Python数据之道

PyDataLab

Python知识手册

V3.0

作者 | Lemon

出品 | Python数据之道

01 Jan 2021

Python 知识手册-v3.0

出品：Python 数据之道

作者：Lemon

01 Jan, 2021

前言

“种一棵树，最好的时间是十年前，其次是现在。”

各位读者好，我是 Lemon 。博客刚兴起不久的时候，我就开始写一些博客文章，但工作之后基本上就没有坚持写。现在想来，有些可惜了。

从 2016 年底开始，再次坚持写作，并于 2017 年开通了微信公众号「Python 数据之道」。期间陆陆续续，写了些许 Python 及数据科学相关的内容，以及翻译了部分外网文章，希望借此机会，将这些显得零散的内容整合起来，形成一本小册子，名曰《Python 知识手册》。

虽然我写了一些关于 Python 零基础入门的文章，但《Python 知识手册》并没有比较完整的覆盖 Python 的基础知识。目前，《Python 知识手册》的内容主要涉及 Python 基础、数据分析，数据可视化等内容。因此，针对手册的阅读，各位读者最好有一些 Python 的基本功底。

若对我写的内容有兴趣，欢迎大家通过以下途径来关注。

1. 微信公众号

公众号「Python 数据之道」秉承“让数据更有价值”的理念，主要分享数据相关的内容，包括数据分析，挖掘，可视化，机器学习，深度学习等, 欢迎关注。

《Python 知识手册》中部分文章提供了源代码，可以在公众号「Python 数据之道」后台回复「code」 来获取。

2. 个人网站

网址：http://liyangbit.com

我的个人网站中，包含更多的文章，并且在不断的进行更新。目前，网站中涉及了 Python 相关一系列内容，包括 Python 基础、Python 数据科学、项目实战等内容，欢迎访问。

3. 感谢的话

最后聊一下 Lemon 为什么写这本手册：作为一个非科班人士，原本跟 Python 没有直接的关联，在长期的工作中我希望自己能尽快走出舒适圈，做一些有意义的事情，所以我跑去研究当前比较火热的人工智能，发现 Python 的强大之处，从而开始深入到 Python 的学习中去了。过去的时间里，在公众号「Python 数据之道」里写了一些内容，希望整合起来，对大家有些启发，于是有了《Python 知识手册》。

坚持写作，少了许多与家人陪伴的时光，感谢家人对我的支持与理解。通过「Python 数据之道」，结识了不少业界前辈以及新朋友，通过交流，让我愈发感到自己的渺小与不足，希望能与诸位共同成长。衷心感谢诸位前辈以及众多读者对「Python 数据之道」的支持。手册封面图片 由 Free-Photos 在 Pixabay 上发布，在此表示感谢。

当然，由于个人水平有限，文章中内容难免有不准确的地方，《Python 知识手册》会不定期进行更新，更新后的内容可以在公众号「Python 数据之道」后台回复数字「600」 进行获取。

有你们同行，Lemon 相信《Python 知识手册》的内容会越来越完善，欢迎大家提供反馈建议，Lemon 的微信联系方式如下：

4. 做一个跨领域学习者

此外，Lemon 还开设了一个非技术类的微信公众号，名称是「价值前瞻」 。在这里推荐下，也算是一点小小私心，希望大家能关注下。

「价值前瞻」 的初衷是希望自己以及更多的人能“做一个有远见的人”，「价值前瞻」 聚焦于书籍阅读、投资创业、自我成长等领域，希望能前瞻思考，不断适应外部环境变化，成就更好的自己。

Tips:

> 您现在看到的可能已经不是最新版本，请在公众号「Python 数据之道」后台回复数字「600」 获取最新版。

Contents

1 导读 **11**

1.1 手册内容 . . . 11
1.2 与优秀者同行 . . . 11
1.3 如何阅读和使用《手册》 . . . 12
1.4 手册版本历史信息 . . . 12

2 Python 语言的简单介绍 **13**

2.1 起源 . . . 13
2.2 为什么名字叫 “Python” . . . 15
2.3 Python 的发展历史 . . . 15
2.4 Python 语言的特点 . . . 16
2.5 Python 语言的应用 . . . 17
2.6 Python 语言的一些缺点 . . . 18

3 Python 安装 **19**

3.1 安装方法一： . . . 19
3.2 安装方法二：（推荐） . . . 24
3.3 总结 . . . 27

4 第一个 Python 程序 **28**

4.1 输出函数 print() . . . 28
4.2 输入函数 input() . . . 30
4.3 小练习 . . . 31

5 创建第一个“.py” 和 “.ipynb” 文件 **32**

5.1 创建和运行 “.py” 文件 . . . 32
5.2 文本编辑器 . . . 39
5.3 创建和运行 “.ipynb” 文件 . . . 39
5.4 总结 . . . 45

6 VS Code 中 Python 环境配置指南 **46**

6.1 软件准备 . . . 46
6.2 在终端运行程序 . . . 47
6.3 Python 环境配置 . . . 48

7 Python 数据类型-List 介绍 60
7.1 初识 list . . . 60
7.2 访问列表中的值 . . . 61
7.3 列表的操作、函数及方法 . . . 62
7.4 list 中元素的类型可以多样 . . . 65
7.5 list 的切片（slices） . . . 66
7.6 列表推导式 (List Comprehension) . . . 68
7.7 小结 . . . 71

8 Python 中字典合并的实现方法 72
8.1 背景 . . . 72
8.2 Method1：适用于 Python3.5 及以上版本呢 . . . 72
8.3 Method2：自定义函数 . . . 72
8.4 多个 dict 进行合并 . . . 73

9 Python 内置时间模块：Datetime 74
9.1 前言 . . . 74
9.2 构建时间对象实例 . . . 75
9.3 时间转换 . . . 79
9.4 时间对象的运算 . . . 83

10 Python 内置时间模块：Time 86
10.1 模块介绍 . . . 86
10.2 时间获取 . . . 87
10.3 时间转换 . . . 87

11 Python 内置时间模块：Calendar 90
11.1 获取某年的日历 . . . 90
11.2 获取某月的日历 . . . 91
11.3 其他方法 . . . 92

12 Jupyter Notebook 主题设置、字体修改等 93
12.1 新的风格预览 . . . 93
12.2 主题更换过程 . . . 94
12.3 我的主题设置参数 . . . 95

13 Jupyter Notebook 中自动补全代码 97

14 Jupyter Notebook 输出 pdf 并支持中文显示 99
14.1 安装顺序 . . . 99

15 Anaconda 安装虚拟环境到指定路径 105
15.1 曾经的困扰 . . . 105

15.2 安装虚拟环境 . . . 105
15.3 虚拟环境下安装 python 库 . . . 108

16 Numpy 的基本用法 110
16.1 Numpy 数组对象 . . . 110
16.2 创建 ndarray 数组 . . . 111
16.3 Numpy 的数值类型 . . . 113
16.4 ndarray 数组的属性 . . . 114
16.5 ndarray 数组的切片和索引 . . . 118
16.6 处理数组形状 . . . 119
16.7 数组的类型转换 . . . 127
16.8 numpy 常用统计函数 . . . 127
16.9 数组的广播 . . . 129
16.10 写在最后 . . . 130

17 Numpy 的 random 函数简介 131
17.1 numpy.random.rand() . . . 131
17.2 numpy.random.randn() . . . 132
17.3 numpy.random.randint() . . . 132
17.4 生成 [0,1) 之间的浮点数 . . . 133
17.5 numpy.random.choice() . . . 134
17.6 numpy.random.seed() . . . 135

18 Numpy 中 Meshgrid 函数介绍及 2 种应用场景 136
18.1 Meshgrid 函数的基本用法 . . . 136
18.2 Meshgrid 函数的一些应用场景 . . . 139
18.3 Method-1 . . . 141
18.4 Method-2 . . . 143

19 Pandas 日期数据处理 145
19.1 读取并整理数据 . . . 145
19.2 按日期筛选数据 . . . 146
19.3 按日期显示数据 . . . 148
19.4 按日期统计数据 . . . 152

20 Pandas 的 DataFrame 如何按指定 list 排序 156
20.1 分析过程 . . . 158
20.2 指定 list 元素多的情况： . . . 160
20.3 指定 list 元素少的情况： . . . 161
20.4 总结 . . . 162

21 Pandas 小册子：根据条件创建新的列 163
21.1 第一种方法 . . . 163

21.2 第二种方法 . . . 164

22 Pandas 中 apply 使用技巧! 166
22.1 使用 apply 函数拆分文本 . . . 166
22.2 应用有两个参数的自定义函数 . . . 168

23 Pandas 的 Groupby 操作 171
23.1 groupby 的基础操作 . . . 171
23.2 分组后选择列进行运算 . . . 172
23.3 聚合方法 . . . 173
23.4 分组运算方法 agg() . . . 174
23.5 分组运算方法 apply() . . . 175
23.6 分组运算方法 transform() . . . 176
23.7 将某列数据按数据值分成不同范围段进行分组（groupby）运算 . . . 177

24 Matplotlib 饼图简介 179
24.1 官方 Demo . . . 179
24.2 将实际数据应用于官方 Demo . . . 180
24.3 一些改善措施 . . . 181

25 Matplotlib 中等高线图（contour）的绘制 191
25.1 仅绘制等高线 . . . 192
25.2 仅填充等高区域颜色: . . . 192
25.3 绘制完整的等高线图 . . . 193

26 Matplotlib 可视化最有价值的 50 个图表 195
26.1 介绍 . . . 195
26.2 准备工作 . . . 195
26.3 关联（Correlation） . . . 196
26.4 偏差（Deviation） . . . 209
26.5 排序（Ranking） . . . 217
26.6 分布（Distribution） . . . 224
26.7 组成（Composition） . . . 237
26.8 变化（Change） . . . 244
26.9 分组（Groups） . . . 261

27 轻松用 Seaborn 进行数据可视化 268
27.1 直方图（Distplot） . . . 268
27.2 联合分布图（Jointplot） . . . 270
27.3 矩阵图（Pairplot） . . . 273
27.4 条形图（Barplot） . . . 274
27.5 箱形图（Boxplot） . . . 276
27.6 LM Plot . . . 278

28 Seaborn 可视化：图形个性化设置的几个小技巧 280
28.1 概述 . . . 280
28.2 未个性化设置的情形 . . . 280
28.3 进行个性化设置 . . . 281

29 Seaborn 热力图使用进阶 284
29.1 构造数据 . . . 284
29.2 Seaborn 的 heatmap 各个参数介绍 . . . 286
29.3 案例应用：突出显示某些数据 . . . 296

30 Bokeh 入门 298
30.1 常规步骤 . . . 298
30.2 绘制 circle() . . . 298

31 Bokeh：figure 详细解读 304
31.1 Bokeh 中绘图的一般步骤 . . . 304
31.2 综合小结 . . . 315

32 Bokeh: 29 种基础可视化图形 317
32.1 Bokeh 中绘图的一般步骤 . . . 317
32.2 综合小结 . . . 328

33 Bokeh 中独特的数据类型简介：ColumnDataSource 329
33.1 直接提供数据 . . . 329
33.2 通过 ColumnDataSource 来提供数据 . . . 330
33.3 小结 . . . 335

34 Bokeh 中数据的添加、修改和筛选 336
34.1 添加新的数据 . . . 336
34.2 数据更新 . . . 338
34.3 筛选数据 . . . 341

35 Bokeh 中图形与组件的布局简介 345
35.1 图形的布局 . . . 345
35.2 组件的布局 . . . 351
35.3 图形和组件混合布局 . . . 352

36 在 Pycharm 等编辑器下使用 Plotly Express 354
36.1 第一个示例 . . . 354
36.2 第二个示例 . . . 355

37 Plotly 中如何保存 jpeg 等图片？ 357
37.1 保存静态图片的方法 . . . 357

37.2 项目案例演示 357

38 用 Python 可视化神器 Plotly 动态演示全球疫情变化趋势 360
38.1 数据来源 360
38.2 准备工作 361
38.3 获取国外的疫情数据 361

39 用 Python 动态曲线图来对全球疫情进行演示 366
39.1 数据来源 367
39.2 准备工作 368
39.3 获取国外和国内的疫情数据 369
39.4 疫情可视化 370

40 升级版，用 Python 来进行多条曲线动态演示全球疫情变化 376
40.1 数据来源 378
40.2 准备工作 378
40.3 获取国外和国内的疫情数据 379
40.4 疫情可视化 381
40.5 动态曲线演示疫情情况 383

41 Plotly 中 4 种文本类型设置详解 387
41.1 项目环境背景 389
41.2 数据获取 389
41.3 Plotly 中文本设置 390

42 推荐一个牛逼的生物信息 Python 库 - Dash Bio 395
42.1 探索 3d 状态下的小分子 397
42.2 分析显微镜图像中的细胞 398
42.3 进行药代动力学分析 399
42.4 可视化 FASTA 数据 400
42.5 突出基因组相似性 401
42.6 可视化微阵列结果 402
42.7 搜索和选择序列 403
42.8 3d 分子的环境遮挡 404
42.9 使用 Dash Ideogram 可视化染色体 405
42.10 可视化基因突变 406
42.11 测量和注释医学图像 407
42.12 构建系统发育树和网络图 408
42.13 Final 409

43 项目实战：用 Python 读取巴菲特最近的持仓数据 410
43.1 数据表格情况 410
43.2 数据读取 411

43.3 xml 格式内容读取 . . . 411

44 项目实战：Python 数据分析，UFO 长啥样? 414
44.1 数据整理与清洗 . . . 414
44.2 UFO 长啥样? . . . 417
44.3 UFO 在美国那些州（state）出现的次数比较多? . . . 420
44.4 UFO 在哪些年份出现的次数较多? . . . 421
44.5 1997 年以后的 UFO 事件分析 . . . 423

45 项目实战：世界杯系列 427
45.1 第一篇：2018 世界杯：用 Python 分析热门夺冠球队 . . . 427
45.2 第二篇：德国是 2018 世界杯夺冠最大热门? . . . 440

46 项目实战：福布斯系列 460
46.1 数据分析思路 . . . 460
46.2 数据采集 . . . 460
46.3 数据完整性检查 . . . 464
46.4 补充数据收集 . . . 468
46.5 数据清洗-2007 年数据 . . . 472
46.6 数据清洗-2008-2010 年 . . . 479
46.7 数据清洗-2011-2015 年 . . . 482
46.8 数据清洗-2016 年 . . . 486
46.9 数据清洗-2017 年 . . . 491

47 学习资料: 精选文章 498
47.1 付费阅读文章 . . . 498
47.2 免费阅读文章 . . . 498
47.3 读者投稿文章 . . . 498

48 学习资料: Python 网站 500
48.1 官方网站 . . . 500
48.2 个人博客 . . . 500
48.3 金融量化 . . . 500
48.4 优质网站 . . . 500

49 学习资料：Python 书籍 502
49.1 《笨办法学 Python3》 . . . 502
49.2 《Python 编程从入门到实践》，第 2 版 . . . 507

50 延伸阅读 513
50.1 我是谁 . . . 513
50.2 推荐内容 . . . 514

1 导读

大家好，我是 Lemon 。

《Python 知识手册》由「Python 数据之道」出品，最新版会在公众号上进行发布，大家可以在公众号「Python 数据之道」后台回复 **600** 获取最新版。

虽然我写了一些关于 Python 零基础入门的文章，但《Python 知识手册》并没有比较完整的覆盖 Python 的基础知识。因此，针对手册的阅读，各位读者最好有一些 Python 的基本功底。

1.1 手册内容

《Python 知识手册》V3.0 版主要更新了 2020 年下半年我写的一些内容，涉及主要涉及 VS Code、Pandas、Plotly 以及学习资料等内容。

当前版本中，手册主要包含以下几方面相关的内容：

- Python 基础知识
- Jupyter Notebook
- Numpy
- Pandas
- Matplotlib
- Seaborn
- Bokeh
- Plotly
- 若干个项目实战案例
- Python 学习资料

如没有特殊的说明，手册中代码的运行环境如下：

- windows 7 或 windows 10 系统或 MacOS 系统
- python 3.5 或 python 3.5+
- Anaconda / Jupyter Notebook / VS Code

1.2 与优秀者同行

从 V3.0 版开始，加强了延伸阅读的内容，一个人的水平、时间和经验总是有限的。在《手册》中，以介绍或链接的方式，添加了部分内容，包括书籍介绍、网站及博客介绍、翻译内容链接、读者投稿文章的链接等，这些内容，可以在

《手册》的学习资料章节进行了解。

1.3 如何阅读和使用《手册》

《Python 知识手册》的大部分章节来自公众号「Python 数据之道」发布的内容，每个章节，基本上是一个独立的内容。因此，读者可以根据自己的知识储备情况，有针对性的阅读，熟悉的章节可以快速的阅览或者跳过，对于有兴趣的内容，可以详细阅读和练习。

1.4 手册版本历史信息

《Python 知识手册》更新信息情况如下：

版本号	日期	主要内容描述
v3.0	2021-01-01	更新 VS Code、Pandas、Plotly 以及学习资料等内容
v2.2	2020-05-11	更新 Plotly 相关内容和 Python 学习资料的章节
v2.1	2020-02-14	添加“导读” 和 “Python 网站” 章节，移除部分项目
v2.0.1	2020-02-02	修复 2.0 版没有书签的 bug
v2.0	2020-02-02	添加 Python 基础知识；添加 Plotly 内容
v2018	2019-02-25	第一版，初始版本

2 Python 语言的简单介绍

2.1 起源

Python 的创立者，Guido van Rossum，荷兰人。1982 年，Guido 从阿姆斯特丹大学获得了数学和计算机硕士学位。然而，尽管他算得上是一位数学家，但他更加享受计算机带来的乐趣。用他的话说，尽管拥有数学和计算机双料资质，他总趋向于做计算机相关的工作，并热衷于做任何和编程相关的活儿。

Python 的创始人吉多・范罗苏姆（Guido van Rossum），在 1989 年 12 月的圣诞节期间，为了打发时间，决定开发一种新的脚本解释程序，作为 ABC 语言的继承。1991 年 2 月，第一个 Python 编译器诞生。现在，python 以其优美，清晰，简单的特性在全世界广泛流行，成为最主流的编程语言之一。

Guido van Rossum 有一句著名的话就是 “**Life is short, you need Python**”，译为：“人生苦短，我用 Python”。

截止至 2017 年 11 月，python 在 TIOBE 流行编程语言排行榜，已经上升到第四名。并且从趋势图中可以看出其还在呈增长趋势。

Nov 2017	Nov 2016	Change	Programming Language	Ratings	Change
1	1		Java	13.231%	-5.52%
2	2		C	9.293%	+0.09%
3	3		C++	5.343%	-0.07%
4	**5**	▲	**Python**	**4.482%**	**+0.91%**
5	4	▼	C#	3.012%	-0.65%
6	8	▲	JavaScript	2.972%	+0.27%
7	6	▼	Visual Basic .NET	2.909%	-0.26%
8	7	▼	PHP	1.897%	-1.23%
9	16	⇈	Delphi/Object Pascal	1.744%	-0.21%
10	9	▼	Assembly language	1.722%	-0.72%

Figure 2.1: 排名情况-2017 年 11 月

Jan 2020	Jan 2019	Change	Programming Language	Ratings	Change
1	1		Java	16.896%	-0.01%
2	2		C	15.773%	+2.44%
3	3		Python	9.704%	+1.41%
4	4		C++	5.574%	-2.58%
5	7	^	C#	5.349%	+2.07%
6	5	v	Visual Basic .NET	5.287%	-1.17%
7	6	v	JavaScript	2.451%	-0.85%
8	8		PHP	2.405%	-0.28%
9	15	^^	Swift	1.795%	+0.61%
10	9	v	SQL	1.504%	-0.77%

Figure 2.2: 排名情况-2020 年 1 月

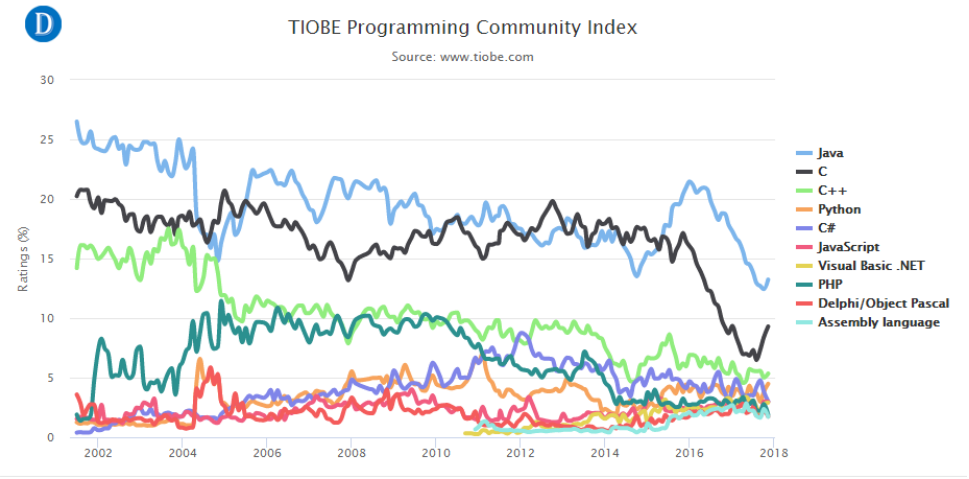


Figure 2.3: 历年排名情况-截至 2017 年 11 月

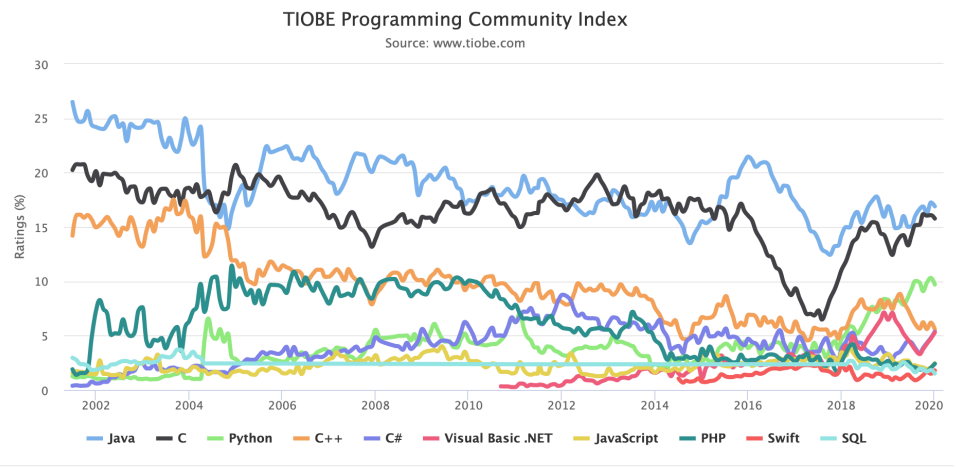


Figure 2.4: 历年排名情况-截至 2020 年 1 月

2.2 为什么名字叫 “Python”

我们知道英文单词 “Python” 的中文翻译是“蟒蛇”或者“巨蛇”的意思。但作为一名编程语言的名称，Python 的初始意思并不是这样的。

Python 这个名字，来自 Guido 所挚爱的电视剧 “Monty Python’s Flying Circus” 。他希望这个新的叫做 Python 的语言，能符合他的理想：创造一种 C 和 shell 之间，功能全面，易学易用，可拓展的语言。

2.3 Python 的发展历史

- Python 3.8 - 2019/10/15
- Python 3.7 - 2018/06/27
- Python 3.6 - 2016/12/23
- Python 3.5 - 2015/09/13
- Python 3.4 - 2014/03/16
- Python 3.3 - 2012/09/29
- Python 3.2 - 2011/02/20
- Python 3.1 - 2009/06/27
- Python 3.0 - 2008/12/03
- Python 2.7 - 2010/07/03
- Python 2.6 - 2008/10/1
- Python 2.5 - 2006/09/19
- Python 2.4 – 2004/11/30, 同年目前最流行的 WEB 框架 Django 诞生
- Python 2.0 - 2000/10/16，加入了内存回收机制，构成了现在 Python 语言框架的基础
- 1999 年 Python 的 web 框架之祖——Zope 1 发布

- Python 1.0 - January 1994 增加了 lambda, map, filter and reduce.
- 1991 年，第一个 Python 编译器诞生。它是用 C 语言实现的，并能够调用 C 语言的库文件。从一出生，Python 已经具有了：类，函数，异常处理，包含表和词典在内的核心数据类型，以及模块为基础的拓展系统。
- 1989 年的圣诞节，Guido 开始编写 Python 语言的编译器。

2014 年 11 月，**Python2.7 将在 2020 年停止支持的消息被发布，并且不会再发布 2.8 版本，建议用户尽可能的迁移到 3.4+**, Python 最初发布时，在设计上有一些缺陷，比如 Unicode 标准晚于 Python 出现，所以一直以来对 Unicode 的支持并不完全，而 ASCII 编码支持的字符有限。例如：对中文支持不好 Python3 相对 Python 早期的版本是一个较大的升级，Py3 在设计的时候没有考虑向下兼容，所以很多早期版本的 Python 的程序无法再 Py3 上运行。

2.4 Python 语言的特点

1）简单易学

Python 语言简洁、优雅，相比其他程序语言（比如：Java，C++ 等），Python 语言更容易阅读和编写。Python 语言使用起来很有趣，且该语言更多的是专注于解决方案而不是语法本身。

初学者学 Python，不但入门容易，而且将来深入下去，可以编写那些非常非常复杂的程序。

2）开源且免费

Python 是一门开源的编程语言，可以免费使用，甚至可以用于商业用途。当然，你也可以提交相关贡献代码。Python 有一个庞大的社区来维护并保持快速发展。

3）可移植性

Python 是跨平台的编程语言，它可以运行在 windows、Mac 和各种 Linux/Unix 系统上。也就是说，假设在 windows 系统下编写的 Python 程序，在 Mac 或 Linux 系统下也是可以运行的。

4）开发效率非常高

Python 有非常强大的第三方库，基本上你想通过计算机实现任何功能，Python 官方库里都有相应的模块进行支持，直接下载调用后，在基础库的基础上再进行开发，大大降低开发周期，避免重复造轮子。

5）高级语言

当你用 Python 语言编写程序的时候，你无需考虑诸如如何管理你的程序使用的内存以及垃圾回收一类的底层细节。

6）可扩展性

如果你需要你的一段关键代码运行得更快或者希望某些算法不公开，你可以把你的部分程序用 C 或 C++ 编写，然后在你的 Python 程序中使用它们。

7）可嵌入性

你可以把 Python 代码嵌入你的 C/C++ 程序，从而向你的程序用户提供脚本功能。

8）面向对象

Python 中的一切都是对象。面向对象编程（OOP）可以帮助您直观地解决复杂的问题。有了 OOP，你可以通过创建对象把这些复杂的问题划分成更小的集合。

2.5 Python 语言的应用

2.5.1 Python 语言的应用领域

Python 被广泛的应用于诸多领域，比如：

- **数据科学：**数据科学涉及领域很广，涉及到当前火热的人工智能领域。Python 的数据科学包括数据分析、数据可视化、数据挖掘、自然语言处理、机器学习、深度学习等。典型的 Python 库有：Numpy，Scipy，Pandas，Matplotlib，Seaborn，Scikit-learn，tensorflow 等。
- **云计算：**典型应用—— Python 开发的 OpenStack
- **WEB 开发：**众多优秀的 WEB 框架，比如：Django、flask、tornado
- **网络爬虫：**使用 Python 可以便捷的编写网络爬虫，从网页上爬取相关信息，常用的 Python 库包括：Requests，BeautifulSoup，Scrapy 等。
- **系统运维：**运维人员必备，slatstack（系统自动化配置和管理工具），Ansible（自动化运维工具）
- **图形界面开发：** wxPython、PyQT、TKinter

2.5.2 使用 Python 的部分知名公司

越来越多的公司选在 python 作为其主要开发语言，例如：

- Google - Google Groups、Gmail、Google Maps、AlphaGo 等，Google App Engine 支持 python 作为开发语言
- NASA - 美国宇航局，从 1994 年起把 python 作为主要开发语言
- Dropbox - 美国最大的在线云存储网站，全部用 Python 实现，每天网站处理 10 亿个文件的上传和下载
- 豆瓣网 - 图书、唱片、电影等文化产品的资料数据库网站
- BitTorrent - bt 下载软件客户端
- gedit - Linux 平台的文本编辑器
- GIMP - Linux 平台的图像处理软件 (Linux 下的 PS)
- 知乎（国外的 Quora）- 社交问答网站，国内最大的问答社区，通过 Python 开发
- Autodesk Maya - 3D 建模软件，支持 python 作为脚本语言
- YouTube: 世界上最大的视频网站 YouTube 就是用 Python 开发的
- Facebook: 大量的基础库均通过 Python 实现的
- Redhat: 世界上最流行的 Linux 发行版本中的 yum 包管理工具就是用 python 开发的

除上面之外，还有搜狐、金山、腾讯、盛大、网易、百度、阿里、淘宝、土豆、新浪、果壳等公司都在使用 Python 完成各种各样的任务。

更多案例：

https://www.python.org/about/success/

2.6 Python 语言的一些缺点

任何编程语言都有缺点，Python 也不例外。比如：

（1）速度慢，Python 的运行速度相比 C 语言确实慢很多，跟 JAVA 相比也要慢一些，因此这也是很多所谓的大牛不屑于使用 Python 的主要原因，但其实这里所指的运行速度慢在大多数情况下用户是无法直接感知到的，必须借助测试工具才能体现出来，比如你用 C 运一个程序花了 0.01s, 用 Python 是 0.1s, 这样 C 语言直接比 Python 快了 10 倍, 算是非常夸张了，但是你是无法直接通过肉眼感知的，因为一个正常人所能感知的时间最小单位是 0.15-0.4s 左右。其实在大多数情况下 Python 已经完全可以满足你对程序速度的要求，除非你要写对速度要求极高的搜索引擎等，这种情况下，当然还是建议你用 C 去实现的。

随着硬件设备及计算力的增强，运行速度慢的这个缺点相对来说有所改善。在编写代码所花时间与程序运行所用时间进行平衡后，很多企业都开始选择使用 Python。

（2）代码不能加密，因为 PYTHON 是解释性语言，它的源码都是以明文形式存放的，不过我不认为这算是一个缺点，如果你的项目要求源代码必须是加密的，那你一开始就不应该用 Python 来去实现。

参考文献：

1. TIOBE Index for November 2017
2. Python 的历史
3. Python 简介
4. 一门编程语言的发展简史
5. Learn Python Programming-The Definitive Guide

3 Python 安装

Python 是跨平台的编程语言，它可以运行在 windows、Mac 和各种 Linux/Unix 系统上。也就是说，假设在 windows 系统下编写的 Python 程序，在 Mac 或 Linux 系统下也是可以运行的。

目前，Python 的版本主要分为两大类，即 Python3.X 版和 Python2.x 版，这两个版本是不兼容的。由于 Python3.x 版本越来越普及，我们接下来的教程使用 Python3.x 版（Python3.5 版或 Python3.6 版及以上版本均可）。

由于个人 windows 系统用的多，所以这里的 Python 安装以 windows 系统为例来演示。

其他系统下的安装，建议大家可以在网上查找相关安装方法。

下面介绍在 Windows 系统下两种常用的安装方法，第一种最基础，第二种会在以后带来一些便利，个人推荐第二种安装方法。

3.1 安装方法一：

3.1.1 安装官方版 Python3.6

下面以在 windows 系统下安装 Python3.6 版为例：

首先，根据你的 Windows 版本（64 位还是 32 位）从 Python 的官方网站下载 Python3.6 对应的 64 位安装程序 或 32 位安装程序 ，然后，运行下载的 EXE 安装包：

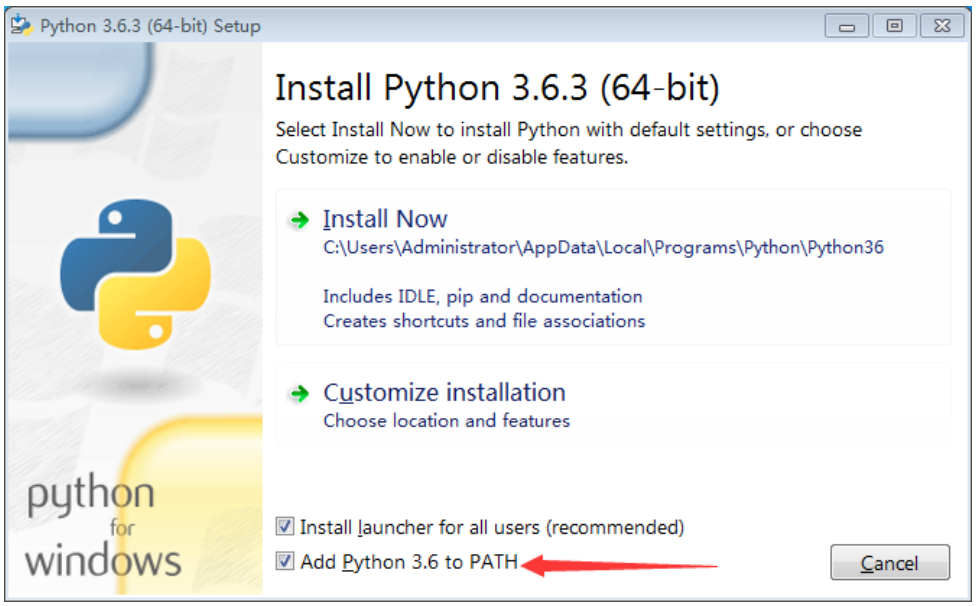


Figure 3.1: 安装过程

特别要注意勾上 `Add Python 3.6 to PATH`，然后点“Install Now”即可完成安装，如下：

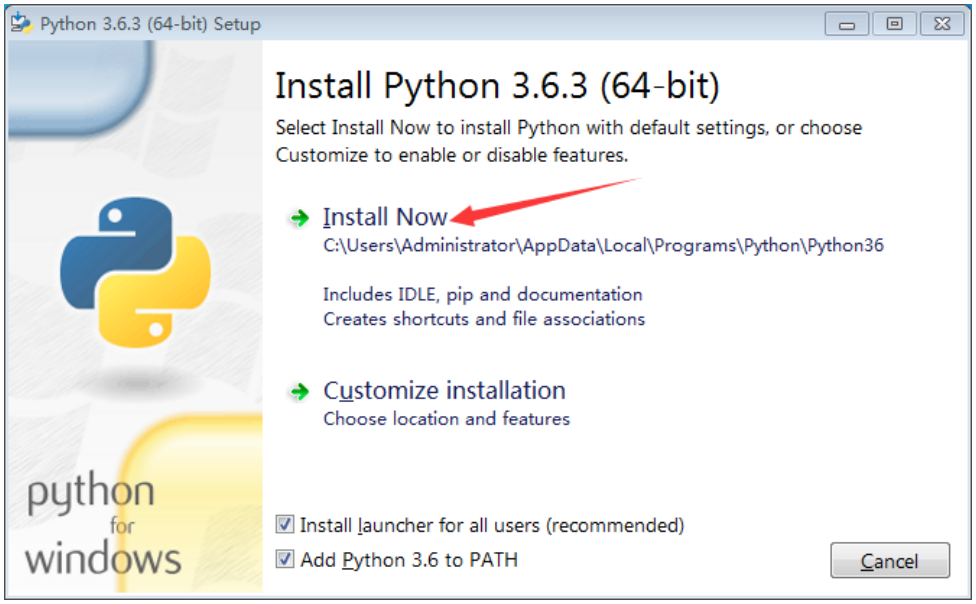


Figure 3.2: 安装过程

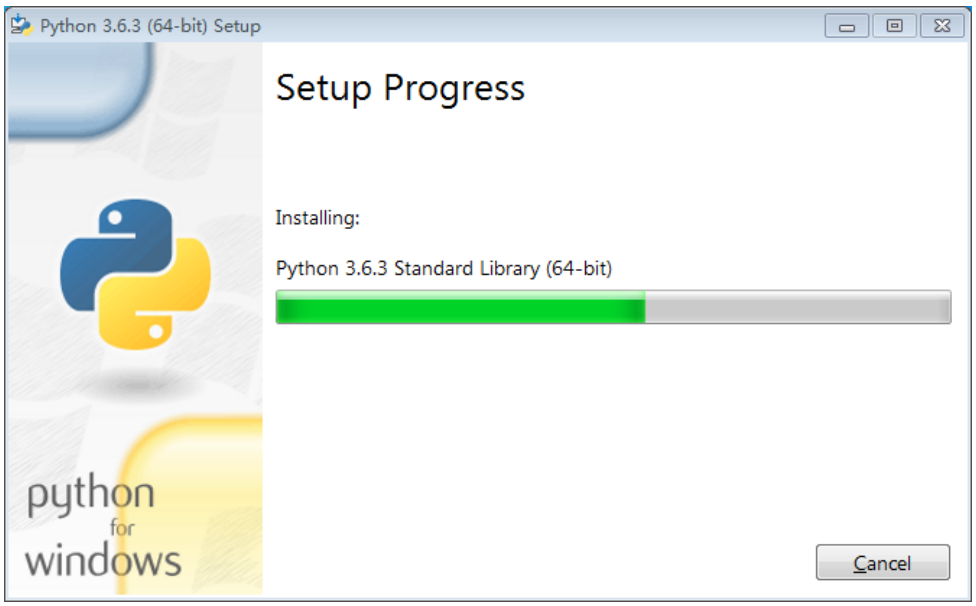


Figure 3.3: 安装过程

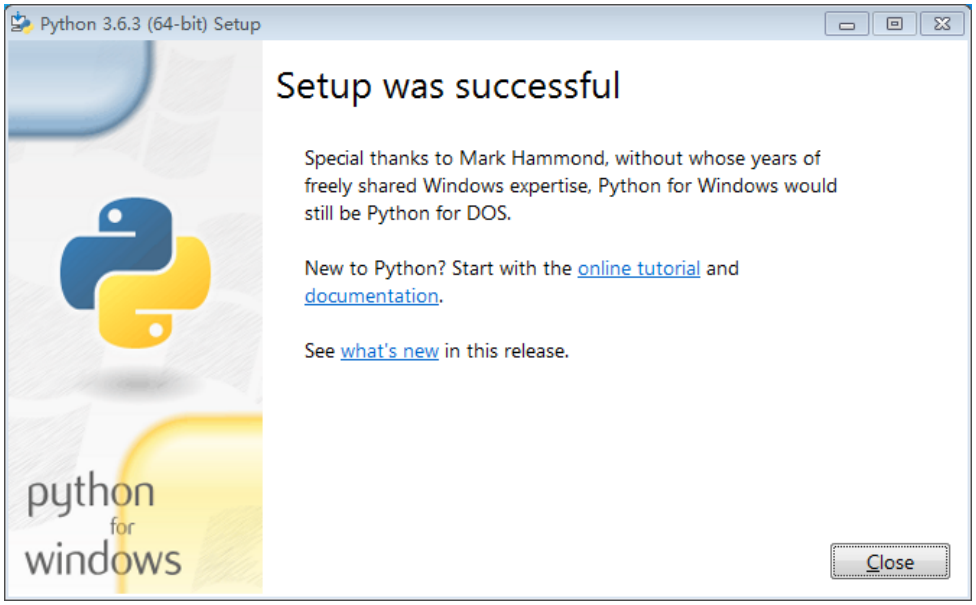


Figure 3.4: 安装过程

3.1.2 运行 Python3.6

安装完成后，我们可以运行下 python 的解释器里，来查看是否安装成功。

首先，在 windows 开始菜单里找到 Python3.6，如下：

Figure 3.5: 开始菜单

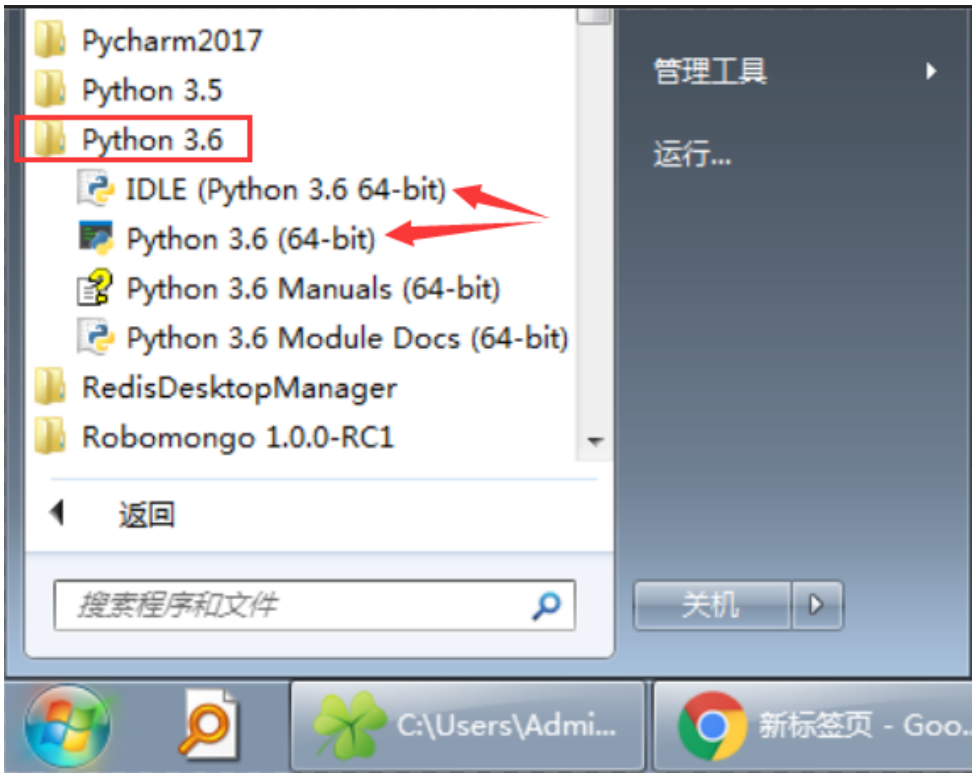


Figure 3.6: 开始菜单

在这里，Python3.6 程序下，有两个可以运行 python 的解释器，我们可以分别打开来看看。

- IDLE （Python 3.6 64-bit）:

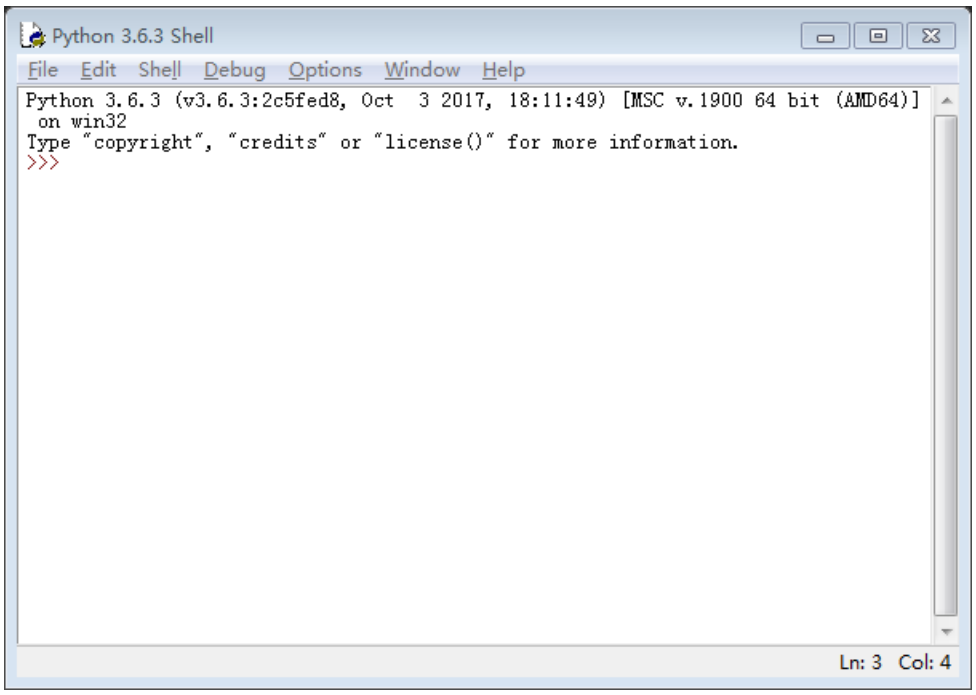


Figure 3.7: IDLE

- Python 3.6 （64-bit）

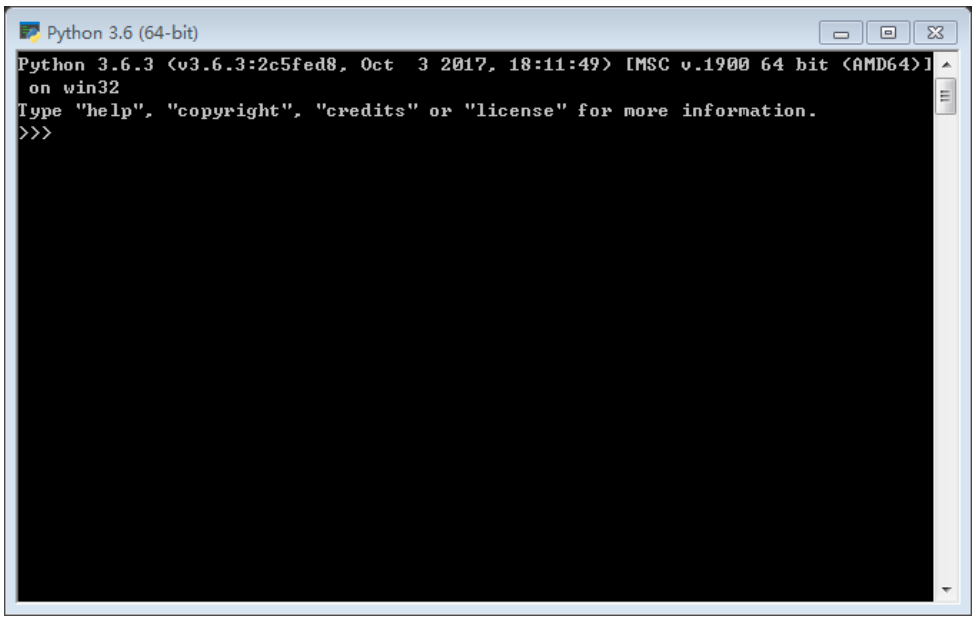


Figure 3.8: CMD

这两个界面下，都是可以运行 python 程序的。这里以 IDLE 界面位例，来测试下 python3.6 是否安装成功。

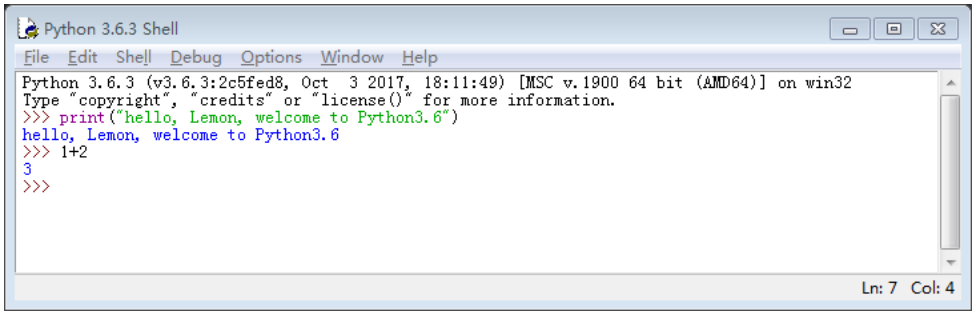


Figure 3.9: IDLE

从上述界面来看，我们的安装程序能够运行 python 语句，已成功的安装了 Python3.6 。

3.2 安装方法二：（推荐）

第二种安装方法是推荐直接安装 Anaconda，它已经包含了 Python 的对应版本。

为什么要推荐安装 Anaconda 呢，针对想用 Python 来进行数据挖掘、机器学习等的童鞋，安装 Anaconda 会带来一些优势，比如：

1. 不需要配置 PYTHON 环境变量；

2. 集成很多 packages，尤其是科学计算相关的库，省去安装这些库带来的烦扰；
3. packages 的安装很简单，conda 一键解决；

3.2.1 安装 Anaconda

安装最新版的 Anaconda 可以去官网下载，地址为：

Anaconda 官网下载地址

windows 系统下，如下：

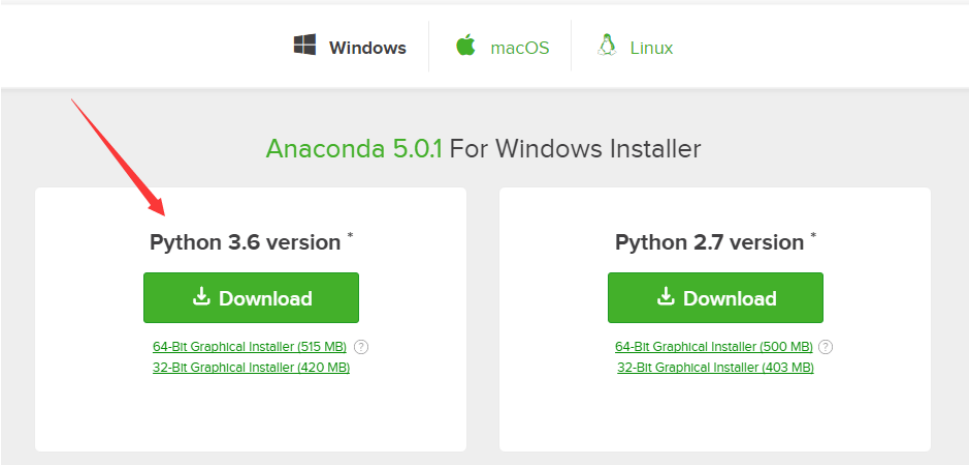


Figure 3.10: Anaconda

根据系统，下载对应的版本，然后根据提示进行安装，选默认设置就可以，一路 “next”，即可安装完毕。

安装完毕后，打开 “cmd”命令行，查看 Anaconda 是否安装好。

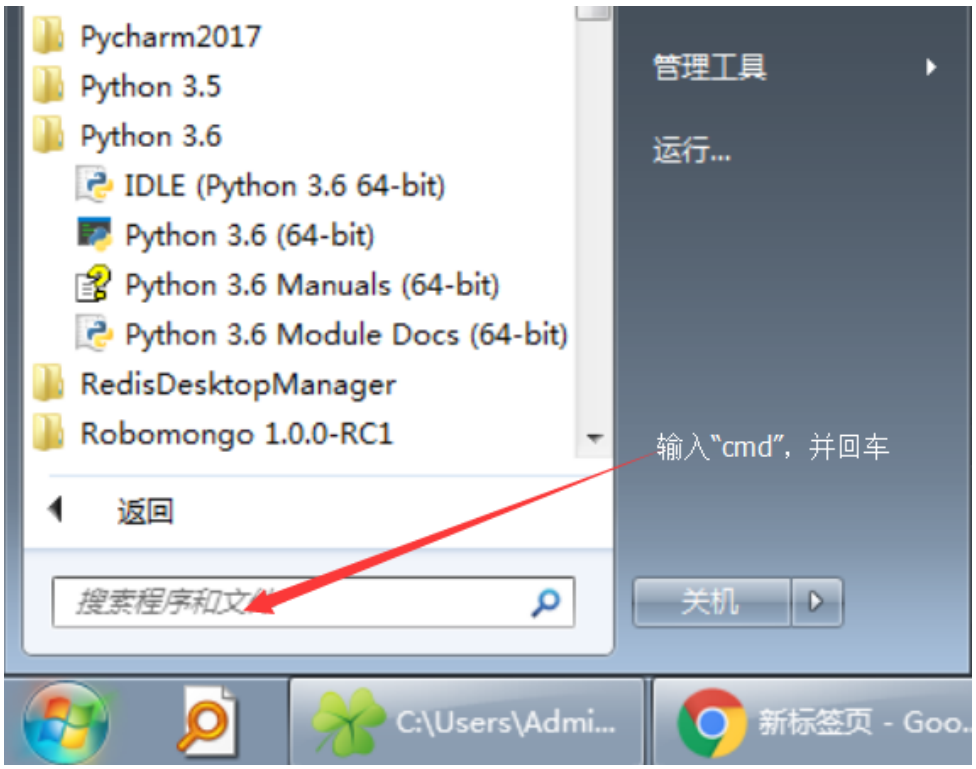


Figure 3.11: cmd

在命令行中输入 `conda info`，查看 Anaconda 安装的版本信息，以及对应的 Python 版本信息。

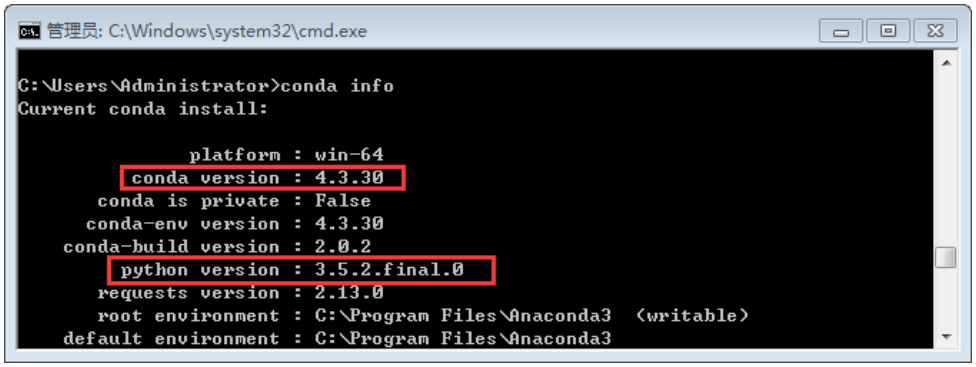


Figure 3.12: conda info

如果你得到了和上图类似的界面，恭喜你已经成功安装了 Anaconda 。

从上图来看，可知我安装的是 Anaconda 4.3.30 版，对应的 Python 版本是 3.5.2 版，**并不是目前的最新版**。

大家可以安装最新的 Anaconda 版本，也可以选择安装跟我一样的版本。

如果想安装的版本不是最新版，可以在下述网址查找相应的版本。

https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/

Figure 3.13: 版本

3.2.2 运行 Python 程序

Anaconda 安装完成后，可以在几个地方运行 Python 程序，最常用的是 “IPython”。

在 windows 开始菜单的所有程序中找到 anaconda 文件夹，点开文件夹，下面可以找到 “IPython”。

点击 “IPython”，得到如下界面:

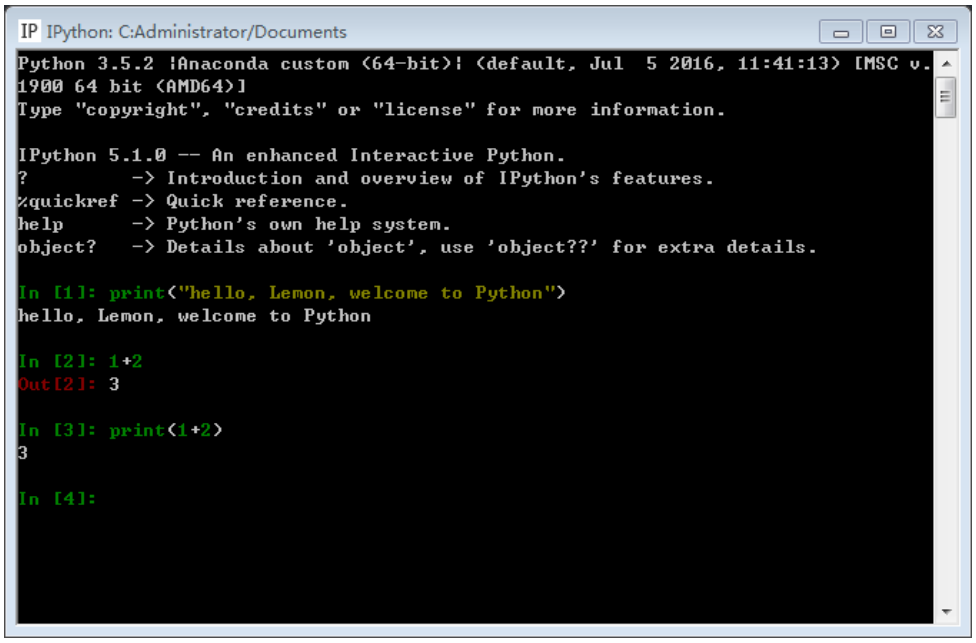


Figure 3.14: ipython

从上述界面来看，我们的安装程序能够运行 python 语句，已成功的安装了 Anaconda 及 Python 。

3.3 总结

对于上述两种安装 Python 的方法，个人推荐第二种方法。尤其是想用 Python 来进行数序分析、数据挖掘、机器学习等，第二种方法会带来很多便捷性。

4 第一个 Python 程序

在安装好 Python 程序后，我们在 Anaconda 中找到 iPython 来编写代码并输出。

在编写代码前，先介绍下运行环境：

- windows 7
- Python 3.5

后续的教程，如果没有特别说明，基本上也是在 Python 3.x （当前用的是 3.5 版）下运行。

4.1 输出函数 print()

跟其他很多程序一样，我们第一个程序用 Python 来输出 “Hello, world”。

Python 代码如下：

```
1 print("hello, world")
```

输入结果如下：

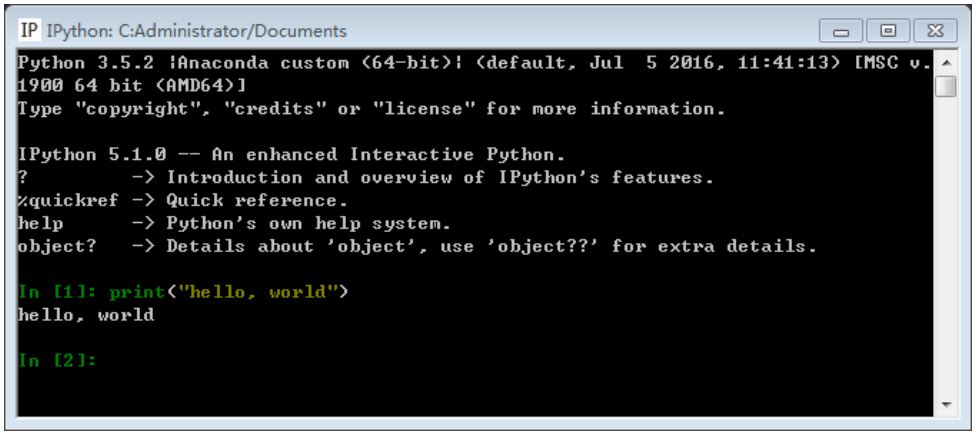


Figure 4.1: hello-world

语法解释：

python 语言中，输出用 `print()` 函数来描述，括号内可以是字符串（比如 “hello, world”），也可以是数据的运算。

当内容是字符串时，可以用双引号，也可以用单引号。代码演示如下：

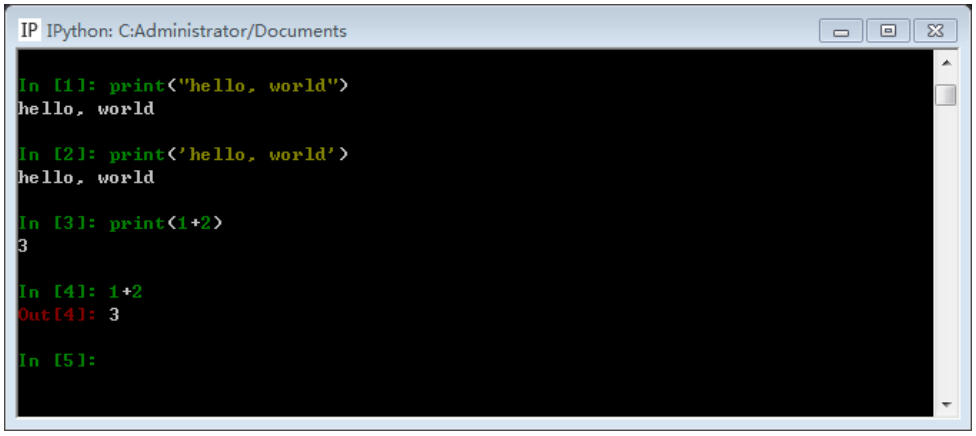


Figure 4.2: demo

注意到，上图中输出 “1+2” 时，可以用 `print(1+2)` 或者 `1+2` ，并且数据运算时是不需要加引号的。

当然，`print(1+2)` 显示的结果为 “3”，有时候，你可能希望显示这样的结果 [1+2=3]，演示如下：

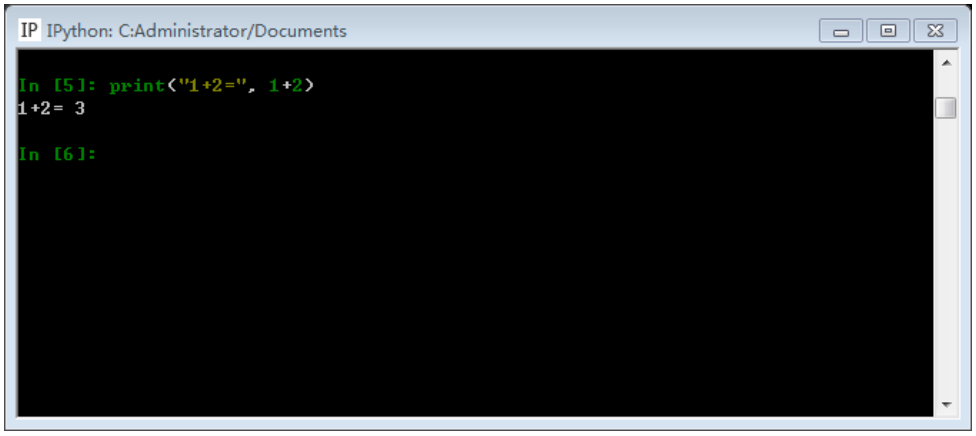


Figure 4.3: demo

从上面可以看出，`print()` 可以接收多个参数，参数之间以逗号隔开，遇到逗号会输出一个空格。

还是以 “hello, world” 为例，可以再看下输出多个字符串的情形。

```
1 print("hello,", "world", "welcome!")
```

演示如下：

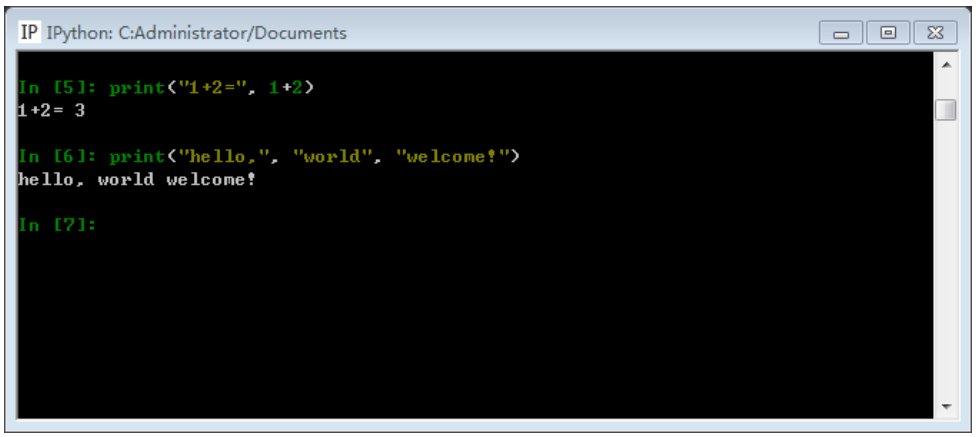


Figure 4.4: demo

4.2 输入函数 input()

上面提到的是输出函数，经常会遇到的一种情形是，需要用户输入相关信息，然后程序才继续运行，这种情况怎么办呢?

Python 提供了 `input()` 函数来让用户输入信息。

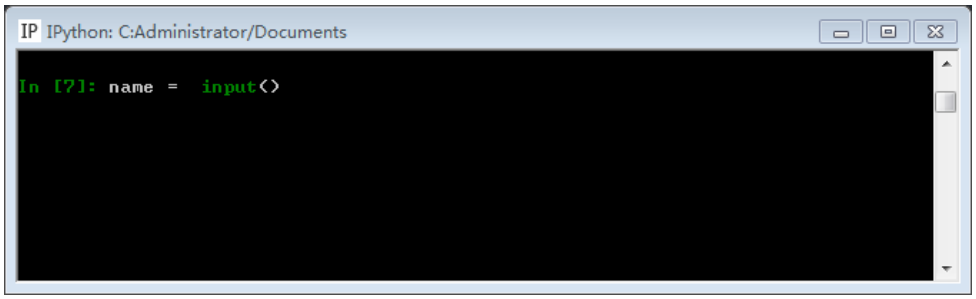


Figure 4.5: demo

在上面程序中，运行后需要用户输入用户名信息，程序才能继续运行，我们输入“Lemon”（不含引号）来试试。

输入“Lemon”后，我们再运行代码 `print(name)`，可以看到其输出结果，演示如下：

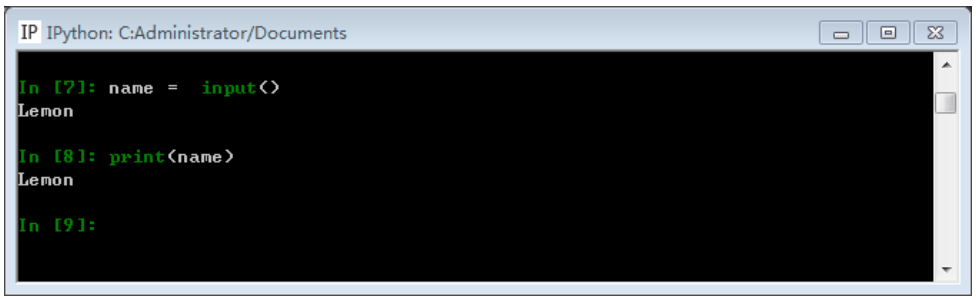


Figure 4.6: demo

上图中，有 2 个“Lemon”（不含引号），其中第一个是用户输入的信息，第二个是输出的结果。

4.3 小练习

在“iPython”中输入如下代码，看看运行情况。

```
1 import this
```

> Python 程序是不是很简洁易懂啊，快来动手试试吧。

5 创建第一个“.py”和“.ipynb”文件

上一节中，我们运行了第一个 Python 程序。在上节中，我们是在 Anaconda 程序文件夹下打开 “iPython” 来运行的 Python 程序。可能你已经注意到，在上节中，写好的 Python 程序运行后，输出了相关结果。但是，Python 代码文件并没有保存下来，如果下次还要运行相同的代码，或者已有代码需要进行修改时，在 “iPython” 中不好实现。

这里，我们来介绍下，在通常情况下，我们应该怎样保存和修改 Python 程序文件。

通常，我们看到的用 Python 语言编写的源代码文件，其文件后缀是 “.py” 或 “.ipynb”。这里，我列了两种文件，其中 “.py” 文件是标准的 Python 源代码文件，通常情况下，我们会使用 “.py” 的 python 源代码文件。

另一种是 “.ipynb” 文件，会在本文后续部分进行描述。

5.1 创建和运行 “.py” 文件

5.1.1 创建 “.py” 文件

在 E 盘下创建名称为 “tutorials” 的文件夹，打开“tutorials” 文件夹，鼠标右击创建文本文档，修改文件名称为 “hello.py” ，演示如下：

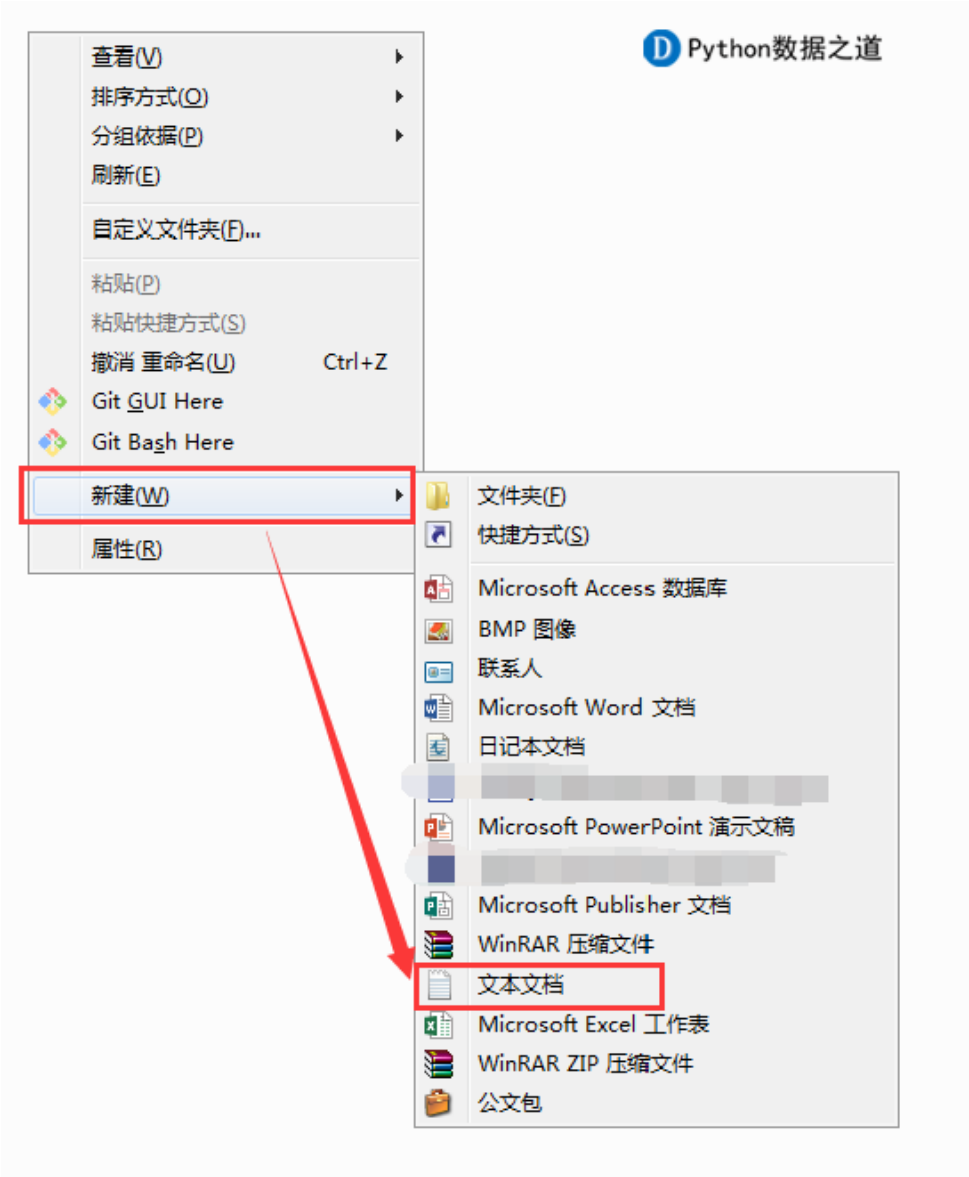


Figure 5.1: demo

注意要修改文件的后缀为 “.py” （默认是 “.txt”）

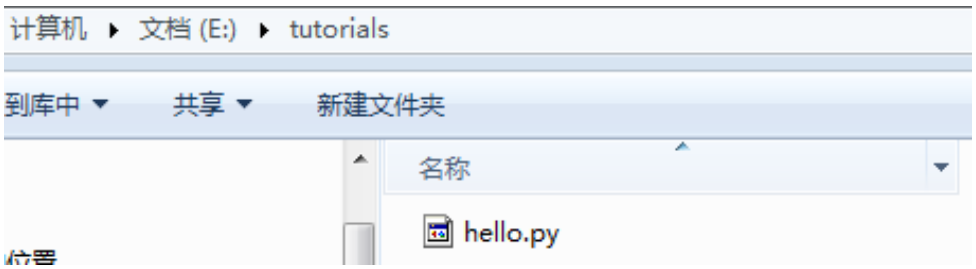


Figure 5.2: demo

5.1.2 打开 “.py” 文件

依次点击“windows 开始菜单”，“所有程序”，在 Anaconda 程序文件夹下打开 “spyder” 程序（对应下图红色数字 1、2、3）

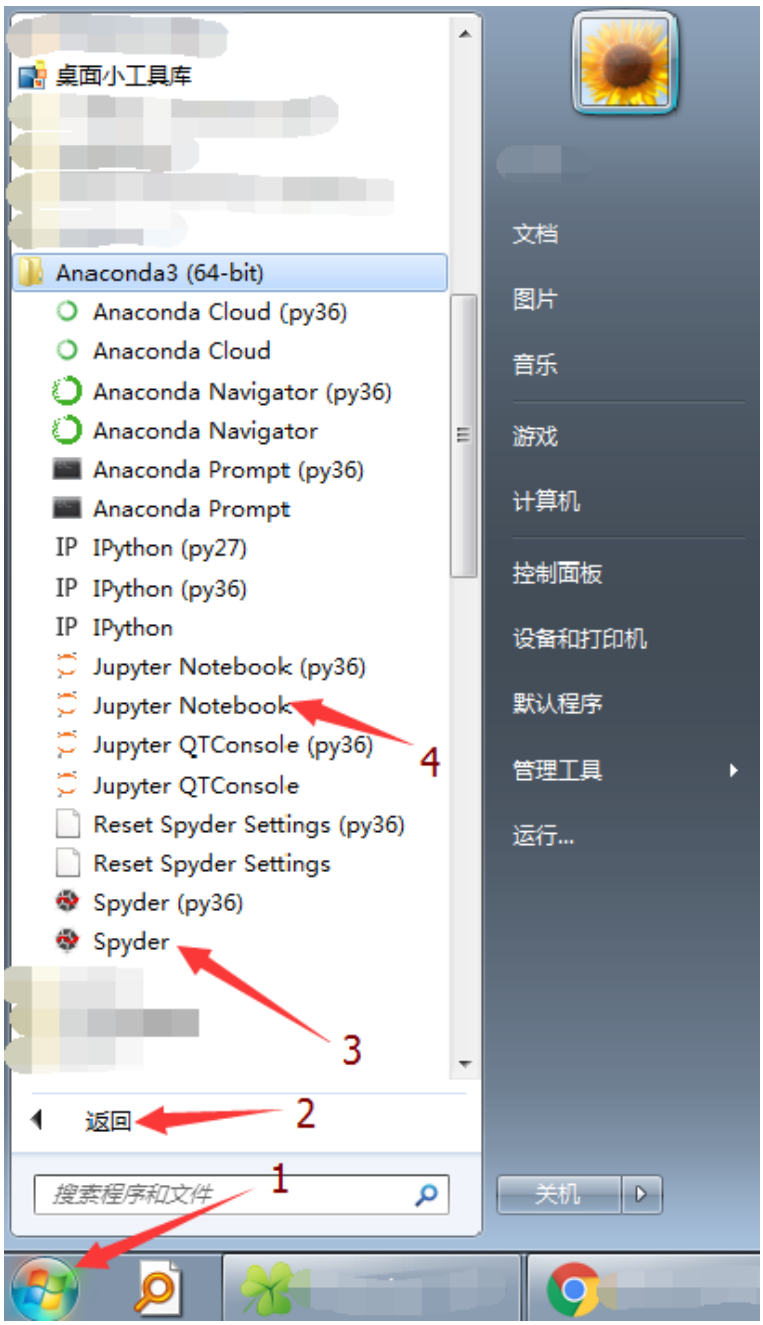


Figure 5.3: demo

然后从 “spyder” 中打开 “hello.py” 程序文件（下图红色箭头所指），得到如下界面：

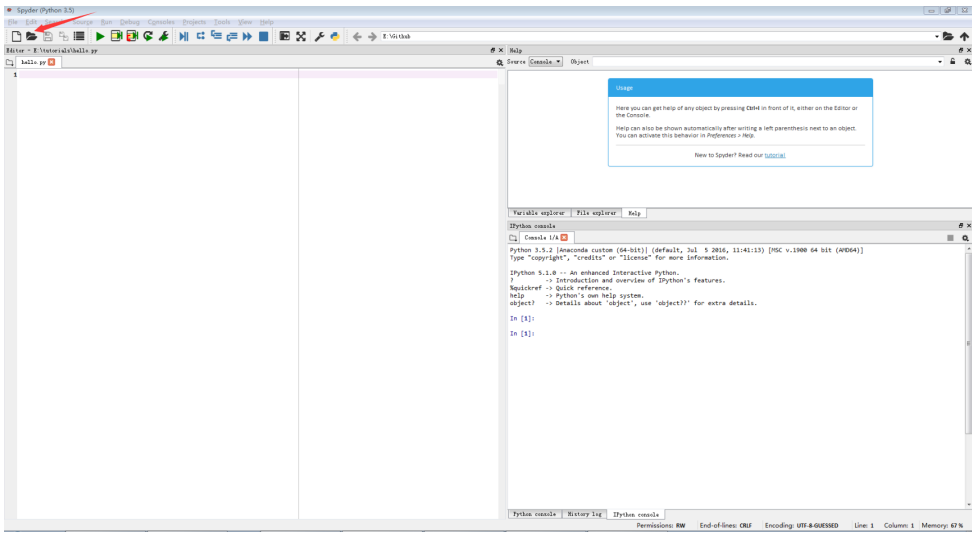


Figure 5.4: demo

打开 “hello.py” 文件后，我们可以从左边看到，这是一个空白文件，里面没有任何代码。

在上图界面的右下角，我们可以找到 “iPython console” （iPython 控制台），我们可以选择在此输出 Python 程序的运行结果。

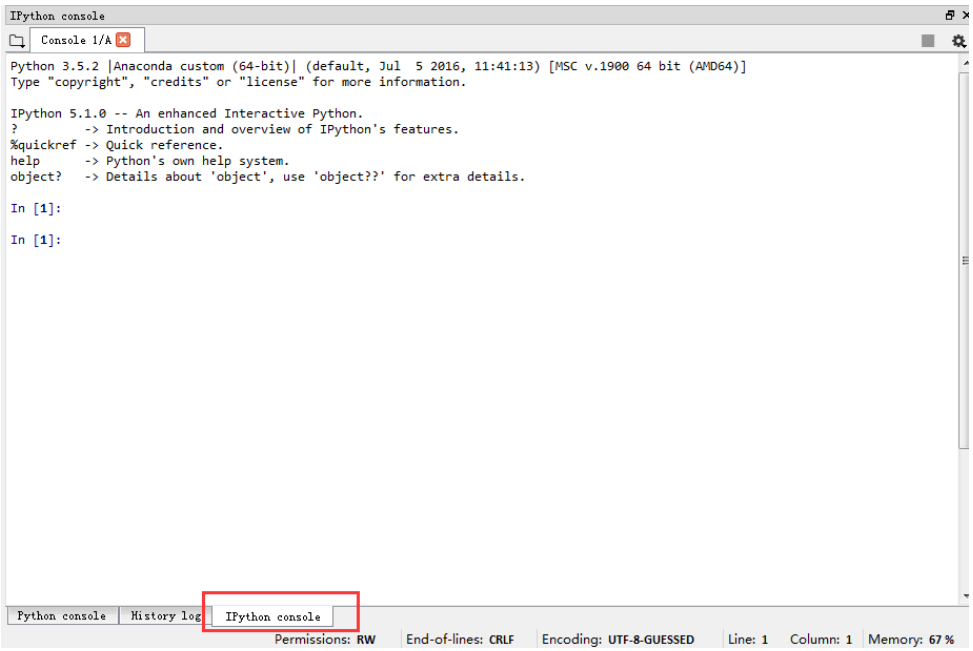


Figure 5.5: demo

5.1.3 运行 “.py” 文件

在 “hello.py” 文件中输入如下代码：

```
1 print("hello, world")
```

保存文件，然后点击绿色的“三角符号”来运行程序（spyder 中运行 python 程序的快捷键是 “F5”），如下：

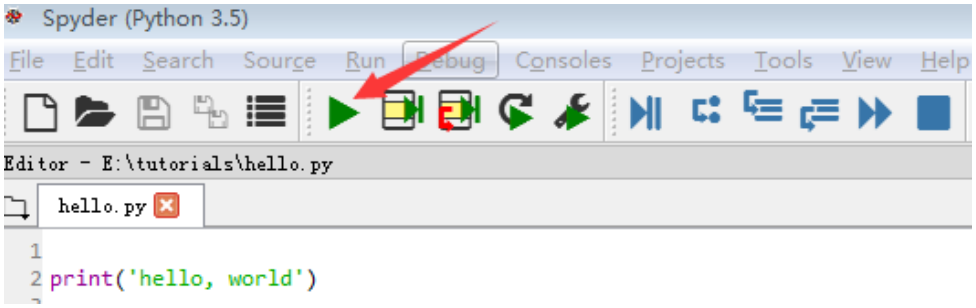


Figure 5.6: demo

如果是第一次运行，点击绿色的“三角符号”后，会弹出如下对话界面：

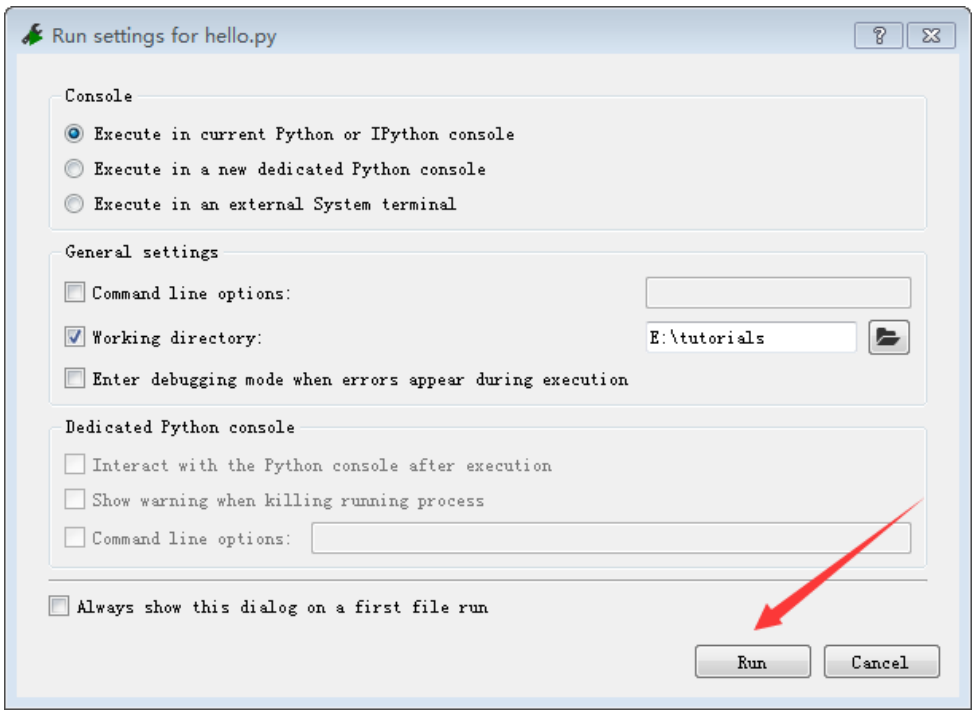


Figure 5.7: demo

选择上图所示默认设置，然后点击 “run”，运行 python 程序，此时，我们会在右下角的 IPython 控制台输出结果，如下：

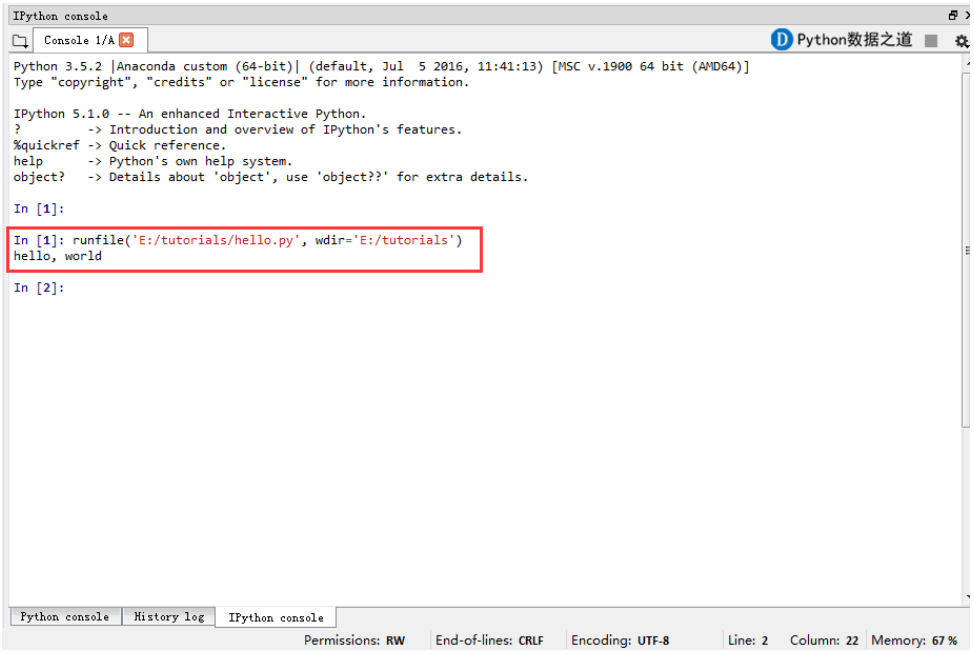


Figure 5.8: demo

可以看到，我们输出的结果是 “hello, world” （不含引号），跟上节在 iPython 界面下的输出结果是一致的。

我们再来试试在“hello.py” 文件中加入更多代码：

```
print("hello, world")

print("hello,", "world", "welcome!")

print(1+2)

print('1+2=', 1+2)
```

保存文件，然后运行程序，运行结果如下：

```
hello, world
hello, world welcome!
3
1+2= 3
```

请注意，不同于 iPython 界面，在 “hello.py” 程序文件中，每一次需要打印输出结果的地方，都需要用 `print()` 函数来输出结果。

5.2 文本编辑器

上面介绍的是在 spyder 中编辑并运行 python 程序，有时候，我们可能想在其他文本编辑器中编辑 python 程序。有很多的文本编辑器可以用来编写 python 程序，作为刚开始接触的同学，可以使用 notepad++ 或者 sublime text 来编写 python 程序。这两个文本编辑器都是免费可以使用的，并且程序不大。

上述 “hello.py” 文件，在 notepad++ 中打开的界面如下：

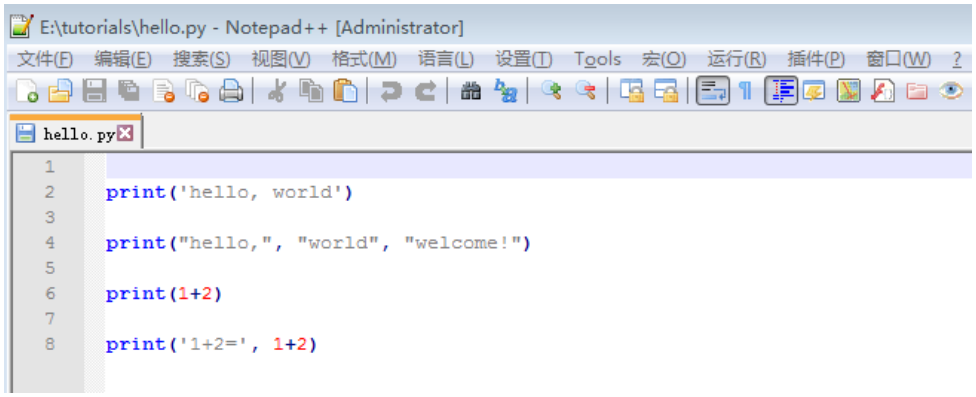


Figure 5.9: demo

关于这两个编辑器的使用，可以自行查找下。

5.3 创建和运行 “.ipynb” 文件

“.ipynb” 文件是使用 Jupyter Notebook 来编写 Python 程序时的文件。

Jupyter Notebook（此前被称为 IPython notebook）是一个交互式笔记本，支持运行 40 多种编程语言。它会在浏览器中打开并运行相关程序，在这里，我们主要介绍其在编写和运行 Python 程序方面的应用。

在安装好 Anaconda 后，已经自动安装好了 Jupyter Notebook，如下所示（红色数字 4 所指）：

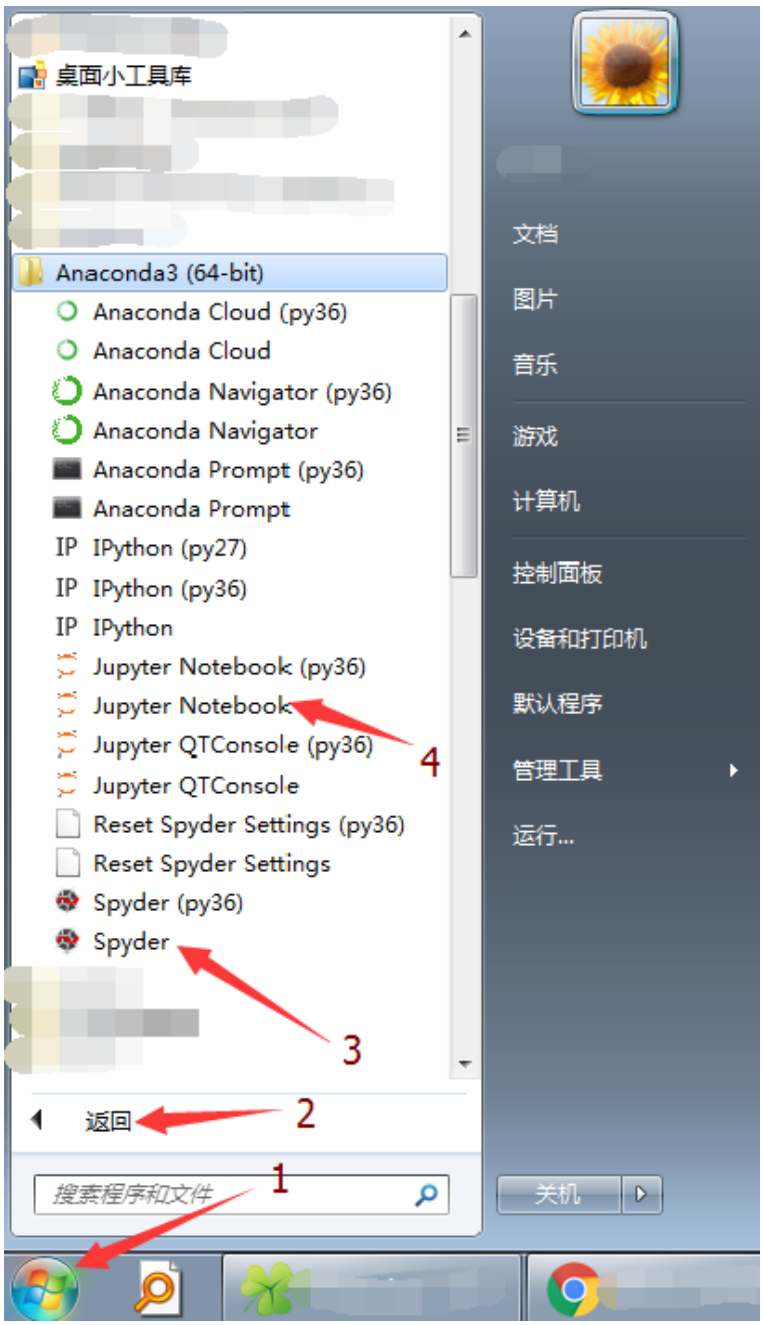


Figure 5.10: demo

点击 “Jupyter Notebook”，可以得到如下启动的命令行界面，以及在浏览器中打开的 Jupyter Notebook 界面。

命令行界面：

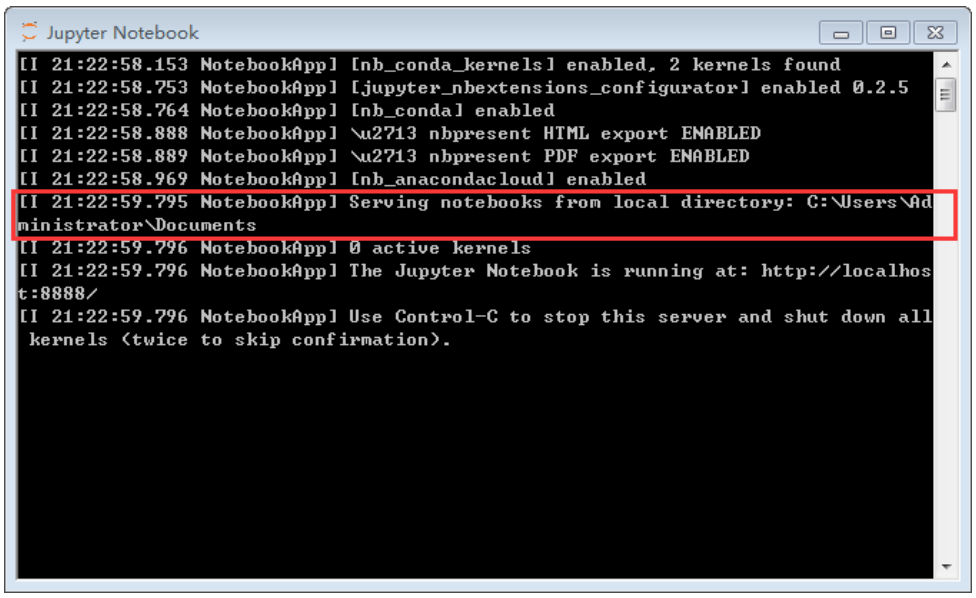


Figure 5.11: demo

Jupyter Notebook 界面：

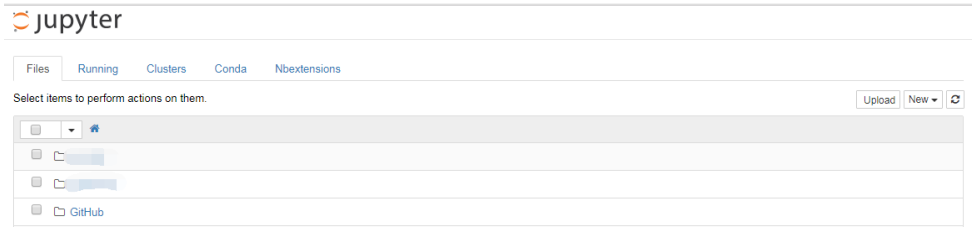


Figure 5.12: demo

从上面可看出，默认的启动界面是以 Jupyter notebook 的安装路径为初始路径的（即从 C 盘启动）。

如果我们希望在需要运行的程序所在文件夹启动，该如何操作呢?

以上述 E 盘 “tutorials”文件夹为例，如果我们希望在该文件夹下创建新的 “.ipynb” 文件或者运行该文件夹下的 “.ipynb” 文件，我们可以先打开 E 盘 “tutorials” 文件夹，然后按下 “Shift”键同时点击右键，得到如下界面：

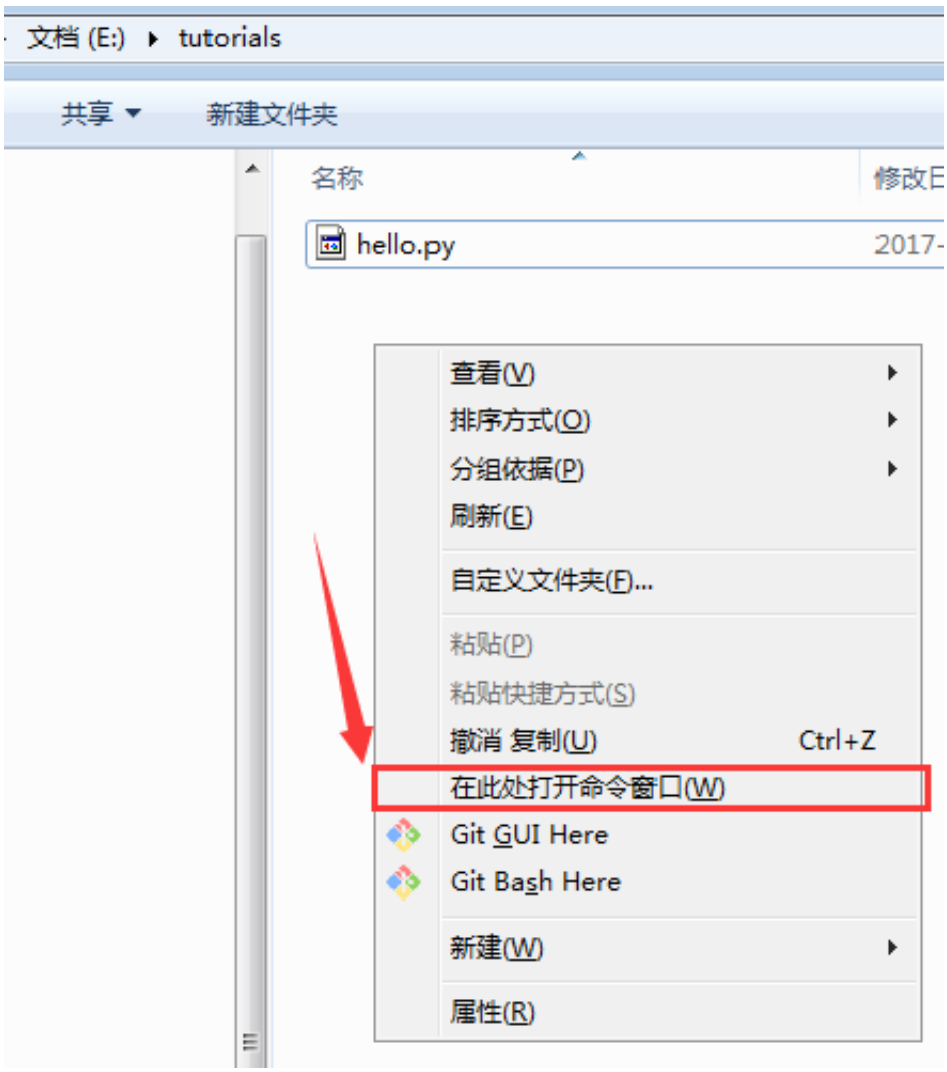


Figure 5.13: demo

点击 “在此处打开命令窗口 (W)”，在弹出的命令行窗口中输入 “jupyter notebook”，在浏览器中得到当前路径下的 jupyter notebook 文件列表，如下：

命令行界面：

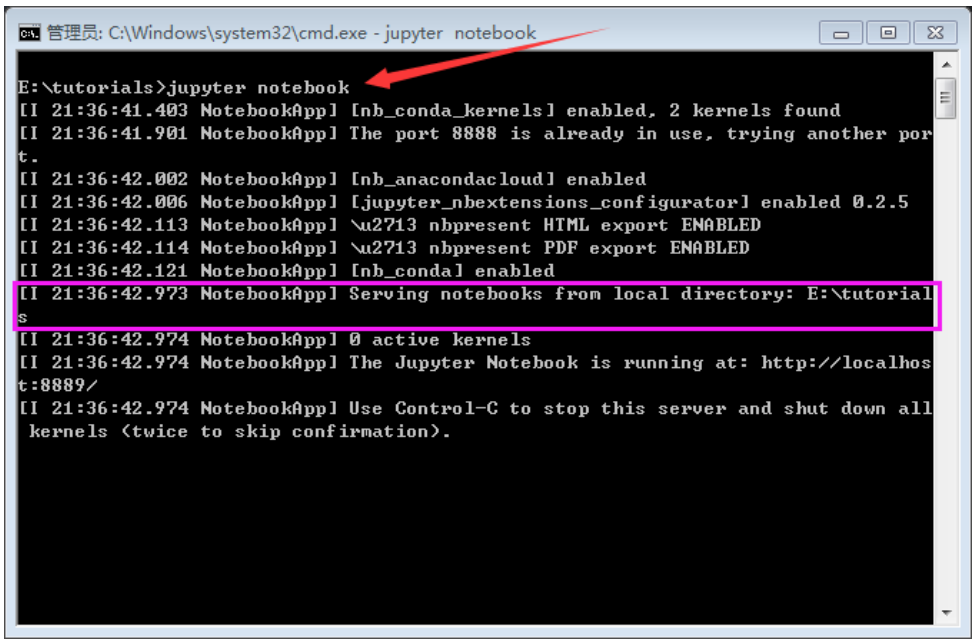


Figure 5.14: demo

Jupyter Notebook 界面：

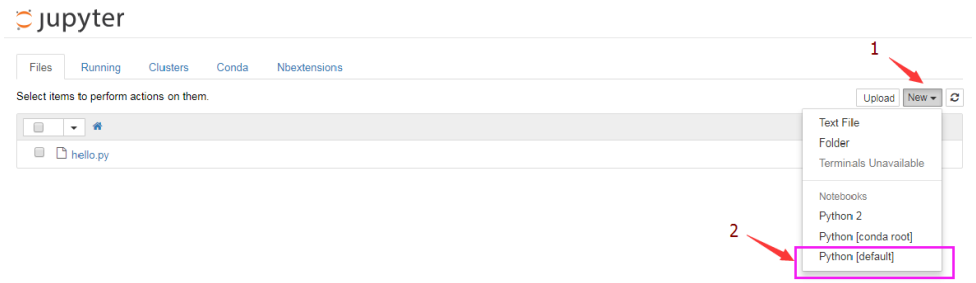


Figure 5.15: demo

请注意，**命令行界面不能关闭**，它是 Jupyter notebook 运行必须要有的环境。

点击上面的 “New”，可以开始创建新的 “.ipynb” 文件。

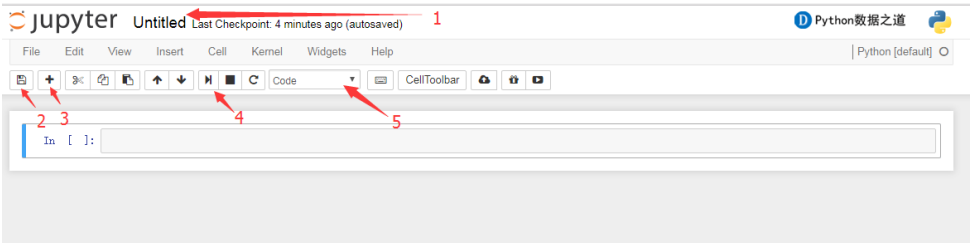


Figure 5.16: demo

在上述新打开的界面中：

- 红色数字 1 处，可以修改文件名称，比如我们修改为 “hello”(不含引号)
- 红色数字 2 处，点击保存文件
- 红色数字 3 处，点击添加新的代码行
- 红色数字 4 处，点击运行代码
- 红色数字 5 处，选择当前是编写代码还是以其他格式进行编写（比如 Markdown），默认是编写代码模式，即 “code”。

现在，我们输入之前在 “hello.py” 文件中的代码，每次输入一行代码，然后依次运行（点击红色数字 4 处的符号，或者按快捷键 “Shift”+“Enter”），运行结果如下：

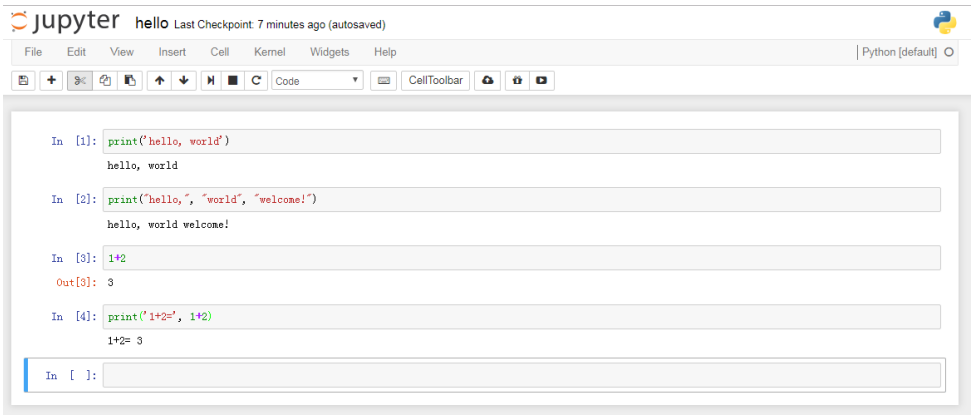


Figure 5.17: demo

此时，我们可以在 E 盘 “tutorials”文件夹下看到有一个名称为 “hello.ipynb” 的文件，或者在 Jupyter notebook 的首页下，也可以看到该文件，如下：

Figure 5.18: demo

5.4 总结

前面我们分别介绍了 “.py” 文件和 “.ipynb” 文件的创建及运行方法，在以后的章节中，我们会大量涉及到这两类文件的使用。

关于 Jupyter notebook，这里也作了简单的介绍。需要说明的是，Jupyter notebook 是一个强大的工具，在 Python 的数据科学领域会频繁的使用，在后续也会进一步介绍。

6 VS Code 中 Python 环境配置指南

之前在公众号发了关于 PyCharm 与 VS Code 对比的文章，大家也是很有感慨。

鉴于 PyCharm 是收费的工具，不少同学在想方设法寻找破解之法。

而 VS Code 是免费开源的工具，并且是微软旗下产品，如果能够媲美 PyCharm 的话，也不失为一个良好的选择。

VS Code 是一个相当优秀的 IDE ，具备开源、跨平台、模块化、插件丰富、启动时间快、颜值高、可高度定制等等优秀的特质，最近几年流行度也是越来越高。

今天来跟大家分享下在 VS Code 下配置和使用 Python，作为对 PyCharm 的替代选择之一。

6.1 软件准备

既然是在 VS Code 中进行 Python 环境配置，在配置之前需要安装好 Python 和 VS Code。

6.1.1 Python 安装

关于 Python 的安装，可以从 Python 官网下载 Python 进行安装，也可以通过 Anaconda 来安装，Lemon 建议直接安装 Anaconda 。

Anaconda 是一个开源的 Python 发行版本，其包含了 conda、Python 等 100 多个科学包及其依赖项。

Anaconda 官方网站下载地址如下：

https://www.anaconda.com/products/individual

在 Lemon 编写此文时，Anaconda 对应的 Python 版本是 Python 3.8，大家可以下载跟电脑系统对应的版本进行安装。

Figure 6.1: Anaconda

6.1.2 VS Code 安装

关于 VS Code 安装，在微软官方网站：https://code.visualstudio.com/ 提供免费下载，VS Code 支持 Windows , Linux , Mac OS 三种操作系统，建议去官方下载，不要在第三方软件下载网站下载，第三方软件下载站的猫腻太多了。

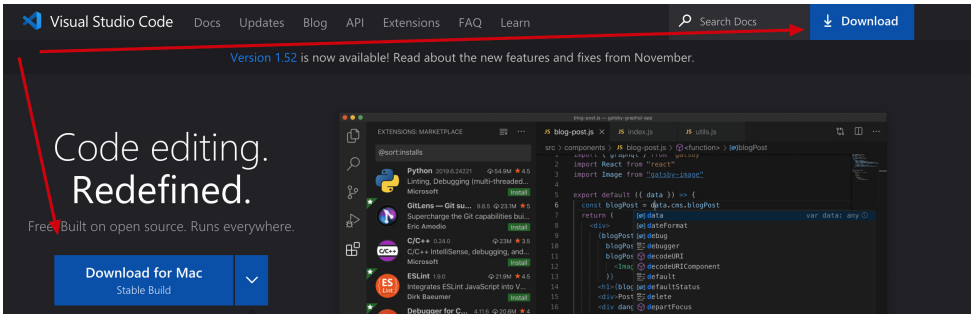


Figure 6.2: VS Code 官网

6.2 在终端运行程序

在没有安装其他 Python 插件的情况下，可以在 VS Code 中通过使用终端（Terminal）直接运行 python 代码，使用这种方式，可以省去配置过程。

在 VS Code 菜单栏选择 “view - Terminal”（中文菜单：查看—终端），或者直接快捷键「Ctrl + ‘ 」打开终端，会在下方产生一个 CMD 控制台。

可以在终端中通过输入 `python xxx.py` 来运行程序，效果如下：

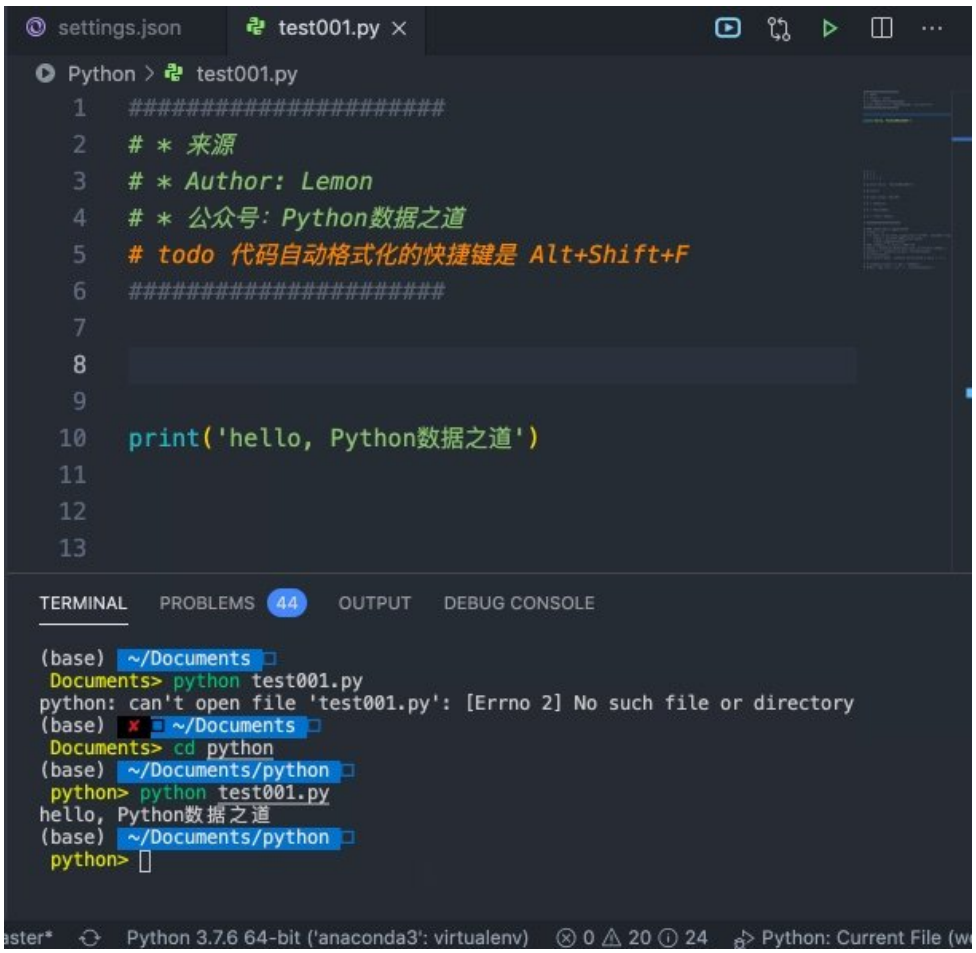


Figure 6.3: img23

动态效果参见：

神器 VS Code，超详细 Python 配置使用指南

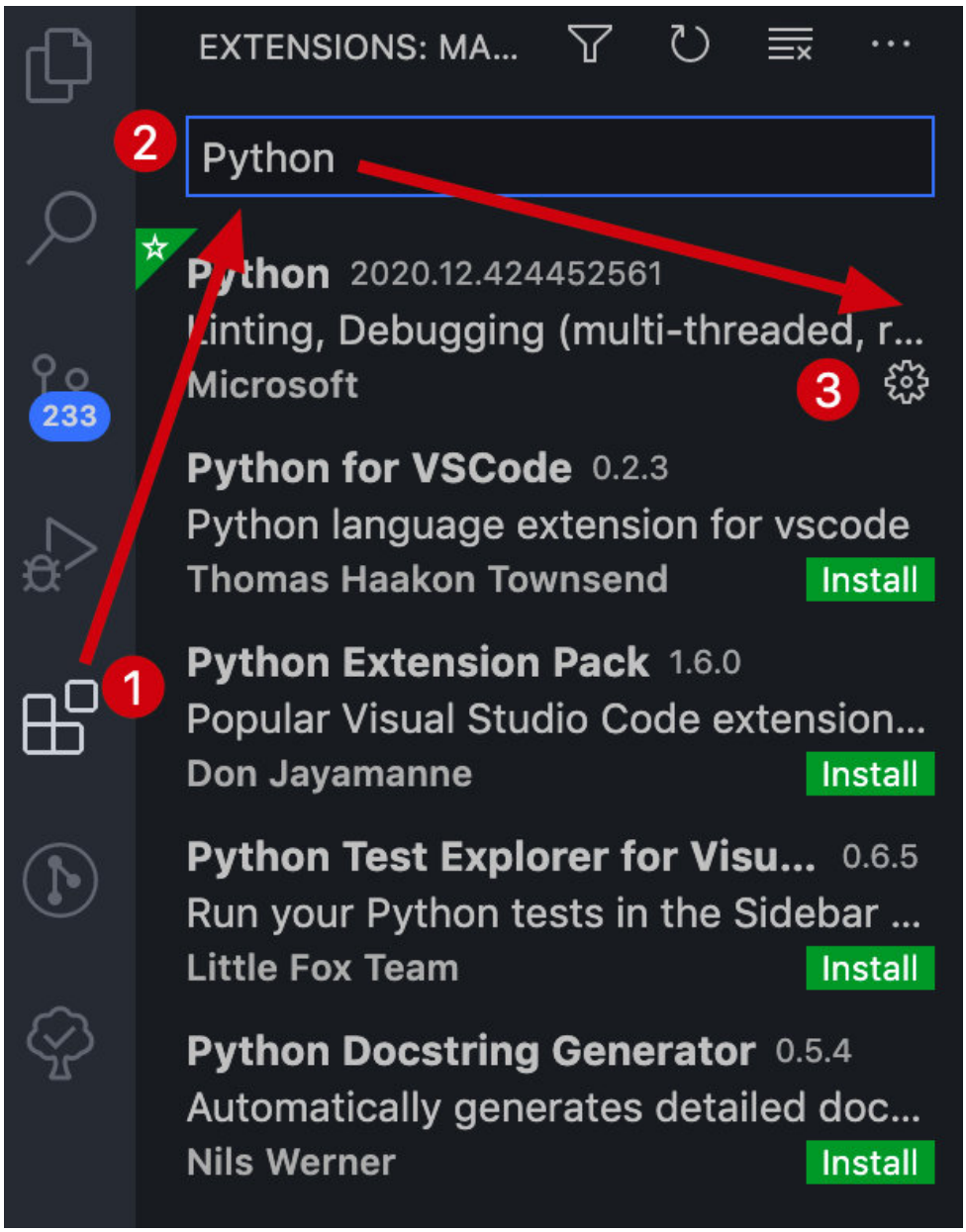
需要注意一下的是，当前的文件夹是否和 Python 脚本文件处于同一个目录，如果不在同一个目录则要 cd 进去。

6.3 Python 环境配置

6.3.1 Python 插件

在 VS Code 中配置 Python 环境，第一个要安装的插件，就是 `Python` 插件。

`Python` 插件的全称是 `Python extension for Visual Studio Code`，它是一个 Visual Studio Code 扩展，具有对 Python 语言的丰富支持（对于该语言的所有受支持版本：>=3.6），包括诸如 IntelliSense，linting，调试，代码导航，代码格式，Jupyter Notebook 支持，重构，变量之类的功能资源管理器，测试资源管理器，代码段等等！

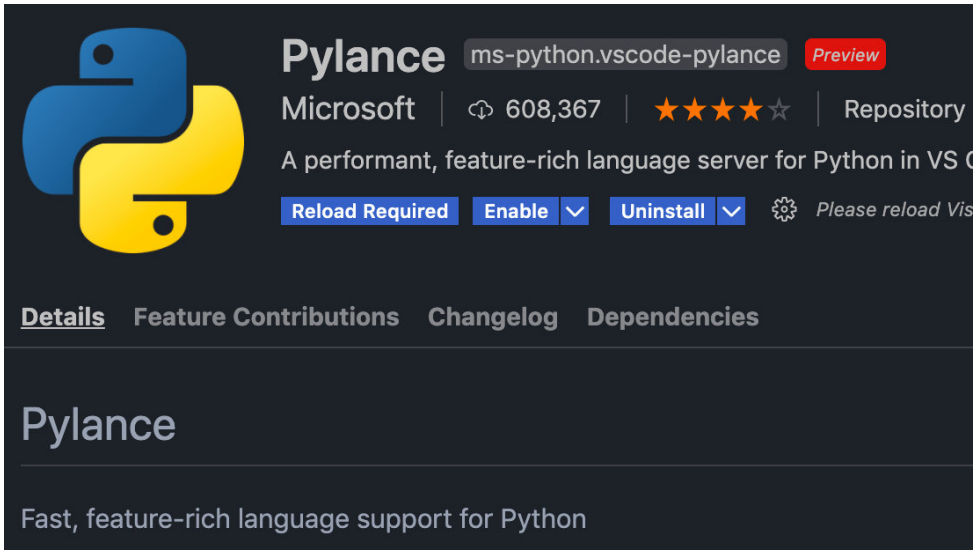

EXTENSIONS: MA...
2
Python
Python 2020.12.424452561
Linting, Debugging (multi-threaded, r...
Microsoft
3
233
Python for VSCode 0.2.3
Python language extension for vscode
Thomas Haakon Townsend
Install
1
Python Extension Pack 1.6.0
Popular Visual Studio Code extension...
Don Jayamanne
Install
Python Test Explorer for Visu... 0.6.5
Run your Python tests in the Sidebar ...
Little Fox Team
Install
Python Docstring Generator 0.5.4
Automatically generates detailed doc...
Nils Werner
Install

Python 插件对 Jupyter Notebook 的支持建立在安装 Jupyter 插件的基础上，这个在本文后续会提到。

6.3.2 代码补齐

在 VS Code 插件市场中有多个代码补齐的工具，这里 Lemon 还是推荐 Pylance 。

Pylance 是微软推出的一个快速而且功能丰富的 Python 语言扩展。Pylance 于 2020 年 7 月发布，目前在 VS Code 市场中，安装量已有 60 多万，足见其关注度。



Pylance 主要有如下特性：

- Docstrings
- 参数提示
- 自动导入
- 代码补全
- 代码诊断
- 引用和跳转
- 代码大纲
- 类型检查
- 支持多工作区
- 带有类型信息的签名帮助
- 兼容 IntelliCode 和 Jupyter notebook

安装后，代码输入效果如下：

Figure 6.4: img7

动态效果参见：

神器 VS Code，超详细 Python 配置使用指南

需要运行 Python 程序时，可以在程序文件（py文件）里右键，选择 `Run Python File in Terminal` 来运行程序，或者点击右上角的绿色三角形来运行程序，演示如下：

Figure 6.5: img17

动态效果参见：

神器 VS Code，超详细 Python 配置使用指南

6.3.3 自动格式化代码

Yapf 是谷歌开源的一个用于格式化 Python 代码的工具，可以一键美化代码。支持两种规范：PEP8 和 Google Style 。

在 VS Code 中使用 yapf 需要先安装该工具，步骤如下：

step 1：打开终端, 输入 “pip install yapf”, 安装 yapf ，如下：

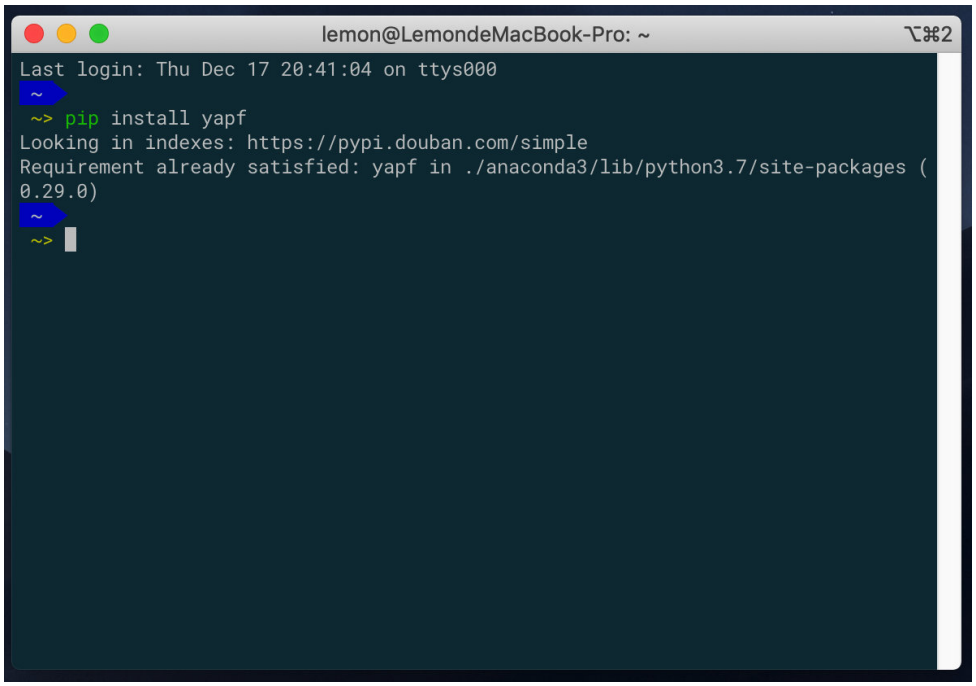
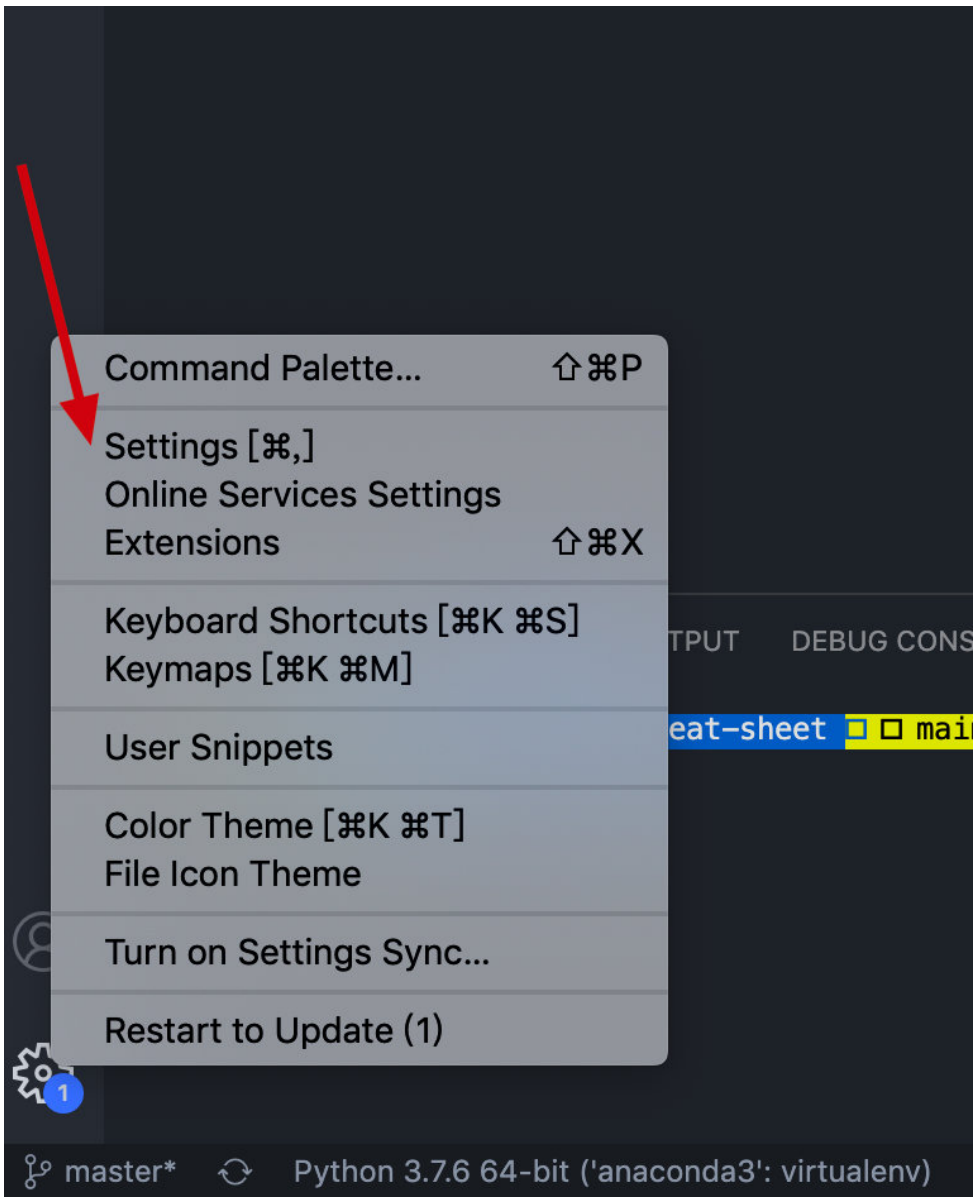


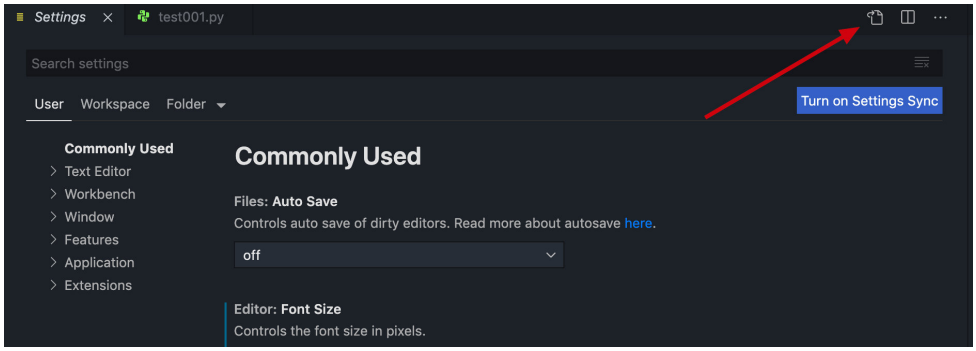
Figure 6.6: 在终端安装 yapf

step 2：在设置中设置使用 yapf，具体设置过程如下：

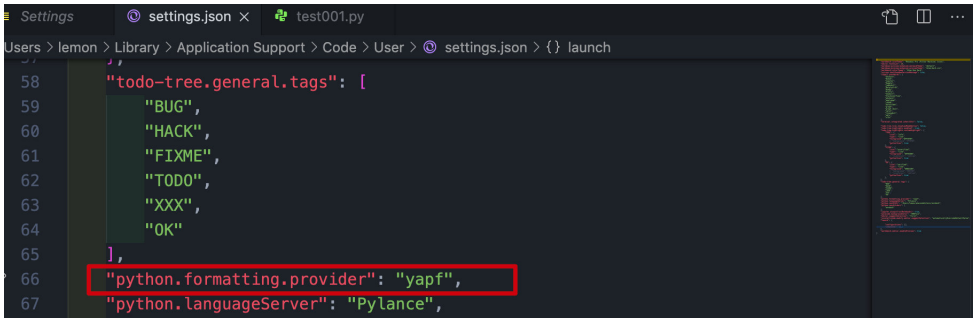
在 VS Code 左下角点击设置按钮，选择 “Settings”，如下：



在下面界面的右上角点击箭头所示，转到 `settings.json`，



在 `settings.json` 里进行设置 “python.formatting.provider”: “yapf”，图示如下：



step 3：完成上述设置后，就可以对代码格式进行自动格式化了，快捷键是 Alt+Shift+F 。

下面是代码自动格式化演示的效果：

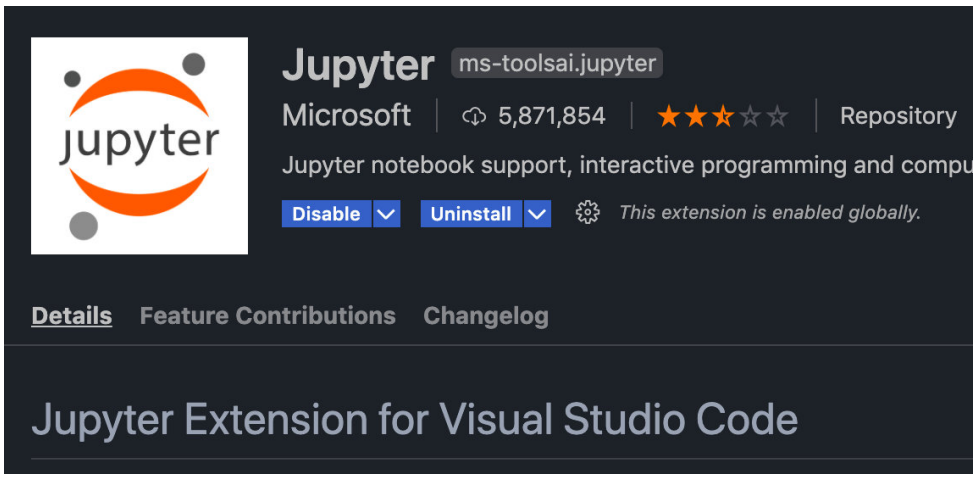
Figure 6.7: img22

动态效果参见：

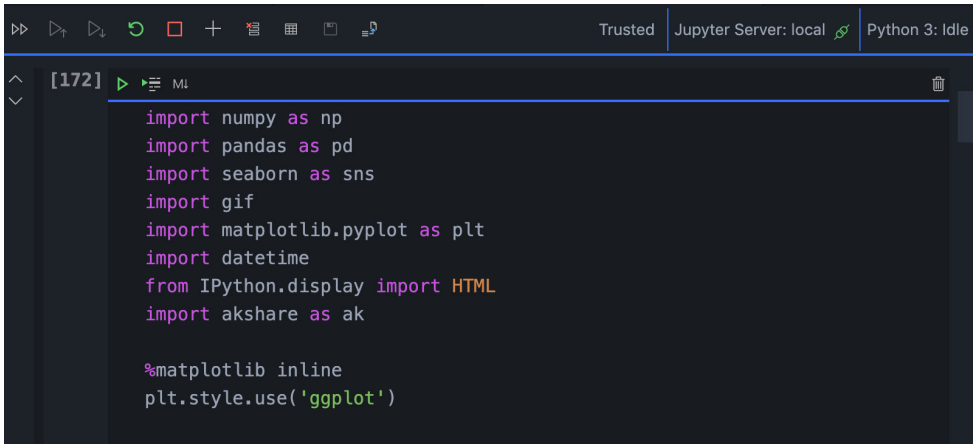
神器 VS Code，超详细 Python 配置使用指南

6.3.4 Jupyter Notebook 支持

众所周知，python 代码，除了通常使用的 `.py` 文件外，对于数据科学而言，Jupyter Notebook 也是大家的重要选项，因此，VS Code 中也提供了 Jupyter Notebook 的支持插件，在 VS Code 扩展市场中的名称是 `Jupyter`，如下：

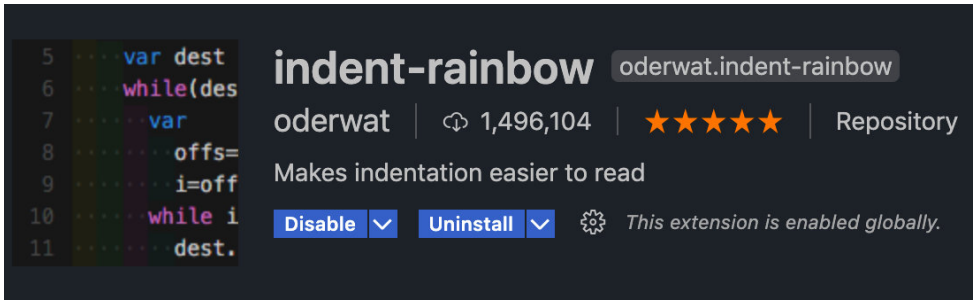


在安装 `Jupyter` 插件之后，Jupyter Notebook 的 `ipynb` 文件打开后的效果如下：



6.3.5 高亮缩进

当代码缩进层次较多时，为了视觉上容易识别，一般希望可以多个缩进以不同颜色进行高亮显示，VS Code 中 `indent-rainbow` 插件可以实现这个功能。



在安装之前，代码的效果如下：

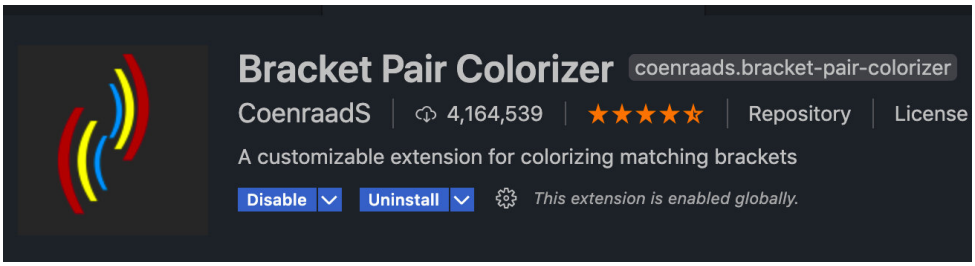
```
Python > test001.py
@gif.frame
def plot(df,date):
    df = df.loc[df.index[0]:pd.Timestamp(date)]
    fig, (ax1) = plt.subplots(1,figsize=(20,10),dpi=100)
    ax1.plot(
        df.close, marker='o',
        linestyle='--',
        linewidth=3,
        markersize=15,
        color = 'tab:blue'
        )
    y_max = round(df.close.max()*1.05)
    y_min = round(df.close.min()*0.95)
    ax1.set_title('沪深300指数2015年至2020年月均指数点位',fontsize=30)
```

在安装 indent-rainbow 之后，代码的效果如下：

```
@gif.frame
def plot(df,date):
    df = df.loc[df.index[0]:pd.Timestamp(date)]
    fig, (ax1) = plt.subplots(1,figsize=(20,10),dpi=100)
    ax1.plot(
        df.close, marker='o',
        linestyle='--',
        linewidth=3,
        markersize=15,
        color = 'tab:blue'
        )
    y_max = round(df.close.max()*1.05)
    y_min = round(df.close.min()*0.95)
    ax1.set_title('沪深300指数2015年至2020年月均指数点位',fontsize=30)
#     ax1.set_title('average close price of Hushen 300 index durin
    ax1.set_xlim([START,END])
```

6.3.6 括号高亮

当代码层次很多时，除了缩进的效果外，还有一个经常遇到的情况，就是各类括号，包括 ()、[]、{}；VS Code 中 Bracket Pair Colorizer 插件可以列出来括号高亮的效果，尤其是多组嵌套括号。



在安装 Bracket Pair Colorizer 之后，代码的效果如下：

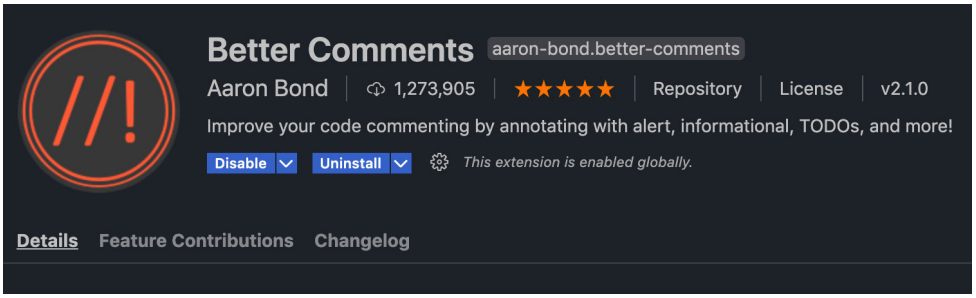
```
######################
# Author: Lemon
# 公众号: Python数据之道
######################

layout = html.Div([
        html.Div([Header(app)]),

        # page
        html.Div([
            # 估值数据表格
            html.Div([
                html.H1("估值数据表格",
                    style={
                        'textAlign': 'center',
                        'color': 'white',
                        'backgroundColor': '#0
                        'fontWeight': 'bold',
                    }
                    ),
```

6.3.7 注释高亮

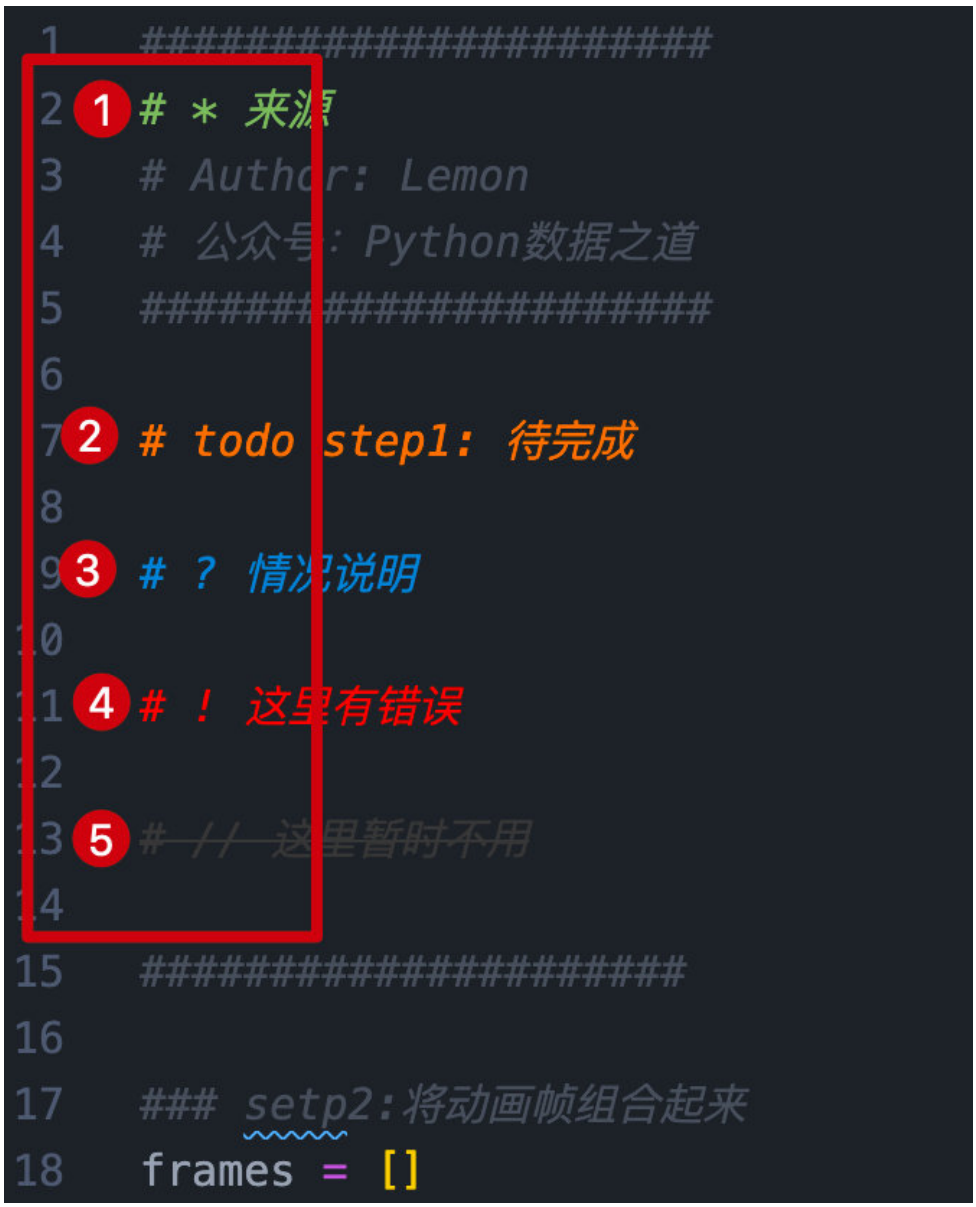
Lemon 在这里还推荐一个插件，这个插件用于程序语言编写时对注释的高亮。

咱们在编写程序的时候，总会涉及一些需要注意或者后续继续完成，或者需要提醒团队其他人员的注释内容，因此，有必要在代码中进行不同的说明，这个时候，在注释中进行不同的高亮，就会有很好的效果。

VS Code 扩展市场中，提供了 `Better Comments` 等类似的插件，来高亮注释。



在 `Better Comments` 插件中，有 5 中类型的注释高亮，分别用符号 `*` `?` `!` `//` `todo` 来区分，示例效果如下：



VS Code 中还有许多优秀的插件，欢迎大家来留言区推荐你最喜欢的插件或编辑工具！

7 Python 数据类型-List 介绍

本文的运行环境：

- windows7 系统
- python3.5
- jupyter notebook

7.1 初识 list

list 是 Python 内置的一种高级数据类型。list 是一种有序的集合，在 python 中应用很广泛。

```
names = ['James', 'Michael', 'Emma', 'Emily']

print("names的数据类型：",type(names))
print(names)
```

输出结果:

```
names的数据类型：  <class 'list'>
['James', 'Michael', 'Emma', 'Emily']
```

用 `len()` 可获得 list 的长度，即 list 集合所包含的元素个数，如下:

```
n = len(names)
print(n)
```

输出结果:

```
4
```

空的 list

如果 list 中一个元素也没有，我们可以定义一个空的 list ，则其长度为 0 ，如下:

```
empty_list = []
empty_list
```

输出结果:

```
[]
```

```
len(empty_list)
```

输出结果:

```
0
```

7.2 访问列表中的值

用索引来访问 list 中的每个元素，请注意索引是从 0 开始，最后一个的索引编号为 n-1，即所有元素的编号依次为(0,1, 2, ..., n-1)。

list 中单个元素的访问如下:

```
names[0]
```

输出结果:

```
'James'
```

```
names[1]
```

输出结果:

```
'Michael'
```

如果要获取最后一个元素，也可以用 -1 来做索引，倒数第二个元素可以用 -2 来索引，其他以此类推。

```
names[-1]
```

输出结果:

```
'Emily'
```

```
names[-2]
```

输出结果:

```
'Emma'
```

可以通过 for 循环来列出所有元素。有两种方法可以实现，如下:

方法 1

```
for name in names:
    print(name)
```

输出结果:

```
    James
```

```
    Michael
    Emma
    Emily
```

方法 2

```
for i in range(len(names)):
    print(names[i])
```

输出结果:

```
    James
    Michael
    Emma
    Emily
```

7.3 列表的操作、函数及方法

7.3.1 list 中元素的增、改、删等操作

list 是一个可变的有序列表，可以通过添加、修改、删除等操作来操作 list 中的元素。

往 list 中添加元素

可以通过 `append()` 和 `insert()` 方法来往 list 中添加元素。

其中，`append()` 方法是在 list 的末尾添加元素；`insert()` 是在指定位置添加元素。

如下:

```
names.append('Jacob')
names
```

输出结果:

```
    ['James', 'Michael', 'Emma', 'Emily', 'Jacob']
```

```
names.insert(1, 'Ava')
names
```

输出结果:

```
    ['James', 'Ava', 'Michael', 'Emma', 'Emily', 'Jacob']
```

请注意，上述 `insert()` 方法中，“1” 表示在第 2 个位置添加新的元素（索引是从 0 开始的）

删除 list 中的元素

- **用 `pop()` 方法删除 list 末尾的元素**

```
names.pop()
```

输出结果:

```
    'Jacob'
```

```
names
```

输出结果:

```
    ['James', 'Ava', 'Michael', 'Emma', 'Emily']
```

- **删除指定位置的元素，用 pop(i) 方法**

```
names.pop(0)
```

输出结果:

```
    'James'
```

```
names
```

输出结果:

```
    ['Ava', 'Michael', 'Emma', 'Emily']
```

修改 list 中的元素

如果需要修改 list 中的元素，可以直接通过 list 的索引进行赋值来实现，如下:

```
names[2] = 'Lemon'
names
```

输出结果:

```
    ['Ava', 'Michael', 'Lemon', 'Emily']
```

7.3.2 列表操作符

列表可以进行相加 “+” 和相乘 “*” 运算，“+” 相当于拼接列表，“*” 相当于重复列表。

此外，还可以判断元素是否存在于列表中。

```
# 列表相加
print("列表相加: ",[1,2,3]+['a','b'])

# 列表相乘
print("列表相乘: ",['a','b']*3)

# 判断元素是否存在于列表中
print("判断元素是否存在于列表中: ", 'a' in ['a', 'b'])
print("判断元素是否存在于列表中: ", 'a' not in ['a', 'b'])
```

输出结果:

```
列表相加： [1, 2, 3, 'a', 'b']
列表相乘： ['a', 'b', 'a', 'b', 'a', 'b']
判断元素是否存在于列表中： True
判断元素是否存在于列表中： False
```

7.3.3 列表函数 & 方法

列表的函数包括:

- len(list)，列表元素个数，即列表的长度
- max(list)，返回列表元素最大值
- min(list)，返回列表元素最小值
- list(sep)，将元组转为列表

列表的方法

列表的方法除了前面提到的增、改、删等方法外，还有其他一些方法，如下:

- list.count(obj)，统计某个元素在列表中出现的次数
- list.extend(seq)，在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
- list.index(obj)，从列表中找出某个值第一个匹配项的索引位置
- list.remove(obj)，移除列表中某个值的第一个匹配项
- list.sort()，对原列表进行排序
- list.reverse()，对原列表进行反向排序

```
list1 = [1,2,3,9,6,3]
list2 = [3, 8, 5, 4, 7,4]
tuple1 = (3,9,6)

# 列表的函数
print("列表最大值： ",max(list1))
print("列表最小值： ",min(list1))
print("将元组转为列表： ",list(tuple1))

# 列表的方法
print("count： ",list1.count(3))

list1.extend(list2)
print("extend： ",list1)

print("index： ",list2.index(4))

list2.remove(4)
print("remove： ",list2)

list2.sort()
print("sort： ",list2)
```

```
23
24 list2.reverse()
25 print("reverse: ",list2)
```

输出结果:

```
    列表最大值:  9
    列表最小值:  1
    将元组转为列表:  [3, 9, 6]
    count:  2
    extend:  [1, 2, 3, 9, 6, 3, 3, 8, 5, 4, 7, 4]
    index:  3
    remove:  [3, 8, 5, 7, 4]
    sort:  [3, 4, 5, 7, 8]
    reverse:  [8, 7, 5, 4, 3]
```

7.4 list 中元素的类型可以多样

同一个 list 中的元素的类型可以是字符串（str)、整型（int)、布尔型（Boolean)、以及嵌套的 list 等，举例如下:

```
a_list = ['Lemon', 100, ['a', 'b', 'c', 'd'], True]
a_list
```

输出结果:

```
    ['Lemon', 100, ['a', 'b', 'c', 'd'], True]
```

```
a_list[0]
```

输出结果:

```
    'Lemon'
```

```
a_list[2]
```

输出结果:

```
    ['a', 'b', 'c', 'd']
```

在上述的 a_list 中，第 3 个元素 (即 alist[2]) 其实本身也是一个 list。

list 中的元素也可以是 list，这样的话，可以进行多重 list 的嵌套。

上述 list，也可以按下述方式来理解。

```
b_list = ['a', 'b', 'c', 'd']
a_list = ['Lemon', 100, b_list, True]
a_list
```

输出结果:

```
['Lemon', 100, ['a', 'b', 'c', 'd'], True]
```

针对 a_list ，如果想获取其中元素 b_list 里面的单个元素 “b”，该如何实现呢?

其实，这个类似二维数组，用二维的索引可以获取，如下：

```
item_b = a_list[2][1]
item_b
```

输出结果:

```
'b'
```

上述 `[2]` 表示获取 a_list 的第 3 个元素，即 b_list，`[1]` 表示获取 `b_list` 的第 2 个元素，即“b”

7.5 list 的切片（slices）

前面描述了 list 中单个元素如何获取，如果想获取其中连续的部分元素，该如何实现呢。

这里可以通过切片 (slices) 的形式来获取部分连续的元素。

```
c_list = ['James', 'Ava', 'Michael', 'Emma', 'Emily', 'Jacob']
c_list
```

输出结果:

```
['James', 'Ava', 'Michael', 'Emma', 'Emily', 'Jacob']
```

list 中以切片形式使用时，其结构可参考 `new_list[start: end : step]`

其中 “start” 和 “end” 表示索引位置的开始和结束，选取的元素包含 “start”，**但不包含 “end”**。

“step” 表示步长，默认情况下，“step”为 1，演示如下：

Example-1:

```
c_list[1:3]
```

输出结果:

```
['Ava', 'Michael']
```

example-1 中，`1:3` 表示切片选取的是第 2 个元素和第 3 个元素，即包含索引为 1 和索引为 2 的元素。相当于获取 c_list[1] 和 c_list[2]

Example-2:

```
c_list[::2]
```

输出结果:

```
['James', 'Michael', 'Emily']
```

example-2 中 “start” 和 “end” 为空的时候，默认是全选，即 “start” 为 0，“end” 为 `len(c_list)-1`。

所以 `c_list[::2]` 表示的是从索引为 0 开始，以步长为 2 来选择元素。

接下来看以下结合上述两个例子的演示：

```
c_list[1:3:2]
```

输出结果:

```
['Ava']
```

Example-3:

前面提到，new_list[-1] 表示获取最后一个元素，在切片的步长 “step” 中，也可以是负数，比如 “-1”:

```
c_list[::-1]
```

输出结果:

```
['Jacob', 'Emily', 'Emma', 'Michael', 'Ava', 'James']
```

从上述结果可以看出，当 “step”为 `-1` 时，我们发现是将列表进行了逆序排序。

再看看步长为 “-2” 时的结果：

```
c_list[::-2]
```

输出结果:

```
['Jacob', 'Emma', 'Ava']
```

可以这么理解，当步长为**正数**时，是**从左到右以该步长**来获取列表中的元素；

而当步长为**负数**时，是**从右到左以该步长的绝对值**来获取列表中的元素。

Example-4:

如果想获取离散的元素，比如想获得第 1、2、4 个元素，能不能通过离散的索引值来获取呢?

我们先来试验一下：

```
c_list[0, 1, 3]
```

```
out:
TypeError Traceback (most recent call last) in () —-> 1 c_list[0, 1, 3]
TypeError: list indices must be integers or slices, not tuple
```

从运行结果可以之道，直接取离散的索引号是不行的，它提示 list 的索引必须是整数或者 slices。

那么，我们有没有方法来获取这些离散的元素呢?

方法肯定是有的，其中一种方法就是使用列表推导式。

7.6 列表推导式 (List Comprehension)

7.6.1 列表推导式的一般情况

列表推导式的一般语法结构:

```
new_list = [x for x in iterable]
```

其中的 iterable 表示可迭代的对象，包括字符串（str)、列表（list)，元组（tuple)、字典（dict)、集合（set)，以及生成器（generator）等。

先来看几个简单的例子:

```
str_list = [x.lower() for x in "Lemon"]
str_list
```

输出结果:

```
    ['l', 'e', 'm', 'o', 'n']
```

```
list_list = [x**2 for x in [1,2,3,4]]
list_list
```

输出结果:

```
    [1, 4, 9, 16]
```

```
tuple_list = [x+2 for x in (1,2,3,4)]
tuple_list
```

输出结果:

```
    [3, 4, 5, 6]
```

```
ge_list = [x for x in range(8)]
ge_list
```

输出结果:

```
    [0, 1, 2, 3, 4, 5, 6, 7]
```

7.6.2 两层 for 循环的列表推导式

列表推导式中，可以同时包含多个 for 循环，比如同时两个 for 循环，如下:

```
[x**2+y for x in range(5) for y in range(4,7)]
```

输出结果:

```
    [4, 5, 6, 5, 6, 7, 8, 9, 10, 13, 14, 15, 20, 21, 22]
```

上述结果跟下面的写法结果是一致的:

```
two_for_list = []

for x in range(5):
    for y in range(4,7):
        two_for_list.append(x**2+y)
print(two_for_list)
```

输出结果:

```
    [4, 5, 6, 5, 6, 7, 8, 9, 10, 13, 14, 15, 20, 21, 22]
```

列表推导式中，三层或者三层以上的 for 循环一般很少用到。

7.6.3 使用两个变量来生成 list

列表推导式也可以使用两个或多个变量来生成 list，结合字典的使用，举例如下:

```
d = {'x': '1', 'y':'2', 'z':'4'}
d_list = [k+'='+v for k,v in d.items()]
d_list
```

输出结果:

```
    ['z=4', 'y=2', 'x=1']
```

7.6.4 含 if 语句的列表推导式

列表推导式中还可以引入 if 条件语句，如下:

```
if_list = [x**2 for x in range(10) if x%2==0]
if_list
```

输出结果:

```
    [0, 4, 16, 36, 64]
```

上述列表推导式，如果用普通的 for 循环来编写的话，内容如下:

```
if_list_1 = []
for x in range(10):
    if x%2==0:
```

```
        if_list_1.append(x**2)

print(if_list_1)
```

输出结果:

```
    [0, 4, 16, 36, 64]
```

通过对比，可以看出列表推导式的方式更显得 Pythonic。

当然，如果仅仅是编写上更简洁，可能不一定能显现出列表推导式的优势，下面我们来对比下上述两种方式的运行效率。

```
%%timeit

if_list = [x**2 for x in range(10) if x%2==0]
if_list
```

输出结果:

```
    100000 loops, best of 3: 6.06 µs per loop
```

```
%%timeit

def for_loop():
    if_list_1 = []
    for x in range(10):
        if x%2==0:
            if_list_1.append(x**2)
    return if_list_1

for_loop()
```

输出结果:

```
    100000 loops, best of 3: 6.76 µs per loop
```

从上面的运行结果进行对比，可以看出列表推导式的运行效率要高于普通的 for 循环编写方式。

因为，建议大家使用列表推导式，不仅写法更 Pythonic，且运行效率更高。

包含 if-else 语句的列表推导式

```
[x**2 if x%2==0 else x+2 for x in range(10)]
```

输出结果:

```
    [0, 3, 4, 5, 16, 7, 36, 9, 64, 11]
```

包含两个 if 语句的列表推导式

```
[x**2 for x in range(10) if x%2==0 if x%3==0]
```

输出结果:

```
    [0, 36]
```

7.7 小结

如果想获取离散的元素，比如想获得第 1、2、4 个元素，能不能通过离散的索引值来获取呢?

前面已经实践过，直接通过离散的索引值来获取列表中的元素是不行的。

通过列表推导式的学习，我们可以换一种思路，来实现列表中离散的元素的获取，如下:

```
[c_list[i] for i in [0,1,3]]
```

输出结果:

```
    ['James', 'Ava', 'Emma']
```

列表（list）作为 python 最基础也是最重要的数据类型之一，在 python 数据分析以及其他用途中有着重要的作用，希望上述内容能对于熟悉 list 有所帮助。

8 Python 中字典合并的实现方法

8.1 背景

Python 中有时候需要将字典进行合并，今天我们来探讨下这个主题。

先来看看下面的运行结果

```
x = {'a':1, 'b':2}
y = {'b':4, 'c':5}
z = x.update(y)
```

来看看 z 的返回结果：

```
print(z)
```

```
None
```

发现 z 返回的是 None 值，并不是我们想要的结果。

再来看一下此时 x 的返回结果，发现 x 已经是合并的结果。

```
x
```

```
{'a': 1, 'b': 4, 'c': 5}
```

那么有没有比较简单的实现方法呢。下面我们来探索一下：

8.2 Method1：适用于 Python3.5 及以上版本呢

在 Python3.5 以上版本（含 3.5），可以通过下列方法实现

```
z1 = {**x, **y}
z1
```

```
{'a': 1, 'b': 4, 'c': 5}
```

8.3 Method2：自定义函数

如果在 Python2 或 Python3.4 以及低于 3.4 的版本，要实现两个 dict 的合并，该怎样做呢?

先定义一个函数，然后用自定义函数来运行

```
def merge(x,y):
    z = x.copy()
    z.update(y)
    return z
```

```
z2 = merge(x,y)
z2
```

```
{'a': 1, 'b': 4, 'c': 5}
```

Method2 在 Python3.5 以上版本也是可以运行的。

此外，请注意 x 中 “b”=2，y 中 “b”=4, 而运算结果中 “b”=4，是用 y 中’b’ 的值来对字典进行更新。

8.4 多个 dict 进行合并

如果是多个 dictionary 需要进行合并呢?

同样可以通过自定义的形式来实现。

```
def merge_dicts(*dict_args):
    result = {}
    for item in dict_args:
        result.update(item)
    return result
```

```
x1 = {'a':1, 'b':2}
y1 = {'b':4, 'c':5}
x2 = {'d':8, 'e':10}
```

```
z3 = merge_dicts(x1,y1,x2)
z3
```

```
{'a': 1, 'b': 4, 'c': 5, 'd': 8, 'e': 10}
```

此方法在 Python2 和 Python3 下都是可以运行的

9 Python 内置时间模块：Datetime

9.1 前言

在用 Python 进行数据处理，尤其是时间序列梳理的处理，经常会涉及处理时间或日期的地方，有些看似简单的问题，经常会混淆，甚至被困住。

虽然网上已有不少关于 Python 时间模块介绍的内容，但总体来说较为零散，因此 Lemon 对 Python 内置的时间模块来进行了整理，Python 内置时间处理模块主要包括：

- Datetime
- Time
- Calendar

概括来说，时间和日期处理的模块和库，有几个主要的概念都是会涉及的，包括：

`日期`，`时间`，`时间戳`，`时间和字符串的转换`，`时区` 等。

`Datetime`、`Time`、`Calendar` 这三个模块中，我们用的较多的是 `Datetime`，在这里我们介绍的也较为详细。

针对 `Datetime` 模块，从构建时间对象实例、时间转换、时间对象的运算三个方面进行了梳理，共涉及 13 个知识点，整理的大纲如下：

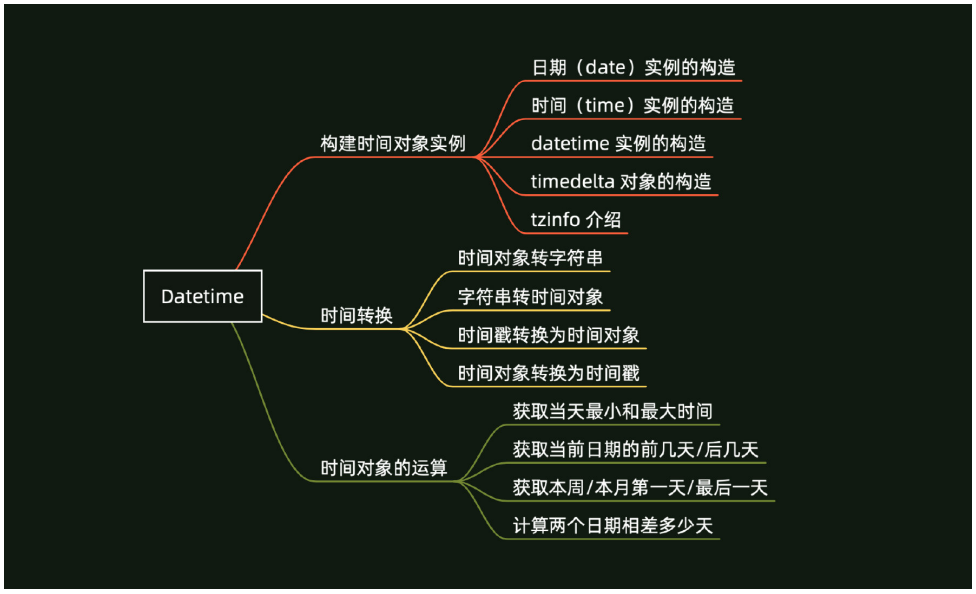


Figure 9.1: datetime

针对 Time 模块，从时间获取和时间转换两个方面进行了梳理，共涉及 6 个知识点，整理的大纲如下：

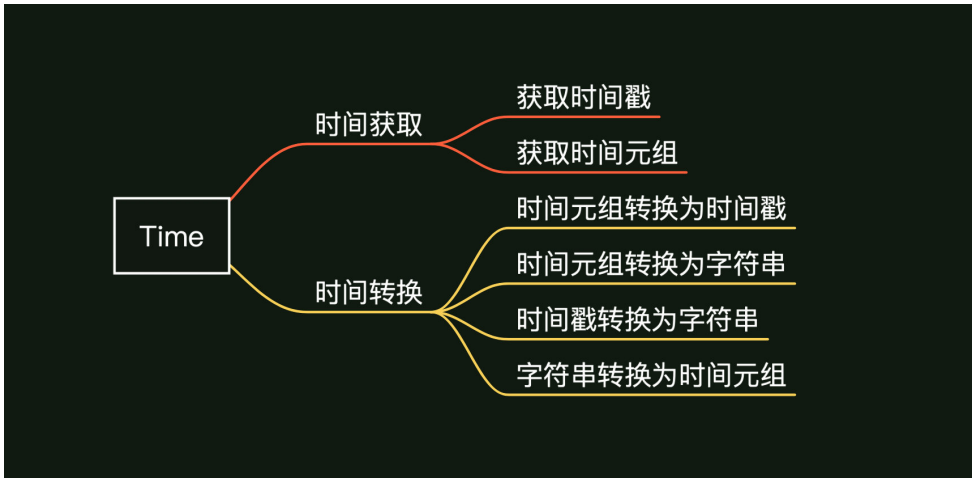


Figure 9.2: time

首先要介绍的，是大家平时用的较多的 datetime 模块, 它属于 Python 的内置模块，功能相对较全。

9.2 构建时间对象实例

```
import datetime
# from datetime import date, datetime,time,timedelta,tzinfo
import time
```

时间实例的构造包括日期（如 2020 年 10 月 12 日），时间（如 20 点 10 分 01 秒），或者是包含 date 和 time 的 datetime （如 2020 年 10 月 12 日 20 点 10 分 01 秒），下面 Lemon 来跟大家介绍下具体的构造过程。

9.2.1 日期（date）实例的构造

date 是一个理想化的简单型日期，属性有 year , month , day 。

```
# 构造日期 date 实例
d1 = datetime.date(2020,10,12)
d1
```

```
datetime.date(2020, 10, 12)
```

```
# 构造日期 date 实例
d1 = datetime.date(2020,10,12)
print(d1)
```

```
2020-10-12
```

除了上面的构造方式，在 date 实例化的时候，还可以通过 date.today() 来构造当天的日期。

```
datetime.date.today()
```

```
datetime.date(2020, 10, 14)
```

date 类型的日期，可以通过 .year , .month , .day 来获取日期所属的年份，月份，和具体的日期号，这几个方法在数据分析中经常会用到。

```
# 获取日期所属的 年份，月份，和具体的日期号

print(f'日期所属的年份为：{d1.year}')
print(f'日期所属的月份为：{d1.month}')
print(f'日期具体的日期号为：{d1.day}')
```

```
日期所属的年份为：2020
日期所属的月份为：10
日期具体的日期号为：12
```

9.2.2 时间 time 实例的构造

time 是一个独立于任何特定日期的理想化时间，其属性有 hour，minute，second，microsecond 和 tzinfo。

```
# 构造时间 time 实例
t1 = datetime.time(20,10,1)
t1
```

```
datetime.time(20, 10, 1)
```

```
# 获取日期所属的 年份，月份，和具体的日期号

print(f'time 所属的小时为：{t1.hour}')
print(f'time 所属的分钟为：{t1.minute}')
print(f'time 所属的秒为：{t1.second}')
```

```
time 所属的小时为：20
time 所属的分钟为：10
time 所属的秒为：1
```

9.2.3 datetime 实例的构造

datetime 是日期和时间的结合，其属性有 year，month，day，hour，minute，second，microsecond 和 tzinfo。

```
# 构造时间 datetime 实例
dt1 = datetime.datetime(2020,10,11,20,10,1)
```

```
dt1
```

```
datetime.datetime(2020, 10, 11, 20, 10, 1)
```

除了上面的构造方式，在 datetime 实例化的时候，还有其他的一些方式，包括使用 `datetime.now()` 和 `datetime.today()`，以及在 date 的基础上使用 `combine` 方法等。

```
dt2 = datetime.datetime.now()
dt2
```

```
datetime.datetime(2020, 10, 14, 15, 12, 20, 303269)
```

```
dt3 = datetime.datetime.today()
dt3
```

```
datetime.datetime(2020, 10, 14, 15, 12, 20, 308733)
```

```
dt4 = datetime.datetime.combine(d1,t1)
dt4
```

```
datetime.datetime(2020, 10, 12, 20, 10, 1)
```

通过 datetime 的实例化，是我们使用时间是经常用到的方法，在日常使用过程中，我们可能只需要具体到某天，或者只需要具体到某天的某个时间点，这时候，也可以通过 datetime 的一些方法来实现。

```
# 从 datetime 来获取日期
dt4.date()
```

```
datetime.date(2020, 10, 12)
```

```
# 从 datetime 来获取具体的时间点
dt4.time()
```

```
datetime.time(20, 10, 1)
```

同样的 `datetime` 类型的时间，可以通过 `.year`，`.month`，`.day` 来获取日期所属的年份，月份，和具体的日期号。

```
# 获取日期所属的 年份，月份，和具体的日期号

print(f'日期所属的年份为：{dt4.year}')
print(f'日期所属的月份为：{dt4.month}')
print(f'日期具体的日期号为：{dt4.day}')
```

```
日期所属的年份为：2020
日期所属的月份为：10
日期具体的日期号为：12
```

还有一个可能涉及到的时间是获取某天属于星期几，可以通过 `weekday()` 和 `isoweekday()` 方法来实现。

```
# 从 datetime 来获取日期是星期几
# 使用 weekday 方法
# 数字从0开始，0代表星期一，1代表星期二，以此类推
dt4.weekday()
```

```
0
```

```
# 从 datetime 来获取日期是星期几
# 使用 isoweekday 方法
# 数字从1开始，1代表星期一，2代表星期二，以此类推
dt4.isoweekday()
```

```
1
```

datetime 还有一种方法，在这里也值得介绍下，如果 datetime 的值由于某些原因弄错了，我们也可以通过 `replace()` 方法来进行更正。这个方法在进行数据清洗的时候会有用。

replace 方法的参数如下：

```
datetime.replace(year=self.year, month=self.month, day=self.day, hour=self.hour,
 minute=self.minute, second=self.second, microsecond=self.microsecond, tzinfo=self.
tzinfo, * fold=0)
```

```
dt5 = dt4.replace(year=2019)
dt5
```

```
datetime.datetime(2019, 10, 12, 20, 10, 1)
```

9.2.4 timedelta 对象的构造

timedelta 对象表示两个 date 或者 time 或者 datetime 的时间间隔。

```
class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0,
minutes=0, hours=0, weeks=0)
```

所有参数都是可选的并且默认为 0。这些参数可以是整数或者浮点数，也可以是正数或者负数。

只有 days, seconds 和 microseconds 会存储在内部。参数单位的换算规则如下：

- 1 毫秒会转换成 1000 微秒。
- 1 分钟会转换成 60 秒。
- 1 小时会转换成 3600 秒。
- 1 星期会转换成 7 天。

```
delta = datetime.timedelta(weeks=3,
                           days=10,
                           hours=6,
```

```
                    minutes=50,
                    seconds=30,
                    microseconds=1000,
                    milliseconds=10000,
                   )

delta
```

```
datetime.timedelta(days=31, seconds=24640, microseconds=1000)
```

9.2.5 tzinfo 介绍

datetime.tzinfo 返回 datetime 对象的时区，前提是在创建 datetime 对象时需传入 tzinfo 参数，如果没有传入则返回值为 None 。

```
# 如果没有 pytz 库，则需要自行安装
import pytz

sh = pytz.timezone('Asia/Shanghai')
d_tz = datetime.datetime(2020,10,12,hour=8,tzinfo=sh)

d_tz.tzinfo
```

```
<DstTzInfo 'Asia/Shanghai' LMT+8:06:00 STD>
```

9.3 时间转换

时间的三种存在方式：时间对象，时间字符串，时间戳。

时间对象，比如前面介绍的 date 、datetime 、time 对象等；时间字符串，如：“2020-10-12”；时间戳，如 time.time() 返回的就是时间戳。

在数据处理过程中，经常会遇到需要将不同形式的时间进行转换。这里给大家介绍下常用的方法：

9.3.1 时间对象转字符串

时间对象转换为字符串，可以通过 isoformat 或 strftime 方法来实现。

strftime 的英文全称是 str format time ，根据给定的格式将时间对象转换为字符串

```
# 将 date 时间对象转换为字符串
d1 = datetime.date(2020,10,12)
d1.isoformat()
```

```
'2020-10-12'
```

```
# 用 srtftime 来转换
# YYYY-MM-DD 形式
d1.strftime('%Y-%m-%d')
```

```
'2020-10-12'
```

```
# MM DD, YYYY 形式
d1.strftime('%b %d, %Y')
```

```
'Oct 12, 2020'
```

```
# 将 time 时间对象转换为字符串
t1 = datetime.time(20,10,1)
t1.strftime('%H:%M:%S')
```

```
'20:10:01'
```

```
# 将 datetime 时间对象转换为字符串
dt2 = datetime.datetime.now()
dt2.strftime('%Y-%m-%d %H:%M:%S')
```

```
'2020-10-14 15:12:20'
```

```
dt2.isoformat()
```

```
'2020-10-14T15:12:20.403113'
```

python 中常见的时间日期格式化符号：

指令	意义	示例
%a	当地工作日的缩写。	Sun, Mon, …, Sat (en_US);So, Mo, …, Sa (de_DE)
%A	本地化的星期中每日的完整名称。	Sunday, Monday, …, Saturday (en_US);Sonntag, Montag, …, Samstag (de_DE)
%w	以十进制数显示的工作日，其中 0 表示星期日，6 表示星期六。	0, 1, …, 6
%d	补零后，以十进制数显示的月份中的一天。	01, 02, …, 31
%b	当地月份的缩写。	Jan, Feb, …, Dec (en_US);Jan, Feb, …, Dez (de_DE)

指令	意义	示例
%B	本地化的月份全名。	January, February, …, December (en_US);Januar, Februar, …, Dezember (de_DE)
%m	补零后，以十进制数显示的月份。	01, 02, …, 12
%y	补零后，以十进制数表示的，不带世纪的年份。	00, 01, …, 99
%Y	十进制数表示的带世纪的年份。	0001, 0002, …, 2013, 2014, …, 9998, 9999
%H	以补零后的十进制数表示的小时（24 小时制）。	00, 01, …, 23
%I	以补零后的十进制数表示的小时（12 小时制）。	01, 02, …, 12
%p	本地化的 AM 或 PM 。	AM, PM (en_US);am, pm (de_DE)
%M	补零后，以十进制数显示的分钟。	00, 01, …, 59
%S	补零后，以十进制数显示的秒。	00, 01, …, 59
%f	以十进制数表示的微秒，在左侧补零。	000000, 000001, …, 999999
%z	UTC 偏移量，格式为 ±HHMM[SS[.ffffff]]（如果是简单型对象则为空字符串）。	(空), +0000, -0400, +1030, +063415, -030712.345216
%Z	时区名称（如果对象为简单型则为空字符串）。	(空), UTC, GMT
%j	以补零后的十进制数表示的一年中的日序号。	001, 002, …, 366
%U	以补零后的十进制数表示的一年中的周序号（星期日作为每周的第一天）。在新的一年中第一个星期日之前的所有日子都被视为是在第 0 周。	00, 01, …, 53
%W	以十进制数表示的一年中的周序号（星期一作为每周的第一天）。在新的一年中第一个第期一之前的所有日子都被视为是在第 0 周。	00, 01, …, 53
%c	本地化的适当日期和时间表示。	Tue Aug 16 21:30:00 1988 (en_US);Di 16 Aug 21:30:00 1988 (de_DE)
%x	本地化的适当日期表示。	08/16/88 (None);08/16/1988 (en_US);16.08.1988 (de_DE)
%X	本地化的适当时间表示。	21:30:00 (en_US);21:30:00 (de_DE)
%%	字面的 '%' 字符。	%

9.3.2 字符串转时间对象

字符串转时间对象，用的是 strptime 方法，与 strftime 方法刚好相反。

strptime 的英文全称是 str parse time ，将字符串解析为给定相应格式的时间对象。

```
s1 = '2020-10-09'
d = datetime.datetime.strptime(s1,'%Y-%m-%d')
d
```

```
datetime.datetime(2020, 10, 9, 0, 0)
```

下面提供了 strftime 方法与 strptime 方法的比较：

	strftime	strptime
用法	根据给定的格式将对象转换为字符串	将字符串解析为给定相应格式的 datetime对象
方法类型	实例方法	类方法
方法	date; datetime; time	datetime
签名	strftime(format)	strptime(date_string, format)

需要注意的是，strftime 方法可供 date、time 和 datetime 对象使用，而 strptime 方法仅供 datetime 对象使用。

9.3.3 时间戳转换为时间对象

时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒 (北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒) 起至现在的总秒数。

```
# 获取当前的时间戳
ts_1 = time.time()
ts_1
```

```
1602660751.071785
```

```
# 获取当天00:00:00的时间戳
ts_2 = int(time.time()/86400)*86400
ts_2
```

```
1602633600
```

```
# 获取当天23:59:59的时间戳
# 一天有 24*60*60 = 86400 秒
ts_3 = int(time.time()/86400)*86400+86400-1
ts_3
```

```
1602719999
```

```
# 将时间戳转换为时间对象
datetime.datetime.fromtimestamp(ts_1)
```

```
datetime.datetime(2020, 10, 14, 15, 32, 31, 71785)
```

```
# 将时间戳转换为时间对象
datetime.date.fromtimestamp(ts_1)
```

```
datetime.date(2020, 10, 14)
```

9.3.4 将时间对象转换为时间戳

鉴于，时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒 (北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒) 起至现在的总秒数。

因此，将时间对象转换为时间戳时，直接计算两个时间对象的 timedelta ，并将 timedelta 以 “秒” 来表示就可以了。

```
dt1
```

```
datetime.datetime(2020, 10, 11, 20, 10, 1)
```

```
# 注意这里要用 北京时间
dt_s = datetime.datetime(1970,1,1,8)
dt_s
```

```
datetime.datetime(1970, 1, 1, 8, 0)
```

```
timedelta_1 = dt1 -dt_s
# 返回时间间隔包含了多少秒
timedelta_s = timedelta_1.total_seconds()
timedelta_s
```

```
1602418201.0
```

这里我们来反推下，看我们将时间对象转换为时间戳，是否正确。

```
# 将时间戳转换为时间对象
datetime.datetime.fromtimestamp(timedelta_s)
```

```
datetime.datetime(2020, 10, 11, 20, 10, 1)
```

9.4 时间对象的运算

9.4.1 获取当天最小时间和最大时间

```
# 获取当天最小时间
```

```
datetime.datetime.combine(datetime.date.today(),datetime.time.min)
```

```
datetime.datetime(2020, 10, 14, 0, 0)
```

```
# 获取当天最大时间
datetime.datetime.combine(datetime.date.today(),datetime.time.max)
```

```
datetime.datetime(2020, 10, 14, 23, 59, 59, 999999)
```

9.4.2 获取当前日期的前几天/后几天

```
# 获取明天
datetime.date.today() + datetime.timedelta(days=1)
```

```
datetime.date(2020, 10, 15)
```

```
# 获取明天
datetime.date.today() - datetime.timedelta(days=1)
```

```
datetime.date(2020, 10, 13)
```

9.4.3 获取本周或本月第一天及最后一天

```
d_today = datetime.date.today()
d_today
```

```
datetime.date(2020, 10, 14)
```

```
# 获取本周第一天
d_today - datetime.timedelta(d_today.weekday())
```

```
datetime.date(2020, 10, 12)
```

```
# 获取本周最后一天
d_today + datetime.timedelta(6-d_today.weekday())
```

```
datetime.date(2020, 10, 18)
```

9.4.4 计算两个日期相差多少天

```
# timedelta 对象的计算
td1 = dt2 - dt1
td1
```

```
datetime.timedelta(days=2, seconds=68539, microseconds=403113)
```

```
td1.days
```

```
2
```

注意下，如果需要计算两个日期之间总共相差多少秒，应该用 `total_seconds()` 方法。

```
td1.seconds
```

```
68539
```

```
td1.total_seconds()
```

```
241339.403113
```

参考文档：

- https://docs.python.org/zh-cn/3/library/datetime.html#strftime-strptime-behavior
- https://segmentfault.com/a/1190000012112097
- https://blog.csdn.net/qq_34493908/article/details/80888052
- https://zhuanlan.zhihu.com/p/96384066

10 Python 内置时间模块：Time

10.1 模块介绍

time 模块提供了各种时间相关的函数。

由于时间处理涉及到时区相关的知识，在开始之前，有一些术语和惯例需要阐述下：

- UTC 是协调世界时（以前称为格林威治标准时间，或 GMT）。缩写 UTC 不是错误，而是英语和法语之间的妥协。在中国为 UTC+8。
- DST 是夏令时，在一年中的一部分时间（通常）调整时区一小时。DST 规则很神奇（由当地法律确定），并且每年都会发生变化。C 库有一个包含本地规则的表（通常是从系统文件中读取以获得灵活性），并且在这方面是 True Wisdom 的唯一来源。
- 时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒 (北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒) 起至现在的总秒数。
- 时间元组 (struct_time) 方式：struct_time 元组共有 9 个元素，返回 struct_time 的函数主要有 gmtime()，localtime()，strptime()。下面列出这种方式元组中的几个元素：

索引（Index）	属性（Attribute）	值（Values）
0	tm_year（年）	比如 2020
1	tm_mon（月）	1 - 12
2	tm_mday（日）	1 - 31
3	tm_hour（时）	0 - 23
4	tm_min（分）	0 - 59
5	tm_sec（秒）	0 - 61
6	tm_wday（weekday）	0 - 6（0 表示周日）
7	tm_yday（一年中的第几天）	1 - 366
8	tm_isdst（是否是夏令时）	默认为-1

10.2 时间获取

10.2.1 获取时间戳

time.time() 返回当前时间的时间戳

```
import time
```

```
# 返回当前时间的时间戳
t1 = time.time()
t1
```

```
1602728783.2113311
```

10.2.2 获取时间元组

```
# 返回当前时间的时间元组 （struct_time）
t2 = time.localtime()
t2
```

```
time.struct_time(tm_year=2020, tm_mon=10, tm_mday=15, tm_hour=10, tm_min=26,
    tm_sec=38, tm_wday=3, tm_yday=289, tm_isdst=0)
```

```
# 返回当前时间的 UTC 时区的时间元组 （struct_time）
t3 = time.gmtime()
t3
```

```
time.struct_time(tm_year=2020, tm_mon=10, tm_mday=15, tm_hour=2, tm_min=26,
    tm_sec=49, tm_wday=3, tm_yday=289, tm_isdst=0)
```

struct_time 元组共有 9 个元素，如果想自定义构造时间元组，则可以通过传入一个数组来实现，如下：

```
t3_1 = time.struct_time([2020,10,17,2,2,34,4,89,0])
t3_1
```

```
time.struct_time(tm_year=2020, tm_mon=10, tm_mday=17, tm_hour=2, tm_min=2,
    tm_sec=34, tm_wday=4, tm_yday=89, tm_isdst=0)
```

10.3 时间转换

10.3.1 时间元组转换为时间戳

```
# 将一个 struct_time 转化为时间戳
t4 = time.mktime(t2)
t4
```

```
1602728798.0
```

10.3.2 时间元组为字符串

time.asctime() 转换由 gmtime() 或 localtime() 所返回的表示时间的元组或 struct_time 为以下形式的字符串: “Sun Jun 20 23:21:05 1993”。

简单理解，转换时间元组为字符串形式。

```
# 转换时间元组为字符串形式
t5 = time.asctime(t2)
t5
```

```
'Thu Oct 15 10:26:38 2020'
```

如果想自定义字符串的格式，则可以通过 `strftime()` 来实现。`strftime()` 可以将时间元组转换为由 format 参数指定的字符串。

```
t5_1 = time.strftime('%Y-%m-%d',t2)
t5_1
```

```
'2020-10-15'
```

10.3.3 时间戳转换为字符串

time.ctime() 转换以距离初始纪元的秒数表示的时间为以下形式的字符串: “Sun Jun 20 23:21:05 1993” 代表本地时间。

简单理解，就是转换时间戳为字符串形式。

```
# 转换时间戳为字符串形式
t6 = time.ctime(t4)
t6
```

```
'Thu Oct 15 10:26:38 2020'
```

10.3.4 字符串转换为时间元组

`strptime()` 方式可以将特定格式的字符串解析为时间元组。字符串的默认匹配格式为 `"%a %b %d %H:%M:%S %Y"`。

```
time.strptime('Thu Oct 15 10:30:18 2020')
```

```
time.struct_time(tm_year=2020, tm_mon=10, tm_mday=15, tm_hour=10, tm_min=30,
    tm_sec=18, tm_wday=3, tm_yday=289, tm_isdst=-1)
```

指令表格

指令	意义
%a	本地化的缩写星期中每日的名称。
%A	本地化的星期中每日的完整名称。
%b	本地化的月缩写名称。
%B	本地化的月完整名称。
%c	本地化的适当日期和时间表示。
%d	十进制数 [01,31] 表示的月中日。
%H	十进制数 [00,23] 表示的小时（24 小时制）。
%I	十进制数 [01,12] 表示的小时（12 小时制）。
%j	十进制数 [001,366] 表示的年中日。
%m	十进制数 [01,12] 表示的月。
%M	十进制数 [00,59] 表示的分钟。
%p	本地化的 AM 或 PM 。
%S	十进制数 [00,61] 表示的秒。
%U	十进制数 [00,53] 表示的一年中的周数（星期日作为一周的第一天）作为。在第一个星期日之前的新年中的所有日子都被认为是在第 0 周。
%w	十进制数 [0(星期日),6] 表示的周中日。
%W	十进制数 [00,53] 表示的一年中的周数（星期一作为一周的第一天）作为。在第一个星期一之前的新年中的所有日子被认为是在第 0 周。
%x	本地化的适当日期表示。
%X	本地化的适当时间表示。
%y	十进制数 [00,99] 表示的没有世纪的年份。
%Y	十进制数表示的带世纪的年份。
%z	时区偏移以格式 +HHMM 或 -HHMM 形式的 UTC/GMT 的正或负时差指示，其中 H 表示十进制小时数字，M 表示小数分钟数字 [-23:59, +23:59] 。
%Z	时区名称（如果不存在时区，则不包含字符）。
%%	字面的 '%' 字符。

参考文档：

- https://docs.python.org/zh-cn/3/library/time.html#module-time

11 Python 内置时间模块：Calendar

Python 内置的日历（calendar）模块主要输出跟日历相关的内容，相对 datetime 和 time 模块来说，calendar 模块的功能要简单些。

最常用的两个功能是输出某年的日历，以及某个月的日历。

下面跟大家介绍下常见的情况：

11.1 获取某年的日历

```
import calendar
```

```
# 获取某年的日历并打印出来
calendar.prcal(2020)
```

```
                                  2020

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2                         1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 28 29
                                                    30 31

       April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2  3       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 13 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 20 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 27 28
27 28 29 30               25 26 27 28 29 30 31      29 30

        July                    August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2          1  2  3  4  5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       7  8  9 10 11 12 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      14 15 16 17 18 19 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      21 22 23 24 25 26 27
```

```
27 28 29 30 31            24 25 26 27 28 29 30      28 29 30
                          31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
          1  2  3  4                         1          1  2  3  4  5  6
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 12 13
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 19 20
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 26 27
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
```

11.2 获取某月的日历

```
calendar.month(2020,10)
```

```
'    October 2020\nMo Tu We Th Fr Sa Su\n          1  2  3  4\n 5  6  7  8  9
    10 11\n12 13 14 15 16 17 18\n19 20 21 22 23 24 25\n26 27 28 29 30 31\n'
```

这里需要注意下，在没有使用行数 `print()` 的情况下，输出的是原始的字符串形式。用 `print()` 输出后，显示如下：

```
print(calendar.month(2020,10))
```

```
    October 2020
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

也可以用 `prmonth()` 函数将结果直接打印出来，效果也是一样的。

```
calendar.prmonth(2020,10)
```

```
    October 2020
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

11.3 其他方法

calendar.monthcalendar()

返回表示一个月的日历的矩阵。每一行代表一周；此月份外的日子由零表示。每周从周一开始，除非使用 setfirstweekday() 改变设置。

```
print(calendar.monthcalendar(2020,10))
```

```
[[0, 0, 0, 1, 2, 3, 4], [5, 6, 7, 8, 9, 10, 11], [12, 13, 14, 15, 16, 17, 18],
    [19, 20, 21, 22, 23, 24, 25], [26, 27, 28, 29, 30, 31, 0]]
```

calendar.weekday()

返回某天是星期几，默认情况下 0-6 代表周一到周日。

```
# 获取日期是星期几 (0-6 代表周一到周日)
print(calendar.weekday(2020,10,15))
```

```
3
```

calendar.setfirstweekday(weekday)

设置每一周的开始 (0 表示星期一，6 表示星期天)。calendar 还提供了 MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY 和 SUNDAY 几个常量方便使用。例如，设置每周的第一天为星期天

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)
```

calendar.weekheader(n)

返回一个包含星期几的缩写名的头。n 指定星期几缩写的字符宽度。

```
print(calendar.weekheader(3))
```

```
Mon Tue Wed Thu Fri Sat Sun
```

12 Jupyter Notebook 主题设置、字体修改等

作为用 Python 进行数据分析的人员，jupyter notebook 在平时使用的频率很高。但经常觉得 jupyter notebook 默认的风格不是很爽，总想换一换。今天，我们来分享下给 jupyter notebook 更换主题等内容的方法。

12.1 新的风格预览

首先来看看我现在使用的风格，如下：

notebook 页面：

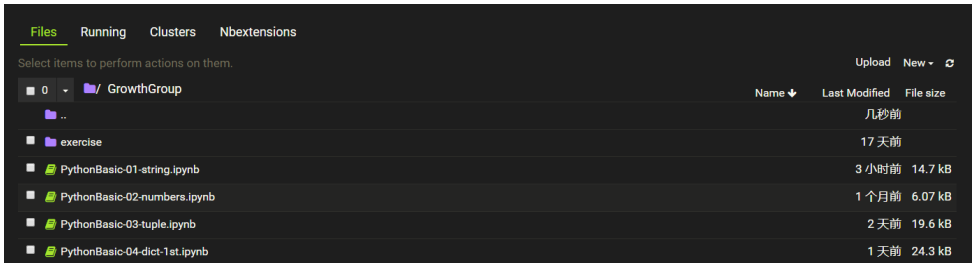


Figure 12.1: notebook 页面

代码文件页面：

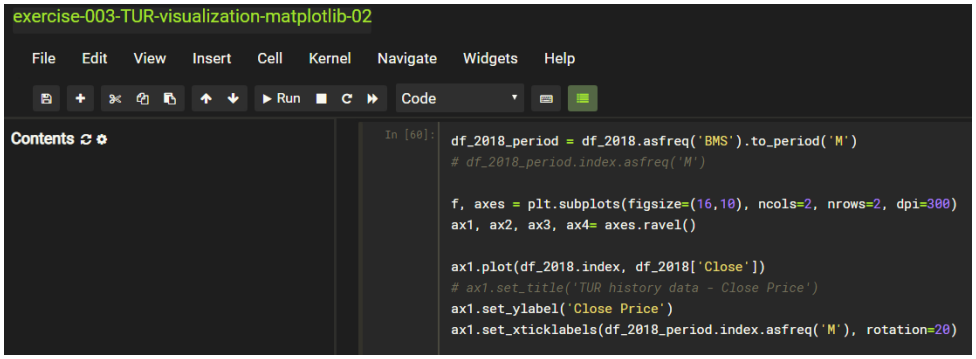


Figure 12.2: 代码文件页面

12.2 主题更换过程

如果你也想更换下 jupyter notebook 的默认风格，不妨接着往下看。

Jupyter Notebook 的默认主题是白色背景的，虽然也简洁大方，便于使用。但是在长时间使用默认界面后，我只有一种感觉，就是亮瞎…… 无比怀念类似 Pycharm 或 vs code 中设置的黑色界面……

于是，想给 jupyter notebook 也换个黑色主题。我搜索了 Jupyter Notebook 的 themes，也就是自定义主题。发现 Github 已经有大神早已解决了这个问题。本次，我们用到的库为 jupyterthemes ，这个第三方库有多个主题可以更换，并且还可以更换字体类型以及大小等，基本满足我们的需求。jupyterthemes 的 github 地址如下：

https://github.com/dunovank/jupyter-themes

默认情况下库的安装如下：

```
pip install jupyterthemes
```

但是，上述方法我没有安装成功，主要是获取该库的请求失败。于是，改用豆瓣的 python 库来源，安装的代码如下：

```
pip install -i https://pypi.doubanio.com/simple/ jupyterthemes
```

Figure 12.3: 豆瓣源

改用豆瓣的来源后，成功安装了 jupyterthemes ，接下来，我们可以来设置自己喜欢的主题。在命令提示符下输入下面的代码来查看有哪些主题可以选择。

```
jt -l
```

有如下一些主题可以选择

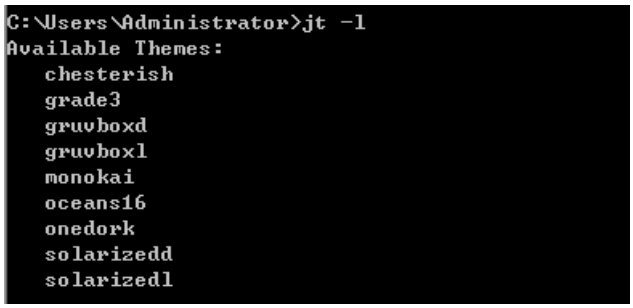


Figure 12.4: 可选主题

有兴趣的同学可以尝试下这几个主题。设置主题的代码如下：

```
jt -t oceans16
```

上面设置的是使用 oceans16 这个主题，更换后的效果如下：

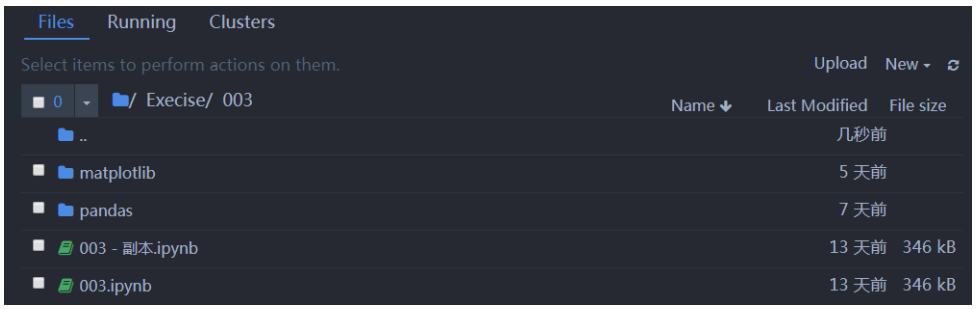


Figure 12.5: oceans16 主题

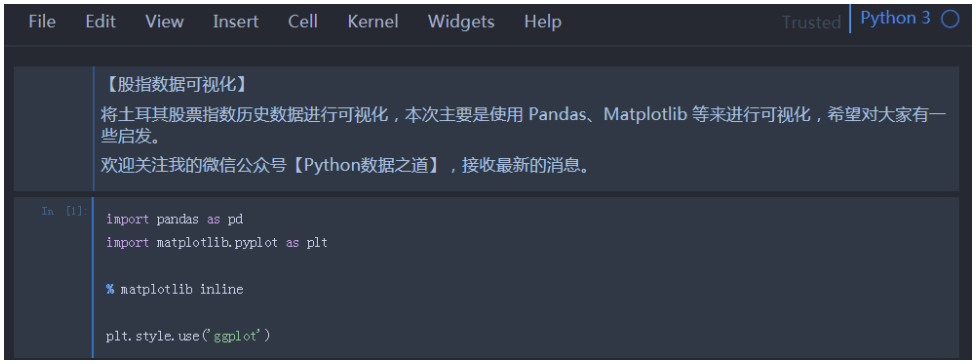


Figure 12.6: oceans16 主题

如果不喜欢上面的主题，可以使用下面的代码来恢复到默认主题，如下：

```
jt -r
```

对比其他几个主题后，相对来说，我更喜欢 monokai 以及 onedork 主题，最后，我选择使用 monokai 主题，也就是本文开始的主题风格。

12.3 我的主题设置参数

我的 monikai 主题设置的详细参数如下：

```
jt -t monokai -f roboto -nf robotosans -tf robotosans -N -T -cellw 70% -dfs 10
    -ofs 10
```

乍一看，看到这串长长的代码，可能有点晕，但其实很好理解。`-t` 是设置主题，`-f` 设置代码的字体，`-nf` 设置 notebook 的字体，等等。更多的参数，请查看 jupyterthemes 的参数设置说明，如下：

Description of Command Line options

cl options	arg	default
Usage help	-h	--
List Themes	-l	--
Theme Name to Install	-t	--
Code Font	-f	--
Code Font-Size	-fs	11
Notebook Font	-nf	--
Notebook Font Size	-nfs	13
Text/MD Cell Font	-tf	--
Text/MD Cell Fontsize	-tfs	13
Pandas DF Fontsize	-dfs	9
Output Area Fontsize	-ofs	8.5
Mathjax Fontsize (%)	-mathfs	100
Intro Page Margins	-m	auto
Cell Width	-cellw	980
Line Height	-lineh	170
Cursor Width	-cursw	2
Cursor Color	-cursc	--
Alt Prompt Layout	-altp	--
Alt Markdown BG Color	-altmd	--
Alt Output BG Color	-altout	--
Style Vim NBExt*	-vim	--
Toolbar Visible	-T	--
Name & Logo Visible	-N	--
Kernel Logo Visible	-kl	--
Reset Default Theme	-r	--
Force Default Fonts	-dfonts	--

Figure 12.7: jupyterthemes 的参数设置说明

怎么样，新的界面不错吧，心动不如行动，赶紧动手尝试下吧。

13 Jupyter Notebook 中自动补全代码

本文来继续介绍 jupyter notebook 中一些实际的技巧。本次要介绍的两个功能是：

（1）针对 jupyter notebook 中的 Markdown 文件自动生成目录

（2）自动补全代码

上述两个功能，都是由 python 的一个 jupyter 扩展插件 Nbextensions 库来实现。安装该库的命令如下：

```
python -m pip install jupyter_contrib_nbextensions
```

然后执行：

```
jupyter contrib nbextension install --user --skip-running-check
```

安装完成后，勾选 `Table of Contents` 以及 `Hinterland`。其中 Hinterland 是用来自动补全代码的，这个拓展的代码补全功能虽然没有 PyCharm 中的那么全面，但比没有是要好多了。设置如下：

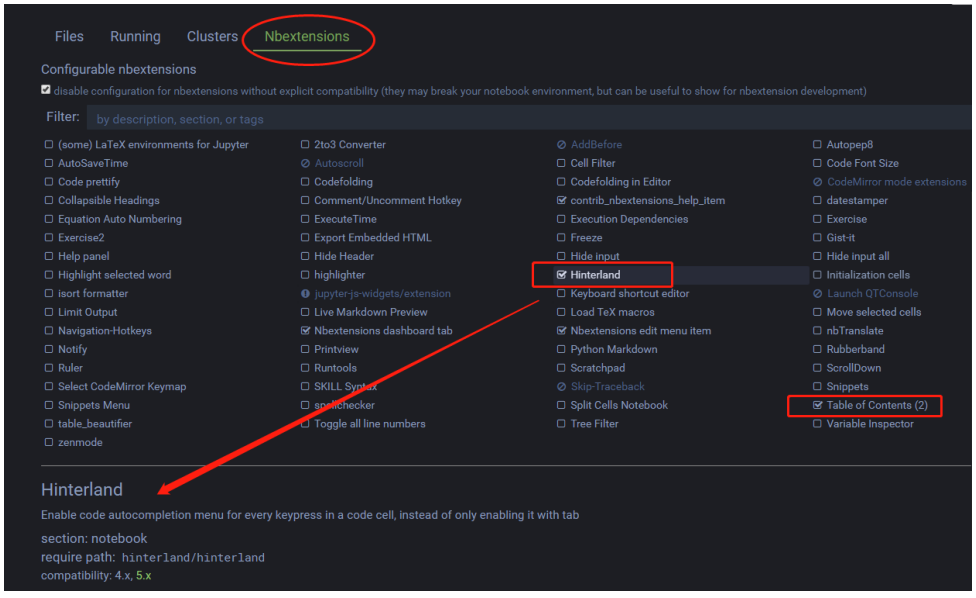


Figure 13.1: Hinterland 设置

自动补全代码的效果如下：

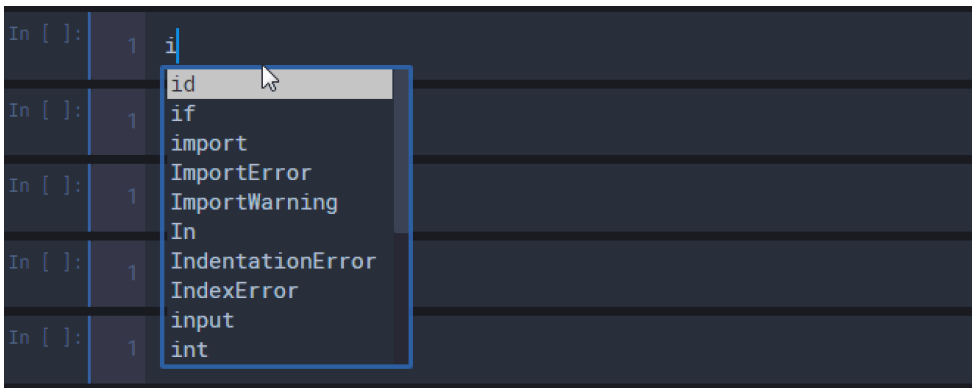


Figure 13.2: 自动补全代码的效果

> 上图在文章中是 gif 的动态图，可以在下面文章中查看：
>
> http://liyangbit.com/jupyter/jupyter-code-auto-complete/

14 Jupyter Notebook 输出 pdf 并支持中文显示

Jupyter Notebook 作为用 Python 进行数据分析的重要工具之一，其最大的特色是可以将代码和结果同步显示在源文件里。Jupyter Notebook 可以用来演示，以及输出各种形式的文件，比如 ipynb，html，md，py，pdf 等，本文主要来阐述如何将写好的内容输出为 pdf 格式的文件。在 Jupyter Notebook 输出 pdf 过程中，相信不少同学因为遇到难以解决的坑而退却，这里给大家分享下我的历程，希望对大家有所帮助。

系统环境：

- windows 7 , windows 10
- Anancoda （基于 Python 3.6 版）

14.1 安装顺序

我是按照下面的顺序安装的，缺少支持的话，`pip install xxxxxx`

1 安装 pandoc

https://github.com/jgm/pandoc/releases

2 安装 MiKTex

https://miktex.org/download

3 中文支持

(1) 直接修改 tex 模版文件。

首先找到 article.tplx 文件，我的 Anaconda 中的路径是 `D:\ProgramData\Anaconda3\pkgs\nbconvert-5.3.1-py36h8dc0fde_0\Lib\site-packages\nbconvert\templates\latex\article.tplx`

你可以根据自己的安装情况，找到该文件，然后用文本编辑器打开 article.tplx ，将 “`\documentclass[11pt]{article}`” 修改为 “`\documentclass{ctexart}`”，对比图示如下：

修改前：

```
% Default to the notebook output style
((* if not cell_style is defined *))
    ((* set cell_style = 'style_ipython.tplx' *))
((* endif *))

% Inherit from the specified cell style.
((* extends cell_style *))


%==============================================================
% Latex Article
%==============================================================

((* block docclass *))
\documentclass[11pt]{article}
((* endblock docclass *))
```

修改前

Figure 14.1: 修改前

修改后：

```
% Default to the notebook output style
((* if not cell_style is defined *))
    ((* set cell_style = 'style_ipython.tplx' *))
((* endif *))

% Inherit from the specified cell style.
((* extends cell_style *))


%==============================================================
% Latex Article
%==============================================================

((* block docclass *))
\documentclass{ctexart}
((* endblock docclass *))
```

修改后

Figure 14.2: 修改后

（2）打开一个含有中文内容的 jupyter notebook 文件（.ipynb 文件），在浏览器中打开，选择输出为 pdf 文件（我这里是新建的一个空白的 ipynb 文档），如下：

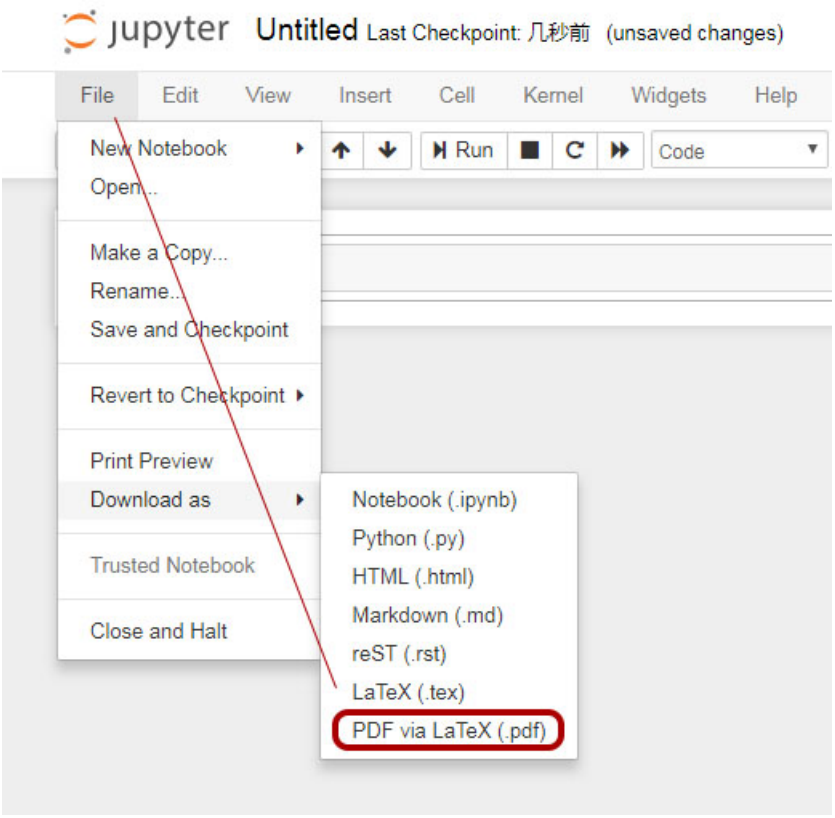


Figure 14.3: ipynb 文档

这里有可能可以正常的输出含有中文的 pdf 文档，如果已输出，那么恭喜你，已经成功啦。但我在输出含有中文内容的 pdf 文档过程中，还遇到了一些问题，主要是显示 “XXX.sty” 文件缺失，这时候，需要安装缺失的文件，由于默认安装情况下，经常会失败。此时，我们需要选择安装源，步骤图示如下：

首先，要通过点击 “Change”来选择

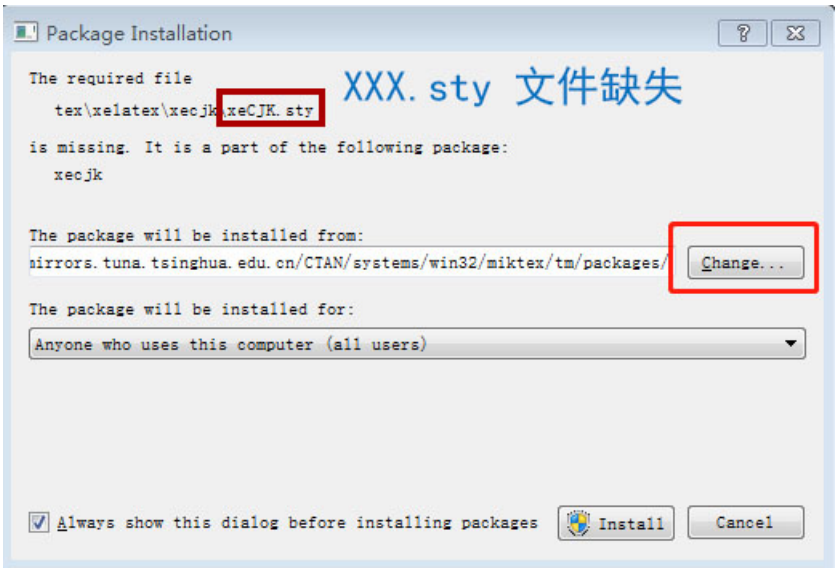


Figure 14.4: 安装缺失文件

按下面图示选项，点击“next”

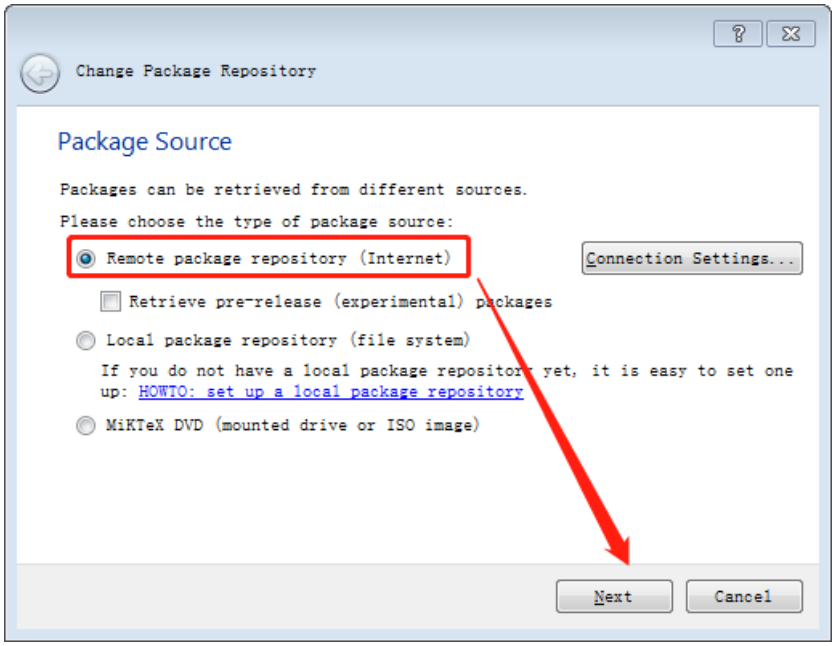


Figure 14.5: 安装缺失文件

在选择安装源的时候，如果有中国的安装来源，请优先选择国内的，这样安装速度较快，成功率相对高些。如果没有，可以多尝试几次其他国家和地区的，比如日本等地。

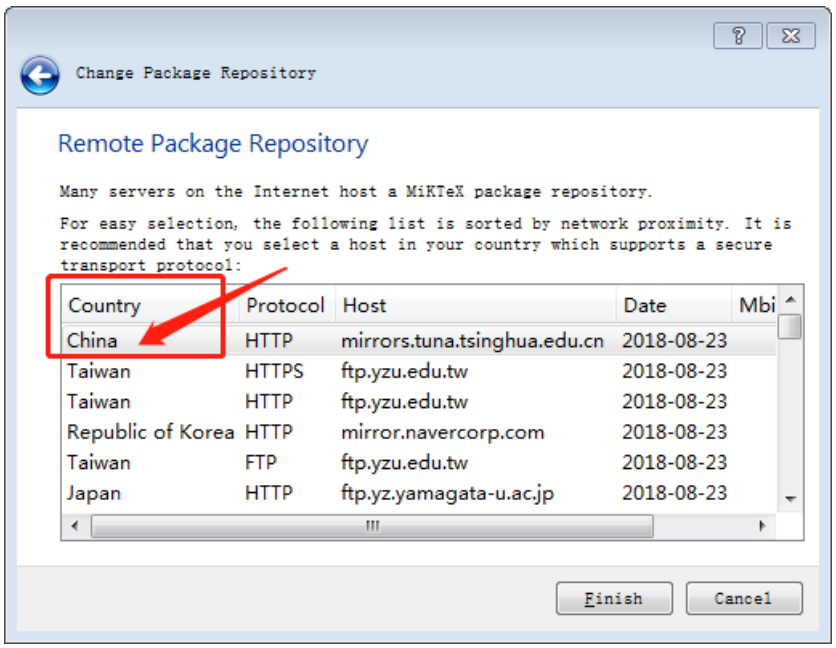


Figure 14.6: 安装缺失文件：选择安装源

选择好后，点击安装即可。

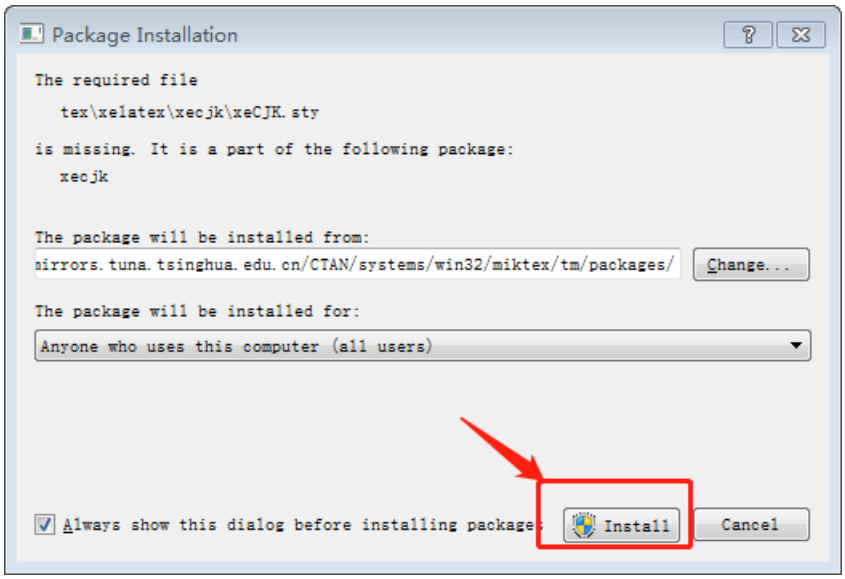


Figure 14.7: 安装缺失文件

一般情况下，将上述缺失的多个.sty 文件安装好后，是可以输出含中文内容的 pdf 文档的。至此，我们本次的目的已基本完成。

但在后面的使用过程中，我还发现了一个小 bug，就是：如果 `.ipynb` 文件的命名中含有中文，则文件名输出是不含中文的，不知道这个问题有没有同学可以提供下解决方案，欢迎献计献策 ~

> 本文参考以下内容：
>
> Jupyter-Notebook-PDF 输出的中文支持

15 Anaconda 安装虚拟环境到指定路径

15.1 曾经的困扰

有段时间，想使用基于不同 python 版本的 anaconda，就直接从官网下载了两个不同的 anaconda 版本进行安装。刚开始的时候，还觉得也没啥问题。用了一小段时间，在安装其他的第三方库时，经常发现安装失败，并且经常出现下面的问题：

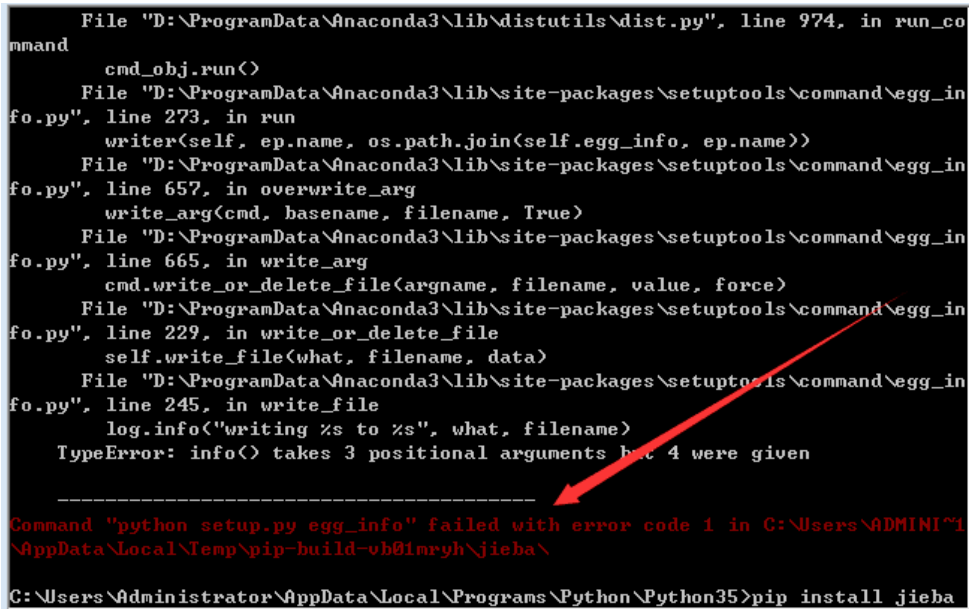


Figure 15.1: 出现的问题

这个问题，我 google、百度等查了好久，也没有解决好。后来，我把两个版本的 anaconda 都卸载了，重新安装了其中一个版本，发现再安装其他第三方库时，上述问题就不存在了。很有可能上述问题就是同时安装两个版本的 anaconda 引起的（不过我也不能完全肯定）。

15.2 安装虚拟环境

虽然只安装一个版本的 anaconda，能顺利的运行，但有时候，还是需要运行基于不同 python 版本的 anaconda 的。后来了解到，anaconda 是可以在虚拟环境下运行不同 python 版本的。下面的步骤演示了我的安装过程，也希望大家能避免一些坑。

先说下我的安装环境：

- windows7,64 位系统
- 目前安装了 anaconda4.2.0 版本（基于 python3.5）

由于经常看到有说 python3.6 版本运行效率比 3.5 版本高，就有一种想安装 3.6 的冲动，但是由于部分 python 库还不支持 3.6，所以主要版本还是以 3.5 为主。anaconda 安装 python3.6 版本的虚拟环境的步骤如下：

输入安装命令：

```
conda create -n py36 python=3.6
```

结果如下：

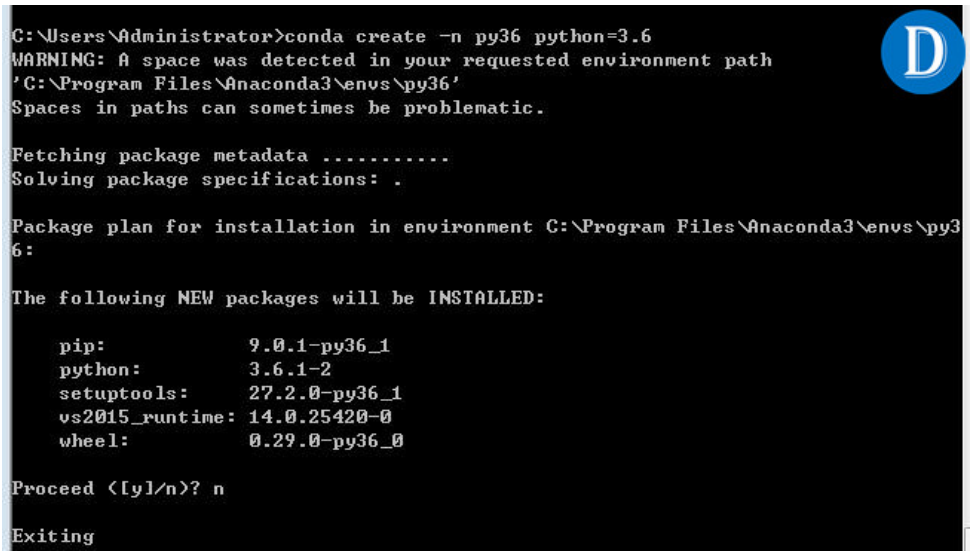


Figure 15.2: 安装演示

“-n”命令，会将虚拟环境安装在 anaconda 已安装路径默认的 envs 目录下。

可以看出，命令行中提示有一个警告，由于我的 anaconda 默认安装路径中有空格，提示可能会引起一些问题。由于有这个警告，我就没有继续往下安装。另外，我的路径在 C 盘，考虑到可能空间也不多，还是不要继续往 C 盘装东西了。于是想，要是能安装在其他指定路径且路径中没有空格，不就解决这个问题了么。

通过查阅 anaconda 的文档，发现是可以进行指定路径安装的。可以输入如下命令进行查看：

```
conda create --help
```

安装虚拟环境到指定路径的命令如下：

```
conda create --prefix=D:\python36\py36 python=3.6
```

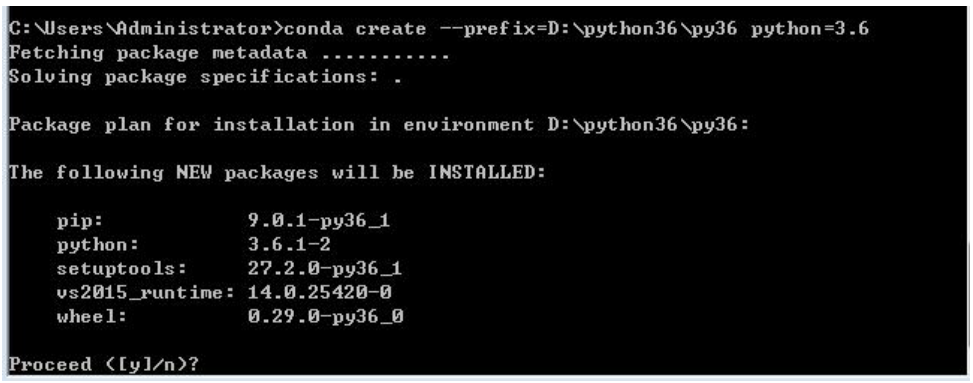


Figure 15.3: 安装到指定路径

上面的命令中，路径 D:\python36 是先建好的文件夹，py36 是需要安装的虚拟环境名称。请注意，安装完成后，虚拟环境的全称包含整个路径，为 D:\python36\py36 。激活指定路径下的虚拟环境的命令如下：

```
activate D:\python36\py36
```

退出虚拟环境的命令如下：

```
deactivate
```

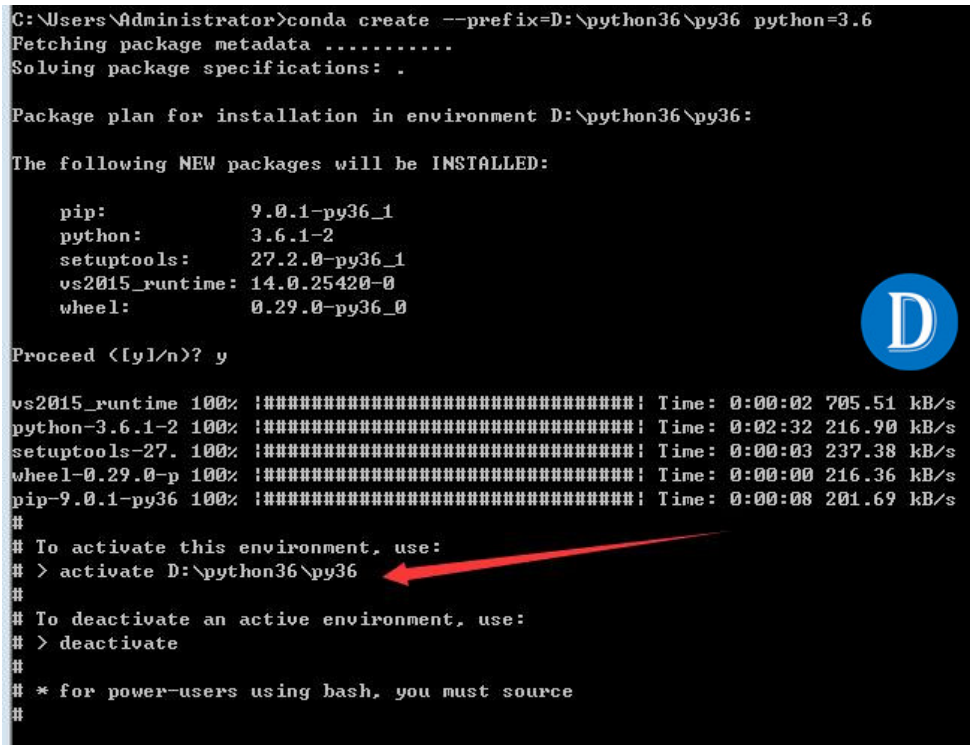


Figure 15.4: 退出虚拟环境

想要删除指定路径下的虚拟环境，使用如下的命令：

```
conda remove --prefix=D:\python36\py36 --all
```

Figure 15.5: 删除指定路径下的虚拟环境

15.3 虚拟环境下安装 python 库

如果还想继续安装 python2.7 的虚拟环境，方法跟安装 python3.6 是一致的。上述虚拟环境 py27 安装完成后，激活后虚拟环境后，可以安装其他 python 库。比如安装 requests 库

```
pip install requests
```

是可以安装成功的。

如果遇到 pip 安是失败，可尝试用下述方法：（指定路径下安装）：

```
conda install -prefix=D:\pyenv\py27 package
```

请注意，默认路径下的命令是“`conda install -n py27 package`”。当然了，**有一些库不论 conda 和 pip 都无法直接安装，只能下载.whl 进行安装**。

最后，来查看已安装好的虚拟环境下都安装了那些 python 库，输入命令：

```
conda list
```

发现已安装好的虚拟环境 py36 下只安装了少量的基本库，如果也想用 3.6 版来进行科学计算，需要安装许多其他的库，如果一个一个安装，则会费时费力，而且容易出错，这时 anaconda 提供了一个命令，可以把基于 python3.6 版本的 anaconda 中的其他库一次安装好，命令如下：

```
conda install -prefix=D:\pyenv\py36 anaconda
```

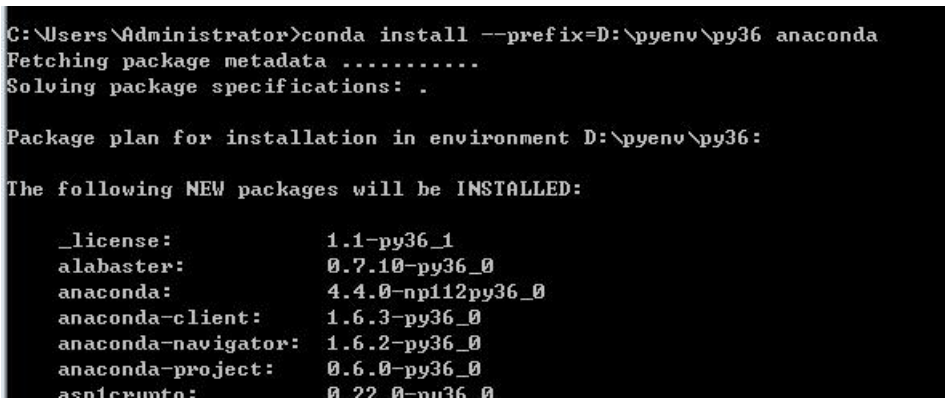


Figure 15.6: 安装 Anaconda 中所有的库

请特别注意，全部安装时，安装包会很多，安装时间比较长，同时占用安装空间也会比较大，请根据自己的需求选择是否安装。我安装 py36 全部的库花了大概 1 个多小时。

16 Numpy 的基本用法

Numpy 是一个开源的 Python 科学计算库，它是 python 科学计算库的基础库，许多其他著名的科学计算库如 Pandas，Scikit-learn 等都要用到 Numpy 库的一些功能。本文主要内容如下：

1. Numpy 数组对象
2. 创建 ndarray 数组
3. Numpy 的数值类型
4. ndarray 数组的属性
5. ndarray 数组的切片和索引
6. 处理数组形状
7. 数组的类型转换
8. numpy 常用统计函数
9. 数组的广播

16.1 Numpy 数组对象

Numpy 中的多维数组称为 ndarray，这是 Numpy 中最常见的数组对象。ndarray 对象通常包含两个部分：

- ndarray 数据本身
- 描述数据的元数据

Numpy 数组的优势

- Numpy 数组通常是由相同种类的元素组成的，即数组中的数据项的类型一致。这样有一个好处，由于知道数组元素的类型相同，所以能快速确定存储数据所需空间的大小。
- Numpy 数组能够运用向量化运算来处理整个数组，速度较快；而 Python 的列表则通常需要借助循环语句遍历列表，运行效率相对来说要差。
- Numpy 使用了优化过的 C API，运算速度较快

关于向量化和标量化运算，对比下面的参考例子就可以看出差异

- 使用 python 的 list 进行循环遍历运算

```
def pySum():
    a = list(range(10000))
    b = list(range(10000))
    c = []
    for i in range(len(a)):
        c.append(a[i]**2 + b[i]**2)
```

```
7
8       return c
```

```
1 %timeit pySum()
2
3 Out[]:
4 10 loops, best of 3: 49.4 ms per loop
```

- 使用 numpy 进行向量化运算

```
1 import numpy as np
2 def npSum():
3     a = np.arange(10000)
4     b = np.arange(10000)
5     c = a**2 + b**2
6     return c
```

```
1 %timeit npSum()
2
3 Out[]:
4 The slowest run took 262.56 times longer than the fastest. This could mean that
      an intermediate result is being cached.
5     1000 loops, best of 3: 128 µs per loop
```

从上面的运行结果可以看出，**numpy 的向量化运算的效率要远远高于 python 的循环遍历运算（效率相差好几百倍）**。(1ms=1000µs)

16.2 创建 ndarray 数组

首先需要导入 numpy 库，在导入 numpy 库时通常使用“np”作为简写，这也是 Numpy 官方倡导的写法。当然，你也可以选择其他简写的方式或者直接写 numpy，但还是建议用“np”，这样你的程序能和大都数人的程序保持一致。

```
1 import numpy as np
```

创建 ndarray 数组的方式有很多种，这里介绍我使用的较多的几种：

Method 1: 基于 list 或 tuple

```
1  # 一维数组
2
3  # 基于list
4  arr1 = np.array([1,2,3,4])
5  print(arr1)
6
7  # 基于tuple
8  arr_tuple = np.array((1,2,3,4))
9  print(arr_tuple)
10
11 # 二维数组 (2*3)
```

```
arr2 = np.array([[1,2,4], [3,4,5]])
arr2

Out[]:
    [1 2 3 4]
    [1 2 3 4]
    array([[1, 2, 4],
           [3, 4, 5]])
```

请注意：

- 一维数组用 print 输出的时候为 [1 2 3 4]，跟 python 的列表是有些差异的，没有“,”
- 在创建二维数组时，在每个子 list 外面还有一个“[]”，形式为“[[list1], [list2]]”

Method 2: 基于 np.arange

```
# 一维数组
arr1 = np.arange(5)
print(arr1)

# 二维数组
arr2 = np.array([np.arange(3), np.arange(3)])
arr2

Out[]:
    [0 1 2 3 4]
    array([[0, 1, 2],
           [0, 1, 2]])
```

Method 3: 基于 arange 以及 reshape 创建多维数组

```
# 创建三维数组
arr = np.arange(24).reshape(2,3,4)
arr

Out[]:
    array([[[ 0,  1,  2,  3],
            [ 4,  5,  6,  7],
            [ 8,  9, 10, 11]],

           [[12, 13, 14, 15],
            [16, 17, 18, 19],
            [20, 21, 22, 23]]])
```

- 请注意：arange 的长度与 ndarray 的维度的乘积要相等，即 24 = 2X3X4
- 用 numpy.random 创建数组的方法，可以参考我的文章：《为什么你用不好Numpy的random函数》
- 其他创建 ndarray 的方法，各位小伙伴们自己可以研究下。

16.3 Numpy 的数值类型

Numpy 的数值类型如下：

Python数据之道

数据类型	说明
bool	布尔类型，True或False，占用1比特
inti	其长度取决于平台的整数，一般是int32或int64
int8	字节长度的整数，取值：[-128, 127]
int16	16位长度的整数，取值：[-32768, 32767]
int32	32位长度的整数，取值：$[-2^{31}, 2^{31}-1]$
int64	64位长度的整数，取值：$[-2^{63}, 2^{63}-1]$
uint8	8位无符号整数，取值：[0, 255]
uint16	16位无符号整数，取值：[0, 65535]
uint32	32位无符号整数，取值：$[0, 2^{32}-1]$
uint64	32位无符号整数，取值：$[0, 2^{64}-1]$
float16	16位半精度浮点数：1位符号位，5位指数，10位尾数
float32	32位半精度浮点数：1位符号位，8位指数，23位尾数
float64或float	双精度浮点数：1位符号位，11位指数，52位尾数
complex64	复数类型，实部和虚部都是32位浮点数
complex128或complex	复数类型，实部和虚部都是64位浮点数

Figure 16.1: Numpy 的数值类型

每一种数据类型都有相应的数据转换函数，参考示例如下：

```
1 np.int8(12.334)
2
3 Out[]:
4     12
```

```
1 np.float64(12)
2
3 Out[]:
```

```
    12.0
```

```
np.float(True)

Out[]:
    1.0
```

```
bool(1)

Out[]:
    True
```

在创建 ndarray 数组时，可以指定数值类型：

```
a = np.arange(5, dtype=float)
a

Out[]:
    array([ 0.,  1.,  2.,  3.,  4.])
```

- 请注意，复数不能转换成为整数类型或者浮点数，比如下面的代码会运行出错

```
float(42 + 1j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-54bfc3e9c0e0> in <module>()
----> 1 float(42 + 1j)

TypeError: can't convert complex to float
```

Figure 16.2: Error

```
# float(42 + 1j)
```

16.4 ndarray 数组的属性

- **dtype 属性**，ndarray 数组的数据类型，数据类型的种类，前面已描述。

```
np.arange(4, dtype=float)

Out[]:
    array([ 0.,  1.,  2.,  3.])
```

```
# 'D'表示复数类型
np.arange(4, dtype='D')

```

```
Out[]:
    array([ 0.+0.j,  1.+0.j,  2.+0.j,  3.+0.j])
```

```
np.array([1.22,3.45,6.779], dtype='int8')

Out[]:
    array([1, 3, 6], dtype=int8)
```

- **ndim 属性**，数组维度的数量

```
a = np.array([[1,2,3], [7,8,9]])
a.ndim

Out[]:
    2
```

- **shape 属性**，数组对象的尺度，对于矩阵，即 n 行 m 列,shape 是一个元组（tuple）

```
a.shape

Out[]:
    (2, 3)
```

- **size 属性**用来保存元素的数量，相当于 shape 中 nXm 的值

```
a.size

Out[]:
    6
```

- **itemsize** 属性返回数组中各个元素所占用的字节数大小。

```
a.itemsize

Out[]:
    4
```

- **nbytes 属性**，如果想知道整个数组所需的字节数量，可以使用 nbytes 属性。其值等于数组的 size 属性值乘以 itemsize 属性值。

```
a.nbytes

Out[]:
    24
```

```
a.size*a.itemsize

Out[]:
    24
```

- **T 属性**，数组转置

```
b = np.arange(24).reshape(4,6)
b

Out[]:
    array([[ 0,  1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11],
           [12, 13, 14, 15, 16, 17],
           [18, 19, 20, 21, 22, 23]])
```

```
b.T

Out[]:
    array([[ 0,  6, 12, 18],
           [ 1,  7, 13, 19],
           [ 2,  8, 14, 20],
           [ 3,  9, 15, 21],
           [ 4, 10, 16, 22],
           [ 5, 11, 17, 23]])
```

- **复数的实部和虚部属性，real 和 imag 属性**

```
d = np.array([1.2+2j, 2+3j])
d

Out[]:
    array([ 1.2+2.j,  2.0+3.j])
```

real 属性返回数组的实部

```
d.real

Out[]:
    array([ 1.2,  2. ])
```

imag 属性返回数组的虚部

```
d.imag

Out[]:
    array([ 2.,  3.])
```

- **flat 属性**，返回一个 numpy.flatiter 对象，即可迭代的对象。

```
e = np.arange(6).reshape(2,3)
e

Out[]:
    array([[0, 1, 2],
           [3, 4, 5]])
```

```
f = e.flat
f

Out[]:
    <numpy.flatiter at 0x65eaca0>
```

```
for item in f:
    print(item)

Out[]:
    0
    1
    2
    3
    4
    5
```

可通过位置进行索引，如下：

```
f[2]

Out[]:
    2
```

```
f[[1,4]]

Out[]:
    array([1, 4])
```

也可以进行赋值

```
e.flat=7
e

Out[]:
    array([[7, 7, 7],
           [7, 7, 7]])
```

```
e.flat[[1,4]]=1
e

Out[]:
    array([[7, 1, 7],
           [7, 1, 7]])
```

下图是对 ndarray 各种属性的一个小结

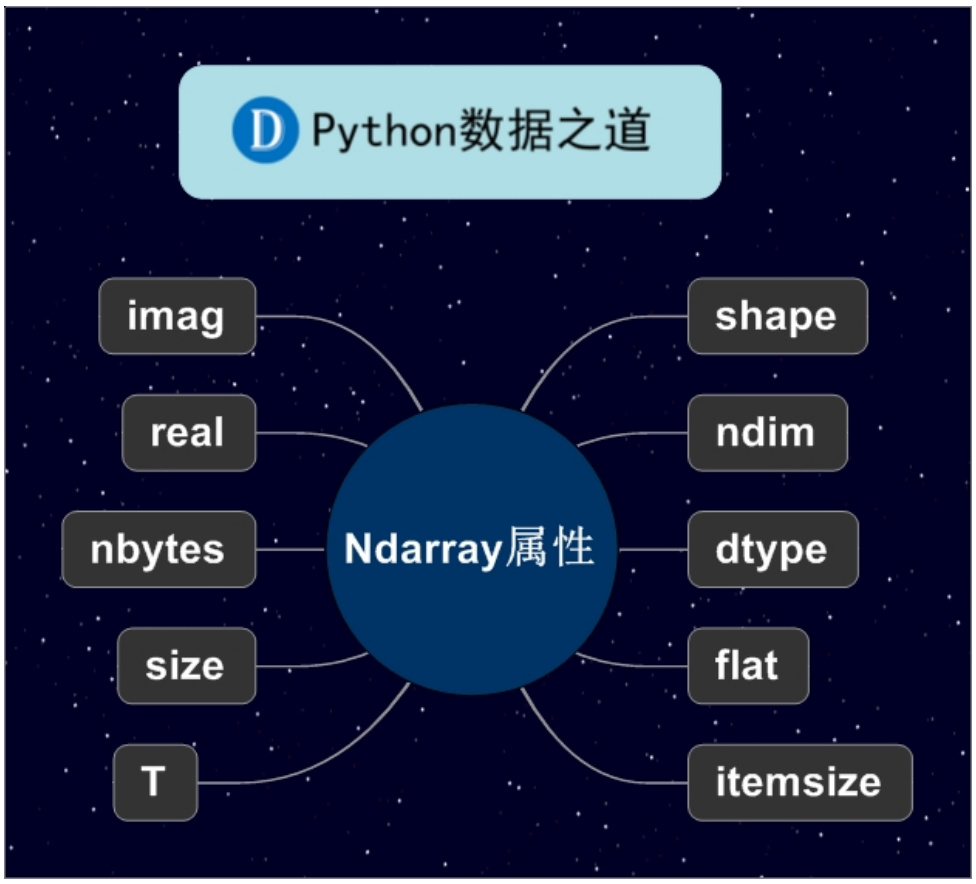


Figure 16.3: ndarray 各种属性小结

16.5 ndarray 数组的切片和索引

- **一维数组**

一维数组的切片和索引与 python 的 list 索引类似。

```
a =  np.arange(7)
a

Out[]:
    array([0, 1, 2, 3, 4, 5, 6])
```

```
a[1:4]

Out[]:
    array([1, 2, 3])
```

```
# 每间隔2个取一个数
a[ : 6: 2]
```

```
3
4 Out[]:
5     array([0, 2, 4])
```

- **二维数组的切片和索引**，如下所示：

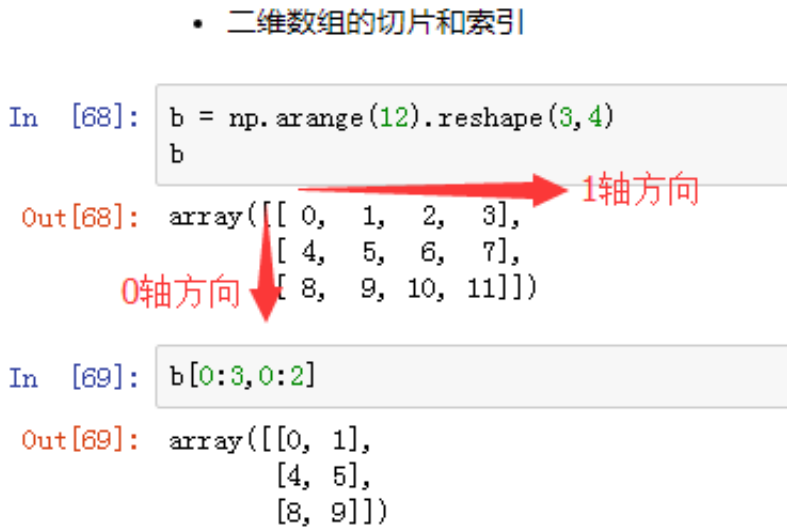


Figure 16.4: 二维数组的切片和索引

16.6 处理数组形状

16.6.1 形状转换

- **reshape() 和 resize()**

```
1 b.reshape(4,3)
2
3 Out[]:
4     array([[ 0,  1,  2],
5            [ 3,  4,  5],
6            [ 6,  7,  8],
7            [ 9, 10, 11]])
```

```
1 b
2
3 Out[]:
4     array([[ 0,  1,  2,  3],
5            [ 4,  5,  6,  7],
6            [ 8,  9, 10, 11]])
```

```
1 b.resize(4,3)
```

```
b

Out[]:
    array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
```

函数 resize（）的作用跟 reshape（）类似，但是会改变所作用的数组，相当于有 inplace=True 的效果

- **ravel() 和 flatten()**，将多维数组转换成一维数组，如下：

```
b.ravel()

Out[]:
    array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
b.flatten()

Out[]:
    array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
b

Out[]:
    array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
```

两者的区别在于返回拷贝（copy）还是返回视图（view），flatten() 返回一份拷贝，需要分配新的内存空间，对拷贝所做的修改不会影响原始矩阵，而 ravel() 返回的是视图（view），会影响原始矩阵。

参考如下代码：

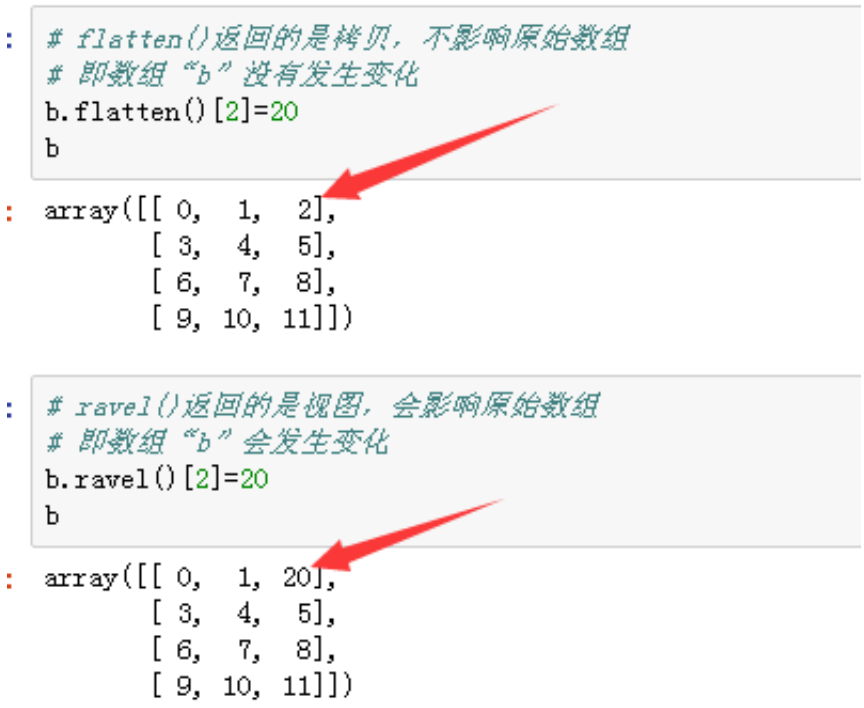


Figure 16.5: 拷贝 vs 视图

- **用 tuple 指定数组的形状**，如下：

```
b.shape=(2,6)
b

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]])
```

- **转置**

前面描述了数组转置的属性（T），也可以通过 transpose() 函数来实现

```
b.transpose()

Out[]:
    array([[ 0,  6],
           [ 1,  7],
           [20,  8],
           [ 3,  9],
           [ 4, 10],
           [ 5, 11]])
```

16.6.2 6.2 堆叠数组

```
b

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]])
```

```
c = b*2
c

Out[]:
    array([[ 0,  2, 40,  6,  8, 10],
           [12, 14, 16, 18, 20, 22]])
```

- **水平叠加**

hstack()

```
np.hstack((b,c))

Out[]:
    array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
           [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

column_stack() 函数以列方式对数组进行叠加，功能类似 hstack ()

```
np.column_stack((b,c))

Out[]:
    array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
           [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

- **垂直叠加**

vstack()

```
np.vstack((b,c))

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11],
           [ 0,  2, 40,  6,  8, 10],
           [12, 14, 16, 18, 20, 22]])
```

row_stack() 函数以行方式对数组进行叠加，功能类似 vstack ()

```
np.row_stack((b,c))

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11],
           [ 0,  2, 40,  6,  8, 10],
```

```
7               [12, 14, 16, 18, 20, 22]])
```

- **concatenate() 方法，通过设置 axis 的值来设置叠加方向**

axis=1 时，沿水平方向叠加

axis=0 时，沿垂直方向叠加

```
1  np.concatenate((b,c),axis=1)
2
3  Out[]:
4      array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
5             [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

```
1  np.concatenate((b,c),axis=0)
2
3  Out[]:
4      array([[ 0,  1, 20,  3,  4,  5],
5             [ 6,  7,  8,  9, 10, 11],
6             [ 0,  2, 40,  6,  8, 10],
7             [12, 14, 16, 18, 20, 22]])
```

由于针对数组的轴为 0 或 1 的方向经常会混淆，通过示意图，或许可以更好的理解。关于数组的轴方向示意图，以及叠加的示意图，如下：

Figure 16.6: 数组的轴方向示意图

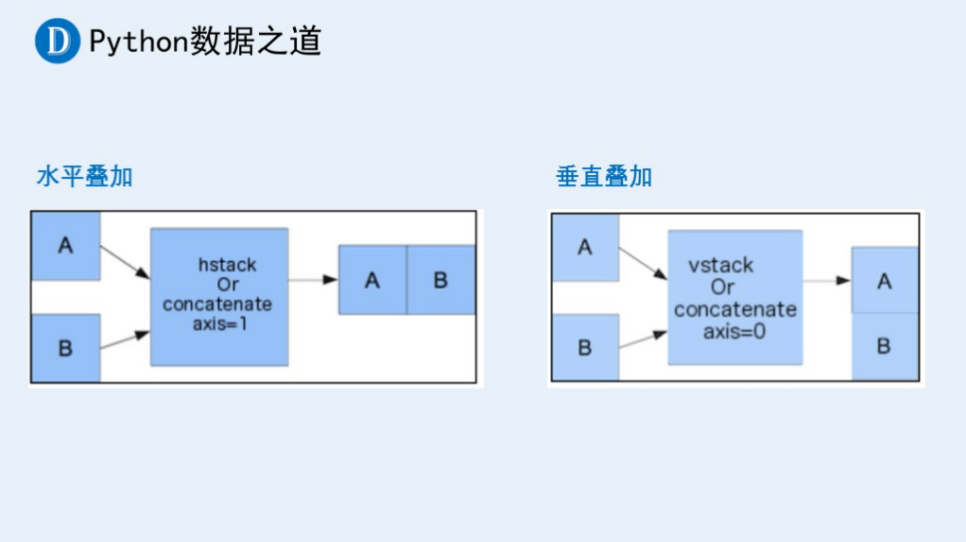


Figure 16.7: 叠加的示意图

深度叠加

这个有点烧脑，举个例子如下，自己可以体会下：

```
arr_dstack = np.dstack((b,c))
print(arr_dstack.shape)
arr_dstack

Out[]:
    (2, 6, 2)

    array([[[ 0,  0],
            [ 1,  2],
            [20, 40],
            [ 3,  6],
            [ 4,  8],
            [ 5, 10]],

           [[ 6, 12],
            [ 7, 14],
            [ 8, 16],
            [ 9, 18],
            [10, 20],
            [11, 22]]])
```

叠加前，b 和 c 均是 shape 为（2,6）的二维数组，叠加后，arr_dstack 是 shape 为（2,6,2）的三维数组。

深度叠加的示意图如下：

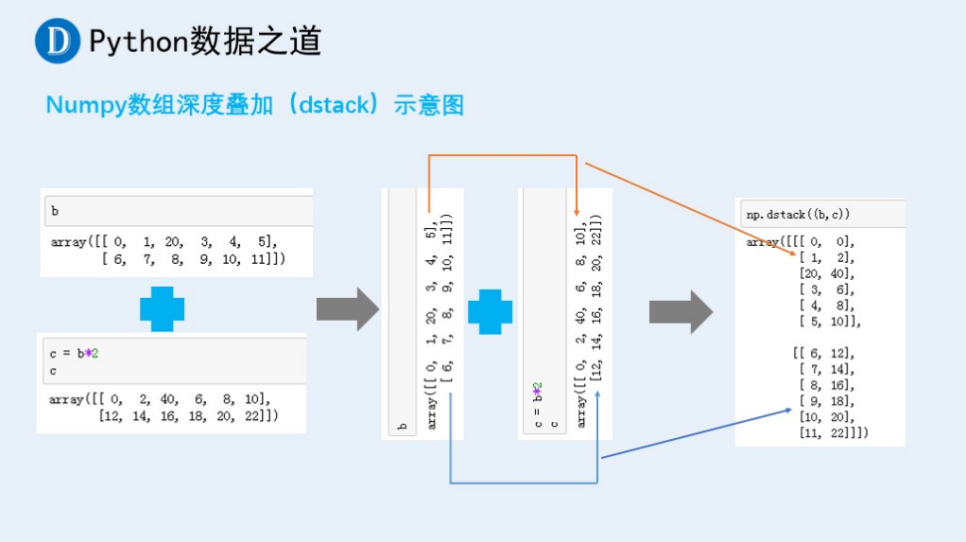


Figure 16.8: 深度叠加的示意图

16.6.3 数组的拆分

跟数组的叠加类似，数组的拆分可以分为横向拆分、纵向拆分以及深度拆分。涉及的函数为 `hsplit()`、`vsplit()`、`dsplit()` 以及 `split()`

```
b

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]])
```

- **沿横向轴拆分（axis=1）**

```
np.hsplit(b, 2)

Out[]:
    [array([[ 0,  1, 20],
            [ 6,  7,  8]]), array([[ 3,  4,  5],
            [ 9, 10, 11]])]
```

```
np.split(b,2, axis=1)

Out[]:
    [array([[ 0,  1, 20],
            [ 6,  7,  8]]), array([[ 3,  4,  5],
            [ 9, 10, 11]])]
```

- **沿纵向轴拆分（axis=0）**

```
np.vsplit(b, 2)

Out[]:
    [array([[ 0,  1, 20,  3,  4,  5]]), array([[ 6,  7,  8,  9, 10, 11]])]
```

```
np.split(b,2,axis=0)

Out[]:
    [array([[ 0,  1, 20,  3,  4,  5]]), array([[ 6,  7,  8,  9, 10, 11]])]
```

- **深度拆分**

```
arr_dstack

Out[]:
    array([[[ 0,  0],
            [ 1,  2],
            [20, 40],
            [ 3,  6],
            [ 4,  8],
            [ 5, 10]],

           [[ 6, 12],
            [ 7, 14],
            [ 8, 16],
            [ 9, 18],
            [10, 20],
            [11, 22]]])
```

```
np.dsplit(arr_dstack,2)

Out[]:
    [array([[[ 0],
             [ 1],
             [20],
             [ 3],
             [ 4],
             [ 5]],

            [[ 6],
             [ 7],
             [ 8],
             [ 9],
             [10],
             [11]]]), array([[[ 0],
             [ 2],
             [40],
             [ 6],
             [ 8],
             [10]],

```

```
23              [[12],
24               [14],
25               [16],
26               [18],
27               [20],
28               [22]])]
```

拆分的结果是原来的三维数组拆分成为两个二维数组。这个烧脑的拆分过程可以自行分析下 ~~

16.7 数组的类型转换

- **数组转换成 list，使用 tolist()**

```
1  b
2
3  Out[]:
4      array([[ 0,  1, 20,  3,  4,  5],
5             [ 6,  7,  8,  9, 10, 11]])
```

```
1  b.tolist()
2
3  Out[]:
4      [[0, 1, 20, 3, 4, 5], [6, 7, 8, 9, 10, 11]]
```

- **转换成指定类型，astype() 函数**

```
1  b.astype(float)
2
3  Out[]:
4      array([[  0.,   1.,  20.,   3.,   4.,   5.],
5             [  6.,   7.,   8.,   9.,  10.,  11.]])
```

16.8 numpy 常用统计函数

常用的函数如下，**请注意函数在使用时需要指定 axis 轴的方向**，若不指定，默认统计整个数组。

- np.sum()，返回求和
- np.mean()，返回均值
- np.max()，返回最大值
- np.min()，返回最小值
- np.ptp()，数组沿指定轴返回最大值减去最小值，即（max-min）
- np.std()，返回标准偏差（standard deviation）
- np.var()，返回方差（variance）
- np.cumsum()，返回累加值
- np.cumprod()，返回累乘积值

```
b

Out[]:
    array([[ 0,  1, 20,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]])
```

```
np.max(b)

Out[]:
    20
```

```
# 沿axis=1轴方向统计
np.max(b,axis=1)

Out[]:
    array([20, 11])
```

```
# 沿axis=0轴方向统计
np.max(b,axis=0)

Out[]:
    array([ 6,  7, 20,  9, 10, 11])
```

```
np.min(b)

Out[]:
    0
```

- **np.ptp()**，返回整个数组的最大值减去最小值，如下：

```
np.ptp(b)

Out[]:
    20
```

```
# 沿axis=0轴方向
np.ptp(b, axis=0)

Out[]:
    array([ 6,  6, 12,  6,  6,  6])
```

```
# 沿axis=1轴方向
np.ptp(b, axis=1)

Out[]:
    array([20,  5])
```

- **np.cumsum()**，沿指定轴方向进行累加

```
b.resize(4,3)
b

Out[]:
    array([[ 0,  1, 20],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
```

```
np.cumsum(b, axis=1)

Out[]:
    array([[ 0,  1, 21],
           [ 3,  7, 12],
           [ 6, 13, 21],
           [ 9, 19, 30]], dtype=int32)
```

```
np.cumsum(b, axis=0)

Out[]:
    array([[ 0,  1, 20],
           [ 3,  5, 25],
           [ 9, 12, 33],
           [18, 22, 44]], dtype=int32)
```

- **np.cumprod()**，沿指定轴方向进行累乘积（Return the cumulative product of the elements along the given axis）

```
np.cumprod(b,axis=1)

Out[]:
    array([[  0,   0,   0],
           [  3,  12,  60],
           [  6,  42, 336],
           [  9,  90, 990]], dtype=int32)
```

```
np.cumprod(b,axis=0)

Out[]:
    array([[   0,    1,   20],
           [   0,    4,  100],
           [   0,   28,  800],
           [   0,  280, 8800]], dtype=int32)
```

16.9 数组的广播

当数组跟一个标量进行数学运算时，标量需要根据数组的形状进行扩展，然后执行运算。这个扩展的过程称为“广播（broadcasting）”

```
b

Out[]:
    array([[ 0,  1, 20],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
```

```
d = b + 2
d

Out[]:
    array([[ 2,  3, 22],
           [ 5,  6,  7],
           [ 8,  9, 10],
           [11, 12, 13]])
```

16.10 写在最后

numpy 涵盖的内容其实是非常丰富的，本文仅仅介绍了 numpy 一些常用的基本功能，算是对 numpy 的一个入门级的简单的较为全面的描述。

numpy 官方的《Numpy Reference》文档，光页面数量就有 1000+ 页，如想要系统的学习 numpy，建议仔细阅读官方的参考文档，可在其官方网站进行查阅。当然，资料都是英文版的，可能看起来难度稍微大点，看习惯了就好。

> 本文涉及的代码量比较多，如需要查看**源代码**，请在微信公众号“Python 数据之道”（ID：PyDataLab）后台回复关键字“code”。

17 Numpy 的 random 函数简介

在 python 数据分析的学习和应用过程中，经常需要用到 numpy 的随机函数，由于随机函数 random 的功能比较多，经常会混淆或记不住，下面我们一起来汇总学习下。

```
import numpy as np
```

17.1 numpy.random.rand()

numpy.random.rand(d0,d1,…,dn)

- rand 函数根据给定维度生成 [0,1) 之间的数据，包含 0，不包含 1
- dn 表格每个维度
- 返回值为指定维度的 array

```
np.random.rand(4,2)

Out[]:
    array([[ 0.02173903,  0.44376568],
           [ 0.25309942,  0.85259262],
           [ 0.56465709,  0.95135013],
           [ 0.14145746,  0.55389458]])
```

```
np.random.rand(4,3,2) # shape: 4*3*2

Out[]:
    array([[[ 0.08256277,  0.11408276],
            [ 0.11182496,  0.51452019],
            [ 0.09731856,  0.18279204]],

           [[ 0.74637005,  0.76065562],
            [ 0.32060311,  0.69410458],
            [ 0.28890543,  0.68532579]],

           [[ 0.72110169,  0.52517524],
            [ 0.32876607,  0.66632414],
            [ 0.45762399,  0.49176764]],

           [[ 0.73886671,  0.81877121],
            [ 0.03984658,  0.99454548],
            [ 0.18205926,  0.99637823]]])
```

17.2 numpy.random.randn()

numpy.random.randn(d0,d1,…,dn)

- randn 函数返回一个或一组样本，具有标准正态分布。
- dn 表格每个维度
- 返回值为指定维度的 array

```
np.random.randn() # 当没有参数时，返回单个数据

Out[]:
    -1.1241580894939212
```

```
np.random.randn(2,4)

Out[]:
    array([[ 0.27795239, -2.57882503,  0.3817649 ,  1.42367345],
           [-1.16724625, -0.22408299,  0.63006614, -0.41714538]])
```

```
np.random.randn(4,3,2)

Out[]:
    array([[[ 1.27820764,  0.92479163],
            [-0.15151257,  1.3428253 ],
            [-1.30948998,  0.15493686]],

           [[-1.49645411, -0.27724089],
            [ 0.71590275,  0.81377671],
            [-0.71833341,  1.61637676]],

           [[ 0.52486563, -1.7345101 ],
            [ 1.24456943, -0.10902915],
            [ 1.27292735, -0.00926068]],

           [[ 0.88303   ,  0.46116413],
            [ 0.13305507,  2.44968809],
            [-0.73132153, -0.88586716]]])
```

标准正态分布介绍

- 标准正态分布—standard normal distribution
- 标准正态分布又称为 u 分布，是以 0 为均值、以 1 为标准差的正态分布，记为 N（0，1）。

17.3 numpy.random.randint()

17.3.1 numpy.random.randint()

numpy.random.randint(low, high=None, size=None, dtype="l")

- 返回随机整数，范围区间为 [low,high)，包含 low，不包含 high
- 参数：low 为最小值，high 为最大值，size 为数组维度大小，dtype 为数据类型，默认的数据类型是 np.int
- high 没有填写时，默认生成随机数的范围是 [0，low)

```
np.random.randint(1,size=5) # 返回[0,1)之间的整数，所以只有0

Out[]:
    array([0, 0, 0, 0, 0])
```

```
np.random.randint(1,5) # 返回1个[1,5)时间的随机整数

Out[]:
    4
```

```
np.random.randint(-5,5,size=(2,2))

Out[]:
    array([[ 2, -1],
           [ 2,  0]])
```

17.3.2 numpy.random.random_integers

numpy.random.random_integers(low, high=None, size=None)

- 返回随机整数，范围区间为 [low,high]，包含 low 和 high
- 参数：low 为最小值，high 为最大值，size 为数组维度大小
- high 没有填写时，默认生成随机数的范围是 [1，low]

该函数在最新的 numpy 版本中已被替代，建议使用 randint 函数

```
np.random.random_integers(1,size=5)

Out[]:
    array([1, 1, 1, 1, 1])
```

17.4 生成 [0,1) 之间的浮点数

- numpy.random.random_sample(size=None)
- numpy.random.random(size=None)
- numpy.random.ranf(size=None)
- numpy.random.sample(size=None)

```
print('----------random_sample--------------')
print(np.random.random_sample(size=(2,2)))
print('----------random--------------')
print(np.random.random(size=(2,2)))
```

```
print('-----------ranf--------------')
print(np.random.ranf(size=(2,2)))
print('-----------sample--------------')
print(np.random.sample(size=(2,2)))


Out[]:
    -----------random_sample--------------
    [[ 0.34966859  0.85655008]
     [ 0.16045328  0.87908218]]
    -----------random--------------
    [[ 0.25303772  0.45417512]
     [ 0.76053763  0.12454433]]
    -----------ranf--------------
    [[ 0.0379055   0.51288667]
     [ 0.71819639  0.97292903]]
    -----------sample--------------
    [[ 0.59942807  0.80211491]
     [ 0.36233939  0.12607092]]
```

17.5 numpy.random.choice()

numpy.random.choice(a, size=None, replace=True, p=None)

- 从给定的一维数组中生成随机数
- 参数：a 为一维数组类似数据或整数；size 为数组维度；p 为数组中的数据出现的概率
- a 为整数时，对应的一维数组为 np.arange(a)

```
np.random.choice(5,3)

Out[]:
    array([4, 1, 4])
```

```
np.random.choice(5, 3, replace=False)
# 当replace为False时，生成的随机数不能有重复的数值

Out[]:
    array([0, 3, 1])
```

```
np.random.choice(5,size=(3,2))

Out[]:
    array([[1, 0],
           [4, 2],
           [3, 3]])
```

```
demo_list = ['lenovo', 'sansumg','moto','xiaomi', 'iphone']
np.random.choice(demo_list,size=(3,3))
```

```
3
4  Out[]:
5      array([['moto', 'iphone', 'xiaomi'],
6             ['lenovo', 'xiaomi', 'xiaomi'],
7             ['xiaomi', 'lenovo', 'iphone']],
8            dtype='<U7')
```

- 参数 p 的长度与参数 a 的长度需要一致；
- 参数 p 为概率，p 里的数据之和应为 1

```
1  demo_list = ['lenovo', 'sansumg','moto','xiaomi', 'iphone']
2  np.random.choice(demo_list,size=(3,3), p=[0.1,0.6,0.1,0.1,0.1])
3
4  Out[]:
5      array([['sansumg', 'sansumg', 'sansumg'],
6             ['sansumg', 'sansumg', 'sansumg'],
7             ['sansumg', 'xiaomi', 'iphone']],
8            dtype='<U7')
```

17.6 numpy.random.seed()

- np.random.seed() 的作用：使得随机数据可预测。
- 当我们设置相同的 seed，每次生成的随机数相同。如果不设置 seed，则每次会生成不同的随机数

```
1  np.random.seed(0)
2  np.random.rand(5)
3
4  Out[]:
5      array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
```

```
1  np.random.seed(1676)
2  np.random.rand(5)
3
4  Out[]:
5      array([ 0.39983389,  0.29426895,  0.89541728,  0.71807369,  0.3531823 ])
```

```
1  np.random.seed(1676)
2  np.random.rand(5)
3
4  Out[]:
5      array([ 0.39983389,  0.29426895,  0.89541728,  0.71807369,  0.3531823 ])
```

18 Numpy 中 Meshgrid 函数介绍及 2 种应用场景

近期在好几个地方都看到 meshgrid 的使用，虽然之前也注意到 meshgrid 的用法。但总觉得印象不深刻，不是太了解 meshgrid 的应用场景。所以，本文将进一步介绍 Numpy 中 meshgrid 的用法。

18.1 Meshgrid 函数的基本用法

在 Numpy 的官方文章里，meshgrid 函数的英文描述也显得文绉绉的，理解起来有些难度。可以这么理解，meshgrid 函数用两个坐标轴上的点在平面上画网格。

用法：

[X,Y]=meshgrid(x,y)

[X,Y]=meshgrid(x) 与 [X,Y]=meshgrid(x,x) 是等同的

[X,Y,Z]=meshgrid(x,y,z) 生成三维数组，可用来计算三变量的函数和绘制三维立体图

这里，主要以`[X,Y]=meshgrid(x,y)`为例，来对该函数进行介绍。`[X,Y] = meshgrid(x,y)` 将向量 x 和 y 定义的区域转换成矩阵 X 和 Y，其中矩阵 X 的行向量是向量 x 的简单复制，而矩阵 Y 的列向量是向量 y 的简单复制 (注：下面代码中 X 和 Y 均是数组，在文中统一称为矩阵了)。假设 x 是长度为 m 的向量，y 是长度为 n 的向量，则最终生成的矩阵 X 和 Y 的维度都是 `n*m` （注意不是 `m*n`)。文字描述可能不是太好理解，下面通过代码演示下：

加载数据

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

m, n = (5, 3)
x = np.linspace(0, 1, m)
y = np.linspace(0, 1, n)

X, Y = np.meshgrid(x,y)
```

查看向量 x 和向量 y

```
x

out:
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

```
5
6  y
7
8  out:
9  array([ 0. ,  0.5,  1. ])
```

查看矩阵 X 和矩阵 Y

```
1  X
2
3  out:
4  array([[ 0.  ,  0.25,  0.5 ,  0.75,  1.  ],
5         [ 0.  ,  0.25,  0.5 ,  0.75,  1.  ],
6         [ 0.  ,  0.25,  0.5 ,  0.75,  1.  ]])
7
8  Y
9
10 out:
11 array([[ 0. ,  0. ,  0. ,  0. ,  0. ],
12        [ 0.5,  0.5,  0.5,  0.5,  0.5],
13        [ 1. ,  1. ,  1. ,  1. ,  1. ]])
```

查看矩阵对应的维度

```
1  X.shape
2
3  out:
4  (3, 5)
5
6  Y.shape
7
8  out:
9  (3, 5)
```

meshgrid 函数的运行过程，可以通过下面的示意图来加深理解：

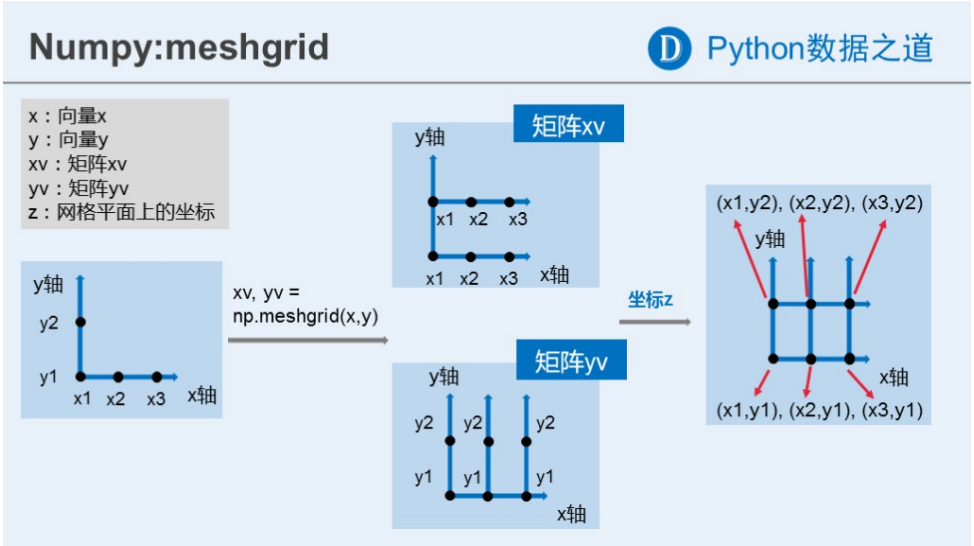


Figure 18.1: meshgrid

再者，也可以通过在 matplotlib 中进行可视化，来查看函数运行后得到的网格化数据的结果

```
plt.plot(X, Y, marker='.', color='blue', linestyle='none')
plt.show()
```

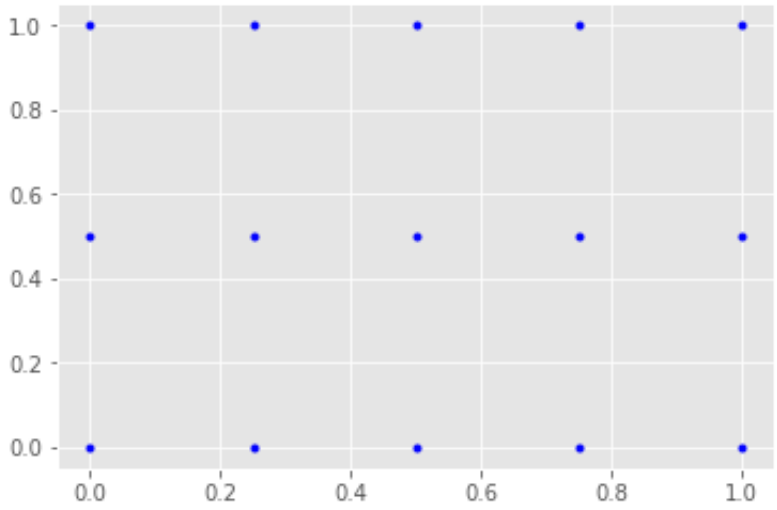


Figure 18.2: result

当然，我们也可以获得网格平面上坐标点的数据，如下：

```
z = [i for i in zip(X.flat,Y.flat)]
z
```

```
3
4  out:
5  [(0.0, 0.0),
6   (0.25, 0.0),
7   (0.5, 0.0),
8   (0.75, 0.0),
9   (1.0, 0.0),
10  (0.0, 0.5),
11  (0.25, 0.5),
12  (0.5, 0.5),
13  (0.75, 0.5),
14  (1.0, 0.5),
15  (0.0, 1.0),
16  (0.25, 1.0),
17  (0.5, 1.0),
18  (0.75, 1.0),
19  (1.0, 1.0)]
```

18.2 Meshgrid 函数的一些应用场景

Meshgrid 函数常用的场景有等高线绘制及机器学习中 SVC 超平面的绘制（二维场景下）。分别图示如下：

（1）等高线

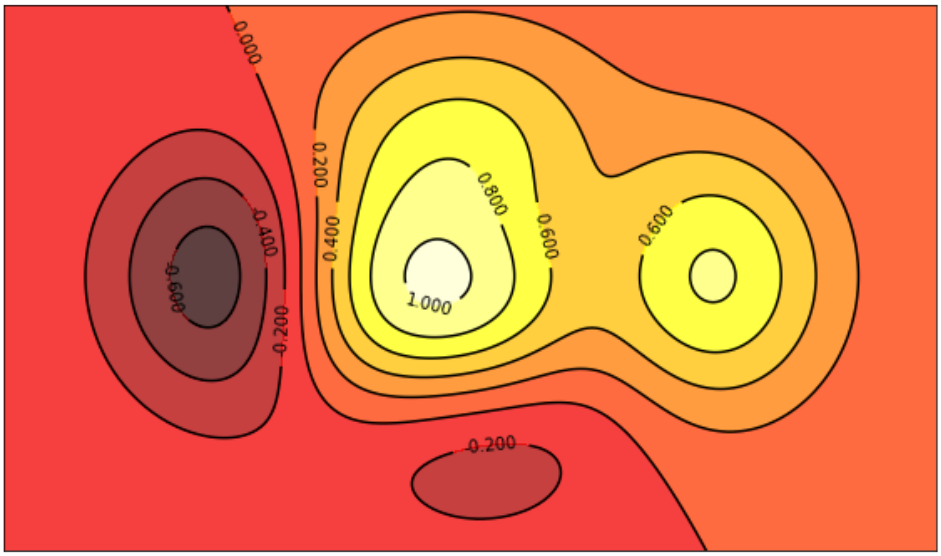


Figure 18.3: contour

（2）SVC 中超平面的绘制：

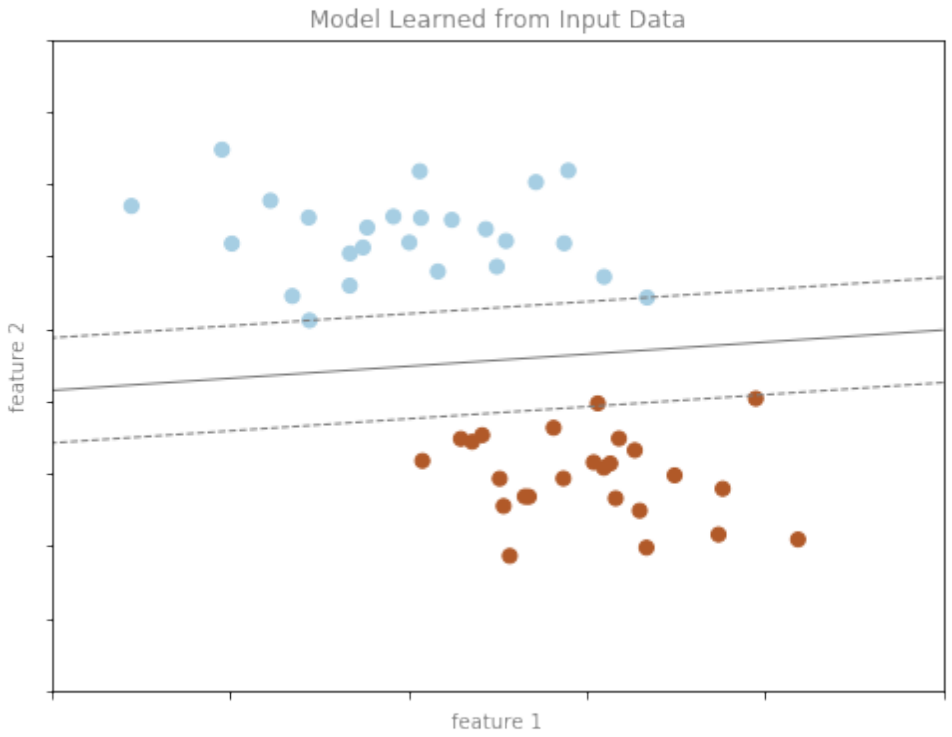
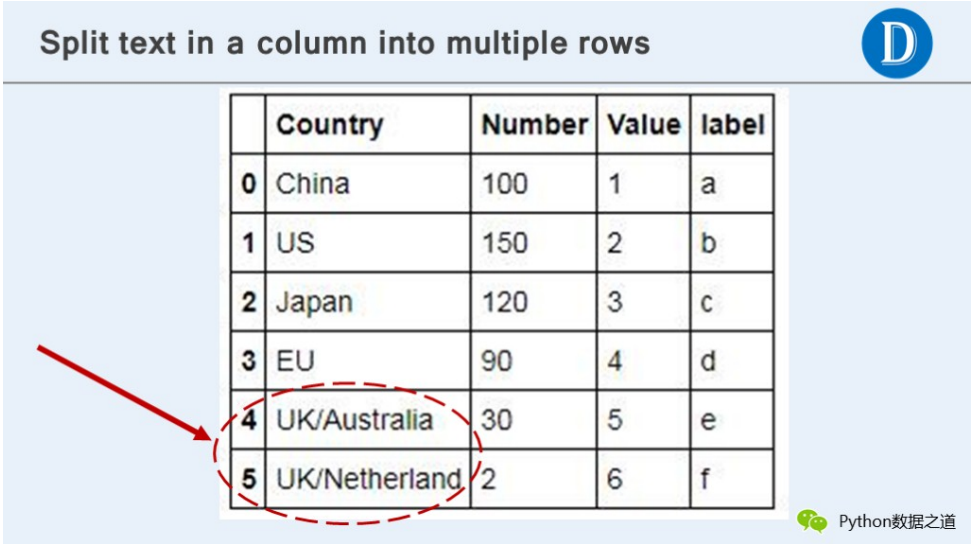


Figure 18.4: svc

关于场景（1）和场景（2），将在后续的文章里做进一步描述。当然，可能还有些其他场景，这里就不做进一步介绍了。# Pandas: 如何将一列中的文本拆分为多行?

在数据处理过程中，经常会遇到以下类型的数据：



	Country	Number	Value	label
0	China	100	1	a
1	US	150	2	b
2	Japan	120	3	c
3	EU	90	4	d
4	UK/Australia	30	5	e
5	UK/Netherland	2	6	f

Figure 18.5: 脏数据

在同一列中，本该分别填入多行中的数据，被填在一行里了，然而在分析的时候，需要拆分成为多行。在上图中，列名为“Country”，index 为 4 和 5 的单元格内，值为“UK/Australia”和“UK/Netherland”。今天，我们来介绍将含有多值的内容分拆成多行的几种方法。加载数据如下：

```
import pandas as pd

df = pd.DataFrame({'Country':['China','US','Japan','EU','UK/Australia', 'UK/
    Netherland'],
                   'Number':[100, 150, 120, 90, 30, 2],
                   'Value': [1, 2, 3, 4, 5, 6],
                   'label': list('abcdef')})
df

Out[2]:
          Country  Number  Value label
0           China     100      1     a
1              US     150      2     b
2           Japan     120      3     c
3              EU      90      4     d
4    UK/Australia      30      5     e
5  UK/Netherland       2      6     f
```

18.3 Method-1

分为如下几步：

1. 将含有多值的列进行拆分，然后通过`stack()`方法进行变换，并通过 index 的设置来完成

2. 用drop()方法从 DataFrame 中删除含有多值的列
3. 然后用join()方法来合并

```
df.drop('Country', axis=1).join(df['Country'].str.split('/', expand=True).stack
    ().reset_index(level=1, drop=True).rename('Country'))
Out[3]:
   Number  Value label     Country
0     100      1     a       China
1     150      2     b          US
2     120      3     c       Japan
3      90      4     d          EU
4      30      5     e          UK
4      30      5     e   Australia
5       2      6     f          UK
5       2      6     f  Netherland
```

过程分步介绍

```
df['Country'].str.split('/', expand=True).stack()
Out[4]:
0  0         China
1  0            US
2  0         Japan
3  0            EU
4  0            UK
   1     Australia
5  0            UK
   1    Netherland
dtype: object

df['Country'].str.split('/', expand=True).stack().reset_index(level=1, drop=
    True)
Out[5]:
0         China
1            US
2         Japan
3            EU
4            UK
4     Australia
5            UK
5    Netherland
dtype: object

df['Country'].str.split('/', expand=True).stack().reset_index(level=1, drop=
    True).rename('Country')
Out[6]:
0         China
1            US
2         Japan
3            EU
4            UK
```

```
4     Australia
5            UK
5    Netherland
Name: Country, dtype: object

df.drop('Country', axis=1)
Out[7]:
   Number  Value label
0     100      1     a
1     150      2     b
2     120      3     c
3      90      4     d
4      30      5     e
5       2      6     f
```

18.4 Method-2

该方法的思路跟 Method-1 基本是一样的，只是在具体的细节方面有些差异。代码如下：

```

df['Country'].str.split('/', expand=True).stack().reset_index(level=0).
    set_index('level_0').rename(columns={0:'Country'}).join(df.drop('Country',
    axis=1))
Out[8]:
      Country  Number  Value label
0       China     100      1     a
1          US     150      2     b
2       Japan     120      3     c
3          EU      90      4     d
4          UK      30      5     e
4   Australia      30      5     e
5          UK       2      6     f
5  Netherland       2      6     f
```

过程分步介绍如下：

```
df['Country'].str.split('/', expand=True).stack().reset_index(level=0)
Out[9]:
   level_0           0
0        0       China
0        1          US
0        2       Japan
0        3          EU
0        4          UK
1        4   Australia
0        5          UK
1        5  Netherland

```

```
df['Country'].str.split('/', expand=True).stack().reset_index(level=0).
    set_index('level_0')
Out[10]:
                  0
level_0
0             China
1                US
2             Japan
3                EU
4                UK
4          Australia
5                UK
5         Netherland

df['Country'].str.split('/', expand=True).stack().reset_index(level=0).
    set_index('level_0').rename(columns={0:'Country'})
Out[11]:
             Country
level_0
0              China
1                 US
2              Japan
3                 EU
4                 UK
4           Australia
5                 UK
5          Netherland

df.drop('Country', axis=1)
Out[12]:
   Number  Value label
0     100      1     a
1     150      2     b
2     120      3     c
3      90      4     d
4      30      5     e
5       2      6     f
```

闲谈

当然，将某列中含有多值的单元拆分成多行，还有其他方法，各位小伙伴们可以研究下 ~~

19 Pandas 日期数据处理

Pandas 有着强大的日期数据处理功能，本期我们来了解下 pandas 处理日期数据的一些基本功能，主要包括以下三个方面：

- 按日期筛选数据
- 按日期显示数据
- 按日期统计数据

运行环境为 windows 系统，64 位，python3.5。

19.1 读取并整理数据

- 首先引入 pandas 库

```
import pandas as pd
```

- 从 csv 文件中读取数据

```
df = pd.read_csv('date.csv', header=None)
print(df.head(2))
```

```
            0  1
0  2013-10-24  3
1  2013-10-25  4
```

- 整理数据

```
df.columns = ['date','number']
df['date'] = pd.to_datetime(df['date']) #将数据类型转换为日期类型
df = df.set_index('date') #  将date设置为index
print(df.head(2))
print(df.tail(2))
print(df.shape)
```

```
            number
date
2013-10-24       3
2013-10-25       4
            number
date
```

```
2017-02-14       6
2017-02-22       6
(425, 1)
```

- df 的行数一共是 425 行。

查看 Dataframe 的数据类型

```
print(type(df))
print(df.index)
print(type(df.index))
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex(['2013-10-24', '2013-10-25', '2013-10-29', '2013-10-30',
               '2013-11-04', '2013-11-06', '2013-11-08', '2013-11-12',
               '2013-11-14', '2013-11-25',
               ...
               '2017-01-03', '2017-01-07', '2017-01-14', '2017-01-17',
               '2017-01-23', '2017-01-25', '2017-01-26', '2017-02-07',
               '2017-02-14', '2017-02-22'],
              dtype='datetime64[ns]', name='date', length=425, freq=None)
<class 'pandas.tseries.index.DatetimeIndex'>
```

构造 Series 类型数据

```
s = pd.Series(df['number'], index=df.index)
print(type(s))
s.head(2)
```

```
<class 'pandas.core.series.Series'>

date
2013-10-24    3
2013-10-25    4
Name: number, dtype: int64
```

19.2 按日期筛选数据

按年度获取数据

```
print('---------获取2013年的数据-----------')
print(df['2013'].head(2)) # 获取2013年的数据
print(df['2013'].tail(2)) # 获取2013年的数据
```

```
---------获取2013年的数据-----------
            number
date
2013-10-24       3
2013-10-25       4
```

```
            number
date
2013-12-27       2
2013-12-30       2
```

获取 2016 至 2017 年的数据

```
print('----------获取2016至2017年的数据-----------')
print(df['2016':'2017'].head(2))  #获取2016至2017年的数据
print(df['2016':'2017'].tail(2))  #获取2016至2017年的数据
```

```
----------获取2016至2017年的数据-----------
            number
date
2016-01-04       4
2016-01-07       6
            number
date
2017-02-14       6
2017-02-22       6
```

获取某月的数据

```
print('----------获取某月的数据-----------')
print(df['2013-11']) # 获取某月的数据
```

```
----------获取某月的数据-----------
            number
date
2013-11-04       1
2013-11-06       3
2013-11-08       1
2013-11-12       5
2013-11-14       2
2013-11-25       1
2013-11-29       1
```

获取具体某天的数据

- 请注意 dataframe 类型的数据，获取具体某天的数据时，跟 series 是有些差异的，详细情况如下述代码所示：

```
# 按日期筛选数据
print('----------获取具体某天的数据-----------')
# 获取具体某天的数据
print(s['2013-11-06'])

# 获取具体某天的数据，用datafrme直接选取某天时会报错，而series的数据就没有问题
# print(df['2013-11-06'])

#可以考虑用区间来获取某天的数据
print(df['2013-11-06':'2013-11-06'])
```

```
---------获取具体某天的数据-----------
3
            number
date
2013-11-06       3
```

- dataframe 的 truncate 函数可以获取某个时期之前或之后的数据，或者某个时间区间的数据
- 但一般建议直接用切片（slice），这样更为直观，方便

```
# dataframe的truncate函数可以获取某个时期之前或之后的数据，或者某个时间区间的数据
# 但一般建议直接用切片（slice），这样更为直观，方便
print('---------获取某个时期之前或之后的数据-----------')
print('--------after------------')
print(df.truncate(after = '2013-11'))
print('--------before------------')
print(df.truncate(before='2017-02'))
```

```
---------获取某个时期之前或之后的数据-----------
--------after------------
            number
date
2013-10-24       3
2013-10-25       4
2013-10-29       2
2013-10-30       1
--------before------------
            number
date
2017-02-07       8
2017-02-14       6
2017-02-22       6
```

> 2020 年 12 月补充：
>
> 关于日期数据筛选，这里有一点需要提醒下，就是先要对日期数据进行排序，不然，类似 `df["xxx":"xxx"]` 这种写法会报错。此外，还可以使用 `df.loc["xxx":"xxx"]` 这种方法来进行日期数据的筛选

19.3 按日期显示数据

19.3.1 to_period() 方法

- 请注意 df.index 的数据类型是 DatetimeIndex;
- df_peirod 的数据类型是 PeriodIndex

按月显示，但不统计

```
df_period = df.to_period('M') #按月显示，但不统计
```

```
print(type(df_period))

print(type(df_period.index))
# 请注意df.index的数据类型是DatetimeIndex;
# df_peirod的数据类型是PeriodIndex

print(df_period.head())
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.tseries.period.PeriodIndex'>
         number
date
2013-10       3
2013-10       4
2013-10       2
2013-10       1
2013-11       1
```

按季度显示，但不统计

```
print(df.to_period('Q').head()) #按季度显示，但不统计
```

```
        number
date
2013Q4       3
2013Q4       4
2013Q4       2
2013Q4       1
2013Q4       1
```

按年度显示，但不统计

```
print(df.to_period('A').head()) #按年度显示，但不统计
```

```
      number
date
2013       3
2013       4
2013       2
2013       1
2013       1
```

19.3.2 asfreq() 方法

按年度频率显示

```
df_period.index.asfreq('A') # 'A'默认是'A-DEC',其他如'A-JAN'
```

```
PeriodIndex(['2013', '2013', '2013', '2013', '2013', '2013', '2013', '2013',
```

```
             '2013', '2013',
             ...
             '2017', '2017', '2017', '2017', '2017', '2017', '2017', '2017',
             '2017', '2017'],
            dtype='period[A-DEC]', name='date', length=425, freq='A-DEC')
```

```
df_period.index.asfreq('A-JAN') # 'A'默认是'A-DEC',其他如'A-JAN'
```

```
PeriodIndex(['2014', '2014', '2014', '2014', '2014', '2014', '2014', '2014',
             '2014', '2014',
             ...
             '2017', '2017', '2017', '2017', '2017', '2017', '2017', '2018',
             '2018', '2018'],
            dtype='period[A-JAN]', name='date', length=425, freq='A-JAN')
```

- 按年度频率在不同情形下的显示，可参考下图所示：

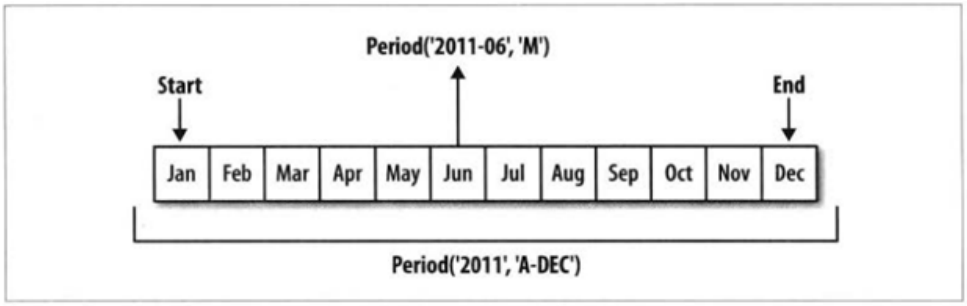


Figure 19.1: Period

按季度频率显示

```
df_period.index.asfreq('Q') # 'Q'默认是'Q-DEC',其他如“Q-SEP”，“Q-FEB”
```

```
PeriodIndex(['2013Q4', '2013Q4', '2013Q4', '2013Q4', '2013Q4', '2013Q4',
             '2013Q4', '2013Q4', '2013Q4', '2013Q4',
             ...
             '2017Q1', '2017Q1', '2017Q1', '2017Q1', '2017Q1', '2017Q1',
             '2017Q1', '2017Q1', '2017Q1', '2017Q1'],
            dtype='period[Q-DEC]', name='date', length=425, freq='Q-DEC')
```

```
df_period.index.asfreq('Q-SEP') # 可以显示不同的季度财年，“Q-SEP”，“Q-FEB”
# df_period.index = df_period.index.asfreq('Q-DEC') # 可以显示不同的季度财
    年，“Q-SEP”，“Q-FEB”
# print(df_period.head())
```

```
PeriodIndex(['2014Q1', '2014Q1', '2014Q1', '2014Q1', '2014Q1', '2014Q1',
             '2014Q1', '2014Q1', '2014Q1', '2014Q1',
             ...
             '2017Q2', '2017Q2', '2017Q2', '2017Q2', '2017Q2', '2017Q2',
```

```
             '2017Q2', '2017Q2', '2017Q2', '2017Q2'],
            dtype='period[Q-SEP]', name='date', length=425, freq='Q-SEP')
```

- 按季度频率在不同情形下的显示，可参考下图所示：

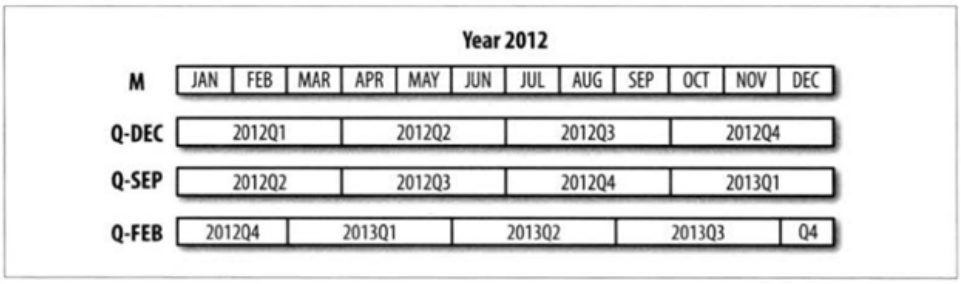


Figure 19.2: Season

按月度频率显示

```
df_period.index.asfreq('M') # 按月份显示
```

```
PeriodIndex(['2013-10', '2013-10', '2013-10', '2013-10', '2013-11', '2013-11',
             '2013-11', '2013-11', '2013-11', '2013-11',
             ...
             '2017-01', '2017-01', '2017-01', '2017-01', '2017-01', '2017-01',
             '2017-01', '2017-02', '2017-02', '2017-02'],
            dtype='period[M]', name='date', length=425, freq='M')
```

按工作日显示

- method 1

```
df_period.index.asfreq('B', how='start') # 按工作日期显示
```

```
PeriodIndex(['2013-10-01', '2013-10-01', '2013-10-01', '2013-10-01',
             '2013-11-01', '2013-11-01', '2013-11-01', '2013-11-01',
             '2013-11-01', '2013-11-01',
             ...
             '2017-01-02', '2017-01-02', '2017-01-02', '2017-01-02',
             '2017-01-02', '2017-01-02', '2017-01-02', '2017-02-01',
             '2017-02-01', '2017-02-01'],
            dtype='period[B]', name='date', length=425, freq='B')
```

- method 2

```
df_period.index.asfreq('B', how='end') # 按工作日期显示
```

```
PeriodIndex(['2013-10-31', '2013-10-31', '2013-10-31', '2013-10-31',
             '2013-11-29', '2013-11-29', '2013-11-29', '2013-11-29',
             '2013-11-29', '2013-11-29',
```

```
             ...
             '2017-01-31', '2017-01-31', '2017-01-31', '2017-01-31',
             '2017-01-31', '2017-01-31', '2017-01-31', '2017-02-28',
             '2017-02-28', '2017-02-28'],
            dtype='period[B]', name='date', length=425, freq='B')
```

19.4 按日期统计数据

19.4.1 按日期统计数据

按周统计数据

```
print(df.resample('w').sum().head())
#  “w” , week
```

```
            number
date
2013-10-27     7.0
2013-11-03     3.0
2013-11-10     5.0
2013-11-17     7.0
2013-11-24     NaN
```

按月统计数据

```
print(df.resample('M').sum().head())
# "MS"是每个月第一天为开始日期, "M"是每个月最后一天
```

```
            number
date
2013-10-31      10
2013-11-30      14
2013-12-31      27
2014-01-31      16
2014-02-28       4
```

按季度统计数据

```
print(df.resample('Q').sum().head())
# "QS"是每个季度第一天为开始日期, "Q"是每个季度最后一天
```

```
            number
date
2013-12-31      51
2014-03-31      73
2014-06-30      96
2014-09-30     136
2014-12-31     148
```

按年统计数据

```
1
2 print(df.resample('AS').sum())
3 # "AS"是每年第一天为开始日期，"A是每年最后一天
```

```
1             number
2 date
3 2013-01-01      51
4 2014-01-01     453
5 2015-01-01     743
6 2016-01-01    1552
7 2017-01-01      92
```

- 关于日期的类型，按参考下图所示来选择合适的分期频率：

Alias	Description
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency
BQ	business quarter endfrequency
QS	quarter start frequency
BQS	business quarter start frequency
A	year end frequency
BA	business year end frequency
AS	year start frequency
BAS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

Figure 19.3: Data Type

19.4.2 按日期统计后，按年或季度或月份显示

按年统计并显示

```
print(df.resample('AS').sum().to_period('A'))
# 按年统计并显示
```

```
      number
date
2013      51
2014     453
```

```
2015     743
2016    1552
2017      92
```

按季度统计并显示

```
print(df.resample('Q').sum().to_period('Q').head())
# 按季度统计并显示
```

```
        number
date
2013Q4      51
2014Q1      73
2014Q2      96
2014Q3     136
2014Q4     148
```

按月度统计并显示

```
print(df.resample('M').sum().to_period('M').head())
# 按月度统计并显示
```

```
         number
date
2013-10      10
2013-11      14
2013-12      27
2014-01      16
2014-02       4
```

20 Pandas 的 DataFrame 如何按指定 list 排序

写这篇文章的起由是有一天微信上一位朋友问到一个问题，问题大体意思概述如下：

> 现在有一个 pandas 的 Series 和一个 python 的 list ，想让 Series 按指定的 list 进行排序，如何实现?

这个问题的需求用流程图描述如下：

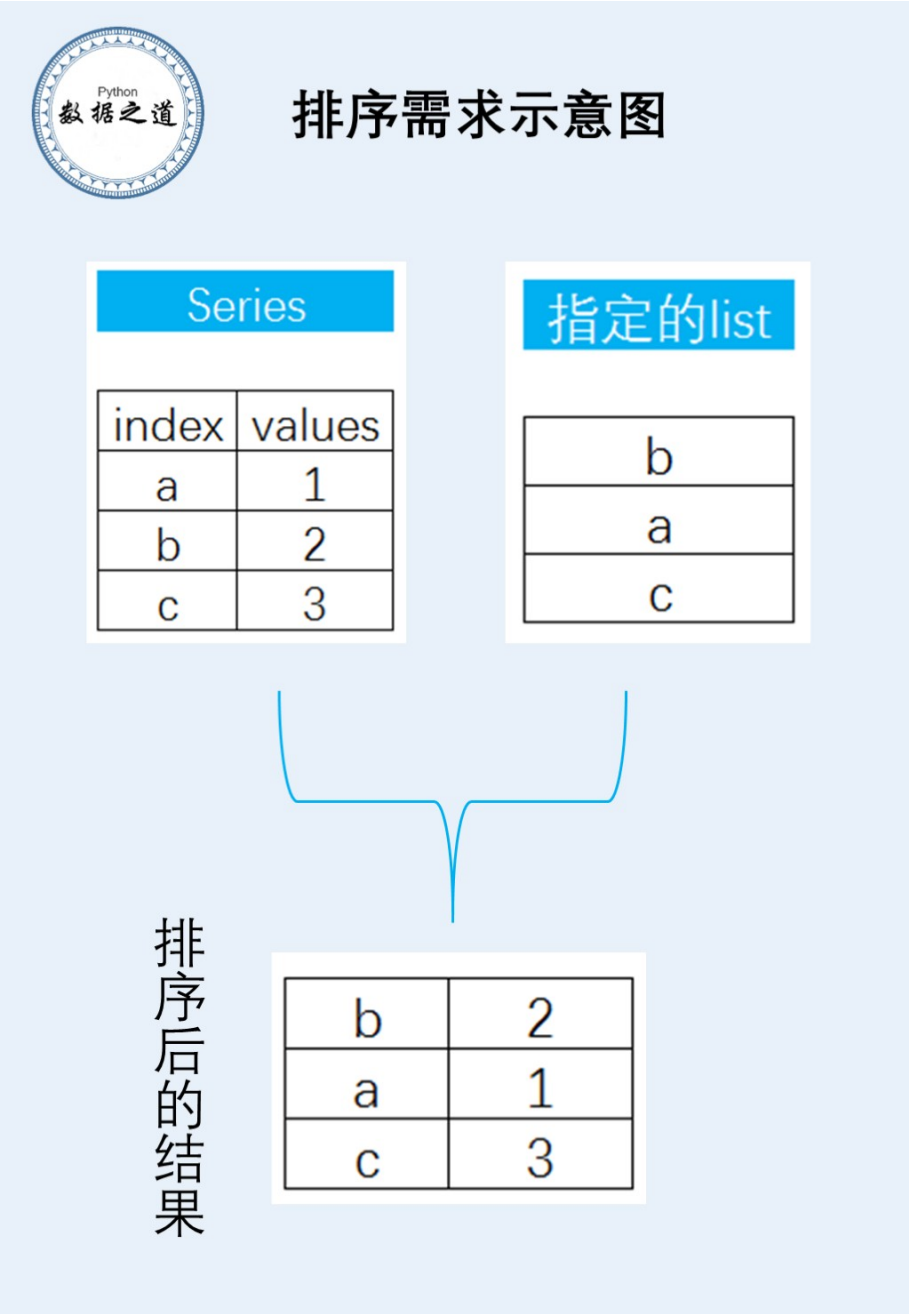


Figure 20.1: 排序需求示意图

我思考了一下，这个问题解决的核心是引入 pandas 的数据类型 “category”，从而进行排序。在具体的分析过程中，先将 pandas 的 Series 转换成为 DataFrame ，然后设置数据类型，再进行排序。思路用流程图表示如下：

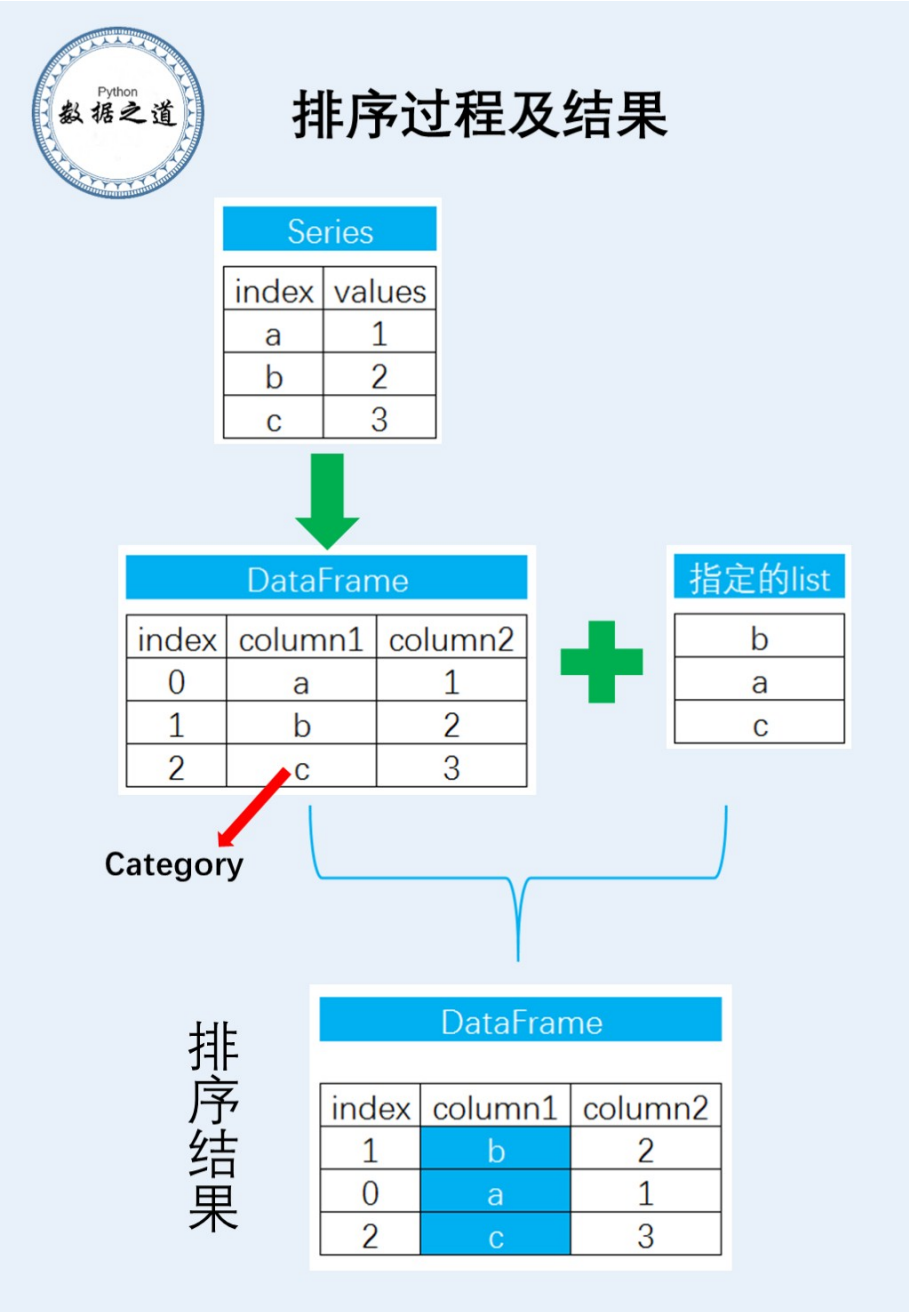


Figure 20.2: 排序过程及结果

20.1 分析过程

- 引入 pandas 库

```
1 import pandas as pd
```

- 构造 Series 数据

```
s = pd.Series({'a':1,'b':2,'c':3})
s

Out[]:
    a    1
    b    2
    c    3
    dtype: int64
```

```
s.index

Out[]:
    Index(['a', 'b', 'c'], dtype='object')
```

- 指定的 list，后续按指定 list 的元素顺序进行排序

```
list_custom = ['b', 'a', 'c']
list_custom

Out[]:
    ['b', 'a', 'c']
```

- 将 Series 转换成 DataFrame

```
df = pd.DataFrame(s)
df = df.reset_index()
df.columns = ['words', 'number']
df
```

	words	number
0	a	1
1	b	2
2	c	3

Figure 20.3: demo

设置成 “category” 数据类型

```
# 设置成“category”数据类型
df['words'] = df['words'].astype('category')
```

```
# inplace = True，使 recorder_categories生效
df['words'].cat.reorder_categories(list_custom, inplace=True)

# inplace = True，使 df生效
df.sort_values('words', inplace=True)
df
```

	words	number
1	b	2
0	a	1
2	c	3

Figure 20.4: demo

20.2 指定 list 元素多的情况:

若指定的 list 所包含元素比 Dataframe 中需要排序的列的元素 **多**，怎么办?

- reorder_catgories（）方法不能继续使用，因为该方法使用时要求新的 categories 和 dataframe 中的 categories 的元素个数和内容必须一致，只是顺序不同。
- 这种情况下，可以使用 set_categories() 方法来实现。新的 list 可以比 dataframe 中元素多。

```
list_custom_new = ['d', 'c', 'b','a','e']
dict_new = {'e':1, 'b':2, 'c':3}
df_new = pd.DataFrame(list(dict_new.items()), columns=['words', 'value'])
print(list_custom_new)
df_new.sort_values('words', inplace=True)
df_new
```

```
['d', 'c', 'b', 'a', 'e']
```

	words	value
0	b	2
1	c	3
2	e	1

Figure 20.5: demo

```
df_new['words'] = df_new['words'].astype('category')

# inplace = True，使 set_categories生效
df_new['words'].cat.set_categories(list_custom_new, inplace=True)

df_new.sort_values('words', ascending=True)
```

	words	value
1	c	3
0	b	2
2	e	1

Figure 20.6: demo

20.3 指定 list 元素少的情况：

若指定的 list 所包含元素比 Dataframe 中需要排序的列的元素 **少**，怎么办?

- 这种情况下，set_categories() 方法还是可以使用的，只是没有的元素会以 NaN 表示

注意下面的 list 中没有元素“b”

```
list_custom_new = ['d', 'c','a','e']
dict_new = {'e':1, 'b':2, 'c':3}
df_new = pd.DataFrame(list(dict_new.items()), columns=['words', 'value'])
print(list_custom_new)
df_new.sort_values('words', inplace=True)
df_new
```

```
['d', 'c', 'a', 'e']
```

	words	value
0	b	2
1	c	3
2	e	1

Figure 20.7: demo

```
df_new['words'] = df_new['words'].astype('category')

# inplace = True，使 set_categories 生效
df_new['words'].cat.set_categories(list_custom_new, inplace=True)

df_new.sort_values('words', ascending=True)
```

	words	value
0	NaN	2
1	c	3
2	e	1

Figure 20.8: demo

20.4 总结

根据指定的 list 所包含元素比 Dataframe 中需要排序的列的元素的多或少，可以分为三种情况：

- 相等的情况下，可以使用 reorder_categories 和 set_categories 方法；
- list 的元素比较多的情况下，可以使用 set_categories 方法；
- list 的元素比较少的情况下，也可以使用 set_categories 方法，但 list 中没有的元素会在 DataFrame 中以 NaN 表示。

> 源代码：在微信公众号“Python 数据之道”（ID：PyDataLab）后台回复关键字“code”获取。

21 Pandas 小册子：根据条件创建新的列

在进行数据分析时，经常会遇到根据已有的数据列，按照一定条件创建新的数据列，然后进行进一步分析。今天，我们来看一个根据已有数据按照一定条件创建新的数据列的方法。数据如下：

```
import pandas as pd

df = pd.DataFrame({'team_A': ['Spain', 'Germany', 'Brazil', 'France'],
                   'team_B': ['USA', 'Argentina', 'Mexico', 'Belgium'],
                   'score_A': [5, 3, 2, 0],
                   'score_B': [4, 0, 3, 0]},
                  columns = ['team_A', 'team_B', 'score_A', 'score_B'])
df

Out[2]:
    team_A     team_B  score_A  score_B
0    Spain        USA        5        4
1  Germany  Argentina        3        0
2   Brazil     Mexico        2        3
3   France    Belgium        0        0
```

问题：从上面数据中创建新的一个数据列，用来存储获胜队伍的名称。即，根据“score_A”与“score_B”比较的结果，来获取相应的结果。

例如，第一行，“Spain”:“USA”为 5:4，“Spain”获胜，新创建的列中存储的结果为“Spain”。下面介绍两种方法来实现上述要求。

21.1 第一种方法

第一种方法是利用 Pandas 中 DataFrame 的条件选择功能来实现，过程如下：

```
# 创建新的列 "win_team"，赋值为空白
df['win_team'] = ''

# 创建判断条件 mask
mask = df['score_A'] - df['score_B']
df.loc[mask > 0, 'win_team'] = df.loc[mask > 0, 'team_A']
df.loc[mask < 0, 'win_team'] = df.loc[mask < 0, 'team_B']
df.loc[mask == 0, 'win_team'] = 'Draw'

df
```

```
Out[3]:
    team_A     team_B  score_A  score_B win_team
0    Spain        USA        5        4    Spain
1  Germany  Argentina        3        0  Germany
2   Brazil     Mexico        2        3   Mexico
3   France    Belgium        0        0     Draw
```

21.2 第二种方法

第二种方法是结合 DataFrame.iterrows() 以及 Python 的 list 的功能来实现，过程如下：

```
# The second method to get the winners

def find_win_team(df):
    winners = []
    for i, row in df.iterrows():
        if row['score_A'] > row['score_B']:
            winners.append(row['team_A'])
        elif row['score_A'] < row['score_B']:
            winners.append(row['team_B'])
        else:
            winners.append('Draw')
    return winners

df['winner'] = find_win_team(df)
df

Out[4]:
    team_A     team_B  score_A  score_B win_team   winner
0    Spain        USA        5        4    Spain    Spain
1  Germany  Argentina        3        0  Germany  Germany
2   Brazil     Mexico        2        3   Mexico   Mexico
3   France    Belgium        0        0     Draw     Draw
```

关于 DataFrame.iterrows(), 我们先来看看其运行结果。

```
for row_index, row in df.iterrows():
    print('%s\n%s' % (row_index, row))

0
team_A      Spain
team_B        USA
score_A         5
score_B         4
win_team    Spain
winner      Spain
Name: 0, dtype: object
1
```

```
team_A         Germany
team_B       Argentina
score_A              3
score_B              0
win_team       Germany
winner         Germany
Name: 1, dtype: object
2
team_A       Brazil
team_B       Mexico
score_A           2
score_B           3
win_team     Mexico
winner       Mexico
Name: 2, dtype: object
3
team_A        France
team_B       Belgium
score_A            0
score_B            0
win_team        Draw
winner          Draw
Name: 3, dtype: object
```

DataFrame.iterrows() 的作用是将 dataframe 的每行转换成为一个 Series，可以理解为针对于每一行，做了行列转置。图示如下：

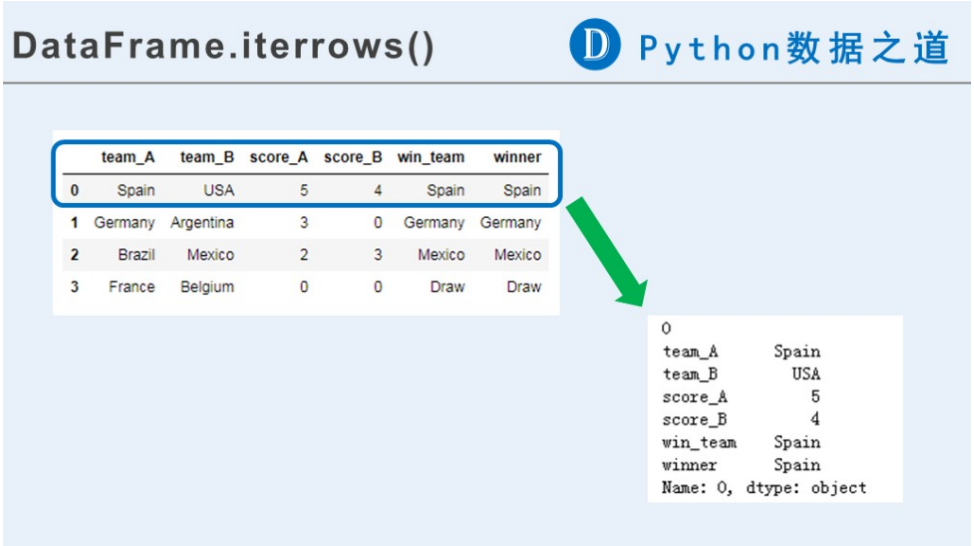


Figure 21.1: iterrows() 的作用

22 Pandas 中 apply 使用技巧!

这里主要介绍使用 apply 函数进行文本拆分，以及如何在 apply 中应用带有两个参数的自定义函数。

22.1 使用 apply 函数拆分文本

Pandas 中 apply 函数，应用广泛，今天要跟大家分享一个使用的技巧，使用 apply 将 dataframe 中内容为 list 的列拆分为多列。

拆分前的数据情况，如下图红色标注所示：

基金代码	基金简称	tmp
000017	财通可持续混合	[2.365, 2020-11-25T00:00:00.000000000]
000020	景顺长城品质投资混合	[3.743, 2020-11-25T00:00:00.000000000]
000021	华夏优势增长混合	[2.689, 2020-11-25T00:00:00.000000000]
000024	大摩双利增强债券A	[1.135, 2020-11-25T00:00:00.000000000]
000025	大摩双利增强债券C	[1.124, 2020-11-25T00:00:00.000000000]

拆分后，如下图所示：

基金代码	基金简称	最新单位净值	净值日期
000017	财通可持续混合	2.365	2020-11-25
000020	景顺长城品质投资混合	3.743	2020-11-25
000021	华夏优势增长混合	2.689	2020-11-25
000024	大摩双利增强债券A	1.135	2020-11-25
000025	大摩双利增强债券C	1.124	2020-11-25

这个案例中，Lemon 使用的数据来自 `akshare`，在开始前，引入相关 package：

```
# -*- coding: utf-8 -*-
"""
@Author: Lemon
@出 品: Python数 据 之 道
@Homepage: liyangbit.com
"""

import numpy as np
import pandas as pd
```

```
import akshare as ak
```

Lemon 使用的几个 Python 库的版本信息如下：

```
print('numpy版本：{}'.format(np.__version__))
print('pandas版本：{}'.format(pd.__version__))
print('akshare版本：{}'.format(ak.__version__))

# numpy版本：1.18.1
# pandas版本：1.0.3
# akshare版本：0.7.53
```

> 如果代码运行出现问题，请先检查下这几个 Python 库的版本是否与上面的一致

先从 akshare 获取需要的数据，分为两步，第一步是获取基金代码的列表，如下：

```
df = ak.fund_em_fund_name().head(20).tail(5)
df = df[['基金代码','基金简称']]
print(df)
```

第二步是获取基金净值数据和净值日期，通过一个自定义函数来获取，自定义函数如下：

```
# 自定义函数只有一个参数的情形
# 获取基金单位净值以及净值日期
def get_mutual_fund(code):
    df = ak.fund_em_open_fund_info(fund=code, indicator="单位净值走势")
    df = df[['净值日期', '单位净值','日增长率']]
    # df.columns = ['净值日期', '单位净值', 'equityReturn', 'unitMoney']
    df['净值日期'] = pd.to_datetime(df['净值日期'])
    df = df.sort_values('净值日期',ascending=False)
    unit_equity = df.head(1)['单位净值'].values[0]
    date_latest = df.head(1)['净值日期'].values[0]
    return [unit_equity, date_latest]
```

对于这个自定义函数，在 pandas 使用 apply 来应用自定义函数，这是使用 apply 的一种常用的方法，如下：

```
# 获取基金最新的单位净值和净值日期
df['tmp'] = df['基金代码'].apply(get_mutual_fund)
print(df)
```

获取的数据截图如下：

基金代码	基金简称	tmp
000017	财通可持续混合	[2.365, 2020-11-25T00:00:00.000000000]
000020	景顺长城品质投资混合	[3.743, 2020-11-25T00:00:00.000000000]
000021	华夏优势增长混合	[2.689, 2020-11-25T00:00:00.000000000]
000024	大摩双利增强债券A	[1.135, 2020-11-25T00:00:00.000000000]
000025	大摩双利增强债券C	[1.124, 2020-11-25T00:00:00.000000000]

22.1.1 文本拆分

上图中的 tmp 列，就是我们这次需要进行处理的对象。

处理方法可以有多种，这里 Lemon 使用 pandas 中的 apply 来处理，相对来说，也是比较便捷的方式。

在 apply 函数中，使用 pd.Series 就可以达到我们的目的。

```
# 将单位净值和净值日期单独成列
df[['最新单位净值','净值日期']] = df['tmp'].apply(pd.Series)
df = df.drop('tmp',axis=1)
print(df)
```

结果如下：

基金代码	基金简称	最新单位	净值	净值日期
000017	财通可持续混合	2.365	2020-11-25	
000020	景顺长城品质投资混合	3.74	2020-11-25	
000021	华夏优势增长混合	2.689	2020-11-25	
000024	大摩双利增强债券A	1.135	2020-11-25	
000025	大摩双利增强债券C	1.124	2020-11-25	

22.2 应用有两个参数的自定义函数

pandas 中的 apply 函数应用自定义函数时，通常情况下，都是没有参数或者一个参数，那么如果有两个参数，是否还可以使用 apply 函数呢?

答案是可以的。

这里我们也来探讨下。

还是以上面的案例为基础雏形，同样的，先从 akshare 获取数据

```
df1 = ak.fund_em_fund_name().head(20).tail(5)
df1 = df1[['基金代码','基金简称']]
```

接下来，自定义一个带有两个参数的函数，如下：

```
# 自定义函数有两个参数的情形
# 获取年度年底基金净值数据
def get_mutual_fund_year(code,year):
    year = str(year)
    df = ak.fund_em_open_fund_info(fund=code, indicator="单位净值走势")
    df = df[['净值日期', '单位净值', '日增长率']]
    # df.columns = ['净值日期', '单位净值', 'equityReturn', 'unitMoney']
    df['净值日期'] = pd.to_datetime(df['净值日期'])
    df = df.sort_values('净值日期',ascending=False)
```

```
    df = df.set_index('净值日期')[year]
    df = df.reset_index()
    unit_equity = df.head(1)['单位净值'].values[0]
    date = df.head(1)['净值日期'].values[0]
    return [unit_equity,date]
```

22.2.1 带有两个参数的自定义函数

然后，使用 apply 来应用上面这个带两个参数的自定义函数，核心要点就是嵌套使用 lambda 函数，固定其中一个参数，具体如下：

```
df1['tmp'] = df1['基金代码'].apply(lambda code: get_mutual_fund_year(code,
    2019))
```

此外，经读者反馈，使用下面的方式，也是可以的：

```
df1['tmp'] = df1['基金代码'].apply(get_mutual_fund_year, year=2019)
```

后续，依旧是文本拆分，实现代码如下：

```
# 将单位净值和净值日期单独成列
df1[['最新单位净值','净值日期']] = df1['tmp'].apply(pd.Series)
df1 = df1.drop('tmp',axis=1)
print(df1)
```

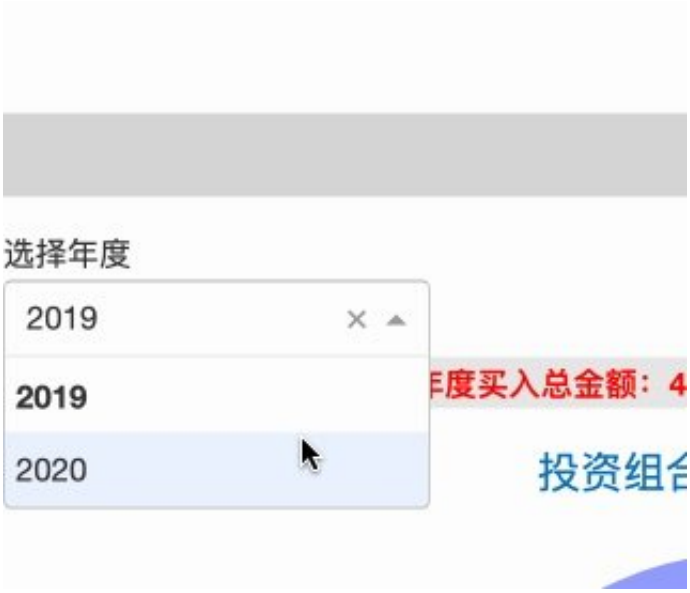
22.2.2 应用场景

有同学可能会问，使用两个参数的自定义函数，有什么用呢？

这里，Lemon 也分享一个应用场景：

根据上面的基础雏形数据，针对具体的年度，建立一个下拉列表，选择不同的年份时，返回不同年份的结果，包括文本数据、表格数据以及图表等。

效果如下：



动态效果请点击下面的链接：

Pandas 的这个知识点，估计 80% 的人都得挂!

涉及到一些个人的数据，就没有完整展示啦 ~~

其他的应用场景，欢迎大家来分享!

23 Pandas 的 Groupby 操作

Pandas 的 `groupby()` 功能很强大，用好了可以方便的解决很多问题，在数据处理以及日常工作中经常能施展拳脚。

今天，我们一起来领略下 `groupby()` 的魅力吧。

首先，引入相关 package：

```
import pandas as pd
import numpy as np
```

23.1 groupby 的基础操作

经常用 `groupby` 对 pandas 中 `dataframe` 的各列进行统计，包括求和、求均值等。

```
In [2]: df = pd.DataFrame({'A': ['a', 'b', 'a', 'c', 'a', 'c', 'b', 'c'],
   ...:                    'B': [2, 8, 1, 4, 3, 2, 5, 9],
   ...:                    'C': [102, 98, 107, 104, 115, 87, 92, 123]})
   ...: df
   ...:

Out[2]:
   A  B    C
0  a  2  102
1  b  8   98
2  a  1  107
3  c  4  104
4  a  3  115
5  c  2   87
6  b  5   92
7  c  9  123
```

按 A 列分组（groupby），获取其他列的均值

```
df.groupby('A').mean()

Out[3]:
     B           C
A
a  2.0  108.000000
b  6.5   95.000000
```

```
c  5.0  104.666667
```

按多列进行分组（groupby）

```
df.groupby(['A','B']).mean()

Out[4]:
       C
A B
a 1  107
  2  102
  3  115
b 5   92
  8   98
c 2   87
  4  104
  9  123
```

23.2 分组后选择列进行运算

分组后，可以选取单列数据，或者多个列组成的列表（list）进行运算

```
In [5]: df = pd.DataFrame([[1, 1, 2], [1, 2, 3], [2, 3, 4]], columns=["A", "B",
     "C"])
   ...: df
   ...:

Out[5]:
   A  B  C
0  1  1  2
1  1  2  3
2  2  3  4

In [6]: g = df.groupby("A")

In [7]: g['B'].mean() # 仅选择B列

Out[7]:
A
1    1.5
2    3.0
Name: B, dtype: float64

In [8]: g[['B', 'C']].mean() # 选择B、C列

Out[8]:
     B    C
A
1  1.5  2.5
```

```
27  2  3.0  4.0
```

可以针对不同的列选用不同的聚合方法

```
In [9]: g.agg({'B':'mean', 'C':'sum'})

Out[9]:
     B  C
A
1  1.5  5
2  3.0  4
```

23.3 聚合方法

聚合方法有 size() 和 count() 。

size 跟 count 的区别：size 计数时包含 NaN 值，而 count 不包含 NaN 值

```
In [10]: df = pd.DataFrame({"Name":["Alice", "Bob", "Mallory", "Mallory", "Bob"
     , "Mallory"],
    ...:                    "City":["Seattle", "Seattle", "Portland", "Seattle",
         "Seattle", "Portland"],
    ...:                    "Val":[4,3,3,np.nan,np.nan,4]})
    ...:
    ...: df
    ...:
Out[10]:
       City     Name  Val
0   Seattle    Alice  4.0
1   Seattle      Bob  3.0
2  Portland  Mallory  3.0
3   Seattle  Mallory  NaN
4   Seattle      Bob  NaN
5  Portland  Mallory  4.0
```

23.3.1 count()

```
In [11]: df.groupby(["Name", "City"], as_index=False)['Val'].count()

Out[11]:
      Name      City  Val
0    Alice   Seattle    1
1      Bob   Seattle    1
2  Mallory  Portland    2
3  Mallory   Seattle    0
```

23.3.2 size()

```
In [12]: df.groupby(["Name", "City"])['Val'].size().reset_index(name='Size')

Out[12]:
      Name      City  Size
0    Alice   Seattle     1
1      Bob   Seattle     2
2  Mallory  Portland     2
3  Mallory   Seattle     1
```

23.4 分组运算方法 agg()

针对某列使用 agg() 时进行不同的统计运算

```
In [13]: df = pd.DataFrame({'A': list('XYZXYZXYZX'), 'B': [1, 2, 1, 3, 1, 2, 3,
     3, 1, 2],
    ...:                            'C': [12, 14, 11, 12, 13, 14, 16, 12, 10,
         19]})
    ...: df
    ...:
Out[13]:
   A  B   C
0  X  1  12
1  Y  2  14
2  Z  1  11
3  X  3  12
4  Y  1  13
5  Z  2  14
6  X  3  16
7  Y  3  12
8  Z  1  10
9  X  2  19

In [14]: df.groupby('A')['B'].agg({'mean':np.mean, 'standard deviation': np.std
    })

Out[14]:
        mean  standard deviation
A
X  2.250000            0.957427
Y  2.000000            1.000000
Z  1.333333            0.577350
```

针对不同的列应用多种不同的统计方法

```
In [15]: df.groupby('A').agg({'B':[np.mean, 'sum'], 'C':['count',np.std]})

Out[15]:
```

```
          B         C
       mean sum count       std
A
X  2.250000   9     4  3.403430
Y  2.000000   6     3  1.000000
Z  1.333333   4     3  2.081666
```

23.5 分组运算方法 apply()

```
In [16]: df = pd.DataFrame({'A': list('XYZXYZXYZX'), 'B': [1, 2, 1, 3, 1, 2, 3,
     3, 1, 2],
    ...:                             'C': [12, 14, 11, 12, 13, 14, 16, 12, 10,
        19]})
    ...: df
    ...:
Out[16]:
   A  B   C
0  X  1  12
1  Y  2  14
2  Z  1  11
3  X  3  12
4  Y  1  13
5  Z  2  14
6  X  3  16
7  Y  3  12
8  Z  1  10
9  X  2  19

In [17]: df.groupby('A').apply(np.mean)
    ...: # 跟下面的方法的运行结果是一致的
    ...: # df.groupby('A').mean()
Out[17]:
          B          C
A
X  2.250000  14.750000
Y  2.000000  13.000000
Z  1.333333  11.666667
```

`apply()` 方法可以应用 lambda 函数，举例如下:

```
In [18]: df.groupby('A').apply(lambda x: x['C']-x['B'])
Out[18]:
A
X  0    11
   3     9
   6    13
   9    17
Y  1    12
   4    12
```

```
10     7     9
11 Z   2    10
12     5    12
13     8     9
14 dtype: int64
15
16 In [19]: df.groupby('A').apply(lambda x: (x['C']-x['B']).mean())
17 Out[19]:
18 A
19 X    12.500000
20 Y    11.000000
21 Z    10.333333
22 dtype: float64
```

23.6 分组运算方法 transform()

前面进行聚合运算的时候，得到的结果是一个以分组名为 index 的结果对象。如果我们想使用原数组的 index 的话，就需要进行 merge 转换。transform(func, *args, **kwargs) 方法简化了这个过程，它会把 func 参数应用到所有分组，然后把结果放置到原数组的 index 上（如果结果是一个标量，就进行广播）：

```
1  In [20]: df = pd.DataFrame({'group1' :  ['A', 'A', 'A', 'A',
2      ...:                                 'B', 'B', 'B', 'B'],
3      ...:                    'group2' :  ['C', 'C', 'C', 'D',
4      ...:                                 'E', 'E', 'F', 'F'],
5      ...:                    'B'      :  ['one', np.NaN, np.NaN, np.NaN,
6      ...:                                 np.NaN, 'two', np.NaN, np.NaN],
7      ...:                    'C'      :  [np.NaN, 1, np.NaN, np.NaN,
8      ...:                                np.NaN, np.NaN, np.NaN, 4]})
9      ...: df
10     ...:
11 Out[20]:
12      B    C group1 group2
13 0  one  NaN      A      C
14 1  NaN  1.0      A      C
15 2  NaN  NaN      A      C
16 3  NaN  NaN      A      D
17 4  NaN  NaN      B      E
18 5  two  NaN      B      E
19 6  NaN  NaN      B      F
20 7  NaN  4.0      B      F
21
22 In [21]: df.groupby(['group1', 'group2'])['B'].transform('count')
23 Out[21]:
24 0    1
25 1    1
26 2    1
27 3    0
28 4    1
```

```
5    1
6    0
7    0
Name: B, dtype: int64

In [22]: df['count_B']=df.groupby(['group1', 'group2'])['B'].transform('count')
    ...: df
    ...:
Out[22]:
     B    C group1 group2  count_B
0  one  NaN      A      C        1
1  NaN  1.0      A      C        1
2  NaN  NaN      A      C        1
3  NaN  NaN      A      D        0
4  NaN  NaN      B      E        1
5  two  NaN      B      E        1
6  NaN  NaN      B      F        0
7  NaN  4.0      B      F        0
```

上面运算的结果分析：{'group1':'A', 'group2':'C'} 的组合共出现 3 次，即 index 为 0,1,2。对应 “B” 列的值分别是 “one”,“NaN”,“NaN”，由于 count() 计数时不包括 NaN 值，因此 {'group1':'A', 'group2':'C'} 的 count 计数值为 1。

transform() 方法会将该计数值在 dataframe 中所有涉及的 rows 都显示出来（我理解应该就进行广播）

23.7 将某列数据按数据值分成不同范围段进行分组（groupby）运算

```
In [23]: np.random.seed(0)
    ...: df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),
    ...:                    'Sex': np.random.choice(['Male', 'Female'], 100),
    ...:                    'number_of_foo': np.random.randint(1, 20, 100)})
    ...: df.head()
    ...:
Out[23]:
   Age     Sex  number_of_foo
0   64  Female             14
1   67  Female             14
2   20  Female             12
3   23    Male             17
4   23  Female             15
```

这里将 “Age” 列分成三类，有两种方法可以实现：

(a) bins=4

(b) bins=[19, 40, 65, np.inf]

```
In [24]: pd.cut(df['Age'], bins=4)
Out[24]:
```

```
...

In [25]: pd.cut(df['Age'], bins=[19,40,65,np.inf])
```

分组结果范围结果如下：

```
97     (40, 65]
98    (65, inf]
99    (65, inf]
Name: Age, dtype: category
Categories (3, object): [(19, 40] < (40, 65] < (65, inf]]
```

Python数据之道

```
In [26]: age_groups = pd.cut(df['Age'], bins=[19,40,65,np.inf])
    ...: df.groupby(age_groups).mean()
```

运行结果如下：

	Age	number_of_foo
Age		
(19, 40]	29.840000	9.880000
(40, 65]	52.833333	9.452381
(65, inf]	67.375000	9.250000

Python数据之道

按 'Age' 分组范围和性别（sex）进行制作交叉表

```
In [27]: pd.crosstab(age_groups, df['Sex'])
```

运行结果如下：

Sex	Female	Male
Age		
(19, 40]	22	28
(40, 65]	18	24
(65, inf]	3	5

Python数据之道

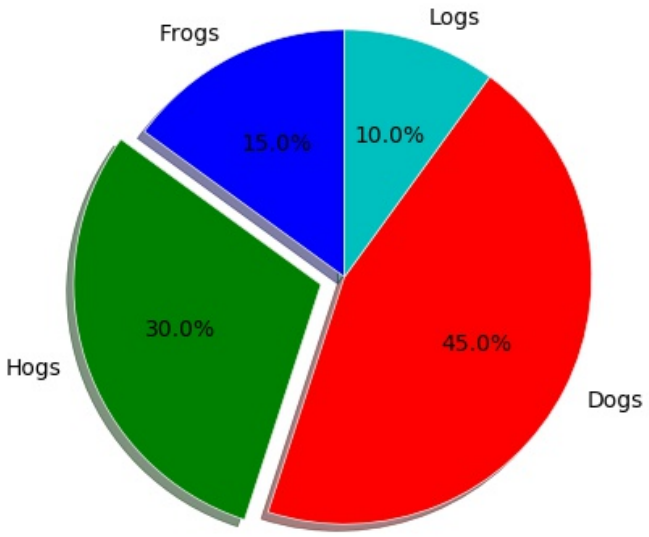
24 Matplotlib 饼图简介

Matplotlib，官方提供的饼图 Demo，功能比较比较简单，在实际应用过程中，往往会有许多个性化的绘制需求，在这里跟大家一起了解下饼图（pie chart）的一些特色的功能的实现。首先引入相关 Python 库。

```
from matplotlib import font_manager as fm
import matplotlib as mpl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
plt.style.use('ggplot')
```

24.1 官方 Demo

```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.savefig('Demo_official.jpg')
plt.show()
```

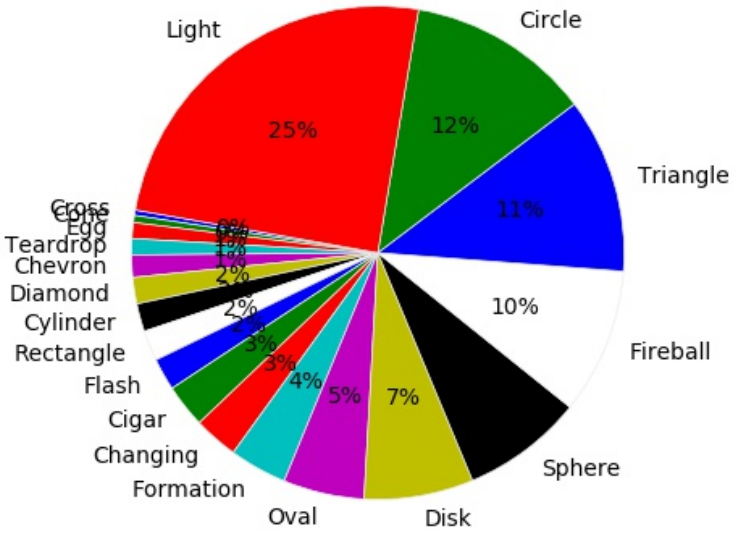


24.2 将实际数据应用于官方 Demo

```
# 原始数据
shapes = ['Cross', 'Cone', 'Egg', 'Teardrop', 'Chevron', 'Diamond', 'Cylinder',
       'Rectangle', 'Flash', 'Cigar', 'Changing', 'Formation', 'Oval', 'Disk',
       'Sphere', 'Fireball', 'Triangle', 'Circle', 'Light']
values = [  287,   383,   842,   866,  1187,  1405,  1495,  1620,  1717,
        2313,  2378,  3070,  4332,  5841,  6482,  7785,  9358,  9818, 20254]

s = pd.Series(values, index=shapes)
s
```

```
from matplotlib import font_manager as fm
import matplotlib as mpl

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = s.index
sizes = s.values
explode = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st
    slice

fig1, ax1 = plt.subplots()
patches, texts, autotexts = ax1.pie(sizes, explode=explode, labels=labels,
    autopct='%1.0f%%',
        shadow=False, startangle=170)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

```

```
14 plt.savefig('Demo_project.jpg')
15 plt.show()
```



上图的一些问题：

1. 颜色比较生硬
2. 部分文字拥挤在一起，绘图显示不齐整

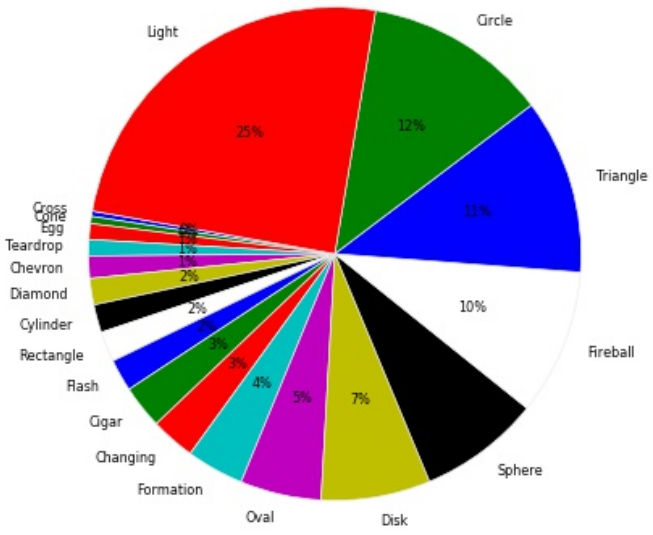
24.3 一些改善措施

- 重新设置字体大小
- 设置自选颜色
- 设置图例
- 将某些类别突出显示

24.3.1 重新设置字体大小

```
1 from matplotlib import font_manager as fm
2 import matplotlib as mpl
3
4 labels = s.index
5 sizes = s.values
6 explode = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st
      slice
7
8 fig1, ax1 = plt.subplots()
```

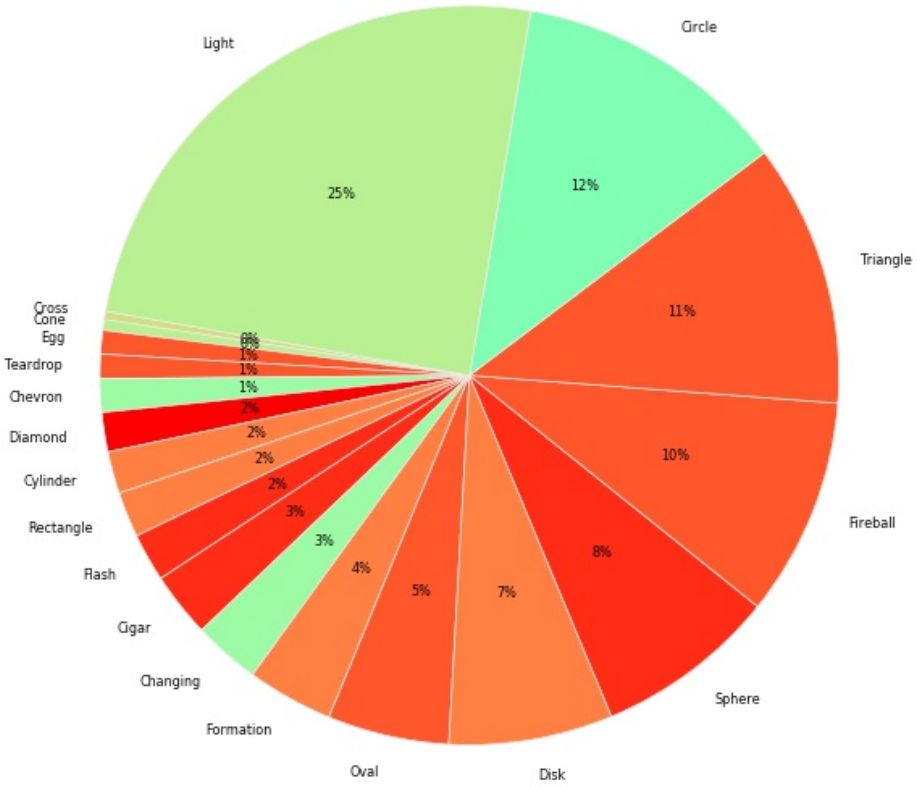
```
9
10 patches, texts, autotexts = ax1.pie(sizes, explode=explode, labels=labels,
       autopct='%1.0f%%',
11         shadow=False, startangle=170)
12 ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
13
14 # 重新设置字体大小
15 proptease = fm.FontProperties()
16 proptease.set_size('xx-small')
17 # font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-
       large’ , ‘xx-large’ or number, e.g. '12'
18 plt.setp(autotexts, fontproperties=proptease)
19 plt.setp(texts, fontproperties=proptease)
20
21 plt.savefig('Demo_project_set_font.jpg')
22 plt.show()
```



24.3.2 设置显示颜色，Method 1:

```
1 from matplotlib import font_manager as fm
2 import matplotlib as mpl
3
4 # Pie chart, where the slices will be ordered and plotted counter-clockwise:
5 labels = s.index
6 sizes = s.values
7 explode = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st
      slice
8
```

```
fig1, ax1 = plt.subplots(figsize=(6,6)) # 设置绘图区域大小

a = np.random.rand(1,19)
color_vals = list(a[0])
my_norm = mpl.colors.Normalize(-1, 1) # 将颜色数据的范围设置为 [0, 1]
my_cmap = mpl.cm.get_cmap('rainbow', len(color_vals)) # 可选择合适的colormap，如：'rainbow'

patches, texts, autotexts = ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.0f%%',
        shadow=False, startangle=170, colors=my_cmap(my_norm(color_vals)))

ax1.axis('equal')

# 重新设置字体大小
proptease = fm.FontProperties()
proptease.set_size('xx-small')
# font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-large’ , ‘xx-large’ or number, e.g. '12'
plt.setp(autotexts, fontproperties=proptease)
plt.setp(texts, fontproperties=proptease)

plt.savefig('Demo_project_set_color_1.jpg')
plt.show()
```

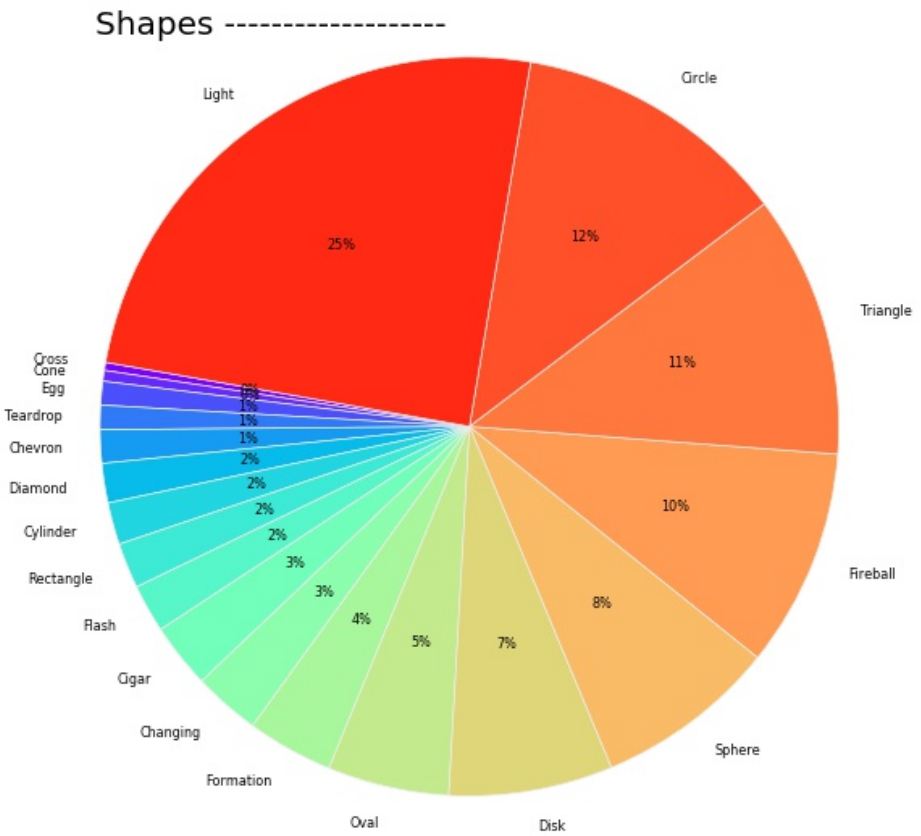


上面这种方法设置颜色时，但类别比较多时，部分颜色的填充会重复。有时候，我们可能想设置成连续的颜色，可以有另外一种方法来实现。

24.3.3 设置显示颜色，Method 2：

```
from matplotlib import font_manager as fm
from  matplotlib import cm

labels = s.index
sizes = s.values
# explode = (0.2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st
    slice

fig, ax = plt.subplots(figsize=(6,6)) # 设置绘图区域大小


colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired,
   autumn, rainbow, gray,spring,Darks
```

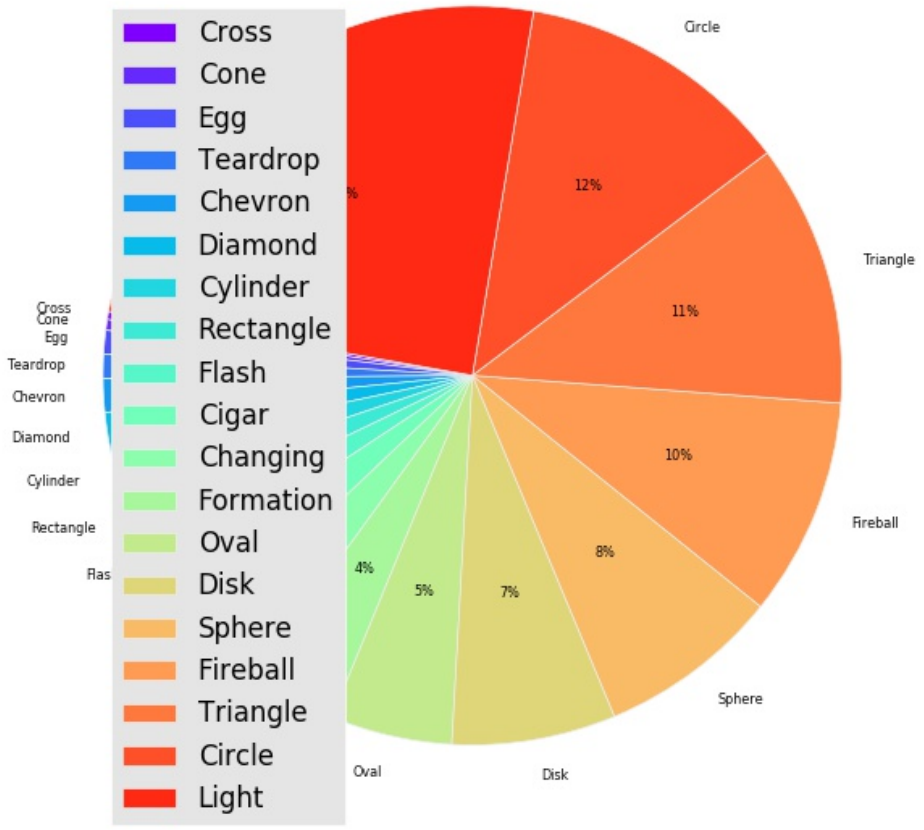
```
12 patches, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.0f%%',
13         shadow=False, startangle=170, colors=colors)
14
15 ax.axis('equal')
16 ax.set_title('Shapes ------------------', loc='left')
17
18 # 重新设置字体大小
19 proptease = fm.FontProperties()
20 proptease.set_size('xx-small')
21 # font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-
       large’ , ‘xx-large’ or number, e.g. '12'
22 plt.setp(autotexts, fontproperties=proptease)
23 plt.setp(texts, fontproperties=proptease)
24
25 plt.savefig('Demo_project_set_color_2.jpg')
26 plt.show()
```



从上图可以看出，颜色显示是连续的，实现了我们想要的效果

24.3.4 设置图例（legend）

```
from matplotlib import font_manager as fm
from  matplotlib import cm

labels = s.index
sizes = s.values
# explode = (0.2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st slice

fig, ax = plt.subplots(figsize=(6,6)) # 设置绘图区域大小


colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks
patches, texts, autotexts = ax.pie(sizes, labels=labels, autopct='%1.0f%%',
        shadow=False, startangle=170, colors=colors)

ax.axis('equal')

# 重新设置字体大小
proptease = fm.FontProperties()
proptease.set_size('xx-small')
# font size include:  'xx-small' ,x-small' ,'small' ,'medium' , 'large' , 'x-large' , 'xx-large'  or number, e.g. '12'
plt.setp(autotexts, fontproperties=proptease)
plt.setp(texts, fontproperties=proptease)


ax.legend(labels, loc=2)

plt.savefig('Demo_project_set_legend_error.jpg')
plt.show()
```

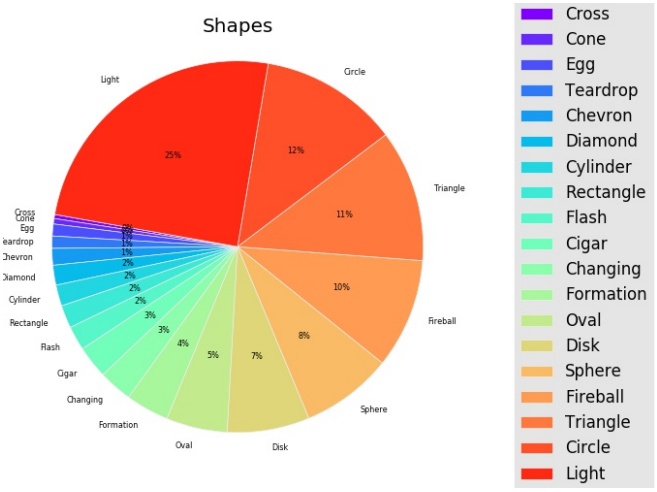


从上面可看出，当类别较多时，图例（legend）的位置摆放显示有重叠，显示有些问题，需要进行调整。

24.3.5 重新设置图例（legend）

```
from matplotlib import font_manager as fm
from  matplotlib import cm

labels = s.index
sizes = s.values
# explode = (0.2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st slice

fig, axes = plt.subplots(figsize=(10,5),ncols=2) # 设置绘图区域大小
ax1, ax2 = axes.ravel()

colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks
patches, texts, autotexts = ax1.pie(sizes, labels=labels, autopct='%1.0f%%',
        shadow=False, startangle=170, colors=colors)
```

```
14
15 ax1.axis('equal')
16
17 # 重新设置字体大小
18 proptease = fm.FontProperties()
19 proptease.set_size('xx-small')
20 # font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-
       large’ , ‘xx-large’ or number, e.g. '12'
21 plt.setp(autotexts, fontproperties=proptease)
22 plt.setp(texts, fontproperties=proptease)
23
24 ax1.set_title('Shapes', loc='center')
25
26 # ax2 只显示图例（legend）
27 ax2.axis('off')
28 ax2.legend(patches, labels, loc='center left')
29
30 plt.tight_layout()
31 plt.savefig('Demo_project_set_legend_good.jpg')
32 plt.show()
```

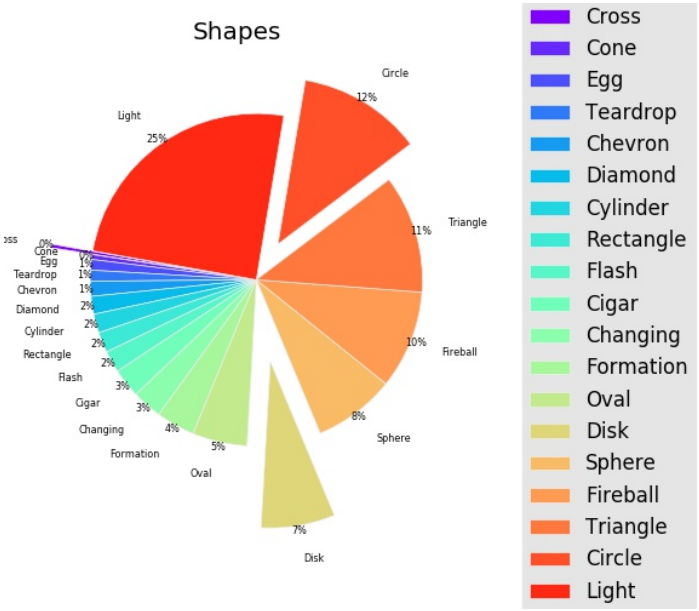


24.3.6 将某些类别突出显示

- 将某些类别突出显示
- 控制 label 的显示位置
- 控制百分比的显示位置
- 控制突出位置的大小

```
1 from matplotlib import font_manager as fm
2 from  matplotlib import cm
3
4 labels = s.index
5 sizes = s.values
```

```
explode = (0.1,0,0,0,0,0,0,0,0,0,0,0,0,0.2,0,0,0,0.1,0)  # "explode" ,  show the selected slice

fig, axes = plt.subplots(figsize=(8,5),ncols=2) # 设置绘图区域大小
ax1, ax2 = axes.ravel()

colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks
patches, texts, autotexts = ax1.pie(sizes, labels=labels, autopct='%1.0f%%', explode=explode,
        shadow=False, startangle=170, colors=colors, labeldistance=1.2, pctdistance=1.03, radius=0.4)
# labeldistance: 控制labels显示的位置
# pctdistance: 控制百分比显示的位置
# radius: 控制切片突出的距离

ax1.axis('equal')

# 重新设置字体大小
proptease = fm.FontProperties()
proptease.set_size('xx-small')
# font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-large’ , ‘xx-large’  or number, e.g. '12'
plt.setp(autotexts, fontproperties=proptease)
plt.setp(texts, fontproperties=proptease)

ax1.set_title('Shapes', loc='center')

# ax2 只显示图例（legend）
ax2.axis('off')
ax2.legend(patches, labels, loc='center left')

plt.tight_layout()
# plt.savefig("pie_shape_ufo.png", bbox_inches='tight')
plt.savefig('Demo_project_final.jpg')
plt.show()
```



> 本文涉及的代码量比较多，如需要查看源代码，请在微信公众号“Python 数据之道”（ID：PyDataLab）后台回复关键字“code”。

25 Matplotlib 中等高线图（contour）的绘制

在此前的文章里，我们介绍了Numpy 中 meshgrid 函数的应用，并提到了等高线图的使用。下面，我们来介绍一下 matplotlib 中等高线图的绘制。先上图来看一下等高线图的绘制效果，是不是很炫啊。

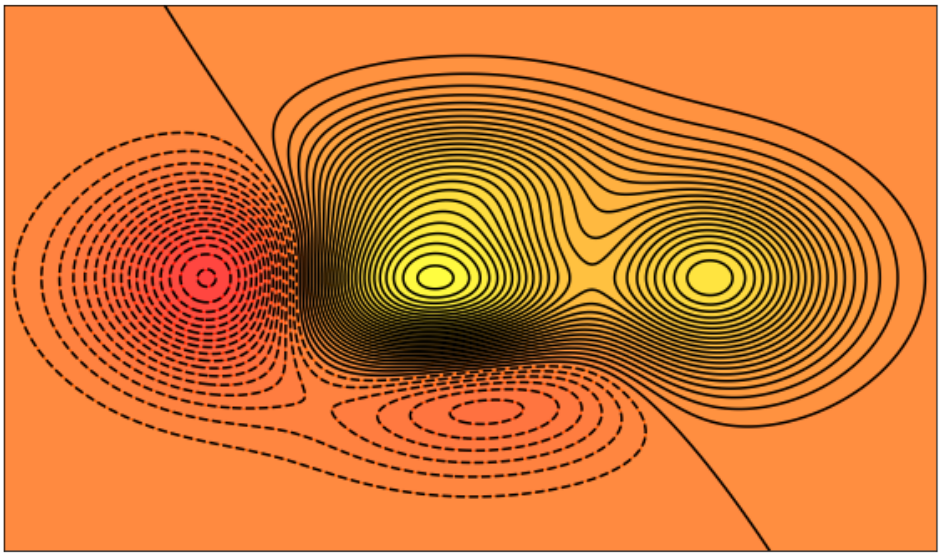


Figure 25.1: 等高线图的效果图

下面我们来分步骤介绍 matplotlib 中等高线图（contour）的绘制。

数据初始化

```
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

%matplotlib inline

# 定义等高线高度函数
def f(x, y):
    return (1 - x / 2 + x ** 5 + y ** 3) * np.exp(-x ** 2 - y ** 2)

```

```
# 数据数目
n = 256
# 定义x, y
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)

# 生成网格数据
X, Y = np.meshgrid(x, y)
```

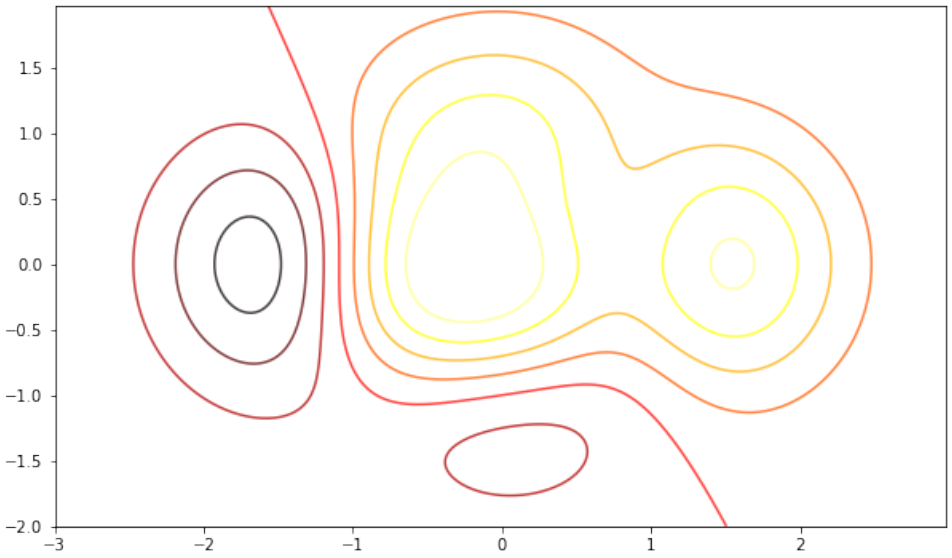
注：数据初始化过程中用到了 Numpy 的 meshgrid 函数，有兴趣的同学可以了解下这个函数的使用，链接如下：

- Numpy 中 Meshgrid 函数介绍及 2 种应用场景

25.1 仅绘制等高线

```
# 设置图像大小尺寸
plt.figure(figsize=(10,6))

# 填充等高线的颜色，8是等高线分为几部分
plt.contour(X, Y,f(X, Y), 8,alpha = 0.75, cmap = plt.cm.hot)
```

结果如下：

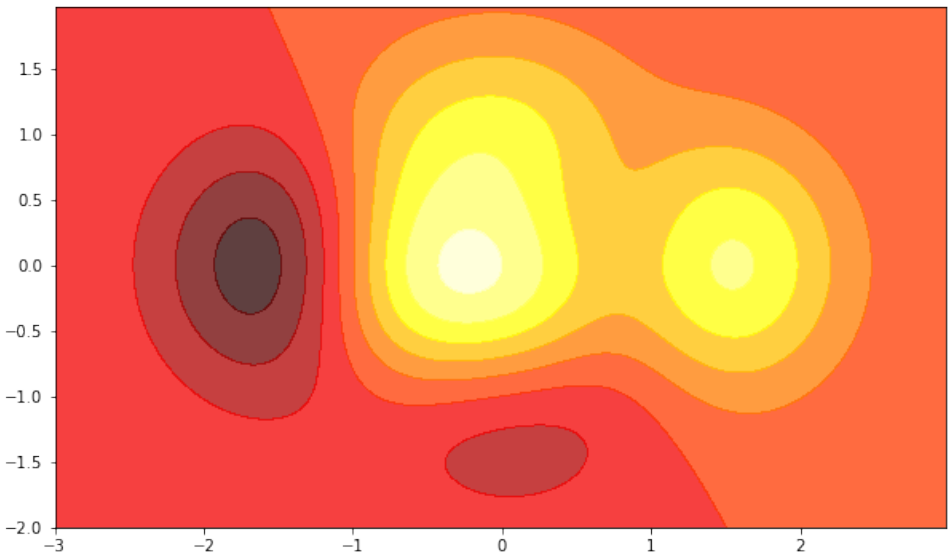


25.2 仅填充等高区域颜色：

```
# 设置图像大小尺寸
plt.figure(figsize=(10,6))

```

```
# 填充等高区域的颜色，8是等高线分为几部分
plt.contourf(X, Y,f(X, Y), 8, alpha = 0.75, cmap = plt.cm.hot)
```

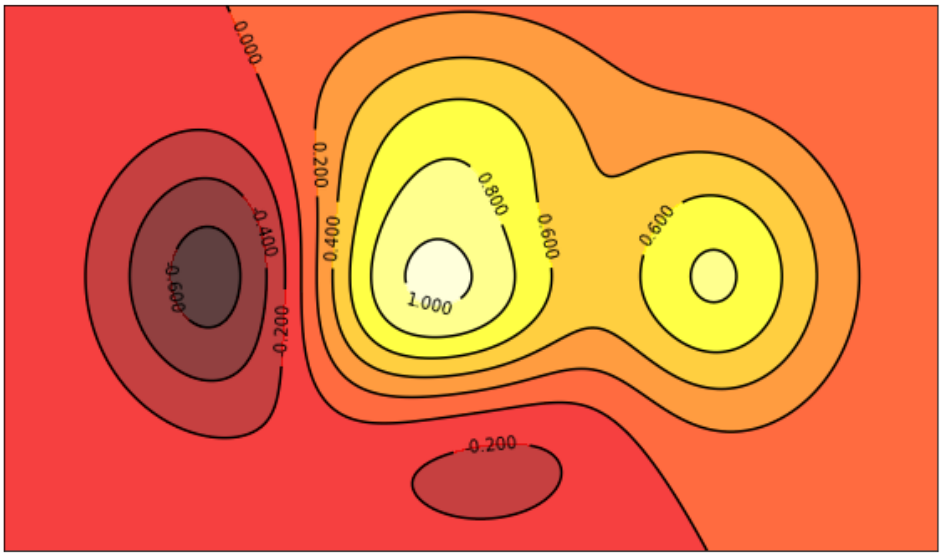
结果如下：



25.3 绘制完整的等高线图

```
# 设置图像大小尺寸
plt.figure(figsize=(10,6))

# 填充等高线的颜色，8是等高线分为几部分
plt.contourf(X, Y, f(X, Y), 8, alpha = 0.75, cmap = plt.cm.hot)

C = plt.contour(X,Y, f(X,Y), 8, colors='black', linewidth=8)

# 绘制等高线数据
plt.clabel(C, inline = True, fontsize = 10)

# 去除坐标轴
plt.xticks(())
plt.yticks(())

plt.show()
```

结果如下：



是不是很容易实现啊，赶快动手来试试吧。

26 Matplotlib 可视化最有价值的 50 个图表

原文标题：深度好文 | Matplotlib 可视化最有价值的 50 个图表（附完整 Python 源代码）

本文总结了 Matplotlib 以及 Seaborn 用的最多的 50 个图形，数量掌握这些图形的绘制，对于数据分析的可视化有莫大的作用，强烈推荐大家阅读后续内容。

> Tips:
>
> (1) 本文原文部分代码有不准确的地方，已进行修改；
>
> (2) 所有正确的源代码，我已整合到 jupyter notebook 文件中，可以在公众号『Python 数据之道』后台回复“code”，可获得本文源代码；
>
> (3) 运行本文代码，除了安装 matplotlib 和 seaborn 可视化库外，还需要安装其他的一些辅助可视化库，已在代码部分作标注，具体内容请查看下面文章内容

在数据分析和可视化中最有用的 50 个 Matplotlib 图表。这些图表列表允许您使用 python 的 matplotlib 和 seaborn 库选择要显示的可视化对象。

26.1 介绍

这些图表根据可视化目标的 7 个不同情景进行分组。例如，如果要想象两个变量之间的关系，请查看“相关”部分下的图表。或者，如果您想要显示值如何随时间变化，请查看“更改”部分，依此类推。

有效图表的重要特征：

- 在不歪曲事实的情况下传达正确和必要的信息。
- 设计简单，您不必太费力就能理解它。
- 从审美角度支持信息而不是掩盖信息。
- 信息没有超负荷。

26.2 准备工作

在代码运行前先引入下面的设置内容。当然，单独的图表，可以重新设置显示要素。

```
1
2 # !pip install brewer2mpl
3 import numpy as np
4 import pandas as pd
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import warnings; warnings.filterwarnings(action='once')

large = 22; med = 16; small = 12
params = {'axes.titlesize': large,
          'legend.fontsize': med,
          'figure.figsize': (16, 10),
          'axes.labelsize': med,
          'axes.titlesize': med,
          'xtick.labelsize': med,
          'ytick.labelsize': med,
          'figure.titlesize': large}
plt.rcParams.update(params)
plt.style.use('seaborn-whitegrid')
sns.set_style("white")
%matplotlib inline

# Version
print(mpl.__version__)  #> 3.0.0
print(sns.__version__)  #> 0.9.0

out:
3.0.2
0.9.0
```

26.3 关联（Correlation）

关联图表用于可视化 2 个或更多变量之间的关系。也就是说，一个变量如何相对于另一个变化。

26.3.1 散点图（Scatter plot）

散点图是用于研究两个变量之间关系的经典的和基本的图表。如果数据中有多个组，则可能需要以不同颜色可视化每个组。在 matplotlib 中，您可以使用 `plt.scatterplot()` 方便地执行此操作。

```
# Import dataset
midwest = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/midwest_filter.csv")

# Prepare Data
# Create as many colors as there are unique midwest['category']
categories = np.unique(midwest['category'])
colors = [plt.cm.tab10(i/float(len(categories)-1)) for i in range(len(categories))]

# Draw Plot for Each Category
```

```
plt.figure(figsize=(16, 10), dpi= 80, facecolor='w', edgecolor='k')

for i, category in enumerate(categories):
    plt.scatter('area', 'poptotal',
                data=midwest.loc[midwest.category==category, :],
                s=20, cmap=colors[i], label=str(category))
    # "c=" 修改为 "cmap=", Python数据之道 备注

# Decorations
plt.gca().set(xlim=(0.0, 0.1), ylim=(0, 90000),
              xlabel='Area', ylabel='Population')

plt.xticks(fontsize=12); plt.yticks(fontsize=12)
plt.title("Scatterplot of Midwest Area vs Population", fontsize=22)
plt.legend(fontsize=12)
plt.show()
```

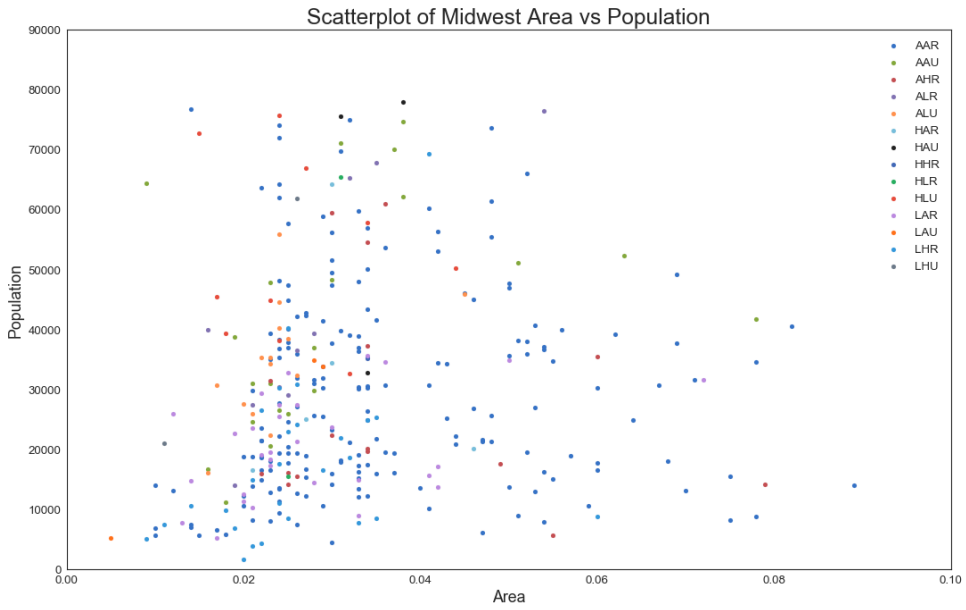


Figure 26.1: 散点图

26.3.2 带边界的气泡图（Bubble plot with Encircling）

有时，您希望在边界内显示一组点以强调其重要性。在这个例子中，你从数据框中获取记录，并用下面代码中描述的 `encircle()` 来使边界显示出来。

```
from matplotlib import patches
from scipy.spatial import ConvexHull
import warnings; warnings.simplefilter('ignore')
sns.set_style("white")

```

```
# Step 1: Prepare Data
midwest = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/midwest_filter.csv")

# As many colors as there are unique midwest['category']
categories = np.unique(midwest['category'])
colors = [plt.cm.tab10(i/float(len(categories)-1)) for i in range(len(categories))]

# Step 2: Draw Scatterplot with unique color for each category
fig = plt.figure(figsize=(16, 10), dpi= 80, facecolor='w', edgecolor='k')

for i, category in enumerate(categories):
    plt.scatter('area', 'poptotal', data=midwest.loc[midwest.category==category, :],
                s='dot_size', cmap=colors[i], label=str(category), edgecolors='black', linewidths=.5)
    # "c=" 修改为 "cmap=", Python数据之道 备注

# Step 3: Encircling
# https://stackoverflow.com/questions/44575681/how-do-i-encircle-different-data-sets-in-scatter-plot
def encircle(x,y, ax=None, **kw):
    if not ax: ax=plt.gca()
    p = np.c_[x,y]
    hull = ConvexHull(p)
    poly = plt.Polygon(p[hull.vertices,:], **kw)
    ax.add_patch(poly)

# Select data to be encircled
midwest_encircle_data = midwest.loc[midwest.state=='IN', :]

# Draw polygon surrounding vertices
encircle(midwest_encircle_data.area, midwest_encircle_data.poptotal, ec="k", fc="gold", alpha=0.1)
encircle(midwest_encircle_data.area, midwest_encircle_data.poptotal, ec="firebrick", fc="none", linewidth=1.5)

# Step 4: Decorations
plt.gca().set(xlim=(0.0, 0.1), ylim=(0, 90000),
              xlabel='Area', ylabel='Population')

plt.xticks(fontsize=12); plt.yticks(fontsize=12)
plt.title("Bubble Plot with Encircling", fontsize=22)
plt.legend(fontsize=12)
plt.show()
```

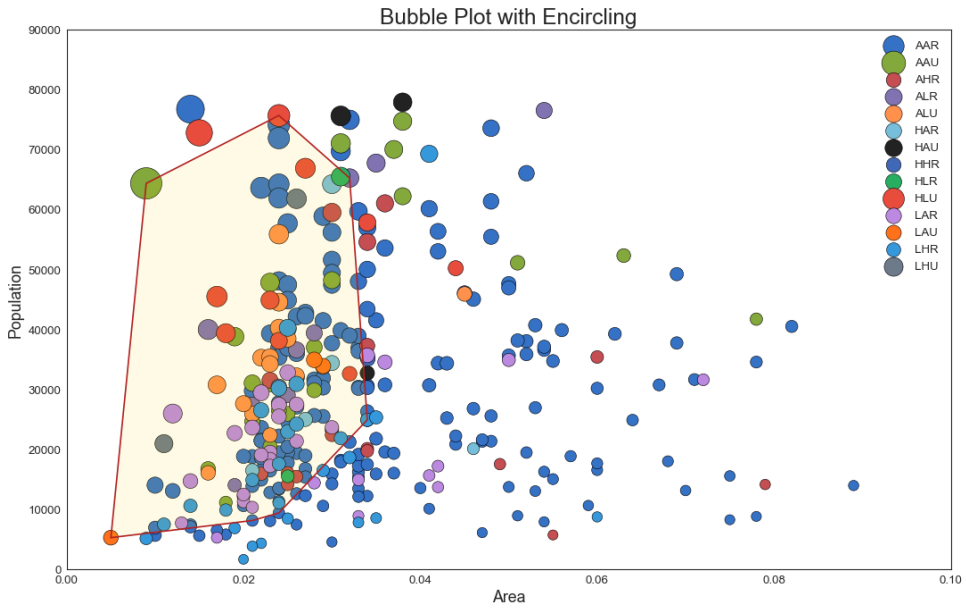


Figure 26.2: 气泡图

26.3.3 带线性回归最佳拟合线的散点图（Scatter plot with linear regression line of best fit）

如果你想了解两个变量如何相互改变，那么最佳拟合线就是常用的方法。下图显示了数据中各组之间最佳拟合线的差异。要禁用分组并仅为整个数据集绘制一条最佳拟合线，请从下面的`sns.lmplot()`调用中删除`hue='cyl'`参数。

```
# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    mpg_ggplot2.csv")
df_select = df.loc[df.cyl.isin([4,8]), :]

# Plot
sns.set_style("white")
gridobj = sns.lmplot(x="displ", y="hwy", hue="cyl", data=df_select,
                     height=7, aspect=1.6, robust=True, palette='tab10',
                     scatter_kws=dict(s=60, linewidths=.7, edgecolors='black'))

# Decorations
gridobj.set(xlim=(0.5, 7.5), ylim=(0, 50))
plt.title("Scatterplot with line of best fit grouped by number of cylinders",
    fontsize=20)
plt.show()
```

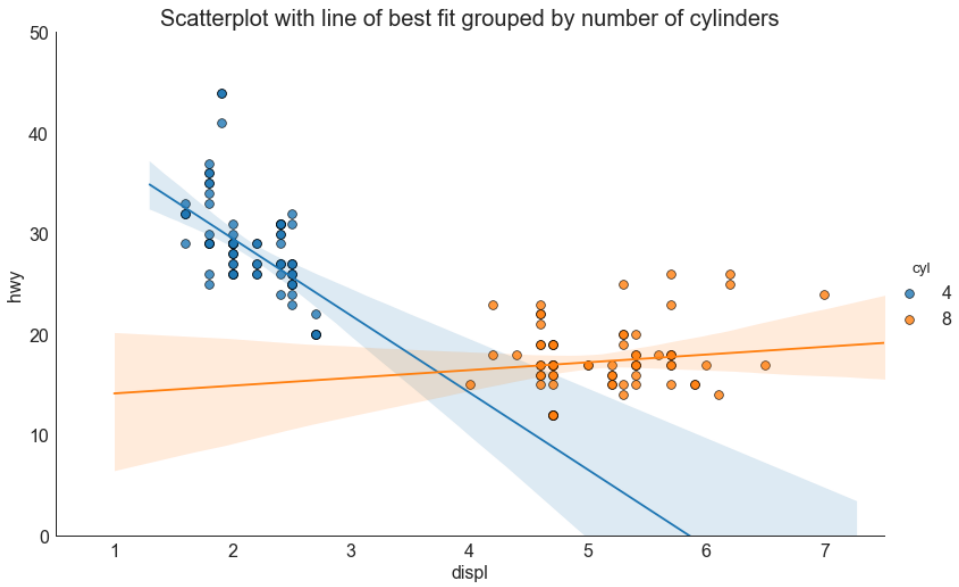


Figure 26.3: 带线性回归最佳拟合线的散点图

针对每列绘制线性回归线

或者，可以在其每列中显示每个组的最佳拟合线。可以通过在 `sns.lmplot()` 中设置 `col=groupingcolumn` 参数来实现，如下：

```
# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    mpg_ggplot2.csv")
df_select = df.loc[df.cyl.isin([4,8]), :]

# Each line in its own column
sns.set_style("white")
gridobj = sns.lmplot(x="displ", y="hwy",
                     data=df_select,
                     height=7,
                     robust=True,
                     palette='Set1',
                     col="cyl",
                     scatter_kws=dict(s=60, linewidths=.7, edgecolors='black'))

# Decorations
gridobj.set(xlim=(0.5, 7.5), ylim=(0, 50))
plt.show()
```

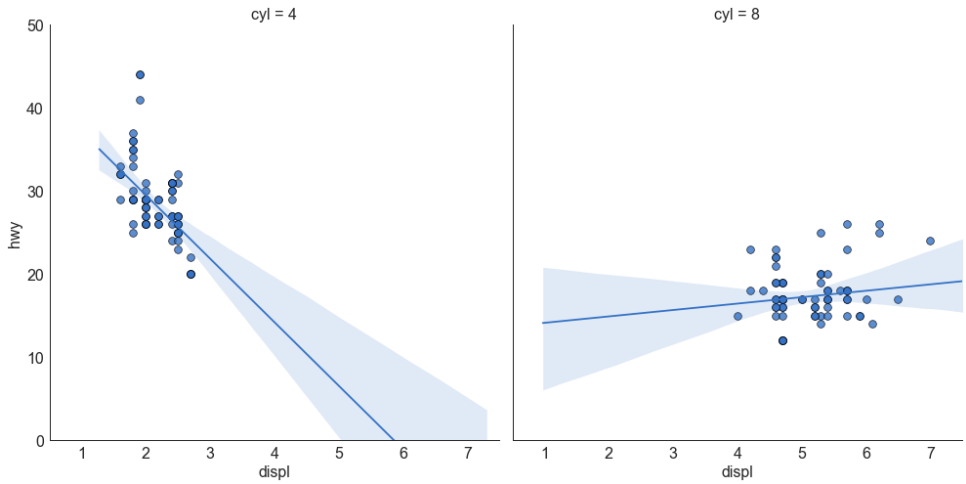


Figure 26.4: 带线性回归最佳拟合线的散点图

26.3.4 抖动图（Jittering with stripplot）

通常，多个数据点具有完全相同的 X 和 Y 值。结果，多个点绘制会重叠并隐藏。为避免这种情况，请将数据点稍微抖动，以便您可以直观地看到它们。使用 seaborn 的 `stripplot()` 很方便实现这个功能。

```
# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    mpg_ggplot2.csv")

# Draw Stripplot
fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
sns.stripplot(df.cty, df.hwy, jitter=0.25, size=8, ax=ax, linewidth=.5)

# Decorations
plt.title('Use jittered plots to avoid overlapping of points', fontsize=22)
plt.show()
```

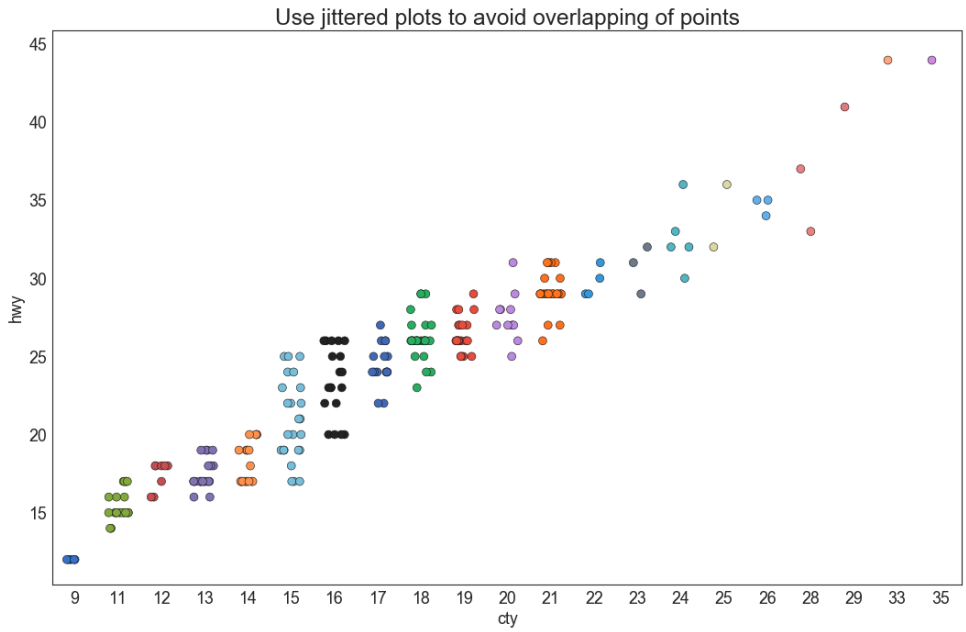


Figure 26.5: 抖动图

26.3.5 计数图（Counts Plot）

避免点重叠问题的另一个选择是增加点的大小，这取决于该点中有多少点。因此，点的大小越大，其周围的点的集中度越高。

```
# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/mpg_ggplot2.csv")
df_counts = df.groupby(['hwy', 'cty']).size().reset_index(name='counts')

# Draw Stripplot
fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
sns.stripplot(df_counts.cty, df_counts.hwy, size=df_counts.counts*2, ax=ax)

# Decorations
plt.title('Counts Plot - Size of circle is bigger as more points overlap', fontsize=22)
plt.show()
```

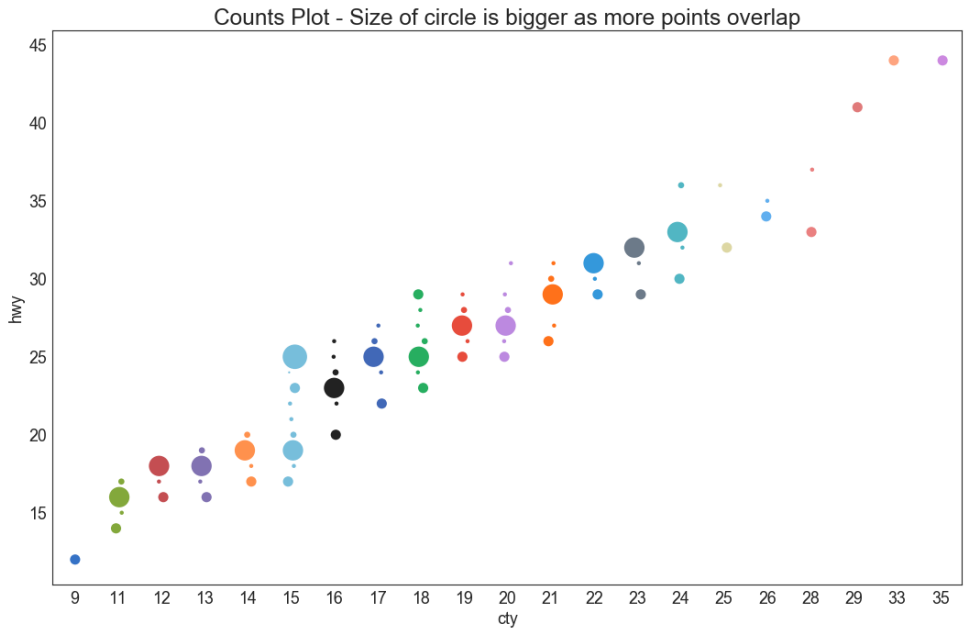


Figure 26.6: 计数图

26.3.6 边缘直方图（Marginal Histogram）

边缘直方图具有沿 X 和 Y 轴变量的直方图。这用于可视化 X 和 Y 之间的关系以及单独的 X 和 Y 的单变量分布。这种图经常用于探索性数据分析（EDA）。

```
# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    mpg_ggplot2.csv")

# Create Fig and gridspec
fig = plt.figure(figsize=(16, 10), dpi= 80)
grid = plt.GridSpec(4, 4, hspace=0.5, wspace=0.2)

# Define the axes
ax_main = fig.add_subplot(grid[:-1, :-1])
ax_right = fig.add_subplot(grid[:-1, -1], xticklabels=[], yticklabels=[])
ax_bottom = fig.add_subplot(grid[-1, 0:-1], xticklabels=[], yticklabels=[])

# Scatterplot on main ax
ax_main.scatter('displ', 'hwy', s=df.cty*4, c=df.manufacturer.astype('category'
    ).cat.codes, alpha=.9, data=df, cmap="tab10", edgecolors='gray', linewidths
    =.5)

# histogram on the right
ax_bottom.hist(df.displ, 40, histtype='stepfilled', orientation='vertical',
    color='deeppink')
ax_bottom.invert_yaxis()
```

```
19
20  # histogram in the bottom
21  ax_right.hist(df.hwy, 40, histtype='stepfilled', orientation='horizontal',
        color='deeppink')
22
23  # Decorations
24  ax_main.set(title='Scatterplot with Histograms \n displ vs hwy', xlabel='displ'
        , ylabel='hwy')
25  ax_main.title.set_fontsize(20)
26  for item in ([ax_main.xaxis.label, ax_main.yaxis.label] + ax_main.
        get_xticklabels() + ax_main.get_yticklabels()):
27      item.set_fontsize(14)
28
29  xlabels = ax_main.get_xticks().tolist()
30  ax_main.set_xticklabels(xlabels)
31  plt.show()
```

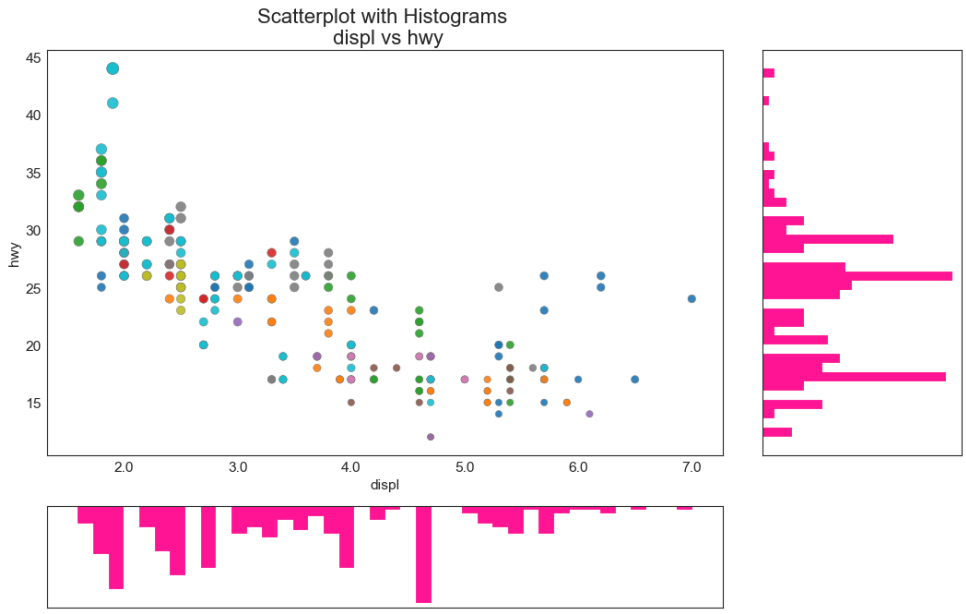


Figure 26.7: 边缘直方图

26.3.7 边缘箱形图（Marginal Boxplot）

边缘箱形图与边缘直方图具有相似的用途。然而，箱线图有助于精确定位 X 和 Y 的中位数、第 25 和第 75 百分位数。

```
1  # Import Data
2  df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
       mpg_ggplot2.csv")
3
4  # Create Fig and gridspec
```

```
fig = plt.figure(figsize=(16, 10), dpi= 80)
grid = plt.GridSpec(4, 4, hspace=0.5, wspace=0.2)

# Define the axes
ax_main = fig.add_subplot(grid[:-1, :-1])
ax_right = fig.add_subplot(grid[:-1, -1], xticklabels=[], yticklabels=[])
ax_bottom = fig.add_subplot(grid[-1, 0:-1], xticklabels=[], yticklabels=[])

# Scatterplot on main ax
ax_main.scatter('displ', 'hwy', s=df.cty*5, c=df.manufacturer.astype('category'
    ).cat.codes, alpha=.9, data=df, cmap="Set1", edgecolors='black', linewidths
    =.5)

# Add a graph in each part
sns.boxplot(df.hwy, ax=ax_right, orient="v")
sns.boxplot(df.displ, ax=ax_bottom, orient="h")

# Decorations ------------------
# Remove x axis name for the boxplot
ax_bottom.set(xlabel='')
ax_right.set(ylabel='')

# Main Title, Xlabel and YLabel
ax_main.set(title='Scatterplot with Histograms \n displ vs hwy', xlabel='displ'
    , ylabel='hwy')

# Set font size of different components
ax_main.title.set_fontsize(20)
for item in ([ax_main.xaxis.label, ax_main.yaxis.label] + ax_main.
    get_xticklabels() + ax_main.get_yticklabels()):
    item.set_fontsize(14)

plt.show()
```

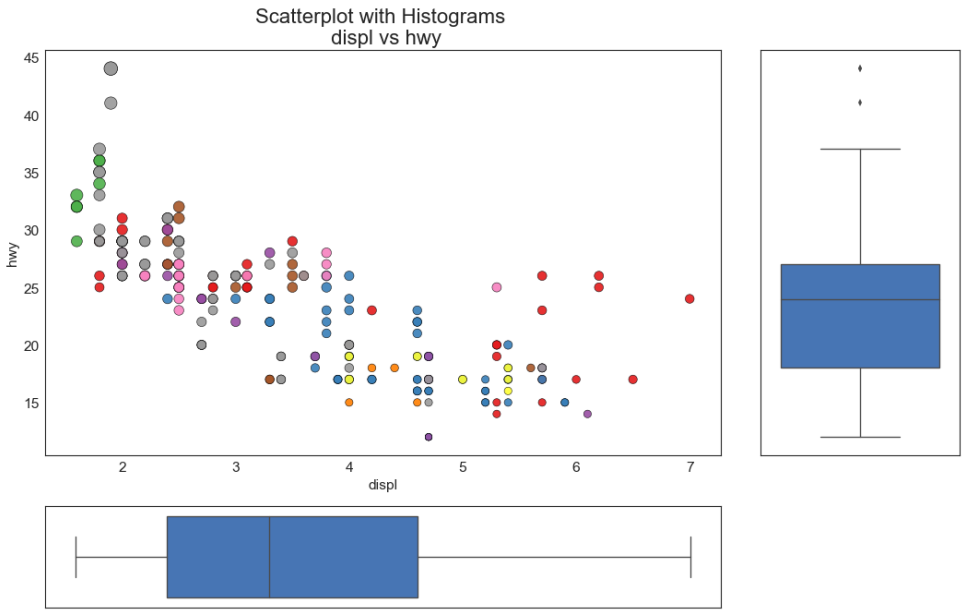


Figure 26.8: 边缘箱形图

26.3.8 相关图（Correllogram）

相关图用于直观地查看给定数据框（或二维数组）中所有可能的数值变量对之间的相关度量。

```
# Import Dataset
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")

# Plot
plt.figure(figsize=(12,10), dpi= 80)
sns.heatmap(df.corr(), xticklabels=df.corr().columns, yticklabels=df.corr().
    columns, cmap='RdYlGn', center=0, annot=True)

# Decorations
plt.title('Correlogram of mtcars', fontsize=22)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

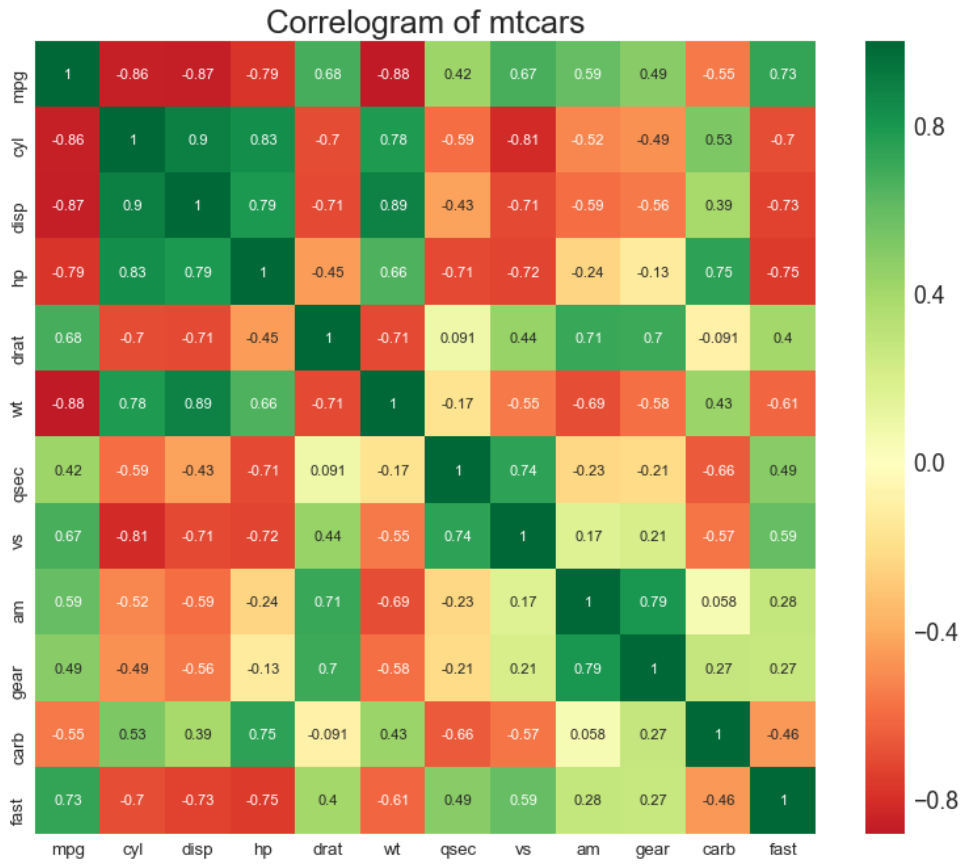


Figure 26.9: 相关图

26.3.9 矩阵图（Pairwise Plot）

矩阵图是探索性分析中的最爱，用于理解所有可能的数值变量对之间的关系。它是双变量分析的必备工具。

```
# Load Dataset
df = sns.load_dataset('iris')

# Plot
plt.figure(figsize=(10,8), dpi= 80)
sns.pairplot(df, kind="scatter", hue="species", plot_kws=dict(s=80, edgecolor="
    white", linewidth=2.5))
plt.show()
```

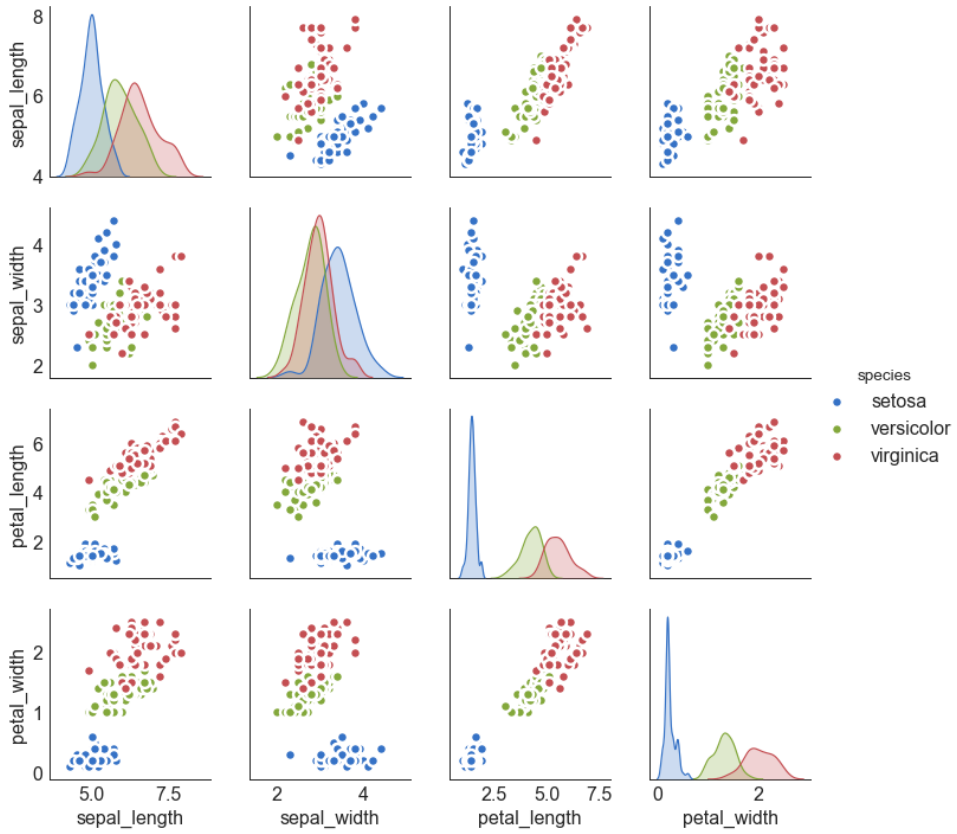


Figure 26.10: 矩阵图

```
# Load Dataset
df = sns.load_dataset('iris')

# Plot
plt.figure(figsize=(10,8), dpi= 80)
sns.pairplot(df, kind="reg", hue="species")
plt.show()
```

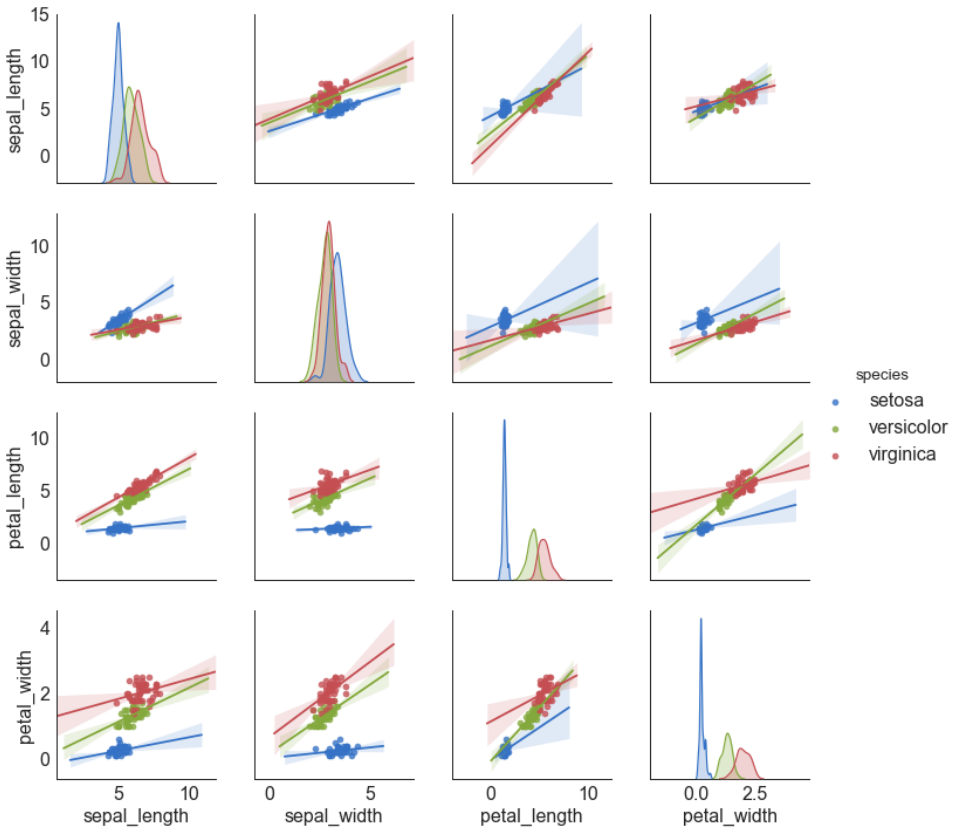


Figure 26.11: 矩阵图

26.4 偏差（Deviation）

26.4.1 发散型条形图（Diverging Bars）

如果您想根据单个指标查看项目的变化情况，并可视化此差异的顺序和数量，那么散型条形图（Diverging Bars）是一个很好的工具。它有助于快速区分数据中组的性能，并且非常直观，并且可以立即传达这一点。

```
# Prepare Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
x = df.loc[:, ['mpg']]
df['mpg_z'] = (x - x.mean())/x.std()
df['colors'] = ['red' if x < 0 else 'green' for x in df['mpg_z']]
df.sort_values('mpg_z', inplace=True)
df.reset_index(inplace=True)

# Draw plot
plt.figure(figsize=(14,10), dpi= 80)
plt.hlines(y=df.index, xmin=0, xmax=df.mpg_z, color=df.colors, alpha=0.4,
    linewidth=5)

```

```
# Decorations
plt.gca().set(ylabel='$Model$', xlabel='$Mileage$')
plt.yticks(df.index, df.cars, fontsize=12)
plt.title('Diverging Bars of Car Mileage', fontdict={'size':20})
plt.grid(linestyle='--', alpha=0.5)
plt.show()
```

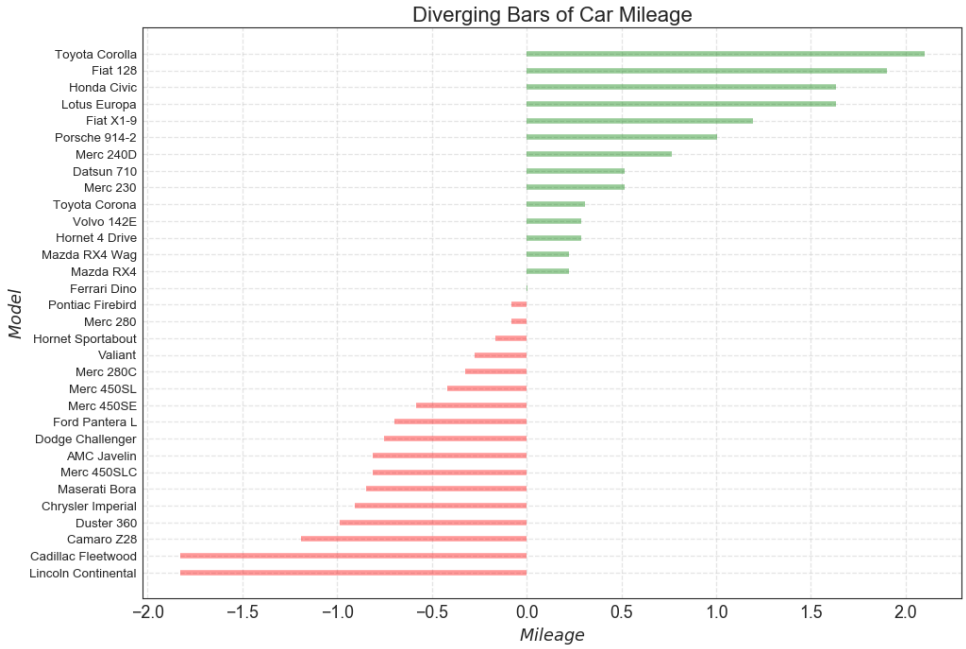


Figure 26.12: 发散型条形图

26.4.2 发散型文本（Diverging Texts）

发散型文本（Diverging Texts）与发散型条形图（Diverging Bars）相似，如果你想以一种漂亮和可呈现的方式显示图表中每个项目的价值，就可以使用这种方法。

```
# Prepare Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
x = df.loc[:, ['mpg']]
df['mpg_z'] = (x - x.mean())/x.std()
df['colors'] = ['red' if x < 0 else 'green' for x in df['mpg_z']]
df.sort_values('mpg_z', inplace=True)
df.reset_index(inplace=True)

# Draw plot
plt.figure(figsize=(14,14), dpi= 80)
plt.hlines(y=df.index, xmin=0, xmax=df.mpg_z)
for x, y, tex in zip(df.mpg_z, df.index, df.mpg_z):
    t = plt.text(x, y, round(tex, 2), horizontalalignment='right' if x < 0 else
        'left',
```

```
                      verticalalignment='center', fontdict={'color':'red' if x < 0
                          else 'green', 'size':14})

# Decorations
plt.yticks(df.index, df.cars, fontsize=12)
plt.title('Diverging Text Bars of Car Mileage', fontdict={'size':20})
plt.grid(linestyle='--', alpha=0.5)
plt.xlim(-2.5, 2.5)
plt.show()
```

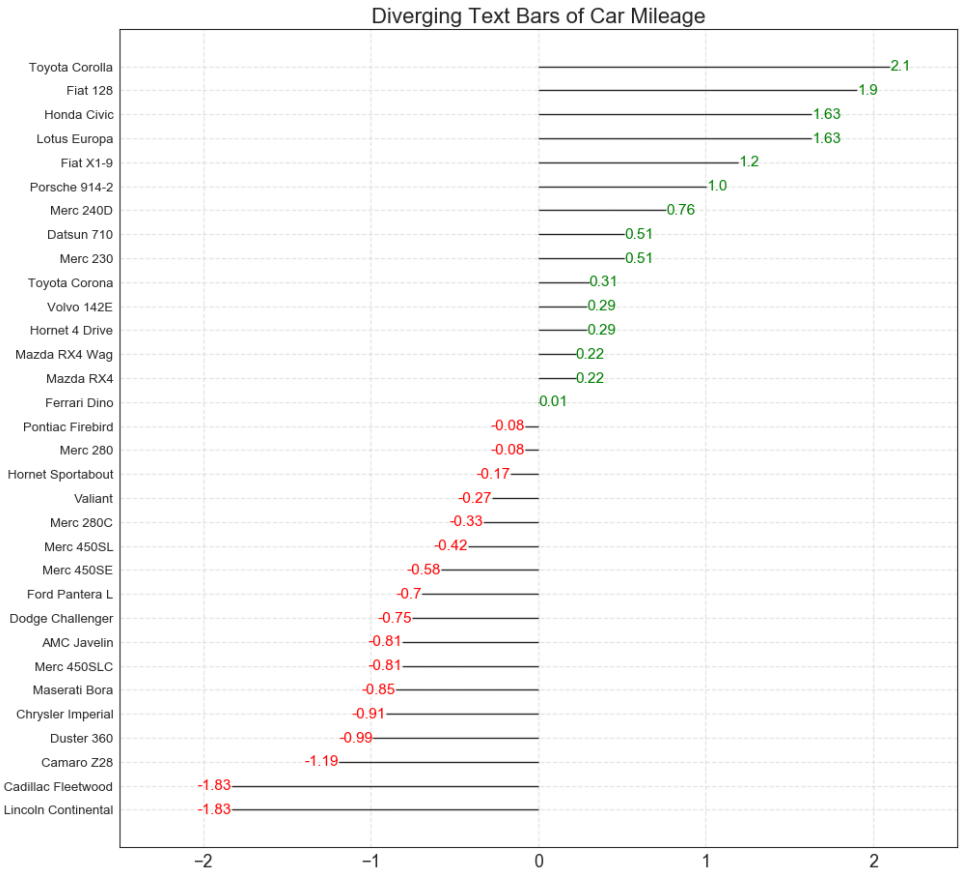


Figure 26.13: 发散型文本

26.4.3 发散型包点图（Diverging Dot Plot）

发散型包点图（Diverging Dot Plot）也类似于发散型条形图（Diverging Bars）。然而，与发散型条形图（Diverging Bars）相比，条的缺失减少了组之间的对比度和差异。

```
# Prepare Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
x = df.loc[:, ['mpg']]
df['mpg_z'] = (x - x.mean())/x.std()
```

```
df['colors'] = ['red' if x < 0 else 'darkgreen' for x in df['mpg_z']]
df.sort_values('mpg_z', inplace=True)
df.reset_index(inplace=True)

# Draw plot
plt.figure(figsize=(14,16), dpi= 80)
plt.scatter(df.mpg_z, df.index, s=450, alpha=.6, color=df.colors)
for x, y, tex in zip(df.mpg_z, df.index, df.mpg_z):
    t = plt.text(x, y, round(tex, 1), horizontalalignment='center',
                 verticalalignment='center', fontdict={'color':'white'})

# Decorations
# Lighten borders
plt.gca().spines["top"].set_alpha(.3)
plt.gca().spines["bottom"].set_alpha(.3)
plt.gca().spines["right"].set_alpha(.3)
plt.gca().spines["left"].set_alpha(.3)

plt.yticks(df.index, df.cars)
plt.title('Diverging Dotplot of Car Mileage', fontdict={'size':20})
plt.xlabel('$Mileage$')
plt.grid(linestyle='--', alpha=0.5)
plt.xlim(-2.5, 2.5)
plt.show()
```

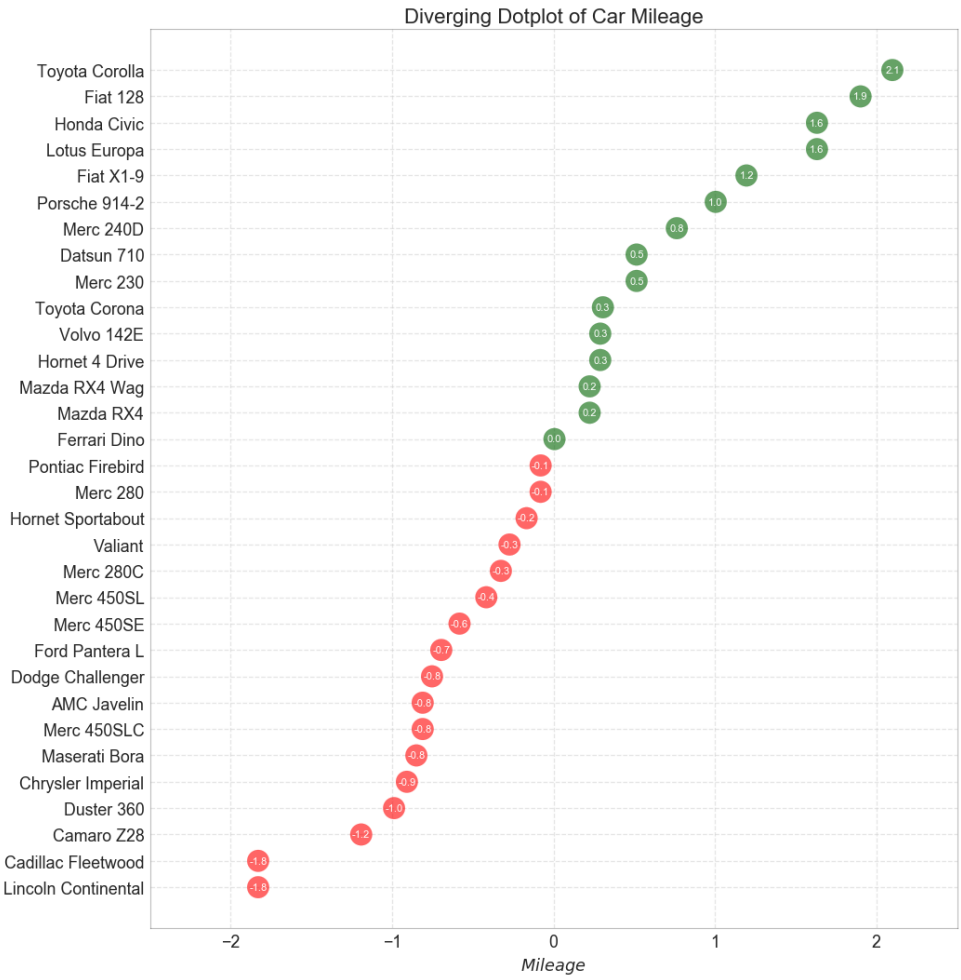


Figure 26.14: 发散型包点图

26.4.4 带标记的发散型棒棒糖图（Diverging Lollipop Chart with Markers）

带标记的棒棒糖图通过强调您想要引起注意的任何重要数据点并在图表中适当地给出推理，提供了一种对差异进行可视化的灵活方式。

```
# Prepare Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
x = df.loc[:, ['mpg']]
df['mpg_z'] = (x - x.mean())/x.std()
df['colors'] = 'black'

# color fiat differently
df.loc[df.cars == 'Fiat X1-9', 'colors'] = 'darkorange'
df.sort_values('mpg_z', inplace=True)
df.reset_index(inplace=True)

```

```
12
13  # Draw plot
14  import matplotlib.patches as patches
15
16  plt.figure(figsize=(14,16), dpi= 80)
17  plt.hlines(y=df.index, xmin=0, xmax=df.mpg_z, color=df.colors, alpha=0.4,
        linewidth=1)
18  plt.scatter(df.mpg_z, df.index, color=df.colors, s=[600 if x == 'Fiat X1-9'
        else 300 for x in df.cars], alpha=0.6)
19  plt.yticks(df.index, df.cars)
20  plt.xticks(fontsize=12)
21
22  # Annotate
23  plt.annotate('Mercedes Models', xy=(0.0, 11.0), xytext=(1.0, 11), xycoords='
        data',
24               fontsize=15, ha='center', va='center',
25               bbox=dict(boxstyle='square', fc='firebrick'),
26               arrowprops=dict(arrowstyle='-[, widthB=2.0, lengthB=1.5', lw=2.0,
                     color='steelblue'), color='white')
27
28  # Add Patches
29  p1 = patches.Rectangle((-2.0, -1), width=.3, height=3, alpha=.2, facecolor='red
        ')
30  p2 = patches.Rectangle((1.5, 27), width=.8, height=5, alpha=.2, facecolor='
        green')
31  plt.gca().add_patch(p1)
32  plt.gca().add_patch(p2)
33
34  # Decorate
35  plt.title('Diverging Bars of Car Mileage', fontdict={'size':20})
36  plt.grid(linestyle='--', alpha=0.5)
37  plt.show()
```

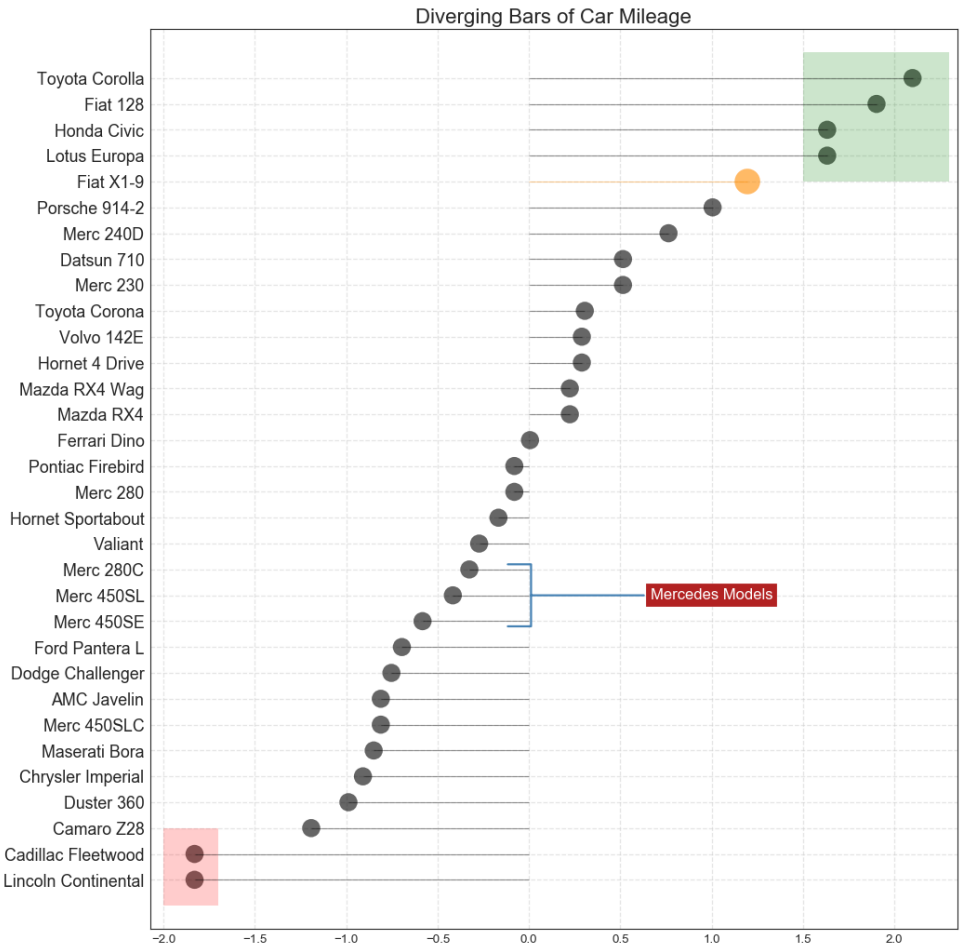


Figure 26.15: 带标记的发散型棒棒糖图

26.4.5 面积图（Area Chart）

通过对轴和线之间的区域进行着色，面积图不仅强调峰和谷，而且还强调高点和低点的持续时间。高点持续时间越长，线下面积越大。

```
import numpy as np
import pandas as pd

# Prepare Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/economics.csv"
    , parse_dates=['date']).head(100)
x = np.arange(df.shape[0])
y_returns = (df.psavert.diff().fillna(0)/df.psavert.shift(1)).fillna(0) * 100

# Plot
plt.figure(figsize=(16,10), dpi= 80)
```

```
plt.fill_between(x[1:], y_returns[1:], 0, where=y_returns[1:] >= 0, facecolor='green', interpolate=True, alpha=0.7)
plt.fill_between(x[1:], y_returns[1:], 0, where=y_returns[1:] <= 0, facecolor='red', interpolate=True, alpha=0.7)

# Annotate
plt.annotate('Peak \n1975', xy=(94.0, 21.0), xytext=(88.0, 28),
             bbox=dict(boxstyle='square', fc='firebrick'),
             arrowprops=dict(facecolor='steelblue', shrink=0.05), fontsize=15, color='white')


# Decorations
xtickvals = [str(m)[:3].upper()+"-"+str(y) for y,m in zip(df.date.dt.year, df.date.dt.month_name())]
plt.gca().set_xticks(x[::6])
plt.gca().set_xticklabels(xtickvals[::6], rotation=90, fontdict={'horizontalalignment': 'center', 'verticalalignment': 'center_baseline'})
plt.ylim(-35,35)
plt.xlim(1,100)
plt.title("Month Economics Return %", fontsize=22)
plt.ylabel('Monthly returns %')
plt.grid(alpha=0.5)
plt.show()
```

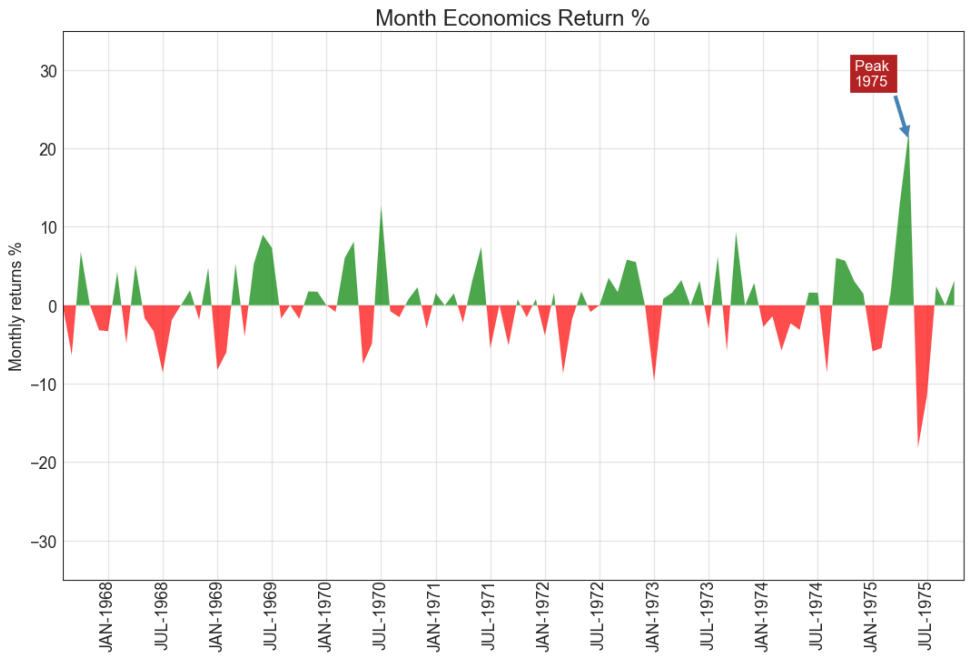


Figure 26.16: 面积图

26.5 排序（Ranking）

26.5.1 有序条形图（Ordered Bar Chart）

有序条形图有效地传达了项目的排名顺序。但是，在图表上方添加度量标准的值，用户可以从图表本身获取精确信息。

```
# Prepare Data
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")
df = df_raw[['cty', 'manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
df.sort_values('cty', inplace=True)
df.reset_index(inplace=True)

# Draw plot
import matplotlib.patches as patches

fig, ax = plt.subplots(figsize=(16,10), facecolor='white', dpi= 80)
ax.vlines(x=df.index, ymin=0, ymax=df.cty, color='firebrick', alpha=0.7, linewidth=20)

# Annotate Text
for i, cty in enumerate(df.cty):
    ax.text(i, cty+0.5, round(cty, 1), horizontalalignment='center')


# Title, Label, Ticks and Ylim
ax.set_title('Bar Chart for Highway Mileage', fontdict={'size':22})
ax.set(ylabel='Miles Per Gallon', ylim=(0, 30))
plt.xticks(df.index, df.manufacturer.str.upper(), rotation=60, horizontalalignment='right', fontsize=12)

# Add patches to color the X axis labels
p1 = patches.Rectangle((.57, -0.005), width=.33, height=.13, alpha=.1, facecolor='green', transform=fig.transFigure)
p2 = patches.Rectangle((.124, -0.005), width=.446, height=.13, alpha=.1, facecolor='red', transform=fig.transFigure)
fig.add_artist(p1)
fig.add_artist(p2)
plt.show()
```

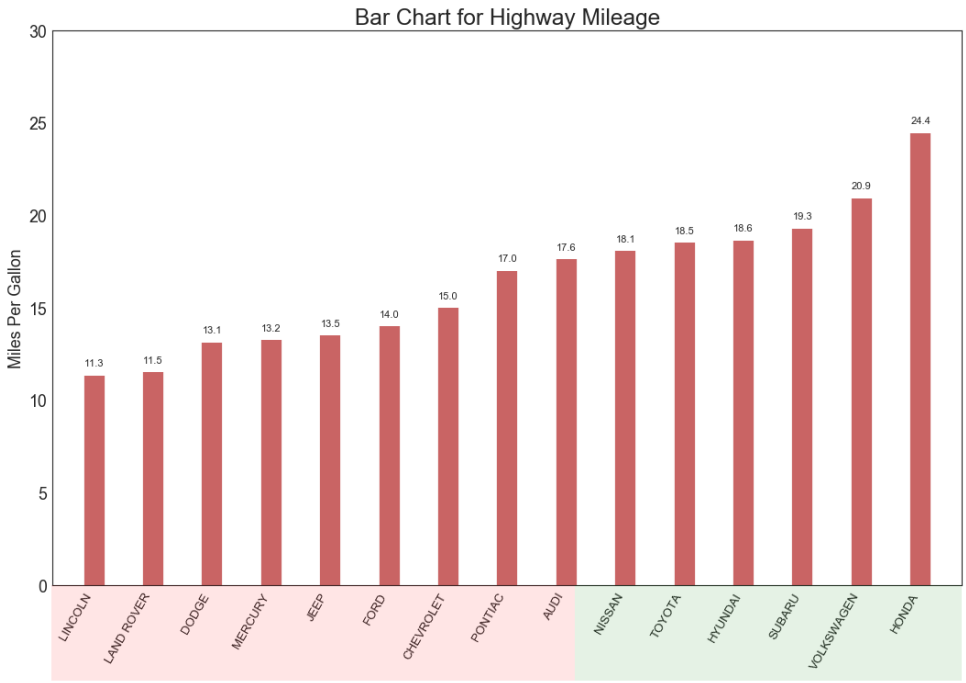


Figure 26.17: 有序条形图

26.5.2 棒棒糖图（Lollipop Chart）

棒棒糖图表以一种视觉上令人愉悦的方式提供与有序条形图类似的目的。

```
# Prepare Data
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
    mpg_ggplot2.csv")
df = df_raw[['cty', 'manufacturer']].groupby('manufacturer').apply(lambda x: x.
    mean())
df.sort_values('cty', inplace=True)
df.reset_index(inplace=True)

# Draw plot
fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
ax.vlines(x=df.index, ymin=0, ymax=df.cty, color='firebrick', alpha=0.7,
    linewidth=2)
ax.scatter(x=df.index, y=df.cty, s=75, color='firebrick', alpha=0.7)

# Title, Label, Ticks and Ylim
ax.set_title('Lollipop Chart for Highway Mileage', fontdict={'size':22})
ax.set_ylabel('Miles Per Gallon')
ax.set_xticks(df.index)
ax.set_xticklabels(df.manufacturer.str.upper(), rotation=60, fontdict={'
    horizontalalignment': 'right', 'size':12})
ax.set_ylim(0, 30)
```

```
18
19 # Annotate
20 for row in df.itertuples():
21     ax.text(row.Index, row.cty+.5, s=round(row.cty, 2), horizontalalignment= '
       center', verticalalignment='bottom', fontsize=14)
22
23 plt.show()
```

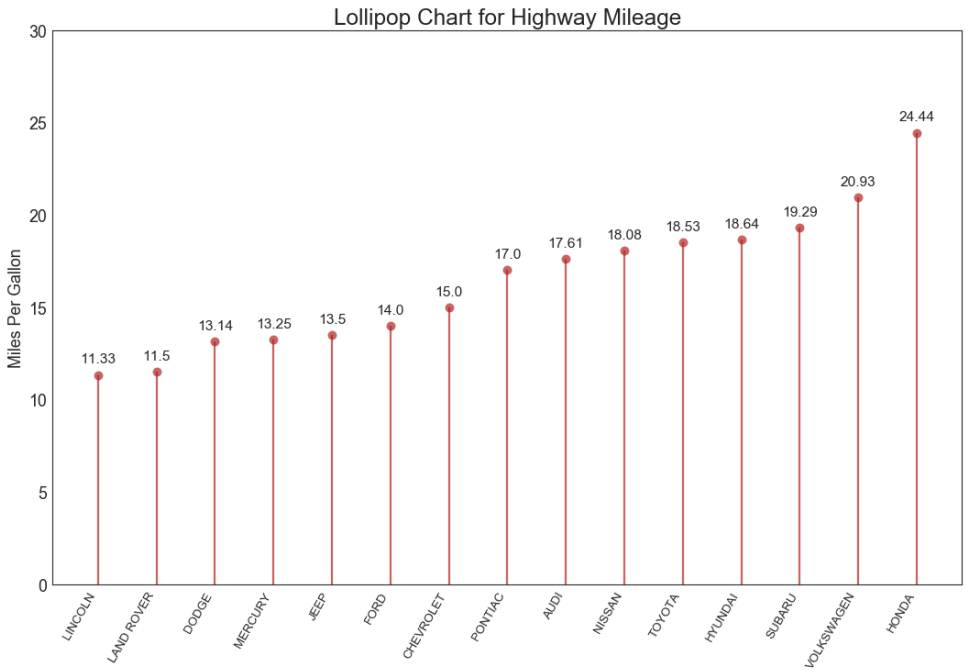


Figure 26.18: 棒棒糖图

26.5.3 包点图（Dot Plot）

包点图表传达了项目的排名顺序，并且由于它沿水平轴对齐，因此您可以更容易地看到点彼此之间的距离。

```
1  # Prepare Data
2  df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
       mpg_ggplot2.csv")
3  df = df_raw[['cty', 'manufacturer']].groupby('manufacturer').apply(lambda x: x.
       mean())
4  df.sort_values('cty', inplace=True)
5  df.reset_index(inplace=True)
6
7  # Draw plot
8  fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
9  ax.hlines(y=df.index, xmin=11, xmax=26, color='gray', alpha=0.7, linewidth=1,
       linestyles='dashdot')
10 ax.scatter(y=df.index, x=df.cty, s=75, color='firebrick', alpha=0.7)
```

```
11
12 # Title, Label, Ticks and Ylim
13 ax.set_title('Dot Plot for Highway Mileage', fontdict={'size':22})
14 ax.set_xlabel('Miles Per Gallon')
15 ax.set_yticks(df.index)
16 ax.set_yticklabels(df.manufacturer.str.title(), fontdict={'horizontalalignment'
       : 'right'})
17 ax.set_xlim(10, 27)
18 plt.show()
```

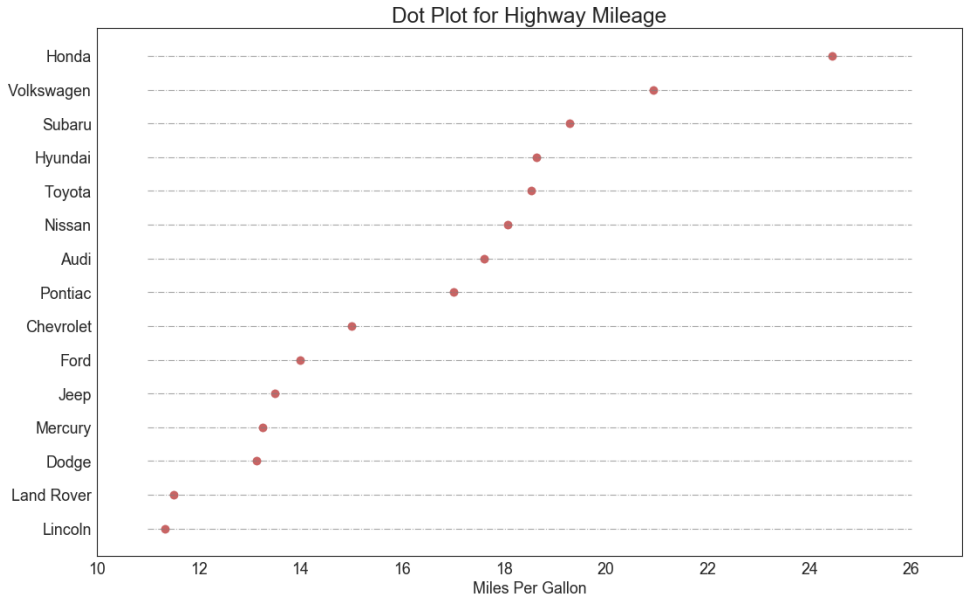


Figure 26.19: 包点图

26.5.4 坡度图（Slope Chart）

坡度图最适合比较给定人/项目的“前”和“后”位置。

```
1 import matplotlib.lines as mlines
2 # Import Data
3 df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
      gdppercap.csv")
4
5 left_label = [str(c) + ', '+ str(round(y)) for c, y in zip(df.continent, df['
      1952'])]
6 right_label = [str(c) + ', '+ str(round(y)) for c, y in zip(df.continent, df['
      1957'])]
7 klass = ['red' if (y1-y2) < 0 else 'green' for y1, y2 in zip(df['1952'], df['
      1957'])]
8
9 # draw line
```

```
# https://stackoverflow.com/questions/36470343/how-to-draw-a-line-with-matplotlib/36479941
def newline(p1, p2, color='black'):
    ax = plt.gca()
    l = mlines.Line2D([p1[0],p2[0]], [p1[1],p2[1]], color='red' if p1[1]-p2[1] > 0 else 'green', marker='o', markersize=6)
    ax.add_line(l)
    return l

fig, ax = plt.subplots(1,1,figsize=(14,14), dpi= 80)

# Vertical Lines
ax.vlines(x=1, ymin=500, ymax=13000, color='black', alpha=0.7, linewidth=1, linestyles='dotted')
ax.vlines(x=3, ymin=500, ymax=13000, color='black', alpha=0.7, linewidth=1, linestyles='dotted')

# Points
ax.scatter(y=df['1952'], x=np.repeat(1, df.shape[0]), s=10, color='black', alpha=0.7)
ax.scatter(y=df['1957'], x=np.repeat(3, df.shape[0]), s=10, color='black', alpha=0.7)

# Line Segmentsand Annotation
for p1, p2, c in zip(df['1952'], df['1957'], df['continent']):
    newline([1,p1], [3,p2])
    ax.text(1-0.05, p1, c + ', ' + str(round(p1)), horizontalalignment='right', verticalalignment='center', fontdict={'size':14})
    ax.text(3+0.05, p2, c + ', ' + str(round(p2)), horizontalalignment='left', verticalalignment='center', fontdict={'size':14})

# 'Before' and 'After' Annotations
ax.text(1-0.05, 13000, 'BEFORE', horizontalalignment='right', verticalalignment='center', fontdict={'size':18, 'weight':700})
ax.text(3+0.05, 13000, 'AFTER', horizontalalignment='left', verticalalignment='center', fontdict={'size':18, 'weight':700})

# Decoration
ax.set_title("Slopechart: Comparing GDP Per Capita between 1952 vs 1957", fontdict={'size':22})
ax.set(xlim=(0,4), ylim=(0,14000), ylabel='Mean GDP Per Capita')
ax.set_xticks([1,3])
ax.set_xticklabels(["1952", "1957"])
plt.yticks(np.arange(500, 13000, 2000), fontsize=12)

# Lighten borders
plt.gca().spines["top"].set_alpha(.0)
plt.gca().spines["bottom"].set_alpha(.0)
plt.gca().spines["right"].set_alpha(.0)
plt.gca().spines["left"].set_alpha(.0)
plt.show()
```

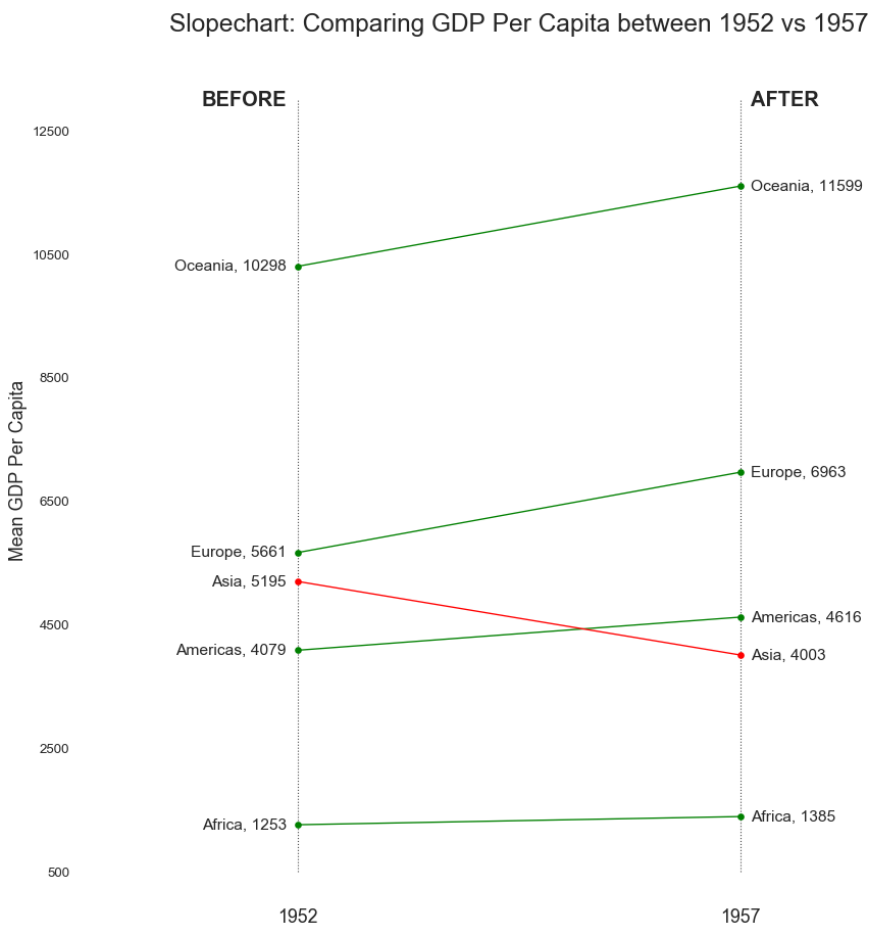


Figure 26.20: 坡度图

26.5.5 哑铃图（Dumbbell Plot）

哑铃图表传达了各种项目的“前”和“后”位置以及项目的等级排序。如果您想要将特定项目/计划对不同对象的影响可视化，那么它非常有用。

```
import matplotlib.lines as mlines

# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    health.csv")
df.sort_values('pct_2014', inplace=True)
df.reset_index(inplace=True)

# Func to draw line segment
def newline(p1, p2, color='black'):
```

```
    ax = plt.gca()
    l = mlines.Line2D([p1[0],p2[0]], [p1[1],p2[1]], color='skyblue')
    ax.add_line(l)
    return l

# Figure and Axes
fig, ax = plt.subplots(1,1,figsize=(14,14), facecolor='#f7f7f7', dpi= 80)

# Vertical Lines
ax.vlines(x=.05, ymin=0, ymax=26, color='black', alpha=1, linewidth=1,
    linestyles='dotted')
ax.vlines(x=.10, ymin=0, ymax=26, color='black', alpha=1, linewidth=1,
    linestyles='dotted')
ax.vlines(x=.15, ymin=0, ymax=26, color='black', alpha=1, linewidth=1,
    linestyles='dotted')
ax.vlines(x=.20, ymin=0, ymax=26, color='black', alpha=1, linewidth=1,
    linestyles='dotted')

# Points
ax.scatter(y=df['index'], x=df['pct_2013'], s=50, color='#0e668b', alpha=0.7)
ax.scatter(y=df['index'], x=df['pct_2014'], s=50, color='#a3c4dc', alpha=0.7)

# Line Segments
for i, p1, p2 in zip(df['index'], df['pct_2013'], df['pct_2014']):
    newline([p1, i], [p2, i])

# Decoration
ax.set_facecolor('#f7f7f7')
ax.set_title("Dumbell Chart: Pct Change - 2013 vs 2014", fontdict={'size':22})
ax.set(xlim=(0,.25), ylim=(-1, 27), ylabel='Mean GDP Per Capita')
ax.set_xticks([.05, .1, .15, .20])
ax.set_xticklabels(['5%', '15%', '20%', '25%'])
ax.set_xticklabels(['5%', '15%', '20%', '25%'])
plt.show()
```

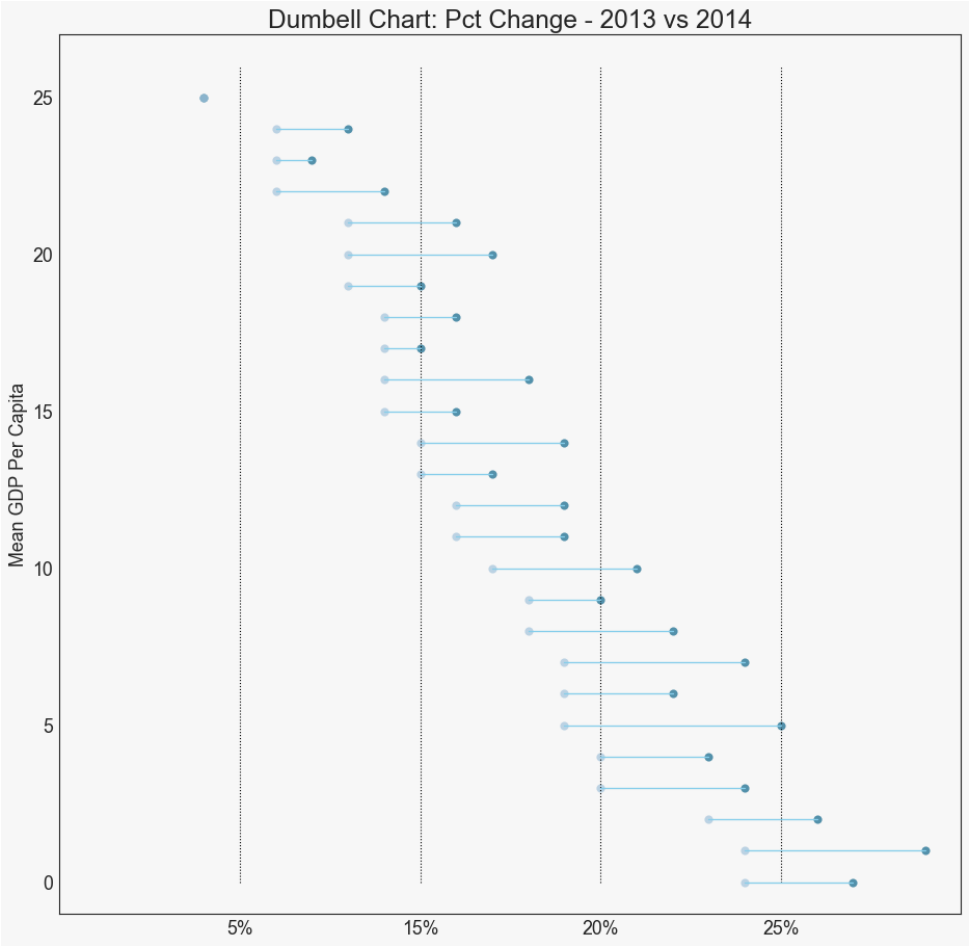


Figure 26.21: 哑铃图

26.6 分布（Distribution）

26.6.1 连续变量的直方图（Histogram for Continuous Variable）

直方图显示给定变量的频率分布。下面的图表示基于类型变量对频率条进行分组，从而更好地了解连续变量和类型变量。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Prepare data
x_var = 'displ'
groupby_var = 'class'
df_agg = df.loc[:, [x_var, groupby_var]].groupby(groupby_var)
vals = [df[x_var].values.tolist() for i, df in df_agg]
```

```
9
10 # Draw
11 plt.figure(figsize=(16,9), dpi= 80)
12 colors = [plt.cm.Spectral(i/float(len(vals)-1)) for i in range(len(vals))]
13 n, bins, patches = plt.hist(vals, 30, stacked=True, density=False, color=colors
       [:len(vals)])
14
15 # Decoration
16 plt.legend({group:col for group, col in zip(np.unique(df[groupby_var]).tolist()
       , colors[:len(vals)])})
17 plt.title(f"Stacked Histogram of ${x_var}$ colored by ${groupby_var}$",
       fontsize=22)
18 plt.xlabel(x_var)
19 plt.ylabel("Frequency")
20 plt.ylim(0, 25)
21 plt.xticks(ticks=bins[::3], labels=[round(b,1) for b in bins[::3]])
22 plt.show()
```

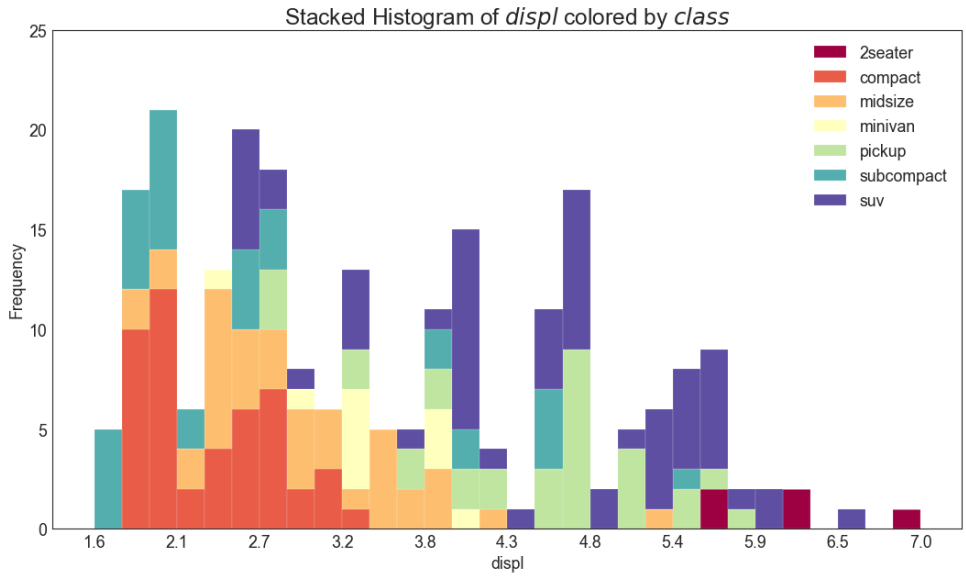


Figure 26.22: 连续变量的直方图

26.6.2 类型变量的直方图（Histogram for Categorical Variable）

类型变量的直方图显示该变量的频率分布。通过对条形图进行着色，可以将分布与表示颜色的另一个类型变量相关联。

```
1 # Import Data
2 df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.
      csv")
3
4 # Prepare data
```

```
x_var = 'manufacturer'
groupby_var = 'class'
df_agg = df.loc[:, [x_var, groupby_var]].groupby(groupby_var)
vals = [df[x_var].values.tolist() for i, df in df_agg]

# Draw
plt.figure(figsize=(16,9), dpi= 80)
colors = [plt.cm.Spectral(i/float(len(vals)-1)) for i in range(len(vals))]
n, bins, patches = plt.hist(vals, df[x_var].unique().__len__(), stacked=True,
    density=False, color=colors[:len(vals)])

# Decoration
plt.legend({group:col for group, col in zip(np.unique(df[groupby_var]).tolist()
    , colors[:len(vals)])})
plt.title(f"Stacked Histogram of ${x_var}$ colored by ${groupby_var}$",
    fontsize=22)
plt.xlabel(x_var)
plt.ylabel("Frequency")
plt.ylim(0, 40)
plt.xticks(ticks=bins, labels=np.unique(df[x_var]).tolist(), rotation=90,
    horizontalalignment='left')
plt.show()
```

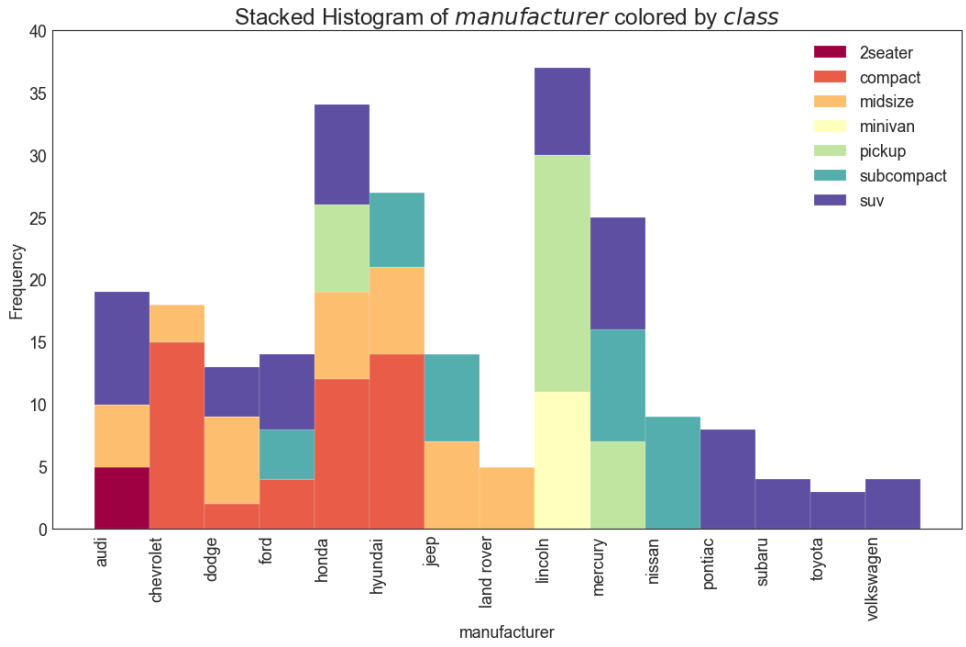


Figure 26.23: 类型变量的直方图

26.6.3 密度图（Density Plot）

密度图是一种常用工具，用于可视化连续变量的分布。通过“响应”变量对它们进行分组，您可以检查 X 和 Y 之间的关系。以下情况用于表示目的，以描述城市里程的分布如何随着汽缸数的变化而变化。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(16,10), dpi= 80)
sns.kdeplot(df.loc[df['cyl'] == 4, "cty"], shade=True, color="g", label="Cyl=4", alpha=.7)
sns.kdeplot(df.loc[df['cyl'] == 5, "cty"], shade=True, color="deeppink", label="Cyl=5", alpha=.7)
sns.kdeplot(df.loc[df['cyl'] == 6, "cty"], shade=True, color="dodgerblue", label="Cyl=6", alpha=.7)
sns.kdeplot(df.loc[df['cyl'] == 8, "cty"], shade=True, color="orange", label="Cyl=8", alpha=.7)

# Decoration
plt.title('Density Plot of City Mileage by n_Cylinders', fontsize=22)
plt.legend()
plt.show()
```

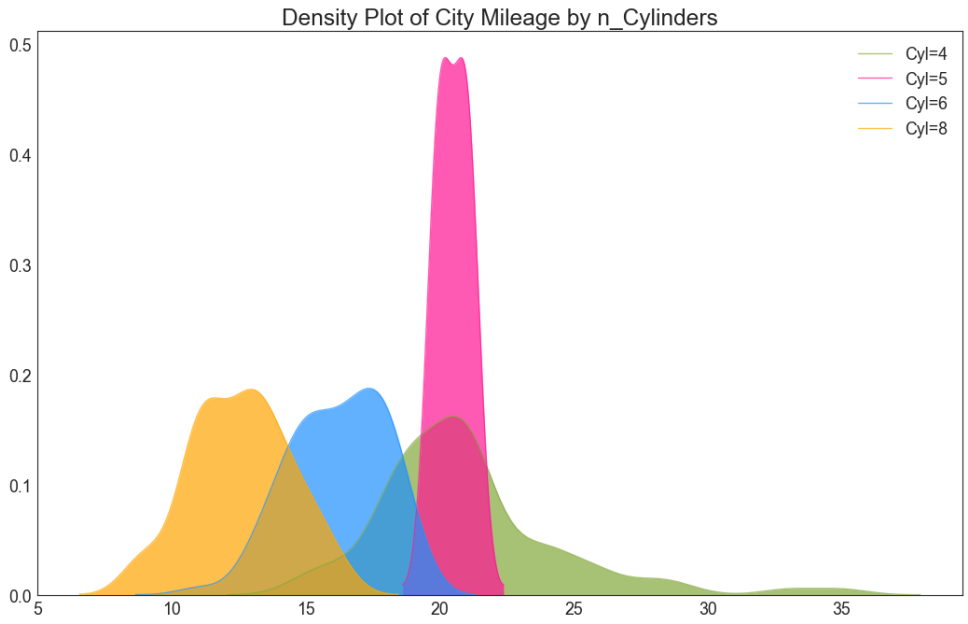


Figure 26.24: 密度图

26.6.4 直方密度线图（Density Curves with Histogram）

带有直方图的密度曲线汇集了两个图所传达的集体信息，因此您可以将它们放在一个图中而不是两个图中。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(13,10), dpi= 80)
sns.distplot(df.loc[df['class'] == 'compact', "cty"], color="dodgerblue", label="Compact", hist_kws={'alpha':.7}, kde_kws={'linewidth':3})
sns.distplot(df.loc[df['class'] == 'suv', "cty"], color="orange", label="SUV", hist_kws={'alpha':.7}, kde_kws={'linewidth':3})
sns.distplot(df.loc[df['class'] == 'minivan', "cty"], color="g", label="minivan", hist_kws={'alpha':.7}, kde_kws={'linewidth':3})
plt.ylim(0, 0.35)

# Decoration
plt.title('Density Plot of City Mileage by Vehicle Type', fontsize=22)
plt.legend()
plt.show()
```

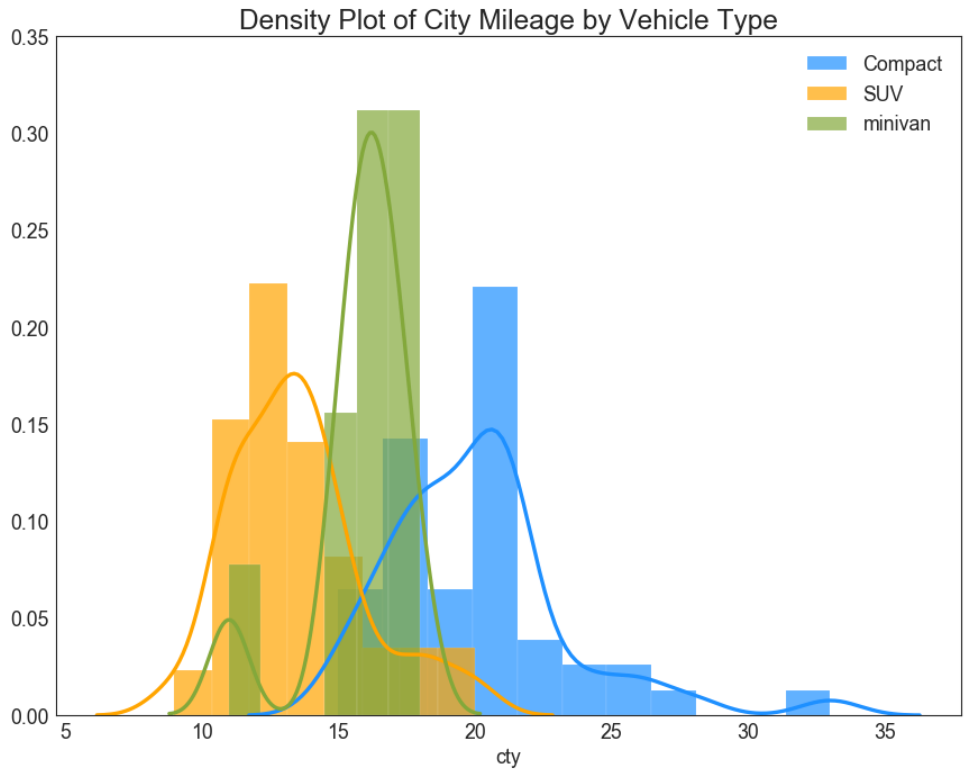


Figure 26.25: 直方密度线图

26.6.5 Joy Plot

Joy Plot 允许不同组的密度曲线重叠，这是一种可视化大量分组数据的彼此关系分布的好方法。它看起来很悦目，并清楚地传达了正确的信息。它可以使用基于 matplotlib 的 joypy 包轻松构建。(『Python 数据之道』注：需要安装 joypy 库)

```
# !pip install joypy
# Python数据之道  备注
import joypy

# Import Data
mpg = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(16,10), dpi= 80)
fig, axes = joypy.joyplot(mpg, column=['hwy', 'cty'], by="class", ylim='own', figsize=(14,10))

# Decoration
plt.title('Joy Plot of City and Highway Mileage by Class', fontsize=22)
plt.show()
```

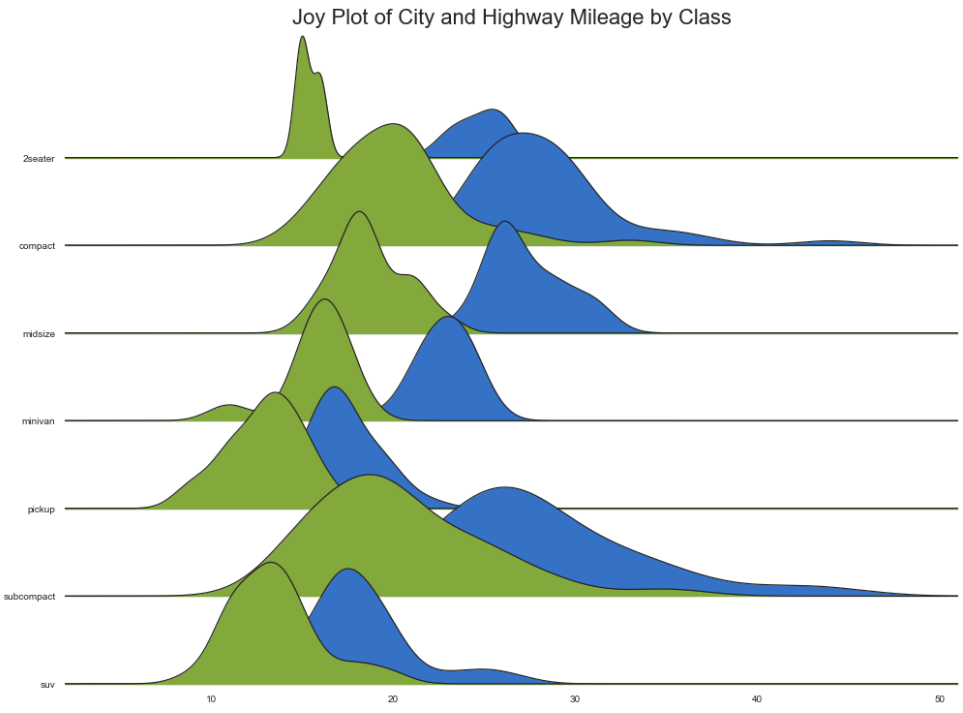


Figure 26.26: Joy Plot

26.6.6 分布式包点图（Distributed Dot Plot）

分布式包点图显示按组分割的点的单变量分布。点数越暗，该区域的数据点集中度越高。通过对中位数进行不同着色，组的真实定位立即变得明显。

```
import matplotlib.patches as mpatches

# Prepare Data
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")
cyl_colors = {4:'tab:red', 5:'tab:green', 6:'tab:blue', 8:'tab:orange'}
df_raw['cyl_color'] = df_raw.cyl.map(cyl_colors)

# Mean and Median city mileage by make
df = df_raw[['cty', 'manufacturer']].groupby('manufacturer').apply(lambda x: x.mean())
df.sort_values('cty', ascending=False, inplace=True)
df.reset_index(inplace=True)
df_median = df_raw[['cty', 'manufacturer']].groupby('manufacturer').apply(lambda x: x.median())

# Draw horizontal lines
fig, ax = plt.subplots(figsize=(16,10), dpi= 80)
ax.hlines(y=df.index, xmin=0, xmax=40, color='gray', alpha=0.5, linewidth=.5, linestyles='dashdot')

# Draw the Dots
for i, make in enumerate(df.manufacturer):
    df_make = df_raw.loc[df_raw.manufacturer==make, :]
    ax.scatter(y=np.repeat(i, df_make.shape[0]), x='cty', data=df_make, s=75, edgecolors='gray', c='w', alpha=0.5)
    ax.scatter(y=i, x='cty', data=df_median.loc[df_median.index==make, :], s=75, c='firebrick')

# Annotate
ax.text(33, 13, "$red \; dots \; are \; the \: median$", fontdict={'size':12}, color='firebrick')

# Decorations
red_patch = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='firebrick', label="Median")
plt.legend(handles=red_patch)
ax.set_title('Distribution of City Mileage by Make', fontdict={'size':22})
ax.set_xlabel('Miles Per Gallon (City)', alpha=0.7)
ax.set_yticks(df.index)
ax.set_yticklabels(df.manufacturer.str.title(), fontdict={'horizontalalignment': 'right'}, alpha=0.7)
ax.set_xlim(1, 40)
plt.xticks(alpha=0.7)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["bottom"].set_visible(False)
```

```
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis='both', alpha=.4, linewidth=.1)
plt.show()
```

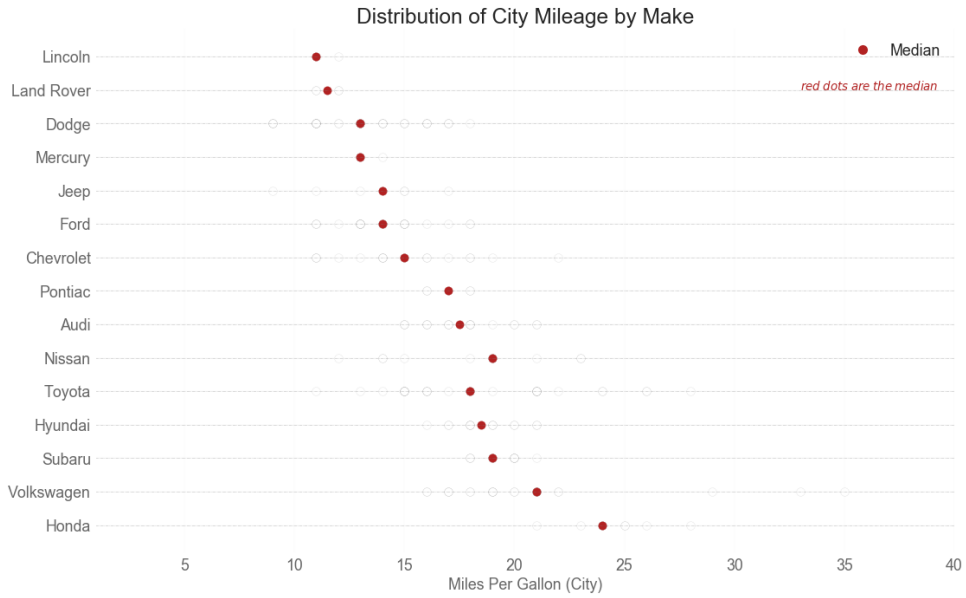


Figure 26.27: 分布式包点图

26.6.7 箱形图（Box Plot）

箱形图是一种可视化分布的好方法，记住中位数、第 25 个第 45 个四分位数和异常值。但是，您需要注意解释可能会扭曲该组中包含的点数的框的大小。因此，手动提供每个框中的观察数量可以帮助克服这个缺点。

例如，左边的前两个框具有相同大小的框，即使它们的值分别是 5 和 47。因此，写入该组中的观察数量是必要的。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(13,10), dpi= 80)
sns.boxplot(x='class', y='hwy', data=df, notch=False)

# Add N Obs inside boxplot (optional)
def add_n_obs(df,group_col,y):
    medians_dict = {grp[0]:grp[1][y].median() for grp in df.groupby(group_col)}
    xticklabels = [x.get_text() for x in plt.gca().get_xticklabels()]
    n_obs = df.groupby(group_col)[y].size().values
    for (x, xticklabel), n_ob in zip(enumerate(xticklabels), n_obs):
        plt.text(x, medians_dict[xticklabel]*1.01, "#obs : "+str(n_ob),
            horizontalalignment='center', fontdict={'size':14}, color='white')
```

```
15
16 add_n_obs(df,group_col='class',y='hwy')
17
18 # Decoration
19 plt.title('Box Plot of Highway Mileage by Vehicle Class', fontsize=22)
20 plt.ylim(10, 40)
21 plt.show()
```

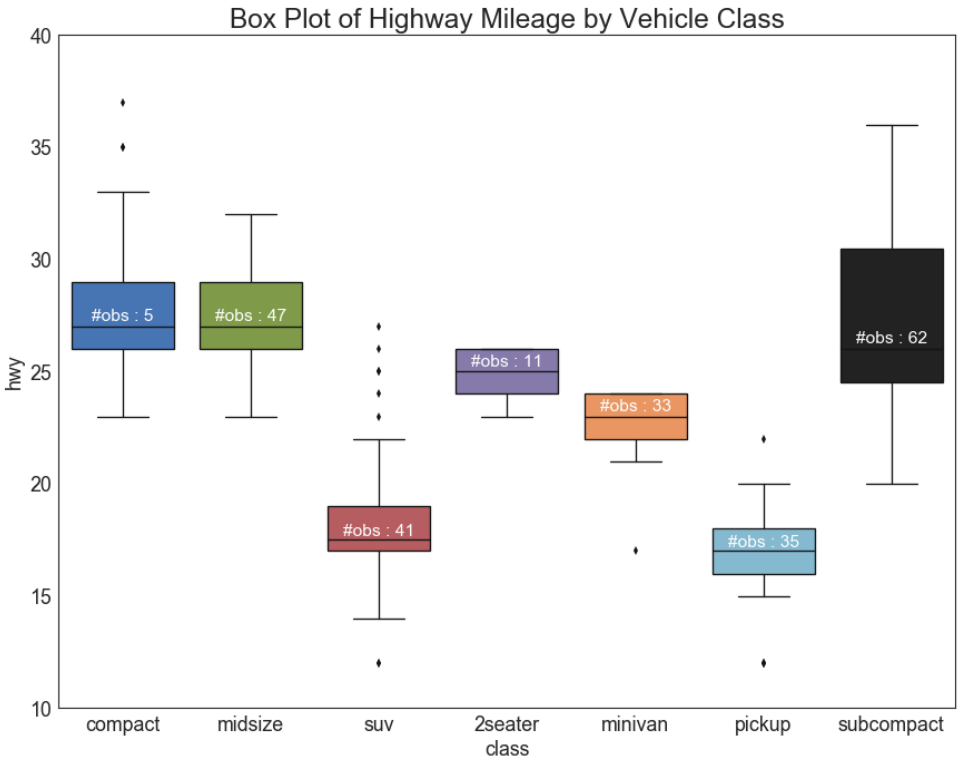


Figure 26.28: 箱形图

26.6.8 包点 + 箱形图（Dot + Box Plot）

包点 + 箱形图（Dot + Box Plot）传达类似于分组的箱形图信息。此外，这些点可以了解每组中有多少数据点。

```
1 # Import Data
2 df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.
      csv")
3
4 # Draw Plot
5 plt.figure(figsize=(13,10), dpi= 80)
6 sns.boxplot(x='class', y='hwy', data=df, hue='cyl')
7 sns.stripplot(x='class', y='hwy', data=df, color='black', size=3, jitter=1)
8
9 for i in range(len(df['class'].unique())-1):
```

```
    plt.vlines(i+.5, 10, 45, linestyles='solid', colors='gray', alpha=0.2)

# Decoration
plt.title('Box Plot of Highway Mileage by Vehicle Class', fontsize=22)
plt.legend(title='Cylinders')
plt.show()
```

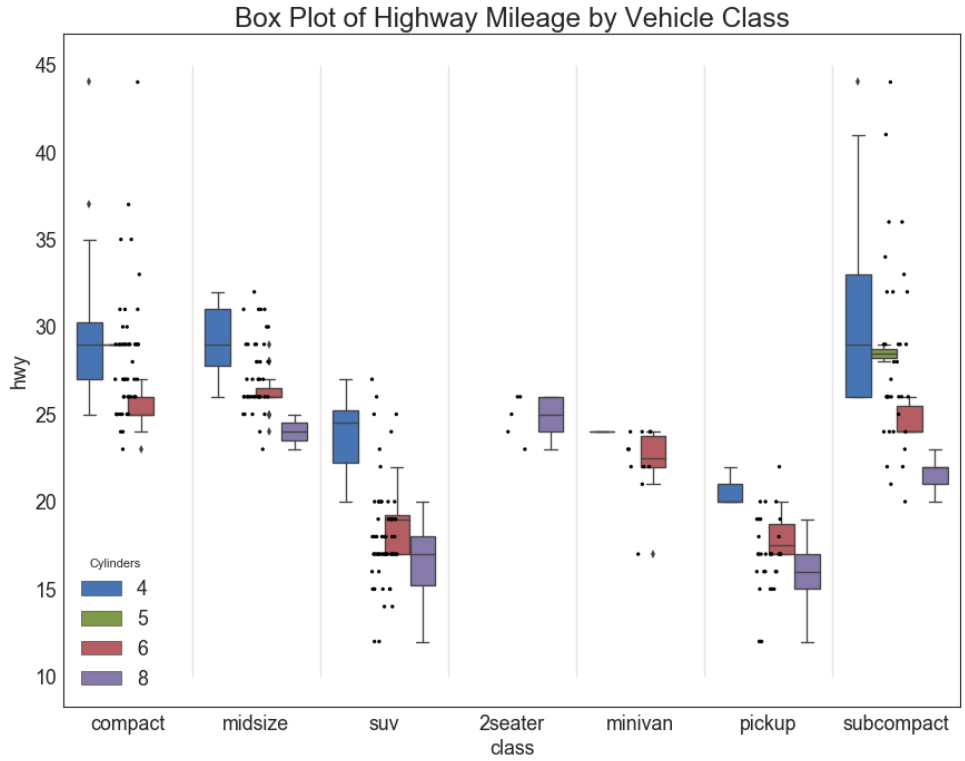


Figure 26.29: 包点 + 箱形图

26.6.9 小提琴图（Violin Plot）

小提琴图是箱形图在视觉上令人愉悦的替代品。小提琴的形状或面积取决于它所持有的观察次数。但是，小提琴图可能更难以阅读，并且在专业设置中不常用。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(13,10), dpi= 80)
sns.violinplot(x='class', y='hwy', data=df, scale='width', inner='quartile')

# Decoration
plt.title('Violin Plot of Highway Mileage by Vehicle Class', fontsize=22)
```

```
plt.show()
```

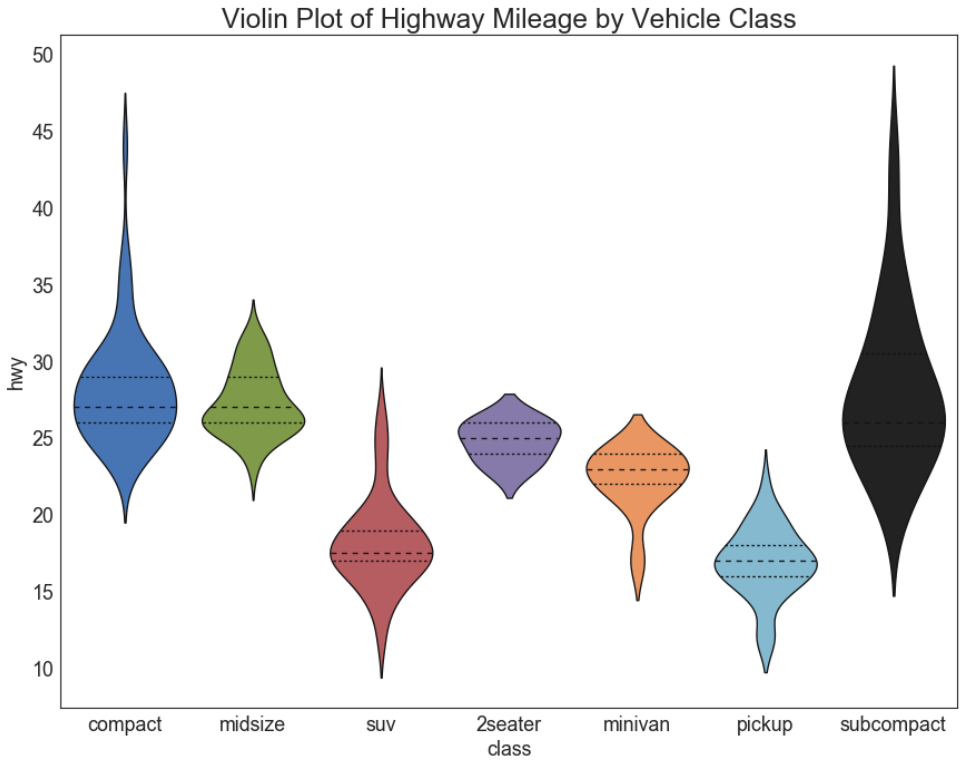


Figure 26.30: 小提琴图

26.6.10 人口金字塔（Population Pyramid）

人口金字塔可用于显示由数量排序的组的分布。或者它也可以用于显示人口的逐级过滤，因为它在下面用于显示有多少人通过营销渠道的每个阶段。

```
# Read data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/email_campaign_funnel.csv")

# Draw Plot
plt.figure(figsize=(13,10), dpi= 80)
group_col = 'Gender'
order_of_bars = df.Stage.unique()[::-1]
colors = [plt.cm.Spectral(i/float(len(df[group_col].unique())-1)) for i in range(len(df[group_col].unique()))]

for c, group in zip(colors, df[group_col].unique()):
    sns.barplot(x='Users', y='Stage', data=df.loc[df[group_col]==group, :], order=order_of_bars, color=c, label=group)

```

```
# Decorations
plt.xlabel("$Users$")
plt.ylabel("Stage of Purchase")
plt.yticks(fontsize=12)
plt.title("Population Pyramid of the Marketing Funnel", fontsize=22)
plt.legend()
plt.show()
```

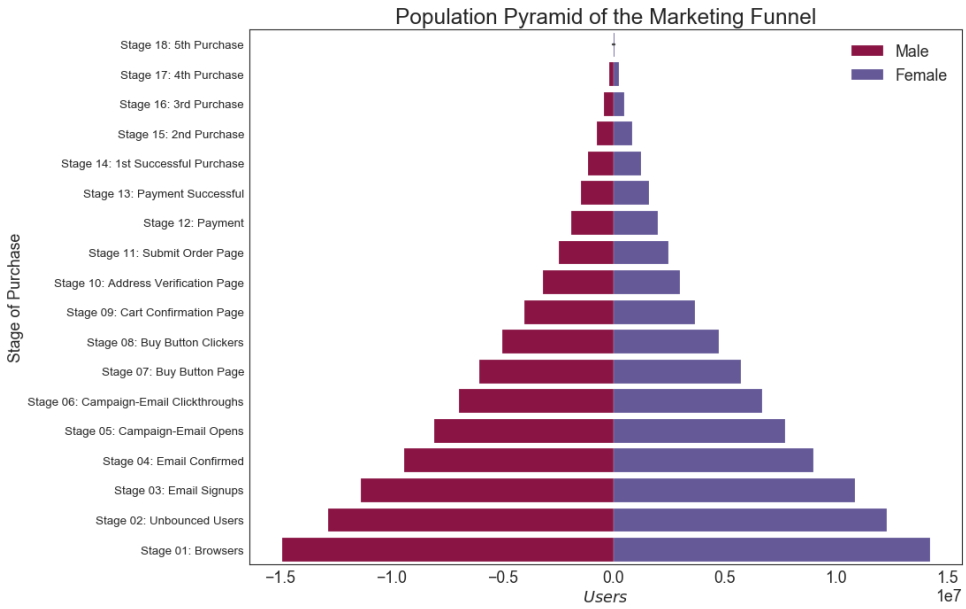


Figure 26.31: 人口金字塔

26.6.11 分类图（Categorical Plots）

由 seaborn 库提供的分类图可用于可视化彼此相关的 2 个或更多分类变量的计数分布。

```
# Load Dataset
titanic = sns.load_dataset("titanic")

# Plot
g = sns.catplot("alive", col="deck", col_wrap=4,
                data=titanic[titanic.deck.notnull()],
                kind="count", height=3.5, aspect=.8,
                palette='tab20')

fig.suptitle('sf')
plt.show()
```

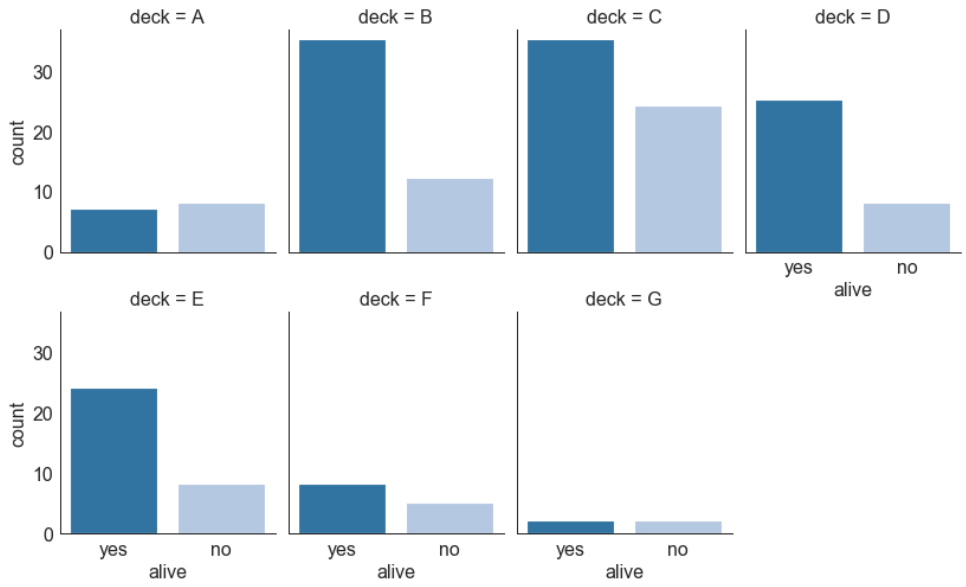


Figure 26.32: 分类图

```
# Load Dataset
titanic = sns.load_dataset("titanic")

# Plot
sns.catplot(x="age", y="embark_town",
            hue="sex", col="class",
            data=titanic[titanic.embark_town.notnull()],
            orient="h", height=5, aspect=1, palette="tab10",
            kind="violin", dodge=True, cut=0, bw=.2)
```

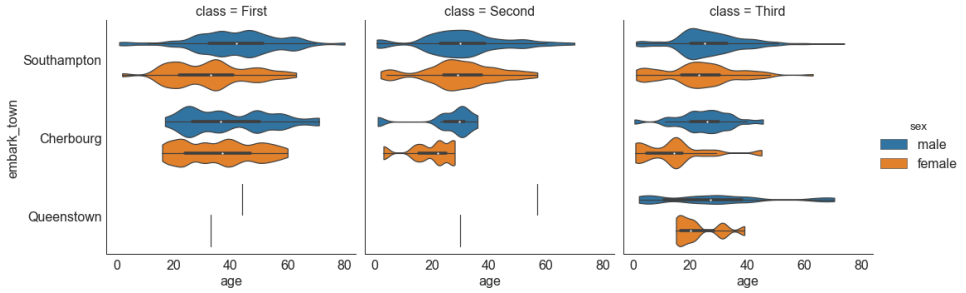


Figure 26.33: 分类图

26.7 组成（Composition）

26.7.1 华夫饼图（Waffle Chart）

可以使用 pywaffle 包创建华夫饼图，并用于显示更大群体中的组的组成。（『Python 数据之道』注：需要安装 pywaffle 库）

```
#! pip install pywaffle
# Reference: https://stackoverflow.com/questions/41400136/how-to-do-waffle-charts-in-python-square-piechart
from pywaffle import Waffle

# Import
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Prepare Data
df = df_raw.groupby('class').size().reset_index(name='counts')
n_categories = df.shape[0]
colors = [plt.cm.inferno_r(i/float(n_categories)) for i in range(n_categories)]

# Draw Plot and Decorate
fig = plt.figure(
    FigureClass=Waffle,
    plots={
        '111': {
            'values': df['counts'],
            'labels': ["{0} ({1})".format(n[0], n[1]) for n in df[['class', 'counts']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 12},
            'title': {'label': '# Vehicles by Class', 'loc': 'center', 'fontsize':18}
        },
    },
    rows=7,
    colors=colors,
    figsize=(16, 9)
)
```

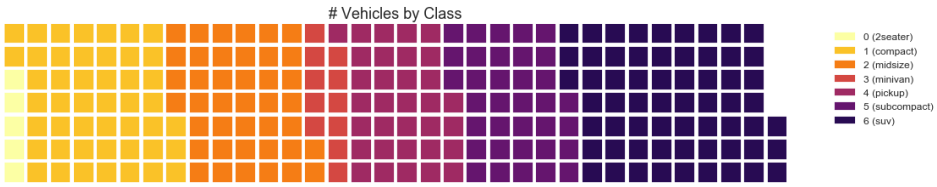


Figure 26.34: 华夫饼图

```
#! pip install pywaffle
from pywaffle import Waffle

# Import
# df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Prepare Data
# By Class Data
df_class = df_raw.groupby('class').size().reset_index(name='counts_class')
n_categories = df_class.shape[0]
colors_class = [plt.cm.Set3(i/float(n_categories)) for i in range(n_categories)]

# By Cylinders Data
df_cyl = df_raw.groupby('cyl').size().reset_index(name='counts_cyl')
n_categories = df_cyl.shape[0]
colors_cyl = [plt.cm.Spectral(i/float(n_categories)) for i in range(n_categories)]

# By Make Data
df_make = df_raw.groupby('manufacturer').size().reset_index(name='counts_make')
n_categories = df_make.shape[0]
colors_make = [plt.cm.tab20b(i/float(n_categories)) for i in range(n_categories)]


# Draw Plot and Decorate
fig = plt.figure(
    FigureClass=Waffle,
    plots={
        '311': {
            'values': df_class['counts_class'],
            'labels': ["{1}".format(n[0], n[1]) for n in df_class[['class', 'counts_class']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 12, 'title':'Class'},
            'title': {'label': '# Vehicles by Class', 'loc': 'center', 'fontsize':18},
            'colors': colors_class
        },
        '312': {
            'values': df_cyl['counts_cyl'],
            'labels': ["{1}".format(n[0], n[1]) for n in df_cyl[['cyl', 'counts_cyl']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 12, 'title':'Cyl'},
            'title': {'label': '# Vehicles by Cyl', 'loc': 'center', 'fontsize':18},
            'colors': colors_cyl
```

```
        },
        '313': {
            'values': df_make['counts_make'],
            'labels': ["{1}".format(n[0], n[1]) for n in df_make[['manufacturer
                ', 'counts_make']].itertuples()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), '
                fontsize': 12, 'title':'Manufacturer'},
            'title': {'label': '# Vehicles by Make', 'loc': 'center', 'fontsize
                ':18},
            'colors': colors_make
        }
    },
    rows=9,
    figsize=(16, 14)
)
```

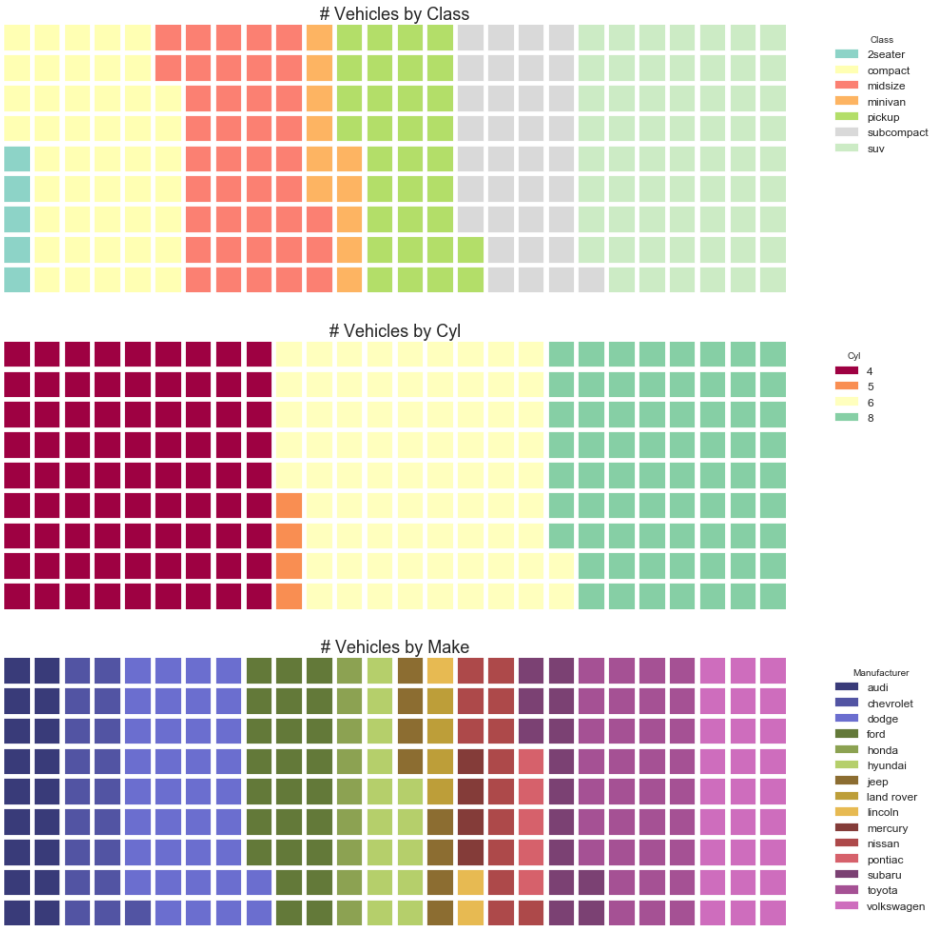


Figure 26.35: 华夫饼图

26.7.2 饼图（Pie Chart）

饼图是显示组成的经典方式。然而，现在通常不建议使用它，因为馅饼部分的面积有时会变得误导。因此，如果您要使用饼图，强烈建议明确记下饼图每个部分的百分比或数字。

```
# Import
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
    mpg_ggplot2.csv")

# Prepare Data
df = df_raw.groupby('class').size()

# Make the plot with pandas
df.plot(kind='pie', subplots=True, figsize=(8, 8))
plt.title("Pie Chart of Vehicle Class - Bad")
plt.ylabel("")
plt.show()
```

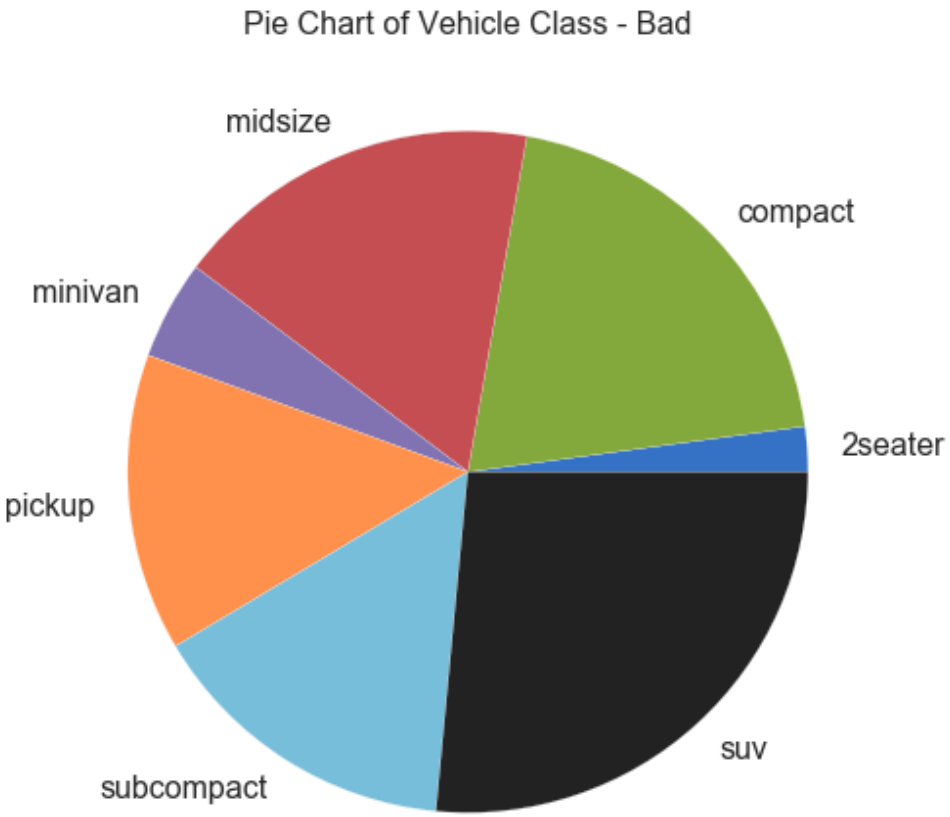


Figure 26.36: 饼图

```
# Import
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
    mpg_ggplot2.csv")

# Prepare Data
df = df_raw.groupby('class').size().reset_index(name='counts')

# Draw Plot
fig, ax = plt.subplots(figsize=(12, 7), subplot_kw=dict(aspect="equal"), dpi=
    80)

data = df['counts']
categories = df['class']
explode = [0,0,0,0,0,0.1,0]

def func(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}% ({:d} )".format(pct, absolute)

wedges, texts, autotexts = ax.pie(data,
                                  autopct=lambda pct: func(pct, data),
                                  textprops=dict(color="w"),
                                  colors=plt.cm.Dark2.colors,
                                 startangle=140,
                                 explode=explode)

# Decoration
ax.legend(wedges, categories, title="Vehicle Class", loc="center left",
    bbox_to_anchor=(1, 0, 0.5, 1))
plt.setp(autotexts, size=10, weight=700)
ax.set_title("Class of Vehicles: Pie Chart")
plt.show()
```

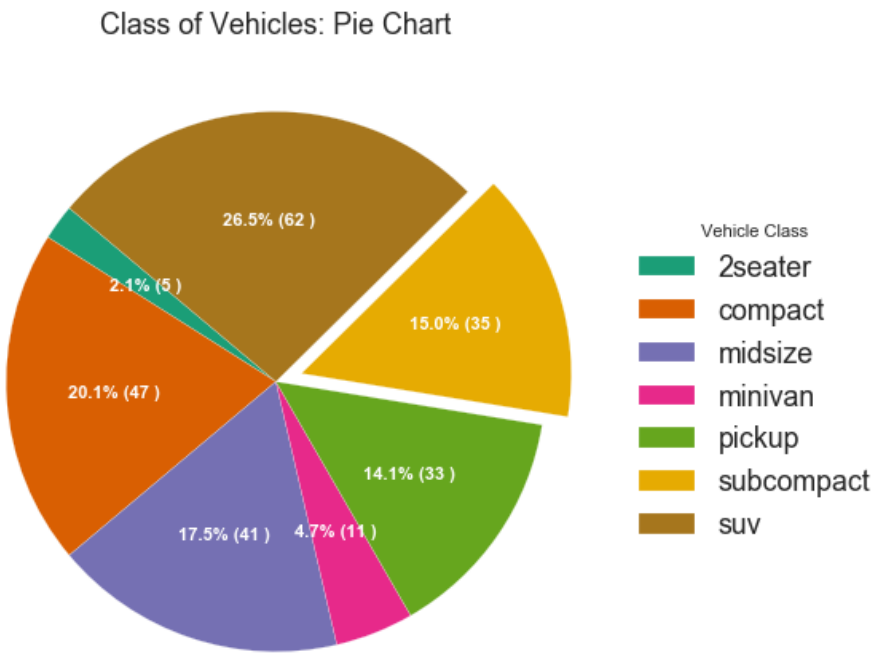


Figure 26.37: 饼图

26.7.3 树形图（Treemap）

树形图类似于饼图，它可以更好地完成工作而不会误导每个组的贡献。（『Python 数据之道』注：需要安装 squarify 库）

```
# pip install squarify
import squarify

# Import Data
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
    mpg_ggplot2.csv")

# Prepare Data
df = df_raw.groupby('class').size().reset_index(name='counts')
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels))]

# Draw Plot
plt.figure(figsize=(12,8), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8)

# Decorate
```

```
plt.title('Treemap of Vechile Class')
plt.axis('off')
plt.show()
```

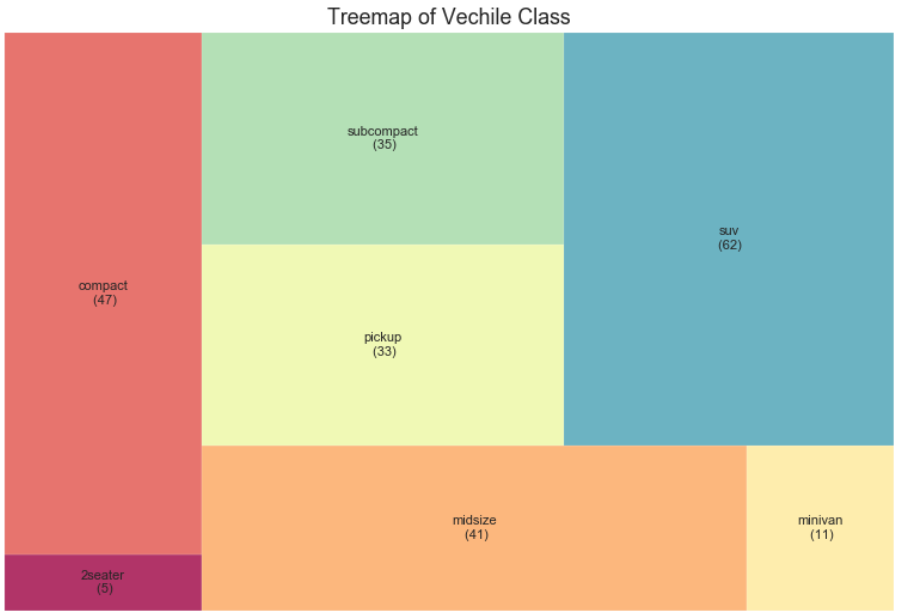


Figure 26.38: 树形图

26.7.4 条形图（Bar Chart）

条形图是基于计数或任何给定指标可视化项目的经典方式。在下面的图表中，我为每个项目使用了不同的颜色，但您通常可能希望为所有项目选择一种颜色，除非您按组对其进行着色。颜色名称存储在下面代码中的 all_colors 中。您可以通过在`plt.plot ()` 中设置颜色参数来更改条的颜色。

```
import random

# Import Data
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/
    mpg_ggplot2.csv")

# Prepare Data
df = df_raw.groupby('manufacturer').size().reset_index(name='counts')
n = df['manufacturer'].unique().__len__()+1
all_colors = list(plt.cm.colors.cnames.keys())
random.seed(100)
c = random.choices(all_colors, k=n)

# Plot Bars
plt.figure(figsize=(16,10), dpi= 80)
```

```
plt.bar(df['manufacturer'], df['counts'], color=c, width=.5)
for i, val in enumerate(df['counts'].values):
    plt.text(i, val, float(val), horizontalalignment='center',
        verticalalignment='bottom', fontdict={'fontweight':500, 'size':12})

# Decoration
plt.gca().set_xticklabels(df['manufacturer'], rotation=60, horizontalalignment=
    'right')
plt.title("Number of Vehicles by Manaufacturers", fontsize=22)
plt.ylabel('# Vehicles')
plt.ylim(0, 45)
plt.show()
```

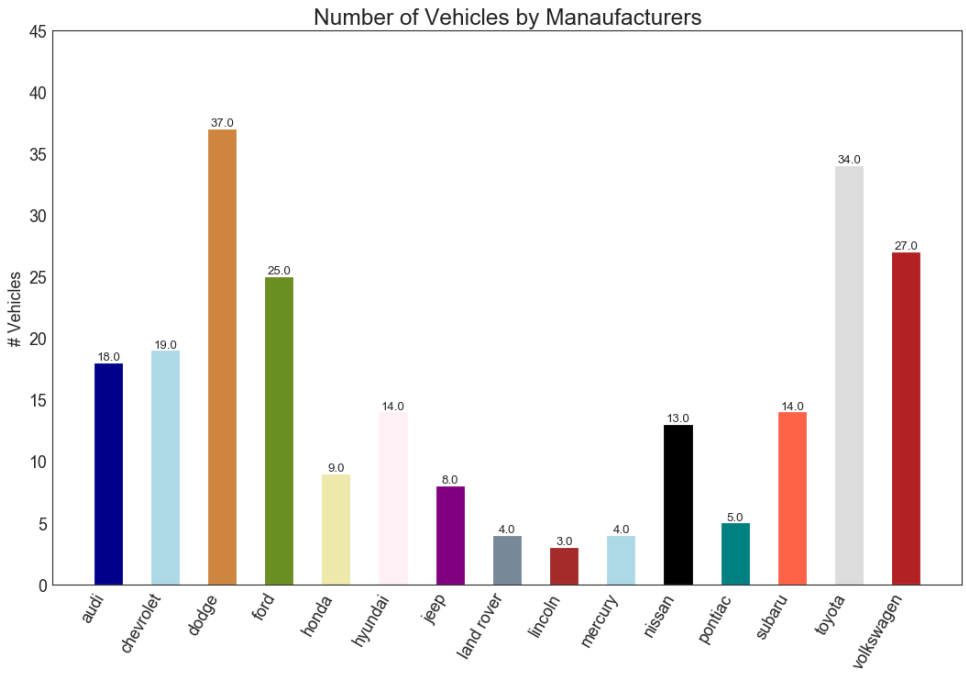


Figure 26.39: 条形图

26.8 变化（Change）

26.8.1 时间序列图（Time Series Plot）

时间序列图用于显示给定度量随时间变化的方式。在这里，您可以看到 1949 年至 1969 年间航空客运量的变化情况。

```
# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/AirPassengers.
    csv')

```

```
# Draw Plot
plt.figure(figsize=(16,10), dpi= 80)
plt.plot('date', 'traffic', data=df, color='tab:red')

# Decoration
plt.ylim(50, 750)
xtick_location = df.index.tolist()[::12]
xtick_labels = [x[-4:] for x in df.date.tolist()[::12]]
plt.xticks(ticks=xtick_location, labels=xtick_labels, rotation=0, fontsize=12,
    horizontalalignment='center', alpha=.7)
plt.yticks(fontsize=12, alpha=.7)
plt.title("Air Passengers Traffic (1949 - 1969)", fontsize=22)
plt.grid(axis='both', alpha=.3)

# Remove borders
plt.gca().spines["top"].set_alpha(0.0)
plt.gca().spines["bottom"].set_alpha(0.3)
plt.gca().spines["right"].set_alpha(0.0)
plt.gca().spines["left"].set_alpha(0.3)
plt.show()
```

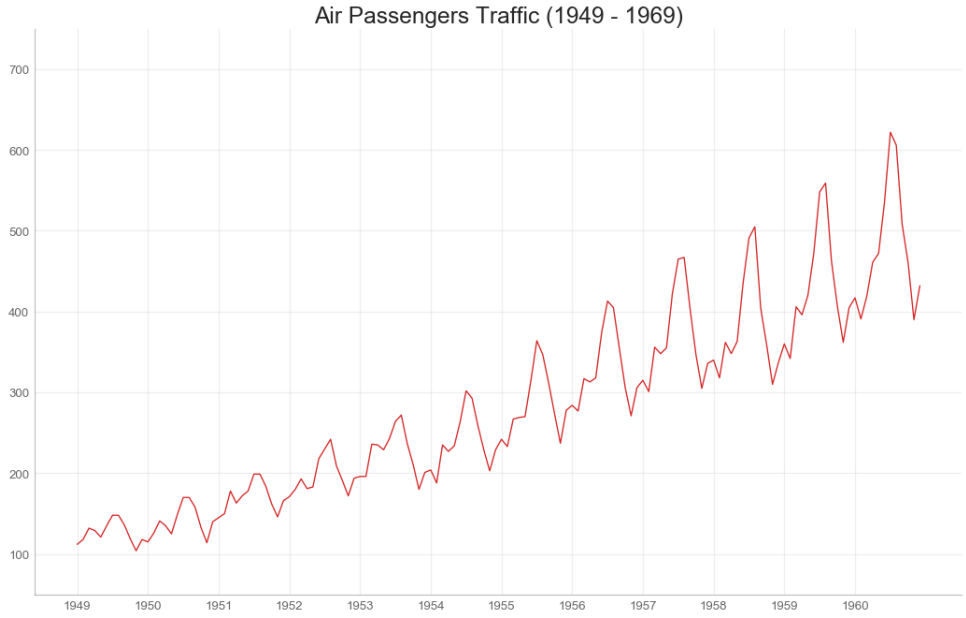


Figure 26.40: 时间序列图

26.8.2 带波峰波谷标记的时序图（Time Series with Peaks and Troughs Annotated）

下面的时间序列绘制了所有峰值和低谷，并注释了所选特殊事件的发生。

```
# Import Data
```

```
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/AirPassengers.csv')

# Get the Peaks and Troughs
data = df['traffic'].values
doublediff = np.diff(np.sign(np.diff(data)))
peak_locations = np.where(doublediff == -2)[0] + 1

doublediff2 = np.diff(np.sign(np.diff(-1*data)))
trough_locations = np.where(doublediff2 == -2)[0] + 1

# Draw Plot
plt.figure(figsize=(16,10), dpi= 80)
plt.plot('date', 'traffic', data=df, color='tab:blue', label='Air Traffic')
plt.scatter(df.date[peak_locations], df.traffic[peak_locations], marker=mpl.markers.CARETUPBASE, color='tab:green', s=100, label='Peaks')
plt.scatter(df.date[trough_locations], df.traffic[trough_locations], marker=mpl.markers.CARETDOWNBASE, color='tab:red', s=100, label='Troughs')

# Annotate
for t, p in zip(trough_locations[1::5], peak_locations[::3]):
    plt.text(df.date[p], df.traffic[p]+15, df.date[p], horizontalalignment='center', color='darkgreen')
    plt.text(df.date[t], df.traffic[t]-35, df.date[t], horizontalalignment='center', color='darkred')

# Decoration
plt.ylim(50,750)
xtick_location = df.index.tolist()[::6]
xtick_labels = df.date.tolist()[::6]
plt.xticks(ticks=xtick_location, labels=xtick_labels, rotation=90, fontsize=12, alpha=.7)
plt.title("Peak and Troughs of Air Passengers Traffic (1949 - 1969)", fontsize=22)
plt.yticks(fontsize=12, alpha=.7)

# Lighten borders
plt.gca().spines["top"].set_alpha(.0)
plt.gca().spines["bottom"].set_alpha(.3)
plt.gca().spines["right"].set_alpha(.0)
plt.gca().spines["left"].set_alpha(.3)

plt.legend(loc='upper left')
plt.grid(axis='y', alpha=.3)
plt.show()
```

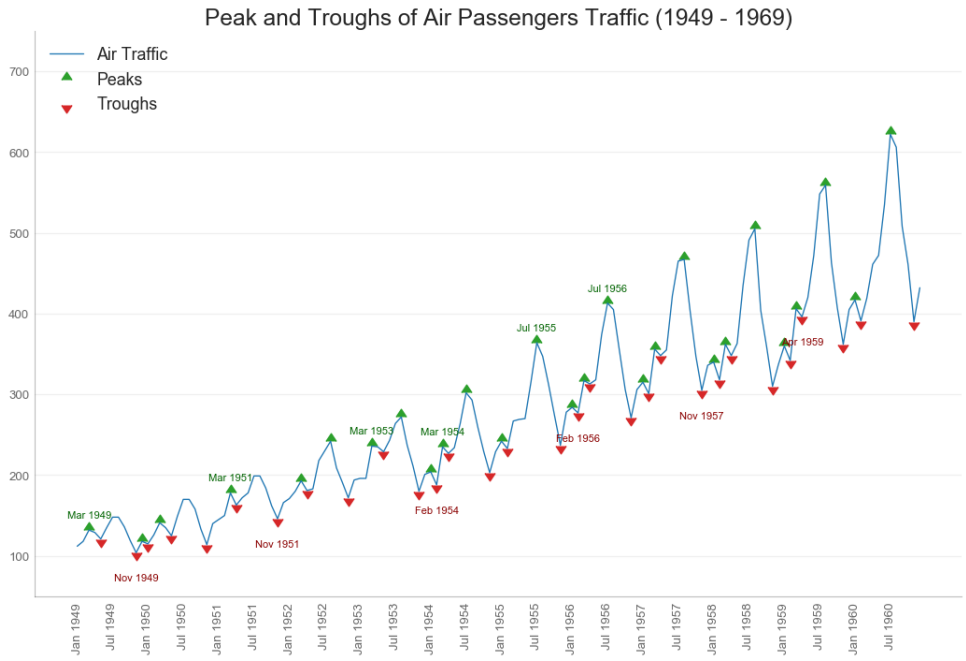


Figure 26.41: 带波峰波谷标记的时序图

26.8.3 自相关和部分自相关图（Autocorrelation (ACF) and Partial Autocorrelation (PACF) Plot）

自相关图（ACF 图）显示时间序列与其自身滞后的相关性。每条垂直线（在自相关图上）表示系列与滞后 0 之间的滞后之间的相关性。图中的蓝色阴影区域是显着性水平。那些位于蓝线之上的滞后是显着的滞后。

那么如何解读呢？对于空乘旅客，我们看到多达 14 个滞后跨越蓝线，因此非常重要。这意味着，14 年前的航空旅客交通量对今天的交通状况有影响。PACF 在另一方面显示了任何给定滞后（时间序列）与当前序列的自相关，但是删除了滞后的贡献。

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/AirPassengers.
    csv')

# Draw Plot
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(16,6), dpi= 80)
plot_acf(df.traffic.tolist(), ax=ax1, lags=50)
plot_pacf(df.traffic.tolist(), ax=ax2, lags=20)

# Decorate
# lighten the borders
ax1.spines["top"].set_alpha(.3); ax2.spines["top"].set_alpha(.3)
ax1.spines["bottom"].set_alpha(.3); ax2.spines["bottom"].set_alpha(.3)
```

```
ax1.spines["right"].set_alpha(.3); ax2.spines["right"].set_alpha(.3)
ax1.spines["left"].set_alpha(.3); ax2.spines["left"].set_alpha(.3)

# font size of tick labels
ax1.tick_params(axis='both', labelsize=12)
ax2.tick_params(axis='both', labelsize=12)
plt.show()
```

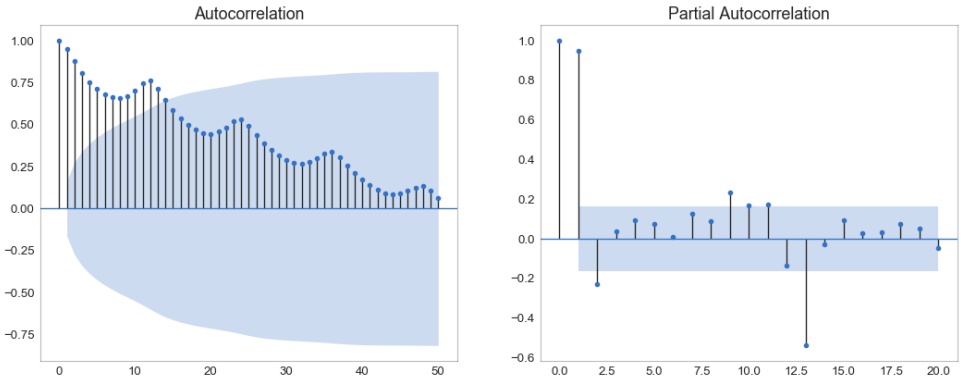


Figure 26.42: 自相关和部分自相关图

26.8.4 交叉相关图（Cross Correlation plot）

交叉相关图显示了两个时间序列相互之间的滞后。

```
import statsmodels.tsa.stattools as stattools

# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/mortality.csv'
    )
x = df['mdeaths']
y = df['fdeaths']

# Compute Cross Correlations
ccs = stattools.ccf(x, y)[:100]
nlags = len(ccs)

# Compute the Significance level
# ref: https://stats.stackexchange.com/questions/3115/cross-correlation-
    significance-in-r/3128#3128
conf_level = 2 / np.sqrt(nlags)

# Draw Plot
plt.figure(figsize=(12,7), dpi= 80)

plt.hlines(0, xmin=0, xmax=100, color='gray')  # 0 axis
plt.hlines(conf_level, xmin=0, xmax=100, color='gray')
```

```
plt.hlines(-conf_level, xmin=0, xmax=100, color='gray')

plt.bar(x=np.arange(len(ccs)), height=ccs, width=.3)

# Decoration
plt.title('$Cross\; Correlation\; Plot:\; mdeaths\; vs\; fdeaths$', fontsize
    =22)
plt.xlim(0,len(ccs))
plt.show()
```

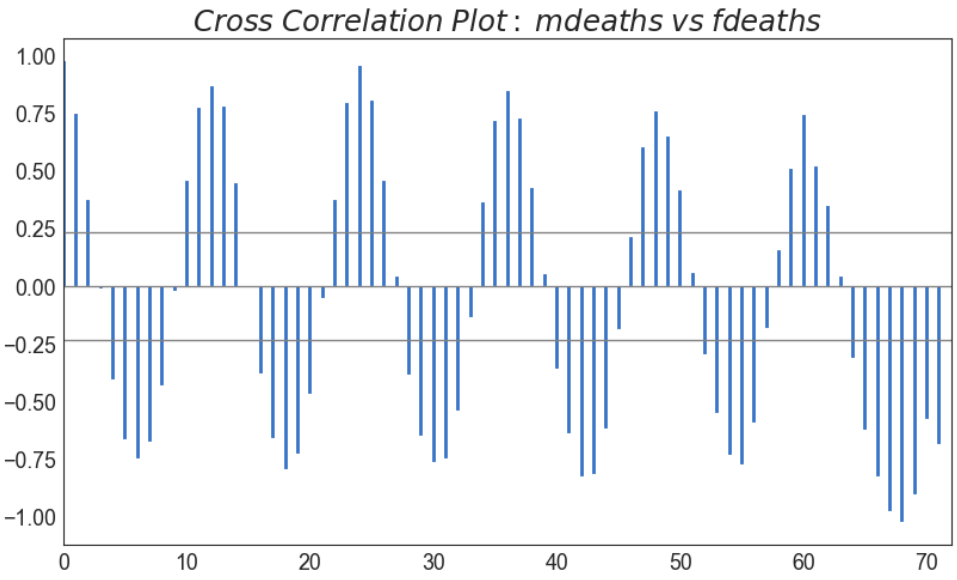


Figure 26.43: 交叉相关图

26.8.5 时间序列分解图（Time Series Decomposition Plot）

时间序列分解图显示时间序列分解为趋势，季节和残差分量。

```
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse

# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/AirPassengers.
    csv')
dates = pd.DatetimeIndex([parse(d).strftime('%Y-%m-01') for d in df['date']])
df.set_index(dates, inplace=True)

# Decompose
result = seasonal_decompose(df['traffic'], model='multiplicative')

# Plot
plt.rcParams.update({'figure.figsize': (10,10)})
```

```
result.plot().suptitle('Time Series Decomposition of Air Passengers')
plt.show()
```

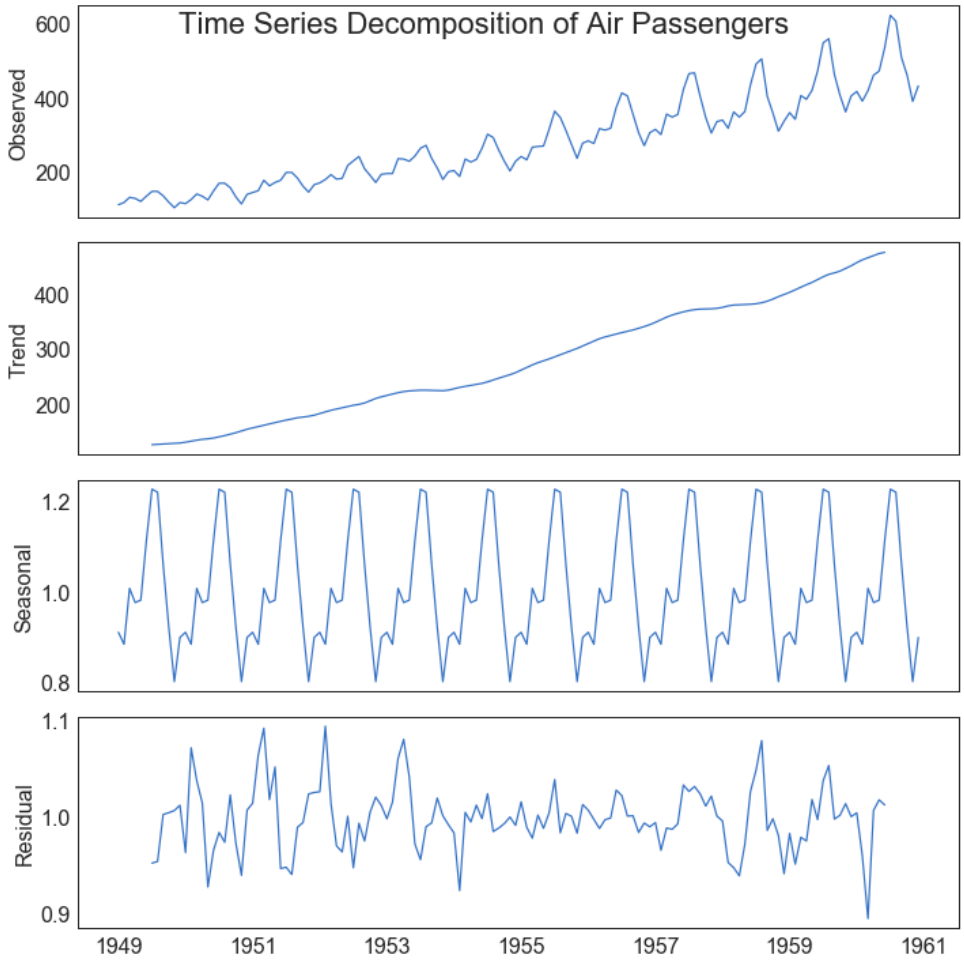


Figure 26.44: 时间序列分解图

26.8.6 多个时间序列（Multiple Time Series）

您可以绘制多个时间序列，在同一图表上测量相同的值，如下所示。

```
# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/mortality.csv'
    )

# Define the upper limit, lower limit, interval of Y axis and colors
y_LL = 100
y_UL = int(df.iloc[:, 1:].max().max()*1.1)
y_interval = 400
mycolors = ['tab:red', 'tab:blue', 'tab:green', 'tab:orange']
```

```
9
10 # Draw Plot and Annotate
11 fig, ax = plt.subplots(1,1,figsize=(16, 9), dpi= 80)
12
13 columns = df.columns[1:]
14 for i, column in enumerate(columns):
15     # 原文此处有误，Python数据之道 备注
16     # 访问 liyangbit.com ， 查看本文完整内容
17     plt.plot(df.date.values, df[column].values, lw=1.5, color=mycolors[i])
18     plt.text(df.shape[0]+1, df[column].values[-1], column, fontsize=14, color=
           mycolors[i])
19
20 # Draw Tick lines
21 for y in range(y_LL, y_UL, y_interval):
22     plt.hlines(y, xmin=0, xmax=71, colors='black', alpha=0.3, linestyles="--",
           lw=0.5)
23
24 # Decorations
25 plt.tick_params(axis="both", which="both", bottom=False, top=False,
26                 labelbottom=True, left=False, right=False, labelleft=True)
27
28 # Lighten borders
29 plt.gca().spines["top"].set_alpha(.3)
30 plt.gca().spines["bottom"].set_alpha(.3)
31 plt.gca().spines["right"].set_alpha(.3)
32 plt.gca().spines["left"].set_alpha(.3)
33
34 plt.title('Number of Deaths from Lung Diseases in the UK (1974-1979)', fontsize
       =22)
35 plt.yticks(range(y_LL, y_UL, y_interval), [str(y) for y in range(y_LL, y_UL,
       y_interval)], fontsize=12)
36 plt.xticks(range(0, df.shape[0], 12), df.date.values[::12], horizontalalignment
       ='left', fontsize=12)
37 plt.ylim(y_LL, y_UL)
38 plt.xlim(-2, 80)
39 plt.show()
```

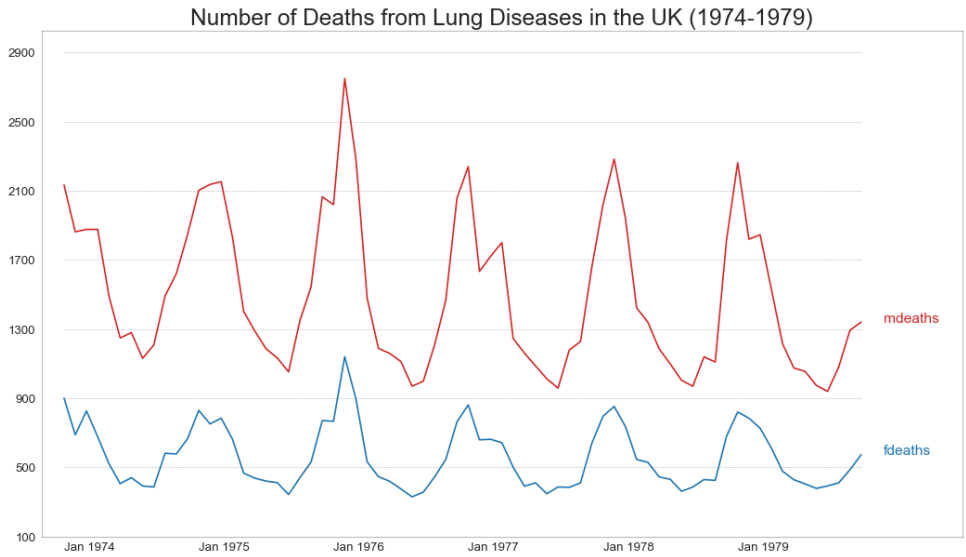


Figure 26.45: 多个时间序列

26.8.7 使用辅助 Y 轴来绘制不同范围的图形（Plotting with different scales using secondary Y axis）

如果要显示在同一时间点测量两个不同数量的两个时间序列，则可以在右侧的辅助 Y 轴上再绘制第二个系列。

```
# Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/economics.csv"
    )

x = df['date']
y1 = df['psavert']
y2 = df['unemploy']

# Plot Line1 (Left Y Axis)
fig, ax1 = plt.subplots(1,1,figsize=(16,9), dpi= 80)
ax1.plot(x, y1, color='tab:red')

# Plot Line2 (Right Y Axis)
ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
ax2.plot(x, y2, color='tab:blue')

# Decorations
# ax1 (left Y axis)
ax1.set_xlabel('Year', fontsize=20)
ax1.tick_params(axis='x', rotation=0, labelsize=12)
ax1.set_ylabel('Personal Savings Rate', color='tab:red', fontsize=20)
ax1.tick_params(axis='y', rotation=0, labelcolor='tab:red' )
ax1.grid(alpha=.4)

```

```
# ax2 (right Y axis)
ax2.set_ylabel("# Unemployed (1000's)", color='tab:blue', fontsize=20)
ax2.tick_params(axis='y', labelcolor='tab:blue')
ax2.set_xticks(np.arange(0, len(x), 60))
ax2.set_xticklabels(x[::60], rotation=90, fontdict={'fontsize':10})
ax2.set_title("Personal Savings Rate vs Unemployed: Plotting in Secondary Y Axis", fontsize=22)
fig.tight_layout()
plt.show()
```

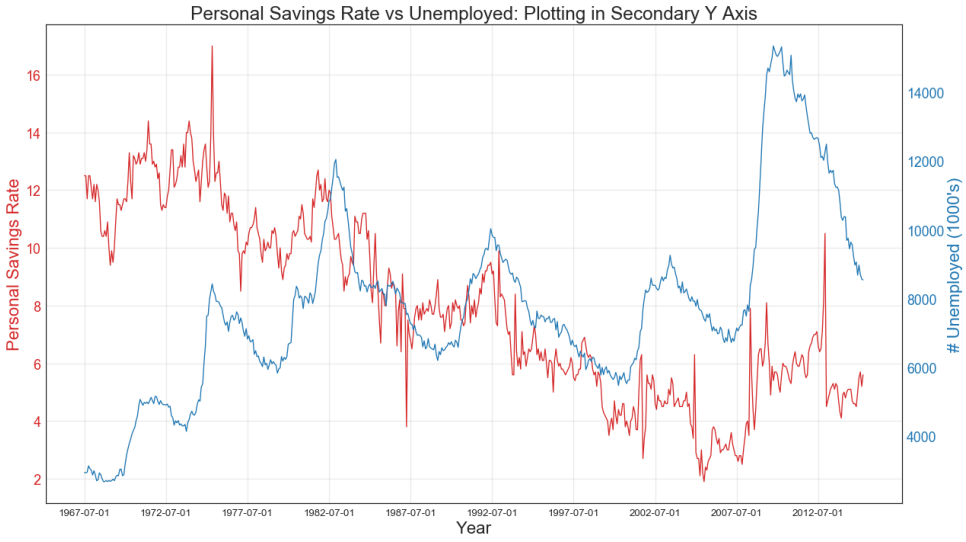


Figure 26.46: 使用辅助 Y 轴来绘制不同范围的图形

26.8.8 带有误差带的时间序列（Time Series with Error Bands）

如果您有一个时间序列数据集，每个时间点（日期/时间戳）有多个观测值，则可以构建带有误差带的时间序列。您可以在下面看到一些基于每天不同时间订单的示例。另一个关于 45 天持续到达的订单数量的例子。

在该方法中，订单数量的平均值由白线表示。并且计算 95%置信区间并围绕均值绘制。

```
from scipy.stats import sem

# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/user_orders_hourofday.csv")
df_mean = df.groupby('order_hour_of_day').quantity.mean()
df_se = df.groupby('order_hour_of_day').quantity.apply(sem).mul(1.96)

# Plot
plt.figure(figsize=(16,10), dpi= 80)
plt.ylabel("# Orders", fontsize=16)
x = df_mean.index
```

```
plt.plot(x, df_mean, color="white", lw=2)
plt.fill_between(x, df_mean - df_se, df_mean + df_se, color="#3F5D7D")

# Decorations
# Lighten borders
plt.gca().spines["top"].set_alpha(0)
plt.gca().spines["bottom"].set_alpha(1)
plt.gca().spines["right"].set_alpha(0)
plt.gca().spines["left"].set_alpha(1)
plt.xticks(x[::2], [str(d) for d in x[::2]] , fontsize=12)
plt.title("User Orders by Hour of Day (95% confidence)", fontsize=22)
plt.xlabel("Hour of Day")

s, e = plt.gca().get_xlim()
plt.xlim(s, e)

# Draw Horizontal Tick lines
for y in range(8, 20, 2):
    plt.hlines(y, xmin=s, xmax=e, colors='black', alpha=0.5, linestyles="--",
        lw=0.5)

plt.show()
```

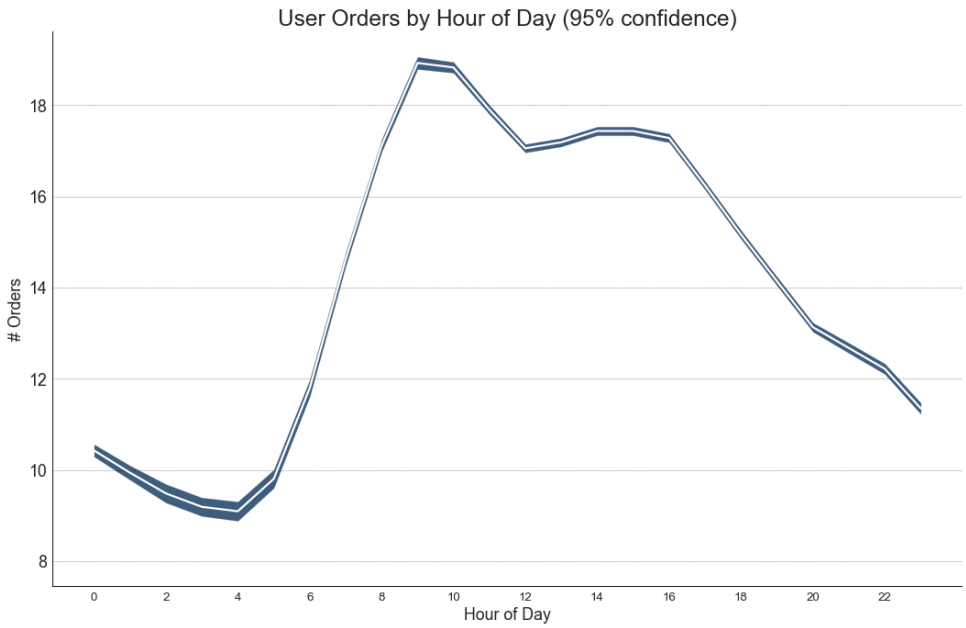


Figure 26.47: 带有误差带的时间序列

```
# "Data Source: https://www.kaggle.com/olistbr/brazilian-ecommerce#
    olist_orders_dataset.csv"
from dateutil.parser import parse
from scipy.stats import sem
```

```
4
5  # Import Data
6  df_raw = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master
       /orders_45d.csv',
7                       parse_dates=['purchase_time', 'purchase_date'])
8
9  # Prepare Data: Daily Mean and SE Bands
10 df_mean = df_raw.groupby('purchase_date').quantity.mean()
11 df_se = df_raw.groupby('purchase_date').quantity.apply(sem).mul(1.96)
12
13 # Plot
14 plt.figure(figsize=(16,10), dpi= 80)
15 plt.ylabel("# Daily Orders", fontsize=16)
16 x = [d.date().strftime('%Y-%m-%d') for d in df_mean.index]
17 plt.plot(x, df_mean, color="white", lw=2)
18 plt.fill_between(x, df_mean - df_se, df_mean + df_se, color="#3F5D7D")
19
20 # Decorations
21 # Lighten borders
22 plt.gca().spines["top"].set_alpha(0)
23 plt.gca().spines["bottom"].set_alpha(1)
24 plt.gca().spines["right"].set_alpha(0)
25 plt.gca().spines["left"].set_alpha(1)
26 plt.xticks(x[::6], [str(d) for d in x[::6]] , fontsize=12)
27 plt.title("Daily Order Quantity of Brazilian Retail with Error Bands (95%
       confidence)", fontsize=20)
28
29 # Axis limits
30 s, e = plt.gca().get_xlim()
31 plt.xlim(s, e-2)
32 plt.ylim(4, 10)
33
34 # Draw Horizontal Tick lines
35 for y in range(5, 10, 1):
36     plt.hlines(y, xmin=s, xmax=e, colors='black', alpha=0.5, linestyles="--",
           lw=0.5)
37
38 plt.show()
```

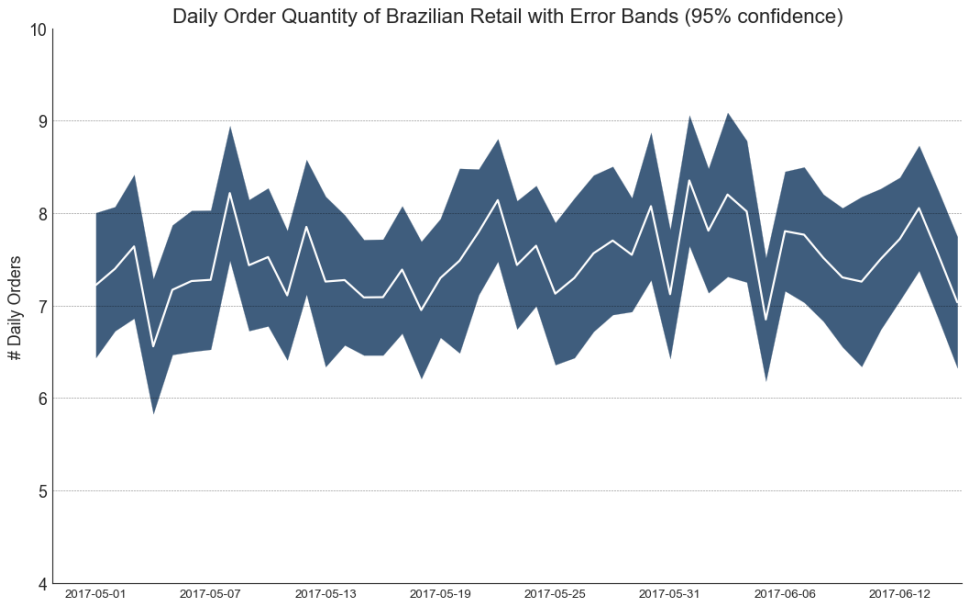


Figure 26.48: 带有误差带的时间序列

26.8.9 堆积面积图（Stacked Area Chart）

堆积面积图可以直观地显示多个时间序列的贡献程度，因此很容易相互比较。

```
# Import Data
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/nightvisitors.csv')

# Decide Colors
mycolors = ['tab:red', 'tab:blue', 'tab:green', 'tab:orange', 'tab:brown', 'tab:grey', 'tab:pink', 'tab:olive']

# Draw Plot and Annotate
fig, ax = plt.subplots(1,1,figsize=(16, 9), dpi= 80)
columns = df.columns[1:]
labs = columns.values.tolist()

# Prepare data
x  = df['yearmon'].values.tolist()
y0 = df[columns[0]].values.tolist()
y1 = df[columns[1]].values.tolist()
y2 = df[columns[2]].values.tolist()
y3 = df[columns[3]].values.tolist()
y4 = df[columns[4]].values.tolist()
y5 = df[columns[5]].values.tolist()
y6 = df[columns[6]].values.tolist()
y7 = df[columns[7]].values.tolist()
y = np.vstack([y0, y2, y4, y6, y7, y5, y1, y3])
```

```
23
24  # Plot for each column
25  labs = columns.values.tolist()
26  ax = plt.gca()
27  ax.stackplot(x, y, labels=labs, colors=mycolors, alpha=0.8)
28
29  # Decorations
30  ax.set_title('Night Visitors in Australian Regions', fontsize=18)
31  ax.set(ylim=[0, 100000])
32  ax.legend(fontsize=10, ncol=4)
33  plt.xticks(x[::5], fontsize=10, horizontalalignment='center')
34  plt.yticks(np.arange(10000, 100000, 20000), fontsize=10)
35  plt.xlim(x[0], x[-1])
36
37  # Lighten borders
38  plt.gca().spines["top"].set_alpha(0)
39  plt.gca().spines["bottom"].set_alpha(.3)
40  plt.gca().spines["right"].set_alpha(0)
41  plt.gca().spines["left"].set_alpha(.3)
42
43  plt.show()
```

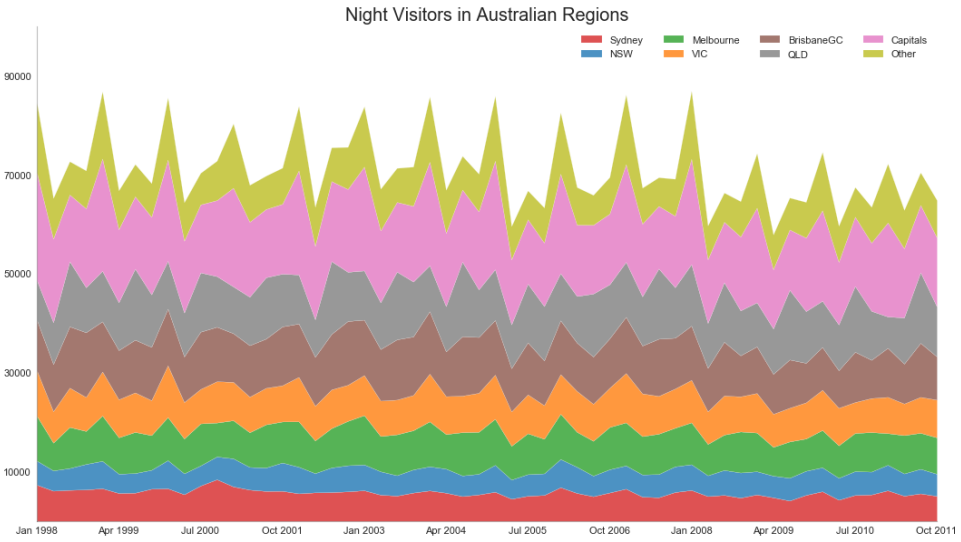


Figure 26.49: 堆积面积图

26.8.10 未堆积的面积图（Area Chart UnStacked）

未堆积面积图用于可视化两个或更多个系列相对于彼此的进度（起伏）。在下面的图表中，您可以清楚地看到随着失业中位数持续时间的增加，个人储蓄率会下降。未堆积面积图表很好地展示了这种现象。

```
1  # Import Data
```

```
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/economics.csv")

# Prepare Data
x = df['date'].values.tolist()
y1 = df['psavert'].values.tolist()
y2 = df['uempmed'].values.tolist()
mycolors = ['tab:red', 'tab:blue', 'tab:green', 'tab:orange', 'tab:brown', 'tab:grey', 'tab:pink', 'tab:olive']
columns = ['psavert', 'uempmed']

# Draw Plot
fig, ax = plt.subplots(1, 1, figsize=(16,9), dpi= 80)
ax.fill_between(x, y1=y1, y2=0, label=columns[1], alpha=0.5, color=mycolors[1], linewidth=2)
ax.fill_between(x, y1=y2, y2=0, label=columns[0], alpha=0.5, color=mycolors[0], linewidth=2)

# Decorations
ax.set_title('Personal Savings Rate vs Median Duration of Unemployment', fontsize=18)
ax.set(ylim=[0, 30])
ax.legend(loc='best', fontsize=12)
plt.xticks(x[::50], fontsize=10, horizontalalignment='center')
plt.yticks(np.arange(2.5, 30.0, 2.5), fontsize=10)
plt.xlim(-10, x[-1])

# Draw Tick lines
for y in np.arange(2.5, 30.0, 2.5):
    plt.hlines(y, xmin=0, xmax=len(x), colors='black', alpha=0.3, linestyles="--", lw=0.5)

# Lighten borders
plt.gca().spines["top"].set_alpha(0)
plt.gca().spines["bottom"].set_alpha(.3)
plt.gca().spines["right"].set_alpha(0)
plt.gca().spines["left"].set_alpha(.3)
plt.show()
```

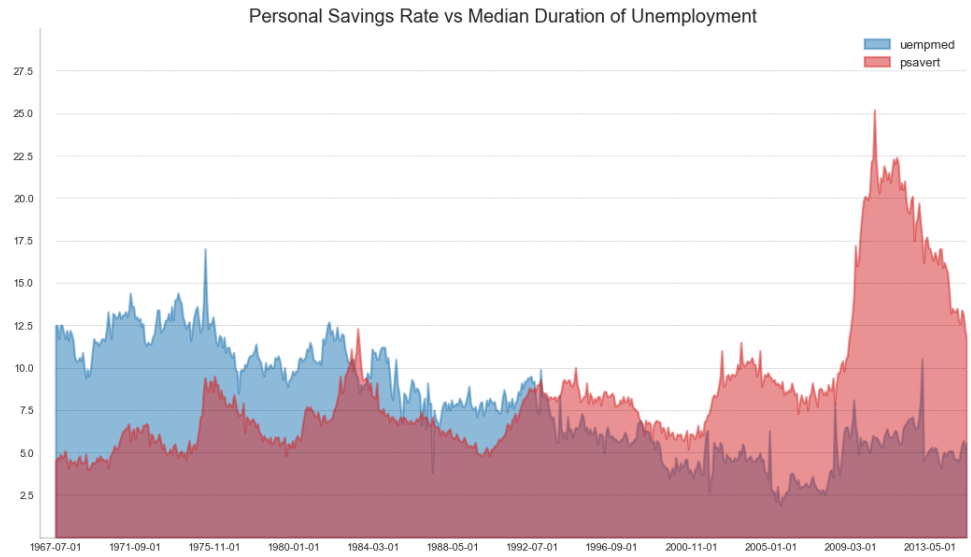


Figure 26.50: 未堆积的面积图

26.8.11 日历热力图（Calendar Heat Map）

与时间序列相比，日历地图是可视化基于时间的数据的备选和不太优选的选项。虽然可以在视觉上吸引人，但数值并不十分明显。然而，它可以很好地描绘极端值和假日效果。（『Python 数据之道』注：需要安装 calmap 库）

```
import matplotlib as mpl

# pip install calmap
# Python数据之道  备注
import calmap

# Import Data
df = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/master/
    yahoo.csv", parse_dates=['date'])
df.set_index('date', inplace=True)

# Plot
plt.figure(figsize=(16,10), dpi= 80)
calmap.calendarplot(df['2014']['VIX.Close'], fig_kws={'figsize': (16,10)},
    yearlabel_kws={'color':'black', 'fontsize':14}, subplot_kws={'title':'Yahoo
    Stock Prices'})
plt.show()
```

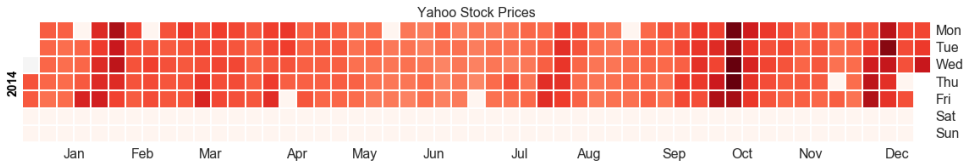


Figure 26.51: 未堆积的面积图

26.8.12 季节图（Seasonal Plot）

季节图可用于比较上一季中同一天（年/月/周等）的时间序列。

```
from dateutil.parser import parse

# Import Data
df = pd.read_csv('https://github.com/selva86/datasets/raw/master/AirPassengers.csv')

# Prepare data
df['year'] = [parse(d).year for d in df.date]
df['month'] = [parse(d).strftime('%b') for d in df.date]
years = df['year'].unique()

# Draw Plot
mycolors = ['tab:red', 'tab:blue', 'tab:green', 'tab:orange', 'tab:brown', 'tab:grey', 'tab:pink', 'tab:olive', 'deeppink', 'steelblue', 'firebrick', 'mediumseagreen']
plt.figure(figsize=(16,10), dpi= 80)

for i, y in enumerate(years):
    plt.plot('month', 'traffic', data=df.loc[df.year==y, :], color=mycolors[i], label=y)
    plt.text(df.loc[df.year==y, :].shape[0]-.9, df.loc[df.year==y, 'traffic'][-1:].values[0], y, fontsize=12, color=mycolors[i])

# Decoration
plt.ylim(50,750)
plt.xlim(-0.3, 11)
plt.ylabel('$Air Traffic$')
plt.yticks(fontsize=12, alpha=.7)
plt.title("Monthly Seasonal Plot: Air Passengers Traffic (1949 - 1969)", fontsize=22)
plt.grid(axis='y', alpha=.3)

# Remove borders
plt.gca().spines["top"].set_alpha(0.0)
plt.gca().spines["bottom"].set_alpha(0.5)
plt.gca().spines["right"].set_alpha(0.0)
plt.gca().spines["left"].set_alpha(0.5)
```

```
# plt.legend(loc='upper right', ncol=2, fontsize=12)
plt.show()
```

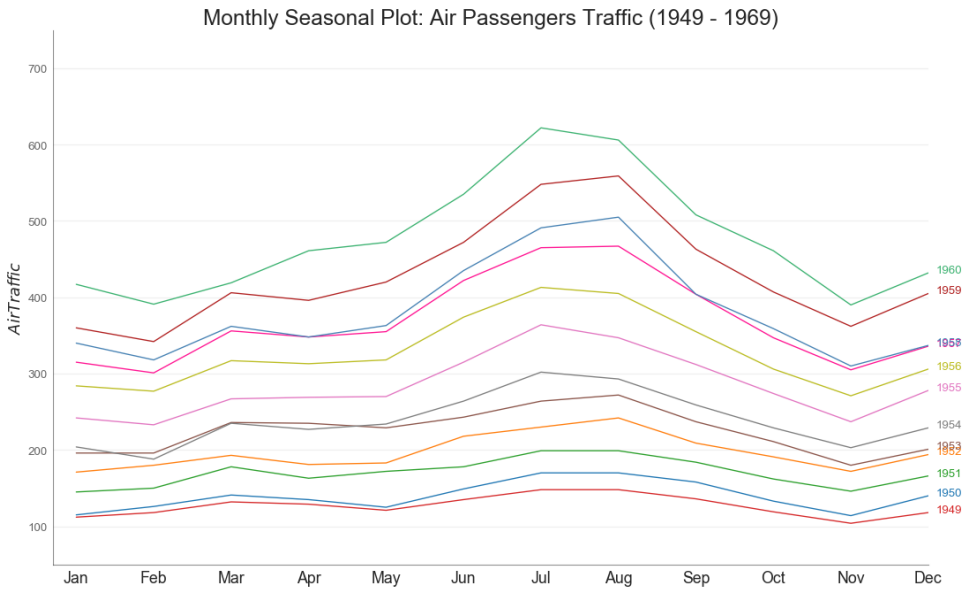


Figure 26.52: 季节图

26.9 分组（Groups）

26.9.1 树状图（Dendrogram）

树形图基于给定的距离度量将相似的点组合在一起，并基于点的相似性将它们组织在树状链接中。

```
import scipy.cluster.hierarchy as shc

# Import Data
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/
    USArrests.csv')

# Plot
plt.figure(figsize=(16, 10), dpi= 80)
plt.title("USArrests Dendograms", fontsize=22)
dend = shc.dendrogram(shc.linkage(df[['Murder', 'Assault', 'UrbanPop', 'Rape'
    ]], method='ward'), labels=df.State.values, color_threshold=100)
plt.xticks(fontsize=12)
plt.show()
```

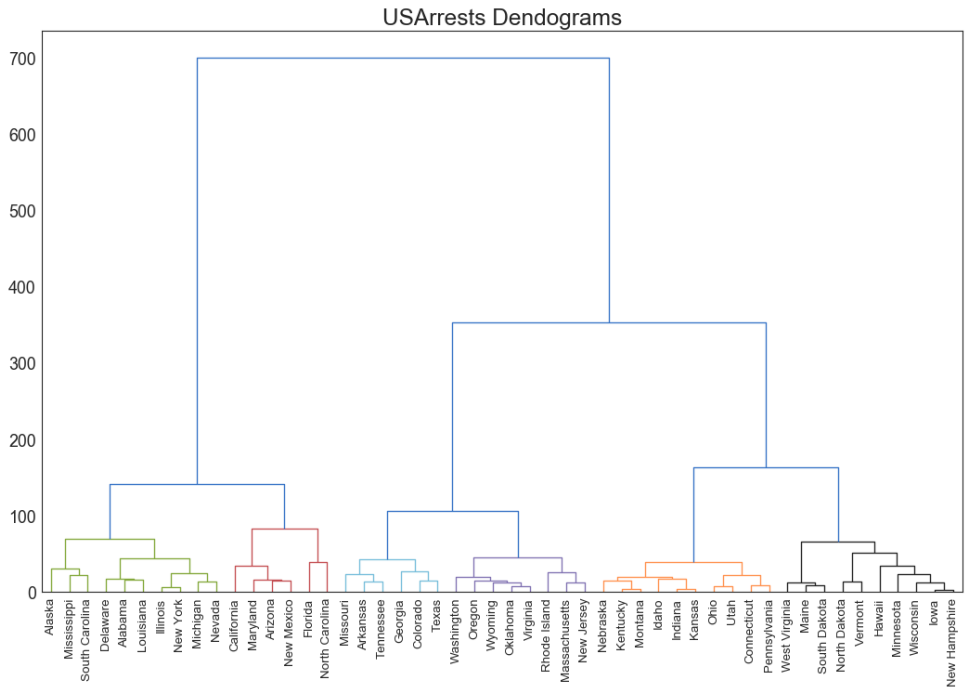


Figure 26.53: 树状图

26.9.2 簇状图（Cluster Plot）

簇状图（Cluster Plot）可用于划分属于同一群集的点。下面是根据 USArrests 数据集将美国各州分为 5 组的代表性示例。此图使用“谋杀”和“攻击”列作为 X 和 Y 轴。或者，您可以将第一个到主要组件用作 X 轴和 Y 轴。

```
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial import ConvexHull

# Import Data
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/
    USArrests.csv')

# Agglomerative Clustering
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='
    ward')
cluster.fit_predict(df[['Murder', 'Assault', 'UrbanPop', 'Rape']])

# Plot
plt.figure(figsize=(14, 10), dpi= 80)
plt.scatter(df.iloc[:,0], df.iloc[:,1], c=cluster.labels_, cmap='tab10')

# Encircle
def encircle(x,y, ax=None, **kw):
    if not ax: ax=plt.gca()
    p = np.c_[x,y]
```

```
    hull = ConvexHull(p)
    poly = plt.Polygon(p[hull.vertices,:], **kw)
    ax.add_patch(poly)

# Draw polygon surrounding vertices
encircle(df.loc[cluster.labels_ == 0, 'Murder'], df.loc[cluster.labels_ == 0, '
    Assault'], ec="k", fc="gold", alpha=0.2, linewidth=0)
encircle(df.loc[cluster.labels_ == 1, 'Murder'], df.loc[cluster.labels_ == 1, '
    Assault'], ec="k", fc="tab:blue", alpha=0.2, linewidth=0)
encircle(df.loc[cluster.labels_ == 2, 'Murder'], df.loc[cluster.labels_ == 2, '
    Assault'], ec="k", fc="tab:red", alpha=0.2, linewidth=0)
encircle(df.loc[cluster.labels_ == 3, 'Murder'], df.loc[cluster.labels_ == 3, '
    Assault'], ec="k", fc="tab:green", alpha=0.2, linewidth=0)
encircle(df.loc[cluster.labels_ == 4, 'Murder'], df.loc[cluster.labels_ == 4, '
    Assault'], ec="k", fc="tab:orange", alpha=0.2, linewidth=0)

# Decorations
plt.xlabel('Murder'); plt.xticks(fontsize=12)
plt.ylabel('Assault'); plt.yticks(fontsize=12)
plt.title('Agglomerative Clustering of USArrests (5 Groups)', fontsize=22)
plt.show()
```

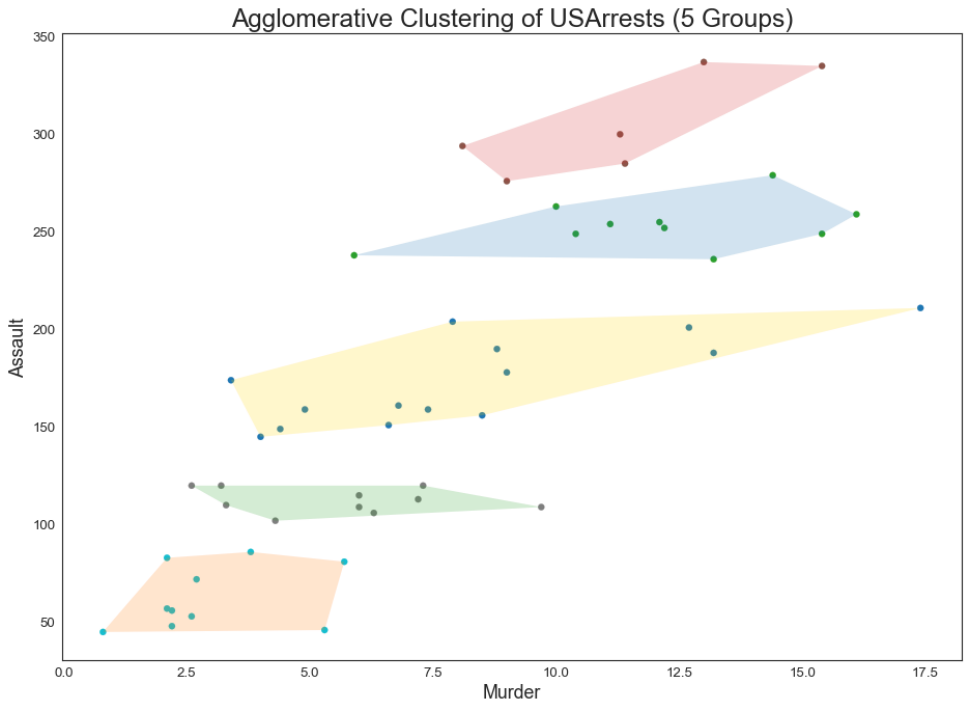


Figure 26.54: 簇状图

26.9.3 安德鲁斯曲线（Andrews Curve）

安德鲁斯曲线有助于可视化是否存在基于给定分组的数字特征的固有分组。如果要素（数据集中的列）无法区分组（cyl），那么这些线将不会很好地隔离，如下所示。

```
from pandas.plotting import andrews_curves

# Import
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mtcars.csv")
df.drop(['cars', 'carname'], axis=1, inplace=True)

# Plot
plt.figure(figsize=(12,9), dpi= 80)
andrews_curves(df, 'cyl', colormap='Set1')

# Lighten borders
plt.gca().spines["top"].set_alpha(0)
plt.gca().spines["bottom"].set_alpha(.3)
plt.gca().spines["right"].set_alpha(0)
plt.gca().spines["left"].set_alpha(.3)

plt.title('Andrews Curves of mtcars', fontsize=22)
plt.xlim(-3,3)
plt.grid(alpha=0.3)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

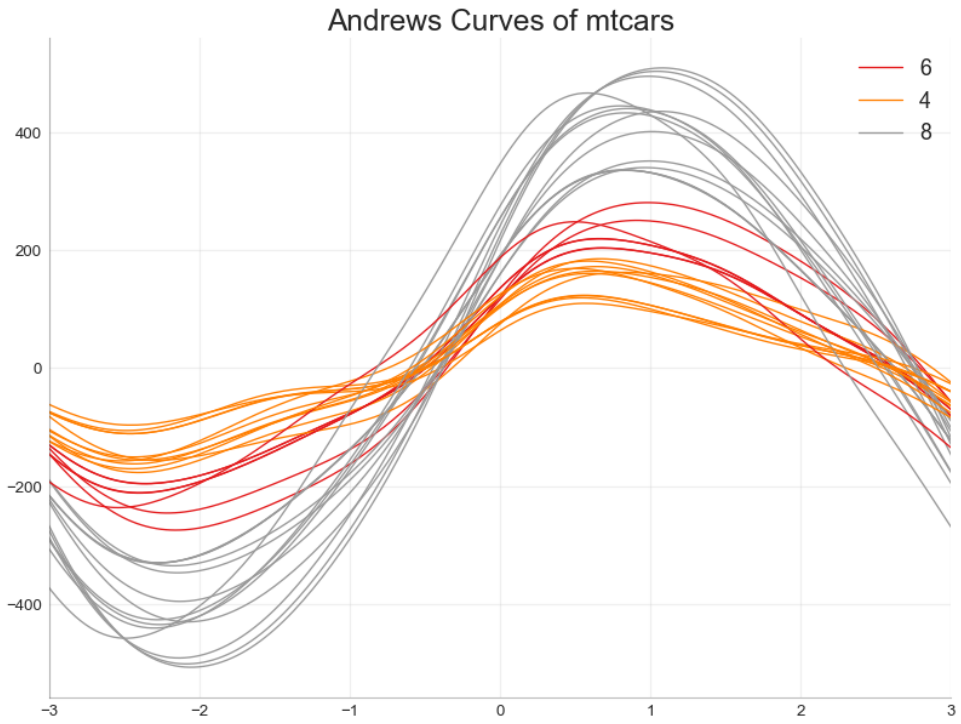


Figure 26.55: 安德鲁斯曲线

26.9.4 平行坐标（Parallel Coordinates）

平行坐标有助于可视化特征是否有助于有效地隔离组。如果实现隔离，则该特征可能在预测该组时非常有用。

```
from pandas.plotting import parallel_coordinates

# Import Data
df_final = pd.read_csv("https://raw.githubusercontent.com/selva86/datasets/
    master/diamonds_filter.csv")

# Plot
plt.figure(figsize=(12,9), dpi= 80)
parallel_coordinates(df_final, 'cut', colormap='Dark2')

# Lighten borders
plt.gca().spines["top"].set_alpha(0)
plt.gca().spines["bottom"].set_alpha(.3)
plt.gca().spines["right"].set_alpha(0)
plt.gca().spines["left"].set_alpha(.3)

plt.title('Parallel Coordinated of Diamonds', fontsize=22)
plt.grid(alpha=0.3)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
20  plt.show()
```

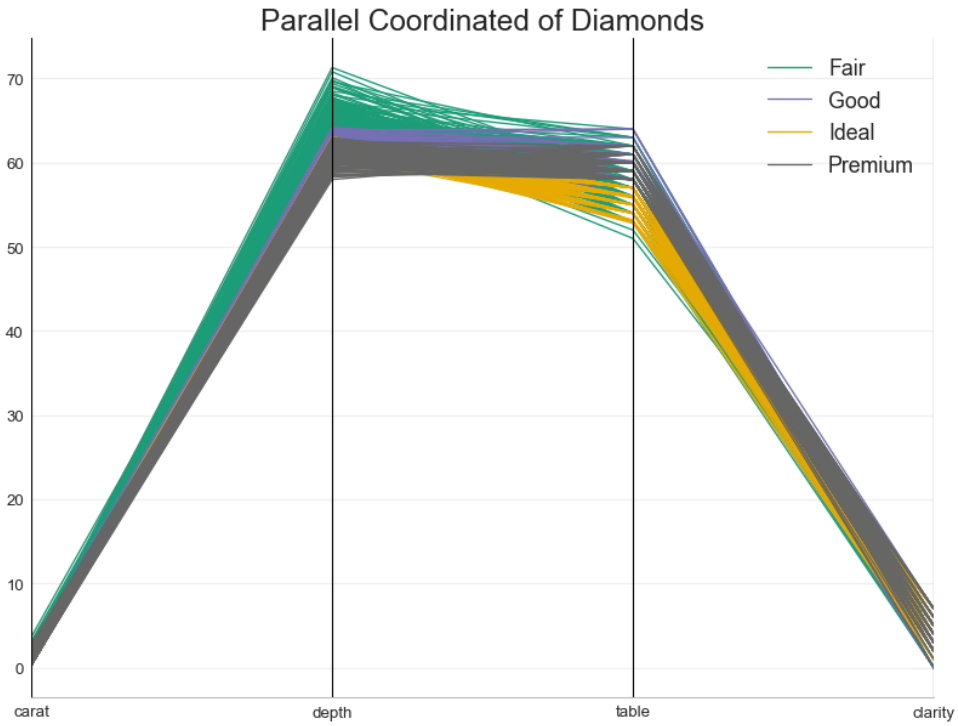


Figure 26.56: 平行坐标

> 原文：
>
> Top 50 matplotlib Visualizations – The Master Plots (with full python code)

> Tips：
>
> (1) 本文原文部分代码有不准确的地方，已进行修改；
>
> (2) 所有正确的源代码，我已整合到 jupyter notebook 文件中，可以在公众号『Python 数据之道』后台回复“code”，可获得本文源代码；
>
> (3) 运行本文代码，除了安装 matplotlib 和 seaborn 可视化库外，还需要安装其他的一些辅助可视化库，已在代码部分作标注。

对我的文章感兴趣的朋友，欢迎关注我的微信公众号『Python 数据之道』(ID：PyDataLab)，接收我的更新通知。

Figure 26.57: 微信公众号: Python 数据之道

26.9.5 延伸阅读

- Matplotlib 实操干货，100 个案例带你从入门到进阶！-Part 1
- Matplotlib 实操干货，100 个案例带你从入门到进阶！-Part 2

27 轻松用 Seaborn 进行数据可视化

> 作者 | Mukul Singh Chauhan
>
> 译者 | Lemon
>
> 出品 | Python 数据之道（ID:PyDataLab）

Seaborn 是一个数据可视化库，可帮助在 Python 中创建有趣的数据可视化。大多数数据分析需要识别趋势和建立模型。本文将帮助您开始使用 Seaborn 库创建数据可视化。

Seaborn 是一个出色的数据可视化库，它让我们的生活变得轻松。首先，您应该在 jupyter notebook 中键入以下命令。

```
import pandas as pd # Pandas
import numpy as np # Numpy
import matplotlib.pyplot as plt # Matplotlibrary
import seaborn as sns # Seaborn Library
%matplotlib inline
sns.set()
```

27.1 直方图（Distplot）

sns.distplot（）结合直方图并绘制核密度估计图。这里 bin 区间大小是自动计算的。

$$sns.distplot(data[“variablename”])$$

我们将使用以下代码在 jupyter notebook 中加载数据集。

```
# Load the Dataset in Python
tips = sns.load_dataset("tips")
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Figure 27.1: tips 数据集

现在，由于我们已经加载了数据集，我们将使用 “total_bill” 变量创建第一个图。让我们从 tips 数据集创建 “total_bill” 变量的 distplot。

```
sns.distplot(tips["total_bill"], bins=16, color="purple")
# Binsize is calculated using square-root of row count.
```

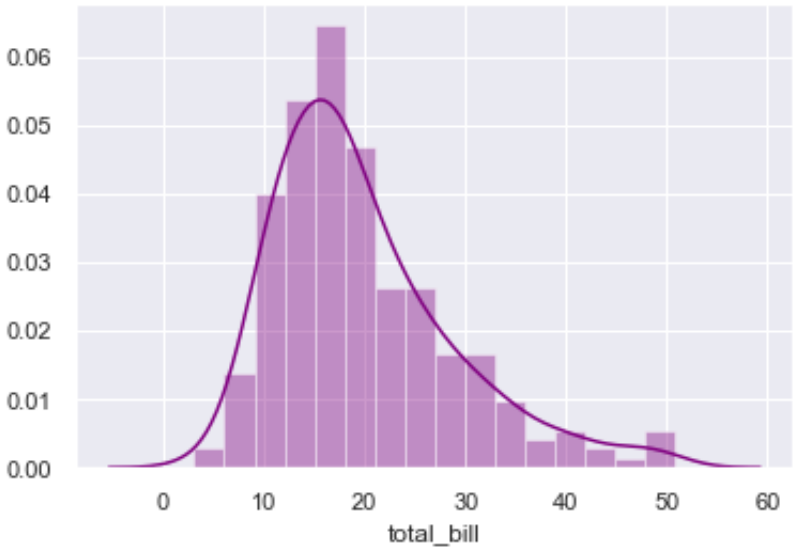


Figure 27.2: 直方图

现在，我们来对上述代码进行进一步描述：

- **sns.distplot**— 这个命令将启动 distplot 的初始创建
- **tips[“total_bill”]**— 从 tips 数据集（数据框）中取出列（total_bill）。在这里，我们应该观察一下，可以使用方括号来拉取列值，并且列名应该用引号括起来（双引号/单引号）都被接受。

我们推测 —“total_bill”变量本质上是倾斜的，大多数帐单值都在 $ 10 - $ 20 范围内。

27.2 联合分布图（Jointplot）

联合分布图（Jointplot）采用两个变量并一起创建直方图和散点图。让我们看一下 jointplot 的语法。

$$sns.jointplot(x =, y =, data =)$$

让我们从 tips 数据集创建 total_bill 和 tip 变量的联合分布图。通常，任何餐厅的小费金额取决于总账单/账单大小。让我们看看这个情景下是什么样的。代码如下：

```
1 sns.jointplot(x = "total_bill", y = "tip", data = tips, color="purple")
```

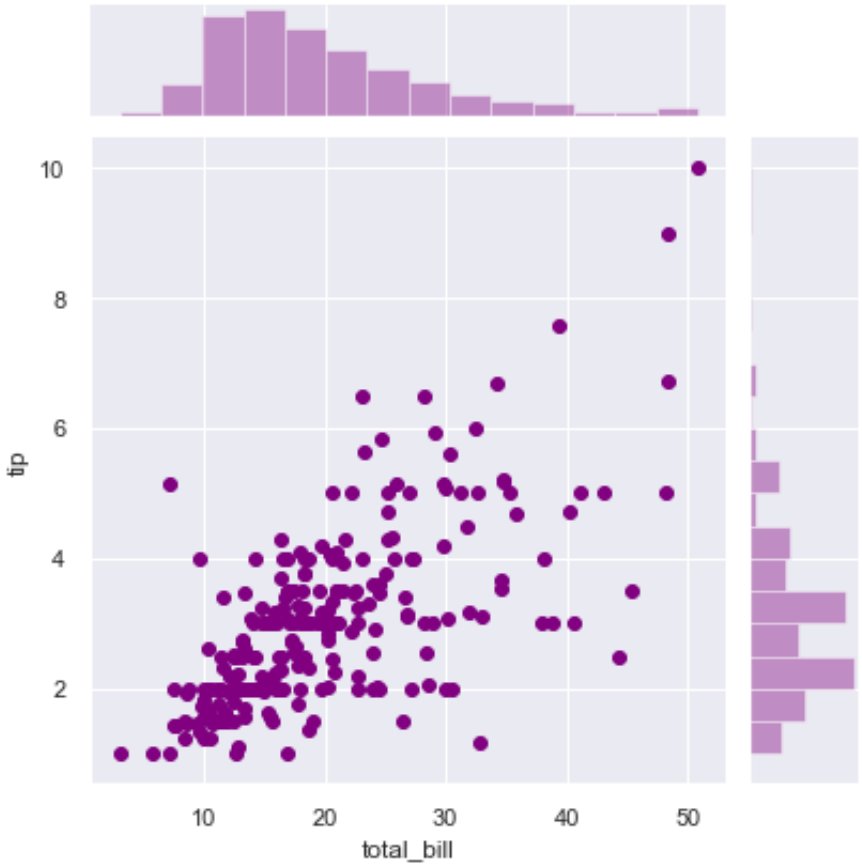


Figure 27.3: 联合分布图

如上所述，散点图似乎显示总账单和小费金额之间的强相关性。在它的顶部，我们可以看到各个变量的直方图。

27.2.1 Jointplot :: kind =”hex”

直方图的双变量类比称为“hexbin”图，因为它显示了六边形区间内的观察计数。此图对于相对较大的数据集最有效。也称为 Hexbin Plots。

$$sns.jointplot(x =, y =, data =, kind = ”hex”)$$

```
# Jointplot - Scatterplot and Histogram
sns.jointplot(x = "total_bill", y = "tip", data = tips, kind ="hex",
color="lightcoral")
```

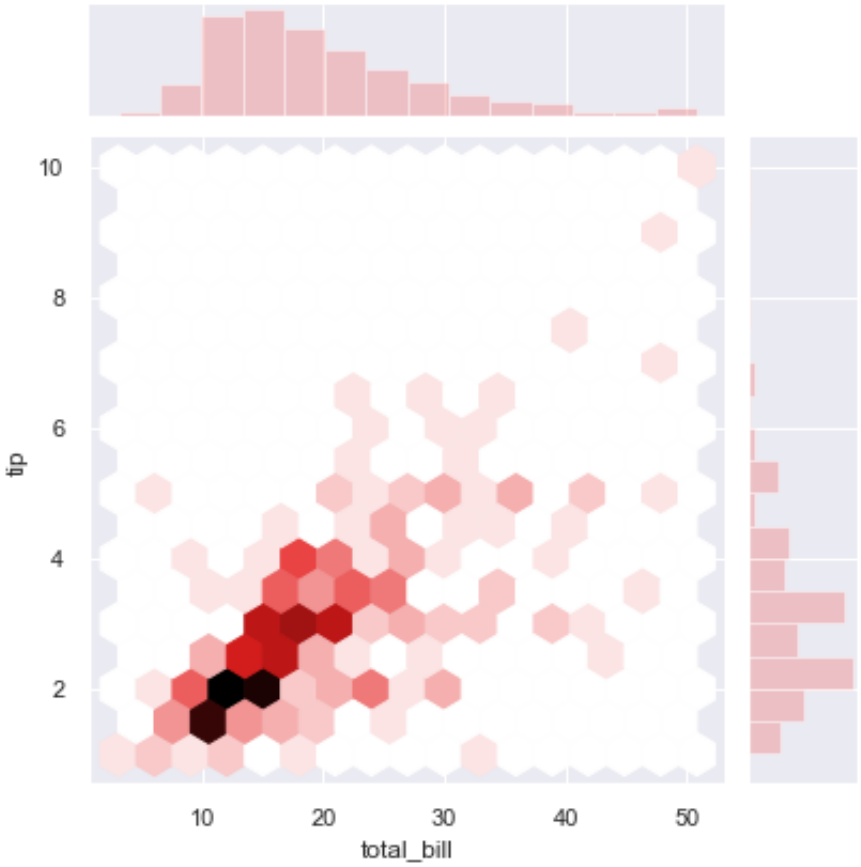


Figure 27.4: 联合分布图: kind=“hex”

有几种类型的值可以放在 sns.jointplot 中来创建不同的图。默认情况下，联合分布图显示散点图。现在，在上面的情节图中，它显示了六边形。六边形的深色表示数据点的高密度，其中较浅的颜色表示较少的点。

kind 参数值可以是以下取值：

> kind : { “scatter” | “reg” | “resid” | “kde” | “hex” }

下面，我们来看看 kind=“kde” 的情形。

27.2.2 Jointplot :: kind =”kde”

```
# Jointplot - Scatterplot and Histogram
sns.jointplot(x = tips["total_bill"], y = tips["tip"],kind = "kde",
color="purple") # contour plot
```

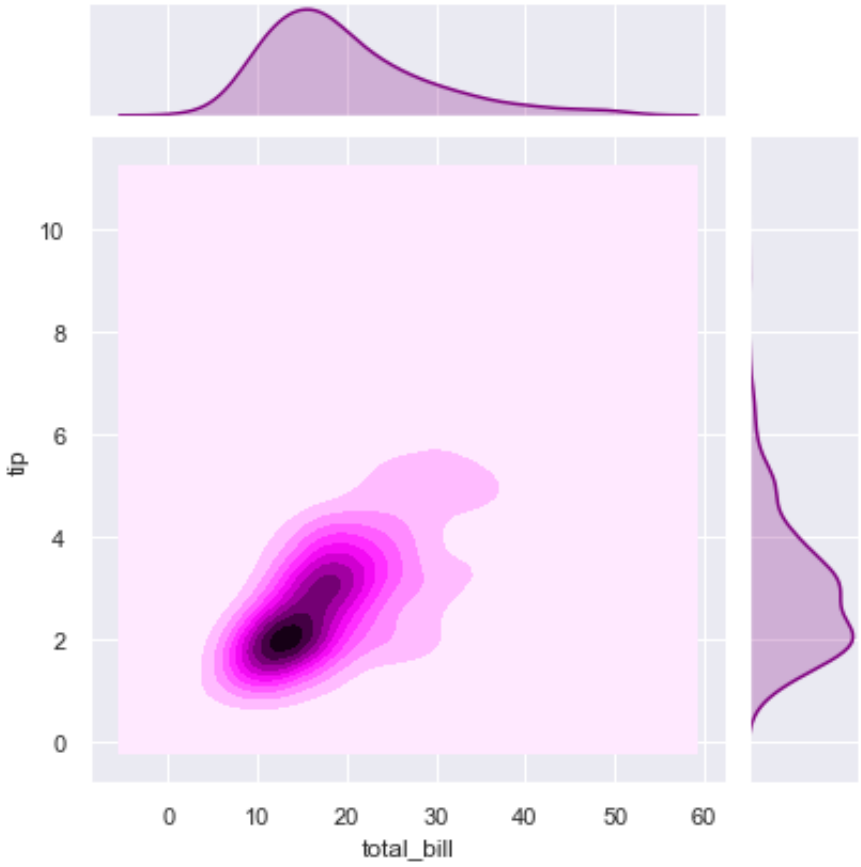


Figure 27.5: 联合分布图: kind=“kde”

上面显示的图表称为轮廓图。轮廓图（有时称为“水平图”）是一种在二维平面上显示三维表面的方法。它绘制了 y 轴上的两个预测变量 X Y 和轮廓的响应变量 Z.

27.3 矩阵图（Pairplot）

矩阵图基本上绘制了变量之间的成对关系。它支持用 “hue” 来为类别变量绘图着色。

$$sns.pairplot(“dataframe”)$$

```
# Pairplot of Tips
sns.pairplot(tips, hue = "sex", palette="Set2")
# this  will color the plot gender wise
```

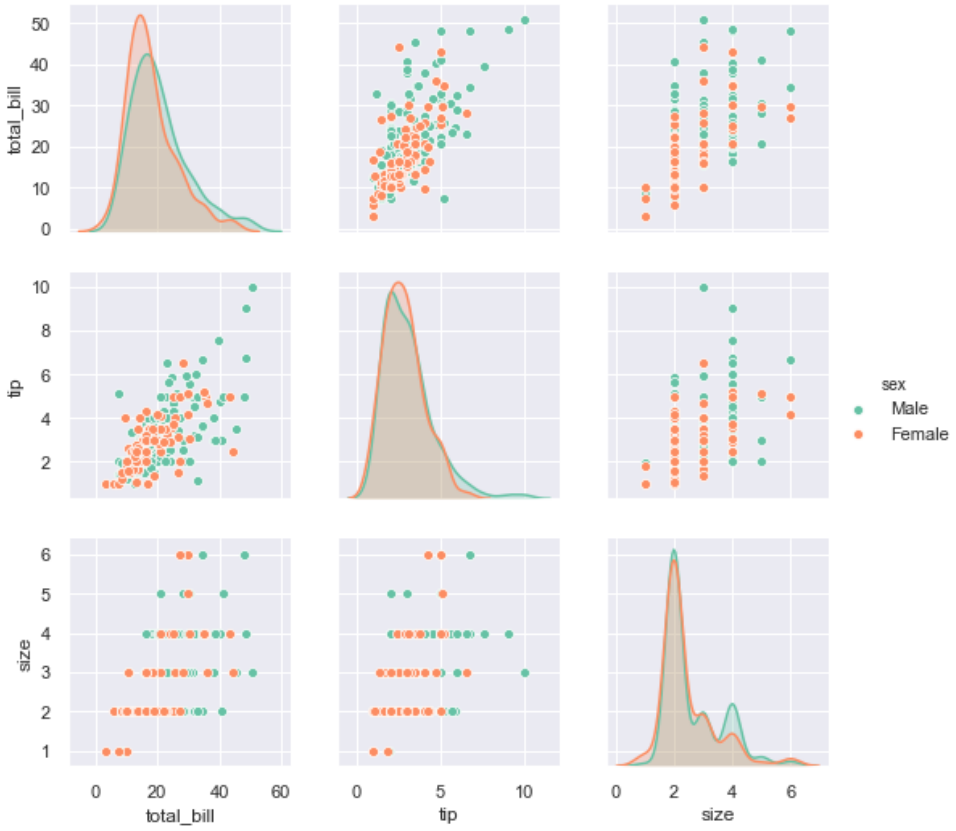


Figure 27.6: 矩阵图

下面我们来了解下矩阵图的含义。对角线部分显示了具有核密度估计的 distplot 图或直方图。矩阵图的上部和下部显示散点图。“hue”使用列的类别为绘图着色。

- hue = “sex” — 设置为按不同的性别进行着色
- palette = “Set2” - “Set2” 是颜色的一个系列。

27.4 条形图（Barplot）

条形图用于绘制分类列和数字列。它在可视化中创建了条形。让我们用“性别”创建一个“total_bill”的条形图，让我们看看哪类人支付更多。

$$sns.barplot(x =, y =, data =)$$

```
# Barplot
sns.barplot(x ="sex" , y ="total_bill" , data=tips)
# Inference - Total Bill Amount for males is more than Females.
```

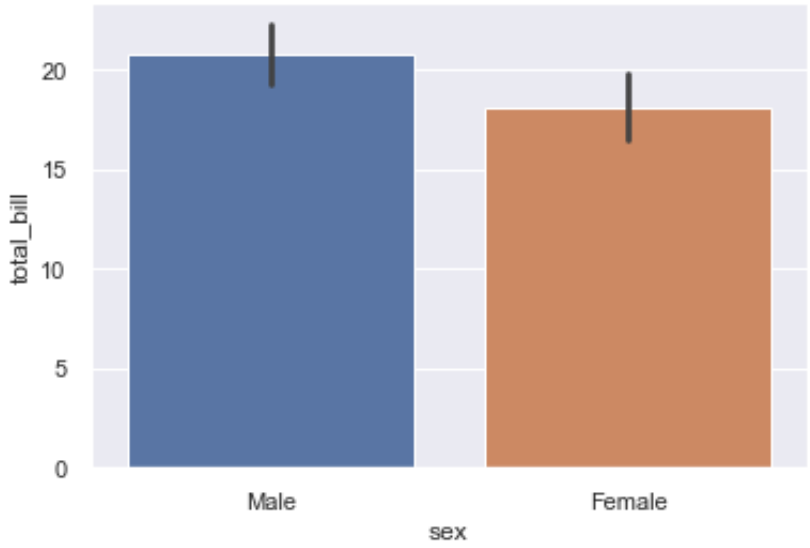


Figure 27.7: 条形图

```
# Lets Plot Smoker Vs Total Bill :: The purpose is to find out if
# Smokers pay more bill than Non Smokers
sns.barplot(x = "smoker", y = "total_bill", data =tips)
# Inference - More Bill for Smokers
```

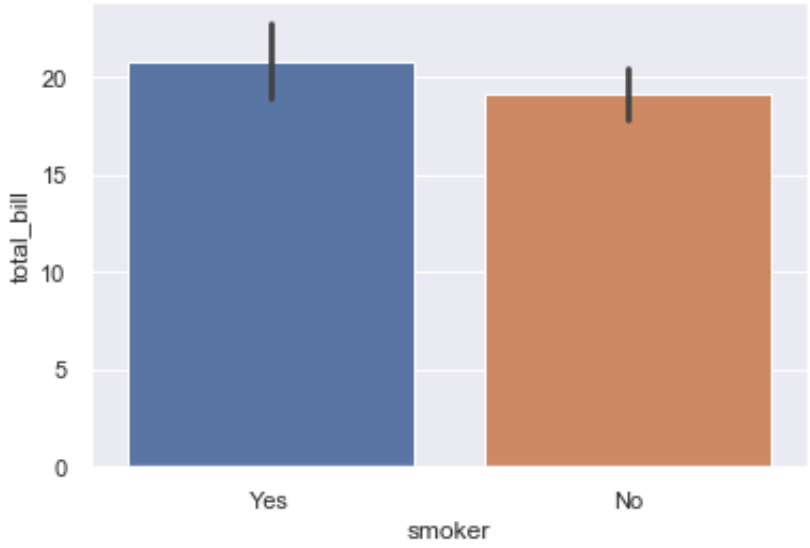


Figure 27.8: 条形图

```
# Lets Find If There is more Bill In Weekend or Weekdays
sns.barplot(x = "day", y = "total_bill", data =tips)
# People tend to visit more on weekends
```

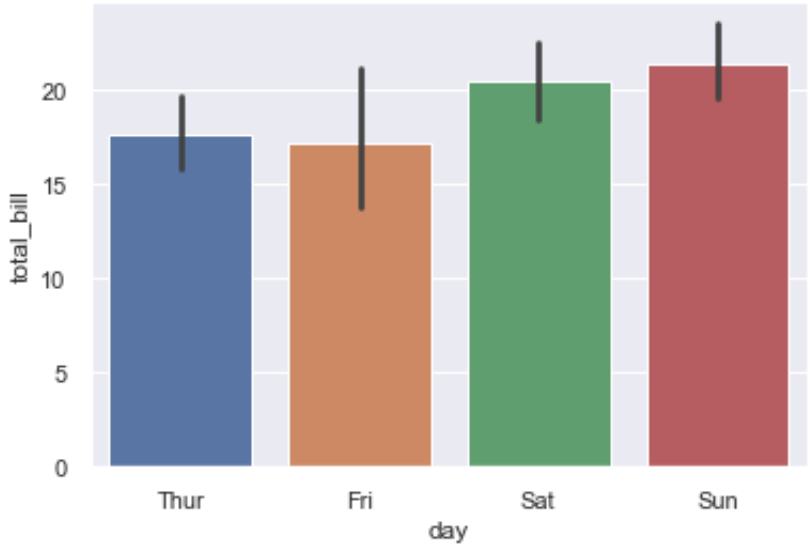


Figure 27.9: 条形图

27.5 箱形图（Boxplot）

箱形图（Boxplot）是给定数据集的五点汇总统计的直观表示。五个数字摘要包括：

- Minimum 最小值
- First Quartile 1/4 值
- Median (Second Quartile) 中位数
- Third Quartile 3/4 值
- Maximum 最大值

此外，值得注意的一点是，为分类 - 连续变量创建了一个箱线图，这意味着如果 x 轴是分类的并且 y 轴是连续的，则应创建箱线图或小提琴图。

让我们从 tips 数据集创建一个 “day” 和 “total_bill” 的箱线图。

$$sns.boxplot(x =, y =, data =)$$

```
# Boxplot
sns.boxplot(x = "day", y = "total_bill", data=tips)
```

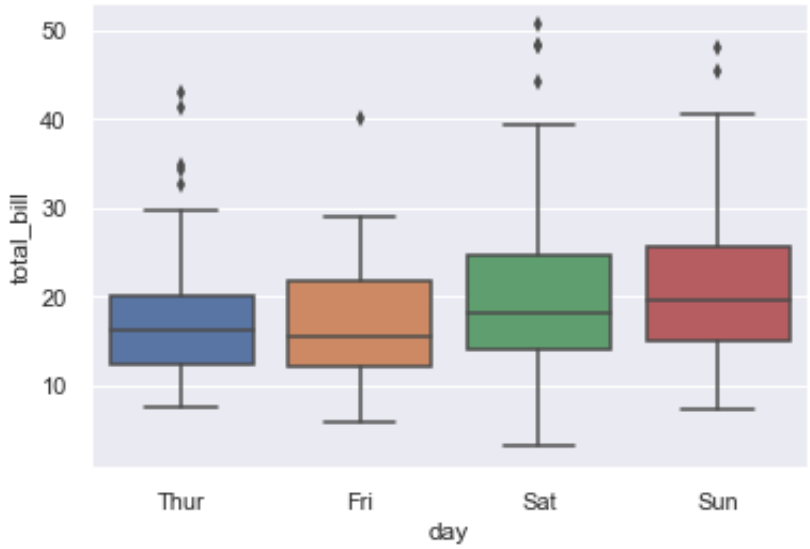


Figure 27.10: 箱形图

```
# Add hue to split the barplot. Making it more fancier
sns.boxplot(x = "day", y = "total_bill", data=tips, hue = "smoker")
# On Friday people have more bill if they are a Non smoker vs smoker
```

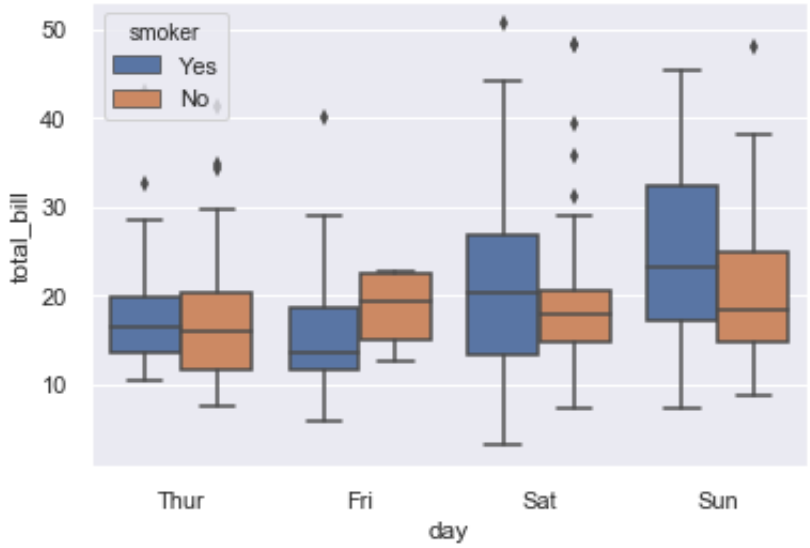


Figure 27.11: 箱形图

hue ="smoker": - 它为吸烟者和非吸烟者创造了一个箱线图。例如：在星期五的情况下，可以清楚地看到，与当天的吸烟者相比，非吸烟者的食物费用更多。

```
# Violin Plots
sns.violinplot(x = "day", y = "total_bill", data = tips)
```

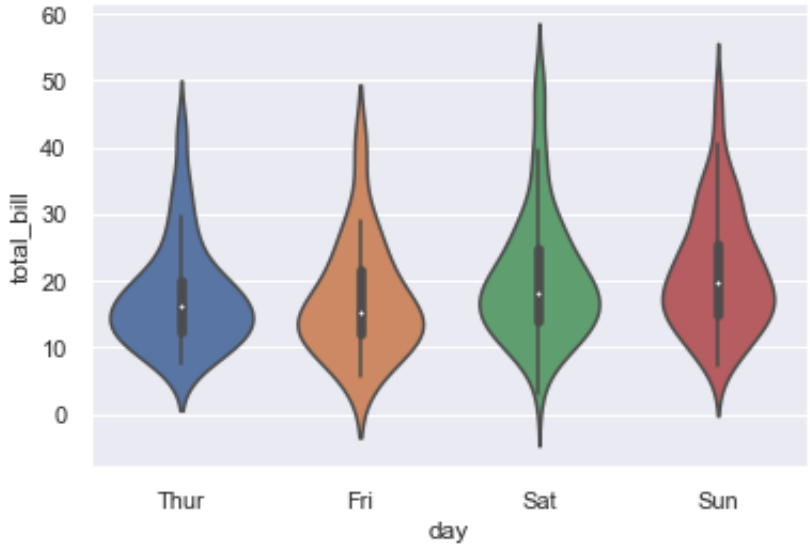


Figure 27.12: 小提琴图

小提琴图跟箱形图有些类似。他们之间的描述可以参考下面的图示内容：

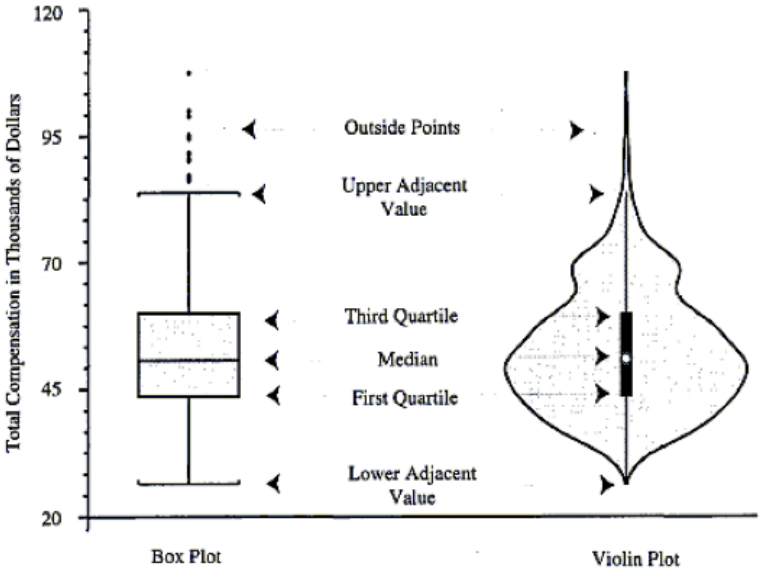


Figure 1. Common Components of Box Plot and Violin Plot. Total compensation for all academic ranks.

Image source: Google

Figure 27.13: 小提琴图示意图

27.6 LM Plot

sns.lmplot 是绘制一个在散点图中进行线性回归拟合的直线。它遵循普通的最小二乘法，线代表最佳拟合线。为了更好地理解这一点，建议进一步了解一下线性回归的相关知识。

代码如下：

```
# LM PLot
sns.lmplot(x = "total_bill", y = "tip", data = tips, hue="day")
```

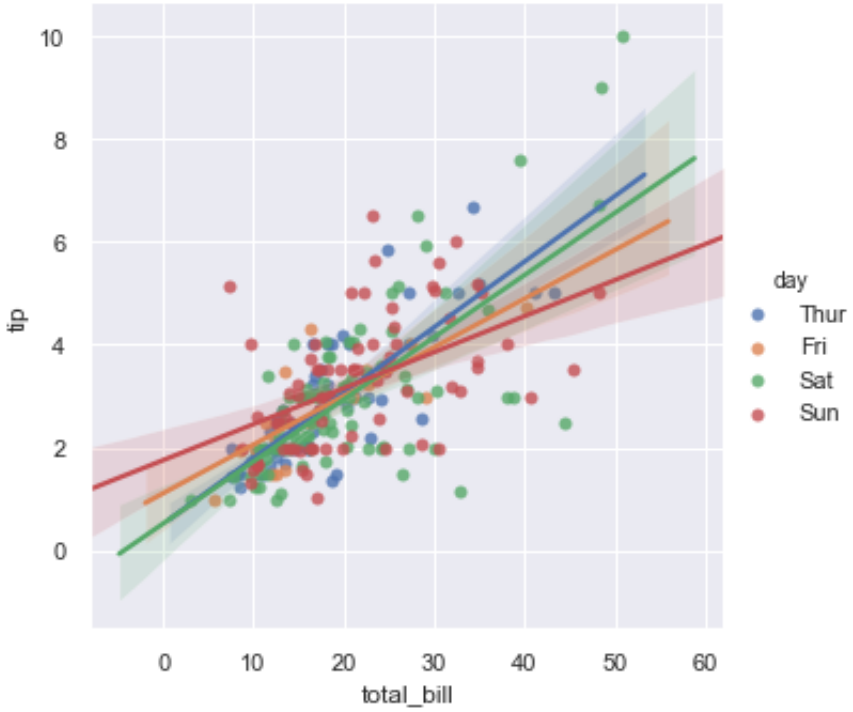


Figure 27.14: LM Plot

上图显示了不同日期的 total_bill 变量的线性回归拟合，如图例中所示，这是在 sns.lmplot 中使用 hue =“day”获得的。

恭喜！您已完成 Seaborn 初学者教程。希望这篇文章能够提供有关 Seaborn 的基本知识，并且可以帮助您创建所有这些图。

源代码：在微信公众号“Python 数据之道”（ID：PyDataLab）后台回复关键字“code”获取。

> 原文：
>
> Data Visualisation Using Seaborn

28 Seaborn 可视化：图形个性化设置的几个小技巧

28.1 概述

在可视化过程中，经常会对默认的制图效果不满意，希望能个性化进行各种设置。

本文通过一个简单的示例，来介绍 seaborn 可视化过程中的个性化设置。包括常用的设置，如：

1. 设置图表显示颜色
2. 设置图表标题，包括显示位置，字体大小，颜色等
3. 设置 x 轴和 y 轴标题，包括颜色，字体大小
4. 设置 x 轴和 y 轴刻度内容，包括颜色、字体大小、字体方向等
5. 将 x 轴和 y 轴内容逆序显示
6. 设置 x 轴或 y 轴显示位置

本文的运行环境：

1. windows 7
2. python 3.5
3. jupyter notebook
4. seaborn 0.7.1
5. matplotlib 2.0.2

28.2 未个性化设置的情形

本文的数据来自 UCI 的数据集“sonar”，用 pandas 直接读取数据。如下：

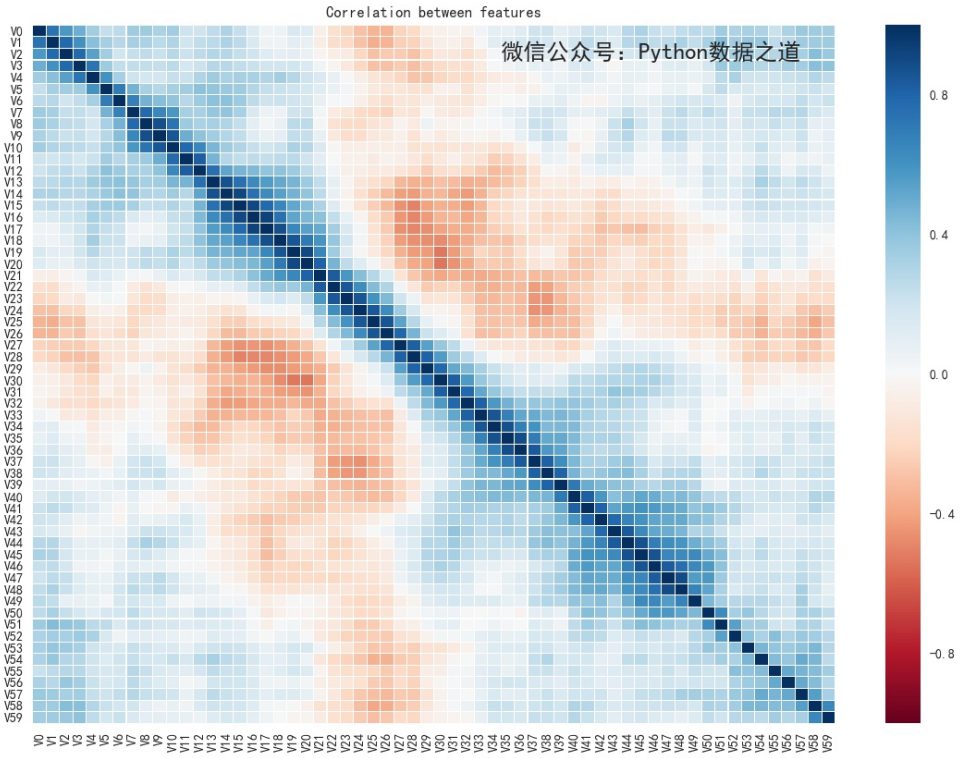
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline

target_url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/
    undocumented/connectionist-bench/sonar/sonar.all-data'
df = pd.read_csv(target_url, header=None, prefix='V')
corr = df.corr()
```

首先来看看没有进行个性化设置时的显示情况，如下：

```
f, ax= plt.subplots(figsize = (14, 10))
```

```
2
3  sns.heatmap(corr,cmap='RdBu', linewidths = 0.05, ax = ax)
4
5  # 设置Axes的标题
6  ax.set_title('Correlation between features')
7
8  f.savefig('sns_style_origin.jpg', dpi=100, bbox_inches='tight')
```

图片显示效果如下：



seaborn 制图的默认效果其实还是不错的。

28.3 进行个性化设置

对于上面这张图，可能让 y 轴从下到上，从 v0 开始显示，这样显示出来的对角线可能更符合我们的视觉显示效果。

这就要用到 **将 y 轴内容进行可逆显示**，涉及的代码如下：

```
1  # 将y轴或x轴进行逆序
2  ax.invert_yaxis()
3  # ax.invert_xaxis()
```

其他的个性化设置的代码，包括：

将 x 轴刻度放置在 top 位置的几种方法

```
# 将x轴刻度放置在top位置的几种方法
# ax.xaxis.set_ticks_position('top')
ax.xaxis.tick_top()
# ax.tick_params(axis='x',labelsize=6, colors='b', labeltop=True, labelbottom=
    False) # x轴
```

设置坐标轴刻度参数，“axis”不写的时候，默认是 x 轴和 y 轴的参数同时调整。

```
# 设置坐标轴刻度的字体大小
# matplotlib.axes.Axes.tick_params
ax.tick_params(axis='y',labelsize=8) # y轴
```

旋转轴刻度上文字方向的两种方法

```
# 旋转轴刻度上文字方向的两种方法
ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)
# ax.set_xticklabels(corr.index, rotation=90)
```

保存图片，设置 bbox_inches=“tight”，保存的图片则不会出现部分内容显示不全的现象。

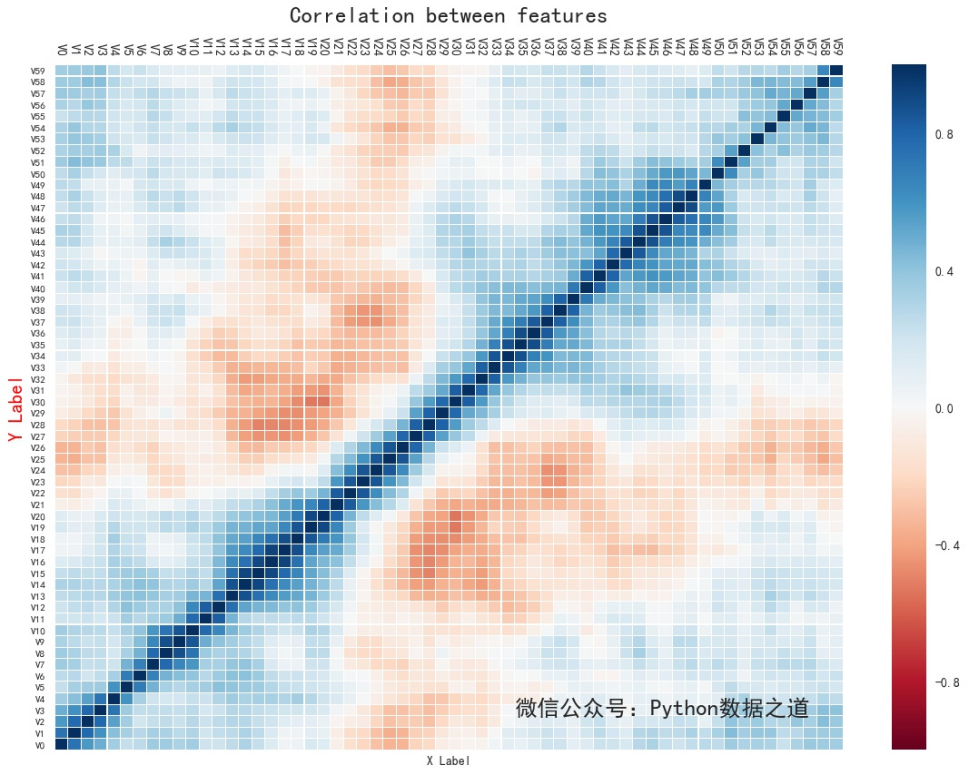
```
f.savefig('sns_style_update.jpg', dpi=100, bbox_inches='tight')
```

整合好的代码如下，大家可以运行试试效果。

```
f, ax = plt.subplots(figsize = (14, 10))

# 设置颜色
cmap = sns.cubehelix_palette(start = 1, rot = 3, gamma=0.8, as_cmap = True)

# color: https://matplotlib.org/users/colormaps.html
sns.heatmap(corr,cmap='RdBu', linewidths = 0.05, ax = ax)

# 设置Axes的标题
ax.set_title('Correlation between features', fontsize=18, position=(0.5,1.05))

# 将y轴或x轴进行逆序
ax.invert_yaxis()
# ax.invert_xaxis()

ax.set_xlabel('X Label',fontsize=10)

# 设置Y轴标签的字体大小和字体颜色
ax.set_ylabel('Y Label',fontsize=15, color='r')

# 设置坐标轴刻度的字体大小
# matplotlib.axes.Axes.tick_params
ax.tick_params(axis='y',labelsize=8) # y轴
# ax.tick_params(axis='x',labelsize=6, colors='b', labeltop=True, labelbottom=
    False) # x轴

# 将x轴刻度放置在top位置的几种方法
```

```
# ax.xaxis.set_ticks_position('top')
ax.xaxis.tick_top()
# ax.tick_params(axis='x',labelsize=6, colors='b', labeltop=True, labelbottom=
    False) # x轴

# 修改tick的字体颜色
# ax.tick_params(axis='x', colors='b') # x轴

# 旋转轴刻度上文字方向的两种方法
ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)
# ax.set_xticklabels(corr.index, rotation=90)

# 单独设置y轴或x轴刻度的字体大小，调整字体方向
# ax.set_yticklabels(ax.get_yticklabels(),fontsize=6)
# ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)

f.savefig('sns_style_update.jpg', dpi=100, bbox_inches='tight')
```

图形显示效果如下：



这些个性化的设置，其实大部分都是使用的 matplotlib 的内容，seaborn 是基于 matplotlib 衍生的，所以可以跟 matplotlib 进行融合使用。

当然，并不是每次都需要进行个性定制，具体可以根据自己的需求来设置。

29 Seaborn 热力图使用进阶

在日常工作中，经常可以见到各种各种精美的热力图，热力图的应用非常广泛，下面一起来学习下 Python 的 Seaborn 库中热力图（heatmap）如何来进行使用。

本次运行的环境为：

- windows 64 位系统
- python 3.5
- jupyter notebook

29.1 构造数据

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

```
region = ['Albania', 'Algeria', 'Angola', 'Argentina', 'Armenia', 'Azerbaijan',
       'Bahamas', 'Bangladesh', 'Belize', 'Bhutan', 'Bolivia',
       'Bosnia and Herzegovina', 'Brazil', 'Burkina Faso', 'Burundi',
       'Cambodia', 'Cameroon', 'Cape Verde', 'Chile', 'China', 'Colombia',
       'Costa Rica', 'Cote d Ivoire', 'Cuba', 'Cyprus',
       "Democratic People's Republic of Korea",
       'Democratic Republic of the Congo', 'Dominican Republic', 'Ecuador',
       'Egypt', 'El Salvador', 'Equatorial Guinea', 'Ethiopia', 'Fiji',
       'Gambia', 'Georgia', 'Ghana', 'Guatemala', 'Guyana', 'Honduras']


kind = ['Afforestation & reforestation', 'Biofuels', 'Biogas',
        'Biomass', 'Cement', 'Energy efficiency', 'Fuel switch',
       'HFC reduction/avoidance', 'Hydro power',
        'Leak reduction', 'Material use', 'Methane avoidance',
       'N2O decomposition', 'Other renewable energies',
       'PFC reduction and substitution','PV',
       'SF6 replacement', 'Transportation', 'Waste gas/heat utilization',
      'Wind power']
```

```
print(len(region))
```

```
print(len(kind))
```

```
40
20
```

```
np.random.seed(100)
arr_region = np.random.choice(region, size=(10000,))
list_region = list(arr_region)

arr_kind = np.random.choice(kind, size=(10000,))
list_kind = list(arr_kind)

values = np.random.randint(50, 1000, 10000)
list_values = list(values)

df = pd.DataFrame({'region':list_region,
                   'kind': list_kind,
                   'values':list_values})
df.head()
```

	kind	region	values
0	Other renewable energies	Belize	74
1	Wind power	Cyprus	809
2	Material use	Argentina	593
3	HFC reduction/avoidance	Honduras	78
4	SF6 replacement	Cuba	206

微信公众号：Python数据之道

Figure 29.1: data

```
pt = df.pivot_table(index='kind', columns='region', values='values', aggfunc=np
    .sum)
pt.head()
```

region	Albania	Algeria	Angola	Argentina	Armenia	Azerbaijan	Bahamas	Bangladesh	Belize	Bhutan	...	El Salvador	Equatorial Guinea	Ethiopia	Fiji	Gambia	Georgia
kind																	
Afforestation & reforestation	5432	11676	6613	7570	4776	8941	5374	12947	4103	6257	...	5869	9744	6278	7281	6720	8856
Biofuels	6455	5128	8017	9016	8028	5995	5705	9404	7088	6510	...	6387	6042	7057	5320	5775	3621
Biogas	6691	5962	7253	9310	6153	8268	6420	6682	7111	6982	...	7657	7016	3334	6148	9031	3137
Biomass	9488	6292	7629	9487	6598	7922	6869	6007	5963	7537	...	5434	8764	4049	9007	8353	2800
Cement	2677	5759	5631	6666	9029	6712	7972	2894	7288	10161	...	9143	8585	2941	4523	4447	8909

5 rows × 40 columns

微信公众号：Python数据之道

Figure 29.2: data

```
f, ax = plt.subplots(figsize = (10, 4))
cmap = sns.cubehelix_palette(start = 1, rot = 3, gamma=0.8, as_cmap = True)
sns.heatmap(pt, cmap = cmap, linewidths = 0.05, ax = ax)
ax.set_title('Amounts per kind and region')
ax.set_xlabel('region')
ax.set_ylabel('kind')

f.savefig('sns_heatmap_normal.jpg', bbox_inches='tight')
# ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)
```

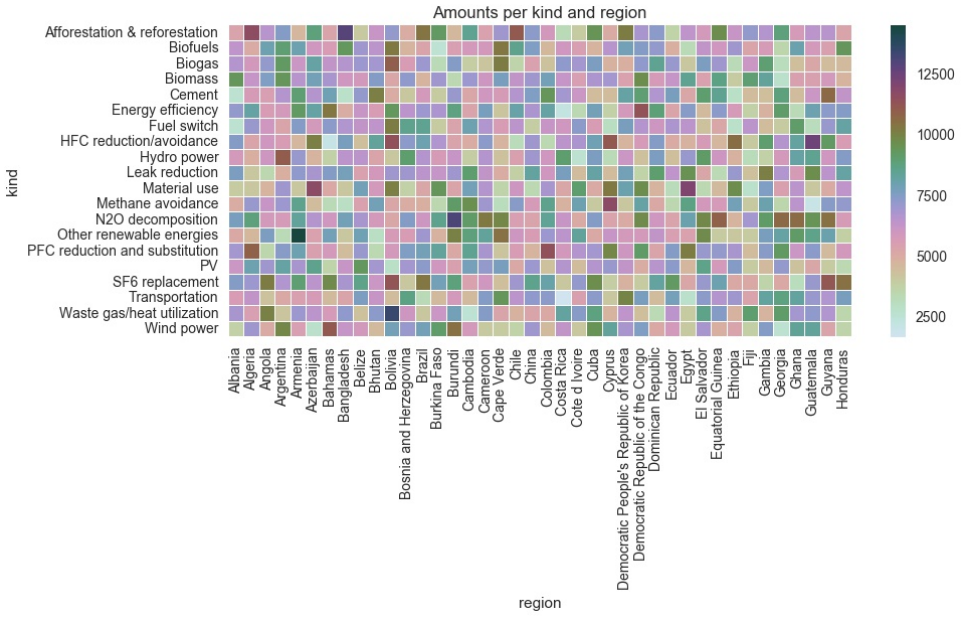


Figure 29.3: img-normal

29.2 Seaborn 的 heatmap 各个参数介绍

seaborn.heatmap

seaborn.heatmap(data, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt=".2g", annot_kws=None, linewidths=0, linecolor="white", cbar=True, cbar_kws=None, cbar_ax=None, square=False, ax=None, xticklabels=True, yticklabels=True, mask=None, **kwargs)

- data: 矩阵数据集，可以使 numpy 的数组 (array)，如果是 pandas 的 dataframe，则 df 的 index/column 信息会分别对应到 heatmap 的 columns 和 rows
- linewidths, 热力图矩阵之间的间隔大小
- vmax,vmin, 图例中最大值和最小值的显示值，没有该参数时默认不显示

29.2.1 cmap

- cmap：matplotlib 的 colormap 名称或颜色对象；如果没有提供，默认为 cubehelix map (数据集为连续数据集时) 或 RdBu_r (数据集为离散数据集时)

```
f, (ax1,ax2) = plt.subplots(figsize = (10, 8),nrows=2)

# cubehelix map颜色
cmap = sns.cubehelix_palette(start = 1.5, rot = 3, gamma=0.8, as_cmap = True)
sns.heatmap(pt, linewidths = 0.05, ax = ax1, vmax=15000, vmin=0, cmap=cmap)
ax1.set_title('cubehelix map')
ax1.set_xlabel('')
ax1.set_xticklabels([]) #设置x轴图例为空值
ax1.set_ylabel('kind')

# matplotlib colormap
sns.heatmap(pt, linewidths = 0.05, ax = ax2, vmax=15000, vmin=0, cmap='rainbow'
    )
# rainbow为 matplotlib 的colormap名称
ax2.set_title('matplotlib colormap')
ax2.set_xlabel('region')
ax2.set_ylabel('kind')

f.savefig('sns_heatmap_cmap.jpg', bbox_inches='tight')
```

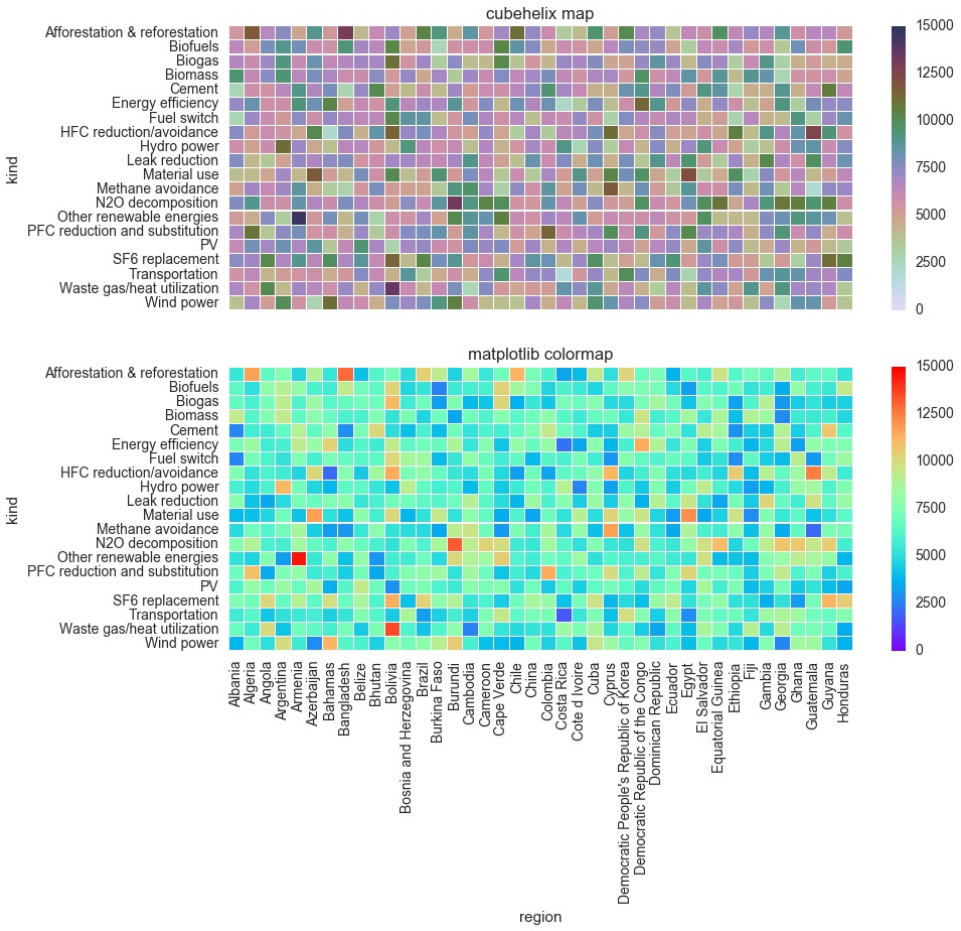


Figure 29.4: img-camp

29.2.2 center

- center: 将数据设置为图例中的均值数据，即图例中心的数据值；通过设置 center 值，可以调整生成的图像颜色的整体深浅；设置 center 数据时，如果有数据溢出，则手动设置的 vmax、vmin 会自动改变

```
f, (ax1,ax2) = plt.subplots(figsize = (10, 8),nrows=2)

cmap = sns.cubehelix_palette(start = 1.5, rot = 3, gamma=0.8, as_cmap = True)

sns.heatmap(pt, linewidths = 0.05, ax = ax1, vmax=15000, vmin=0, cmap=cmap,
    center=None )
# center为None时，由于最小值为0，最大值为15000，相当于center值为vamx和vmin的均值，即7500
ax1.set_title('center=None')
ax1.set_xlabel('')
ax1.set_xticklabels([]) #设置x轴图例为空值
ax1.set_ylabel('kind')
```

```
11
12 sns.heatmap(pt, linewidths = 0.05, ax = ax2, vmax=15000, vmin=0, cmap=cmap,
       center=3000 )
13 # 由于均值为2000，当center设置为3000时，大部分数据会比7500大，所以center=3000
       时，生成的图片颜色要深
14 # 设置center数据时，如果有数据溢出，则手动设置的vmax或vmin会自动改变
15 ax2.set_title('center=3000')
16 ax2.set_xlabel('region')
17 ax2.set_ylabel('kind')
18
19 f.savefig('sns_heatmap_center.jpg', bbox_inches='tight')
```

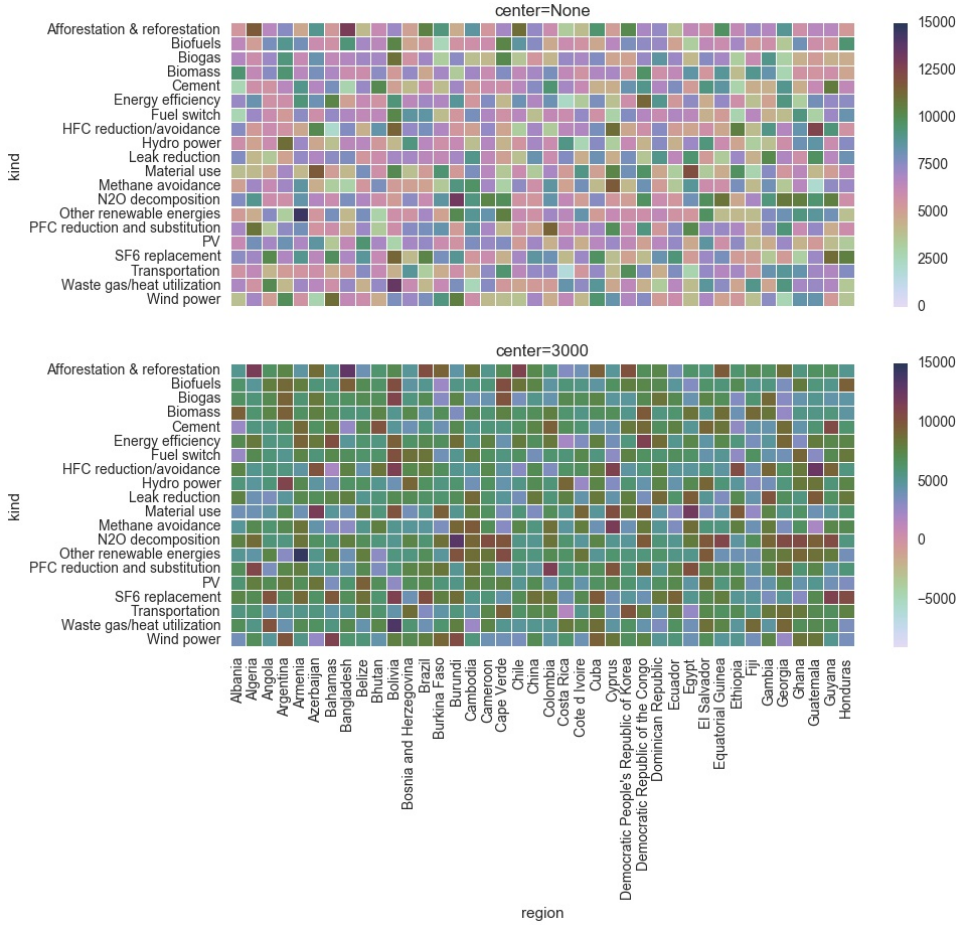


Figure 29.5: img-center

29.2.3 robust

```
1 f, (ax1,ax2) = plt.subplots(figsize = (10, 8),nrows=2)
2
3 cmap = sns.cubehelix_palette(start = 1.5, rot = 3, gamma=0.8, as_cmap = True)
```

```
4
5  sns.heatmap(pt, linewidths = 0.05, ax = ax1, cmap=cmap, center=None, robust=
       False )
6  # robust默认为False
7  ax1.set_title('robust=False')
8  ax1.set_xlabel('')
9  ax1.set_xticklabels([]) #设置x轴图例为空值
10 ax1.set_ylabel('kind')
11
12 sns.heatmap(pt, linewidths = 0.05, ax = ax2, cmap=cmap, center=None, robust=
       True )
13 # If True and vmin or vmax are absent, the colormap range is computed with
       robust quantiles instead of the extreme values.
14 ax2.set_title('robust=True')
15 ax2.set_xlabel('region')
16 ax2.set_ylabel('kind')
17
18 f.savefig('sns_heatmap_robust.jpg', bbox_inches='tight')
```

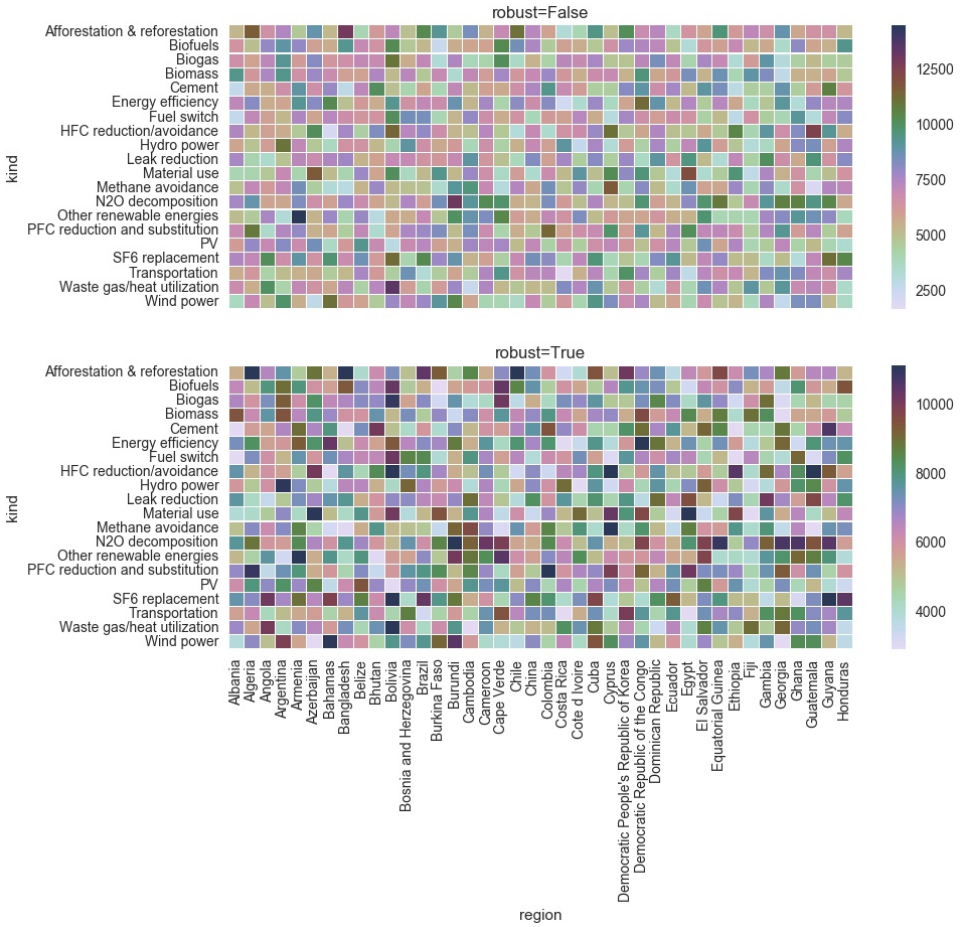


Figure 29.6: img-robust

29.2.4 mask

```
f, (ax1,ax2) = plt.subplots(figsize = (10, 8),nrows=2)

cmap = sns.cubehelix_palette(start = 1.5, rot = 3, gamma=0.8, as_cmap = True)

p1 = sns.heatmap(pt, linewidths = 0.05,ax=ax1, vmax=15000, vmin=0, cmap=cmap,
    center=None, robust=False, mask=None )
# robust默认为False
ax1.set_title('mask=None')
ax1.set_xlabel('')
ax1.set_xticklabels([]) #设置x轴图例为空值
ax1.set_ylabel('kind')

p2 = sns.heatmap(pt, linewidths = 0.05, ax=ax2, vmax=15000, vmin=0, cmap=cmap,
    center=None, robust=False, annot=False,mask=pt<10000 )
# mask: boolean array or DataFrame

ax2.set_title('mask: boolean DataFrame')
ax2.set_xlabel('region')
ax2.set_ylabel('kind')

f.savefig('sns_heatmap_mask.jpg', bbox_inches='tight')
```

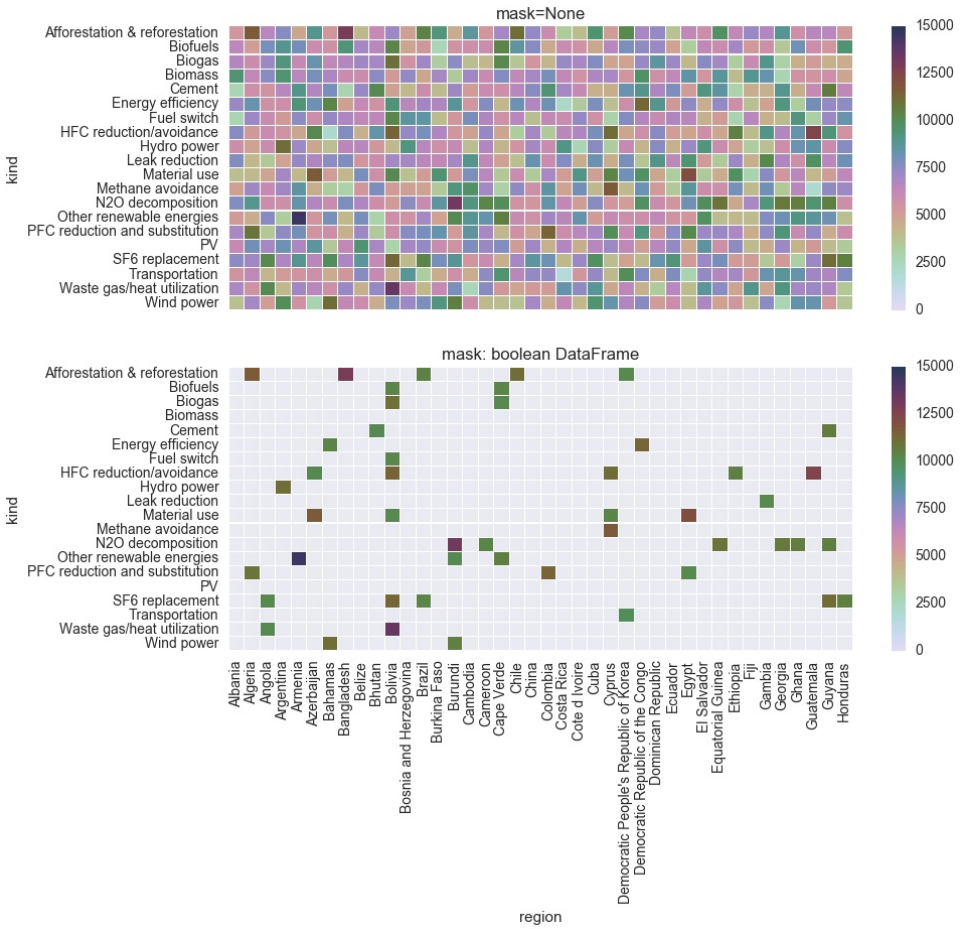


Figure 29.7: img-mask

29.2.5 xticklabels, yticklabels

- xticklabels: 如果是 True，则绘制 dataframe 的列名。如果是 False，则不绘制列名。如果是列表，则绘制列表中的内容作为 xticklabels。如果是整数 n，则绘制列名，但每个 n 绘制一个 label。默认为 True。
- yticklabels: 如果是 True，则绘制 dataframe 的行名。如果是 False，则不绘制行名。如果是列表，则绘制列表中的内容作为 yticklabels。如果是整数 n，则绘制列名，但每个 n 绘制一个 label。默认为 True。默认为 True。

```
f, (ax1,ax2) = plt.subplots(figsize = (10, 8),nrows=2)

cmap = sns.cubehelix_palette(start = 1.5, rot = 3, gamma=0.8, as_cmap = True)

p1 = sns.heatmap(pt, linewidths = 0.05,ax=ax1, vmax=15000, vmin=0, cmap=cmap,
    center=None, robust=False, mask=None, xticklabels=False )
# robust默认为False
ax1.set_title('xticklabels=None')
```

```
ax1.set_xlabel('')
# ax1.set_xticklabels([]) #设置x轴图例为空值
ax1.set_ylabel('kind')

p2 = sns.heatmap(pt, linewidths = 0.05, ax=ax2, vmax=15000, vmin=0, cmap=cmap,
    center=None, robust=False, annot=False,mask=None,xticklabels=3, yticklabels=
    list(range(20)) )
# mask: boolean array or DataFrame

ax2.set_title('xticklabels=3, yticklabels is a list')
ax2.set_xlabel('region')
ax2.set_ylabel('kind')

f.savefig('sns_heatmap_xyticklabels.jpg', bbox_inches='tight')
```

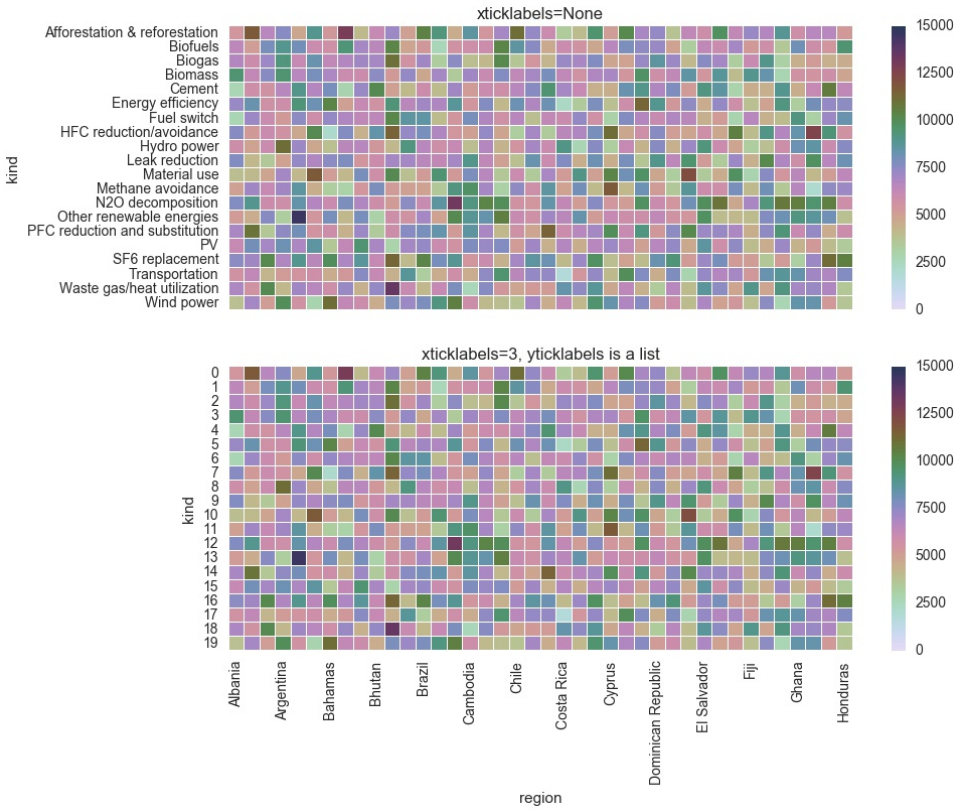


Figure 29.8: img-ticklabels

29.2.6 annot

- annotate 的缩写，annot 默认为 False，当 annot 为 True 时，在 heatmap 中每个方格写入数据
- annot_kws，当 annot 为 True 时，可设置各个参数，包括大小，颜色，加粗，斜体字等

```
np.random.seed(0)
x = np.random.randn(10, 10)
f, (ax1, ax2) = plt.subplots(figsize=(8,8),nrows=2)

sns.heatmap(x, annot=True, ax=ax1)
sns.heatmap(x, annot=True, ax=ax2, annot_kws={'size':9,'weight':'bold', 'color'
    :'blue'})
# Keyword arguments for ax.text when annot is True.
# http://stackoverflow.com/questions/35024475/seaborn-heatmap-key-words

f.savefig('sns_heatmap_annot.jpg')
```

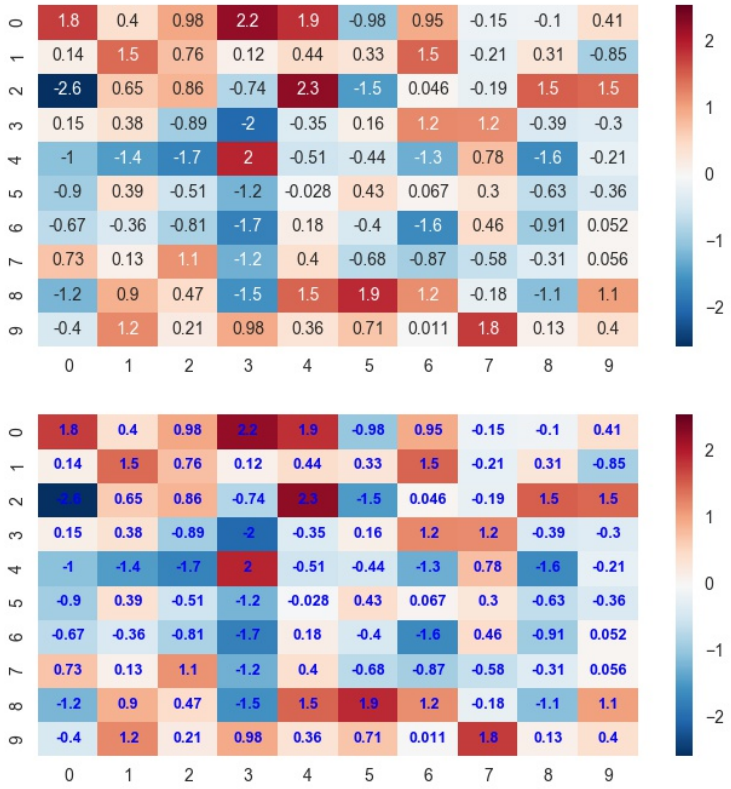


Figure 29.9: img-annot

关于 annot_kws 的设置，还有很多值得研究的地方，ax.text 有很多属性，有兴趣的可以去研究下；

ax.text 可参考官方文档：http://matplotlib.org/api/text_api.html#matplotlib.text.Text

29.2.7 fmt

- fmt，格式设置

```
np.random.seed(0)
x = np.random.randn(10, 10)
f, (ax1, ax2) = plt.subplots(figsize=(8,8),nrows=2)

sns.heatmap(x, annot=True, ax=ax1)
sns.heatmap(x, annot=True, fmt='.1f', ax=ax2)

f.savefig('sns_heatmap_fmt.jpg')
```

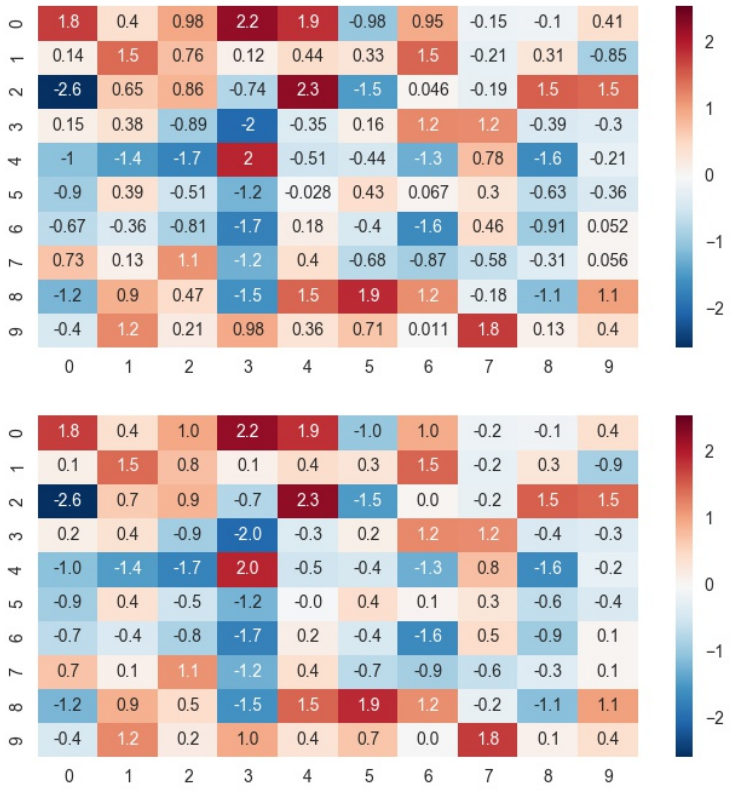


Figure 29.10: img-fmt

29.3 案例应用：突出显示某些数据

29.3.1 method 1：利用 mask 来实现

```
f,ax=plt.subplots(figsize=(10,5))

x = np.random.randn(10, 10)
sns.heatmap(x, annot=True, ax=ax)
sns.heatmap(x, mask=x < 1, cbar=False, ax=ax,
            annot=True, annot_kws={"weight": "bold"})

f.savefig('sns_heatmap_eg1.jpg')
```

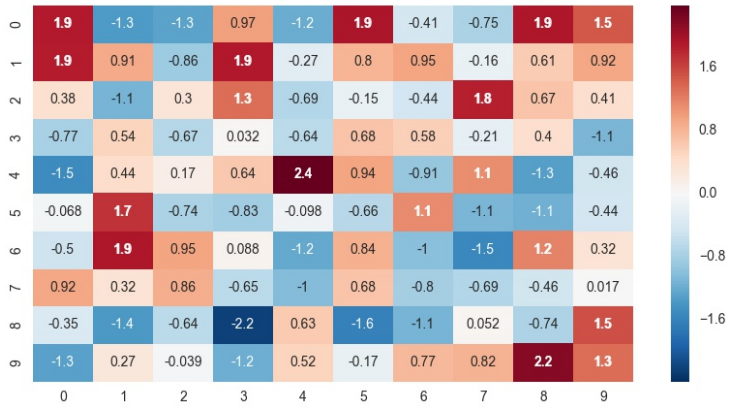


Figure 29.11: img-mask

29.3.2 method 2：利用 ax.texts 来实现

```
f,ax=plt.subplots(figsize=(10,5))

flights = sns.load_dataset("flights")
flights = flights.pivot("month", "year", "passengers")
pic = sns.heatmap(flights, annot=True, fmt="d", ax=ax)

for text in pic.texts:
    text.set_size(8)
    if text.get_text() == '118':
        text.set_size(12)
        text.set_weight('bold')
        text.set_style('italic')

f.savefig('sns_heatmap_eg2.jpg')
```

Figure 29.12: img-eg2

> 你可能会发现本文中 seaborn 的 heatmap 中还有些参数没有进行介绍，介于篇幅，这里就不在啰嗦了，建议各位小伙伴自己可以研究下其他参数如何使用。

30 Bokeh 入门

一直觉得 Bokeh 的图很漂亮，今天我们来开始用 bokeh 制图，本文是开篇，希望大家喜欢 ~

本次运行环境为：

- win7
- jupyter notebook
- python 3.6
- bokeh 0.13.0

30.1 常规步骤

bokeh 中绘图有两类主要的接口，bokeh.plotting 和 bokeh.models ，其中 plotting 是高级接口，一般情况下我们使用 bokeh,plotting 较多。下面，我们介绍的内容，主要是针对 bokeh.plotting

在 bokeh 中绘制图表的常规步骤：

1. 加载 bokeh 库，声明在 notebook 或 html 文件中显示或输出绘制的图表
2. 绘制图表框架 figure()
3. 在 figure 上绘制具体的图形，比如 circle，line，bar 等
4. 显示图片，show()

30.2 绘制 circle()

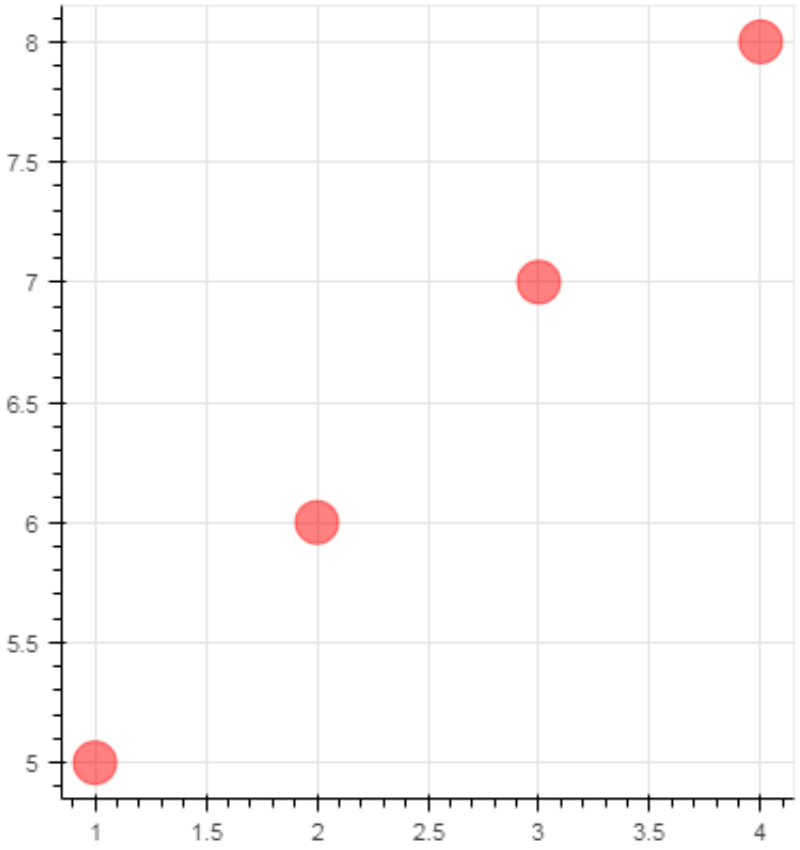
首先，在绘制图表框架 figure() 时，可以设置图表大小（plot_wihth,plot_height），也可以设置图表框架旁边所用的工具 (tools)

- 工具可以是以下 value，可以根据实际情况来选择合作的工具使用。
- “crosshair,pan,wheel_zoom,box_zoom,reset,box_select,lasso_select,save”

```
from bokeh.plotting import output_notebook, figure, show
import pandas as pd
import numpy as np

output_notebook()

# tools="pan,box_zoom,wheel_zoom,reset,save"

```

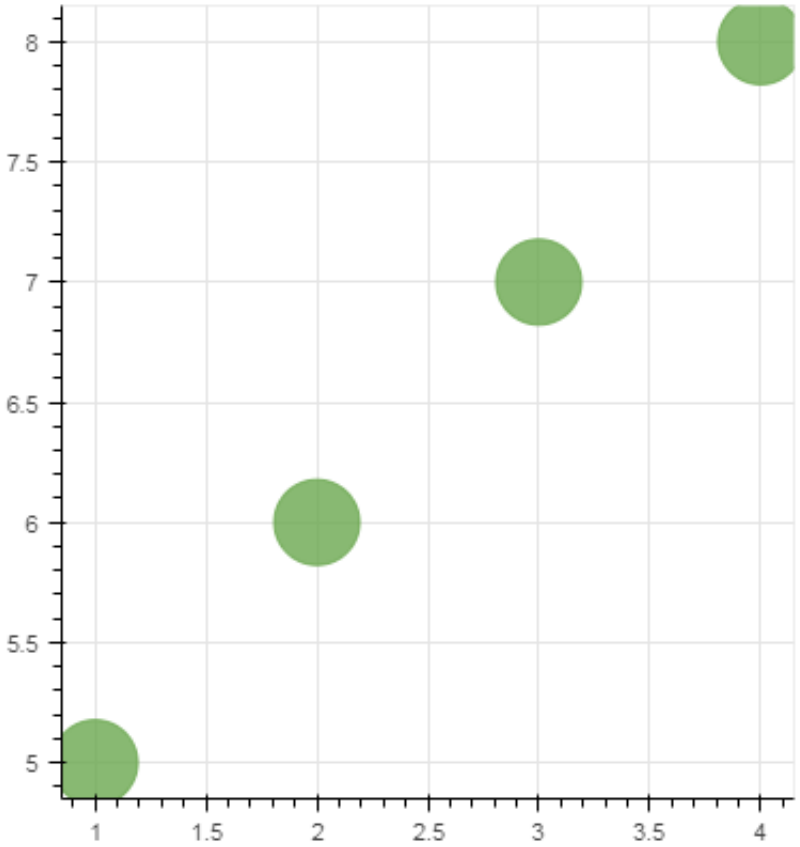
```
# 工具可以是以下 value，可以根据实际情况来选择合作的工具使用
tools = "crosshair,pan,wheel_zoom,box_zoom,reset,box_select,lasso_select,save"

p = figure(plot_width=400, plot_height = 400, tools=tools)

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

其中，参数：alpha，数值越小，透明度越高



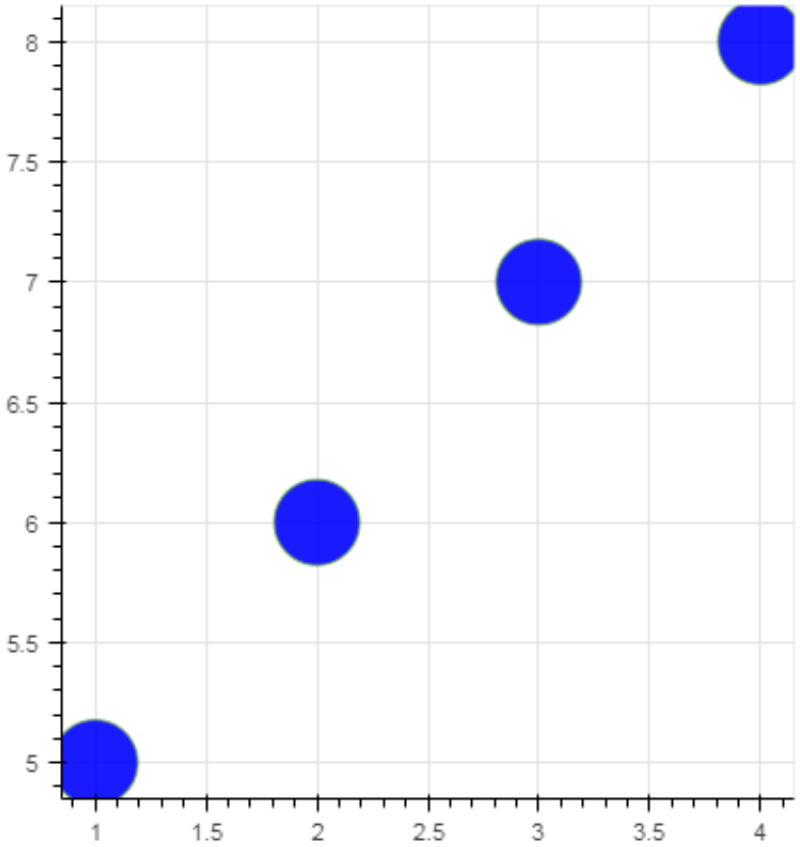
参数 angle, 对于 circle 而言，angle 参数似乎没有用……（因为本身就是圆的，旋转角度没有影响）

```
p = figure(plot_width=400, plot_height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=40, color='#6aa84f', alpha=0.8, angle=0.8)

show(p)
```



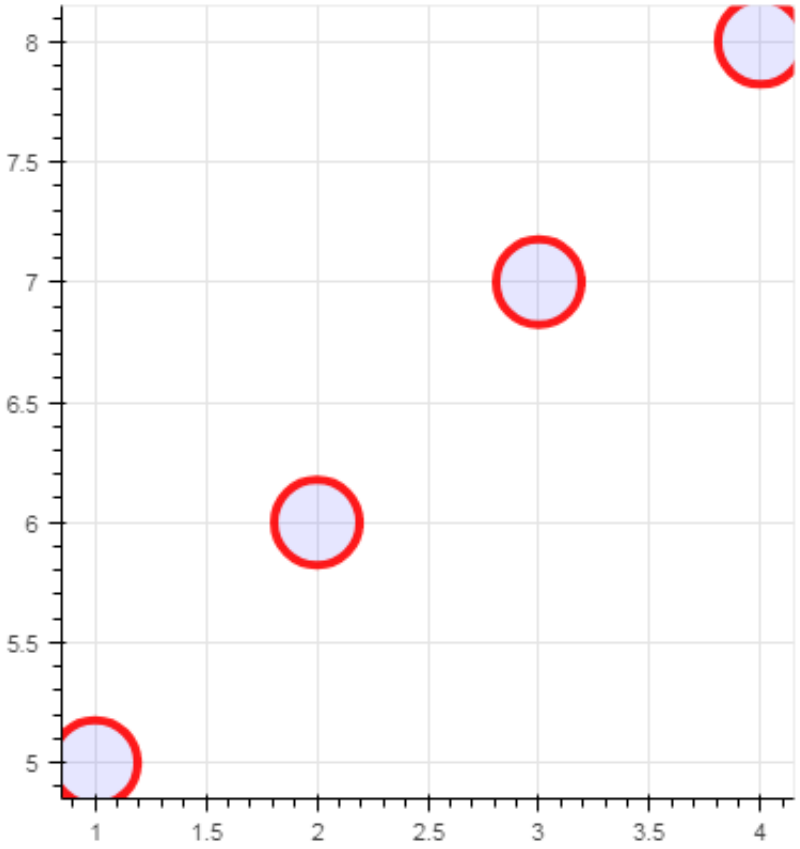
fill_color, fill_alpha, 这两个参数用来设置圆体的填充颜色和透明度

```
p = figure(plot_width=400, plot_height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=40, color='#6aa84f', alpha=0.8,
         angle=0.8,fill_alpha=0.9, fill_color='blue')

show(p)
```



- line_alpha 线条透明度
- line_color 线条颜色
- line_cap 的参数似乎没有效果，line_cap 的值可以是 (butt, round, square)
- line_width, 圆圈边缘线条的宽度

```
p = figure(plot_width=400, plot_height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=40, color='#6aa84f', alpha=0.8,
         angle=0.8,fill_alpha=0.1, fill_color='blue',
        line_color='red', line_alpha=0.9, line_width=4,
        line_cap='square')

# LineCap = Enumeration(butt, round, square)
# Specify how stroked lines should be terminated

show(p)
```

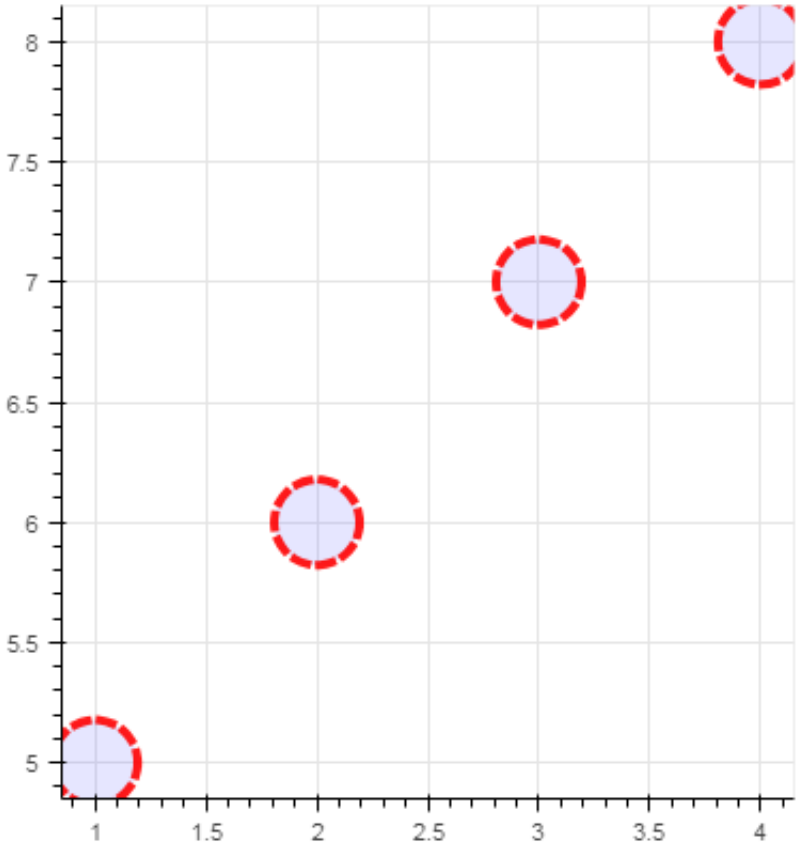


line_dash

官方描述如下

> Accept line dash specifications.
>
> Express patterns that describe line dashes. DashPattern values can be specified in a variety of ways:
>
> An enum: “solid”, “dashed”, “dotted”, “dotdash”, “dashdot” a tuple or list of integers in the HTML5 Canvas dash specification style. Note that if the list of integers has an odd number of elements, then it is duplicated, and that duplicated list becomes the new dash list. To indicate that dashing is turned off (solid lines), specify the empty list [].

```
p = figure(plot_width=400, plot_height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=40, color='#6aa84f', alpha=0.8,
         angle=0.8,fill_alpha=0.1, fill_color='blue',
        line_color='red', line_alpha=0.9, line_width=4,
        line_cap='square', line_dash='dashed')

show(p)
```



当然，`circle()` 中还有其他的参数可以设置，有兴趣的童鞋不妨自行摸索下。

> 对我的文章感兴趣的朋友，可以关注我的微信公众号『Python 数据之道』（ID：PyDataLab），接收我的更新通知。

31 Bokeh：figure 详细解读

figure 在 bokeh 绘图中有着重要的作用，今天我们来开始了解 figure 的用法，希望对大家有所帮助。

本次运行环境为：

- win7
- jupyter notebook
- python 3.6
- bokeh 0.13.0

31.1 Bokeh 中绘图的一般步骤

1. 加载 bokeh 库，声明在 notebook 或 html 文件中显示或输出绘制的图表
2. 绘制图表框架 figure()
3. 在 figure 上绘制具体的图形，比如 circle，line，bar 等
4. 显示图片，show()

本文主要来介绍 figure() 的基本用法。

首先，加载 bokeh 库

```
from bokeh.plotting import figure, output_notebook, show

output_notebook()
```

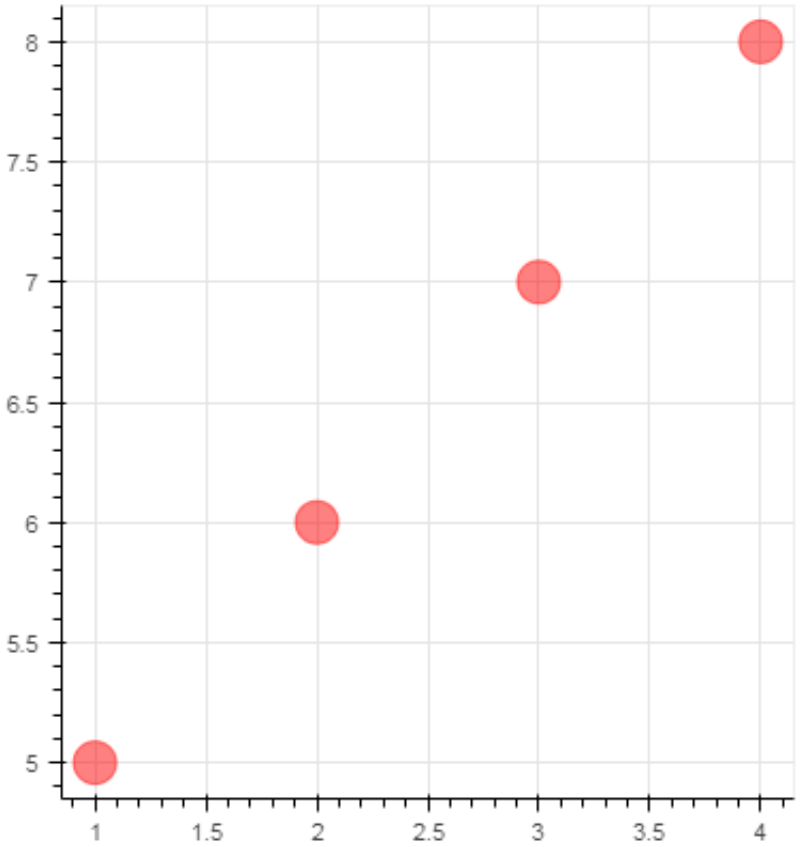
31.1.1 plot_width, plot_weight

plot_width, plot_weight 可以设置绘图区的宽度和高度。

设置参数值，如下：

```
p = figure(plot_width=400, plot_height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：



width, weight

width, weight 也可以设置绘图区的宽度和高度。

查看官方文档，用的是 plot_width 和 plot_weight 两个参数，但实际上 width 和 height 也有同样的效果，大家不妨探索下。

代码如下：

```
p = figure(width=400, height = 400)

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

31.1.2 tools

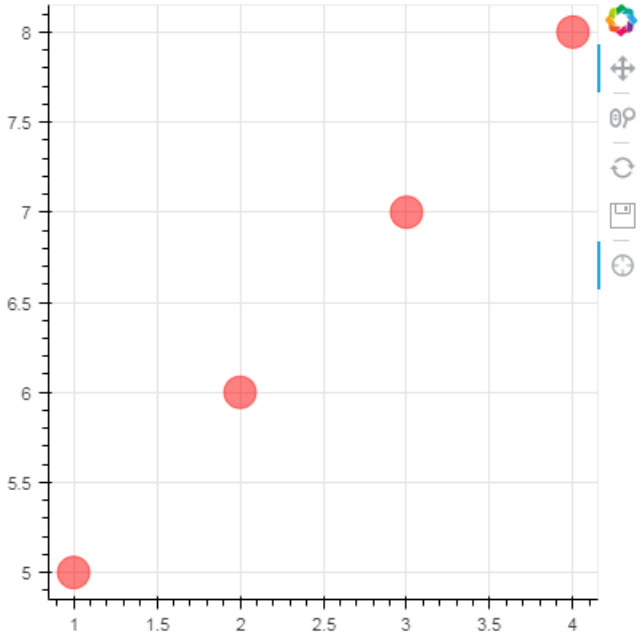
工具可以是以下 value，可以根据实际情况来选择合作的工具使用。

“crosshair,pan,wheel_zoom,box_zoom,reset,box_select,lasso_select,save,help”

```
# 工具可以是以下 value，可以根据实际情况来选择合作的工具使用
# tools = "crosshair,pan,wheel_zoom,box_zoom,reset,box_select,lasso_select,save
    ,help"
```

```
3  tools = "crosshair,pan,wheel_zoom,reset,save"
4
5
6  p = figure(width=400, height = 400, tools=tools)
7
8  p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)
9
10 show(p)
```

图示如下：

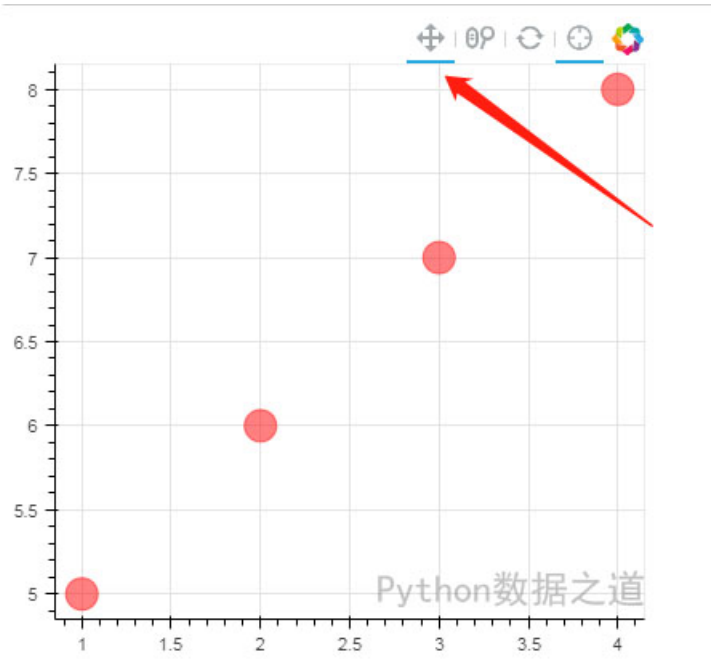


31.1.3 toolbar_location

toolbar_location，设置工具栏显示的位置，参数值可以是：“below, above, left, right”，默认值应该是“auto”

```
1  tools = "crosshair,pan,wheel_zoom,reset"
2
3
4  p = figure(width=400, height = 400, tools=tools, toolbar_location='above')
5
6  p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)
7
8  show(p)
```

图示如下：



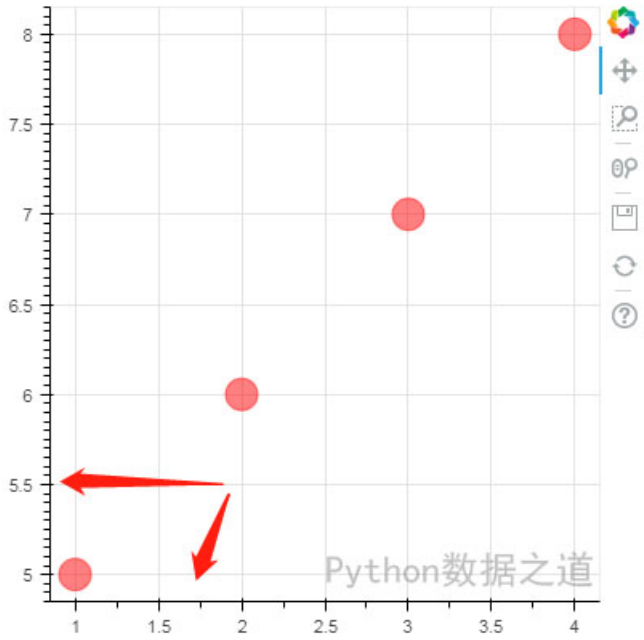
31.1.4 x_minor_ticks, y_minor_ticks

x_minor_ticks, y_minor_ticks 默认值是 “auto”，其他值可以是大于 1 的整数

示例如下：

```
p = figure(width=400, height = 400,
           x_minor_ticks=2, y_minor_ticks=10)

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：



31.1.5 x_range, y_range

x_range, y_range 可以是 list 或 tuple 表示范围的形式数值

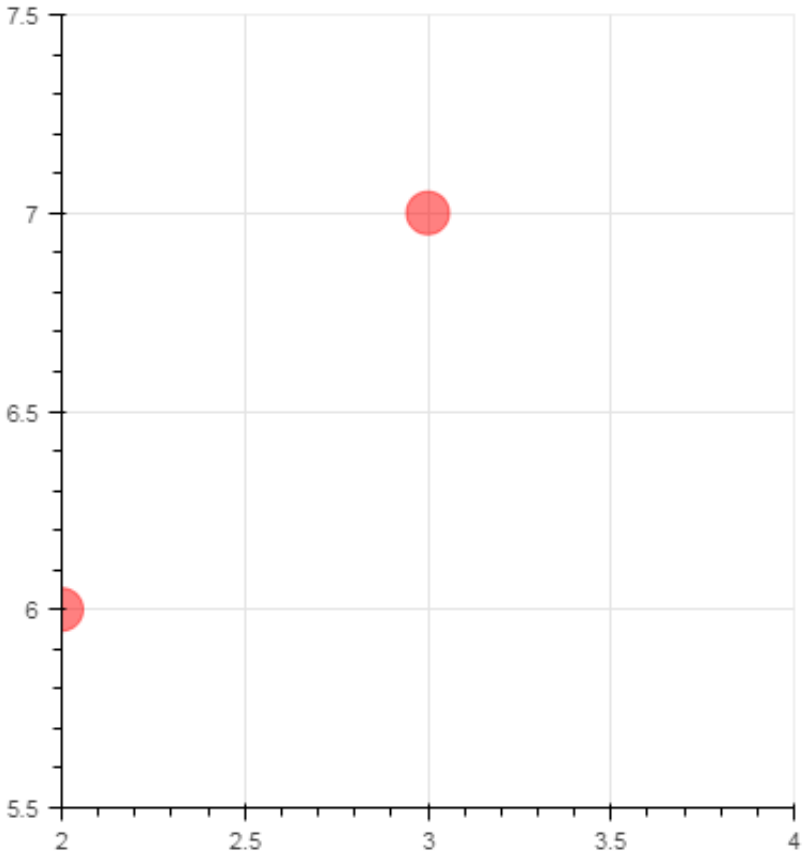
示例如下：

```
p = figure(width=400, height = 400, x_range=[2,4], y_range=[5.5, 7.5])

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

或者

```
p = figure(width=400, height = 400, x_range=(2,4), y_range=(5.5, 7.5))

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：



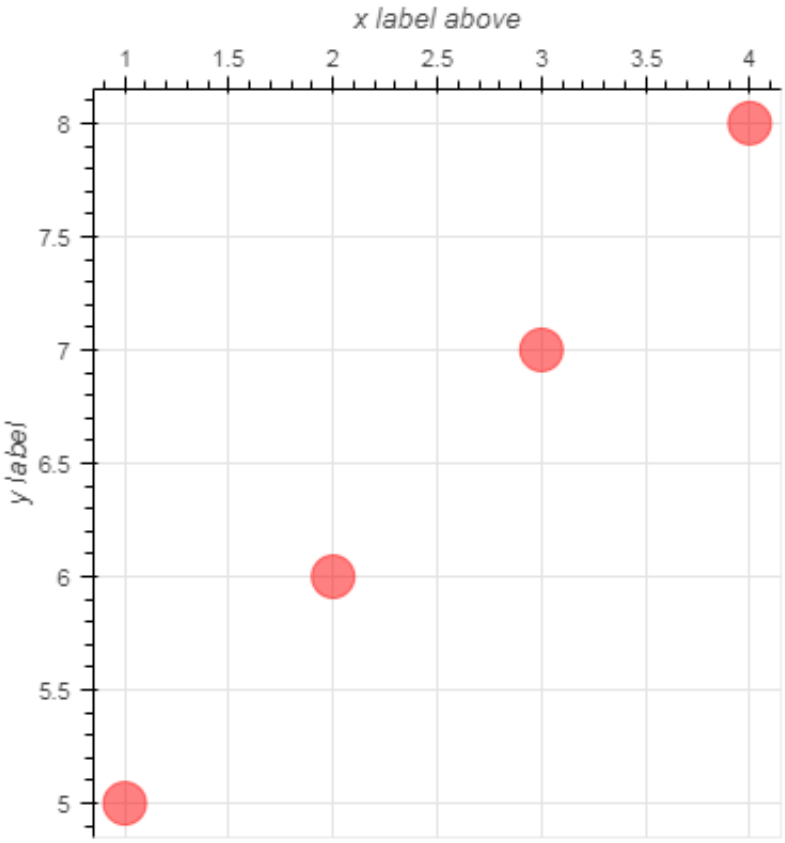
31.1.6 x_axis_label, x_axis_location

x_axis_label, 设置 x 轴的名称

x_axis_location, 设置 x 轴的位置，有两个值，在上面显示（“above”）和在下面显示（“below”），默认值是“below”

```
p = figure(width=400, height = 400,
           x_axis_label='x label above', x_axis_location='above',
           y_axis_label='y label')

# x_axis_location, default value: "below", values are [ above, below]

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：



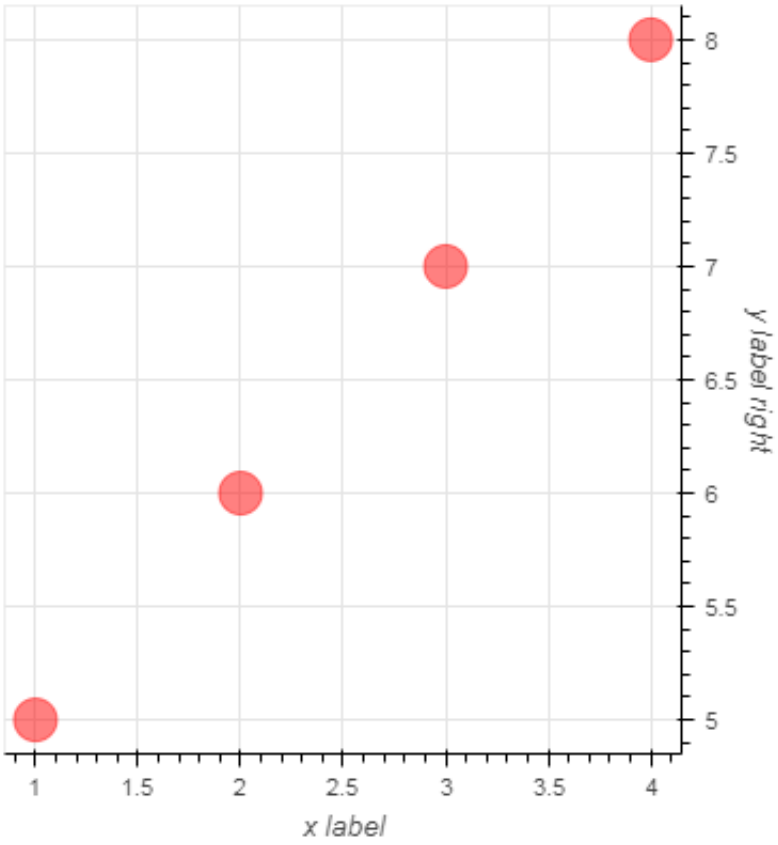
31.1.7 y_axis_label, y_axis_location

y_axis_label, 设置 y 轴的名称

y_axis_location, 设置 y 轴的位置，有两个值，在左边显示（“left”）和在右边显示（“right”），默认值是 “left”

```
p = figure(width=400, height = 400,
           x_axis_label='x label', x_axis_location='below',
           y_axis_label='y label right', y_axis_location='right')

# x_axis_location, default value: "below", values are [ above, below]
# y_axis_location, default value: "left", values are [ left, right]

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：



31.1.8 x_aixs_type，y_axis_type

(1) x_aixs_type: “datetime”

若是时间数据，可以将 x 轴或者 y 轴的数值设置为日期形式

```
import pandas as pd

rng = pd.date_range('2018-1-1', periods=7)

rng
```

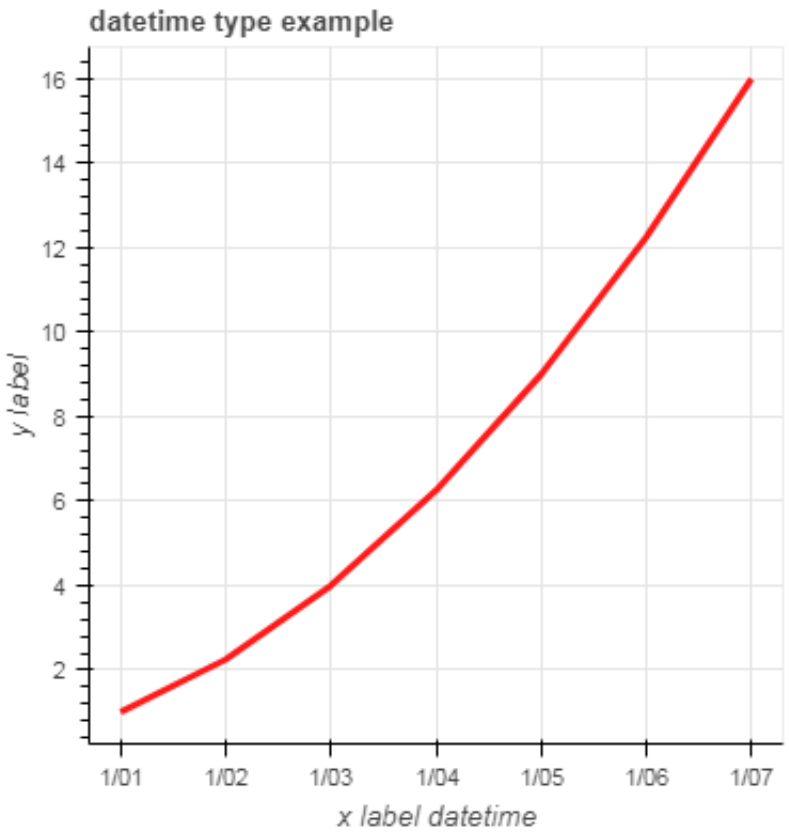
out:

```
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
               '2018-01-05', '2018-01-06', '2018-01-07'],
              dtype='datetime64[ns]', freq='D')
```

```
x = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
# y = [10**(i**2) for i in x]
y = [i**2 for i in x]

p = figure(width=400, height = 400, title='datetime type example',
```

```
            x_axis_label='x label datetime', x_axis_location='below',
            y_axis_label='y label', y_axis_location='left',
            y_axis_type='linear', x_axis_type='datetime')
# y_axis_type, "auto" 和 "linear" 的效果是一样的

p.line(rng,y,color='red',line_width=3, alpha=0.9)

show(p)
```

图示如下：



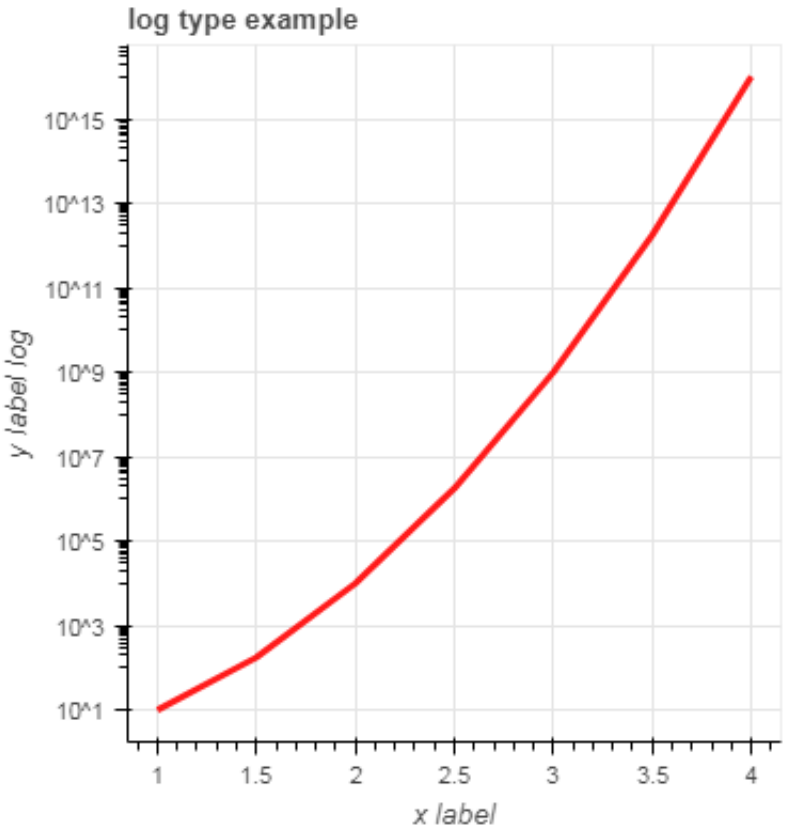
(2) y_aixs_type: “log”

有时候，我们需要将 x 或 y 轴的数据设置成对数（log）形式，如下：

```
x = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
y = [10**(i**2) for i in x]

p = figure(width=400, height = 400, title='log type example',
            x_axis_label='x label', x_axis_location='below',
            y_axis_label='y label log', y_axis_location='left',
            y_axis_type='log')

p.line(x,y,color='red',line_width=3, alpha=0.9)
```

```
10
11 show(p)
```

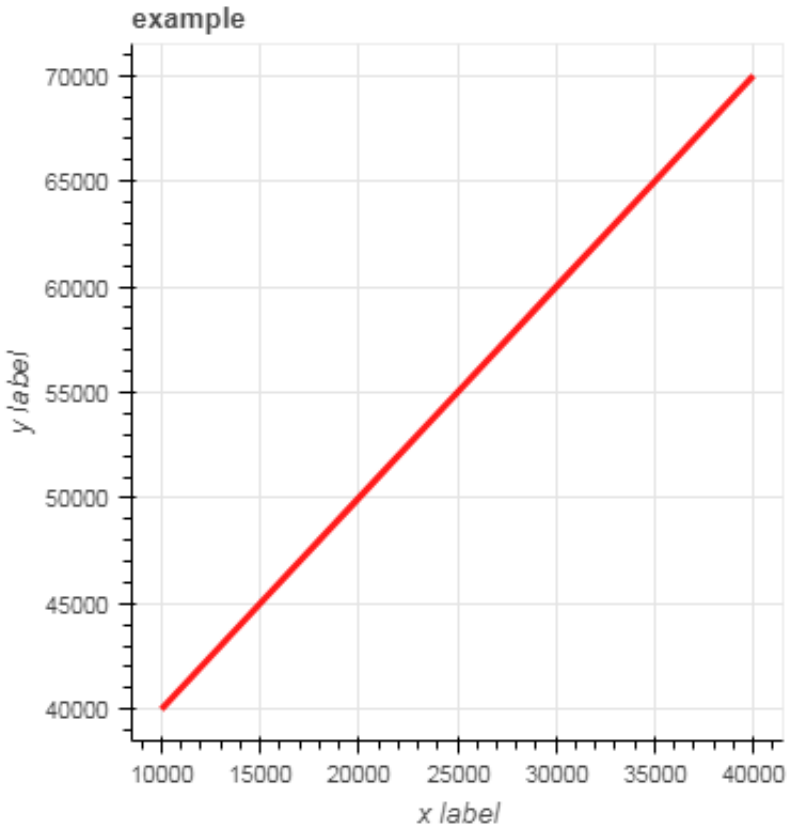
图示如下：



（3）x_axis_type, y_aixs_type: “mercator”

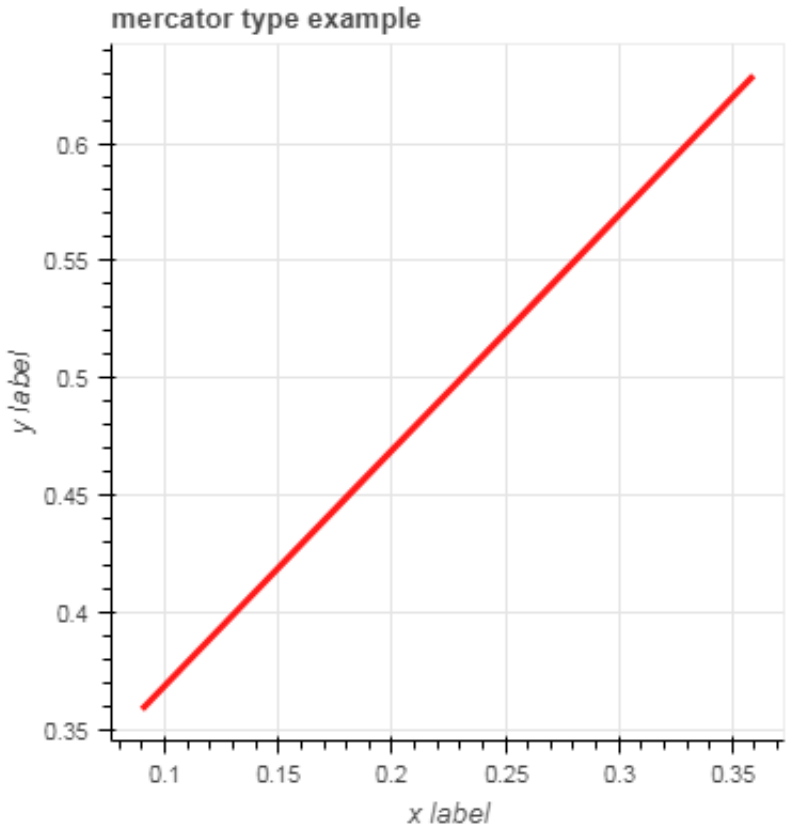
```
1 k = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
2 x = [i*10000 for i in k]
3 y = [i*10000+30000 for i in k]
4
5 p = figure(width=400, height = 400, title='example',
6            x_axis_label='x label', x_axis_location='below',
7            y_axis_label='y label', y_axis_location='left' )
8
9 p.line(x,y,color='red',line_width=3, alpha=0.9)
10
11 show(p)
```

没有设置 mercator 的图示如下：




```
k = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
x = [i*10000 for i in k]
y = [i*10000+30000 for i in k]

p = figure(width=400, height = 400, title='mercator type example',
           x_axis_label='x label', x_axis_location='below',
           y_axis_label='y label', y_axis_location='left',
           x_axis_type='mercator', y_axis_type='mercator')

p.line(x,y,color='red',line_width=3, alpha=0.9)

show(p)
```

设置为 mercator 后的图示如下：



从上图的结果来看，mercator 好像代表的是数值 10 万，简写为 1。

当然，我不知道我的这个理解是否准确。

关于 mercator，我暂时没有看懂是什么用途，在网上查到了 mercator projection。

各位有兴趣的可以探索下。

麦卡托投影法 Mercator projection * https://en.wikipedia.org/wiki/Mercator_projection * https://zh.wikipedia.org/wiki/%E9%BA%A5%E5%8D%A1%E6%89%98%E6%8A%95%E5%BD%B1%E6%B3%95

31.2 综合小结

最后，我们来小结下前面所述内容。

figure 的部分参数：* plot_width, plot_weight 或 width, height，设置绘图区的长度和宽度 * tools，设置绘图区旁边所用的工具 (tools)，其值可以是 “crosshair,pan,wheel_zoom,box_zoom,reset,box_select,lasso_select,save,help” * toolbar_location，设置工具栏显示的位置，参数值可以是：“below, above, left, right”，默认值 “right” * x_minor_ticks, y_minor_ticks 默认值是 “auto”，其他值可以是大于 1 的整数 * x_range, y_range 可以是 list 或 tuple 表示范围的形式数值 * x_axis_label, y_axis_label, 设置 x 轴和 y 轴的名称 * x_axis_location, default value: “below”, values are [above, below] * y_axis_location, default value: “left”, values are [left,

right] * x_aixs_type，y_axis_type，x 和 y 轴数据的表现形式，其值可以是” linear, log, datetime, mercator“, 默认是”auto”

当然，还有一些参数没有提到，各位可以自行研究下。

32 Bokeh: 29 种基础可视化图形

前面，我们分享了关于 bokeh 入门和 figure 使用的内容。今天，我们在前文的基础上，主要来分享 29 种基本的图形绘制方法，很多复杂的图形，都是基于这些基础图形的组合，所以，这些基础图形是我们进阶路上的必由之路。

下面，我们一起来看看都有哪些基础图形吧。

本次运行环境为：* win7 * jupyter notebook * python 3.6 * bokeh 0.13.0

32.1 Bokeh 中绘图的一般步骤

1. 加载 bokeh 库，声明在 notebook 或 html 文件中显示或输出绘制的图表
2. 绘制图表框架 figure()
3. 在 figure 上绘制具体的图形，比如 circle，line，bar 等
4. 显示图片，show()

本文主要来介绍 29 种基础图形的绘制用法。

首先，加载 bokeh 库，以及准备基础数据

```
from bokeh.plotting import figure, output_notebook, show
from bokeh.layouts import gridplot
import numpy as np

output_notebook()

np.random.seed(15)

x=np.random.randint(1,20,size=6)
y=np.random.randint(20,50,size=6)

print(x)
print(y)
```

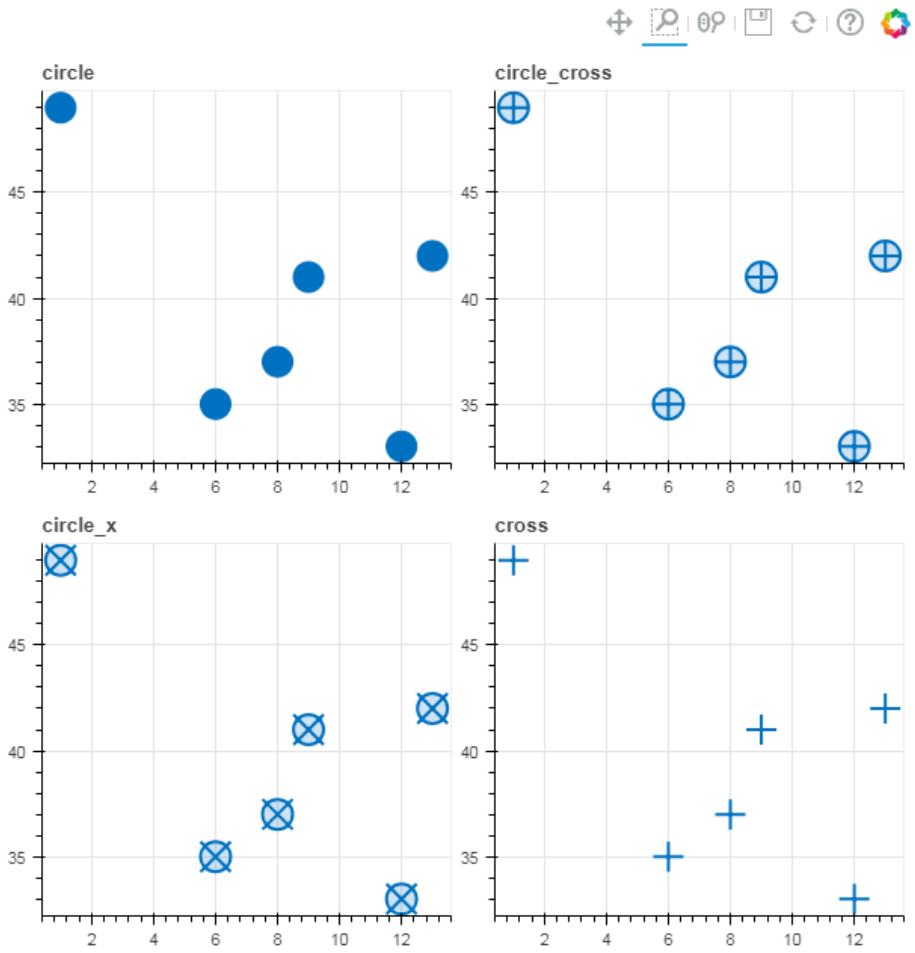
32.1.1 circle, circle_cross, circle_x, cross

首先，我们来看看一组关于圆形及其相关的图形

```
p1 = figure(title='circle')
p1.circle(x,y,size=20, color='#0071c1')

```

```
4  p2 = figure(title='circle_cross')
5  p2.circle_cross(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)
6
7  p3 = figure(title='circle_x')
8  p3.circle_x(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)
9
10 p4 = figure(title='cross')
11 p4.cross(x,y,size=20, color='#0071c1', line_width=2)
12
13 grid=gridplot([p1,p2,p3,p4],ncols=2, plot_width=300,plot_height=300)
14
15 show(grid)
```

图示如下：

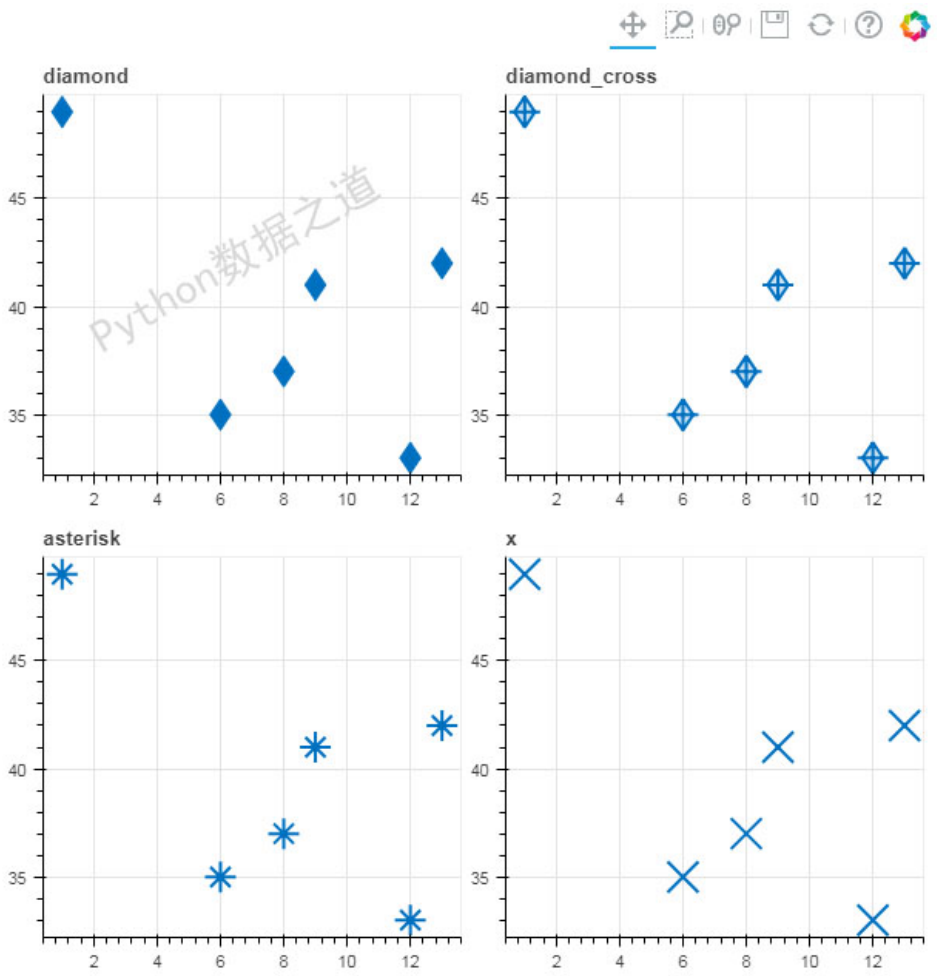


32.1.2 diamond, diamond_cross, asterisk, x

这是一组跟钻石还有星型符号相关的图形绘制，如下：

```
p1 = figure(title='diamond')
p1.diamond(x,y,size=20, color='#0071c1')

p2 = figure(title='diamond_cross')
p2.diamond_cross(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p3 = figure(title='asterisk')
p3.asterisk(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p4 = figure(title='x')
p4.x(x,y,size=20, color='#0071c1', line_width=2)


grid=gridplot([p1,p2,p3,p4],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

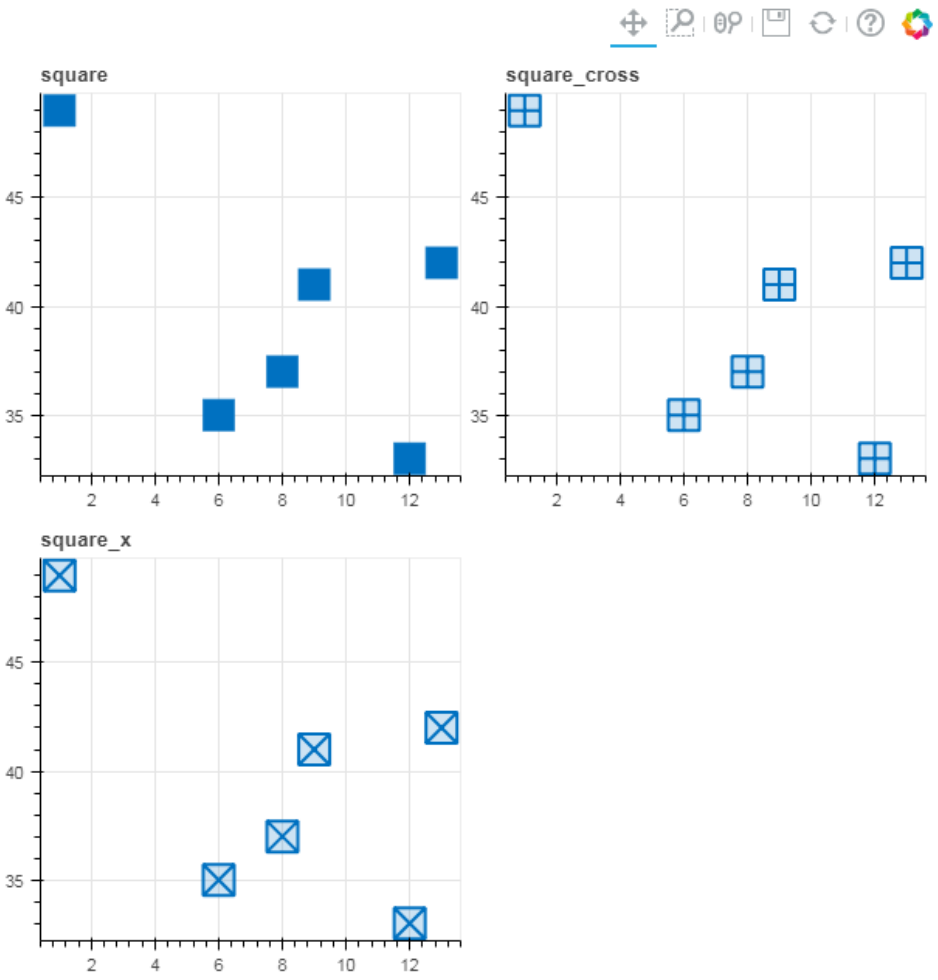


32.1.3 square, square_cross, square_x

这里，还有跟方块相关的一组图形

```
p1 = figure(title='square')
p1.square(x,y,size=20, color='#0071c1')

p2 = figure(title='square_cross')
p2.square_cross(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p3 = figure(title='square_x')
p3.square_x(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)


grid=gridplot([p1,p2,p3],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

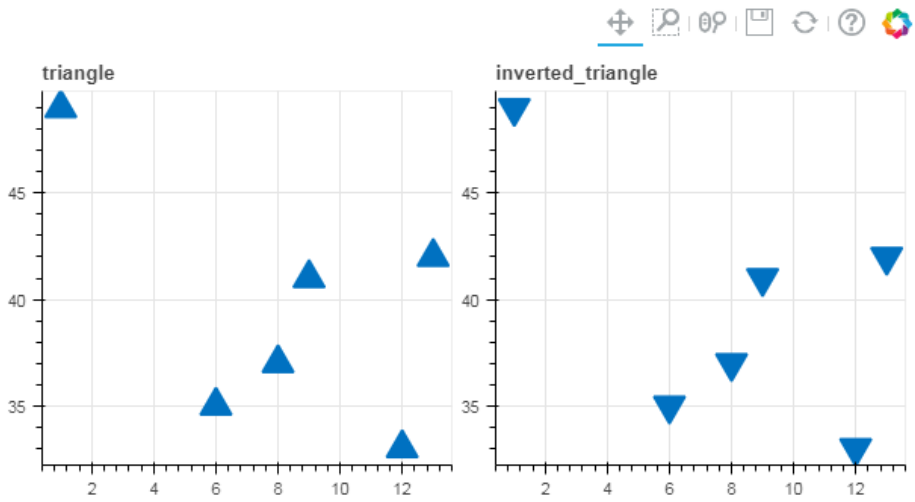


32.1.4 triangle, inverted_triangle

接下来，是三角形的绘制

```
p3 = figure(title='triangle')
p3.triangle(x,y,size=20, color='#0071c1', line_width=2)

p4 = figure(title='inverted_triangle')
p4.inverted_triangle(x,y,size=20, color='#0071c1', line_width=2)

grid=gridplot([p3,p4],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：



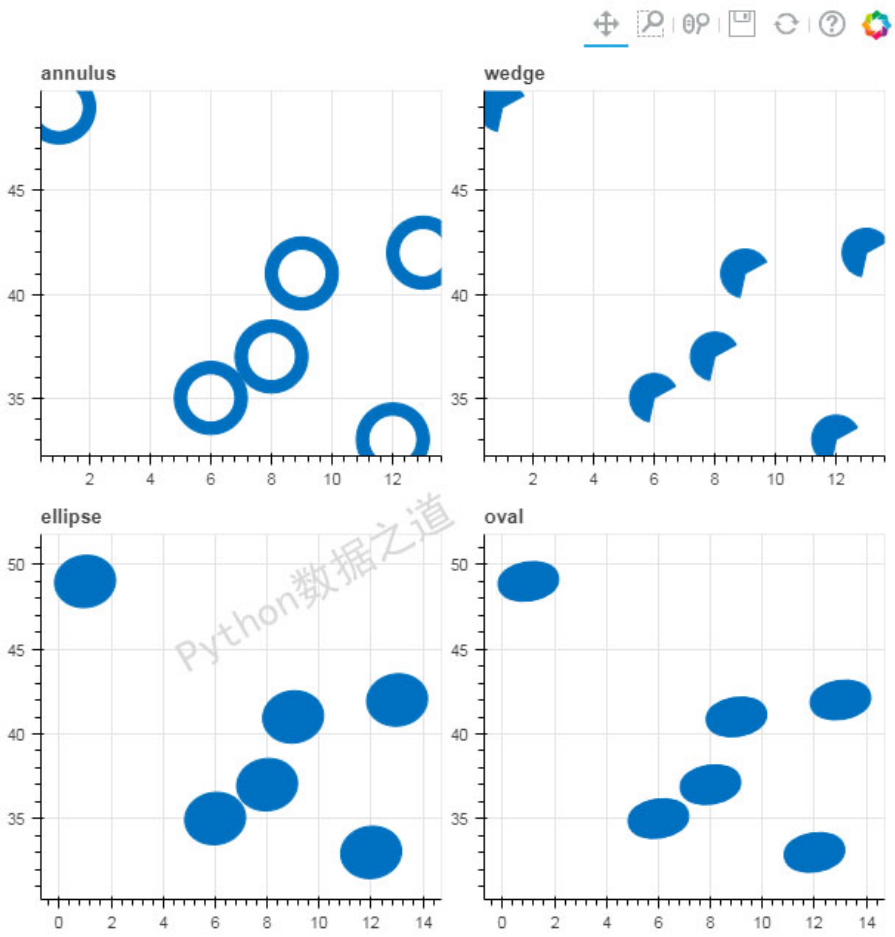
32.1.5 annulus, ellipse, wedge, oval

- annulus, 绘制圆环
- wedge, 楔形图
- ellipse, 绘制椭圆
- oval，绘制椭圆

```
p1 = figure(title='annulus')
p1.annulus(x,y,color='#0071c1', inner_radius=0.8, outer_radius=1.2)

p2 = figure(title='wedge')
p2.wedge(x,y,color='#0071c1', radius=0.8, start_angle=0.5, end_angle=4.5,
    direction='anticlock')

p3 = figure(title='ellipse')
p3.ellipse(x,y,color='#0071c1', width=2, height=3.6, angle=30)
```

```
9
10 p4 = figure(title='oval')
11 p4.oval(x,y,color='#0071c1', width=2, height=3.6, angle=30)
12
13
14 grid=gridplot([p1,p2,p3,p4],ncols=2, plot_width=300,plot_height=300)
15
16 show(grid)
```

图示如下：



从上图来看，同样的数据源，ellipse 和 oval 绘制出来的椭圆形状，看起来是有些差异的。有兴趣的同学可以研究下其中的原因。

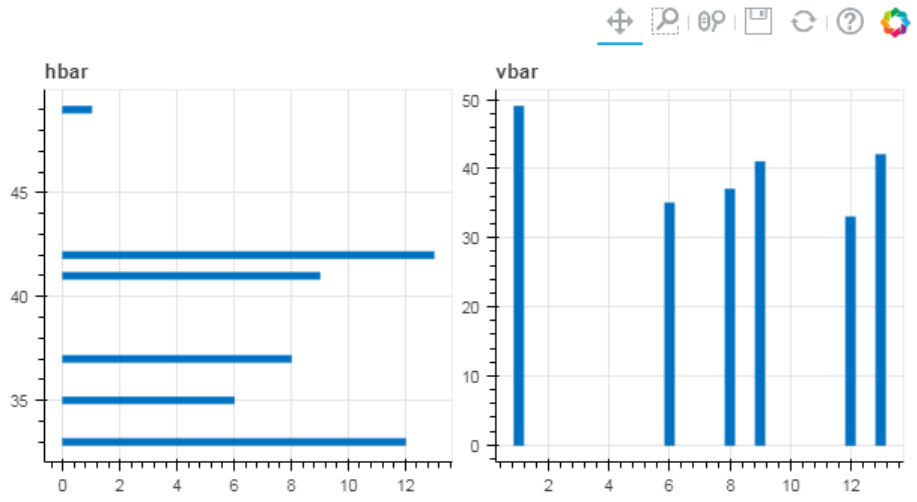
32.1.6 hbar, vbar

绘制柱状图，分为水平柱状图和垂直柱状图，这个也是平时我们用的比较多的类型之一。

```
1 p2 = figure(title='hbar')
2 p2.hbar(y=y, height=0.3, left=0, right=x, color='#0071c1')
```

```
3
4  p3 = figure(title='vbar')
5  p3.vbar(x=x, width=0.3, bottom=0, top=y, color='#0071c1')
6
7  grid=gridplot([p2,p3],ncols=2, plot_width=300,plot_height=300)
8
9  show(grid)
```

图示如下：

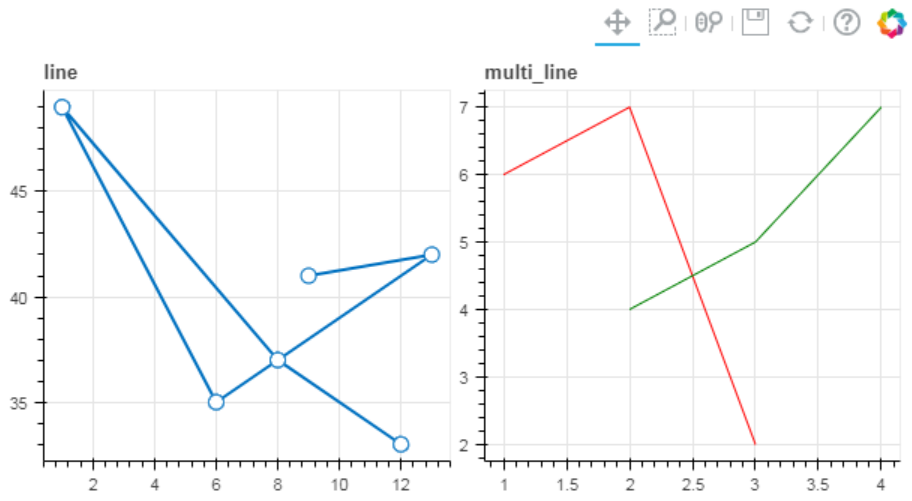


32.1.7 line, multi_line

线状图或者称为折线图，单独只绘制线条时，可能视觉效果没有那么好看，有时候会配合圆形、方形等上面描述过的图形一起使用。

```
1  p1 = figure(title='line')
2  p1.line(x,y,color='#0071c1', line_width=2)
3  p1.circle(x,y,size=10,color='#0071c1',fill_color='white')
4
5  from bokeh.plotting import figure, output_file, show
6
7  p2 = figure(title='multi_line')
8  p2.multi_line(xs=[[1, 2, 3], [2, 3, 4]], ys=[[6, 7, 2], [4, 5, 7]],
9                color=['red','green'])
10
11 grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)
12
13 show(grid)
```

图示如下：



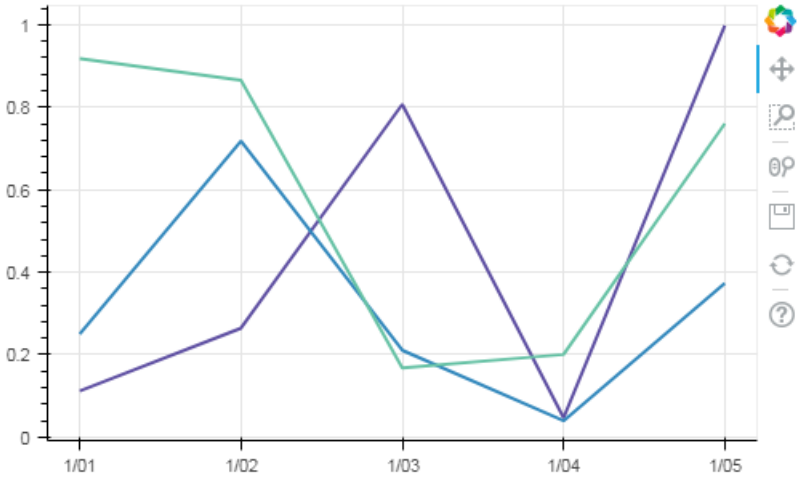
来看看关于 multi_line 的一个稍微复杂一些的例子，如下

```
from bokeh.palettes import Spectral11
import pandas as pd

df = pd.DataFrame(data=np.random.rand(5,3), columns = ('a', 'b' ,'c'),
                  index = pd.DatetimeIndex(start='01-01-2018',periods=5, freq='
                      d'))

df
```

	a	b	c
2018-01-01	0.111741	0.249899	0.917630
2018-01-02	0.264147	0.717774	0.865715
2018-01-03	0.807079	0.210551	0.167243
2018-01-04	0.046706	0.039422	0.200231
2018-01-05	0.998543	0.372787	0.760510

```
numlines=len(df.columns)
mypalette=Spectral11[0:numlines]

p = figure(width=500, height=300, x_axis_type="datetime")
p.multi_line(xs=[df.index.values]*numlines,
                ys=[df[name].values for name in df],
                line_color=mypalette,
                line_width=2)
show(p)
```

图示如下：



关于这个 multi_line，总觉得在 bokeh 里过于复杂，在 pandas 里绘制类似的图形，相对来说比较容易。

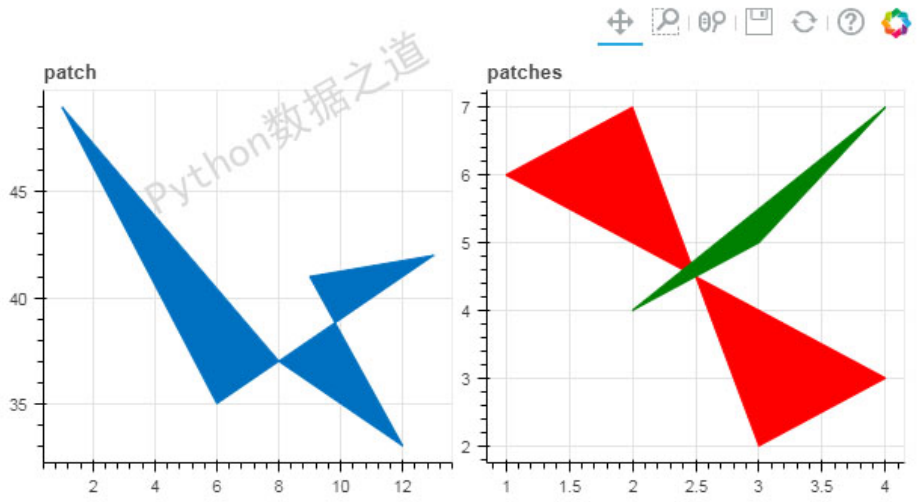
32.1.8 patch, patches

patch/patches，中文翻译不确定如何解释为好，这里姑且称之为“块状图”吧。

- patch(x, y, **kwargs)
- patches(xs, ys, **kwargs)

```
p1 = figure(title='patch')
p1.patch(x,y,color='#0071c1')


p2 = figure(title='patches')
p2.patches(xs=[[1, 2, 3, 4], [2, 3, 4]], ys=[[6, 7, 2, 3], [4, 5, 7]],
             color=['red','green'])

grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

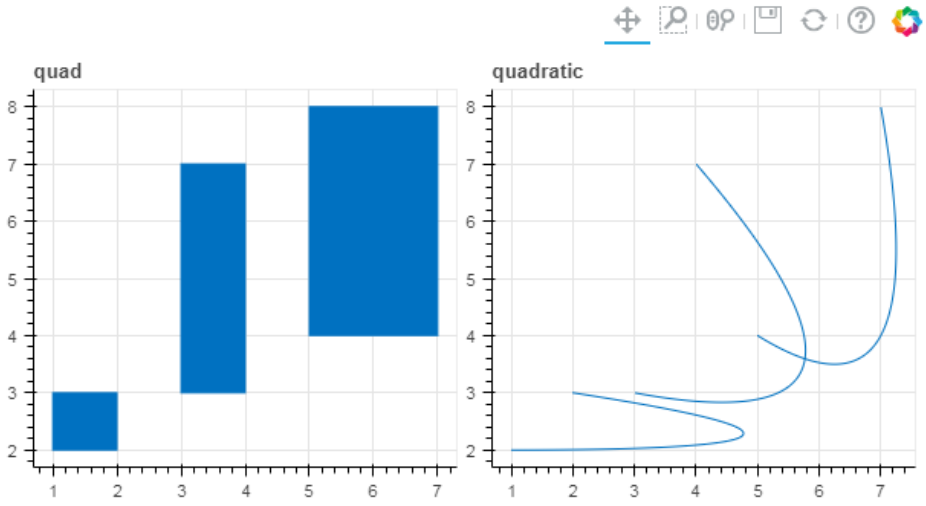


32.1.9 quad, quadratic

- quad(left, right, top, bottom, **kwargs)，四方的
- quadratic(x0, y0, x1, y1, cx, cy, **kwargs)，二次方程的

```
p1 = figure(title='quad')
p1.quad(left=[1,3,5],right=[2,4,7], top=[3,7,8], bottom=[2,3,4], color='#0071c1
    ')


p2 = figure(title='quadratic')
p2.quadratic(x0=[1,3,5],x1=[2,4,7], y1=[3,7,8], y0=[2,3,4], cx=8 , cy=2, color=
    '#0071c1')

grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

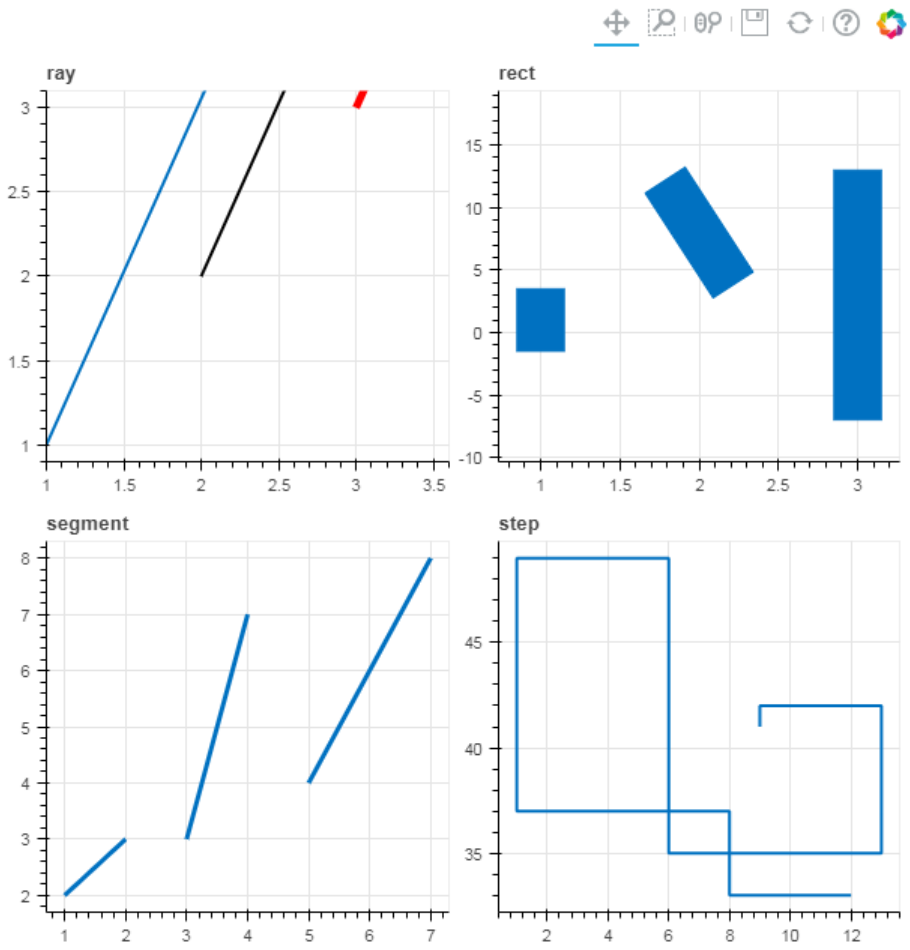


32.1.10 ray, rect, segment, step

- ray(x, y, length, angle, **kwargs)，射线
- rect(x, y, width, height, angle=0.0, dilate=False, **kwargs)，矩形
- segment(x0, y0, x1, y1, **kwargs)，分段，段落

```
p1 = figure(title='ray', x_range=[1,3.6])
p1.ray(x=[1, 2, 3], y=[1, 2, 3], length=45, angle=20, color=["#0071c1", 'black', 'red'],
       line_width=[2,2,5])

p2 = figure(title='rect')
p2.rect(x=[1, 2, 3], y=[1, 8, 3], width=0.3, height=[5,10,20],angle=[0,10,0], color="#0071c1")

p3= figure(title='segment')
p3.segment(x0=[1,3,5],x1=[2,4,7], y1=[3,7,8], y0=[2,3,4], color='#0071c1', line_width=3)

p4 = figure(title='step')
p4.step(x, y, color="#0071c1",line_width=2)

grid=gridplot([p1,p2,p3,p4],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：



32.2 综合小结

这 29 种图形，是 bokeh 中比较基础的类型，其中一部分，我们会在后续分享中多次应用，希望对大家有所启发。

当然，还有一些图形没有提到，各位可以自行研究下。

33 Bokeh 中独特的数据类型简介：ColumnDataSource

本文的环境为

- window 7 系统
- python 3.6
- Jupyter Notebook
- bokeh 0.13.0

数据是进行数据可视化的必要基础，在 bokeh 中，数据有几种呈现方式。

(1) 直接提供数据

(2) 通过 ColumnDataSource 来提供数据

33.1 直接提供数据

首先加载相关 Python 库。

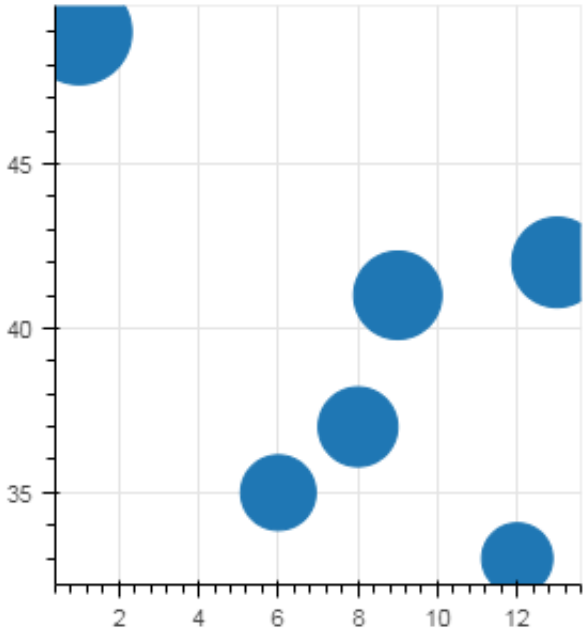
```
from bokeh.plotting import figure, output_notebook, show
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource
import numpy as np
import pandas as pd

output_notebook()
```

可以通过数据列的形式（list）直接提供数据

```
np.random.seed(15)

x=np.random.randint(1,20,size=6)
y=np.random.randint(20,50,size=6)

print(x)
print(y)

p = figure(plot_width=300, plot_height=300)
p.circle(x, y,size=y)

show(p)
```

```
[ 9 13  6  1  8 12]
[41 42 35 49 37 33]
```

图示如下：



33.2 通过 ColumnDataSource 来提供数据

ColumnDataSource 是 Bokeh 中一种重要的数据形式，ColumnDataSource() 方法有一个参数为 “data”，“data”主要有以下三种类型：

（1）data 为字典

（2）data 为 Pandas 的 DataFrame

（3）data 为 Pandas 的 DataFrame 的 groupby 对象

33.2.1 data 为字典

data 的表现形式是一个字典的形式，一般情况下，字典的 key 值是一个字符串，代表列名称，而 value 则是 list 形式或者 numpy 的 array 形式。演示如下：

```
data = {'x_values': [1, 2, 9, 4, 5],
        'y_values': [6, 7, 2, 3, 6]}

source = ColumnDataSource(data=data)
source
```

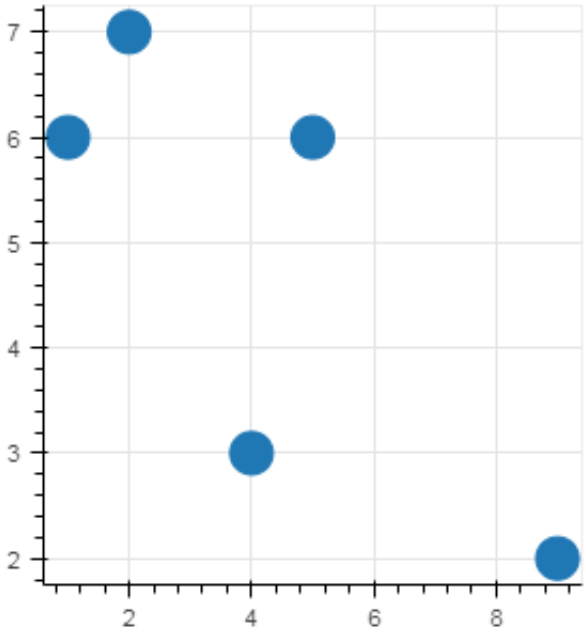
```
type(source)
```

```
bokeh.models.sources.ColumnDataSource
```

从上面结果来看，source 是一个 ColumnDataSource 对象，不能直接打印出来，后续可以在绘图是传入参数进行使用。

```
data = {'x_values': [1, 2, 9, 4, 5],
        'y_values': [6, 7, 2, 3, 6]}

source = ColumnDataSource(data=data)

p = figure(plot_width=300, plot_height=300)
p.circle(x='x_values', y='y_values', source=source, size=20)
show(p)
```

图示如下：



33.2.2 data 为 pandas 的 DataFrame

ColumnDataSource （简称为 CDS）的 data 参数，也可以是 pandas 的 DataFrame。当 CDS 的参数是 DataFrame 时，参数中可以直接用 DataFrame 的列名称，索引名称，也可以直接用 DataFrame 已有的名称，如果没有索引名称，默认情况下，索引名称用 “index” 就可以。

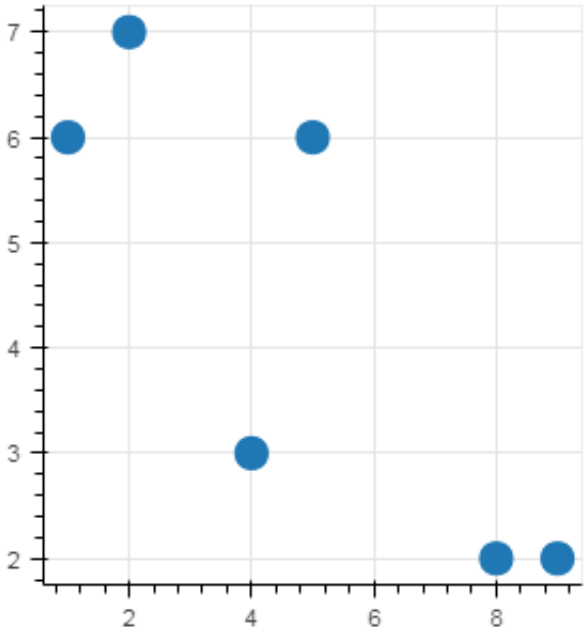
```
data = {'x_column': [1, 2, 9, 4, 5, 8],
        'y_column': [6, 7, 2, 3, 6, 2]}

df = pd.DataFrame(data=data)
```

```
df
```

	x_column	y_column
0	1	6
1	2	7
2	9	2
3	4	3
4	5	6
5	8	2

```
source_df = ColumnDataSource(df)

p = figure(plot_width=300, plot_height=300)
p.circle(x='x_column', y='y_column', source=source_df, size=15)
show(p)
```

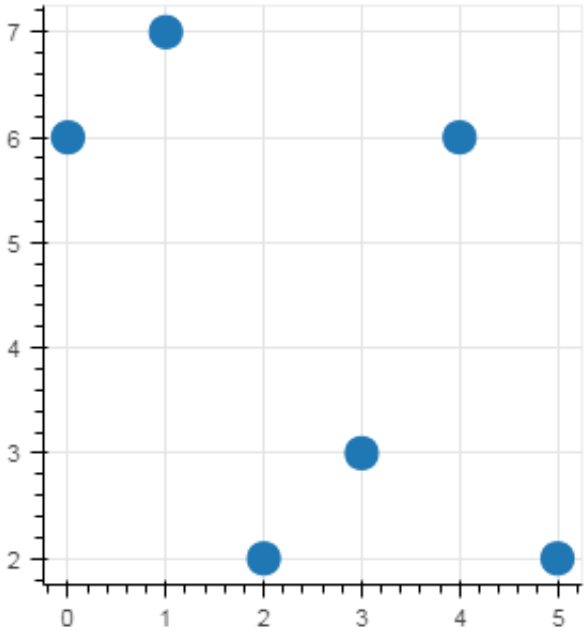
图示如下：




```
source_df = ColumnDataSource(df)

p = figure(plot_width=300, plot_height=300)

# 使用 “index” 作为 DataFrame 的默认索引名称
p.circle(x= 'index', y='y_column', source=source_df, size=15)
```

```
7 show(p)
```

图示如下：



33.2.3 data 为 pandas 的 DataFrame 的 groupby 对象

ColumnDataSource （简称为 CDS）的 data 参数，还也可以是 pandas 的 DataFrame 的 groupby 对象。

当 CDS 的参数是 DataFrame 的 groupby 对象时，在绘图时使用的列名为 groupby 对象的 groupby.describe() 方法中的列名称。

由于 groupby 会有多个统计参数，在引用时，列表会合并到一起，形式如：column_mean 等。

```
1 dates = pd.date_range('20180101', periods=360)
2
3 df = pd.DataFrame(np.random.randn(360,2), index=dates, columns=list('AB'))
4
5 df['C'] = ['Good', 'Bad', 'Common', 'Good','Good']*72
6 df['month'] = df.index.month
7
8 df
```

图示如下：

	A	B	C	month
2018-01-01	0.107071	2.460188	Good	1
2018-01-02	-0.517491	0.213835	Bad	1
2018-01-03	-1.444430	-0.080809	Common	1
2018-01-04	-0.072208	0.907390	Good	1
2018-01-05	0.359186	0.888204	Good	1

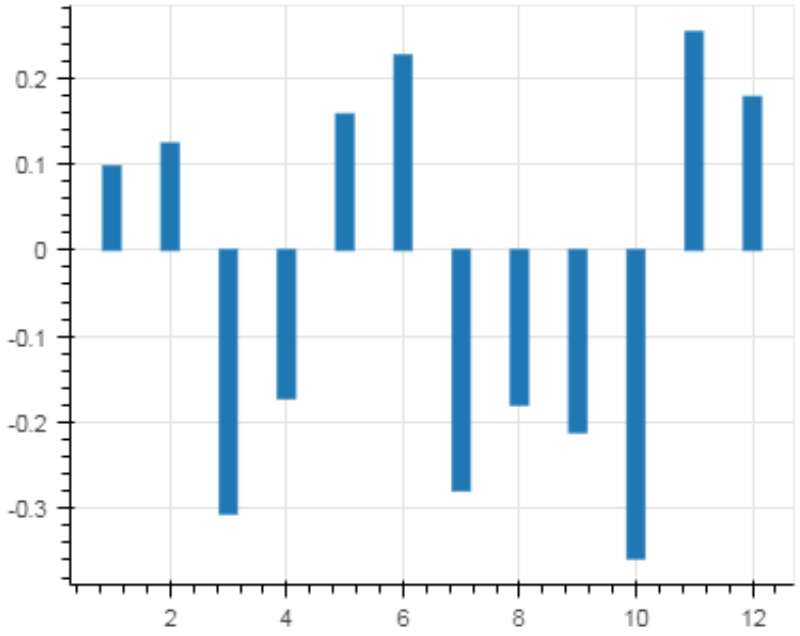
```
g = df.groupby('month')

g.describe()
```

图示如下：

	A								B
	count	mean	std	min	25%	50%	75%	max	co
month									
1	31.0	0.097555	0.977037	-1.797101	-0.421312	0.234122	0.708352	1.911893	31.
2	28.0	0.123796	0.832912	-1.566971	-0.597919	0.249597	0.738137	1.380626	28.
3	31.0	-0.305952	0.865404	-1.761597	-0.811632	-0.325259	0.240729	1.810966	31.
4	30.0	-0.172096	0.864274	-1.796895	-0.851580	-0.120589	0.549985	1.832491	30.
5	31.0	0.157829	0.969460	-1.800019	-0.441239	0.179009	0.759174	2.253888	31.
6	30.0	0.226035	1.054476	-2.456756	-0.308975	0.413759	0.879814	1.940967	30.
7	31.0	-0.279147	1.208461	-3.204534	-1.152630	-0.153925	0.592159	1.808065	31.
8	31.0	-0.179559	0.802671	-2.133487	-0.497727	-0.083627	0.232733	1.444430	31.
9	30.0	-0.211570	0.852524	-1.946128	-0.977727	-0.090862	0.427597	1.349902	30.
10	31.0	-0.357981	1.015426	-2.721714	-0.998281	-0.420551	0.159884	2.114726	31.
11	30.0	0.253082	1.215074	-2.518439	-0.675033	0.356449	1.102798	2.805104	30.
12	26.0	0.177882	1.045153	-1.773598	-0.580895	0.273264	0.827014	2.275739	26.

```
source_g = ColumnDataSource(g)
p = figure(plot_width=400, plot_height=300)
p.vbar(x='month', width=0.3, bottom=0, top='A_mean',source=source_g)
show(p)
```

图示如下：



33.3 小结

相对于 matplotlib，pandas，seaborn 等 Python 会图库，Bokeh 提供了特有的数据源，掌握好 ColumnDataSource 的应用，对于 Bokeh 绘图是至关重要的。后续，我们还会陆续接触到 ColumnDataSource 的相关用法。

34 Bokeh 中数据的添加、修改和筛选

本文的环境为

- window 7 系统
- python 3.6
- Jupyter Notebook
- bokeh 0.13.0

数据是进行数据可视化的必要基础，前文介绍了 bokeh 中数据呈现的几种方式。

本文主要介绍在 bokeh 中对 ColumnDataSource 类型数据中进行：

(1) 添加新的数据（2）修改更新已有数据（3）筛选数据。

首先加载相关 Python 库。

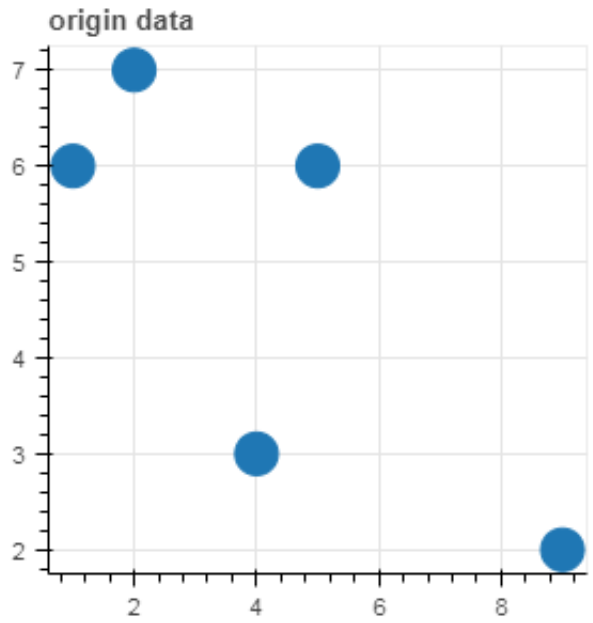
```
from bokeh.plotting import figure, output_notebook, show
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource
import numpy as np
import pandas as pd

output_notebook()
```

34.1 添加新的数据

ColumnDataSource 通过 stream() 方法来添加新的数据

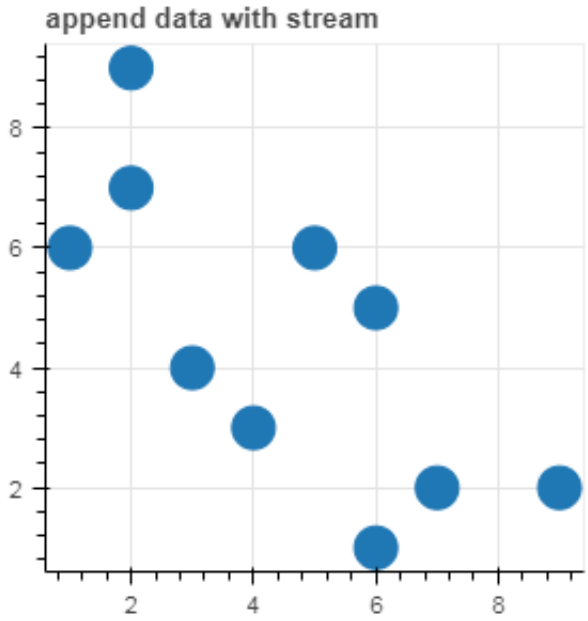
```
data1 = {'x_values': [1, 2, 9, 4, 5],
         'y_values': [6, 7, 2, 3, 6]}

source = ColumnDataSource(data=data1)

p1 = figure(plot_width=300, plot_height=300, title= 'origin data')
p1.circle(x='x_values', y='y_values', source=source, size=20)

show(p1)
```

图示如下：




```
new_data = {'x_values':  [6, 7, 2, 3, 6],
        'y_values': [1, 2, 9, 4, 5]}

# 在已有数据基础上添加新的数据 （append）
source.stream(new_data)

p2 = figure(plot_width=300, plot_height=300,title= 'append data with stream')
p2.circle(x='x_values', y='y_values', source=source, size=20)

show(p2)
```

图示如下：



34.2 数据更新

用 patch 方法可以更新 ColumnDataSource 的数据。

```
data = {'x_column': [1, 2, 9, 4, 5, 8],
        'y_column': [6, 7, 2, 3, 6, 2]}

df = pd.DataFrame(data=data)

df
```

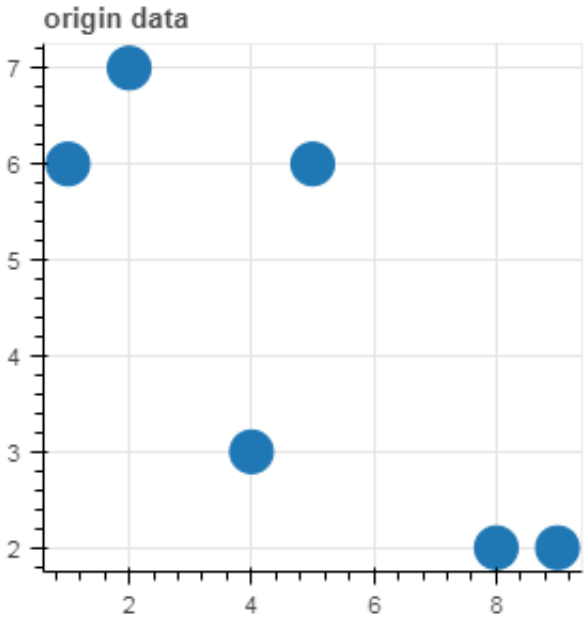
如下：

	x_column	y_column
0	1	6
1	2	7
2	9	2
3	4	3
4	5	6
5	8	2

```
source = ColumnDataSource(data=df)

p1 = figure(plot_width=300, plot_height=300, title= 'origin data')
p1.circle(x='x_column', y='y_column', source=source, size=20)

```

```
6 show(p1)
```

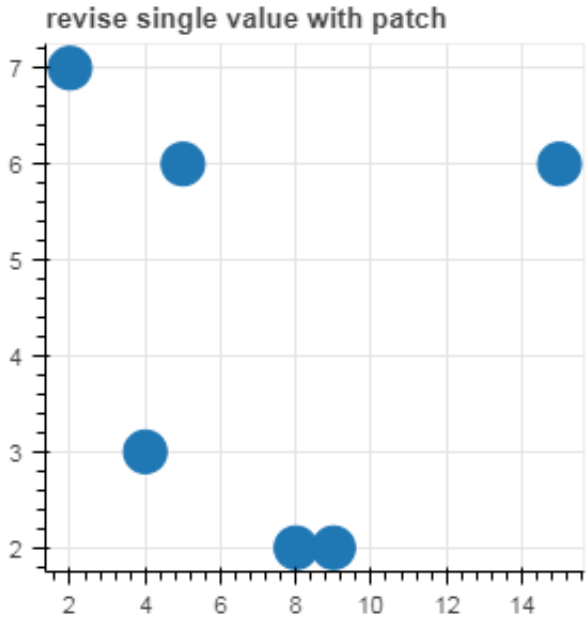
图示如下：



34.2.1 更新单个数据

```
1 # 更新单个数据
2 # {column:(index, new_value) }
3 source.patch({'x_column':[(0,15)]})
4
5 p2 = figure(plot_width=300, plot_height=300,title= 'revise single value with
      patch')
6 p2.circle(x='x_column', y='y_column', source=source, size=20)
7
8 show(p2)
```

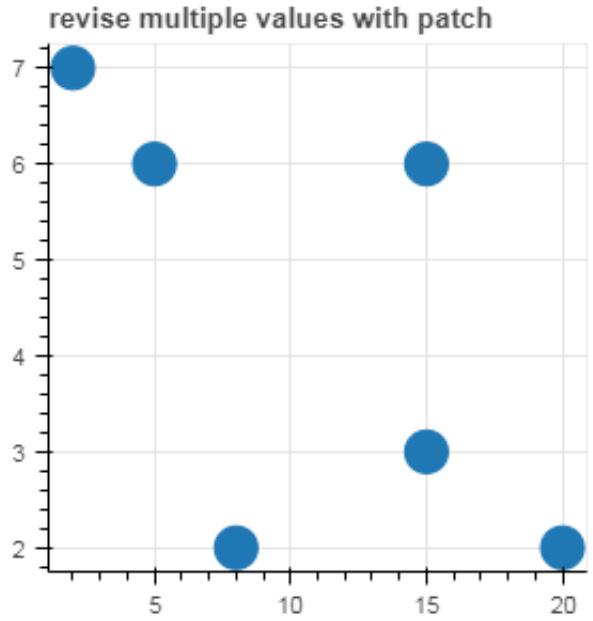
图示如下：



34.2.2 更新多个数据

```
# 更新多个数据
# {column:(slice, new_values) }

s = slice(2,4)
source.patch({'x_column':[(s,[20,15])]})

p2 = figure(plot_width=300, plot_height=300,title= 'revise multiple values with
    patch')
p2.circle(x='x_column', y='y_column', source=source, size=20)

show(p2)
```

图示如下：



34.3 筛选数据

在 Bokeh 中，ColumnDataSource （CDS）提供了 CDSView 来对数据进行筛选

其一般用法如下：

```
from bokeh.models import ColumnDataSource, CDSView
source = ColumnDataSource(some_data)
view = CDSView(source=source, filters=[filter1, filter2])
```

其中 filters 包括 IndexFilter, BooleanFilter, GroupFilter 等

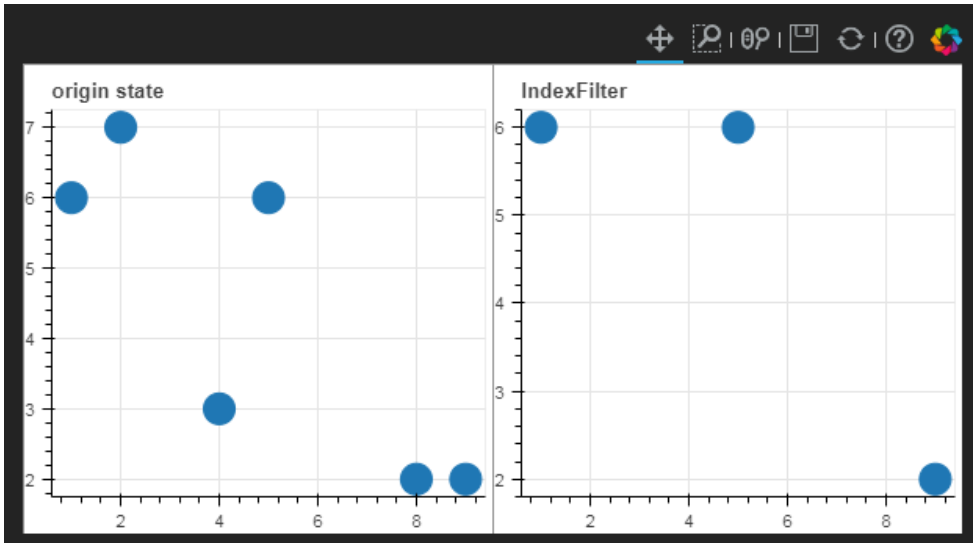
34.3.1 Indexfilter

根据数据的索引来筛选数据

```
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, CDSView, IndexFilter
from bokeh.layouts import gridplot

data = {'x_column': [1, 2, 9, 4, 5, 8],
        'y_column': [6, 7, 2, 3, 6, 2]}

df = pd.DataFrame(data=data)
source = ColumnDataSource(data=df)
view = CDSView(source=source, filters=[IndexFilter([0,2,4])])
```

```
11
12 p1 = figure(plot_width=300, plot_height=300, title='origin state')
13 p1.circle(x='x_column', y='y_column', source=source, size=20)
14
15 p2 = figure(plot_width=300, plot_height=300, title='IndexFilter')
16 p2.circle(x='x_column', y='y_column', source=source, size=20, view=view)
17
18 grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)
19
20 show(grid)
```

图示如下：

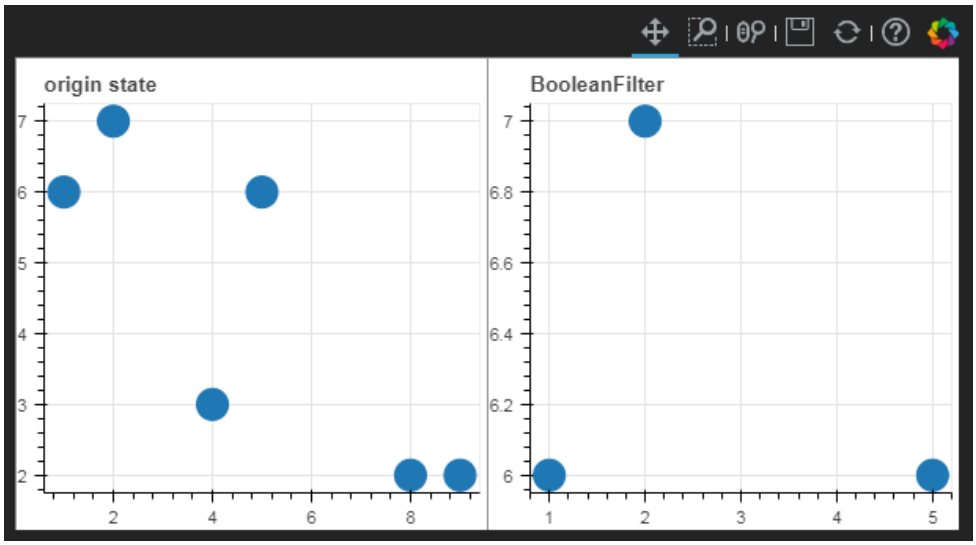


34.3.2 BooleanFilter

根据布尔值，True 或 False 来筛选数据

```
1  from bokeh.models import BooleanFilter
2  from bokeh.layouts import row
3
4  booleans = [True if y_val>4 else False for y_val in source.data['y_column']]
5  view_booleans = CDSView(source=source, filters=[BooleanFilter(booleans)])
6
7  p1 = figure(plot_width=300, plot_height=300,title='origin state')
8  p1.circle(x='x_column', y='y_column', source=source, size=20)
9
10 p2 = figure(plot_width=300, plot_height=300, title='BooleanFilter')
11 p2.circle(x='x_column', y='y_column', source=source, size=20, view=
       view_booleans)
12
13 grid=gridplot([p1,p2],ncols=2,plot_width=300,plot_height=300)
14
15 show(grid)
```

图示如下：



34.3.3 GroupFilter

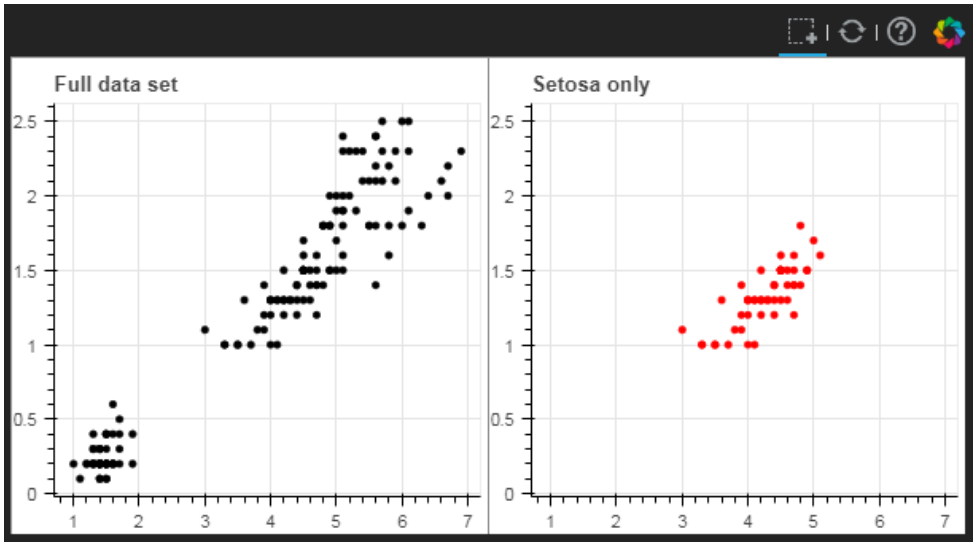
使用 GroupFilter 可以筛选出包含特定类型的所有行数据。GroupFilter 有两个参数，即 Column_name 和 group，也就是列名和类别名称。

如下面的官方例子所述，如果想筛选 iris 数据中特定类别的花，可以使用 GroupFilter 方法。

```
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource, CDSView, GroupFilter

from bokeh.sampledata.iris import flowers

# output_file("group_filter.html")

source = ColumnDataSource(flowers)
view1 = CDSView(source=source, filters=[GroupFilter(column_name='species',
    group='versicolor')])

plot_size_and_tools = {'plot_height': 300, 'plot_width': 300,
                       'tools':['box_select', 'reset', 'help']}

p1 = figure(title="Full data set", **plot_size_and_tools)
p1.circle(x='petal_length', y='petal_width', source=source, color='black')

p2 = figure(title="Setosa only", x_range=p1.x_range, y_range=p1.y_range, **
    plot_size_and_tools)
p2.circle(x='petal_length', y='petal_width', source=source, view=view1, color='
    red')
```

```
20
21 show(gridplot([[p1, p2]]))
```

图示如下：



当然，还有一些其他的筛选方法，有兴趣的同学可以自己挖掘下 ~~

35 Bokeh 中图形与组件的布局简介

35.1 图形的布局

图形（plot）的布局可以通过 column() 、row() 和 gridplot() 方法来实现，其中：

1、column() 方法是将所有图形 (plots) 在一列中分布；

2、row() 方法是将所有图形 (plots) 在一行中分布；

3、gridplot() 方法，可以按需求进行行列分布。

35.1.1 column

把所有图形放在一列中分布，其基本用法为 column([plot_1, plot_2, ……, plot_n])

```
from bokeh.io import output_notebook, show
from bokeh.layouts import column, row, gridplot
from bokeh.plotting import figure
import numpy as np

output_notebook()
```

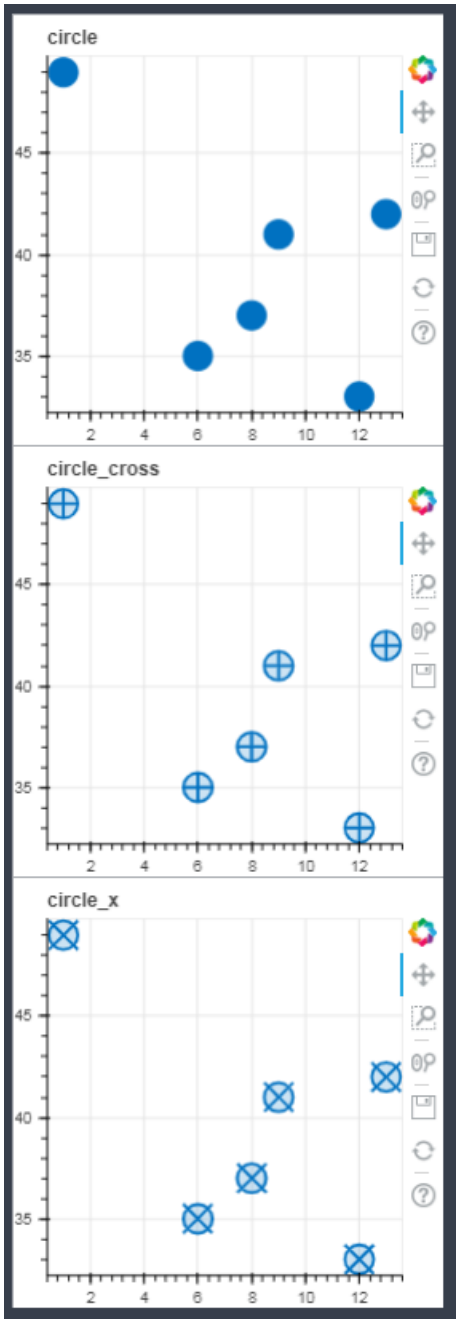
准备基础数据和图形

```
np.random.seed(15)

x=np.random.randint(1,20,size=6)
y=np.random.randint(20,50,size=6)

p1 = figure(title='circle',plot_width=300,plot_height=300)
p1.circle(x,y,size=20, color='#0071c1')

p2 = figure(title='circle_cross',plot_width=300,plot_height=300)
p2.circle_cross(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p3 = figure(title='circle_x',plot_width=300,plot_height=300)
p3.circle_x(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p4 = figure(title='cross',plot_width=300,plot_height=300)
p4.cross(x,y,size=20, color='#0071c1', line_width=2)
```

将图形按列进行布局

```
column_layout = column([p1, p2, p3])
show(column_layout)
```

如下图所示：

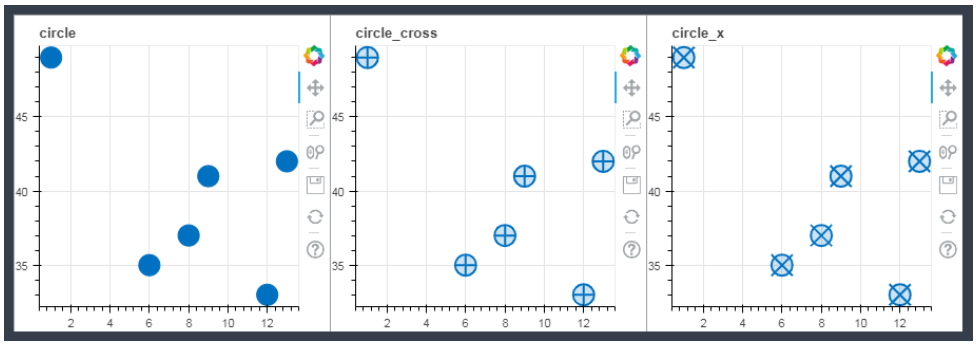


35.1.2 row

把所有图形按行分布，其基本用法为 row([plot_1, plot_2, ……, plot_n])

```
1 row_layout = row(p1,p2,p3)
2 show(row_layout)
```

如下图所示：



35.1.3 gridplot

使用 gridplot 来进行个性化布局，gridplot 的参数如下：

```
gridplot(\*args, \*\*kwargs)
```

Create a grid of plots rendered on separate canvases. gridplot builds a single toolbar for all the plots in the grid. gridplot is designed to layout a set of plots. For general grid layout, use the layout() function.

Parameters:

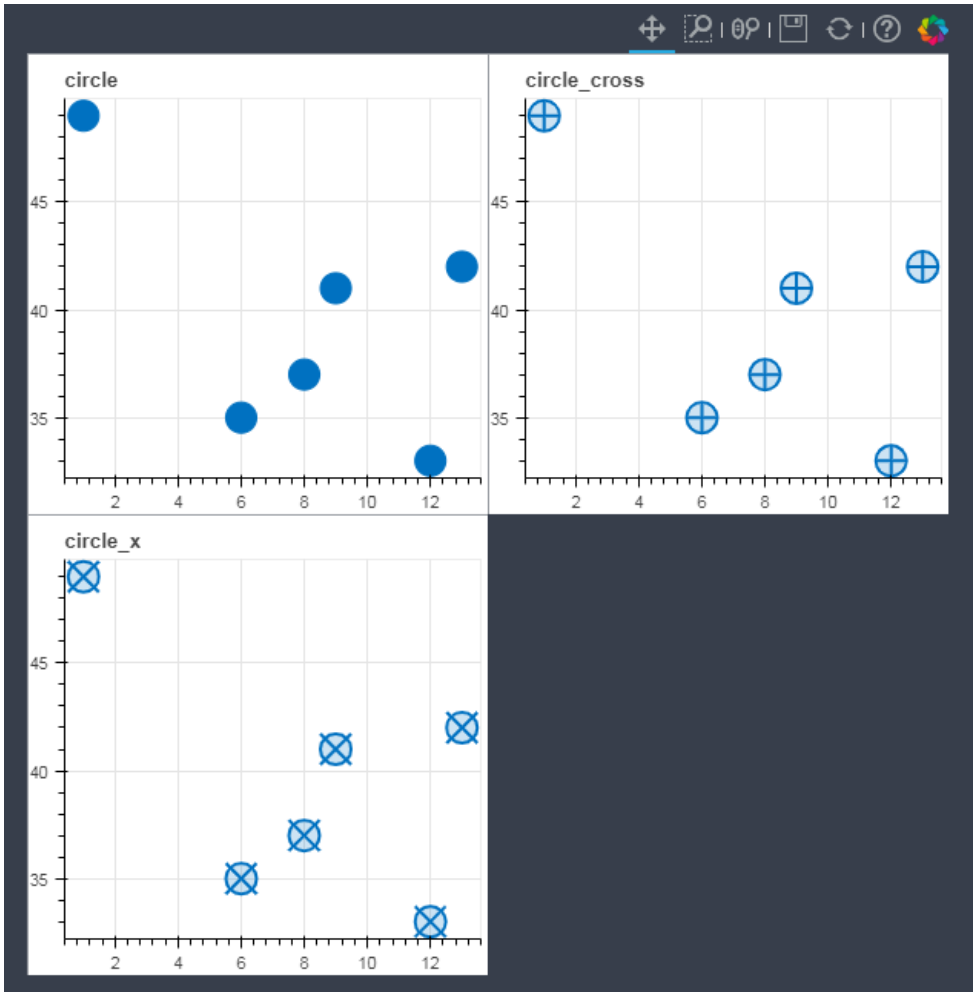
- **children** (list of lists of Plot) – An array of plots to display in a grid, given as a list of lists of Plot objects. To leave a position in the grid empty, pass None for that position in the children list. OR list of Plot if called with ncols. OR an instance of GridSpec.
- **sizing_mode** (“fixed”, “stretch_both”, “scale_width”, “scale_height”, “scale_both”) – How will the items in the layout resize to fill the available space. Default is “fixed”. For more information on the different modes see sizing_mode description on LayoutDOM.
- **toolbar_location** (above, below, left, right) – Where the toolbar will be located, with respect to the grid. Default is above. If set to None, no toolbar will be attached to the grid.
- **ncols** (int, optional) – Specify the number of columns you would like in your grid. You must only pass an un-nested list of plots (as opposed to a list of lists of plots) when using ncols.
- **plot_width** (int, optional) – The width you would like all your plots to be
- **plot_height** (int, optional) – The height you would like all your plots to be.
- **toolbar_options** (dict, optional) – A dictionary of options that will be used to construct the grid’s toolbar (an instance of ToolbarBox). If none is supplied, ToolbarBox’s defaults will be used.
- **merge_tools** (True, False) – Combine tools from all child plots into a single toolbar.

可以在 `gridplot()` 方法中，以列表的形式将 plots 分组按行列的形式表示出来，如果要预留一个空置的位置，

可以用 “None” 来表示。

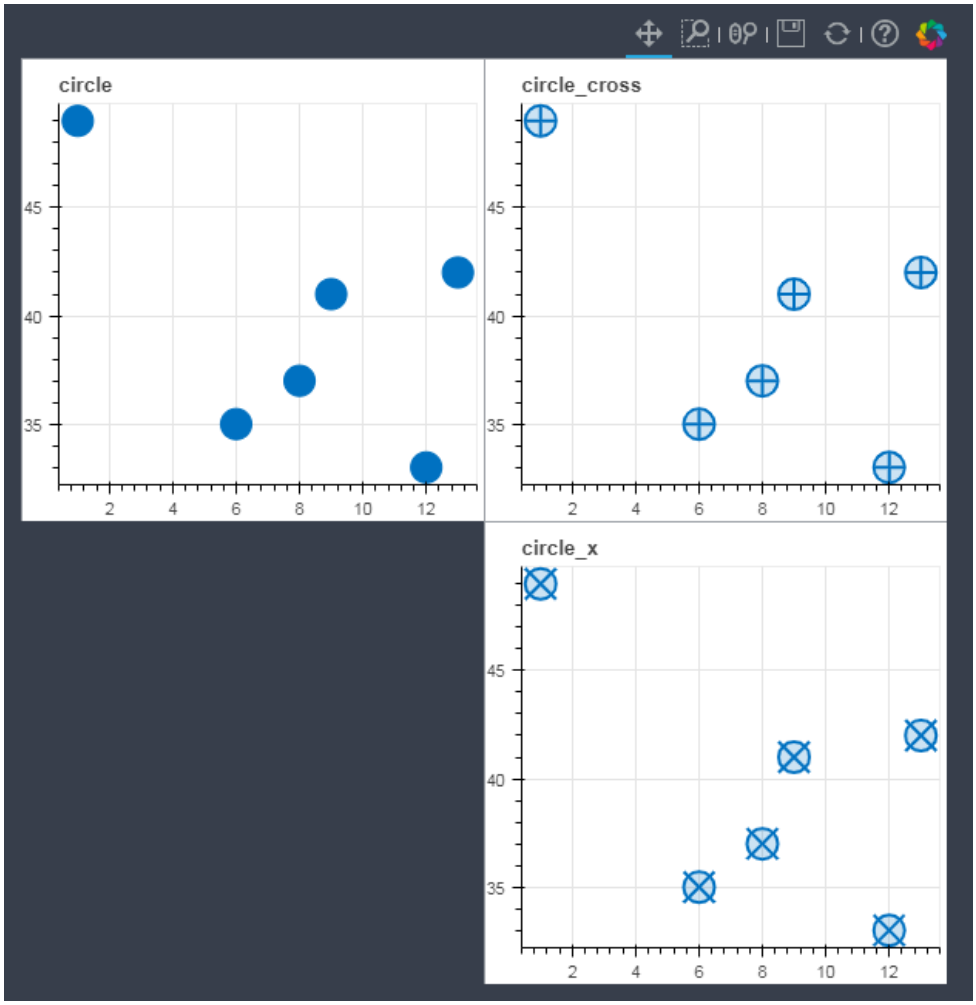
```
grid1=gridplot([p1,p2],[p3,])

show(grid1)
```

如下图所示：




```
grid2=gridplot([p1,p2],[None,p3])

show(grid2)
```

如下图所示：



在 gridplot() 方法中，还可以引入参数 ncols 来控制显示的列数，这里所有的 plots 放在一个列表中即可。

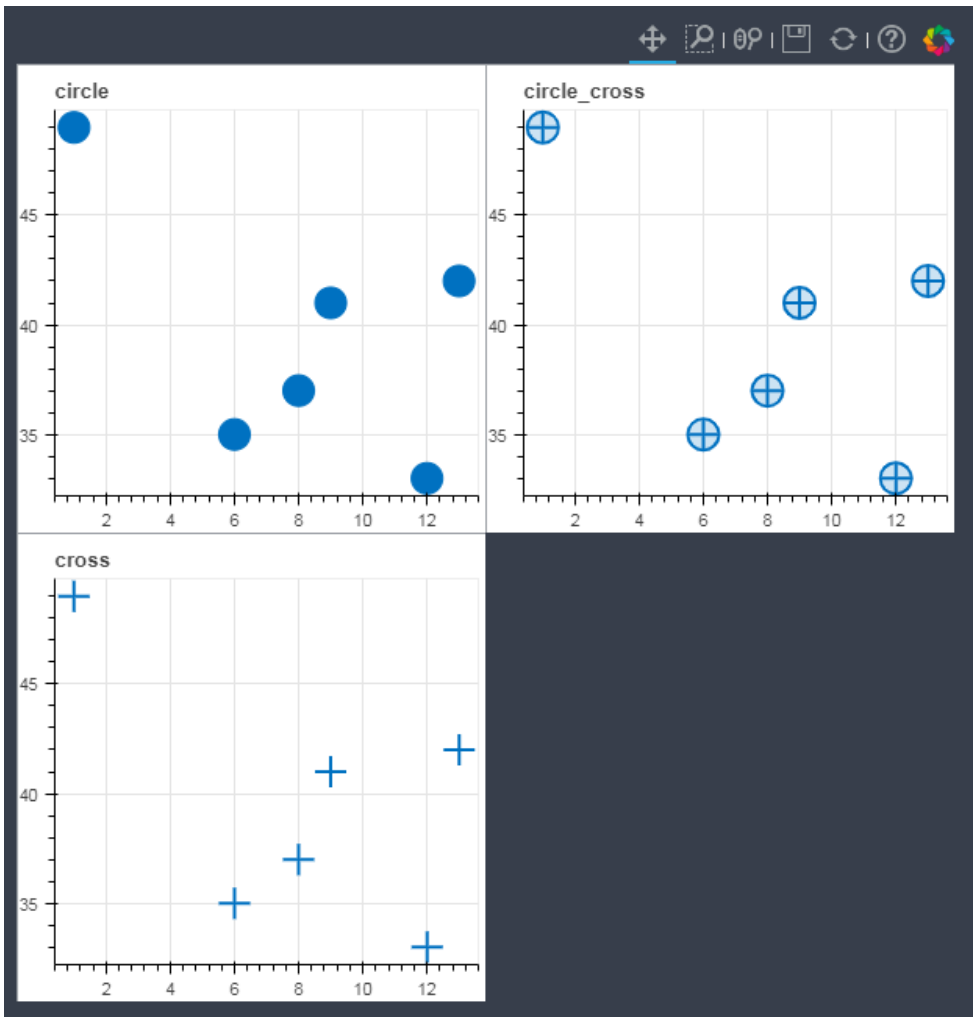
P.S. 官方文档中，提到有 “ncols” 参数时，不能同时使用 “None”，但我尝试了一下，是可以同时使用 “None” 的。有兴趣的小伙伴也可以试试。

官方的原文如下：

> You cannot use None with the ncols argument. It must only be a list of Plot objects at once.

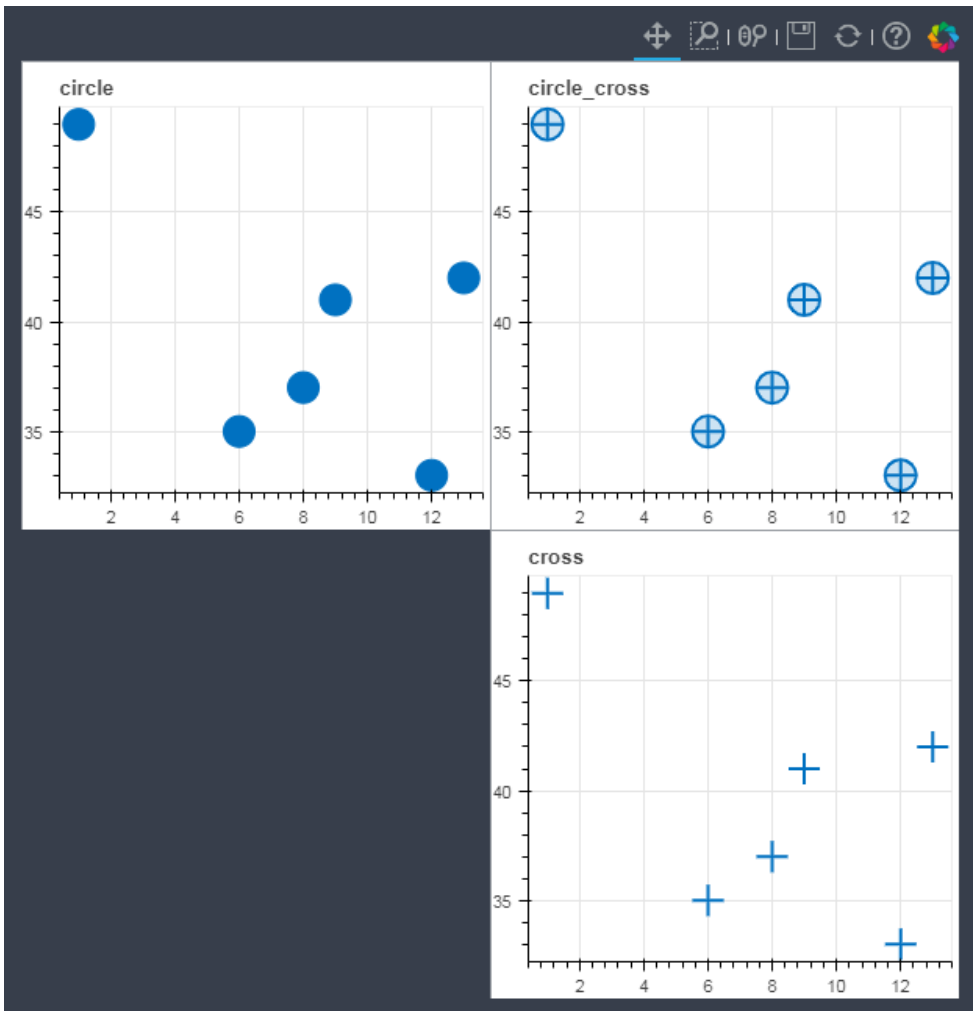
```
grid3=gridplot([p1,p2,p4],ncols=2, plot_width=300,plot_height=300)

show(grid3)
```

如下图所示：




```
grid4=gridplot([p1,p2,None,p4],ncols=2, plot_width=300,plot_height=300)

show(grid4)
```

如下图所示：



35.2 组件的布局

bokeh 中，组件（widgets）包括按钮（button），选项（Group），滑动块（slider）等等；组件的布局通过 widgerbox() 方法来实现

```
from bokeh.layouts import widgetbox
from bokeh.models.widgets import Button, RadioButtonGroup, Select, Slider
from bokeh.models.widgets import Dropdown, Toggle

# 创建一些组件
slider = Slider(start=0, end=20, value=1, step=0.5, title="Slider")
button_group = RadioButtonGroup(labels=["Option 1", "Option 2", "Option 3"],
    active=0)
select = Select(title="Option:", value="Lemon", options=["Lemon", "Python", "
    Java", "PHP"])
button_1 = Button(label="Button 1")
```

```
button_2 = Button(label="Button 2")

menu = [("Item 1", "item_1"), ("Item 2", "item_2"), None, ("Item 3", "item_3")]
dropdown = Dropdown(label="Dropdown button", button_type="warning", menu=menu)

# put the results in a row
show(widgetbox(button_1, slider,
               button_group, select,
               button_2, dropdown, width=300))
```

如下图所示：

> 上图在文章中是 gif 的动态图，可以在下面文章中查看：
>
> http://liyangbit.com/pythonvisualization/Bokeh-Laying-out-Plots-and-Widgets/

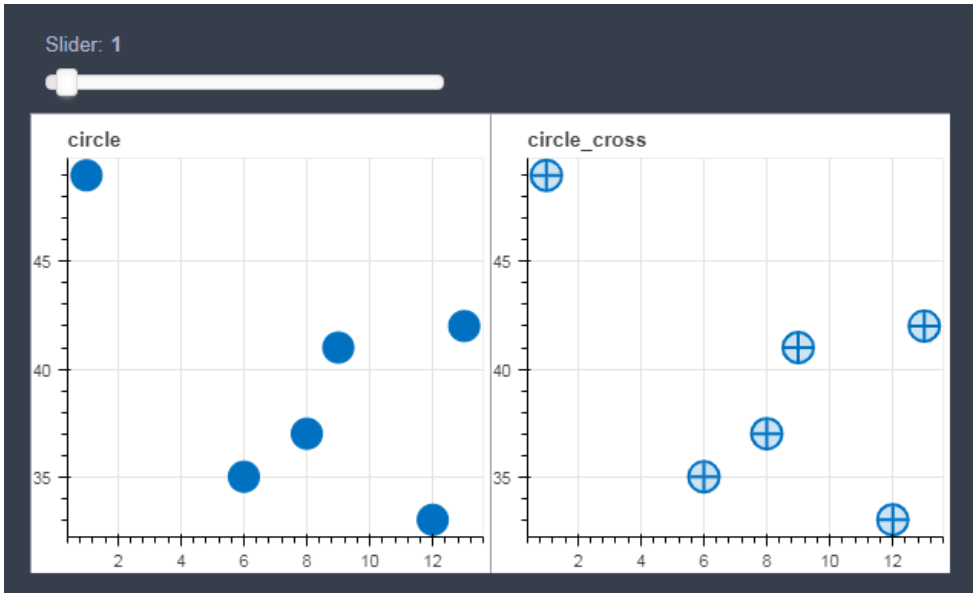
关于组件的具体内容介绍，我们会在后续进一步学习。

35.3 图形和组件混合布局

通过 layout() 方法，可以实现图形（plots）和组件（widgets）的混合布局。

```
from bokeh.layouts import layout

layout_01 =layout([slider],[p1,p2])

show(layout_01)
```

如下图所示：



这里需要注意的是，slider 和 plot 是放置在一起，但它们之间是没有内在联系的。

对比 Python 中常用的可视化库 Matplotlib，在 Bokeh 中，对图形和组件进行布局还是比较方便的。布局的功能，会在以后的实践中经常进行使用。

36 在 Pycharm 等编辑器下使用 Plotly Express

在这篇文章里，展示了 Plotly Express 在交互式可视化方面的强大功能，但同时也遇到一些小问题，比如，上述代码是在 Jupyter Notebook 中运行的，有不少童鞋问到在 Pycharm 中如何运行?

刚开始遇到这个问题时，我基本回复的都是“建议参考 Plotly 来进行修改”。

但估计很多童鞋还是不清楚怎么运行，这里，我提供一种方法，大家可以在 Pycharm 、VScode 等 IDE 中运行代码，由于 Pycharm、VSCode 我用的相对较多，只测试了这两种。其他的 IDE，个人觉得也是差不多的，大家可以自己试试。

36.1 第一个示例

我们以 iris 数据集为例来演示，代码如下（.py 文件）：

```
# Code based on Python 3.x
# _*_ coding: utf-8 _*_
# Author: "Lemon"
# Website: http://liyangbit.com
# 公众号：Python数据之道
# ID: PyDataLab

import plotly_express as px
import plotly
import plotly.graph_objs as go
plotly.offline.init_notebook_mode(connected=True)

iris = px.data.iris()

iris_plot = px.scatter(iris, x='sepal_width', y='sepal_length',
           color='species', marginal_y='histogram',
          marginal_x='box', trendline='ols')

plotly.offline.plot(iris_plot)
```

代码运行后，结果会出现在浏览器端，效果如下：

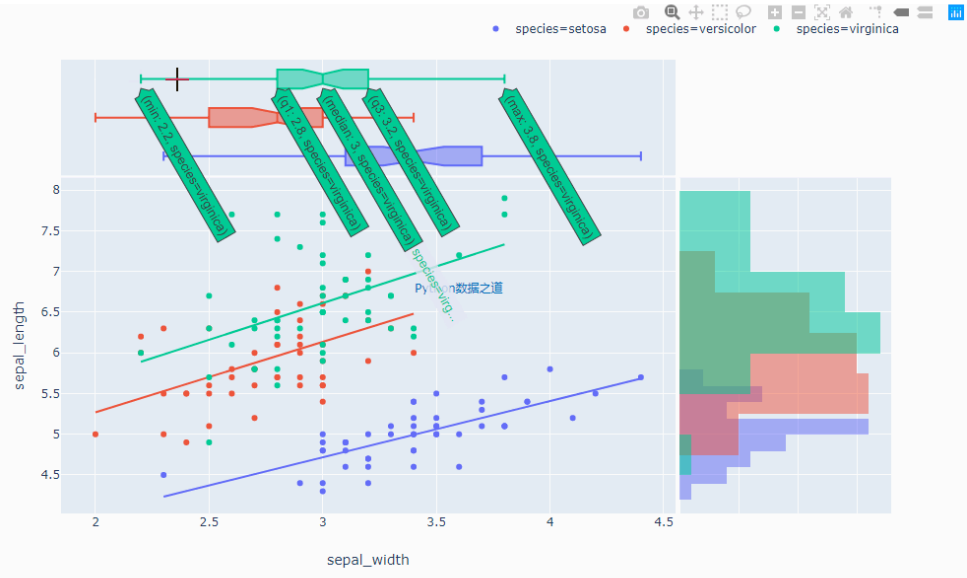


Figure 36.1: image-动态 gif

36.2 第二个示例

以 wind 数据集为例来演示，代码如下（.py 文件）：

```
# Code based on Python 3.x
# _*_ coding: utf-8 _*_
# Author: "Lemon"
# Website: http://liyangbit.com
# 公众号：Python数据之道
# ID: PyDataLab

import plotly_express as px
import plotly
plotly.offline.init_notebook_mode(connected=True)

wind = px.data.wind()
wind_plot = px.bar_polar(wind, r="value", theta="direction", color="strength",
    template="plotly_dark",
            color_discrete_sequence= px.colors.sequential.Plotly[-2::-1])

plotly.offline.plot(wind_plot)
```

代码运行后，结果会出现在浏览器端，图示如下：

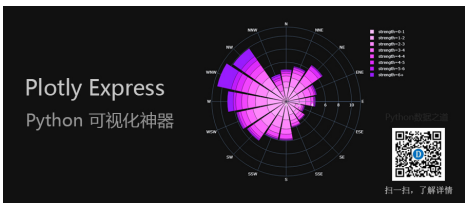


Figure 36.2: image

看完上面的代码，是不是觉得 so easy，赶紧动手试试吧。

37 Plotly 中如何保存 jpeg 等图片?

大家好，我是 Lemon 。

如本文开头的专辑所示，在过去的一段时间里，我写了一些用 Python 的交互式可视化工具 Plotly 来进行数据分析的文章。

有不少同学都问到如何保存 Plotly 中的图片，今天 Lemon 来跟大家分享下如何保存静态图片，比如 png、jpeg、svg 等。

37.1 保存静态图片的方法

在 Plotly 中保存静态图片，需要安装 3 个依赖包才可以，分别是 `orca` 、`psutil` 、和 `requests` 。

在安装上述依赖包时，需要用到 conda 管理工具，安装命令如下：

```
conda install -c plotly plotly-orca==1.2.1 psutil requests
```

正常情况下，通过上述命令后，应该会安装成功。

我自己安装的时候，发现有因为网络延迟原因，安装失败的情况。**如果遇到安装失败，可以多试几次，或者更换网络。**

上面的安装命令中，`orca` 需要通过 conda 管理工具来安装。

psutil 和 requests 可以通过 pip 命令来安装：

```
pip install psutil requests
```

在上述依赖包安装成功后，我们可以使用 `write_image` 方法来保存静态图片，支持的图片格式包括 png、jpeg、svg、webp、pdf、eps 等。

> An image format must be specified as one of the following string values: [“eps”, “jpeg”, “jpg”, “pdf”, “png”, “svg”, “webp”]

下面，Lemon 通过一个项目案例来给大家展示下静态图片的保存。

37.2 项目案例演示

这个案例，主要是通过获取标普 500 指数数据，对标普 500 的走势进行可视化，并将展示结果的图片进行保存。

本文的数据来自开源项目 yfinance，大家在运行案例中的代码时，需要先安装 yfinance，安装命令如下：

```
pip install yfinance
```

本项目案例是在 vs code，Pycharm 等环境中运行，Python 环境版本及其他主要工具如下：

- Python 3.7
- pandas version: 1.0.3
- plotly version: 4.5.0

本项目案例的代码如下：

```
# -*- coding: utf-8 -*-
"""
@Author: Lemon
@出品: Python数据之道
@Homepage: liyangbit.com
"""

import pandas as pd
import plotly.graph_objs as go
import plotly
from datetime import datetime
import yfinance as yf
import os

pyplt = plotly.offline.plot

# 如果项目路径下没有 "images" 文件夹，则创建该文件夹
if not os.path.exists("images"):
    os.mkdir("images")

# 如果项目路径下没有 "htmls" 文件夹，则创建该文件夹
if not os.path.exists("htmls"):
    os.mkdir("htmls")

date_end = datetime.today().strftime('%Y-%m-%d')

# yfinance，标准普尔500指数 雅虎的代码是 ^GSPC
# 在其他地方，标准普尔500指数，代码通常是 SPX
data_spx = yf.download('^GSPC', start='1950-01-01', end=date_end)

# plotly，基础图
trace0 = go.Scatter(x=data_spx.index, y=data_spx['Close'],
                    mode='lines',
                    marker=dict(color='#d66101'),  # 设置K线颜色
                    )
data = [trace0]
fig_spx = go.Figure(data=data)
fig_spx.update_layout(template='ggplot2')
fig_spx.update_layout(title=dict(text='价值前瞻-SPX', font=dict(size=24, color=
    '#d66101'), x=0.5))
```

```
40  # 保存静态图片
41  fig_spx.write_image('images/spx.png')
42  # 在浏览器中输出结果
43  pyplt(fig_spx, filename='htmls/export-image.html')
```

上述案例中，是以保存 png 为示例，其他格式的静态图片，各位可以自己去试试。

38 用 Python 可视化神器 Plotly 动态演示全球疫情变化趋势

近期对疫情数据进行可视化的内容比较多，今天我来用 Python 可视化申请 Plotly 对国外的疫情发展情况进行可视化，以项目实战的形式，在分析和了解国外疫情变化趋势的同时，加深大家对 Plotly 的学习应用。

在开始之前，我们先来看看最终制作的部分效果图，如果你觉得有兴趣，不妨继续往下看。

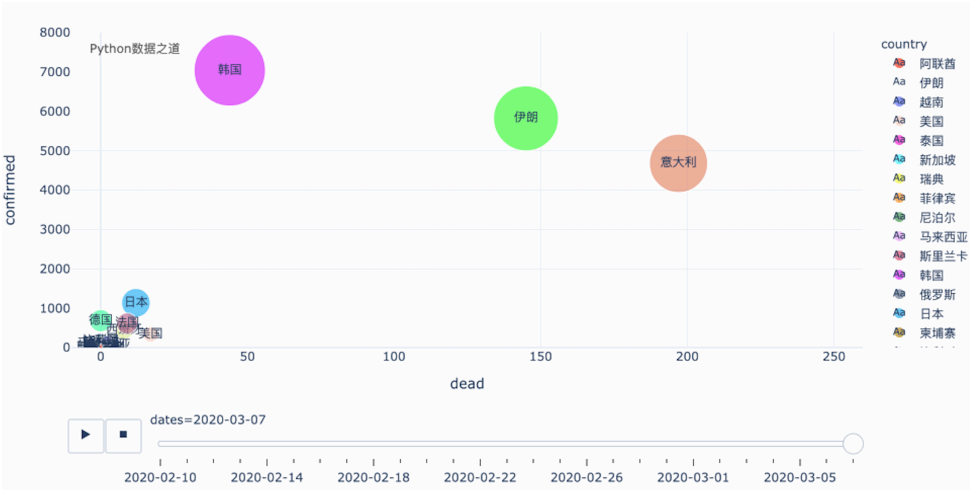


Figure 38.1: img-2

动态效果请 点击此处链接

38.1 数据来源

国内的疫情，目前已逐步受到控制，各项指标已开始明显好转，但国外的情况，看起来不是太乐观，已有的作业也没有抄好，所以，本次我们主要来可视化分析国外疫情的发展情况。

疫情的数据来源于开源项目 Akshare，由于使用该项目获取数据时，有时不太稳定，可能会遇到连接失败的情况，所以，这里我提供了保存好的数据供大家使用。

38.2 准备工作

照例，还是先介绍下我运行的环境

- Mac 系统
- Anaconda（Python 3.7）
- Jupyter Notebook

本次使用到的 Python 库包括 akshare, pandas, plotly 等，导入如下：

```
import akshare as ak
import pandas as pd
import plotly
from plotly.offline import iplot, init_notebook_mode
import plotly.express as px
from datetime import datetime

init_notebook_mode()
```

接着，我们读取已获得的数据，已保存的数据是截至 3 月 7 日的。

```
# 从 akshare 获取数据
# df_all_history = ak.epidemic_history()

# 从csv文件获取数据
df_all_history = pd.read_csv('epidemic_all_20200307.csv',index_col=0)

df_all_history
```

从上面获取的数据，有些数据格式需要加以调整，对于日期，我们这里会组织两列数据，一列是时间格式的日期（['date']），一列是字符串格式的日期（['dates']）。这样设置的原因，是因为我们后续分别需要用到这两种格式的日期。

```
df_all = df_all_history

# 将字符串格式的日期 另保存为一列
df_all['dates'] = df_all_history['date']

# 将字符串格式的日期转换为 日期格式
df_all['date'] = pd.to_datetime(df_all['date'])
```

38.3 获取国外的疫情数据

上面的数据，是全球的数据，我们可以把其中属于中国的剔除，就可以得到国外的数据了。

```
# 国外，按国家统计
df_oversea = df_all.query("country!='中国'")
df_oversea.fillna(value="", inplace=True)

```

```
df_oversea
```

先来用 plotly express 看下国外疫情分国家的整体走势。

```
fig_oversea = px.line(df_oversea, x='dates', y='confirmed',
                      line_group='country',
                      color='country',
                      color_discrete_sequence=px.colors.qualitative.D3,
                      hover_name='country',
                     )

fig_oversea.show()
```

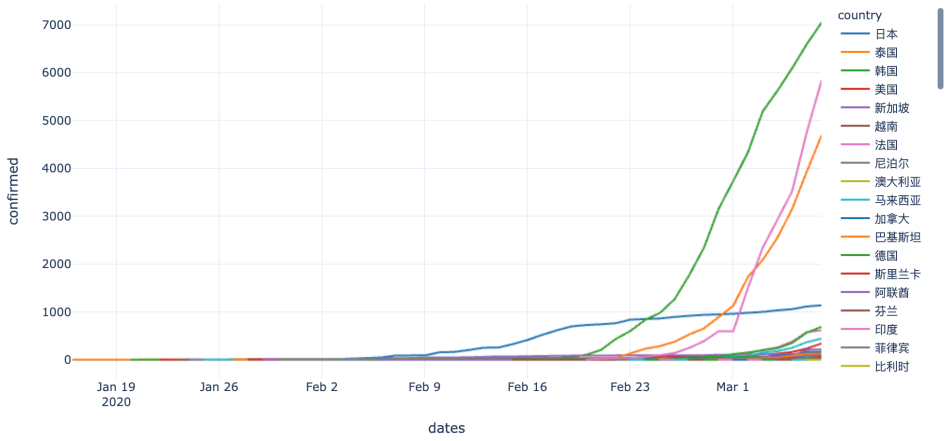


Figure 38.2: img-1

从上图可以看出，国外的疫情发展情况，大部分国家从 2 月 10 日期，发展趋势较为明显，因此，后面我们重点分析这段时间之后的情况。

```
# 现有数据演示从 2020年2月10日开始
df_oversea_recent = df_oversea.set_index('date')
df_oversea_recent = df_oversea_recent['2020-02-10':]
df_oversea_recent
```

由于部分国家的数据不是从 2020 年 2 月 10 日开始记录的，所以要补充数据。我们可以手动新建一个 excel 数据表，将补充日期的数值填充为 0 。

我这里主要补充的是伊朗的数据，因为伊朗实现是发展太快了，必须纳入分析的范围内。其他国家，如果有需要补充的，后续可以继续完善。

```
# 由于部分国家，数据不是从2020年2月10日开始的，所以要补充数据，数值为 0
# 数据在 excel 表格中进行补充，这里进行读取

df_oversea_buchong = pd.read_excel('epidemic_buchong.xlsx')
df_oversea_buchong['dates'] = df_oversea_buchong['date'].apply(lambda x:x.
    strftime('%Y-%m-%d'))
```

```
df_oversea_buchong.set_index('date', inplace=True)
df_oversea_buchong.fillna(value="", inplace=True)
print(df_oversea_buchong.info())
df_oversea_buchong
```

将需要补充的数据弄好后，我们可以合并上面这两部分数据，一起进行分析。

```
# 合并补充数据

df_oversea_recent_new = df_oversea_recent.append(df_oversea_buchong)

df_oversea_recent_new.sort_index(inplace=True)

df_oversea_recent_new
```

得到合并的数据后，首先，我们用气泡图来对变化情况进行可视化，这里用的是 plotly express 的散点图。

plotly express 现在已经合并到 plotly 中，个人觉得跟 plotly 原生内容的协同性相对 cufflinks 要好用点。

```
# -*- coding: utf-8 -*-
"""
@Author: Lemon
@出品: Python数据之道
@Homepage: liyangbit.com
"""

fig_oversea_recent = px.scatter(df_oversea_recent_new, x='dead', y='confirmed',
                                size='confirmed', text='country', color='country',
                                color_discrete_sequence=px.colors.qualitative.Light24,
                                animation_frame='dates',animation_group='country',
                                hover_name='country',
                                range_x=[-10,260],
                                range_y=[0,8000],
                                size_max=50,
                                template='plotly_white',
                               )

fig_oversea_recent.show()
```

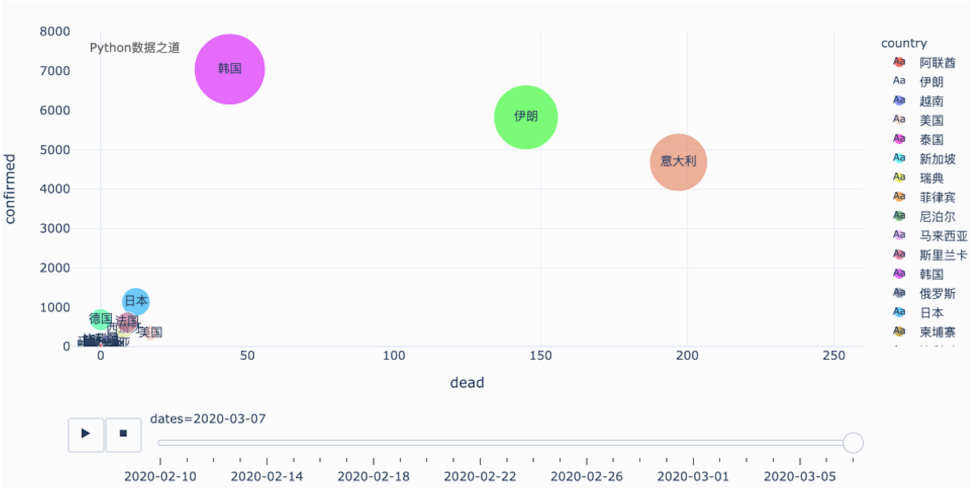


Figure 38.3: img-2

从上面这个动态图可以清晰的看出，当前，在海外的国家中，韩国、伊朗、意大利三个国家最为严重。这三个国家中，就增长趋势而言，伊朗和意大利又比韩国要更明显，目前韩国的增长有所放缓，而伊朗和意大利还处于快速增长的过程中，后续情况不容乐观。

此外，在这个图中，另外还有几个国家值得关注，日本，除去钻石号之外，从数据来看，本土的增长目前还算是在稍微较好的范围内。反倒是，德国、法国、西班牙，个人觉得已成逐步壮大之势，不得不防。

而且，由于整个欧盟国家之间，人员是自由流通的，现在看来，整个欧盟很可能会成为疫情的重灾区，其影响巨大。

上图左下角的这些国家的走势，我们可以拉近来看，走势如下，这样，对于德国、法国、西班牙就会看的更明显了。

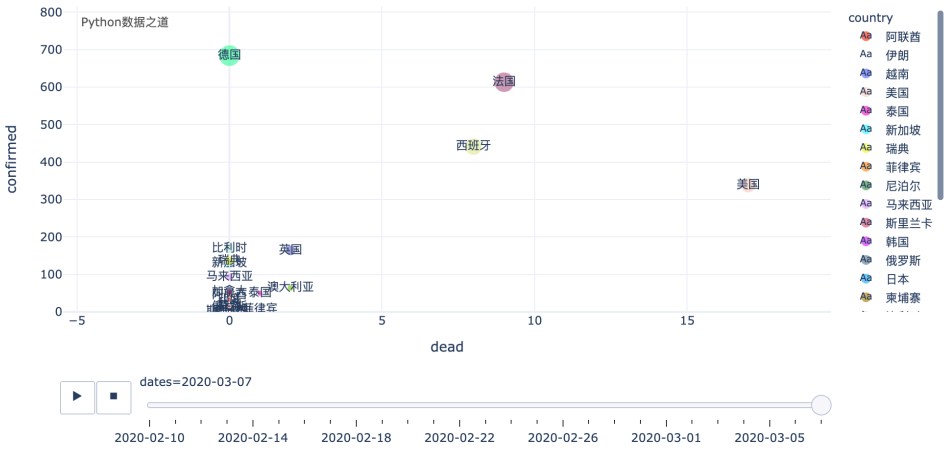


Figure 38.4: img-4

这里，还有一个国家，就是美国，虽然从数据以及增长情况来看，好像美国还好，但恐怕实际情况，要糟糕很多。

上面是以气泡图的方式来演示变化过程，我们也可以以柱状图的形式来演示，关于柱状图，各位可以自行研究下，欢迎交流。

> 需要说明的是，本文代码是在 Jupyter Notebook 中运行的，如果是在 PyCharm 等 IDE 中运行，需要稍微修改下代码。

最后，再次感谢开源项目 Akshare 提供了数据接口。为方便大家进行操作，Lemon 给大家提供了数据文件，可以在公众号「Python 数据之道」后台回复数字「691」进行获取。

39 用 Python 动态曲线图来对全球疫情进行演示

各位同学好，我是 Lemon 。

最近一段时间，我写了几篇用 Python 的交互式可视化工具 Plotly 来演示全球疫情情况的文章，如下：

- 用 Python 可视化神器 Plotly 动态演示全球疫情变化趋势
- 用 Plotly 动态柱状图来演示全球疫情变化趋势
- 超火动态排序疫情变化图，这次我们用 Plotly 来绘制

如今（截至 2020 年 3 月 28 日），全球确诊数量超过 60 万人，美国确诊数量已超过 10 万人，成为目前人数最多的国家，而且，疫情扩散的趋势还在快速发展过程中，多个国家首脑或高官也确诊，总体情况令人担忧，多国经济陷入困境，金融市场动荡不安。

今天，Lemon 继续来分享用 Plotly 对疫情情况进行可视化，本次我们用动态曲线图来进行演示。先来看最终的效果：

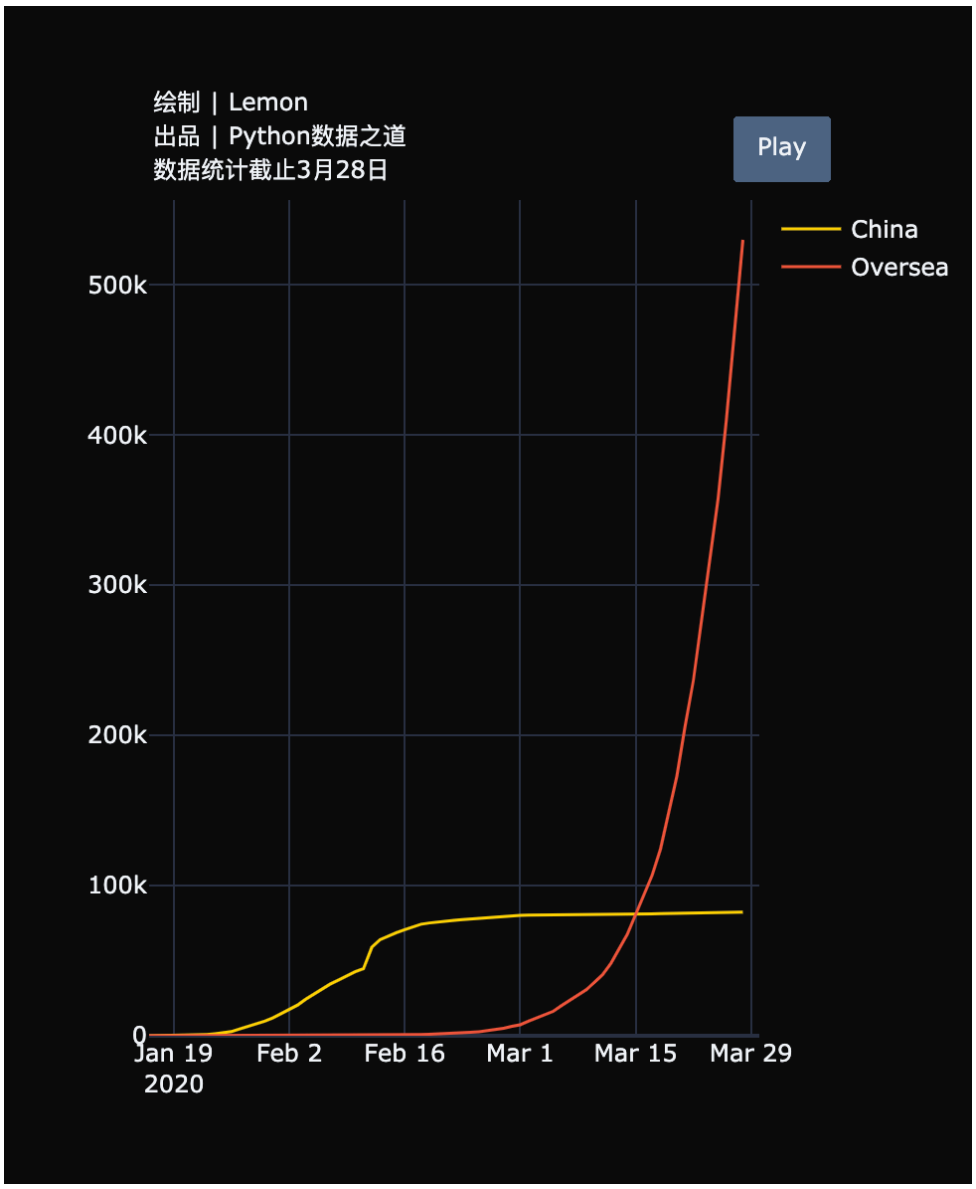


Figure 39.1: covid-19

上图的动态效果，请在下面文章中查看：

用 Python 动态曲线图来对全球疫情进行演示

39.1 数据来源

本次我们主要使用动态曲线来可视化分析疫情的发展情况，疫情的数据来源于开源项目 Akshare 。为了代码和演示情况能够复现，这里我提供了保存好的数据供大家练习使用，**数据文件在文末提供了获取方式。**

39.2 准备工作

照例，还是先介绍下我运行的环境。

- Mac 系统
- Anaconda（Python 3.7）
- Jupyter Notebook

我是在 Jupyter Notebook 中运行代码的，本次使用到的 Python 库包括 akshare, pandas, plotly 等，首先我们需要将这些工具进行导入。

```
import akshare as ak
import pandas as pd
import plotly
from plotly.offline import iplot, init_notebook_mode,plot
import plotly.express as px
from datetime import datetime
import numpy as np
import plotly.graph_objs as go

init_notebook_mode()
```

本次，Lemon 使用的几个 Python 库的版本如下：

```
print(f'pandas version: {pd.__version__}')
print(f'akshare version: {ak.__version__}')
print(f'plotly version: {plotly.__version__}')

# pandas version: 1.0.1
# akshare version: 0.4.27
# plotly version: 4.5.0
```

接着，我们读取已获得的数据（已保存的数据是截至 3 月 28 日）。

```
# 从 akshare 获取数据
# df_all_history = ak.epidemic_history()

# 从csv文件获取数据，这个数据文件的数据截止到3月28日
df_all_history = pd.read_csv('epidemic_all_20200328.csv',index_col=0)

df_all_history
```

获取数据后，根据本次数据分析和可视化的目的，Lemon 对数据进行了初步整理

```
# 整理数据

# 将数据复制一份
df_all = df_all_history

# 将字符串格式的日期 另保存为一列
# df_all['dates'] = df_all_history['date']
```

```
8
9  # 将字符串格式的日期转换为 日期格式
10 df_all['date'] = pd.to_datetime(df_all['date'])
11
12 # 将时间格式转为字符串格式的日期，以 YYYY-mm-dd 的形式保存
13 df_all['dates'] = df_all['date'].apply(lambda x:x.strftime('%Y-%m-%d'))
14
15 # 添加现存确诊列
16 df_all['current'] = df_all['confirmed'] - df_all['cured'] - df_all['dead']
17
18 df_all.fillna('', inplace=True)
19
20 print(df_all.info())
21
22 df_all
```

39.3 获取国外和国内的疫情数据

上面的数据，是全球的数据，其中也包括国内各个省市的数据。我们可以将数据进行整理，分别提取出中国和海外国家的数据。

```
1 # 国内总计数量
2 df_china_total = df_all.query("country=='中国' and province==''")
3 df_china_total = df_china_total.sort_values('date',ascending=False)
4 # df_china_total
5
6 # 国外，按国家统计
7 df_oversea = df_all.query("country!='中国'")
8 df_oversea.fillna(value="", inplace=True)
9 # df_oversea
```

进一步，我们可以梳理出海外国家的总计概况，同时，将国内的数据也提取出相应的字段。

```
1  df_oversea_total = df_oversea.groupby(['date','dates'])['confirmed','cured','
       dead','current'].sum()
2  df_oversea_total.reset_index(level=1,inplace=True)
3  df_oversea_total['district'] = 'oversea'
4  # df_oversea_total
5
6  df_china_sum = df_china_total[['date','dates','confirmed','cured','dead','
       current']]
7  df_china_sum.set_index('date',inplace=True)
8  df_china_sum['district'] = 'China'
9  # df_china_sum
10
11 df_total = df_oversea_total.append(df_china_sum)
12 df_total.sort_index(ascending=True,inplace=True)
13 df_total
```

由于国外从 1 月 16 日起，才开始有统计数据，因此我们的可视化从这个日期开始。

```
# 国外从 1月16日起，才开始有统计数据
df_total_analysis = df_total['20200116':]
df_total_analysis
```

date	dates	confirmed	cured	dead	current	district
2020-01-16	2020-01-16	45	15	2	28	China
2020-01-16	2020-01-16	2	0	0	2	oversea
2020-01-17	2020-01-17	62	19	2	41	China
2020-01-18	2020-01-18	198	24	3	171	China
2020-01-19	2020-01-19	275	25	4	246	China
...	...	...	...	...	...	...

Figure 39.2: covid-19

39.4 疫情可视化

我们先来用 plotly express 对海外国家和国内情况的发展来做一个总的概览。

```
fig_total = px.line(df_total_analysis, x='dates', y='confirmed', line_group='
    district',
                    color='district', color_discrete_sequence=px.colors.
                        qualitative.D3,
                    hover_name='district',template='plotly_white',
                   width=500,height=600,
                   title=dict(text='Covid-19-trend',
                              font=dict(size=16,color='#0071c1'),
                             x=0.5)
                    )

fig_total.show()
```

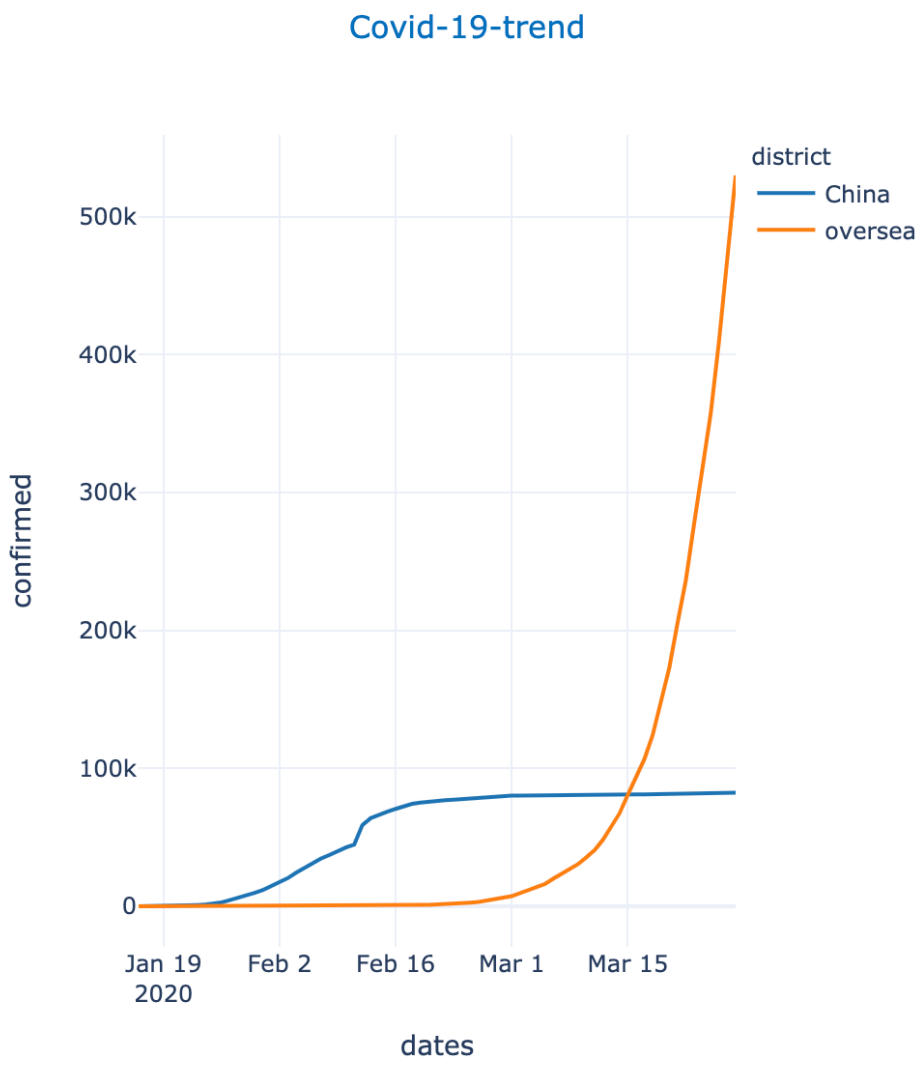


Figure 39.3: covid-19

从上面的趋势来看，国内基本趋稳了，而海外国家作为总体来看，还在快速发展。

> 这里仅仅是做简单的可视化，至于将中国和海外国家总体做对比分析，是否合理，这里只是个示例，不做进一步阐述。

上图中的曲线，并不能动态的演示变化过程，在 `px.line` 中，Lemon 也研究了下，暂时没有实现将曲线进行动态可视化的功能。

下面，Lemon 将用 Plotly 原生的功能来实现这个效果。

39.4.1 动态曲线演示疫情情况

在 Plotly 中，将曲线（Line）进行动态演示，需要通过几个步骤来实现

1. 对初始状态进行可视化，每条曲线将起始的两个点绘制成曲线；
2. 通过构造字典的形式，在 frames 中实现曲线的动态变化；
3. 添加演示按钮，

代码如下：

```
# 日期
d = datetime.today().day
m = datetime.today().month
text_today = f'数据统计截止{m}月{d}日'

# 海外数据
df_oversea_analysis = df_total_analysis.query('district=="oversea"')
# 国内数据
df_cn_analysis = df_total_analysis.query('district=="China"')

# 计算 最大的确诊人数
y_max_cn = df_cn_analysis['confirmed'].max()
y_max_oversea = df_oversea_analysis['confirmed'].max()
y_max = max(y_max_cn,y_max_oversea)

# China
trace1 = go.Scatter(x=df_cn_analysis.index[:2],
                    y=df_cn_analysis['confirmed'][:2],
                    mode='lines',
                    name='China',
                    line=dict(width=1.5,
                              color='#FFD300'))

# oversea
trace2 = go.Scatter(x = df_oversea_analysis.index[:2],
                    y = df_oversea_analysis['confirmed'][:2],
                    mode='lines', # markers+lines
                    name='Oversea',
                    line=dict(width=1.5))

frames = [dict(data= [dict(type='scatter',
                           x=df_cn_analysis.index[:k+1],
                           y=df_cn_analysis['confirmed'][:k+1]),
                      dict(type='scatter',
                           x=df_oversea_analysis.index[:k+1],
                           y=df_oversea_analysis['confirmed'][:k+1])],
               traces= [0, 1],
               # 0: frames[k]['data'][0], 代表 trace1, 1: frames[k]['data'][1], trace2
              )for k  in  range(1, len(df_cn_analysis))]

```

```
layout = go.Layout(width=500,
                   height=600,
                   showlegend=True,
                   template='plotly_dark',
                   hovermode='closest',
                   updatemenus=[dict(type='buttons', showactive=False,
                                     y=1.10,
                                     x=1.15,
                                     xanchor='right',
                                     yanchor='top',
                                     pad=dict(t=0, r=10),
                                     buttons=[dict(label='Play',
                                                   method='animate',
                                                   args=[None,
                                                         dict(frame=dict(duration
                                                             =100,
                                                                         redraw=
                                                                             False),
                                                              transition=dict(
                                                                  duration=1),
                                                              fromcurrent=True,
                                                              mode='immediate')])])
                                                                  ],
                   )

layout.update(xaxis =dict(range=[df_cn_analysis.index[0],
                                 df_cn_analysis.index[len(df_cn_analysis)-1]+pd
                                     .Timedelta(days=2)
                                 ],
                          autorange=False),
              yaxis =dict(range=[0, y_max*1.05], autorange=False))
fig = go.Figure(data=[trace1, trace2], frames=frames, layout=layout)
fig.show()
```

上面代码中，go.Layout 中的 duration 可以来控制按钮点击后变化的速度。

运行上述代码后，得到的效果如下：

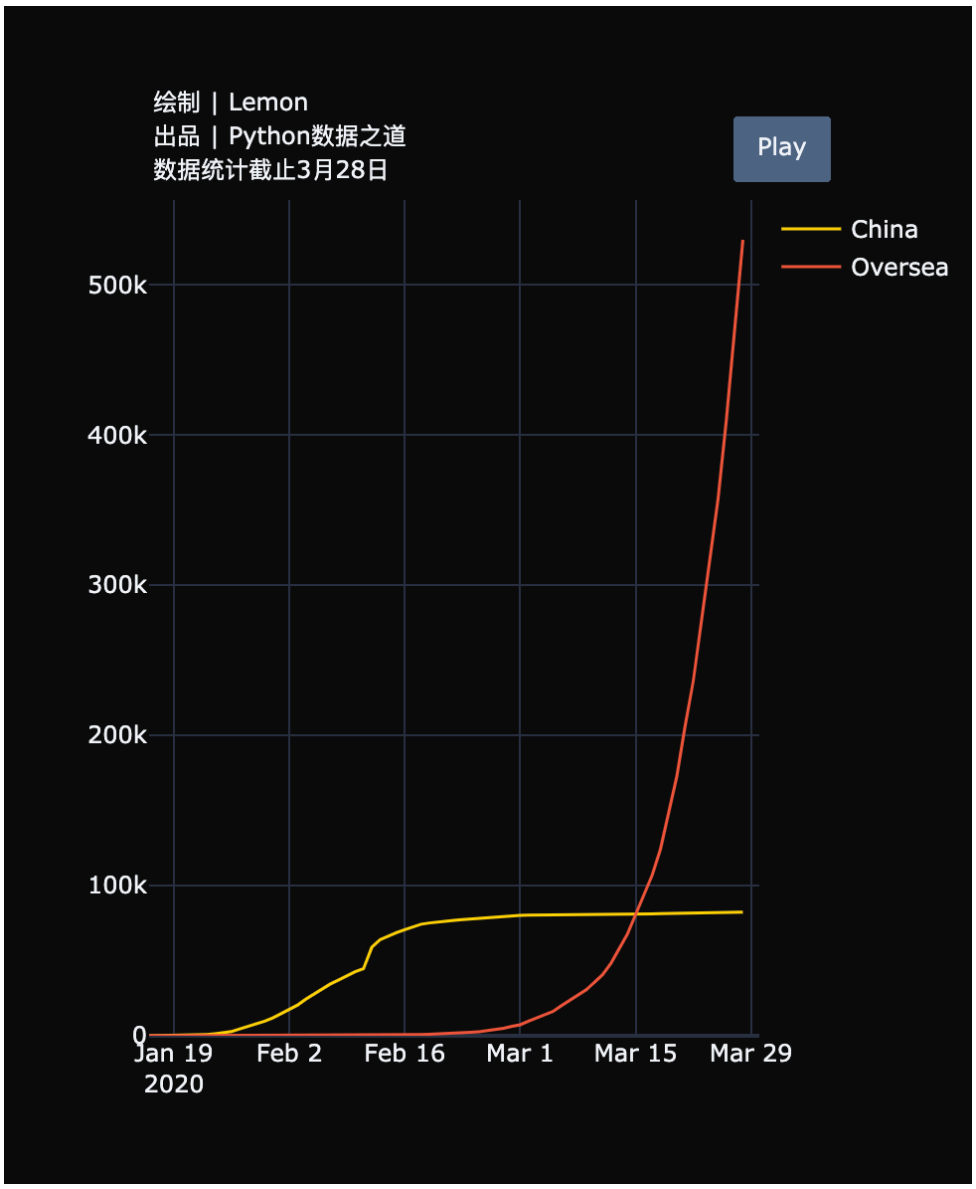


Figure 39.4: covid-19

曲线可以动起来了，是不是很棒啊。

上面的动态曲线是两条曲线的，这里给大家留一个作业，如果是 3 条或者 3 条以上的曲线，该如何实现动态演示，大家可以思考下这个问题。

多条动态曲线的效果如下：

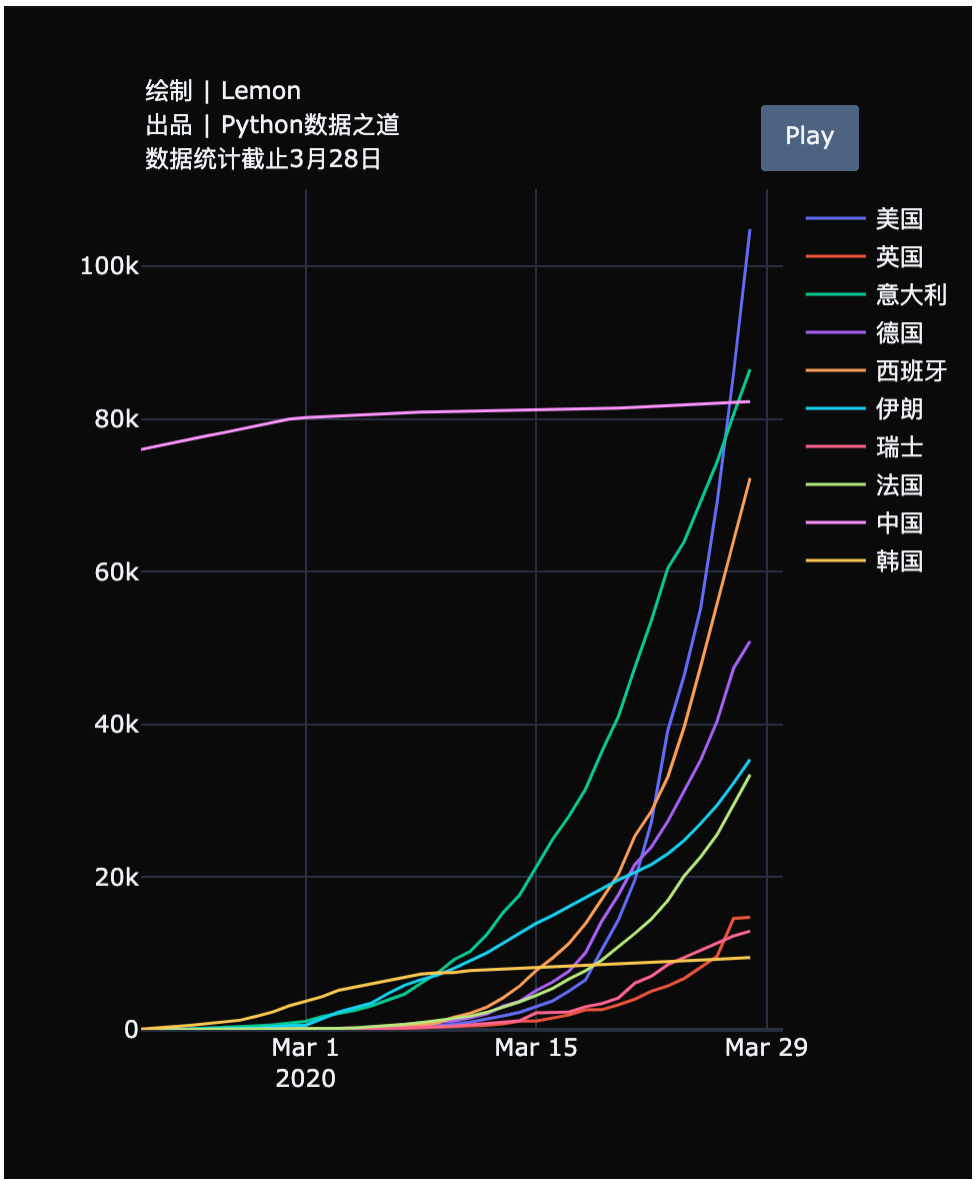


Figure 39.5: covid-19

> PS：如果曲线数量较少，你可以手动添加几条曲线，如果曲线数量较多，或者数量经常变化，则需要更加灵活的方式来实现。

为方便大家进行操作，Lemon 给大家提供了本文的数据文件，可以在公众号 **「Python 数据之道」**后台回复数字「660」进行获取。

40 升级版，用 Python 来进行多条曲线动态演示全球疫情变化

2020 年 3 月左右，我写了几篇用 Python 的交互式可视化工具 Plotly 来演示全球疫情情况的文章，如下：

- 用 Python 可视化神器 Plotly 动态演示全球疫情变化趋势
- 用 Plotly 动态柱状图来演示全球疫情变化趋势
- 超火动态排序疫情变化图，这次我们用 Plotly 来绘制
- 用 Python 动态曲线图来对全球疫情进行演示

如今（截至 2020 年 3 月 31 日），全球确诊数量将近 80 万人，美国确诊数量已超过 16 万人，成为目前人数最多的国家，美国疫情的快速发展也给全球增添了更大的不确定性。

今天，Lemon 继续来分享用 Plotly 对疫情情况进行可视化。作为上次文章内容的升级版，本次我们用 Plotly 来实现多条曲线的动态演示。先来看最终的效果：

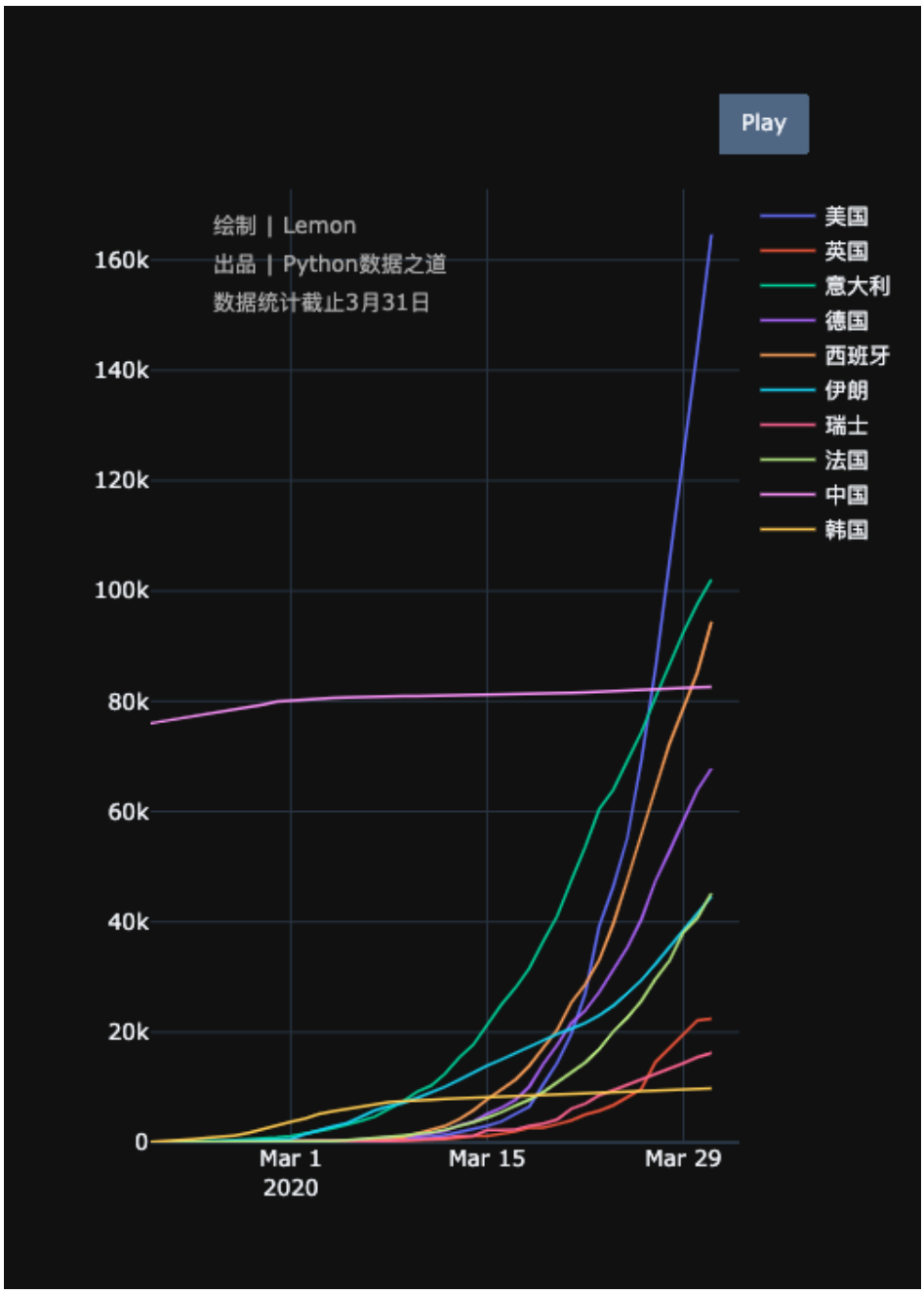


Figure 40.1: covid-19

上图的动态效果，请点击下面链接查看:

https://mp.weixin.qq.com/s/hUkk99SXN0kncQOCj8gHXw

40.1 数据来源

本次我们主要使用动态曲线来可视化分析疫情的发展情况，疫情的数据来源于开源项目 Akshare 。为了代码和演示情况能够复现，这里我提供了保存好的数据供大家练习使用，**本文的完整代码及数据文件在文末提供了获取方式。**

40.2 准备工作

照例，还是先介绍下我运行的环境。

- Mac 系统
- Anaconda（Python 3.7）
- Jupyter Notebook

我是在 Jupyter Notebook 中运行代码的，本次使用到的 Python 库包括 akshare, pandas, plotly 等，首先我们需要将这些工具进行导入。

```
import akshare as ak
import pandas as pd
import plotly
from plotly.offline import iplot, init_notebook_mode,plot
import plotly.express as px
from datetime import datetime
import numpy as np
import plotly.graph_objs as go

init_notebook_mode()
```

本次，Lemon 使用的几个 Python 库的版本如下：

```
print(f'pandas version: {pd.__version__}')
print(f'akshare version: {ak.__version__}')
print(f'plotly version: {plotly.__version__}')

# pandas version: 1.0.1
# akshare version: 0.4.27
# plotly version: 4.5.0
```

接着，我们读取已获得的数据（已保存的数据是截至 3 月 31 日）。

```
# 从 akshare 获取数据
# df_all_history = ak.epidemic_history()

# 从csv文件获取数据，这个数据文件的数据截止到3月31日
df_all_history = pd.read_csv('epidemic_all_20200331.csv',index_col=0)

df_all_history
```

获取数据后，根据本次数据分析和可视化的目的，Lemon 对数据进行了初步整理

```
# 整理数据

# 将数据复制一份
df_all = df_all_history

# 将字符串格式的日期 另保存为一列
# df_all['dates'] = df_all_history['date']

# 将字符串格式的日期转换为 日期格式
df_all['date'] = pd.to_datetime(df_all['date'])

# 将时间格式转为字符串格式的日期，以 YYYY-mm-dd 的形式保存
df_all['dates'] = df_all['date'].apply(lambda x:x.strftime('%Y-%m-%d'))

# 添加现存确诊列
df_all['current'] = df_all['confirmed'] - df_all['cured'] - df_all['dead']

df_all.fillna('', inplace=True)

print(df_all.info())

df_all
```

40.3 获取国外和国内的疫情数据

上面的数据，是全球的数据，其中也包括国内各个省市的数据。我们可以将数据进行整理，分别提取出中国和海外国家的数据。

```
# 国内总计数量
df_china_total = df_all.query("country=='中国' and province==''")
df_china_total = df_china_total.sort_values('date',ascending=False)
# df_china_total

# 国外，按国家统计
df_oversea = df_all.query("country!='中国'")
df_oversea.fillna(value="", inplace=True)
# df_oversea

# # 全球总计
df_global =  df_china_total.append(df_oversea)
df_global = df_global.sort_values(['country','date'])
df_global
```

获取全球最近日期的数据：

```
# 按日期进行排序
df_global_date = df_global.sort_values('date',ascending=False)

```

```
# 获取最新的日期
latest_day = df_global_date['dates'].values[0]

# query 函数中，变量需要加 @ ，
df_global_latest = df_global_date.query('dates==@latest_day')

df_global_latest = df_global_latest.sort_values('confirmed',ascending=False)

df_global_latest.head(10)
```

进一步，我们可以梳理出海外国家的总计概况，同时，将国内的数据也提取出相应的字段。

```
df_oversea_total = df_oversea.groupby(['date','dates'])['confirmed','cured','
    dead','current'].sum()
df_oversea_total.reset_index(level=1,inplace=True)
df_oversea_total['district'] = 'oversea'
# df_oversea_total

df_china_sum = df_china_total[['date','dates','confirmed','cured','dead','
    current']]
df_china_sum.set_index('date',inplace=True)
df_china_sum['district'] = 'China'
# df_china_sum

df_total = df_oversea_total.append(df_china_sum)
df_total.sort_index(ascending=True,inplace=True)
df_total
```

由于国外从 1 月 16 日起，才开始有统计数据，因此我们的可视化从这个日期开始。

```
# 国外从 1月16日起，才开始有统计数据
df_total_analysis = df_total['20200116':]
df_total_analysis
```

date	dates	confirmed	cured	dead	current	district
2020-01-16	2020-01-16	45	15	2	28	China
2020-01-16	2020-01-16	2	0	0	2	oversea
2020-01-17	2020-01-17	62	19	2	41	China
2020-01-18	2020-01-18	198	24	3	171	China
2020-01-19	2020-01-19	275	25	4	246	China
...	...	...	...	...	...	...

Figure 40.2: covid-19

40.4 疫情可视化

我们先来用 plotly express 对海外国家和国内情况的发展来做一个总的概览。

```
fig_total = px.line(df_total_analysis, x='dates', y='confirmed', line_group='
    district',
                      color='district', color_discrete_sequence=px.colors.
                          qualitative.D3,
                      hover_name='district',template='plotly_white',
                     width=500,height=600,
                     title=dict(text='Covid-19-trend',
                                font=dict(size=16,color='#0071c1'),
                               x=0.5)
                      )

fig_total.show()
```

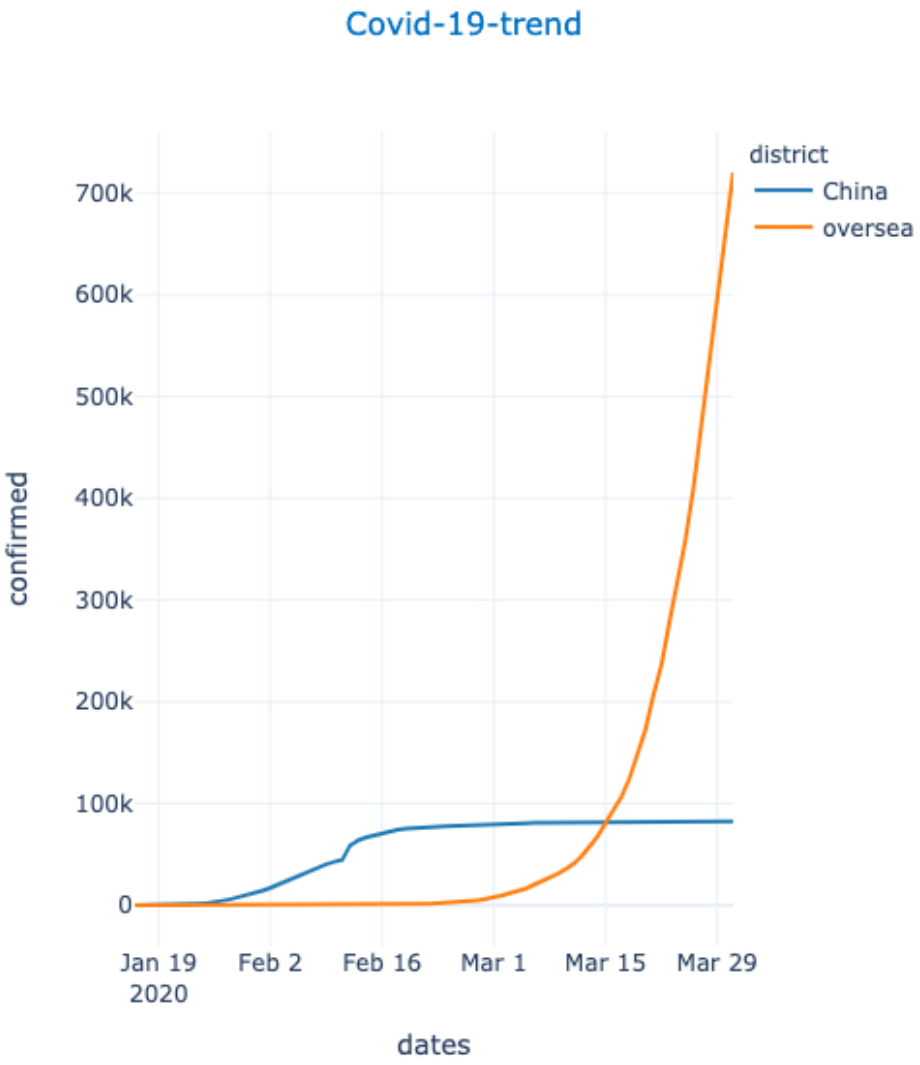


Figure 40.3: covid-19

从上面的趋势来看，国内基本趋稳了，而海外国家作为总体来看，还在快速发展。

> 这里仅仅是做简单的可视化，至于将中国和海外国家总体做对比分析，是否合理，这里只是个示例，不做进一步阐述。

上图中的曲线，并不能动态的演示变化过程，在 `px.line` 中，Lemon 也研究了下，暂时没有实现将曲线进行动态可视化的功能。

下面，Lemon 将用 Plotly 原生的功能来实现多条曲线的动态可视化。

40.5 动态曲线演示疫情情况

在 Plotly 中，将曲线（Line）进行动态演示，需要通过几个步骤来实现

1. 对初始状态进行可视化，每条曲线将起始的两个点绘制成曲线；
2. 通过构造字典的形式，在 `frames` 中实现曲线的动态变化；
3. 添加演示按钮，

多条曲线的动态可视化，关键是如何便捷的添加多条曲线，尤其考虑有时曲线数量较多，有时数量经常发生变化。

针对上述情况，我们需要将程序设计的相对灵活些，这里介绍的方法，主要是将 `dataframe` 以 `value` 的形式装入到字典里。

至于曲线的多少，可以将元素包含在列表 `list` 中，这样就可以灵活的实现曲线数量的变化。

多条曲线动态演示的代码如下：

```
# 日期
m = df_global_latest.iloc[0]['date'].month
d = df_global_latest.iloc[0]['date'].day
text_today = f'数据统计截止{m}月{d}日'

df_compare = df_global.set_index('date')
df_compare = df_compare['20200220':]

list_countries = ["美国","英国","意大利","德国","西班牙","伊朗","瑞士","法国","
    中国","韩国"]
n_countries = len(list_countries)
n_list = [i for i in range(n_countries)] # 曲线数量编号
# print(n_list)

n_frame={}

for country_i in list_countries:
    df_i = df_compare.query('country==@country_i') # @ 代表变量
    df_i = df_i.sort_index(ascending=True)
    n_frame[country_i] = df_i

# 计算 最大的确诊人数
# 初始值为0
y_max = 0
for country_i in list_countries:
    y_max_i = n_frame[country_i]['confirmed'].max()
    y_max = max(y_max,y_max_i)

df_01 = n_frame[list_countries[0]] # 第一个国家
# print(df_01)

traces = [go.Scatter(
                    x = value.index[:2],   # value 是 国家的数据，dataframe
                    y = value['confirmed'][:2],
```

```
                      mode='lines', # markers+lines
                      name=key,  # key 是国家名称
                      line=dict(width=1.5)
                      ) for key,value in n_frame.items()
          ]

frames = [dict(data= [dict(type='scatter',
                           x=value.index[:k+1],
                           y=value['confirmed'][:k+1]) for key,value in n_frame
                               .items()
                     ],
               traces= n_list,
              )for k  in  range(1, len(n_frame[list_countries[0]]))]

layout = go.Layout(width=500,
                   height=700,
                   showlegend=True,
                   template='plotly_dark',
                   hovermode='closest',
                   updatemenus=[dict(type='buttons', showactive=False,
                               y=1.10,
                               x=1.15,
                               xanchor='right',
                               yanchor='top',
                               pad=dict(t=0, r=10),
                               buttons=[dict(label='Play',
                                             method='animate',
                                             args=[None,
                                                   dict(frame=dict(duration
                                                       =100,
                                                                   redraw=
                                                                       False),
                                                        transition=dict(
                                                            duration=1),
                                                        fromcurrent=True,
                                                        mode='immediate')])])
                                                            ],
                   )

layout.update(xaxis =dict(range=[df_01.index[0],
                                 df_01.index[len(df_01)-1]+pd.Timedelta(days=2)
                                ],
                          autorange=False),
              yaxis =dict(range=[0, y_max*1.05], autorange=False))
fig = go.Figure(data=traces, frames=frames, layout=layout)

# Pycharm，VS Code， Spider 等模式下
# Jupyter Notebook 下也可以用
plot(fig,filename='covid-19-multiple-lines.html')

# Jupyter Notebook 中
```

```
80  # fig.show()
```

上面代码中，`go.Layout` 中的 `duration` 可以来控制按钮点击后变化的速度。

运行上述代码后，得到的效果如下：

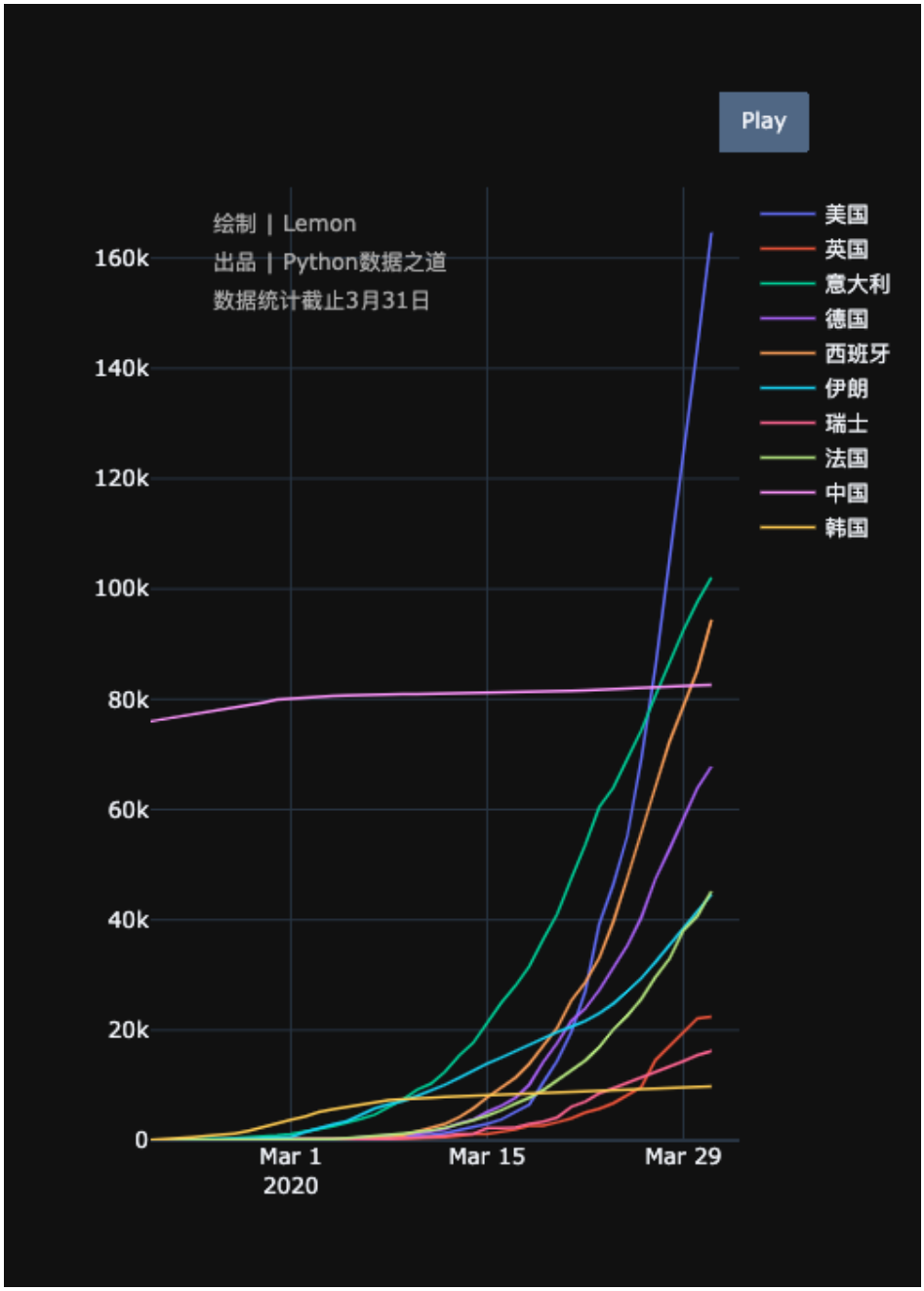


Figure 40.4: covid-19

曲线可以动起来了，是不是很棒啊。

为方便大家进行操作，Lemon 给大家提供了本文的 PDF 版内容（含完整的代码）以及数据文件，可以在公众号**「Python 数据之道」**后台回复数字「661」进行获取。

41 Plotly 中 4 种文本类型设置详解

在过去的一段时间里，我写了几篇用 Python 的交互式可视化工具 Plotly 来演示全球疫情情况的文章，如下：

- 用 Python 可视化神器 Plotly 动态演示全球疫情变化趋势
- 用 Plotly 动态柱状图来演示全球疫情变化趋势
- 超火动态排序疫情变化图，这次我们用 Plotly 来绘制
- 用 Python 动态曲线图来对全球疫情进行演示
- 升级版，用 Python 来进行多条曲线动态演示全球疫情变化

有不少同学都问到图表中文本如何设置，今天 Lemon 来跟大家分享下。

Plotly 中文本设置大概可以分为四种类型，效果如下：

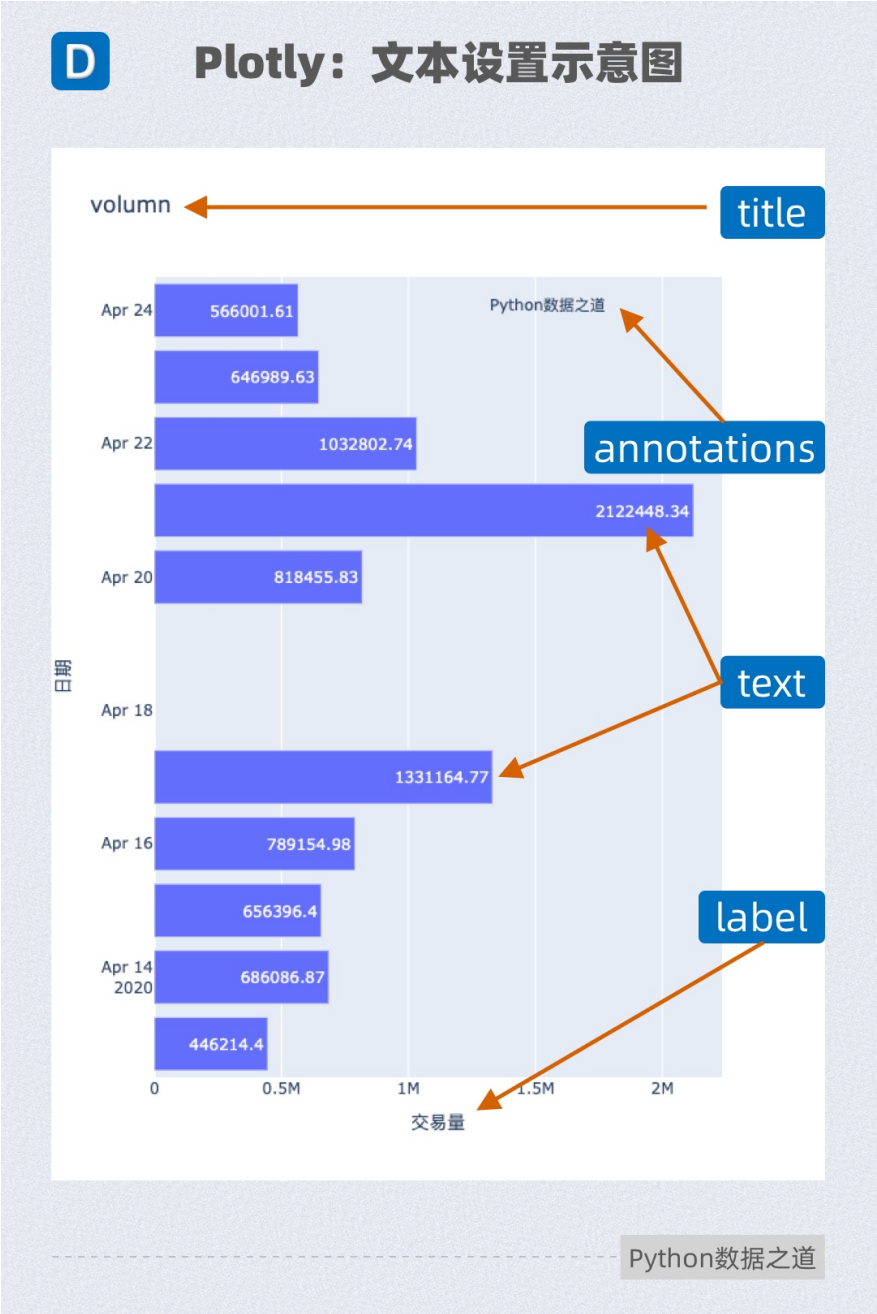


Figure 41.1: img1

下面将以 Plotly Express 中的设置为基础，来介绍这 4 种类型的详细情况。

41.1 项目环境背景

本文的数据来自开源项目 tushare，从 tushare 中获取数据，首先要进行注册获取 token（一串字母和数字组成的文本），然后才可以获取到数据，大家可以通过以下链接来注册：

https://tushare.pro/register?reg=129033

本文的代码是在 Jupyter notebook 环境中运行，Python 环境版本及其他工具如下：

- Python 3.7
- pandas version: 1.0.3
- tushare version: 1.2.51
- plotly version: 4.5.0

首先导入相关工具：

```
import tushare as ts
import plotly
from plotly.offline import iplot,plot, init_notebook_mode
import plotly_express as px
import plotly.graph_objs as go
import pandas as pd
import plotly.io as pio

pio.templates.default = 'plotly_white'
init_notebook_mode()
```

41.2 数据获取

在 tushare 中注册后，通过 “个人主页”——“接口 TOKEN” 可以找到自己的 token 值，界面如下：

Figure 41.2: img2

复制 token 值，然后在代码中进行如下设置：

```
# 设置 token
# tushare 注册地址： https://tushare.pro/register?reg=129033
# 以上方法只需要在第一次或者token失效后调用，完成调取tushare数据凭证的设置，正常情况下不需要重复设置。
ts.set_token('你的token值')

pro = ts.pro_api()
```

在设置好 token 值后，我们就可以开始获取数据，这里以获取股票代码是“000001.SZ”的企业为例，来描述 Plotly 中文本设置。

数据获取及初步整理如下：

```
df = pro.daily(ts_code='000001.SZ',
               start_date='19900101',
               end_date = '20200425'
              )
# 设置时间格式
df['trade_date'] = pd.to_datetime(df['trade_date'])
df
```

41.3 Plotly 中文本设置

这次，我们以 Plotly express 为例，来阐述文本的设置，在原生的 plotly 中，文本设置的方式也是一致的。

文本的设置分为 4 中类型，分别是 `title`，`text`，`label`，以及 `annotations`。

`title` 是图表的标题；`text` 是显示 x 轴或 y 轴数据的值；`label` 是设置数据列在图表中的显示名称；`annotations` 是设置图表中的注释文本。

这 4 种类型的文本，前 3 种相对而言比较好设置，`annotations` 要稍微复杂些。

下面以柱状图为例来进行说明，代码如下：

```
df_bar = df[:10]
# df_bar

fig_bar = px.bar(df_bar,
                 x='vol',
                 y='trade_date',
                 orientation='h',
                 text='vol',      # 设置柱状图柱子上的显示文本
                 title='volumn', # 设置图表标题
                 width=600,
                 height=800,
#                  template='plotly_white',
                 labels={'vol':'交易量',       # 设置标签显示名称
                         'trade_date':'日期',
```

```
                            }
                    )

# 设置文字注释内容
annotation = [dict(x=0.8,
                    xref='paper',  #使用相对坐标
                    y=0.98,
                    yref='paper',
                    text='Python数据之道',
                    showarrow=False,  # 不显示箭头

                    ),
                ]

fig_bar.update_layout(annotations= annotation)

fig_bar.show()
```

效果如下：

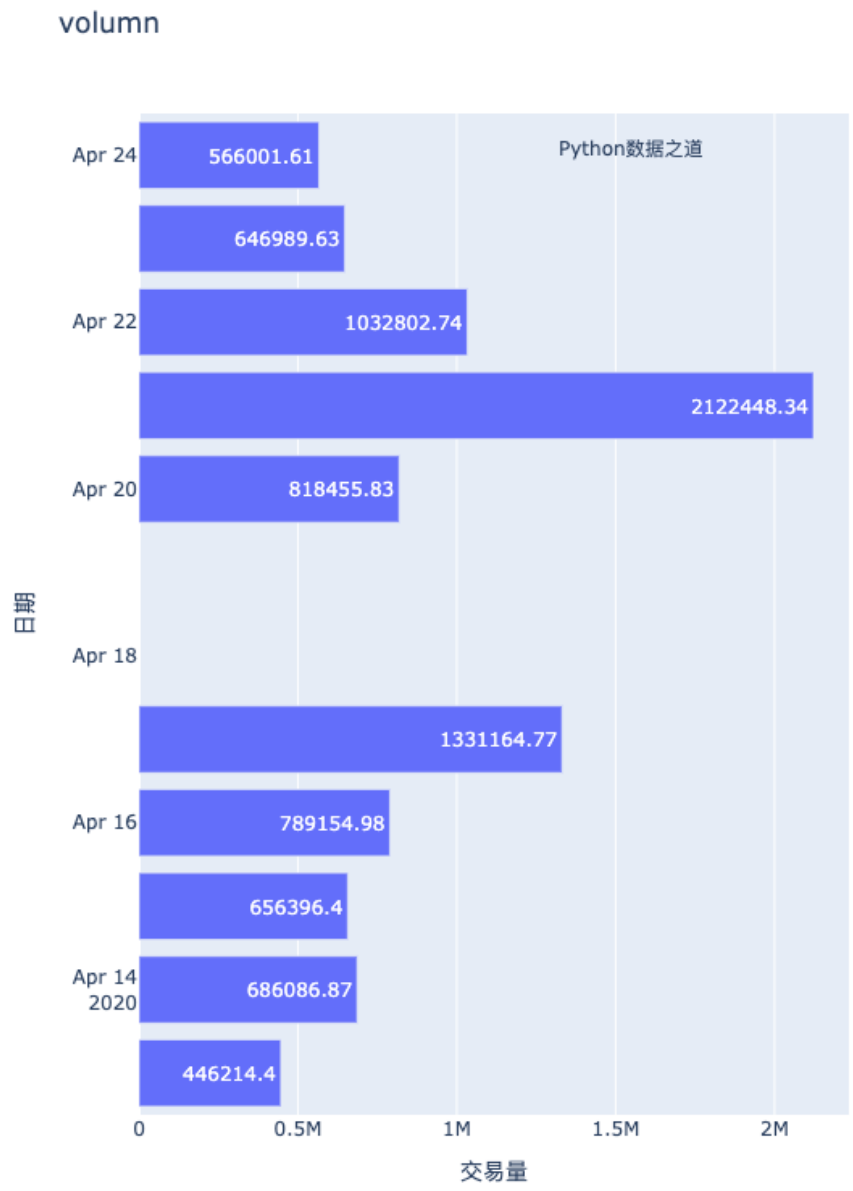


Figure 41.3: img3

上图中，对 4 种类型的文本进行了初步设置，当然你可能觉得有点丑，想更个性化的设置，比如标题（title）居中、设置标题文本颜色，等等。

下面以股票收盘价为例，描述在线形图中的文本设置情况，详细代码如下：

```
fig = px.line(df,
              x='trade_date',
```

```
                y='close',
                title=dict(text='交易价格',
                           x=0.5,
                           font=dict(size=20,
                                     color='blue',
                                    )
                          ),
                color='ts_code',
                color_discrete_sequence=['#0071c1'],
                template='plotly_white',
                labels={'close':'收盘价',
                        'trade_date':'日期',
                       }
               )

annotations = [dict(x=0.1,
                    xref='paper',  #使用相对坐标
                    y=0.98,
                    yref='paper',
                    text='Python数据之道',
                    font=dict(size=14,
                              color='red',
                             ),
                    textangle=40,
                    showarrow=False,  # 不显示箭头

                    ),

                # 标注
                 dict(
                     xref='x', # 使用绝对坐标系
                     x='2007-10-31',
                     yref='y',
                     y=48.05,
                     text='high position',
                     font=dict(
                         size=12,
                         color='#ffffff',
                     ),
    #                 textangle=45,
                     showarrow=True,
                     arrowcolor='#595959',
                     arrowwidth=3,
                     arrowhead=1,
                     ax=-40,
                     ay=-40,
                     bgcolor='#d66101',
                     borderpad=4,
    #                                  bordercolor="#c7c7c7",
    #                                  borderwidth=2,
                     ),
```

```
54                ]
55
56  fig.update_layout(showlegend=False,
57                    annotations =annotations,
58                   )
59
60  fig.show()
```

图示效果如下：

Figure 41.4: img4

42 推荐一个牛逼的生物信息 Python 库 - Dash Bio

Dash Bio 是一个免费的开源 Python 库，用于生物信息学和药物开发应用。

要开始使用 Dash Bio，请使用 `pip install dash_bio` 安装，然后转到 Dash Bio 的文档：http://dash.plot.ly/dash-bio

如果您是 Dash 的新手，您可能需要从 Dash 的入门指南开始：https://dash.plot.ly/getting-started

自发布以来，Plotly 用于 Python 的开源应用程序构建软件 Dash 在生物信息学家和药物开发人员中一直在稳步上升。交互式 Web 图形、Python 创作和易用性的结合使 Dash 成为生物信息学家工具箱的自然之选。

例如，麻省理工大学的吉福德实验室（Gifford Lab）已经发布了一个完全用 Dash 编写的 CRISPR 预测工具。麻省理工大学的团队在 Nature 中详细介绍了他们的方法，并免费提供在线的 Dash 应用程序和 Python 代码。

Figure 42.1: image01

图片来源：theinDelphi CRISPR prediction app

同样，南卡罗来纳医科大学的 Hammer 实验室正在开发一种带有 Dash 的开源细胞显微镜工具，并在 bioRxiv 上发表了他们的方法：

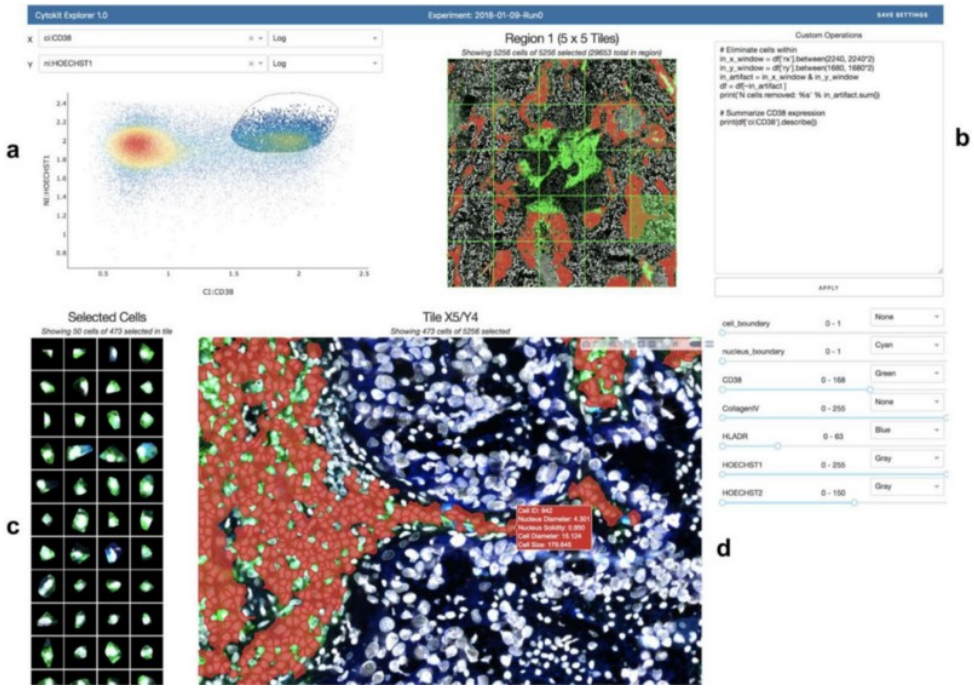


Figure 42.2: image02

去年夏天，Plotly 与加拿大研究机构 Canadian Research Chair Aïda Ouangraou 合作，利用 Dash 开发新的基因组数据可视化开源工具。该研究尚未发布，但您可以在该机构网站上查看新闻。

今年，Plotly 正在利用 Dash Bio 重建其对生命科学的承诺 - Dash Bio 是一个用于在 Python 中构建生物信息学和药物开发应用程序的开源工具包。

许多 Dash Bio 组件都建立在 JavaScript 库之上，这些库已经在全栈、生物信息学应用程序开发人员中流行。我们重新设计了这些 JavaScript 小部件，以便 Python 开发人员现在可以访问它们。换句话说，科学计算的 Python 开发人员现在可以将这些小部件工作，而无需了解 JavaScript 。所有你需要知道的是 Python 和 Dash 。

生命科学软件的许多重量级工具都是基于 Java 的桌面程序 - `PyMol` 用于生物制剂开发，`ImageJ` 用于显微镜图像分析，`IGV` 用于基因组数据查看是一些例子。使用 Dash 可以为 Web 编写未来版本的应用程序，完全使用 Python ，在移动设备上工作，并根据特定的研究目标进行定制。由于 Dash 是开源的，因此整个软件应用程序的代码可以以同行评审的方式自由分发和发布。CRISPR、NGS 和生物制剂在生命科学领域引发了新的创新和商业化浪潮– Dash 是基于 Python 的分析库，可以跟上。

以下是 12 个 Dash 应用程序，这些应用程序显示了这种基于 Web、敏捷和交互式的生物信息学和药物开发分析方法。有了 Dash Bio，我们希望尽可能扩大可能性。

42.1 探索 3d 状态下的小分子

这个 Dash 应用程序从磁盘、数据库或 Python 中的 API 读取 PDB（“蛋白质数据库”）文件，然后在 Dash 中可视化 3d 结构。当您单击原子，旋转分子或更改结构时，Dash 会触发 Python 回调函数。您还可以突出显示单个原子

(如蛋白质的活性位点)。

- 查看 Dash app：https://dash-bio.plotly.host/dash-molecule3d
- 查看 Dash app 的 Python 代码：https://github.com/plotly/dash-bio/blob/master/tests/dash-bio_demos/app_molecule3d.py

Figure 42.3: image03

用 Dash 3d 分子组分可视化的 DNA 螺旋

42.2 分析显微镜图像中的细胞

这个 Dash 应用程序是由 “scikit-image” 背后的主要开发人员之一 Emma Gouillart 制作的。将鼠标悬停在图像中的白细胞上以突出显示相邻表格中的细胞属性。您还可以使用该表来过滤具有特定属性的单元格（例如，面积小于 1500μm^2 的单元格)。你需要安装Dash Canvas https://dash.plot.ly/canvas 来运行这个应用程序。

- Dash 程序地址：https://dash-canvas.plotly.host/object-properties/
- Python 代码地址：https://github.com/plotly/canvas-portal/blob/master/apps/object-properties/app.py
- Dash Canvas 程序地址：https://dash-canvas.plotly.host/Portal/

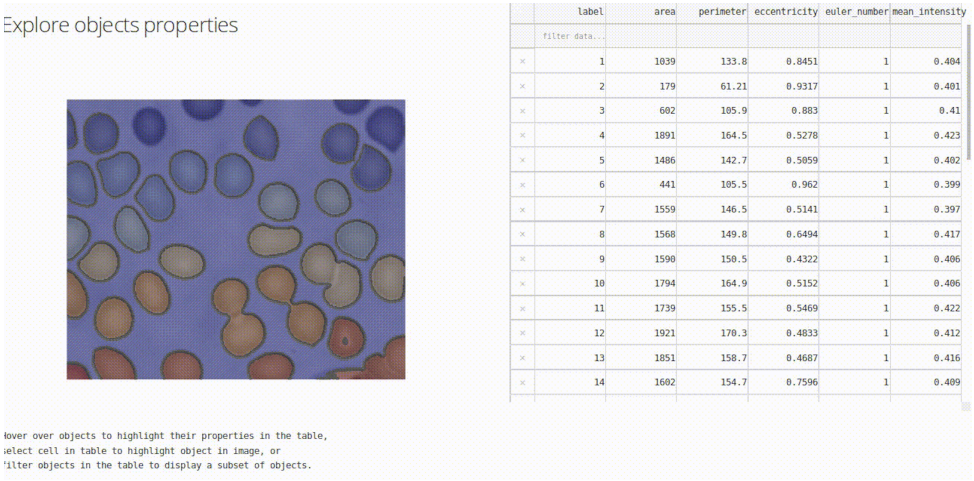


Figure 42.4: image04

光学显微镜下的单核细胞 - 用 Dash Canvas 进行交互式探索。

42.3 进行药代动力学分析

此 Dash 应用程序旨在允许进行药代动力学研究的人员输入数据（手动或通过复制粘贴）。然后显示浓度 - 时间曲线以及用 Python 计算的各种参数表。

- 程序地址：https://dash-bio.plotly.host/dash-pk-calc/
- Python 代码地址：https://github.com/plotly/dash-sample-apps/blob/master/apps/dash-pk-calc/app.py

Figure 42.5: image05

使用自定义的 Dash 应用程序简化了 pk 分析

42.4 可视化 FASTA 数据

有许多非常好的、交互式的多序列比对（MSA）工具。我们从这 3 个中获取灵感：

- MATLAB 的 `seqalignviewer` https:// www.mathworks.com/ help/ bioinfo/ ref/ seqalignviewer.html
- Seb Wilzbach's JavaScript MSA： https://msa.biojs.net/
- Jalview：http://www.jalview.org/

对于 Dash MSA Viewer，我们使用 WebGL 在浏览器中实现超快的交互式性能。由于 Dash MSA Viewer 是 Dash 组件，因此您只需要知道 Python 即可使用它。下面的 Dash 应用程序从 Python 中读取 FASTA 文件中的序列数据，然后使用 Dash MSA 查看器绘制数据。

- Dash 应用程序：https://dash-bio.plotly.host/dash-alignment-viewer/
- Python 代码：https:// github.com/ plotly/ dash-bio/ blob/ master/ tests/ dashbio_demos/ app_alignment_viewer.py

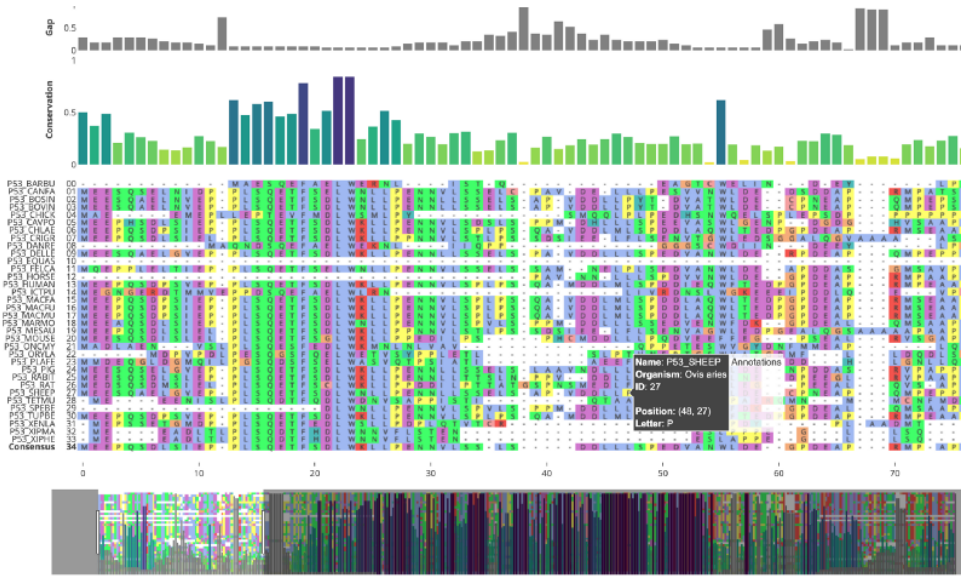


Figure 42.6: image06

使用 Dash MSA Viewer 以交互方式可视化 FASTA 数据

42.5 突出基因组相似性

Circos 图通常用于比较基因组学。在下面的 Dash 应用程序中，21 个染色体以圆圈绘制，并且它们的基因组区域之间的关系与线或带相关联。根据 `circos.ca` [http://circos.ca/guide/genomic/img/circos-conservation.png]，可以定义这些关系：

> [基因组] 位置之间的关系可以反映任何类型的对应关系。例如，它可以根据相似性（序列或蛋白质）或类别（功能或结构）来定义

由于 Dash Circos 是一个 Dash 组件，它会在 Web 浏览器中显示，而您只需要知道 Python 即可使用它构建应用程序。

- Dash 程序地址：https://dash-bio.plotly.host/dash-circos/
- Python 代码：https:// github.com/ plotly/ dash-bio/ blob/ master/ tests/ dashbio_demos/ app_circos.py

Figure 42.7: image07

使用 Dash Circos 突出基因组相似性

42.6 可视化微阵列结果

集群图是具有树形图的热图，其可视化分层数据聚类。它们通常与微阵列数据一起使用。Dash Clustergram 响应单击、悬停和缩放事件。使用 Dash Clustergram 创建应用程序，你只需要了解 Python 即可。

- Dash 程序：https://dash-bio.plotly.host/dash-clustergram/
- Python 代码：https:// github.com/ plotly/ dash-bio/ blob/ master/ tests/ dashbio_demos/ app_clustergram.py

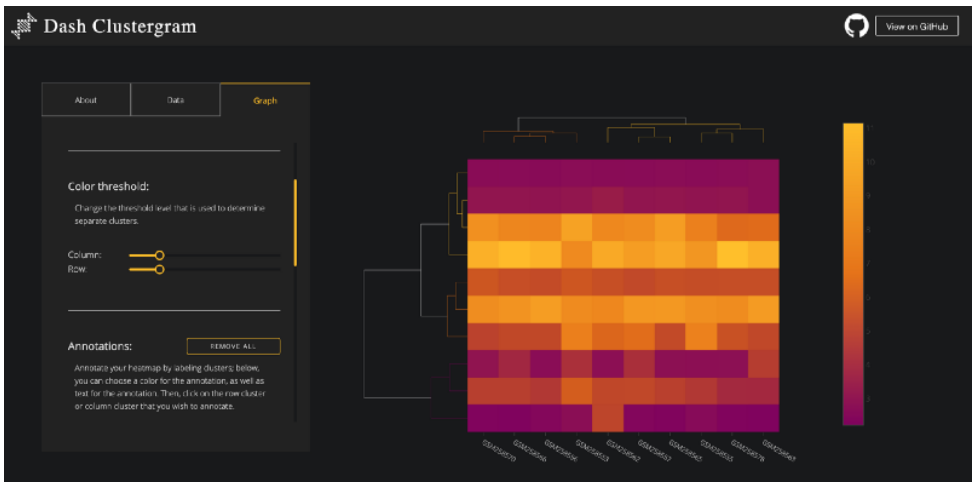


Figure 42.8: image08

使用 Dash Clustergram 可视化微阵列结果

42.7 搜索和选择序列

Dash 序列查看器简化了序列搜索和选择的 UI。该组件的核心是瑞士生物信息学研究所最初开发的 JavaScript 库。我们为 Dash 重新设计了它，以便它可以在 Python 中轻松使用。

- Dash 应用程序：https://dash-bio.plotly.host/dash-sequence-viewer/
- Python 代码：https:// github.com/ plotly/ dash-bio/ blob/ master/ tests/ dashbio_demos/ app_sequence_viewer.py

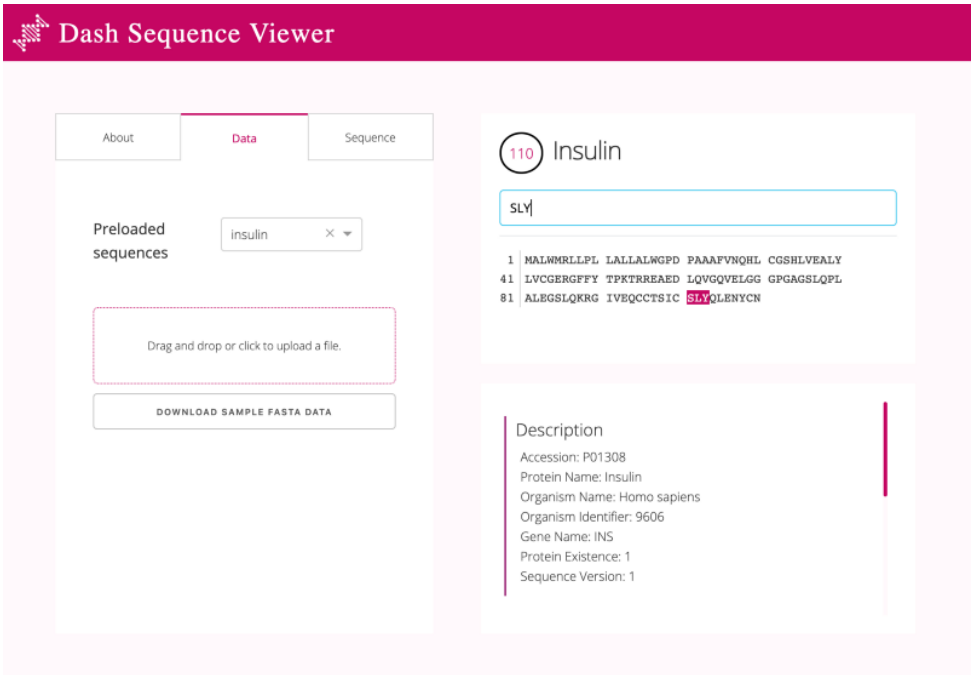


Figure 42.9: image09

使用 Dash Sequence Viewer 搜索和选择生物序列

42.8 3d 分子的环境遮挡

这个 3d 分子查看器使用 WebGL 和环境遮挡来提供更好的深度感知。最初的 JavaScript 库 - Speck- 由 Rye Terrell 开发。我们为 Dash 重新设计了这个库，使构建分析应用程序的 Python 用户可以访问它。

- 应用程序：https://dash-bio.plotly.host/dash-speck/
- 代码：https://github.com/plotly/dash-bio/blob/master/tests/dashbio_demos/app_speck.py

Figure 42.10: image10

Dash Speck 显示的比较优美的交互式 3D 分子

42.9 使用 Dash Ideogram 可视化染色体

Dash Ideogram 的核心是 Broad Institute 开发的 JavaScript 库。我们将它重新设计为 Dash 组件，以便科学计算的 Python 开发人员可以轻松访问它。

- 应用程序：https://dash-bio.plotly.host/dash-ideogram/
- 代码:https://github.com/plotly/dash-bio/blob/master/tests/dashbio_demos/app_ideogram.py

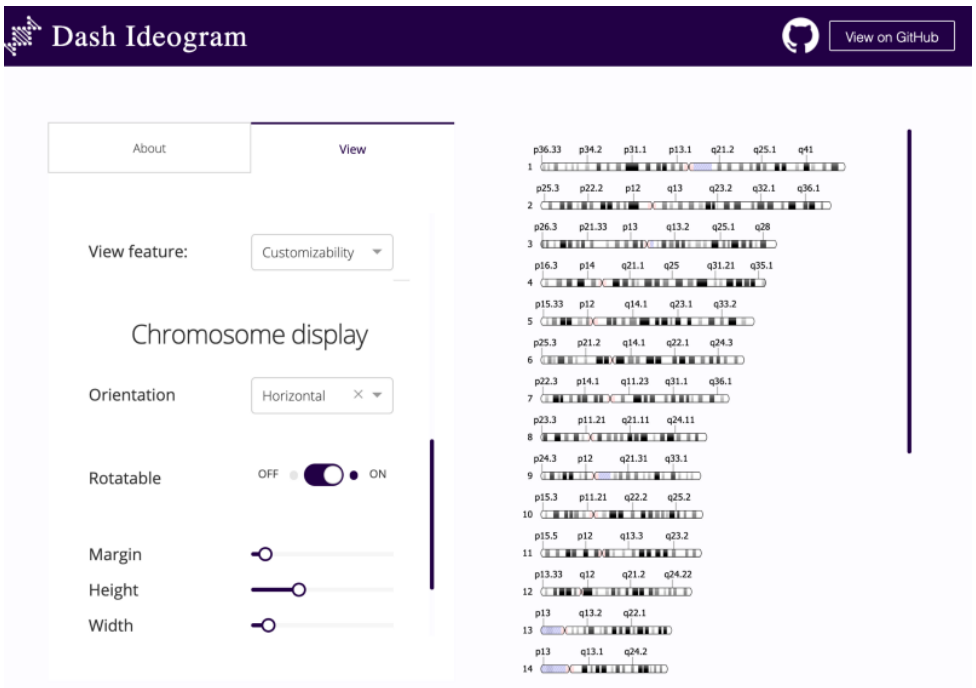


Figure 42.11: image11

在 Dash 应用程序中嵌入交互式染色体可视化

42.10 可视化基因突变

交互式针线图现在可以在 Python 中轻松组合并嵌入到 Dash 应用程序中。这个 Dash 组件的设计灵感来自 Barcelona Biomedical Genomics Lab 的 JavaScript 库 “muts-needle-plot” 。

- 应用程序：https://dash-bio.plotly.host/dash-needle-plot/
- Python 代码：https:// github.com/ plotly/ dash-bio/ blob/ master/ tests/ dashbio_demos/ app_needle_plot.py

Figure 42.12: image12

42.11 测量和注释医学图像

成像和图像分析是生命科学研究的基础。Dash Canvas 允许您以交互方式注释医学图像，并根据用户与图像的交互运行 Python 程序。此 Dash 应用程序显示如何使用 Dash Canvas 和 Python 计算 X 射线图像上的距离。

- 应用程序：https://dash-canvas.plotly.host/measure-length/
- Python 代码：https:// github.com/ plotly/ canvas-portal/ blob/ master/ apps/ measure-length/ app.py
- Dash Canvas：https://dash-canvas.plotly.host/Portal/

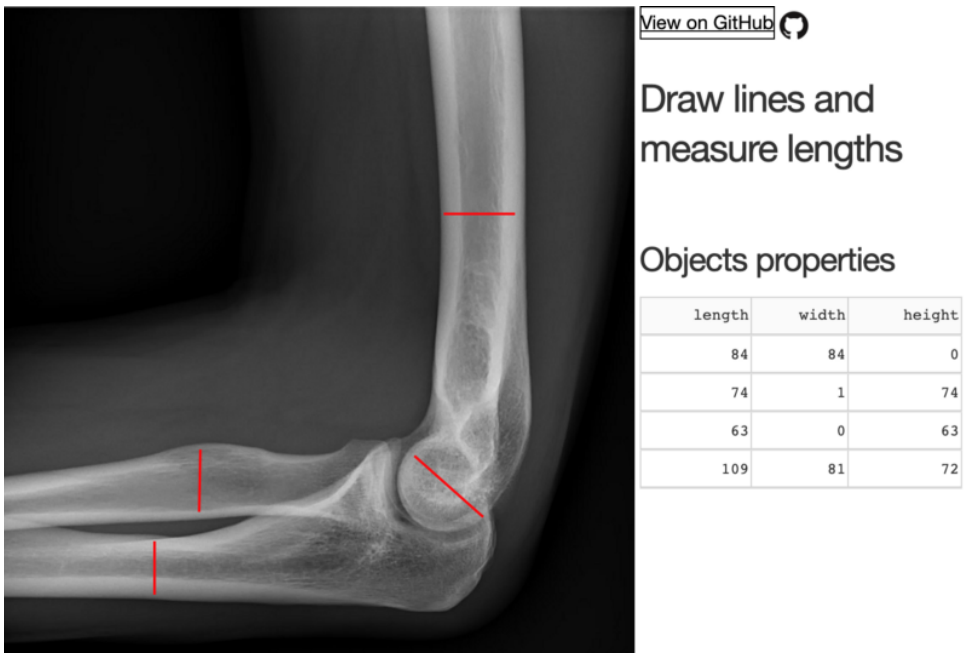


Figure 42.13: image13

使用 Dash Canvas 在医学图像上注释，保存和运行 Python 程序。

42.12 构建系统发育树和网络图

BioJS 上最受欢迎的 JavaScript 库是 Cytoscape，一个高性能网络图库。去年，Plotly 与 Cytoscape 作者密切合作，为 Dash 和 Python 用户提供了这个库。与本文中的所有 Dash 组件一样，Dash Cytoscape 是免费的开源软件。

- Dash Cytoscape 发布：https:// medium.com/ @plotlygraphs/ introducing-dash-cytoscape-ce96cac824e4
- Dash Cytoscape 文档：https://dash.plot.ly/cytoscape

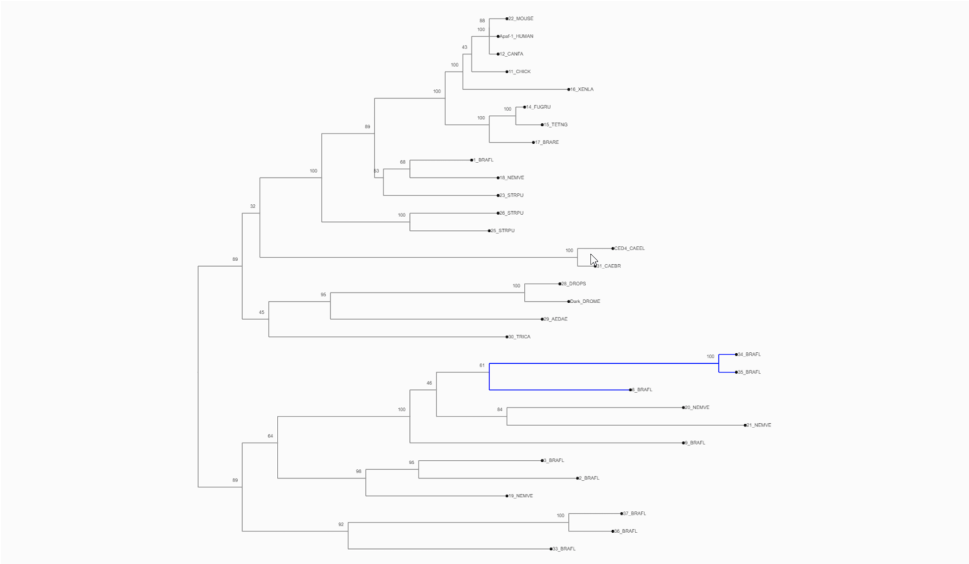


Figure 42.14: image14

使用 Dash Cytoscape 构建交互式网络图和系统发育树

42.13 Final

最后，如果你对上述内容有兴趣，可以去访问下述 Github 的内容:

- https://github.com/plotly/dash-bio
- https://github.com/plotly/dash-canvas

> 翻译: Lemon
>
> 译文出品: Python 数据之道 (PyDataLab)
>
> 来源：Plotly

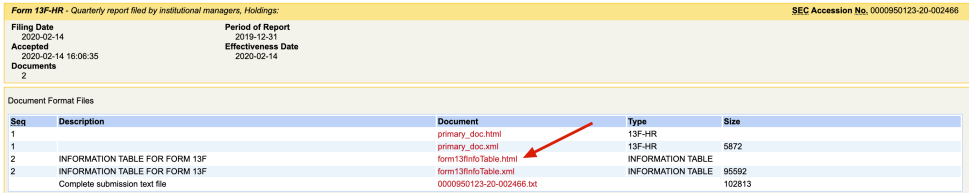
43 项目实战：用 Python 读取巴菲特最近的持仓数据

大家好，我是 Lemon ，最近关注巴菲特的同学比较多，结合 2020 年巴菲特致股东信，我也去了解了下伯克希尔最近的持仓情况。

43.1 数据表格情况

持仓情况的披露文件发布在美国证监会网站上，在美国，13F 文件是机构公布季度持仓时需要披露的文件。

这个文件可能跟国内的有点不一样，我们一起来看看吧。



Form 13F-HR - *Quarterly report filed by institutional managers, Holdings:* **SEC Accession No.** 0000950123-20-002466

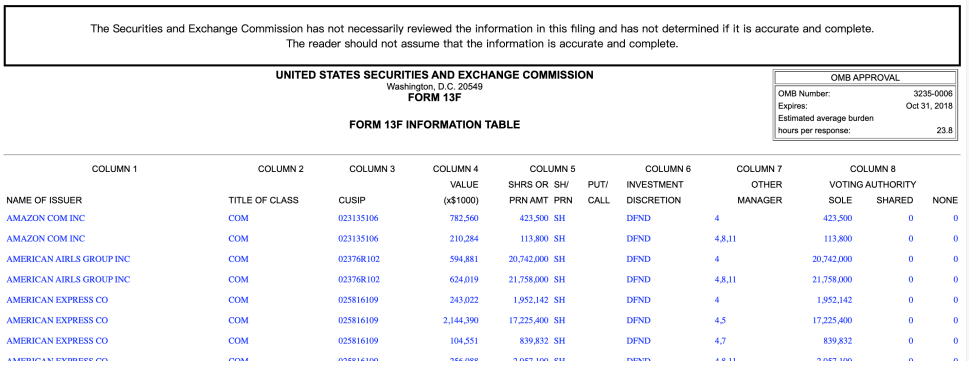
Filing Date 2020-02-14
Accepted 2020-02-14 16:06:35
Documents 2
Period of Report 2019-12-31
Effectiveness Date 2020-02-14

Document Format Files

Seq	Description	Document	Type	Size
1		primary_doc.html	13F-HR	
1		primary_doc.xml	13F-HR	5872
2	INFORMATION TABLE FOR FORM 13F	form13fInfoTable.html	INFORMATION TABLE	
2	INFORMATION TABLE FOR FORM 13F	form13fInfoTable.xml	INFORMATION TABLE	95592
	Complete submission text file	0000950123-20-002466.txt		102813

Figure 43.1: 下载页面

在官方网站上，是以网页格式进行披露的（html 和 xml），点击上图中红色箭头所指的 “html” 文件后，在会在网页格式中打开持仓情况。如下：



The Securities and Exchange Commission has not necessarily reviewed the information in this filing and has not determined if it is accurate and complete.
The reader should not assume that the information is accurate and complete.

UNITED STATES SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549
FORM 13F

OMB APPROVAL
OMB Number: 3235-0006
Expires: Oct 31, 2018
Estimated average burden hours per response: 23.8

FORM 13F INFORMATION TABLE

COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5			COLUMN 6	COLUMN 7	COLUMN 8		
			VALUE	SHRS OR	SH/	PUT/	INVESTMENT	OTHER	VOTING AUTHORITY		
NAME OF ISSUER	TITLE OF CLASS	CUSIP	(x$1000)	PRN AMT	PRN	CALL	DISCRETION	MANAGER	SOLE	SHARED	NONE
AMAZON COM INC	COM	023135106	782,560	423,500	SH		DFND	4	423,500	0	0
AMAZON COM INC	COM	023135106	210,284	113,800	SH		DFND	4,8,11	113,800	0	0
AMERICAN AIRLS GROUP INC	COM	02376R102	594,881	20,742,000	SH		DFND	4	20,742,000	0	0
AMERICAN AIRLS GROUP INC	COM	02376R102	624,019	21,758,000	SH		DFND	4,8,11	21,758,000	0	0
AMERICAN EXPRESS CO	COM	025816109	243,022	1,952,142	SH		DFND	4	1,952,142	0	0
AMERICAN EXPRESS CO	COM	025816109	2,144,390	17,225,400	SH		DFND	4,5	17,225,400	0	0
AMERICAN EXPRESS CO	COM	025816109	104,551	839,832	SH		DFND	4,7	839,832	0	0

Figure 43.2: 网页情况

从上面的表格中，可以清晰的看出伯克希尔的持仓情况。如果我们想对伯克希尔的持仓情况进行分析，则可以将数据保存到本地，然后进行分析，尤其是涉及多份数据进行对比和分析的时候，对这些数据的读取和梳理就有必要了。

43.2 数据读取

下面我们来看看如何用 Python 进行读取。首先，我们将下面红色箭头所指的“xml”文件下载到本地，名称为“test.xml”。

Document Format Files

Seq	Description	Document
1		primary_doc.html
1		primary_doc.xml
2	INFORMATION TABLE FOR FORM 13F	form13fInfoTable.html
2	INFORMATION TABLE FOR FORM 13F	form13fInfoTable.xml
	Complete submission text file	0000950123-20-002466.txt

Figure 43.3: 本地文件

初步观察网页格式上的数据，可以看出，这个数据表格是一个嵌套型的表格，我们在数据分析的时候，最好进行转化为二维的 dataframe 。

UNITED STATES SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549
FORM 13F

FORM 13F INFORMATION TABLE

OMB APPROVAL	
OMB Number:	3235-0006
Expires:	Oct 31, 2018
Estimated average burden hours per response:	23.8

COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5			COLUMN 6	COLUMN 7	COLUMN 8		
			VALUE	SHRS OR	SH/	PUT/	INVESTMENT	OTHER	VOTING AUTHORITY		
NAME OF ISSUER	TITLE OF CLASS	CUSIP	(x$1000)	PRN AMT	PRN	CALL	DISCRETION	MANAGER	SOLE	SHARED	NONE
AMAZON COM INC	COM	023135106	782,560	423,500	SH		DFND	4	423,500	0	0
AMAZON COM INC	COM	023135106	210,284	113,800	SH		DFND	4,8,11	113,800	0	0
AMERICAN AIRLS GROUP INC	COM	02376R102	594,881	20,742,000	SH		DFND	4	20,742,000	0	0
AMERICAN AIRLS GROUP INC	COM	02376R102	624,019	21,758,000	SH		DFND	4,8,11	21,758,000	0	0
AMERICAN EXPRESS CO	COM	025816109	243,022	1,952,142	SH		DFND	4	1,952,142	0	0

Figure 43.4: 数据格式

本来我是想用 Pandas 直接进行数据读取的，后来发现是嵌套表格之后，有点麻烦。于是改用 BeautifulSoup 来读取，可以将嵌套的数据也单独提取出来，方便后续分析。

用 BeautifulSoup 进行解析的时候，思路其实比较简单：

1. 读取 XML 文件的内容
2. 构造一个 pandas 的 dataframe ，用来保存解析出来的数据

43.3 xml 格式内容读取

我们知道，BeautifulSoup 可以用来解析 html、xml 等网页的标记语言内容。因此，对于 XML 文件，我们可以直接进行读取。

下面的代码中的“test.xml”文件是我们保存的本地的文件，在本文结尾处也提供了获取方式。

```
# -*- coding: utf-8 -*-
"""
@Author: Lemon
@出 品： Python数 据 之 道
```

```
@Homepage: liyangbit.com
"""

from bs4 import BeautifulSoup
import pandas as pd

path = r'./test.xml'

htmlfile = open(path, encoding='utf-8')

htmlhandle = htmlfile.read()

soup = BeautifulSoup(htmlhandle, 'xml')

print(soup)
```

读取标记语言的内容后，我们再来通过 BeautifulSoup 的 `find_all` 和 `find` 功能来读取所需要的数据即可。

对数据进行封装和保存的方法有多种，我们这里采取的是构造一个空的 Pandas 的 DataFrame 来进行封装，然后用 Pandas 的 `to_excel` 来保存文件，这种方法还是比较方便的，有兴趣的同学可以研究下。

```

count = 0
# 构造一个空的 dataframe
result = pd.DataFrame({}, index=[0])

# 根据持仓数据表格，添加需要保存的字段（dataframe 的 列名称）
result['nameOfIssuer'] = ''
result['titleOfClass'] = ''
result['cusip'] = ''
result['value'] = ''
result['sshPrnamt'] = ''
result['sshPrnamtType'] = ''
result['investmentDiscretion'] = ''
result['otherManager'] = ''
result['votingAuthority-Sole'] = ''
result['votingAuthority-Share'] = ''
result['votingAuthority-None'] = ''

new = result
for item in soup.find_all('infoTable'):
    nameOfIssuer = item.find('nameOfIssuer').get_text()
    new['nameOfIssuer'] = nameOfIssuer
    titleOfClass = item.find('titleOfClass').get_text()
    new['titleOfClass'] = titleOfClass
    cusip = item.find('cusip').get_text()
    new['cusip'] = cusip
    value = item.find('value').get_text()
    new['value'] = value
    sshPrnamt = item.find('shrsOrPrnAmt').find('sshPrnamt').get_text()
    new['sshPrnamt'] = sshPrnamt
```

```
    sshPrnamtType = item.find('shrsOrPrnAmt').find('sshPrnamtType').get_text()
    new['sshPrnamtType'] = sshPrnamtType
    investmentDiscretion = item.find('investmentDiscretion').get_text()
    new['investmentDiscretion'] = investmentDiscretion
    otherManager = item.find('otherManager').get_text()
    new['otherManager'] = otherManager
    Sole = item.find('votingAuthority').find('Sole').get_text()
    new['votingAuthority-Sole'] = Sole
    Shared = item.find('votingAuthority').find('Shared').get_text()
    new['votingAuthority-Share'] = Shared
    Non = item.find('votingAuthority').find('None').get_text()
    new['votingAuthority-None'] = Non

    count += 1
    result = result.append(new, ignore_index=True)

print(count)

print(result)

# 保存数据到 Excel 表格
result.to_excel('BRKA-2019-Q4.xlsx')
```

如果后续需要对数据进行分析和可视化，可以借助 Pandas、Matplotlib、Plotly 等 Python 库来处理。

当然，如果你只是想看看或分析这一份文件，也可以直接从网页上复制内容到 Excel 表格中即可，获取数据的方法有多种，选择适合自己的就好哈。

最后，为方便大家进行操作，Lemon 给大家提供了源代码和数据文件，可以在公众号「Python 数据之道」后台回复数字「690」进行获取。

44 项目实战：Python 数据分析，UFO 长啥样?

> 真心讲，长这么大，还没有见过 UFO 长啥样，偶然看到美国 UFO 报告中心有关于 UFO 时间记录的详细信息，突然想分析下这些记录里都包含了那些有趣的信息，于是有了这次的分析过程。
>
> 当然，原始数据包含的记录信息比较多，我只是进了了比较简单的分析，有兴趣的童鞋可以一起来分析，别忘了也给大家分享下您的分析情况哦。

本次分析的主要内容涉及以下几个方面：

- UFO 长啥样?
- UFO 在哪些地方出现的次数较多?
- UFO 在哪些年份出现的次数较多?
- 热力图同时显示哪些州和哪些年 UFO 出现次数最多

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
plt.style.use('ggplot')
```

44.1 数据整理与清洗

```
df =  pd.read_csv('nuforc_events.csv')
```

```
print(df.shape) # 查看数据的结构
print(df.head())
```

```
(110265, 13)
             Event_Time  Event_Date    Year  Month   Day  Hour  Minute  \
0  2017-04-20T14:15:00Z  2017-04-20  2017.0    4.0  20.0  14.0    15.0
1  2017-04-20T04:56:00Z  2017-04-20  2017.0    4.0  20.0   4.0    56.0
2  2017-04-19T23:55:00Z  2017-04-19  2017.0    4.0  19.0  23.0    55.0
3  2017-04-19T23:50:00Z  2017-04-19  2017.0    4.0  19.0  23.0    50.0
4  2017-04-19T23:29:00Z  2017-04-19  2017.0    4.0  19.0  23.0    29.0

          City State     Shape     Duration  \
0     Palmyra    NJ     Other    5 minutes
1  Bridgeview    IL     Light   20 seconds
2      Newton    AL  Triangle    5 seconds
3      Newton    AL  Triangle  5-6 minutes
```

```
4      Denver    CO     Light       1 hour

                                                Summary  \
0    I observed an aircraft that seemed to look odd.
1  Bridgeview, IL, blue light.  ((anonymous report))
2                               Silent triangle UFO.
3  My friend and I stepped outside hoping to catc...
4  Moved slow but made quick turns staying and ci...

                                              Event_URL
0  http://www.nuforc.org/webreports/133/S133726.html
1  http://www.nuforc.org/webreports/133/S133720.html
2  http://www.nuforc.org/webreports/133/S133724.html
3  http://www.nuforc.org/webreports/133/S133723.html
4  http://www.nuforc.org/webreports/133/S133721.html
```

- 由于存在许多包含 NaN 的数据信息，在进行分析之前，先用 dropna() 方法去除包含 NaN 的行数

```
df_clean = df.dropna()
print(df_clean.shape) # 查看去除Nan后还有多少行
print(df_clean.head())
```

```
(95004, 13)
             Event_Time  Event_Date    Year  Month   Day  Hour  Minute  \
0  2017-04-20T14:15:00Z  2017-04-20  2017.0    4.0  20.0  14.0    15.0
1  2017-04-20T04:56:00Z  2017-04-20  2017.0    4.0  20.0   4.0    56.0
2  2017-04-19T23:55:00Z  2017-04-19  2017.0    4.0  19.0  23.0    55.0
3  2017-04-19T23:50:00Z  2017-04-19  2017.0    4.0  19.0  23.0    50.0
4  2017-04-19T23:29:00Z  2017-04-19  2017.0    4.0  19.0  23.0    29.0

          City State     Shape     Duration  \
0     Palmyra    NJ     Other    5 minutes
1  Bridgeview    IL     Light   20 seconds
2      Newton    AL  Triangle    5 seconds
3      Newton    AL  Triangle  5-6 minutes
4      Denver    CO     Light       1 hour

                                                Summary  \
0    I observed an aircraft that seemed to look odd.
1  Bridgeview, IL, blue light.  ((anonymous report))
2                               Silent triangle UFO.
3  My friend and I stepped outside hoping to catc...
4  Moved slow but made quick turns staying and ci...

                                              Event_URL
0  http://www.nuforc.org/webreports/133/S133726.html
1  http://www.nuforc.org/webreports/133/S133720.html
2  http://www.nuforc.org/webreports/133/S133724.html
3  http://www.nuforc.org/webreports/133/S133723.html
4  http://www.nuforc.org/webreports/133/S133721.html
```

- 由于 1900 年以前的数据较少，这里选择 1900 年以后的数据来进行分析，如下：

```
df_clean = df_clean[df_clean['Year']>=1900] # 获取1900年以后的数据来进行分析
```

- 查看导入的每列数据的数据类型，通过运行结果，可以看到，“Event_Date”列并不是日期类型，因此要将之转换。
- 可以采用 pd.to_datetime() 方法来操作

```
df_clean.dtypes
```

```
Event_Time     object
Event_Date     object
Year          float64
Month         float64
Day           float64
Hour          float64
Minute        float64
City           object
State          object
Shape          object
Duration       object
Summary        object
Event_URL      object
dtype: object
```

- 用 pd.to_datetime() 方法来将 str 格式的日期转换成日期类型

```
pd.to_datetime(df_clean['Event_Date']) # 1061-12-31年不能显示
# OutOfBoundsDatetime: Out of bounds nanosecond timestamp: 1061-12-31 00:00:00
df_clean.dtypes
```

```
Event_Time     object
Event_Date     object
Year          float64
Month         float64
Day           float64
Hour          float64
Minute        float64
City           object
State          object
Shape          object
Duration       object
Summary        object
Event_URL      object
dtype: object
```

44.2 UFO 长啥样?

- 按 UFO 出现的形状类型来分析, 统计不同类型的 UFO 出现的次数

```
s_shape = df_clean.groupby('Shape')['Event_Date'].count()
print(type(s_shape))
s_shape.sort_values(inplace=True)
s_shape
```

```
<class 'pandas.core.series.Series'>

Shape
Changed          1
Hexagon          1
Pyramid          1
Flare            1
Round            2
Crescent         2
Delta            7
Cross          287
Cone           383
Egg            842
Teardrop       866
Chevron       1187
Diamond       1405
Cylinder      1495
Rectangle     1620
Flash         1717
Cigar         2313
Changing      2378
Formation     3070
Oval          4332
Disk          5841
Sphere        6482
Other         6658
Unknown       6887
Fireball      7785
Triangle      9358
Circle        9818
Light        20254
Name: Event_Date, dtype: int64
```

剔除特殊情况 * 剔除出现次数少于 10 次的类型 * 剔除“Unknown”及“Other”类型

```
s_shape_normal = s_shape[s_shape.values>10]
s_shape_normal
```

```
Shape
Cross          287
Cone           383
```

```
Egg            842
Teardrop       866
Chevron       1187
Diamond       1405
Cylinder      1495
Rectangle     1620
Flash         1717
Cigar         2313
Changing      2378
Formation     3070
Oval          4332
Disk          5841
Sphere        6482
Other         6658
Unknown       6887
Fireball      7785
Triangle      9358
Circle        9818
Light        20254
Name: Event_Date, dtype: int64
```

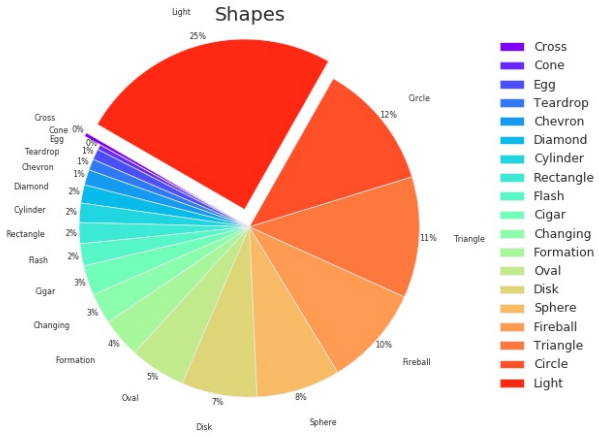
```
s_shape_normal = s_shape_normal[s_shape_normal.index.isin(['Unknown', 'Other'])
    ==False]
s_shape_normal
```

```
Shape
Cross          287
Cone           383
Egg            842
Teardrop       866
Chevron       1187
Diamond       1405
Cylinder      1495
Rectangle     1620
Flash         1717
Cigar         2313
Changing      2378
Formation     3070
Oval          4332
Disk          5841
Sphere        6482
Fireball      7785
Triangle      9358
Circle        9818
Light        20254
Name: Event_Date, dtype: int64
```

```
from matplotlib import font_manager as fm
from  matplotlib import cm

```

```
labels = s_shape_normal.index
sizes = s_shape_normal.values

explode = (0.1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.1)  # "explode" ,  show the selected slice

fig, axes = plt.subplots(figsize=(10,5),ncols=2) # 设置绘图区域大小
ax1, ax2 = axes.ravel()

colors = cm.rainbow(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn, rainbow, gray,spring,Darks
patches, texts, autotexts = ax1.pie(sizes, labels=labels, autopct='%1.0f%%', explode=explode,
        shadow=False, startangle=150, colors=colors, labeldistance=1.2, pctdistance=1.05, radius=0.95)
# labeldistance: 控制labels显示的位置
# pctdistance: 控制百分比显示的位置
# radius: 控制切片突出的距离

ax1.axis('equal')

# 重新设置字体大小
proptease = fm.FontProperties()
proptease.set_size('xx-small')
# font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-large’ , ‘xx-large’ or number, e.g. '12'
plt.setp(autotexts, fontproperties=proptease)
plt.setp(texts, fontproperties=proptease)

ax1.set_title('Shapes', loc='center')

# ax2 只显示图例（legend）
ax2.axis('off')
ax2.legend(patches, labels, loc='center left', fontsize=9)

# plt.tight_layout()
# plt.savefig("pie_shape_ufo.png", bbox_inches='tight')
plt.savefig('ufo_shapes.jpg')
plt.show()
```

运行结果如下：



44.3 UFO 在美国那些州（state）出现的次数比较多?

按“State”进行分组运算，统计 ufo 在各个州出现的次数

```
s_state = df_clean.groupby('State')['Event_Date'].count()
print(type(s_state))
s_state.head()
```

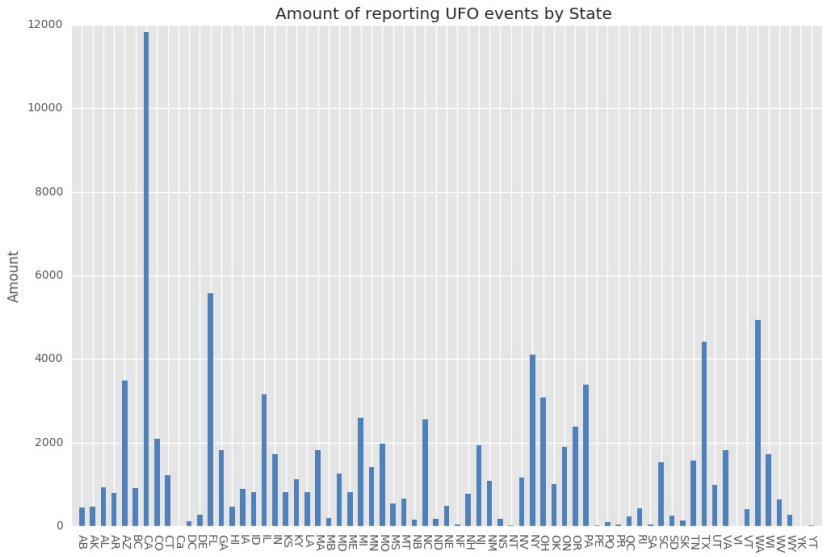
```
<class 'pandas.core.series.Series'>

State
AB      438
AK      472
AL      930
AR      791
AZ     3488
Name: Event_Date, dtype: int64
```

将分析得到的结果进行可视化显示，如下：

```
fig, ax1 = plt.subplots(figsize=(12,8))

width = 0.5
state = s_state.index
x_pos1 = np.arange(len(state))
y1 = s_state.values
ax1.bar(x_pos1, y1,color='#4F81BD',align='center', width=width, label='Amounts'
    , linewidth=0)
ax1.set_title('Amount of reporting UFO events by State ')
ax1.set_xlim(-1, len(state))
ax1.set_xticks(x_pos1)
ax1.set_xticklabels(state, rotation = -90)
ax1.set_ylabel('Amount')
```

```
13
14 fig.savefig('ufo_state.jpg')
15 plt.show()
```

运行结果如下：



从上图可看出，ufo 在加州（CA）出现的总次数明显比其他地方多，难道是 ufo 偏爱加州人民?

44.4 UFO 在哪些年份出现的次数较多?

按“Year”进行分组运算，统计 ufo 在各个年份出现的次数

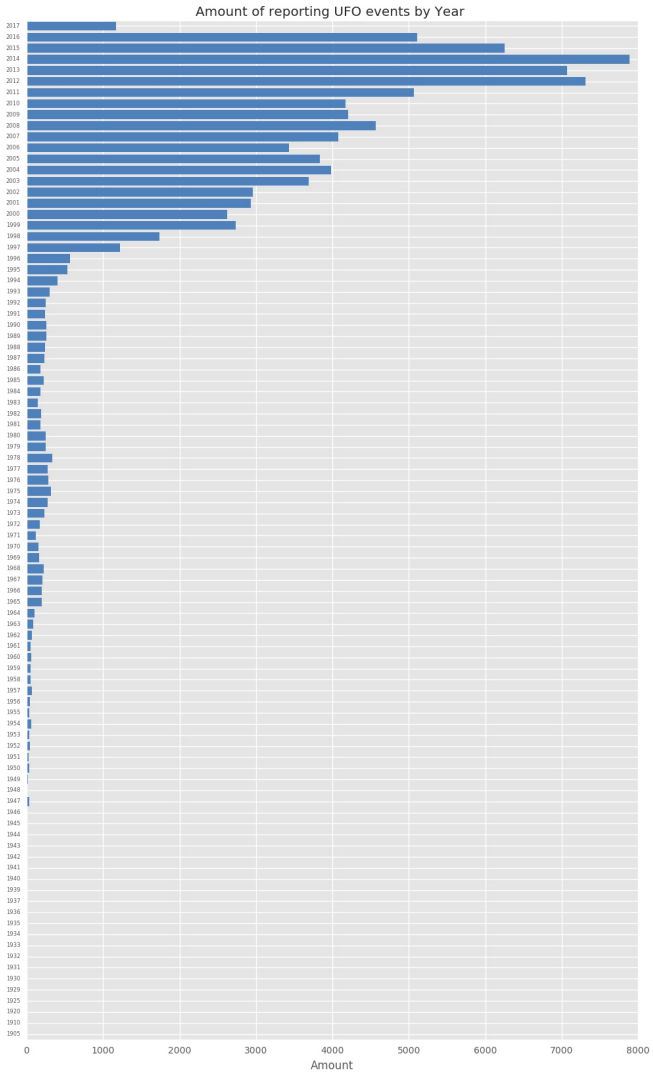
```
1 # df_clean['Year'].astype(int)
2 s_year = df_clean.groupby(df_clean['Year'].astype(int))['Event_Date'].count()
3 print(type(s_year))
4 s_year.head()
```

```
1 <class 'pandas.core.series.Series'>
2
3 Year
4 1905    1
5 1910    2
6 1920    1
7 1925    1
8 1929    1
9 Name: Event_Date, dtype: int64
```

将分析得到的结果进行可视化显示，如下：

```
fig, ax = plt.subplots(figsize=(12,20))
# fig, ax1 = plt.subplots(figsize=(12,8))
# fig, axes = plt.subplots(nrows=2, figsize=(12,8))
# fig, axes = plt.subplots(ncols=2, figsize=(18,4))

year = s_year.index
y_pos = np.arange(len(year))
x_value = s_year.values
ax.barh(y_pos, x_value,color='#4F81BD',align='center', label='Amounts',
    linewidth=0)
ax.set_title('Amount of reporting UFO events by Year ')
ax.set_ylim(-0.5, len(year)-0.5)
ax.set_yticks(y_pos)
ax.set_yticklabels(year, rotation = 0, fontsize=6)
ax.set_xlabel('Amount')

plt.savefig('ufo_year.jpg')
plt.show()
```

运行结果如下：



从上图可看出，近年来 UFO 出现的报告次数最多

44.5 1997 年以后的 UFO 事件分析

- 通过上述分析可看出，1997 年以前，报告发现 UFO 的事件相对较少，下面将针对 1997 年以后的情况进行分析

```
1 df_97 = df_clean[(df_clean['Year']>=1997)]
```

```
df_97['Year'] = df_97['Year'].astype(int)
# df_97.astype({'Year':int})
print(df_97.shape)
print(df_97.head())
```

```
(86041, 13)
             Event_Time  Event_Date  Year  Month   Day  Hour  Minute  \
0  2017-04-20T14:15:00Z  2017-04-20  2017    4.0  20.0  14.0    15.0
1  2017-04-20T04:56:00Z  2017-04-20  2017    4.0  20.0   4.0    56.0
2  2017-04-19T23:55:00Z  2017-04-19  2017    4.0  19.0  23.0    55.0
3  2017-04-19T23:50:00Z  2017-04-19  2017    4.0  19.0  23.0    50.0
4  2017-04-19T23:29:00Z  2017-04-19  2017    4.0  19.0  23.0    29.0

          City State     Shape     Duration  \
0     Palmyra    NJ     Other    5 minutes
1  Bridgeview    IL     Light   20 seconds
2      Newton    AL  Triangle    5 seconds
3      Newton    AL  Triangle  5-6 minutes
4      Denver    CO     Light       1 hour

                                          Summary  \
0   I observed an aircraft that seemed to look odd.
1  Bridgeview, IL, blue light.  ((anonymous report))
2                              Silent triangle UFO.
3  My friend and I stepped outside hoping to catc...
4  Moved slow but made quick turns staying and ci...

                                        Event_URL
0  http://www.nuforc.org/webreports/133/S133726.html
1  http://www.nuforc.org/webreports/133/S133720.html
2  http://www.nuforc.org/webreports/133/S133724.html
3  http://www.nuforc.org/webreports/133/S133723.html
4  http://www.nuforc.org/webreports/133/S133721.html
```

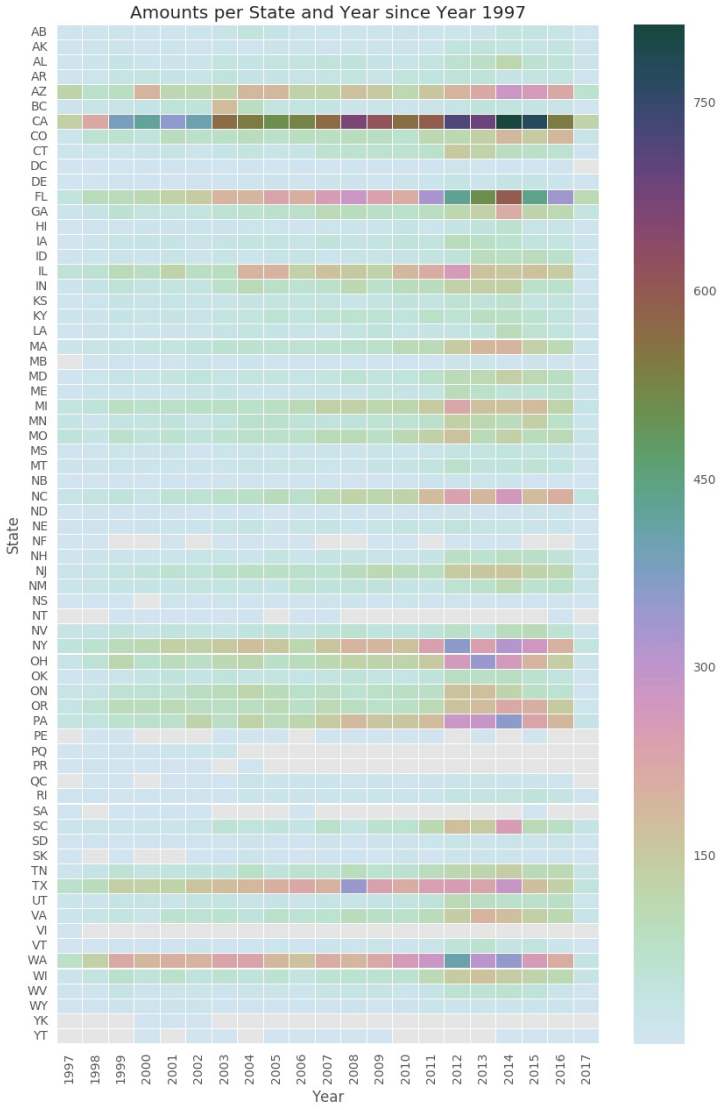
将数据按“Year”和“State”进行分组运算，如下：

```
df_amount_year = df_97.groupby(['Year', 'State'])['Event_Date'].size().
    reset_index()
df_amount_year.columns = ['Year', 'State', 'Amount']
print(df_amount_year.head())
```

```
   Year State  Amount
0  1997    AB       6
1  1997    AK       5
2  1997    AL       8
3  1997    AR      10
4  1997    AZ     127
```

```
import seaborn as sns

```

```
df_pivot = df_amount_year.pivot_table(index='State', columns='Year', values='
    Amount')

f, ax = plt.subplots(figsize = (10, 15))
cmap = sns.cubehelix_palette(start = 1, rot = 3, gamma=0.8, as_cmap = True)
sns.heatmap(df_pivot, cmap = cmap, linewidths = 0.05, ax = ax)

ax.set_title('Amounts per State and Year since Year 1997')
ax.set_xlabel('Year')
ax.set_ylabel('State')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
f.savefig('ufo_per_year_state.jpg')
```



- 上图中，颜色越深的地方，表示 UFO 事件报告的次数越多。

> 原始数据来源于美国的 UFO 事件报告中心。

45 项目实战：世界杯系列

45.1 第一篇：2018 世界杯：用 Python 分析热门夺冠球队

2018 年，火热的世界杯即将拉开序幕。在比赛开始之前，我们不妨用 Python 来对参赛队伍的实力情况进行分析，并大胆的预测下本届世界杯的夺冠热门球队。

通过数据分析，可以发现很多有趣的结果，比如：

- 找出哪些队伍是首次进入世界杯的黑马队伍
- 找出 2018 年 32 强中之前已经进入过世界杯，但在世界杯上没有赢得过一场比赛的队伍

当然，我们本次的主要任务是要通过数据分析来预测 2018 年世界杯的夺冠热门队伍。

本次分析的数据来源于 Kaggle，包含从 1872 年到今年的数据，包括世界杯比赛、世界杯预选赛、亚洲杯、欧洲杯、国家之间的友谊赛等比赛，一共大约 40000 场比赛的情况。

本次的环境为

- window 7 系统
- python 3.6
- Jupyter Notebook
- pandas version 0.22.0

先来看看数据的情况：

```
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('ggplot')

df = pd.read_csv('results.csv')
df.head()
```

该数据集包含的数据列的信息如下：

- 日期
- 主队名称
- 客队名称
- 主队进球数（不含点球）
- 客队进球数（不含点球）
- 比赛的类型

- 比赛所在城市
- 比赛所在国家
- 是否中立

结果如下：

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral
0	1872-11-30	Scotland	England	0	0	Friendly	Glasgow	Scotland	False
1	1873-03-08	England	Scotland	4	2	Friendly	London	England	False
2	1874-03-07	Scotland	England	2	1	Friendly	Glasgow	Scotland	False
3	1875-03-06	England	Scotland	2	2	Friendly	London	England	False
4	1876-03-04	Scotland	England	3	0	Friendly	Glasgow	Scotland	False

45.1.1 获取所有世界杯比赛的数据（不含预选赛）

```
df_FIFA_all = df[df['tournament'].str.contains('FIFA', regex=True)]
df_FIFA = df_FIFA_all[df_FIFA_all['tournament']=='FIFA World Cup']
df_FIFA.head()
```

结果如下：

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral
1277	1930-07-13	Belgium	USA	0	3	FIFA World Cup	Montevideo	Uruguay	True
1278	1930-07-13	France	Mexico	4	1	FIFA World Cup	Montevideo	Uruguay	True
1279	1930-07-14	Brazil	Yugoslavia	1	2	FIFA World Cup	Montevideo	Uruguay	True
1280	1930-07-14	Peru	Romania	1	3	FIFA World Cup	Montevideo	Uruguay	True
1281	1930-07-15	Argentina	France	1	0	FIFA World Cup	Montevideo	Uruguay	True

数据做一个初步整理

```
df_FIFA.loc[:,'date'] = pd.to_datetime(df_FIFA.loc[:,'date'])
df_FIFA['year'] = df_FIFA['date'].dt.year
df_FIFA['diff_score'] = df_FIFA['home_score']-df_FIFA['away_score']
df_FIFA['win_team'] = ''
df_FIFA['diff_score'] = pd.to_numeric(df_FIFA['diff_score'])
```

创建一个新的列数据，包含获胜队伍的信息

```
# The first method to get the winners

df_FIFA.loc[df_FIFA['diff_score']> 0, 'win_team'] = df_FIFA.loc[df_FIFA['
    diff_score']> 0, 'home_team']
df_FIFA.loc[df_FIFA['diff_score']< 0, 'win_team'] = df_FIFA.loc[df_FIFA['
    diff_score']< 0, 'away_team']
df_FIFA.loc[df_FIFA['diff_score']== 0, 'win_team'] = 'Draw'

df_FIFA.head()

```

```
# The second method to get the winners

def find_win_team(df):
    winners = []
    for i, row in df.iterrows():
        if row['home_score'] > row['away_score']:
            winners.append(row['home_team'])
        elif row['home_score'] < row['away_score']:
            winners.append(row['away_team'])
        else:
            winners.append('Draw')
    return winners

df_FIFA['winner'] = find_win_team(df_FIFA)
df_FIFA.head()
```

结果如下：

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral	year	diff_score	win_team	winner
1277	1930-07-13	Belgium	USA	0	3	FIFA World Cup	Montevideo	Uruguay	True	1930	-3	USA	USA
1278	1930-07-13	France	Mexico	4	1	FIFA World Cup	Montevideo	Uruguay	True	1930	3	France	France
1279	1930-07-14	Brazil	Yugoslavia	1	2	FIFA World Cup	Montevideo	Uruguay	True	1930	-1	Yugoslavia	Yugoslavia
1280	1930-07-14	Peru	Romania	1	3	FIFA World Cup	Montevideo	Uruguay	True	1930	-2	Romania	Romania
1281	1930-07-15	Argentina	France	1	0	FIFA World Cup	Montevideo	Uruguay	True	1930	1	Argentina	Argentina

45.1.2 获取世界杯所有比赛的前 20 强数据情况

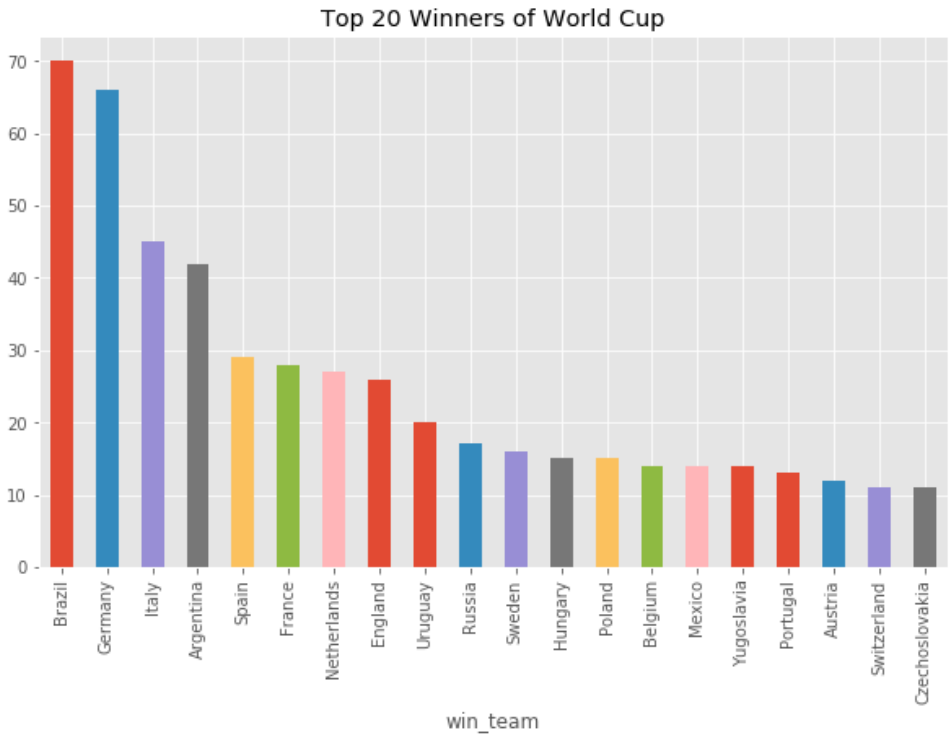
45.1.2.1 获取世界杯所有比赛获胜场数最多的前 20 强数据

```
s = df_FIFA.groupby('win_team')['win_team'].count()
s.sort_values(ascending=False, inplace=True)
s.drop(labels=['Draw'], inplace=True)
```

用 pandas 可视化如下：

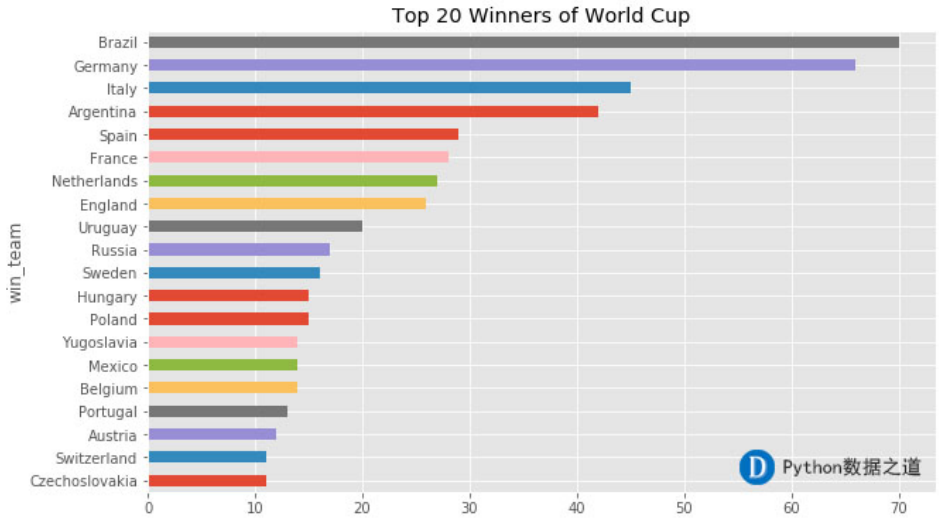
柱状图

```
s.head(20).plot(kind='bar', figsize=(10,6), title='Top 20 Winners of World Cup'
    )
```



水平柱状图

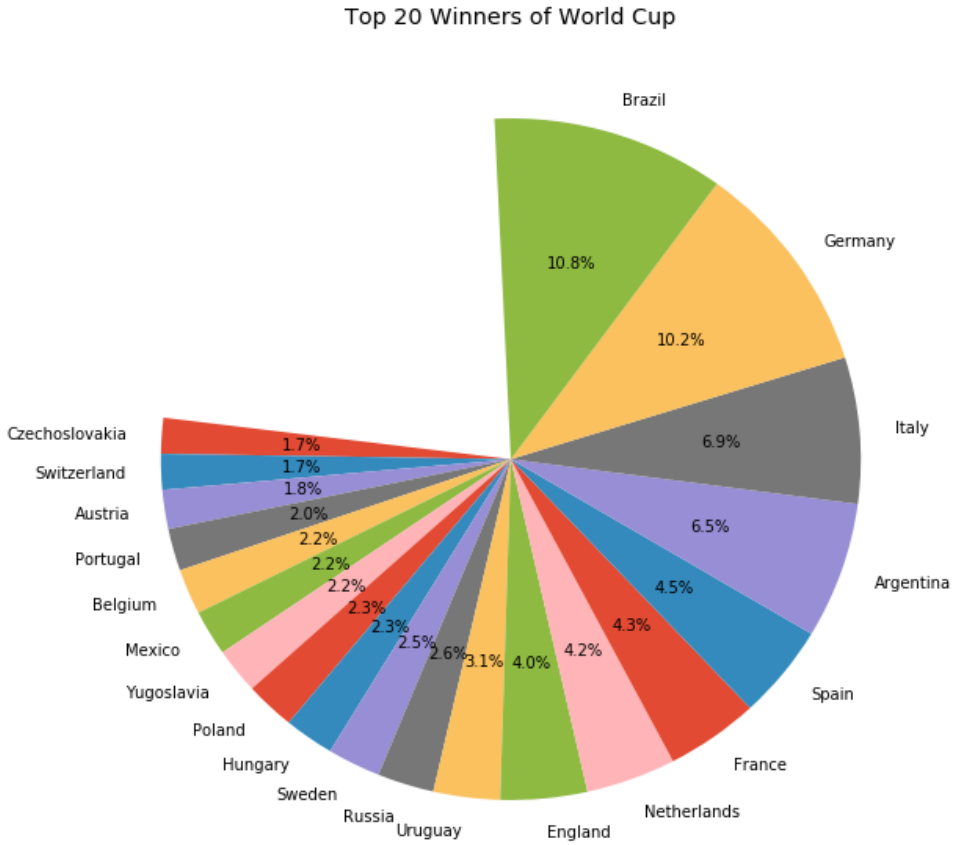
```
s.sort_values(ascending=True,inplace=True)
s.tail(20).plot(kind='barh', figsize=(10,6), title='Top 20 Winners of World Cup
    ')
```



饼图

```
s_percentage = s/s.sum()
s_percentage
```

```
s_percentage.tail(20).plot(kind='pie', figsize=(10,10), autopct='%.1f%%',
                           startangle=173, title='Top 20 Winners of World Cup',
                                label='')
```



分析结论 1：从赢球场数来看，巴西、德国、意大利、阿根廷四支球队实力最强。

通过上面的分析，我们还可以来查看部分国家的获胜情况

```
s.get('China', default = 'NA')
s.get('Japan', default = 'NA')
s.get('Korea DPR', default = 'NA')
s.get('Korea Republic', default = 'NA')
s.get('Egypt', default = 'NA')
```

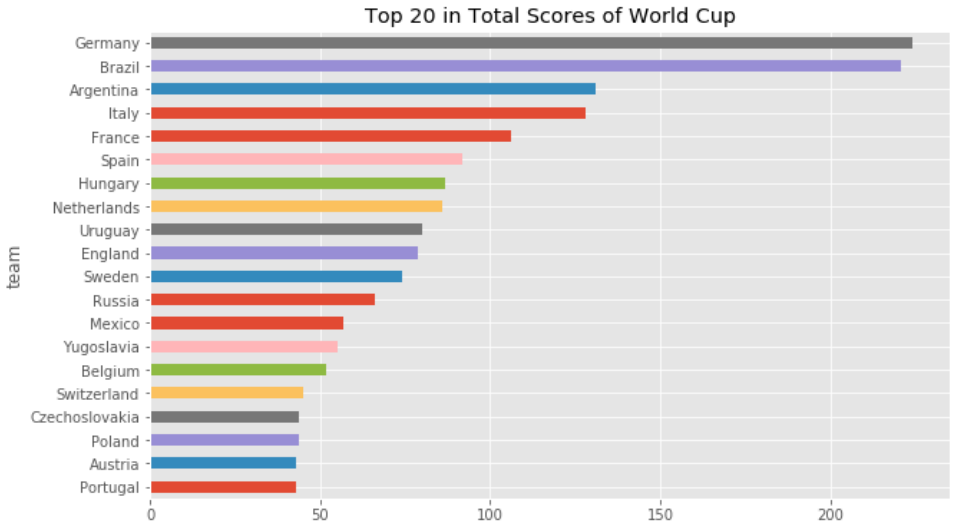
运行结果分别是“NA”，4，1，5，“NA”。

从结果来看，中国队，在世界杯比赛上（不含预选赛）还没有赢过。当然，**本次世界杯的黑马-埃及队，之前两度进入世界杯上，但也没有赢过 ~~**

上面分析的是赢球场数的情况，下面我们来看下进球总数情况。

45.1.2.2 各个国家队进球总数量情况

```
df_score_home = df_FIFA[['home_team', 'home_score']]
column_update = ['team', 'score']
df_score_home.columns = column_update
df_score_away = df_FIFA[['away_team', 'away_score']]
df_score_away.columns = column_update
df_score = pd.concat([df_score_home,df_score_away], ignore_index=True)
s_score = df_score.groupby('team')['score'].sum()
s_score.sort_values(ascending=False, inplace=True)
s_score.sort_values(ascending=True, inplace=True)
s_score.tail(20).plot(kind='barh', figsize=(10,6), title='Top 20 in Total
    Scores of World Cup')
```



分析结论 2：从进球总数量来看，德国、巴西、阿根廷、意大利四支球实力最强。

上面分析的是自 1872 年以来的所有球队的数据情况，下面，我们重点来分析下 2018 年世界杯 32 强的数据情况。

45.1.3 2018 年世界杯 32 强分析

2018 年世界杯的分组情况如下：

第一组：俄罗斯、德国、巴西、葡萄牙、阿根廷、比利时、波兰、法国

第二组：西班牙、秘鲁、瑞士、英格兰、哥伦比亚、墨西哥、乌拉圭、克罗地亚

第三组：丹麦、冰岛、哥斯达黎加、瑞典、突尼斯、埃及、塞内加尔、伊朗

第四组：塞尔维亚、尼日利亚、澳大利亚、日本、摩洛哥、巴拿马、韩国、沙特阿拉伯

获取 32 强的所有数据

首先，判断是否有队伍首次打入世界杯。

```
team_list = ['Russia', 'Germany', 'Brazil', 'Portugal', 'Argentina', 'Belgium',
    'Poland', 'France',
```

```
             'Spain', 'Peru', 'Switzerland', 'England', 'Colombia', 'Mexico', '
                Uruguay', 'Croatia',
            'Denmark', 'Iceland', 'Costa Rica', 'Sweden', 'Tunisia', 'Egypt', '
                Senegal', 'Iran',
            'Serbia', 'Nigeria', 'Australia', 'Japan', 'Morocco', 'Panama', '
                Korea Republic', 'Saudi Arabia']
for item in team_list:
    if item not in s_score.index:
        print(item)

out:
Iceland
Panama
```

通过上述分析可知，**冰岛队和巴拿马队是首次打入世界杯的。**

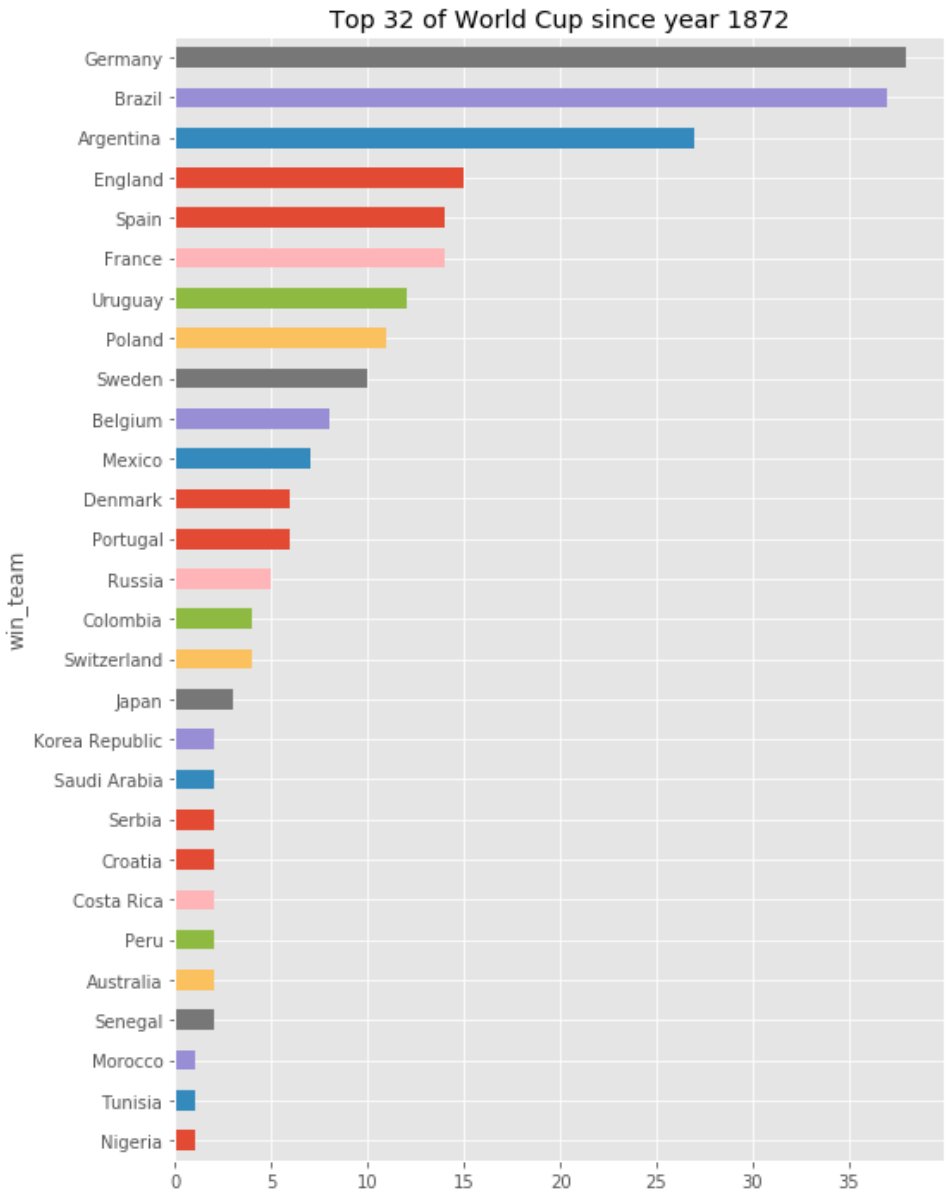
由于冰岛队和巴拿马队是首次进入世界杯，所以这里的 32 强数据，事实上是没有这两支队伍的历史数据的。

```
df_top32 = df_FIFA[(df_FIFA['home_team'].isin(team_list))&(df_FIFA['away_team'
    ].isin(team_list))]
```

45.1.3.1 自 1872 年以来，32 强数据情况

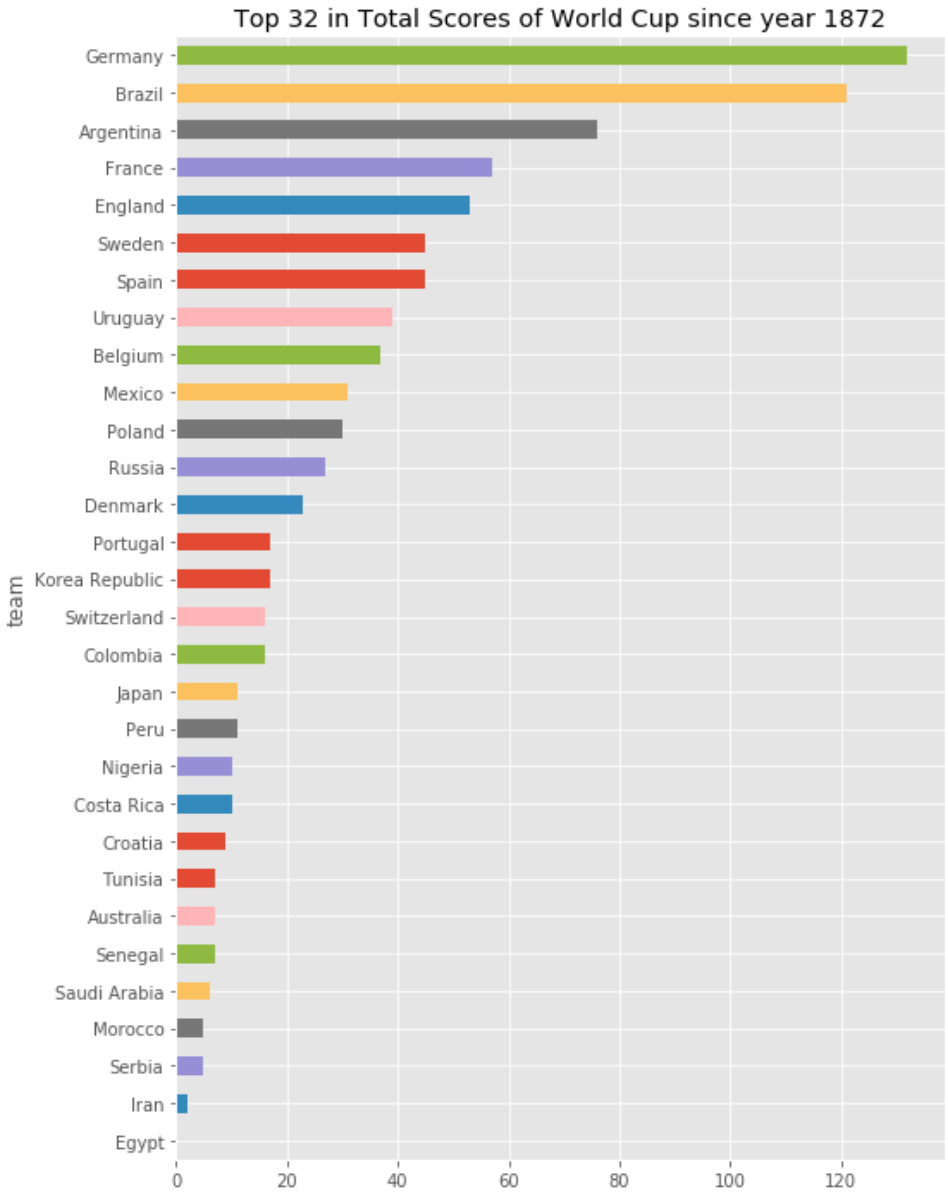
赢球场数情况

```
s_32 = df_top32.groupby('win_team')['win_team'].count()
s_32.sort_values(ascending=False, inplace=True)
s_32.drop(labels=['Draw'], inplace=True)
s_32.sort_values(ascending=True,inplace=True)
s_32.plot(kind='barh', figsize=(8,12), title='Top 32 of World Cup since year
    1872')
```



进球数据情况

```
df_score_home_32 = df_top32[['home_team', 'home_score']]
column_update = ['team', 'score']
df_score_home_32.columns = column_update
df_score_away_32 = df_top32[['away_team', 'away_score']]
df_score_away_32.columns = column_update
df_score_32 = pd.concat([df_score_home_32,df_score_away_32], ignore_index=True)
s_score_32 = df_score_32.groupby('team')['score'].sum()
s_score_32.sort_values(ascending=False, inplace=True)
s_score_32.sort_values(ascending=True, inplace=True)
s_score_32.plot(kind='barh', figsize=(8,12), title='Top 32 in Total Scores of
    World Cup since year 1872')
```



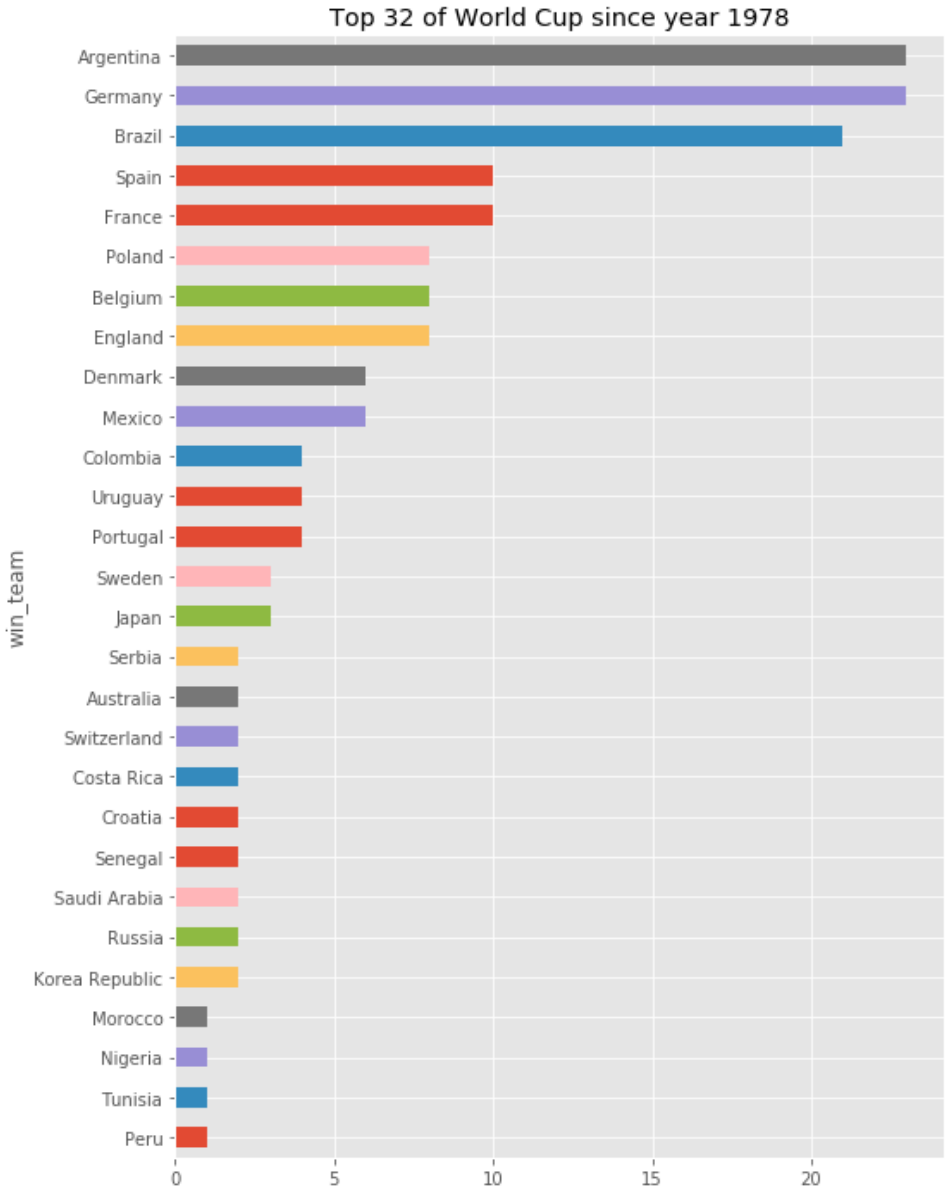
分析结论 3：自 1872 年以来，32 强之间的世界杯比赛，从赢球场数和进球数量来看，德国、巴西、阿根廷三支球队实力最强。

自 1872 年到现在，已经有 100 多年，时间跨度较大，有些国家已发生重大变化，后续分别分析自 1978 年（近 10 届）以及 2002 年（近 4 届）以来的比赛情况。

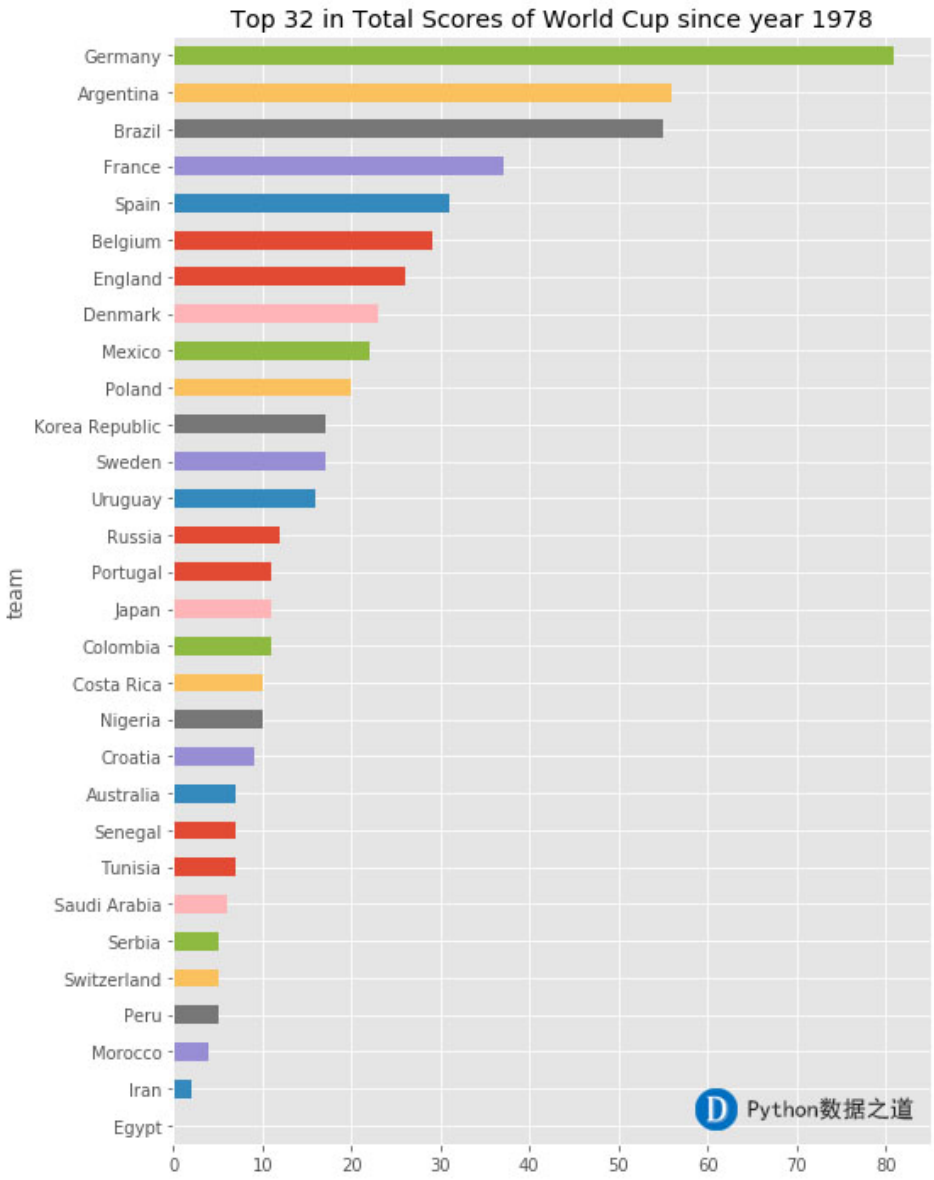
程序代码是类似的，这里只显示可视化的结果。

45.1.3.2 自 1978 年以来，32 强数据情况

赢球场数情况



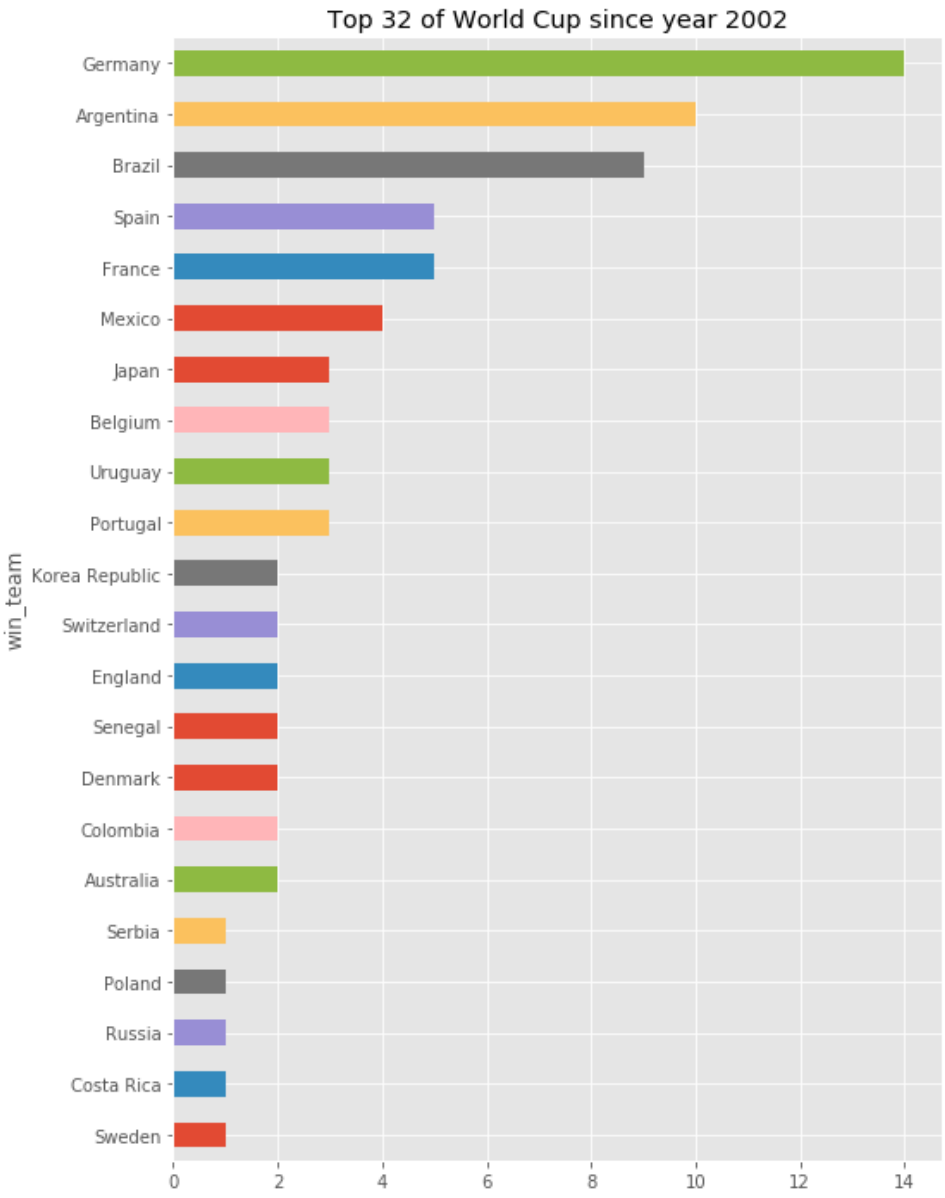
进球数据情况



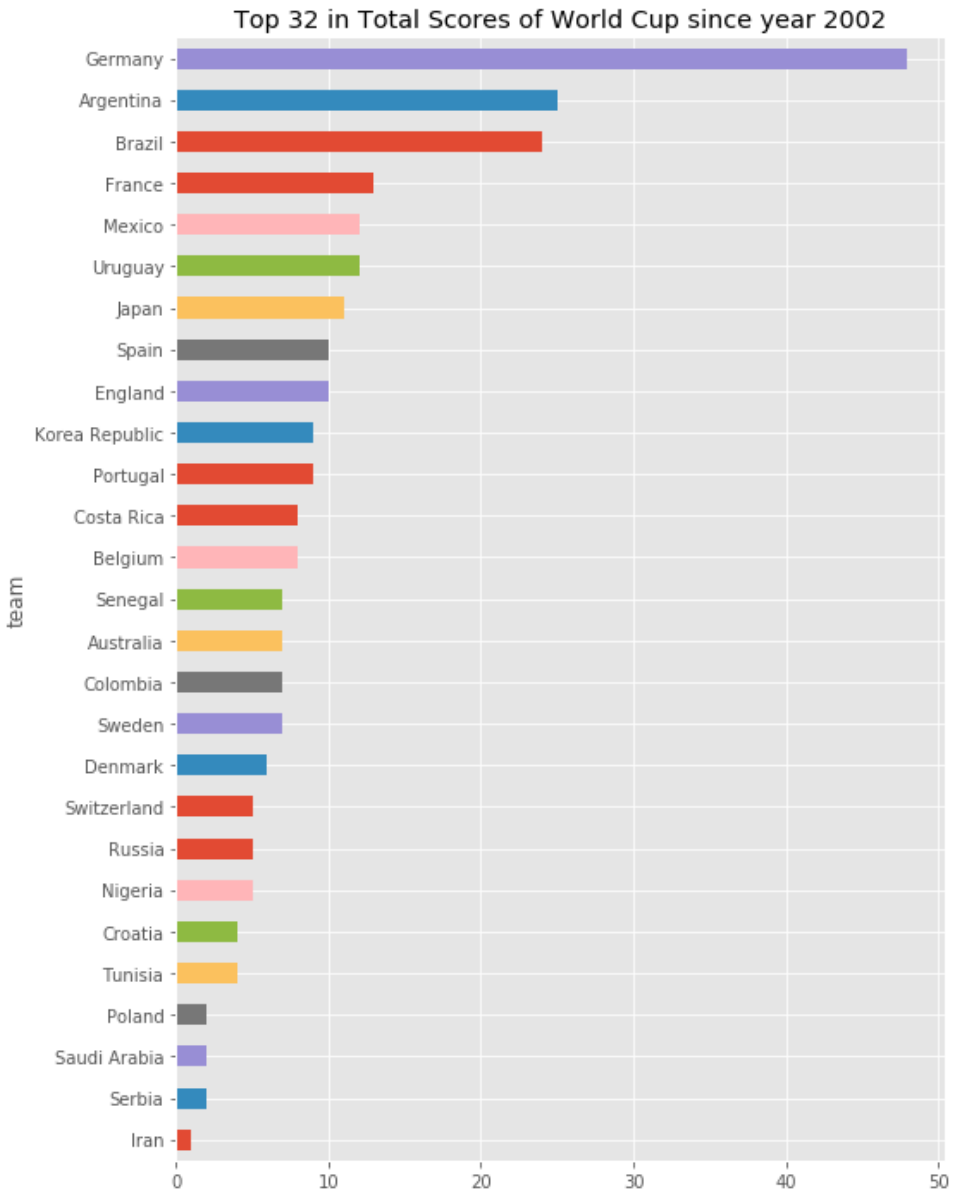
分析结论 4：自 1978 年以来，32 强之间的世界杯比赛，从赢球场数来看，阿根廷、德国、巴西三支球队实力最强。从进球数量来看，前 3 强也是这三支球队，但德国队的数据优势更明显。

45.1.3.3 自 2002 年以来，32 强数据情况

赢球场数情况


Top 32 of World Cup since year 2002
win_team
Germany
Argentina
Brazil
Spain
France
Mexico
Japan
Belgium
Uruguay
Portugal
Korea Republic
Switzerland
England
Senegal
Denmark
Colombia
Australia
Serbia
Poland
Russia
Costa Rica
Sweden
0
2
4
6
8
10
12
14

进球数据情况



分析结论 5：自 2002 年以来，32 强之间的世界杯比赛，从赢球场数和进球数量来看，德国、阿根廷、巴西三支球队实力最强。其中，德国队的数据优势更明显。

45.1.4 综合结论

2018 年世界杯的 32 支队伍，根据以往的世界杯比赛数据来看，预测前三强为德国、阿根廷和巴西，其中德国队应该是夺冠的最大热门。

> **特别说明：**以上数据分析，纯属个人学习用，预测结果与实际情况可能偏差很大，不能用于其他用途。
>
> 如需获取本文源代码，请在公众号『Python 数据之道』后台回复“code”

45.2 第二篇：德国是 2018 世界杯夺冠最大热门?

2018 年，世界杯小组赛已经在如火如荼的进行中。在上篇文章的基础上，我们继续分析世界杯 32 强的实力情况，以期能够更进一步分析本次世界杯的夺冠热门球队。

三十年河东三十年河西，对于世界杯而言，这个时间可能 4 年就足够。

前几场爆冷，使得天台上已经拥挤不堪，跳水的股市更是让天台一度混乱。

在文章开始之前，提醒大家：

> 赌球有风险，看球须尽兴

本文的重点是通过分析 32 强之间的比赛，透过历史数据来预测夺冠热门球队。

本次分析的数据来源于 Kaggle，包含从 1872 年到今年的数据，包括世界杯比赛、世界杯预选赛、亚洲杯、欧洲杯、国家之间的友谊赛等比赛，一共大约 40000 场比赛的情况。

本次的环境为

- window 7 系统
- python 3.6
- Jupyter Notebook
- pandas version 0.22.0

先来看看数据的情况：

```
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('ggplot')

# 解决matplotlib显示中文问题
plt.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题

df = pd.read_csv('results.csv')
df.head()
```

该数据集包含的数据列的信息如下：

- 日期
- 主队名称
- 客队名称
- 主队进球数（不含点球）
- 客队进球数（不含点球）
- 比赛的类型
- 比赛所在城市
- 比赛所在国家

- 是否中立

结果如下：

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral
0	1872-11-30	Scotland	England	0	0	Friendly	Glasgow	Scotland	False
1	1873-03-08	England	Scotland	4	2	Friendly	London	England	False
2	1874-03-07	Scotland	England	2	1	Friendly	Glasgow	Scotland	False
3	1875-03-06	England	Scotland	2	2	Friendly	London	England	False
4	1876-03-04	Scotland	England	3	0	Friendly	Glasgow	Scotland	False

45.2.1 获取所有世界杯比赛的数据（含预选赛）

创建一个新的列数据，包含获胜队伍的信息，以及获取所有世界杯比赛的数据，**包含预选赛。**

```
mask = df['home_score'] - df['away_score']
df.loc[mask > 0, 'win_team'] = df.loc[mask > 0, 'home_team']
df.loc[mask < 0, 'win_team'] = df.loc[mask < 0, 'away_team']
df.loc[mask == 0, 'win_team'] = 'Draw'

df_FIFA_all = df[df['tournament'].str.contains('FIFA', regex=True)]
```

结果如下：

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral	win_team
1277	1930-07-13	Belgium	USA	0	3	FIFA World Cup	Montevideo	Uruguay	True	USA
1278	1930-07-13	France	Mexico	4	1	FIFA World Cup	Montevideo	Uruguay	True	France
1279	1930-07-14	Brazil	Yugoslavia	1	2	FIFA World Cup	Montevideo	Uruguay	True	Yugoslavia
1280	1930-07-14	Peru	Romania	1	3	FIFA World Cup	Montevideo	Uruguay	True	Romania
1281	1930-07-15	Argentina	France	1	0	FIFA World Cup	Montevideo	Uruguay	True	Argentina

45.2.2 世界杯战绩分析（含预选赛）

从前文来看，在世界杯历史上，实力最强的 5 支球队是德国、阿根廷、巴西、法国、西班牙。

接下来，我们将比赛的范围扩大至包含世界杯预选赛，通过 5 支球队之间的比赛情况来进行分析。

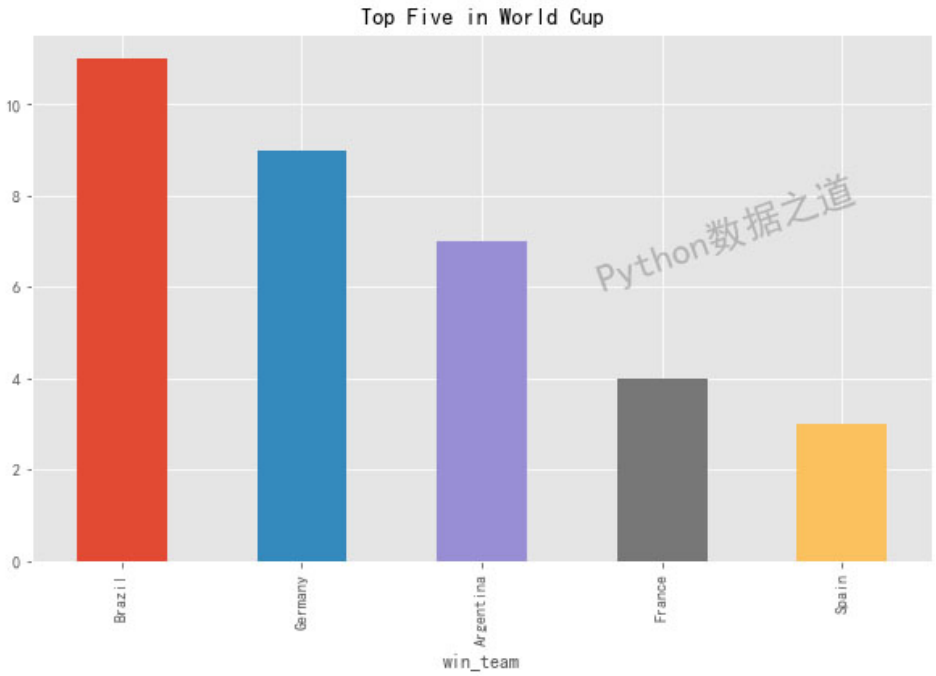
```
team_top5 = ['Germany', 'Argentina', 'Brazil', 'France', 'Spain']
df_FIFA_top5 = df_FIFA_all[(df_FIFA_all['home_team'].isin(team_top5))&
                                        (df_FIFA_all['away_team'].isin(
                                            team_top5))]
df_FIFA_top5.shape


out:
(43, 10)
```

在世界杯历史上，5 支球队在共有 43 场比赛相遇。

通过这 43 场比赛分析后，5 支球队的胜负场数排名如下：

```
s_FIFA_top5 = df_FIFA_top5.groupby('win_team')['win_team'].count()
s_FIFA_top5.drop('Draw', inplace=True)
s_FIFA_top5.sort_values(ascending=False, inplace=True)
s_FIFA_top5.plot(kind='bar', figsize=(10,6), title='Top Five in World Cup')
```

结果如下：



下面，着重来分析下这 5 支球队，在世界杯上，两两对阵时的胜负情况。

首先自定义两个函数，分别获得两支球队获胜场数情况以及自定义绘图函数

```
# 自定义函数，返回两支球队获胜场数情况
def team_vs(df,team_A,team_B):
    df_team_A_B = df[(df['home_team'].isin([team_A,team_B]))&
                                        (df['away_team'].isin([team_A,team_B]))
                                            ]
    s_win_team = df_team_A_B.groupby('win_team')['win_team'].count()
    return s_win_team

# 如需获取本文源代码，请关注公众号“Python数据之道”，
# 在公众号后台回复 “code” ，谢谢大家支持。

# 自定义函数，两支球队在世界杯的对阵胜负情况制图
def team_vs_plot(df,team_A,team_B,ax):
    s_win_FIFA = team_vs(df,team_A,team_B)
    title = team_A + ' vs ' +team_B
    s_win_FIFA.plot(kind='bar', ax =ax)
    ax.set_xlabel('')
```

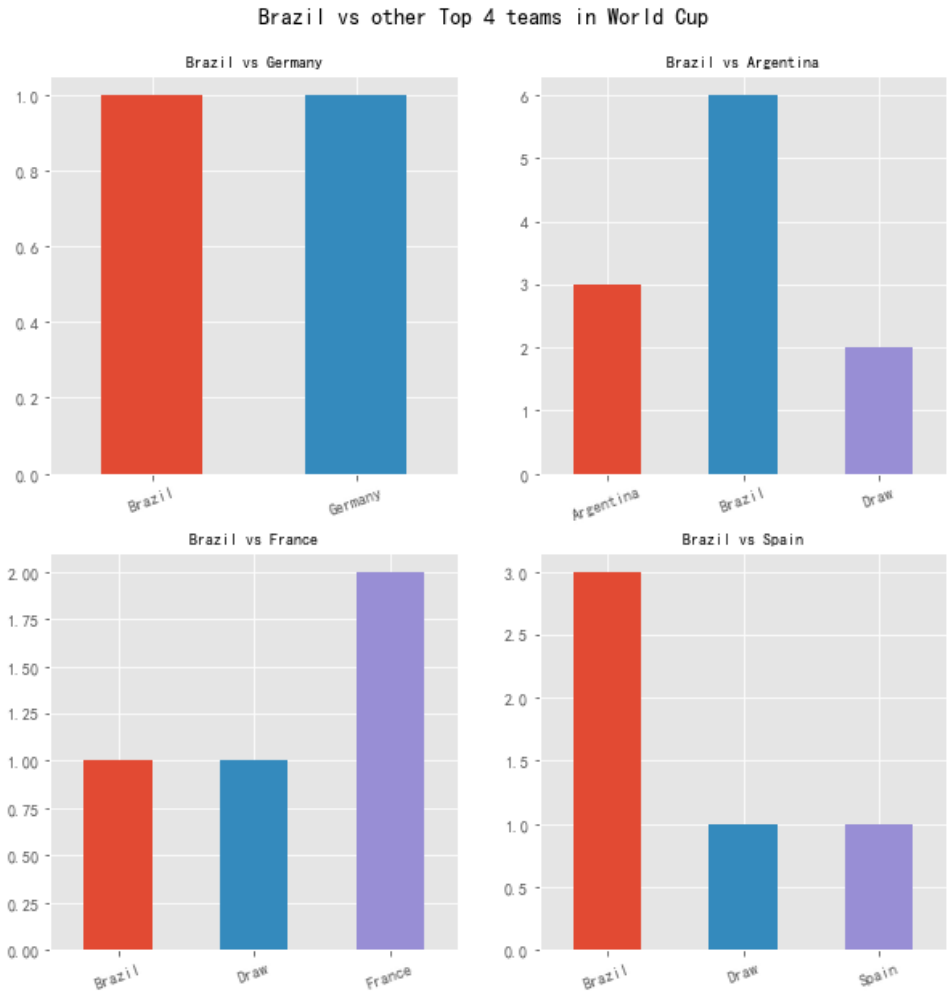
```
    ax.set_title(title,fontdict={'fontsize':10})
    ax.set_xticklabels(s_win_FIFA.index, rotation=20)
```

基于上述函数，分析结果如下：

45.2.2.1 世界杯战绩：巴西 vs 其他 4 支球队

```
f, axes = plt.subplots(figsize=(10,10), ncols=2, nrows=2)
ax1, ax2,ax3,ax4 = axes.ravel()
team_vs_plot(df_FIFA_all,'Brazil','Germany',ax=ax1)
team_vs_plot(df_FIFA_all,'Brazil','Argentina',ax=ax2)
team_vs_plot(df_FIFA_all,'Brazil','France',ax=ax3)
team_vs_plot(df_FIFA_all,'Brazil','Spain',ax=ax4)

# 如需获取本文源代码，请关注公众号“Python数据之道”，
# 在公众号后台回复 “code” ，谢谢大家支持。

# set main title of the figure
plt.suptitle('Brazil vs other Top 4 teams in World Cup', fontsize=14,
    fontweight='bold', x=0.5, y=0.94)
plt.show()
```

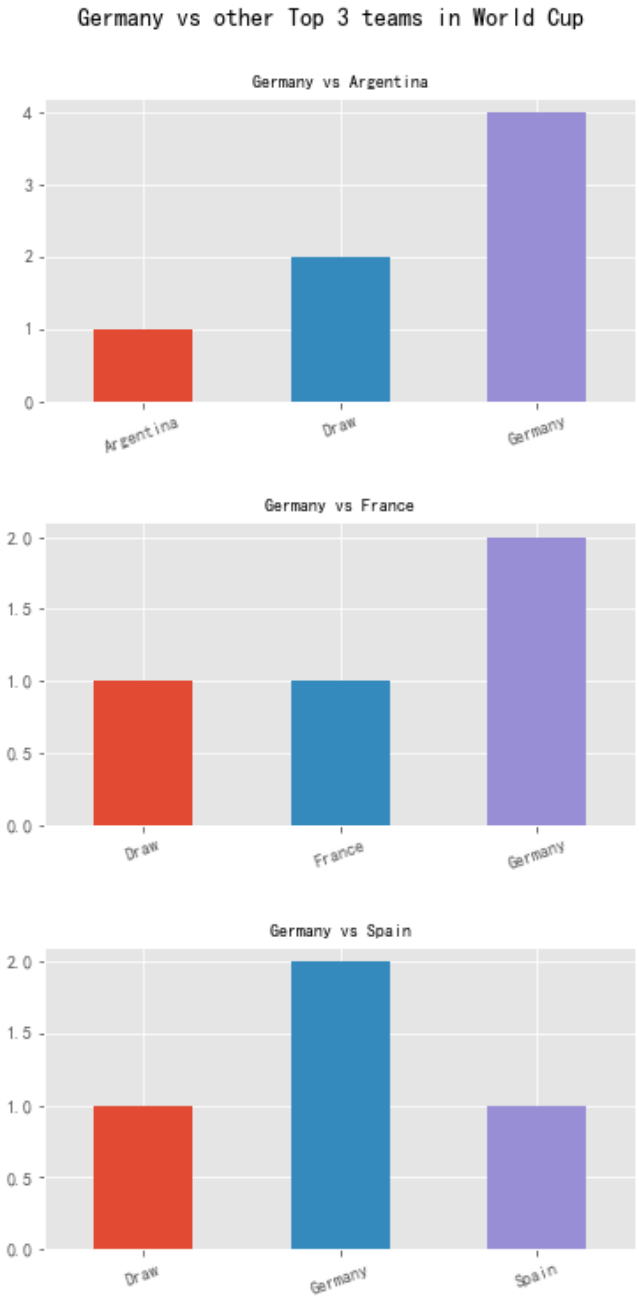
结果如下：



> **统计现象 1：**在世界杯上的战绩，统计获胜场数如下（不含平局）：巴西 1：1 德国，巴西 6：3 阿根廷，巴西 1：2 法国，巴西 3：1 西班牙巴西队，输赢不好判断……

45.2.2.2 世界杯战绩：德国 vs 其他 3 支球队

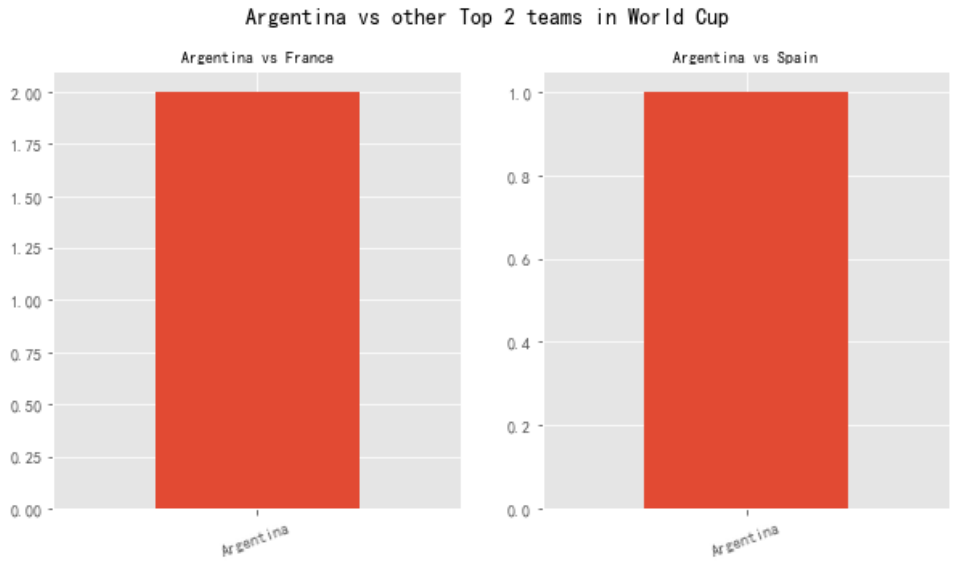
代码跟 2.1 部分是类似的，结果如下：



> **统计现象 2：**
>
> 在世界杯上的战绩，统计获胜场数如下（不含平局）：德国 4：1 阿根廷，德国 2：1 法国，德国 2：1 西班牙，德国在这 5 支持球队里，获胜的优势相对比较明显。

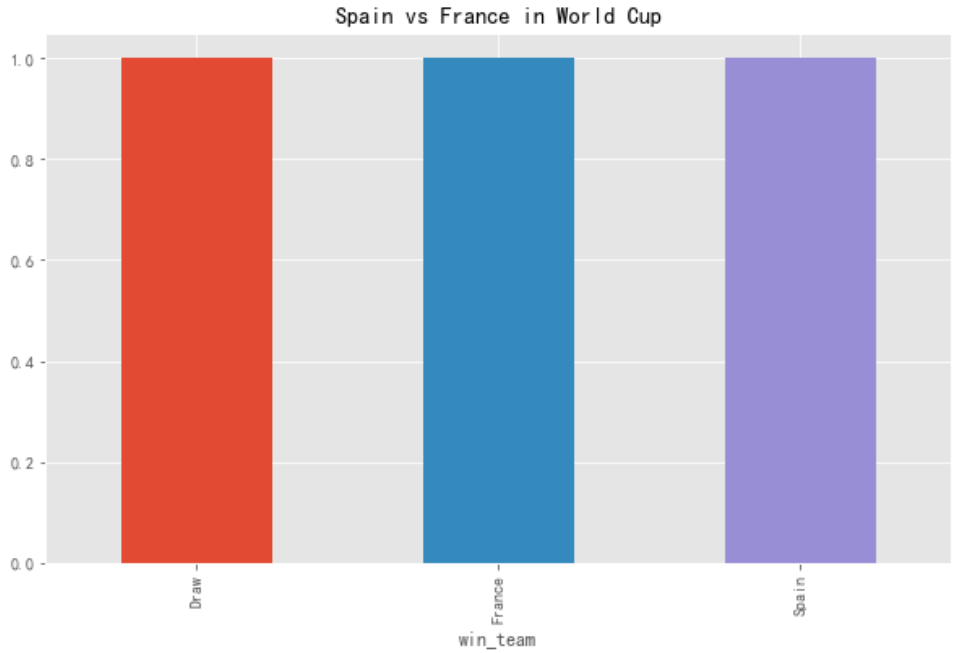
45.2.2.3 世界杯战绩：阿根廷 vs 其他 2 支球队

代码跟 2.1 部分是类似的，结果如下：



> **统计现象 3：**在世界杯上的战绩，统计获胜场数如下（不含平局）：阿根廷 2：0 法国，阿根廷 1：0 西班牙但阿根廷不敌巴西和德国

45.2.2.4 世界杯战绩：西班牙 vs 法国



> **综合小结论：**从历届世界杯上的表现情况来看，分析 5 强之间两两对阵后，发现德国队的表现是最好的。其次巴西和阿根廷的表现也不错。

考虑到，历届世界杯的数据，时间跨度很大，很多球队其实已经发生了很大变化。

球队真实的情况，可能选择近几年的比赛，以及包含不同级别的比赛，可能分析效果要更好些。

下面，重点来分析 2014 年以来包含所有比赛的情况。

45.2.3 2014 年以来，所有比赛的战绩对比

首先，时间选择 2014 年之后（含 2014 年），距离现在的时间比较近，相对来说，球队人员的组成变化小一些。

当然，这里的时间选择，对于结果是有影响的。大家可以探讨下这个因素带来的影响。

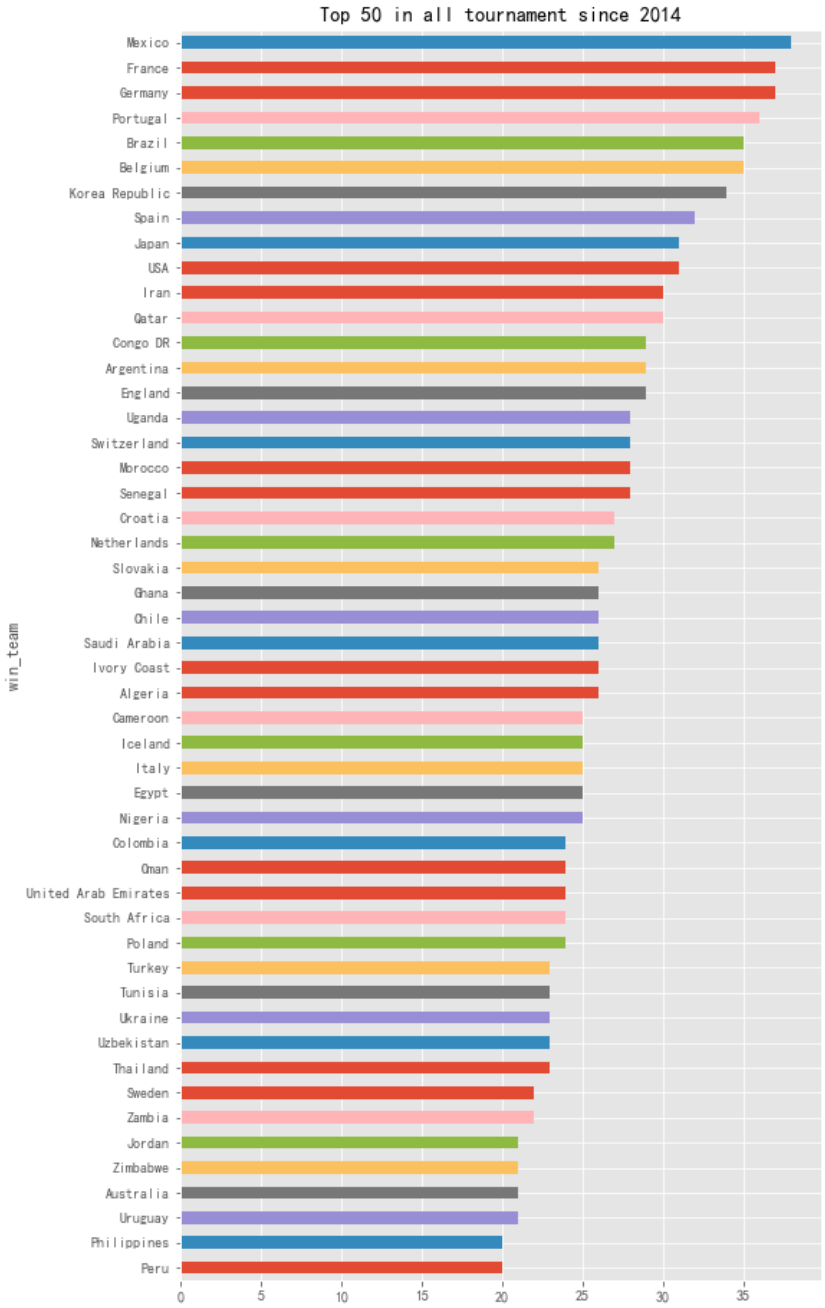
45.2.3.1 2014 年以来所有球队所有比赛胜负情况概览

```
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df_since_2014 = df[df['year']>=2014]
df_since_2014.shape
```

2014 年以来，共有 3600 多场比赛。

针对 3600 多场比赛分析后，胜负场数情况如下：

```
s_all = df_since_2014.groupby('win_team')['win_team'].count()
s_all.drop('Draw', inplace=True)
s_all.sort_values(ascending=True, inplace=True)
s_all.tail(50).plot(kind='barh', figsize=(8,16), tick_label='',title='Top 50 in
    all tournament since 2014')
```



从上图来看，**2014 年以来，墨西哥，法国，德国、葡萄牙、巴西、比利时、韩国和西班牙表现相对较好。**

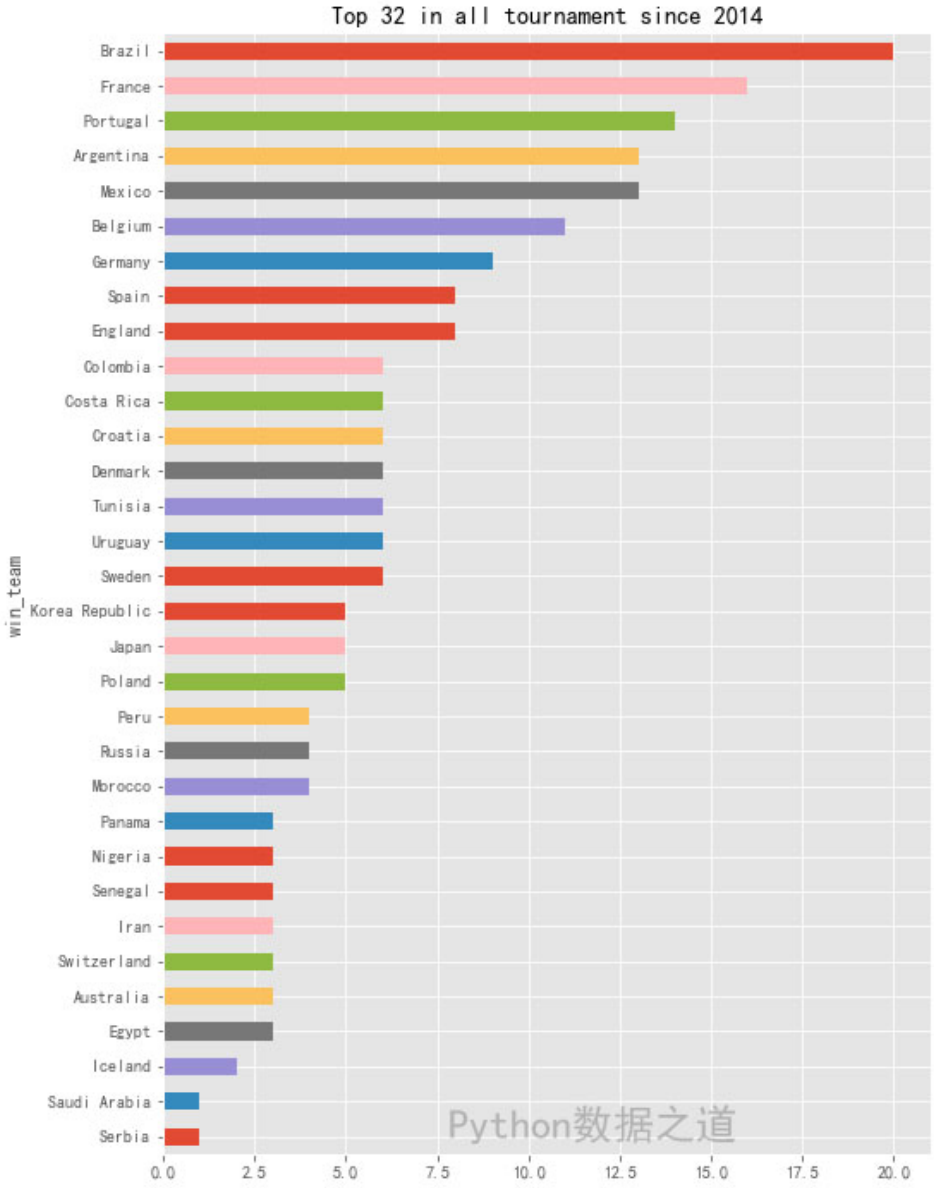
结果是不是跟想象中的有些差异?

6 月 17 日的小组赛，德国不敌墨西哥，看来也不是全无理由的。

但是，**本次我们主要还是要考虑 32 强之间的对阵，这样更能反映现实情况。**

45.2.3.2 2014 年以来 32 强相互之间在所有比赛中的概览情况

```
team_list = ['Russia', 'Germany', 'Brazil', 'Portugal', 'Argentina', 'Belgium', 'Poland', 'France',
             'Spain', 'Peru', 'Switzerland', 'England', 'Colombia', 'Mexico', 'Uruguay', 'Croatia',
            'Denmark', 'Iceland', 'Costa Rica', 'Sweden', 'Tunisia', 'Egypt', 'Senegal', 'Iran',
            'Serbia', 'Nigeria', 'Australia', 'Japan', 'Morocco', 'Panama', 'Korea Republic', 'Saudi Arabia']

df_top32 = df_since_2014[(df_since_2014['home_team'].isin(team_list))&(df_since_2014['away_team'].isin(team_list))]

s_top32 = df_top32.groupby('win_team')['win_team'].count()
s_top32.drop('Draw', inplace=True)
s_top32.sort_values(ascending=True, inplace=True)
s_top32.plot(kind='barh', figsize=(8,12), tick_label='',title='Top 32 in all tournament since 2014')
# plt.ylabel('')
```



从上图来看，自 2014 年以来，**巴西、法国、葡萄牙、阿根廷、墨西哥、比利时、德国、西班牙、英国为前 9 强**。

下面我们来分析 top9 之间的胜负情况。

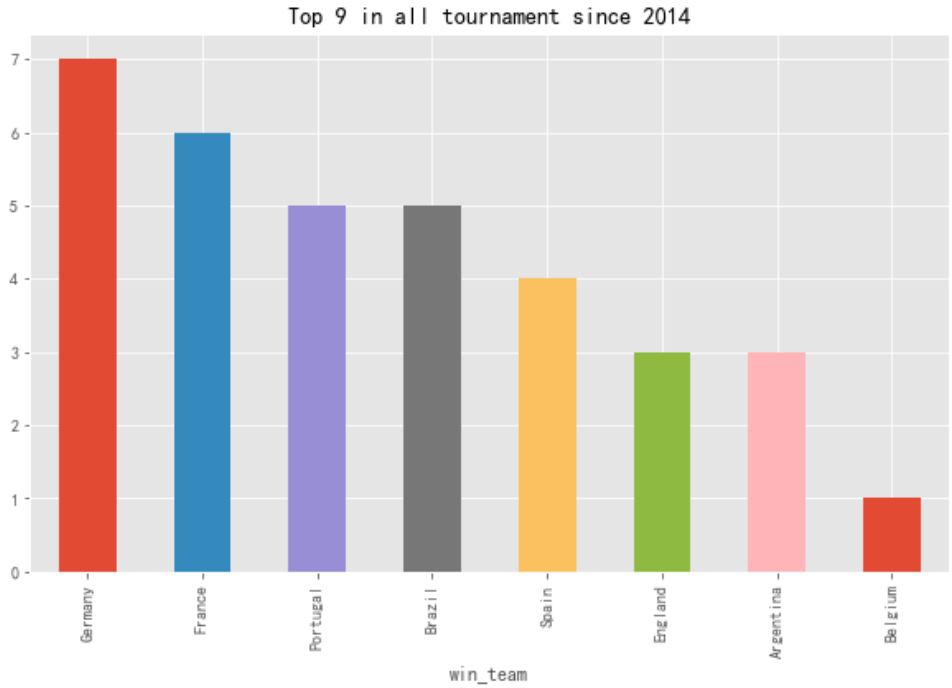
```
team_top9 = [ 'Brazil', 'France', 'Portugal',
              'Argentina','Mexico','Belgium',
              'Germany','Spain','England']
df_top9 = df_since_2014[(df_since_2014['home_team'].isin(team_top9))&
                        (df_since_2014['away_team'].isin(team_top9))]
df_top9.shape
```

2014 年以来，top 9 之间一共踢了 44 场比赛（包括友谊赛）。

总体来说，比赛的场数不是太多，基于这些数据来分析，可能对结果会有较大的影响。

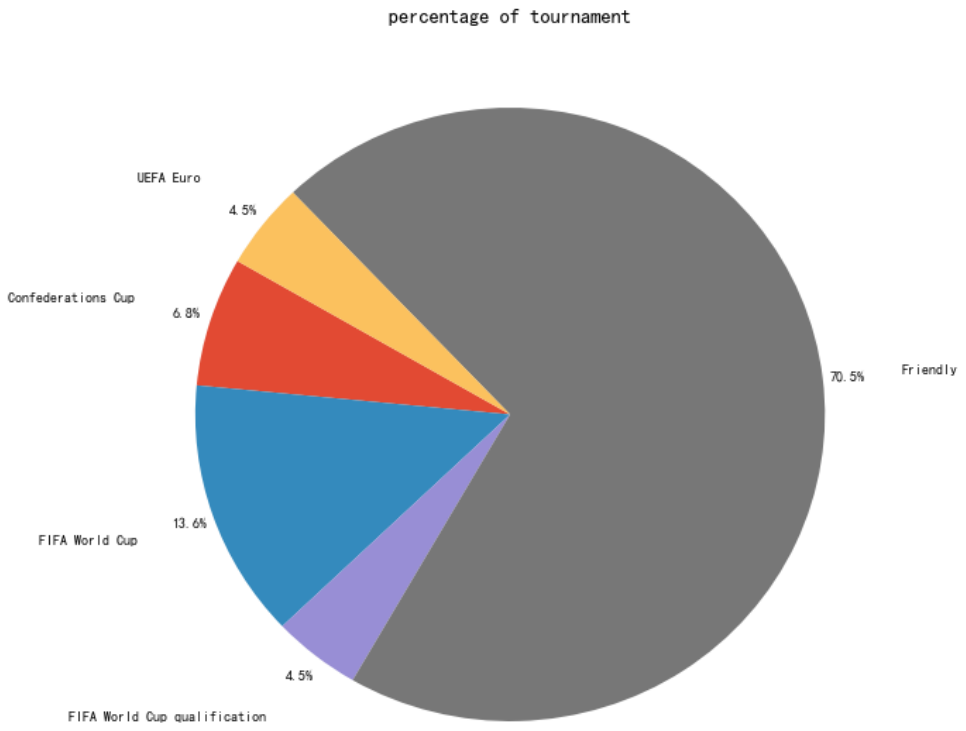
九强排名如下：

```
s_top9 = df_top9.groupby('win_team')['win_team'].count()
s_top9.drop('Draw', inplace=True)
s_top9.sort_values(ascending=False, inplace=True)
s_top9.plot(kind='bar', figsize=(10,6), title='Top 9 in all tournament since
    2014')
```



来查看下都统计了哪些类型的比赛

```
s_tournament = df_top9.groupby('tournament')['tournament'].count()
s_tournament_percentage = s_tournament/s_tournament.sum()
# s_tournament_percentage.sort_values(ascending=False, inplace=True)
s_tournament_percentage.tail(20).plot(kind='pie', figsize=(10,10), autopct='%.1
    f%%',
                          startangle=150, title='percentage of tournament',
                              label='',
                                    labeldistance=1.25, pctdistance=1.08)
```



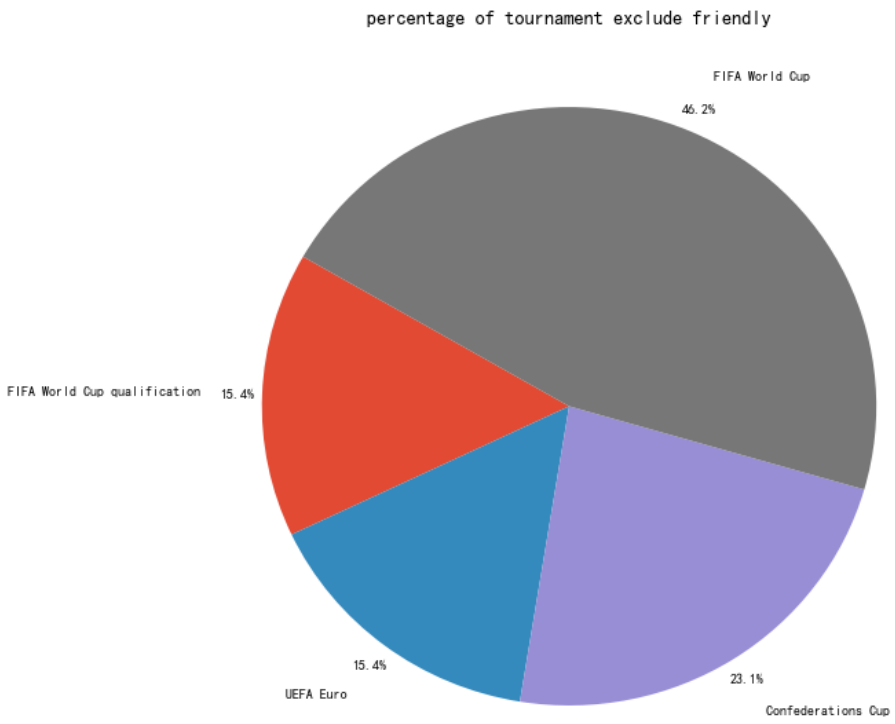
从上面来看，友谊赛占的比例较大。

考虑到友谊赛在有些情况下可能不能比较准确的反映出球队的真实水平，且友谊赛站的场数比例较大，我们剔除友谊赛再来看看结果情况。

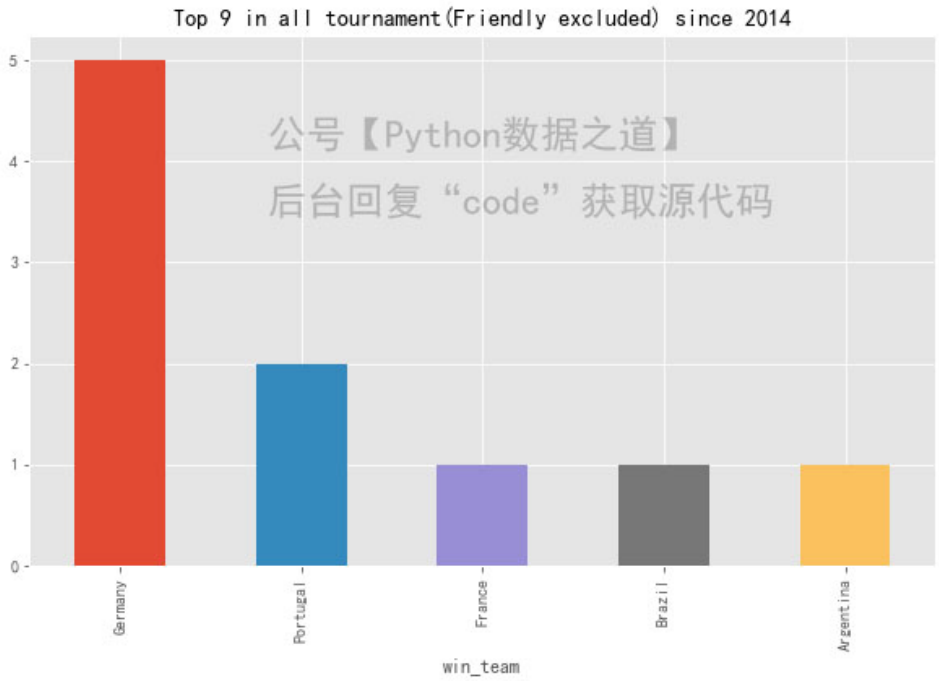
45.2.3.3 2014 年以来 32 强剔除友谊赛后的胜负情况概览

```
df_top9_no_friendly = df_top9[df_top9['tournament']!= 'Friendly']
df_top9_no_friendly.groupby('tournament')['tournament'].count()

out:
tournament
Confederations Cup              3
FIFA World Cup                  6
FIFA World Cup qualification    2
UEFA Euro                       2
Name: tournament, dtype: int64
```

剔除友谊赛后，比赛类型分布如下：



剔除友谊赛后，Top 9 的情况如下：



在概览中可以看出，是否剔除友谊赛（Friendly），对排名还是有影响的。

另外，剔除友谊赛后，总的比赛场数更少了（只有 13 场），9 强之间有些队伍没有比赛，或者没有赢过，这个数据用来分析的作用更有限。

当然，在分析中 **是否要剔除友谊赛，应该是值得商榷的。**

45.2.3.4 九强两两对阵的胜负情况概览

这里，我们后续分析采用包含友谊赛的数据，来分别分析 9 强之间两两对阵的情况，看看哪支球队的胜率更高些。

首先自定义几个函数，方便进行分析。

自定义获取球队某年至今获胜比例函数

```
# 自定义获取球队某年至今获胜比例函数
def probability(df,year,team_A,team_B):
    prob = []
    df_year = df[df['year']>= year]
    s = team_vs(df_year,team_A,team_B)
    s_team_A = 0 if s.get(team_A) is None else s.get(team_A)
    s_A_win = s_team_A/s.sum()
    s_team_B = 0 if s.get(team_B) is None else s.get(team_B)
    s_B_win = s_team_B/s.sum()
    s_draw = 1 - s_A_win - s_B_win
    prob.append(year)
    prob.append(s_A_win)
    prob.append(s_B_win)
    prob.append(s_draw)
    return prob
```

自定义获取两支球队历史获胜情况对比函数

```
# 自定义获取两支球队历史获胜情况对比函数
def his_team_data(df,year_start,year_end,team_A,team_B):
    row_team = []
#     df_team = pd.DataFrame(columns=('year', 'team_A_win', 'team_B_win', 'draw
    '))
    for yr in list(range(year_start,year_end+1)):
        team_A_vs_team_B = probability(df,yr,team_A,team_B)
        row_team.append(team_A_vs_team_B)
    team_A_win = team_A + '_win_percentage'
    team_B_win = team_B + '_win_percentage'
    df_team = pd.DataFrame(row_team, columns=('year', team_A_win, team_B_win, '
        draw_percentage'))
    return df_team
```

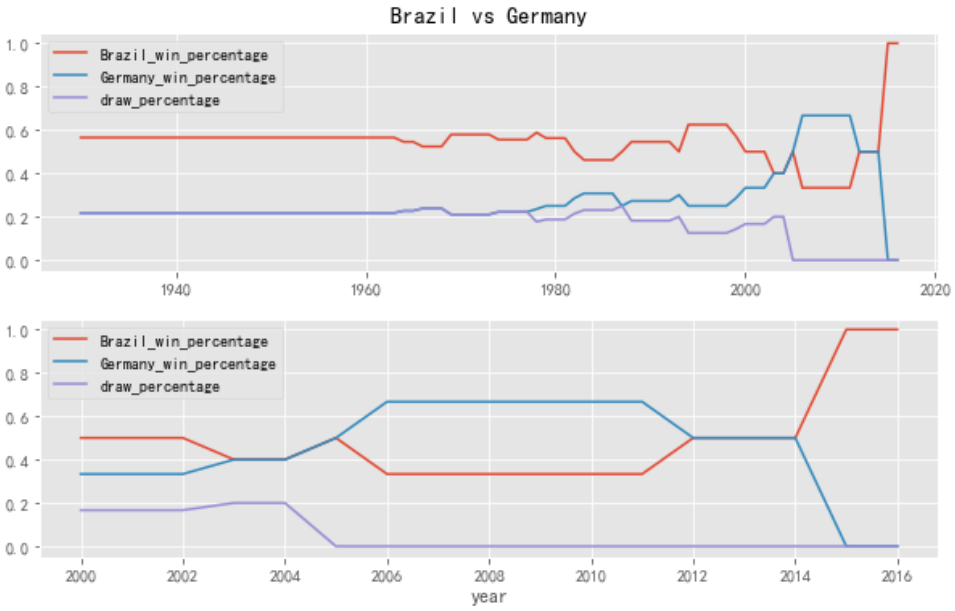
自定义两支球队历史获胜情况制图函数

```
# 自定义两支球队历史获胜情况制图函数
def team_plot(df,year_start,year_end,team_A,team_B,ax):
    team_A_vs_team_B = team_A + '_vs_' + team_B
    team_A_vs_team_B = his_team_data(df,year_start,year_end,team_A,team_B)
```

```
    title = team_A + ' vs ' + team_B
    columns = [team_A+'_win_percentage',team_B+'_win_percentage','
        draw_percentage']
    team_A_vs_team_B.set_index('year')[columns].plot(kind='line',figsize=(10,6)
        , title=title,ax=ax)
```

这些函数有什么用呢，首先我们来分析下 **巴西 vs 德国**的情况，如下：

```
team_A = 'Brazil'
team_B = 'Germany'

f, axes = plt.subplots(figsize=(6,12), ncols=1, nrows=2)
ax1, ax2 = axes.ravel()

team_plot(df,1930,2016,team_A,team_B,ax1)
ax1.set_xlabel('')
team_plot(df,2000,2016,team_A,team_B,ax2)
ax2.set_title('')

plt.show()
```



上述图中，x 轴代表的含义是从某年至今（数据集含有部分 2018 年的比赛数据），两支球队的胜负情况。例如 2012 对应的是德国跟巴西从 2012 年至今，两支球队的胜负情况。

所以，时间越早，两支球队的比赛数量越多，数据曲线的波动可能要小些。但由于球队的成员组成在不断的变化，会导致越早的数据，其分析价值越弱。**因此，选择合适的年份进行分析就显得很重要。**

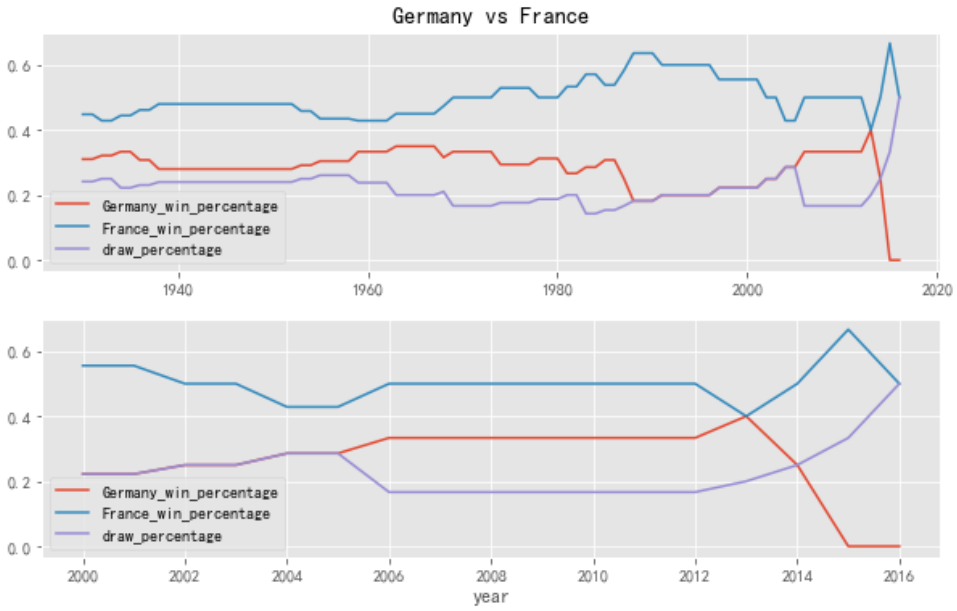
有童鞋说，如果我要 **同时分析德国对阵另外 8 支球队呢?**

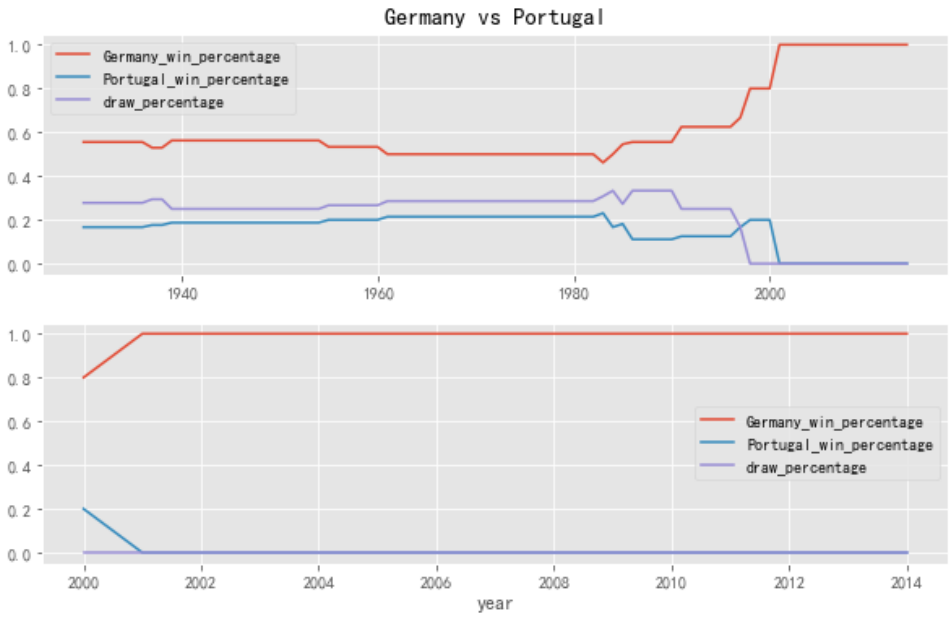
这里，用上面的函数，也是很迅速的，代码如下：

```
team_A = 'Germany'

for team in ['France','Portugal', 'Argentina','Mexico','Belgium','Brazil','
    Spain','England']:
    team_B = team

    f, axes = plt.subplots(figsize=(6,12), ncols=1, nrows=2)
    ax1, ax2 = axes.ravel()

    team_plot(df,1930,2016,team_A,team_B,ax1)
    ax1.set_xlabel('')
    team_plot(df,2000,2016,team_A,team_B,ax2)
    ax2.set_title('')

    plt.show()
```

运行上述代码后，将会绘制 8 张图，下面只放上其中几张图。

同理，如果你喜欢巴西队或者别的球队，也可以用上述代码进行分析。





用上述函数可以快速的分析两支球队的历史胜负情况，当然，有些球队之间，相遇很少，或者近些年没有遭遇过，那分析结果可能就不好用了。

当然，数据分析的只是历史情况，足球是圆的，场上瞬息万变。比如，**阿根廷现在岌岌可危，梅西内心慌得一逼……**

45.2.4 预测

本届世界杯真的是爆冷太多。

意大利，荷兰，连小组赛都没进；阿根廷，可以说现在已凉了半截；德国队，若不是最后的绝杀，也差不过可以送首凉凉了，不过现在看已回血大半。

最后，来放上我的神预测。

黑马年年有，今年特别多，预测不准，坐等 pia pia 打脸。

当然，**其实我内心深处希望是下面这样的。**

怎么样，为强大的内心点赞吧 ~~

> **特别说明：**以上数据分析，纯属个人学习用，预测结果与实际情况可能偏差很大，不能用于其他用途。
>
> 如需获取本文源代码，请在公众号『Python 数据之道』后台回复“code”

46 项目实战：福布斯系列

46.1 数据分析思路

福布斯每年都会发布福布斯全球上市企业 2000 强排行榜（Forbes Global 2000），这个排行榜每年发布的时候，国内外总有新闻会热闹的讨论一番，但很少见到比较全面的分析。

因此才有了这样一个想法，搜集近些年每年发布的排行榜，做一个进一步的分析。

在准备做这个小小的项目前，先理了一下整个思路，大概可以分为下面这几个步骤：

1. 数据采集
2. 原始数据完整性检查
3. 数据清洗、整理
4. 从不同角度对数据进行分析
5. 数据可视化
6. 总结

整个分析过程会涉及多篇文章，主要使用 Python 来进行分析。

- 数据采集: 主要涉及的 python 库包括 requests，BeautifulSoup，csv，以及一些其他常用工具。
- 数据完整性检查: 包括不同数据来源的对比，以及其他一些常识性的知识。需要对比数据量的多少是否完整，以及有些数据是否缺失。

当然，在拿到数据的初期，其实只能做一个初步的判断，有些内容是在整个分析过程中发现的。

- 数据清洗与整理: 主要用到 Pandas、Numpy 以及其他常用库和函数。由于数据比较杂乱，数据清洗与整理涉及的内容比较多，可以说是整个福布斯系列的重点之一。

同时，这个也印证了通常我们所说的数据清洗与整理可能占整个分析的 50~80%。

- 数据分析与可视化: 经常是伴随在一起的。主要根据不同分析目的进行分析与可视化。用到的工具包括 Pandas、Numpy、Matplotlib、Seaborn 以及其他一些相关库。

希望能通过福布斯系列的实战来对数据分析的知识点与工具作一个简单的示例整理与分享。

46.2 数据采集

46.2.1 数据采集概述

开始一个数据分析项目，首先需要做的就是 get 到原始数据，获得原始数据的方法有多种途径。比如：

1. 获取数据集（dataset）文件
2. 使用爬虫采集数据
3. 直接获得 excel、csv 及其他数据文件
4. 其他途径...

本次福布斯系列数据分析项目实战，数据采集方面，主要数据来源于使用爬虫进行数据采集，同时也辅助其他数据进行对比。

本节主要是介绍使用爬虫进行数据采集的思路和步骤。

本次采集的福布斯全球上市企业 2000 强排行榜数据，涉及年份从 2007 年到 2017 年，跨越 10 多年。

本次采集的目标网站，是多个网页，但多个网页的分布结构都有所不同，虽然思路和步骤都差不多，但需要分开来编写，分别采集。

46.2.2 数据采集步骤

数据采集大体分为几步：

1. 目标主网页内容的 Download
2. 主网页上数据的采集
3. 主网页上其他分发页面网站链接的采集
4. 各分发网页数据的 download 与采集
5. 将采集的数据保存

涉及到的 python 库包括，requests、BeautifulSoup 以及 csv。下面以采集某年的数据为案例，来描述下数据采集的步骤。

```
import requests
from bs4 import BeautifulSoup
import csv
```

46.2.2.1 数据 Download 模块

主要是基于 requests，代码如下：

```
def download(url):
    headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
        AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari
        /537.36'}
    response = requests.get(url,headers=headers)
    # print(response.status_code)
    return response.text
```

这个模块会在主网页数据下载，以及各个分页面数据下载时使用，是一个比较通用的模块。

46.2.2.2 主网页上数据的采集

主网页的页面结构，主要分为两个部分，一类是包含其他页面数据的网页链接，一类是主网页上的公司数据列表，以表格形式在网页上显示。

用 BeautifulSoup 可以把这些数据解析出来。代码模块如下：

- 解析主网页上的公司数据列表信息

```
def get_content_first_page(html, year):
    '''
    获取排名在1-100的公司列表，且包含表头
    '''
    soup = BeautifulSoup(html, 'lxml')
    body = soup.body
    body_content = body.find('div', {'id': 'bodyContent'})
    tables = body_content.find_all('table', {'class': 'XXXXtable'})

    # tables一共有3个，最后一个才是我们想要的
    trs = tables[-1].find_all('tr')

    # 获取表头名称
    # trs[1]，这里跟其他年份不一样
    row_title = [item.text.strip() for item in trs[1].find_all('th')]
    row_title.insert(0, '年份')

    rank_list = []
    rank_list.append(row_title)
    for i, tr in enumerate(trs):
        if i == 0 or i == 1:
            continue
        tds = tr.find_all('td')

        # 获取公司排名及列表
        row = [ item.text.strip() for item in tds]
        row.insert(0, year)
        rank_list.append(row)
    return rank_list
```

- 解析主网页上其他页面的网页链接

```
def get_page_urls(html):
    '''
    获取排名在101-2000的公司的网页链接
    '''
    soup = BeautifulSoup(html, 'lxml')
    body = soup.body
    body_content = body.find('div', {'id': 'bodyContent'})
    label_div = body_content.find('div', {'align':'center'})
    label_a = label_div.find('p').find('b').find_all('a')

    page_urls = ['basic_url' + item.get('href') for item in label_a]
```

```
12      return page_urls
```

46.2.2.3 各个分发页面上的数据采集

步骤也是网页页面下载和表格类数据爬取。代码内容跟主网页页面类似，只是细节上有些差异，这里就不作赘述了。

46.2.2.4 数据存储

采集的数据，最后保存到 csv 文件中。模块代码如下：

```
1  def save_data_to_csv_file(data, file_name):
2      '''
3      保存数据到csv文件中
4      '''
5      with open(file_name, 'a', errors='ignore', newline='') as f:
6          f_csv = csv.writer(f)
7          f_csv.writerows(data)
```

46.2.2.5 数据采集主函数

```
1  def get_forbes_global_year_2007(year=2007):
2      url = 'url'
3      html = download(url)
4      # print(html)
5
6      data_first_page = get_content_first_page(html, year)
7      # print(data_first_page)
8      save_data_to_csv_file(data_first_page, 'forbes_'+str(year)+'.csv')
9
10     page_urls = get_page_urls(html)
11     # print(page_urls)
12
13     for url in page_urls:
14         html = download(url)
15         data_other_page = get_content_other_page(html, year)
16         # print(data_other_page)
17         print('saving data ...', url)
18         save_data_to_csv_file(data_other_page, 'forbes_'+str(year)+'.csv')
19
20
21 if __name__ == '__main__':
22
23     # get data from Forbes Global 2000 in Year 2009
24     get_forbes_global_year_2007()
```

46.2.3 小结

本节只介绍了数据采集的思路与各个模块，并没有提供目标网页的链接，一方面由于原始网页的数据信息比较杂乱，采集的时候需要写多个采集程序，另外一方面，由于我们的重点在于后续的数据分析部分，希望不要着重于数据爬取。

46.3 数据完整性检查

福布斯系列之数据完整性检查 - Python 数据分析项目实战

在上文中，介绍了使用 python 爬取福布斯全球上市企业 2000 强排行榜（Forbes Global 2000）数据的思路和步骤。

在获取数据后，首先需要对获得的数据的完整性情况进行检查，主要查看数据是否有缺失、是否有重复、是否有数据错位或其他异常情况。

数据完整性检查也包括不同数据来源的对比，以及其他一些常识性的知识。当然，在拿到数据的初期，其实只能做一个初步的判断，有些内容是在整个分析过程中发现的。

46.3.1 每年企业数量检查

福布斯全球上市企业 2000 强排行榜，企业数量应该是 2000 家，或者稍微多几家（因为排名可能相同，会增加少量企业）。

所以首先需要检查这个排行榜上每年的企业数量，如果明显多余 2000 家或者少于 2000 家企业，则说明获得的数据可能异常。

在使用 Python 分析时，可以用 Pandas 来进行检查，使用`DataFrame.shape`就可以查看企业数量。

下面以 2007 年的数据为例来介绍。代码如下：

```
df_2007 = pd.read_csv('./data/data_forbes_2007.csv', encoding='gbk', thousands=
    ',')
print('the shape of DataFrame: ', df_2007.shape)

out:
the shape of DataFrame:  (2000, 9)
```

从运行结果可以看出，2007 年的企业数量是 2000 家，是符合要求的。用同样的方法，逐个检查其他年份，发现 2013 年和 2015 年的企业数量有异常。

通过查看，发现 **2013 年的企业数量只有 1991 家，少于 2000 家。**

```
df_2013 = pd.read_csv('./data/data_forbes_2013.csv', encoding='gbk',header=None
    )
print('the shape of DataFrame: ', df_2013.shape)

out:
the shape of DataFrame:  (1991, 9)
```

同时，**2015 年的企业数量为 2020 家，明显多余 2000 家，可能有重复的内容。**

```
df_2015 = pd.read_csv('./data/data_forbes_2015.csv', encoding='gbk',header=None
    )
print('the shape of DataFrame: ', df_2015.shape)

out:
the shape of DataFrame:  (2020, 9)
```

上述的这些问题，都需要在后续的数据清洗、分析中进一步处理。

46.3.2 数据错位、数据缺失等情况检查

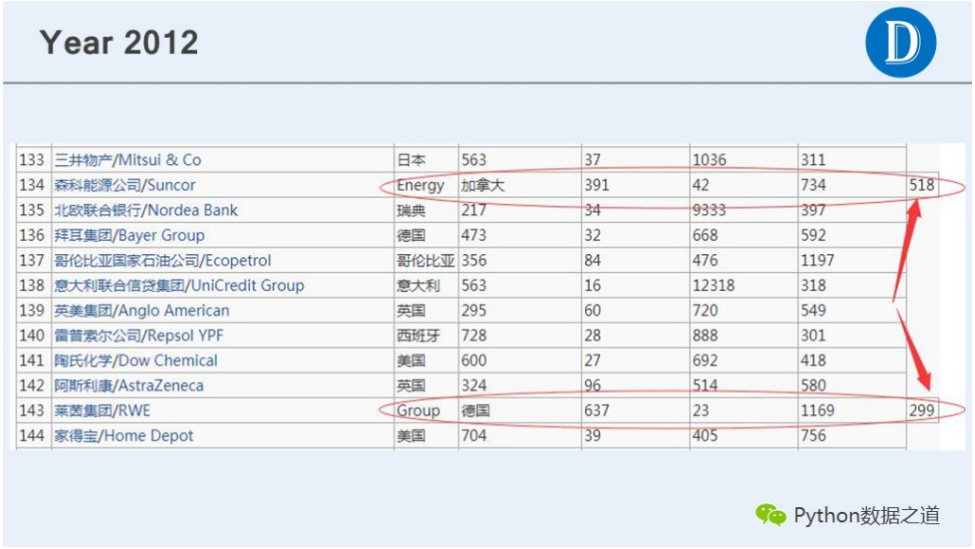
通过查看 csv 数据文件以及目标网页等文件，发现有部分数据错位或缺失等情况，需要分别处理。

46.3.2.1 数据错位

通过检查发现，2012 年和 2014 年的数据有错位的现象，通过手动在 csv 文件中进行调整相关数据，以便在后续分析中继续使用这些数据。

2012 年的情况

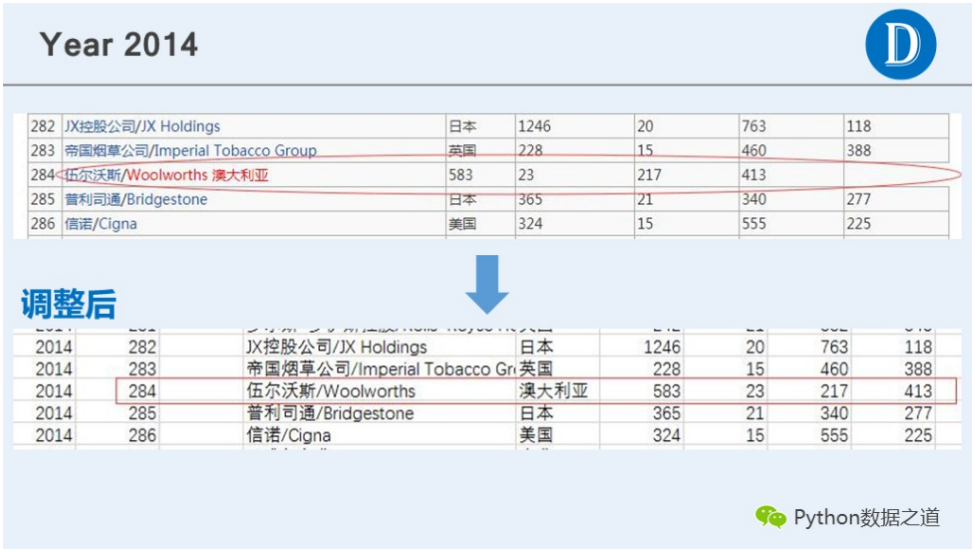
原始数据如下：



调整后的 csv 文件如下：

2014 年的情况

原始数据以及调整后的 csv 文件如下：



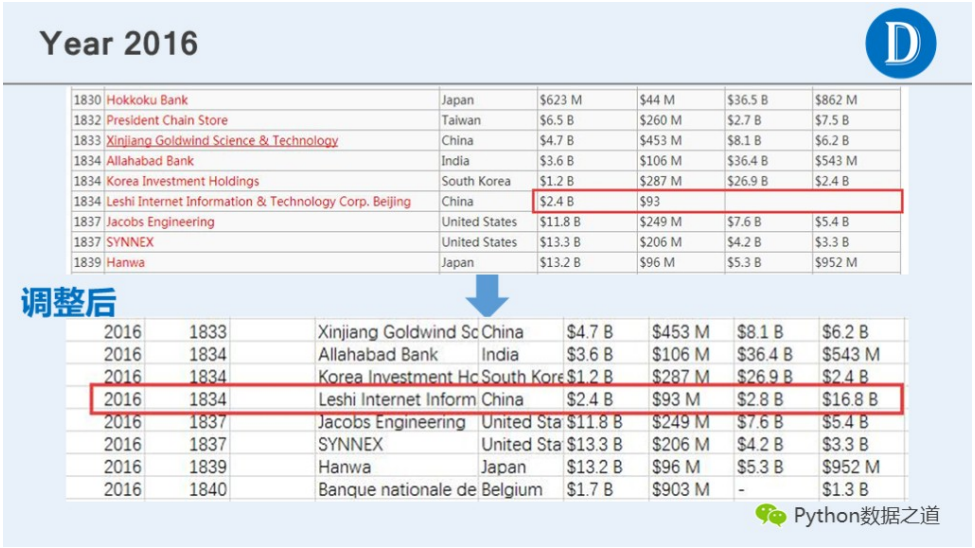
46.3.2.2 数据缺失

通过检查发现，2016 年的数据有缺失的现象，通过查找其他信息，发现其他网站有完整的信息。

对比网站的数据，进行了手动修改。用来对比的网站信息如下：

http://www.askci.com/news/finance/20160530/13364822511_7.shtml

通过手动在 csv 文件中进行调整相关数据，以便在后续分析中继续使用这些数据。



46.3.3 小结

本次数据完整性检查主要包括企业数量是否完整，以及相关数据是否缺失、错位等情况的检查。

当然，有可能还有其他问题暂时没有发现，在后续数据清洗以及分析过程中，若发现有相关问题，可以进一步处理。

46.4 补充数据收集

福布斯系列之补充数据收集 - Python 数据分析项目实战

在之前分析中，提到 2013 年的企业数量只有 1991 家，少于 2000 家。数据可能存在缺失与不完整。

于是，继续来寻找其他可用的数据源。

46.4.1 使用爬虫获取数据

通过网站搜索，发现 Economy Watch 网站有相关数据，且数据结构比较规范，质量较好，于是进行数据爬取。

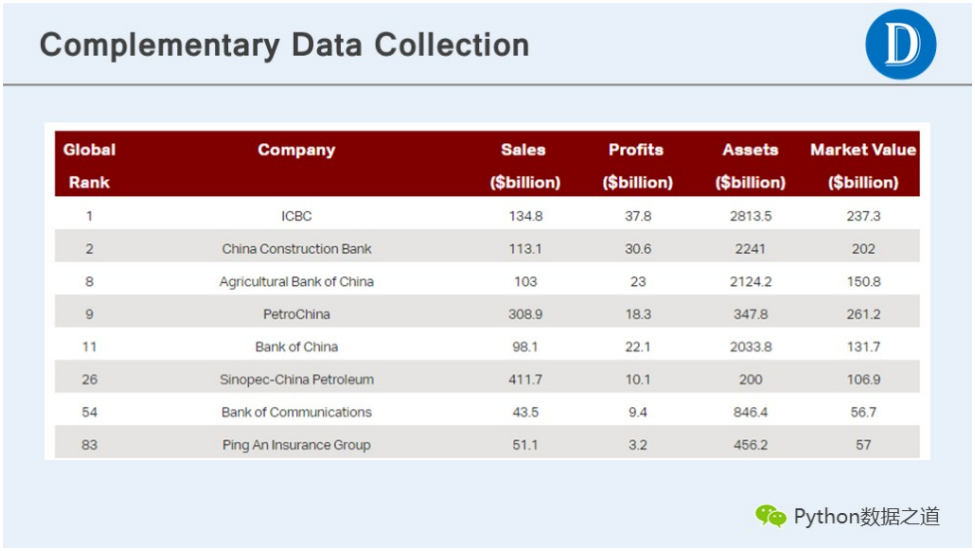
网站地址如下：

- Economy Watch：Forbes Global 2000: China's Largest Companies

先打开网站，来查看数据情况，在这个页面上，我们关心的内容主要有两个部分。

1. 某个国家的数据列表，即数据表格。
2. 获得所有国家的信息列表的网页链接，方便后续从这些网页中爬取相关国家的数据。

这两部分在网站上对应的情况如下图所示：



Global Rank	Company	Sales ($billion)	Profits ($billion)	Assets ($billion)	Market Value ($billion)
1	ICBC	134.8	37.8	2813.5	237.3
2	China Construction Bank	113.1	30.6	2241	202
8	Agricultural Bank of China	103	23	2124.2	150.8
9	PetroChina	308.9	18.3	347.8	261.2
11	Bank of China	98.1	22.1	2033.8	131.7
26	Sinopec-China Petroleum	411.7	10.1	200	106.9
54	Bank of Communications	43.5	9.4	846.4	56.7
83	Ping An Insurance Group	51.1	3.2	456.2	57

本次数据爬取大体分为几步：

1. 起始页面内容的爬取
2. 起始页面上其他国家的网站链接的采集
3. 其他国家的网页数据的 download 与企业列表数据信息采集
4. 将采集的数据保存

涉及到的 python 库主要包括 requests、BeautifulSoup 以及 csv。

下面来描述下数据采集的步骤。

先引入相关 python 库：

```
# _*_ coding: utf-8 _*_

'''
# Code based on Python 3.x
# Author: "LEMON"
# 微信公众号ID: PyDataLab
'''

import requests
from bs4 import BeautifulSoup
import csv
```

模块：页面 Download

网站页面 download 页面是一个通用的模块，在所有国家的数据获取时都要用到。代码如下：

```
def download(url):
    headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
        AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari
        /537.36'}
    response = requests.get(url,headers=headers)
```

```
    # print(response.status_code)
    return response.text
```

模块：企业表格信息

在分析完网页结构之后，首先我们来获取起始页面上的信息，分别包括国家的企业信息数据和其他国家的网页链接信息。

可以分为两个模块来实现这些信息的获取。

国家的企业列表表格上的信息获取的模块代码如下：

```
def get_content_first_page(html, country='China'):
    '''
    获取China的公司列表，且包含表头
    '''
    soup = BeautifulSoup(html, 'lxml')
    body = soup.body
    label_div = body.find('div', {'class': 'content post'})
    tables = label_div.find_all('table')

    # tables一共有2个，第一个才是我们想要的数据
    trs = tables[0].find_all('tr')

    # 获取标题行信息
    row_title = [td.text for td in trs[0].find_all('td')]
    row_title.insert(2, 'Country')

    data_list = []
    data_list.append(row_title)
    for i, tr in enumerate(trs):
        if i == 0:
            continue
        tds = tr.find_all('td')

        # 获取公司排名及列表
        row = [ item.text.strip() for item in tds]
        row.insert(2, country)
        data_list.append(row)
    return data_list
```

模块：其他国家的网页链接

从起始页面上获取其他国家的网页链接信息的模块如下：

```
def get_country_info(html):
    '''
    获取除China外其他国家的网页链接及国家名称
    '''
    soup = BeautifulSoup(html, 'lxml')
    body = soup.body
    label_div = body.find('div', {'class': 'content post'})
```

```
    tables = label_div.find_all('table')

    label_a = tables[1].find_all('a')
    country_names = [item.text for item in label_a]
    page_urls = [ item.get('href') for item in label_a]
    country_info = list(zip(country_names, page_urls))
    return country_info
```

模块：其他国家的企业表格信息

对于其他国家而言，只需要在其对应的页面上获取企业列表信息即可，代码内容跟起始页面上的企业信息获取代码基本是一样的。

有一点差异，就是不需要再获取表格的标题信息了。代码就不作赘述了。

模块：信息保存

由于信息量不大，且结构相对比较简单，保存到 csv 文件即可。

```
def save_data_to_csv_file(data, file_name):
    '''
    保存数据到csv文件中
    '''
    with open(file_name, 'a', errors='ignore', newline='') as f:
        f_csv = csv.writer(f)
        f_csv.writerows(data)
```

编写网页爬取主函数文件

在各个模块编写好后，就可以在主函数里进行调用，从而爬取数据。如下：

```
def get_forbes_global_year_2013():
    url = 'http://www.economywatch.com/companies/forbes-list/china.html'
    html = download(url)
    # print(html)

    data_first_page = get_content_first_page(html)
    # print(data_first_page)
    print('saving data ...', 'China')
    save_data_to_csv_file(data_first_page, 'forbes_economywatch_2013.csv')

    country_info = get_country_info(html)
    # print(country_info)

    for item in country_info:
        country_name = item[0]
        country_url = item[1]

        if country_name == 'China':
            continue

        html = download(country_url)
```

```
22          data_other_country = get_content_other_country(html, country_name)
23          # print(data_other_country)
24          print('saving data ...', country_name)
25          save_data_to_csv_file(data_other_country, 'forbes_economywatch_2013.csv
                ')
26
27
28  if __name__ == '__main__':
29
30      # get data from Forbes Global 2000 in Year 2013
31      get_forbes_global_year_2013()
```

将数据爬取下来后，查看数据是否完整，通过 pandas 查看，发现 Economy Watch 上，2013 年的企业数量只有 **1984 家**，数据也不完整。

需要继续查找其他可靠的数据。

46.4.2 其他数据源收集

后来，在网站找到一个 excel 文件，发现其企业数量是 2000 家，初步认为数据是完整的。

在后续分析过程中，将会使用这个 excel 文件的企业数据信息。

所以说，数据收集的途径很多，关键是数据的质量要好，数据要完整，尽量少的出现异常情况。

至此，福布斯系列文章已经发布了多篇文章，**接下来要进入数据清洗的深水区**，欢迎大家持续关注 ~~

46.5 数据清洗-2007 年数据

福布斯系列之数据清洗（1）- Python 数据分析项目实战

46.5.1 前言

本文作为数据清洗的第一篇，将详细的描述福布斯全球上市企业 2000 强排行榜数据中 2007 年数据的处理过程。

通常，我们都希望自己拿到的数据是比较规范，看起来令人清爽的数据。然而，实际上我们获得的信息或数据，会因为各种各样的原因，存在数据缺失、不准确、不规范等异常情况。需要我们在进行数据分析之前进行预处理。

福布斯全球上市企业 2000 强排行榜数据，从 2007 年到 2017 年，各个年份的数据都存在一些不规范的地方。从本文开始至接下来的几篇文章，将通过福布斯项目来进行数据清洗的项目实战，希望能给大家带来些收获。

本项目运行环境：

- windows7
- jupyter notebook

46.5.2 数据清洗的目的

将不规范的数据进行处理，包括：

1. 替换 NaN 值
2. 将字符串型数字转换为数字类型
3. 将数字后面多余的字母等文字剔除
4. 将公司和国家进行拆分
5. 将国家名称进行统一
6. 按列名将 DataFrame 重新排序

最终达到的效果如下：

数据清洗前

数据类型

```
年份                      int64
排名(Rank)                int64
公司名称(Company)          object
所在国家或地区(Country)      object
所在行业(Industry)         object
销售收入(Sales)            object
利润(Profits)            object
总资产(Assets)            object
市值(Market Vaue)       float64
dtype: object
```

	年份	排名(Rank)	公司名称(Company)	所在国家或地区(Country)	所在行业(Industry)	销售收入(Sales)	利润(Profits)	总资产(Assets)	市值(Market Vaue)
0	2007	1	Citigroup /花旗集团	美国(US)	银行	146.56	21.54	1,884.32	247.42
1	2007	2	Bank of America /美国银行	美国(US)	银行	116.57	21.13	1,459.74	226.61
2	2007	3	HSBC Holdings/汇丰集团	英国(UK)	银行	121.51	16.63	1,860.76	202.29

数据清洗后

数据类型

```
Year               int64
Rank               int64
Company_cn_en     object
Company_en        object
Company_cn        object
Country_cn_en     object
Country_cn        object
Country_en        object
Industry_cn       object
Industry_en       object
Sales            float64
Profits          float64
Assets           float64
Market_value     float64
dtype: object
```

	Year	Rank	Company_cn_en	Company_en	Company_cn	Country_cn_en	Country_cn	Country_en	Industry_cn	Industry_en	Sales	Profits	Assets	Market_value
0	2007	1	Citigroup /花旗集团	Citigroup	花旗集团	美国(US)	美国	US	银行		146.56	21.54	1884.32	247.42
1	2007	2	Bank of America /美国银行	Bank of America	美国银行	美国(US)	美国	US	银行		116.57	21.13	1459.74	226.61
2	2007	3	HSBC Holdings/汇丰集团	HSBC Holdings	汇丰集团	英国(UK)	英国	UK	银行		121.51	16.63	1860.76	202.29

数据清洗后，sales、Profits、Assets 及 Market_value 列的数据均为数字类型，方便后续分析时计算使用。

46.5.3 数据清洗的详细过程

```
import pandas as pd
import numpy as np
```

2007 年的数据，原始数据的单位为十亿美元。

46.5.3.1 先查看数据的类型（dtype）和结构

```
df_2007 = pd.read_csv('./data/data_forbes_2007.csv', encoding='gbk', thousands=
    ',')
print('the shape of DataFrame: ', df_2007.shape)
print(df_2007.dtypes)
df_2007.head(3)

out:
the shape of DataFrame:  (2000, 9)
年份                      int64
排名(Rank)                int64
公司名称(Company)          object
所在国家或地区(Country)      object
所在行业(Industry)         object
销售收入(Sales)           object
利润(Profits)            object
总资产(Assets)            object
市值(Market Vaue)       float64
dtype: object
```

	年份	排名(Rank)	公司名称(Company)	所在国家或地区(Country)	所在行业(Industry)	销售收入(Sales)	利润(Profits)	总资产(Assets)	市值(Market Vaue)
0	2007	1	Citigroup /花旗集团	美国(US)	银行	146.56	21.54	1,884.32	247.42
1	2007	2	Bank of America /美国银行	美国(US)	银行	116.57	21.13	1,459.74	226.61
2	2007	3	HSBC Holdings/汇丰集团	英国(UK)	银行	121.51	16.63	1,860.76	202.29

更新 columns 的命名

```
column_update = ['Year', 'Rank', 'Company_cn_en', 'Country_cn_en',
                 'Industry_cn', 'Sales', 'Profits', 'Assets', 'Market_value']
df_2007.columns = column_update
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
0	2007	1	Citigroup /花旗集团	美国(US)	银行	146.56	21.54	1,884.32	247.42
1	2007	2	Bank of America /美国银行	美国(US)	银行	116.57	21.13	1,459.74	226.61
2	2007	3	HSBC Holdings/汇丰集团	英国(UK)	银行	121.51	16.63	1,860.76	202.29

46.5.3.2 将字符串转换为数字类型

通过前面的分析可看出，只有“Market_value”是数字类型，**找出’Sales’,’Profits’ 及’Assets’ 中非数字的内容**

```
df_2007[df_2007['Sales'].str.contains('.*[A-Za-z]', regex=True)]
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
117	2007	118	Repsol-YPF /瑞普索	西班牙(SP)	炼油	64.20 E	4.12	58.43	38.75
616	2007	617	Inpex Holdings	日本(JA)	炼油	6.49 E	1.02 E	10.77 E	19.65
880	2007	881	Asahi Breweries/朝日啤酒	日本(JA)	食品、饮料和烟草	7.97 E	0.38	10.66	7.71

用 replace() 方法替换“Sales”列中含有字母的内容

```
df_2007['Sales'] = df_2007['Sales'].replace('([A-Za-z])', '', regex=True)
```

查看替换后的结果

```
df_2007.loc[[117,616,880], :]
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
117	2007	118	Repsol-YPF /瑞普索	西班牙(SP)	炼油	64.20	4.12	58.43	38.75
616	2007	617	Inpex Holdings	日本(JA)	炼油	6.49	1.02 E	10.77 E	19.65
880	2007	881	Asahi Breweries/朝日啤酒	日本(JA)	食品、饮料和烟草	7.97	0.38	10.66	7.71

查看“Assets”列中非数字的内容

```
df_2007[df_2007['Assets'].str.contains('.*[A-Za-z]', regex=True)]
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
616	2007	617	Inpex Holdings	日本(JA)	炼油	6.49	1.02 E	10.77 E	19.65

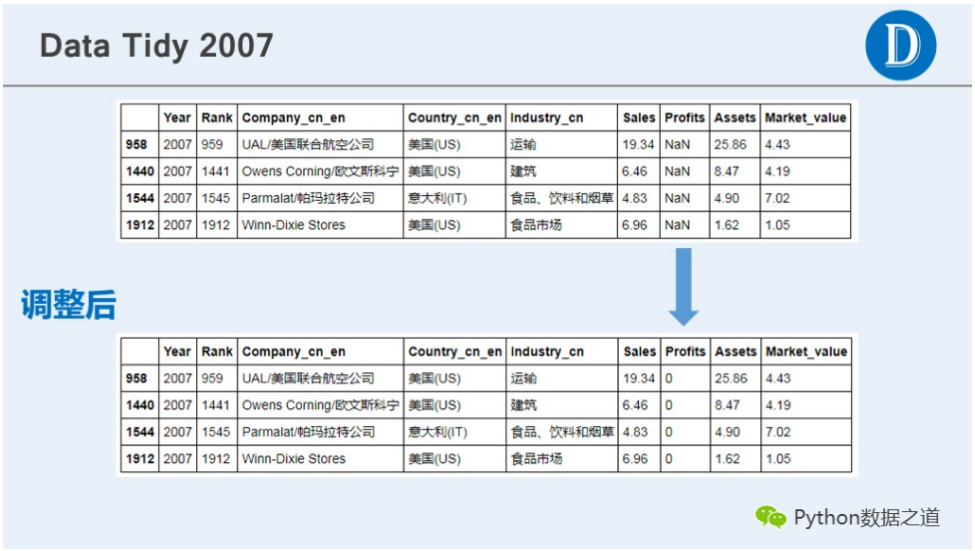
替换非数字的内容，以及替换千分位间隔符号

```
# 将数字后面的字母进行替换
df_2007['Assets'] = df_2007['Assets'].replace('([A-Za-z])', '', regex=True)

# 千分位数字的逗号被识别为string了，需要替换
df_2007['Assets'] = df_2007['Assets'].replace(',', '', regex=True)
df_2007.loc[616, :]

out:
Year                          2007
Rank                           617
Company_cn_en    Inpex Holdings
Country_cn_en             日本(JA)
Industry_cn                   炼油
Sales                         6.49
Profits                     1.02 E
Assets                       10.77
Market_value                 19.65
```

```
18 Name: 616, dtype: object
```

发现“Profits”中有 NaN 值，需要先进行替换

```
df_2007[pd.isnull(df_2007['Profits'])]

#  将NaN值替换为0
df_2007['Profits'].fillna(0, inplace=True)
df_2007.loc[[958,1440,1544,1912], :]
```



	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
958	2007	959	UAL/美国联合航空公司	美国(US)	运输	19.34	NaN	25.86	4.43
1440	2007	1441	Owens Corning/欧文斯科宁	美国(US)	建筑	6.46	NaN	8.47	4.19
1544	2007	1545	Parmalat/帕玛拉特公司	意大利(IT)	食品、饮料和烟草	4.83	NaN	4.90	7.02
1912	2007	1912	Winn-Dixie Stores	美国(US)	食品市场	6.96	NaN	1.62	1.05

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
958	2007	959	UAL/美国联合航空公司	美国(US)	运输	19.34	0	25.86	4.43
1440	2007	1441	Owens Corning/欧文斯科宁	美国(US)	建筑	6.46	0	8.47	4.19
1544	2007	1545	Parmalat/帕玛拉特公司	意大利(IT)	食品、饮料和烟草	4.83	0	4.90	7.02
1912	2007	1912	Winn-Dixie Stores	美国(US)	食品市场	6.96	0	1.62	1.05

将“Profits”列中非数字的内容进行替换，并查看替换后的结果

```
df_2007['Profits'] = df_2007['Profits'].replace('([A-Za-z])', '', regex=True)
df_2007.loc[[117,616,880], :]
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value
117	2007	118	Repsol-YPF /瑞普索	西班牙(SP)	炼油	64.20	4.12	58.43	38.75
616	2007	617	Inpex Holdings	日本(JA)	炼油	6.49	1.02	10.77	19.65
880	2007	881	Asahi Breweries/朝日啤酒	日本(JA)	食品、饮料和烟草	7.97	0.38	10.66	7.71

将 string 类型的数字转换为数据类型，这里使用 `pd.to_numeric()` 方法

```
df_2007['Sales'] = pd.to_numeric(df_2007['Sales'])
df_2007['Profits'] = pd.to_numeric(df_2007['Profits'])
df_2007['Assets'] = pd.to_numeric(df_2007['Assets'])
df_2007.dtypes

out:
Year                int64
Rank                int64
Company_cn_en      object
Country_cn_en      object
Industry_cn        object
```

```
Sales            float64
Profits          float64
Assets           float64
Market_value     float64
dtype: object
```

46.5.3.3 拆分“Company_cn_en”列

新生成两列，分别为公司英文名称和中文名称

```
df_2007['Company_en'],df_2007['Company_cn'] = df_2007['Company_cn_en'].str.
    split('/', 1).str
print(df_2007['Company_en'][:5])
print(df_2007['Company_cn'] [-5:])
df_2007.tail(3)

out:
0           Citigroup
1     Bank of America
2        HSBC Holdings
3    General Electric
4      JPMorgan Chase
Name: Company_en, dtype: object
1995    NaN
1996    NaN
1997    NaN
1998    NaN
1999    NaN
Name: Company_cn, dtype: object
```

	Year	Rank	Company_cn_en	Country_cn_en	Industry_cn	Sales	Profits	Assets	Market_value	Company_en	Company_cn
1997	2007	1998	CBOT Holdings	美国(US)	综合金融	0.64	0.17	0.81	8.54	CBOT Holdings	NaN
1998	2007	1998	Singapore Petroleum	新加坡(SI)	炼油	5.59	0.19	2.05	1.50	Singapore Petroleum	NaN
1999	2007	2000	DVB Bank	德国(GE)	银行	0.77	0.06	12.74	1.26	DVB Bank	NaN

46.5.3.4 拆分“Country_cn_en”列

新生成两列，分别为国家中文名称和英文名称

```
df_2007['Country_cn'],df_2007['Country_en'] = df_2007['Country_cn_en'].str.
    split('(', 1).str
print(df_2007['Country_cn'][:5])
print(df_2007['Country_en'][-5:])

out:
0    美 国
1    美 国
2    英 国
3    美 国
```

```
4     美国
Name: Country_cn, dtype: object
1995    US)
1996    US)
1997    US)
1998    SI)
1999    GE)
Name: Country_en, dtype: object
```

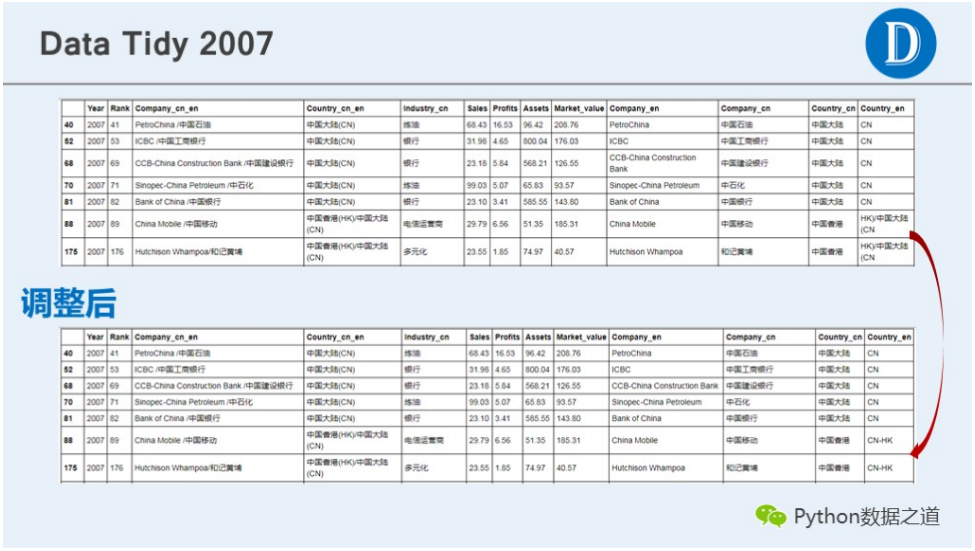
由于国家的英文名称中，最后有半个括号，需要去除，用 Series.str.slice() 方法。参数表示选取从开始到倒数第二个，即不要括号“)”

```
df_2007['Country_en'] = df_2007['Country_en'].str.slice(0,-1)
df_2007.head(3)
```

考虑的中国的企业有区分为中国大陆，中国香港，中国台湾。对应的国家英文名称也需要修改下。

- 中国大陆：CN；
- 中国香港：CN-HK；
- 中国台湾：CN-TA。

```
# 查找含“中国”的数据
df_2007[df_2007['Country_cn'].str.contains('中国',regex=True)]

# 替换并查看结果
df_2007['Country_en'] = df_2007['Country_en'].replace(['HK.*','TA'],['CN-HK', '
    CN-TA'],regex=True)
df_2007[df_2007['Country_en'].str.contains('CN',regex=True)]
```



考虑到其他年份，公司所在行业有用英文名称展示的，这里添加一列英文的行业名称，但内容是空白。

```
df_2007['Industry_en'] = ''
df_2007.tail(3)
```

46.5.3.5 将列名进行重新排序

```
columns_sort = ['Year', 'Rank', 'Company_cn_en','Company_en',
                'Company_cn', 'Country_cn_en', 'Country_cn',
                'Country_en', 'Industry_cn', 'Industry_en',
                'Sales', 'Profits', 'Assets', 'Market_value']

# 按指定list重新将columns进行排序
df_2007 = df_2007.reindex(columns=columns_sort)
print(df_2007.shape)
print(df_2007.dtypes)
df_2007.head(3)

out:
(2000, 14)
Year               int64
Rank               int64
Company_cn_en     object
Company_en        object
Company_cn        object
Country_cn_en     object
Country_cn        object
Country_en        object
Industry_cn       object
Industry_en       object
Sales            float64
Profits          float64
Assets           float64
Market_value     float64
dtype: object
```

	Year	Rank	Company_cn_en	Company_en	Company_cn	Country_cn_en	Country_cn	Country_en	Industry_cn	Industry_en	Sales	Profits	Assets	Market_value
0	2007	1	Citigroup /花旗集团	Citigroup	花旗集团	美国(US)	美国	US	银行		146.56	21.54	1884.32	247.42
1	2007	2	Bank of America /美国银行	Bank of America	美国银行	美国(US)	美国	US	银行		116.57	21.13	1459.74	226.61
2	2007	3	HSBC Holdings/汇丰集团	HSBC Holdings	汇丰集团	英国(UK)	英国	UK	银行		121.51	16.63	1860.76	202.29

46.5.4 小结

至此，将 2007 年的原始数据进行了初步处理，当然，在后续的分析过程中，可能还会有少量的处理。经过数据清洗后，获得的数据信息相对来说比较规范，数据的质量较好，可以用来后续分析使用或者进一步清洗。

数据清洗是数据分析、数据挖掘中一项基本且非常重要的技能，通过在整个数据分析中所用时间的比例会比较高。

当然，本次数据清洗的过程，还有可以优化的地方，各位同学可以自行研究下。

46.6 数据清洗-2008-2010 年

福布斯系列之数据清洗（2）- Python 数据分析项目实战

46.6.1 前言

本文作为 **数据清洗的第二篇**，内容包含福布斯全球上市企业 2000 强排行榜数据中 **2008-2010 年**数据的初步处理过程。

福布斯全球上市企业 2000 强排行榜数据，从 2007 年到 2017 年，各个年份的数据都存在一些不规范的地方。

本文以 2008 年为例，描述 2008 年至 2010 年的数据清洗过程。

本项目运行环境：

- windows7
- jupyter notebook

46.6.2 数据清洗的目的

将不规范的数据进行处理，包括：

1. 替换 NaN 值
2. 将字符串型数字转换为数字类型
3. 将数字后面多余的字母等文字剔除
4. 将公司和国家进行拆分
5. 按列名将 DataFrame 重新排序

最终达到的效果如下：

数据清洗前

数据类型

```
the shape of DataFrame:  (2000, 10)
年份                            int64
Rank                           int64
公司名称（英文）                      object
公司名称（中文）                      object
Country/area（国家或地区）          object
Industry（行业）                  object
Sales （销售额）($bil十亿美元)        object
Profits （利润）($bil)           object
Assets 资产($bil)              object
Market Value 市值($bil)       float64
dtype: object
```

	年份	Rank	公司名称（英文）	公司名称（中文）	Country/area（国家或地区）	Industry（行业）	Sales （销售额）($bil十亿美元)	Profits （利润）($bil)	Assets 资产($bil)	Market Value 市值($bil)
0	2008	1	HSBC Holdings	汇丰集团	United Kingdom	Banking	146.5	19.13	2,348.96	180.81
1	2008	2	General Electric	通用电气公司	United States	Conglomerates	172.74	22.21	795.34	330.93
2	2008	3	Bank of America	美国银行	United States	Banking	119.19	14.98	1,715.75	176.53
3	2008	4	JPMorgan Chase	摩根大通公司	United States	Banking	116.35	15.37	1,562.15	136.88
4	2008	5	ExxonMobil	埃克森美孚公司	United States	Oil & Gas Operations	358.6	40.61	242.08	465.51

数据清洗后

数据类型

```
Year              int64
Rank              int64
Company_cn_en    object
Company_en       object
Company_cn       object
Country_cn_en    object
Country_cn       object
Country_en       object
Industry_cn      object
Industry_en      object
Sales           float64
Profits         float64
Assets          float64
Market_value    float64
```

	Year	Rank	Company_cn_en	Company_en	Company_cn	Country_cn_en	Country_cn	Country_en	Industry_cn	Industry_en	Sales	Profits	Assets	Market_value
0	2008	1		HSBC Holdings	汇丰集团			United Kingdom		Banking	146.50	19.13	2348.98	180.81
1	2008	2		General Electric	通用电气公司			United States		Conglomerates	172.74	22.21	795.34	330.93
2	2008	3		Bank of America	美国银行			United States		Banking	119.19	14.98	1715.75	176.53
3	2008	4		JPMorgan Chase	摩根大通公司			United States		Banking	116.35	15.37	1562.15	136.88
4	2008	5		ExxonMobil	埃克森美孚公司			United States		Oil & Gas Operations	358.60	40.61	242.08	465.51

数据清洗后，sales、Profits、Assets 及 Market_value 列的数据均为数字类型，方便后续分析时计算使用。

46.6.3 2008 年数据清洗的详细过程

导入相关 python 库

```
import pandas as pd
import numpy as np
```

从 csv 文件中读取数据

```
df_2008 = pd.read_csv('./data/data_forbes_2008.csv', encoding='gbk', thousands=
    ',')
print('the shape of DataFrame: ', df_2008.shape)
print(df_2008.dtypes)
df_2008.head()
```

更新列名

```
df_2008.columns = ['Year', 'Rank', 'Company_en', 'Company_cn','Country_en', '
    Industry_en', 'Sales', 'Profits', 'Assets', 'Market_value']
df_2008.head()
```

添加空白列，使之与其他年份的格式保持一致

```
df_2008['Company_cn_en'], df_2008['Country_cn_en'], df_2008['Country_cn'],
    df_2008['Industry_cn'] = ['','','','']
df_2008.head()
```

将字符串转换为数字类型

在前文 2007 年的数据清洗过程中，单独针对’Sales’,’Profits’ 及’Assets’ 进行处理。

本文中，**对处理过程进行了一些优化**，通过 for 循环，一次性处理不同的字段，大大减少了代码量，以及程序的繁冗程度，使整个过程变得更简洁。

处理过程如下：

```
col_digit = ['Sales', 'Profits', 'Assets', 'Market_value']

for col in col_digit:
    # 将数字后面的字母进行替换
    df_2008[col] = df_2008[col].replace('([A-Za-z])', '', regex=True)

    # 千分位数字的逗号被识别为string了，需要替换
    df_2008[col] = df_2008[col].replace(',', '', regex=True)

    #将数字型字符串转换为可进行计算的数据类型
    df_2008[col] = pd.to_numeric(df_2008[col])
```

按指定 list 重新将 columns 进行排序

```
# 按指定list重新将columns进行排序
columns_sort = ['Year', 'Rank', 'Company_cn_en','Company_en',
                'Company_cn', 'Country_cn_en', 'Country_cn',
                'Country_en', 'Industry_cn', 'Industry_en',
                'Sales', 'Profits', 'Assets', 'Market_value']

df_2008 = df_2008.reindex(columns=columns_sort)
print(df_2008.shape)
print(df_2008.dtypes)
df_2008.head()
```

46.6.4 后续

2009 年和 2010 年的数据清洗过程跟 2008 年基本一致，处理过程就不在本文中描述了（**提供源代码供参考**）。

细心的同学可能会发现，本次没有提到针对国家数据的规整，因为国家数据的规整将统一到最后进行处理，敬请后续关注。

46.7 数据清洗-2011-2015 年

福布斯系列之数据清洗（3）- Python 数据分析项目实战

46.7.1 前言

本文作为 **数据清洗的第三篇**，内容包含福布斯全球上市企业 2000 强排行榜数据中 **2011-2015 年**数据的初步处理过程。

福布斯全球上市企业 2000 强排行榜数据，从 2007 年到 2017 年，各个年份的数据都存在一些不规范的地方。

本文以 2013 年为例，描述 2011 年至 2015 年的数据清洗过程。

本项目运行环境：

- windows7
- jupyter notebook
- Python 3.5

46.7.2 数据清洗的目的

将不规范的数据进行处理，包括：

1. 检查数据完整性
2. 去除重复内容
3. 替换 NaN 值
4. 将字符串型数字转换为数字类型
5. 将数字后面多余的字母等文字剔除
6. 将公司和国家进行拆分
7. 按列名将 DataFrame 重新排序

最终达到的效果如下：

数据清洗后

数据类型

```
Year              int64
Rank              int64
Company_cn_en    object
Company_en       object
Company_cn       object
Country_cn_en    object
Country_cn       object
Country_en       object
Industry_cn      object
Industry_en      object
Sales           float64
Profits         float64
Assets          float64
Market_value    float64
dtype: object
```

	Year	Rank	Company_cn_en	Company_en	Company_cn	Country_cn_en	Country_cn	Country_en	Industry_cn	Industry_en	Sales	Profits	Assets	Market_value
0	2013	1	中国工商银行/ICBC	ICBC	中国工商银行		中国大陆				134.8	37.8	2813.5	237.3
1	2013	2	中国建设银行/China Construction Bank	China Construction Bank	中国建设银行		中国大陆				113.1	30.6	2241.0	202.0
2	2013	3	摩根大通/JPMorgan Chase	JPMorgan Chase	摩根大通		美国				108.2	21.3	2359.1	191.4
3	2013	4	通用电气/General Electric	General Electric	通用电气		美国				147.4	13.6	685.3	243.7
4	2013	5	埃克森美孚/Exxon Mobil	Exxon Mobil	埃克森美孚		美国				420.7	44.9	333.8	400.4

46.7.3 2013 年数据清洗的详细过程

导入相关 python 库

```
import pandas as pd
import numpy as np
```

2013 年的数据，由于初始爬取的数据不完整，后续又查找了另外 2 个数据源，所以共有 3 个数据源，下面分别介绍。

46.7.3.1 数据源 1

从 csv 文件中读取数据

```
df_2013 = pd.read_csv('./data/data_forbes_2013.csv', encoding='gbk',header=None
    )
print('the shape of DataFrame: ', df_2013.shape)
print(df_2013.dtypes)
df_2013.head()

out:
the shape of DataFrame:  (1991, 9)
```

数据源 1 中，2013 年的数据只有 1991 条记录，数据可能有缺失，待进一步核实。

46.7.3.2 数据源 2

2013 年的数据在网上继续寻找，发现 Economy Watch 网站有相关数据，于是进行数据爬取。前文已描述了从该网站获取数据的过程。

从 csv 文件中读取数据

```
df_2013_economy = pd.read_csv('./data/data_forbes_2013_economywatch.csv',
    encoding='gbk')
print('the shape of DataFrame: ', df_2013_economy.shape)
print(df_2013_economy.dtypes)
df_2013_economy.head()

out:
the shape of DataFrame:  (1984, 7)
```

发现数据只有 1984 条记录，也缺少相关记录，需要继续寻找其他记录。

46.7.3.3 数据源 3

后来，找到一个 excel 文件，发现其数据记录是完整的，于是用该文件。

首先，从 csv 文件中读取数据

```
df_2013_all = pd.read_excel('./data/data_forbes_2013_all.xlsx')
print('the shape of DataFrame: ', df_2013_all.shape)
print(df_2013_all.dtypes)
df_2013_all.head()

out:
the shape of DataFrame:  (2000, 7)
排名              int64
公司名           object
国家（地区）        object
销售额（亿美元）       int64
利润（亿美元）        int64
资产（亿美元）        int64
市值（亿美元）        int64
dtype: object
```

这里在前面两篇数据清洗文章的基础上，将数据清洗的过程整合到一起了。处理过程如下：

```
# 更新列名
df_2013_all.columns = ['Rank', 'Company_cn_en',
                       'Country_cn', 'Sales', 'Profits', 'Assets', 'Market_value']

# 拆分"Company_cn_en"列，新生成两列，分别为公司英文名称和中文名称
df_2013_all['Company_cn'],df_2013_all['Company_en'] = df_2013_all['
    Company_cn_en'].str.split('/', 1).str
# print(df_2013_all['Company_cn'][:5])
# print(df_2013_all['Company_en'] [-5:])

# 将数据单位转换成十亿美元
df_2013_all[['Sales','Profits','Assets','Market_value']] =df_2013_all[['Sales',
    'Profits','Assets','Market_value']].apply(lambda x: x/10)

# 添加年份2013
df_2013_all['Year'] = 2013

# 添加空白列
df_2013_all['Country_en'],df_2013_all['Country_cn_en'], df_2013_all['
    Industry_cn'], df_2013_all['Industry_en'] = ['','','','']

df_2013_all['Rank'] = pd.to_numeric(df_2013_all['Rank'])

# 按指定list重新将columns进行排序
df_2013_all = df_2013_all.reindex(columns=columns_sort)
print(df_2013_all.shape)
print(df_2013_all.dtypes)
df_2013_all.head()
```

请注意，2013 年的数据，sales、Profits、Assets 及 Market_value 列的本身就是为数字类型，不需要进一步处理。

46.7.4 2015 年的个别情况

在前文中提到，2015 年的企业数量有重复的，因此，在数据处理过程中，需要剔除重复的内容。

去重的代码如下：

```
# 数据有2020行，有重复行，需要去除重复行
# inplace=True，使去重生效
df_2015.drop_duplicates('Company_cn_en', inplace=True)
```

查看是否还有重复的内容：

```
# 查看'Company_cn_en'是否还有重复行
df_2015[df_2015['Company_cn_en'].duplicated()]
```

通过查看结果可知，已无重复内容。

2015 年的其他处理过程与 2013 年类似，就不做进一步描述了。

46.7.5 小结

2011 年至 2015 年的数据，相互之间的相似度较高，数据清洗与处理过程和 2013 年的基本类似，具体过程就不再详细描述了（**提供源代码供参考**）。

2011 年-2015 年的数据，**Sales、Profits、Assets 及 Market_value 列的本身就是数字类型**，可以在后续分析时计算使用，所以不需要像 2008-2010 年的数据那样处理。

细心的同学可能会发现，本次依然没有提到针对国家数据的规整，因为国家数据的规整将统一到最后进行处理，敬请后续关注。

46.8 数据清洗-2016 年

Pandas 数据处理实战：福布斯全球上市企业排行榜数据整理

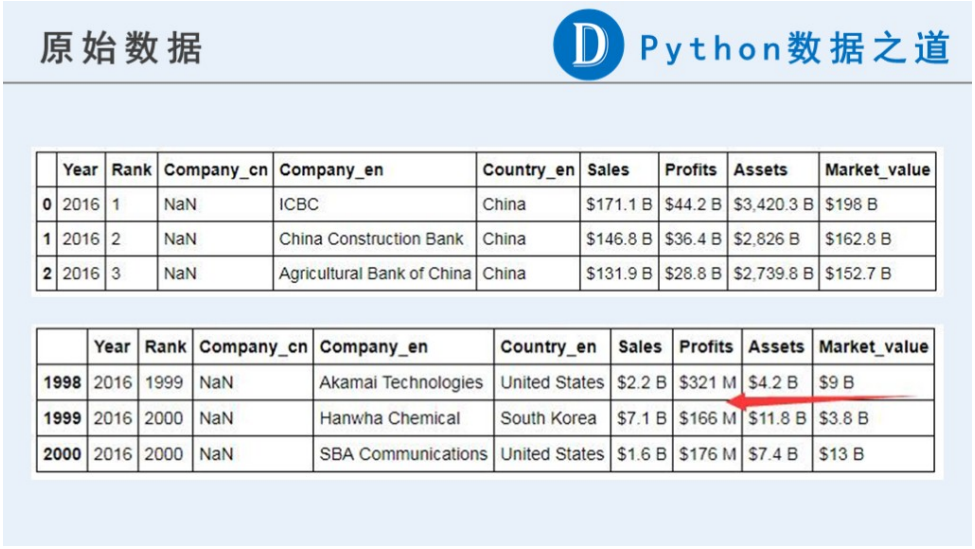
本文通过实例操作来介绍用 pandas 进行数据整理。

先说下我的运行环境，如下：

- windows 7，64 位
- python 3.5
- pandas 0.19.2 版本

在拿到原始数据后，我们先来看看数据的情况，并思考下我们需要什么样的数据结果。

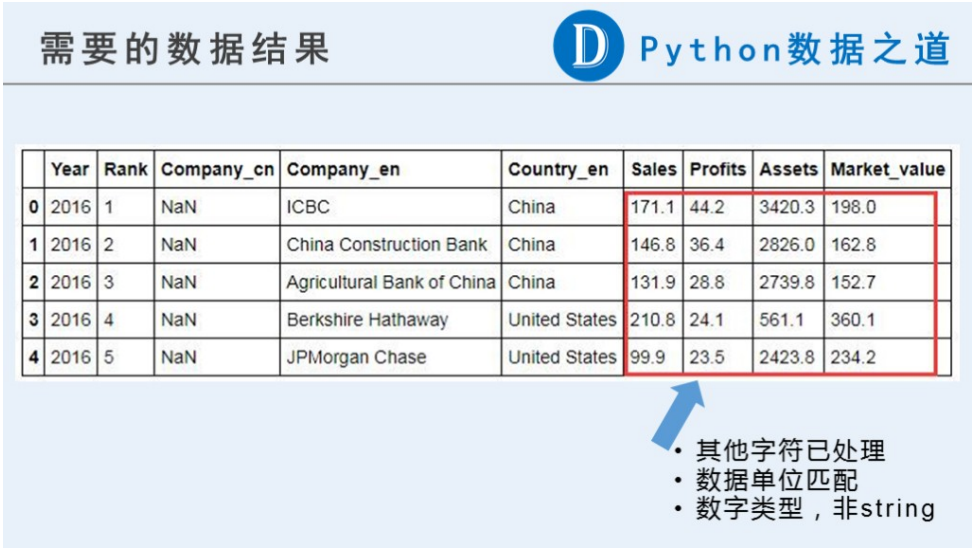
下面是原始数据：



	Year	Rank	Company_cn	Company_en	Country_en	Sales	Profits	Assets	Market_value
0	2016	1	NaN	ICBC	China	$171.1 B	$44.2 B	$3,420.3 B	$198 B
1	2016	2	NaN	China Construction Bank	China	$146.8 B	$36.4 B	$2,826 B	$162.8 B
2	2016	3	NaN	Agricultural Bank of China	China	$131.9 B	$28.8 B	$2,739.8 B	$152.7 B

	Year	Rank	Company_cn	Company_en	Country_en	Sales	Profits	Assets	Market_value
1998	2016	1999	NaN	Akamai Technologies	United States	$2.2 B	$321 M	$4.2 B	$9 B
1999	2016	2000	NaN	Hanwha Chemical	South Korea	$7.1 B	$166 M	$11.8 B	$3.8 B
2000	2016	2000	NaN	SBA Communications	United States	$1.6 B	$176 M	$7.4 B	$13 B

在本文中，我们需要以下的初步结果，以供以后继续使用。



	Year	Rank	Company_cn	Company_en	Country_en	Sales	Profits	Assets	Market_value
0	2016	1	NaN	ICBC	China	171.1	44.2	3420.3	198.0
1	2016	2	NaN	China Construction Bank	China	146.8	36.4	2826.0	162.8
2	2016	3	NaN	Agricultural Bank of China	China	131.9	28.8	2739.8	152.7
3	2016	4	NaN	Berkshire Hathaway	United States	210.8	24.1	561.1	360.1
4	2016	5	NaN	JPMorgan Chase	United States	99.9	23.5	2423.8	234.2

可以看到，原始数据中，跟企业相关的数据中（“Sales”，“Profits”，“Assets”，“Market_value”），目前都是不是可以用来计算的数字类型。

原始内容中包含货币符号”$“，“-”，纯字母组成的字符串以及其他一些我们认为异常的信息。更重要的是，这些数据的单位并不一致。分别有以“B”（Billion，十亿）和“M”（Million，百万）表示的。在后续计算之前需要进行单位统一。

46.8.1 处理方法 Method-1

首先想到的处理思路就是将数据信息分别按十亿（“B”）和百万 (“M”) 进行拆分，分别进行处理，最后在合并到一起。过程如下所示。

- 加载数据，并添加列的名称

```
import pandas as pd

df_2016 = pd.read_csv('data_2016.csv', encoding='gbk',header=None)

# 更新列名
df_2016.columns = ['Year', 'Rank', 'Company_cn','Company_en',
                   'Country_en', 'Sales', 'Profits', 'Assets', 'Market_value']

print('the shape of DataFrame: ', df_2016.shape)
print(df_2016.dtypes)
df_2016.head(3)
```

- 获取单位为十亿（“B”）的数据

```
# 数据单位为 B的数据 (Billion, 十亿)
df_2016_b = df_2016[df_2016['Sales'].str.endswith('B')]
print(df_2016_b.shape)
df_2016_b
```

- 获取单位为百万 (“M”) 的数据

```
# 数据单位为 M的数据 (Million, 百万)
df_2016_m = df_2016[df_2016['Sales'].str.endswith('M')]
print(df_2016_m.shape)
df_2016_m
```

这种方法理解起来比较简单，但操作起来会比较繁琐，尤其是如果有很多列数据需要处理的话，会花费很多时间。

进一步的处理，我这里就不描述了。当然，各位可以试试这个方法。

下面介绍稍微简单一点的方法。

46.8.2 处理方法 Method-2

46.8.2.1 加载数据

第一步还是加载数据，跟 Method-1 是一样的。

下面来处理’Sales’ 列

46.8.2.2 替换相关的异常字符

首先是替换相关的异常字符，包括美元的货币符号’$“，纯字母的字符串’undefined”，以及’B’。这里，我们想统一把数据的单位整理成十亿，所以’B’ 可以直接进行替换。而’M’ 需要更多的处理步骤。

46.8.2.3 处理’M’ 相关的数据

处理含有百万“M”为单位的数据，即以“M”结尾的数据，思路如下：

（1）设定查找条件 mask；

（2）替换字符串“M”为空值

（3）用 pd.to_numeric() 转换为数字

（4）除以 1000，转换为十亿美元，与其他行的数据一致

上面两个步骤相关的代码如下：

```
# 替换美元符号
df_2016['Sales'] = df_2016['Sales'].str.replace('$','')

# # 查看异常值，均为字母（“undefined”）
# df_2016[df_2016['Sales'].str.isalpha()]

# 替换异常值“undefined”为空白
# df_2016['Sales'] = df_2016['Sales'].str.replace('undefined','')
df_2016['Sales'] = df_2016['Sales'].str.replace('^[A-Za-z]+$','')

# 替换符号十亿美元“B”为空白，数字本身代表的就是十亿美元为单位
df_2016['Sales'] = df_2016['Sales'].str.replace('B','')


# 处理含有百万“M”为单位的数据，即以“M”结尾的数据
# 思路：
# （1）设定查找条件mask；
# （2）替换字符串“M”为空值
# （3）用pd.to_numeric()转换为数字
# （4）除以1000，转换为十亿美元，与其他行的数据一致
mask = df_2016['Sales'].str.endswith('M')
df_2016.loc[mask, 'Sales'] = pd.to_numeric(df_2016.loc[mask, 'Sales'].str.
    replace('M', ''))/1000

df_2016['Sales'] = pd.to_numeric(df_2016['Sales'])
print('the shape of DataFrame: ', df_2016.shape)
print(df_2016.dtypes)
df_2016.head(3)
```

用同样类似的方法处理其他列

可以看到，这个方法比第一种方法还是要方便很多。当然，这个方法针对 DataFrame 的每列数据都要进行相关的操作，如果列数多了，也还是比较繁琐的。

有没有更方便一点的方法呢。答案是有的。

46.8.3 处理方法 Method-3

在 Method-2 的基础上，将处理方法写成更通用的数据处理函数，根据数据的结构，拓展更多的适用性，则可以比较方便的处理相关数据。

46.8.3.1 加载数据

第一步还是加载数据，跟 Method-1 是一样的。

46.8.3.2 编写数据处理的自定义函数

参考 Method-2 的处理过程，编写数据处理的自定义函数'pro_col'，并在 Method-2 的基础上拓展其他替换功能，使之适用于这四列数据（“Sales”，“Profits”，“Assets”，“Market_value”）。

函数编写的代码如下：

```
def pro_col(df, col):
    # 替换相关字符串，如有更多的替换情形，可以自行添加
    df[col] = df[col].str.replace('$','')
    df[col] = df[col].str.replace('^[A-Za-z]+$','')
    df[col] = df[col].str.replace('B','')

    # 注意这里是'-$'，即以'-'结尾，而不是'-'，因为有负数
    df[col] = df[col].str.replace('-$','')
    df[col] = df[col].str.replace(',','')

    # 处理含有百万“M”为单位的数据，即以“M”结尾的数据
    # 思路：
    # （1）设定查找条件mask；
    # （2）替换字符串“M”为空值
    # （3）用pd.to_numeric()转换为数字
    # （4）除以1000，转换为十亿美元，与其他行的数据一致
    mask = df[col].str.endswith('M')
    df.loc[mask, col] = pd.to_numeric(df.loc[mask, col].str.replace('M',''))
        /1000

    # 将字符型的数字转换为数字类型
    df[col] = pd.to_numeric(df[col])
    return df
```

46.8.3.3 将自定义函数进行应用

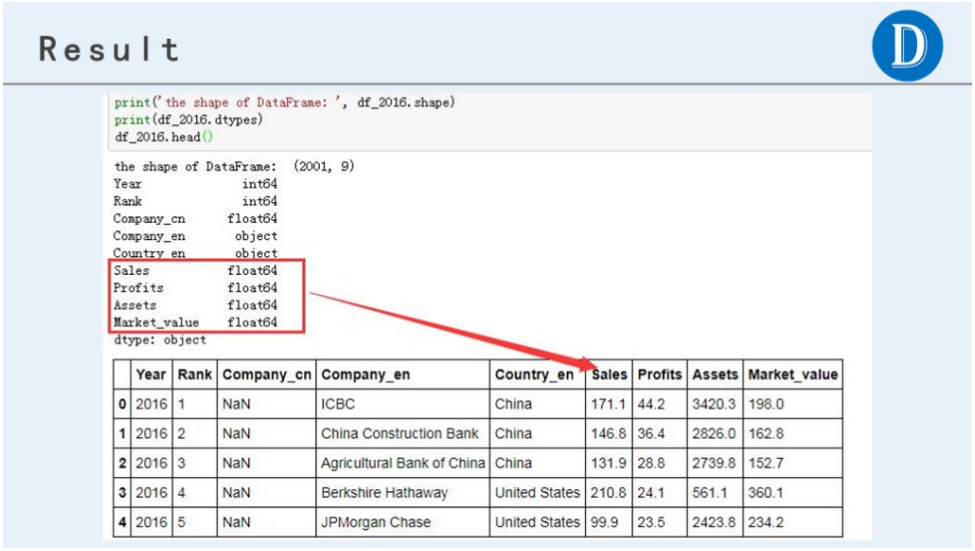
针对 DataFrame 的每列，应用该自定义函数，进行数据处理，得到需要的结果。

```
pro_col(df_2016, 'Sales')
pro_col(df_2016, 'Profits')
pro_col(df_2016, 'Assets')
pro_col(df_2016, 'Market_value')

print('the shape of DataFrame: ', df_2016.shape)
print(df_2016.dtypes)
df_2016.head()
```

当然，如果 DataFrame 的列数特别多，可以用 for 循环，这样代码更简洁。代码如下：

```
cols = ['Sales', 'Profits', 'Assets', 'Market_value']
for col in cols:
    pro_col(df_2016, col)

print('the shape of DataFrame: ', df_2016.shape)
print(df_2016.dtypes)
df_2016.head()
```

最终处理后，获得的数据结果如下：



	Year	Rank	Company_cn	Company_en	Country_en	Sales	Profits	Assets	Market_value
0	2016	1	NaN	ICBC	China	171.1	44.2	3420.3	198.0
1	2016	2	NaN	China Construction Bank	China	146.8	36.4	2826.0	162.8
2	2016	3	NaN	Agricultural Bank of China	China	131.9	28.8	2739.8	152.7
3	2016	4	NaN	Berkshire Hathaway	United States	210.8	24.1	561.1	360.1
4	2016	5	NaN	JPMorgan Chase	United States	99.9	23.5	2423.8	234.2

46.9 数据清洗-2017 年

福布斯系列之数据清洗（5）- Python 数据分析项目实战

46.9.1 前言

本文作为 **数据清洗的第五篇**，内容包含：

（1）**2016-2017 年**数据的初步处理过程。

（2）将 2007-2017 年数据合并，并统一国家名称

本项目运行环境：

- windows7
- jupyter notebook
- Python 3.5

46.9.2 2016-2017 年数据处理

2016 年数据的清洗，前文已做详细描述。

2017 年的数据信息，跟 2016 年基本一致。具体过程就不描述了。代码如下：

```
df_2017 = pd.read_csv('./data/data_forbes_2017.csv', encoding='gbk')

# 更新列名
df_2017.columns = ['Year', 'Rank', 'Company_cn','Company_en',
                   'Country_en', 'Sales', 'Profits', 'Assets', 'Market_value']
print('the shape of DataFrame: ', df_2017.shape)
df_2017.head()

cols = ['Sales', 'Profits', 'Assets', 'Market_value']
for col in cols:
    pro_col(df_2017, col)
print('the shape of DataFrame: ', df_2017.shape)
print(df_2017.dtypes)
df_2017.head()

# 添加空白列
df_2017['Company_cn_en'],df_2017['Country_cn_en'], df_2017['Country_cn'],
    df_2017['Industry_cn'], df_2017['Industry_en'] = ['','','','','']

# 按指定list重新将columns进行排序
columns_sort = ['Year', 'Rank', 'Company_cn_en','Company_en',
                'Company_cn', 'Country_cn_en', 'Country_cn',
                'Country_en', 'Industry_cn', 'Industry_en',
                'Sales', 'Profits', 'Assets', 'Market_value']
df_2017 = df_2017.reindex(columns=columns_sort)
print(df_2017.shape)
df_2017.head()
```

其中，函数`pro_col()`来自 2016 年处理过程的函数。

46.9.3 2007-2017 年数据的合并

用 pandas 的`concat()`方法，将 2007 年至 2017 年的数据进行合并，统一到一个 Dataframe 中。

当然，也可以保存到 csv 文件中，以便后续分析时使用。

```
df_concat = pd.concat([df_2007, df_2008, df_2009,
                       df_2010, df_2011, df_2012,
                       df_2013_all, df_2014,
                       df_2015, df_2016, df_2017], ignore_index=True)
# 保存数据
# df_concat.to_csv('data_forbes_concat.csv')
df_concat.head()
```

46.9.4 国家名称的统一

福布斯全球上市企业排行榜数据，我们获得的数据相对比较杂乱。

我们希望将国家名称进行规范，这里以国家英文名称的列“Country_en”作为清洗的目标。

国家英文名称中，不规范的情况包括：

1. 有些样本中国家的英文名称为空白
2. 有些是中文名称
3. 有些一个国家对应多个英文名称
4. 有些一个缩写对应多个不同的国家
5. 一家企业可能在多个国家有登记

46.9.4.1 国家英文名称为空值的处理

若 “Country_en”列有空值，则用“Conutry_cn”列的值替换

```
df_concat.loc[df_concat['Country_en']=='', 'Country_en'] = df_concat.loc[
    df_concat['Country_en']=='', 'Country_cn']
```

46.9.4.2 将国家名称统一为英文

将国家名称统一成英文名称，这里主要用的是replace()方法

```
list_origin = [']','AR', 'AS','AU', 'US', 'BE', 'BR', 'BS', 'BU', 'CA', 'CH', '
    CI',
               'CN-HK', 'CN-TA', 'CO', 'CZ', 'DE', 'EG', 'FI', 'FR', 'GE',
               'GR', 'Hong Kong', 'Hong Kong/China', 'HU', 'IC', 'ID', 'IN', 'IR
                   ',
               'IS','IT', 'JA', 'JO', 'KO','LI', 'LU', 'MX', 'NL', 'NO', 'NZ',
                'PA', 'PE', 'PH', 'PK', 'PL', 'PO', 'RU', 'SI', 'SP', 'SU',
               'SW', 'SZ', 'Taiwan', 'TH', 'TU', 'UK', 'VE', '阿根廷', '阿联酋',
               '阿曼', '埃及', '爱尔兰', '奥地利', '澳大利亚', '巴基斯坦', '巴林
                   ',
               '巴拿马', '巴西', '百慕大', '比利时', '波多黎各', '波兰', '丹麦',
               '德国', '多哥', '俄罗斯', '法国', '菲律宾', '芬兰', '哥伦比亚',
               '哈萨克斯坦', '海峡群岛', '韩国', '荷兰', '加拿大', '捷克共和国',
               '卡塔尔', '开曼群岛', '科威特', '克罗地亚', '黎巴嫩', '利比里亚',
               '列支敦士登', '列支敦斯登', '卢森堡', '马来西亚', '毛里求斯',
               '美国', '秘鲁', '摩洛哥', '墨西哥', '南非', '尼日利亚', '挪威',
               '葡萄牙', '日本', '瑞典', '瑞士', '塞浦路斯', '沙特$',
               '沙特阿拉伯', '斯洛伐克', '泰国', '土耳其', '委内瑞拉', '西班牙',
               '希腊', '新加坡', '新西兰', '匈牙利', '以色列', '意大利',
               '印度$', '印度尼西亚', '印尼', '英国', '约旦', '越南', '智利',
               '中国大陆', '中国台湾', '中国香港', 'CN$', '中国$', 'Netherlands'
                   ]

```

```
list_replace = ['','Argentina', 'Austria','Australia', 'United States', 'Belgium',
                'Brazil', 'Bahamas', 'Bermuda', 'Canada', 'Chile', 'Cayman Islands',
               'China-HongKong', 'China-Taiwan', 'Colombia', 'Czech Republic',
               'Denmark', 'Egypt', 'Finland', 'France', 'Germany', 'Greece', 'China-HongKong',
               'China-HongKong', 'Hungary', 'Iceland', 'Indonesia', 'India', 'Ireland', 'Israel',
               'Italy', 'Japan', 'Jordan', 'South Korea','Liberia', 'Luxembourg', 'Mexico',
               'Netherland', 'Norway', 'New Zealand', 'Panama', 'Peru', 'Philippines',
               'Pakistan', 'Poland', 'Portugal', 'Russia', 'Singapore', 'Spain',
               'Saudi Arabia', 'Sweden', 'Switzerland', 'China-Taiwan',
               'Thailand', 'Turkey', 'United Kingdom', 'Venezuela', 'Argentina',
               'United Arab Emirates', 'Oman', 'Egypt', 'Ireland', 'Austria', 'Australia',
               'Pakistan', 'Bahrain', 'Panama', 'Brazil', 'Bermuda', 'Belgium',
               'Puerto Rico', 'Poland', 'Denmark', 'Germany', 'Togo', 'Russia',
               'France', 'Philippines', 'Finland', 'Colombia', 'Kazakhstan',
               'Channel Islands', 'South Korea', 'Netherland', 'Canada',
               'Czech Republic', 'Qatar', 'Cayman Islands', 'Kuwait', 'Croatia', 'Lebanon',
               'Liberia', 'Liechtenstein', 'Liechtenstein', 'Luxembourg', 'Malaysia',
               'Mauritius', 'United States', 'Peru', 'Morocco', 'Mexico', 'South Africa',
               'Nigeria', 'Norway', 'Portugal', 'Japan', 'Sweden', 'Switzerland', 'Cyprus',
               'Saudi Arabia', 'Saudi Arabia', 'Slovakia', 'Thailand', 'Turkey', 'Venezuela',
               'Spain', 'Greece', 'Singapore', 'New Zealand', 'Hungary', 'Israel', 'Italy',
               'India', 'Indonesia', 'Indonesia', 'United Kingdom', 'Jordan', 'Vietnam',
               'Chile',  'China', 'China-Taiwan', 'China-HongKong', 'China', 'China',
               'Netherland']

df_concat['Country_en'] = df_concat['Country_en'].replace(list_origin, list_replace, regex=True)
```

两个 list 数据的录入，没有捷径，自己录入的。

46.9.4.3 一家公司属于多个国家的情况

进一步替换，包括一家公司属于多个国家的情况。

```
# 进一步替换，包括一家公司属于多个国家的情况
# 注意这里没有 regex=True
df_concat['Country_en'] = df_concat['Country_en'].replace(['China-China-Taiwan',r'China-HongKong/China', r'Australia)/United Kingdom(United Kingdom',
                                                           r'Netherland)/United Kingdom(United Kingdom', r'Panama)/United Kingdom(United Kingdom',
                                                           r'SA)/United Kingdom(United Kingdom', r'United Kingdom)/Australia(SA', r'United Kingdom)/Netherland(Netherland',
                                                           r'United Kingdom)/South Africa(SA', 'Netherland/United Kingdom', 'Panama/United Kingdom',
                                                           'Australia/United Kingdom'],
                                                          ['China-Taiwan', 'China-HongKong', 'United Kingdom/Australia', 'United Kingdom/Netherland',
                                                           'United Kingdom/Panama', 'United Kingdom/Australia', 'United Kingdom/Australia',
                                                           'United Kingdom/Netherland', 'United Kingdom/South Africa', 'United Kingdom/Netherland',
                                                           'United Kingdom/Panama', 'United Kingdom/Australia'])
```

46.9.4.4 同一个英文缩写代表不同国家的情况

通过观察数据，发现有一类特殊情况，同样的英文缩写，代表的国家可能不同，需要进一步处理。

如 **SA 可以代表 Australia 或 South Africa**，MA 可以代表 Malaysia 或 Morocco

处理这类特殊情况的思路与步骤为：

（a）将此类英文缩写用空白值替换

（b）将空白值替换为当前样本的中文国家名称

（c）将中文国家名称替换为各自的英文国家名称

```
# 注意需要以 "$"结尾
df_concat['Country_en'] = df_concat['Country_en'].replace(['MA$','SA$'],['', ''], regex=True)

df_concat.loc[df_concat['Country_en']=='', 'Country_en'] = df_concat.loc[df_concat['Country_en']=='', 'Country_cn']

df_concat['Country_en'] = df_concat['Country_en'].replace(['澳大利亚','马来西亚', '摩洛哥', '南非'],
                                                          ['Australia', 'Malaysia', 'Morocco', 'South Africa'])
```

最后，保存数据

```
df_concat.to_csv('data_forbes_concat.csv')
```

46.9.5 总结

2016 年至 2017 年的数据清洗，前文已作详细描述，本文主要是描述国家名称的统一处理。

至此，福布斯全球上市企业排行榜数据（2007 年至 2017 年）的清洗告一段落。

结合前文 2007 年-2015 年的数据清洗，将 2007-2017 年的数据清洗的代码合并到一起，感兴趣的同学可以回复关键字获取相关代码。

> 关注微信公众号公众号“Python 数据之道”，后台回复“**code**”，获取本文的源代码及原始数据文件。

47 学习资料: 精选文章

47.1 付费阅读文章

- 自律是一种觉醒
- 阅读是投资自己最有效的方式
- 用 Plotly 动态柱状图来演示全球疫情变化趋势
- 超火动态排序疫情变化图，这次我们用 Plotly 来绘制

47.2 免费阅读文章

- 用 Python 读取巴菲特近期持仓数据
- 推荐：这才是你寻寻觅觅想要的 Python 可视化神器
- 干货推荐: 掌握这几点，轻松玩转 Bokeh 可视化（项目实战经验分享）
- Matplotlib 实操干货，100 个案例带你从入门到进阶！-Part 1
- Matplotlib 实操干货，100 个案例带你从入门到进阶！-Part 2
- 巧用 Matplotlib 动画, 让你的 Python 可视化大放异彩
- Flask vs Django，Python Web 开发用哪个框架更好?
- 手把手教你用 Python 进行机器学习
- 不会时间序列预测？不要紧，大神来教你
- Python 中必须知道的 5 对魔术方法

47.3 读者投稿文章

- Anaconda 零基础使用说明
- Python 正则表达式，从入门到实战，精华都在这里！
- 讲真，Markdown 有啥魅力，竟让如此多人入迷！
- 玩转词云图，推荐一个 Pyecharts 和 Plotly 数据分析实战项目

- 炫酷！用 Pyecharts 制作漂亮的流动桑基图
- Pyecharts 绘制饼图原来可以如此漂亮！
- Pyecharts 组合图形绘制实践
- 图解超经典的 KNN 算法 - 机器学习算法入门
- TensorFlow 零基础入门指南
- 超经典 MySQL 练习 50 题，做完这些你的 SQL 就过关了

48 学习资料: Python 网站

《Python 知识手册》的内容毕竟有限，互联网上有很多优秀的资源，平时在浏览过程中收集一些 Python 相关的网站，在此罗列，大家挑选适合的就好。

这个资料列表会不定期更新，大家如有觉得不错的资料，欢迎推荐给我。

48.1 官方网站

- Python 官方网站
- Anaconda 官方网站
- Pandas
- Matplotlib
- Plotly

48.2 个人博客

- Python 数据之道的官方网站，这个是我个人的网站，内容不定期更新。
- 廖雪峰的官方网站，廖老师的 Python 学习网站，知名度还是挺高的。

48.3 金融量化

- Python for Finance

48.4 优质网站

- Kaggle
- Stackoverflow
- Towards Data Science
- Planet Python
- Python Awesome
- Practical Business Python
- Python weekly

如果您觉得有不错的资料，欢迎推荐给我。

49 学习资料：Python 书籍

经常有读者问我，想让我推荐一些关于 Python 学习的书籍，鉴于每个人的知识背景以及跟我交流的深浅程度不一样，可能每次推荐的书籍都会有些差异。

最近有些想法，打算陆陆续续介绍一些书籍，希望能给大家带来一些益处。

需要注意的是，我本身不会刻意对书籍的内容进行点评，尽量做到相对客观的对书籍进行介绍。

至于你是否需要去学习文章中介绍的书籍，这个还是需要结合自己的实际情况来做筛选。毕竟每个人的时间和精力都有限，市面上优秀的书籍数不胜数，我们挑选适合自己的就好。

49.1 《笨办法学 Python3》

> 笨办法学 Python，其实一点都不笨

本次来介绍的书籍是《笨办法学 Python 3》，这本书的作者是泽德 A. 肖（Zed A.Shaw）。

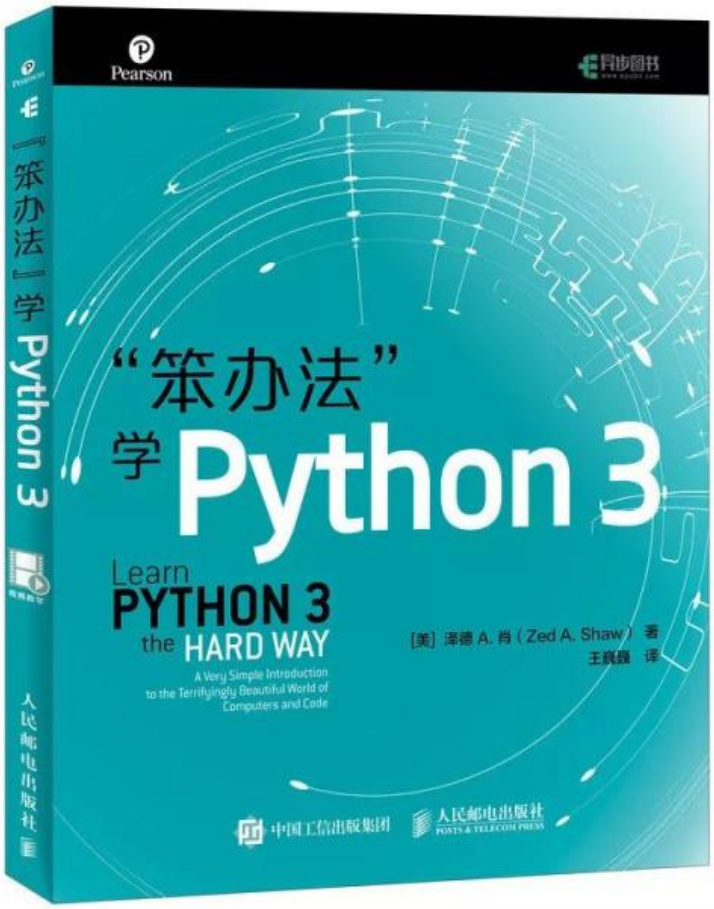


Figure 49.1: img-笨办法学 Python3-封面

本书作者是《“笨办法”学 Python》《“笨办法”学 Ruby》《“笨办法”学 C 语言》等几本备受欢迎的图书的作者，他的书在全世界有数百万读者。他还是诸多开源项目的创建者，拥有近 20 年的编程和写作经验。

《笨办法学 Python 3》基于 Python 3.6，是一本从零基础入门开始介绍 Python 3 的书籍。本书介绍了一个非常有用的学习窍门，就是把编程书籍习题化，全书共 52 道习题。

习题的安排遵循了循序渐进的原则，每道题 2~5 页不等，内容相对通俗易懂，对于初学者而言，可以减少不少枯燥之处。

下面将就书中的部分内容进行描述，希望大家对本书有所了解。

49.1.1 习题 1：第一个程序

作者建议先编写一个完整的 Python 程序，即 `.py` 文件，然后在终端运行代码。

例如，文件 `ex1.py` 的代码如下

```
print('Hello World!')
print('Hello again')
print('I like typing this.')
print('This is fun.')
print('Yay! Printing.')
print("I'd much rather you 'not'.")
print('I "said" do not touch this.')
```

在终端运行下面代码

```
python ex1.py
```

我运行的效果如下：

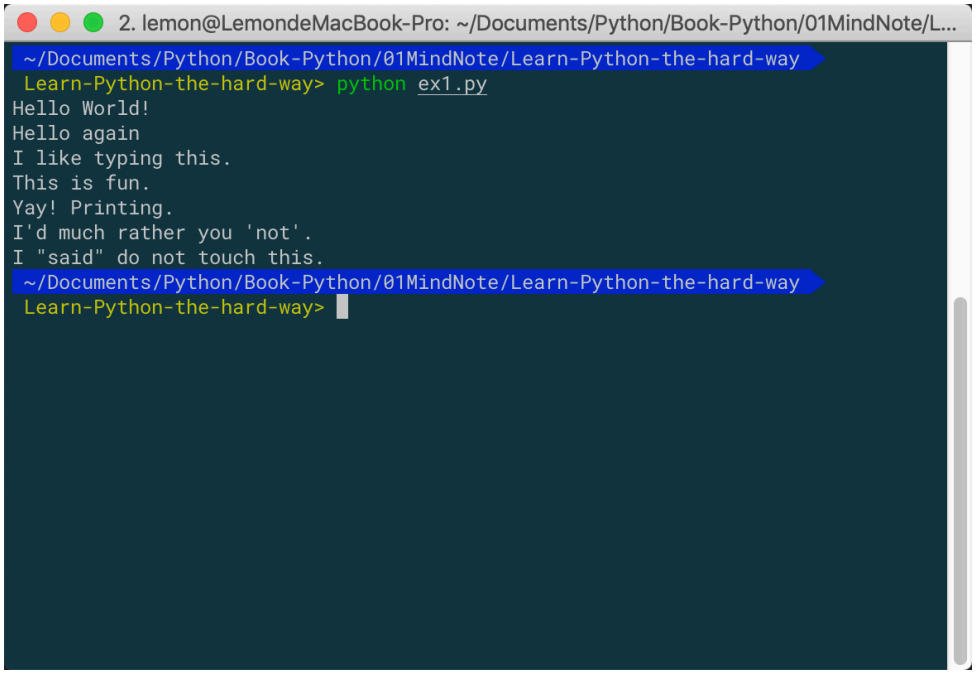


Figure 49.2: img-演示

49.1.2 习题 4：变量和命名

在这个习题中，提到了一些我们经常遇到的小问题。

比如：

> = （单等号）和 ==（双等号）有什么不同?

这个问题的答案，请自行查找（本书也进行了阐述）。

比如：

> 书写 x=100 和 x = 100 有什么差异?

上面两种写法，本质上都是可以的，但操作符两边加上空格，会让代码可阅读性更好。

49.1.3 习题 6：字符串和文本

在 Python 3 中，字符串有几种常见的输入表达方法，本书着重介绍了一种，称为 “f-string”，其语法格式如下：

```
f"some stuff here {avariable}"
```

在 ipython 中实际运行下面代码：

```
num = 10

y = f"there are {num} apples."

print(y)
```

效果如下：

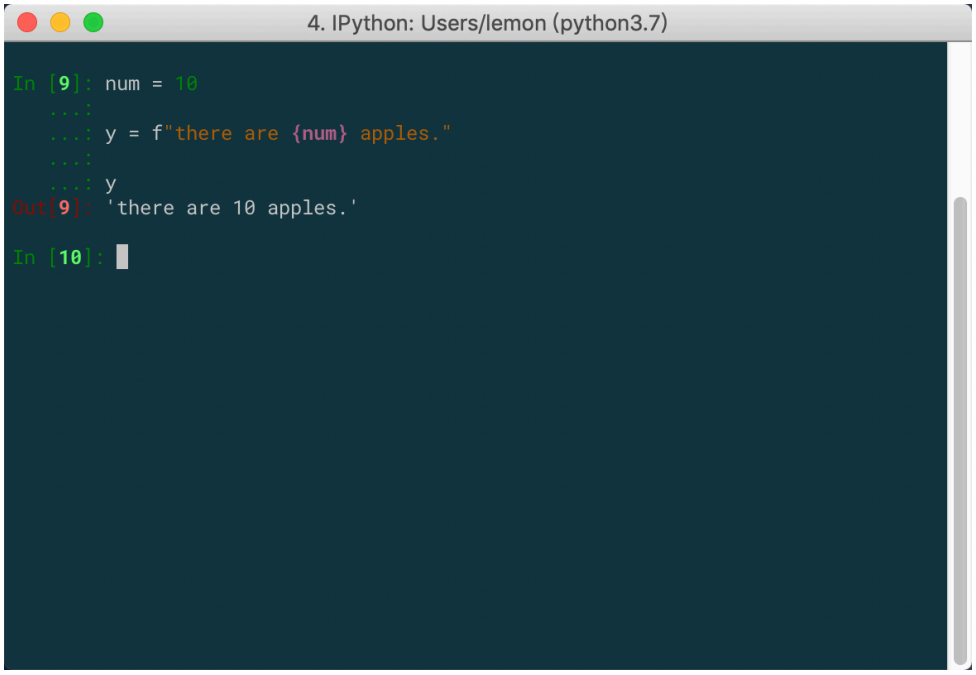


Figure 49.3: img 演示

同时，关于 字符串中 .format() 语法格式的使用，书中亦有描述。

49.1.4 总结

总体而言，本书 52 道习题，内容逐步深入，在最后几道习题，作者通过使用 flask 来建立一个微小型的网页，使本书内容延伸到 web 编程的范畴。

在本书的结尾部分，作者也给出了一些建议：

> 你已经完成了这本书并且打算继续编程。也许这会成为你的职业，也许你只是作为业余爱好玩玩而已。无论如何，你都需要一些建议以确保你在正确的道路上继续前行，并且让这项新的爱好最大程序为你带来享受。
>
> .....
>
> **编程语言这东西并不重要，重要的是你用这些编程语言做的事情。**

可以说，本书以一种较为新颖的方式，帮助广大读者完成了 Python 学习从 0 到 0.5 的过程，如果需要继续深入了解 Python，可以配合其他书籍进一步学习。

最后，**「Python 数据之道」**将本书的内容概览进行了整理，以便更加清晰的了解本书的内容结构，在公众号「Python 数据之道」后台回复 “**pdf**” 可以获取本书高清版 pdf 版的思维导图。

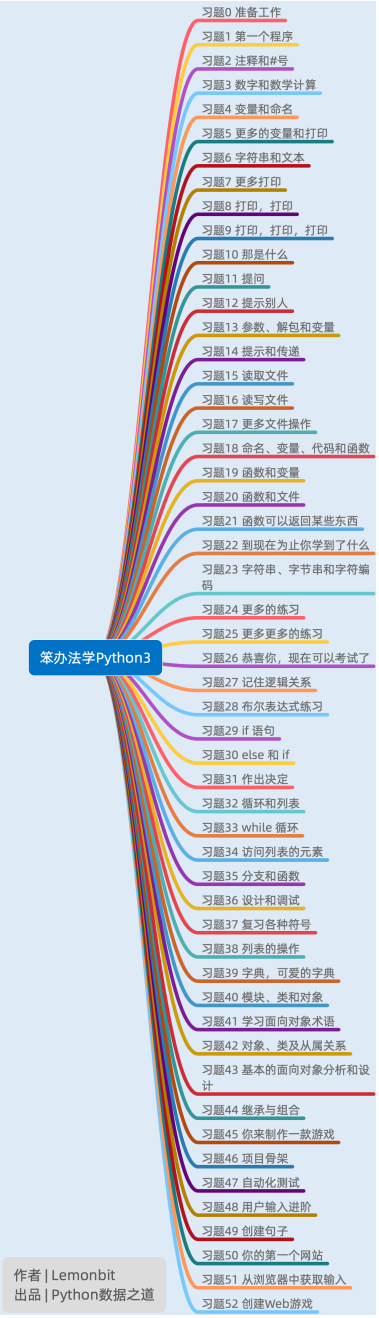
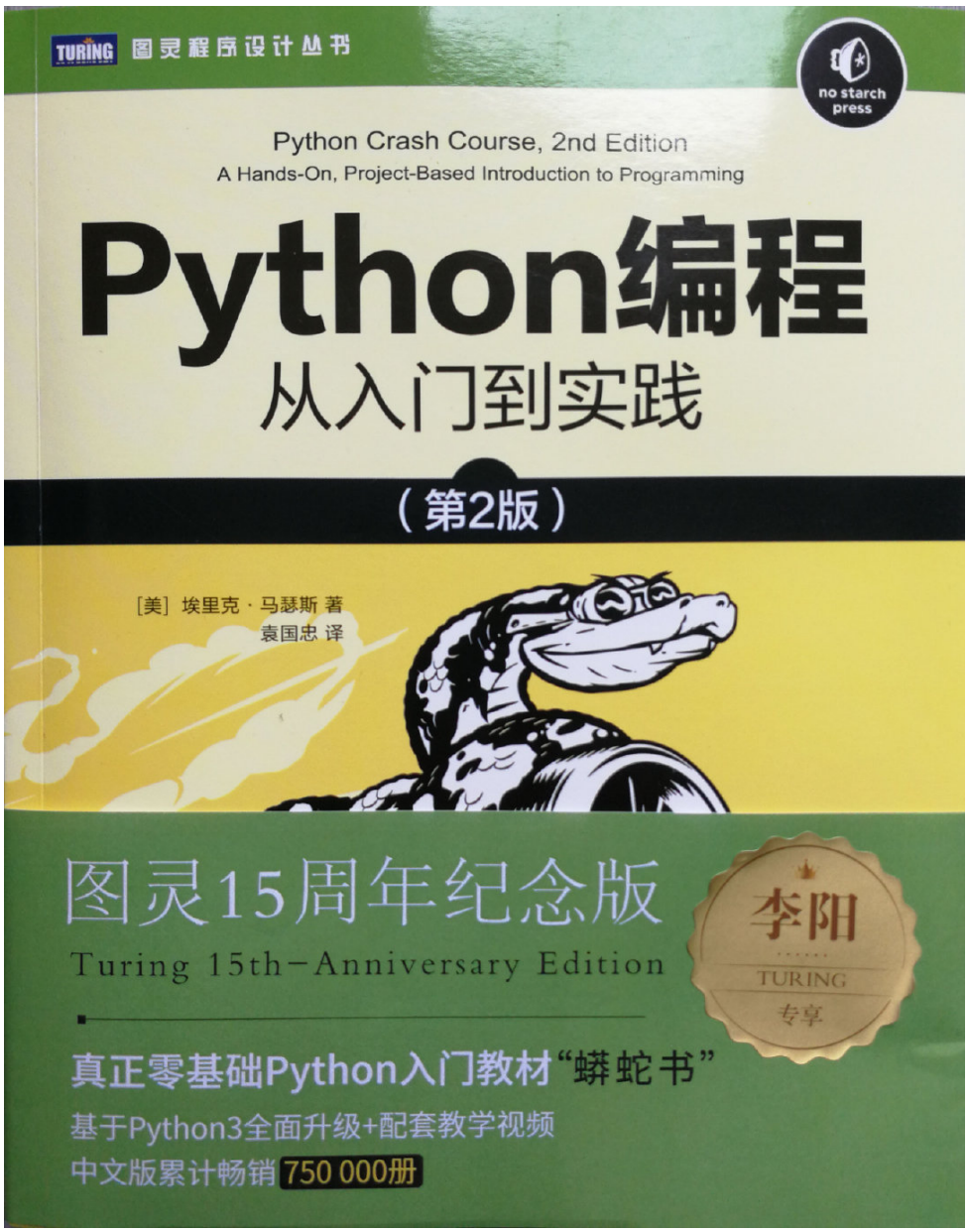


Figure 49.4: img-思维导图

49.2 《Python 编程从入门到实践》，第 2 版

49.2.1 前言

下面来介绍的书籍是《Python 编程从入门到实践》（第 2 版）。



49.2.2 书籍背景

49.2.2.1 作者介绍

埃里克·马瑟斯（Eric Matthes），高中科学和数学老师，现居住在美国阿拉斯加，在当地讲授 Python 入门课程。他从 5 岁开始就一直在编写程序。

受其父亲的影响，5 岁的埃里克·马瑟斯开始编写自己的第一个程序——一个简单的猜数字游戏。30 岁时，作为 Python 爱好者，他开始在技术社区中义务教授 Python 。源于对 Python 的好奇心，他的儿子 Ever 每天不断提问，这才驱使他有了写作这本书的想法。所以，与其说它是一本书，倒不如说它是对父子两代人编程初心的传承。

49.2.2.2 书籍获得的赞誉

蟒营创始人表示：

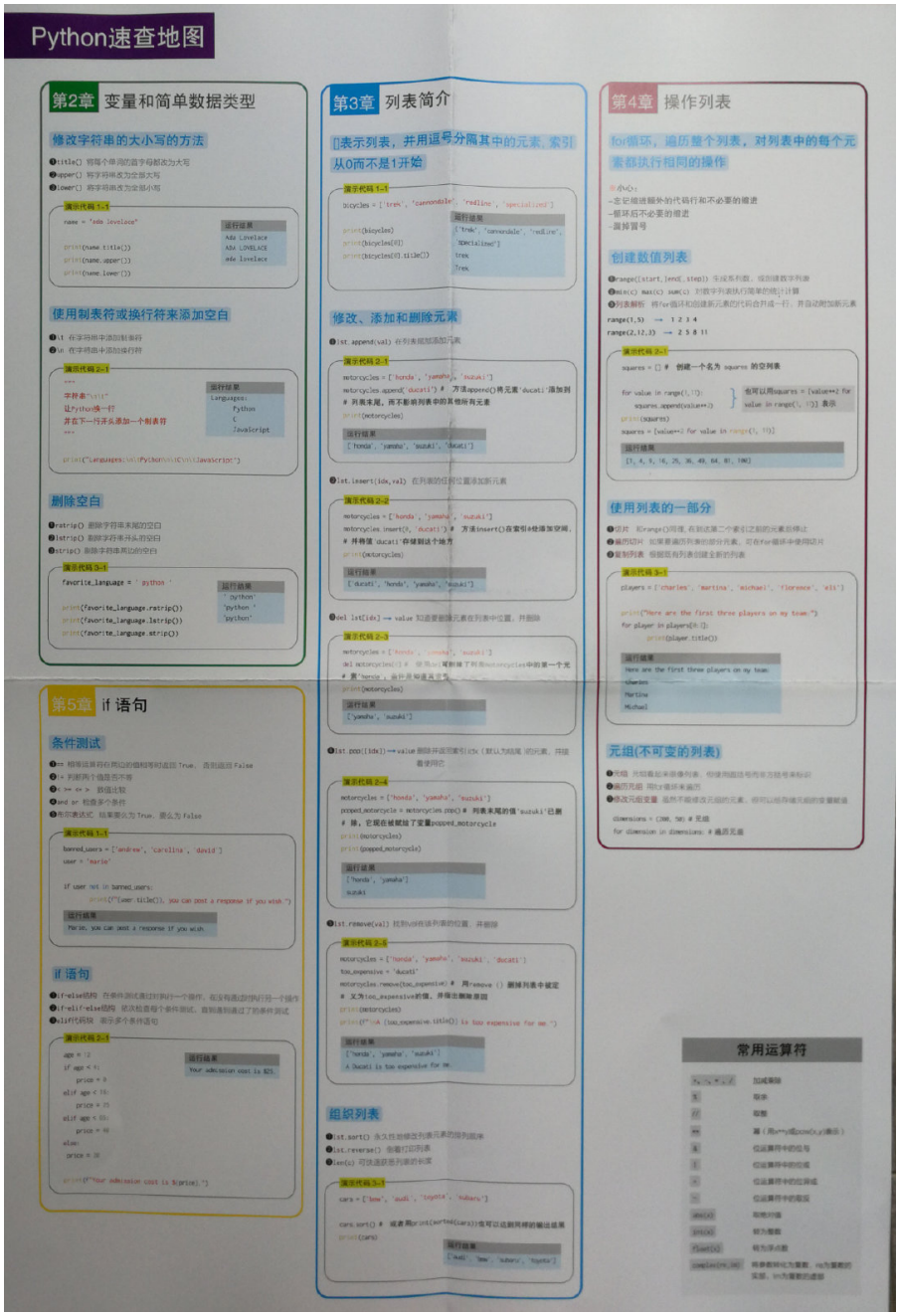
> 20 多年过去了，市面上一直不乏各种教授“零基础入门 Python”的图书，但至今只有两本摸到了门径。一本是《笨办法学 Python》，另外一本，就是这本“Python 蟒蛇书”《Python 编程从入门到实践》。

49.2.3 本书内容

《Python 编程从入门到实践》有两个大的部分组成，分别是 Python 基础知识和项目实践。

如果你已经有一定的 Python 基础或者其他编程基础，第一部分可以先快速的阅览，在项目实践部分再来回顾，这样可能效果更好些。

此外，本书随书提供了纸质版的速查表，我拍了下照片，但不是太清晰（手机太渣……）



不过不要紧，在这里，我也给大家整理了本书的内容框架，虽然没有示例的代码，但通过下面的思维导图也能对本书有个大概的了解。

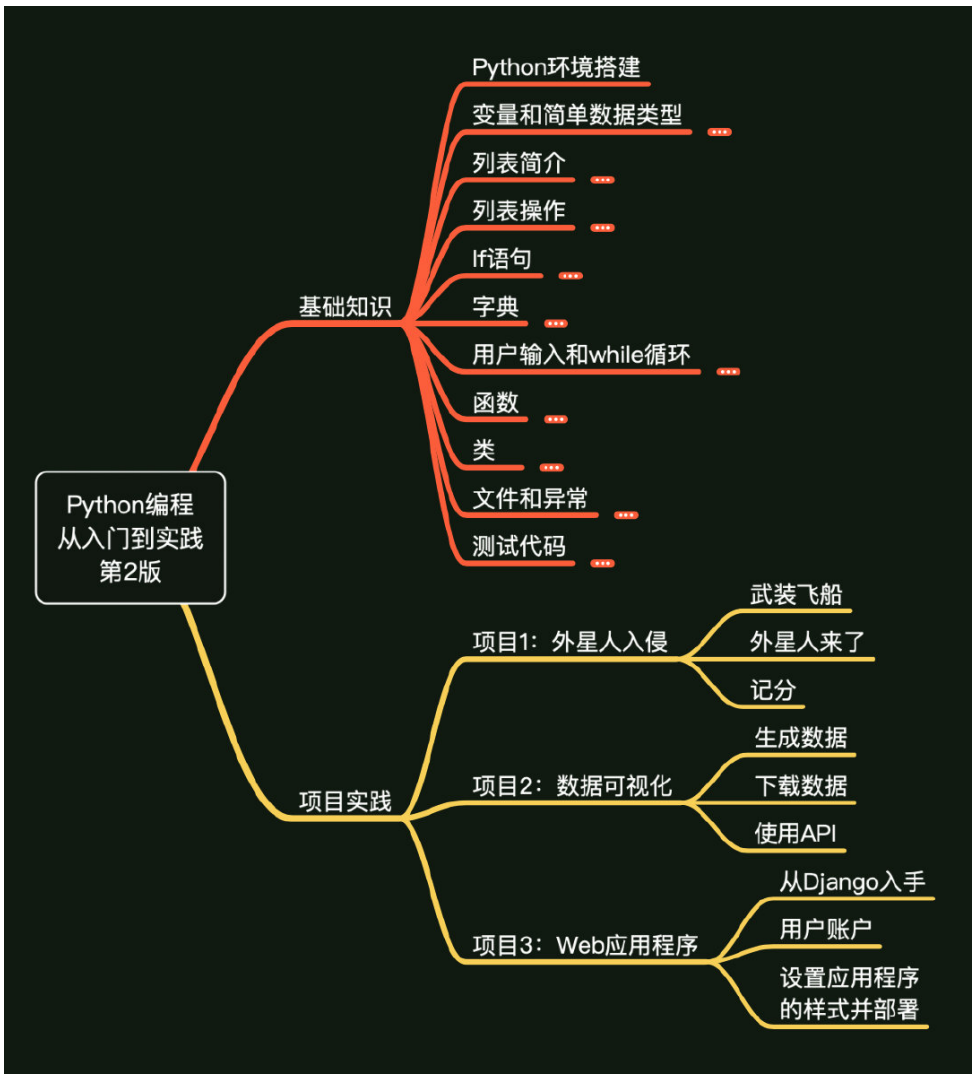


Figure 49.5: 本书的整体框架

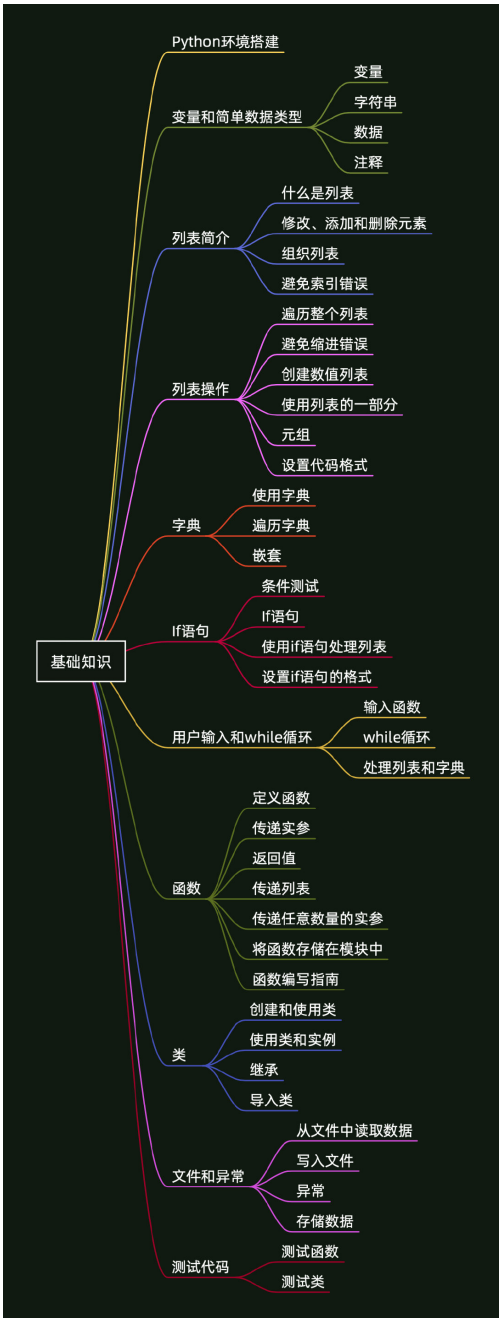


Figure 49.6: Python 基础知识部分

50 延伸阅读

50.1 我是谁

大家好，我是 Lemon ，公众号「Python 数据之道」号主。

公众号「Python 数据之道」秉承“让数据更有价值”的理念，主要分享数据相关的内容，包括数据分析，挖掘，可视化，机器学习，深度学习等，希望能给大家分享有价值的内容。

若对我写的内容有兴趣，欢迎大家通过以下途径来关注。

50.1.1 微信公众号

Figure 50.1: Python 数据之道

「Python 数据之道」是 Lemon 分享关于 Python 及数据分析相关内容的主阵地。

《Python 知识手册》会不定期更新，大家可以在公众号「Python 数据之道」后台回复 **600** 获取最新版。

50.1.2 个人网站

网址：http://liyangbit.com

Lemon 的个人网站中，包含更多的文章，并且在不断的进行更新。目前，网站中涉及了 Python 相关一系列内容，包括 Python 基础、Python 数据科学、项目实战等内容，欢迎访问。

50.2 推荐内容

50.2.1 《Markdown 速查表》

Lemon 给大家整理了一份 Markdown 速查表，先晒一张图：

Figure 50.2: Markdown 速查表

完整的 Markdown 速查表 （高清 pdf 版），**一共 3 页**，大家可以在下面的公众号 **「柠檬数据」**回复 **markdown** 来获取。

Figure 50.3: 柠檬数据

50.2.2 精选书单

Lemon 是一个爱好读书的人，平时喜欢在公众号「价值前瞻」上写一些读书笔记内容，目前发布的有《如何阅读一本书》、《巴菲特之道》、《金字塔原理》、高瓴张磊的《价值》、《投资最重要的事》、《戴维斯王朝》等书籍的笔记内容和思维导图等。

大家可以在公众号「价值前瞻」后台回复 **「书单」**来获取精选书单和思维导图等。

Figure 50.4: 价值前瞻

部分书籍内容如下：

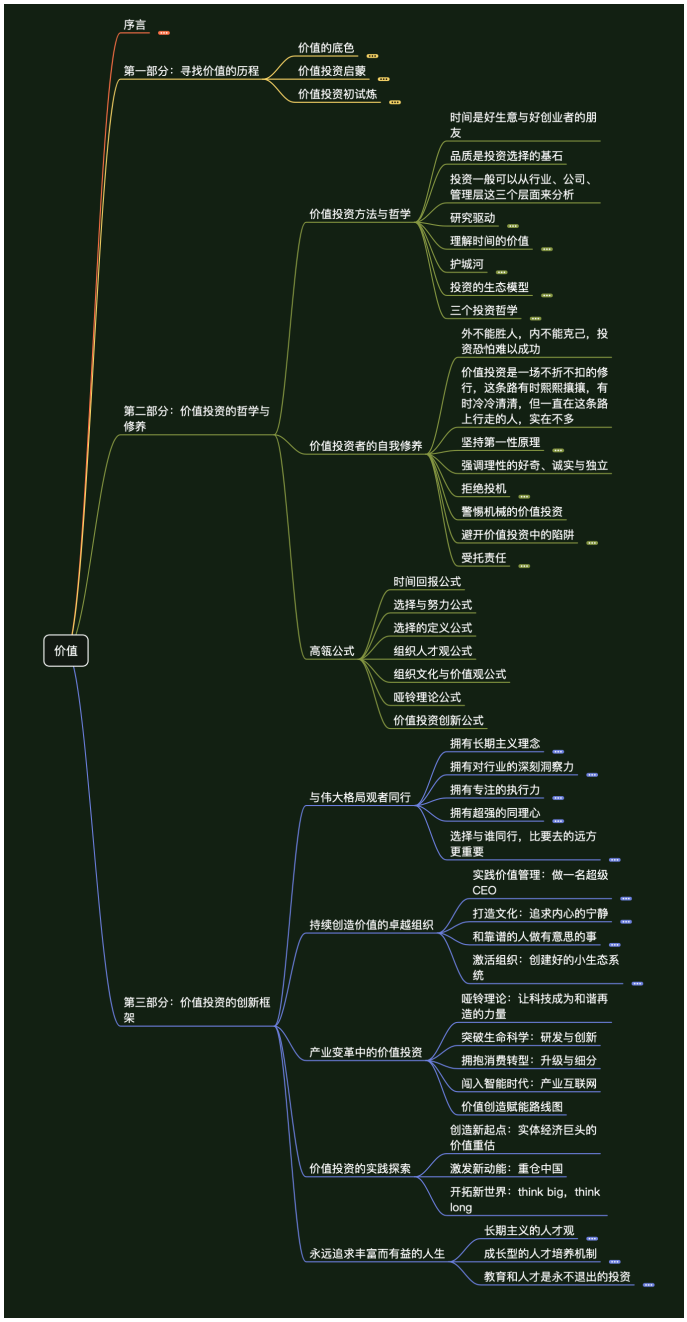


Figure 50.5: 《价值》

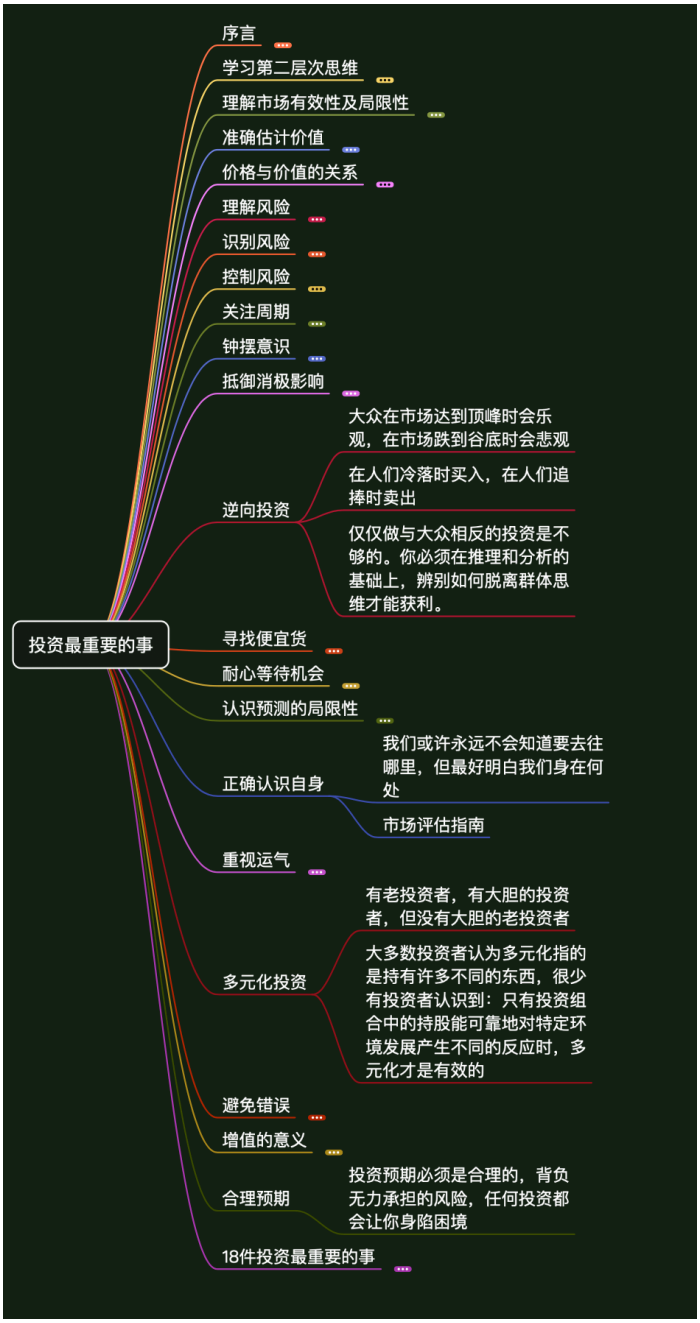


Figure 50.6: 《投资最重要的事》

读书备忘录：《戴维斯王朝》

Version: 01

出品：价值前瞻

背景

这本书是一本类似传记的书，书中记录的戴维斯家族投资历程：

- 经历了 2 次漫长的牛市、25 次调整、2 次严酷的熊市、1 次大盘崩溃、7 次温和的熊市、9 次衰退；
- 第二次世界大战、冷战、人类登月、石油危机、“漂亮五十”的崛起和崩溃；
- 1 次总统遇刺、1 次总统辞职、1 次总统遭遇弹劾；34 年利率上升和 18 年的利率下降 ……

戴维斯家族打破了“富不过三代”的魔咒，历经经济的盛衰兴替，股市的牛熊起伏，屹立华尔街 50 年不倒，的确值得我们认真研读。

戴维斯家族人物介绍

书中涉及到戴维斯家族祖孙三代人：

- 戴维斯，全称：谢尔比·卡洛姆·戴维斯，后续简称：“戴维斯”，1906 年出生，普林斯顿大学毕业；
- 戴维斯的儿子：谢尔比·戴维斯，是的，名字与他父亲一样的，后续简称“谢尔比”，1937 年出生，普林斯顿大学毕业；
- 戴维斯的孙辈：安德鲁、克里斯和维多利亚，出生在 20 世纪 60 年代。

戴维斯

戴维斯的传奇，用一句话来概括，就是：“从 5 万美元到 9 亿元美元的传奇”。

戴维斯的故事的结尾的确是激动人心和充满希望的：一个 38 岁才开始投资的中年男人，最终成为一个在 20 世纪 90 年代就身价近 10 亿美元的大富翁。

谢尔比

这本书最初的想法是描述基金经理谢尔比·戴维斯。他管理纽约风险基金长达 28 年，为他的投资人创造了辉煌的业绩，他将每一个最初投资的 1 万美元变成了 37.9 万美元。但谢尔比和他的基金很低调。在这 28 年终，他有 22 年击败市场大盘。这个记录足以令他与富达公司传奇的基金经理彼得·林奇比肩。

十大投资原则

在书中，总结了戴维斯家族的十大投资原则：

1. 避开廉价股
2. 避开高价股
3. 以合理的价格购买适度增长的公司股票
4. 等待合理价格的出现
5. 顺势而为
6. 主题投资
7. 让你的赢家继续奔跑
8. 投资于优秀的管理层
9. 忽略后视镜
10. 坚持到底

投资检查清单

谢尔比就其主持的纽约风险投资基金中每一只股票的持股原因写下了一个备忘录。这些持股，即便不能满足所有条件，至少应该满足大部分特征。

特征列表如下：

- 一流的管理以及信守承诺的良好记录
- 创新研究，使用科技手段将优势最大化
- 在国外市场的运营如同在国内市场一样优秀
- 销售的产品或服务永不会过时
- 能提供给股东强劲资本回报的公司，以及心怀投资者利益的管理层
- 能保持最低成本，是的公司具有低成本优势
- 在一个成长的市场中，能占有主导地位或能不断扩大市场份额的公司
- 善于并购竞争对手，并提高其利润率的公司
- 财务强健的公司

儿童的财商教育

谢尔比鼓励孩子们将打零工的收入进行投资，以取得复利式增长。为了让孩子们认识到财务杠杆的威力，孩子们每投资自己的一美元，谢尔比就再借给他们一美元，以 1：1 的配资形式让他们进行投资。

戴维斯双击

什么是戴维斯双击？

当一家公司利润上升时，推动股票价格上涨，投资者赋予该股更高估值，推动股票价格再次上涨。戴维斯将这种有利可图的转化称之为“戴维斯双击”。

戴维斯双杀

戴维斯双击的反方向效应，称之为“戴维斯双杀”。

让我们假设一下，一只热门的成长股，每股盈利 1 美元，以 30 倍市盈率标价，该股价格应该为每股 30 美元。如果该股盈利下降一半，同时投资者不在抱有幻想，只愿意支付 15 倍市盈率，那么这只 30 美元的股票瞬间便会跌至 7.5 美元。这种情况通常称之为“戴维斯双杀”。

金句分享

- 乐观是每一个成功的投资者不可或缺的特质
- 只有股票才是超级财富的创造者
- 全体一致的意见是危险的信号
- 从历史上看，股市升跌与经济并非完全同步。股市崩溃的时候，经济往往还不错；而股市飙升的时候，经济有时还很悲惨。

精选书单

在公众号「**价值前瞻**」回复「**书单**」获取精选书单，包括《如何阅读一本书》、《巴菲特之道》、《价值》、《投资最重要的事》、《金字塔原理》等

作者：Lemon 出品：价值前瞻 2020-12-17 1 / 1

Figure 50.7: 《戴维斯王朝》

作者：Lemon
公号：Python数据之道

《Python知识手册》会不定期更新，在公号回复数字「600」可获取最新版