Python知识手册

v2018

作者｜李阳

出品｜Python数据之道

Python 知识手册-v2018

『Python 数据之道』

李阳

February, 2019

前言

“种一棵树，最好的时间是十年前，其次是现在。”

博客刚兴起不久的时候，大概 04 年左右，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了。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 相关的内容，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》不是一本从 Python 零基础开始的手册，针对手册的阅读，各位读者最好有一些 Python 的基本功底。当前版本中（v2018），手册主要包含以下几方面相关的内容：

- Jupyter Notebook
- Numpy
- Pandas
- Matplotlib
- Seabon
- Bokeh
- 若干个项目实战案例

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
- python 3.5 或 python 3.6
- Anaconda / PyCharm / Jupyter Notebook

若上述这些内容，能给诸位读者带来益处，我觉得付出些辛劳，也是值得的。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》将会不定期进行更新，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。

若对我写的内容有兴趣，欢迎大家通过以下途径来关注或者加入社群。

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。公众号所有文章都可以免费阅读，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. 知识星球

由于个人时间和精力有限，对于众多读者的提问和需求不能一一交流与回复。开通了付费的知识星球『Python 数据之道成长圈』，希望能够集中有限的时间来更好的交流和互动，同时可以汇聚一部分志同道合的人员，共同成长。

4. 感谢的话

最后聊一下我为什么写这本手册：作为气候变化和新能源领域的从业者，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 1 年多时间里，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』，结识了不少业界大佬以及新的朋友，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。感谢 @ 陈典铖为《Python 知识手册》制作了精美的封面，衷心感谢诸位大佬以及众多读者对手册内容的支持。

当然，由于个人水平有限，文章中内容难免有不准确的地方，欢迎通过公众号联系我。

李阳

25 Feb 2019

Contents

1 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。

1.1 新的风格预览

首先来看看我现在使用的风格，如下：

notebook 页面：

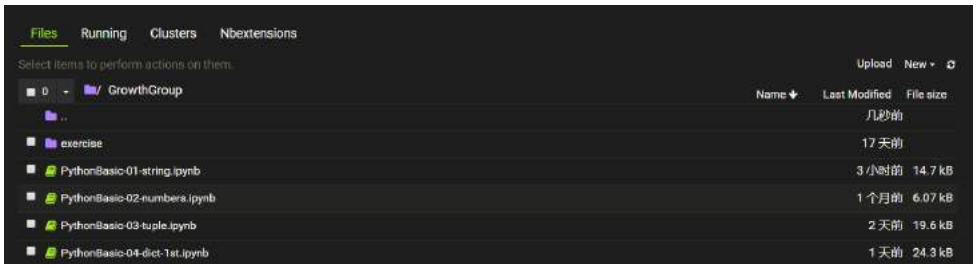


Figure 1.1: notebook 页面

代码文件页面：

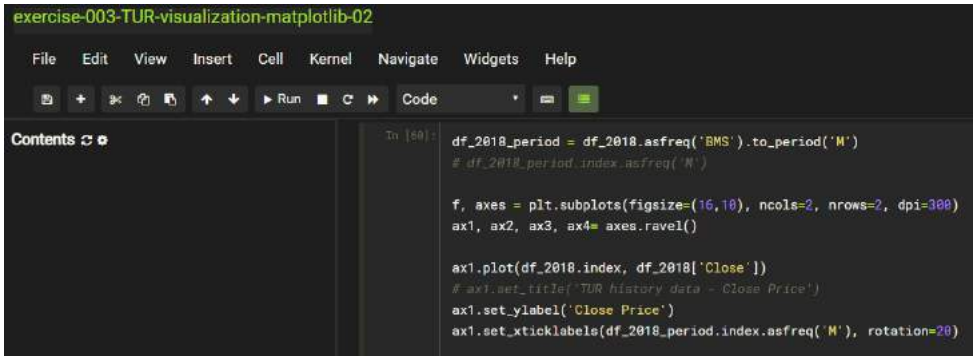


Figure 1.2: 代码文件页面

1.2 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 1.3: 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

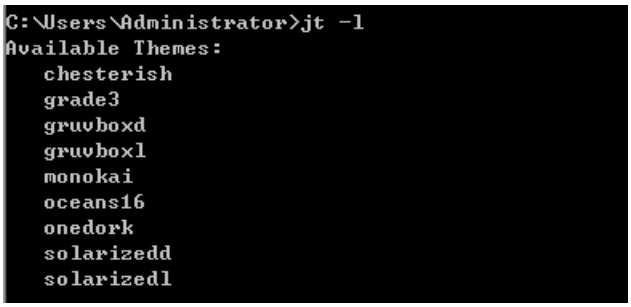


Figure 1.4: 可选主题

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：

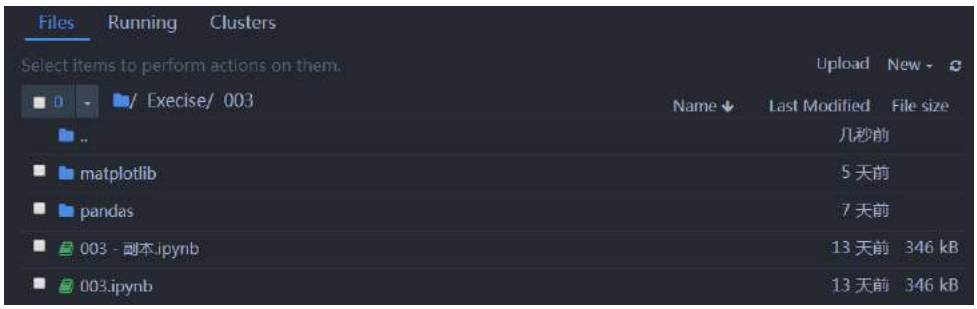


Figure 1.5: oceans16 主题

Figure 1.6: 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。

1.3 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 1.7: 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。

2 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：

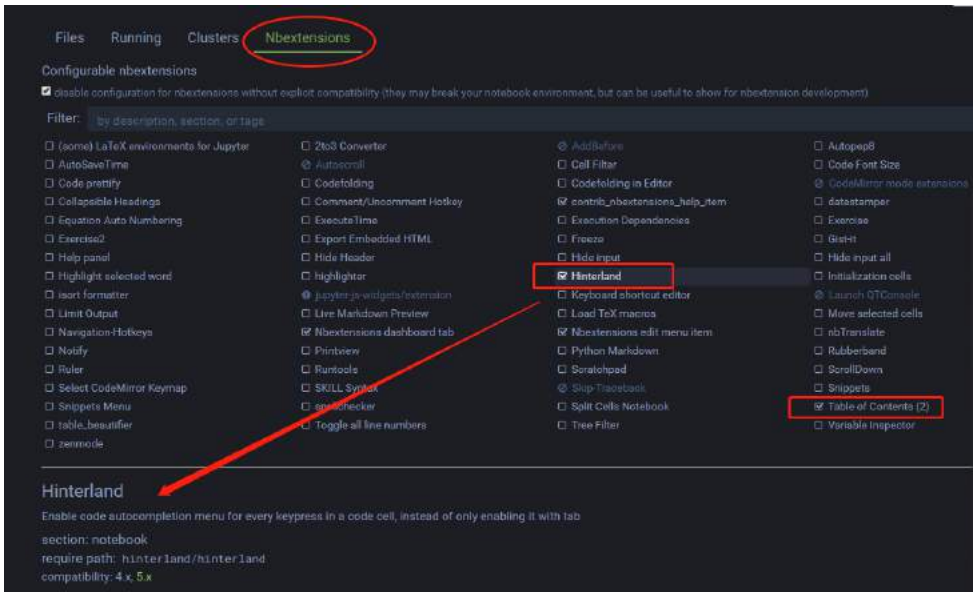


Figure 2.1: Hinterland 设置

自动补全代码的效果如下：

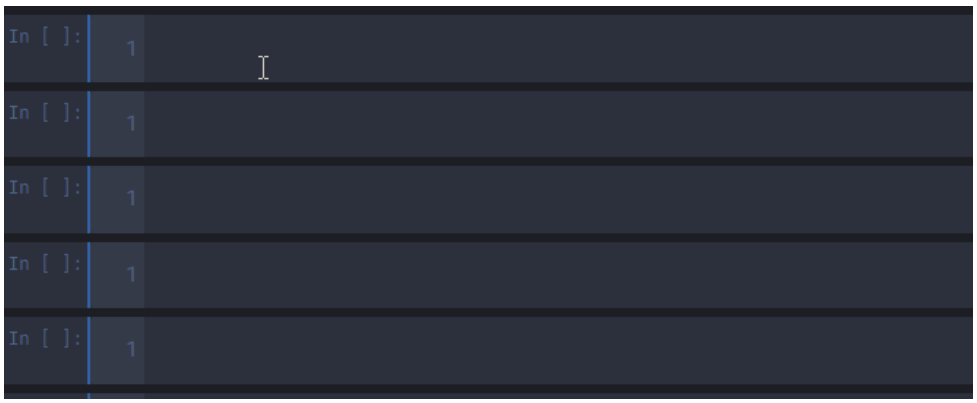


Figure 2.2: 自动补全代码的效果

> 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/

3 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）

3.1 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：

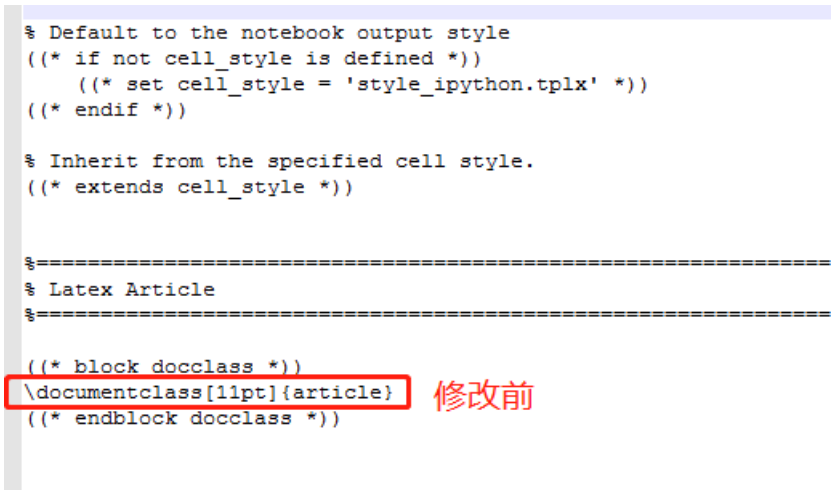


Figure 3.1: 修改前

修改后：

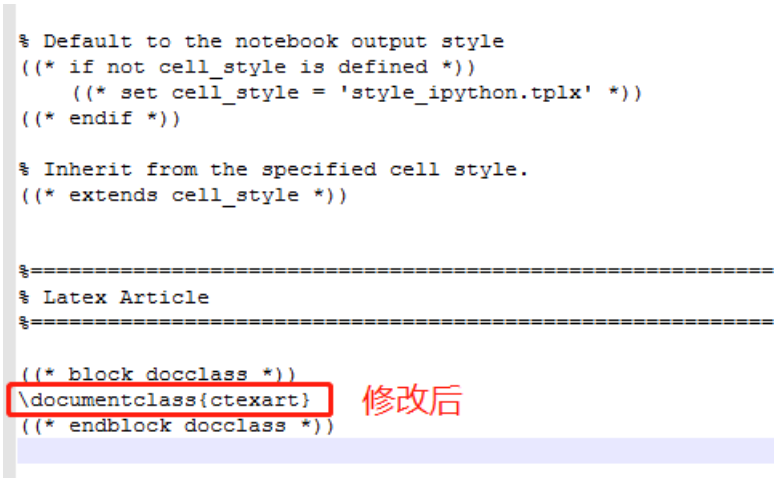


Figure 3.2: 修改后

（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：

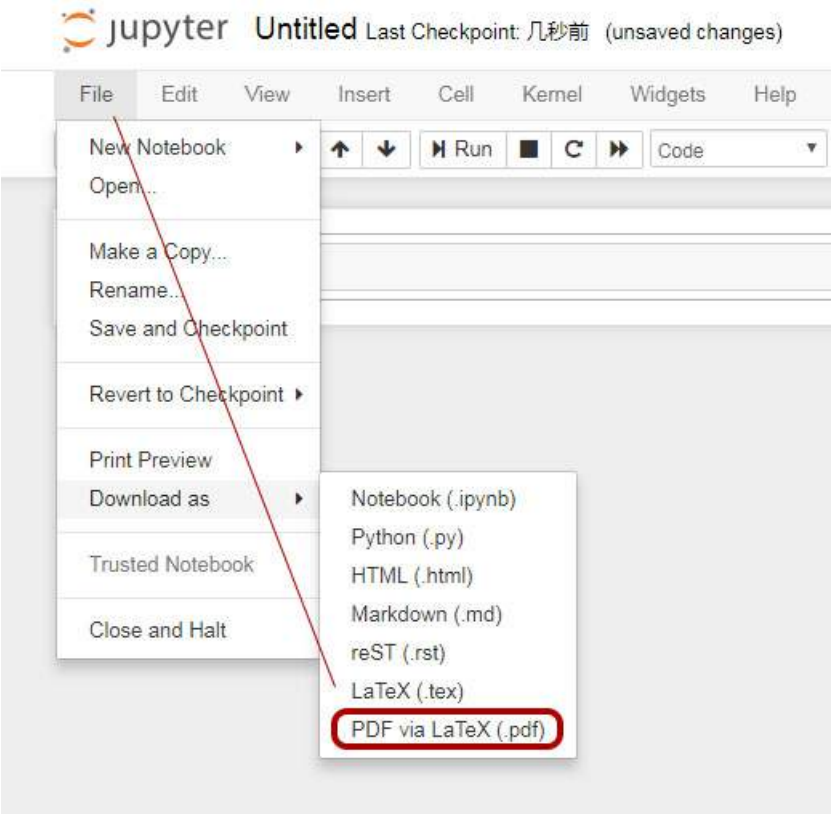


Figure 3.3: ipynb 文档

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

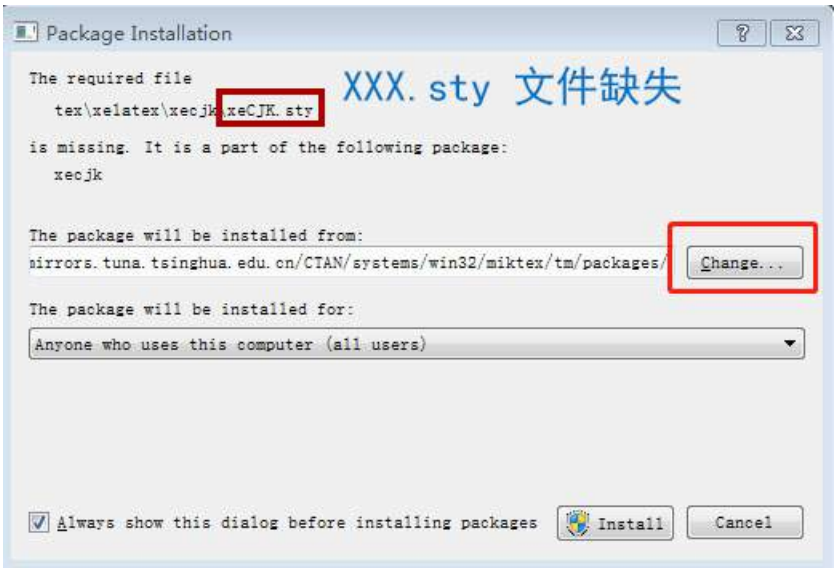


Figure 3.4: 安装缺失文件

按下面图示选项，点击“next”

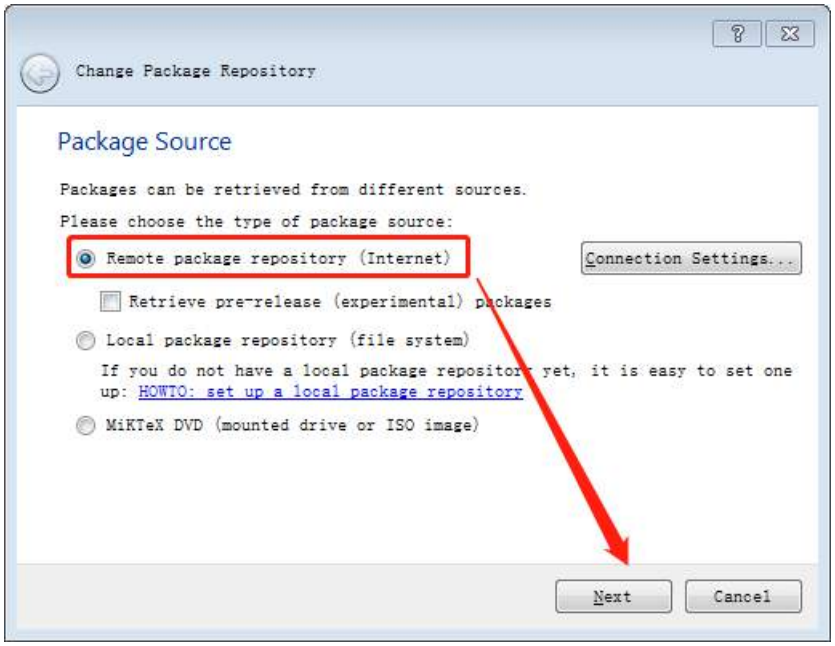


Figure 3.5: 安装缺失文件

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。

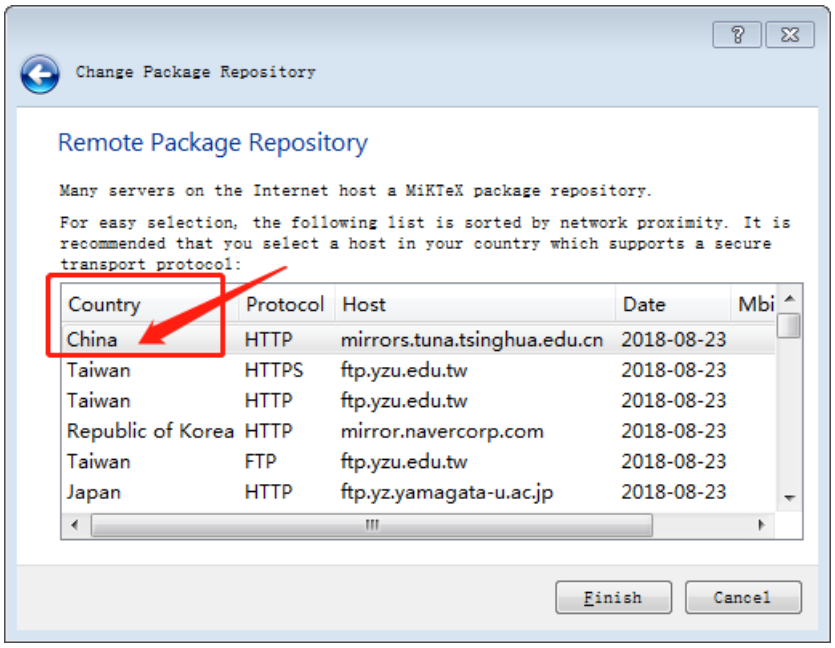


Figure 3.6: 安装缺失文件：选择安装源

选择好后，点击安装即可。

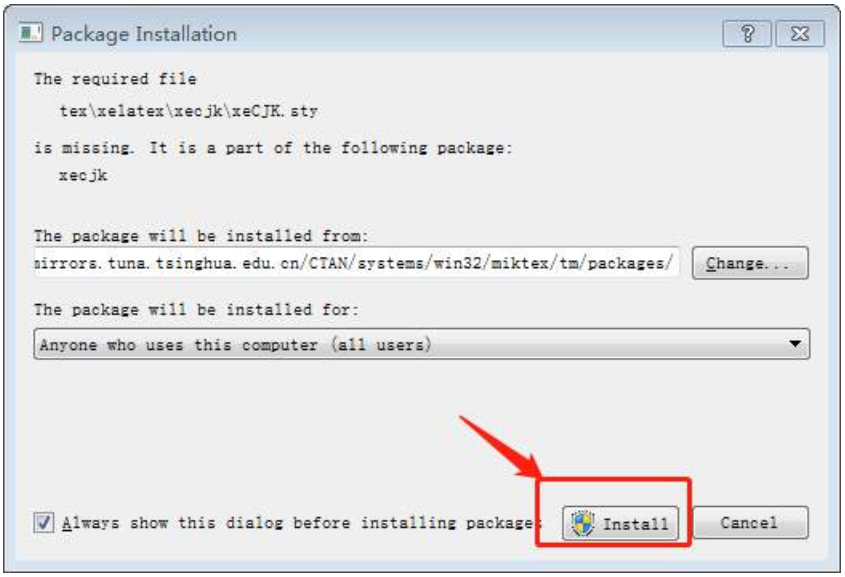


Figure 3.7: 安装缺失文件

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

4 Anaconda 安装虚拟环境到指定路径

4.1 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：

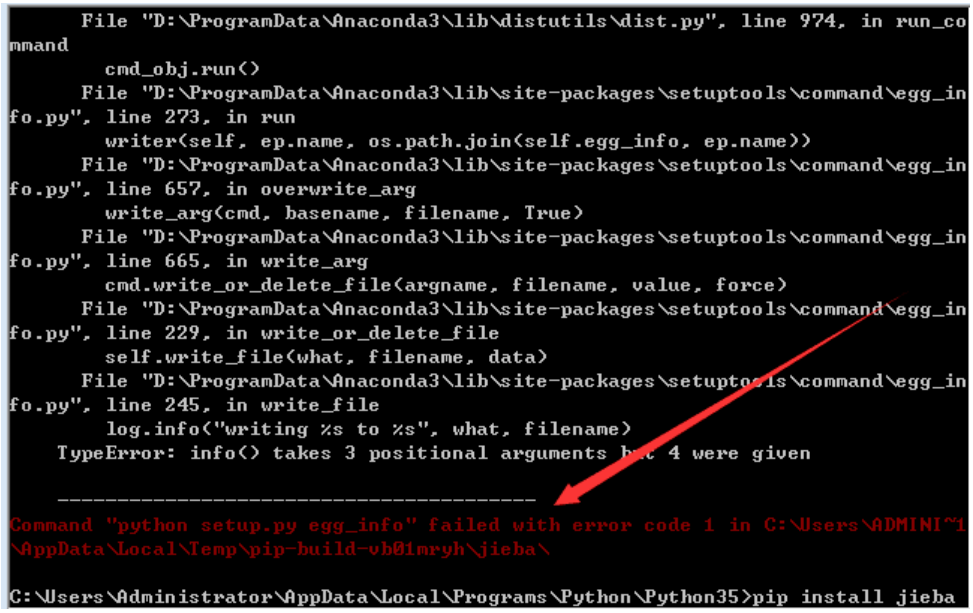


Figure 4.1: 出现的问题

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）。

4.2 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：

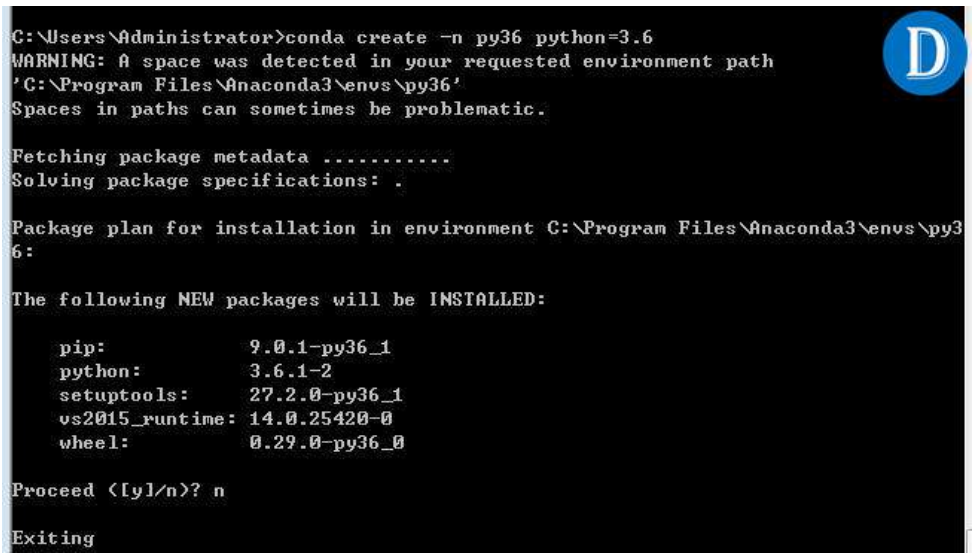


Figure 4.2: 安装演示

“-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
```

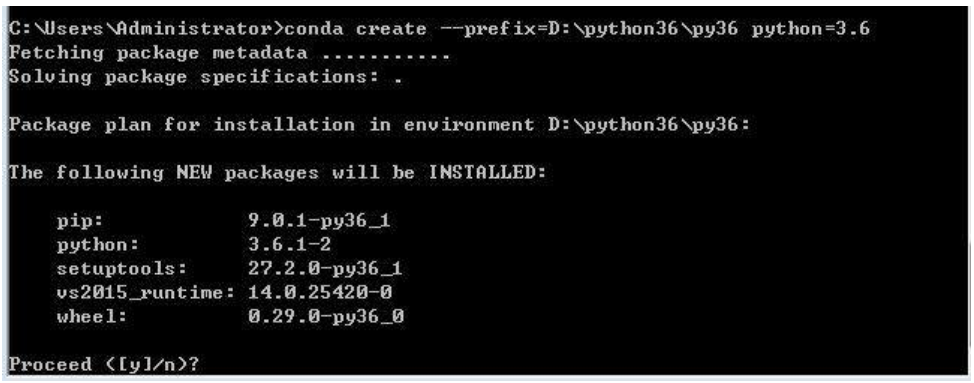


Figure 4.3: 安装到指定路径

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
```

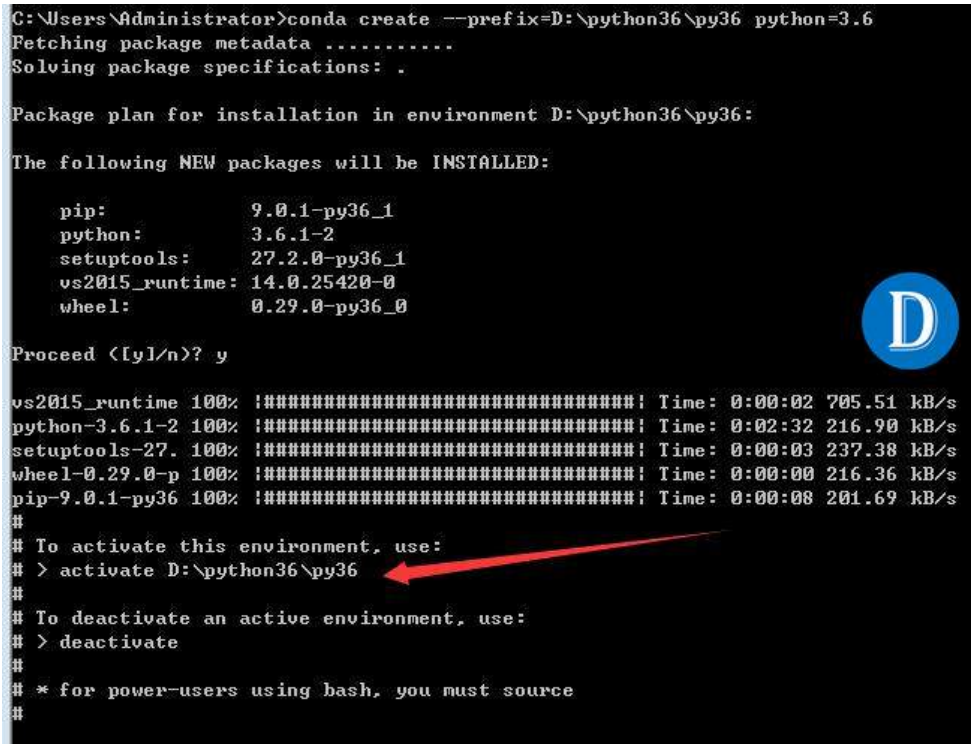


Figure 4.4: 退出虚拟环境

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
```

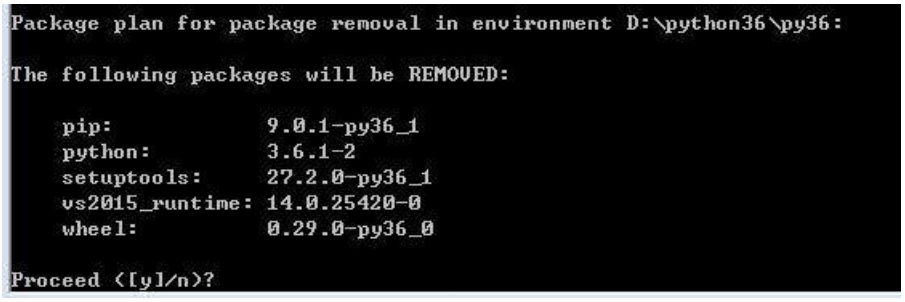


Figure 4.5: 删除指定路径下的虚拟环境

4.3 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
```

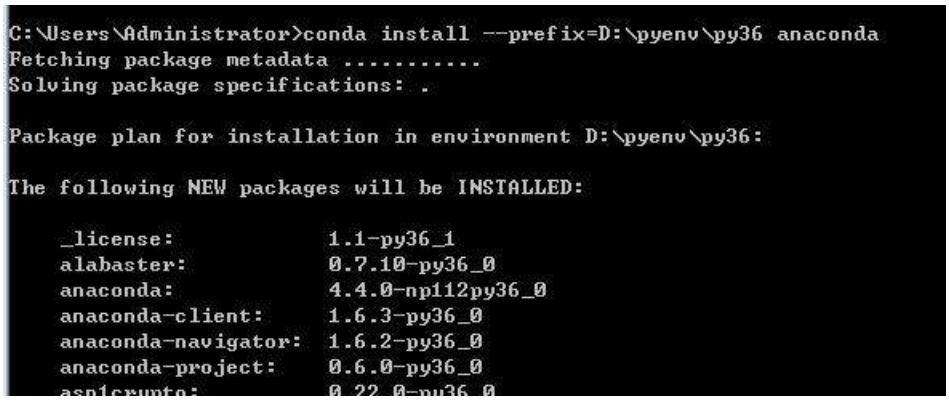


Figure 4.6: 安装 Anaconda 中所有的库

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。

5 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

5.1 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）

5.2 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
```

```
# 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。

5.3 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 5.1: 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 5.2: Error

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

5.4 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 5.3: ndarray 各种属性小结

5.5 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：

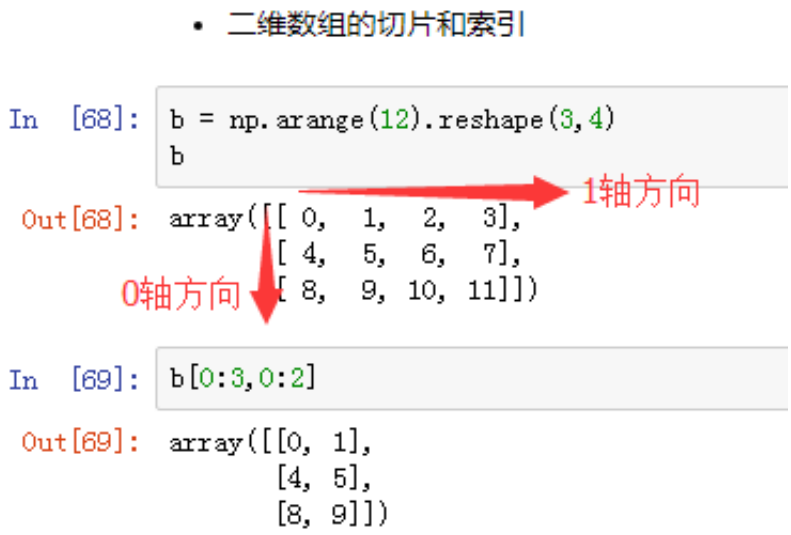


Figure 5.4: 二维数组的切片和索引

5.6 处理数组形状

5.6.1 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]])
```

```
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：

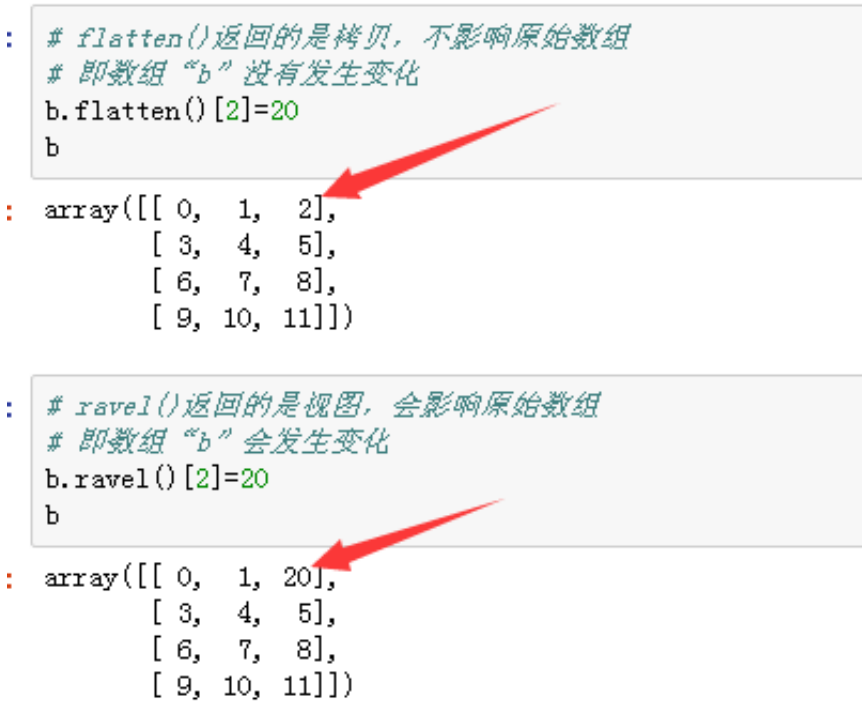


Figure 5.5: 拷贝 vs 视图

- **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]])
```

5.6.2 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],
            [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

```
np.concatenate((b,c),axis=1)

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]])
```

```
np.concatenate((b,c),axis=0)

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]])
```

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 5.6: 数组的轴方向示意图

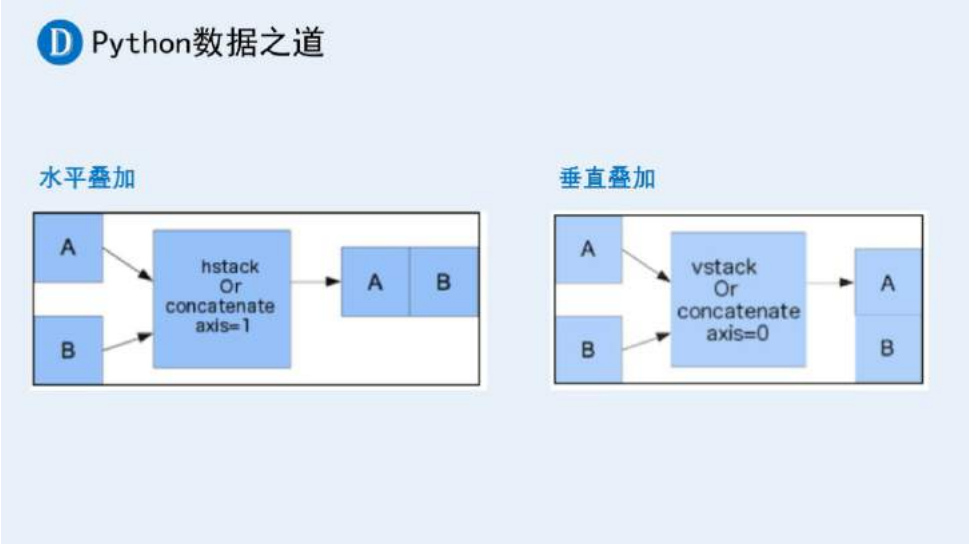


Figure 5.7: 叠加的示意图

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：

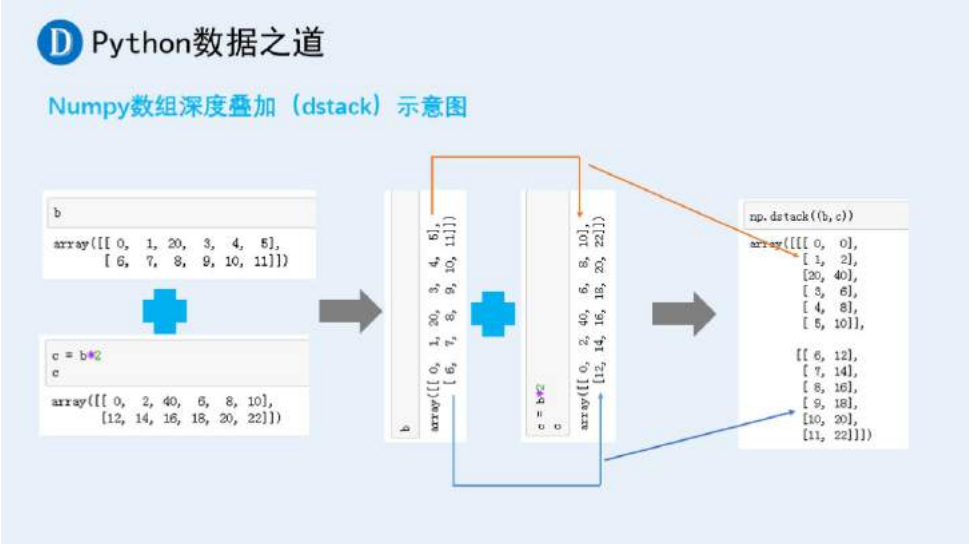


Figure 5.8: 深度叠加的示意图

5.6.3 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]],
```

```
22
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 ~~

5.7 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.]])
```

5.8 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)
```

5.9 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]])
```

5.10 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：PyDataRoad）后台回复关键字“code”。

6 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
```

6.1 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]]])
```

6.2 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）。

6.3 numpy.random.randint()

6.3.1 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]])
```

6.3.2 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])
```

6.4 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]]
```

6.5 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))

```

```
Out[]:
    array([['moto', 'iphone', 'xiaomi'],
           ['lenovo', 'xiaomi', 'xiaomi'],
           ['xiaomi', 'lenovo', 'iphone']],
          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

```
demo_list = ['lenovo', 'sansumg','moto','xiaomi', 'iphone']
np.random.choice(demo_list,size=(3,3), p=[0.1,0.6,0.1,0.1,0.1])

Out[]:
    array([['sansumg', 'sansumg', 'sansumg'],
           ['sansumg', 'sansumg', 'sansumg'],
           ['sansumg', 'xiaomi', 'iphone']],
          dtype='<U7')
```

6.6 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

```
np.random.seed(0)
np.random.rand(5)

Out[]:
    array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
```

```
np.random.seed(1676)
np.random.rand(5)

Out[]:
    array([ 0.39983389,  0.29426895,  0.89541728,  0.71807369,  0.3531823 ])
```

```
np.random.seed(1676)
np.random.rand(5)

Out[]:
    array([ 0.39983389,  0.29426895,  0.89541728,  0.71807369,  0.3531823 ])
```

7 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。

7.1 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.  ])
```

```
y

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

查看矩阵 X 和矩阵 Y

```
X

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

Y

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

查看矩阵对应的维度

```
X.shape

out:
(3, 5)

Y.shape

out:
(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：

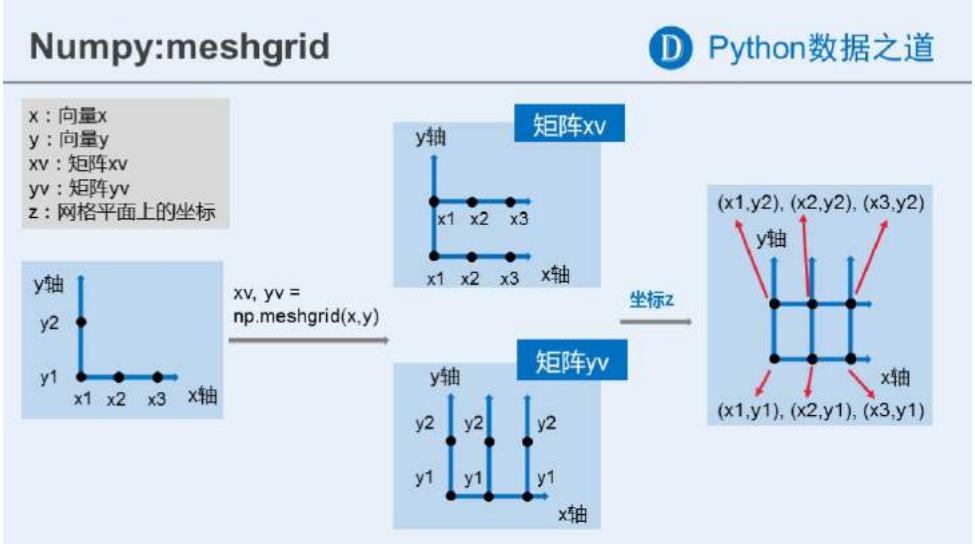


Figure 7.1: meshgrid

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()
```

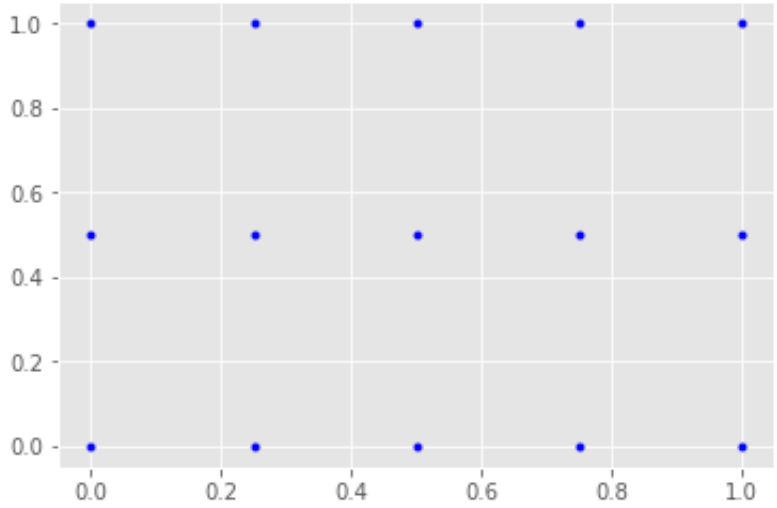


Figure 7.2: result

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)]
```

7.2 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

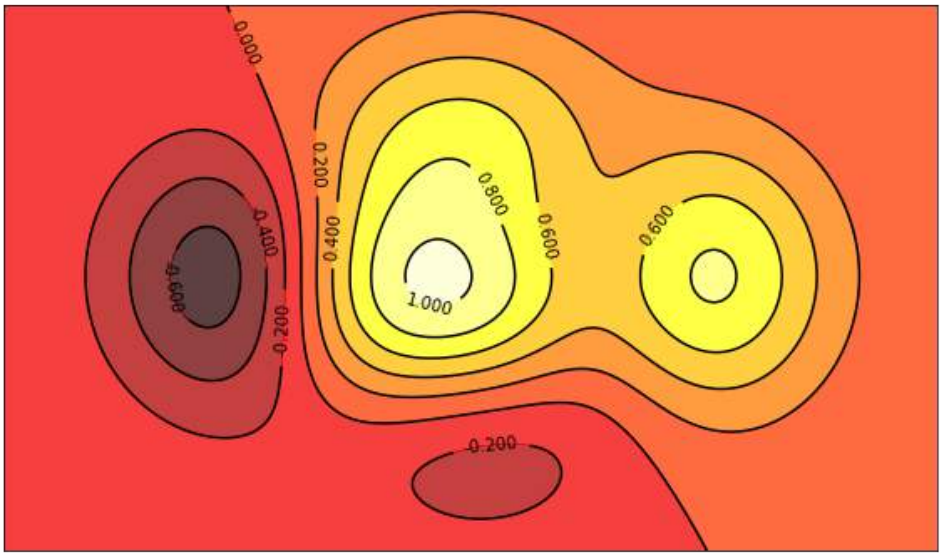


Figure 7.3: contour

（2）SVC 中超平面的绘制：

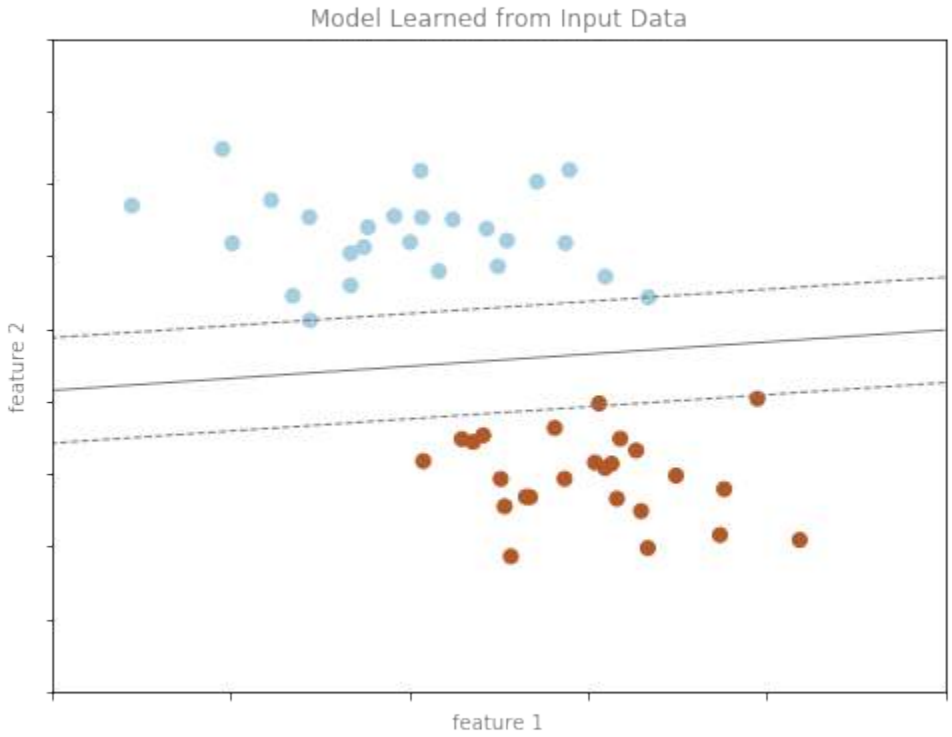


Figure 7.4: svc

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。

8 Pandas 日期数据处理

原文标题：Pandas 日期数据处理：如何按日期筛选、显示及统计数据

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。

8.1 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
```

8.2 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dataframe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
```

8.3 按日期显示数据

8.3.1 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
```

8.3.2 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：

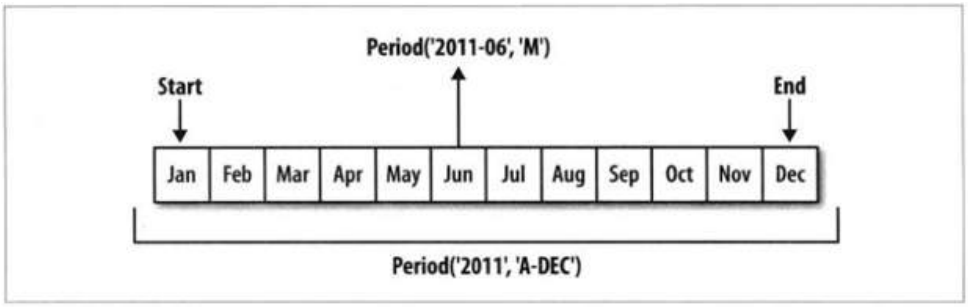


Figure 8.1

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：

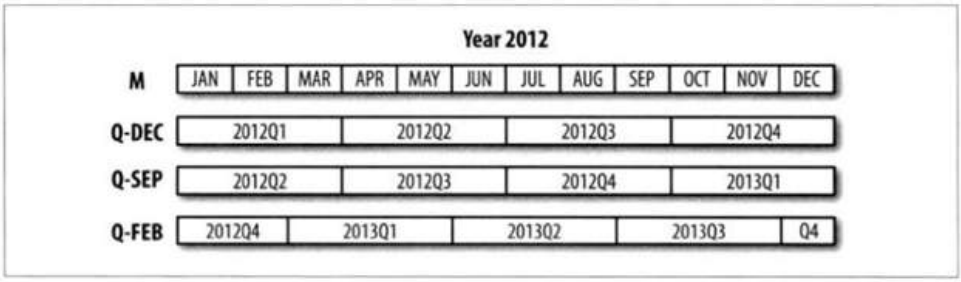


Figure 8.2

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')
```

8.4 按日期统计数据

8.4.1 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

```

print(df.resample('AS').sum())
# ”AS”是每年第一天为开始日期, ”A是每年最后一天
```

```
            number
date
2013-01-01      51
2014-01-01     453
2015-01-01     743
2016-01-01    1552
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 8.3

8.4.2 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
```

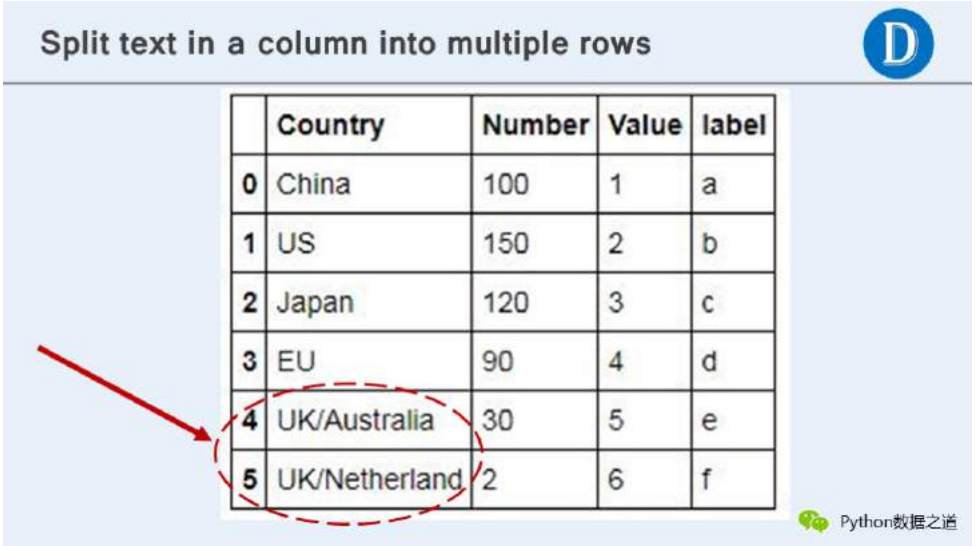
按月度统计并显示

```
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
```

9 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 9.1: 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
```

9.1 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 1. 用drop()方法从 DataFrame 中删除含有多值的列 1. 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
```

9.2 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 ~~

10 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：

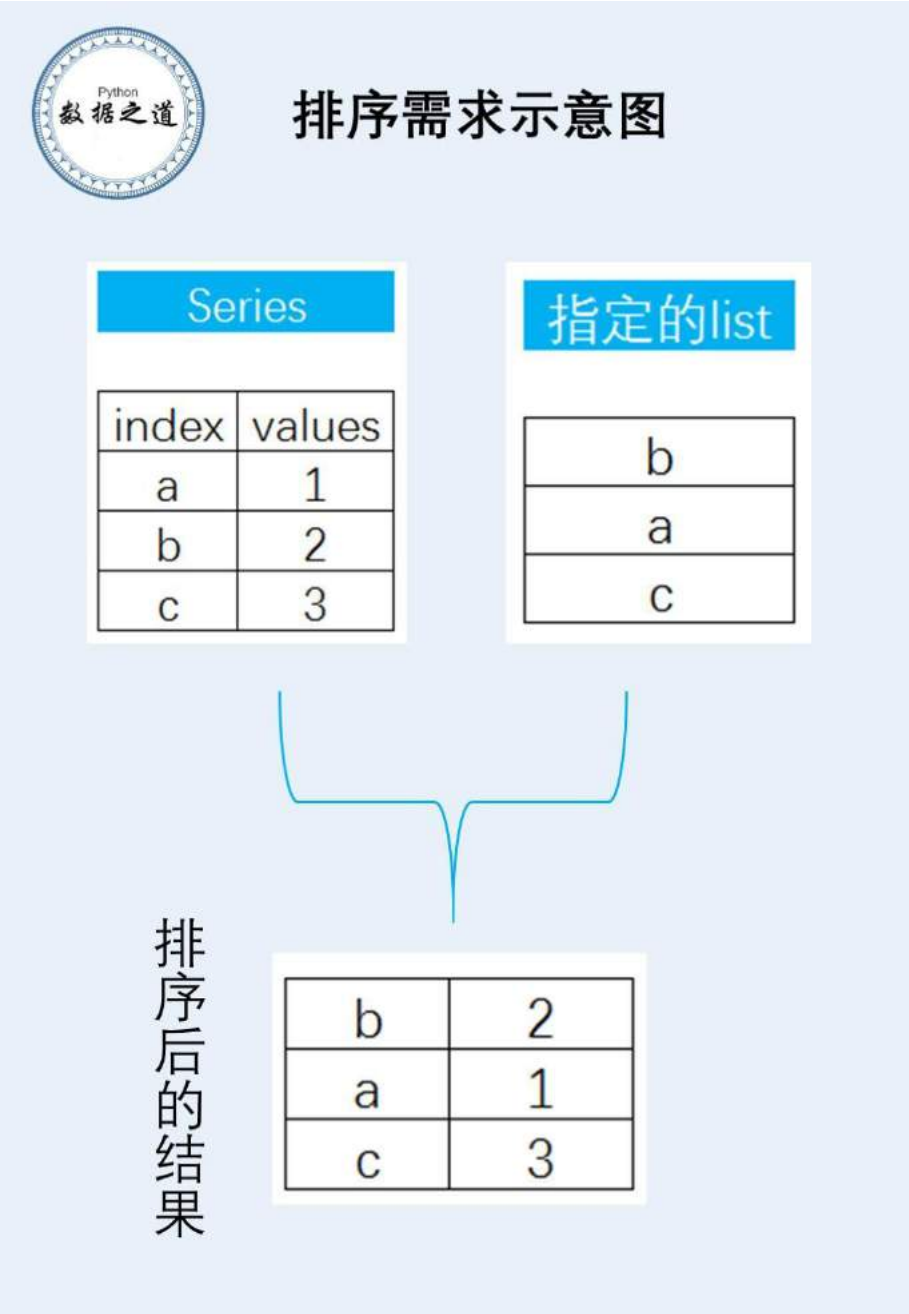


Figure 10.1: 排序需求示意图

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：

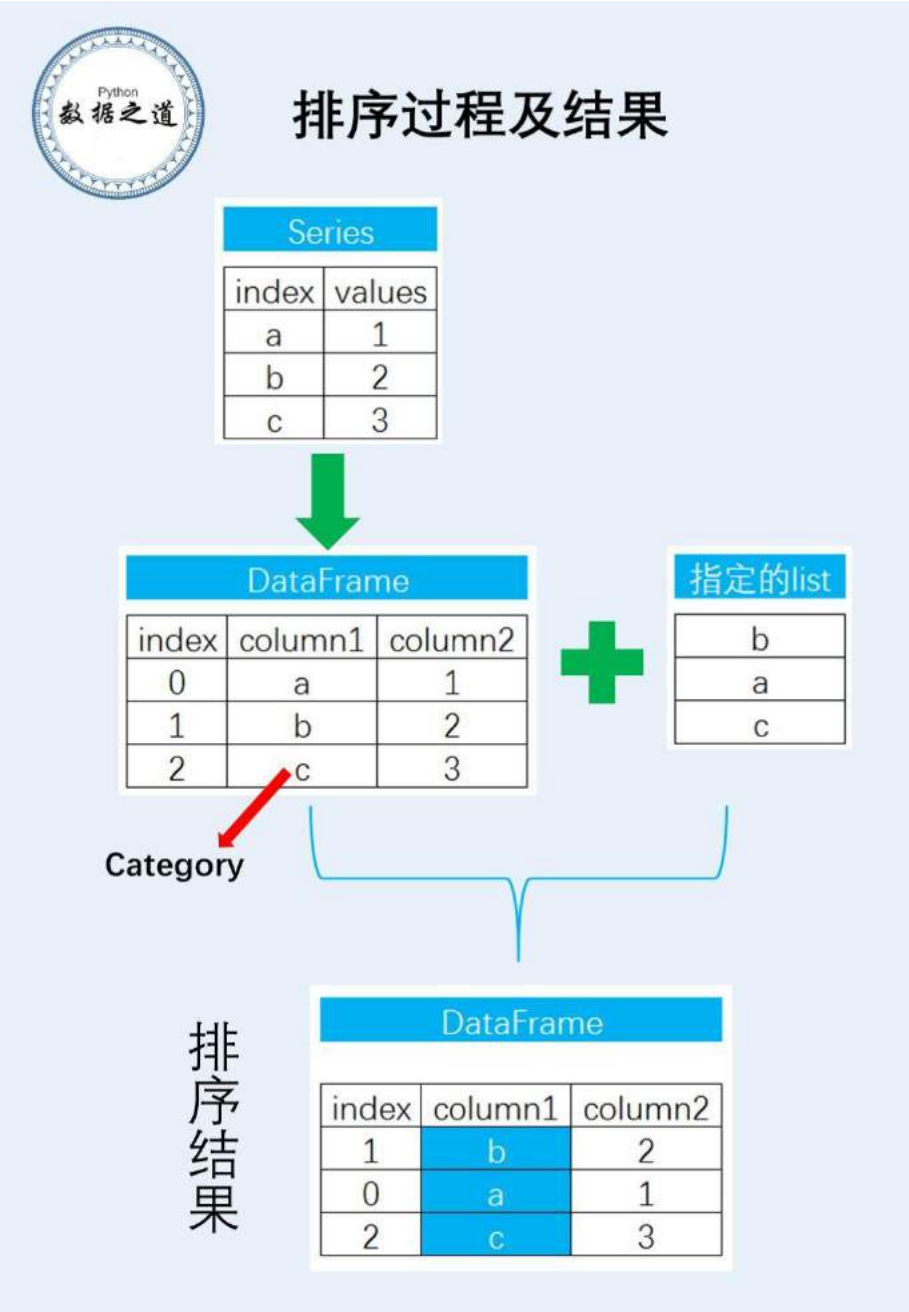


Figure 10.2: 排序过程及结果

10.1 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 10.3

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 10.4

10.2 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 10.5

```
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 10.6

10.3 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 10.7

```
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 10.8

10.4 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：PyDataRoad）后台回复关键字“code”获取。

11 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。

11.1 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
```

11.2 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：

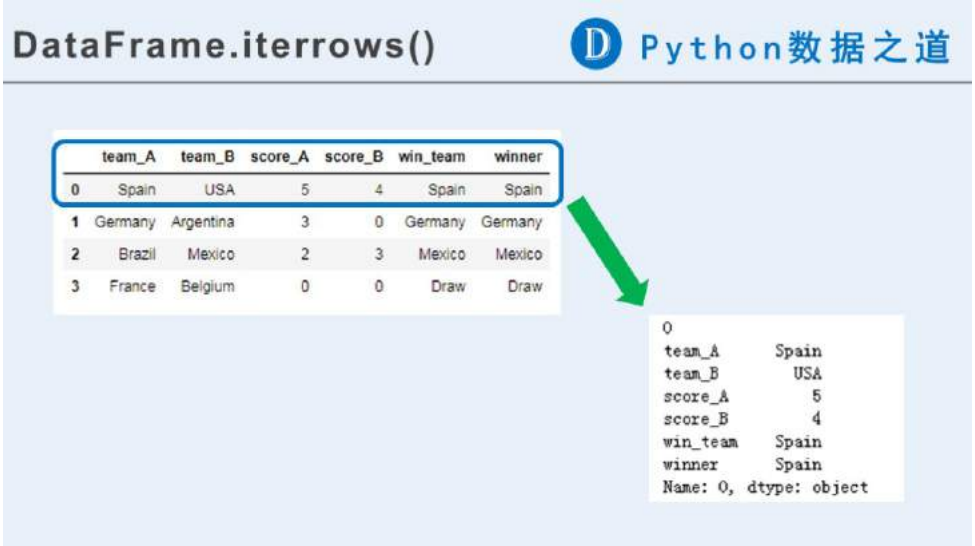


Figure 11.1: iterrows() 的作用

12 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')
```

12.1 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()
```

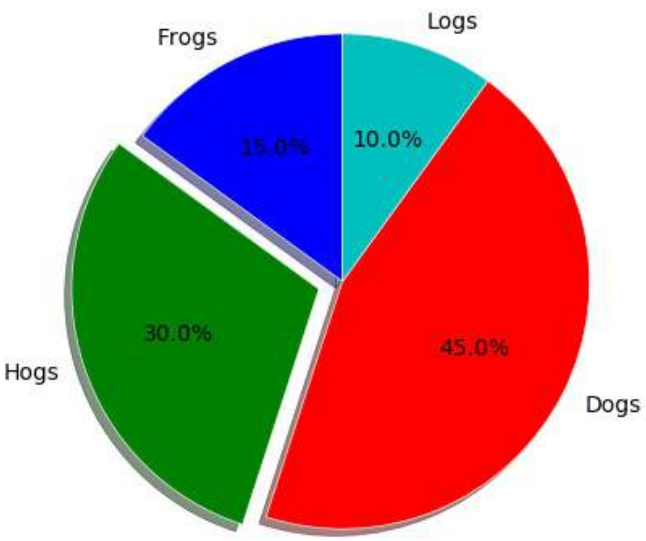


Figure 12.1

12.2 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)
```

```
12 ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
13
14 plt.savefig('Demo_project.jpg')
15 plt.show()
```

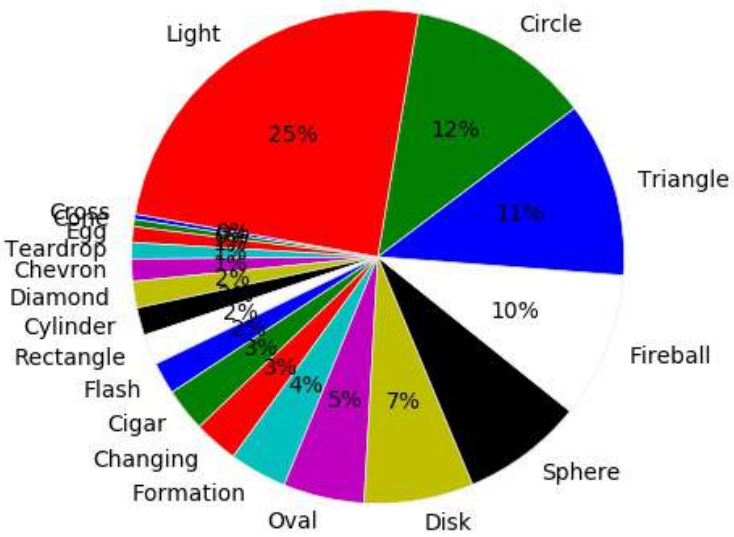


Figure 12.2

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

12.3 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

12.3.1 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
```

```
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.

# 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_font.jpg')
plt.show()
```

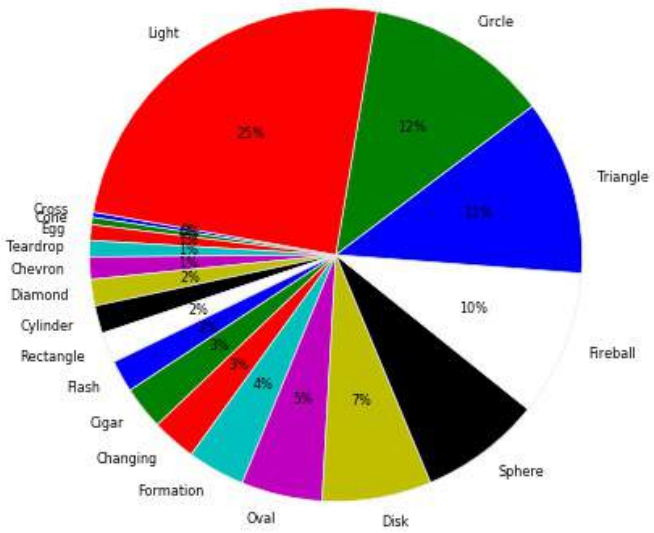


Figure 12.3

12.3.2 设置显示颜色，Method 1：

```
from matplotlib import font_manager as fm
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)  # only ”explode” the 1st
       slice
8
9  fig1, ax1 = plt.subplots(figsize=(6,6)) # 设置绘图区域大小
10
11 a = np.random.rand(1,19)
12 color_vals = list(a[0])
13 my_norm = mpl.colors.Normalize(-1, 1) # 将颜色数据的范围设置为 [0, 1]
14 my_cmap = mpl.cm.get_cmap('rainbow', len(color_vals)) # 可选择合适的colormap,
       如：'rainbow'
15
16 patches, texts, autotexts = ax1.pie(sizes, explode=explode, labels=labels,
       autopct='%1.0f%%',
17         shadow=False, startangle=170, colors=my_cmap(my_norm(color_vals)))
18
19 ax1.axis('equal')
20
21 # 重新设置字体大小
22 proptease = fm.FontProperties()
23 proptease.set_size('xx-small')
24 # font size include: ‘xx-small’ ,x-small’ ,'small’ ,'medium’ , ‘large’ , ‘x-
       large’ , ‘xx-large’ or number, e.g. '12'
25 plt.setp(autotexts, fontproperties=proptease)
26 plt.setp(texts, fontproperties=proptease)
27
28 plt.savefig('Demo_project_set_color_1.jpg')
29 plt.show()
```

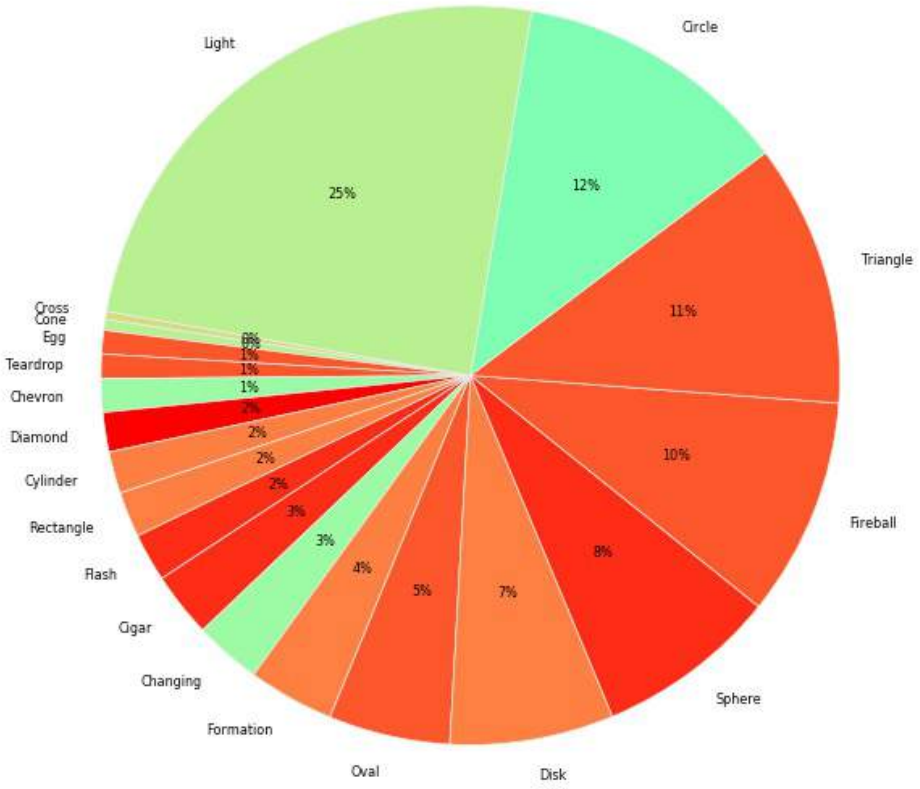


Figure 12.4

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。

12.3.3 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

```

```
10
11 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()
```

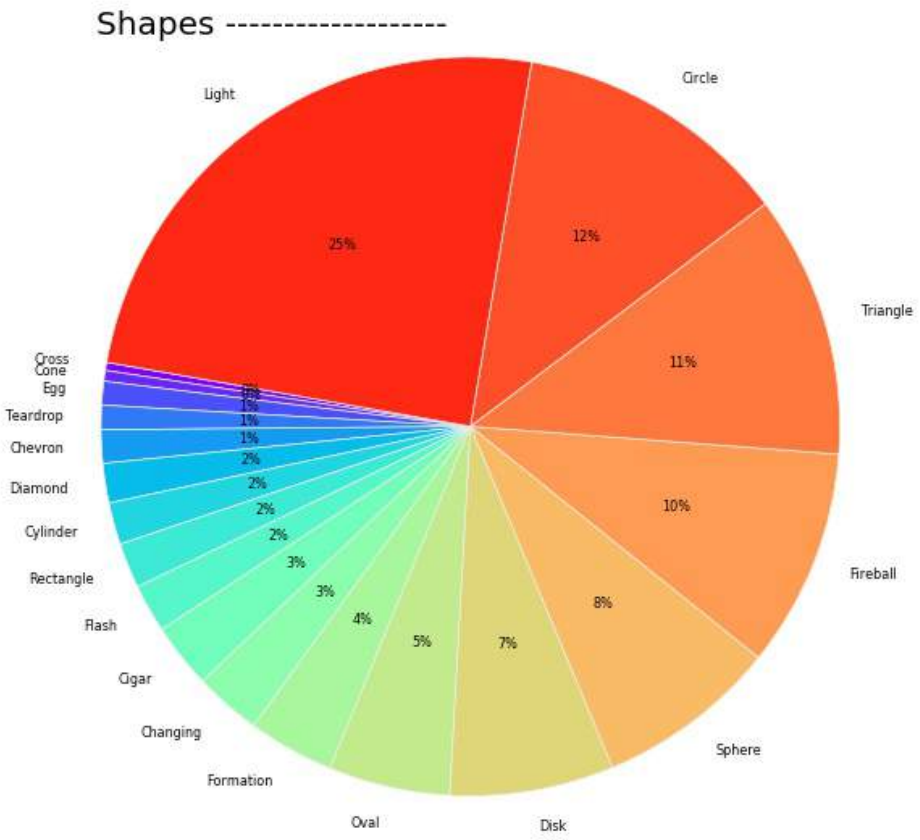


Figure 12.5

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

12.3.4 设置图例（legend）

```
from matplotlib import font_manager as fm
from  matplotlib import cm

labels = s.index
sizes = s.values
# explode = (0.2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)  # only "explode" the 1st
    slice

fig, ax = plt.subplots(figsize=(6,6)) # 设置绘图区域大小


```

```
11 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
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
25 ax.legend(labels, loc=2)
26
27 plt.savefig('Demo_project_set_legend_error.jpg')
28 plt.show()
```

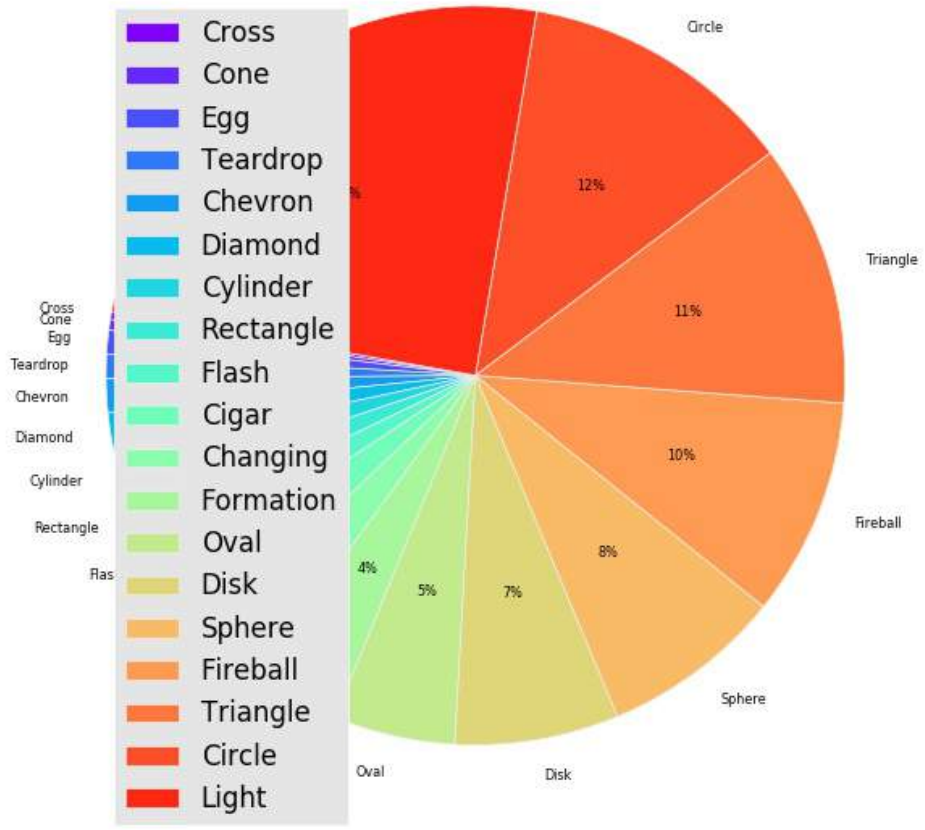


Figure 12.6

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。

12.3.5 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)  # 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)

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('Demo_project_set_legend_good.jpg')
plt.show()
```

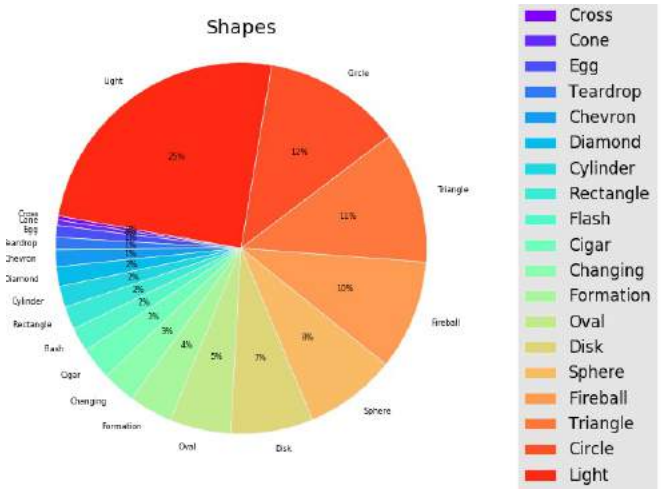


Figure 12.7

12.3.6 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

```
from matplotlib import font_manager as fm
from  matplotlib import cm

labels = s.index
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()
```

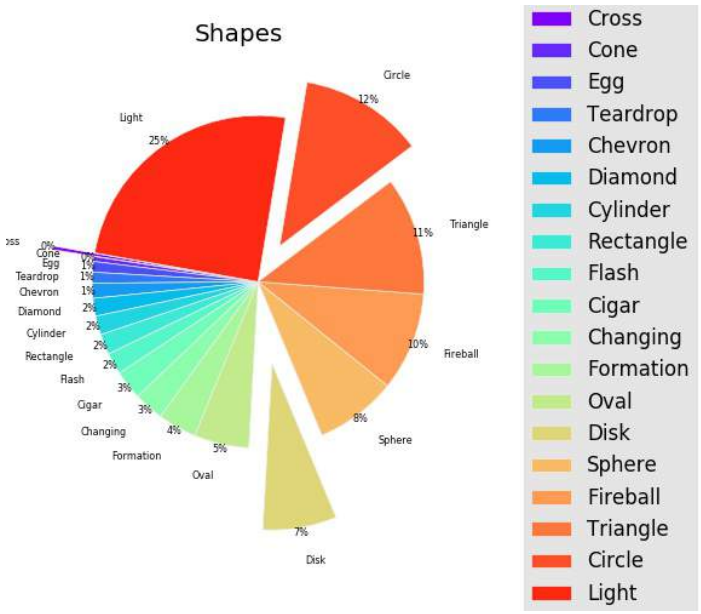


Figure 12.8

> 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：PyDataRoad）后台回复关键字“code”。

13 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。

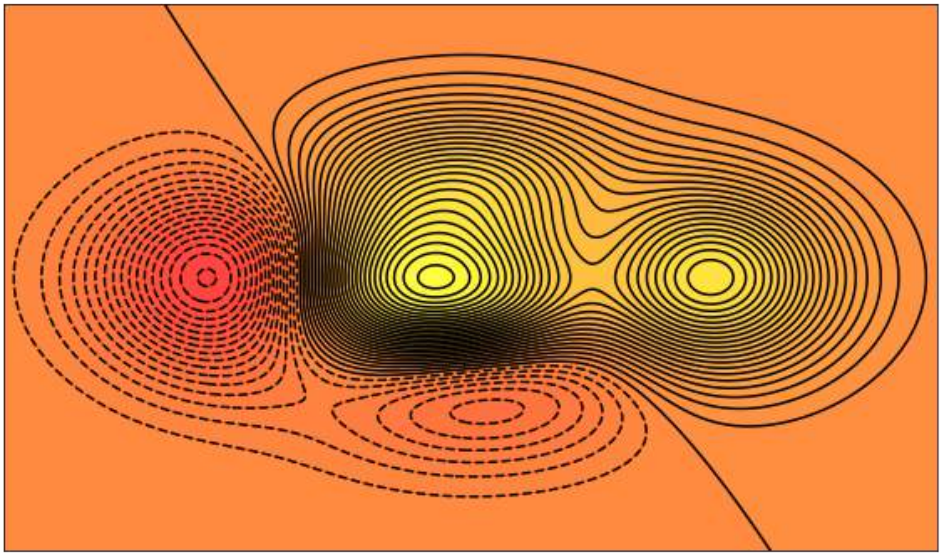


Figure 13.1: 等高线图的效果图

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

13.1 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：

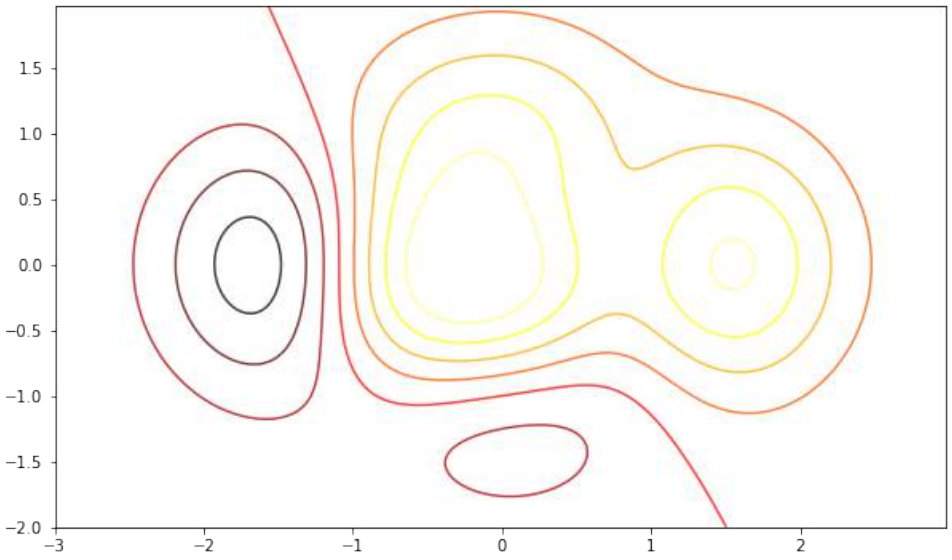


Figure 13.2

13.2 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：

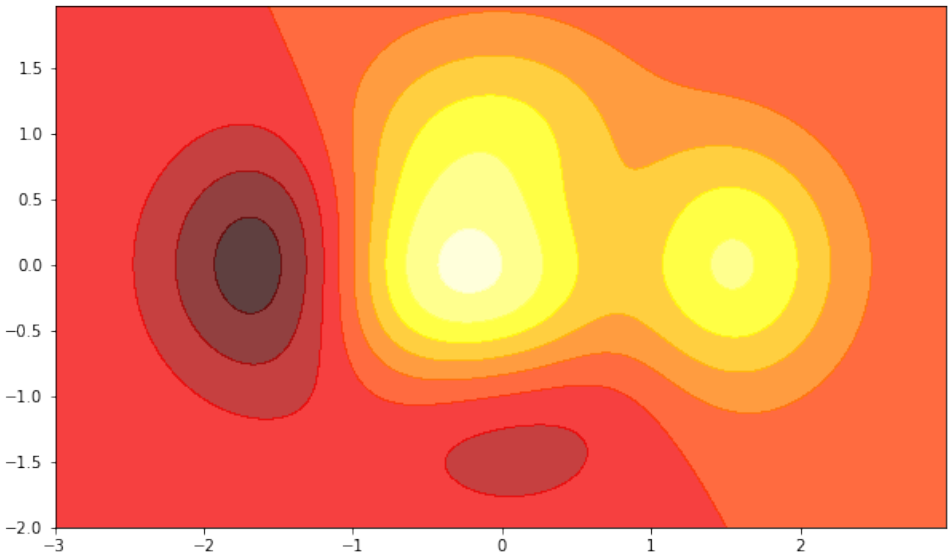


Figure 13.3

13.3 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：

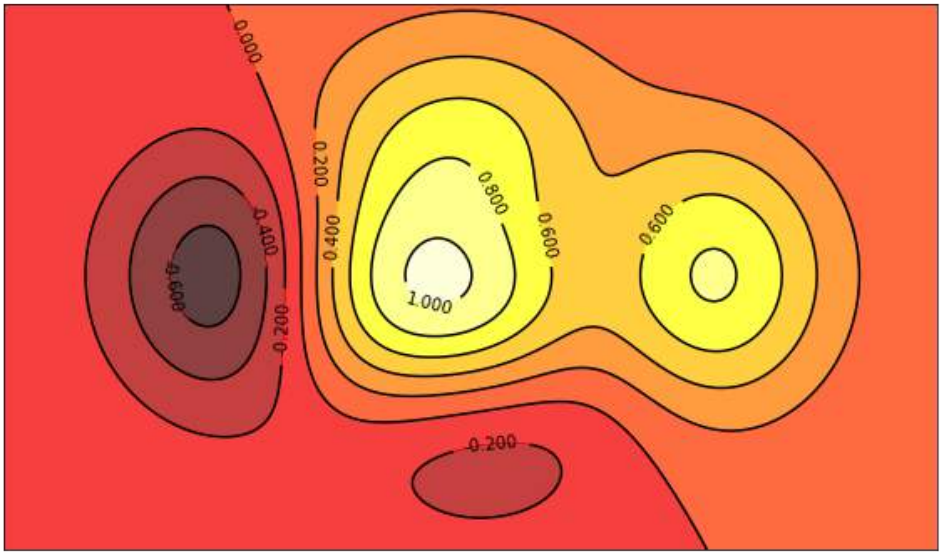


Figure 13.4

是不是很容易实现啊，赶快动手来试试吧。

14 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。

14.1 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。

14.2 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
```

14.3 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。

14.3.1 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')
```

```
11
12 for i, category in enumerate(categories):
13     plt.scatter('area', 'poptotal',
14                 data=midwest.loc[midwest.category==category, :],
15                 s=20, cmap=colors[i], label=str(category))
16     # ”c=” 修改为 ”cmap=”, Python数据之道 备注
17
18 # Decorations
19 plt.gca().set(xlim=(0.0, 0.1), ylim=(0, 90000),
20               xlabel='Area', ylabel='Population')
21
22 plt.xticks(fontsize=12); plt.yticks(fontsize=12)
23 plt.title(”Scatterplot of Midwest Area vs Population”, fontsize=22)
24 plt.legend(fontsize=12)
25 plt.show()
```

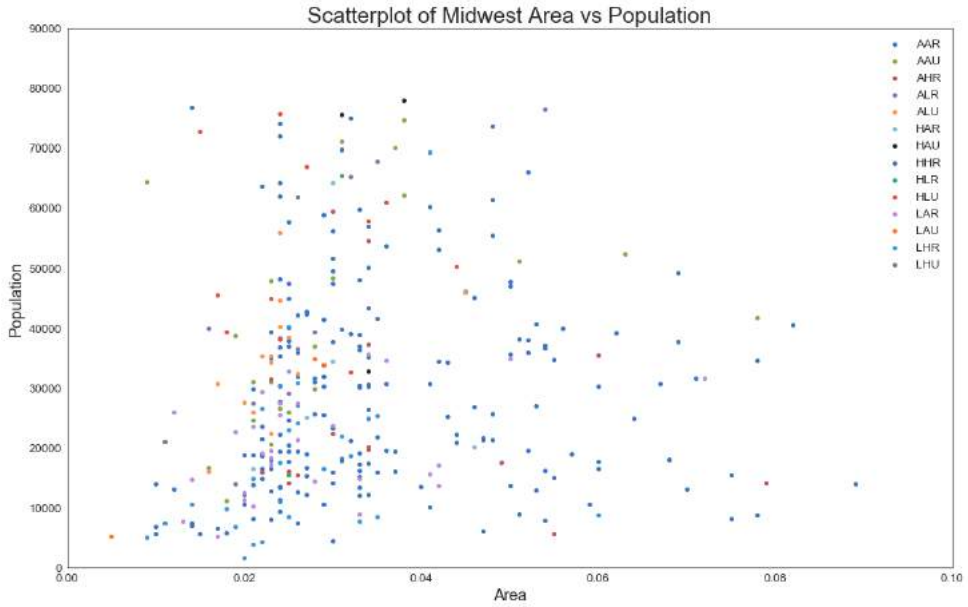


Figure 14.1: 散点图

14.3.2 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。

```
1 from matplotlib import patches
2 from scipy.spatial import ConvexHull
3 import warnings; warnings.simplefilter('ignore')
4 sns.set_style(”white”)
5
6 # 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()
```

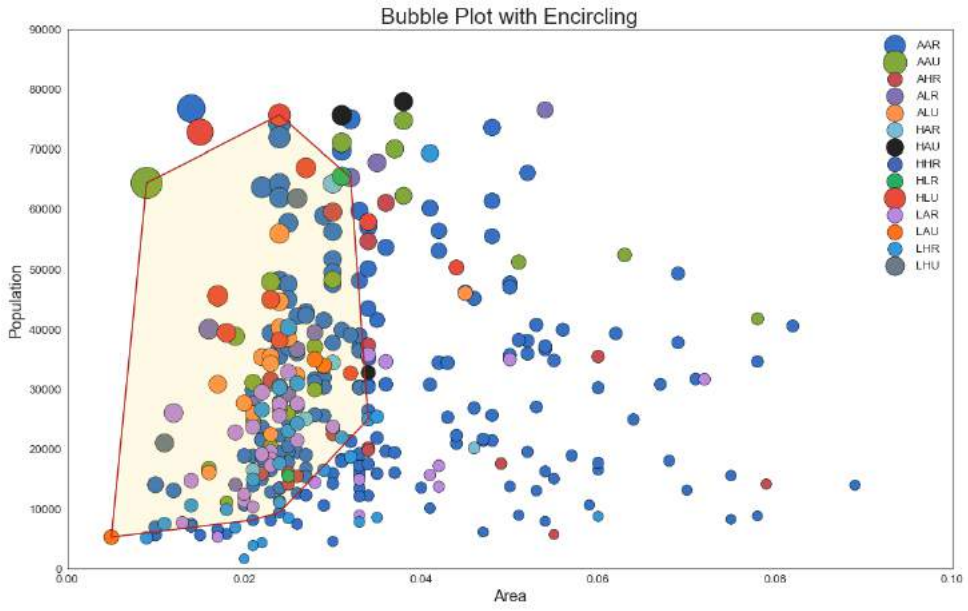


Figure 14.2: 气泡图

14.3.3 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()
```

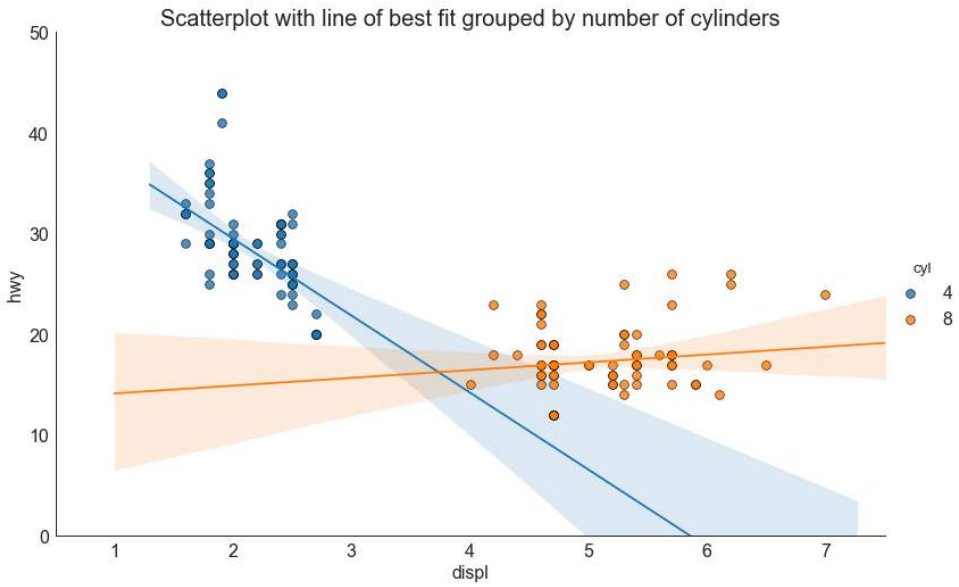


Figure 14.3: 带线性回归最佳拟合线的散点图

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()
```

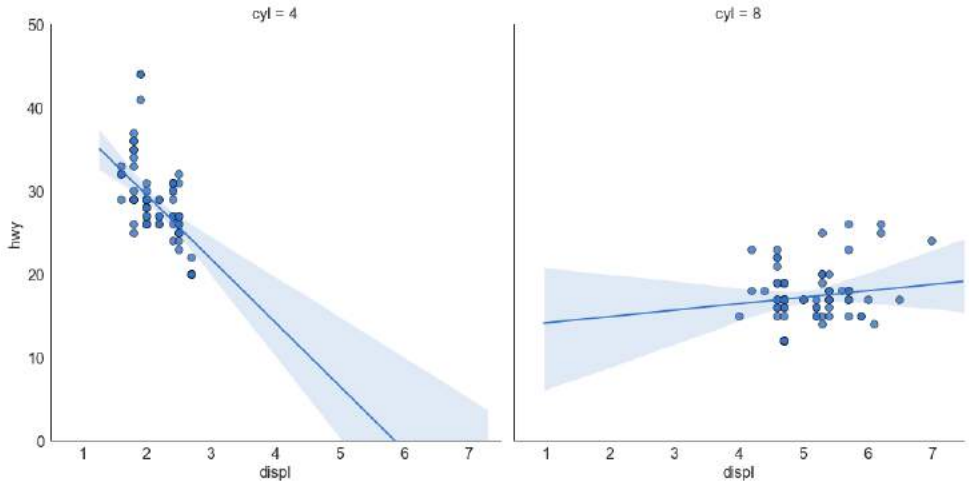


Figure 14.4: 带线性回归最佳拟合线的散点图

14.3.4 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()
```

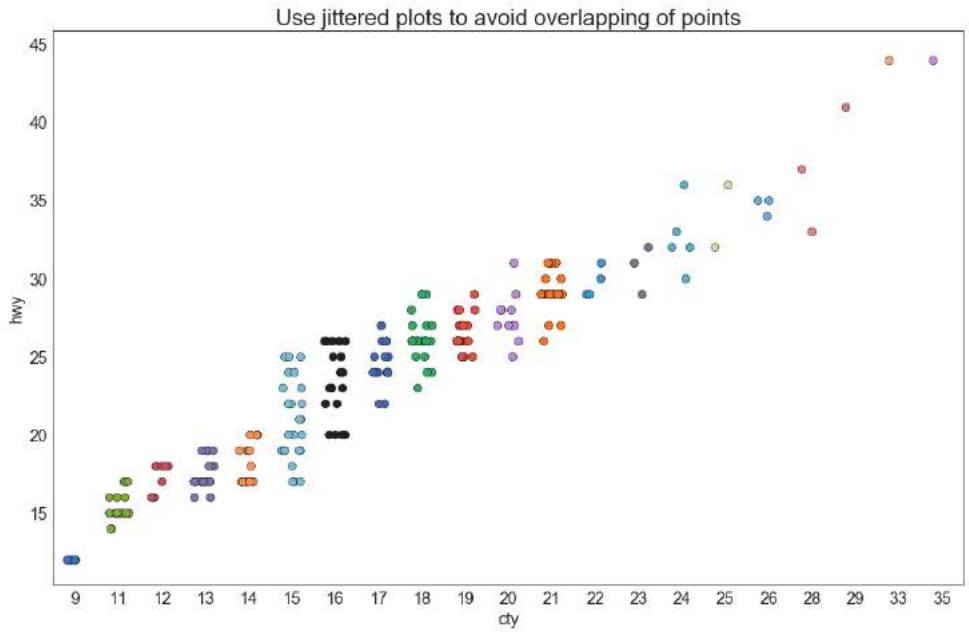


Figure 14.5: 抖动图

14.3.5 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()
```

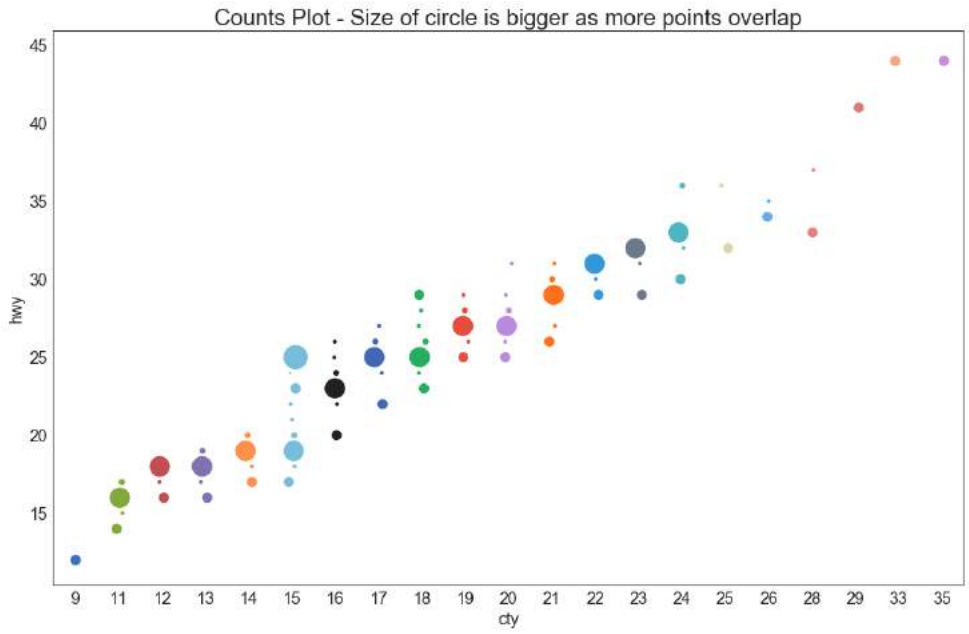


Figure 14.6: 计数图

14.3.6 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()
```

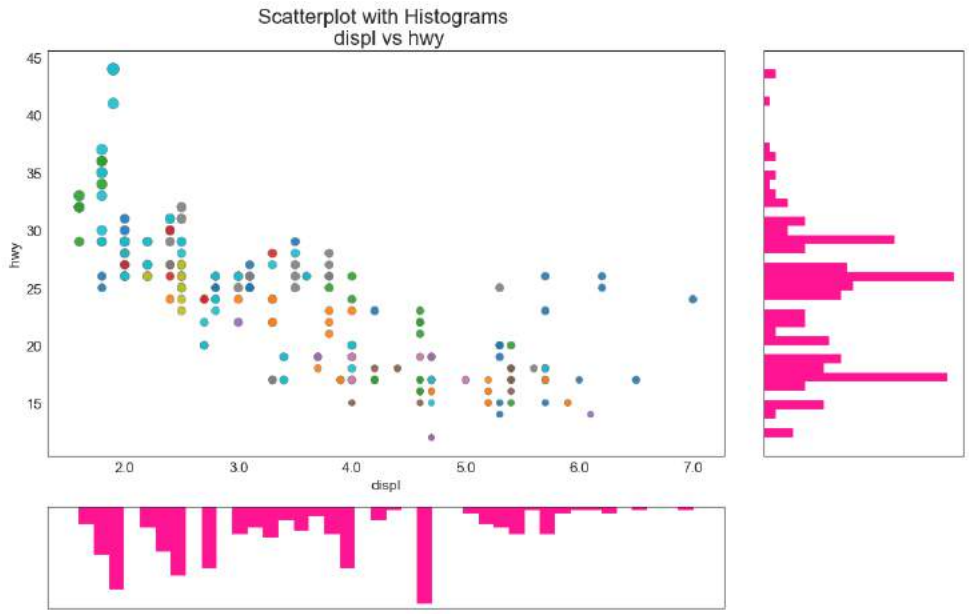


Figure 14.7: 边缘直方图

14.3.7 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()
```

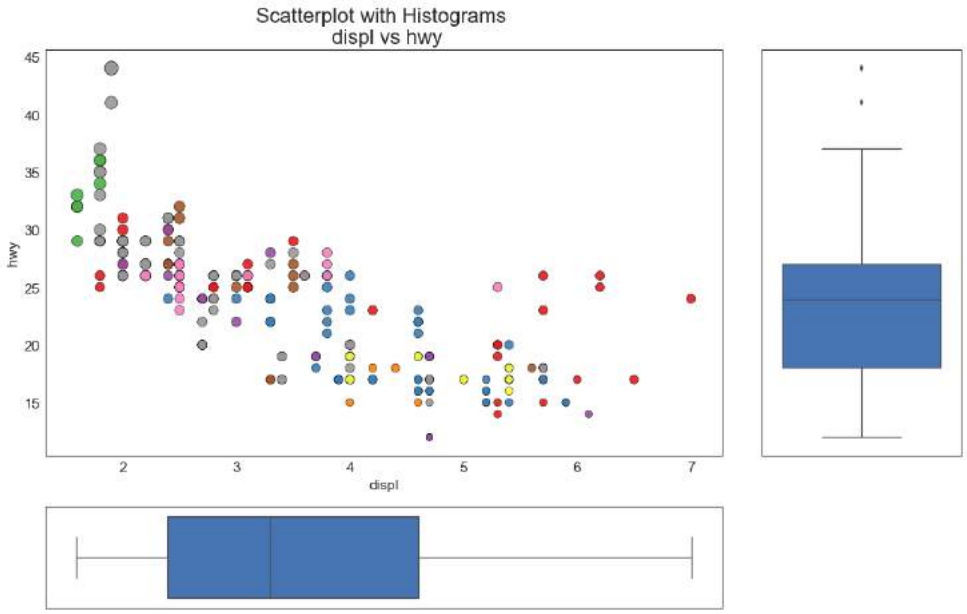


Figure 14.8: 边缘箱形图

14.3.8 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()
```

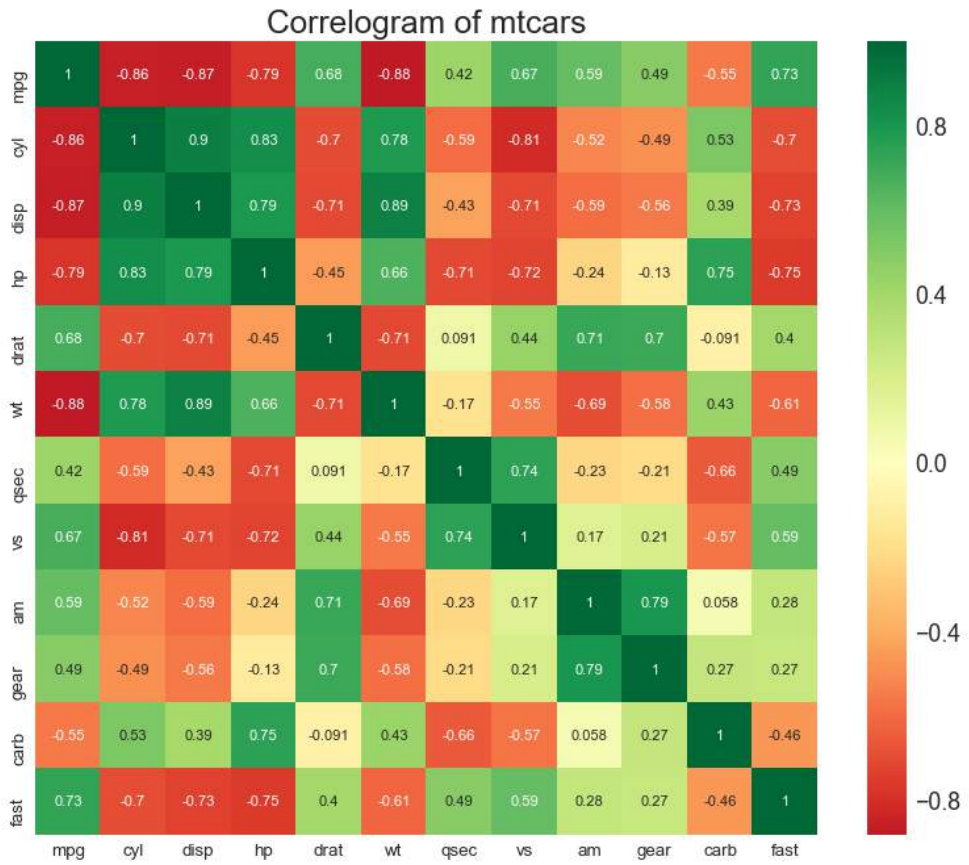


Figure 14.9: 相关图

14.3.9 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()
```

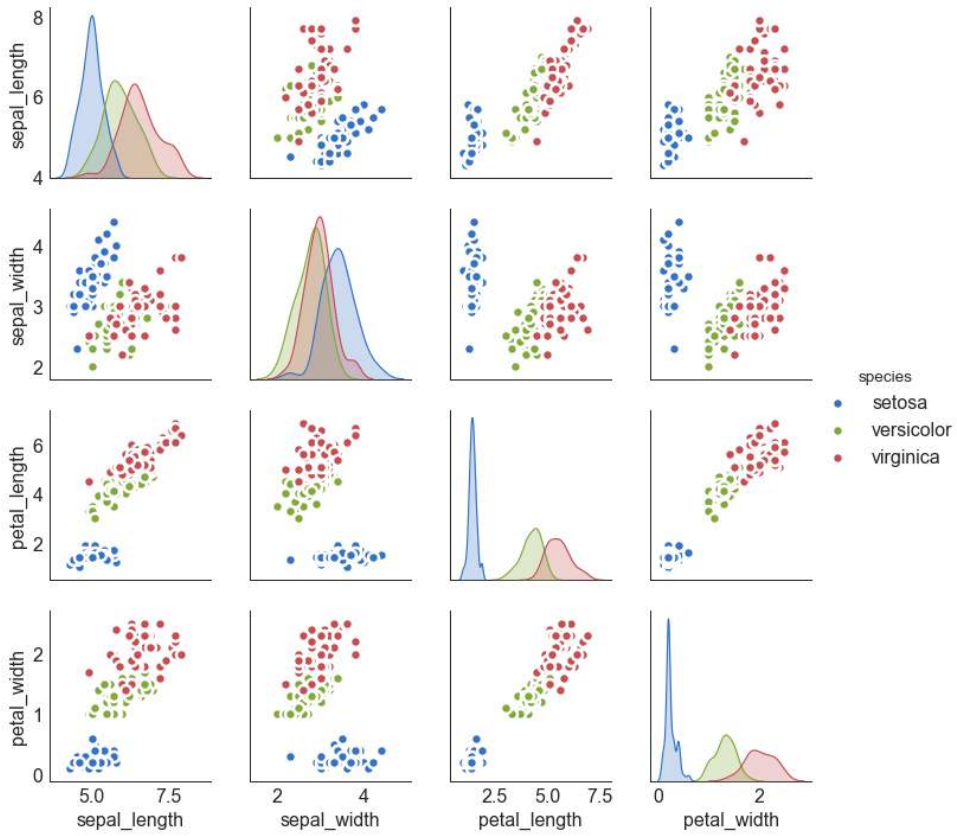


Figure 14.10: 矩阵图

```
# Load Dataset
df = sns.load_dataset('iris')

# Plot
plt.figure(figsize=(10,8), dpi= 80)
sns.pairplot(df, kind="reg", hue="species")
plt.show()
```

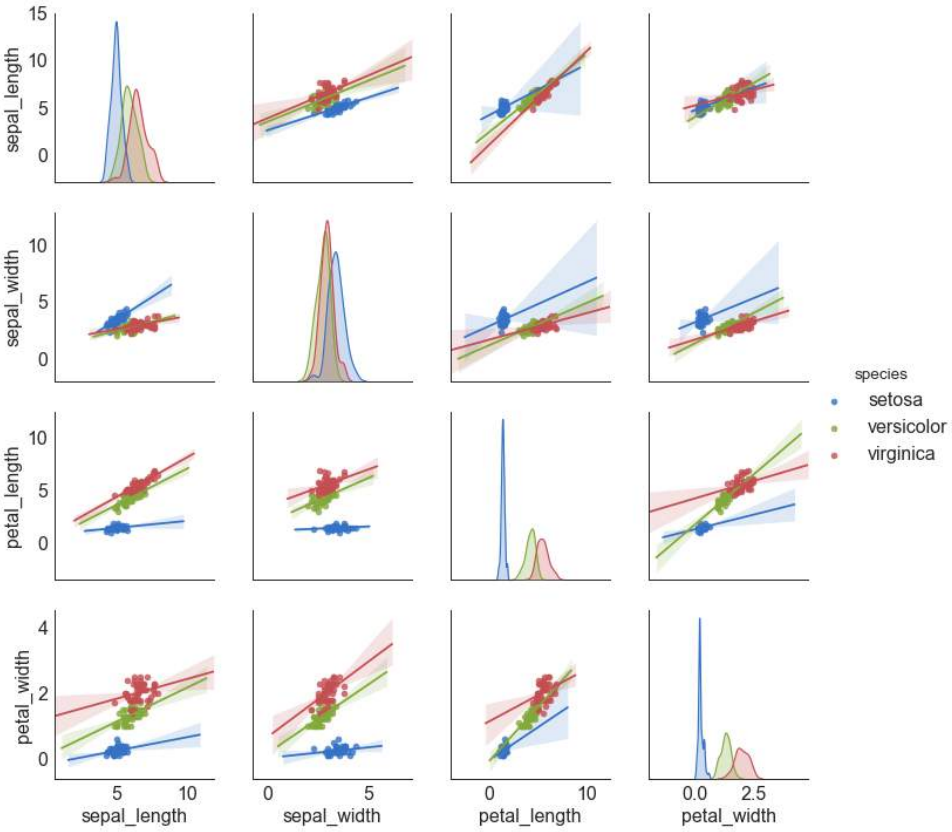


Figure 14.11: 矩阵图

14.4 偏差（Deviation）

14.4.1 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()
```

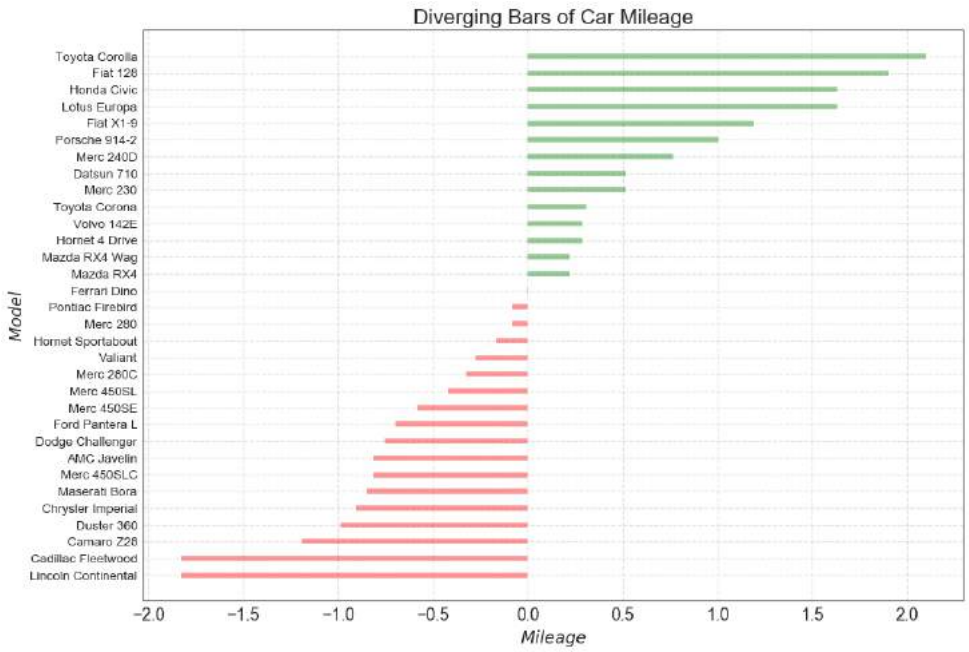


Figure 14.12: 发散型条形图

14.4.2 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()
```

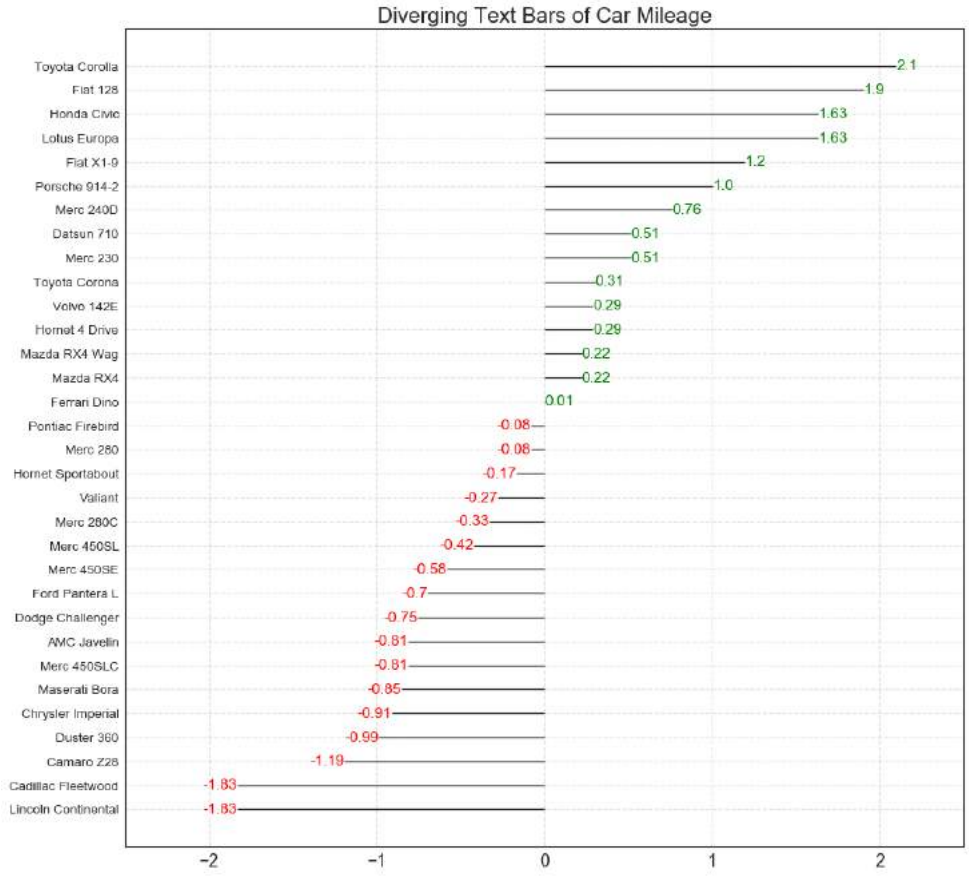


Figure 14.13: 发散型文本

14.4.3 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()
```

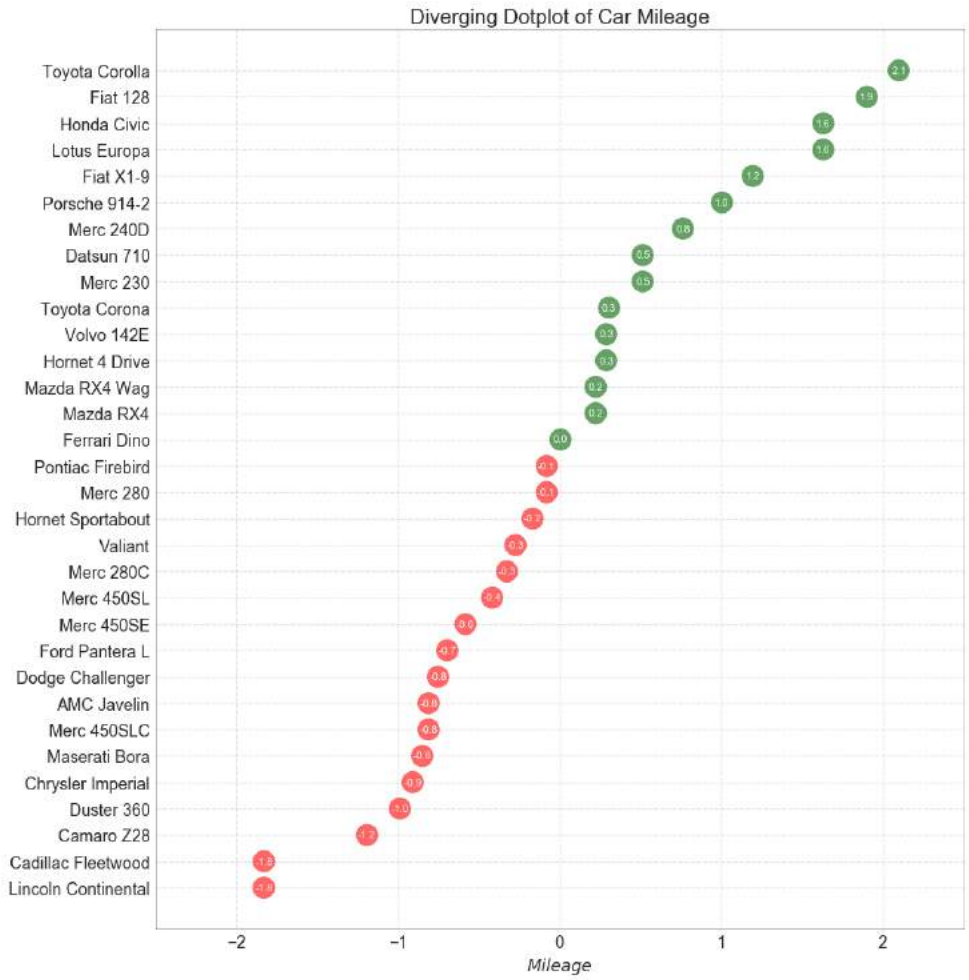


Figure 14.14: 发散型包点图

14.4.4 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()
```

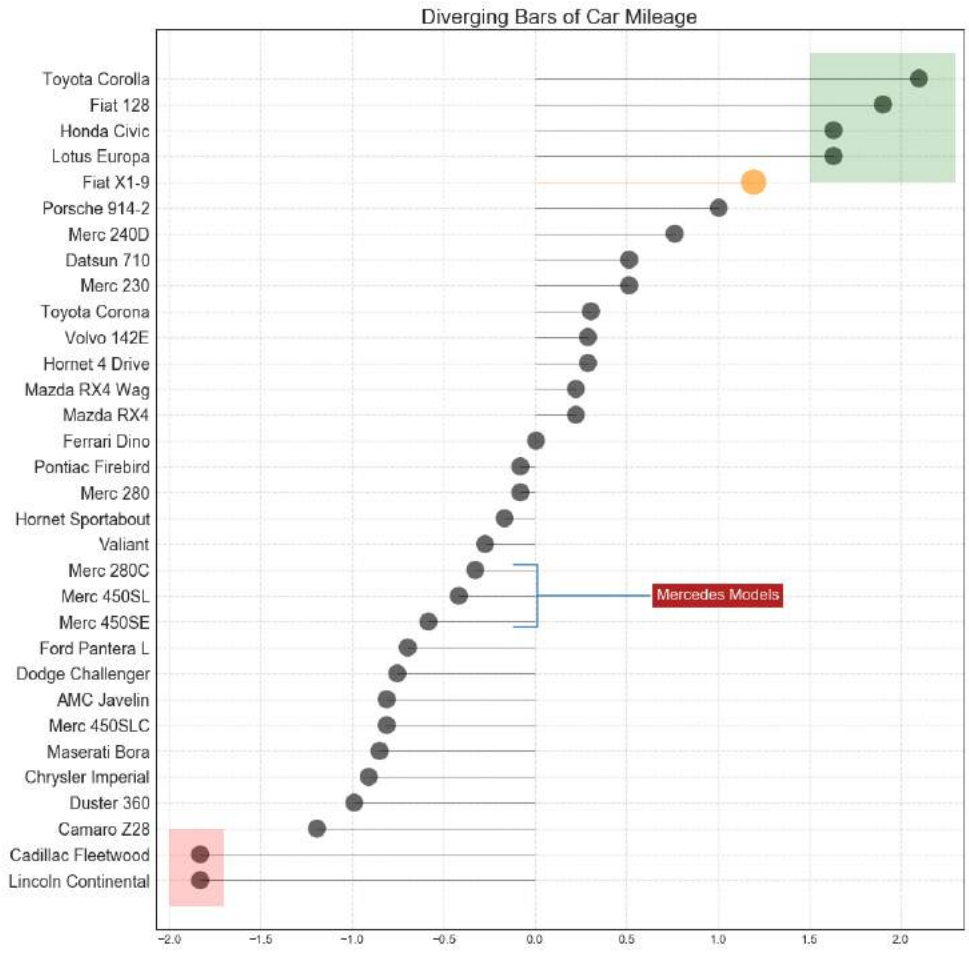


Figure 14.15: 带标记的发散型棒棒糖图

14.4.5 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()
```

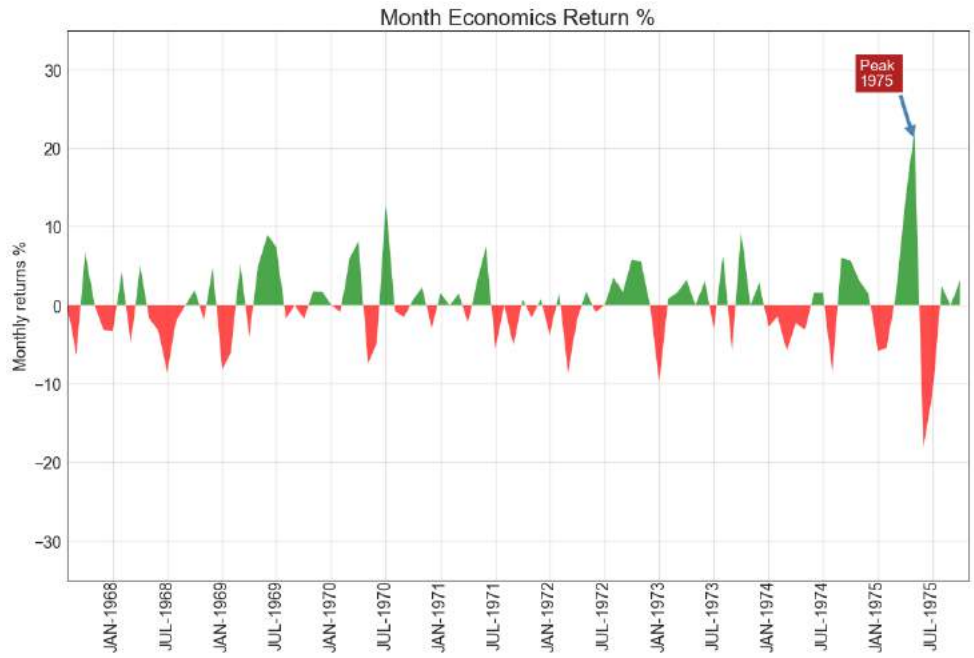


Figure 14.16: 面积图

14.5 排序（Ranking）

14.5.1 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()
```

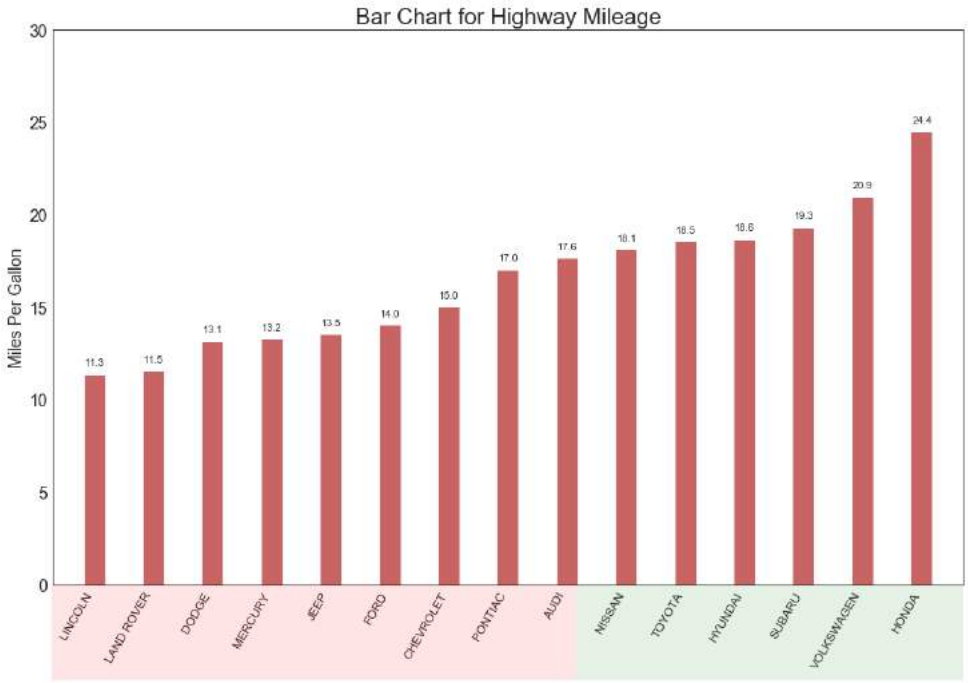


Figure 14.17: 有序条形图

14.5.2 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()
```

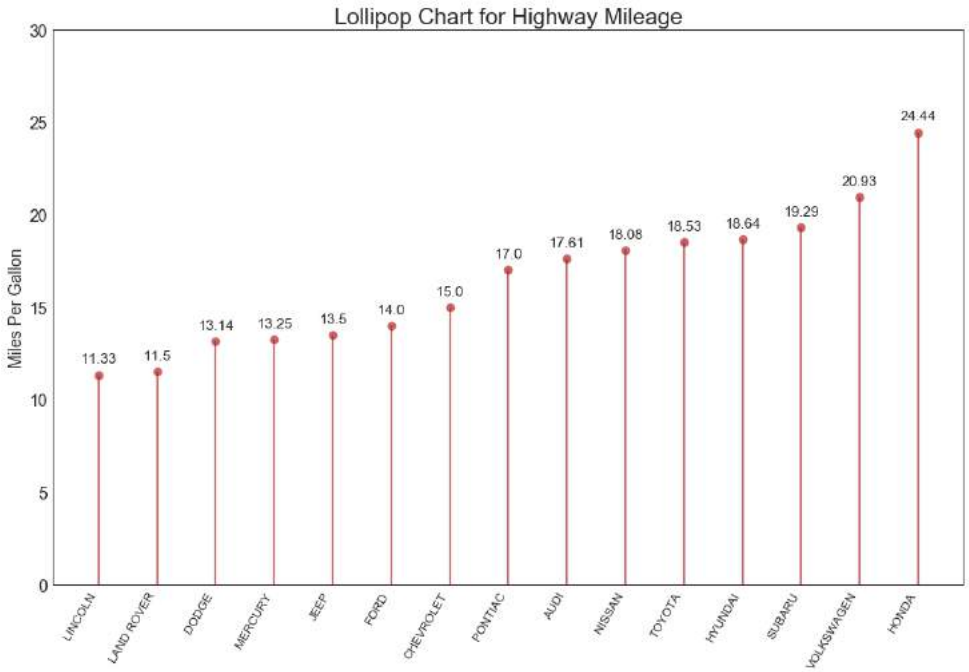


Figure 14.18: 棒棒糖图

14.5.3 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()
```

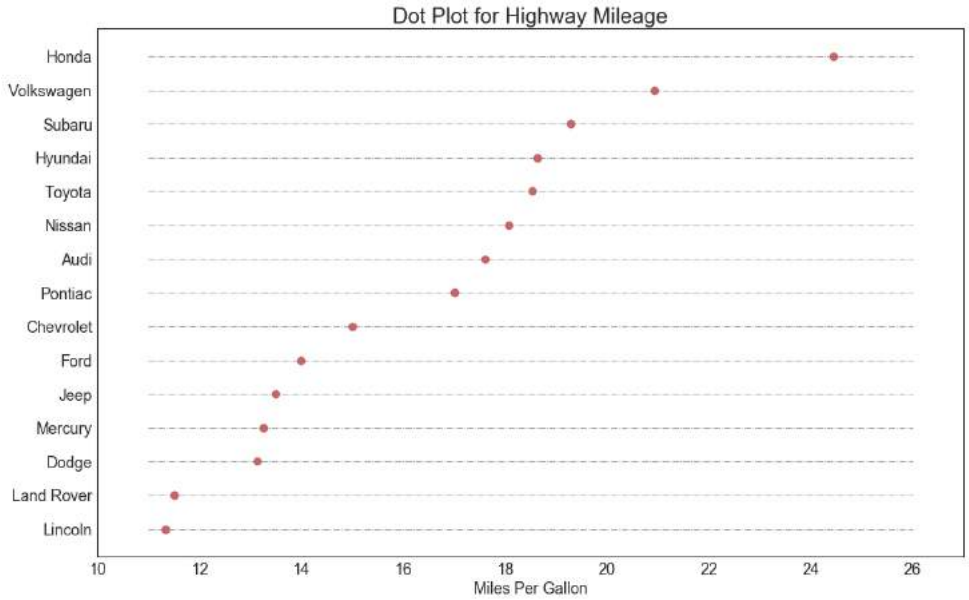


Figure 14.19: 包点图

14.5.4 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()
```

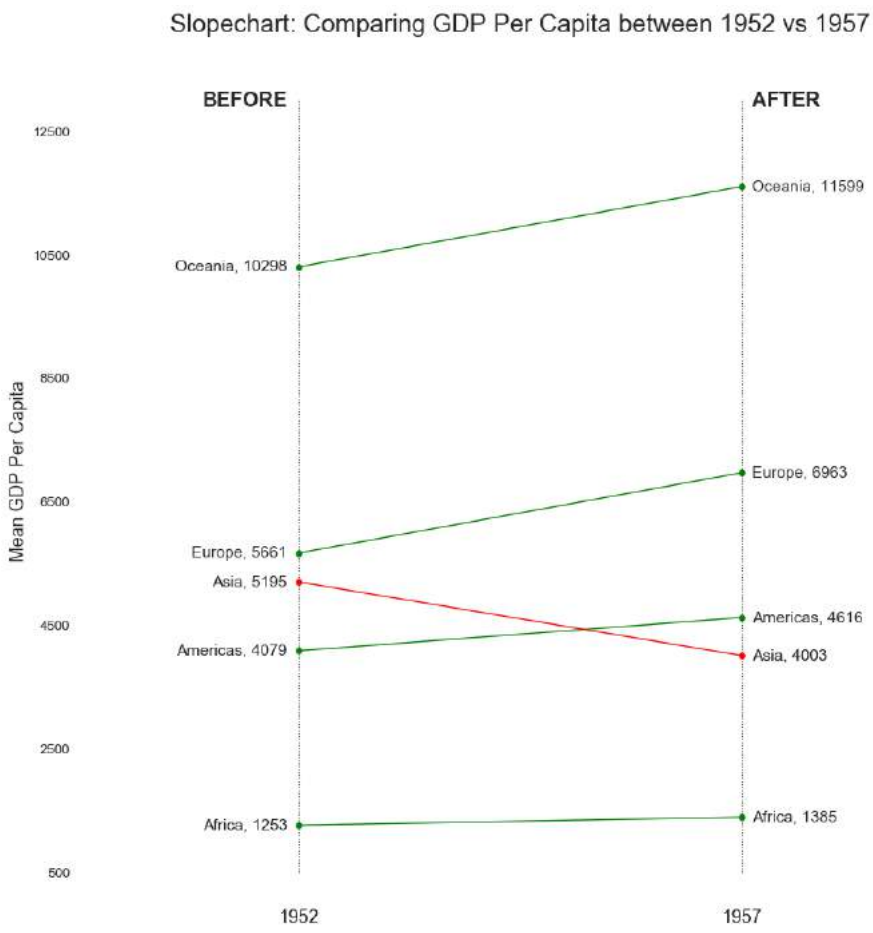


Figure 14.20: 坡度图

14.5.5 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()
```

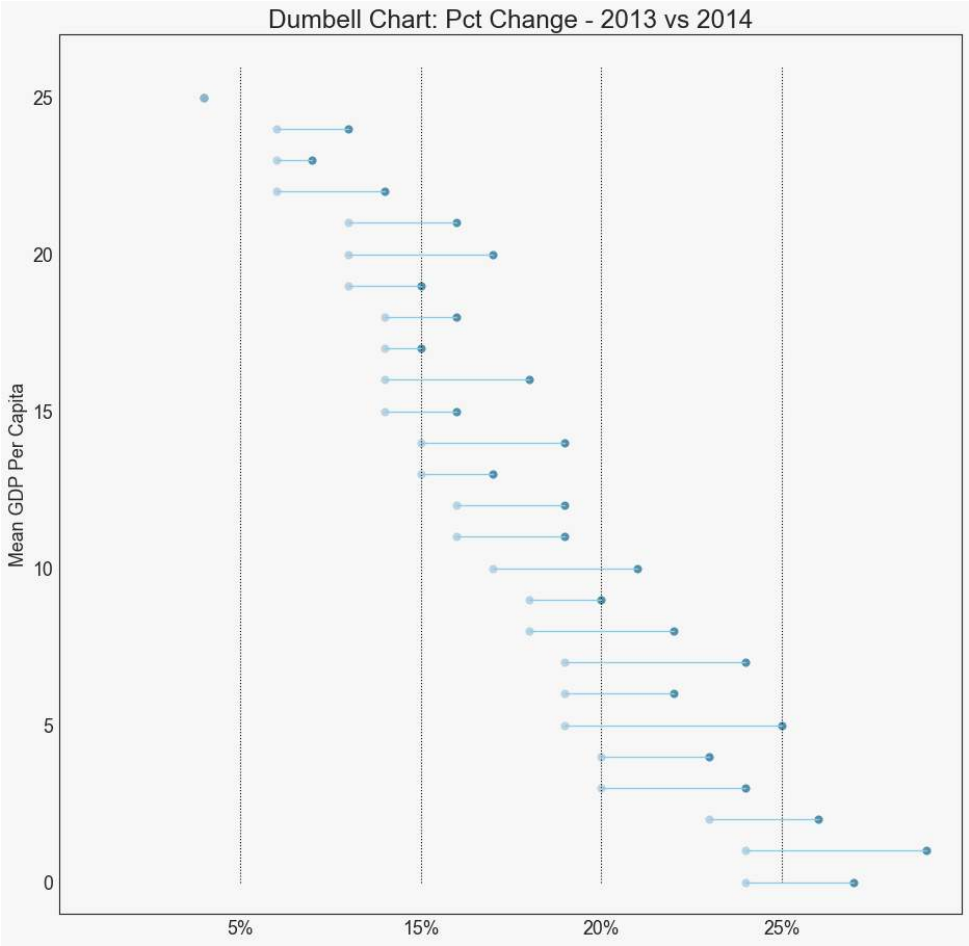


Figure 14.21: 哑铃图

14.6 分布（Distribution）

14.6.1 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()
```

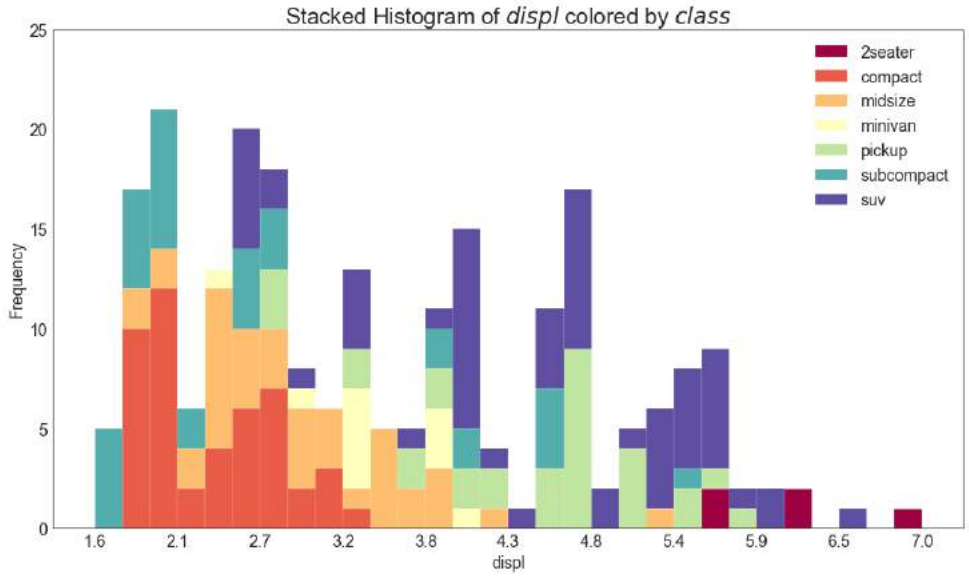


Figure 14.22: 连续变量的直方图

14.6.2 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()
```

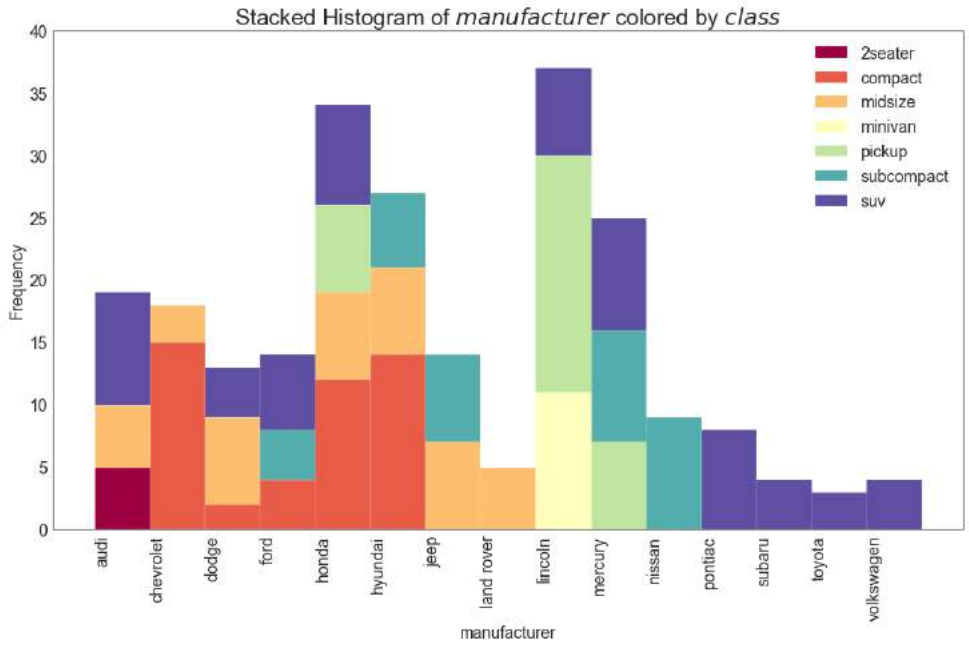


Figure 14.23: 类型变量的直方图

14.6.3 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()
```

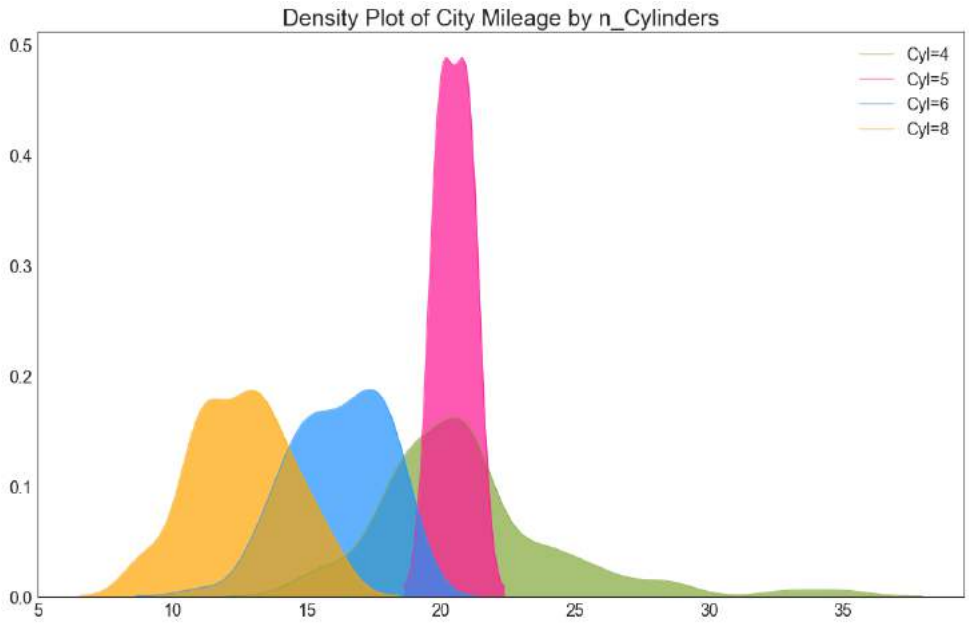


Figure 14.24: 密度图

14.6.4 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()
```

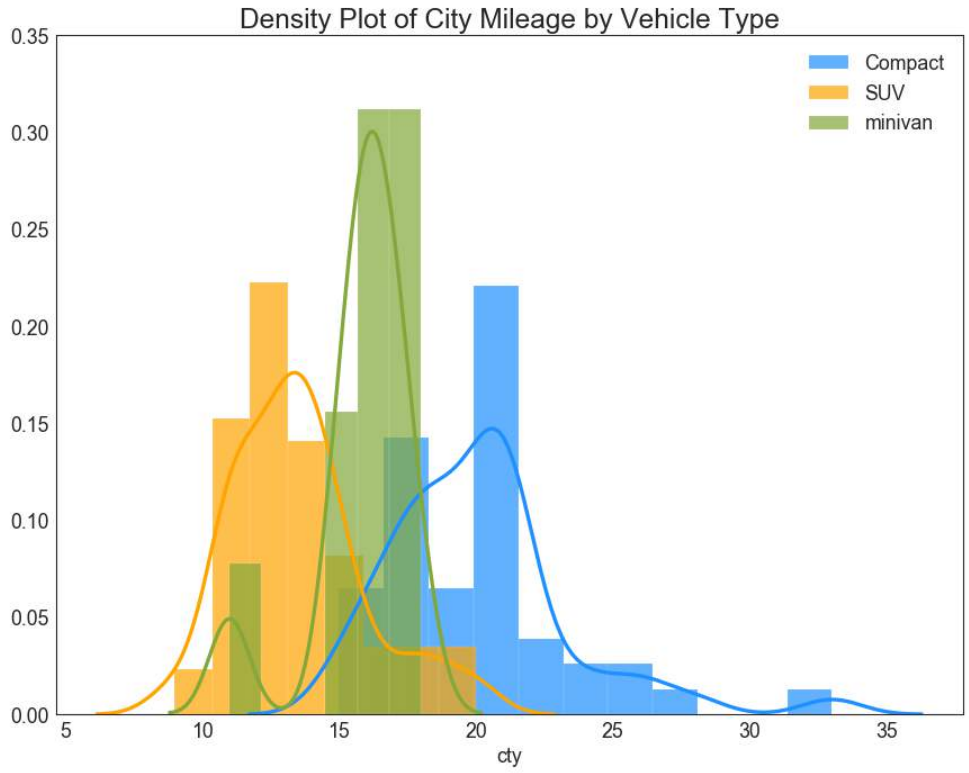


Figure 14.25: 直方密度线图

14.6.5 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()
```

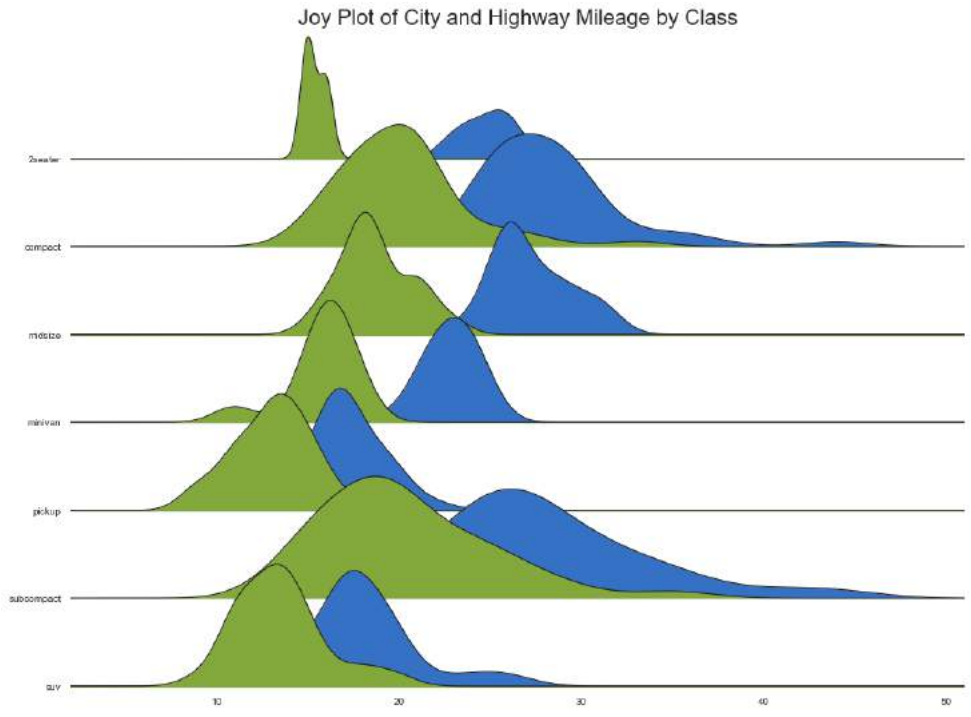


Figure 14.26: Joy Plot

14.6.6 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()
```

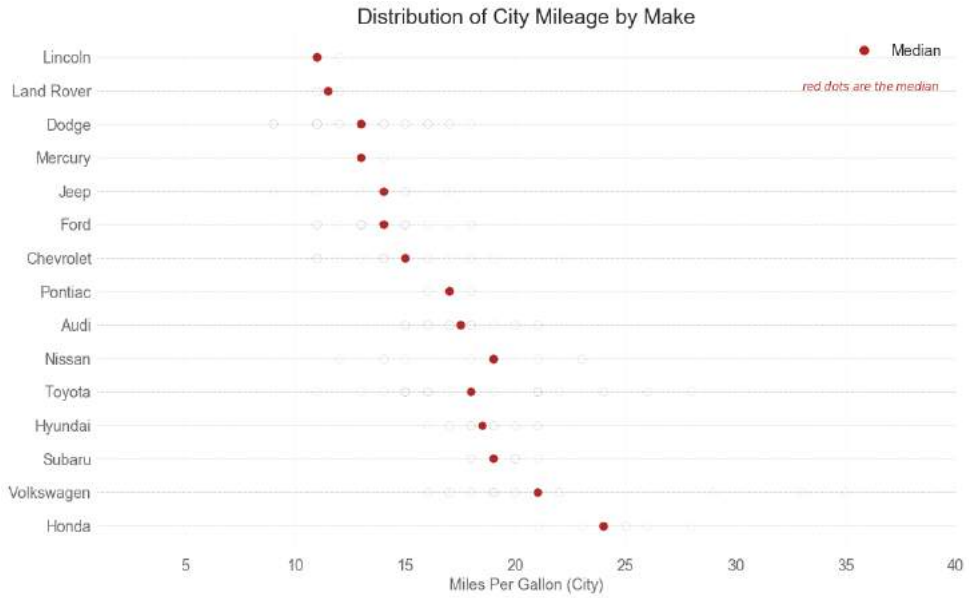


Figure 14.27: 分布式包点图

14.6.7 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')

add_n_obs(df,group_col='class',y='hwy')

# Decoration
plt.title('Box Plot of Highway Mileage by Vehicle Class', fontsize=22)
plt.ylim(10, 40)
plt.show()
```

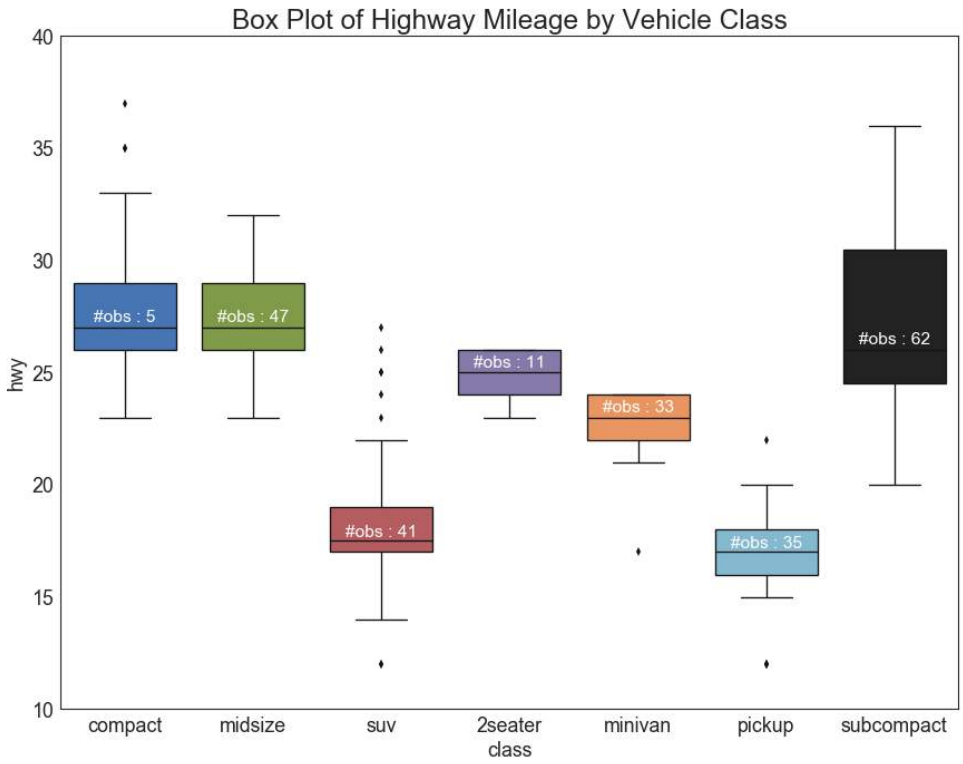


Figure 14.28: 箱形图

14.6.8 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。

```
# 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, hue='cyl')
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):
10     plt.vlines(i+.5, 10, 45, linestyles='solid', colors='gray', alpha=0.2)
11
12 # Decoration
13 plt.title('Box Plot of Highway Mileage by Vehicle Class', fontsize=22)
14 plt.legend(title='Cylinders')
15 plt.show()
```

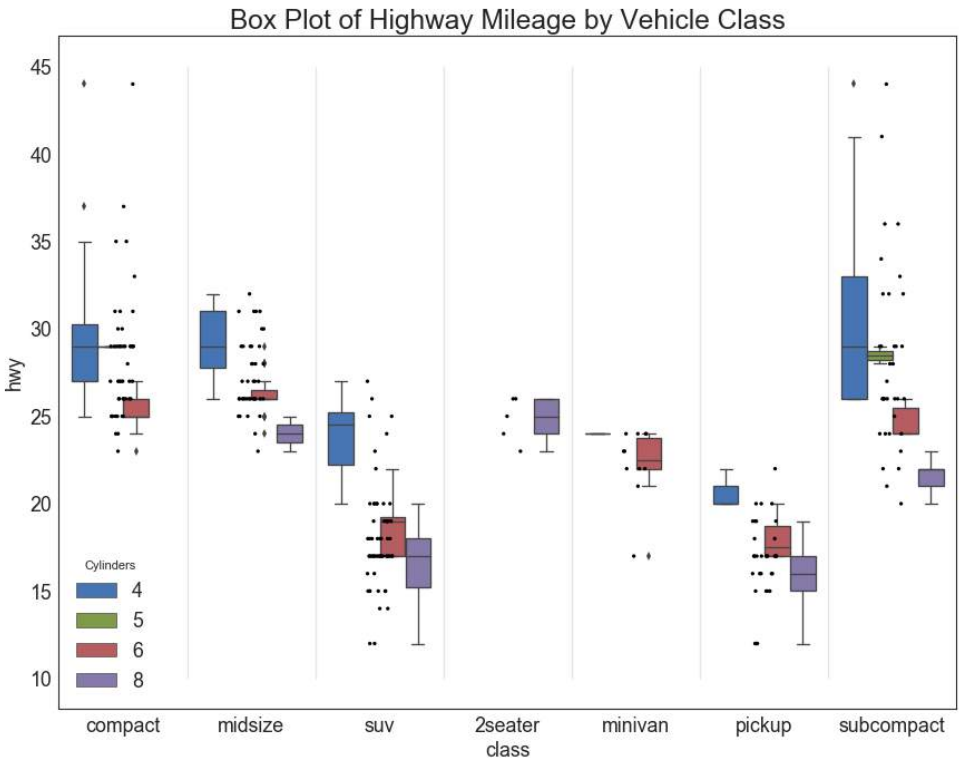


Figure 14.29: 包点 + 箱形图

14.6.9 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。

```
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.violinplot(x='class', y='hwy', data=df, scale='width', inner='quartile')
7
```

```
# Decoration
plt.title('Violin Plot of Highway Mileage by Vehicle Class', fontsize=22)
plt.show()
```

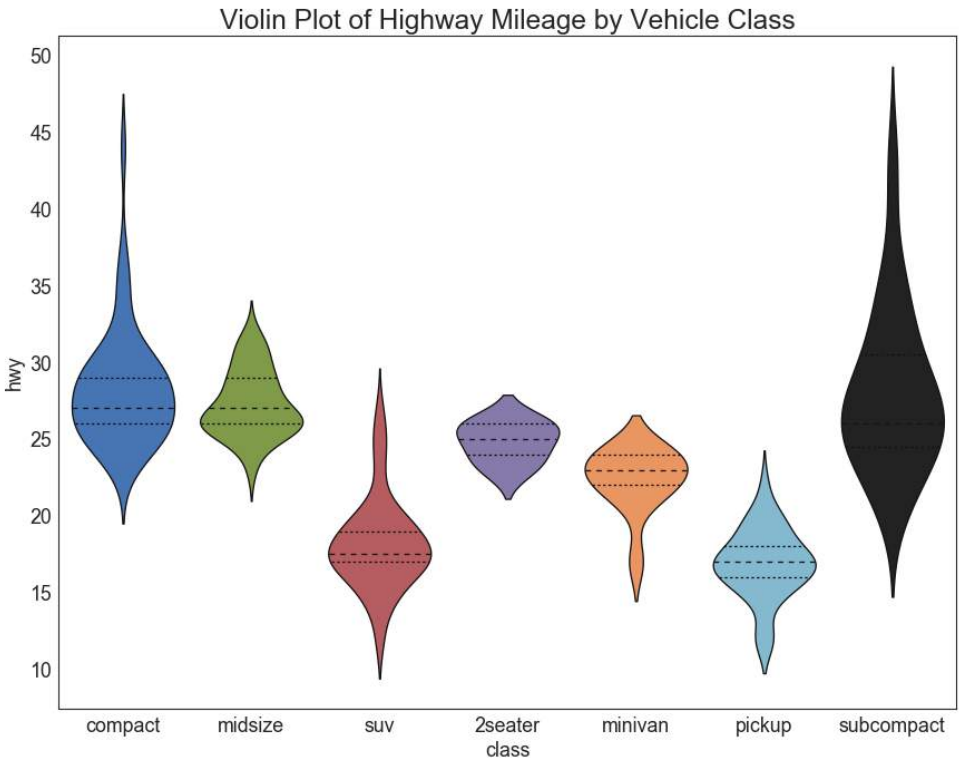


Figure 14.30: 小提琴图

14.6.10 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()
```

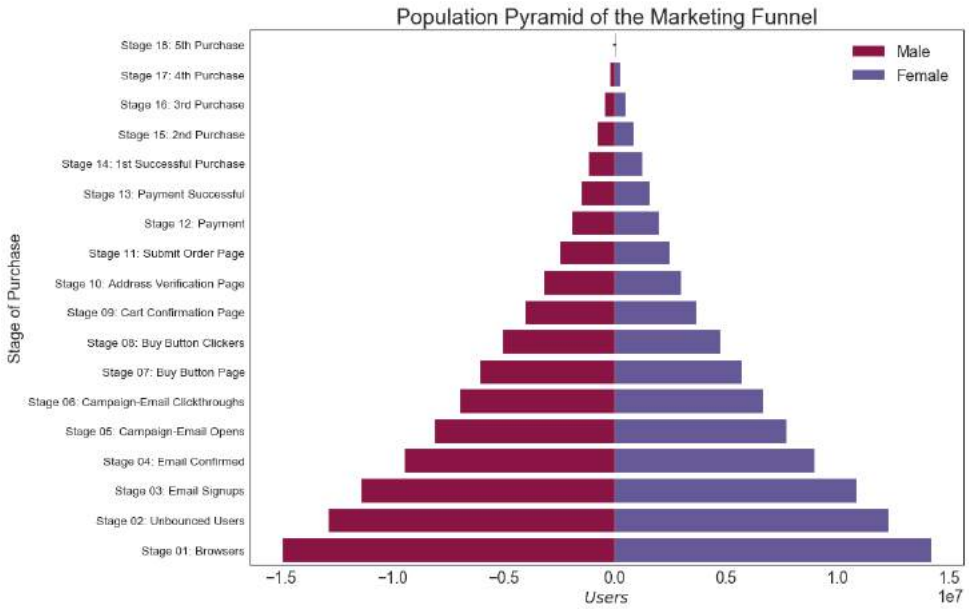


Figure 14.31: 人口金字塔

14.6.11 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()
```

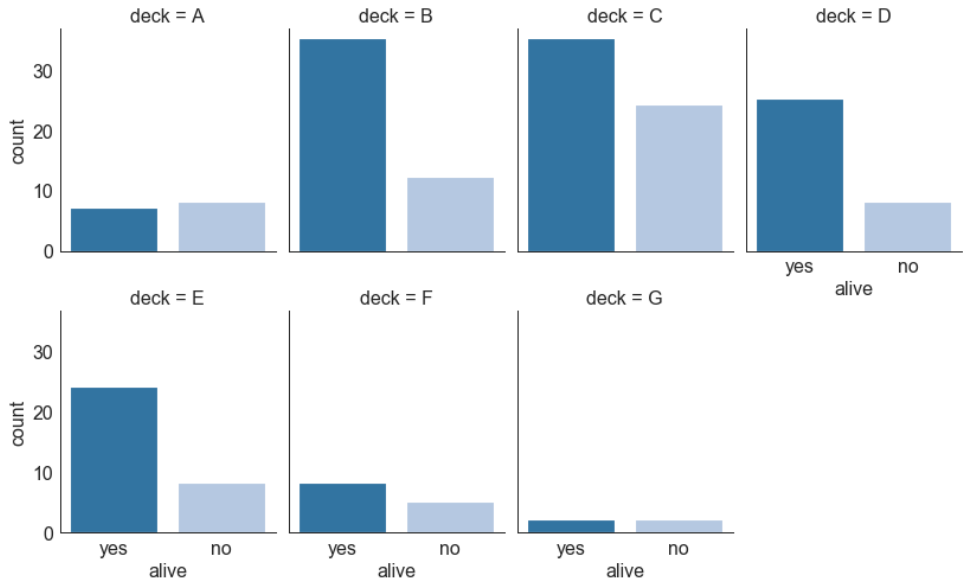


Figure 14.32: 分类图

```
# 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)
```

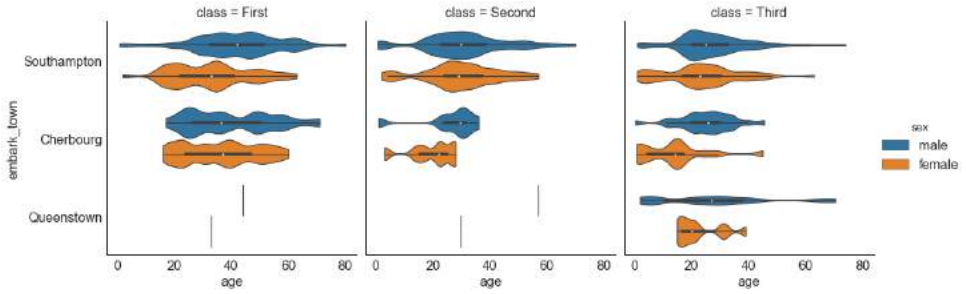


Figure 14.33: 分类图

14.7 组成（Composition）

14.7.1 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)
)
```

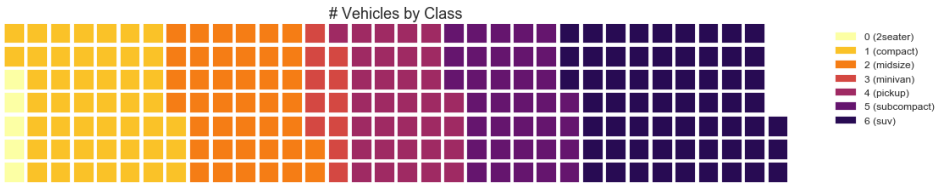


Figure 14.34: 华夫饼图

```
#! 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)
)
```

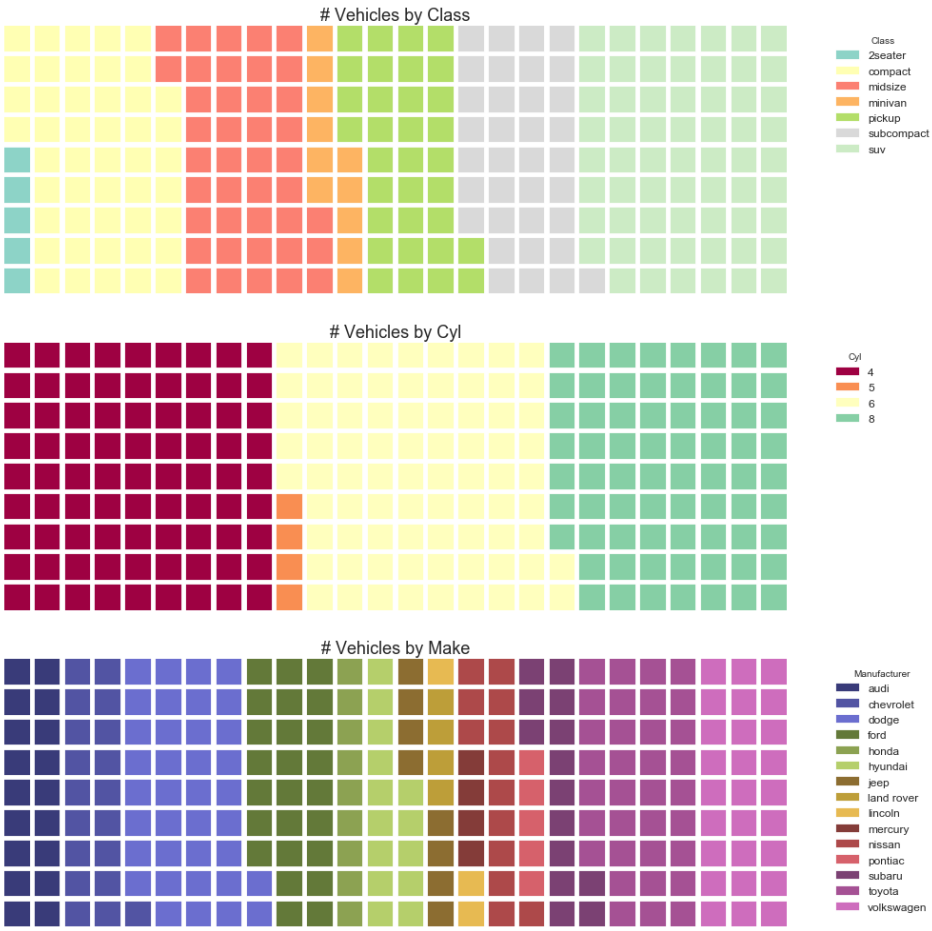


Figure 14.35: 华夫饼图

14.7.2 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()
```

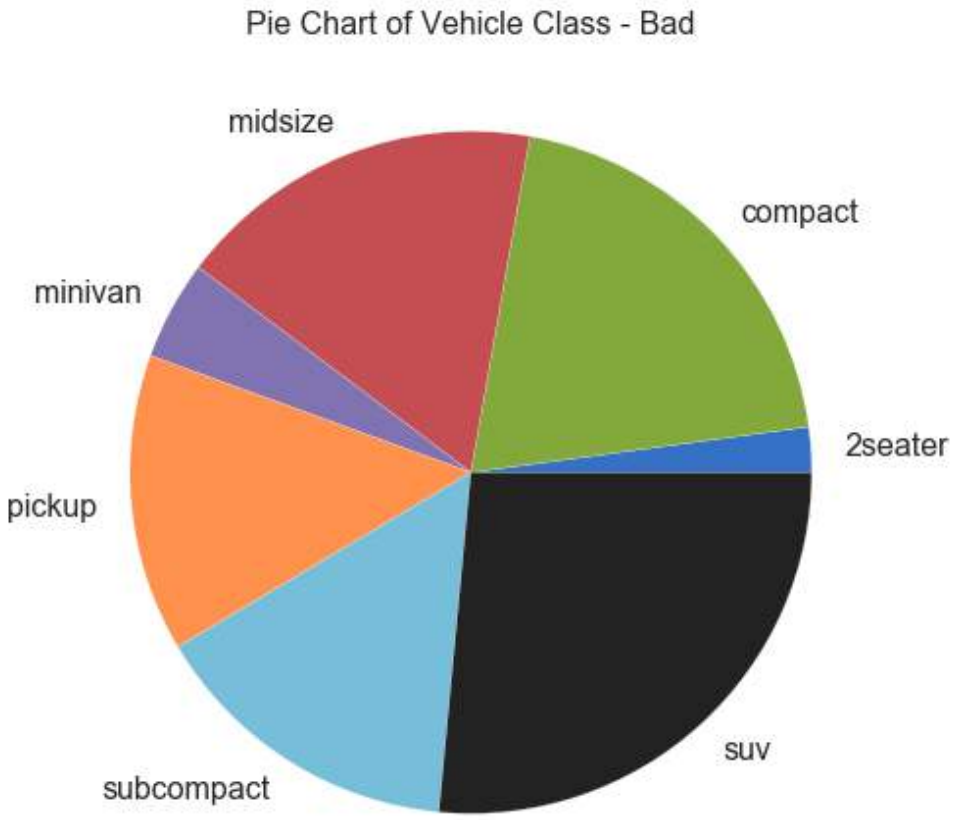


Figure 14.36: 饼图

```
# 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()
```

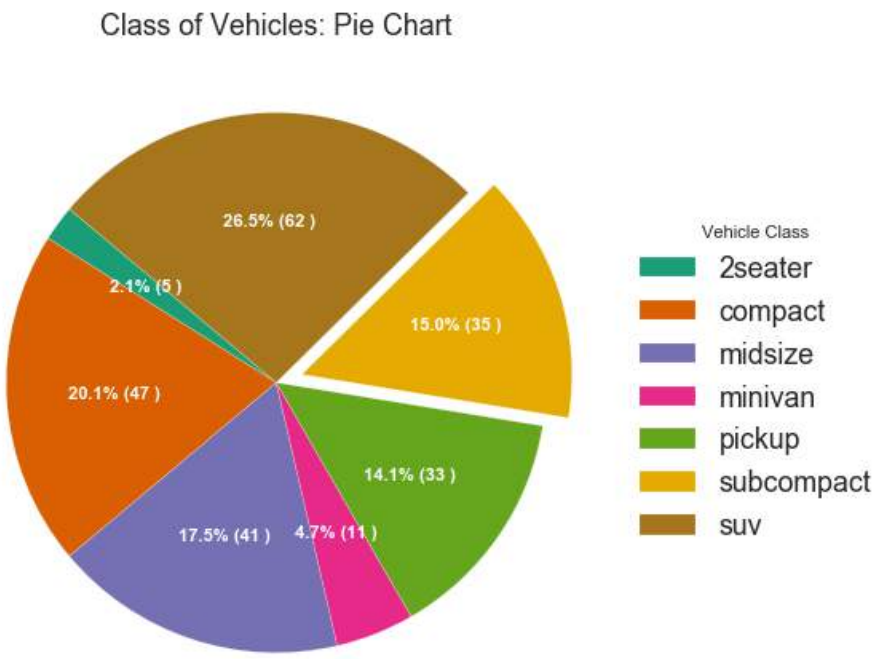


Figure 14.37: 饼图

14.7.3 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()
```

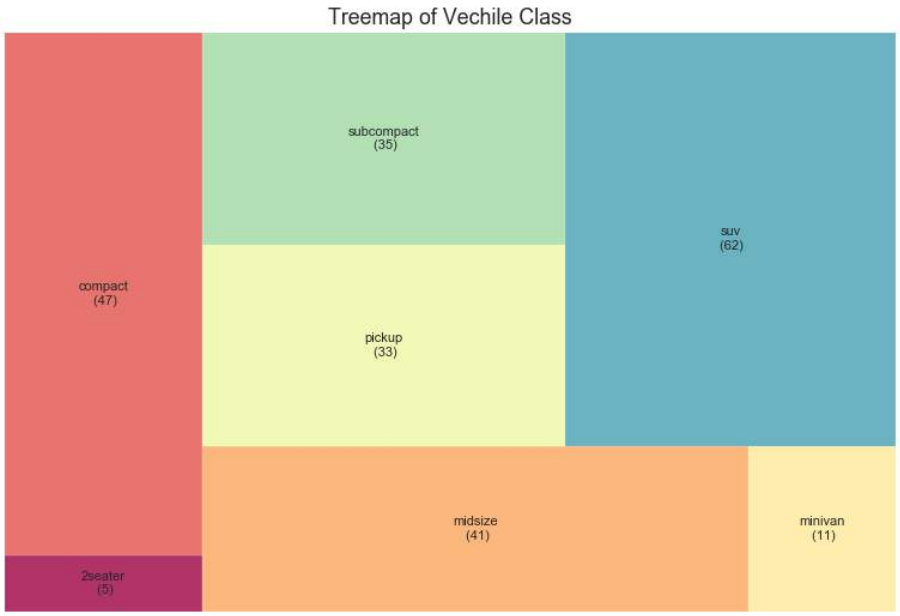


Figure 14.38: 树形图

14.7.4 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()
```

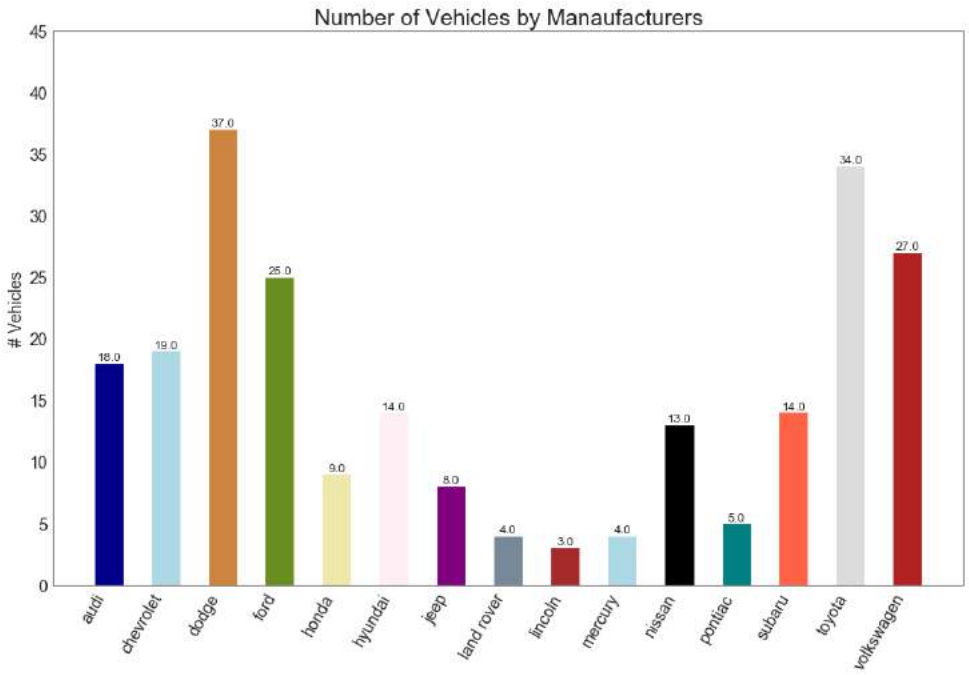


Figure 14.39: 条形图

14.8 变化（Change）

14.8.1 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()
```

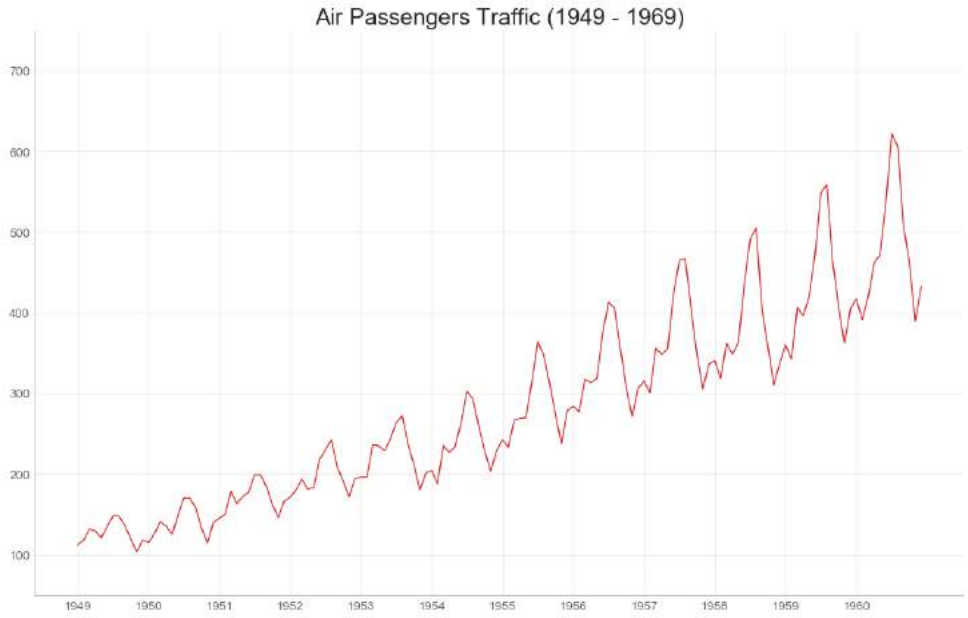


Figure 14.40: 时间序列图

14.8.2 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()
```

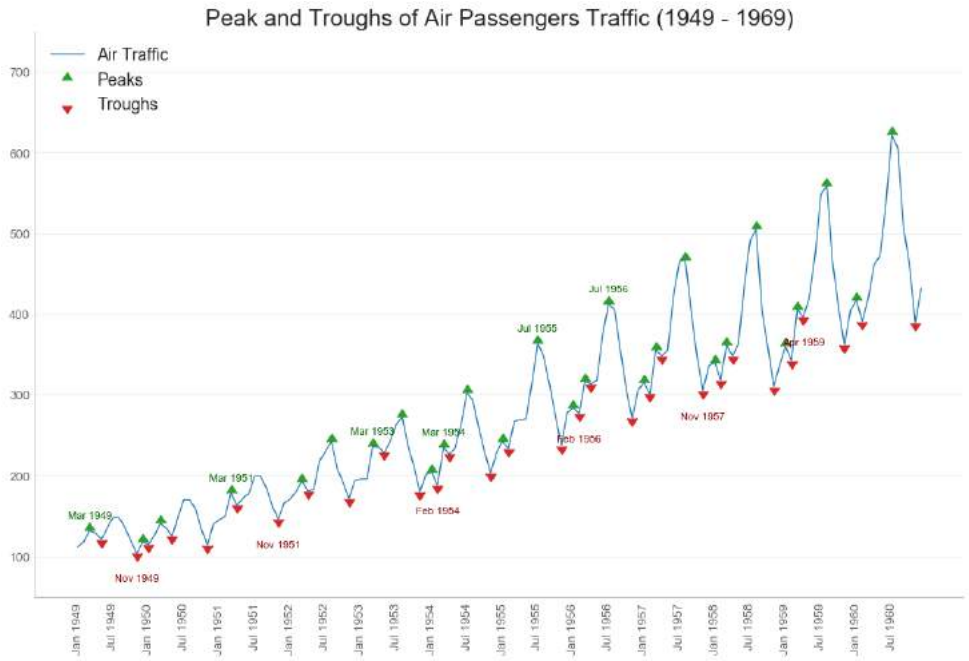


Figure 14.41: 带波峰波谷标记的时序图

14.8.3 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()
```

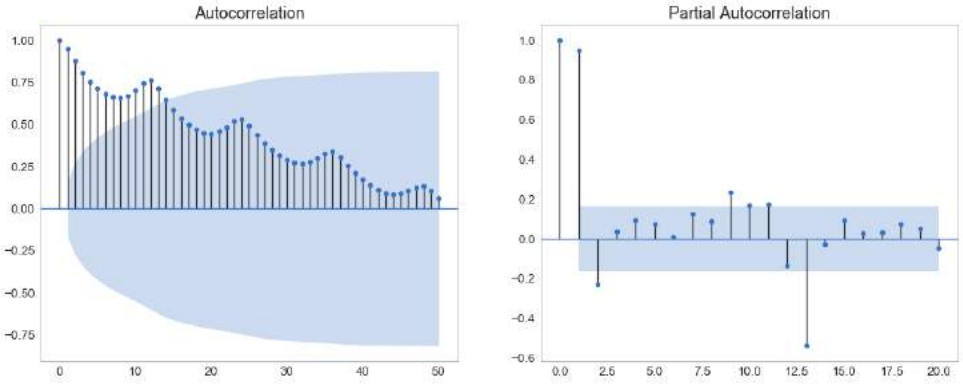


Figure 14.42: 自相关和部分自相关图

14.8.4 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()
```

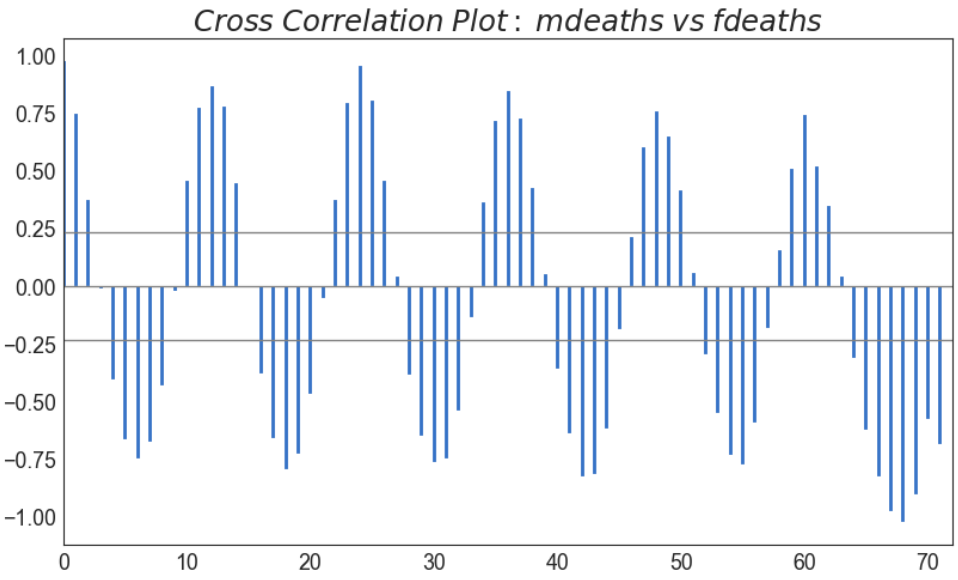


Figure 14.43: 交叉相关图

14.8.5 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()
```

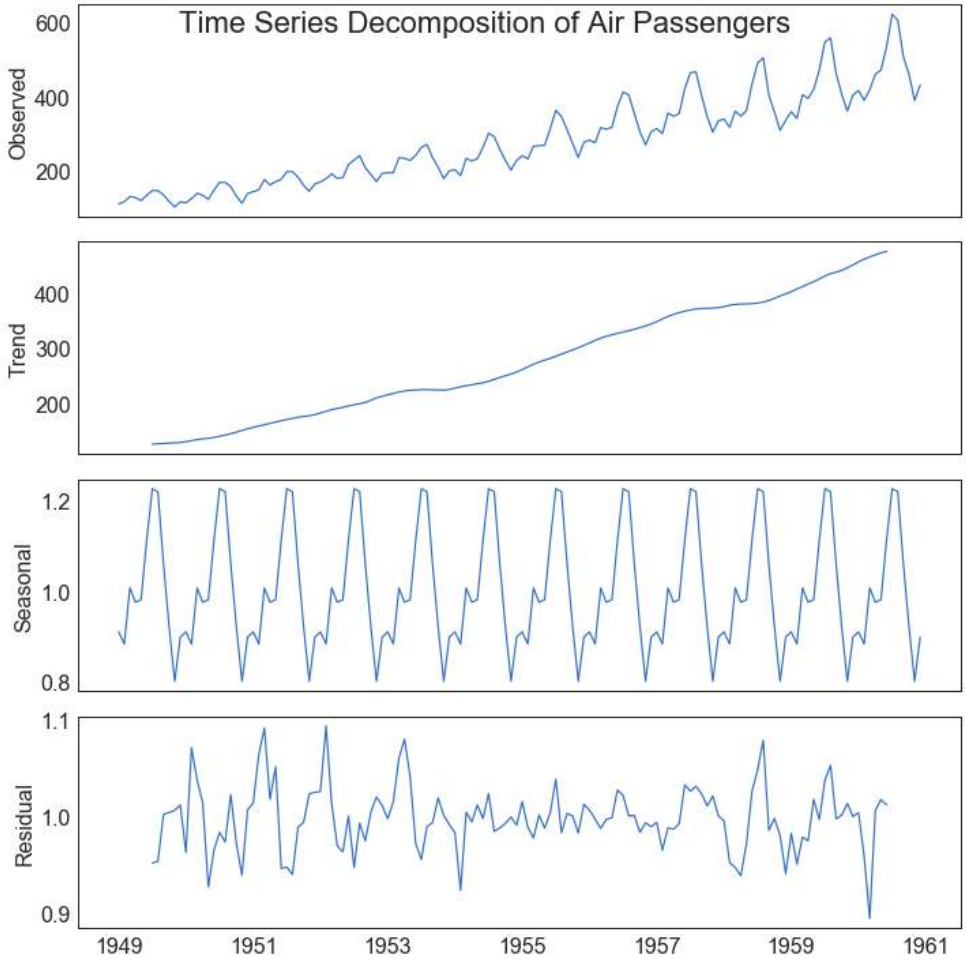


Figure 14.44: 时间序列分解图

14.8.6 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()
```

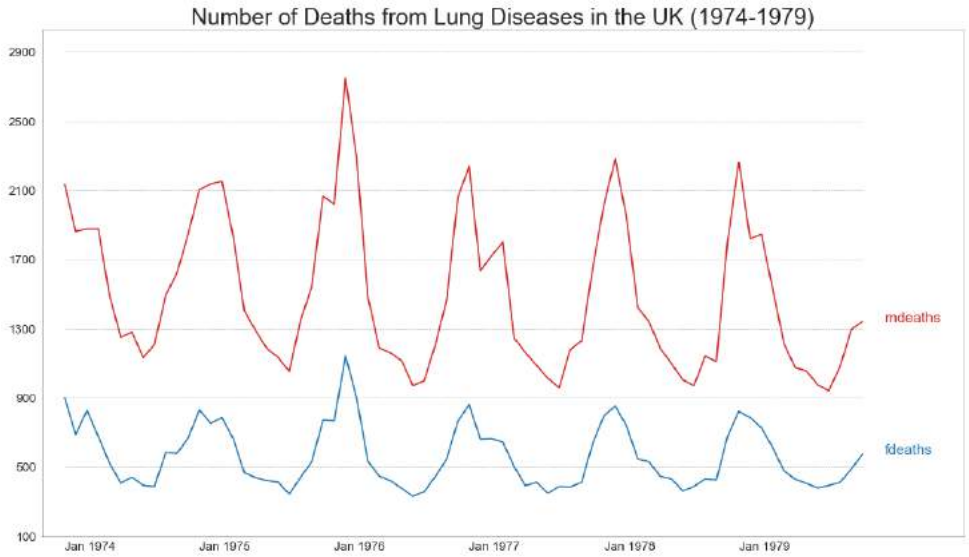


Figure 14.45: 多个时间序列

14.8.7 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()
```

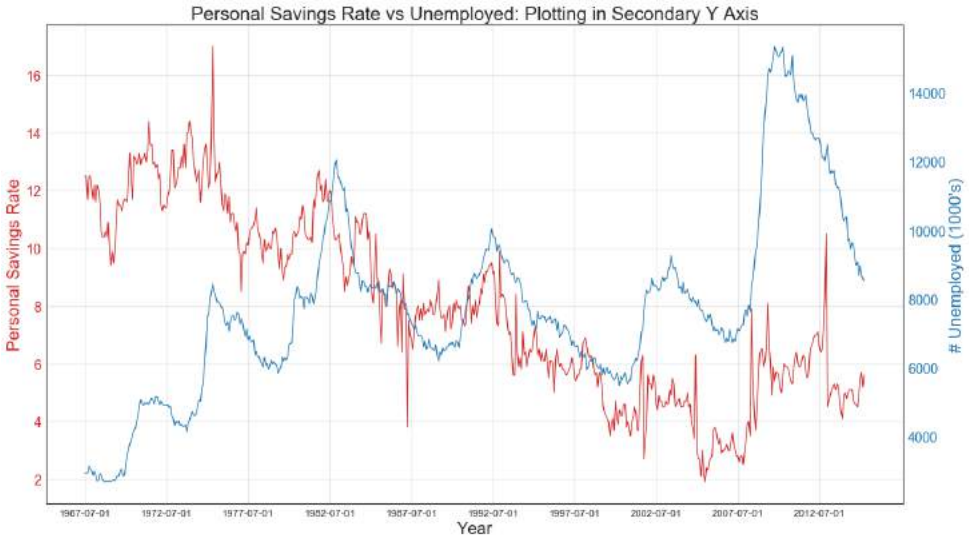


Figure 14.46: 使用辅助 Y 轴来绘制不同范围的图形

14.8.8 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()
```

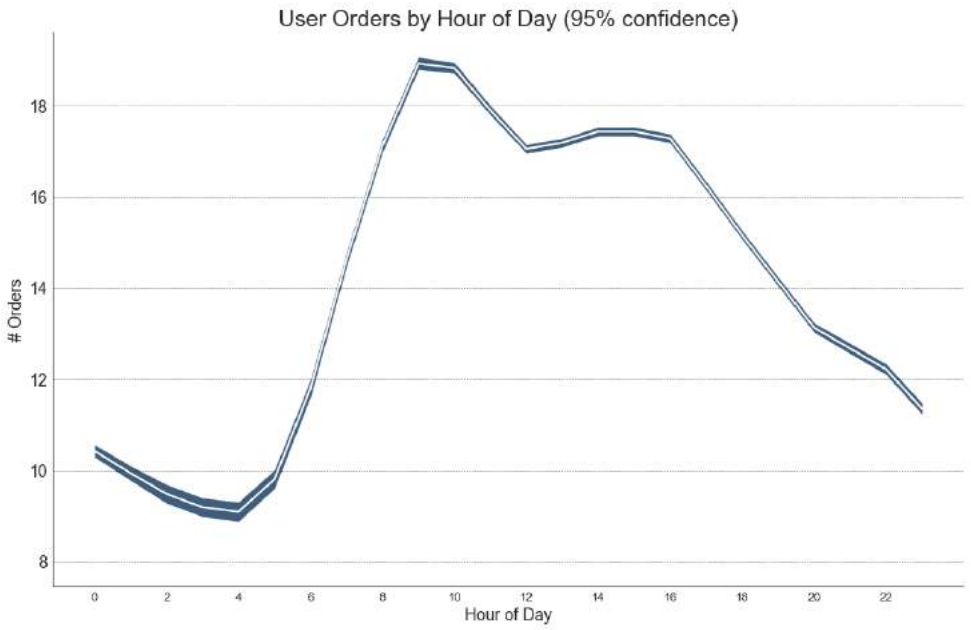


Figure 14.47: 带有误差带的时间序列

```
# "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()
```

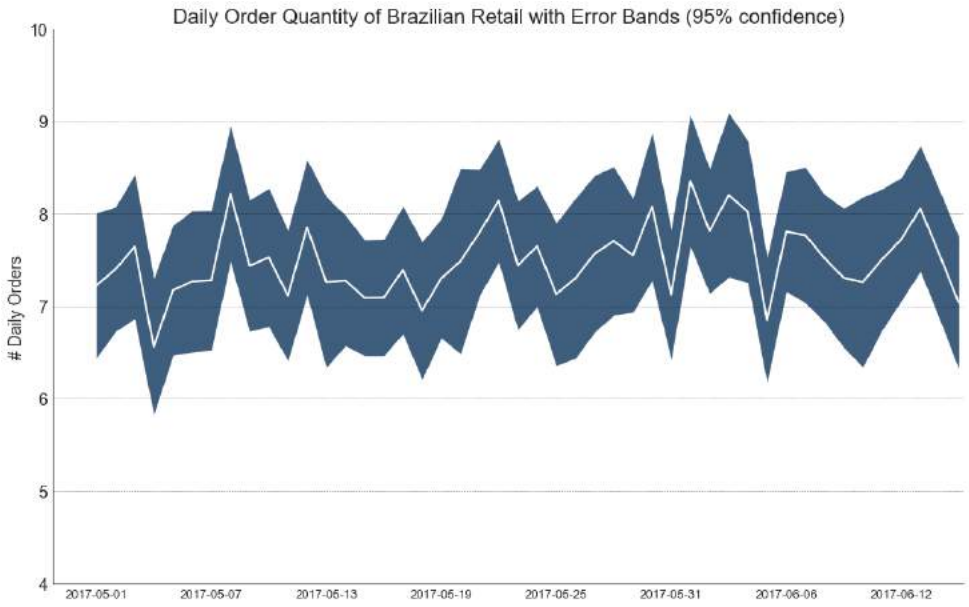


Figure 14.48: 带有误差带的时间序列

14.8.9 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()
```

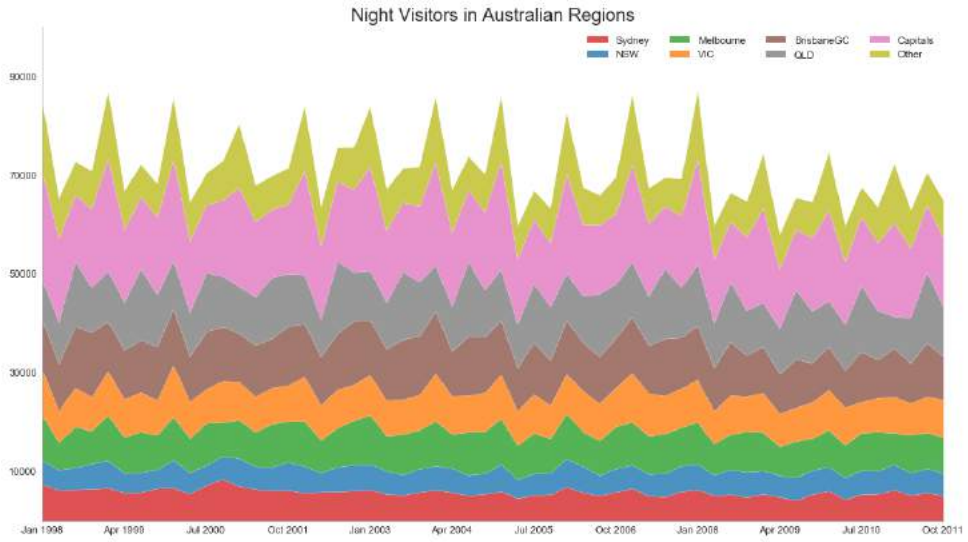


Figure 14.49: 堆积面积图

14.8.10 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。

```
# 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()
```

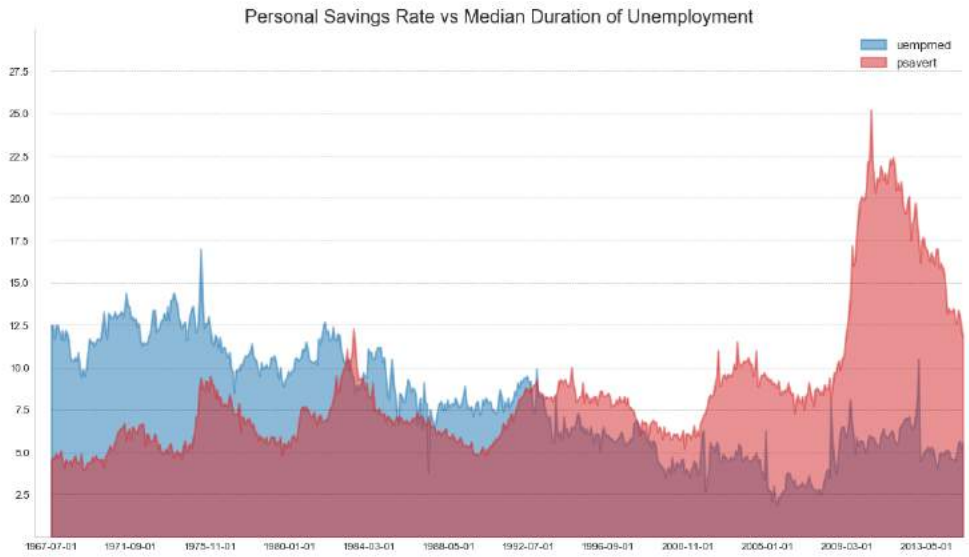


Figure 14.50: 未堆积的面积图

14.8.11 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()
```

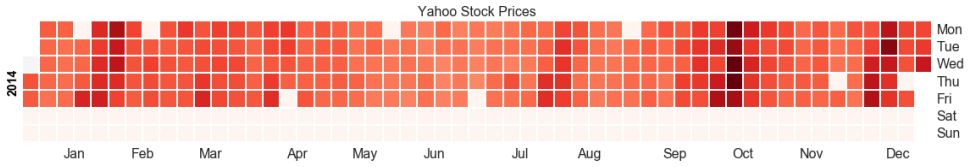


Figure 14.51: 未堆积的面积图

14.8.12 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)
```

```
33 plt.show()
```

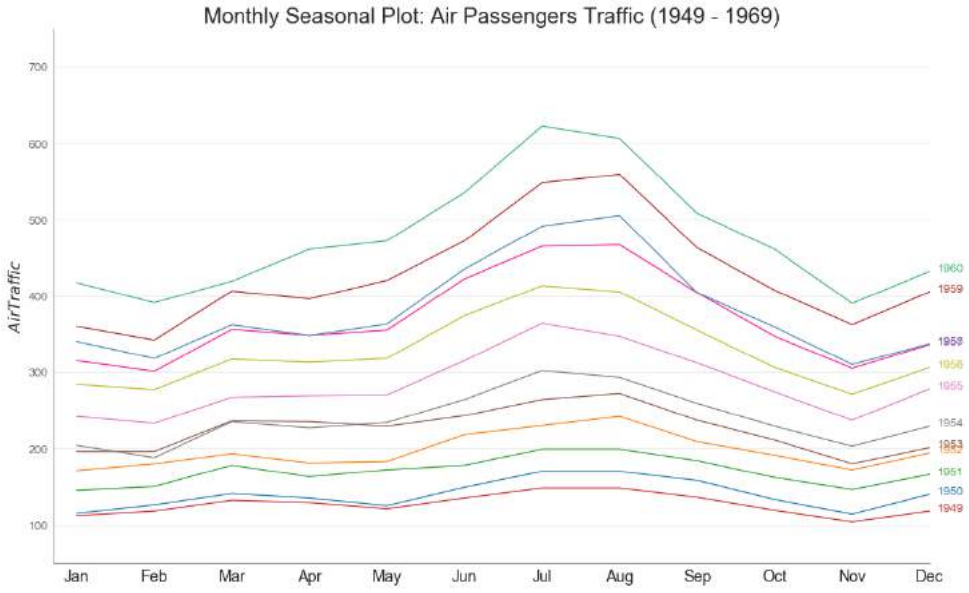


Figure 14.52: 季节图

14.9 分组（Groups）

14.9.1 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。

```
1  import scipy.cluster.hierarchy as shc
2
3  # Import Data
4  df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/
       USArrests.csv')
5
6  # Plot
7  plt.figure(figsize=(16, 10), dpi= 80)
8  plt.title("USArrests Dendograms", fontsize=22)
9  dend = shc.dendrogram(shc.linkage(df[['Murder', 'Assault', 'UrbanPop', 'Rape'
       ]], method='ward'), labels=df.State.values, color_threshold=100)
10 plt.xticks(fontsize=12)
11 plt.show()
```

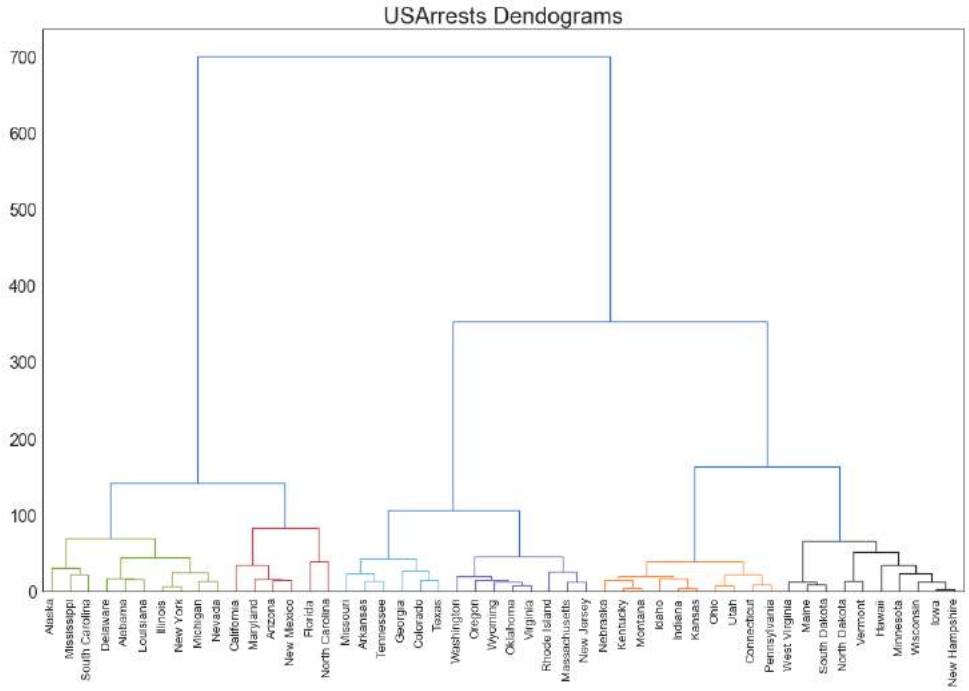


Figure 14.53: 树状图

14.9.2 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()
```

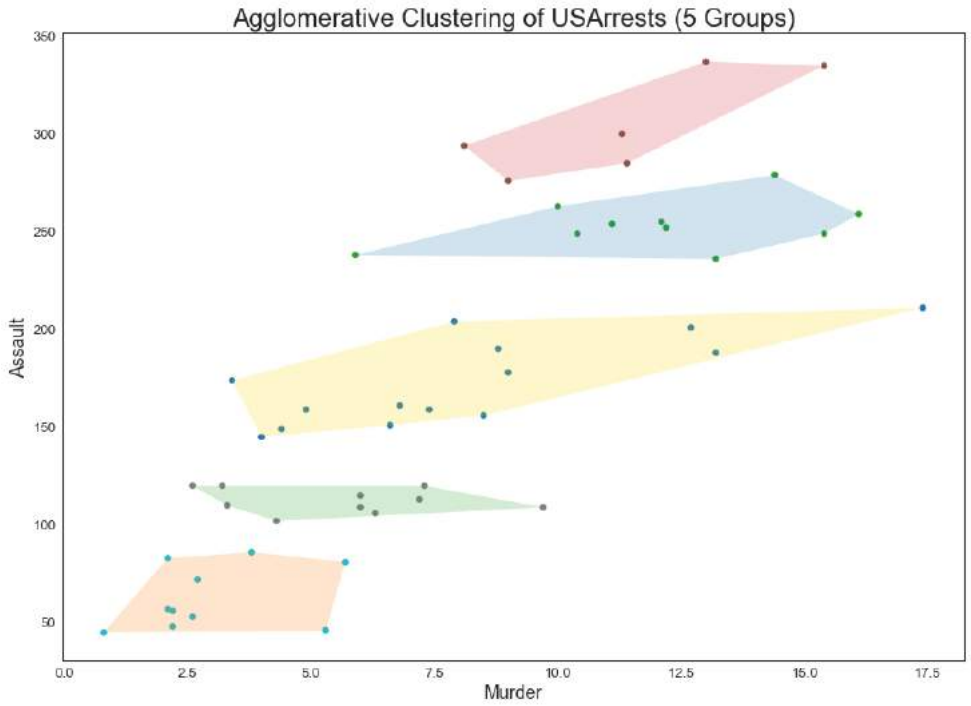


Figure 14.54: 簇状图

14.9.3 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()
```

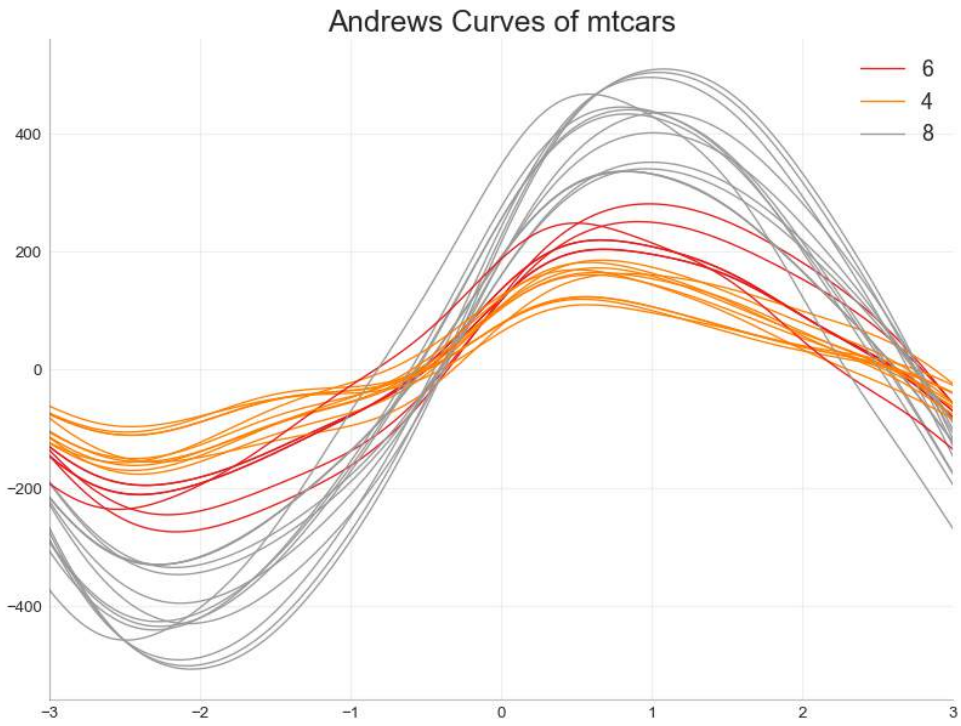


Figure 14.55: 安德鲁斯曲线

14.9.4 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()
```

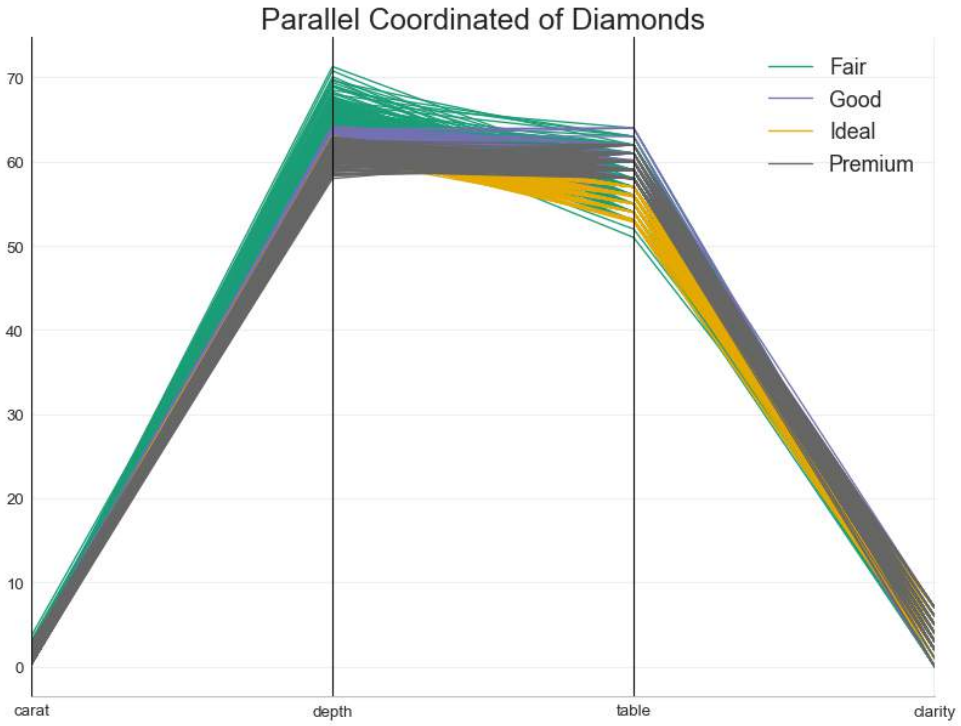


Figure 14.56: 平行坐标

> 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：PyDataRoad），接收我的更新通知。

Figure 14.57: 微信公众号: Python 数据之道

15 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:PyDataRoad）

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()
```

15.1 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 15.1: 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.
```

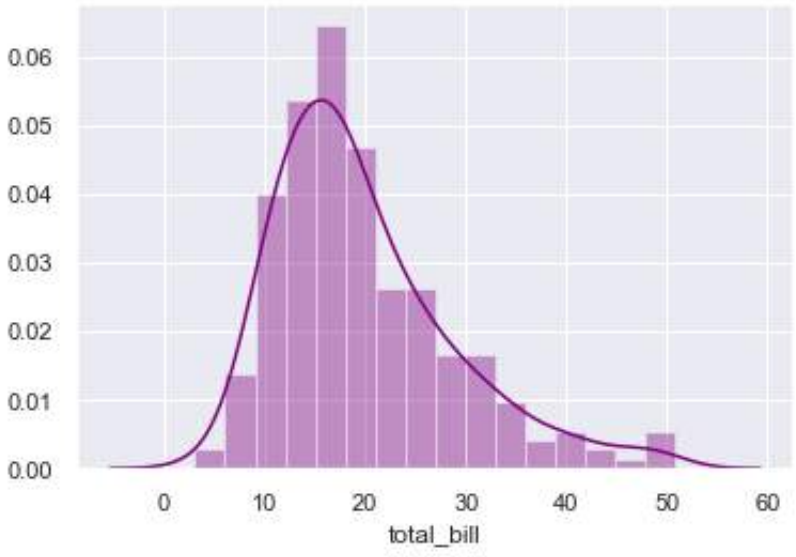


Figure 15.2: 直方图

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。

15.2 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")
```

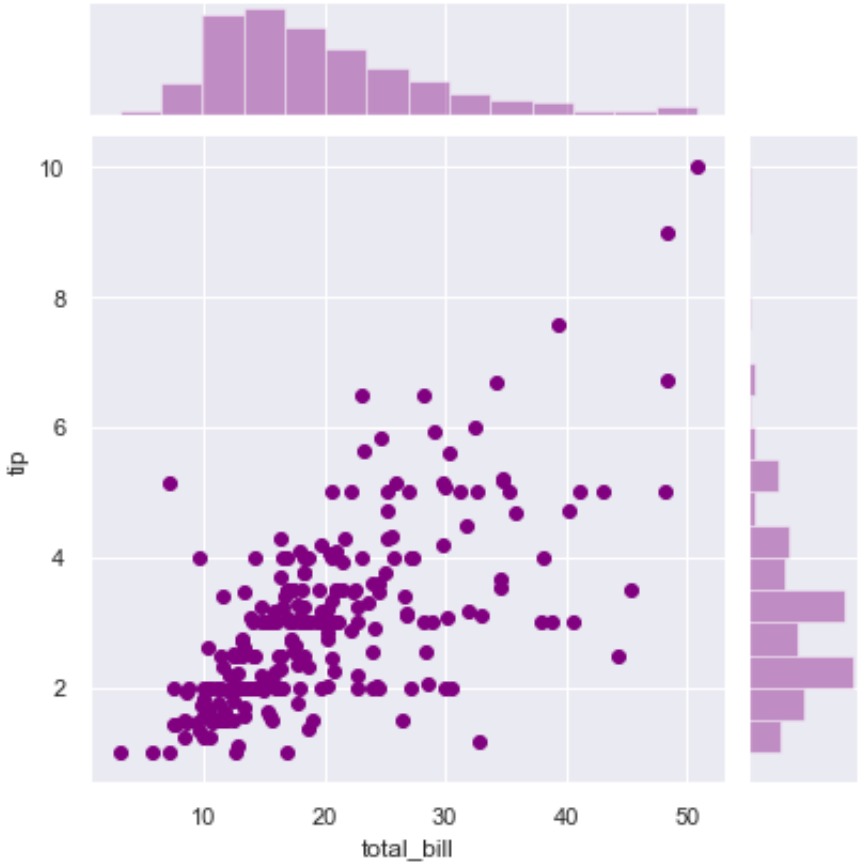


Figure 15.3: 联合分布图

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。

15.2.1 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”)
```

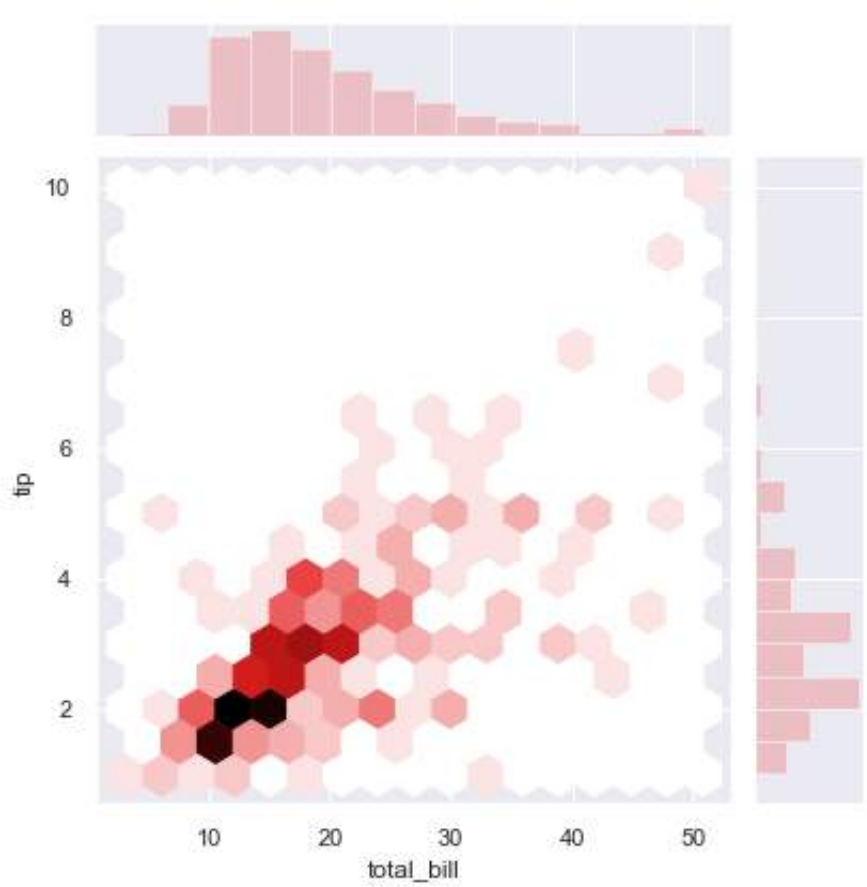


Figure 15.4: 联合分布图: kind=“hex”

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。

15.2.2 Jointplot :: kind =”kde”

```
# Jointplot - Scatterplot and Histogram
sns.jointplot(x = tips[”total_bill”], y = tips[”tip”],kind = ”kde”,
color=”purple”) # contour plot
```

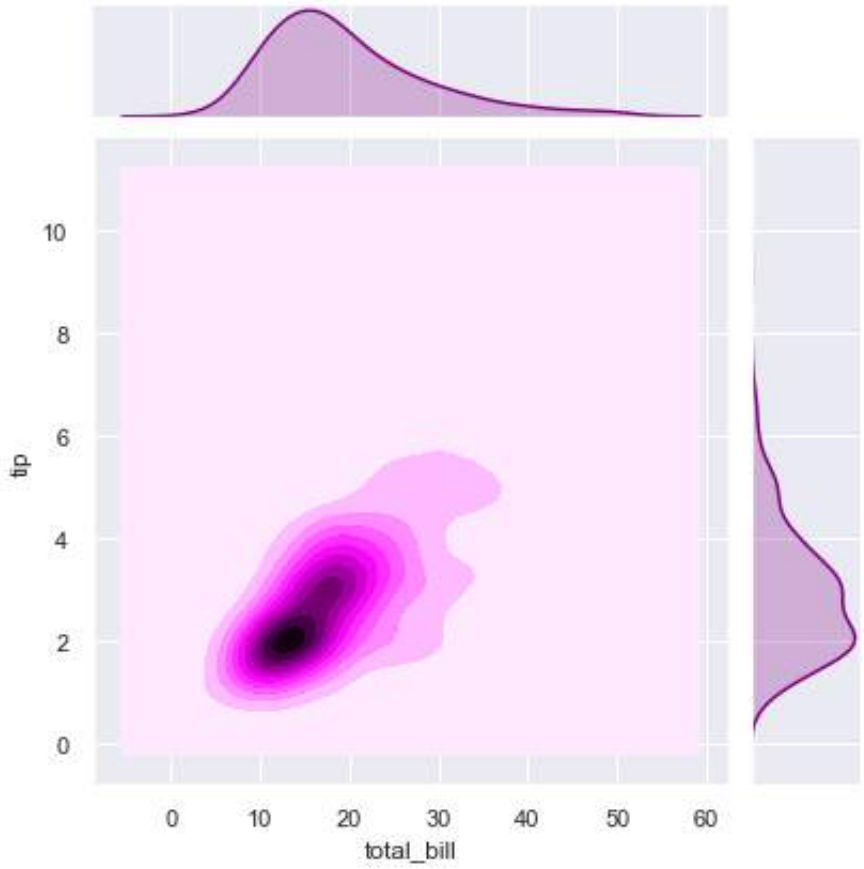


Figure 15.5: 联合分布图: kind=“kde”

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.

15.3 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
```

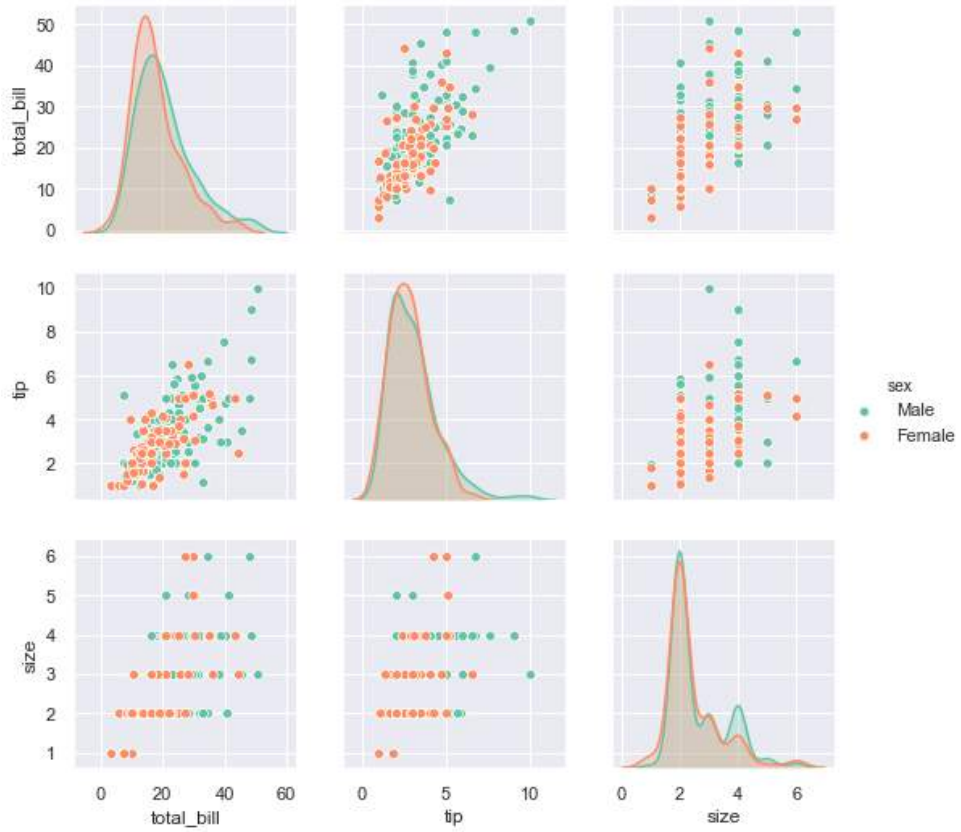


Figure 15.6: 矩阵图

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。

15.4 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.
```

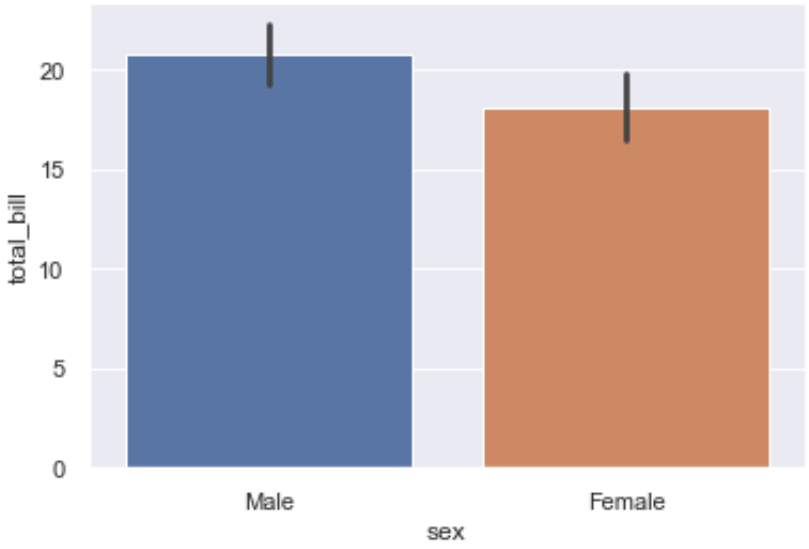


Figure 15.7: 条形图

```
# 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
```

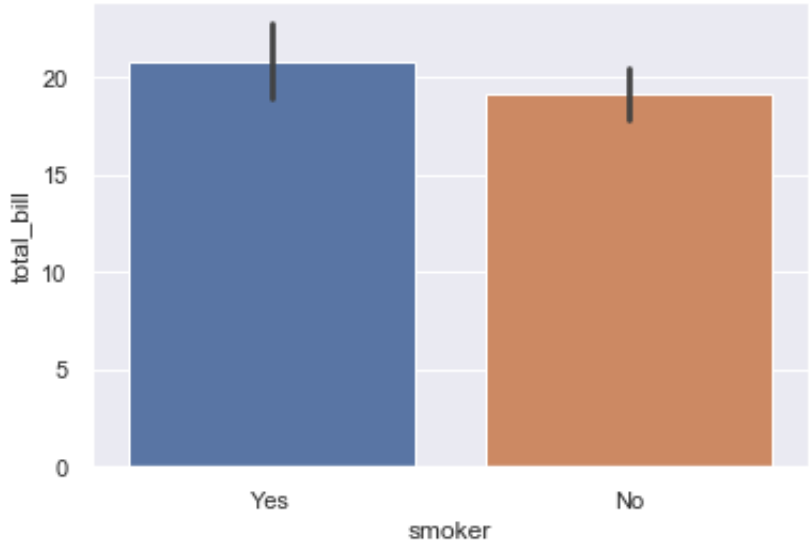


Figure 15.8: 条形图

```
# 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
```

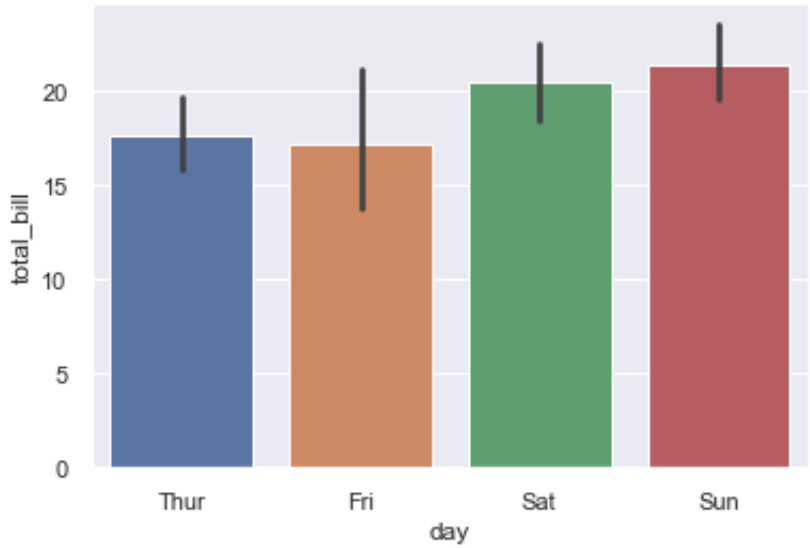


Figure 15.9: 条形图

15.5 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)
```

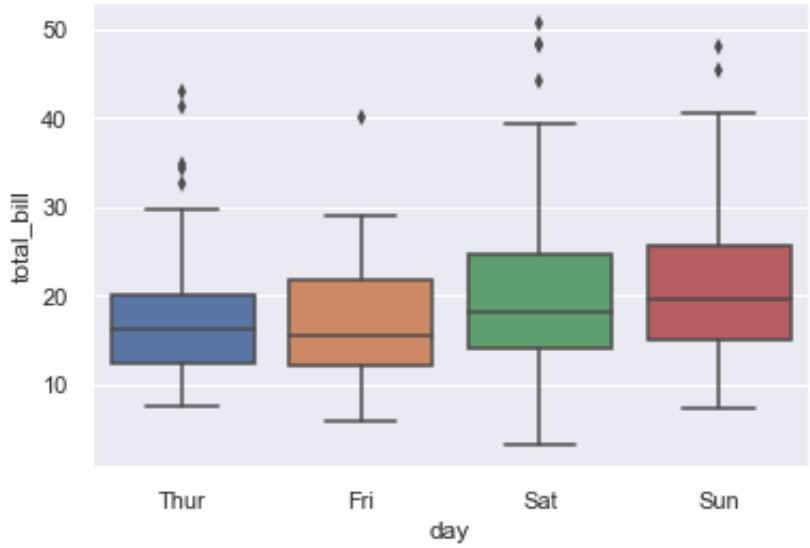


Figure 15.10: 箱形图

```
# 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
```

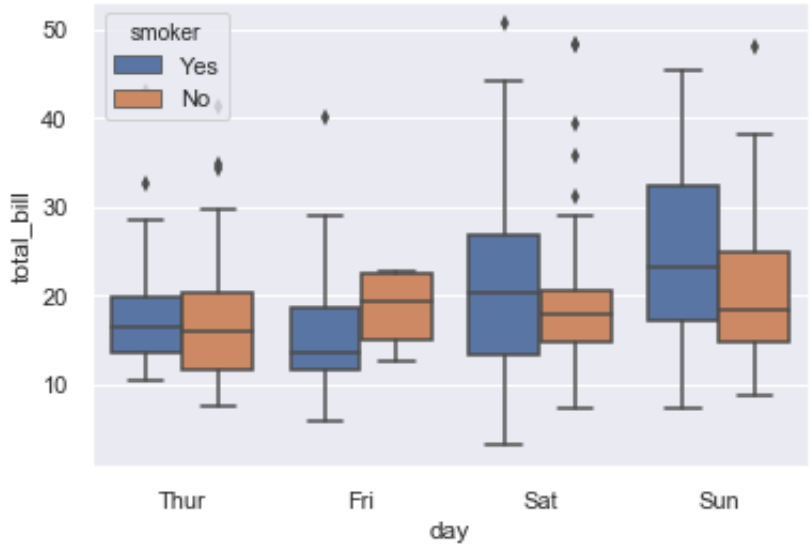


Figure 15.11: 箱形图

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。

```
1  # Violin Plots
2  sns.violinplot(x = ”day”, y = ”total_bill”, data = tips)
```

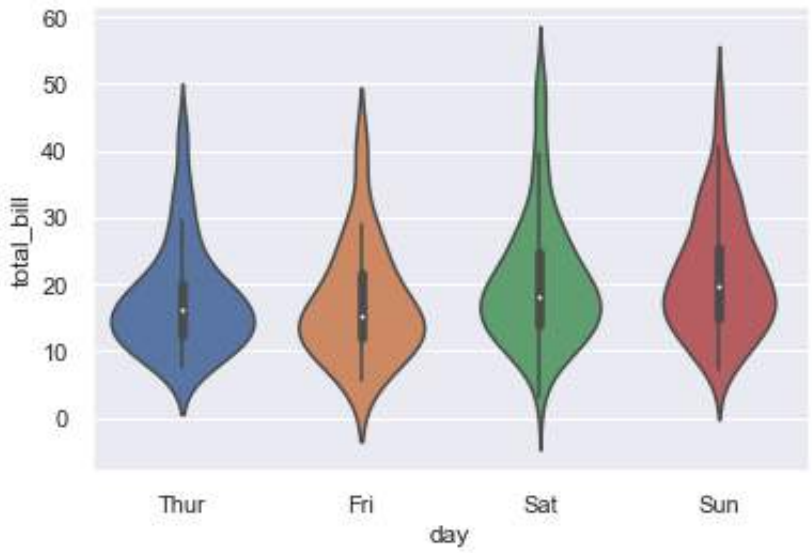


Figure 15.12: 小提琴图

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：

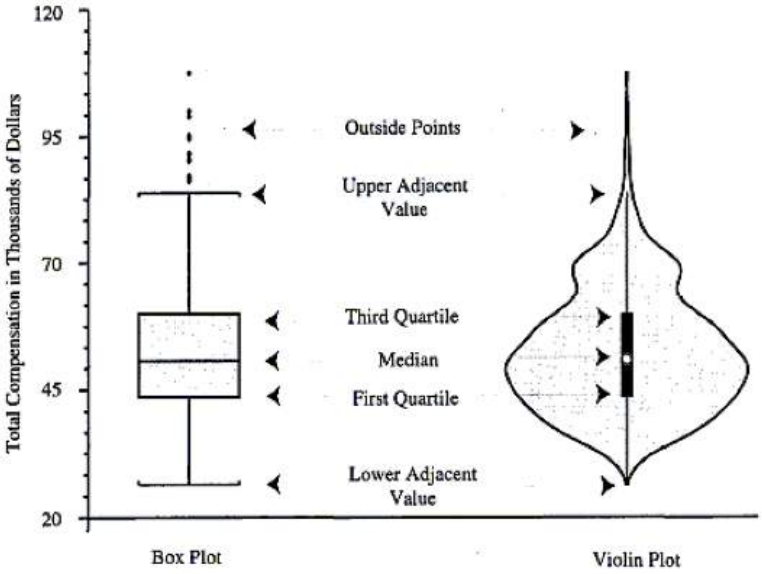


Figure 1. Common Components of Box Plot and Violin Plot. Total compensation for all academic ranks.

Image source: Google

Figure 15.13: 小提琴图示意图

15.6 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：

```
1 # LM PLot
2 sns.lmplot(x = "total_bill", y = "tip", data = tips, hue="day")
```

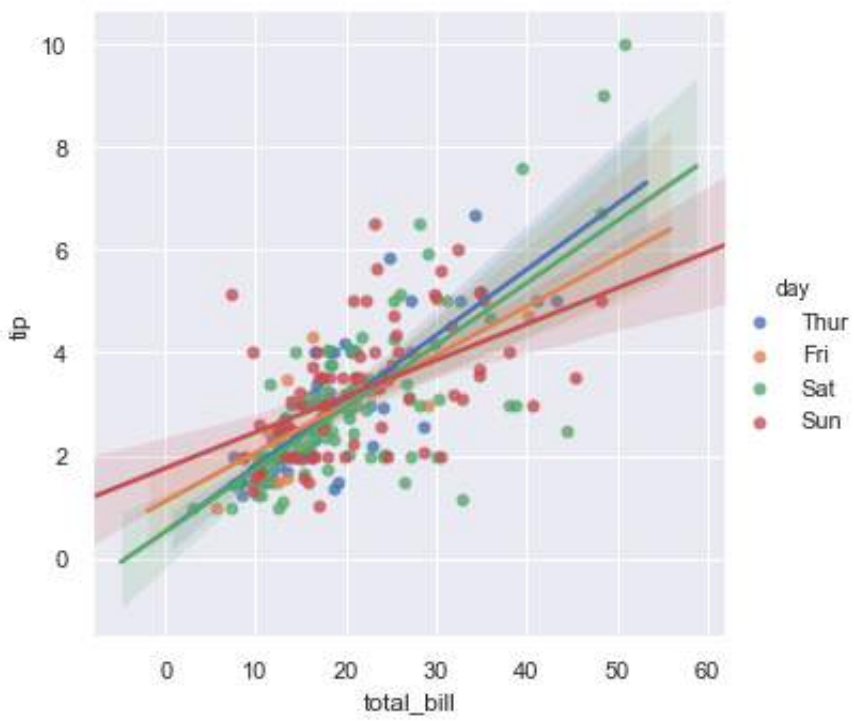


Figure 15.14: LM Plot

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：PyDataRoad）后台回复关键字“code”获取。

> 原文：
>
> Data Visualisation Using Seaborn

16 Seaborn 可视化：图形个性化设置的几个小技巧

16.1 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

16.2 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))

sns.heatmap(corr,cmap='RdBu', linewidths = 0.05, ax = ax)

# 设置Axes的标题
ax.set_title('Correlation between features')

f.savefig('sns_style_origin.jpg', dpi=100, bbox_inches='tight')
```

图片显示效果如下：

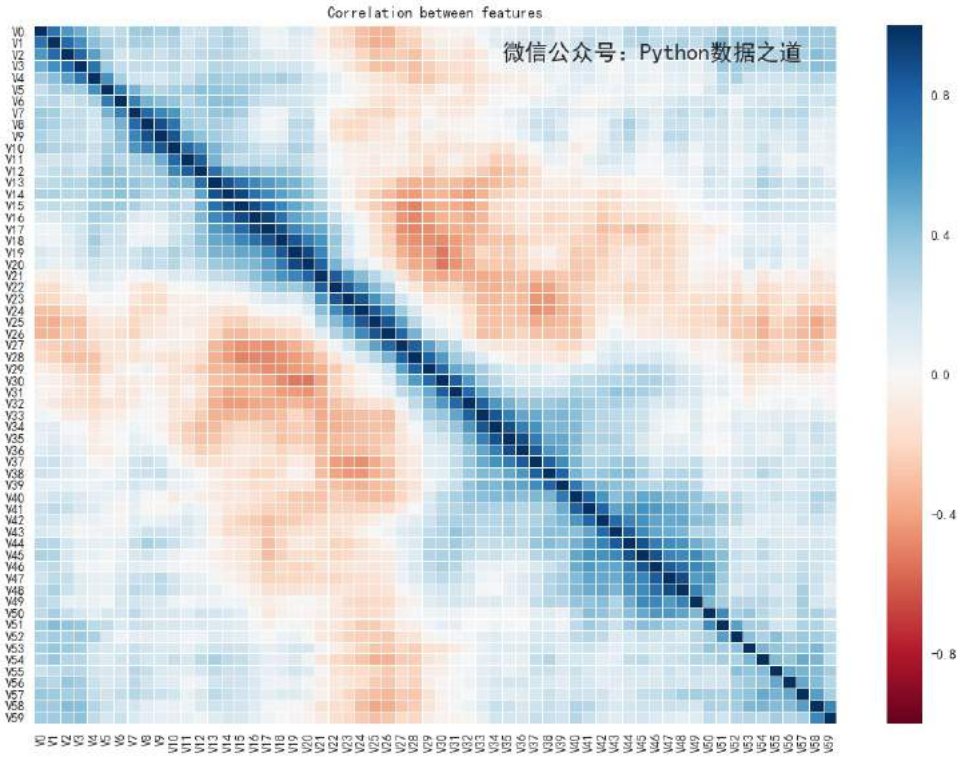


Figure 16.1

seaborn 制图的默认效果其实还是不错的。

16.3 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：

```
# 将y轴或x轴进行逆序
ax.invert_yaxis()
# 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')
```

```
20
21 # 设置坐标轴刻度的字体大小
22 # matplotlib.axes.Axes.tick_params
23 ax.tick_params(axis='y',labelsize=8) # y轴
24 # ax.tick_params(axis='x',labelsize=6, colors='b', labeltop=True, labelbottom=
       False) # x轴
25
26 # 将x轴刻度放置在top位置的几种方法
27 # ax.xaxis.set_ticks_position('top')
28 ax.xaxis.tick_top()
29 # ax.tick_params(axis='x',labelsize=6, colors='b', labeltop=True, labelbottom=
       False) # x轴
30
31 # 修改tick的字体颜色
32 # ax.tick_params(axis='x', colors='b') # x轴
33
34 # 旋转轴刻度上文字方向的两种方法
35 ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)
36 # ax.set_xticklabels(corr.index, rotation=90)
37
38 # 单独设置y轴或x轴刻度的字体大小，调整字体方向
39 # ax.set_yticklabels(ax.get_yticklabels(),fontsize=6)
40 # ax.set_xticklabels(ax.get_xticklabels(), rotation=-90)
41
42 f.savefig('sns_style_update.jpg', dpi=100, bbox_inches='tight')
```

图形显示效果如下：

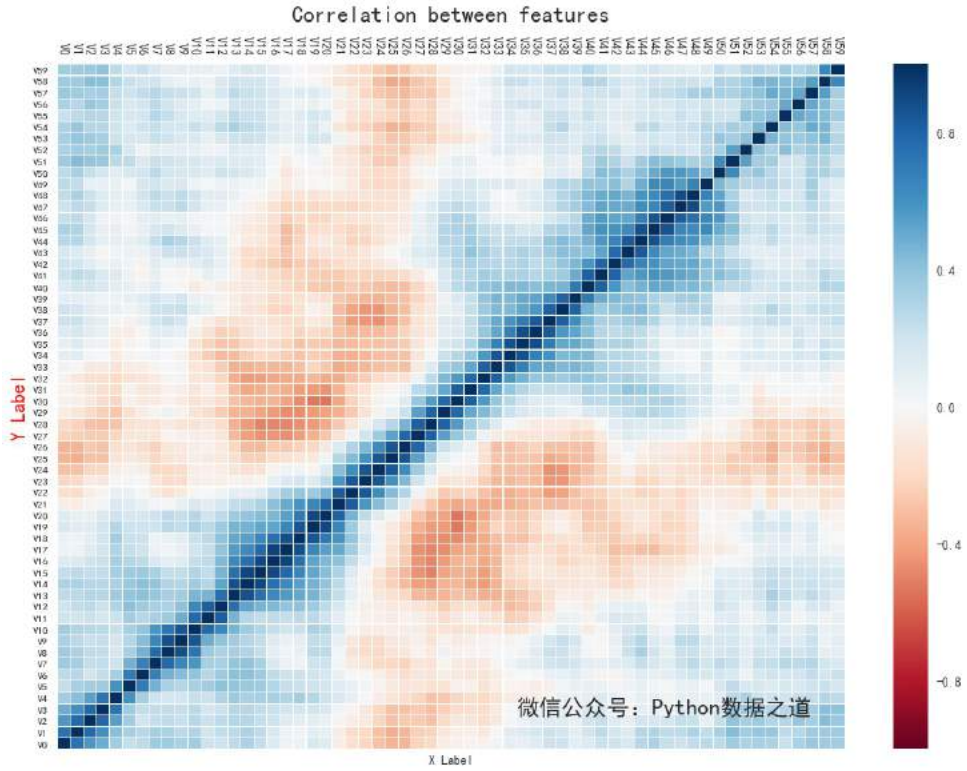


Figure 16.2

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。

17 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

17.1 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

Figure 17.1

```
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	5706	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	6698	7922	6869	6007	5963	7537	...	5434	8764	4049	9007	8363	2800
Cement	2677	5759	5631	6666	9029	6712	7972	2894	7288	10161	...	9143	8585	2941	4523	4447	8909

5 rows × 40 columns

Figure 17.2

```
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)
```

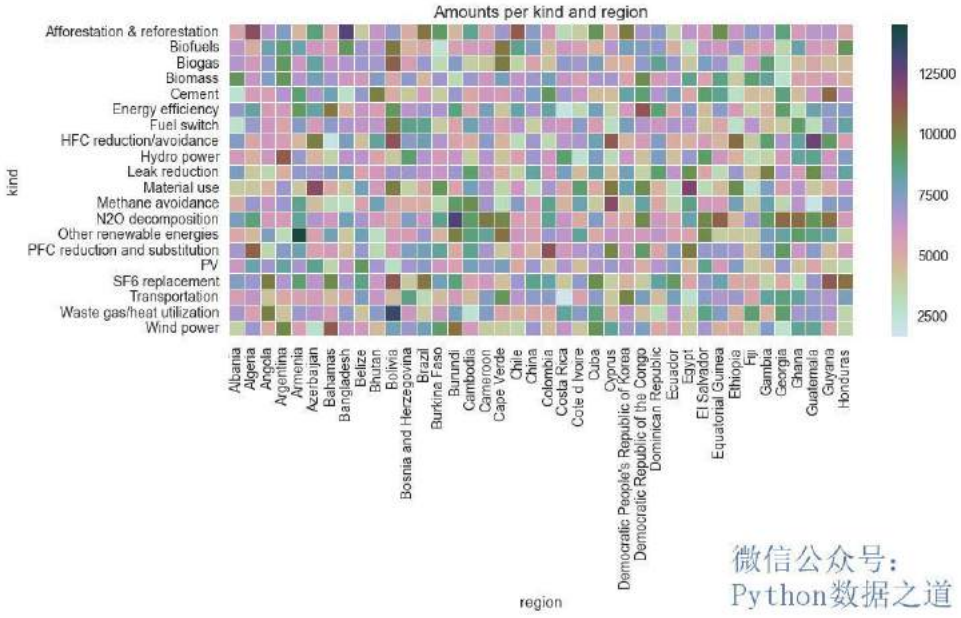


Figure 17.3

17.2 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

17.2.1 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')
```

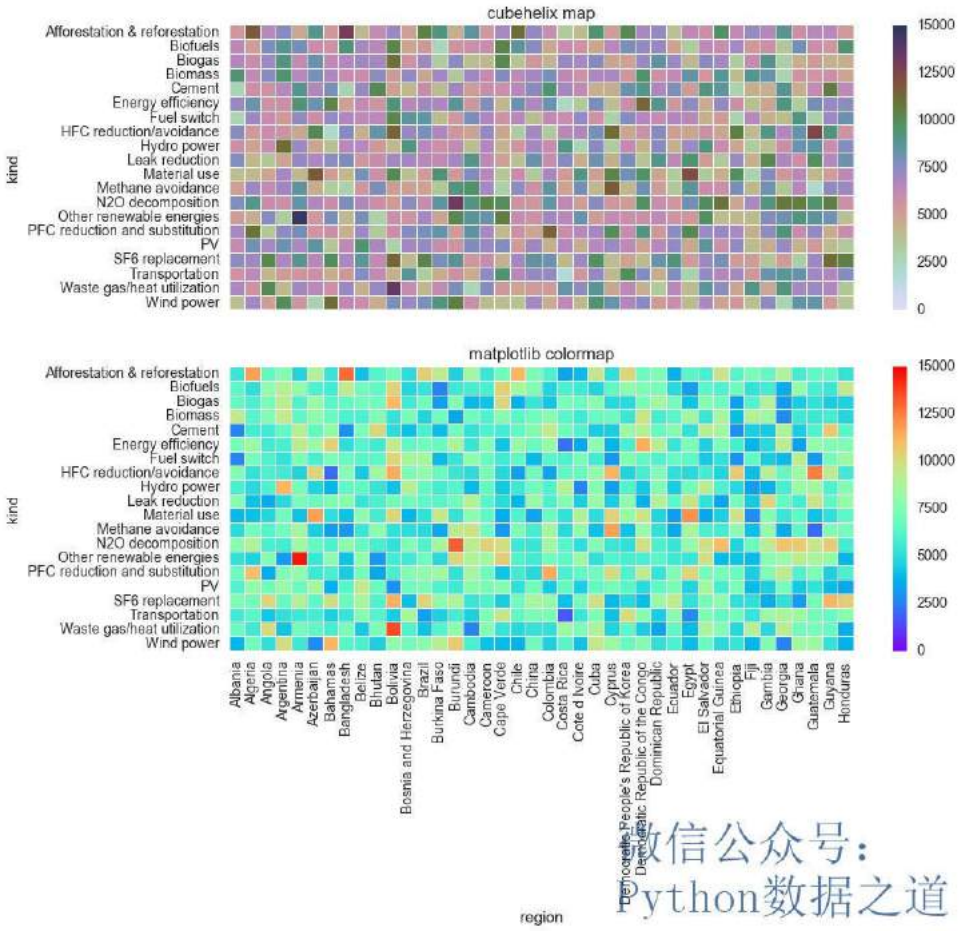


Figure 17.4

17.2.2 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')
```

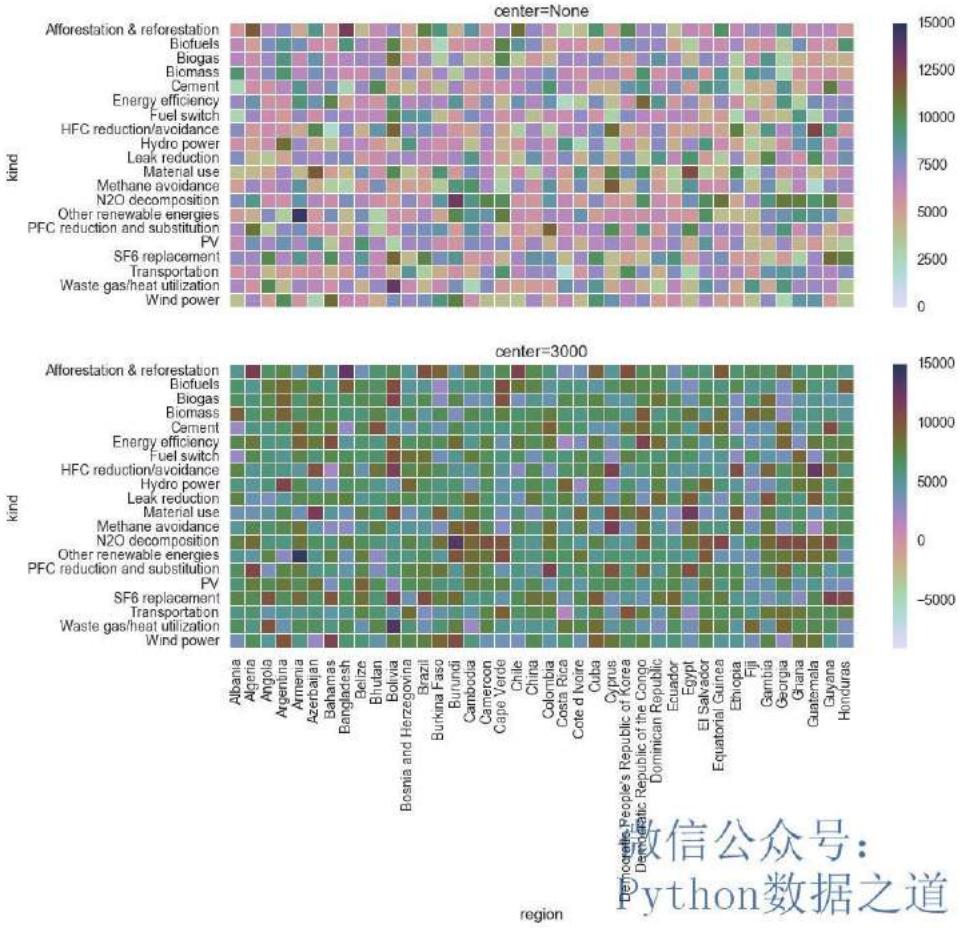


Figure 17.5

17.2.3 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')
```

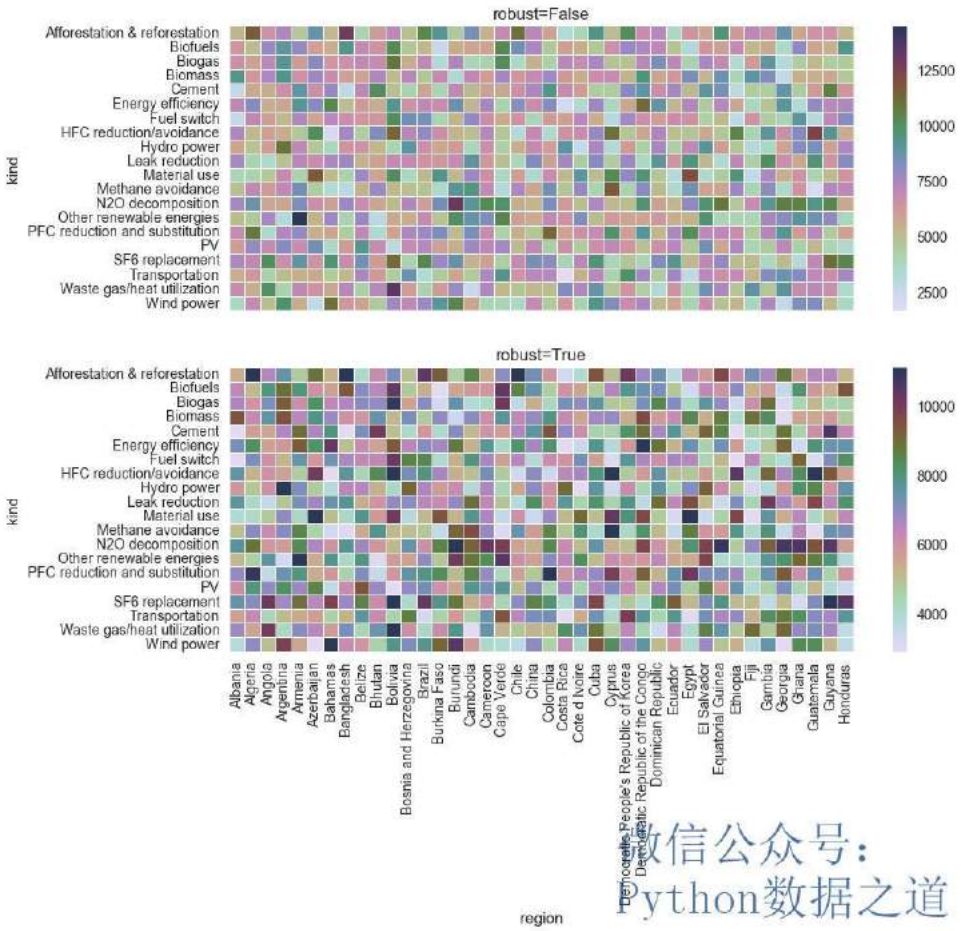


Figure 17.6

17.2.4 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')
```

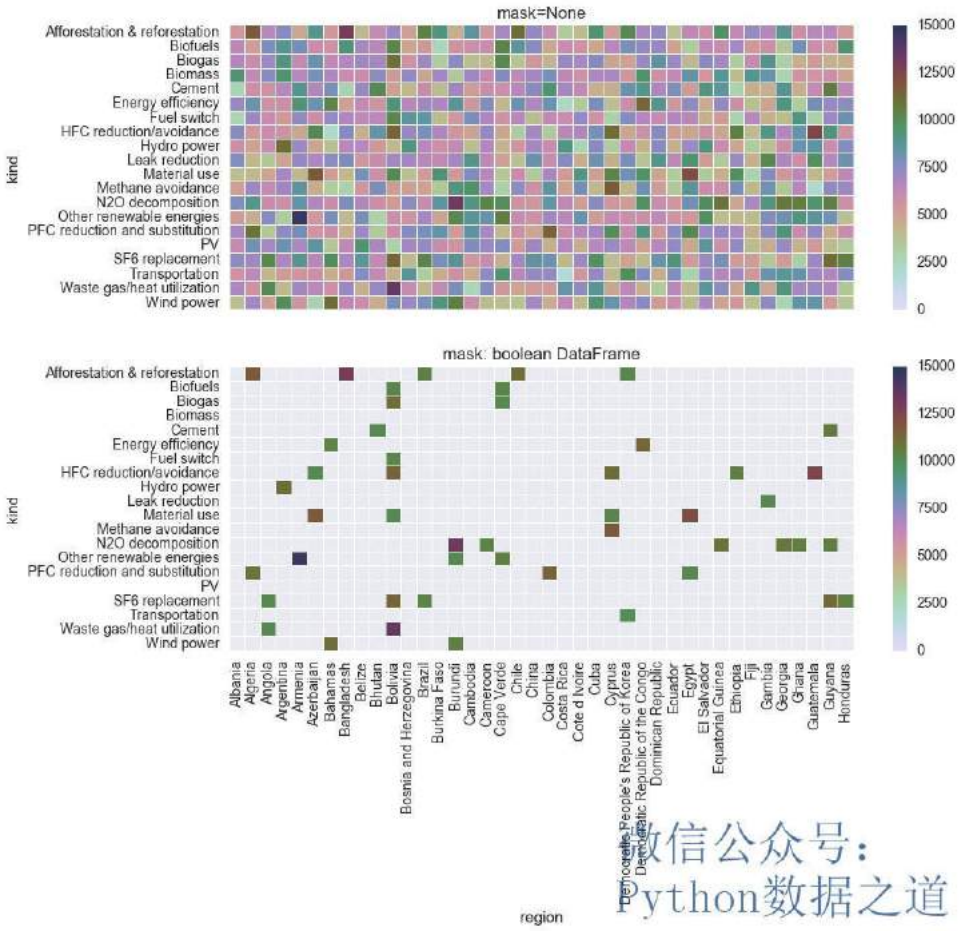


Figure 17.7

17.2.5 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')
```

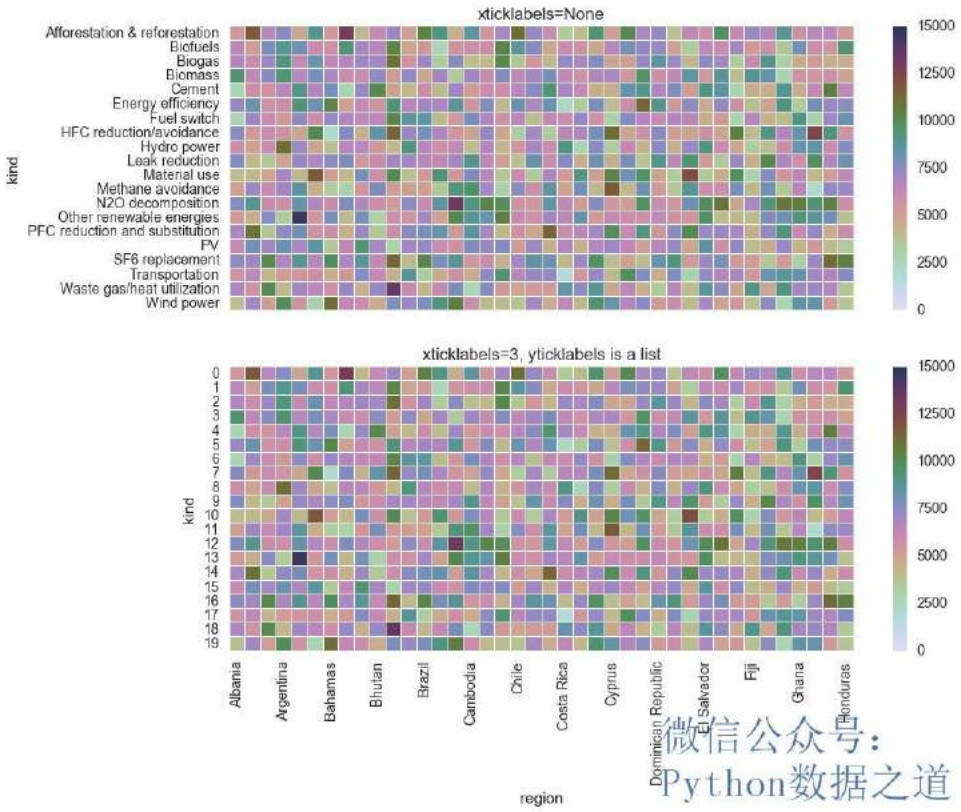


Figure 17.8

17.2.6 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')
```

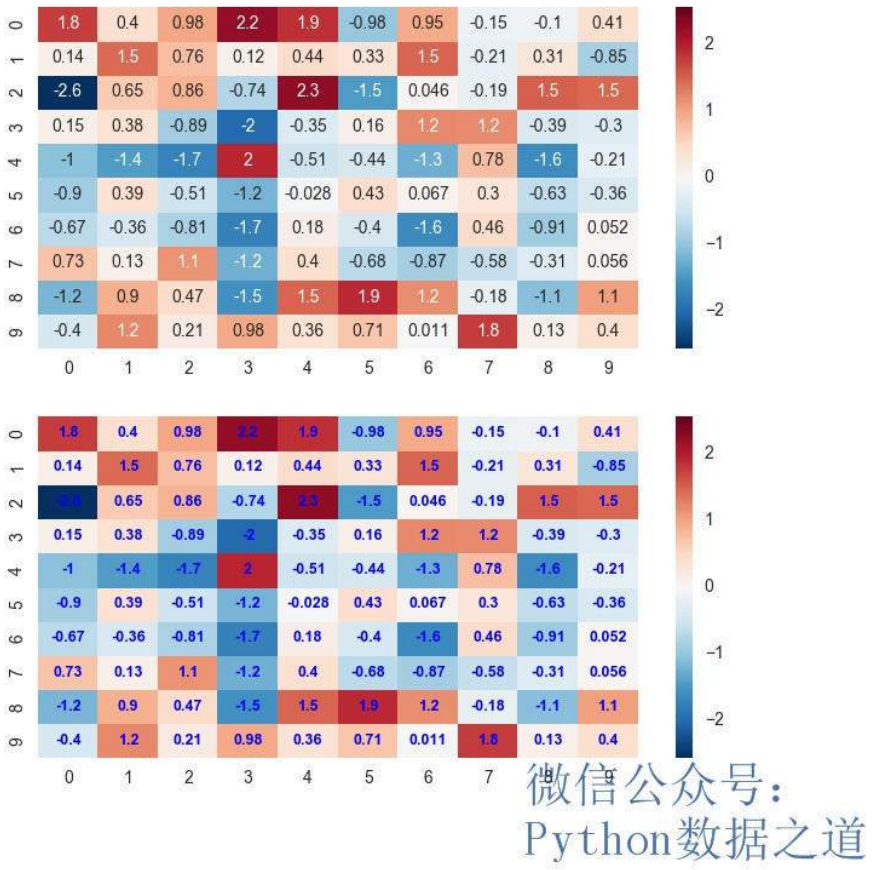


Figure 17.9

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

17.2.7 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')
```

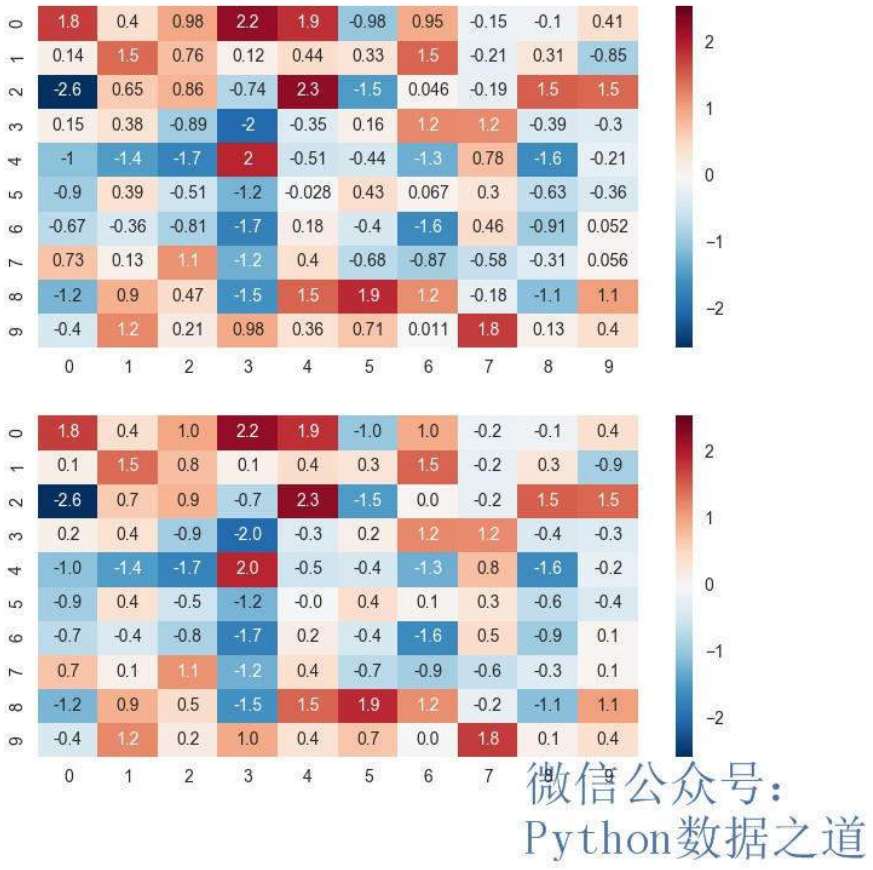


Figure 17.10

17.3 案例应用：突出显示某些数据

17.3.1 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')
```

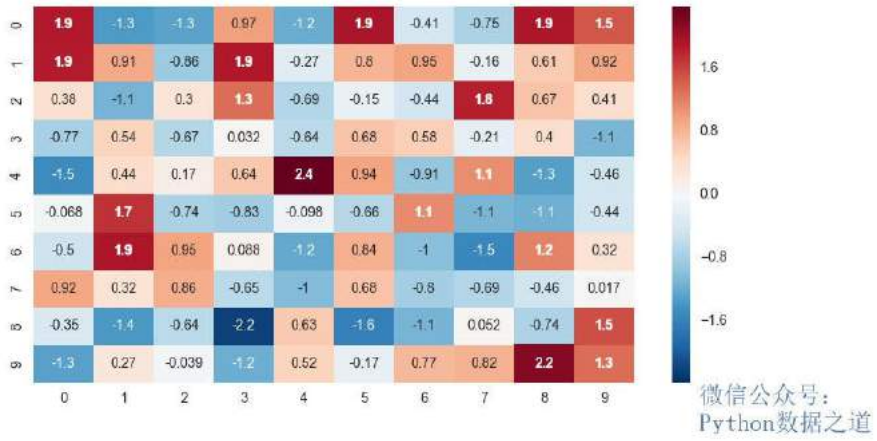


Figure 17.11

17.3.2 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')
```

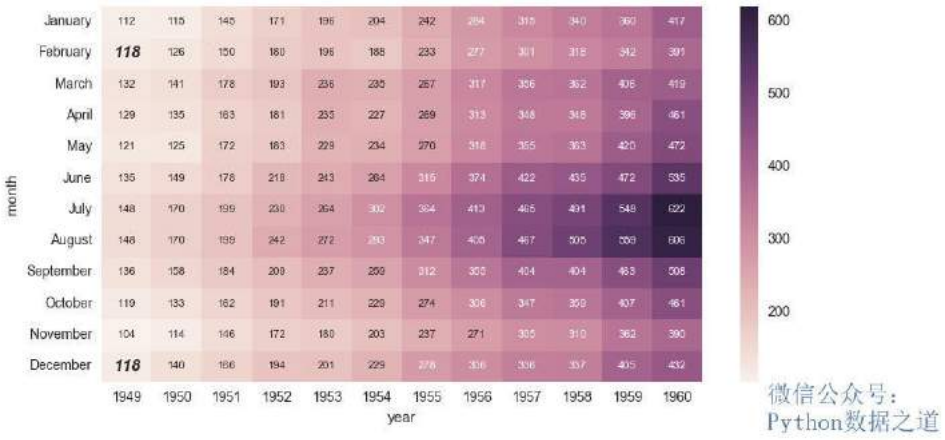


Figure 17.12

> 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。

18 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

18.1 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()

18.2 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

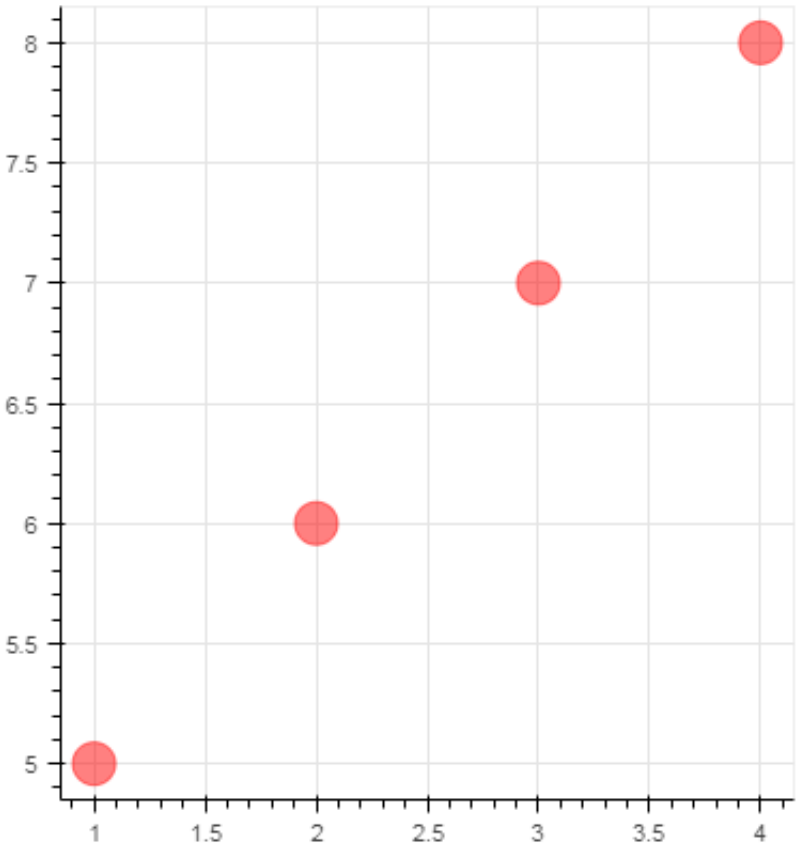


Figure 18.1

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)
```

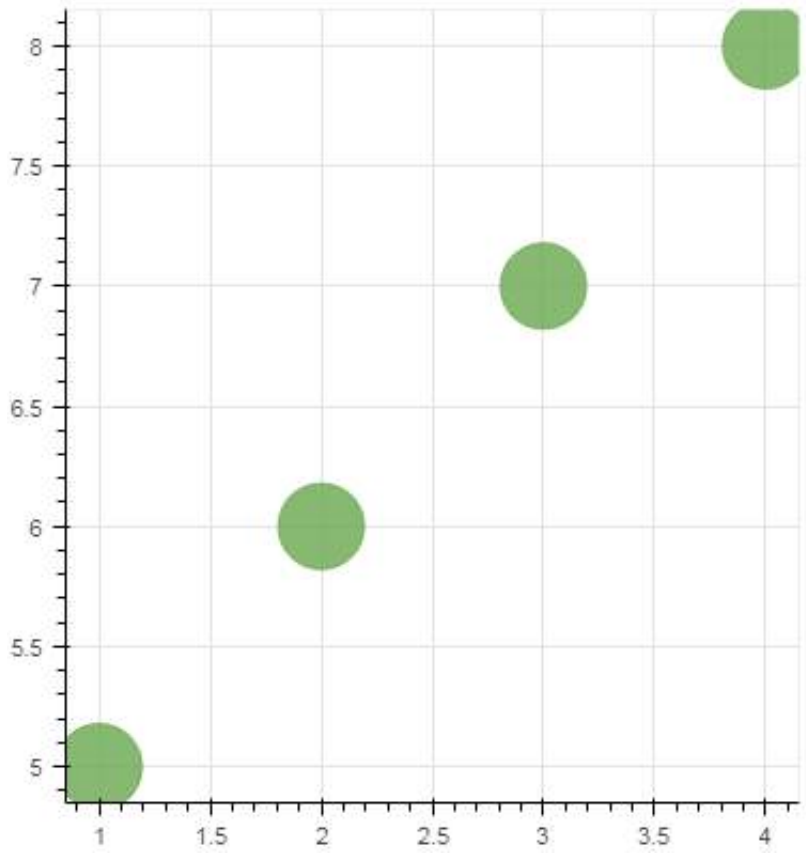


Figure 18.2

`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)
```

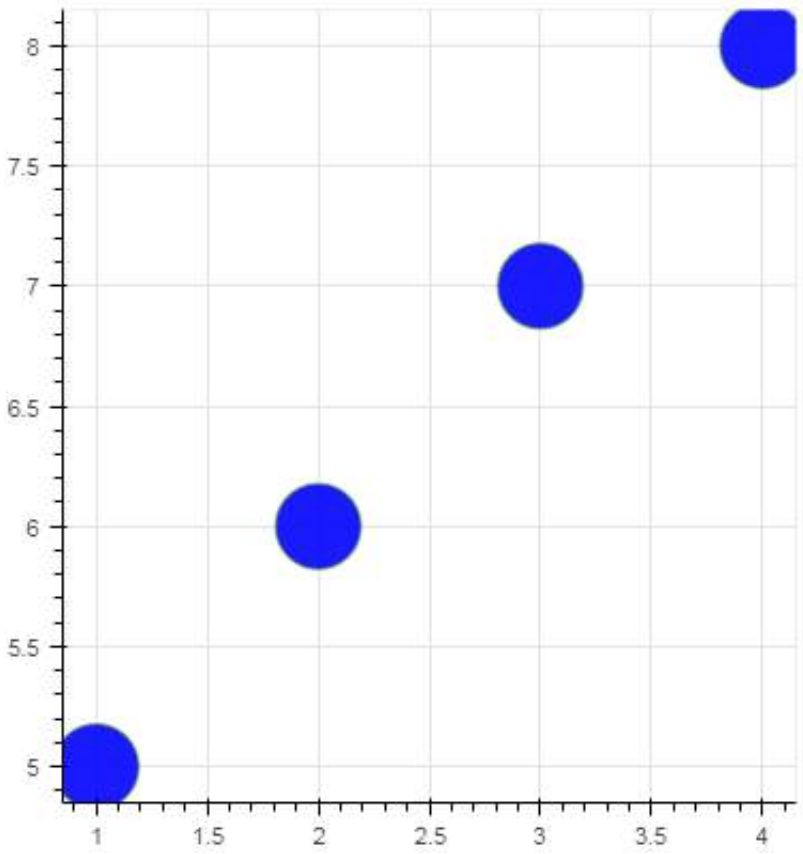


Figure 18.3

- `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)
```

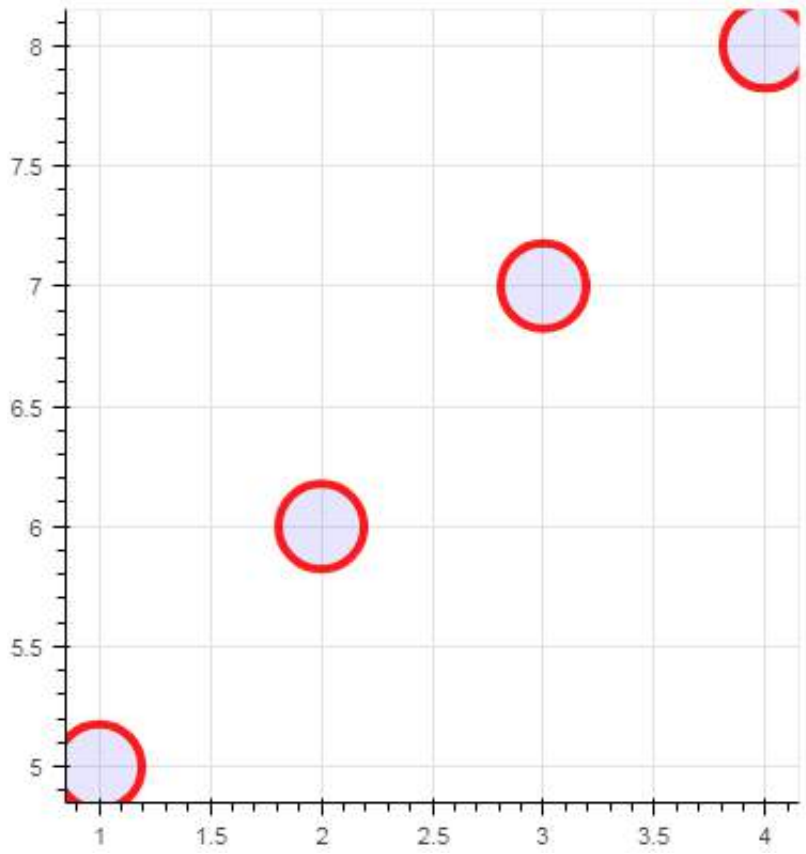


Figure 18.4

`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')

```

```
8  show(p)
```

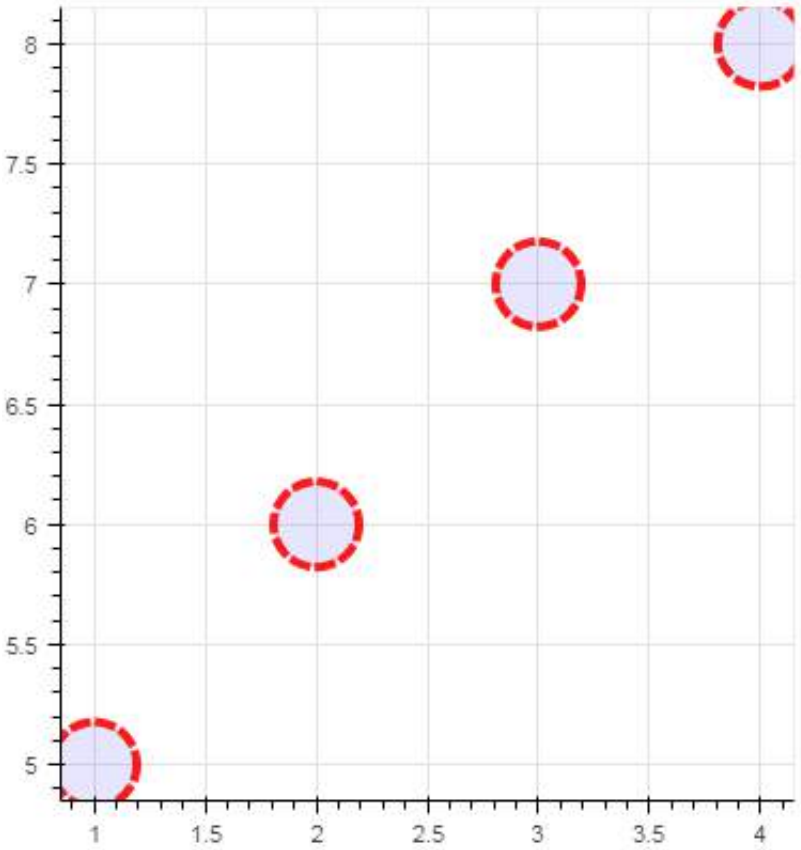


Figure 18.5

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：PyDataRoad），接收我的更新通知。

19 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

19.1 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()
```

19.1.1 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：

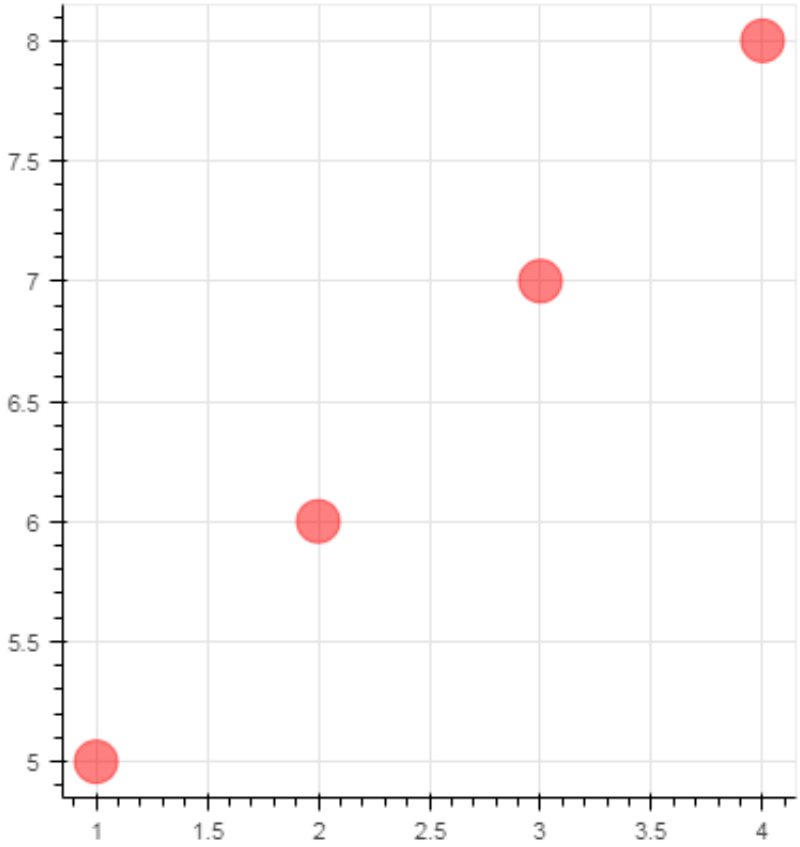


Figure 19.1

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)
```

19.1.2 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"
tools = "crosshair,pan,wheel_zoom,reset,save"


p = figure(width=400, height = 400, tools=tools)

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

show(p)
```

图示如下：

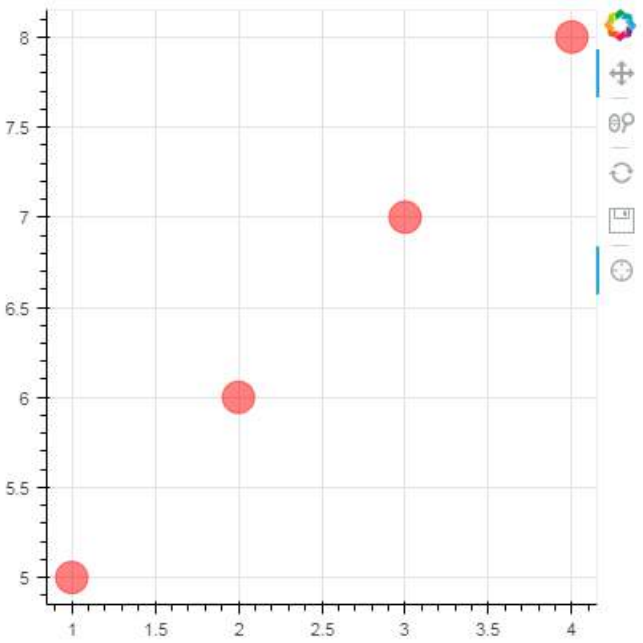


Figure 19.2

19.1.3 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”

```
tools = "crosshair,pan,wheel_zoom,reset"


p = figure(width=400, height = 400, tools=tools, toolbar_location='above')

p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)

```

```
8 show(p)
```

图示如下：

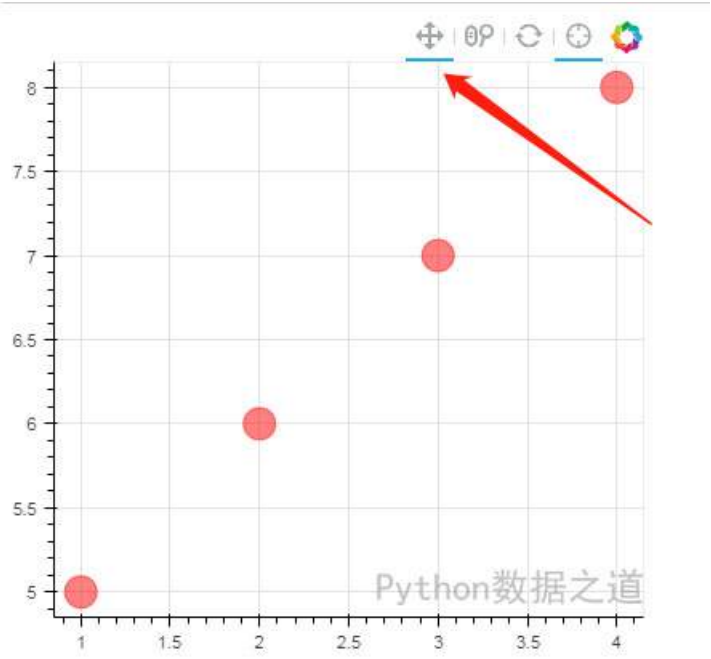


Figure 19.3

19.1.4 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：

```
1 p = figure(width=400, height = 400,
2           x_minor_ticks=2, y_minor_ticks=10)
3
4 p.circle([1,2,3,4],[5,6,7,8],size=20, color='red', alpha=0.5)
5
6 show(p)
```

图示如下：

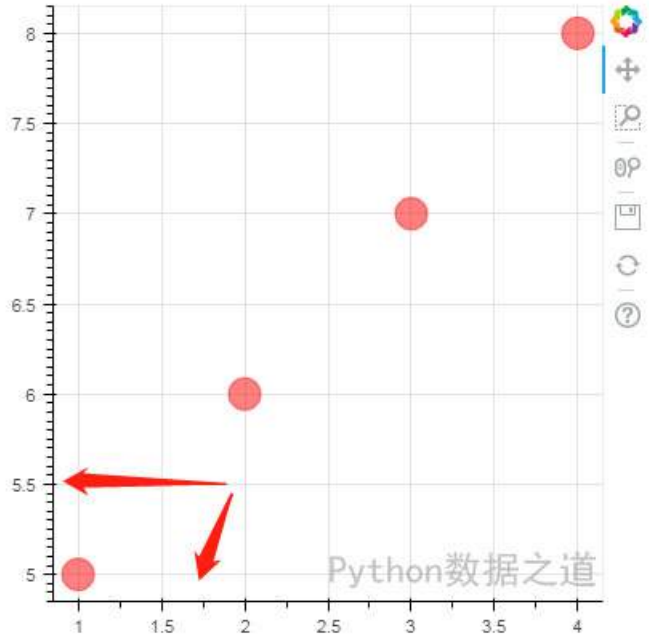


Figure 19.4

19.1.5 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:

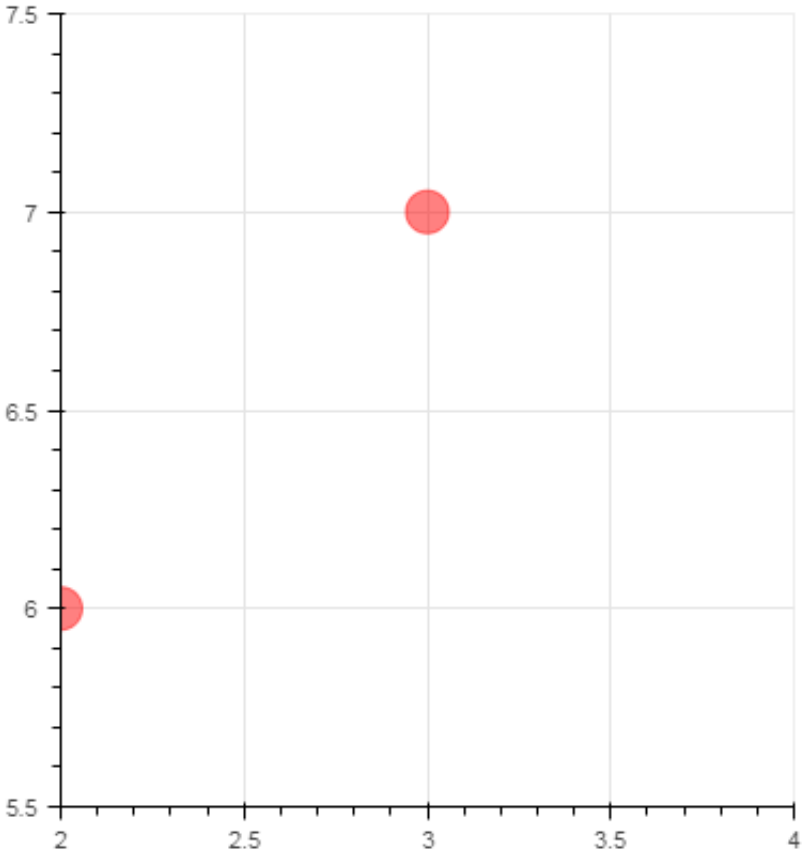


Figure 19.5

19.1.6 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：

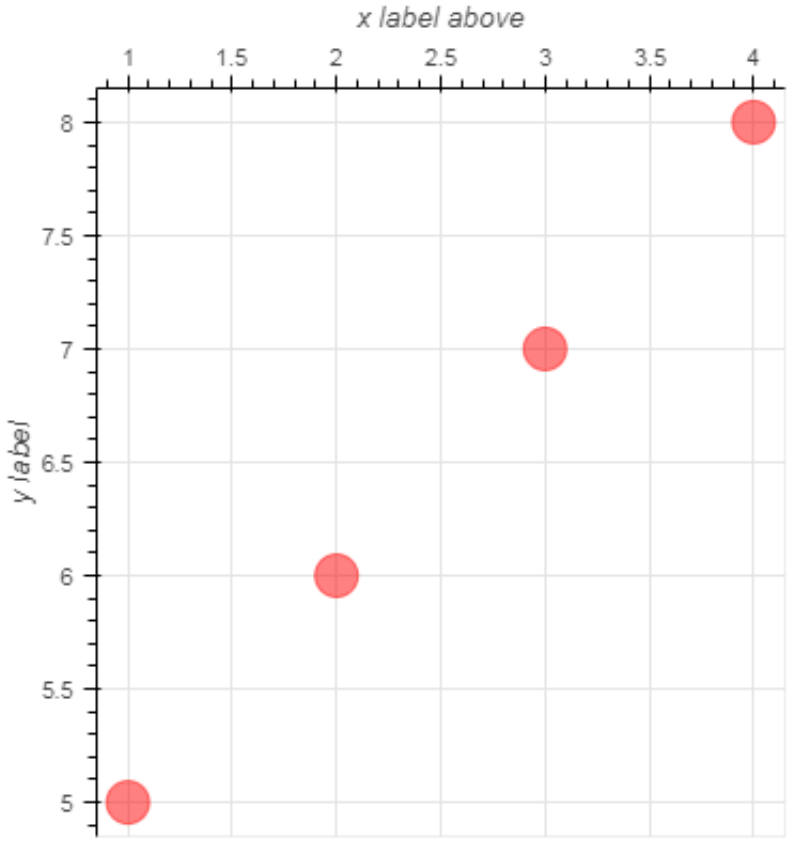


Figure 19.6

19.1.7 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：

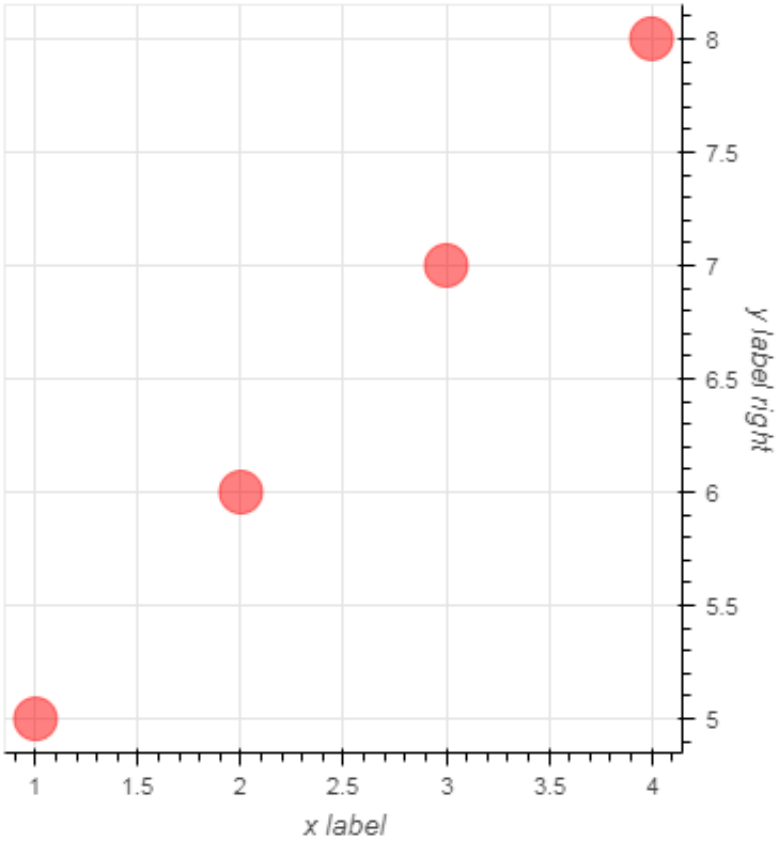


Figure 19.7

19.1.8 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]
```

```
4
5  p = figure(width=400, height = 400, title='datetime type example',
6             x_axis_label='x label datetime', x_axis_location='below',
7             y_axis_label='y label', y_axis_location='left',
8             y_axis_type='linear', x_axis_type='datetime')
9  # y_axis_type, "auto" 和 "linear" 的效果是一样的
10
11 p.line(rng,y,color='red',line_width=3, alpha=0.9)
12
13 show(p)
```

图示如下：

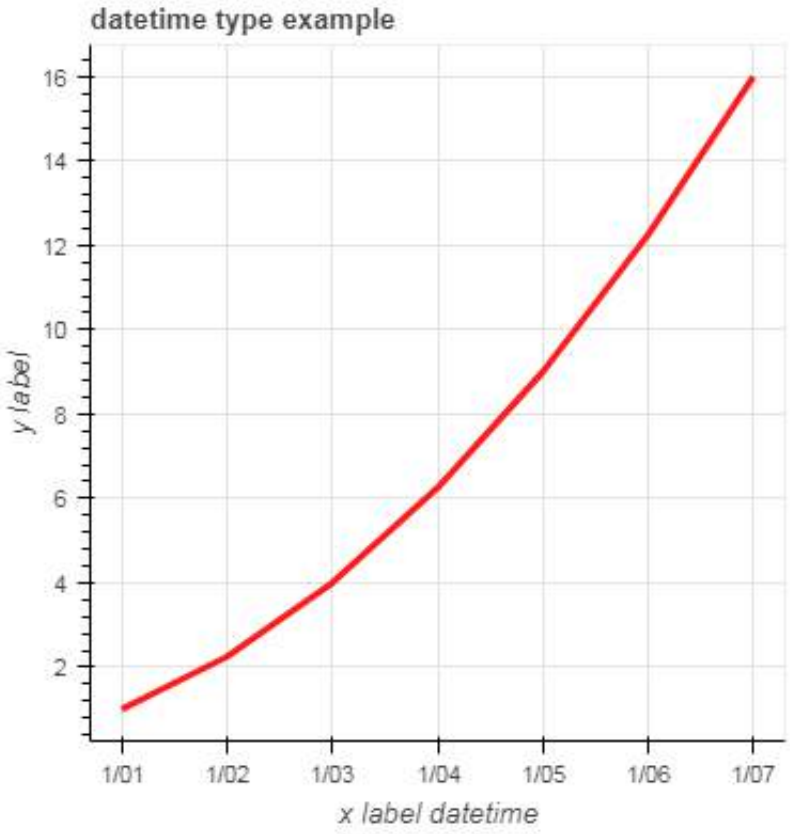


Figure 19.8

（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：

```
1  x = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
2  y = [10**(i**2) for i in x]
3
4  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)

show(p)
```

图示如下：

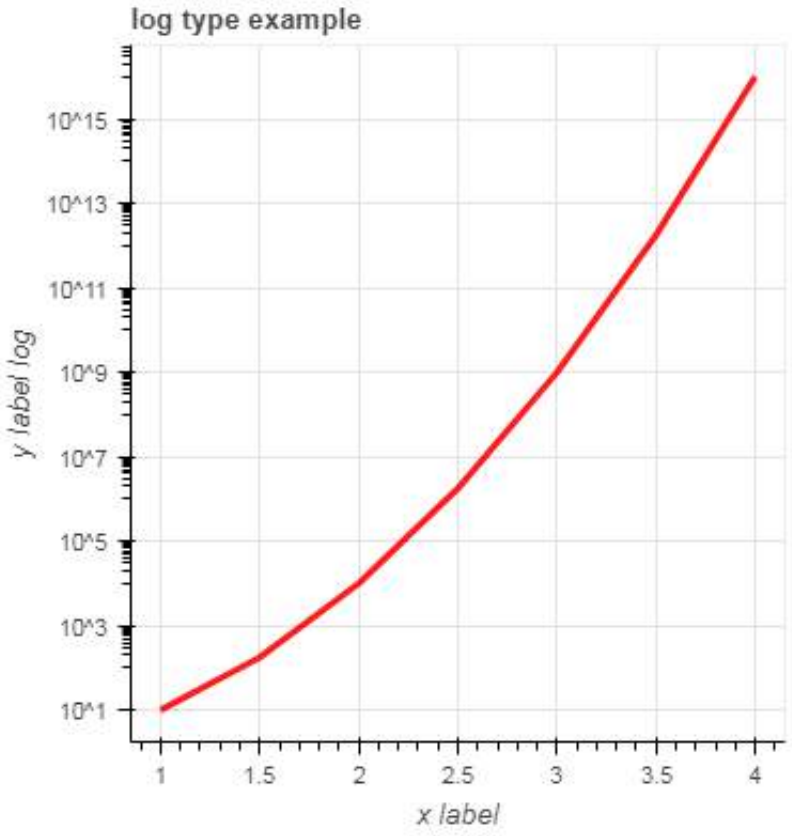


Figure 19.9

（3）x_axis_type, y_aixs_type: “mercator”

```
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='example',
            x_axis_label='x label', x_axis_location='below',
            y_axis_label='y label', y_axis_location='left' )

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：

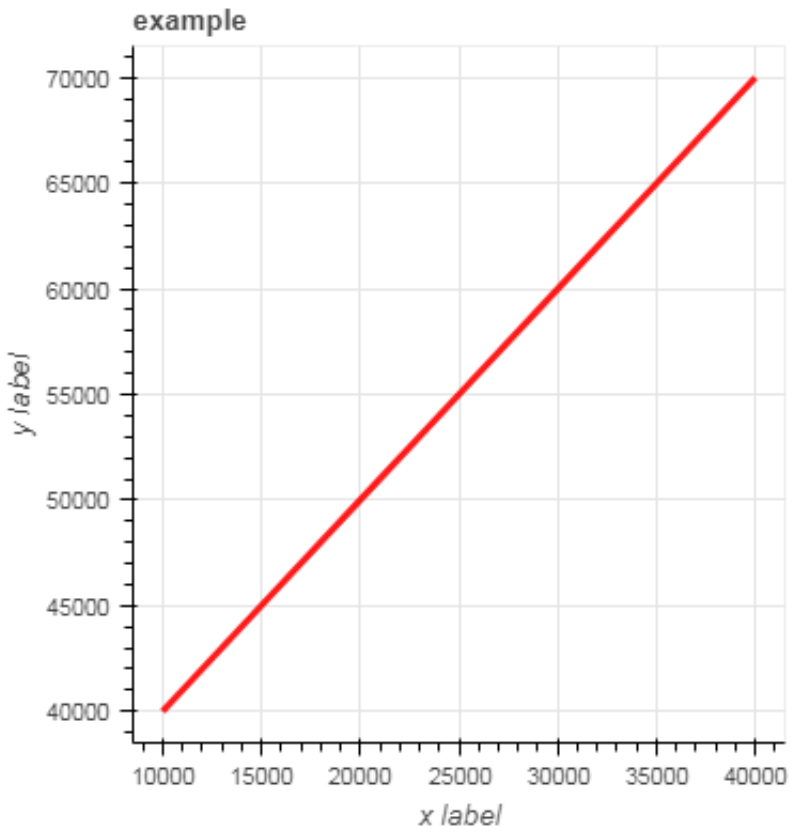


Figure 19.10

```
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='mercator type example',
6             x_axis_label='x label', x_axis_location='below',
7             y_axis_label='y label', y_axis_location='left',
8             x_axis_type='mercator', y_axis_type='mercator')
9
10 p.line(x,y,color='red',line_width=3, alpha=0.9)
11
12 show(p)
```

设置为 mercator 后的图示如下：

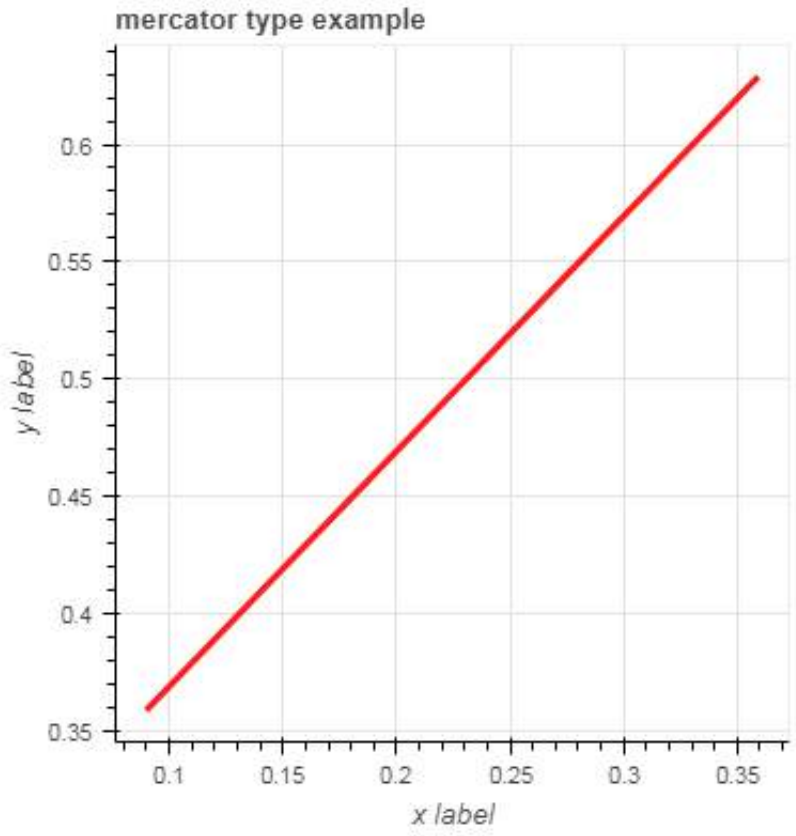


Figure 19.11

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

19.2 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。

20 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

20.1 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)
```

20.1.1 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')

p2 = figure(title='circle_cross')
p2.circle_cross(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p3 = figure(title='circle_x')
p3.circle_x(x,y,size=20, color='#0071c1',fill_alpha=0.2, line_width=2)

p4 = figure(title='cross')
p4.cross(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:

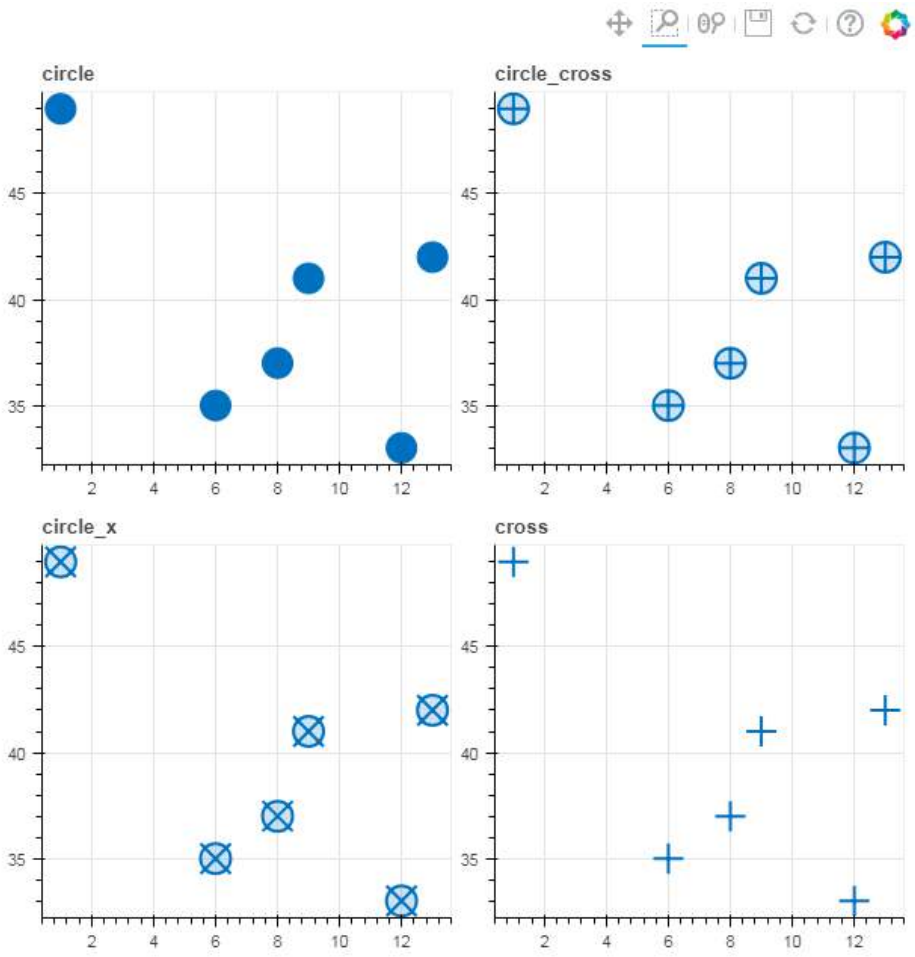


Figure 20.1

20.1.2 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：

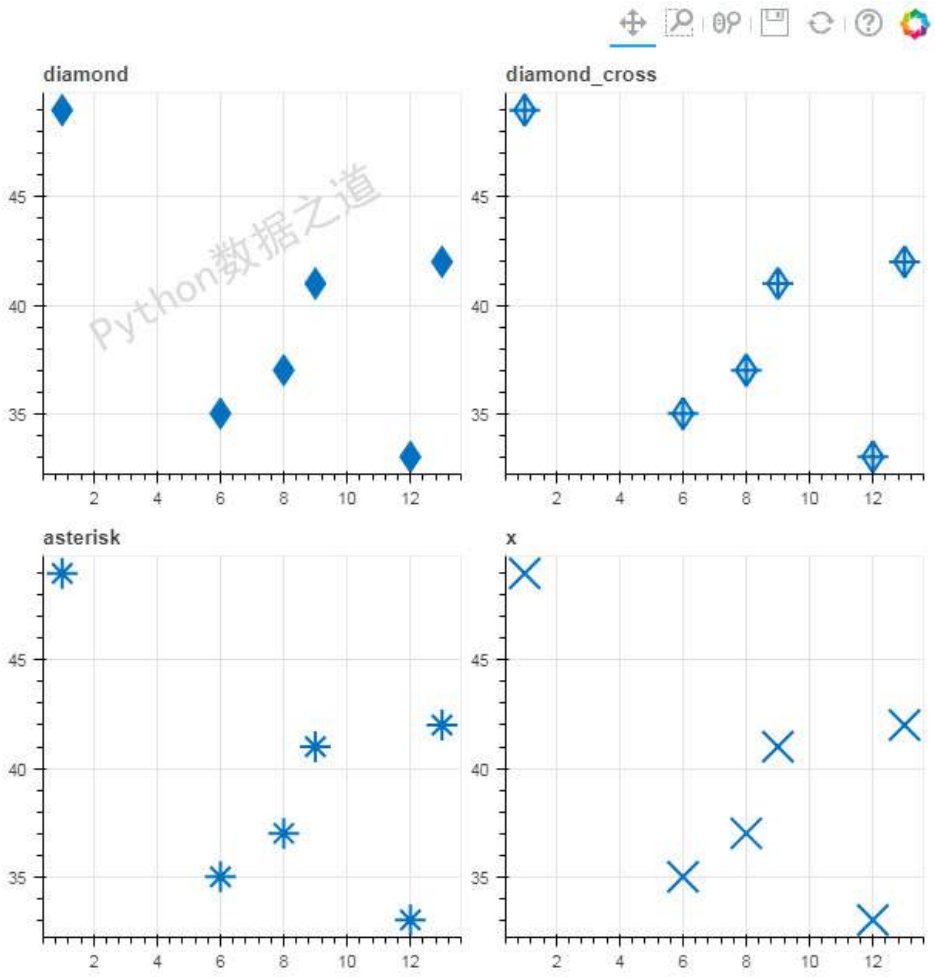


Figure 20.2

20.1.3 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)

```

```
13 show(grid)
```

图示如下：

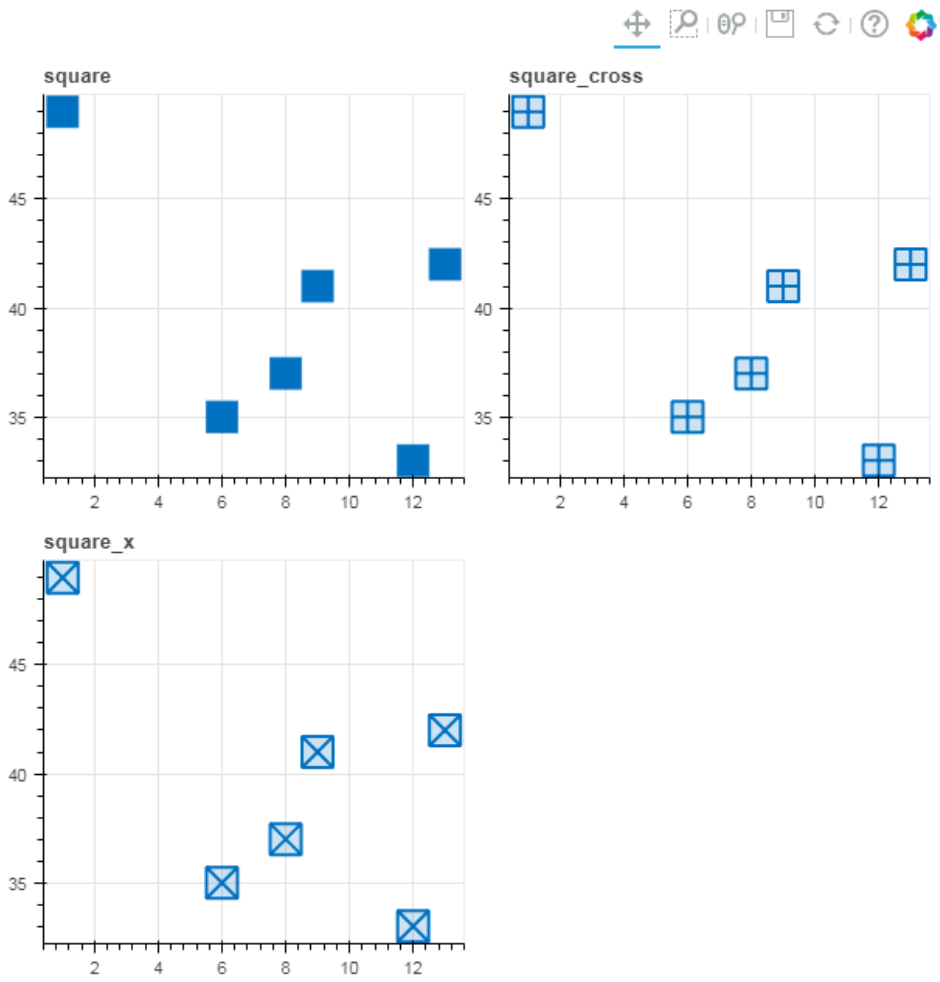


Figure 20.3

20.1.4 triangle, inverted_triangle

接下来，是三角形的绘制

```
1 p3 = figure(title='triangle')
2 p3.triangle(x,y,size=20, color='#0071c1', line_width=2)
3
4 p4 = figure(title='inverted_triangle')
5 p4.inverted_triangle(x,y,size=20, color='#0071c1', line_width=2)
6
7 grid=gridplot([p3,p4],ncols=2, plot_width=300,plot_height=300)
8
```

```
show(grid)
```

图示如下:

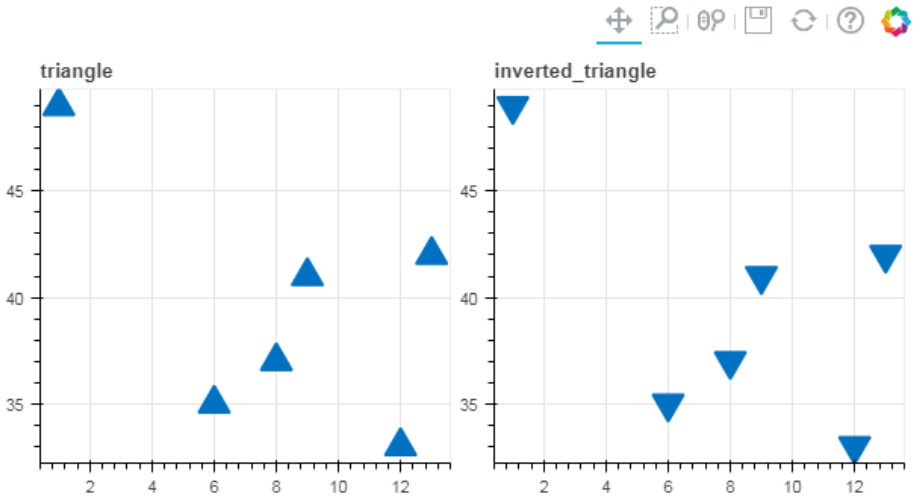


Figure 20.4

20.1.5 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)

p4 = figure(title='oval')
p4.oval(x,y,color='#0071c1', width=2, height=3.6, angle=30)


grid=gridplot([p1,p2,p3,p4],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下:

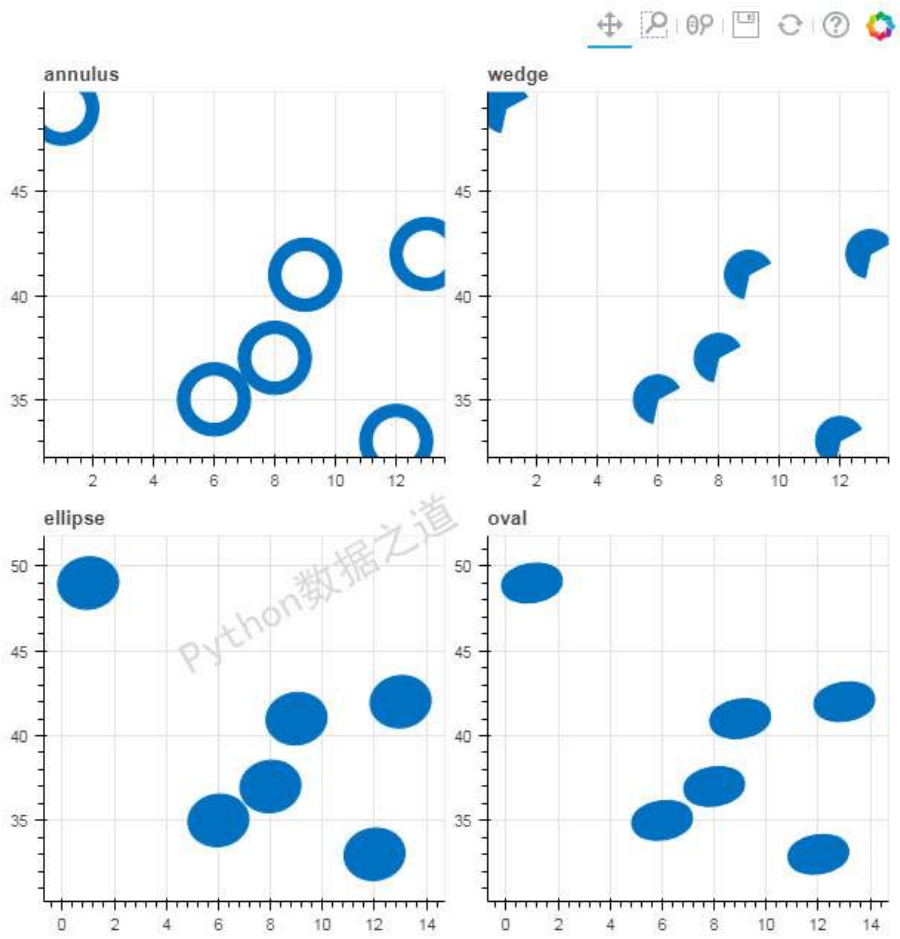


Figure 20.5

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。

20.1.6 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。

```
p2 = figure(title='hbar')
p2.hbar(y=y, height=0.3, left=0, right=x, color='#0071c1')

p3 = figure(title='vbar')
p3.vbar(x=x, width=0.3, bottom=0, top=y, color='#0071c1')

grid=gridplot([p2,p3],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

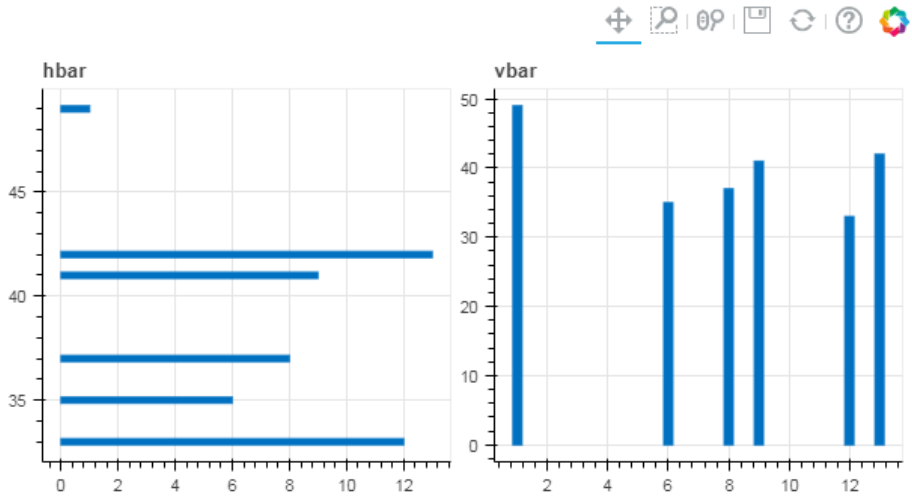


Figure 20.6

20.1.7 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。

```
p1 = figure(title='line')
p1.line(x,y,color='#0071c1', line_width=2)
p1.circle(x,y,size=10,color='#0071c1',fill_color='white')

from bokeh.plotting import figure, output_file, show

p2 = figure(title='multi_line')
p2.multi_line(xs=[[1, 2, 3], [2, 3, 4]], ys=[[6, 7, 2], [4, 5, 7]],
              color=['red','green'])

grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下：

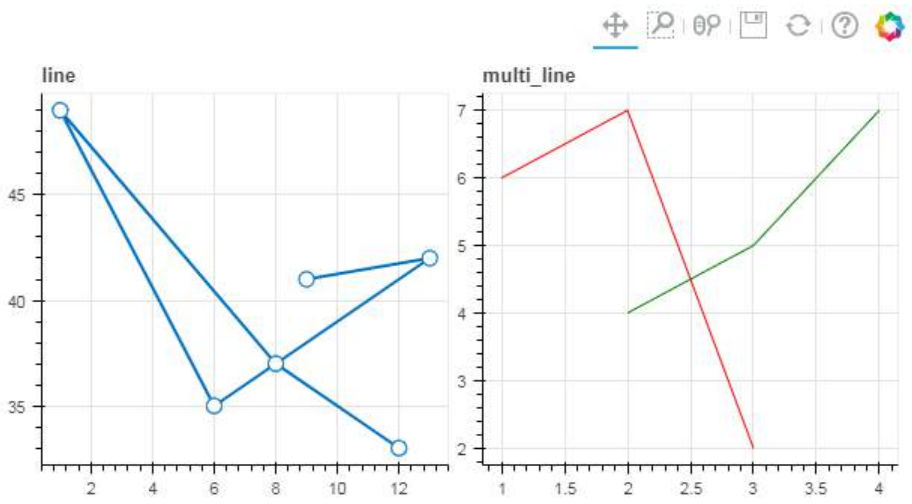


Figure 20.7

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

Figure 20.8

```
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,
```

```
8                   line_width=2)
9  show(p)
```

图示如下：

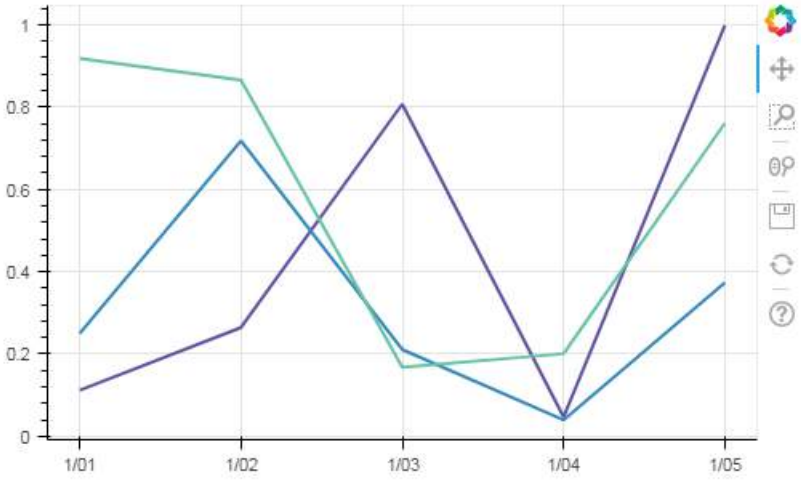


Figure 20.9

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。

20.1.8 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)

```
1  p1 = figure(title='patch')
2  p1.patch(x,y,color='#0071c1')
3
4
5  p2 = figure(title='patches')
6  p2.patches(xs=[[1, 2, 3, 4], [2, 3, 4]], ys=[[6, 7, 2, 3], [4, 5, 7]],
7             color=['red','green'])
8
9  grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)
10
11 show(grid)
```

图示如下：

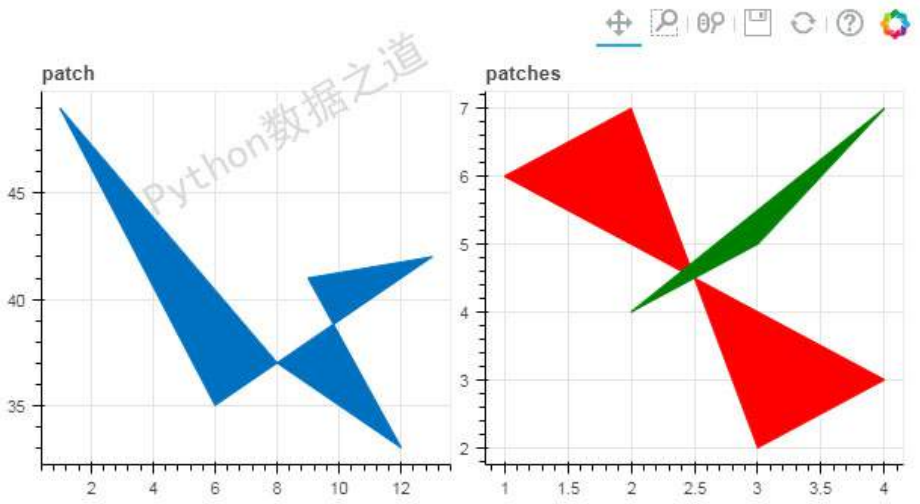


Figure 20.10

20.1.9 quad, quadratic

- quad(left, right, top, bottom, **kwargs)，四方的
- quadratic(x0, y0, x1, y1, cx, cy, **kwargs)，二次方程的

```
1  p1 = figure(title='quad')
2  p1.quad(left=[1,3,5],right=[2,4,7], top=[3,7,8], bottom=[2,3,4], color='#0071c1
       ')
3
4
5  p2 = figure(title='quadratic')
6  p2.quadratic(x0=[1,3,5],x1=[2,4,7], y1=[3,7,8], y0=[2,3,4], cx=8 , cy=2, color=
       '#0071c1')
7
8  grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)
9
10 show(grid)
```

图示如下：

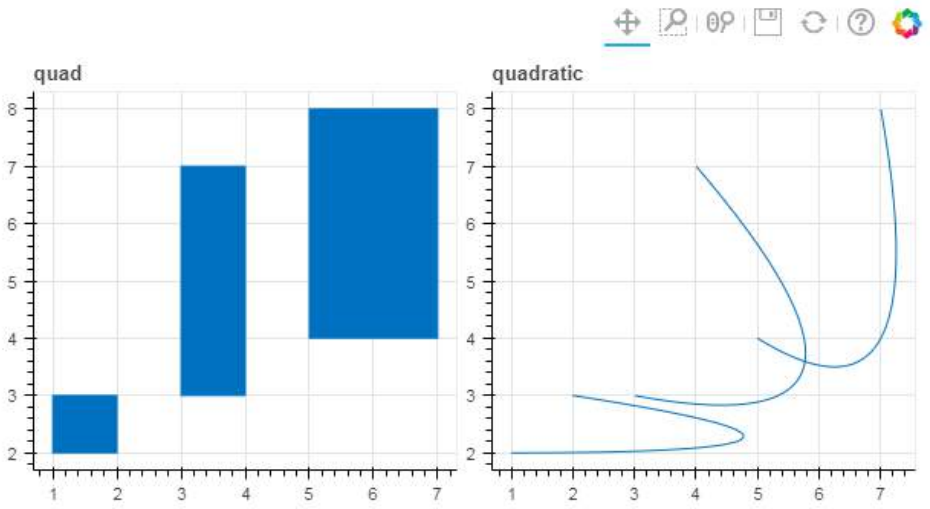


Figure 20.11

20.1.10 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：

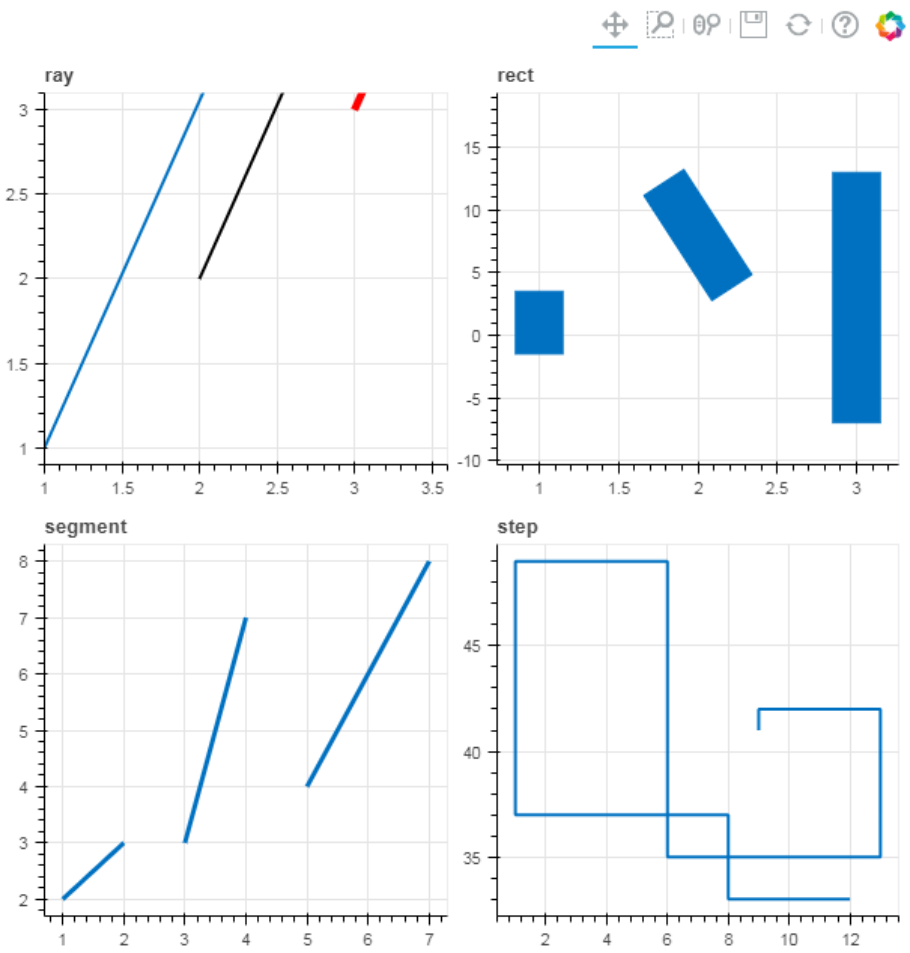


Figure 20.12

20.2 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。

21 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

21.1 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：

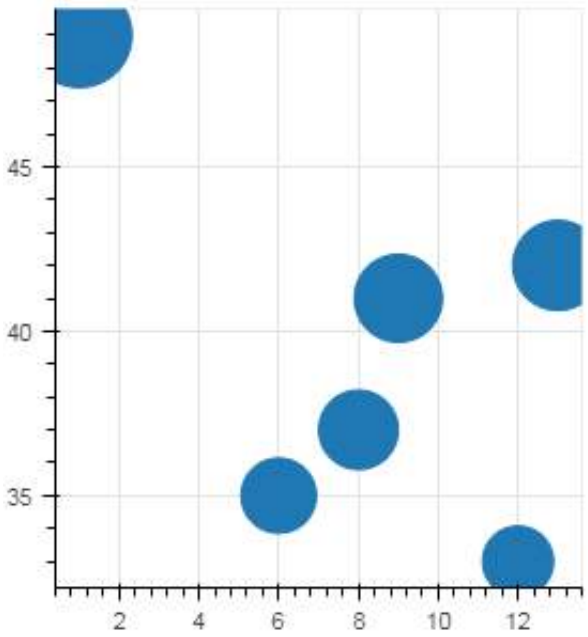


Figure 21.1

21.2 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

21.2.1 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：

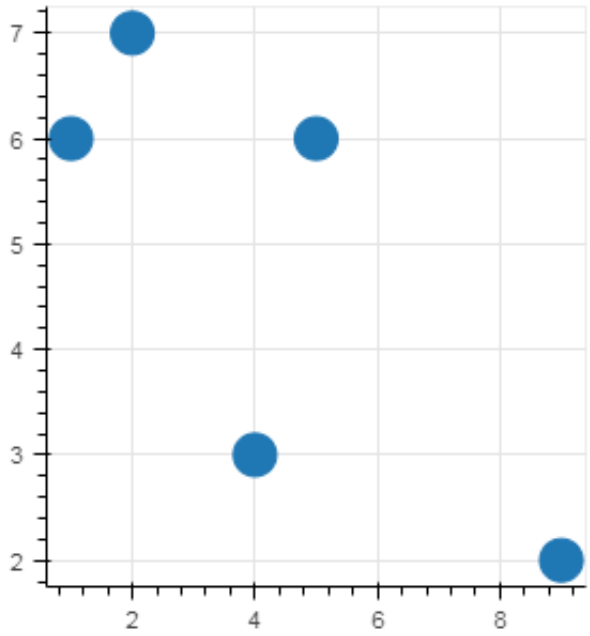


Figure 21.2

21.2.2 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：

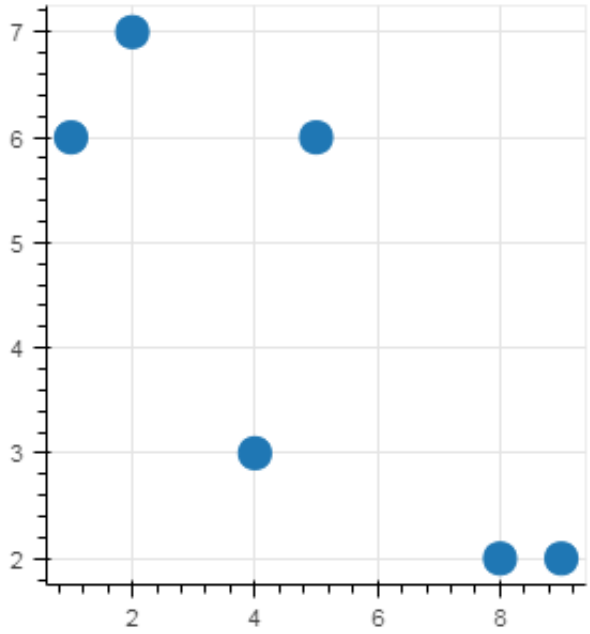


Figure 21.3

```
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)
show(p)
```

图示如下：

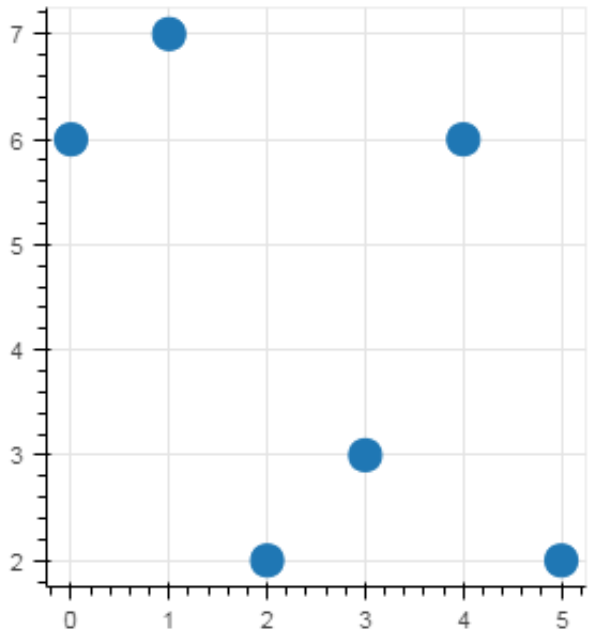


Figure 21.4

21.2.3 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。

```
dates = pd.date_range('20180101', periods=360)

df = pd.DataFrame(np.random.randn(360,2), index=dates, columns=list('AB'))

df['C'] = ['Good', 'Bad', 'Common', 'Good','Good']*72
df['month'] = df.index.month

df
```

图示如下：

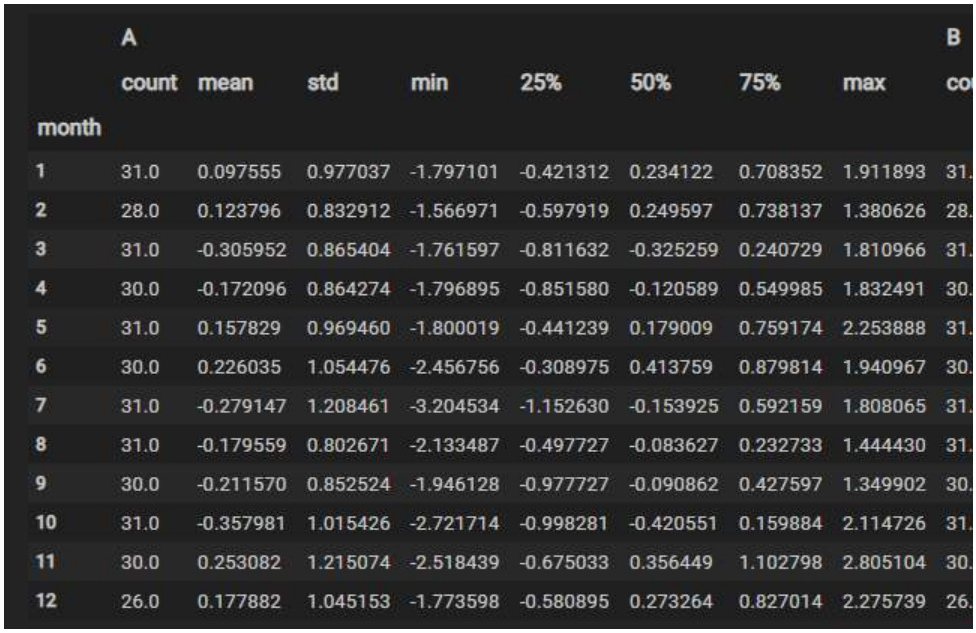
	A	B	C	month
2018-01-01	0.107071	2.460188	Good	1
2018-01-02	-0.517491	0.213835	Bad	1
2018-01-03	-1.444430	-0.080809	Common	1
2018-01-04	-0.072208	0.907390	Good	1
2018-01-05	0.359186	0.888204	Good	1

Figure 21.5

```
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.

Figure 21.6

```
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：

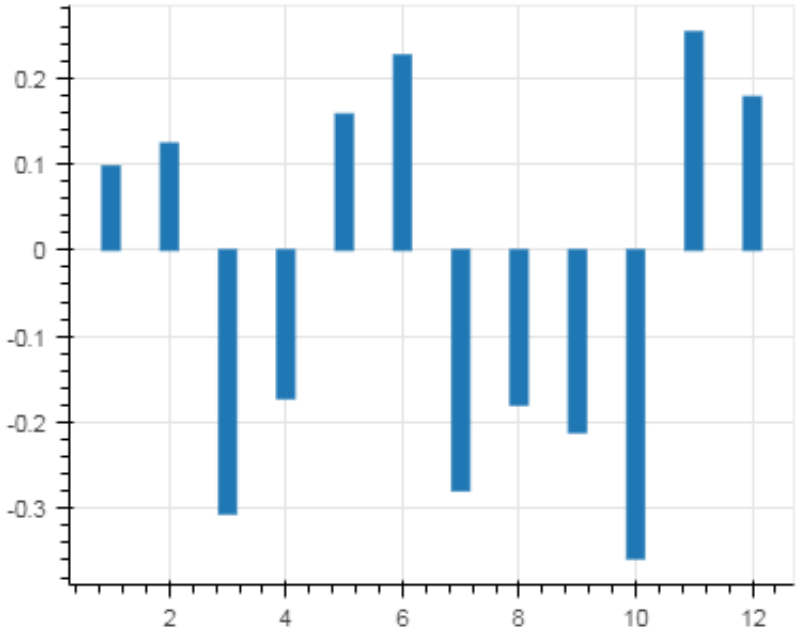


Figure 21.7

21.3 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。

22 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()
```

22.1 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：

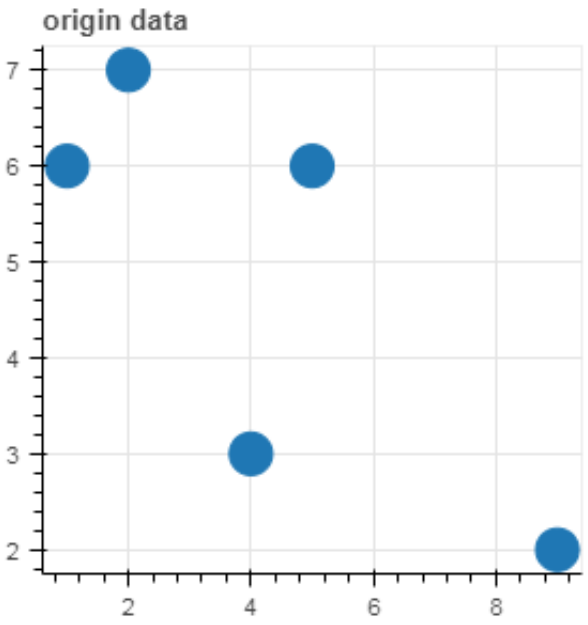


Figure 22.1

```
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:

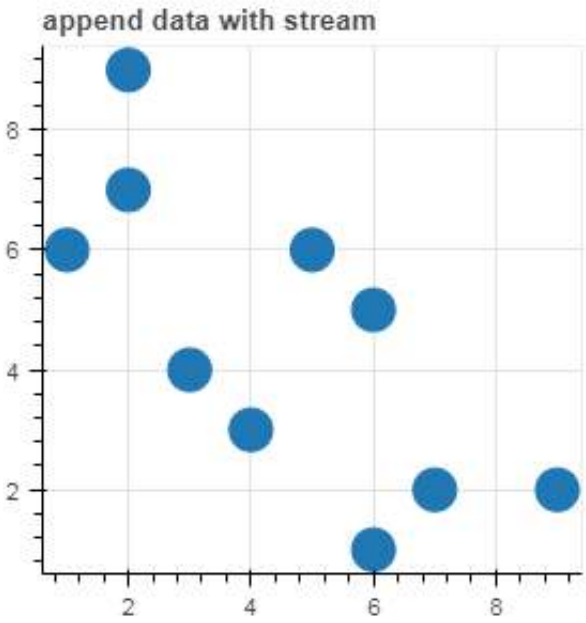


Figure 22.2

22.2 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

Figure 22.3

```
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)

show(p1)
```

图示如下:

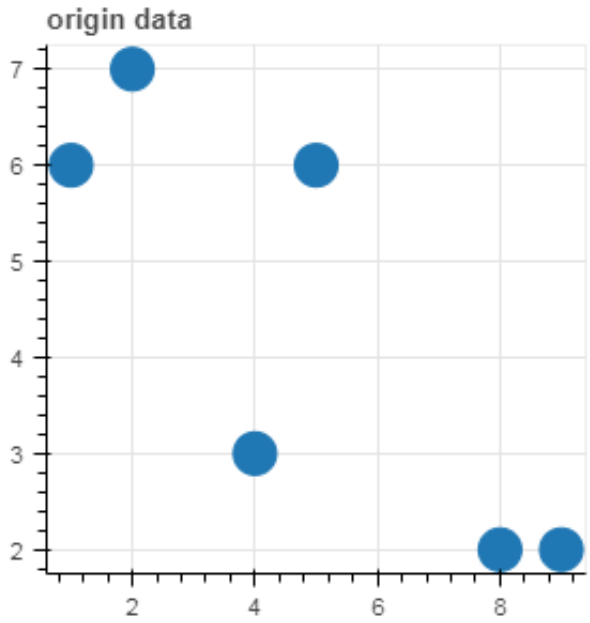


Figure 22.4

22.2.1 更新单个数据

```
# 更新单个数据
# {column:(index, new_value) }
source.patch({'x_column':[(0,15)]})

p2 = figure(plot_width=300, plot_height=300,title= 'revise single value with
    patch')
p2.circle(x='x_column', y='y_column', source=source, size=20)

show(p2)
```

图示如下:

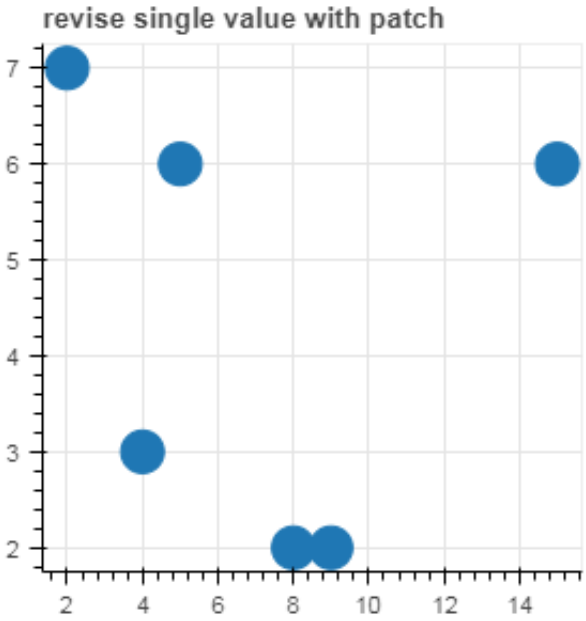


Figure 22.5

22.2.2 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:

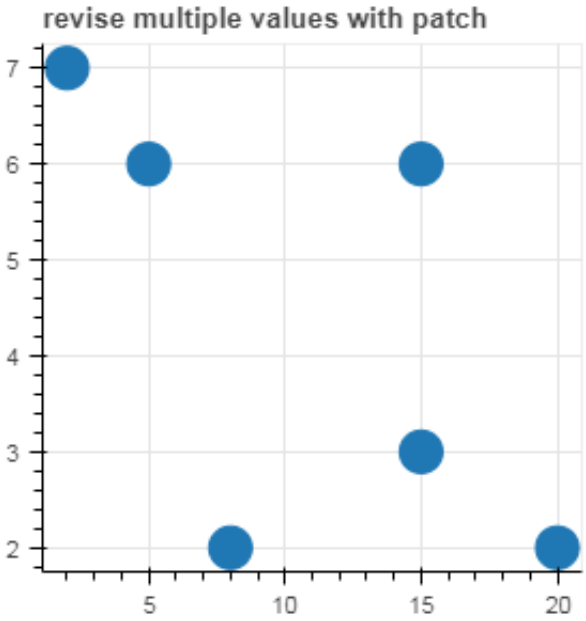


Figure 22.6

22.3 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

22.3.1 Indexfilter

根据数据的索引来筛选数据

```
1 from bokeh.plotting import figure
2 from bokeh.models import ColumnDataSource, CDSView, IndexFilter
3 from bokeh.layouts import gridplot
4
5 data = {'x_column': [1, 2, 9, 4, 5, 8],
6         'y_column': [6, 7, 2, 3, 6, 2]}
7
8 df = pd.DataFrame(data=data)
```

```
source = ColumnDataSource(data=df)
view = CDSView(source=source, filters=[IndexFilter([0,2,4])])

p1 = figure(plot_width=300, plot_height=300, title='origin state')
p1.circle(x='x_column', y='y_column', source=source, size=20)

p2 = figure(plot_width=300, plot_height=300, title='IndexFilter')
p2.circle(x='x_column', y='y_column', source=source, size=20, view=view)

grid=gridplot([p1,p2],ncols=2, plot_width=300,plot_height=300)

show(grid)
```

图示如下:

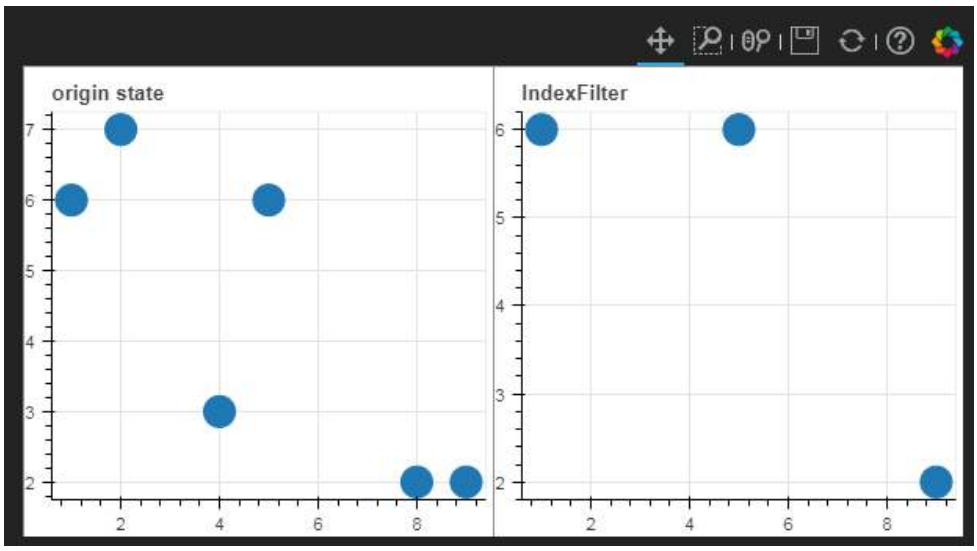


Figure 22.7

22.3.2 BooleanFilter

根据布尔值，True 或 False 来筛选数据

```
from bokeh.models import BooleanFilter
from bokeh.layouts import row

booleans = [True if y_val>4 else False for y_val in source.data['y_column']]
view_booleans = CDSView(source=source, filters=[BooleanFilter(booleans)])

p1 = figure(plot_width=300, plot_height=300,title='origin state')
p1.circle(x='x_column', y='y_column', source=source, size=20)

p2 = figure(plot_width=300, plot_height=300, title='BooleanFilter')
```

```
p2.circle(x='x_column', y='y_column', source=source, size=20, view=
    view_booleans)

grid=gridplot([p1,p2],ncols=2,plot_width=300,plot_height=300)

show(grid)
```

图示如下：

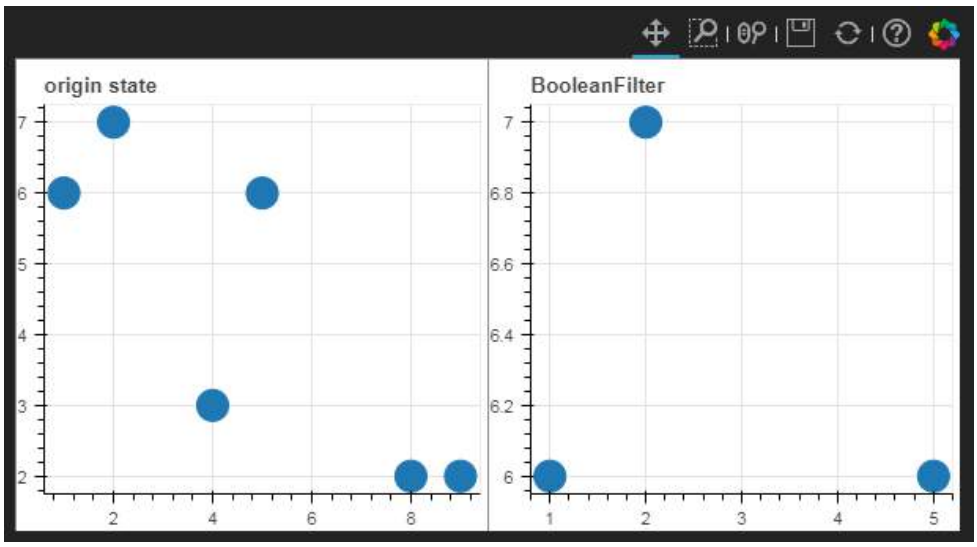


Figure 22.8

22.3.3 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']}
```

```
14
15 p1 = figure(title="Full data set", **plot_size_and_tools)
16 p1.circle(x='petal_length', y='petal_width', source=source, color='black')
17
18 p2 = figure(title="Setosa only", x_range=p1.x_range, y_range=p1.y_range, **
       plot_size_and_tools)
19 p2.circle(x='petal_length', y='petal_width', source=source, view=view1, color='
       red')
20
21 show(gridplot([[p1, p2]]))
```

图示如下：

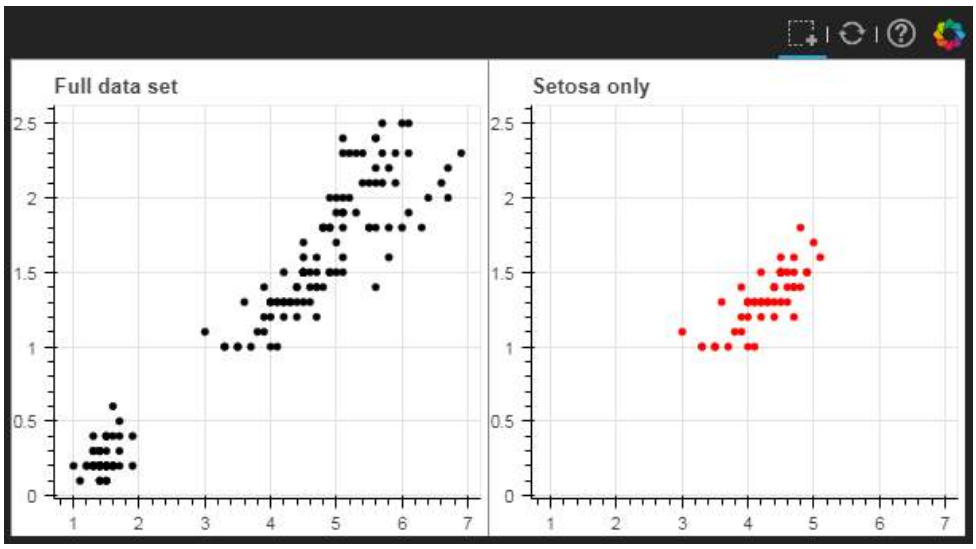


Figure 22.9

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 ~~

23 Bokeh 中图形与组件的布局简介

23.1 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。

23.1.1 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:

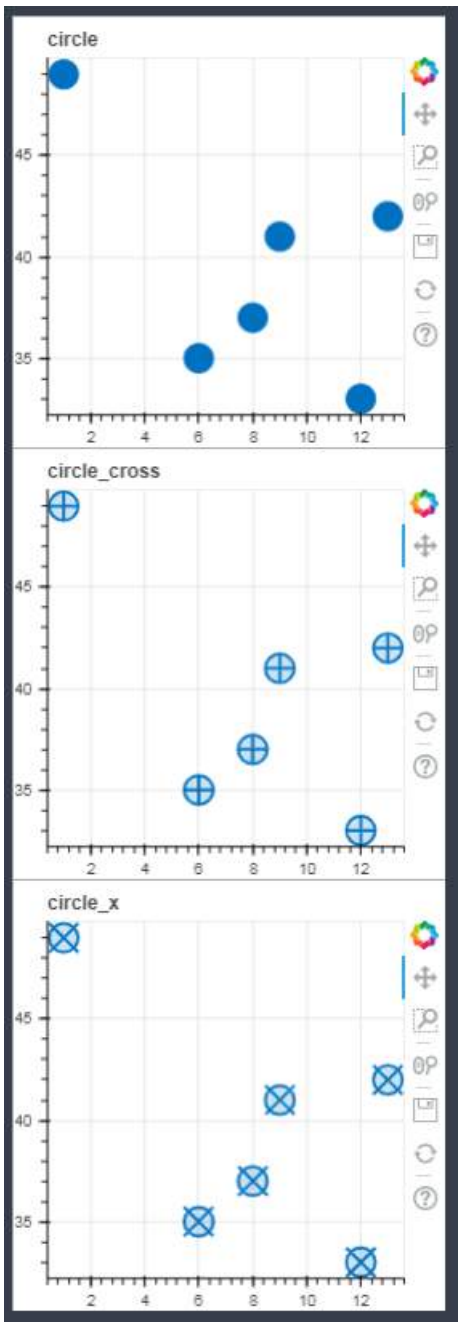


Figure 23.1

23.1.2 row

把所有图形按行分布，其基本用法为 row([plot_1, plot_2, ……, plot_n])

```
row_layout = row(p1,p2,p3)
show(row_layout)
```

如下图所示：

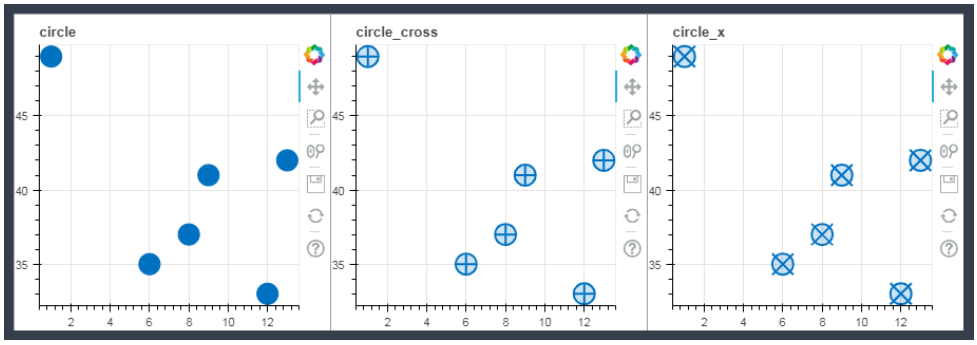


Figure 23.2

23.1.3 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：

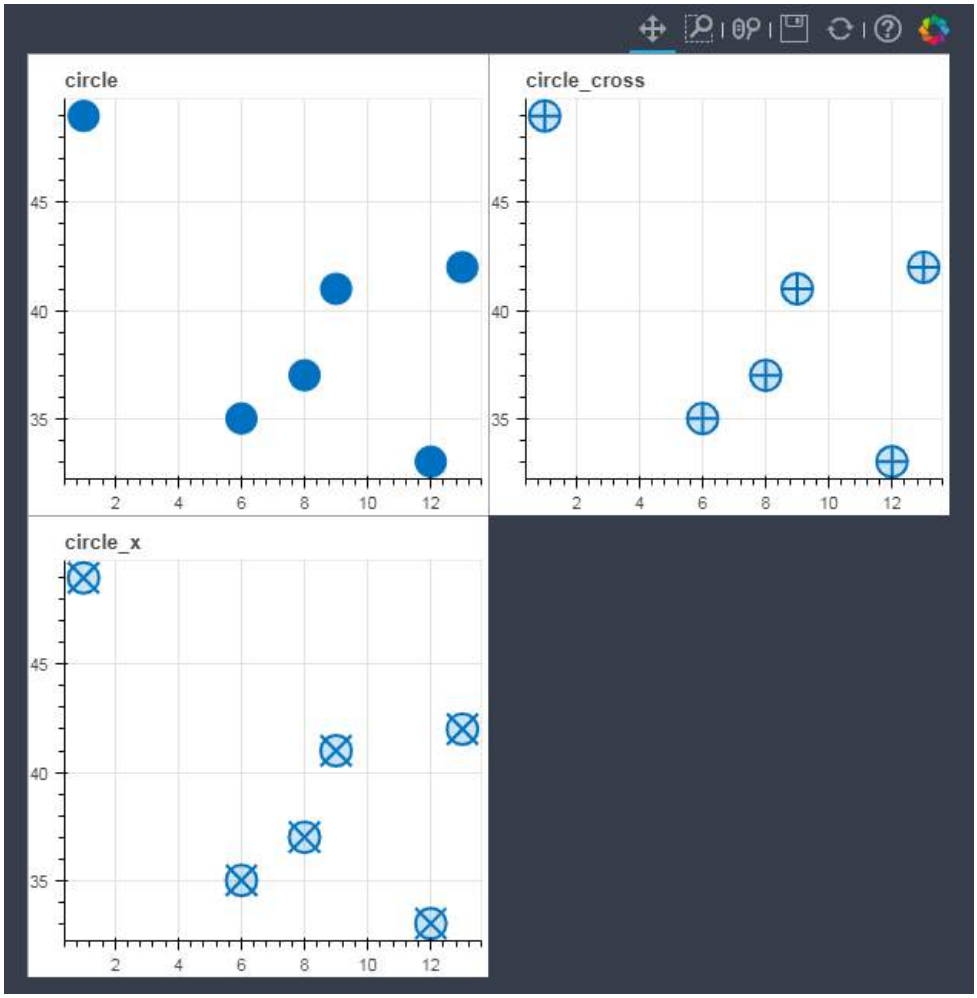


Figure 23.3

```
grid2=gridplot([p1,p2],[None,p3])

show(grid2)
```

如下图所示：

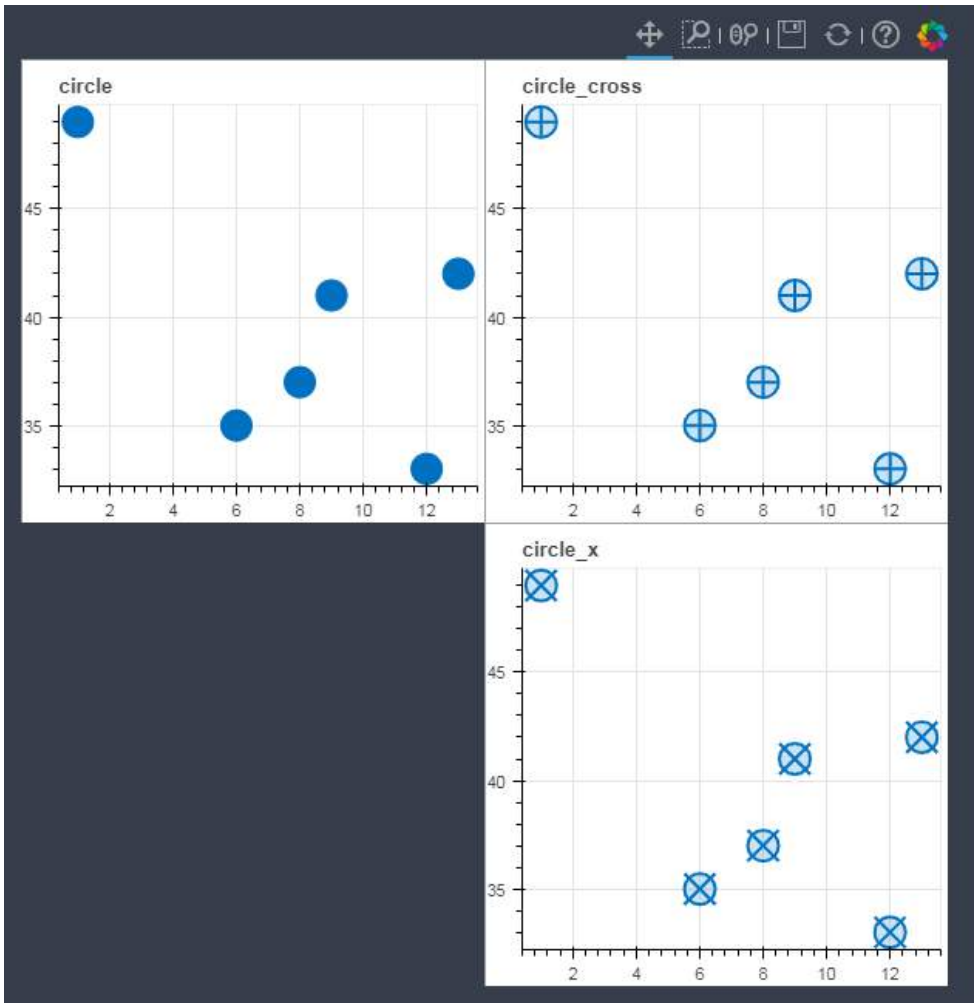


Figure 23.4

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:

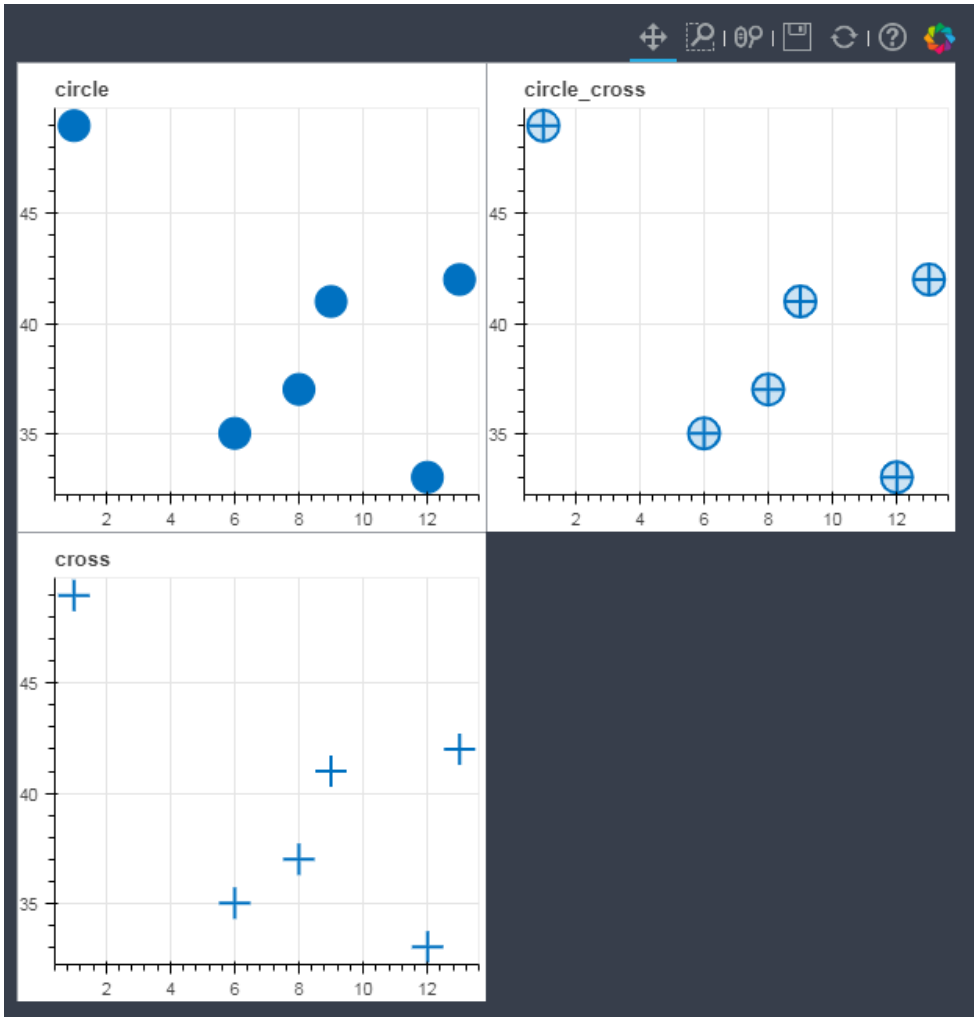


Figure 23.5

```
grid4=gridplot([p1,p2,None,p4],ncols=2, plot_width=300,plot_height=300)

show(grid4)
```

如下图所示:

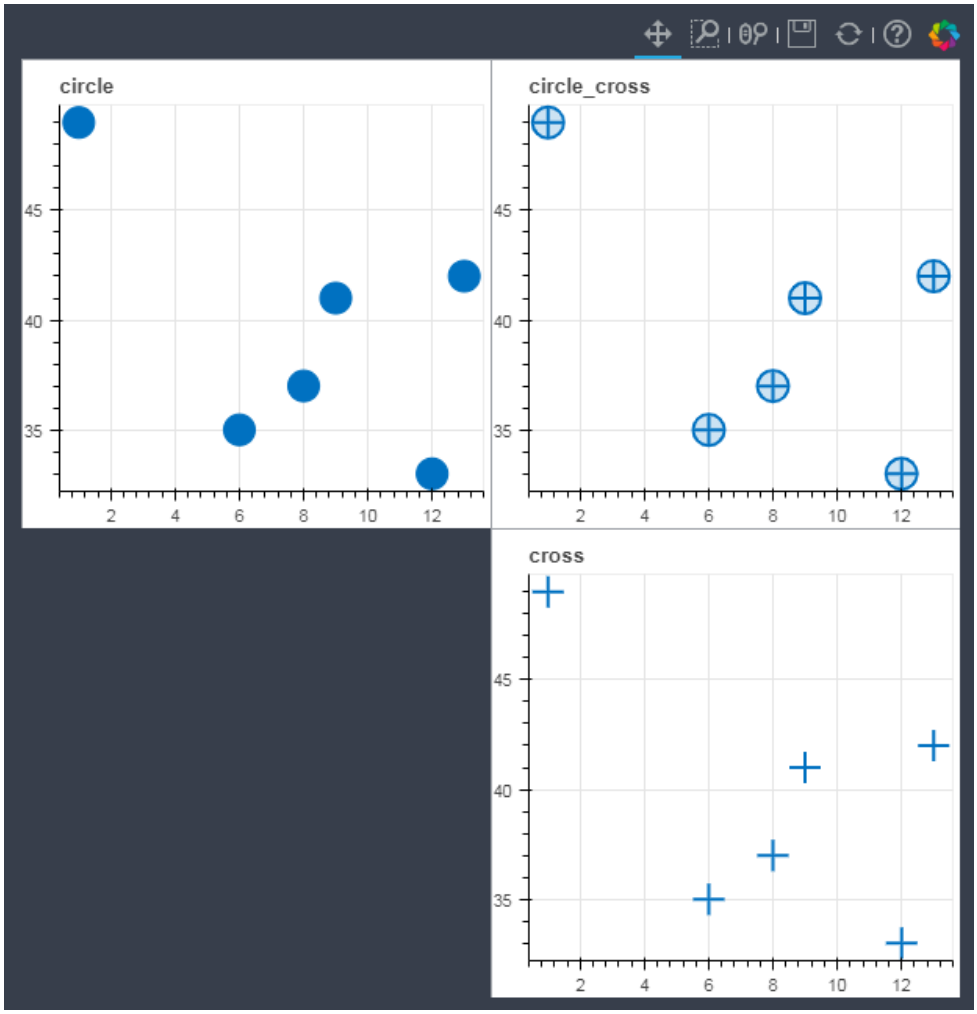


Figure 23.6

23.2 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

```
1 from bokeh.layouts import widgetbox
2 from bokeh.models.widgets import Button, RadioButtonGroup, Select, Slider
3 from bokeh.models.widgets import Dropdown, Toggle
4
5 # 创建一些组件
6 slider = Slider(start=0, end=20, value=1, step=0.5, title="Slider")
7 button_group = RadioButtonGroup(labels=["Option 1", "Option 2", "Option 3"],
      active=0)
8 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：

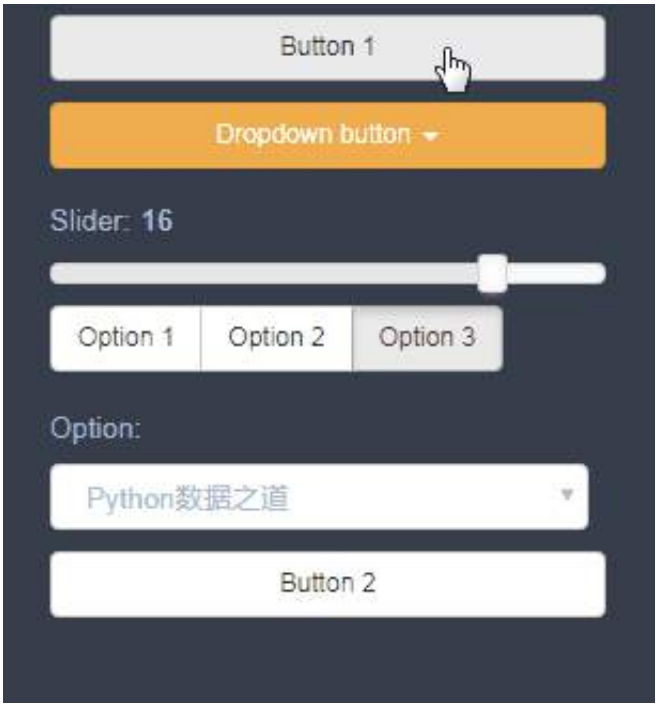


Figure 23.7

> 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。

23.3 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:

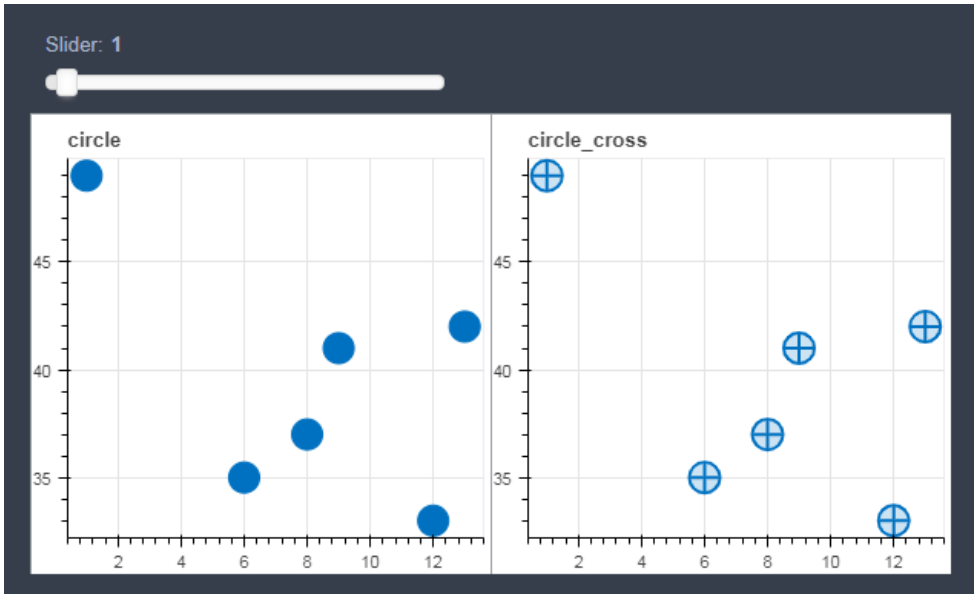


Figure 23.8

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。

24 项目实战：智联求职系列

24.1 第一篇：数据采集并保存到 MongoDB 数据库

24.1.1 前言

本次主题分两篇文章来介绍：

- 一、数据采集
- 二、数据分析

第一篇先来介绍数据采集，即用 python 爬取网站数据。

24.1.2 运行环境和 python 库

先说下运行环境：

- python3.5
- windows 7，64 位系统

本次智联招聘的网站爬取，主要涉及以下一些 python 库：

- requests
- BeautifulSoup
- multiprocessing
- pymongo
- itertools

24.1.3 爬取的主要步骤

- 根据关键字、城市、以及页面编号生成需要爬取的网页链接
- 用 requests 获取相应的网页内容
- 用 BeautifulSoup 解析，获取需要的关键信息
- 将爬取的信息存入 MongoDB 数据库中，插入新记录或更新已有记录
- 用 multiprocessing 启动多进程进行爬取，提高运行效率

24.1.4 文件组成

- 信息配置文件“zhilian_kw_config.py”

- 爬虫主运行文件“zhilian_kw_spider.py”

在配置文件中设置需要爬取的信息，然后运行主程序进行内容抓取。

配置文件“zhilian_kw_config.py”的内容如下：

```
# Code based on Python 3.x
# _*_ coding: utf-8 _*_
# __Author: "LEMON"

TOTAL_PAGE_NUMBER = 90  # PAGE_NUMBER: total number of pages, 可进行修改

KEYWORDS = ['大数据', 'python', '投资经理'] # 需爬取的关键字可以自己添加或修改

# 爬取主要城市的记录
ADDRESS = ['全国', '北京', '上海', '广州', '深圳',
           '天津', '武汉', '西安', '成都', '大连',
           '长春', '沈阳', '南京', '济南', '青岛',
           '杭州', '苏州', '无锡', '宁波', '重庆',
           '郑州', '长沙', '福州', '厦门', '哈尔滨',
           '石家庄', '合肥', '惠州', '太原', '昆明',
           '烟台', '佛山', '南昌', '贵阳', '南宁']

MONGO_URI = 'localhost'
MONGO_DB = 'zhilian'
```

爬虫主运行文件“zhilian_kw_spider.py”的内容如下：

```
# Code based on Python 3.x
# _*_ coding: utf-8 _*_
# __Author: "LEMON"

from datetime import datetime
from urllib.parse import urlencode
from multiprocessing import Pool
import requests
from bs4 import BeautifulSoup
import pymongo
from zhilian.zhilian_kw_config import *
import time
from itertools import product

client = pymongo.MongoClient(MONGO_URI)
db = client[MONGO_DB]


def download(url):
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0)
        Gecko/20100101 Firefox/51.0'}
    response = requests.get(url, headers=headers)
    return response.text

```

```
24
25  def get_content(html):
26      # 记录保存日期
27      date = datetime.now().date()
28      date = datetime.strftime(date, '%Y-%m-%d')  # 转变成str
29
30      soup = BeautifulSoup(html, 'lxml')
31      body = soup.body
32      data_main = body.find('div', {'class': 'newlist_list_content'})
33
34      if data_main:
35          tables = data_main.find_all('table')
36
37          for i, table_info in enumerate(tables):
38              if i == 0:
39                  continue
40              tds = table_info.find('tr').find_all('td')
41              zwmc = tds[0].find('a').get_text()  # 职位名称
42              zw_link = tds[0].find('a').get('href')  # 职位链接
43              fkl = tds[1].find('span').get_text()  # 反馈率
44              gsmc = tds[2].find('a').get_text()  # 公司名称
45              zwyx = tds[3].get_text()  # 职位月薪
46              gzdd = tds[4].get_text()  # 工作地点
47              gbsj = tds[5].find('span').get_text()  # 发布日期
48
49              tr_brief = table_info.find('tr', {'class': 'newlist_tr_detail'})
50              # 招聘简介
51              brief = tr_brief.find('li', {'class': 'newlist_deatil_last'}).
                    get_text()
52
53              # 用生成器获取信息
54              yield {'zwmc': zwmc,  # 职位名称
55                     'fkl': fkl,  # 反馈率
56                     'gsmc': gsmc,  # 公司名称
57                     'zwyx': zwyx,  # 职位月薪
58                     'gzdd': gzdd,  # 工作地点
59                     'gbsj': gbsj,  # 公布时间
60                     'brief': brief,  # 招聘简介
61                     'zw_link': zw_link,  # 网页链接
62                     'save_date': date  # 记录信息保存的日期
63                     }
64
65
66  def main(args):
67      basic_url = 'http://sou.zhaopin.com/jobs/searchresult.ashx?'
68
69      for keyword in KEYWORDS:
70          mongo_table = db[keyword]
71          paras = {'jl': args[0],
72                   'kw': keyword,
73                   'p': args[1]  # 第X页
```

```
                }
        url = basic_url + urlencode(paras)
        # print(url)
        html = download(url)
        # print(html)
        if html:
            data = get_content(html)
            for item in data:
                if mongo_table.update({'zw_link': item['zw_link']}, {'$set':
                    item}, True):
                    print('已保存记录：', item)

if __name__ == '__main__':
    start = time.time()
    number_list = list(range(TOTAL_PAGE_NUMBER))
    args = product(ADDRESS, number_list)
    pool = Pool()
    pool.map(main, args) # 多进程运行
    end = time.time()
    print('Finished, task runs %s seconds.' % (end - start))
```

24.2 第二篇：python 求职 Top10 城市分析

从智联招聘爬取相关信息后，我们关心的是如何对内容进行分析，获取用用的信息。本次以上篇文章中爬取的数据为基础，分析关键词为“python”的爬取数据的情况，获取包括全国 python 招聘数量 Top10 的城市列表以及其他相关信息。

24.2.1 主要分析步骤

- 数据读取
- 数据整理
- 对职位数量在全国主要城市的分布情况进行分析
- 对全国范围内的职位月薪情况进行分析
- 对该职位招聘岗位要求描述进行词云图分析，获取频率最高的关键字
- 选取两个城市，分别分析月薪分布情况以及招聘要求的词云图分析

24.2.2 具体分析过程

```
import pymongo
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
% 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
```

24.2.2.1 读取数据

```
client = pymongo.MongoClient('localhost')
db = client['zhilian']
table = db['python']

columns = ['zwmc',
           'gsmc',
           'zwyx',
           'gbsj',
           'gzdd',
           'fkl',
           'brief',
           'zw_link',
           '_id',
           'save_date']

# url_set =  set([records['zw_link'] for records in table.find()])
# print(url_set)

df = pd.DataFrame([records for records in table.find()], columns=columns)

# columns_update = ['职位名称',
#                   '公司名称',
#                   '职位月薪',
#                   '公布时间',
#                   '工作地点',
#                   '反馈率',
#                   '招聘简介',
#                   '网页链接',
#                   '_id',
#                   '信息保存日期']
# df.columns = columns_update
print('总行数为：{}行'.format(df.shape[0]))
df.head(2)
```

结果如下图所示:

总行数为：23972行

	zwmc	gsmc	zwyx	gbsj	gzdd	fkl	brief	zw_link	_id	save_date
0	PYTHON程序员	义乌市邮朋电子商务商行	6000-10000	05-10	杭州-萧山区	71%	岗位职责 1、负责从前端界面到底层服务器的数据平台的整体设计；2、负责数据中心...	http://jobs.zhaopin.com/428183986250013.htm	59009cdc42949e35dbe920dc	2017-05-20
1	Python开发工程师助理/Python开发学徒/Python程序员实习生可学徒	苏州软世通信息技术有限公司	4001-6000	16天前	苏州-姑苏区		岗位职责：1.负责智能搜索；2.Python相关的数据处理；3、大专及以上学...	http://jobs.zhaopin.com/466382095250279.htm	59009cdc42949e35dbe920de	2017-05-20

Figure 24.1

24.2.2.2 数据整理

将 str 格式的日期变为 datatime

```
df['save_date'] = pd.to_datetime(df['save_date'])
print(df['save_date'].dtype)
# df['save_date']
```

```
datetime64[ns]
```

筛选月薪格式为“XXXX-XXXX”的信息

```
df_clean = df[['zwmc',
               'gsmc',
               'zwyx',
               'gbsj',
               'gzdd',
               'fkl',
               'brief',
               'zw_link',
               'save_date']]

# 对月薪的数据进行筛选，选取格式为“XXXX-XXXX”的信息，方面后续分析
df_clean = df_clean[df_clean['zwyx'].str.contains('\d+-\d+', regex=True)]
print('总行数为：{}行'.format(df_clean.shape[0]))
# df_clean.head()
```

```
总行数为：22605行
```

分割月薪字段，分别获取月薪的下限值和上限值

```
# http://stackoverflow.com/questions/14745022/pandas-dataframe-how-do-i-split-a-column-into-two

# http://stackoverflow.com/questions/20602947/append-column-to-pandas-dataframe

# df_temp.loc[: ,'zwyx_min'],df_temp.loc[: , 'zwyx_max'] = df_temp.loc[: , 'zwyx'].str.split('-',1).str #会有警告
s_min, s_max = df_clean.loc[: , 'zwyx'].str.split('-',1).str
df_min = pd.DataFrame(s_min)
df_min.columns = ['zwyx_min']
df_max = pd.DataFrame(s_max)
df_max.columns = ['zwyx_max']

df_clean_concat = pd.concat([df_clean, df_min, df_max], axis=1)
# df_clean['zwyx_min'].astype(int)
df_clean_concat['zwyx_min'] = pd.to_numeric(df_clean_concat['zwyx_min'])
df_clean_concat['zwyx_max'] = pd.to_numeric(df_clean_concat['zwyx_max'])
# print(df_clean['zwyx_min'].dtype)
print(df_clean_concat.dtypes)
df_clean_concat.head(2)
```

运行结果如下图所示：

```
zwmc                object
gsmc                object
zwyx                object
gbsj                object
gzdd                object
fkl                 object
brief               object
zw_link             object
save_date    datetime64[ns]
zwyx_min             int64
zwyx_max             int64
dtype: object
```

	zwmc	gsmc	zwyx	gbsj	gzdd	fkl	brief	zw_link	save_date	zwyx_min	zwyx_max
0	PYTHON程序员	义乌市都朋电子商务有行	6000-10000	05-10	杭州-萧山区	71%	岗位职责 1.负责从前端界面到底层服务器的数据平台的整体设计；2.负责数据中心...	http://jobs.zhaopin.com/428183986250013.htm	2017-05-20	6000	10000
1	Python开发工程师助理/Python开发学徒/Python程序员实习生可学徒	苏州软世通信息技术有限公司	4001-6000	15天前	苏州-姑苏区		岗位职责：1.负责智能搜索；2.Python相关的数据处理；3、大专及以上学...	http://jobs.zhaopin.com/466382035250279.htm	2017-05-20	4001	6000

Figure 24.2

- 将数据信息按职位月薪进行排序

```
df_clean_concat.sort_values('zwyx_min',inplace=True)
# df_clean_concat.tail()
```

- 判断爬取的数据是否有重复值

```
# 判断爬取的数据是否有重复值
print(df_clean_concat[df_clean_concat.duplicated('zw_link')==True])
```

```
Empty DataFrame
Columns: [zwmc, gsmc, zwyx, gbsj, gzdd, fkl, brief, zw_link, save_date,
    zwyx_min, zwyx_max]
Index: []
```

- 从上述结果可看出，数据是没有重复的。

24.2.3 对全国范围内的职位进行分析

24.2.3.1 主要城市的招聘职位数量分布情况

```
# from IPython.core.display import display, HTML
ADDRESS = [ '北京', '上海', '广州', '深圳',
            '天津', '武汉', '西安', '成都', '大连',
            '长春', '沈阳', '南京', '济南', '青岛',
            '杭州', '苏州', '无锡', '宁波', '重庆',
            '郑州', '长沙', '福州', '厦门', '哈尔滨',
            '石家庄', '合肥', '惠州', '太原', '昆明',
            '烟台', '佛山', '南昌', '贵阳', '南宁']
df_city = df_clean_concat.copy()

# 由于工作地点的写上，比如北京，包含许多地址为北京-朝阳区等
# 可以用替换的方式进行整理，这里用pandas的replace()方法
for city in ADDRESS:
    df_city['gzdd'] = df_city['gzdd'].replace([(city+'.*')],[city],regex=True)

# 针对全国主要城市进行分析
df_city_main = df_city[df_city['gzdd'].isin(ADDRESS)]

df_city_main_count = df_city_main.groupby('gzdd')['zwmc','gsmc'].count()
df_city_main_count['gsmc'] = df_city_main_count['gsmc']/(df_city_main_count['
    gsmc'].sum())
df_city_main_count.columns = ['number', 'percentage']

# 按职位数量进行排序
df_city_main_count.sort_values(by='number', ascending=False, inplace=True)

# 添加辅助列，标注城市和百分比，方面在后续绘图时使用
df_city_main_count['label']=df_city_main_count.index+ ' '+  ((
    df_city_main_count['percentage']*100).round()).astype('int').astype('str')+'
    %'
print(type(df_city_main_count))

# 职位数量最多的Top10城市的列表
print(df_city_main_count.head(10))
```

```
<class 'pandas.core.frame.DataFrame'>
      number  percentage   label
gzdd
北京       6936    0.315948  北京 32%
上海       3213    0.146358  上海 15%
深圳       1908    0.086913   深圳 9%
成都       1290    0.058762   成都 6%
杭州       1174    0.053478   杭州 5%
广州       1167    0.053159   广州 5%
南京        826    0.037626   南京 4%
郑州        741    0.033754   郑州 3%
武汉        552    0.025145   武汉 3%
西安        473    0.021546   西安 2%
```

- 对结果进行绘图：

```
from  matplotlib import cm

label = df_city_main_count['label']
sizes = df_city_main_count['number']

# 设置绘图区域大小
fig, axes = plt.subplots(figsize=(10,6),ncols=2)
ax1, ax2 = axes.ravel()

colors = cm.PiYG(np.arange(len(sizes))/len(sizes)) # colormaps: Paired, autumn,
    rainbow, gray,spring,Darks

# 由于城市数量太多，饼图中不显示labels和百分比
patches, texts = ax1.pie(sizes,labels=None, shadow=False, startangle=0, colors=
    colors)

ax1.axis('equal')

ax1.set_title('职位数量分布', loc='center')

# ax2 只显示图例（legend）
ax2.axis('off')
ax2.legend(patches, label, loc='center left', fontsize=9)

plt.savefig('job_distribute.jpg')
plt.show()
```

运行结果如下述饼图所示：

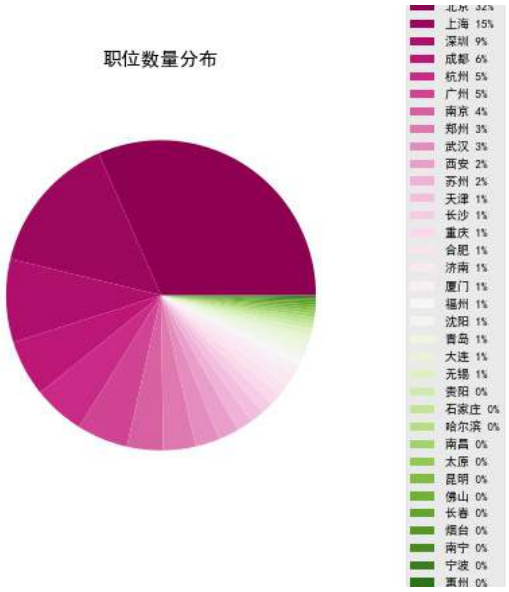


Figure 24.3

24.2.3.2 月薪分布情况（全国）

```
from matplotlib.ticker import FormatStrFormatter

fig, (ax1, ax2) = plt.subplots(figsize=(10,8), nrows=2)

x_pos = list(range(df_clean_concat.shape[0]))
y1 = df_clean_concat['zwyx_min']

ax1.plot(x_pos, y1)
ax1.set_title('Trend of min monthly salary in China', size=14)
ax1.set_xticklabels('')
ax1.set_ylabel('min monthly salary(RMB)')

bins = [3000,6000, 9000, 12000, 15000, 18000, 21000, 24000, 100000]
counts, bins, patches = ax2.hist(y1, bins, normed=1, histtype='bar', facecolor=
    'g', rwidth=0.8)
ax2.set_title('Hist of min monthly salary in China', size=14)
ax2.set_yticklabels('')
# ax2.set_xlabel('min monthly salary(RMB)')

# http://stackoverflow.com/questions/6352740/matplotlib-label-each-bin
ax2.set_xticks(bins) #将bins设置为xticks
ax2.set_xticklabels(bins, rotation=-90) # 设置为xticklabels的方向

# Label the raw counts and the percentages below the x-axis...
bin_centers = 0.5 * np.diff(bins) + bins[:-1]
for count, x in zip(counts, bin_centers):
```

```
#     # Label the raw counts
#     ax2.annotate(str(count), xy=(x, 0), xycoords=('data', 'axes fraction'),
#         xytext=(0, -70), textcoords='offset points', va='top', ha='center',
    rotation=-90)

    # Label the percentages
    percent = '%0.0f%%' % (100 * float(count) / counts.sum())
    ax2.annotate(percent, xy=(x, 0), xycoords=('data', 'axes fraction'),
        xytext=(0, -40), textcoords='offset points', va='top', ha='center',
            rotation=-90, color='b', size=14)

fig.savefig('salary_quanguo_min.jpg')
```

运行结果如下述图所示：

Figure 24.4

不考虑部分极值后，分析月薪分布情况

```
df_zwyx_adjust = df_clean_concat[df_clean_concat['zwyx_min']<=20000]

fig, (ax1, ax2) = plt.subplots(figsize=(10,8), nrows=2)

x_pos = list(range(df_zwyx_adjust.shape[0]))
y1 = df_zwyx_adjust['zwyx_min']
```

```
7
8  ax1.plot(x_pos, y1)
9  ax1.set_title('Trend of min monthly salary in China (adjust)', size=14)
10 ax1.set_xticklabels('')
11 ax1.set_ylabel('min monthly salary(RMB)')
12
13 bins = [3000,6000, 9000, 12000, 15000, 18000, 21000]
14 counts, bins, patches = ax2.hist(y1, bins, normed=1, histtype='bar', facecolor=
       'g', rwidth=0.8)
15 ax2.set_title('Hist of min monthly salary in China (adjust)', size=14)
16 ax2.set_yticklabels('')
17 # ax2.set_xlabel('min monthly salary(RMB)')
18
19 # http://stackoverflow.com/questions/6352740/matplotlib-label-each-bin
20 ax2.set_xticks(bins) #将bins设置为xticks
21 ax2.set_xticklabels(bins, rotation=-90) # 设置为xticklabels的方向
22
23 # Label the raw counts and the percentages below the x-axis...
24 bin_centers = 0.5 * np.diff(bins) + bins[:-1]
25 for count, x in zip(counts, bin_centers):
26 #     # Label the raw counts
27 #     ax2.annotate(str(count), xy=(x, 0), xycoords=('data', 'axes fraction'),
28 #         xytext=(0, -70), textcoords='offset points', va='top', ha='center',
       rotation=-90)
29
30     # Label the percentages
31     percent = '%0.0f%%' % (100 * float(count) / counts.sum())
32     ax2.annotate(percent, xy=(x, 0), xycoords=('data', 'axes fraction'),
33         xytext=(0, -40), textcoords='offset points', va='top', ha='center',
              rotation=-90, color='b', size=14)
34
35 fig.savefig('salary_quanguo_min_adjust.jpg')
```

运行结果如下述图所示：

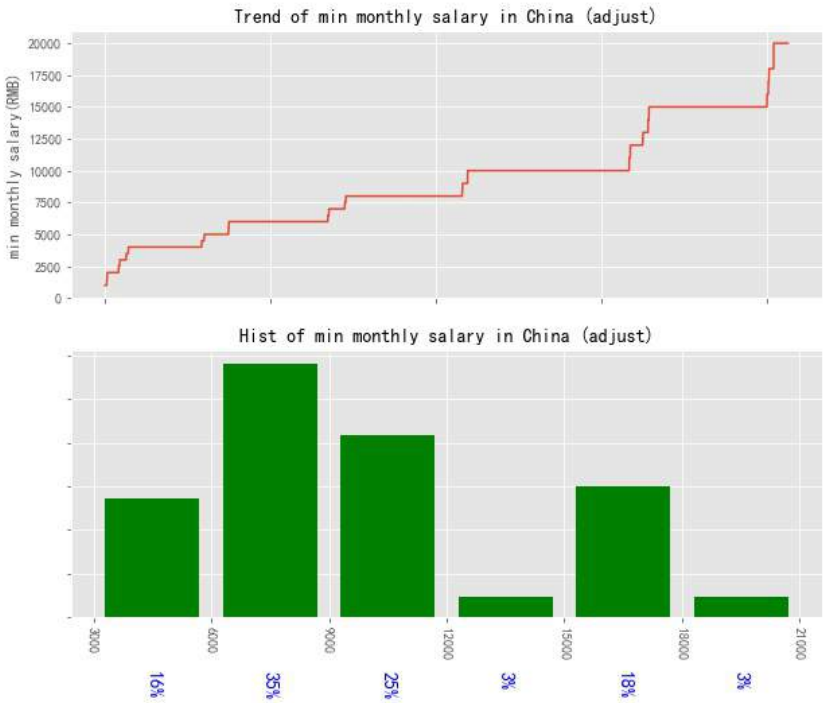


Figure 24.5

24.2.3.3 相关技能要求

```
brief_list = list(df_clean_concat['brief'])
brief_str = ''.join(brief_list)
print(type(brief_str))
# print(brief_str)
# with open('brief_quanguo.txt', 'w', encoding='utf-8') as f:
#     f.write(brief_str)
```

```
<class 'str'>
```

对获取到的职位招聘要求进行词云图分析，代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 17 2017

@author: lemon
"""

import jieba
from wordcloud import WordCloud, ImageColorGenerator
```

```
import matplotlib.pyplot as plt
import os
import PIL.Image as Image
import numpy as np


with open('brief_quanguo.txt', 'rb') as f: # 读取文件内容
    text = f.read()
    f.close()


# 首先使用 jieba 中文分词工具进行分词
wordlist = jieba.cut(text, cut_all=False)
# cut_all, True为全模式, False为精确模式

wordlist_space_split = ' '.join(wordlist)

d = os.path.dirname(__file__)
alice_coloring = np.array(Image.open(os.path.join(d,'colors.png')))
my_wordcloud = WordCloud(background_color='#F0F8FF', max_words=100, mask=
    alice_coloring,
                         max_font_size=300, random_state=42).generate(
                             wordlist_space_split)

image_colors = ImageColorGenerator(alice_coloring)

plt.show(my_wordcloud.recolor(color_func=image_colors))
plt.imshow(my_wordcloud)            # 以图片的形式显示词云
plt.axis('off')                     # 关闭坐标轴
plt.show()

my_wordcloud.to_file(os.path.join(d, 'brief_quanguo_colors_cloud.png'))
```

得到结果如下:

Figure 24.6

24.2.4 北京

24.2.4.1 月薪分布情况

```
df_beijing = df_clean_concat[df_clean_concat['gzdd'].str.contains('北京.*',
    regex=True)]
df_beijing.to_excel('zhilian_kw_python_bj.xlsx')
print('总行数为：{}行'.format(df_beijing.shape[0]))
# df_beijing.head()
```

```
总行数为：6936行
```

参考全国分析时的代码，月薪分布情况图如下：

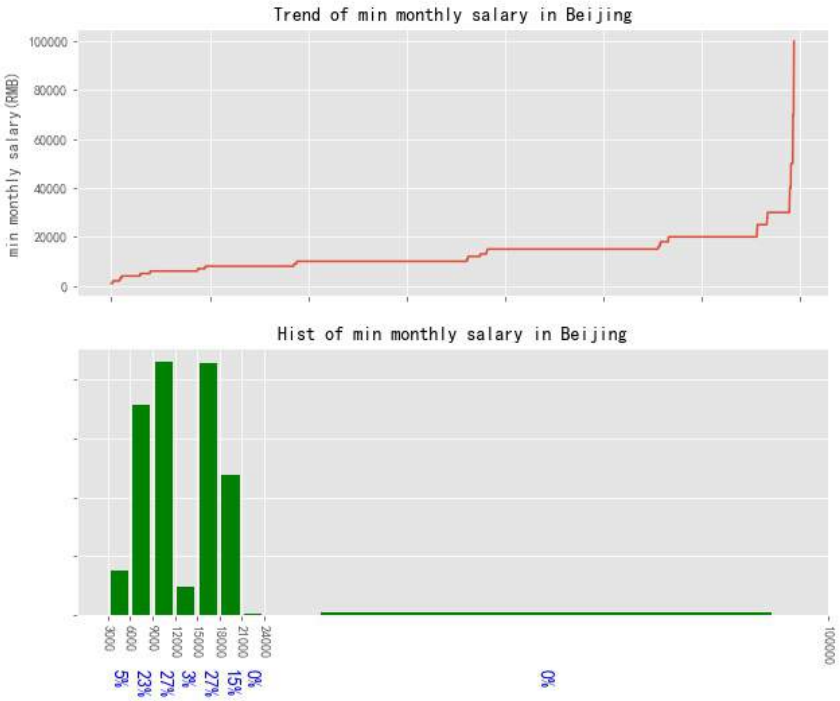


Figure 24.7

24.2.4.2 相关技能要求

```
brief_list_bj = list(df_beijing['brief'])
brief_str_bj = ''.join(brief_list_bj)
print(type(brief_str_bj))
# print(brief_str_bj)
# with open('brief_beijing.txt', 'w', encoding='utf-8') as f:
#     f.write(brief_str_bj)
```

```
<class 'str'>
```

词云图如下：

Figure 24.8

24.2.5 长沙

24.2.5.1 月薪分布情况

```
df_changsha = df_clean_concat[df_clean_concat['gzdd'].str.contains('长沙.*',
    regex=True)]
# df_changsha = pd.DataFrame(df_changsha, ignore_index=True)
df_changsha.to_excel('zhilian_kw_python_cs.xlsx')
print('总行数为：{}行'.format(df_changsha.shape[0]))
# df_changsha.tail()
```

```
总行数为：280行
```

参考全国分析时的代码，月薪分布情况图如下：

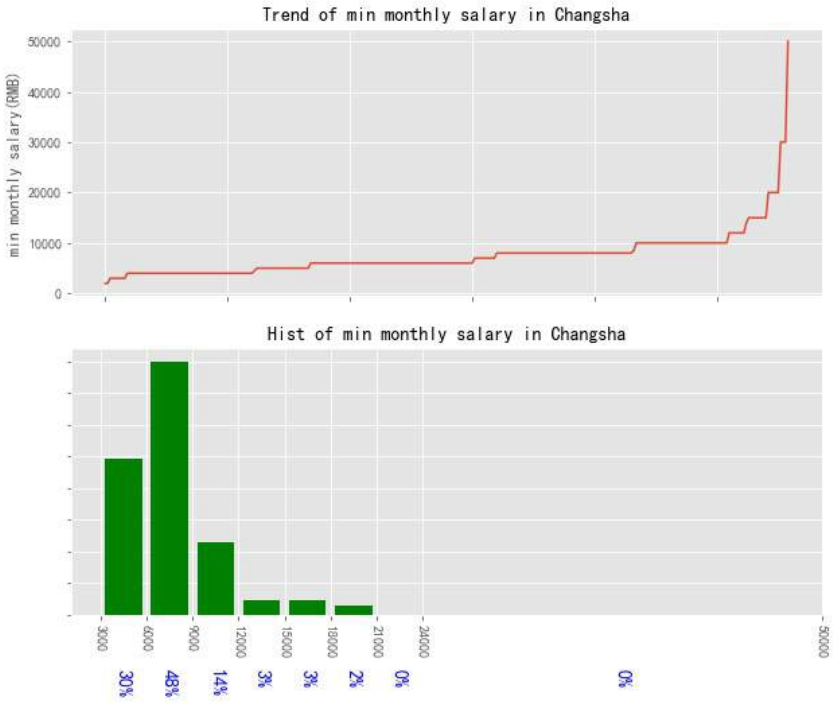


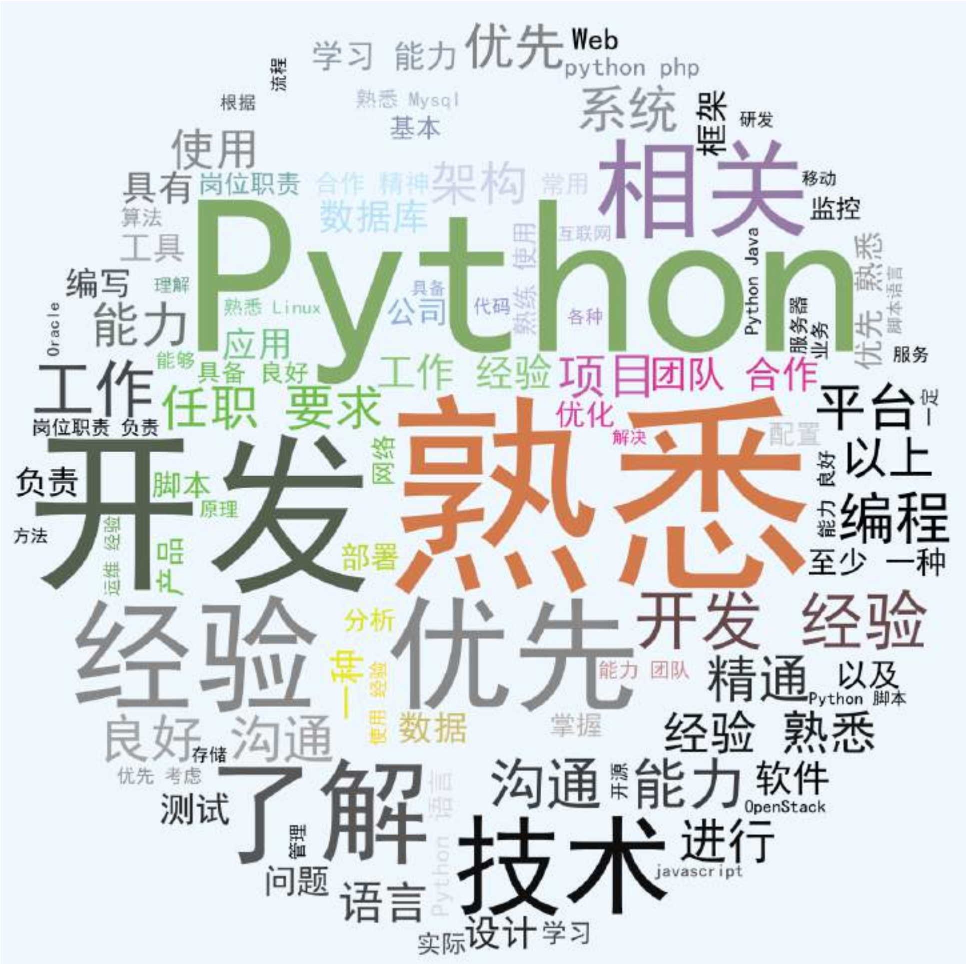
Figure 24.9

24.2.5.2 相关技能要求

```
brief_list_cs = list(df_changsha['brief'])
brief_str_cs = ''.join(brief_list_cs)
print(type(brief_str_cs))
# print(brief_str_cs)
# with open('brief_changsha.txt', 'w', encoding='utf-8') as f:
#     f.write(brief_str_cs)
```

```
<class 'str'>
```

词云图如下：



学习 能力 优先 Web
python php
流程
根据
熟悉 Mysql
系统
基本
框架
研发
使用
相关
具有 岗位职责
合作 精神 架构 常用
移动
监控
算法
数据库
Python
工具
互联网
熟悉
编写
理解
具备
代码
Python Java
熟悉 Linux
公司
各种
脚本语言
能力
Oracle
服务器
业务
优先
应用
能够
具备 良好
服务
工作
工作 经验
项目团队
合作
平台
一定
任职
要求
优化
岗位职责 负责
配置
解决
开发
熟悉
以上
负责
网络
脚本
良好
原理
编程
方法
能力
运维 经验
产品
部署
至少 一种
分析
经验
优先
开发
经验
一种
能力 团队
精通
以及
经验
Python 脚本
良好
沟通
数据
掌握
经验 熟悉
存储
使用
优先 考虑
语言
开源
能力
软件
了解
沟通
测试
OpenStack
管理
Python
技术
进行
问题
javascript
语言
实际
设计
学习

Figure 24.10

25 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')
```

25.1 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
```

25.2 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：

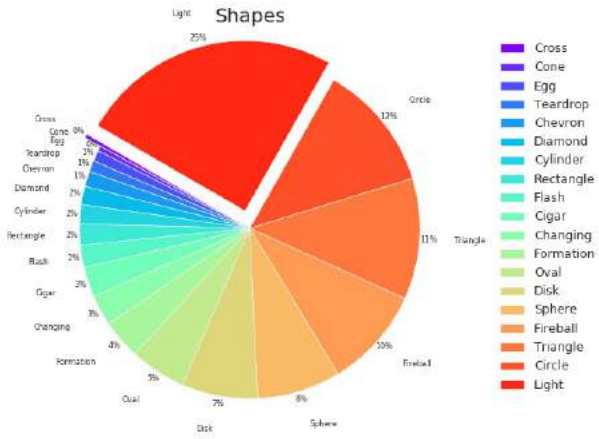


Figure 25.1

25.3 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')

fig.savefig('ufo_state.jpg')
plt.show()
```

运行结果如下：

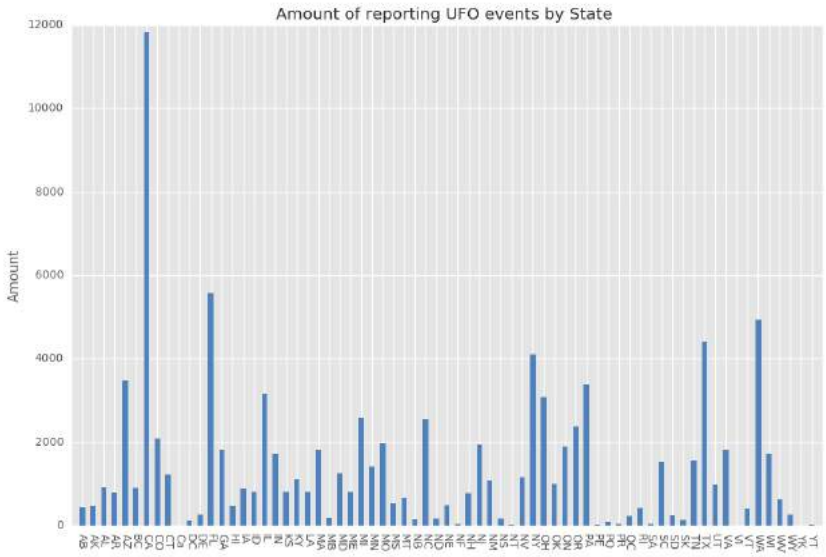


Figure 25.2

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？

25.4 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

```
# df_clean['Year'].astype(int)
s_year = df_clean.groupby(df_clean['Year'].astype(int))['Event_Date'].count()
print(type(s_year))
s_year.head()
```

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

Year
1905    1
1910    2
1920    1
```

```
1925    1
1929    1
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：

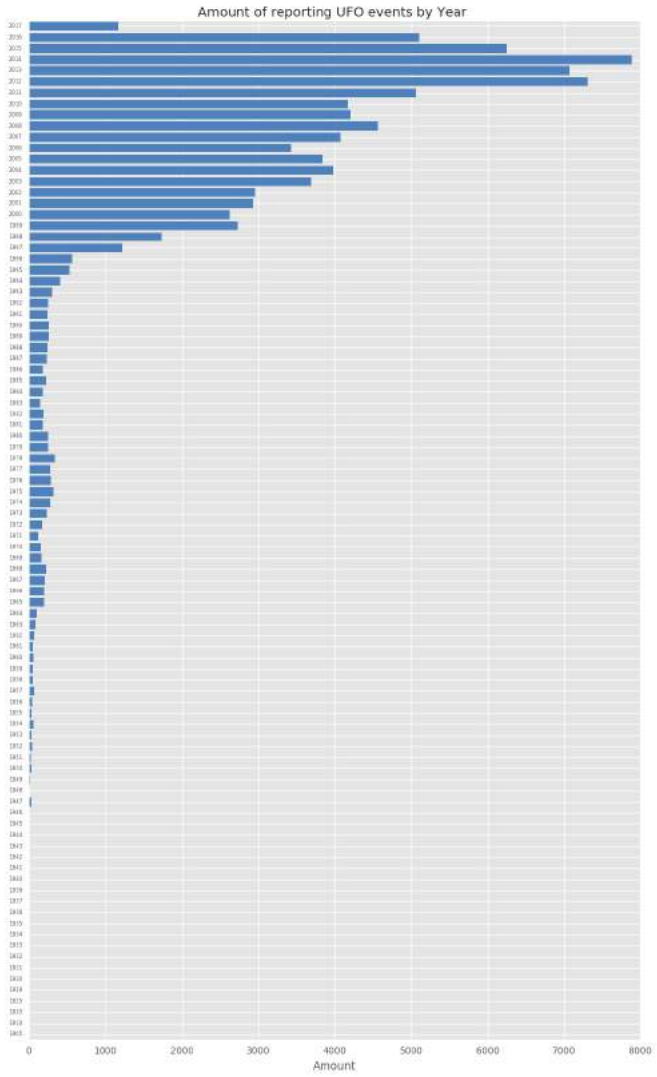


Figure 25.3

从上图可看出，近年来 UFO 出现的报告次数最多

25.5 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

```
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')
```

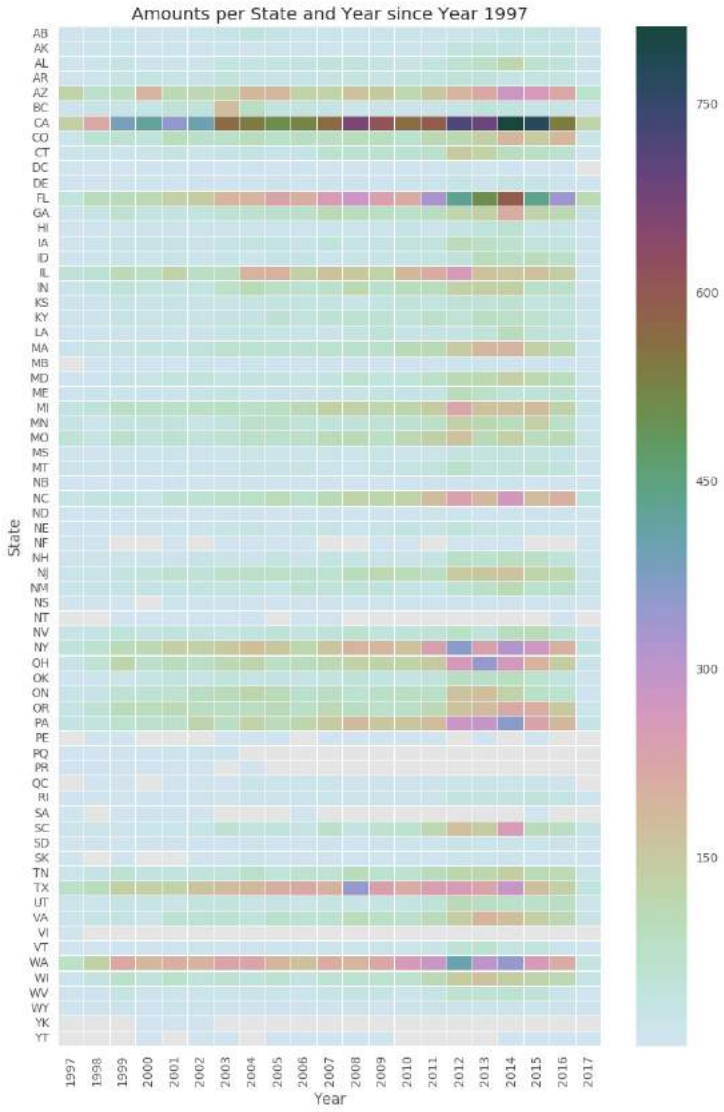


Figure 25.4

- 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。

26 项目实战：世界杯系列

26.1 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

Figure 26.1

26.1.1 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

Figure 26.2

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

Figure 26.3

26.1.2 获取世界杯所有比赛的前 20 强数据情况

26.1.2.1 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'
    )
```

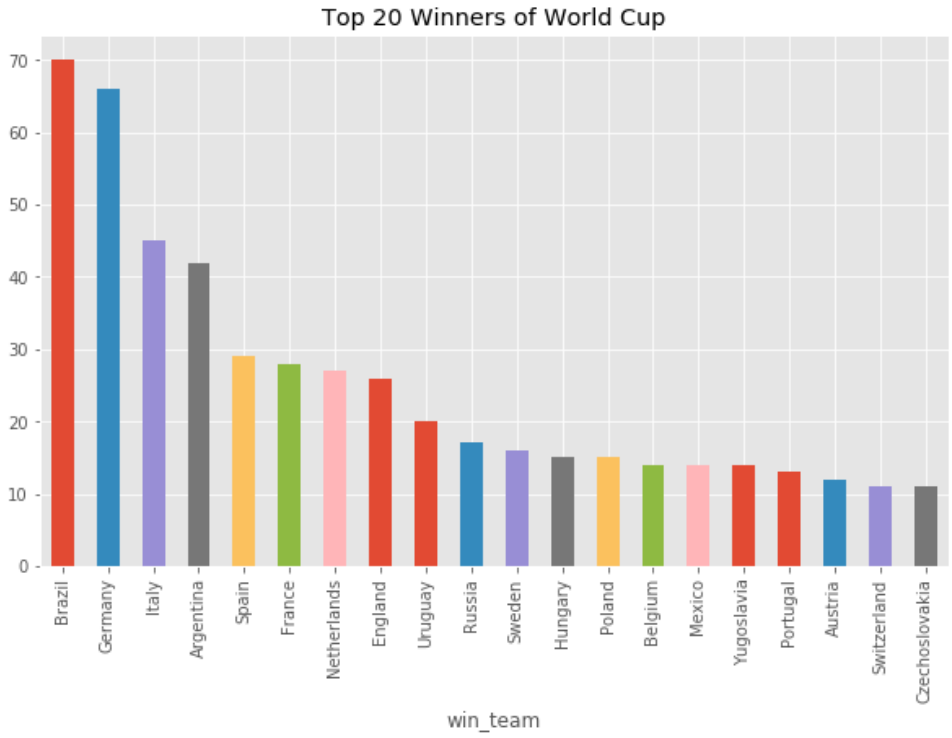


Figure 26.4

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
    ')
```

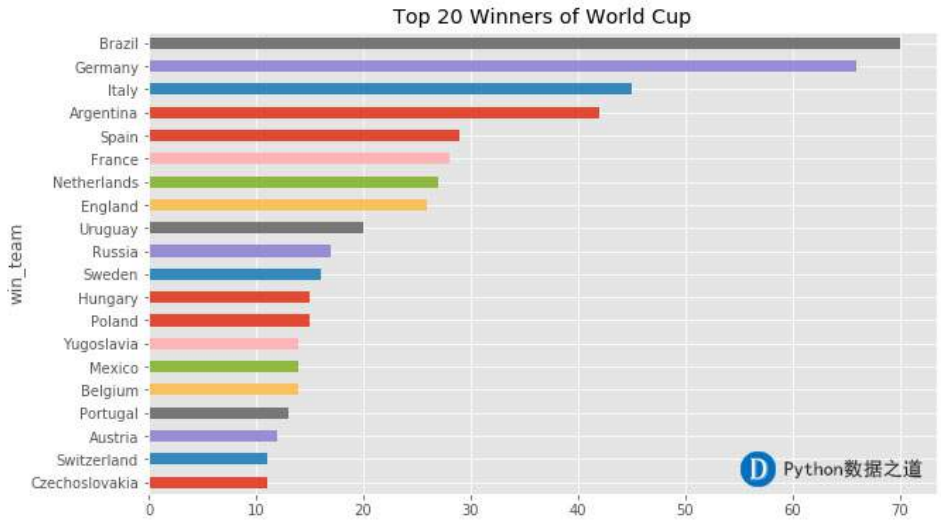


Figure 26.5

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='')
```

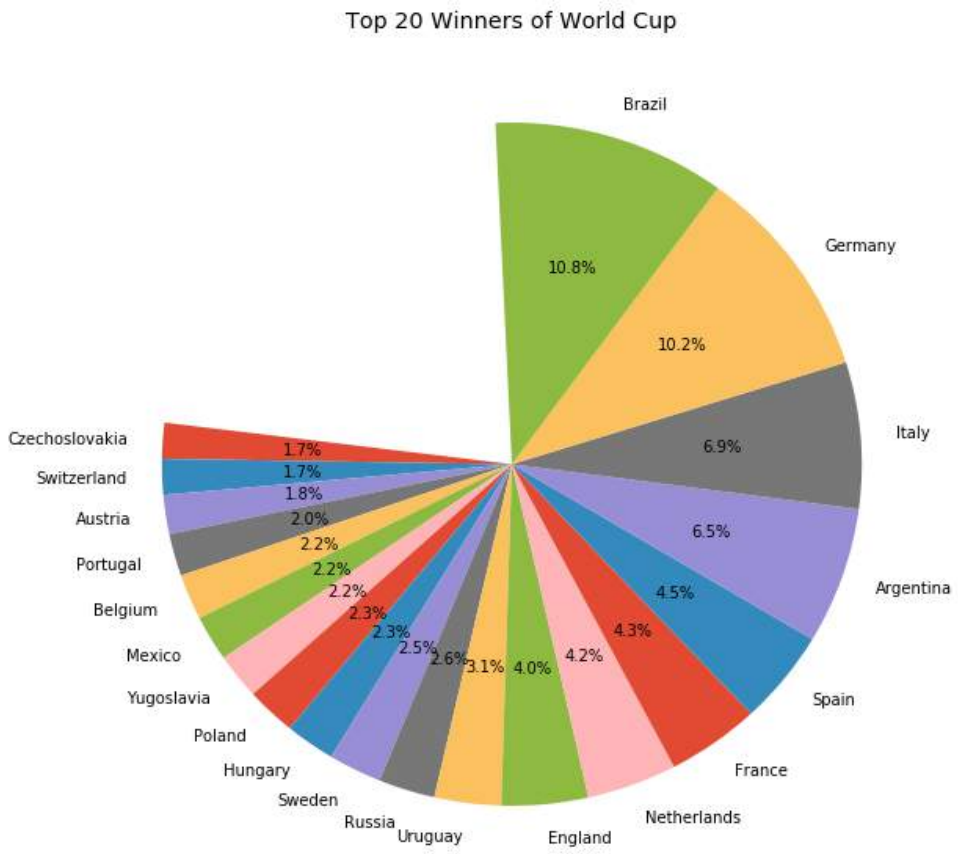


Figure 26.6

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。

26.1.2.2 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')
```

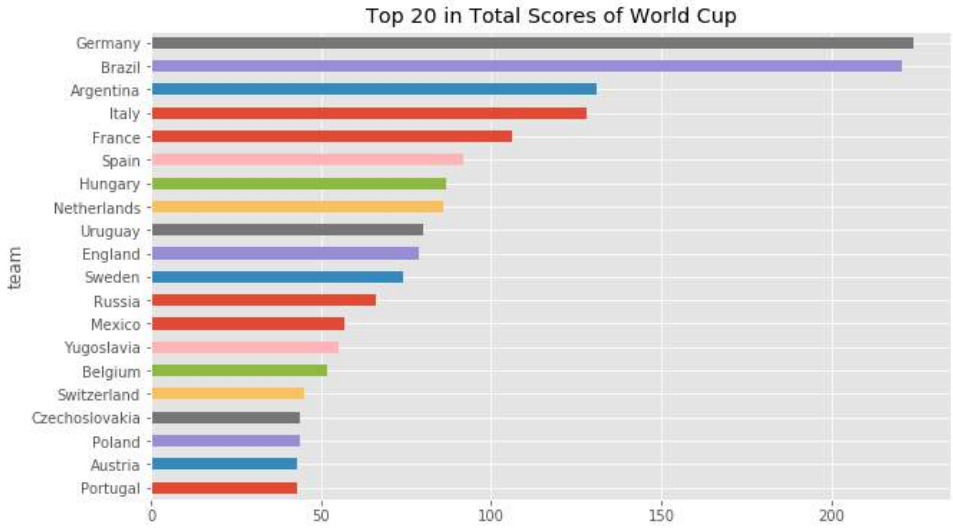


Figure 26.7

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。

26.1.3 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))]
```

26.1.3.1 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')
```

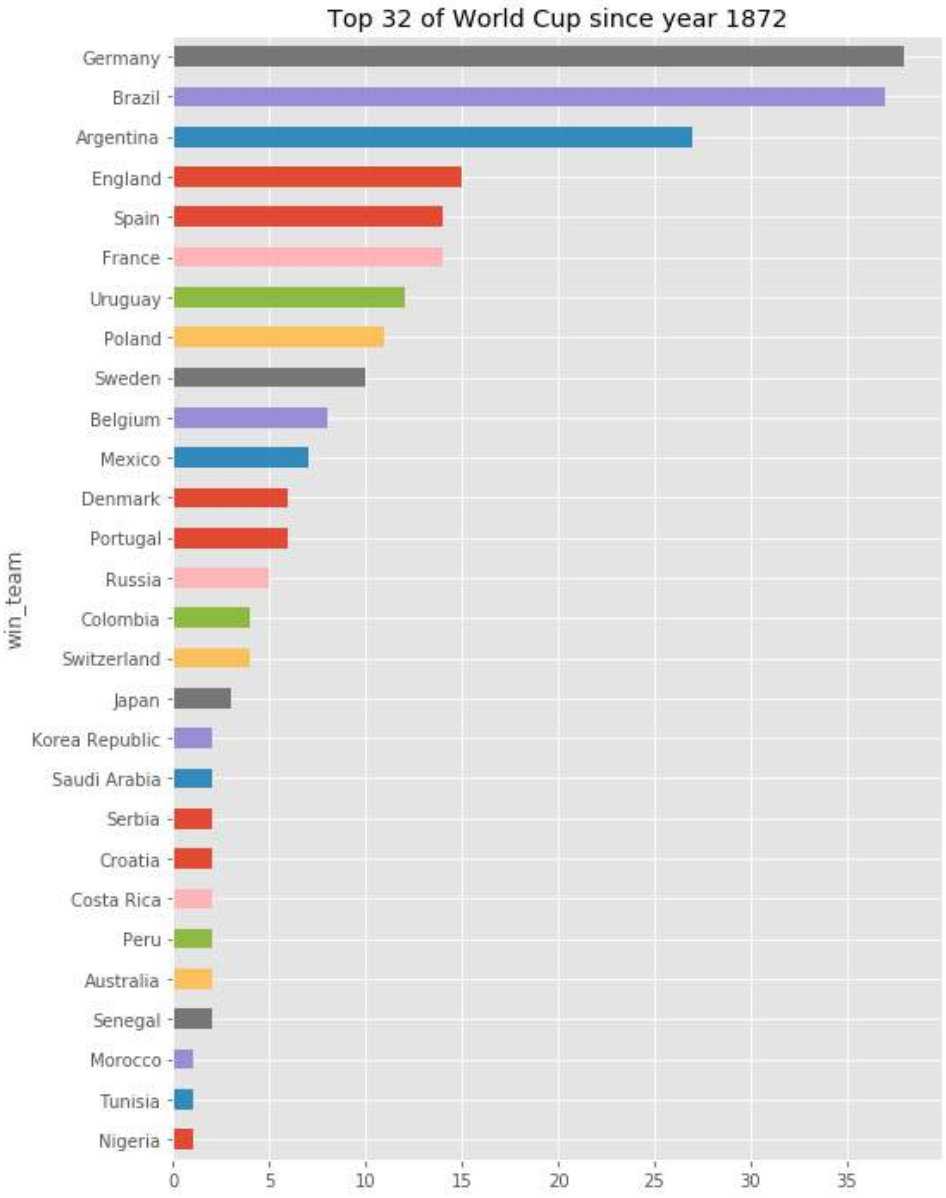


Figure 26.8

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)
```

```
9  s_score_32.sort_values(ascending=True, inplace=True)
10 s_score_32.plot(kind='barh', figsize=(8,12), title='Top 32 in Total Scores of
       World Cup since year 1872')
```

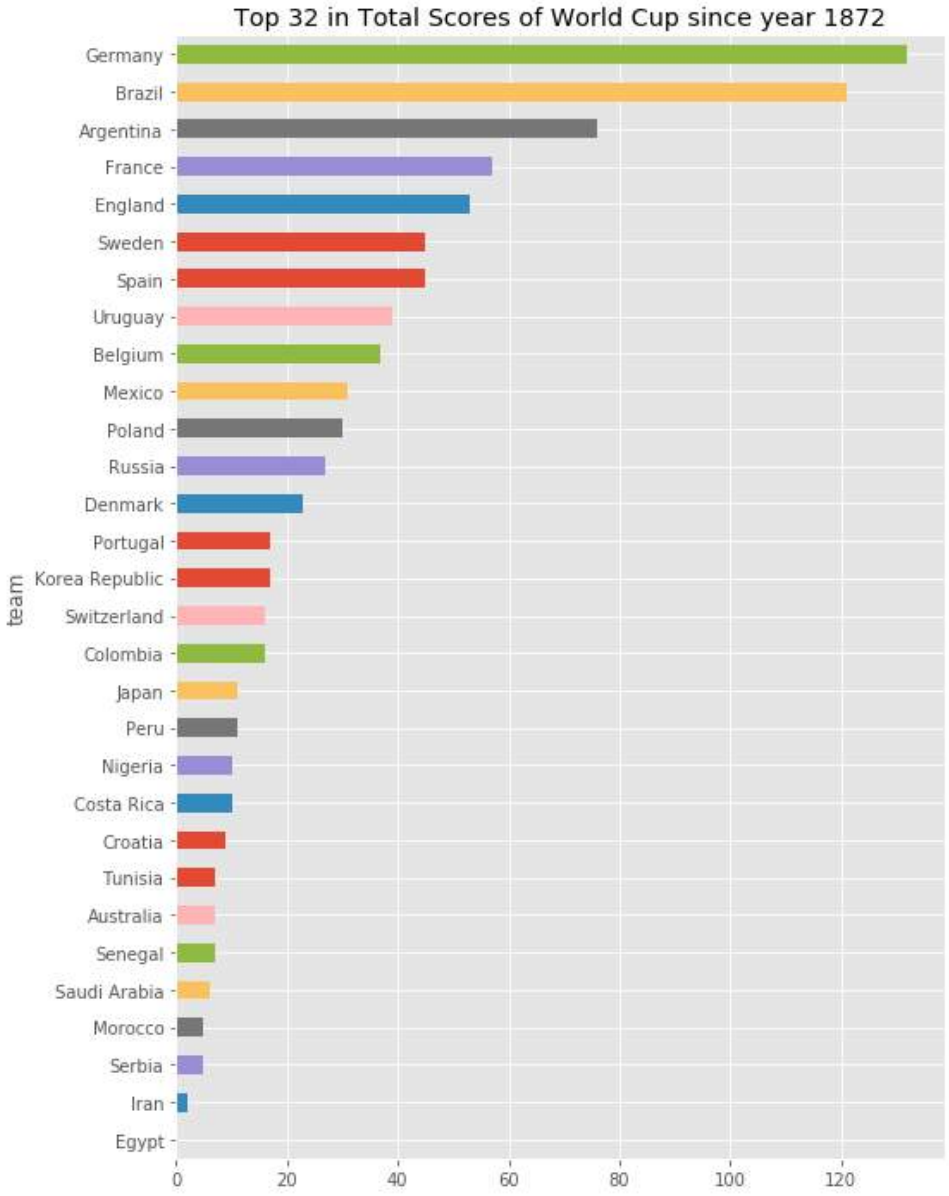


Figure 26.9

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。

26.1.3.2 自 1978 年以来，32 强数据情况

赢球场数情况

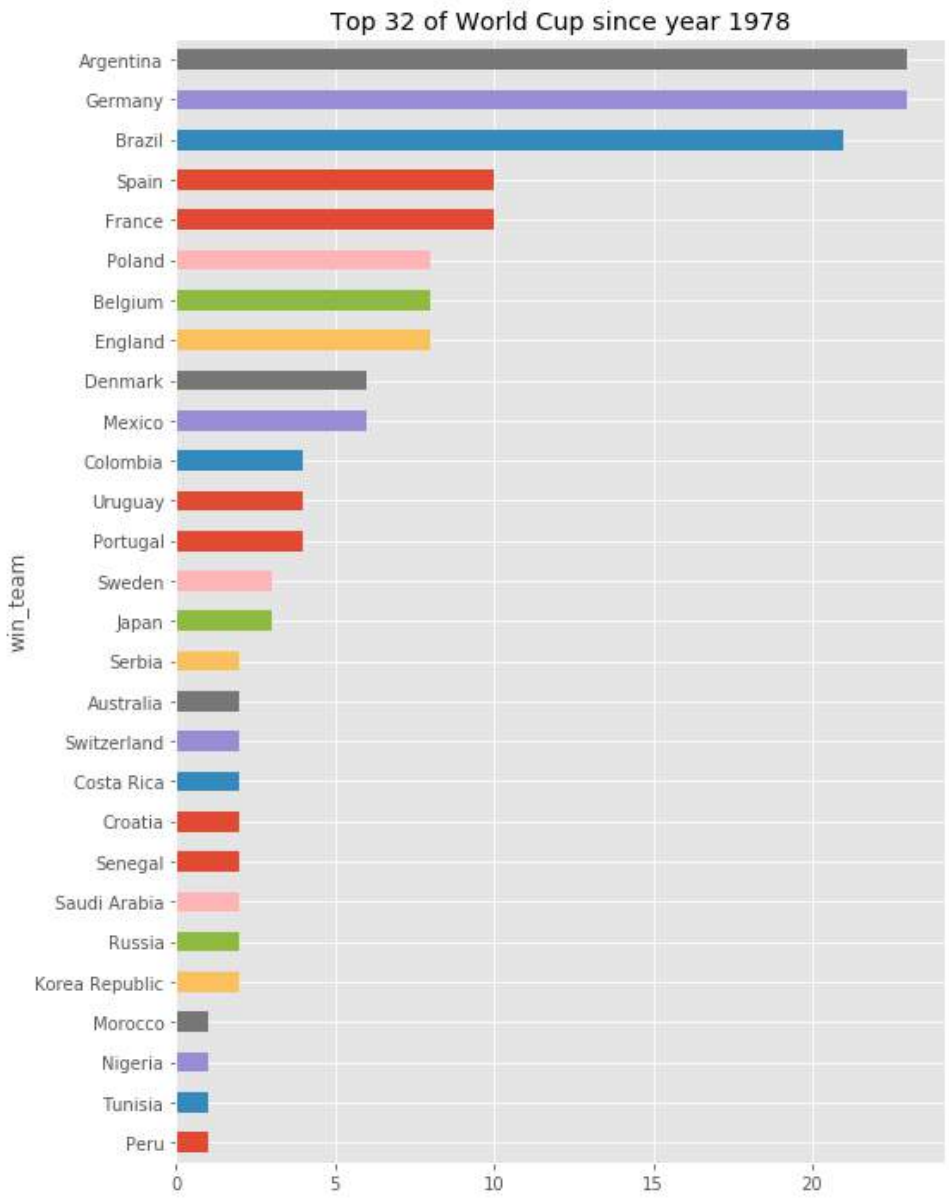


Figure 26.10

进球数据情况

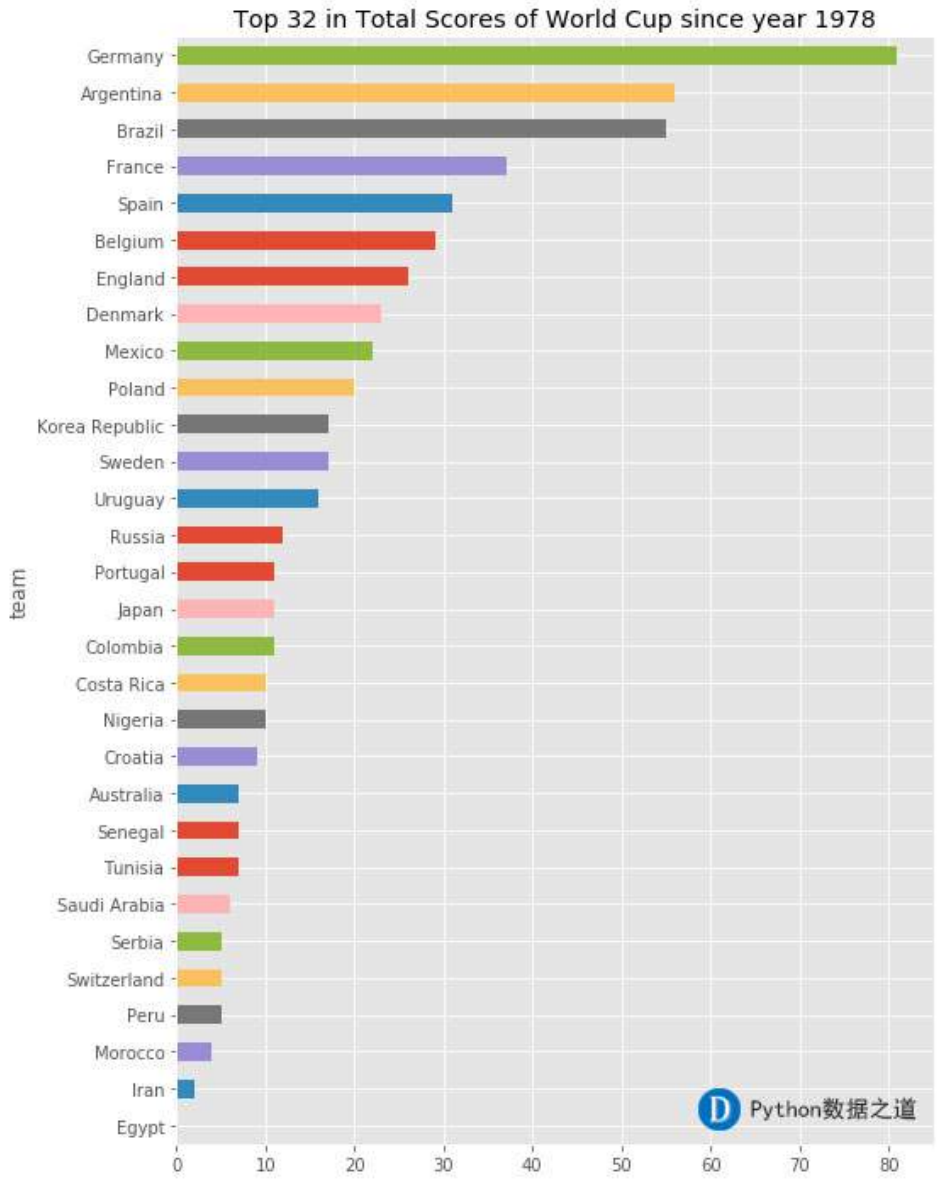


Figure 26.11

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。

26.1.3.3 自 2002 年以来，32 强数据情况

赢球场数情况

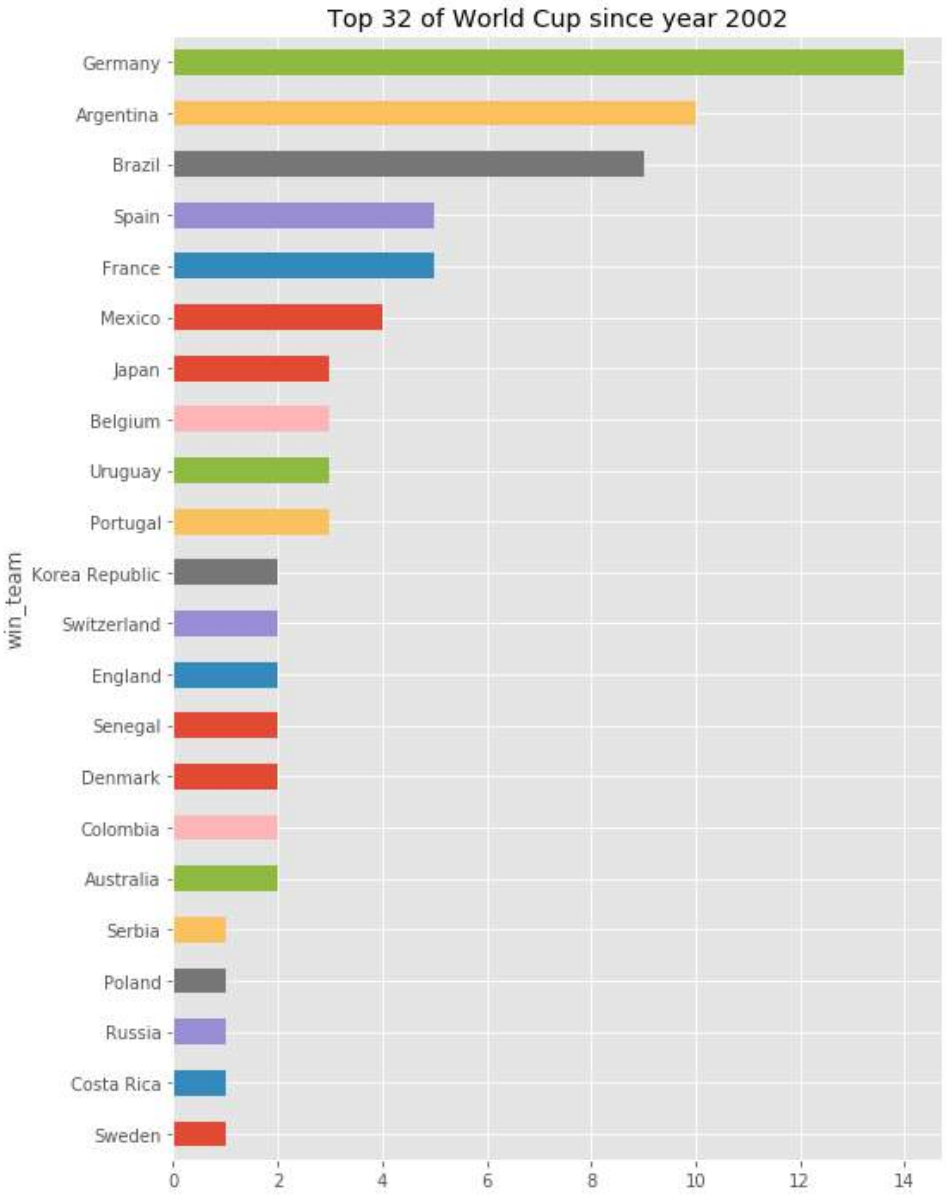


Figure 26.12

进球数据情况

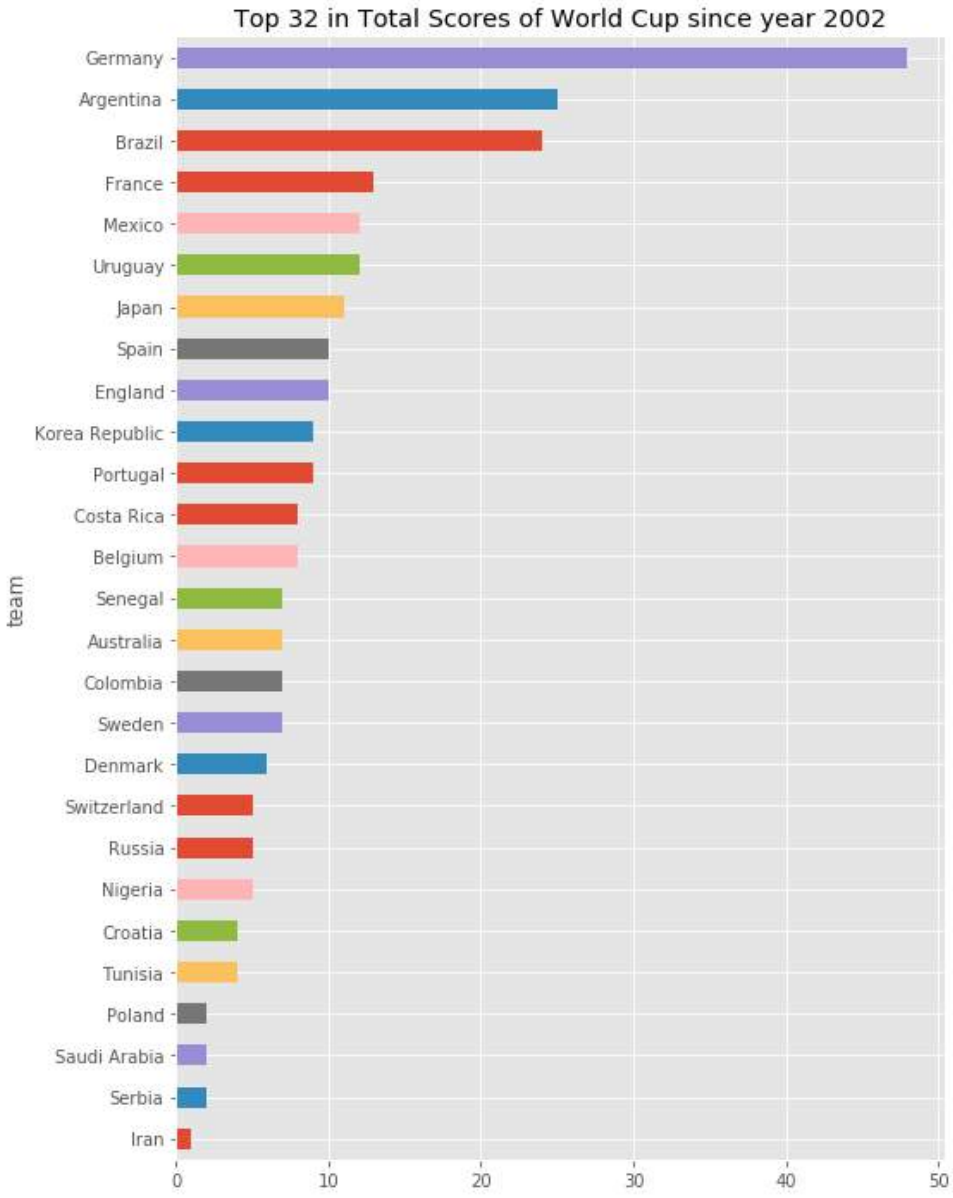


Figure 26.13

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。

26.1.4 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”

26.2 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

Figure 26.14

26.2.1 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

Figure 26.15

26.2.2 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：

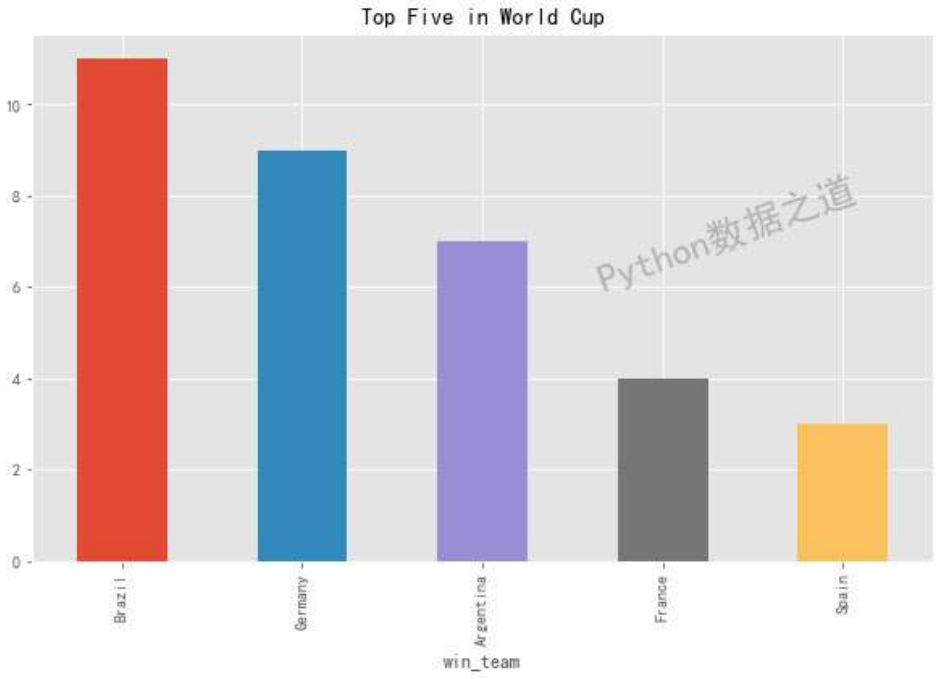


Figure 26.16

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：

26.2.2.1 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：

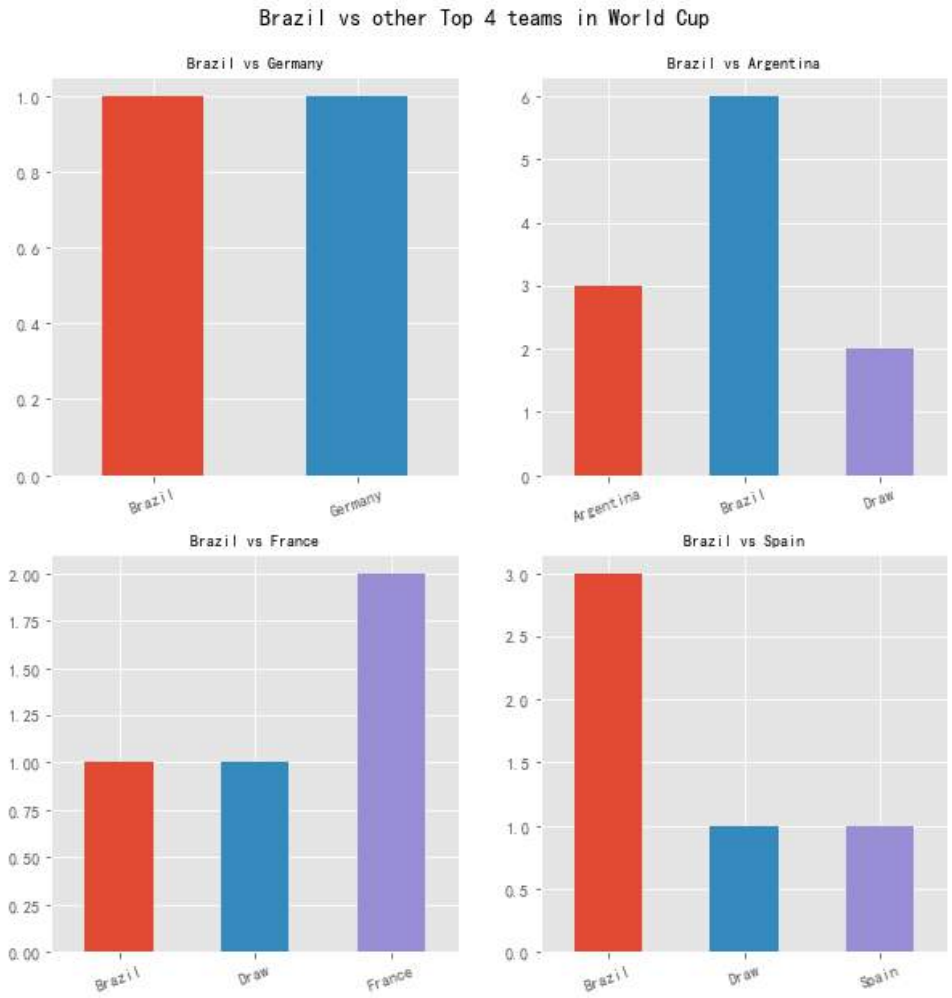


Figure 26.17

> **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……

26.2.2.2 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：

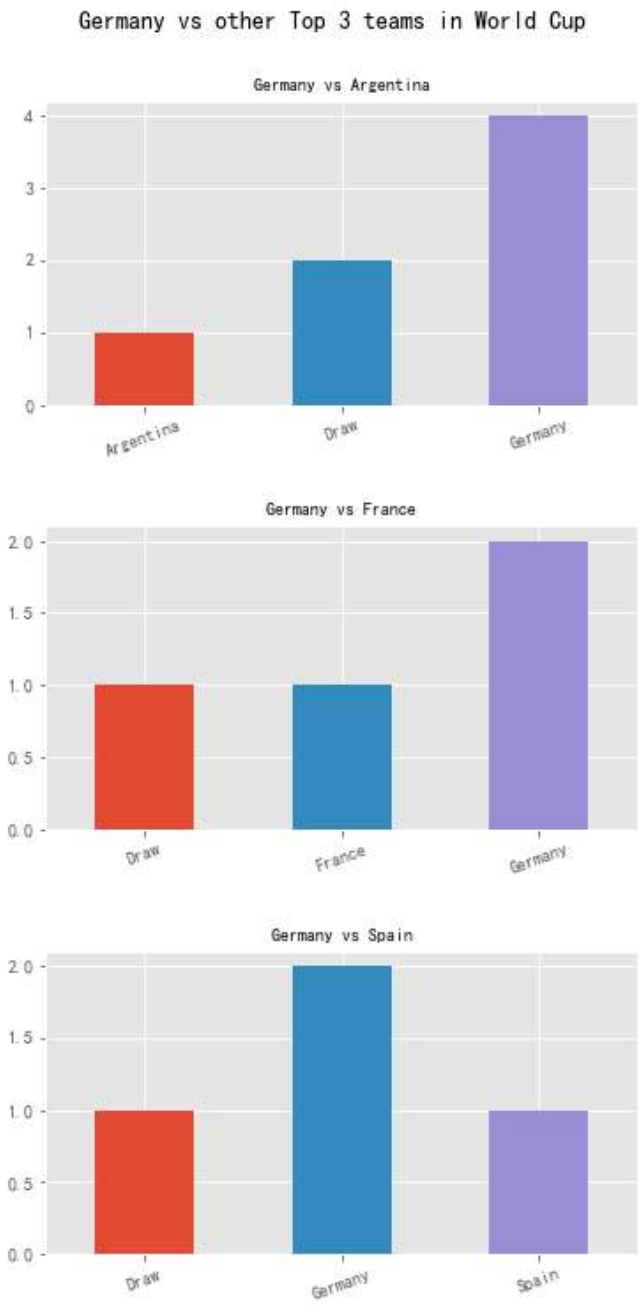


Figure 26.18

> **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。

26.2.2.3 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：

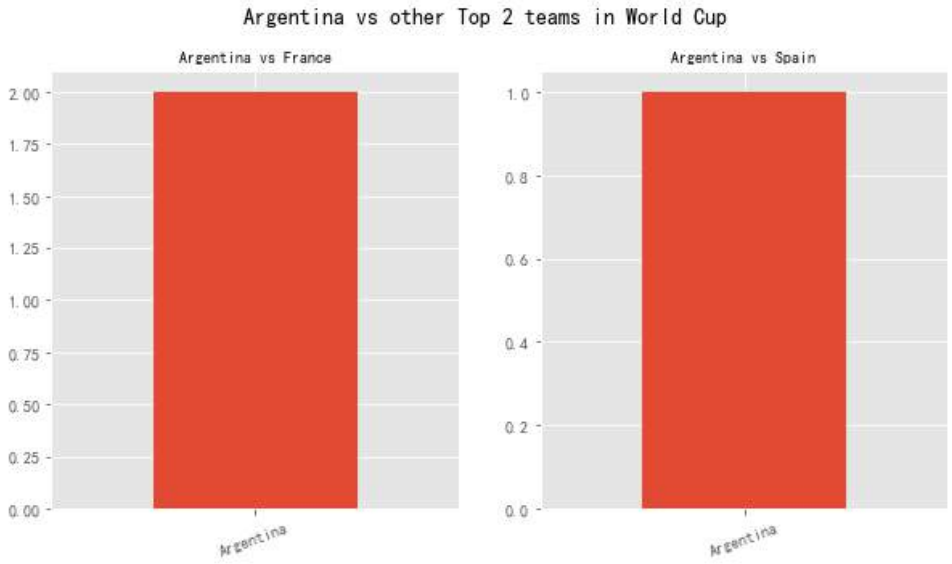


Figure 26.19

> **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

26.2.2.4 世界杯战绩：西班牙 vs 法国

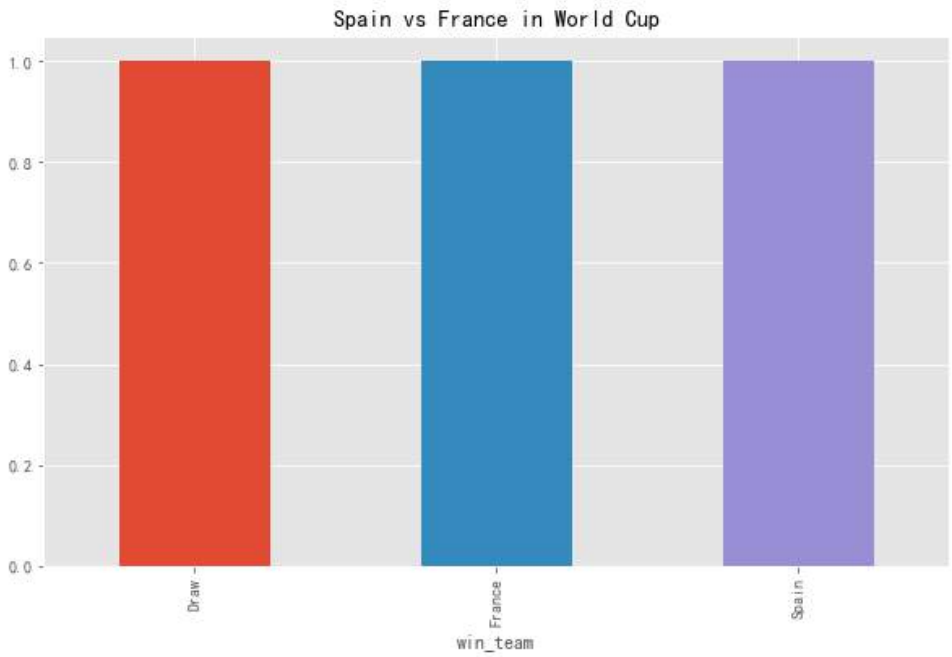


Figure 26.20

> **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。

26.2.3 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。

26.2.3.1 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')
```

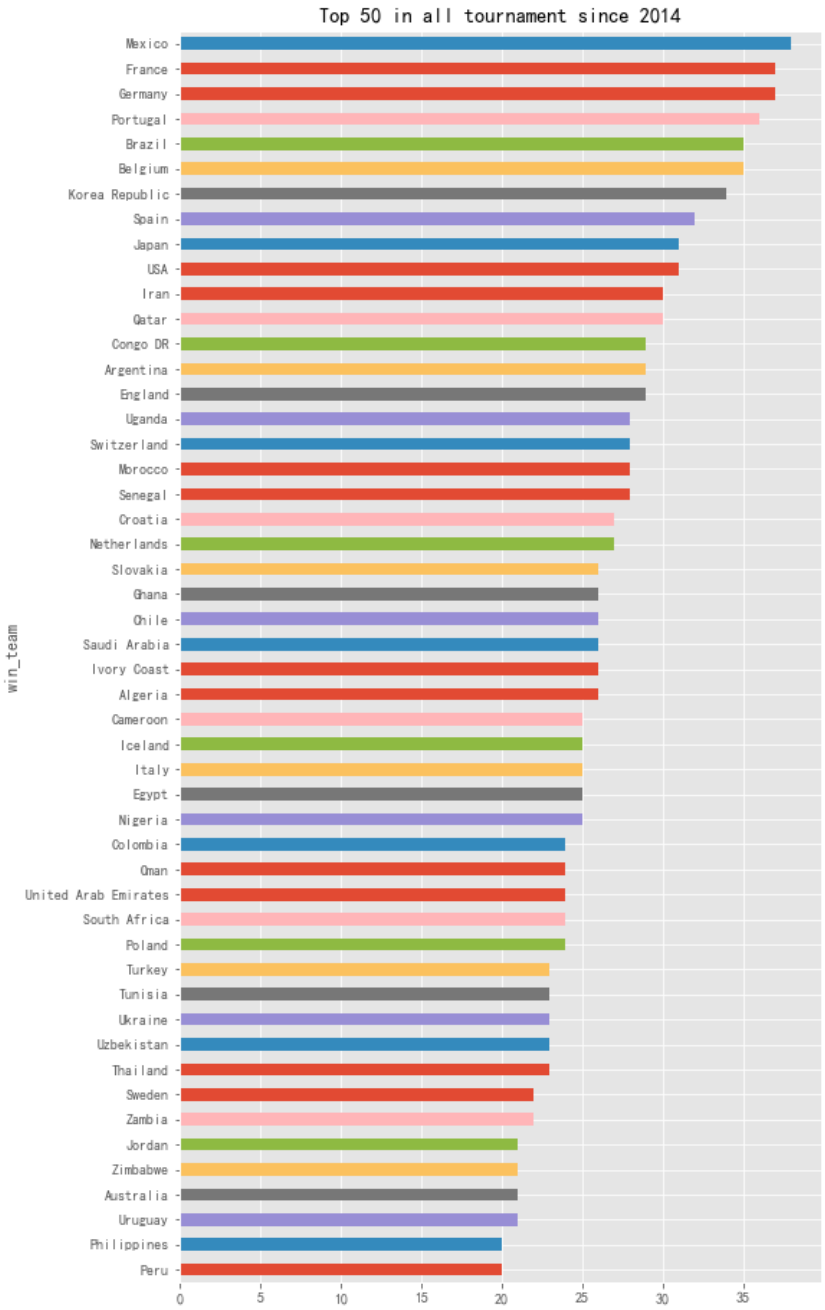


Figure 26.21

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。**

26.2.3.2 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('')
```

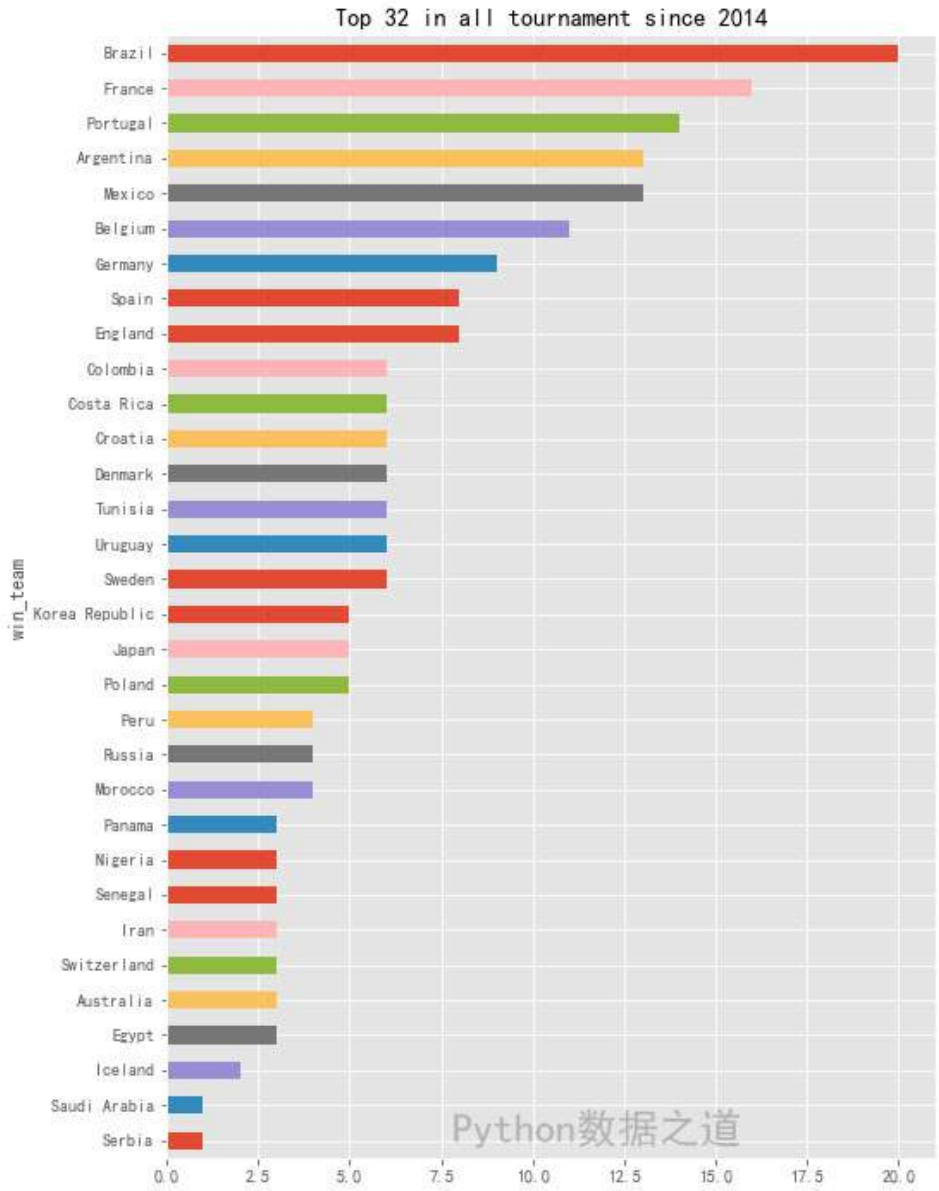


Figure 26.22

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')
```

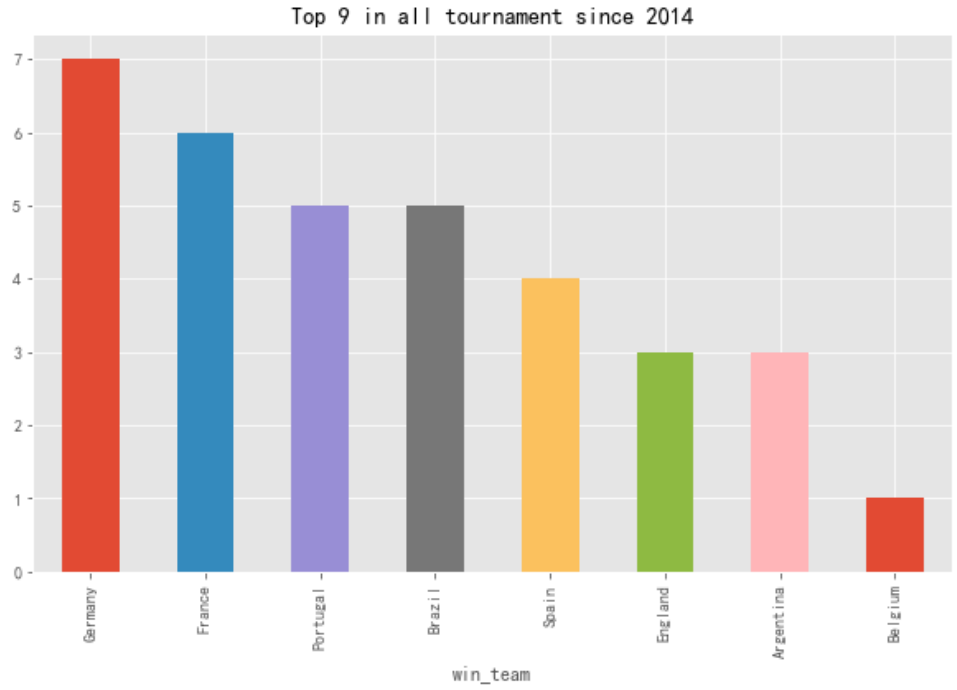


Figure 26.23

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)
```

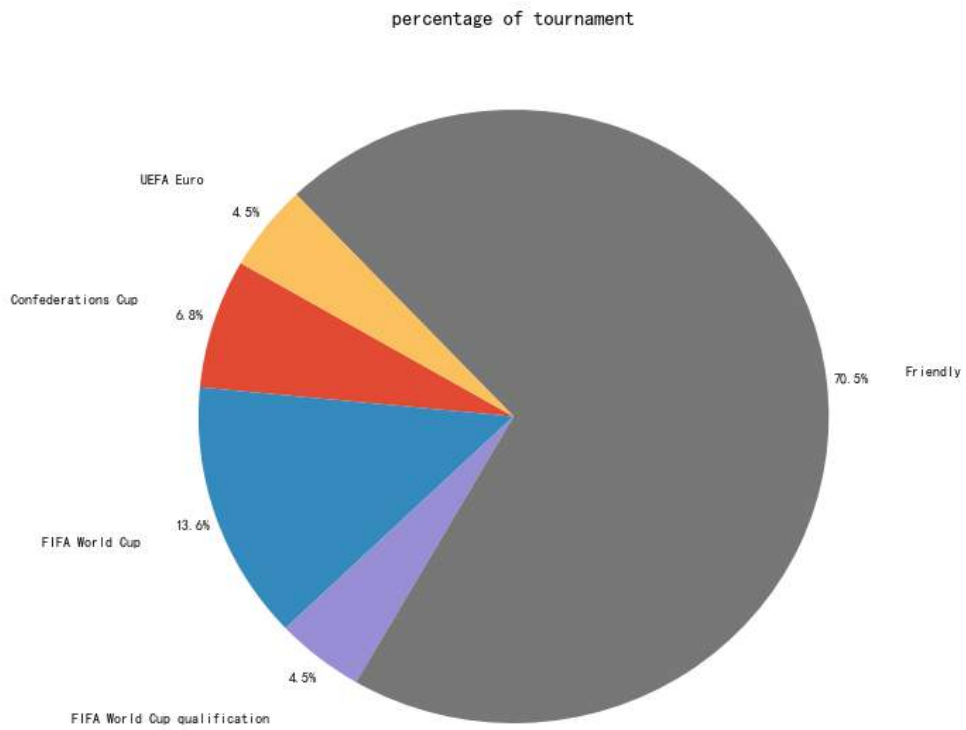


Figure 26.24

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。

26.2.3.3 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：

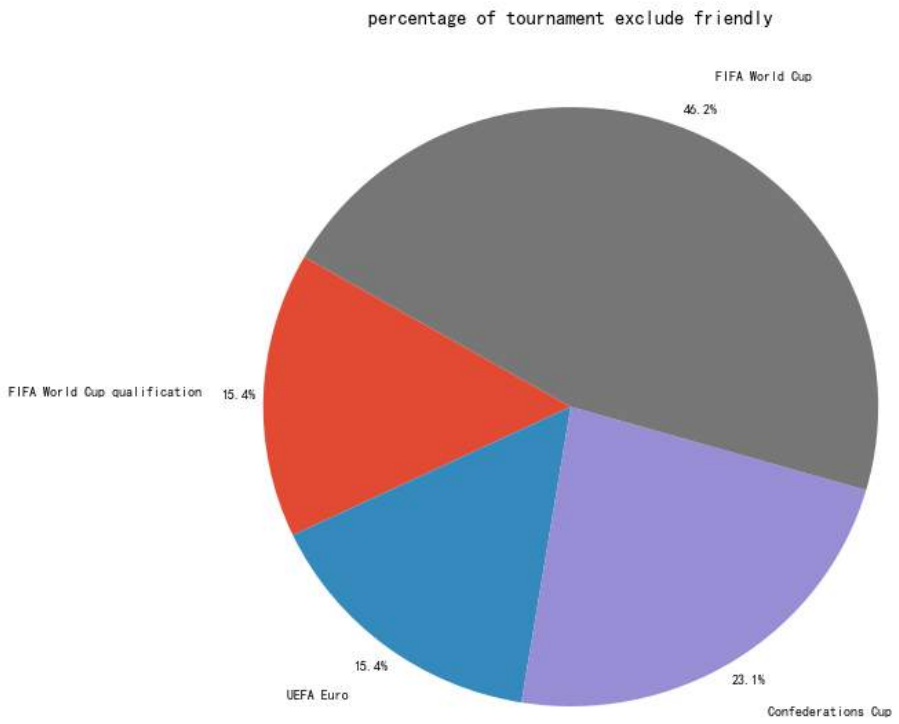


Figure 26.25

剔除友谊赛后，Top 9 的情况如下：

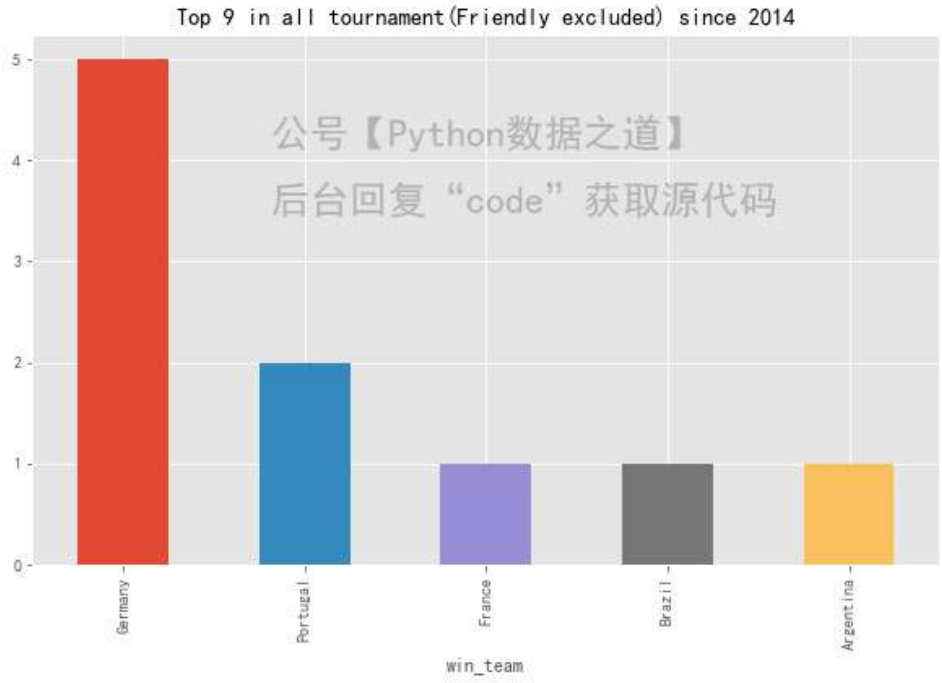


Figure 26.26

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**。

26.2.3.4 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()
```

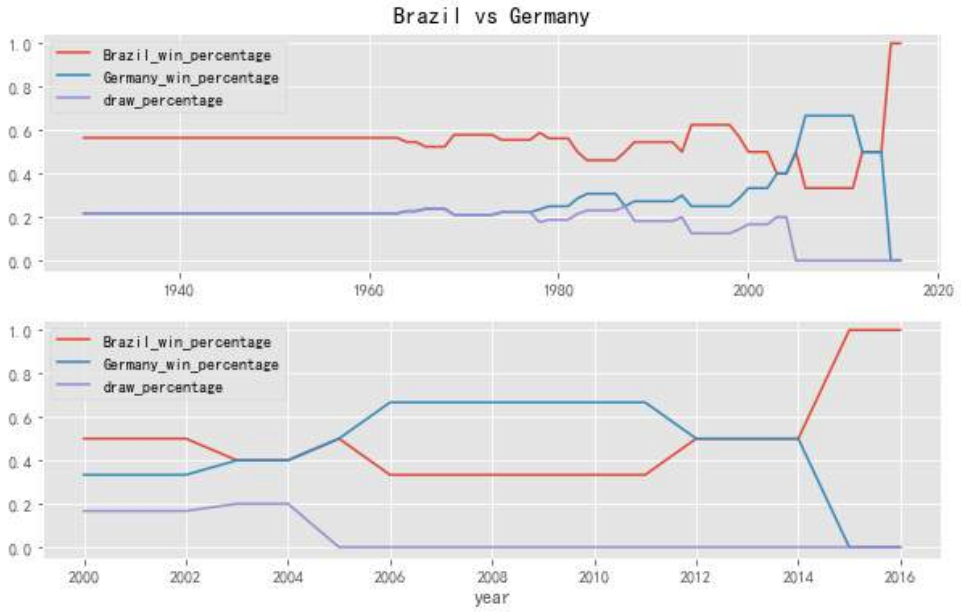


Figure 26.27

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。

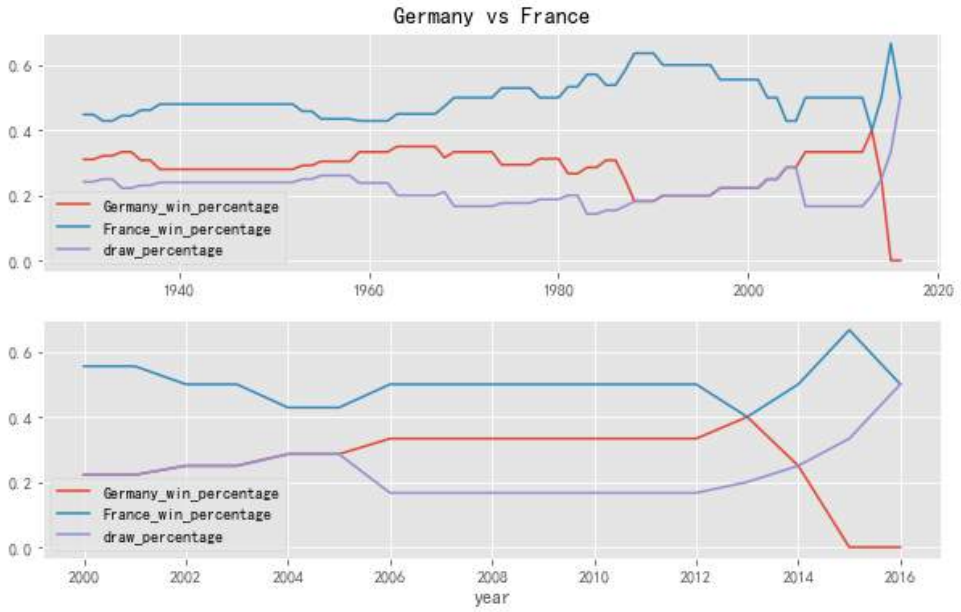


Figure 26.28

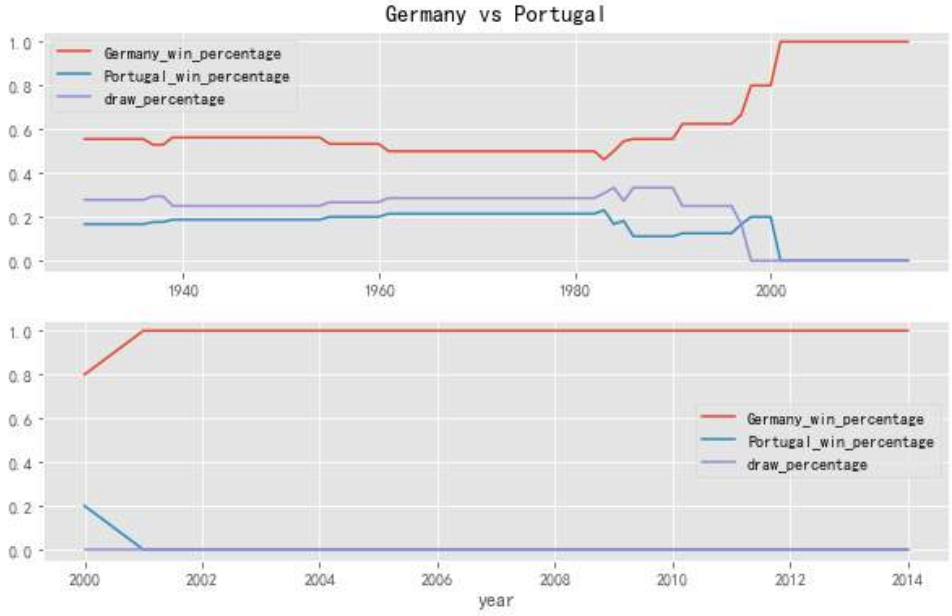


Figure 26.29

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……

26.2.4 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。

Figure 26.30

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 ~~

Figure 26.31

> **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”

27 项目实战：福布斯系列

27.1 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。

27.2 数据采集

27.2.1 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。

27.2.2 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
```

27.2.2.1 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。

27.2.2.2 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
```

27.2.2.3 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。

27.2.2.4 数据存储

采集的数据，最后保存到 csv 文件中。模块代码如下：

```
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)
```

27.2.2.5 数据采集主函数

```
def get_forbes_global_year_2007(year=2007):
    url = 'url'
    html = download(url)
    # print(html)

    data_first_page = get_content_first_page(html, year)
    # print(data_first_page)
    save_data_to_csv_file(data_first_page, 'forbes_'+str(year)+'.csv')

    page_urls = get_page_urls(html)
    # print(page_urls)

    for url in page_urls:
        html = download(url)
        data_other_page = get_content_other_page(html, year)
        # print(data_other_page)
        print('saving data ...', url)
        save_data_to_csv_file(data_other_page, 'forbes_'+str(year)+'.csv')


if __name__ == '__main__':

    # get data from Forbes Global 2000 in Year 2009
    get_forbes_global_year_2007()
```

27.2.3 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。

27.3 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。

27.3.1 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。

27.3.2 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。

27.3.2.1 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：

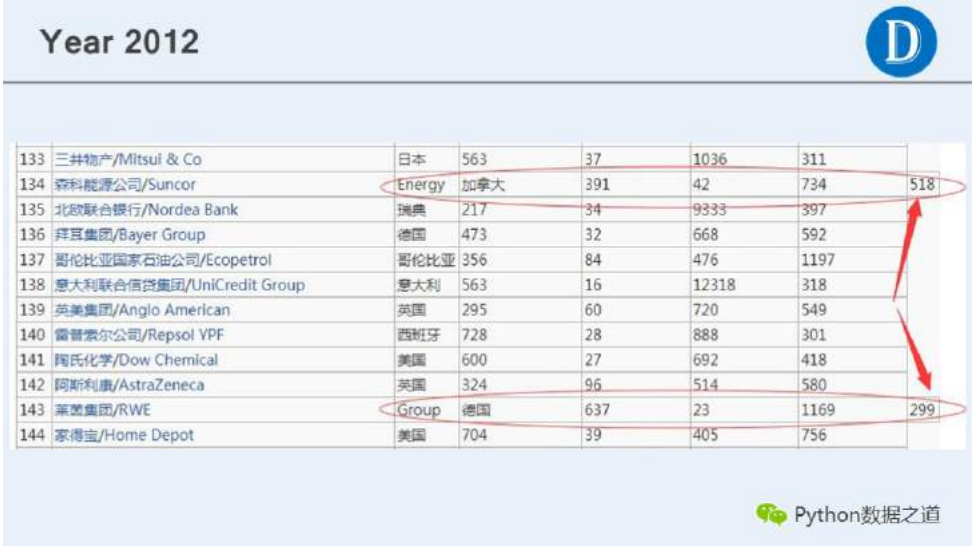


Figure 27.1

调整后的 csv 文件如下：

Figure 27.2

2014 年的情况

原始数据以及调整后的 csv 文件如下：

Figure 27.3

Figure 27.4

27.3.2.2 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。

Figure 27.5

27.3.3 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。

27.4 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。

27.4.1 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：

Figure 27.6

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

Figure 27.7

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: PyDataRoad
'''

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)
        data_other_country = get_content_other_country(html, country_name)
        # print(data_other_country)
        print('saving data ...', country_name)
        save_data_to_csv_file(data_other_country, 'forbes_economywatch_2013.csv
            ')


if __name__ == '__main__':

    # get data from Forbes Global 2000 in Year 2013
    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。

27.4.2 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 ~~

27.5 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

27.5.1 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

27.5.2 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

Figure 27.8

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	1458.74	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

Figure 27.9

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。

27.5.3 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。

27.5.3.1 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

Figure 27.10

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

Figure 27.11

27.5.3.2 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

Figure 27.12

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

Figure 27.13

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

Figure 27.14

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
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

Figure 27.15

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

Figure 27.16

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
```

27.5.3.3 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

Figure 27.17

27.5.3.4 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)]
```

Figure 27.18

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)
```

27.5.3.5 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	1458.74	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

Figure 27.19

27.5.4 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。

27.6 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

27.6.1 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

27.6.2 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	2,348.98	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	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	356.6	40.61	242.08	465.51

Figure 27.20

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	136.89
4	2008	5		ExxonMobil	埃克森美孚公司			United States		Oil & Gas Operations	356.60	40.61	242.08	465.51

Figure 27.21

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。

27.6.3 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()
```

27.6.4 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。

27.7 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

27.7.1 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

27.7.2 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	202.8
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

Figure 27.22

27.7.3 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。

27.7.3.1 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。

27.7.3.2 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。

27.7.3.3 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:])
```

```
9
10 # 将数据单位转换成十亿美元
11 df_2013_all[['Sales','Profits','Assets','Market_value']] =df_2013_all[['Sales',
       'Profits','Assets','Market_value']].apply(lambda x: x/10)
12
13 # 添加年份2013
14 df_2013_all['Year'] = 2013
15
16 # 添加空白列
17 df_2013_all['Country_en'],df_2013_all['Country_cn_en'], df_2013_all['
       Industry_cn'], df_2013_all['Industry_en'] = ['','','','']
18
19 df_2013_all['Rank'] = pd.to_numeric(df_2013_all['Rank'])
20
21 # 按指定list重新将columns进行排序
22 df_2013_all = df_2013_all.reindex(columns=columns_sort)
23 print(df_2013_all.shape)
24 print(df_2013_all.dtypes)
25 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。

27.7.4 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：

```
1 # 数据有2020行，有重复行，需要去除重复行
2 # inplace=True，使去重生效
3 df_2015.drop_duplicates('Company_cn_en', inplace=True)
```

查看是否还有重复的内容：

```
1 # 查看'Company_cn_en'是否还有重复行
2 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。

27.7.5 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。

27.8 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

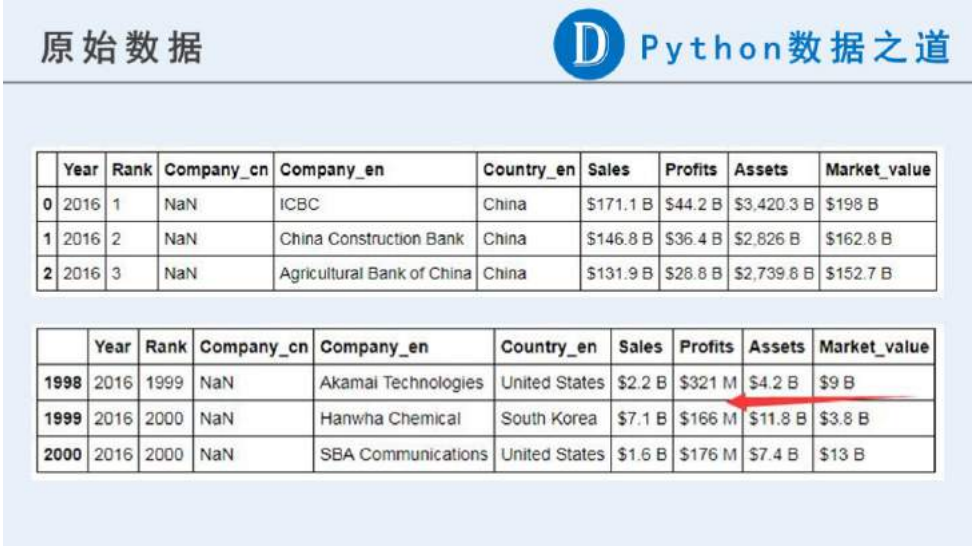
本文通过实例操作来介绍用 pandas 进行数据整理。

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	$2,626 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

Figure 27.23

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

Figure 27.24

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。

27.8.1 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。

27.8.2 处理方法 Method-2

27.8.2.1 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

27.8.2.2 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。

27.8.2.3 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。

27.8.3 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。

27.8.3.1 加载数据

第一步还是加载数据，跟 Method-1 是一样的。

27.8.3.2 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
```

27.8.3.3 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()
```

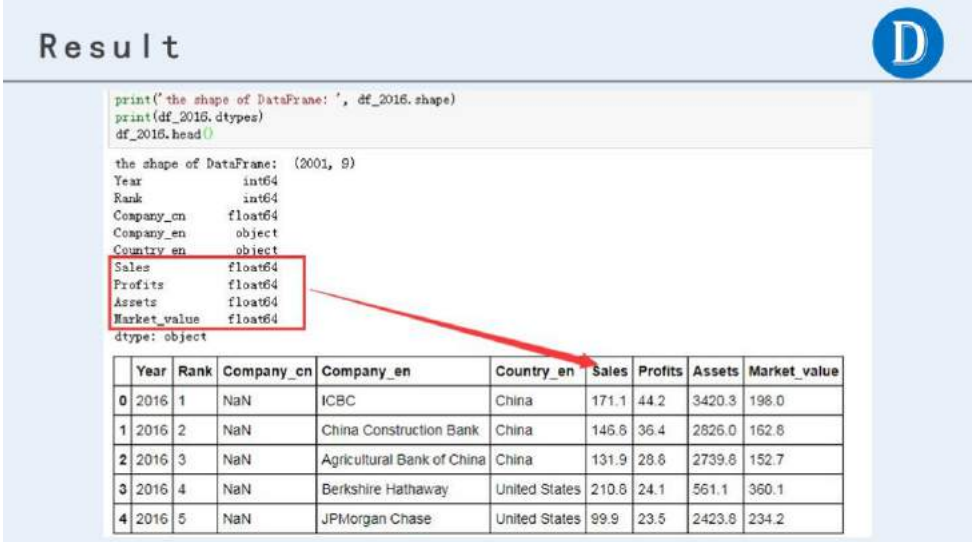
当然，如果 DataFrame 的列数特别多，可以用 for 循环，这样代码更简洁。代码如下：

```
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	146.6	36.4	2826.0	162.6
2	2016	3	NaN	Agricultural Bank of China	China	131.9	28.6	2739.6	152.7
3	2016	4	NaN	Berkshire Hathaway	United States	210.6	24.1	561.1	360.1
4	2016	5	NaN	JPMorgan Chase	United States	99.9	23.5	2423.6	234.2

Figure 27.25

27.9 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

27.9.1 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

27.9.2 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。

27.9.3 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()
```

27.9.4 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

27.9.4.1 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']
```

27.9.4.2 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。

27.9.4.3 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',
```

```
4                                                   r'Netherland)/United
                                                        Kingdom(United
                                                        Kingdom', r'Panama
                                                        )/United Kingdom(
                                                        United Kingdom',
5                                                   r'SA)/United Kingdom(
                                                        United Kingdom', r
                                                        'United Kingdom)/
                                                        Australia(SA', r'
                                                        United Kingdom)/
                                                        Netherland(
                                                        Netherland',
6                                                   r'United Kingdom)/
                                                        South Africa(SA' ,
                                                         'Netherland/
                                                        United Kingdom', '
                                                        Panama/United
                                                        Kingdom',
7                                                   'Australia/United
                                                        Kingdom'],
8                                                   ['China-Taiwan', '
                                                        China-HongKong', '
                                                        United Kingdom/
                                                        Australia', '
                                                        United Kingdom/
                                                        Netherland',
9                                                   'United Kingdom/
                                                        Panama', 'United
                                                        Kingdom/Australia'
                                                        , 'United Kingdom/
                                                        Australia',
10                                                  'United Kingdom/
                                                        Netherland', '
                                                        United Kingdom/
                                                        South Africa', '
                                                        United Kingdom/
                                                        Netherland',
11                                                  'United Kingdom/
                                                        Panama', 'United
                                                        Kingdom/Australia'
                                                        ])
```

27.9.4.4 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')
```

27.9.5 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。

Figure 27.26