Python 数据科学速查表
导入数据

用 Python 导入数据

大多数情况下，都是用 Numpy 或 Pandas 导入数据。

```
>>> import numpy as np
>>> import pandas as pd
```

调用帮助

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

文本文件

纯文本文件

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

以只读方式读取文件
读取文件内容
查看文件是否已经关闭
关闭文件

使用上下文管理器 with

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

读取一行

表格数据：文本文件

用 Numpy 导入文本文件

单数据类型文件

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

用于分割各列值的字符
跳过前两行
读取并使用第1列和第3列
使用的数据类型

多数据类型文件

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

导入时查找列名

```
>>> data_array = np.recfromcsv(filename)
```

np.recfromcsv() 函数的 dtype 默认值为 None。

用 Pandas 导入文本文件

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t'
                       comment='#',
                       na_values[""])
```

读取的行数
用哪一行做列名
用于分隔各列的字符
用于分割注释的字符
读取时，哪些值为NA/NaN

Excel表

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

使用sheet_names属性访问表单名称：

```
>>> data.sheet_names
```

SAS 文件

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata 文件

```
>>> data = pd.read_stata('urbanpop.dta')
```

关系型数据库文件

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

使用 table_names() 方法获取表名列表：

```
>>> table_names = engine.table_names()
```

查询关系型数据库

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

使用上下文管理器 with

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

使用Pandas 查询关系型数据库

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

探索数据

Numpy 数组

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

查看数组元素的数据类型
查看数组维度
查看数组长度

Pandas 数据框

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

返回数据框的前几行，默认为5行
放回数据框的后几行，默认为5行
查看数据框的索引
查看数据框的列名
查看数据框各列的信息
将数据框转换为 Numpy 数组

Pickled 文件

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 文件

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab 文件

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

探索字典

通过函数访问数据元素

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

输出字典的键值 (Key)
输出字典的键值 (Key)
返回字典的值
返回由元组构成字典键值对列表

通过键访问数据

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

探索 HDF5 的结构
提取某个键对应的值

探寻文件系统

魔法命令

```
!ls
%cd ..
%pwd
```

列出目录里的子目录和文件夹
改变当前工作目录
返回当前工作目录的路径

os 库

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

将当前工作目录存为字符串
将目录里的内容输出为列表
改变当前的工作目录
重命名文件
删除现有文件
新建文件夹

原文作者 DataCamp
Learn R for Data Science Interactively

Python 数据科学 *速查表*

Python 基础

变量与数据类型

变量赋值

```
>>> x=5
>>> x
 5
```

变量计算

```
>>> x+2          加
 7
>>> x-2          减
 3
>>> x*2          乘
 10
>>> x**2         幂
 25
>>> x%2          取余
 1
>>> x/float(2)   除
 2.5
```

类型与类型转换

str()	'5', '3.45', 'True'	转为字符串
int()	5, 3, 1	转为整数
float()	5.0, 1.0	转为浮点数
bool()	True, True, True	转为布尔值

调用帮助

```
>>> help(str)
```

列表

参阅 Numpy 数组

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

选择列表元素

索引始于0

子集

```
>>> my_list[1]      选择索引1对应的值
>>> my_list[-3]     选择倒数第3个索引对应的值
```

切片

```
>>> my_list[1:3]    选取索引1和2对应的值
>>> my_list[1:]     选取索引0之后对应的值
>>> my_list[:3]     选取索引3之前对应的值
>>> my_list[:]      复制列表
```

子集列表的列表

```
>>> my_list2[1][0]  my_list[list][itemOfList]
>>> my_list2[1][:2]
```

列表操作

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

列表方法

```
>>> my_list.index(a)          获取某值的索引统计
>>> my_list.count(a)          某值出现的次数追加
>>> my_list.append('!')       某值
>>> my_list.remove('!')       移除某值
>>> del(my_list[0:1])         删除某值
>>> my_list.reverse()         反转列表
>>> my_list.extend('!')       添加某值
>>> my_list.pop(-1)           移除某值
>>> my_list.insert(0,'!')     插入某值
>>> my_list.sort()            列表排序
```

字符串

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

字符串运算

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

字符串操作

索引始于0

```
>>> my_string[3]
>>> my_string[4:9]
```

字符串方法

```
>>> my_string.upper()                  设为大写字符
>>> my_string.lower()                  设为小写字符
>>> my_string.count('w')               统计某字符出现的次数
>>> my_string.replace('e', 'i')        替换字符
>>> my_string.strip()                  清除空格
```

Python库

导入库

```
>>> import numpy
>>> import numpy as np
```

导入指定功能

```
>>> from math import pi
```

pandas 数据分析　　learn 机器学习

NumPy 科学计算　　matplotlib 二维视图

安装 Python

ANACONDA. Python 首选开源数据科学平台

spyder Anaconda 内置的免费IDE

jupyter 创建包含代码、可视图与文本的文档

Numpy 数组

参阅 列表

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

选取 Numpy 数组的值

索引始于0

子集

```
>>> my_array[1]
   2
```

选择索引1对应的值

切片

```
>>> my_array[0:2]
   array([1, 2])
```

选择索引0和1对应的值

二维 Numpy 数组的子集

```
>>> my_2darray[:,0]
   array([1, 4])
```

my_2darray[rows, columns]

Numpy 数组运算

```
>>> my_array > 3
  array([False, False, False,  True], dtype=bool)
>>> my_array * 2
   array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
  array([6, 8, 10, 12])
```

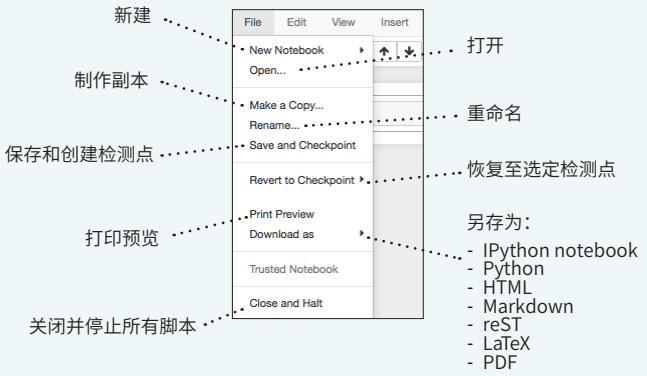
Numpy 数组函数

```
>>> my_array.shape                获取数组形状
>>> np.append(other_array)        追加数据
>>> np.insert(my_array, 1, 5)     插入数据
>>> np.delete(my_array,[1])       删除数据
>>> np.mean(my_array)             平均值
>>> np.median(my_array)           中位数
>>> my_array.corrcoef()           相关系数
>>> np.std(my_array)              标准差
```

原文作者

Python 数据科学 速查表
Jupyter Notebook

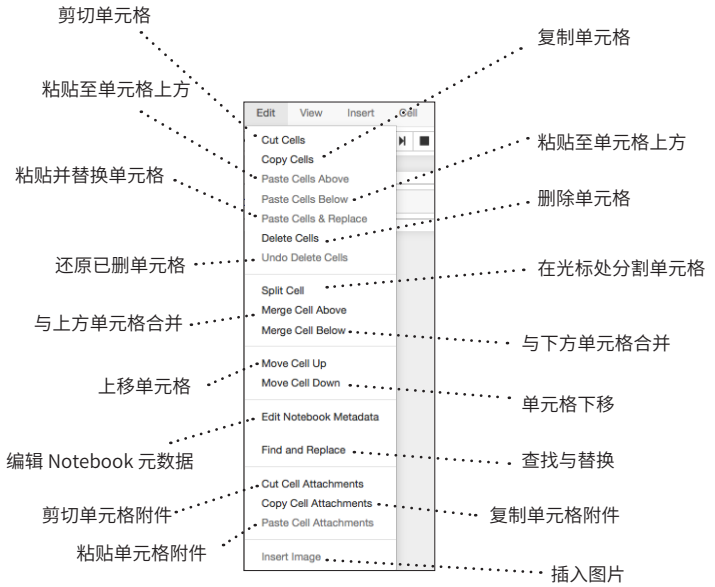
保存/加载



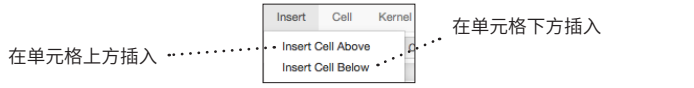
编写代码与文本

Jupyter 将代码与文本封装为三种类型的单元格：Markdown、代码与 NBConvert。

编辑单元格



插入单元格

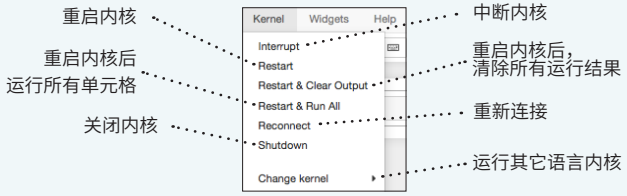


适用多种编程语言

Jupyter 提供了三种编程语言内核：

IP[y]: IPython　IRkernel　IJ[o] IJulia

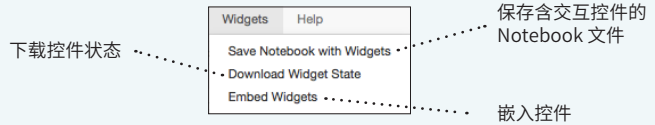
安装 Jupyter 时会自动安装 IPython 内核



Widget 控件

Widget 控件用于控制数据、实现数据可视化，包括滚动条、文本框等。

可用于创建交互式 GUI，或在 Python 和 JavaScript 之间同步状态。

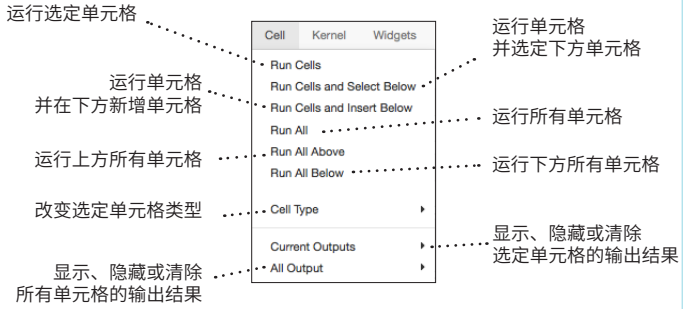


命令模式：

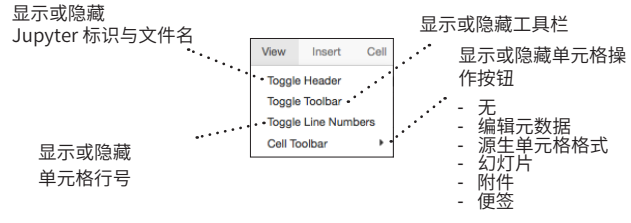
编辑模式：

1. 保存文件和检测点
2. 在下方插入单元格
3. 剪切单元格
4. 复制单元格
5. 在下方粘贴单元格
6. 单元格上移
7. 单元格下移
8. 运行当前单元格
9. 中断内核
10. 重启内核
11. 单元格类型
12. 打开命令控制台
13. 当前内核
14. 内核状态
15. 注销 Notebook 服务器

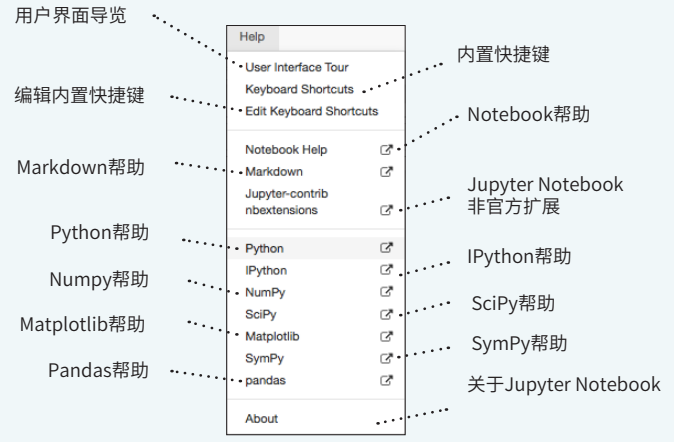
运行单元格



查看单元格



帮助



原文作者

Python数据科学 *速查表*

Matplotlib

Matplotlib

Matplotlib 是 Python 的二维绘图库，用于生成符合出版质量或跨平台交互环境的各类图形。

matplotlib

1 准备数据

参阅 列表与 NumPy

一维数据

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

二维数据或图片

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 绘制图形

```
>>> import matplotlib.pyplot as plt
```

画布

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

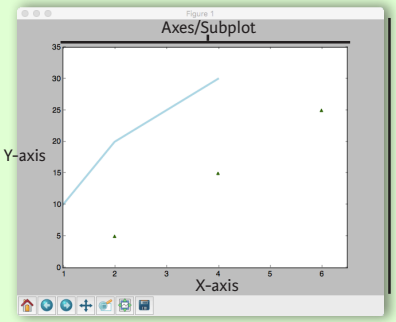
坐标轴

图形是以坐标轴为核心绘制的，大多数情况下，子图就可以满足需求。子图是栅格系统的坐标轴。

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

图形解析与工作流

图形解析



Figure

工作流

Matplotlib 绘图的基本步骤：

1 准备数据 2 创建图形 3 绘图 4 自定义设置 5 保存图形 6 显示图形

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 自定义图形

颜色、色条与色彩表

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

标记

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

线型

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

文本与标注

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

数学符号

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

尺寸限制、图例和布局

尺寸限制与自动调整

```
>>> ax.margins(x=0.0,y=0.1)                  添加内边距
>>> ax.axis('equal')                         将图形纵横比设置为1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    设置x轴与y轴的限制
>>> ax.set_xlim(0,10.5)                      设置x轴的限制
```

图例

```
>>> ax.set(title='An Example Axes',          设置标题与x、y轴的标签
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    自动选择最佳的图例位置
```

标记

```
>>> ax.xaxis.set(ticks=range(1,5),           手动设置X轴刻度
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 设置Y轴长度与方向
                   direction='inout',
                   length=10)
```

子图间距

```
>>> fig3.subplots_adjust(wspace=0.5,         调整子图间距
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       设置画布的子图布局
```

坐标轴边线

```
>>> ax1.spines['top'].set_visible(False)     隐藏顶部坐标轴线
>>> ax1.spines['bottom'].set_position(('outward',10))   设置底部边线的位置为outward
```

3 绘图例程

一维数据

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     用线或标记连接点
>>> ax.scatter(x,y)                          缩放或着色未连接的点
>>> axes[0,0].bar([1,2,3],[3,4,5])           绘制等宽纵向矩形
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      绘制等高横向矩形
>>> axes[1,1].axhline(0.45)                  绘制与轴平行的横线
>>> axes[0,1].axvline(0.65)                  绘制与轴垂直的竖线
>>> ax.fill(x,y,color='blue')                绘制填充多边形
>>> ax.fill_between(x,y,color='yellow')      填充y值和0之间
```

向量场

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             为坐标轴添加箭头
>>> axes[1,1].quiver(y,z)                    二维箭头
>>> axes[0,1].streamplot(X,Y,U,V)            二维箭头
```

数据分布

```
>>> ax1.hist(y)                              直方图
>>> ax3.boxplot(y)                           箱形图
>>> ax3.violinplot(z)                        小提琴图
```

二维数据或图片

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      色彩表或RGB数组
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   二维数组伪彩色图
>>> axes2[0].pcolormesh(data)                二维数组等高线伪彩色图
>>> CS = plt.contour(Y,X,U)                  等高线图
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)                  等高线图标签
```

5 保存

保存画布

```
>>> plt.savefig('foo.png')
```

保存透明画布

```
>>> plt.savefig('foo.png', transparent=True)
```

6 显示图形

```
>>> plt.show()
```

关闭与清除

```
>>> plt.cla()                                清除坐标轴
>>> plt.clf()                                清除画布
>>> plt.close()                              关闭窗口
```

Python 数据科学 速查表
Seaborn

用 Seaborn 绘制统计型数据可视图

Seaborn 是基于 matplotlib 开发的高阶Python 数据可视图库，
用于绘制优雅、美观的统计图形。

使用下列别名导入该库：

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

使用 Seaborn 创建图形的基本步骤：

1.准备数据
2.设定画布外观
3.使用 Seaborn 绘图
4.自定义图形

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    第1步
>>> sns.set_style("whitegrid")                         第2步
>>> g = sns.lmplot(x="tip",                            第3步
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         第4步
>>> plt.title("title")
>>> plt.show(g)                                        第5步
```

1 数据 参阅列表、Numpy 及 Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn 提供了内置数据集：

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 画布外观 参阅 Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	创建画布与子图

Seaborn 样式

`>>> sns.set()`	设置或重置 Seaborn 默认值
`>>> sns.set_style("whitegrid")`	设置 matplotlib 参数
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	
`>>> sns.axes_style("whitegrid")`	返回参数字典或用with设置临时样式

上下文函数

`>>> sns.set_context("talk")`	将上下文设置为 "talk"
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	将上下文设置为 "notebook"，缩放字体，覆盖参数映射

调色板

`>>> sns.set_palette("husl",3)`	定义调色板
`>>> sns.color_palette("husl")`	使用 with 临时设置调色板
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	设置调色板

3 使用 Seaborn 绘图

坐标轴栅格

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	绘制条件关系的子图栅格
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	在分面栅格上绘制分类图
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	绘制适配分面栅格的数据与回归模型

`>>> h = sns.PairGrid(iris)`	绘制配对关系的子图栅格
`>>> h = h.map(plt.scatter)`	绘制配对的双变量分布
`>>> sns.pairplot(iris)`	绘制双变量图的边际单变量图栅格
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	绘制双变量分布

各类图形

散点图

`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	含分类变量的散点图
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	不重叠分类散点图

条形图

`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	用散点图示符显示点估计值和置信区间

计数图

`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	显示观测数量

点图

`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	用柱状图显示点估计和置信区间

箱型图

`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	箱形图
`>>> sns.boxplot(data=iris,orient="h")`	使用宽表数据的箱型图

小提琴图

`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	小提琴图

回归图

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	绘制与线性回归模型拟合的数据

分布图

`>>> plot = sns.distplot(data.y, kde=False, color="b")`	绘制单变量分布

矩阵图

`>>> sns.heatmap(uniform_data,vmin=0,vmax=1)`	热力图

4 深度自定义 参阅 Matplotlib

Axisgrid 对象

`>>> g.despine(left=True)`	移除左框
`>>> g.set_ylabels("Survived")`	设置Y轴的标签
`>>> g.set_xticklabels(rotation=45)`	设置X轴刻度标签
`>>> g.set_axis_labels("Survived", "Sex")`	设置坐标轴标签
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	设置X与Y轴的限制和刻度

图形

`>>> plt.title("A Title")`	添加图形标题
`>>> plt.ylabel("Survived")`	调整y轴标签
`>>> plt.xlabel("Sex")`	调整x轴标签
`>>> plt.ylim(0,100)`	调整y轴限制
`>>> plt.xlim(0,10)`	调整x轴限制
`>>> plt.setp(ax,yticks=[0,5])`	调整图形属性
`>>> plt.tight_layout()`	调整子图参数

5 显示或保存图形 参阅 Matplotlib

`>>> plt.show()`	显示图形
`>>> plt.savefig("foo.png")`	将画布保存为图形
`>>> plt.savefig("foo.png", transparent=True)`	保存透明画布

关闭与清除 参阅 Matplotlib

`>>> plt.cla()`	清除坐标轴
`>>> plt.clf()`	清除画布
`>>> plt.close()`	关闭窗口

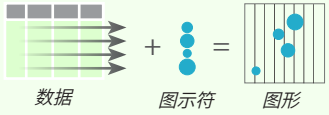
原文作者

Python数据科学 *速查表*
Bokeh

使用 Bokeh 绘图

Bokeh 是 Python 的交互式可视图库，用于生成在浏览器里显示的大规模数据集高性能可视图。

Bokeh 的中间层通用 bokeh.plotting 界面主要为两个组件：
数据与图示符。



使用 bokeh.plotting 界面绘图的基本步骤为：

1. 准备数据
   Python列表、Numpy数组、Pandas数据框或其它序列值
2. 创建图形
3. 为数据添加渲染器，自定义可视化图
4. 指定生成的输出类型
5. 显示视图或保存结果

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                          Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",      Step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)   Step 3
>>> output_file("lines.html")                    Step 4
>>> show(p)                                      Step 5
```

1 数据

参阅列表、Numpy 及 Pandas

通常，Bokeh在后台把数据转换为列数据源，不过也可手动转换：

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

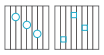
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 绘图

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

3 渲染器与自定义可视化

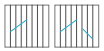
图示符



散点标记

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
```

线型图示符



```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

自定义图示符

参阅 数据

图示符选择与反选

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

绘图区内部

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None,
mode='vline') >>> p3.add_tools(hover)
```

色彩表

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
              color=dict(field='origin',
                         transform=color_mapper),
              legend='Origin')
```

图例位置

绘图区内部

```
>>> p.legend.location = 'bottom_left'
```

绘图区外部

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" ,[p1, r1]),("Two",[r2])],
                    location=(0, -30))
>>> p.add_layout(legend, 'right')
```

图例方向

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

图例背景与边框

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

行列布局

行

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

列

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

行列嵌套

```
>>>layout = row(column(p1,p2), p3)
```

栅格布局

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

标签布局

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

链接图

链接坐标轴

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

链接刷

```
>>> p4 = figure(plot_width = 100,
                tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200,
                tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

4 输出与导出

Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

HTML

脱机HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
```

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

组件

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

5 显示或保存图形

```
>>> show(p1)
>>> save(p1)
```

```
>>> show(layout)
>>> save(layout)
```

原文作者

Python 数据科学 *速查表*
Numpy 基础

NumPy

Numpy 是 Python 数据科学计算的核心库，提供了高性能的多维数组对象及处理数组的工具。

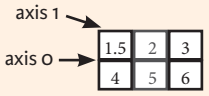
使用以下语句导入 Numpy 库：

```
>>> import numpy as np
```

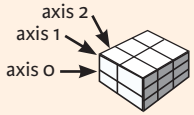
NumPy 数组

1维数组

2维数组



3维数组



创建数组

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

初始化占位符

```
>>> np.zeros((3,4))                创建值为0数组
>>> np.ones((2,3,4),dtype=np.int16) 创建值为1数组
>>> d = np.arange(10,25,5)          创建均匀间隔的数组（步进值）

>>> np.linspace(0,2,9)              创建均匀间隔的数组（样本数）

>>> e = np.full((2,2),7)            创建常数数组
>>> f = np.eye(2)                   创建2x2单位矩阵
>>> np.random.random((2,2))         创建随机值的数组
>>> np.empty((3,2))                 创建空数组
```

输入/输出

保存与载入磁盘上的文件

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

保存与载入文本文件

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

数据类型

```
>>> np.int64       带符号的64位整数
>>> np.float32     标准双精度浮点数
>>> np.complex     显示为128位浮点数的复数
>>> np.bool        布尔值：True值和False值
>>> np.object      Python对象
>>> np.string_     固定长度字符串
>>> np.unicode_    固定长度Unicode
```

数组信息

```
>>> a.shape             数组形状，几行几列
>>> len(a)              数组长度
>>> b.ndim              几维数组
>>> e.size              数组有多少元素
>>> b.dtype             数据类型
>>> b.dtype.name        数据类型的名字
>>> b.astype(int)       数据类型转换
```

调用帮助

```
>>> np.info(np.ndarray.dtype)
```

数组计算

算数运算

```
>>> g = a - b                       减法
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                减法
>>> b + a                           加法
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                     加法
>>> a / b                           除法
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                  除法
>>> a * b                           乘法
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                乘法
>>> np.exp(b)                       幂
>>> np.sqrt(b)                      平方根
>>> np.sin(a)                       正弦
>>> np.cos(b)                       余弦
>>> np.log(a)                       自然对数
>>> e.dot(f)                        点积
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

比较

```
>>> a == b                          对比值
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                           对比值
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)            对比数组
```

聚合函数

```
>>> a.sum()             数组汇总
>>> a.min()             数组最小值
>>> b.max(axis=0)       数组最大值，按行
>>> b.cumsum(axis=1)    数组元素的累加值
>>> a.mean()            平均数
>>> b.median()          中位数
>>> a.corrcoef()        相关系数
>>> np.std(b)           标准差
```

数组复制

```
>>> h = a.view()        使用同一数据创建数组视图
>>> np.copy(a)          创建数组的副本
>>> h = a.copy()        创建数组的深度拷贝
```

数组排序

```
>>> a.sort()            数组排序
>>> c.sort(axis=0)      以轴为依据对数组排序
```

子集、切片、 索引

参阅 列表

子集

```
>>> a[2]                选择索引2对应的值
  3
>>> b[1,2]              选择行1列2对应的值（等同于b[1][2]
  6.0
```

切片

```
>>> a[0:2]              选择索引为0与1对应的值
  array([1, 2])
>>> b[0:2,1]            选择第1列中第0行、第1行的值
  array([ 2.,  5.])
>>> b[:1]               选择第0行的所有值（等同于b[0:1,:1]
  array([[1.5, 2., 3.]])
>>> c[1,...]            等同于 [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]           反转数组a
  array([3, 2, 1])
```

条件索引

```
>>> a[a<2]              选择数组a中所有小于2的值
  array([1])
```

花式索引

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   选择(1,0),(0,1),(1,2) 和(0,0)所对应的值
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   选择矩阵的行列子集
  array([[ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

数组操作

转置数组

```
>>> i = np.transpose(b)     转置数组
>>> i.T                     转置数组
```

改变数组形状

```
>>> b.ravel()               拉平数组
>>> g.reshape(3,-2)         改变数组形状，但不改变数据
```

添加或删除值

```
>>> h.resize((2,6))         返回形状为(2,6)的新数组
>>> np.append(h,g)          追加数据
>>> np.insert(a, 1, 5)      插入数据
>>> np.delete(a,[1])        删除数据
```

合并数组

```
>>> np.concatenate((a,d),axis=0)    拼接数组
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                纵向以行的维度堆叠数组
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                      纵向以行的维度堆叠数组
>>> np.hstack((e,f))                横向以列的维度堆叠数组
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))          以列的维度创建堆叠数组
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                      以列的维度创建堆叠数组
```

分割数组

```
>>> np.hsplit(a,3)          纵向分割数组为3等份
  [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)          横向分割数组为2等份
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python 数据科学 *速查表*
Pandas 基础

Pandas

Pandas 是基于 Numpy 创建的 Python 库，为 Python 提供了易于使用的**数据结构**和**数据分析**工具。

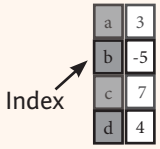
使用以下语句导入 Pandas 库：

```
>>> import pandas as pd
```

Pandas 数据结构

Series - 序列

存储任意类型数据的**一维**数组




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame - 数据框

列 → 存储不同类型数据的**二维**数组

索引

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

调用帮助

```
>>>help(pd.Series.loc)
```

选择

参阅 NumPy Arrays

取值

```
>>> s['b']
 -5
```
取序列的值

```
>>> df[1:]
   Country     Capital   Population
1   India   New Delhi   1303171035
2  Brazil    Brasília    207847528
```
取数据框的子集

选取、布尔索引及设置值

按位置

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```
按行与列的位置选择某值

按标签

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```
按行与列的名称选择某值

按标签/位置

```
>>> df.ix[2]
Country      Brazil
Capital     Brasília
Population  207847528
```
选择某行

```
>>> df.ix[:,'Capital']
0      Brussels
1     New Delhi
2      Brasília
```
选择某列

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

布尔索引

```
>>> s[~(s > 1)]
```
序列 S 中没有大于1的值

```
>>> s[(s < -1)  |  (s > 2)]
```
序列 S 中小于-1或大于2的值

```
>>> df[df['Population']>1200000000]
```
使用筛选器调整数据框

设置值

```
>>> s['a'] = 6
```
将序列 S 中索引为 a 的值设为6

删除数据

```
>>> s.drop(['a', 'c'])
```
按索引删除序列的值 (axis=0)

```
>>> df.drop('Country', axis=1)
```
按列名删除数据框的列(axis=1)

排序和排名

```
>>> df.sort_index()
```
按索引排序

```
>>> df.sort_values(by='Country')
```
按某列的值排序

```
>>> df.rank()
```
数据框排名

查询序列与数据框的信息

基本信息

```
>>> df.shape
```
(行,列))

```
>>> df.index
```
获取索引

```
>>> df.columns
```
获取列名

```
>>> df.info()
```
获取数据框基本信息

```
>>> df.count()
```
非Na值的数量

汇总

```
>>> df.sum()
```
合计

```
>>> df.cumsum()
```
累计

```
>>> df.min()/df.max()
```
最小值除以最大值

```
>>> df.idxmin()/df.idxmax()
```
索引最小值除以索引最大值

```
>>> df.describe()
```
基础统计数据

```
>>> df.mean()
```
平均值

```
>>> df.median()
```
中位数

应用函数

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```
应用匿名函数lambda
应用函数
对每个单元格应用函数

数据对齐

内部数据对齐

如有不一致的索引，则使用NA值：

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a     10.0
b      NaN
c      5.0
d      7.0
```

使用 Fill 方法运算

还可以使用 Fill 方法进行内部对齐运算：

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c    5.0
d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

输入/输出

读取/写入CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

读取/写入Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

读取内含多个表的Excel

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

读取和写入 SQL 查询及数据库表

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql()是 read_sql_table() 与 read_sql_query()的便捷打包器

```
>>> pd.to_sql('myDf', engine)
```

原文作者

Python 数据科学 *速查表*

Pandas 进阶

数据重塑

透视

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

将行变为列

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

→

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

透视表

```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

将行变为列

堆栈 / 反堆栈

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

透视列标签
透视索引标签

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

反堆栈

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

堆栈

融合

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

将列转为行

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

→

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

迭代

```
>>> df.iteritems()
>>> df.iterrows()
```

(列索引，序列) 键值对
(行索引，序列) 键值对

高级索引

参阅 NumPy Arrays

基础选择

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

选择任一值大于1的列
选择所有值大于1的列
选择含 NaN值的列
选择不含NaN值的列

通过isin选择

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

选择为某一类型的数值
选择特定值
选择指定元素

通过Where选择

```
>>> s.where(s > 0)
```

选择子集

通过Query选择

```
>>> df6.query('second > first')
```

查询DataFrame

设置/取消索引

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

设置索引
取消索引
重命名DataFrame列名

重置索引

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

前向填充

```
>>> df.reindex(range(4),
               method='ffill')
   Country     Capital   Population
0  Belgium    Brussels    11190846
1    India   New Delhi  1303171035
2   Brazil    Brasília   207847528
3   Brazil    Brasília   207847528
```

后向填充

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
0  3
1  3
2  3
3  3
4  3
```

多重索引

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

重复数据

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

返回唯一值
查找重复值
去除重复值
查找重复索引

数据分组

聚合

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                              'b': np.sum})
```

转换

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

缺失值

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

去除缺失值NaN
用预设值填充缺失值NaN
用一个值替换另一个值

合并数据

数据1

	X1	X2
a	11.432
b	1.303
c	99.906

数据2

	X1	X3
a	20.784
b	NaN
d	20.784

合并-Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

连接-Join

```
>>> data1.join(data2, how='right')
```

拼接-Concatenate

纵向

```
>>> s.append(s2)
```

横向/纵向

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

日期

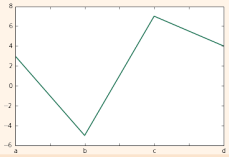
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

可视化

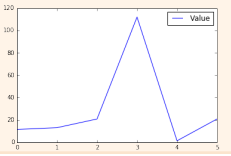
参阅 Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



原文作者

Python 数据科学 *速查表*

SciPy - 线性代数

SciPy

SciPy 是基于 NumPy 创建的 Python 科学计算核心库，提供了众多数学算法与函数。

与NumPy交互

参阅 NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

索引技巧

```
>>> np.mgrid[0:5,0:5]          创建稠密栅格
>>> np.ogrid[0:2,0:2]          创建开放栅格
>>> np.r_[[3,[0]*5,-1:1:10j]   按行纵向堆叠数组按
>>> np.c_[b,c]                 列横向堆叠数组
```

操控形状

```
>>> np.transpose(b)            转置矩阵
>>> b.flatten()                拉平数组
>>> np.hstack((b,c))           按列横向堆叠数组
>>> np.vstack((a,b))           按行纵向堆叠数组
>>> np.hsplit(c,2)             在索引2横向分割数组
>>> np.vpslit(d,2)             在索引2纵向分割数组
```

多项式

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])        创建多项式对象
```

矢量函数

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)       矢量函数
```

类型控制

```
>>> np.real(c)                 返回数组元素的实部
>>> np.imag(c)                 返回数组元素的虚部
>>> np.real_if_close(c,tol=1000)  如果复数接近0，返回实部将
>>> np.cast['f'](np.pi)        对象转化为数据类型
```

常用函数

```
>>> np.angle(b,deg=True)       返回复数的角度
>>> g = np.linspace(0,np.pi,num=5)  创建等差数组（样本数）
>>> g [3:] += np.pi
>>> np.unwrap(g)               解包
>>> np.logspace(0,10,3)        创建等差数组（对数刻度）
>>> np.select([c<4],[c*2])     根据条件返回数组列表的值

>>> misc.factorial(a)          因子
>>> misc.comb(10,3,exact=True) 取K次N项的组合，已改为scipy.special.comb
>>> misc.central_diff_weights(3)  NP点中心导数的权重
>>> misc.derivative(myfunc,1.0)   查找函数在某点的第n个导数
```

线性代数

参阅 NumPy

使用 linalg 和 sparse 模块。注意 scipy.linalg 包含了 numpy.linalg，并扩展了其功能。

```
>>> from scipy import linalg, sparse
```

创建矩阵

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

基础矩阵例程

逆矩阵

```
>>> A.I                        求逆矩阵
>>> linalg.inv(A)              求逆矩阵
>>> A.T                        矩阵转置
>>> A.H                        共轭转置
>>> np.trace(A)                计算对角线元素的和
```

范数

```
>>> linalg.norm(A)             Frobenius 范数
>>> linalg.norm(A,1)           L1 范数 (最大列汇总)
>>> linalg.norm(A,np.inf)      L 范数 (最大列汇总)
```

排名

```
>>> np.linalg.matrix_rank(C)   矩阵排名
```

行列式

```
>>> linalg.det(A)              行列式
```

求解线性问题

```
>>> linalg.solve(A,b)          求解稠密矩阵
>>> E = np.mat(a).T            求解稠密矩阵
>>> linalg.lstsq(D,E)          用最小二乘法求解线性代数方程
```

广义逆

```
>>> linalg.pinv(C)             计算矩阵的伪逆（最小二乘法求解器）

>>> linalg.pinv2(C)            计算矩阵的伪逆（SVD）
```

创建稀疏矩阵

```
>>> F = np.eye(3, k=1)         创建2X2单位矩阵
>>> G = np.mat(np.identity(2)) 创建2X2单位矩阵
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   压缩稀疏行矩阵
>>> I = sparse.csc_matrix(D)   压缩稀疏列矩阵
>>> J = sparse.dok_matrix(A)   DOK矩阵
>>> E.todense()                将稀疏矩阵转为全矩阵
>>> sparse.isspmatrix_csc(A)   单位稀疏矩阵
```

稀疏矩阵例程

逆矩阵

```
>>> sparse.linalg.inv(I)       求逆矩阵
```

范数

```
>>> sparse.linalg.norm(I)      范数
```

解决线性问题

```
>>> sparse.linalg.spsolve(H,I) 稀求解疏矩阵
```

稀疏矩阵函数

```
>>> sparse.linalg.expm(I)      稀疏矩阵指数
```

矩阵函数

加法

```
>>> np.add(A,D)                加法
```

减法

```
>>> np.subtract(A,D)           减法
```

除法

```
>>> np.divide(A,D)             除法
```

乘法

```
>>> np.multiply(D,A)           乘法
>>> np.dot(A,D)                点积
>>> np.vdot(A,D)               向量点积
>>> np.inner(A,D)              内积
>>> np.outer(A,D)              外积
>>> np.tensordot(A,D)          张量点积
>>> np.kron(A,D)               Kronecker 积
```

指数函数

```
>>> linalg.expm(A)             矩阵指数
>>> linalg.expm2(A)            矩阵指数（泰勒级数）
>>> linalg.expm3(D)            矩阵指数（特征值分解）
```

对数函数

```
>>> linalg.logm(A)             矩阵对数
```

三角函数

```
>>> linalg.sinm(D)             矩阵正弦
>>> linalg.cosm(D)             矩阵余弦
>>> linalg.tanm(A)             矩阵切线
```

双曲三角函数

```
>>> linalg.sinhm(D)            双曲矩阵正弦
>>> linalg.coshm(D)            双曲矩阵余弦
>>> linalg.tanhm(A)            双曲矩阵切线
```

矩阵符号函数

```
>>> np.sigm(A)                 矩阵符号函数
```

矩阵平方根

```
>>> linalg.sqrtm(A)            矩阵平方根
```

任意函数

```
>>> linalg.funm(A, lambda x: x*x)  评估矩阵函数
```

矩阵分解

特征值与特征向量

```
>>> la, v = linalg.eig(A)      求解方阵的普通或广义特征值问题

>>> l1, l2 = la                解包特征值
>>> v[:,0]                     第一个特征值
>>> v[:,1]                     第二个特征值
>>> linalg.eigvals(A)          解包特征值
```

奇异值分解

```
>>> U,s,Vh = linalg.svd(B)     奇异值分解（SVD）
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  在 SVD 中构建 Sigma 矩阵
```

LU 分解

```
>>> P,L,U = linalg.lu(C)       LU 分解
```

解构稀疏矩阵

```
>>> la, v = sparse.linalg.eigs(F,1)  特征值与特征向量
>>> sparse.linalg.svds(H, 2)         奇异值分解（SVD）
```

调用帮助

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

原文作者 DataCamp
Learn Python for Data Science Interactively

Python数据科学 *速查表*

Keras

Keras

Keras是强大、易用的深度学习库，基于Theano和TensorFlow提供了高阶神经网络API，用于开发和评估深度学习模型。

示例

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

数据

参阅 NumPy, Pandas & Scikit-Learn

数据要存为 NumPy 数组或数组列表，使用 sklearn.cross_validation 的 train_test_split 模块进行分割将数据分割为训练集与测试集。

Keras 数据集

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

其它

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

模型架构

序贯模型

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

多层感知器（MLP）

二进制分类

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

多级分类

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

回归

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

卷积神经网络（CNN）

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

递归神经网络（RNN）

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

预处理

参阅 NumPy 与 Scikit-Learn

序列填充

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

独热编码

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

训练与测试集

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

标准化/归一化

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

审视模型

```
>>> model.output_shape          模型输出形状
>>> model.summary()             模型摘要展示
>>> model.get_config()          模型配置
>>> model.get_weights()         列出模型的所有权重张量
```

编译模型

多层感知器：二进制分类

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

多层感知器：多级分类

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

多层感知器：回归

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

递归神经网络

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

模型训练

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

评估模型性能

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

预测

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

保存/加载模型

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

模型微调

参数优化

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

早停法

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

原文作者

Python 数据科学 *速查表*
Scikit-learn

Scikit-learn

Scikit-learn 是开源的 Python 库，通过统一的界面实现机器学习、预处理、交叉验证及可视化算法。

简例

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

加载数据

参阅 NumPy 与 Pandas

Scikit-learn 处理的数据是存储为 NumPy 数组或 SciPy 稀疏矩阵的数字，还支持 Pandas 数据框等可转换为数字数组的其它数据类型。

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

训练集与测试集数据

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

数据预处理

标准化

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

归一化

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

二值化

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

编码分类特征

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

输入缺失值

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

生成多项式特征

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

创建模型

有监督学习评估器

线性回归

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

支持向量机(SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

朴素贝叶斯

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

无监督学习评估器

主成分分析(PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

模型拟合

有监督学习

```
>>> lr.fit(X, y)                          拟合数据与模型
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

无监督学习

```
>>> k_means.fit(X_train)                  拟合数据与模型
>>> pca_model = pca.fit_transform(X_train) 拟合并转换数据
```

预测

有监督评估器

```
>>> y_pred = svc.predict(np.random.random((2,5)))   预测标签
>>> y_pred = lr.predict(X_test)                     预测标签
>>> y_pred = knn.predict_proba(X_test)              评估标签概率
```

无监督评估器

```
>>> y_pred = k_means.predict(X_test)      预测聚类算法里的标签
```

评估模型性能

分类指标

准确率

```
>>> knn.score(X_test, y_test)                          评估器评分法
>>> from sklearn.metrics import accuracy_score         指标评分函数
>>> accuracy_score(y_test, y_pred)
```

分类预估评价函数

```
>>> from sklearn.metrics import classification_report  精确度、召回率、F1
>>> print(classification_report(y_test, y_pred))       分数及支持率
```

混淆矩阵

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

回归指标

平均绝对误差

```
>>> from sklearn.metrics import
mean_absolute_error >>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

均方误差

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 评分

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

群集指标

调整兰德系数

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

同质性

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

交叉验证

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

模型调整

栅格搜索

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

随机参数优化

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```


Python 数据科学 *速查表*

PySpark - SQL 基础

PySpark 与 Spark SQL

Spark SQL 是 Apache Spark 处理结构化数据的模块。

初始化 SparkSession

SparkSession 用于创建数据框，将数据框注册为表，执行 SQL 查询，缓存表及读取 Parquet 文件。

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

创建数据框

从 RDD 创建

```
>>> from pyspark.sql.types import *
```

推断 Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

指定 Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +--------+---+
  |    name|age|
  +--------+---+
  |    Mine| 28|
  |   Filip| 29|
  |Jonathan| 30|
  +--------+---+
```

从 Spark 数据源创建

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName |lastName|         phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet 文件

```
>>> df3 = spark.read.load("users.parquet")
```

文本文件

```
>>> df4 = spark.read.text("people.txt")
```

查阅数据信息

```
>>> df.dtypes          返回 df 的列名与数据类型
>>> df.show()          显示 df 的内容
>>> df.head()          返回前 n 行数据
>>> df.first()         返回第 1 行数据
>>> df.take(2)         返回前 n 行数据
>>> df.schema          返回 df 的 Schema
```

```
>>> df.describe().show()    汇总统计数据
>>> df.columns              返回 df 的列名
>>> df.count()              返回 df 的行数
>>> df.distinct().count()   返回 df 中不重复的行数
>>> df.printSchema()        返回 df的 Schema
>>> df.explain()            返回逻辑与实体方案
```

重复值

```
>>> df = df.dropDuplicates()
```

查询

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()                  显示 firstName 列的所有条目
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",                         显示 firstName、age 的所有条目和类型
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
              "firstName",
              "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)        显示 firstName 和 age 列的所有记录，并对 age 记录添加1
      .show()                                      显示所有小于24岁的记录
>>> df.select(df['age'] > 24).show()
```

When

```
>>> df.select("firstName",                         显示 firstName，且大于30岁显示1，小于30岁显示0
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]          显示符合指定条件的 firstName 列的记录
            .collect()
```

Like

```
>>> df.select("firstName",                         显示 lastName 列中包含 Smith 的 firstName 列的记录
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName",                         显示 lastName 列中以 Sm 开头的 firstName 列的记录
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th"))\         显示以 th 结尾的 lastName
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \          返回 firstName 的子字符串
                          .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \            显示介于22岁至24岁之间的 age 列的记录
      .show()
```

添加、修改、删除列

添加列

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

修改列

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

删除列

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

分组

```
>>> df.groupBy("age")\                  按 age 列分组，统计每组人数
      .count() \
      .show()
```

筛选

```
>>> df.filter(df["age"]>24).show()      按 age 列筛选，保留年龄大于24岁的
```

排序

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

替换缺失值

```
>>> df.na.fill(50).show()               用一个值替换空值
>>> df.na.drop().show()                 去除 df 中为空值的行
>>> df.na \                             用一个值替换另一个值
      .replace(10, 20) \
      .show()
```

重分区

```
>>> df.repartition(10)\                 将 df 拆分为10个分区
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()   将 df 合并为1个分区
```

运行 SQL 查询

将数据框注册为视图

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

查询视图

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

输出

数据结构

```
>>> rdd1 = df.rdd                       将 df 转换为 RDD
>>> df.toJSON().first()                 将 df 转换为 RDD 字符串
>>> df.toPandas()                       将 df 的内容转为 Pandas 的数据框
```

保存至文件

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

终止 SparkSession

```
>>> spark.stop()
```

原文作者 DataCamp
Learn Python for Data Science Interactively

Python数据科学速查表

PySpark - RDD 基础

Spark

PySpark 是 Spark 的 Python API，允许 Python 调用 Spark 编程模型。

初始化 Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

核查 SparkContext

```
>>> sc.version                  获取 SparkContext 版本
>>> sc.pythonVer                获取 Python 版本
>>> sc.master                   要连接的 Master URL
>>> str(sc.sparkHome)           Spark 在工作节点的安装路径
>>> str(sc.sparkUser())         获取 SparkContext 的 Spark 用户名

>>> sc.appName                  返回应用名称
>>> sc.applicationId            获取应用程序ID
>>> sc.defaultParallelism       返回默认并行级别
>>> sc.defaultMinPartitions     RDD默认最小分区数
```

配置

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

使用 Shell

PySpark Shell 已经为 SparkContext 创建了名为 sc 的变量。

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

用 --master 参数设定 Context 连接到哪个 Master 服务器，通过传递逗号分隔列表至 --py-files 添加 Python.zip、.egg 或 .py文件到 Runtime 路径。

加载数据

并行集合

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

外部数据

使用 textFile() 函数从HDFS、本地文件或其它支持 Hadoop 的文件系统里读取文本文件，或使用 wholeTextFiles() 函数读取目录里文本文件。

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

提取 RDD 信息

基础信息

```
>>> rdd.getNumPartitions()          列出分区数
>>> rdd.count()                     计算 RDD 实例数量
 3
>>> rdd.countByKey()                按键计算 RDD 实例数量
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()              按值计算 RDD 实例数量
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()              以字典形式返回键值
 {'a': 2,'b': 2}
>>> rdd3.sum()                      汇总 RDD 元素
 4950
>>> sc.parallelize([]).isEmpty()    检查 RDD 是否为空
 True
```

汇总

```
>>> rdd3.max()                      RDD 元素的最大值
  99
>>> rdd3.min()                      RDD 元素的最小值
  0
>>> rdd3.mean()                     RDD 元素的平均值
  49.5
>>> rdd3.stdev()                    RDD 元素的标准差
  28.866070047722118
>>> rdd3.variance()                 计算 RDD 元素的方差
  833.25
>>> rdd3.histogram(3)               分箱（Bin）生成直方图
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                    综合统计
                                    包括：计数、平均值、标准差、最大值和最小值
```

应用函数

```
>>> rdd.map(lambda x: x+(x[1],x[0]))          对每个 RDD 元素执行函数
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))  对每个 RDD 元素执行函数，并拉平结果

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)           不改变键，对 rdd4 的每个键值对执行 flatMap 函数
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

选择数据

获取

```
>>> rdd.collect()                   返回包含所有 RDD 元素的列表
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                     提取前两个 RDD 元素
  [('a', 7), ('a', 2)]
>>> rdd.first()                     提取第一个 RDD 元素
  ('a', 7)
>>> rdd.top(2)                      提取前两个 RDD 元素
  [('b', 2), ('a', 7)]
```

抽样

```
>>> rdd3.sample(False, 0.15, 81).collect()   返回 rdd3 的采样子集
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

筛选

```
>>> rdd.filter(lambda x: "a" in x)          筛选 RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()               返回 RDD 里的唯一值
  ['a',2,'b',7]
>>> rdd.keys().collect()                    返回 RDD 键值对里的键
  ['a', 'a', 'b']
```

迭代

```
>>> def g(x): print(x)
>>> rdd.foreach(g)                  为所有RDD应用函数
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

改变数据形状

规约

```
>>> rdd.reduceByKey(lambda x,y : x+y)       合并每个键的 RDD 值
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)          合并 RDD 的值
  ('a',7,'a',2,'b',2)
```

分组

```
>>> rdd3.groupBy(lambda x: x % 2)           返回 RDD 的分组值
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()                        按键分组 RDD
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

聚合

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)      汇总每个分区里的 RDD 元素，并输出结果
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)  汇总每个 RDD 的键的值
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)                        汇总每个分区里的 RDD 元素，并输出结果
  4950
>>> rdd.foldByKey(0, add)                   合并每个键的值
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)               通过执行函数，创建 RDD 元素的元组
       .collect()
```

数学运算

```
>>> rdd.subtract(rdd2)                      返回在 rdd2 里没有匹配键的 rdd 键值对
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)                 返回 rdd2 里的每个（键，值）对，rdd中没有匹配的键
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()           返回 rdd 和 rdd2 的笛卡尔积
```

排序

```
>>> rdd2.sortBy(lambda x: x[1])             按给定函数排序 RDD
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()                        按键排序 RDD的键值对
       .collect()
  [('a',2),('b',1),('d',1)]
```

重分区

```
>>> rdd.repartition(4)              新建一个含4个分区的 RDD
>>> rdd.coalesce(1)                 将 RDD 中的分区数缩减为1个
```

保存

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

终止 SparkContext

```
>>> sc.stop()
```

执行程序

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

原文作者 DataCamp
Learn Python for Data Science Interactively