



# Python\_Libraries\_for\_Data\_Analytics

Matplotlib, Numpy, Pandas by Haley Kwok

- [Python\\_Libraries\\_for\\_Data\\_Analytics](#)
- [Matplotlib](#)
  - [Matplotlib Pyplot](#)
    - [plot\(\)](#)
  - [Marker 标记](#)
  - [linestyle 自定义线的样式, 包括线的类型、颜色和大小等](#)
  - [label 轴标签和标题](#)
  - [grid 网格线](#)
  - [legend](#)
  - [subplot\(\) 绘制多图](#)
  - [scatter 散点图](#)
    - [颜色条 Colormap](#)
  - [barchart 柱形图](#)
    - [Horizontal Bars](#)
    - [Multiple Bars](#)
  - [Pie 饼图](#)
  - [histograms 直方图](#)
    - [Multiple](#)
    - [Horizontal](#)
- [Numpy](#)
  - [NumPy 数据类型](#)
  - [NumPy 数组属性](#)
    - [ndarray.ndim](#)
    - [ndarray.shape](#)
  - [创建数组](#)
  - [从已有的数组创建数组](#)
    - [numpy.asarray](#)
  - [从数值范围创建数组](#)
  - [切片和索引](#)
    - [一位数组](#)
    - [多维数组](#)

- 高级索引
  - 整数数组索引
  - 布尔索引
- 广播(Broadcast)
- Pandas
  - Series
  - DataFrame
  - 读写数据
    - CSV 文件
      - skiprows/ header = 1
      - names
      - nrows
      - na\_values
    - Excel
    - JSON
    - Dictionary
    - Tuples list
  - 数据清洗
    - dropna()
    - isnull()
    - fillna()
    - mean() for single column
    - mean() for multiple columns
    - interpolate() 中和数值
    - replace()
    - split()
    - 利用数据的均值、中位数值或众数
    - 清洗格式错误数据
    - 清洗错误数据
    - 清洗重复数据
  - 数据处理columns 列
    - df.columns
    - df
    - df
    - set\_index

- `reset_index`
  - `loc()` by label
  - `iloc()` by position
  - `.columns`
- 数据处理 rows 行
  - `head()`
  - `tail()`
  - `info()`
  - `describe()`
- feature engineering 特征工程
  - `replace`
  - `mapping`
  - `apply`
  - `applymap`
  - `get_dummies` 1010
  - `LabelEncoder` 1010 and `OneHotEncoder`
  - `split()`
  - operator: `~`
- `Groupby`
- Concatenation 串联
  - Concatenation And Keys
  - Concatenation Using Index
  - Concatenate dataframe with series
- Merge Using a Dataframe Column 使用数据框架列进行合并
  - Joins
    - `inner`
    - `outer`
    - `left`
    - `right`
    - `indicator flags`
    - `suffix`
- Pivot Table 改造和重塑
- `Melt`
- `stack`
- `crosstab`

- [margins](#)
- [multi index columns and rows](#)
- [normalize](#)
- [aggfunc and values](#)
- [数据操作](#)
  - [内置函数 max\(\), min\(\), std\(\)](#)
  - [操作符 > < =](#)
- [Summary](#)
- [Reference](#)

# Matplotlib

## Matplotlib Pyplot

使用 import 导入 pyplot 库，并设置一个别名 plt:

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

### plot()

syntax:

```
# 画单条线
plot([x], y, [fmt], *, data=None, **kwargs)
# 画多条线
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

参数说明：

x, y：点或线的节点，x 为 x 轴数据，y 为 y 轴数据，数据可以列表或数组。

fmt：可选，定义基本格式（如颜色、标记和线条样式）。

\*\*kwargs：可选，用在二维平面图上，设置指定属性，如标签，线的宽度等。

颜色字符：'b' 蓝色，'m' 洋红色，'g' 绿色，'y' 黄色，'r' 红色，'k' 黑色，'w' 白色，'c' 青绿色，'#008000' RGB 颜色字符串。多条曲线不指定颜色时，会自动选择不同颜色。

线型参数：'-' 实线，'--' 破折线，'-. ' 点划线，'...' 虚线。

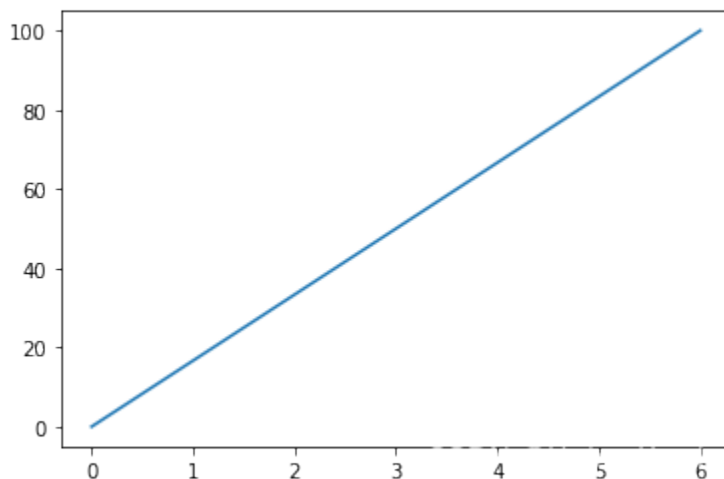
标记字符：'.' 点标记, ',' 像素标记(极小点), 'o' 实心圈标记, 'v' 倒三角标记, '^' 上三角标记, '>' 右三角标记, '<' 左三角标记...等等。

画单条线:

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

xpoints = np.array([0, 6])
ypoints = np.array([0, 100])

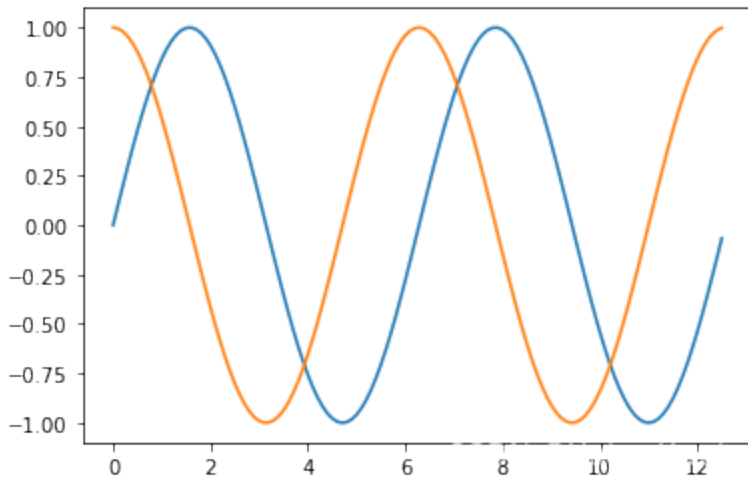
plt.plot(xpoints, ypoints)
plt.show()
```



画多条线:

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

x = np.arange(0, 4*np.pi, 0.1) # start, stop, step
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y, x, z)
plt.show()
```



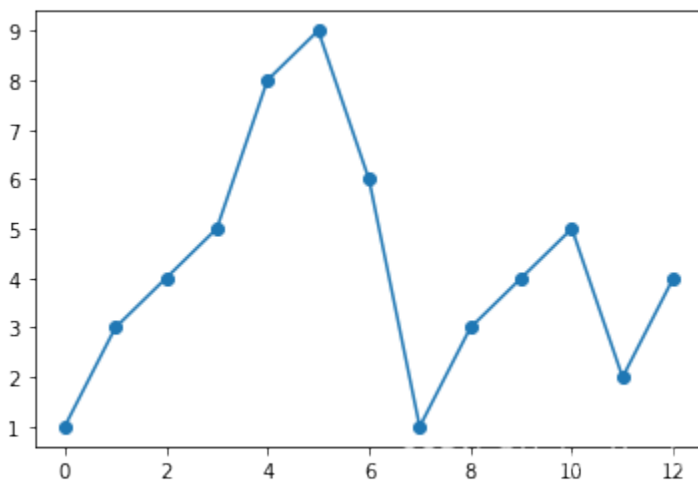
## Marker 标记

坐标自定义一些不一样的标记，可以使用 `plot()` 方法的 `marker` 参数来定义：

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

ypoints = np.array([1,3,4,5,8,9,6,1,3,4,5,2,4])

plt.plot(ypoints, marker = 'o')
plt.show()
```

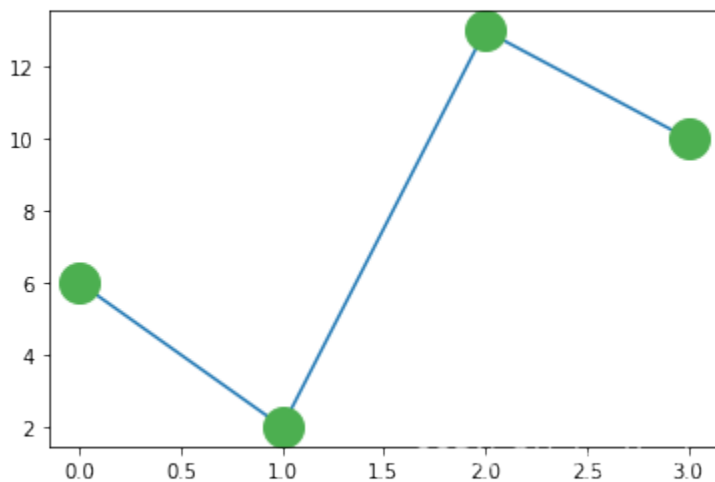


标记大小与颜色:

markersize, 简写为 ms：定义标记的大小。  
markerfacecolor, 简写为 mfc：定义标记内部的颜色。  
markeredgecolor, 简写为 mec：定义标记边框的颜色。

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

ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
plt.show()
```



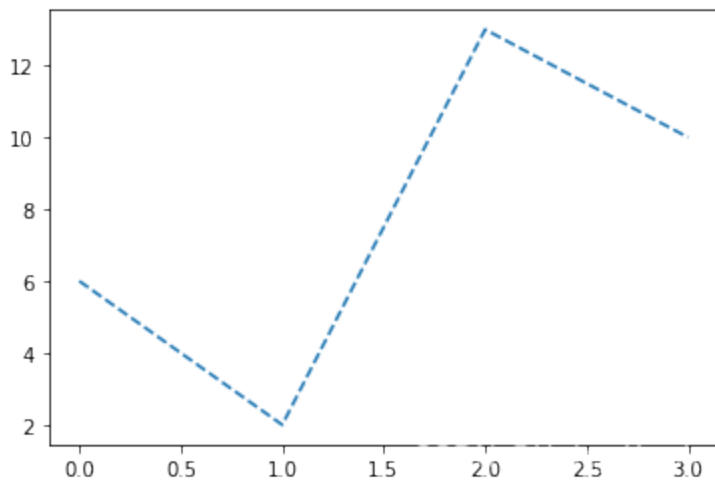
## linestyle 自定义线的样式，包括线的类型、颜色和大小等

线的类型可以使用 linestyle 参数来定义，简写为 ls:

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

ypoints = np.array([6, 2, 13, 10])

plt.plot(ypoints, linestyle = 'dashed')
plt.show()
```

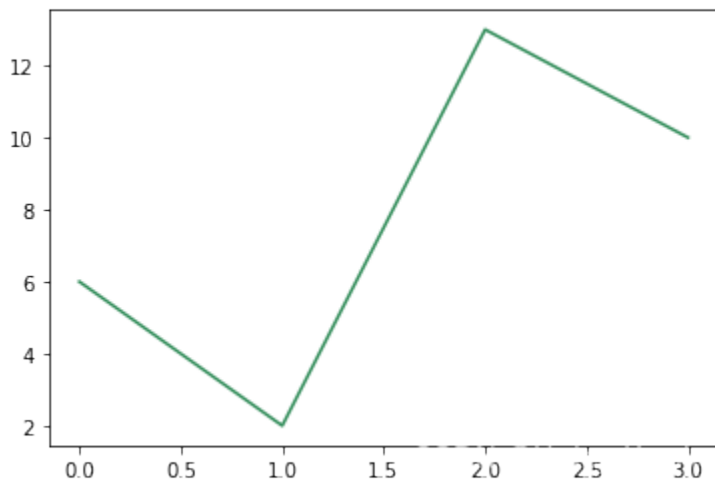


线的颜色可以使用 color 参数来定义，简写为 c:

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

ypoints = np.array([6, 2, 13, 10])

plt.plot(ypoints, c = 'SeaGreen')
plt.show()
```



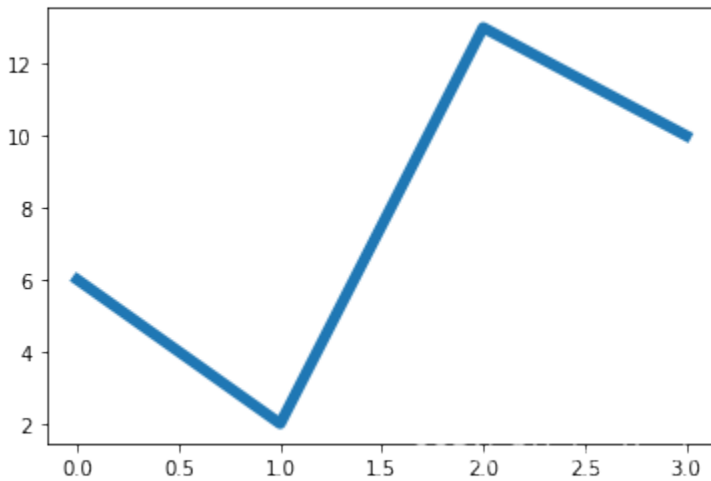
线的宽度可以使用 linewidth 参数来定义，简写为 lw，值可以是浮点数



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

ypoints = np.array([6, 2, 13, 10])

plt.plot(ypoints, linewidth = '5.5')
plt.show()
```



## label 轴标签和标题

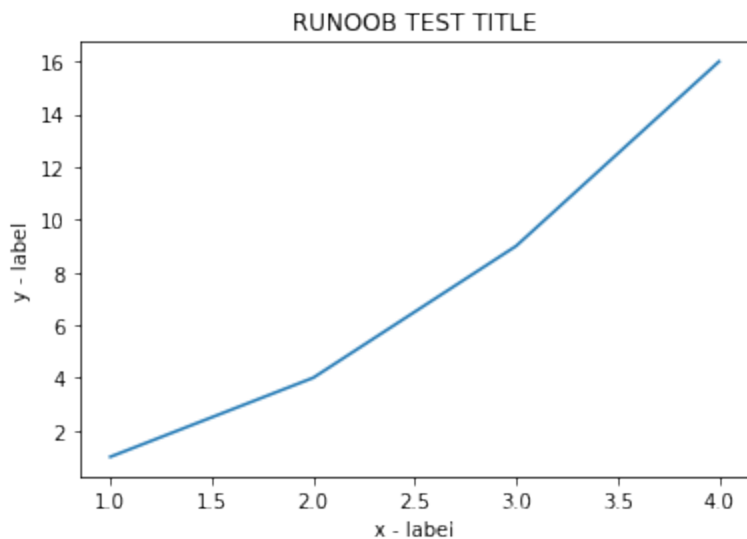
使用 xlabel() 和 ylabel() 方法来设置 x 轴和 y 轴的标签  
使用 title() 方法来设置标题

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

x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
plt.plot(x, y)

plt.title("RUNOOB TEST TITLE")
plt.xlabel("x - label")
plt.ylabel("y - label")

plt.show()
```



## grid 网格线

使用 pyplot 中的 grid() 方法来设置图表中的网格线  
syntax:

```
matplotlib.pyplot.grid(b=None, which='major', axis='both', )
```

参数说明：

b：可选，默认为 None，可以设置布尔值，true 为显示网格线，false 为不显示，如果设置 \*\*kwargs 参数，则值为 true。

which：可选，可选值有 'major'、'minor' 和 'both'，默认为 'major'，表示应用更改的网格线。

axis：可选，设置显示哪个方向的网格线，可以是取 'both'（默认），'x' 或 'y'，分别表示两个方向，x 轴方向或 y 轴方向。

\*\*kwargs：可选，设置网格样式，可以是 color='r', linestyle='-' 和 linewidth=2，分别表示网格线的颜色，样式和宽度。

```

import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])

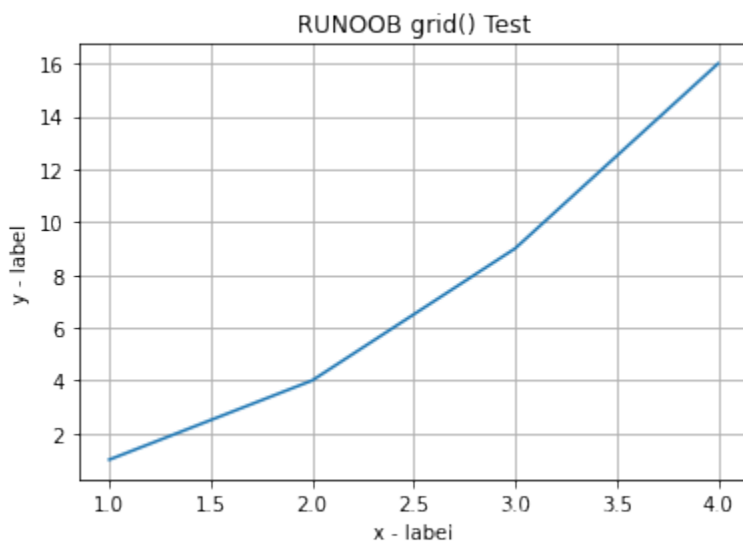
plt.title("RUNOOB grid() Test")
plt.xlabel("x - label")
plt.ylabel("y - label")

plt.plot(x, y)

plt.grid()
# plt.grid(color = 'r', linestyle = '--', linewidth = 0.5) 做装饰

plt.show()

```



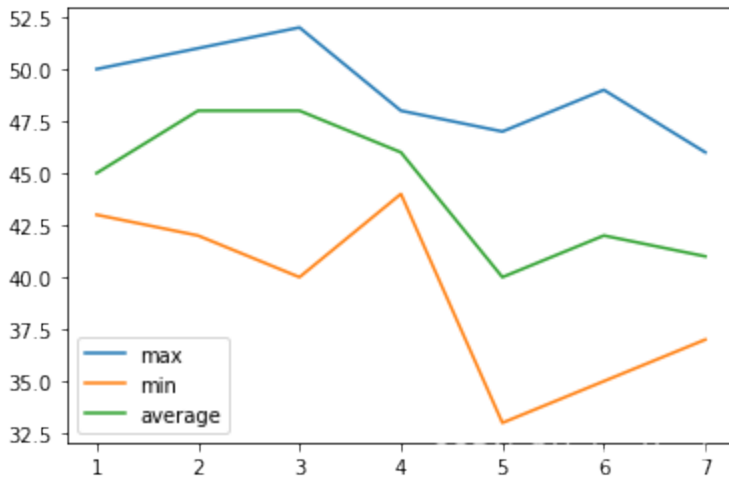
## legend

```

# Show legend
plt.plot(days, max_t, label="max")
plt.plot(days, min_t, label="min")
plt.plot(days, avg_t, label="average")

plt.legend(loc='best')
# plt.legend(loc='upper right', shadow=True, fontsize='large')
# plt.legend(loc='upper right', fontsize="large", shadow=True)

```



## subplot() 绘制多图

使用 pyplot 中的 subplot() 和 subplots() 方法来绘制多个子图; subplot() 方法在绘图时需要指定位置, subplots() 方法可以一次生成多个, 在调用时只需要调用生成对象的 ax 即可

syntax:

```
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(**kwargs)
subplot(ax)
```

```
matplotlib.pyplot.subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True,
```

```

import matplotlib.pyplot as plt
import numpy as np

#plot 1:
xpoints = np.array([0, 6])
ypoints = np.array([0, 100])

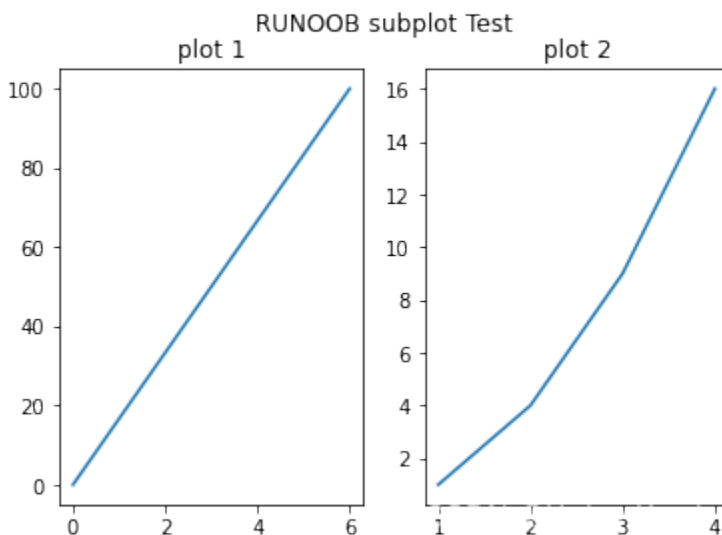
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot
plt.plot(xpoints,ypoints)
plt.title("plot 1")

#plot 2:
x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])

plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd subplot
plt.plot(x,y)
plt.title("plot 2")

plt.suptitle("RUNOOB subplot Test")
plt.show()

```



## scatter 散点图

使用 pyplot 中的 scatter() 方法来绘制散点图:

```

matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None,
vmin=None, vmax=None, alpha=None, linewidths=None, *, edgecolors=None,

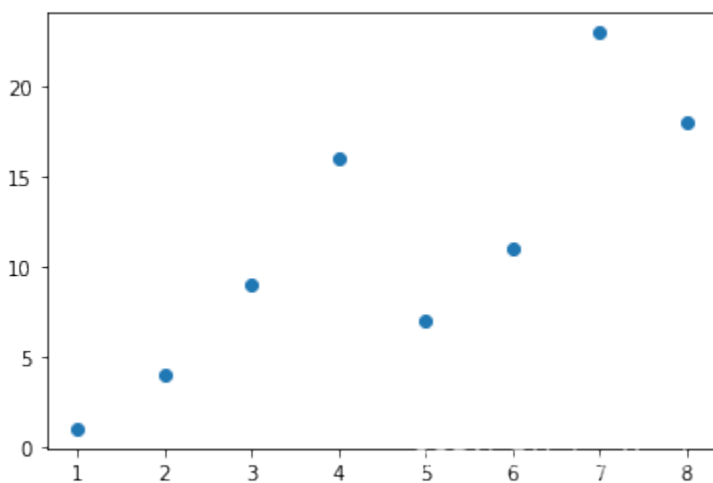
```

```
plotnonfinite=False, data=None, **kwargs)
```

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

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 9, 16, 7, 11, 23, 18])
# sizes = np.array([20,50,100,200,500,1000,60,90]) 设置图标大小
# colors = np.array(["red","green","black","orange","purple","beige","cyan","magenta"]) 自

plt.scatter(x, y)
plt.show()
```



## 颜色条 Colormap

使用 plt.colorbar() 方法：

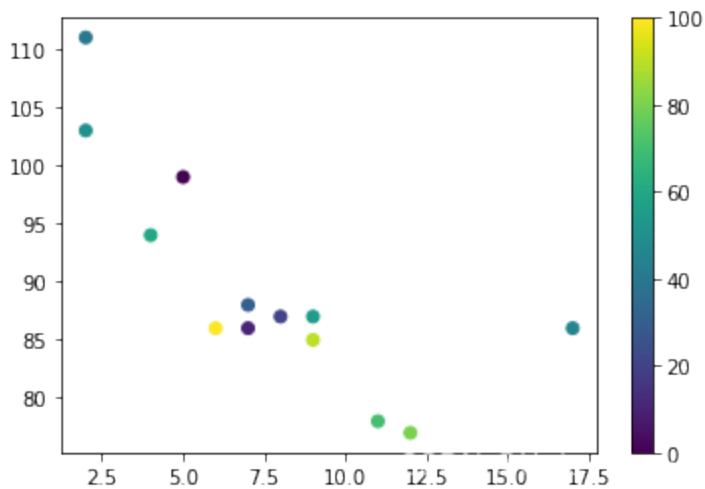
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis') # 换个颜色条参数, cmap 设置为 afmhot_r

plt.colorbar()

plt.show()
```



## barchart 柱形图

使用 pyplot 中的 `bar()` 方法来绘制柱形图:

syntax:

```
matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
```

参数说明：

`x`：浮点型数组，柱形图的 `x` 轴数据。

`height`：浮点型数组，柱形图的高度。

`width`：浮点型数组，柱形图的宽度。

`bottom`：浮点型数组，底座的 `y` 坐标，默认 0。

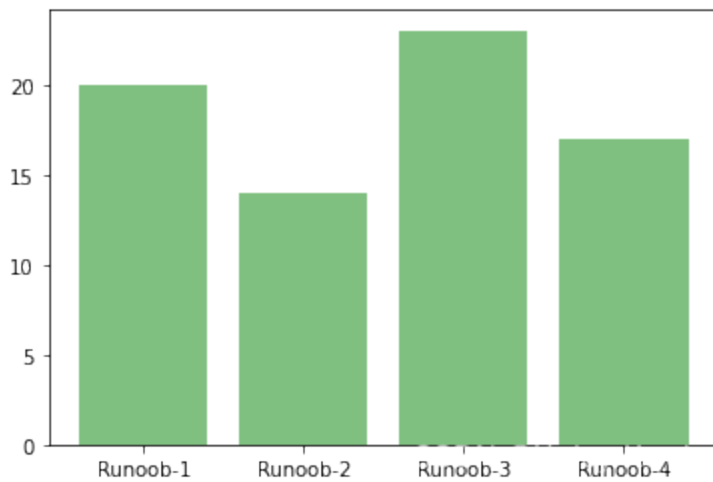
`align`：柱形图与 `x` 坐标的对齐方式，'center' 以 `x` 位置为中心，这是默认值。'edge'：将柱形图的左边缘与 `x` 位置对齐。要对齐右边缘的条形，可以传递负数的宽度值及 `align='edge'`。

`**kwargs`：其他参数。

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

x = np.array(['Runoob-1', 'Runoob-2', 'Runoob-3', 'Runoob-4'])
y = np.array([20, 14, 23, 17])

plt.bar(x, y, align='center', color='green', alpha=0.5)
plt.show()
```



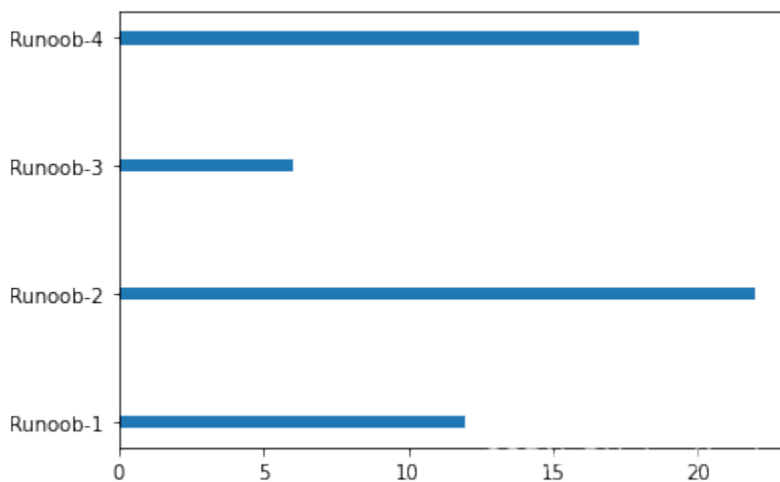
## Horizontal Bars

垂直方向的柱形图可以使用 `barh()` 方法来设置:

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

x = np.array(["Runoob-1", "Runoob-2", "Runoob-3", "Runoob-4"])
y = np.array([12, 22, 6, 18])

plt.barh(x, y, height = 0.1)
plt.show()
```

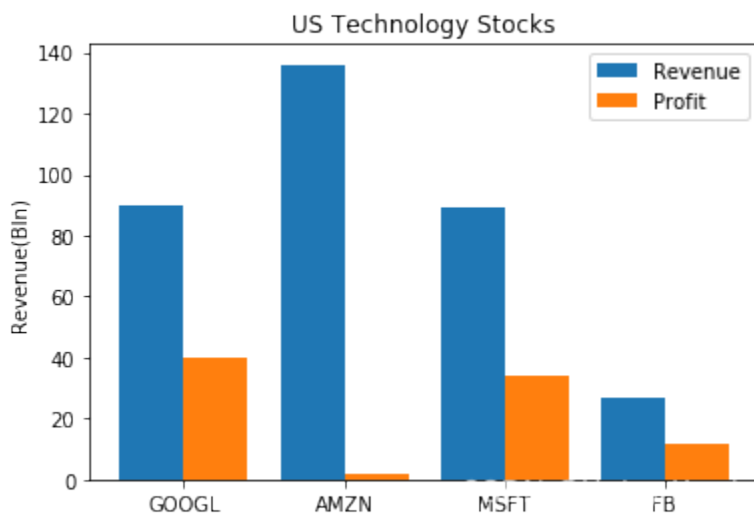




## Multiple Bars

```
plt.bar(xpos-0.2,revenue, width=0.4, label="Revenue")
plt.bar(xpos+0.2,profit, width=0.4,label="Profit")

plt.xticks(xpos,company)
plt.ylabel("Revenue(Bln)")
plt.title('US Technology Stocks')
plt.legend()
```



## Pie 饼图

使用 pyplot 中的 pie() 方法来绘制饼图:

syntax:

```
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6)
```

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

y = np.array([35, 25, 25, 15])

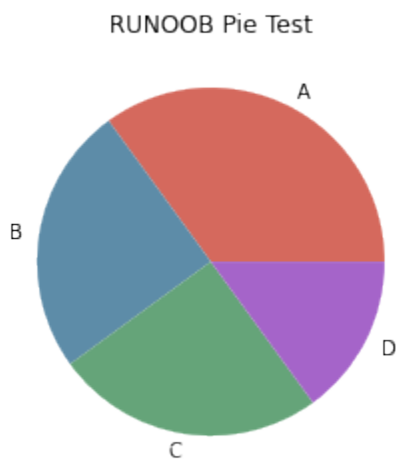
plt.pie(y)
plt.show()
```



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

y = np.array([35, 25, 25, 15])

plt.pie(y,
        labels=['A', 'B', 'C', 'D'], # 设置饼图标签
        colors=["#d5695d", "#5d8ca8", "#65a479", "#a564c9"], # 设置饼图颜色
        )
plt.title("RUNOOB Pie Test") # 设置标题
plt.show()
```

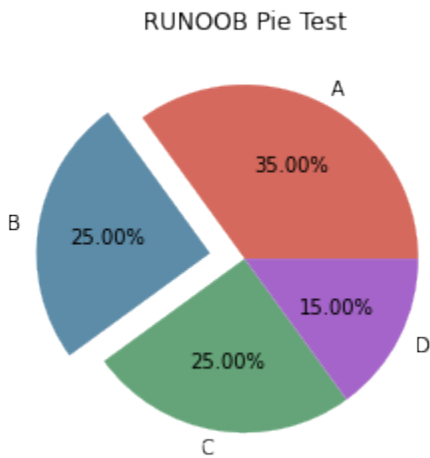


突出显示第二个扇形，并格式化输出百分比：

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

y = np.array([35, 25, 25, 15])

plt.pie(y,
        labels=['A', 'B', 'C', 'D'],
        colors=["#d5695d", "#5d8ca8", "#65a479", "#a564c9"],
        explode=(0, 0.2, 0, 0), # 第二部分突出显示, 值越大, 距离中心越远
        autopct='%.2f%%', # 输出百分比
        )
plt.title("RUNOOB Pie Test")
plt.show()
```

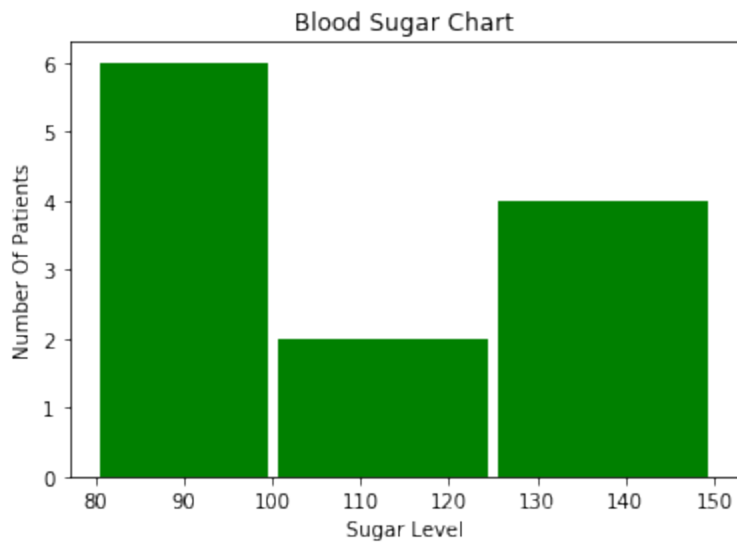


## histograms 直方图

在直方图中, X轴包含一个变量, Y轴将是该变量的频率:

```
plt.xlabel("Sugar Level")
plt.ylabel("Number Of Patients")
plt.title("Blood Sugar Chart")

plt.hist(blood_sugar, bins=[80,100,125,150], rwidth=0.95, color='g')
```

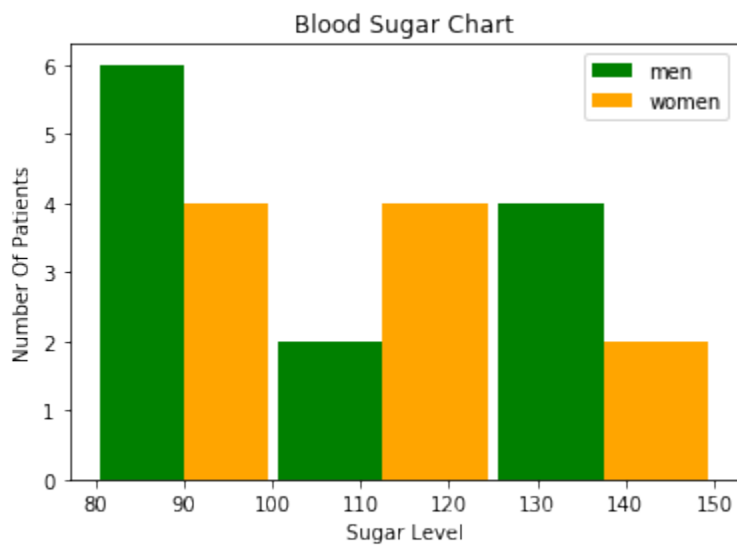


## Multiple

```
plt.xlabel("Sugar Level")
plt.ylabel("Number Of Patients")
plt.title("Blood Sugar Chart")

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([blood_sugar_men, blood_sugar_women], bins=[80, 100, 125, 150], rwidth=0.95, color=['men', 'women'])
plt.legend() # 右上角的标签
```

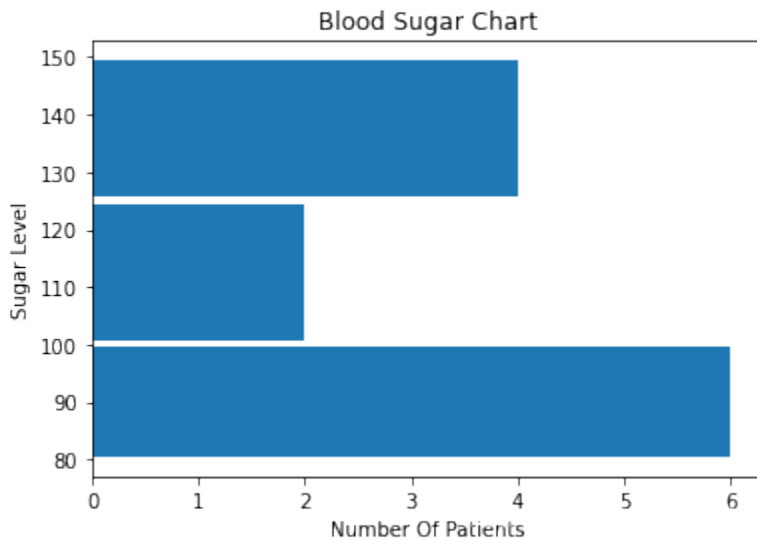


## Horizontal

横向, orientation='horizontal' :

```
plt.xlabel("Number Of Patients")
plt.ylabel("Sugar Level")
plt.title("Blood Sugar Chart")

plt.hist(blood_sugar, bins=[80,100,125,150], rwidth=0.95, orientation='horizontal') # 横向
```



## Numpy

NumPy 最重要的一个特点是其 N 维数组对象 ndarray，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。

## NumPy 数据类型

numpy 支持的数据类型比 Python 内置的类型要多很多，基本上可以和 C 语言的数据类型对应上，其中部分类型对应为 Python 内置的类型。比如bool, int, int16, int32, float16, float32, float64, complex。

# NumPy 数组属性

NumPy 数组的维数称为秩（rank），秩就是轴的数量，即数组的维度，一维数组的秩为 1，二维数组的秩为 2，以此类推。

每一个线性的数组称为是一个轴（axis），也就是维度（dimensions）。比如说，二维数组相当于是两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。

声明 axis：axis=0，表示沿着第 0 轴进行操作，即对每一列进行操作；axis=1，表示沿着第 1 轴进行操作，即对每一行进行操作。

## ndarray.ndim

ndarray.ndim 用于返回数组的维数，等于rank

```
import numpy as np

a = np.arange(24)
print (a.ndim) # one d

b = a.reshape(2,4,3) # three d
print (b.ndim)

# return
1
3
```

## ndarray.shape

ndarray.shape 表示数组的维度，返回一个元组，这个元组的长度就是维度的数目，即 ndim 属性(秩)。比如，一个二维数组，其维度表示"行数"和"列数"。

ndarray.shape 也可以用于调整数组大小。

```
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
a.shape = (3,2)
print (a)
```

```
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print (b)

# return
[[1, 2]
 [3, 4]
 [5, 6]]
```

## 创建数组

`numpy.empty/ones/zeros(shape, dtype = float, order = 'C')`

```
import numpy as np

# 默认为浮点数
x = np.zeros(5)
print(x)

# 设置类型为整数
y = np.zeros((5,), dtype = int)
print(y)

# 自定义类型
z = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print(z)

# return
[0. 0. 0. 0. 0.]
[0 0 0 0 0]
[[ (0, 0) (0, 0) ]
 [ (0, 0) (0, 0) ]]
```

## 从已有的数组创建数组

### **numpy.asarray**

`numpy.asarray` 类似 `numpy.array`，但 `numpy.asarray` 参数只有三个，比 `numpy.array` 少两个。

```
numpy.asarray(a, dtype = None, order = None)
```

参数说明：

a 任意形式的输入参数，可以是，列表, 列表的元组, 元组, 元组的元组, 元组的列表, 多维数组

dtype 数据类型，可选

order 可选，有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

```
# 将列表转换为 ndarray:
```

```
import numpy as np
```

```
x = [1,2,3]
```

```
a = np.asarray(x)
```

```
print (a)
```

```
# return
```

```
[1 2 3]
```

```
# 将元组转换为 ndarray:
```

```
import numpy as np
```

```
x = (1,2,3)
```

```
a = np.asarray(x)
```

```
print (a)
```

```
# return
```

```
[1 2 3]
```

## 从数值范围创建数组

根据 start 与 stop 指定的范围以及 step 设定的步长，生成一个 ndarray:

```
numpy.arange(start, stop, step, dtype)
```

```
import numpy as np
```

```
x = np.arange(10,20,2)
```

```
print (x)
```

```
# return
```

```
[10 12 14 16 18]
```



# 切片和索引

## 一位数组

与python一样

```
import numpy as np

a = np.arange(10)
s = slice(2,7,2)    # 从索引 2 开始到索引 7 停止，间隔为2
print (a[s])

import numpy as np

a = np.arange(10)
b = a[2:7:2]    # 从索引 2 开始到索引 7 停止，间隔为 2
print(b)

# return
[2 4 6]
[2 4 6]
```

## 多维数组

```
import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
# 从某个索引处开始切割
print('从数组索引 a[1:] 处开始切割')
print(a[1:])

# return
[[1 2 3]
 [3 4 5]
 [4 5 6]]
从数组索引 a[1:] 处开始切割
[[3 4 5]
 [4 5 6]]
```

切片还可以包括省略号 ...，来使选择元组的长度与数组的维度相同。如果在行位置使用省略号，它将返回包含行中元素的 ndarray。

## 行列

```
import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print (a[... ,1])    # 第2列元素
print (a[1,...])     # 第2行元素
print (a[... ,1:])    # 第2列及剩下的所有元素

# return
[2 4 5]
[3 4 5]
[[2 3]
 [4 5]
 [5 6]]
```

## 高级索引

### 整数数组索引

以下实例获取数组中(0,0)，(1,1)和(2,0)位置处的元素。

```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print (y)

# return
[1 4 5]
```

获取 4X3 数组中的四个角的元素。行索引是 [0,0] 和 [3,3]，而列索引是 [0,2] 和 [0,2]。

```

import numpy as np

x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
print ('我们的数组是：' )
print (x)
print ('\n')
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]
print ('这个数组的四个角元素是：')
print (y)

# return
我们的数组是：
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

这个数组的四个角元素是：
[[ 0  2]
 [ 9 11]]

```

借助切片：或 ... 与索引数组组合：

```

import numpy as np

a = np.array([[1,2,3], [4,5,6], [7,8,9]])
b = a[1:3, 1:3]
c = a[1:3, [1,2]]
d = a[... ,1:]
print(b)
print(c)
print(d)

# return
[[5 6]
 [8 9]]
[[5 6]
 [8 9]]
[[2 3]
 [5 6]
 [8 9]]

```

# 布尔索引

布尔索引通过布尔运算（如：比较运算符）来获取符合指定条件的元素的数组  
获取大于 5 的元素：

```
import numpy as np

x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
print ('我们的数组是：')
print (x)
print ('\n')
# 现在我们会打印出大于 5 的元素
print ('大于 5 的元素是：')
print (x[x > 5])

# return
我们的数组是：
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

大于 5 的元素是：
[ 6  7  8  9 10 11]
```

# 广播(Broadcast)

如果两个数组 a 和 b 形状相同，即满足  $a.shape == b.shape$ ，那么  $a*b$  的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同。

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print (c)

# return
[ 10  40  90 160]
```

当运算中的 2 个数组的形状不同时，numpy 将自动触发广播机制。如：

```
import numpy as np

a = np.array([[ 0, 0, 0],
              [10,10,10],
              [20,20,20],
              [30,30,30]])
b = np.array([0,1,2])
print(a + b)

# return
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

---

# Pandas

Pandas 是 Python 语言的一个扩展程序库，用于数据分析；名字衍生自术语 "panel data"（面板数据）和 "Python data analysis"（Python 数据分析）。

## Series

主要数据结构是 Series（一维数据）与 DataFrame（二维数据）：Series 是一种类似于一维数组的对象，它由一组数据（各种Numpy数据类型）以及一组与之相关的数据标签（即索引）组成

syntax:

```
pandas.Series( data, index, dtype, name, copy)
```

data：一组数据(ndarray 类型)。

index：数据索引标签，如果不指定，默认从 0 开始。

dtype：数据类型，默认会自己判断。

name：设置名称。

copy：拷贝数据，默认为 False。

```
import pandas as pd

a = ["Google", "Runoob", "Wiki"]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)

# return
x    Google
y    Runoob
z     Wiki
```

```
import pandas as pd

sites = {1: "Google", 2: "Runoob", 3: "Wiki"}

myvar = pd.Series(sites)

print(myvar)

# return
1    Google
2    Runoob
3     Wiki
```

## DataFrame

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。DataFrame 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共用一个索引）

syntax:

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

data：一组数据(ndarray、series, map, lists, dict 等类型)。  
index：索引值，或者可以称为行标签。  
columns：列标签，默认为 RangeIndex (0, 1, 2, ..., n)。  
dtype：数据类型。

copy：拷贝数据，默认为 False。

```
import pandas as pd

data = [['Google',10],['Runoob',12],['Wiki',13]]

df = pd.DataFrame(data,columns=['Site','Age'],dtype=float)

print(df)
```

或者：

```
import pandas as pd

data = {'Site':['Google', 'Runoob', 'Wiki'], 'Age':[10, 12, 13]}

df = pd.DataFrame(data)

print (df)

# return
   Site  Age
0  Google  10.0
1  Runoob  12.0
2   Wiki  13.0
```

---

## 读写数据

### CSV 文件

CSV（Comma-Separated Values，逗号分隔值，有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。

```
import pandas as pd
df = pd.read_csv('nba.csv')
print(df)
```

## skiprows/ header = 1

```
df = pd.read_csv("stock_data.csv", skiprows=1) # skip first line
```

或者

```
df = pd.read_csv("stock_data.csv", header=1)
```

## names

```
df = pd.read_csv("stock_data.csv", header=None, names = ["ticker","eps","revenue","people",
```

```
# return
```

|         | ticker        | eps     | revenue | price  | people        |
|---------|---------------|---------|---------|--------|---------------|
| tickers | eps           | revenue | price   | people |               |
| GOOGL   | 27.82         | 87      | 845     |        | larry page    |
| WMT     | 4.61          | 484     | 65      | n.a.   |               |
| MSFT    | -1            | 85      | 64      |        | bill gates    |
| RIL     | not available |         | 50      | 1023   | mukesh ambani |
| TATA    | 5.6           | -1      | n.a.    |        | ratan tata    |

## nrows

```
df = pd.read_csv("stock_data.csv", nrows=2)
```

```
# return
```

|   | tickers | eps   | revenue | price | people     |
|---|---------|-------|---------|-------|------------|
| 0 | GOOGL   | 27.82 | 87      | 845   | larry page |
| 1 | WMT     | 4.61  | 484     | 65    | n.a.       |



## na\_values

```
df = pd.read_csv("stock_data.csv", na_values=["n.a.", "not available"]) # n.a. replaced by
```

```
# return
```

|   | tickers | eps   | revenue | price  | people        |
|---|---------|-------|---------|--------|---------------|
| 0 | GOOGL   | 27.82 | 87      | 845.0  | larry page    |
| 1 | WMT     | 4.61  | 484     | 65.0   | NaN           |
| 2 | MSFT    | -1.00 | 85      | 64.0   | bill gates    |
| 3 | RIL     | NaN   | 50      | 1023.0 | mukesh ambani |
| 4 | TATA    | 5.60  | -1      | NaN    | ratan tata    |

```
df = pd.read_csv("stock_data.csv", na_values={
    'eps': ['not available'],
    'revenue': [-1],
    'people': ['not available', 'n.a.'] # not available replaced by n.a.
})
```

```
# return
```

|   | tickers | eps   | revenue | price | people        |
|---|---------|-------|---------|-------|---------------|
| 0 | GOOGL   | 27.82 | 87.0    | 845   | larry page    |
| 1 | WMT     | 4.61  | 484.0   | 65    | NaN           |
| 2 | MSFT    | -1.00 | 85.0    | 64    | bill gates    |
| 3 | RIL     | NaN   | 50.0    | 1023  | mukesh ambani |
| 4 | TATA    | 5.60  | NaN     | n.a.  | ratan tata    |

## Excel

```
df=pd.read_excel('')
```

```

# want to fill the na with specific data
def convert_people_cell(cell):
    if cell=="n.a.":
        return 'Sam Walton'
    return cell

def convert_price_cell(cell):
    if cell=="n.a.":
        return 50
    return cell

df = pd.read_excel("stock_data.xlsx","Sheet1", converters= {
    'people': convert_people_cell,
    'price': convert_price_cell
})
df

```

## JSON

JSON (JavaScript Object Notation, JavaScript 对象表示法)，是存储和交换文本信息的语法，类似 XML。

## Dictionary

```

import pandas as pd
weather_data = {
    'day': ['1/1/2017', '1/2/2017', '1/3/2017'],
    'event': ['Rain', 'Sunny', 'Snow'],
    'temperature': [32, 35, 28],
    'windspeed': [6, 7, 2],
}
df = pd.DataFrame(weather_data)
df

# return

```

| day | event    | temperature | windspeed |   |
|-----|----------|-------------|-----------|---|
| 0   | 1/1/2017 | Rain        | 32        | 6 |
| 1   | 1/2/2017 | Sunny       | 35        | 7 |
| 2   | 1/3/2017 | Snow        | 28        | 2 |

```

weather_data = [
    {'day': '1/1/2017', 'temperature': 32, 'windspeed': 6, 'event': 'Rain'},
    {'day': '1/2/2017', 'temperature': 35, 'windspeed': 7, 'event': 'Sunny'},
    {'day': '1/3/2017', 'temperature': 28, 'windspeed': 2, 'event': 'Snow'},
]
df = pd.DataFrame(data=weather_data, columns=['day', 'temperature', 'windspeed', 'event'])
df

# return

```

|   | day      | temperature | windspeed | event |
|---|----------|-------------|-----------|-------|
| 0 | 1/1/2017 | 32          | 6         | Rain  |
| 1 | 1/2/2017 | 35          | 7         | Sunny |
| 2 | 1/3/2017 | 28          | 2         | Snow  |

## Tuples list

```

weather_data = [
    ('1/1/2017', 32, 6, 'Rain'),
    ('1/2/2017', 35, 7, 'Sunny'),
    ('1/3/2017', 28, 2, 'Snow')
]
df = pd.DataFrame(data=weather_data, columns=['day', 'temperature', 'windspeed', 'event'])
df

# return

```

|   | day      | temperature | windspeed | event |
|---|----------|-------------|-----------|-------|
| 0 | 1/1/2017 | 32          | 6         | Rain  |
| 1 | 1/2/2017 | 35          | 7         | Sunny |
| 2 | 1/3/2017 | 28          | 2         | Snow  |

## 数据清洗

很多数据集存在数据缺失、数据格式错误、错误数据或重复数据的情况，如果要对使数据分析更加准确，就需要对这些没有用的数据进行处理。

### dropna()

axis：默认为 0，表示逢空值剔除整行，如果设置参数 axis=1 表示逢空值去掉整列。

how：默认为 'any' 如果一行（或一列）里任何一个数据有出现 NA 就去掉整行，如果设置 how='all' 一行（或列）都是 NA 才去掉这整行。

thresh：设置需要多少非空值的数据才可以保留下来的。

subset：设置想要检查的列。如果是多个列，可以使用列名的 list 作为参数。

inplace：如果设置 True，将计算得到的值直接覆盖之前的值并返回 None，修改的是源数据。

```
DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

```
import pandas as pd

df = pd.read_csv('property-data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

设置需要多少非空值的数据才可以保留下来：

```
new_df = df.dropna(thresh=2) # 有两个或以上的value就留
new_df
```

```
# return
```

|            | temperature | windspeed | event  |
|------------|-------------|-----------|--------|
| day        |             |           |        |
| 2017-01-01 | 32.0        | 6.0       | Rain   |
| 2017-01-04 | NaN         | 9.0       | Sunny  |
| 2017-01-05 | 28.0        | NaN       | Snow   |
| 2017-01-07 | 32.0        | NaN       | Rain   |
| 2017-01-10 | 34.0        | 8.0       | Cloudy |
| 2017-01-11 | 40.0        | 12.0      | Sunny  |

移除指定列有空值的行：

```
import pandas as pd

df = pd.read_csv('property-data.csv')

df.dropna(subset=['ST_NUM'], inplace = True)

print(df.to_string())
```

## isnull()

可以通过 `isnull()` 判断各个单元格是否为空

```
import pandas as pd

df = pd.read_csv('property-data.csv')

print (df['NUM_BEDROOMS'])
print (df['NUM_BEDROOMS'].isnull())
```

指定空数据类型：

```
import pandas as pd

missing_values = ["n/a", "na", "--"]
df = pd.read_csv('property-data.csv', na_values = missing_values)

print (df['NUM_BEDROOMS'])
print (df['NUM_BEDROOMS'].isnull())
```

## fillna()

`fillna()` 方法替换一些空字段：

- `fillna()`

```
import pandas as pd

df = pd.read_csv('property-data.csv')

df.fillna(12345, inplace = True) # 使用 12345 替换空字段

print(df.to_string())
```

```
import pandas as pd
df = pd.read_csv("weather_data.csv", parse_dates=["day"])
df.set_index('day', inplace=True)
df

new_df = df.fillna(0)
new_df
```

```
# return
```

|            | temperature | windspeed | event  |
|------------|-------------|-----------|--------|
| day        |             |           |        |
| 2017-01-01 | 32.0        | 6.0       | Rain   |
| 2017-01-04 | 0.0         | 9.0       | Sunny  |
| 2017-01-05 | 28.0        | 0.0       | Snow   |
| 2017-01-06 | 0.0         | 7.0       | 0      |
| 2017-01-07 | 32.0        | 0.0       | Rain   |
| 2017-01-08 | 0.0         | 0.0       | Sunny  |
| 2017-01-09 | 0.0         | 0.0       | 0      |
| 2017-01-10 | 34.0        | 8.0       | Cloudy |
| 2017-01-11 | 40.0        | 12.0      | Sunny  |

指定某一个列来替换数据：

```
import pandas as pd

df = pd.read_csv('property-data.csv')

df['PID'].fillna(12345, inplace = True) # 使用 12345 替换 PID 为空数据

print(df.to_string())
```

```

new_df = df.fillna({
    'temperature': 0,
    'windspeed': 0,
    'event': 'no event'
})
new_df

# return

```

|            | temperature | windspeed | event    |
|------------|-------------|-----------|----------|
| day        |             |           |          |
| 2017-01-01 | 32.0        | 6.0       | Rain     |
| 2017-01-04 | 0.0         | 9.0       | Sunny    |
| 2017-01-05 | 28.0        | 0.0       | Snow     |
| 2017-01-06 | 0.0         | 7.0       | no event |
| 2017-01-07 | 32.0        | 0.0       | Rain     |
| 2017-01-08 | 0.0         | 0.0       | Sunny    |
| 2017-01-09 | 0.0         | 0.0       | no event |
| 2017-01-10 | 34.0        | 8.0       | Cloudy   |
| 2017-01-11 | 40.0        | 12.0      | Sunny    |

复制前或后数据 ffill / bfill :

- 加上limit = 1 只可copy一次
- 加上 axis = 'columns' 垂直 copy / 'index' 水平 copy

```

new_df = df.fillna(method="ffill")
new_df

# return

```

|            | temperature | windspeed | event  |
|------------|-------------|-----------|--------|
| day        |             |           |        |
| 2017-01-01 | 32.0        | 6.0       | Rain   |
| 2017-01-04 | 32.0        | 9.0       | Sunny  |
| 2017-01-05 | 28.0        | 9.0       | Snow   |
| 2017-01-06 | 28.0        | 7.0       | Snow   |
| 2017-01-07 | 32.0        | 7.0       | Rain   |
| 2017-01-08 | 32.0        | 7.0       | Sunny  |
| 2017-01-09 | 32.0        | NaN       | Sunny  |
| 2017-01-10 | 34.0        | 8.0       | Cloudy |
| 2017-01-11 | 40.0        | 12.0      | Sunny  |

## mean() for single column

```
import numpy as np
import pandas as pd

# A dictionary with list as values
GFG_dict = { 'G1': [10, 20, 30, 40],
              'G2': [25, np.NaN, np.NaN, 29],
              'G3': [15, 14, 17, 11],
              'G4': [21, 22, 23, 25]}

# Create a DataFrame from dictionary
gfg = pd.DataFrame(GFG_dict)

# Finding the mean of the column having NaN
mean_value = gfg['G2'].mean()

# Replace NaNs in column S2 with the
# mean of values in the same column
gfg['G2'].fillna(value=mean_value, inplace=True)
print('Updated Dataframe:')
print(gfg)
```

## mean() for multiple columns

@ Day1 ML Challenge

```
# from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer as Imputer
# old version: class sklearn.preprocessing.Imputer(missing_values='NaN', strategy='mean',

imputer = Imputer(missing_values=np.nan, strategy="mean")
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
X
```



## interpolate() 中和数值

```
new_df = df.interpolate(method="time")
new_df
```

|            | temperature | windspeed | event  |
|------------|-------------|-----------|--------|
| day        |             |           |        |
| 2017-01-01 | 32.0        | 6.0       | Rain   |
| 2017-01-04 | NaN         | 9.0       | Sunny  |
| 2017-01-05 | 28.0        | NaN       | Snow   |
| 2017-01-06 | NaN         | 7.0       | NaN    |
| 2017-01-07 | 32.0        | NaN       | Rain   |
| 2017-01-08 | NaN         | NaN       | Sunny  |
| 2017-01-09 | NaN         | NaN       | NaN    |
| 2017-01-10 | 34.0        | 8.0       | Cloudy |
| 2017-01-11 | 40.0        | 12.0      | Sunny  |

```
# return
```

|            | temperature | windspeed | event  |
|------------|-------------|-----------|--------|
| day        |             |           |        |
| 2017-01-01 | 32.000000   | 6.00      | Rain   |
| 2017-01-04 | 29.000000   | 9.00      | Sunny  |
| 2017-01-05 | 28.000000   | 8.00      | Snow   |
| 2017-01-06 | 30.000000   | 7.00      | NaN    |
| 2017-01-07 | 32.000000   | 7.25      | Rain   |
| 2017-01-08 | 32.666667   | 7.50      | Sunny  |
| 2017-01-09 | 33.333333   | 7.75      | NaN    |
| 2017-01-10 | 34.000000   | 8.00      | Cloudy |
| 2017-01-11 | 40.000000   | 12.00     | Sunny  |

## replace()

|   | day      | temperature | windspeed | event |
|---|----------|-------------|-----------|-------|
| 0 | 1/1/2017 | 32          | 6         | Rain  |
| 1 | 1/2/2017 | -99999      | 7         | Sunny |
| 2 | 1/3/2017 | 28          | -99999    | Snow  |
| 3 | 1/4/2017 | -99999      | 7         | 0     |
| 4 | 1/5/2017 | 32          | -99999    | Rain  |
| 5 | 1/6/2017 | 31          | 2         | Sunny |
| 6 | 1/6/2017 | 34          | 5         | 0     |

用NaN代替-99999：

```
new_df = df.replace(-99999, value=np.NaN)
new_df
```

```
# return
```

| day | temperature | windspeed | event |
|-----|-------------|-----------|-------|
| 0   | 1/1/2017    | 32.0 6.0  | Rain  |
| 1   | 1/2/2017    | NaN 7.0   | Sunny |
| 2   | 1/3/2017    | 28.0 NaN  | Snow  |
| 3   | 1/4/2017    | NaN 7.0   | 0     |
| 4   | 1/5/2017    | 32.0 NaN  | Rain  |
| 5   | 1/6/2017    | 31.0 2.0  | Sunny |
| 6   | 1/6/2017    | 34.0 5.0  | 0     |

代替两个数值-99999, -88888：

```
new_df = df.replace(to_replace=[-99999, -88888], value=0)
new_df
```

```
# return
```

| day | temperature | windspeed | event |
|-----|-------------|-----------|-------|
| 0   | 1/1/2017    | 32 6      | Rain  |
| 1   | 1/2/2017    | 0 7       | Sunny |
| 2   | 1/3/2017    | 28 0      | Snow  |
| 3   | 1/4/2017    | 0 7       | 0     |
| 4   | 1/5/2017    | 32 0      | Rain  |
| 5   | 1/6/2017    | 31 2      | Sunny |
| 6   | 1/6/2017    | 34 5      | 0     |

代替一行的数值：

```
new_df = df.replace({
    'temperature': -99999,
    'windspeed': -99999,
    'event': '0'
}, np.nan)
new_df
```

```
# return
```

|   | day      | temperature | windspeed | event |
|---|----------|-------------|-----------|-------|
| 0 | 1/1/2017 | 32.0        | 6.0       | Rain  |
| 1 | 1/2/2017 | NaN         | 7.0       | Sunny |
| 2 | 1/3/2017 | 28.0        | NaN       | Snow  |
| 3 | 1/4/2017 | NaN         | 7.0       | NaN   |
| 4 | 1/5/2017 | 32.0        | NaN       | Rain  |
| 5 | 1/6/2017 | 31.0        | 2.0       | Sunny |
| 6 | 1/6/2017 | 34.0        | 5.0       | NaN   |

用mapping代替：

```
new_df = df.replace({
    -99999: np.nan,
    'no event': 'Sunny',
})
new_df
```

```
# return
```

|   | day      | temperature | windspeed | event |
|---|----------|-------------|-----------|-------|
| 0 | 1/1/2017 | 32.0        | 6.0       | Rain  |
| 1 | 1/2/2017 | NaN         | 7.0       | Sunny |
| 2 | 1/3/2017 | 28.0        | NaN       | Snow  |
| 3 | 1/4/2017 | NaN         | 7.0       | 0     |
| 4 | 1/5/2017 | 32.0        | NaN       | Rain  |
| 5 | 1/6/2017 | 31.0        | 2.0       | Sunny |
| 6 | 1/6/2017 | 34.0        | 5.0       | 0     |

代替整个列表：

```
df = pd.DataFrame({
    'score': ['exceptional', 'average', 'good', 'poor', 'average', 'exceptional'],
    'student': ['rob', 'maya', 'parthiv', 'tom', 'julian', 'erica']
})
df

# return

```

|   | score       | student |
|---|-------------|---------|
| 0 | exceptional | rob     |
| 1 | average     | maya    |
| 2 | good        | parthiv |
| 3 | poor        | tom     |
| 4 | average     | julian  |
| 5 | exceptional | erica   |

## split()

练习1：数据科学 with Flask 的例子：

### 1. split''

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)

# Add new feature(integer) for bhk (Bedrooms Hall Kitchen)
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0])) # 拆解' ', 取第一个元素

# return 即为 2, 4....

df3['bhk'].unique()

# return
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
        13, 18])

```

### 2. split'-'

```

array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)

def is_float(x):
    try:
        float(x)
        return True
    except ValueError:
        return False

df3[df3['total_sqft'].apply(is_float)].head(10) # 没有反应, 因为有 ' - '

def convert_sqft_to_num(x):
    token = x.split('-')
    if len(token) == 2:
        return (float(token[0]) + float(token[1])) / 2
    try:
        return float(x)
    except:
        return np.nan

convert_sqft_to_num('1000-1200') # token[0] = 1000 token[1] = 1200
# return
1100.0

```

## 利用数据的均值、中位数值或众数

mean()、median() 和 mode() 方法计算列的均值（所有值加起来的平均值）、中位数值（排序后排在中间的数）和众数（出现频率最高的数）：

mean()：

```

import pandas as pd

df = pd.read_csv('property-data.csv')

x = df["ST_NUM"].mean()

df["ST_NUM"].fillna(x, inplace = True)

print(df.to_string())

```

median()：

```
import pandas as pd

df = pd.read_csv('property-data.csv')

x = df["ST_NUM"].median()

df["ST_NUM"].fillna(x, inplace = True)

print(df.to_string())
```

mode() :

```
import pandas as pd

df = pd.read_csv('property-data.csv')

x = df["ST_NUM"].mode()

df["ST_NUM"].fillna(x, inplace = True)

print(df.to_string())
```

## 清洗格式错误数据

过包含空单元格的行，或者将列中的所有单元格转换为相同格式的数据

```
import pandas as pd

# 第三个日期格式错误
data = {
    "Date": ['2020/12/01', '2020/12/02' , '20201226'],
    "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

df['Date'] = pd.to_datetime(df['Date'])

print(df.to_string())

# return
      Date  duration
day1 2020-12-01      50
day2 2020-12-02      40
day3 2020-12-26      45
```

## 清洗错误数据

对错误的数据进行替换或删除：

```
import pandas as pd

person = {
    "name": ['Google', 'Runoob' , 'Taobao'],
    "age": [50, 40, 12345] # 12345 年龄数据是错误的
}

df = pd.DataFrame(person)

df.loc[2, 'age'] = 30 # 修改数据

print(df.to_string())

# return
      name  age
0  Google   50
1  Runoob   40
2  Taobao   30
```

设置条件语句：

```

import pandas as pd

person = {
    "name": ['Google', 'Runoob' , 'Taobao'],
    "age": [50, 200, 12345]
}

df = pd.DataFrame(person)

for x in df.index:
    if df.loc[x, "age"] > 120:
        df.loc[x, "age"] = 120

print(df.to_string())

# return
   name  age
0  Google   50
1  Runoob  120
2  Taobao  120

```

## 清洗重复数据

如果我们要清洗重复数据，可以使用 `uplicated()` 和 `drop_duplicates()` 方法。  
 如果对应的数据是重复的，`uplicated()` 会返回 `True`，否则返回 `False`。

```

import pandas as pd

person = {
    "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
    "age": [50, 40, 40, 23]
}

df = pd.DataFrame(person)

print(df.duplicated())

# return
0    False
1    False
2     True
3    False
dtype: bool

```

删除重复数据，可以直接使用 `drop_duplicates()` 方法：



```
import pandas as pd

persons = {
    "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
    "age": [50, 40, 40, 23]
}

df = pd.DataFrame(persons)

df.drop_duplicates(inplace = True)
print(df)

# return
   name  age
0  Google   50
1  Runoob   40
3  Taobao   23
```

## 数据处理columns 列

### df.columns

返回index：

```
import pandas as pd
weather_data = {
    'day': ['1/1/2017', '1/2/2017', '1/3/2017', '1/4/2017', '1/5/2017', '1/6/2017'],
    'temperature': [32, 35, 28, 24, 32, 31],
    'windspeed': [6, 7, 2, 7, 4, 2],
    'event': ['Rain', 'Sunny', 'Snow', 'Snow', 'Rain', 'Sunny']
}
df = pd.DataFrame(weather_data)
df

df.columns

# return
Index(['day', 'temperature', 'windspeed', 'event'], dtype='object')
```

## df

`df[' ']`

即为`df.day`，返回`day`列的数据，不带`dataframe`，因为这是一个`pandas.core.series.Series`

```
df["day"]

# return
0    1/1/2017
1    1/2/2017
2    1/3/2017
3    1/4/2017
4    1/5/2017
5    1/6/2017
Name: day, dtype: object
```

## df

`df[[ ]]`

返回`dataframe`格式的数据，且有两个数据列或以上，不然单个`[]`会报错：

```
df[['day', 'temperature']]

# return
day      temperature
0    1/1/2017         32
1    1/2/2017         35
2    1/3/2017         28
3    1/4/2017         24
4    1/5/2017         32
5    1/6/2017         31
```

## set\_index

设置某个`column`为`index`：

```

df.set_index('day')
# df.set_index('day', inplace=True)

#return
temperature      windspeed      event
day
1/1/2017          32           6      Rain
1/2/2017          35           7      Sunny
1/3/2017          28           2      Snow
1/4/2017          24           7      Snow
1/5/2017          32           4      Rain
1/6/2017          31           2      Sunny

df.index

# return
Index(['1/1/2017', '1/2/2017', '1/3/2017', '1/4/2017', '1/5/2017', '1/6/2017'], dtype='object')

```

## reset\_index

```

df.reset_index(inplace=True)
df.head()

# return
      day      temperature      windspeed      event
0  1/1/2017          32           6      Rain
1  1/2/2017          35           7      Sunny
2  1/3/2017          28           2      Snow
3  1/4/2017          24           7      Snow
4  1/5/2017          32           4      Rain

df.set_index('event', inplace=True)
df

# return
      day      temperature      windspeed
event
Rain    1/1/2017          32           6
Sunny   1/2/2017          35           7
Snow    1/3/2017          28           2
Snow    1/4/2017          24           7
Rain    1/5/2017          32           4
Sunny   1/6/2017          31           2

```

## loc() by label

Pandas 可以使用 loc 属性返回指定行里面的数据

1. 第一行索引为 0，第二行索引为 1，以此类推：

```
import pandas as pd

data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}

# 数据载入到 DataFrame 对象
df = pd.DataFrame(data)

# 返回第一行
print(df.loc[0])
# 返回第二行
print(df.loc[1])

# return
calories    420
duration     50
Name: 0, dtype: int64
calories    380
duration     40
Name: 1, dtype: int64
```

2. 设置索引，即[" "]

```
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

# 指定索引
print(df.loc["day2"])

# return
calories    380
duration     40
Name: day2, dtype: int64
```

3. 可以返回多行数据，使用 [[ ... ]] 格式，... 为各行的索引，以逗号隔开：

```
# 数据载入到 DataFrame 对象
df = pd.DataFrame(data)

# 返回第一行和第二行
print(df.loc[[0, 1]])

# return
   calories  duration
0       420        50
1       380        40
```

## iloc() by position

与loc一样，第一行索引为 0，第二行索引为 1，以此类推：

```
# 返回第一行
print(df.iloc[0])
# 返回第二行
print(df.iloc[1])

# return
calories    420
duration     50
Name: 0, dtype: int64
calories    380
duration     40
Name: 1, dtype: int64
```

## .columns

### 练习1：数据科学 with Flask

获取所有数组的index [ ][ ]

```

X.columns
Index(['total_sqft', 'bath', 'bhk', '1st Block Jayanagar',
      '1st Phase JP Nagar', '2nd Phase Judicial Layout',
      '2nd Stage Nagarbhavi', '5th Block Hbr Layout', '5th Phase JP Nagar',
      '6th Phase JP Nagar',
      ...,
      'Vijayanagar', 'Vishveshwarya Layout', 'Vishwapriya Layout',
      'Vittasandra', 'Whitefield', 'Yelachenahalli', 'Yelahanka',
      'Yelahanka New Town', 'Yelenahalli', 'Yeshwanthpur'],
      dtype='object', length=244)

np.where(X.columns == '2nd Phase Judicial Layout')[0][0] # index of 2nd Phase Judicial Lay
# if one [0] only, return array([5])

# return
5

```

## 数据处理rows 行

### head()

head(n) 方法用于读取前面的 n 行，如果不填参数 n，默认返回 5 行。

```

import pandas as pd

df = pd.read_csv('nba.csv')

print(df.head())

```

### tail()

tail(n) 方法用于读取尾部的 n 行，如果不填参数 n，默认返回 5 行，空行各个字段的值返回 NaN。

```
import pandas as pd

df = pd.read_csv('nba.csv')

print(df.tail())
```

## info()

info() 方法返回表格的一些基本信息

```
import pandas as pd

df = pd.read_csv('nba.csv')

print(df.info())
```

# return  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 458 entries, 0 to 457  
Data columns (total 9 columns):

| # | Column   | Non-Null Count | Dtype   |
|---|----------|----------------|---------|
| 0 | Name     | 457 non-null   | object  |
| 1 | Team     | 457 non-null   | object  |
| 2 | Number   | 457 non-null   | float64 |
| 3 | Position | 457 non-null   | object  |
| 4 | Age      | 457 non-null   | float64 |
| 5 | Height   | 457 non-null   | object  |
| 6 | Weight   | 457 non-null   | float64 |
| 7 | College  | 373 non-null   | object  |
| 8 | Salary   | 446 non-null   | float64 |

dtypes: float64(4), object(5)

# 行数, 458 行, 第一行编号为 0  
# 列数, 9列  
# 各列的数据类型  
  
# non-null, 意思为非空的数据  
# 类型

## describe()

生成描述的数据：

```
df.describe()

# return
```

|       | temperature | windspeed |
|-------|-------------|-----------|
| count | 6.000000    | 6.000000  |
| mean  | 30.333333   | 4.666667  |
| std   | 3.829708    | 2.338090  |
| min   | 24.000000   | 2.000000  |
| 25%   | 28.750000   | 2.500000  |
| 50%   | 31.500000   | 5.000000  |
| 75%   | 32.000000   | 6.750000  |
| max   | 35.000000   | 7.000000  |

---

## feature engineering 特征工程

### replace

```
    score      student
0  exceptional      rob
1   average      maya
2    good    parthiv
3    poor       tom
4   average    julian
5  exceptional    erica

df.replace(['poor', 'average', 'good', 'exceptional'], [1,2,3,4])

# return
```

|   | score | student |
|---|-------|---------|
| 0 | 4     | rob     |
| 1 | 2     | maya    |
| 2 | 3     | parthiv |
| 3 | 1     | tom     |
| 4 | 2     | julian  |
| 5 | 4     | erica   |

### mapping

map是针对单列，apply是针对多列/一行也可，applymap是针对全部元素



```

import numpy as np
import pandas as pd
boolean=[True,False]
gender=["male","female"]
color=["white","black","yellow"]
data=pd.DataFrame({
    "height":np.random.randint(150,190,100),
    "weight":np.random.randint(40,90,100),
    "smoker":[boolean[x] for x in np.random.randint(0,2,100)],
    "gender":[gender[x] for x in np.random.randint(0,2,100)],
    "age":np.random.randint(15,90,100),
    "color":[color[x] for x in np.random.randint(0,len(color),100)]
})
data

# return

```

|     | height | weight | smoker | gender | age | color  |
|-----|--------|--------|--------|--------|-----|--------|
| 0   | 188    | 43     | True   | male   | 65  | black  |
| 1   | 177    | 82     | False  | male   | 39  | white  |
| 2   | 186    | 86     | False  | male   | 30  | white  |
| 3   | 159    | 79     | False  | female | 80  | black  |
| 4   | 175    | 80     | True   | female | 88  | black  |
| ... | ...    | ...    | ...    | ...    | ... | ...    |
| 95  | 170    | 48     | True   | male   | 21  | yellow |
| 96  | 187    | 79     | True   | male   | 64  | black  |
| 97  | 170    | 61     | True   | male   | 39  | black  |
| 98  | 172    | 63     | True   | female | 69  | black  |
| 99  | 169    | 48     | True   | female | 60  | white  |

```

)

```

```
data["gender"] = data["gender"].map({"male":1, "female":0})
data.head(10)
```

```
# return
```

|   | height | weight | smoker | gender | age | color  |
|---|--------|--------|--------|--------|-----|--------|
| 0 | 187    | 85     | True   | 1      | 17  | white  |
| 1 | 186    | 88     | False  | 0      | 31  | black  |
| 2 | 170    | 82     | True   | 1      | 55  | white  |
| 3 | 178    | 72     | True   | 0      | 34  | white  |
| 4 | 158    | 65     | False  | 1      | 24  | yellow |
| 5 | 155    | 69     | False  | 0      | 48  | yellow |
| 6 | 154    | 51     | False  | 0      | 76  | yellow |
| 7 | 162    | 86     | False  | 0      | 75  | white  |
| 8 | 157    | 44     | False  | 1      | 15  | white  |
| 9 | 159    | 54     | True   | 0      | 46  | white  |

Another example:

|   | Survived | Pclass | Sex    | Age  | Fare    |
|---|----------|--------|--------|------|---------|
| 0 | 0        | 3      | male   | 22.0 | 7.2500  |
| 1 | 1        | 1      | female | 38.0 | 71.2833 |
| 2 | 1        | 3      | female | 26.0 | 7.9250  |
| 3 | 1        | 1      | female | 35.0 | 53.1000 |
| 4 | 0        | 3      | male   | 35.0 | 8.0500  |

```
df.Sex = df.Sex.map({'female':1, 'male':0})
df.Sex
```

|   |   |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 0 |

|     |   |
|-----|---|
| ..  |   |
| 886 | 0 |
| 887 | 1 |
| 888 | 1 |
| 889 | 0 |
| 890 | 0 |

Name: Sex, Length: 891, dtype: int64

```
# return
```

|     | Survived | Pclass | Sex | Age       | Fare    |
|-----|----------|--------|-----|-----------|---------|
| 0   | 0        | 3      | 0   | 22.000000 | 7.2500  |
| 1   | 1        | 1      | 1   | 38.000000 | 71.2833 |
| 2   | 1        | 3      | 1   | 26.000000 | 7.9250  |
| 3   | 1        | 1      | 1   | 35.000000 | 53.1000 |
| 4   | 0        | 3      | 0   | 35.000000 | 8.0500  |
| ... | ...      | ...    | ... | ...       | ...     |
| 886 | 0        | 2      | 0   | 27.000000 | 13.0000 |
| 887 | 1        | 1      | 1   | 19.000000 | 30.0000 |
| 888 | 0        | 3      | 1   | 29.699118 | 23.4500 |
| 889 | 1        | 1      | 0   | 26.000000 | 30.0000 |
| 890 | 0        | 3      | 0   | 32.000000 | 7.7500  |

## apply

apply方法的作用原理和map方法类似，区别在于apply能够传入功能更为复杂的函数：

可以看到age列都减了3

```
def apply_age(x,bias):
    return x+bias
#以元组的方式传入额外的参数
data["age"] = data["age"].apply(apply_age,args=(-3,))
data

# return
```

| height | weight | smoker | gender | age | color |        |
|--------|--------|--------|--------|-----|-------|--------|
| 0      | 162    | 61     | True   | 1   | 58    | white  |
| 1      | 177    | 89     | True   | 1   | 17    | black  |
| 2      | 168    | 88     | False  | 1   | 79    | white  |
| 3      | 152    | 74     | True   | 1   | 68    | yellow |
| 4      | 170    | 66     | False  | 1   | 70    | black  |
| ...    | ...    | ...    | ...    | ... | ...   | ...    |
| 95     | 163    | 63     | False  | 1   | 59    | white  |
| 96     | 177    | 42     | False  | 0   | 18    | white  |
| 97     | 168    | 65     | True   | 0   | 74    | white  |
| 98     | 169    | 78     | False  | 0   | 16    | white  |
| 99     | 189    | 54     | False  | 1   | 44    | white  |

练习1：数据科学 with Flask 的例子：

1. 全部split', 获取第一个element

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
      '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
      '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
      '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
      '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
      '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)

# Add new feature(integer) for bhk (Bedrooms Hall Kitchen)
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0])) # 拆解' ', 取第一个元素

# return 即为 2, 4....
```

2. 变浮数点

```
df3[df3['total_sqft'].apply(is_float)].head(10)
```

```
df4 = df3.copy()
```

```
df4['total_sqft'] = df4['total_sqft'].apply(convert_sqft_to_num)
```

```
df4.head()
```

```
# return
```

|   | location                 | size      | total_sqft | bath | price  | bhk |
|---|--------------------------|-----------|------------|------|--------|-----|
| 0 | Electronic City Phase II | 2 BHK     | 1056.0     | 2.0  | 39.07  |     |
| 1 | Chikka Tirupathi         | 4 Bedroom | 2600.0     | 5.0  | 120.00 |     |
| 2 | Uttarahalli              | 3 BHK     | 1440.0     | 2.0  | 62.00  | 3   |
| 3 | Lingadheeranahalli       | 3 BHK     | 1521.0     | 3.0  | 95.00  | 3   |
| 4 | Kothanur                 | 2 BHK     | 1200.0     | 2.0  | 51.00  | 2   |

## applymap

applymap会对DataFrame中的每个单元格执行指定函数的操作

```
df = pd.DataFrame(
```

```
{
```

```
    "A":np.random.randn(5),
```

```
    "B":np.random.randn(5),
```

```
    "C":np.random.randn(5),
```

```
    "D":np.random.randn(5),
```

```
    "E":np.random.randn(5),
```

```
})
```

```
)
```

```
df
```

```
df.applymap(lambda x:"%.2f" % x)
```

```
# return
```

| A | B     | C     | D     | E     |       |
|---|-------|-------|-------|-------|-------|
| 0 | -0.70 | -1.04 | -0.14 | -0.57 | 0.58  |
| 1 | -0.43 | 0.55  | 0.06  | -1.61 | -0.57 |
| 2 | -0.96 | 0.43  | 0.19  | 0.63  | 0.03  |
| 3 | 0.45  | -0.28 | 0.95  | -0.72 | -1.35 |
| 4 | -0.65 | 0.86  | -1.60 | -1.28 | -0.07 |

## get\_dummies 1010

Using `get_dummies` while using `pandas` to convert categorical variable into dummy/indicator variables. For example, if there is `female` and `male` columns, and using `get_dummies` will generate two new columns for them. Usually it is used when there is a need to drop one of the column for prediction, like Naive Bayes.

```
data_df = pd.get_dummies(data.gender)
data_df.head(10)
merged = pd.concat([data, data_df], axis = 1) # left, right, columns
merged
```

```
# return
```

| height | weight | smoker | gender | age    | color | female |     |     |
|--------|--------|--------|--------|--------|-------|--------|-----|-----|
| 0      | 155    | 74     | True   | female | 28    | white  | 1   | 0   |
| 1      | 173    | 75     | True   | male   | 47    | yellow | 0   | 1   |
| 2      | 172    | 61     | True   | female | 51    | yellow | 1   | 0   |
| 3      | 171    | 88     | True   | male   | 67    | yellow | 0   | 1   |
| 4      | 163    | 40     | False  | male   | 15    | white  | 0   | 1   |
| ...    | ...    | ...    | ...    | ...    | ...   | ...    | ... | ... |
| 95     | 177    | 70     | False  | male   | 88    | white  | 0   | 1   |
| 96     | 172    | 52     | False  | male   | 79    | black  | 0   | 1   |
| 97     | 158    | 77     | True   | male   | 82    | yellow | 0   | 1   |
| 98     | 159    | 55     | False  | male   | 57    | yellow | 0   | 1   |
| 99     | 182    | 74     | False  | female | 53    | white  | 1   |     |

## LabelEncoder 1010 and OneHotEncoder

Label Encoder is for 10101010

while Oneencoder is for the array ([1,0,1,0...0,1,0,1])

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])

# onehotencoder = OneHotEncoder(categorical_features = [0])
# X = onehotencoder.fit_transform(X).toarray()
# labelencoder_Y = LabelEncoder()
# Y = labelencoder_Y.fit_transform(Y)

# onehotencoder = OneHotEncoder(categorical_features = [0])
onehotencoder = OneHotEncoder()
X = onehotencoder.fit_transform(X).toarray()

labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(y)
```

## split()

移除字符串头尾指定的字符

```

df5.location

# return
0      Electronic City Phase II
1      Chikka Tirupathi
2      Uttarahalli
3      Lingadheeranahalli
4      Kothanur
...
13315      Whitefield
13316      Richards Town
13317      Raja Rajeshwari Nagar
13318      Padmanabhanagar
13319      Doddathoguru
Name: location, Length: 13246, dtype: object

# Examine locations which is a categorical variable. We need to apply dimensionality reduction
df5.location = df5.location.apply(lambda x: x.strip()) # strip()用于移除字符串头尾指定的字符（默认为空格或换行符）

location_stats = df5.groupby('location')['location'].count() # how many types does it have
location_stats

# return
location
1 Annasandrapalya      1
1 Giri Nagar           1
1 Immadihalli          1
1 Ramamurthy Nagar     1
12th cross srinivas nagar banshankari 3rd stage  1
..
t.c palya              1
tc.palya               4
vinayakanagar          1
white field,kadugodi   1
whitefiled             1
Name: location, Length: 1293, dtype: int64

```

## operator: ~

The bitwise operator ~ (tilde) is a complement operator. So ~i on an integer value i is ~i=-i-1



```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
# negate on the criteria if you want to filter the rows
df6.shape
```

```
'''
In[1]: s = pd.Series(range(-3, 4))
Out[1]: s
0    -3
1    -2
2    -1
3     0
4     1
5     2
6     3

In[2]: s[~(s < 0)] -> -(s < 0)-1 -> s > -1
Out[2]:
3     0
4     1
5     2
6     3
dtype: int64
'''
```

## Groupby

筛选某个组别名：

```
g.get_group('mumbai')
```

# return

| day | city     | temperature | windspeed | event |       |
|-----|----------|-------------|-----------|-------|-------|
| 4   | 1/1/2017 | mumbai      | 90        | 5     | Sunny |
| 5   | 1/2/2017 | mumbai      | 85        | 12    | Fog   |
| 6   | 1/3/2017 | mumbai      | 87        | 15    | Fog   |
| 7   | 1/4/2017 | mumbai      | 92        | 5     | Rain  |

自定义组别：

```
def grouper(df, idx, col):
    if 80 <= df[col].loc[idx] <= 90:
        return '80-90'
    elif 50 <= df[col].loc[idx] <= 60:
        return '50-60'
    else:
        return 'others'

g = df.groupby(lambda x: grouper(df, x, 'temperature'))
g

for key, d in g:
```

```
# return
Group by Key: 50-60
```

|    | day      | city  | temperature | windspeed | event  |
|----|----------|-------|-------------|-----------|--------|
| 9  | 1/2/2017 | paris | 50          | 13        | Cloudy |
| 10 | 1/3/2017 | paris | 54          | 8         | Cloudy |

Group by Key: 80-90

|   | day      | city   | temperature | windspeed | event |
|---|----------|--------|-------------|-----------|-------|
| 4 | 1/1/2017 | mumbai | 90          | 5         | Sunny |
| 5 | 1/2/2017 | mumbai | 85          | 12        | Fog   |
| 6 | 1/3/2017 | mumbai | 87          | 15        | Fog   |

Group by Key: others

|    | day      | city     | temperature | windspeed | event  |
|----|----------|----------|-------------|-----------|--------|
| 0  | 1/1/2017 | new york | 32          | 6         | Rain   |
| 1  | 1/2/2017 | new york | 36          | 7         | Sunny  |
| 2  | 1/3/2017 | new york | 28          | 12        | Snow   |
| 3  | 1/4/2017 | new york | 33          | 7         | Sunny  |
| 7  | 1/4/2017 | mumbai   | 92          | 5         | Rain   |
| 8  | 1/1/2017 | paris    | 45          | 20        | Sunny  |
| 11 | 1/4/2017 | paris    | 42          | 10        | Cloudy |

# Concatenation 串联

```
import pandas as pd

india_weather = pd.DataFrame({
    "city": ["mumbai", "delhi", "banglore"],
    "temperature": [32, 45, 30],
    "humidity": [80, 60, 78]
})
india_weather

# return

```

|   | city     | temperature | humidity |
|---|----------|-------------|----------|
| 0 | mumbai   | 32          | 80       |
| 1 | delhi    | 45          | 60       |
| 2 | banglore | 30          | 78       |

```
us_weather = pd.DataFrame({
    "city": ["new york", "chicago", "orlando"],
    "temperature": [21, 14, 35],
    "humidity": [68, 65, 75]
})
us_weather

# return

```

|   | city     | temperature | humidity |
|---|----------|-------------|----------|
| 0 | new york | 21          | 68       |
| 1 | chicago  | 14          | 65       |
| 2 | orlando  | 35          | 75       |

```
df = pd.concat([india_weather, us_weather], ignore_index=True) # it ignores that original df


```

|   | city     | temperature | humidity |
|---|----------|-------------|----------|
| 0 | mumbai   | 32          | 80       |
| 1 | delhi    | 45          | 60       |
| 2 | banglore | 30          | 78       |
| 3 | new york | 21          | 68       |
| 4 | chicago  | 14          | 65       |
| 5 | orlando  | 35          | 75       |

## Concatenation And Keys

```
df = pd.concat([india_weather, us_weather], keys=["india", "us"])
df
```

```
# return
```

```
city      temperature      humidity
```

```
india
```

```
0      mumbai      32      80
```

```
1      delhi      45      60
```

```
2      banglore 30      78
```

```
us
```

```
0      new york 21      68
```

```
1      chicago      14      65
```

```
2      orlando      35      75
```

## Concatenation Using Index

```
temperature_df = pd.DataFrame({
    "city": ["mumbai","delhi","banglore"],
    "temperature": [32,45,30],
}, index=[0,1,2])
temperature_df

# return
city      temperature
0      mumbai      32
1      delhi       45
2      banglore    30

windspeed_df = pd.DataFrame({
    "city": ["delhi","mumbai"],
    "windspeed": [7,12],
}, index=[1,0])
windspeed_df

# return
city      windspeed
1      delhi       7
0      mumbai     12

df = pd.concat([temperature_df,windspeed_df],axis=1)
df

# return
city      temperature      city      windspeed
0      mumbai      32      mumbai     12.0
1      delhi       45      delhi      7.0
2      banglore    30      NaN       NaN
```

## Concatenate dataframe with series

```
s = pd.Series(["Humid","Dry","Rain"], name="event")
s

# return
0    Humid
1     Dry
2     Rain
Name: event, dtype: object

df = pd.concat([temperature_df,s],axis=1)
df

# return
```

|   | city     | temperature | event |
|---|----------|-------------|-------|
| 0 | mumbai   | 32          | Humid |
| 1 | delhi    | 45          | Dry   |
| 2 | banglore | 30          | Rain  |

---

## Merge Using a Dataframe Column 使用数据框架列进行合并

根据columns 行合并, on="city" :

```
import pandas as pd
df1 = pd.DataFrame({
    "city": ["new york", "chicago", "orlando"],
    "temperature": [21, 14, 35],
})
df1
```

```
# return
city      temperature
0      new york      21
1      chicago      14
2      orlando      35
```

```
df2 = pd.DataFrame({
    "city": ["chicago", "new york", "orlando"],
    "humidity": [65, 68, 75],
})
df2
```

```
# return
city      humidity
0      chicago      65
1      new york      68
2      orlando      75
```

```
df3 = pd.merge(df1, df2, on="city")
df3
```

```
# return
city      temperature      humidity
0      new york      21      68
1      chicago      14      65
2      orlando      35      75
```

# Joins

```
df1 = pd.DataFrame({
    "city": ["new york","chicago","orlando", "baltimore"],
    "temperature": [21,14,35, 38],
})
df1

df2 = pd.DataFrame({
    "city": ["chicago","new york","san diego"],
    "humidity": [65,68,71],
})
df2
```

## inner

```
df3=pd.merge(df1,df2,on="city",how="inner") #df1 left df2 right
df3
```

```
# return
```

|   | city     | temperature | humidity |
|---|----------|-------------|----------|
| 0 | new york | 21          | 68       |
| 1 | chicago  | 14          | 65       |

## outer

```
df3=pd.merge(df1,df2,on="city",how="outer")
df3
```

```
# return
```

|   | city      | temperature | humidity |
|---|-----------|-------------|----------|
| 0 | new york  | 21.0        | 68.0     |
| 1 | chicago   | 14.0        | 65.0     |
| 2 | orlando   | 35.0        | NaN      |
| 3 | baltimore | 38.0        | NaN      |
| 4 | san diego | NaN         | 71.0     |



## left

```
df3=pd.merge(df1,df2,on="city",how="left")
df3
```

```
# return
```

|   | city      | temperature | humidity |
|---|-----------|-------------|----------|
| 0 | new york  | 21          | 68.0     |
| 1 | chicago   | 14          | 65.0     |
| 2 | orlando   | 35          | NaN      |
| 3 | baltimore | 38          | NaN      |

## right

```
df3=pd.merge(df1,df2,on="city",how="right")
df3
```

```
# return
```

|   | city      | temperature | humidity |
|---|-----------|-------------|----------|
| 0 | new york  | 21.0        | 68       |
| 1 | chicago   | 14.0        | 65       |
| 2 | san diego | NaN         | 71       |

## indicator flags

```
df3=pd.merge(df1,df2,on="city",how="outer",indicator=True)
df3
```

```
# return
```

|   | city      | temperature | humidity | _merge     |
|---|-----------|-------------|----------|------------|
| 0 | new york  | 21.0        | 68.0     | both       |
| 1 | chicago   | 14.0        | 65.0     | both       |
| 2 | orlando   | 35.0        | NaN      | left_only  |
| 3 | baltimore | 38.0        | NaN      | left_only  |
| 4 | san diego | NaN         | 71.0     | right_only |

## suffix

前缀名字：

```
df3= pd.merge(df1,df2,on="city",how="outer", suffixes=('_first','_second'))
df3
```

```
# return
```

| city | humidity_first | temperature_first | humidity_second | temperature_second |
|------|----------------|-------------------|-----------------|--------------------|
| 0    | new york       | 65.0              | 21.0            | 68.0               |
| 1    | chicago        | 68.0              | 14.0            | 65.0               |
| 2    | orlando        | 71.0              | 35.0            | NaN                |
| 3    | baltimore      | 75.0              | 38.0            | NaN                |
| 4    | san diego      | NaN               | NaN             | 71.0               |

## Pivot Table 改造和重塑

```
df.pivot(index='city',columns='date')
```

```
# return
```

| temperature | humidity |          |          |          |          |    |
|-------------|----------|----------|----------|----------|----------|----|
| date        | 5/1/2017 | 5/2/2017 | 5/3/2017 | 5/1/2017 | 5/2/2017 |    |
| city        |          |          |          |          |          |    |
| beijing     | 80       | 77       | 79       | 26       | 30       | 35 |
| mumbai      | 75       | 78       | 82       | 80       | 83       | 85 |
| new york    | 65       | 66       | 68       | 56       | 58       | 60 |

只获取某个数值，但是保留框架：

```
df.pivot(index='city',columns='date',values="humidity") # only get value humidity
```

```
# return
```

| date     | 5/1/2017 | 5/2/2017 | 5/3/2017 |
|----------|----------|----------|----------|
| city     |          |          |          |
| beijing  | 26       | 30       | 35       |
| mumbai   | 80       | 83       | 85       |
| new york | 56       | 58       | 60       |

## Melt

syntax :

```
pandas.melt(frame, id_vars=None, value_vars=None, var_name=None, value_name='value', col_
```

```
import pandas as pd
df = pd.read_csv("weather.csv")
df
```

```
# return
```

| day | chicago   | chennai | berlin |    |
|-----|-----------|---------|--------|----|
| 0   | Monday    | 32      | 75     | 41 |
| 1   | Tuesday   | 30      | 77     | 43 |
| 2   | Wednesday | 28      | 75     | 45 |
| 3   | Thursday  | 22      | 82     | 38 |
| 4   | Friday    | 30      | 83     | 30 |
| 5   | Saturday  | 20      | 81     | 45 |
| 6   | Sunday    | 25      | 77     | 47 |

```
df1=pd.melt(df, id_vars=["day"]) # Column(s) to use as identifier variables
df1
```

```
# return
```

| day | variable  | value   |    |
|-----|-----------|---------|----|
| 0   | Monday    | chicago | 32 |
| 1   | Tuesday   | chicago | 30 |
| 2   | Wednesday | chicago | 28 |
| 3   | Thursday  | chicago | 22 |
| ... |           |         |    |
| 8   | Tuesday   | chennai | 77 |
| 9   | Wednesday | chennai | 75 |
| 10  | Thursday  | chennai | 82 |
| 11  | Friday    | chennai | 83 |
| 12  | Saturday  | chennai | 81 |
| ... |           |         |    |
| 17  | Thursday  | berlin  | 38 |
| 18  | Friday    | berlin  | 30 |
| 19  | Saturday  | berlin  | 45 |
| 20  | Sunday    | berlin  | 47 |

```

melted = pd.melt(df, id_vars=["day"], var_name='city', value_name='temperature')
# Column(s) to unpivot. If not specified, uses all columns that are not set as id_vars
# Name to use for the 'value' column.
melted

# return

```

|     | day       | city    | temperature |
|-----|-----------|---------|-------------|
| 0   | Monday    | chicago | 32          |
| 1   | Tuesday   | chicago | 30          |
| ... |           |         |             |
| 6   | Sunday    | chicago | 25          |
| 7   | Monday    | chennai | 75          |
| 8   | Tuesday   | chennai | 77          |
| 9   | Wednesday | chennai | 75          |
| ... |           |         |             |
| 15  | Tuesday   | berlin  | 43          |
| 16  | Wednesday | berlin  | 45          |
| 17  | Thursday  | berlin  | 38          |
| 18  | Friday    | berlin  | 30          |

## stack

## crosstab

|    | Name   | Nationality | Sex    | Age | Handedness |
|----|--------|-------------|--------|-----|------------|
| 0  | Kathy  | USA         | Female | 23  | Right      |
| 1  | Linda  | USA         | Female | 18  | Right      |
| 2  | Peter  | USA         | Male   | 19  | Right      |
| 3  | John   | USA         | Male   | 22  | Left       |
| 4  | Fatima | Bangadesh   | Female | 31  | Left       |
| 5  | Kadir  | Bangadesh   | Male   | 25  | Left       |
| 6  | Dhaval | India       | Male   | 35  | Left       |
| 7  | Sudhir | India       | Male   | 31  | Left       |
| 8  | Parvir | India       | Male   | 37  | Right      |
| 9  | Yan    | China       | Female | 52  | Right      |
| 10 | Juan   | China       | Female | 58  | Left       |
| 11 | Liang  | China       | Male   | 43  | Left       |

```
pd.crosstab(df.Nationality,df.Handedness)
```

```
# return
```

| Handedness  | Left | Right |
|-------------|------|-------|
| Nationality |      |       |
| Bangladesh  | 2    | 0     |
| China       | 2    | 1     |
| India       | 2    | 1     |
| USA         | 1    | 3     |

```
pd.crosstab(df.Sex,df.Handedness)
```

```
# return
```

| Handedness | Left | Right |
|------------|------|-------|
| Sex        |      |       |
| Female     | 2    | 3     |
| Male       | 5    | 2     |

## margins

返回all：

```
pd.crosstab(df.Sex,df.Handedness, margins=True) # all
```

```
# return
```

| Handedness | Left | Right | All |
|------------|------|-------|-----|
| Sex        |      |       |     |
| Female     | 2    | 3     | 5   |
| Male       | 5    | 2     | 7   |
| All        | 7    | 5     | 12  |

## multi index columns and rows

```
pd.crosstab(df.Sex, [df.Handedness,df.Nationality], margins=True)
```

```
# return
```

| Handedness  |   | Left      | Right | All   |   |       |     |       |
|-------------|---|-----------|-------|-------|---|-------|-----|-------|
| Nationality |   | Bangadesh |       | China |   | India | USA | China |
| Sex         |   |           |       |       |   |       |     |       |
| Female      | 1 | 1         | 0     | 0     | 1 | 0     | 2   | 5     |
| Male        | 1 | 1         | 2     | 1     | 0 | 1     | 1   | 7     |
| All         | 2 | 2         | 2     | 1     | 1 | 1     | 3   | 12    |

```
pd.crosstab([df.Nationality, df.Sex], [df.Handedness], margins=True)
```

```
# return
```

|             | Handedness |   | Left | Right | All |
|-------------|------------|---|------|-------|-----|
| Nationality | Sex        |   |      |       |     |
| Bangadesh   | Female     | 1 | 0    | 1     |     |
| Male        |            | 0 | 1    |       |     |
| China       | Female     | 1 | 1    | 2     |     |
| Male        |            | 0 | 1    |       |     |
| India       | Male       | 2 | 1    | 3     |     |
| USA         | Female     | 0 | 2    | 2     |     |
| Male        |            | 1 | 2    |       |     |
| All         |            | 7 | 5    | 12    |     |

## normalize

列出百分比：

```
pd.crosstab(df.Sex, df.Handedness, normalize='index') # percentage
```

```
# return
```

| Handedness |          | Left     | Right |
|------------|----------|----------|-------|
| Sex        |          |          |       |
| Female     | 0.400000 | 0.600000 |       |
| Male       | 0.714286 | 0.285714 |       |

## aggfunc and values

```
import numpy as np
pd.crosstab(df.Sex, df.Handedness, values=df.Age, aggfunc=np.average)
...
```

values : array-like, optional  
Array of values to aggregate according to the factors.  
Requires `aggfunc` be specified.

aggfunc : function, optional  
If specified, requires `values` be specified as well.

```
...
```

# return

| Handedness | Left | Right |
|------------|------|-------|
| Sex        |      |       |
| Female     | 44.5 | 31.0  |
| Male       | 31.2 | 28.0  |

---

## 数据操作

### 内置函数 max(), min(), std()

查找某个数值：

```
import pandas as pd
weather_data = {
    'day': ['1/1/2017', '1/2/2017', '1/3/2017', '1/4/2017', '1/5/2017', '1/6/2017'],
    'temperature': [32, 35, 28, 24, 32, 31],
    'windspeed': [6, 7, 2, 7, 4, 2],
    'event': ['Rain', 'Sunny', 'Snow', 'Snow', 'Rain', 'Sunny']
}
df = pd.DataFrame(weather_data)
df

df['temperature'].max()
# return
35
```

```
df['temperature'].std()
```

```
# return  
3.8297084310253524
```

查找某个数值对应的index：

```
df['day'][df['temperature'] == df['temperature'].max()]
```

```
# return  
1    1/2/2017  
Name: day, dtype: object
```

查找某个数值对应的dataframe：

```
df[df['temperature'] == df['temperature'].max()]
```

```
# return  
day      temperature  windspeed  event  
1    1/2/2017         35         7    Sunny
```

## 操作符 > < =

```
df[df['temperature']>32] # df[df.temperature>32]
```

```
# return  
day      temperature  windspeed  event  
1    1/2/2017         35         7    Sunny
```

---

## Summary

1. import data
2. check outlier/ null value
3. preprocessing/ feature engineering
4. split data
5. model



6. hyperparameter
  7. AOC/ ROC score
  8. Data visualization
- 

## Reference

[Runoob: Matplotlib](#)

[Runoob: NumPy](#)

[Runoob: Pandas](#)

[Matplotlib.org](#)

[Numpy.org](#)

[Pandas.pydata.org](#)

[Pandas教程 | 数据处理三板斧——map、apply、applymap详解](#)