数据可视化

1. 安装 `Matplotlib`

在Linux系统中安装`matplotlib`

`Ubuntu17.10`内置Python2版本和Python3版本，可以采用下面的方式安装Matplotlib。

```
$ sudo apt-get install python3-matplotlib
```

如果你使用的是Python 2.7，执行如下命令：

```
$ sudo apt-get install python-matplotlib
```

如果你安装了`pip`就可以使用下面的方式安装：

```
$ pip install matplotlib
```

如果你的安装比较慢，可以尝试这种方式来安装：

```
$ pip3 install matplotlib -i https://pypi.tuna.tsinghua.edu.cn/simple
```

> 常用的国内源地址有：
>
> - 阿里云 `http://mirrors.aliyun.com/pypi/simple/`
> - 中国科技大学 `https://pypi.mirrors.ustc.edu.cn/simple/`
> - 豆瓣(douban) `http://pypi.douban.com/simple/`
> - 清华大学 `https://pypi.tuna.tsinghua.edu.cn/simple/`
> - 中国科学技术大学 `http://pypi.mirrors.ustc.edu.cn/simple/`

在Windows系统中安装`matplotlib`

在Windows下下载Python后记得在安装的时候选择加入`pip`到环境变量。然后用下面的命令：

```
pip install matplotlib -i https://pypi.tuna.tsinghua.edu.cn/simple
```

测试matplotlib

```
>>>import matplotlib
>>>
```

如果没有出现任何错误信息，就代表安装成功！

2. 绘制简单的折线图

下面来使用 `matplotlib` 绘制一个简单的折线图，再对其进行定制，以实现信息更丰富的**数据可视化**。我们将使用平方数序列 1 、 4 、 9 、 16 和 25 来绘制折线图。

```
import matplotlib.pyplot as plt

squares = [1, 4, 9, 16, 25]

plt.plot(squares)

plt.show()
```

`plt.show()` 打开 `matplotlib` 查看器，并显示绘制的图形。

修改标签文字和线条粗细

图形表明数字是越来越大的，但标签文字太小，线条太细。所幸 `matplotlib` 让你能够调整可视化的各个方面。

```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
plt.plot(squares, linewidth=5)

# 设置图表标题，并给坐标轴加上标签
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# 设置刻度标记的大小
plt.tick_params(axis='both', labelsize=14)

plt.show()
```

校正图形

图形更容易阅读后，我们发现没有正确地绘制数据：折线图的终点指出 4.0 的平方为 25 ！下 面来修复这个问题。

```
import matplotlib.pyplot as plt

input_values = [1, 2, 3, 4, 5]

squares = [1, 4, 9, 16, 25]
plt.plot(input_values, squares, linewidth=5)

# 设置图表标题，并给坐标轴加上标签
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# 设置刻度标记的大小
```

```
plt.tick_params(axis='both', labelsize=14)

plt.show()
```

使用`scatter()`绘制散点图并设置其样式

有时候，需要绘制散点图并设置各个数据点的样式。要绘制单个点，可使用函数 scatter() ，并向它传递一对 x 和 y 坐标，它将在指定位置绘制一 个点：

```
import matplotlib.pyplot as plt
plt.scatter(2, 4)
plt.show()
```

下面来设置输出的样式，使其更有趣：添加标题，给轴加上标签，并确保所有文本都大到能够看清：

```
import matplotlib.pyplot as plt

plt.scatter(2, 4, s=200)

# 设置图表标题并给坐标轴加上标签
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# 设置刻度标记的大小
plt.tick_params(axis='both', labelsize=14)

plt.show()
```

使用`scatter()`绘制一系列点

要绘制一系列的点，可向 scatter() 传递两个分别包含 x 值和 y 值的列表，如下所示：

```
import matplotlib.pyplot as plt

x_values = [1, 2, 3, 4, 5]
y_values = [1, 4, 9, 16, 25]

plt.scatter(x_values, y_values, s=100)

# 设置图表标题并给坐标轴加上标签
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# 设置刻度标记的大小
plt.tick_params(axis='both', which='major', labelsize=14)

plt.show()
```

自动计算数据

手工计算列表要包含的值可能效率低下，需要绘制的点很多时尤其如此。

```
import matplotlib.pyplot as plt

x_values = list(range(1, 1001))

y_values = [x**2 for x in x_values]

plt.scatter(x_values, y_values, s=40)

# 设置每个坐标轴的取值范围

plt.axis([0, 1100, 0, 1100000])

plt.show()
```

`matplotlib`允许你给散点图中的各个点指定颜色。**默认为蓝色点和黑色轮廓**，在散点图包含的数据点不多时效果很好。但绘制很多点时，黑色轮廓可能会**粘连**在一起。要删除数据点的轮廓，可在调用`scatter()`时传递实参`edgecolor='none'`：

自定义颜色

要修改数据点的颜色，可向scatter()传递参数c，并将其设置为要使用的颜色的名称，如下所示：

```
plt.scatter(x_values, y_values, c='red', edgecolor='none', s=40)
```

你还可以使用RGB颜色模式自定义颜色。

```
plt.scatter(x_values, y_values, c=(0, 0, 0.8), edgecolor='none', s=40)
```

`(0, 0, 0.8)`它们分别表示红色、绿色和蓝色分量。值越接近0，指定的颜色越深，值越接近1，指定的颜色越浅。

使用颜色映射

颜色映射（colormap）是一系列颜色，它们从起始颜色渐变到结束颜色。在可视化中，颜色映射用于突出数据的规律，例如，你可能用较浅的颜色来显示较小的值，并使用较深的颜色来显示较大的值。

```
import matplotlib.pyplot as plt

x_values = list(range(1001))
y_values = [x**2 for x in x_values]

plt.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, edgecolor='none', s=40)
```

这些代码将y值较小的点显示为浅蓝色，并将y值较大的点显示为深蓝色。

自动保存图表

```
plt.savefig('squares_plot.png', bbox_inches='tight')
```

第二个实参指定将图表多余的空白区域裁剪掉。如果要保留图表周围多余的空白区域，可省略这个实参。

3. 随机漫步

在自然界、物理学、生物学、化学和经济领域，随机漫步都有其实际用途。例如，漂浮在水滴上的花粉因不断受到水分子的挤压而在水面上移动。水滴中的分子运动是随机的，因此花粉在水面上的运动路径犹如随机漫步。我们稍后将编写的代码模拟了现实世界的很多情形。

创建`RandomWalk()`类

为模拟随机漫步，我们将创建一个名为`RandomWalk`的类，它随机地选择前进方向。这个类需要三个属性，其中一个是存储随机漫步次数的变量，其他两个是列表，分别存储随机漫步经过的每个点的x和y坐标。

`RandomWalk`类只包含两个方法：`__init__ ()`和`fill_walk()`，其中后者计算随机漫步经过的所有点。下面先来看看`__init__()`，如下所示：

```
from random import choice

class RandomWalk():
    """一个生成随机漫步数据的类"""
    def __init__(self, num_points=5000):
        """初始化随机漫步的属性"""
        self.num_points = num_points
        # 所有随机漫步都始于(0, 0)
        self.x_values = [0]
        self.y_values = [0]
```

选择方向

我们将使用`fill_walk()`来生成漫步包含的点，并决定每次漫步的方向。

```
def fill_walk(self):
    """计算随机漫步包含的所有点"""

    # 不断漫步，直到列表达到指定的长度
    while len(self.x_values) < self.num_points:
        # 决定前进方向以及沿这个方向前进的距离
        x_direction = choice([1, -1])
        x_distance = choice([0, 1, 2, 3, 4])
        x_step = x_direction * x_distance

        y_direction = choice([1, -1])
        y_distance = choice([0, 1, 2, 3, 4])
        y_step = y_direction * y_distance
```

```
        # 拒绝原地踏步
        if x_step == 0 and y_step == 0:
            continue
        # 计算下一个点的x和y值
        next_x = self.x_values[-1] + x_step
        next_y = self.y_values[-1] + y_step

        self.x_values.append(next_x)
        self.y_values.append(next_y)
```

绘制随机漫步图

下面的代码将随机漫步的所有点都绘制出来：

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个RandomWalk实例，并将其包含的点都绘制出来

rw = RandomWalk()

rw.fill_walk()

plt.scatter(rw.x_values, rw.y_values, s=15)

plt.show()
```

给点着色

我们将使用颜色映射来指出漫步中各点的先后顺序，并删除每个点的黑色轮廓，让它们的颜色更明显。为根据漫步中各点的先后顺序进行着色，我们传递参数c，并将其设置为一个列表，其中包含各点的先后顺序。由于这些点是按顺序绘制的，因此给参数c指定的列表只需包含数字1~5000，如下所示：

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个RandomWalk实例，并将其包含的点都绘制出来

rw = RandomWalk()

rw.fill_walk()

point_numbers = list(range(rw.num_points))
plt.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues, edgecolor='none',s=15)

plt.show()
```

重新绘制起点和终点

除了给随机漫步的各个点着色，以指出它们的先后顺序外，如果还能呈现随机漫步的起点和终点就更好了。为此，可在绘制随机漫步图后重新绘制起点和终点。我们让起点和终点变得更大，并显示为不同的颜色，以突出它们，如下所示：

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个RandomWalk实例，并将其包含的点都绘制出来

rw = RandomWalk()

rw.fill_walk()

point_numbers = list(range(rw.num_points))
plt.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues, edgecolor='none',s=15)

# 突出起点和终点
plt.scatter(0, 0, c='green', edgecolors='none', s=100)
plt.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none', s=100)
plt.show()
```

隐藏坐标轴

如果不想显示坐标的尺寸，可以隐藏：

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个RandomWalk实例，并将其包含的点都绘制出来

rw = RandomWalk()

rw.fill_walk()

point_numbers = list(range(rw.num_points))
plt.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues, edgecolor='none',
s=15)

# 突出起点和终点
plt.scatter(0, 0, c='green', edgecolors='none', s=100)
plt.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none', s=100)

plt.axes().get_xaxis().set_visible(False)

plt.axes().get_yaxis().set_visible(False)

plt.show()
```

调整尺寸以适合屏幕

在不同的电脑上面，由于屏幕的不同，图像的大小也是不同的，为了达到这种效果，我们可以这样做：

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个RandomWalk实例，并将其包含的点都绘制出来

rw = RandomWalk()

rw.fill_walk()

# 设置绘图窗口的尺寸
plt.figure(figsize=(10, 6))

point_numbers = list(range(rw.num_points))
plt.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues, edgecolor='none',
s=15)

# 突出起点和终点
plt.scatter(0, 0, c='green', edgecolors='none', s=100)
plt.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
    s=100)

plt.axes().get_xaxis().set_visible(False)

plt.axes().get_yaxis().set_visible(False)

plt.show()
```

4. Matplotlib进阶-Seaborn

`Seaborn`其实是在`matplotlib`的基础上进行了更高级的`API`封装，从而使得作图更加容易，在大多数情况下使用`seaborn`就能做出很具有吸引力的图。

安装方式

安装方式类似于`matplotlib`，在Windows下和Linux下面都可以采用`pip`安装方式。

set_style( )

`set_style( )`是用来设置主题的，`Seaborn`有五个预设好的主题：`darkgrid , whitegrid , dark , white`,和`ticks`默认：`darkgrid`

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

sns.set_style("whitegrid")
plt.plot(range(10))

plt.show()
```

直方图

直方图的绘制：

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_iris = pd.read_csv(r'D:\Windows 7 Documents\Desktop\iris.csv')

sns.distplot(df_iris['petal_length'], kde = True)   # kde 密度曲线  rug 边际毛毯

plt.show()
```

箱型图

箱形图（Box-plot）又称为盒须图、盒式图或箱线图，是一种用作显示一组数据分散情况资料的统计图。因形状如箱子而得名。

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_iris = pd.read_csv(r'D:\Windows 7 Documents\Desktop\iris.csv')

sns.boxplot(x = df_iris['species'], y = df_iris['sepal_width'])

plt.show()
```

联合分布

两个变量的画图

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_iris = pd.read_csv(r'D:\Windows 7 Documents\Desktop\iris.csv')

sns.jointplot(df_iris['petal_width'], df_iris['sepal_width'])

plt.show()
```

不用圆点表示的话也是可以的，可以用其他方式来表示，比如六角形来表示：

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_iris = pd.read_csv(r'D:\Windows 7 Documents\Desktop\iris.csv')

sns.jointplot(df_iris['petal_width'], df_iris['sepal_width'], kind='hex')

plt.show()
```

热力图

相关系数是最早由统计学家卡尔·皮尔逊设计的统计指标，是研究变量之间线性相关程度的量，一般用字母 r 表示。由于研究对象的不同，相关系数有多种定义方式，较为常用的是皮尔逊相关系数。 相关系数是用以反映变量之间相关关系密切程度的统计指标。相关系数是按积差方法计算，同样以两变量与各自平均值的离差为基础，通过两个离差相乘来反映两变量之间相关程度；着重研究线性的单相关系数。公式：

$$r(X,Y)=\frac{Cov(X,Y)}{\sqrt{Var[X]Var[Y]}}$$

以上公式中：

$Cov(X,Y)=E[XY]-E[X]E[Y]$

$Var(X)=E\{[X-E(X)^2]\}=E(X^2)-[E(X)]^2$

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_iris = pd.read_csv(r'D:\Windows 7 Documents\Desktop\iris.csv')

corrmat = df_iris[df_iris.columns[:4]].corr()

sns.heatmap(corrmat, square=True, linewidths=.5, annot=True)

plt.show()
```

多变量图

关注数据框中各个特征之间的相关关系，呈现图形的展示，给人以直观的感受。而不是"冰冷"的数字。可以非常方便的找到各个特征之间呈现什么样的关系。比如线性，离散等关系。

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv(r"D:\Windows 7 Documents\Desktop\iris.csv")

sns.set(style="ticks")    # 使用默认配色
sns.pairplot(data,hue="species")   # hue 选择分类列

plt.show()
```