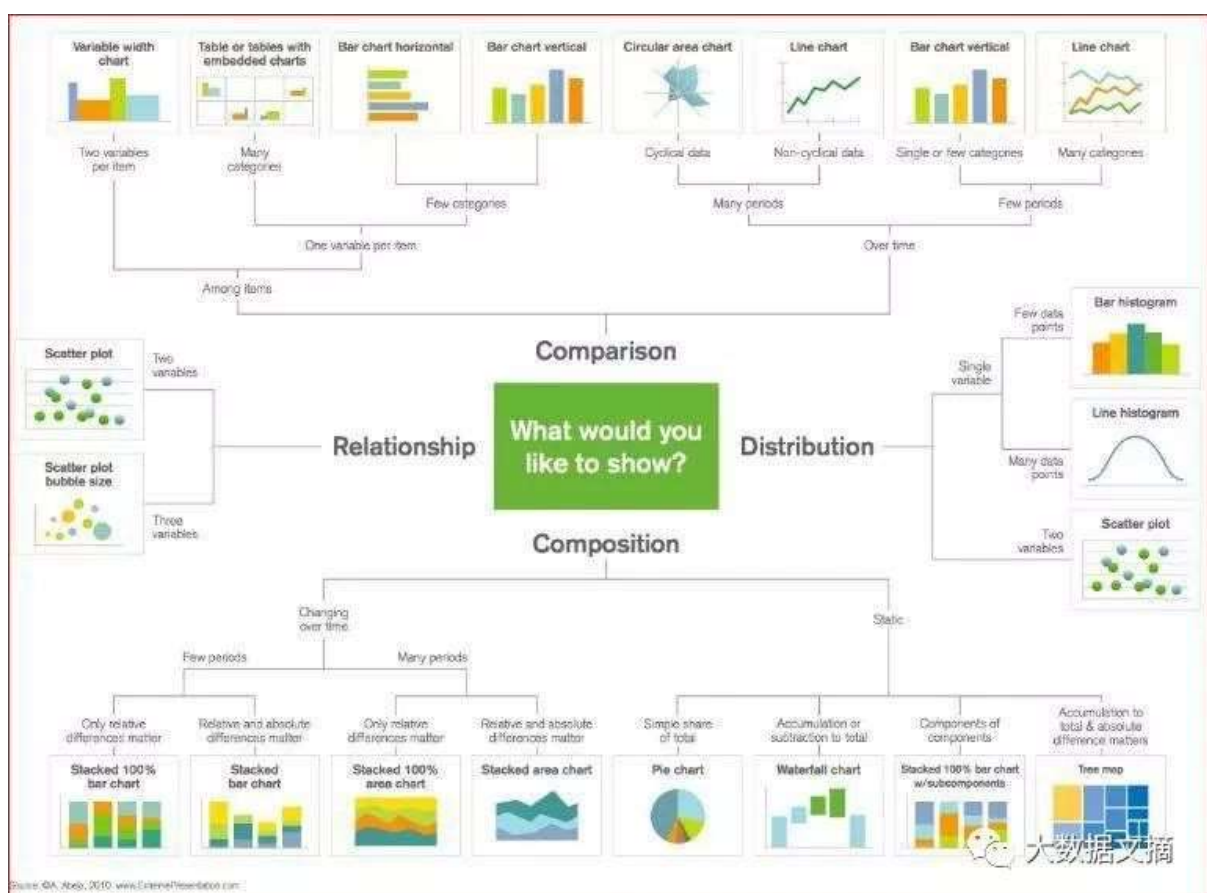


数据可视化 (Python+Matplotlib)

本文要讲的是Matplotlib，一个强大的Python可视化库。一共5小段代码，轻松实现散点图、折线图、直方图、柱状图、箱线图。

Matplotlib是个很流行的Python库，可以轻松实现数据可视化。但是，每次执行新项目的绘图时，设置数据、参数、图形的过程都非常的繁琐。在本文中，我们将着眼于5种数据可视化方法，用Python的Matplotlib库实现一些快速而简单的功能。

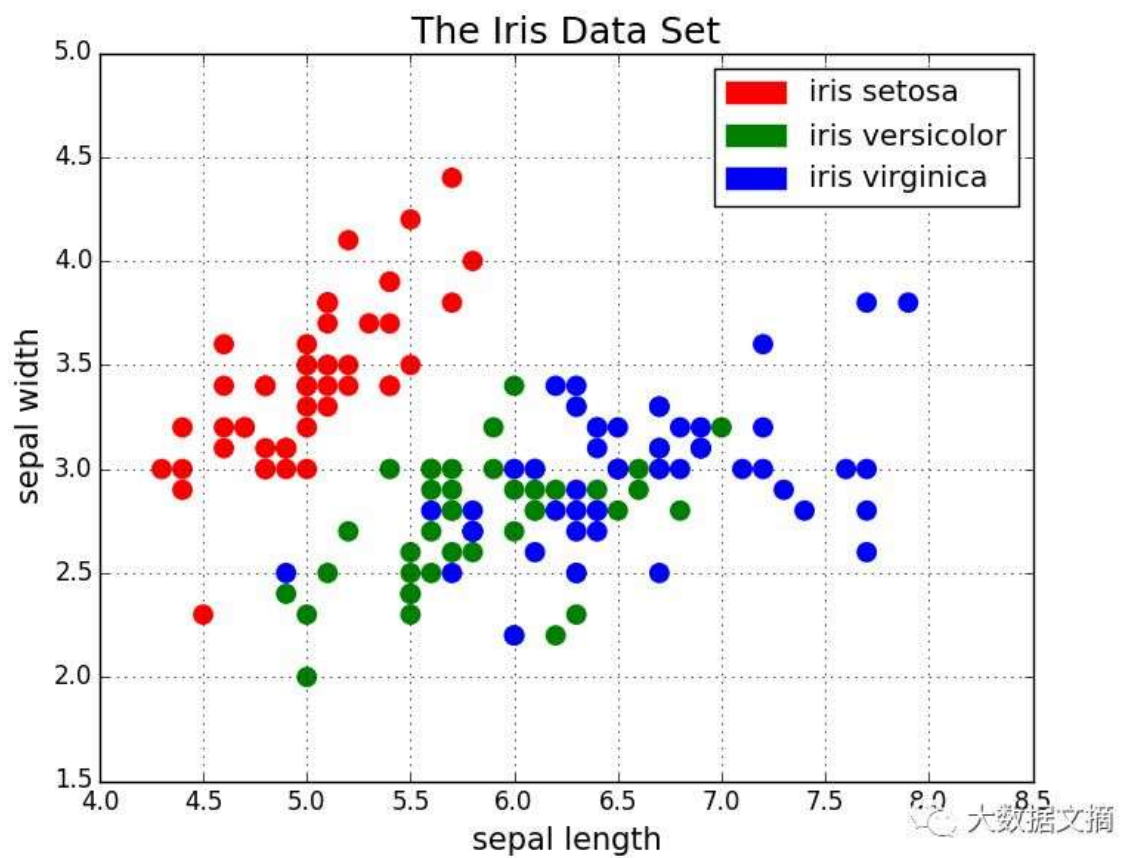
首先，请大家看看这张大的地图，它能指引你根据不同情况，选择正确的可视化方法：



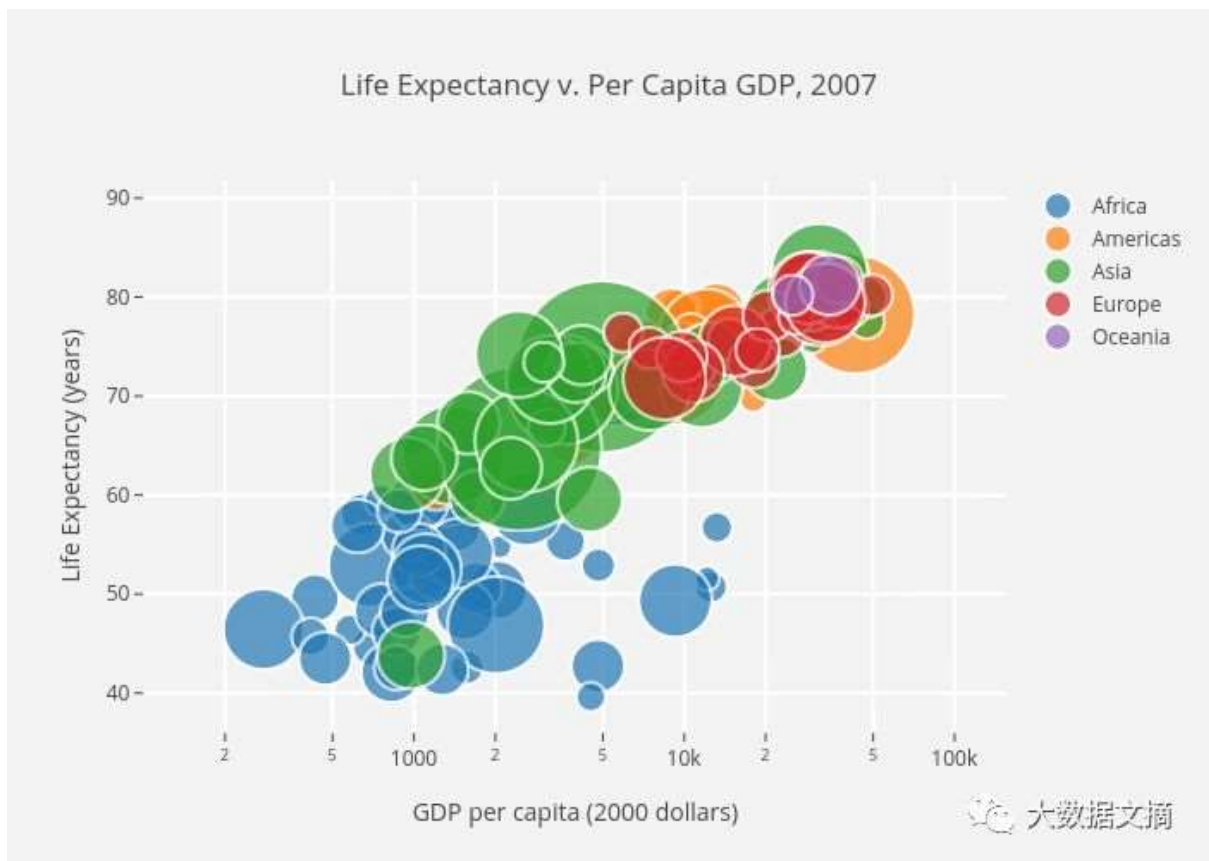
根据情况选择适当的数据可视化技术

散点图

散点图非常适合展现两个变量间关系，因为，图中可以直接看出数据的原始分布。还可以通过设置不同的颜色，轻松地查看不同组数据间的关系，如下图所示。那如果想要可视化三个变量之间的关系呢？没问题！只需再添加一个参数（如点的大小）来表示第三个变量就可以了，如下面第二个图所示。



以颜色分组的散点图



加入新维度：圆圈大小

现在来写代码。首先导入Matplotlib库的pyplot子库，并命名为plt。使用 plt.subplots()命令创建一个新的图。将x轴和y轴数据传递给相应数组x_data和y_data，然后将数组和其他参数传递给ax.scatter()以绘制散点图。我们还可以设置点的大小、颜色和alpha透明度，甚至将y轴设置成对数坐标。最后再为该图设置好必要的标题和轴标签。这个函数轻松地实现了端到端的绘图！

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

def scatterplot(x_data, y_data, x_label="", y_label="", title="", color = "r",
               yscale_log=False):

    # Create the plot object
    _, ax = plt.subplots()

    # Plot the data, set the size (s), color and transparency (alpha)
    # of the points
    ax.scatter(x_data, y_data, s = 10, color = color, alpha = 0.75)

    if yscale_log == True:
        ax.set_yscale('log')

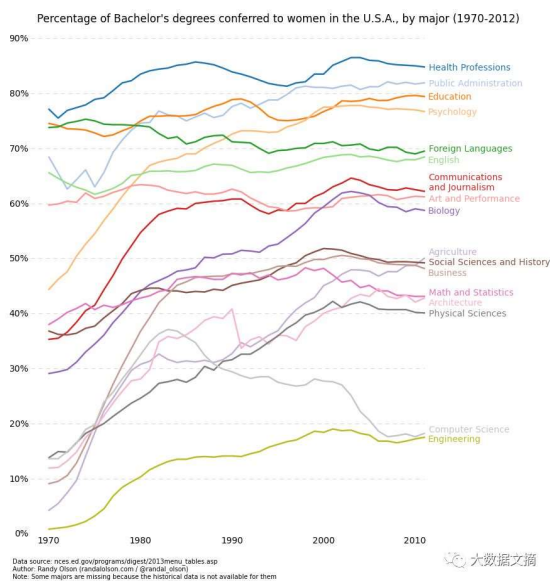
    # Label the axes and provide a title
```

```
ax.set_title(title)
ax.set_xlabel(x_label)
ax.set_ylabel(y_label)
```

折线图

如果一个变量随着另一个变量的变化而大幅度变化（具有很高的协方差），为了清楚地看出变量间的关系，最好使用折线图。例如，根据下图，我们能清楚地看出，不同专业获得学士学位的人群中，女性所占的百分比随时间变化产生很大变化。

此时，若用散点图绘制，数据点容易成簇，显得非常混乱，很难看出数据本身的意义。而折线图就再合适不过了，因为它基本上反映出两个变量（女性占比和时间）协方差的大体情况。同样，也可使用不同颜色来对多组数据分组。



女性获得学士学位的百分比（美国）

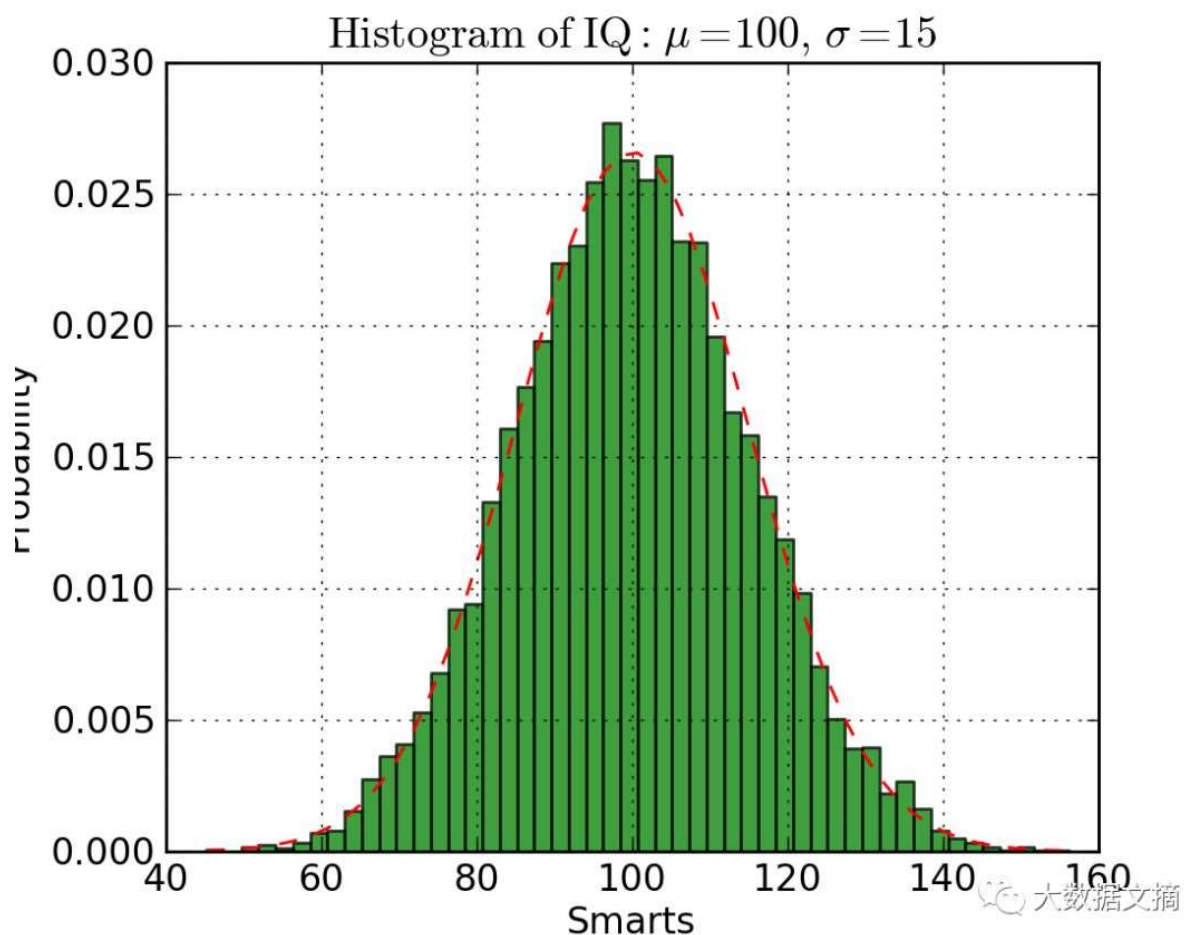
代码与散点图类似，只是一些微小的参数改动。

```
def lineplot(x_data, y_data, x_label="", y_label="", title=""):  
    # Create the plot object  
    _, ax = plt.subplots()  
  
    # Plot the best fit line, set the linewidth (lw), color and  
    # transparency (alpha) of the line  
    ax.plot(x_data, y_data, lw = 2, color = '#539caf', alpha = 1)  
  
    # Label the axes and provide a title
```

```
ax.set_title(title)
ax.set_xlabel(x_label)
ax.set_ylabel(y_label)
```

直方图

直方图适合查看（或发现）数据分布。下图为不同IQ人群所占比例的直方图。从中可以清楚地看出中心期望值和中位数，看出它遵循正态分布。使用直方图（而不是散点图）可以清楚地显示出不同组数据频率之间的相对差异。而且，分组（使数据离散化）有助于看出“更宏观的分布”，若使用未被离散化的数据点，可能会产生大量数据噪声，从而很难看出数据的真实分布。



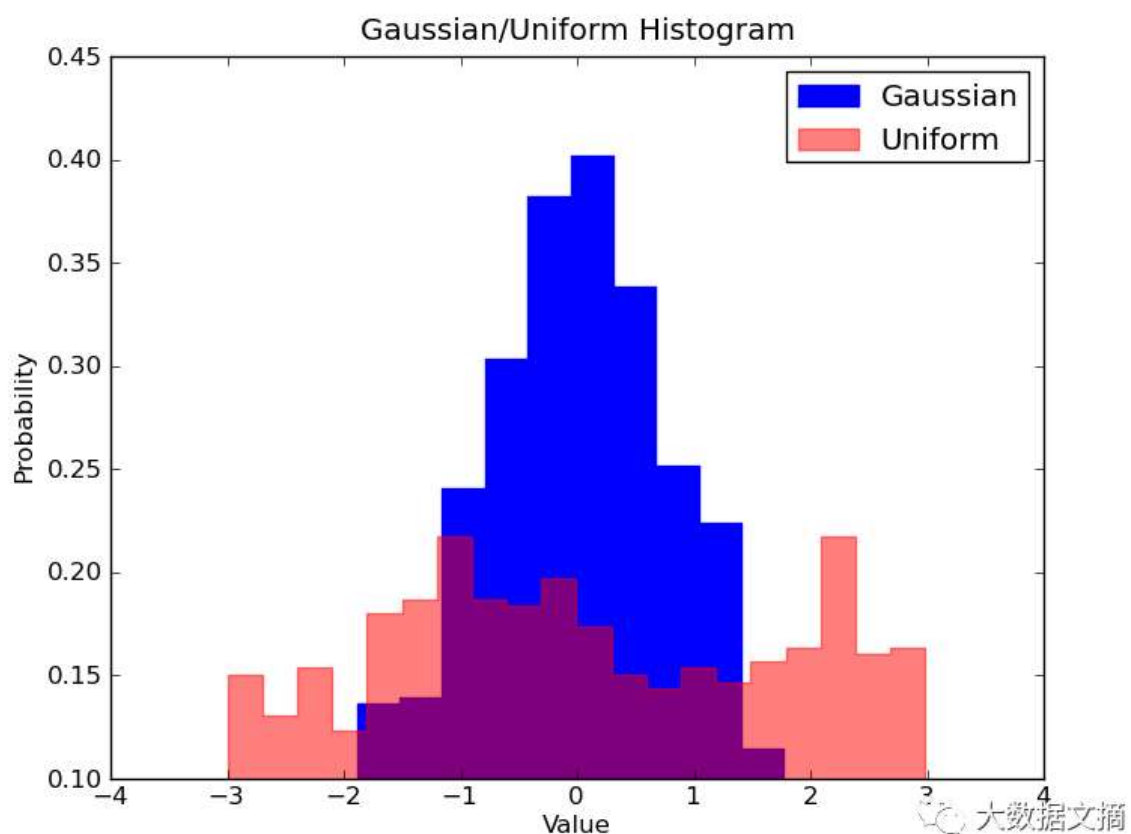
正态分布的IQ

下面是用Matplotlib库创建直方图的代码。这里有两个参数需要注意。第一个参数是`n_bins`参数，用于控制直方图的离散度。一方面，更多的分组数能提供更详细的信息，但可能会引入数据噪声使结果偏离宏观分布；另一方面，更少的分组数能提供更宏观的数据“鸟瞰”，在不需太多细节的情况下能更全面地了解数据整体情况。第二个参数是累积参数`cumulative`，

是一个布尔值，通过它控制直方图是否累积，也就是选择使用概率密度函数（PDF）还是累积密度函数（CDF）。

```
def histogram(data, n_bins, cumulative=False, x_label = "", y_label = "", title = ""):
    _, ax = plt.subplots()
    ax.hist(data, n_bins = n_bins, cumulative = cumulative, color = '#539caf')
    ax.set_ylabel(y_label)
    ax.set_xlabel(x_label)
    ax.set_title(title)
```

如果要比较数据中两个变量的分布情况该怎么办呢？有些人可能会认为，必须要制作两个独立的直方图将它们并排放在一起进行比较。但实际上，有更好的方法：用不同透明度实现直方图的叠加。比如下图，将均匀分布透明度设置为0.5，以便看清后面的正态分布。这样，用户就可以在同一张图上查看两个变量的分布了。



叠加直方图

在实现叠加直方图的代码中需要设置以下几个参数：

- 设置水平范围，以适应两种可变分布；
- 根据这个范围和期望的分组数量，计算并设置组距；
- 设置其中一个变量具有更高透明度，以便在一张图上显示两个分布。

```
# Overlay 2 histograms to compare them
def overlaid_histogram(data1, data2, n_bins = 0, data1_name="", data1_color="#539caf", data2_name="", data2_color="#7663b0", x_label="", y_label="", title=""):
    # Set the bounds for the bins so that the two distributions are fairly
    compared
    max_nbins = 10
    data_range = [min(min(data1), min(data2)), max(max(data1), max(data2))]
    binwidth = (data_range[1] - data_range[0]) / max_nbins

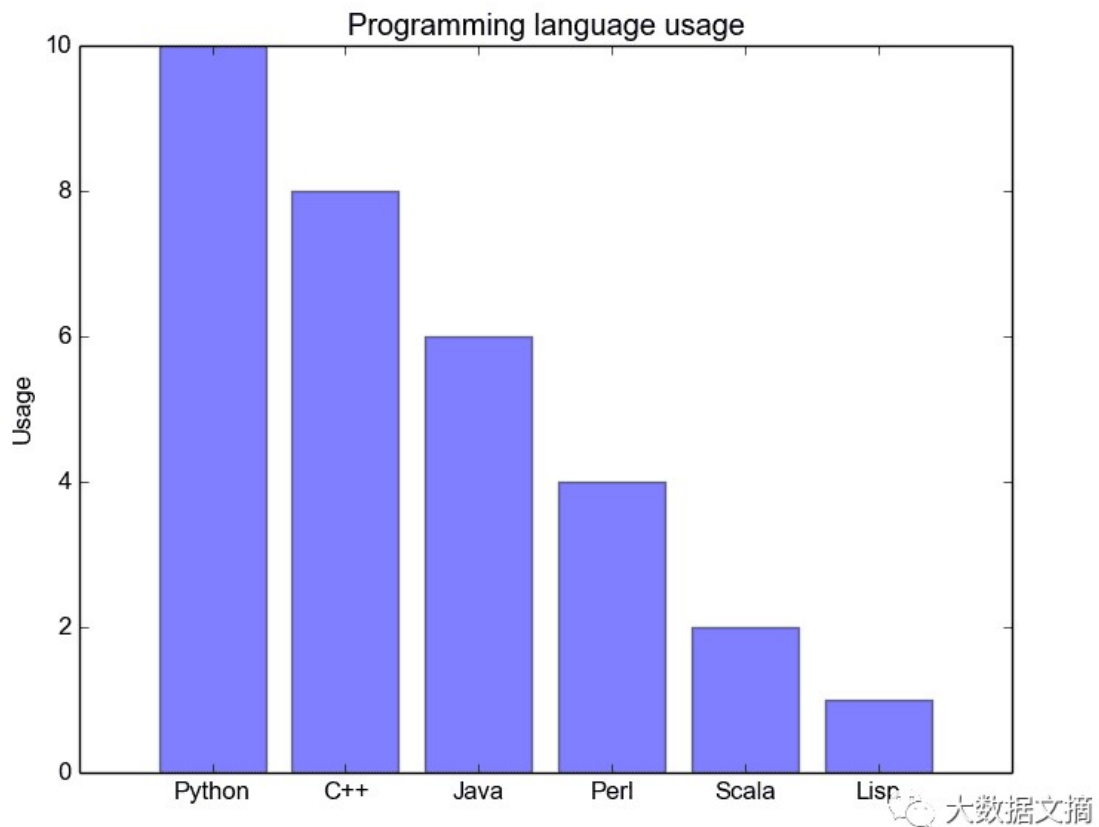
    if n_bins == 0:
        bins = np.arange(data_range[0], data_range[1] + binwidth, binwidth)
    else:
        bins = n_bins

    # Create the plot
    _, ax = plt.subplots()
    ax.hist(data1, bins = bins, color = data1_color, alpha = 1, label = data1_name)
    ax.hist(data2, bins = bins, color = data2_color, alpha = 0.75, label = data2_name)
    ax.set_ylabel(y_label)
    ax.set_xlabel(x_label)
    ax.set_title(title)
    ax.legend(loc = 'best')
```

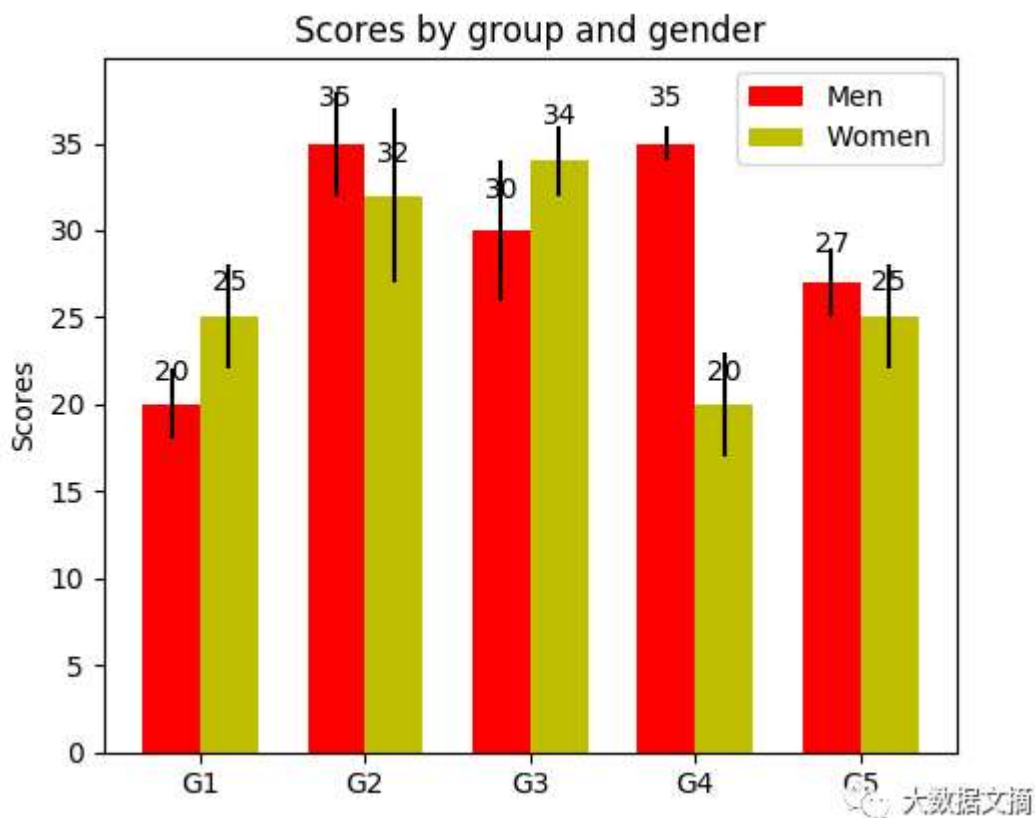
柱状图

柱状图适用于对类别较少（<10个）的分类数据进行可视化。但在类别太多时，图中的柱体就会容易堆在一起，显得非常乱，对数据的理解造成困难。柱状图适合于分类数据的原因，一是能根据柱体的高度（即长短）轻松地看出类别之间的差异，二是很容易将不同类别加以区分，甚至赋予不同颜色。以下介绍三种类型的柱状图：常规柱状图，分组柱状图和堆积柱状图。参考代码来看详细的说明。

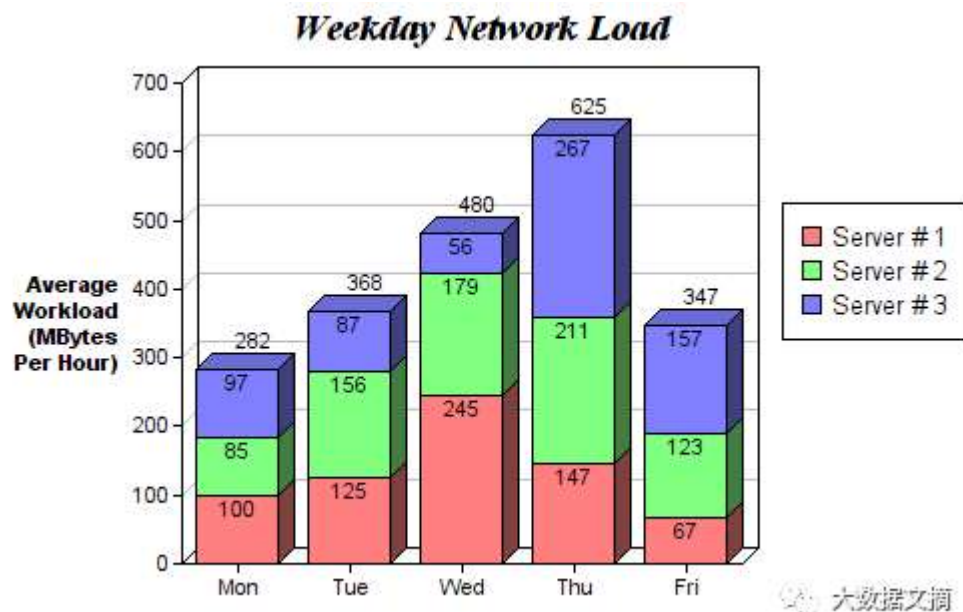
常规柱状图，如下图所示。代码中，`barplot()`函数的`x_data`参数表示x轴坐标，`y_data`代表y轴（柱体的高度）坐标，`yerr`表示在每个柱体顶部中央显示的标准偏差线。



分组柱状图，如下图所示。它允许对多个分类变量进行对比。如图所示，两组关系其一是分数与组（组G1，G2，...等）的关系，其二是用颜色区分的性别之间的关系。代码中，`y_data_list`是一个列表，其中又包含多个子列表，每个子列表代表一个组。对每个列表赋予x坐标，循环遍历其中的每个子列表，设置成不同颜色，绘制出分组柱状图。



堆积柱状图，适合可视化含有子分类的分类数据。下面这张图是用堆积柱状图展示的日常服务器负载情况统计。使用不同颜色进行堆叠，对不同服务器之间进行比较，从而能查看并了解每天中哪台服务器的工作效率最高，负载具体为多少。代码与柱状图样式相同，同样为循环遍历每个组，只是这次是在旧柱体基础上堆叠，而不是在其旁边绘制新柱体。



以下是三种堆积柱状图的代码：

```
def barplot(x_data, y_data, error_data, x_label="", y_label="", title=""):
    _, ax = plt.subplots()
    # Draw bars, position them in the center of the tick mark on the x-axis
    ax.bar(x_data, y_data, color = '#539caf', align = 'center')
    # Draw error bars to show standard deviation, set ls to 'none'
    # to remove line between points
    ax.errorbar(x_data, y_data, yerr = error_data, color = '#297083', ls = 'none',
lw = 2, capthick = 2)
    ax.set_ylabel(y_label)
    ax.set_xlabel(x_label)
    ax.set_title(title)
```

```
def stackedbarplot(x_data, y_data_list, colors, y_data_names="", x_label="",
y_label="", title=""):
    _, ax = plt.subplots()
    # Draw bars, one category at a time
    for i in range(0, len(y_data_list)):
        if i == 0:
            ax.bar(x_data, y_data_list[i], color = colors[i], align = 'center',
label = y_data_names[i])
```

```

    else:
        # For each category after the first, the bottom of the
        # bar will be the top of the last category
        ax.bar(x_data, y_data_list[i], color = colors[i], bottom =
y_data_list[i - 1], align = 'center', label = y_data_names[i])
    ax.set_ylabel(y_label)
    ax.set_xlabel(x_label)
    ax.set_title(title)
    ax.legend(loc = 'upper right')

def groupedbarplot(x_data, y_data_list, colors, y_data_names="", x_label="",
y_label="", title=""):
    _, ax = plt.subplots()
    # Total width for all bars at one x location
    total_width = 0.8
    # Width of each individual bar
    ind_width = total_width / len(y_data_list)
    # This centers each cluster of bars about the x tick mark
    alteration = np.arange(-(total_width/2), total_width/2, ind_width)

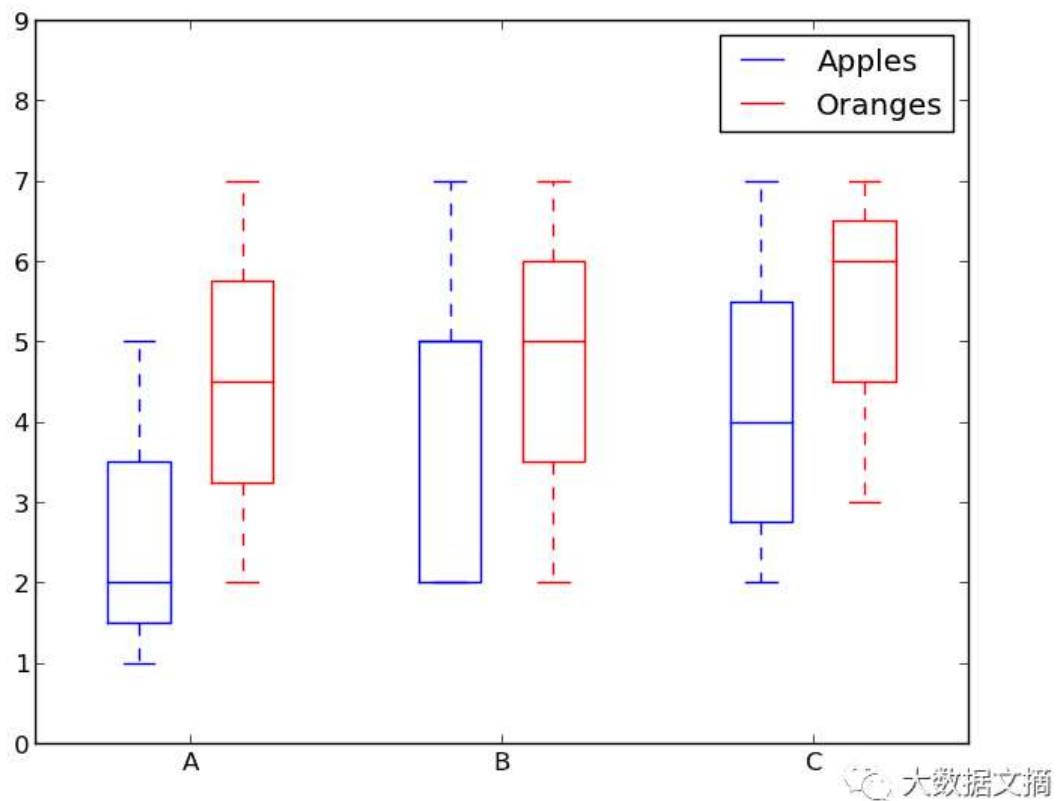
    # Draw bars, one category at a time
    for i in range(0, len(y_data_list)):
        # Move the bar to the right on the x-axis so it doesn't
        # overlap with previously drawn ones
        ax.bar(x_data + alteration[i], y_data_list[i], color = colors[i], label =
y_data_names[i], width = ind_width)
    ax.set_ylabel(y_label)
    ax.set_xlabel(x_label)
    ax.set_title(title)
    ax.legend(loc = 'upper right')

```

箱线图

前文介绍的直方图非常适合于对变量分布的可视化。但是，如果想要将更多的变量信息可视化呢？比如要清楚地看出标准差，或者一些情况下，中位数与平均值存在很大差异，因此是存在很多异常值呢还是数据分布本身就向一端偏移呢？

这里，箱线图就可以表示出上述的所有信息。箱体的底部和顶部分别为第一和第三四分位数（即数据的25%和75%），箱体内的横线为第二四分位数（即中位数）。箱体上下的延伸线（即T型虚线）表示数据的上下限。



由于箱形图是为每个组或变量绘制的，因此设置起来非常容易。x_data是组或变量的列表，x_data中的每个值对应于y_data中的一列值（一个列向量）。用Matplotlib库的函数boxplot()为y_data的每列值（每个列向量）生成一个箱形，然后设定箱线图各个参数就可以了。

```
def boxplot(x_data, y_data, base_color="#539caf", median_color="#297083",
            x_label="", y_label="", title=""):
    _, ax = plt.subplots()

    # Draw boxplots, specifying desired style
    ax.boxplot(y_data
               # patch_artist must be True to control box fill
               , patch_artist = True
               # Properties of median line
               , medianprops = {'color': median_color}
               # Properties of box
               , boxprops = {'color': base_color, 'facecolor': base_color}
               # Properties of whiskers
               , whiskerprops = {'color': base_color}
               # Properties of whisker caps
               , capprops = {'color': base_color}))

    # By default, the tick label starts at 1 and increments by 1 for
    # each box drawn. This sets the labels to the ones we want
```

```
ax.set_xticklabels(x_data)
ax.set_ylabel(y_label)
ax.set_xlabel(x_label)
ax.set_title(title)
```

这就是可供你使用的Matplotlib库的5个快速简单的数据可视化方法了！将功能和方法包装成函数，总是会使代码的编写和阅读都变的更简单！希望这篇文章能对你有所帮助，希望你能从中学到知识！如果喜欢就点个赞吧！

原文链接：

<https://towardsdatascience.com/5-quick-and-easy-data-visualizations-in-python-with-code-a2284bae952f>