9.5.3 用FontProperties直接指定字体

由于matplotlib只搜索TTF字体文件，因此无法通过上述方法使用Windows的Fonts目录下的许多复合字体文件(*.ttc)。

可以直接创建使用字体文件的FontProperties对象，并使用此对象指定图表中的各种文字的字体。

In [ ]:

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
import numpy as np

font1 = FontProperties(fname = "c:\\windows\\fonts\\simsun.ttc",
                       size = 12)
font2 = FontProperties(fname = "c:\\windows\\fonts\\STXIHEI.TTF",
                       size = 14)
font3 = FontProperties(fname = "c:\\windows\\fonts\\simkai.ttf",
                       size = 20)

sns.pointplot(x = ccss.time, y = ccss.index1, capsize = .1)
plt.xlabel("月份", fontproperties = font1)
plt.ylabel("消费者信心指数", fontproperties = font2)
plt.title("信心指数随月份变化的情况", fontproperties = font3, color = 'b')
```

9.6 实战练习

尝试解决S5职业的条图中类别标签重叠的问题，可以用修改标签值，旋转排列方向等各种方法。

分别尝试使用annotate方法和text方法为S5职业的条图中的每一个直条加绘数值标签。

尝试将一个绘图面板分割为多个网格，在每个网格中分别显示不同的样式/配色/字体/格式设置等。

> 提示：如果可能，绘图操作请尽量用循环方式完成。

10 特殊统计图的绘制

本章的主要学习目的是：通过学习各类特殊统计图的绘制，扩展绘图思路，掌握对matplotlib的各种灵活使用方法。

对于已提供了现成模块，直接调用就可以绘制的一些罕用图形，本章将不加以介绍。

> 风玫瑰图：windrose模块
> 桑基图：sankey模块
> 量场图/箭头图：quiver模块

10.1 P-P图和Q-Q图

10.1.1 P-P图

P-P图是将变量的实际分布累积概率与所指定的理论分布累积概率分别作为横、纵坐标而绘制的散点图，用于直观地检测样本数据是否符合某一概率分布。

如果被检验的数据符合所指定的分布，则代表样本数据的点应当基本在代表理论分布的对角线上。

python没有直接提供绘制P-P图的功能，但图形本身并不复杂，因此绘制的关键点在于如何正确计算出实际/理论分布累积概率。

In [ ]:

```
# 对数据进行排序，方便后续计算
ccss.s3.sort_values()
```

In [ ]:

```
# 将数据转换为数据框便于处理
df1 = pd.DataFrame(ccss.s3.sort_values())
df1.head()
```

In [ ]:

```
from scipy import stats

# 使用kde函数计算概率密度（最终是图形观察，因此不需要很精确，默认参数即可）
df1['probd'] = stats.gaussian_kde(df1.s3)(df1.s3)
df1.head()
```

In [ ]:

```
# 去除重复值，以便进一步计算累积概率
df1 = df1[~df1.duplicated()]
# 重建索引以便后续使用
df1.reset_index(drop = True, inplace = True)
df1.head()
```

In [ ]:

```
# 需要考虑是否对小的误差进行处理
df1.probd.sum()
```

In [ ]:

```
df1['cprob'] = df1.probd.cumsum() # 如果希望严格0~1，则 / df1.probd.sum()
df1.tail()
```

In [ ]:

```
# 计算基于正态分布假设的理论累积概率分布
df1['cprob0'] = stats.norm.cdf(df1.s3,
                               loc = df1.s3.mean(),
                               scale = df1.s3.std()
                              )
df1.head()
```

In [ ]:

```
# 完全使用pyplot实现（效果稍差）
plt.scatter(df1.cprob, df1.cprob0)
plt.plot([0,1], [0,1])
```

In [ ]:

```
# 使用seaborn中的功能拟合理论直线
sns.regplot(df1.cprob, df1.cprob0, ci = None)
```

进一步实现去除趋势的P-P图。

In [ ]:

```
# 近似方法
df1['min0'] = df1.cprob - df1.cprob0
df1.head()
```

In [ ]:

```
plt.scatter(df1.index, df1.min0)
plt.plot([0,df1.index.max()], [0,0])
```

10.1.2 Q-Q图

Q代表分位数，将两个概率分布同一个位置的分位数值作为平面坐标进行绘图，该分位值就构成平面中的一个散点，而全体分位数的比较结果就可以构成一张图形。

如果两个分布相似，则该Q-Q图中的所有散点将会趋近于落在 y = x 这一主对角线上。因此Q-Q图也可以用来直观的评估样本数据是否符合假定的理论分布，如对数据进行正态分布的考察。

> Q-Q图只是直观观察，更精确的考察需要使用假设检验。

scipy.stats.probplot(

```
x : 用于绘图的数据，array_like
sparams : 和分布相关的形状参数，tuple, optional
dist = 'norm' : 希望拟合的分布/分布函数名称
    str or stats.distributions instance, optional
fit = True : 是否使用最小二乘法对数据拟合回归线
plot = None : 绘图结果输出的具体对象，如matplotlib.pyplot模块或Axes对象
```

)

注意：probplot()不会自动输出图形，执行后需要使用plt.show()或者 plt.savefig()命令显示/保存图形。

In [ ]:

```
from scipy import stats
stats.probplot(ccss.s3, plot = plt)
```

In [ ]:

```
stats.probplot(ccss.index1, plot = plt)
```

10.2 Pareto图

Pareto图本质上就是原始指标的条图和累积百分比的线图的组合，因此绘制关键点在于：

计算出相应的累积百分比。
将相应的两个图形用双轴图的形式组合起来。

In [ ]:

```
data = ccss.s5.value_counts()
data
```

In [ ]:

```
prob = data.cumsum() / data.sum()
prob
```

In [ ]:

```
data.plot.bar(color = 'b')
```

In [ ]:

```
data.plot.bar(color = 'b')
prob.plot(color = 'g', secondary_y = True, style = '-o', linewidth = 2)
```

10.3 人口金字塔

人口金字塔本质上是两个横向条图的拼接，其绘制关键点在于：

计算出相应的人口构成比，并且根据绘图需求变换为相反数。
为图形增加合适的注解/标签。

In [ ]:

```
ccss['s3cls'] = pd.cut(ccss.s3,
                       bins = [15,20,25,30,35,40,45,50,55,60,65])
ccss.s3cls.head()
```

In [ ]:

```
data = pd.crosstab(ccss.s3cls, ccss.s2, normalize = 'columns')
data
```

In [ ]:

```
data['男'] = - data.男
data
```

In [ ]:

```
# 用拼图的方式绘制人口金字塔
data.男.plot.barh(color = 'r', width = 0.9)
data.女.plot.barh(color = 'b', width = 0.9)
```

In [ ]:

```
# 用拼图的方式绘制人口金字塔
data.男.plot.barh(color = 'r', width = 0.9)
data.女.plot.barh(color = 'b', width = 0.9)

# 修改x轴标签显示以美化图形
plt.xticks([-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3],
           ['0.3', '0.2', '0.1', '0', '0.1', '0.2', '0.3'])

# 在恰当的位置增加数值标签
for i in range(len(data)):
    plt.text(data.女[i] + 0.02, i, '%.2f' % data.女[i],
             ha='center', va='center')
    plt.text(data.男[i] - 0.02, i, '%.2f' % - data.男[i],
             ha='center', va='center')
```

10.4 雷达图

雷达图属于极坐标图的范畴，因此使用极坐标加以绘制即可，其关键点如下：

按照绘图需求生成各散点所对应的弧度数据。
将数据进行收尾拼接，以保证折线绘图时成为闭环。
将这些数据按照对应的顺序正确排列。

In [ ]:

```
# 生成所需的数据
data = ccss.s5.value_counts(sort = False, normalize = True)
data
```

In [ ]:

```
import numpy as np

# 生成绘图用弧度数据，注意要跳开终点数值
angles = np.linspace(0, 2 * np.pi, len(data), endpoint = False)
angles
```

In [ ]:

```
# 添加终点值，使曲线坐标闭合
data1 = np.concatenate((data, [data[0]])) # 闭合
angles = np.concatenate((angles, [angles[0]])) # 闭合
angles
```

In [ ]:

```
# 默认参数时的绘图效果
plt.plot(angles, data1)
```

In [ ]:

```
# 使用极坐标方式绘图
fig = plt.figure()
ax = fig.add_subplot(111, polar = True) # 使用polar参数
ax.plot(angles, data1)# 画线
```

In [ ]:

```
# 使用极坐标方式绘图
fig = plt.figure()
ax = fig.add_subplot(111, polar = True) # 使用polar参数
ax.plot(angles, data1)# 画线
# 在适当的位置显示标签
ax.set_thetagrids(angles * 180 / np.pi, data.index)
```

10.5 多层饼图/圆环图

多层饼图/圆环图的绘制关键点在于采用适当的圆环宽度，使得整个图形看起来比较协调。同时注意不同层圆环的色彩搭配方式不能造成冲突或误解。

In [ ]:

```
# 生成绘图用数据
data = pd.crosstab(ccss.s2, ccss.s5)
data
```

In [ ]:

```
# 设定圆环宽度
size = 0.3
# 绘制最外层圆环，加绘边框色使得扇区更清晰
plt.pie(data.iloc[0,:], radius = 1,
        wedgeprops = dict(width = size, edgecolor='w'))
```

In [ ]:

```
plt.figure(figsize = (5, 5))
# 绘制最外层圆环，并加上标签
plt.pie(data.iloc[0,:], radius = 1,
        wedgeprops = dict(width = size, edgecolor='w'),
        labels = data.columns)
# 绘制内层圆环
plt.pie(data.iloc[1,:], radius = 1 - size,
        wedgeprops = dict(width = size, edgecolor = 'w'))
```

显然，配色不当可能引起对图形信息的误解，因此需要进一步控制圆环配色为相同方式。

In [ ]:

```
# 由于类别数较多，考虑使用等分的HUSL配色方案以免重色
# 在matplotlib中使用，需要将配色方案直接转换为数值映射
colorlist = sns.color_palette("husl", n_colors = len(data.columns))
colorlist
```

In [ ]:

```
plt.figure(figsize = (5, 5))
# 绘制最外层圆环
plt.pie(data.iloc[0,:], radius = 1,
        wedgeprops = dict(width = size, edgecolor='w'),
        colors = colorlist, labels = data.columns)
# 绘制内层圆环
plt.pie(data.iloc[1,:], radius = 1 - size,
        wedgeprops = dict(width = size, edgecolor = 'w'),
        colors = colorlist)
```

10.6 热图

seaborn中的热图在本质上就是把已经整理好的矩阵数据显示为图形，因此所有的数据准备工作必须要事前完成，包括行/列可能需要进行的排序、标签设定等。

相应的操作如果使用Pandas中的数据框整理功能来完成应当更加容易。

seaborn.heatmap(

data : rectangular dataset，用于绘制热图的数据框/ndarray
如果提供Pandas数据框，则行/列名称将用于绘图，成为行/列标签

调色板：
vmin, vmax : floats, colormap的取值范围
cmap : matplotlib colormap名称
center : float, colormap的中间值
robust : bool, 当vmin或vmax缺省时，是否使用分位数而不是极值来更稳健的估计colorma
p range

注释与标签：
annot : bool or rectangular dataset, 是否在单元格中显示数值
fmt = '.2g' : string, 添加注释时的格式
annot_kws : dict of key, value mappings, optional
xticklabels, yticklabels = 'auto' : 行/列标签的设定方式
"auto", bool, list-like, or int, optional

框线与色带：
linewidths = 0 : float, 单元格框线宽度
linecolor = 'white' : color, 单元格框线颜色

cbar = True : boolean, 是否绘制色带
cbar_kws : dict of key, Keyword arguments for fig.colorbar.
cbar_ax : 用于绘制色带的Axes对象名称，否则将在主Axes对象中绘制

square : boolean, 是否强制设定每个单元格为正方形

)

In [ ]:

```
# 汇总出所需的数据
data = ccss.pivot_table(index = 's9', columns = 's5', values = 'index1')
data.head()
```

In [ ]:

```
sns.heatmap(data)
```

In [ ]:

```
sns.heatmap(data, square = True)
```

In [ ]:

```
# 设置图形大小和颜色配置
plt.figure(figsize=(10, 6))
sns.heatmap(data, cmap = 'YlGnBu')
```

In [ ]:

```
# 在图中标注数值
plt.figure(figsize=(10, 6))
sns.heatmap(data, cmap = 'YlGnBu', annot = True, fmt = '.3g')
```

In [ ]:

```
# 重新整理数据用于排序
data2 = ccss.pivot_table(index = 's9', columns = 's5',
                         values = 'index1', margins = True)
data2
```

In [ ]:

```
# 去掉不需要的汇总列
data2 = data2.iloc[:, :-1]
data2
```

In [ ]:

```
# 对数据进行排序
data2.sort_values(by = 'All', axis = 'columns', inplace = True)
data2
```

In [ ]:

```
plt.figure(figsize=(10, 6))
sns.heatmap(data2.iloc[:-1,:], cmap = 'YlGnBu',
            annot = True, fmt = '.3g')
```

10.7 实战练习

请尝试使用matplotlib或者seaborn中的绘图函数实现人口金字塔。

提示：重点在于如何完美的拼接两侧的条图。

请尝试实现分城市比较分学历消费者信心指数均值的多重雷达图。

请实现受教育程度和职业交叉情况下消费者信心指数均值的热图，并且将单元格做行、列排序，将热图左上角设定为均值高的区域，右下角设定为均值低的区域。

11 自由绘图

11.1 区域填充

在指定区域之间上色填充属于基本的绘图功能，在matplotlib中则可以借此实现其他方式难于绘制的一些特殊效果/图形。

11.1.1 填充两个水平曲线间的区域

matplotlib.pyplot.fill_between(

```
    x : 曲线的x坐标，array (length N)
    y1 : 第一条曲线的y坐标，array (length N)
    y2 = 0 : 第二条曲线的y坐标，array (length N)
    where = None : 绘制时是否包括所对应的区域，array of bool (length N)
    interpolate = False : 当使用where参数并且两曲线有交叉时有效
        当两条曲线比较接近时，要求计算精确的交叉位置并加以绘制
    step : 当各x的数值之间y应当是常数时，确定具体的常数取值方式
        "pre": y值恒等于左侧的上一个x,y数据对应的y值
        "post": y值恒等于右侧的下一个x,y数据对应的y值
        "mid": 取两侧y值的平均
```

)

In [ ]:

```
# 一个简单的填充示例，填充和横轴间的空间
plt.fill_between([1,2,3,4,5], [1,2,1,2,1])
```

In [ ]:

```
# 一个简单的填充示例，填充和横轴间的空间
plt.fill_between([1,2,3,4,5], [1,2,1,2,1],
                 where = [False, True, True, True, False])
```

In [ ]:

```
# 一个简单的填充示例
plt.fill_between([1,2,3,4,5], [1,2,1,2,1], [2,1,2,1,2], color = 'g')
```

In [ ]:

```
# 组合填充
plt.fill_between([1,2,3,4,5], [1,2,1,2,1], [2,1,2,1,2], color = 'g')
plt.fill_between([1,2,3,4,5], [2,1,2,1,2], color = 'b')
```