In [ ]:

```
# 交互式定制离散色板
sns.choose_diverging_palette()
```

8.7 实战练习

请自行判断下列数据可视化需求应当使用哪种类型的色板进行修饰：

比较不同职业的人群其当前家庭经济状况信心值（QA3）的差异，该数值以100为中值，上下波动范围为0~200。
比较不同职业的人群其受教育状况的构成比有怎样的差异。
比较不同职业的人群其家庭平均收入有怎样的差异。

尝试使用choose_light_palette函数配制自定义的连续色板，然后利用适当的函数在绘图中使用该色板。

9 统计图的进一步美化与修饰

9.1 专业图表的风格应当是怎样的?

9.2 Seaborn的样式管理

Seaborn 将 matplotlib 的参数划分为两个独立的组合：

第一组是设置绘图Axes对象的整体外观风格
    axes_style()
    set_style()
第二组主要设定图中各种图形元素的大小
    plotting_context()
    set_context()

每对方法中的第一个方法（axes_style(), plotting_context()）会返回一组字典参数。

第二个方法（set_style(), set_context()）会设置matplotlib中对应的默认参数。

但是用户只能通过这个方法来覆盖matplotlib的风格定义中的部分参数。

9.2.1 设定绘图外观风格

axes_style()返回的相关参数设置会影响Axes对象颜色设定、网格设定和其他视觉设定。

直接使用预设主题模板

seaborn.set_style()中提供了一些预设主题模板：

暗网格(darkgrid)
白网格(whitegrid)
全黑(dark)
全白(white)
全刻度(ticks)

In [ ]:

```
sns.set()
plt.plot([1,2,2,1])
```

In [ ]:

```
# 更改图形模板
sns.set_style("white")
plt.plot([1,2,2,1])
```

In [ ]:

```
# 更改图形模板
sns.set_style("whitegrid")
plt.plot([1,2,2,1])
```

修改具体外观参数

根据字典的具体内容，用户可以针对其中的某些具体参数进行修改。

In [ ]:

```
# 查看Axes对象整体外观风格的现有参数设定
sns.axes_style()
```

In [ ]:

```
# 直接修改个别参数
sns.set_style({'ytick.left': True, 'xtick.top': True})
plt.plot([1,2])
```

在预设模板的基础上，也可以进一步对其中的具体参数进行单独修改。

In [ ]:

```
# 在默认主题的基础上进行修改
sns.set_style("white", {"axes.facecolor": "c"})
plt.plot([1,2,2,1])
```

9.2.2 设定图中元素大小

In [ ]:

```
# 查看给图元素大小现有设定
sns.plotting_context()
```

In [ ]:

```
sns.set_style("ticks")
sns.set_context({'xtick.labelsize' : 20, 'ytick.major.size': 16})
plt.plot([1,2])
```

图形元素大小的四种预设格式

paper ： 纸质出版物常用大小\
notebook ： 电脑屏幕常用大小，默认值\
talk ： 公众演讲时常用大小\
poster ： 展板常用大小\

In [ ]:

```
sns.plotting_context('paper')
```

In [ ]:

```
sns.set_context('paper')
plt.plot([1,2])
```

In [ ]:

```
# 重新载入默认样式定义
sns.set()
plt.plot([1,2])
```

9.2.3 设置调色板

seaborn.color_palette()用于显示预设的几种调色板设定

默认的分类色板设置：deep, muted, bright, pastel, dark, colorblind\
无参数时返回当前调色板设定列表

seaborn.set_palette()则可以用来加载预设的几种调色板设定

In [ ]:

```
sns.color_palette()
```

In [ ]:

```
sns.palplot(sns.color_palette('bright'))
```

In [ ]:

```
sns.set_palette('bright')
plt.plot([1,2])
```

In [ ]:

```
sns.set_palette('dark')
plt.plot([1,2])
```

9.2.4 打包加载综合设定

9.2.4.1 直接改变系统设定

seaborn.set(

```
    context = 'notebook' : 设定plotting_context()的有关参数
    style = 'darkgrid' : 设定预设的主题，即axes_style()的有关参数
    palette = 'deep' : 即color_palette()的有关参数
    font = 'sans-serif' : 设定希望使用的字体
    font_scale = 1 : 设定对字体的附加放大倍数
    color_codes = True : 为True时，使用seaborn的调色板字符设定
```

)

In [ ]:

```
sns.set(context = 'talk', style = 'whitegrid', palette = 'bright')
plt.plot([1,2])
```

In [ ]:

```
sns.set(context = 'paper', style = 'white', palette = 'dark')
plt.plot([1,2])
```

In [ ]:

```
# 当不提供任何参数时，直接加载默认设定
sns.set()
plt.plot([1,2])
```

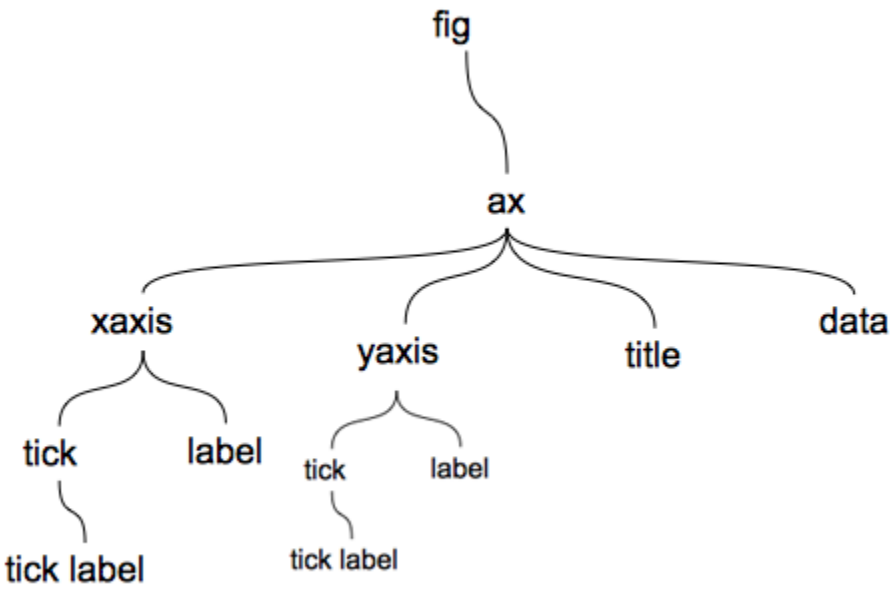
9.2.4.2 临时设置绘图风格

可以在一个with语句中使用axes_style()方法来临时的设置绘图参数。相应的设定只在with范围内有效，这将允许用不同风格的轴来绘图。

In [ ]:

```
def sinplot(flip=1):
    x = np.linspace(0,14,100)
    for i in range(1,7):
        plt.plot(x,np.sin(x+i*.5)*(7-i)*flip)

with sns.axes_style("darkgrid"):
    plt.subplot(211)
    sinplot()

sns.set_style("whitegrid")
plt.subplot(212)
sinplot()
```

9.3 设置坐标轴



Spines：可以直接理解为边框（外框），是用于标记数据区域的边界线，共有上、下、左、右四个。

这四个边框都可以直接被设定为坐标轴加以使用。

完整参考信息：https://matplotlib.org/api/spines_api.html (https://matplotlib.org/api/spines_api.html)

9.3.1 设置坐标轴的样式

设置边框颜色：ax.spines['right'].set_color('blue')

设置边框线宽：ax.spines['left'].set_linewidth(5)

设置边框线型：ax.spines['left'].set_linestyle('--')

In [ ]:

```
# 设置边框颜色
plt.plot([1,2,2,1])
plt.gca().spines['left'].set_color('y')
plt.gca().spines['right'].set_color('g')
plt.gca().spines['bottom'].set_color('c')
plt.gca().spines['top'].set_color('b')
```

In [ ]:

```
# 设置边框线宽和线形
plt.plot([1,2,2,1])
ax1 = plt.gca()
ax1.spines['left'].set_lw(2.5)
ax1.spines['left'].set_ls(':')
ax1.spines['bottom'].set_lw('2.5')
ax1.spines['bottom'].set_ls('--')
ax1.spines['right'].set_color('w')
ax1.spines['top'].set_color('none')
```

9.3.2 设置坐标轴标签

matplotlib.pyplot.xlabel/ylabel(

```
label : 标签文本。
fontdict : 标签文本的格式设定。
labelpad : 标签和数轴的间距。
```

)

In [ ]:

```
plt.xlabel('$label$', fontdict = {'family': 'serif',
        'color' : 'red',
        'weight' : 'normal',
        'size' : 16
        })
```

9.3.3 设置坐标轴刻度范围

matplotlib.pyplot.xlim() : 设置/获取当前Axes对象的x轴范围设定

```
xmin, xmax = xlim()  # return the current xlim
xlim((xmin, xmax))   # set the xlim to xmin, xmax
xlim(xmin, xmax)     # set the xlim to xmin, xmax
```

matplotlib.pyplot.ylim() : 设置/获取当前Axes对象的y轴范围设定

In [ ]:

```
xmin, xmax = plt.xlim()
print(xmin, xmax)
plt.xlim(0, 0.5)
```

9.3.4 设置刻度线和刻度标签

XTick/YTick : 对横轴/纵轴的刻度标记进行各种格式设定。

刻度线可不等距设定。
可使用自定义标签替换默认的刻度数值进行显示。
完整参考信息: https://matplotlib.org/api/axis_api.html

设置刻度格式

matplotlib.pyplot.xticks/yticks() : 设置/获取当前Axes对象的数轴刻度/标签设定

```
locs, labels = xticks()            # Get locations and labels
xticks(locs, [labels], **kwargs)   # Set locations and labels
```

In [ ]:

```
plt.xticks(rotation = 'vertical') # 使用了matplotlib.text类的方法
plt.yticks([-2, -1.5, -1, 1.5, 3],
           ['really bad', 'bad', 'normal', 'good', 'really good'],
           color = 'green')
```

In [ ]:

```
# 使用转义字符串进行格式化
# 可进一步使用LeTex转义字符（注意和中文不兼容）
plt.xticks(rotation = 'vertical') # 使用了matplotlib.text类的方法
plt.yticks([-2, -1.5, -1, 1.5, 3],
           ['$really\ bad$', '$bad$', r'$normal\ \alpha\ level$',
            '$good$', '$really\ good$'],
            color = 'green')
```

设置刻度显示方向

In [ ]:

```
ax = plt.gca()

# ACCEPTS: [ 'top' | 'bottom' | 'both' | 'default' | 'none' ]
ax.xaxis.set_ticks_position('top') # 选择下方的边界用于显示刻度
ax.yaxis.set_ticks_position('right') # 选择右侧的边界用于显示刻度
```

9.3.5 移动坐标轴位置

In [ ]:

```
ax = plt.gca()

# the 1st is in 'outward' | 'axes' | 'data'
ax.spines['bottom'].set_position(('data', 0.5)) # 下方边界交叉于0.5位置
ax.spines['left'].set_position(('data', 0.5)) # 左侧边界交叉于0.5位置
```

In [ ]:

```
# 来点复杂的图形
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(-3, 3, 100)

x = 16*(np.sin(t))**3
y = 13*np.cos(t)-5*np.cos(2*t)-2*np.cos(3*t)-np.cos(4*t)

a = plt.figure(figsize = (5, 5))
plt.plot(x, y)

ax = plt.gca()

ax.spines['bottom'].set_position(('data', 0))
ax.spines['left'].set_position(('data',0))

ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
```

9.3.6 移除坐标轴/外框线

外框线在matplotlib的默认绘图格式，以及很多seaborn的绘图样式设定中默认均显示。

dark和darkgrid两个风格默认不显示顶部和右侧的外框线。

移除外框线的做法：

将相应外框线的颜色设定为'none'。
使用seaborn.despine()方法来进一步控制外框线的位置，甚至移除它们。

seaborn.despine(

```
fig = None, ax = None
top = True, right = True, left = False, bottom = False
    是否移除对应位置的spine（框架线）
offset = None : int or dict, 希望将框线移动的绝对距离
trim = False : 是否将框线限制在数轴的最大和最小值范围之内
```

)

In [ ]:

```
sns.set_style('white')
sns.distplot(ccss.s3)
```

In [ ]:

```
sns.distplot(ccss.s3)
sns.despine()
```

In [ ]:

```
# 有针对性的移除某些框线
sns.distplot(ccss.s3)
sns.despine(bottom = True)
```

可以用offset参数对框线进行挪动，并使用trim参数限制显示范围。

In [ ]:

```
sns.violinplot(ccss.s3)
sns.despine(offset = 10)
```

In [ ]:

```
sns.violinplot(ccss.s3)
sns.despine(offset = 10, trim = True)
```

In [ ]:

```
# offset可以设置为负值
sns.violinplot(ccss.s3)
sns.despine(offset = -20)
```

In [ ]:

```
sns.violinplot(ccss.s3)
sns.despine(offset = {'left': -20})
```

9.4 设置注解

9.4.1 pyplot.annotate()方法

matplotlib.pyplot.annotate(

```
    s : 注解字符串。
    xy : 注解所对应的坐标点位置，(x,y)格式。
    xytext : 注解文字的起始点坐标位置，缺省时使用xy的值。
    xycoords : xy坐标位置的起始坐标基准。
        'figure points'/'figure pixels' Fig左下角起按points/pixels计算。
        'figure fraction'    Figure左下角起按百分比计算。
        'axes points'/'axes pixels'    Axes左下角起按points/pixels计算。
        'axes fraction'    Axes左下角起按百分比计算。
        'data'    default，使用数据绘图时所用的坐标系。
        'polar'    (theta,r)，极坐标格式。
    textcoords : xytext坐标位置的起始坐标基准，缺省时使用xy的值。
        'offset points'/'offset pixels'    基于xy值偏离points/pixels。
    arrowprops : dict，从xytext指向xy位置的箭头格式。
        不使用'arrowstyle'关键字时的设定方式：
            width    箭头身体宽度，points
            headwidth    箭头头部宽度，points
            headlength    头部长度，points
            shrink    箭头两端"回缩"的长度比例
            ?    matplotlib.patches.FancyArrowPatch中的其余关键字
    annotation_clip : 为True时，只有当坐标位于Axes内时才绘制注解。
```

)

In [ ]:

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

x = np.linspace(-3, 3, 50)
y = 2 * x + 1

plt.figure(figsize=(8, 5))
plt.plot(x, y)

ax = plt.gca()
sns.despine()
ax.spines['bottom'].set_position(('data', 0))
ax.spines['left'].set_position(('data', 0))

plt.plot([1, 1], [0, 3], 'k--', linewidth = 2.5)
plt.scatter([1], [3], s = 50, color = 'b')

plt.annotate('$2x+1=3$', xy=(1, 3), xycoords='data', xytext=(+30, -10),
             textcoords = 'offset points', fontsize = 16)
plt.show()
```

9.4.2 自定义连接箭头

FancyArrowPatch类

https://matplotlib.org/api/_as_gen/matplotlib.patches.FancyArrowPatch.html (https://matplotlib.org/api/_as_gen/matplotlib.patches.FancyArrowPatch.html)

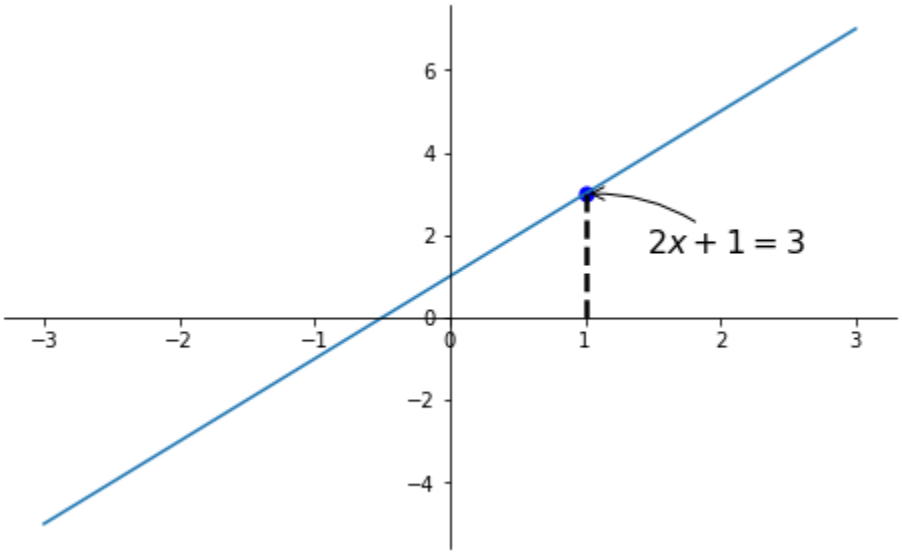
'arrowstyle'的取值列表

Name	Attrs
'-'	None
'->'	head_length=0.4,head_width=0.2
'-['	widthB=1.0,lengthB=0.2,angleB=None
'|-|'	widthA=1.0,widthB=1.0
'-|>'	head_length=0.4,head_width=0.2
'<-'	head_length=0.4,head_width=0.2
'<->'	head_length=0.4,head_width=0.2
'<|-'	head_length=0.4,head_width=0.2
'<|-|>'	head_length=0.4,head_width=0.2
'fancy'	head_length=0.4,head_width=0.4,tail_width=0.4
'simple'	head_length=0.5,head_width=0.5,tail_width=0.2
'wedge'	tail_width=0.3,shrink_factor=0.5

In [ ]:

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

x = np.linspace(-3, 3, 50)
y = 2*x + 1

plt.figure(num=1, figsize=(8, 5))
plt.plot(x, y)

ax = plt.gca()
sns.despine()
ax.spines['bottom'].set_position(('data', 0))
ax.spines['left'].set_position(('data', 0))

plt.plot([1, 1], [0, 3], 'k--', linewidth = 2.5)
plt.scatter([1], [3], s = 50, color = 'b')

plt.annotate('$2x+1=3$', xy=(1, 3), xycoords='data', xytext=(+30, -30),
             textcoords = 'offset points', fontsize = 16,
             arrowprops = dict(arrowstyle = '->',
                               connectionstyle = "arc3, rad = .2"))
plt.show()
```

arrowprops参数目前和notebook有兼容问题，在spyder中可以正常输出：



9.4.3 pyplot.text()方法

pyplot.text()方法相对要简单一些，但实际上也可以实现非常酷的效果。

目前该方法还有中文兼容问题，只能使用英文标签。

matplotlib.pyplot.text(

x, y : 文本放置的坐标位置。
s : 文本内容。
fontdict : 文本格式设定，默认使用rc parameters中的设置。

其余text类的属性：
https://matplotlib.org/api/text_api.html#matplotlib.text.Text

)

In [ ]:

```
plt.text(0.6, 0.5, "$The\ tag\ 1$", size = 50, rotation = 30,
         ha = "center", va = "center",
         bbox = dict(boxstyle = "round",
                     ec = (1, 0.5, 0.5),
                     fc = (1, 0.8, 0.8),
                     )
         )

plt.text(0.5, 0.4, "tag2", size = 50, rotation = -30.,
         ha = "right", va = "top",
         bbox = dict(boxstyle = "square",
                     ec = (0, 1, 1),
                     fc = (1, 0, 0),
                     )
         )
```

9.5 字体设定

自定义图表字体可以满足如下需求：

符合规定的制图文字规范。
让图表能正确显示中文。
在图形中使用多种字体混合显示，以达到更好的效果。

matplotlib中可以有以下几种字体设定的解决方案：

修改配置文件。
用程序修改配置字典rcParams。
在程序中直接指定字体。

9.5.1 统计图表的文字设定原则

9.5.2 在程序中直接指定字体

In [ ]:

```
# 获得所有可用的字体列表
from matplotlib.font_manager import fontManager

fonts = list(f.name for f in fontManager.ttflist)
fontname = list(f.fname for f in fontManager.ttflist)

print ('可用字体:')
for i in range(len(fonts)):
    print(fonts[i] + " : " + fontname[i])
```

In [ ]:

```
# 在程序中指定所使用的字体
from matplotlib.font_manager import fontManager
import matplotlib.pyplot as plt
import os
import os.path

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(111)
plt.subplots_adjust(0, 0, 1, 1, 0, 0)
plt.xticks([])
plt.yticks([])
x, y = 0.05, 0.08

fonts = [font.name for font in fontManager.ttflist if
    os.path.exists(font.fname) and os.stat(font.fname).st_size > 2e6]
dy = (1.0-y)/(len(fonts)/4 + (len(fonts)%4!=0))

for font in fonts:
    t = ax.text(x, y, u"+-中文效果",
                {'fontname' : font, 'fontsize' : 18},
                transform = ax.transAxes)
    ax.text(x, y - dy/2, font, transform = ax.transAxes)
    x += 0.25
    if x >= 1.0:
        y += dy
        x = 0.05
plt.show()
```

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。