## 单系列柱状图

## Matplotlib绘制单系列柱状图：不同城市的房价对比

data = pd.DataFrame({'city':['深圳', '上海', '北京', '广州', '成都'], 'house\_price(w)':[3.5, 4.0, 4.2, 2.1, 1.5]})

fig = plt.figure(figsize=(10,6))

ax1 = fig.add\_axes([0.15,0.15,0.7,0.7]) # [left, bottom, width, height], 它表示添加到画布中的矩形区域的左下角坐标(x, y)，以及宽度和高度

plt.bar(data['city'], data['house\_price(w)'], width=0.6, align='center', orientation='vertical', label='城市')

"""

x 表示x坐标，数据类型为int或float类型，也可以为str

height 表示柱状图的高度，也就是y坐标值，数据类型为int或float类型

width 表示柱状图的宽度，取值在0~1之间，默认为0.8

bottom 柱状图的起始位置，也就是y轴的起始坐标

align 柱状图的中心位置，"center","lege"边缘

color 柱状图颜色

edgecolor 边框颜色

linewidth 边框宽度

tick\_label 下标标签

log 柱状图y周使用科学计算方法，bool类型

orientation 柱状图是竖直还是水平，竖直："vertical"，水平条："horizontal"

"""

plt.title("不同城市的房价对比图") # 在axes1设置标题

plt.xlabel("城市") # 在axes1中设置x标签

plt.ylabel("房价/w") # 在axes1中设置y标签

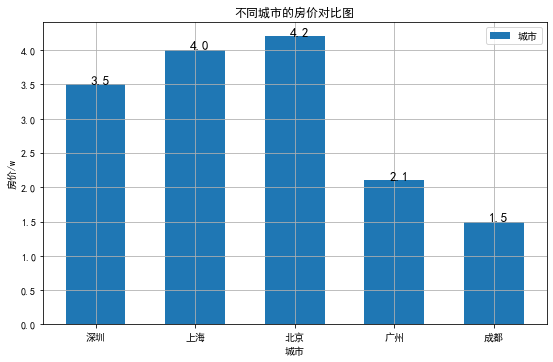
plt.grid(b=True, which='both') # 在axes1中设置设置网格线

for i in range(len(data)):

plt.text(i-0.05, data.iloc[i,]['house\_price(w)']+0.01, data.iloc[i,]['house\_price(w)'],fontsize=13) # 添加数据注释

plt.legend()

plt.show()



## 多系列柱状图

## Matplotlib绘制多系列柱状图：不同城市在不同年份的房价对比

data = pd.DataFrame({

'城市':['深圳', '上海', '北京', '广州', '成都', '深圳', '上海', '北京', '广州', '成都'],

'年份':[2021,2021,2021,2021,2021,2022,2022,2022,2022,2022],

'房价(w)':[3.5, 4.0, 4.2, 2.1, 1.5, 4.0, 4.2, 4.3, 1.6, 1.9]

})

fig = plt.figure(figsize=(10,6))

ax1 = fig.add\_axes([0.15,0.15,0.7,0.7]) # [left, bottom, width, height], 它表示添加到画布中的矩形区域的左下角坐标(x, y)，以及宽度和高度

plt.bar(

np.arange(len(np.unique(data['城市'])))-0.15,

data.loc[data['年份']==2021,'房价(w)'],

width=0.3,

align='center',

orientation='vertical',

label='年份：2021'

)

plt.bar(

np.arange(len(np.unique(data['城市'])))+0.15,

data.loc[data['年份']==2022,'房价(w)'],

width=0.3,

align='center',

orientation='vertical',

label='年份：2022'

)

plt.title("不同城市的房价对比图") # 在axes1设置标题

plt.xlabel("城市") # 在axes1中设置x标签

plt.ylabel("房价/w") # 在axes1中设置y标签

plt.xticks(np.arange(len(np.unique(data['城市']))), np.array(['深圳', '上海', '北京', '广州', '成都']))

plt.grid(b=True, which='both') # 在axes1中设置设置网格线

data\_2021 = data.loc[data['年份']==2021,:]

for i in range(len(data\_2021)):

plt.text(i-0.15-0.05, data\_2021.iloc[i,2]+0.05, data\_2021.iloc[i,2],fontsize=13) # 添加数据注释

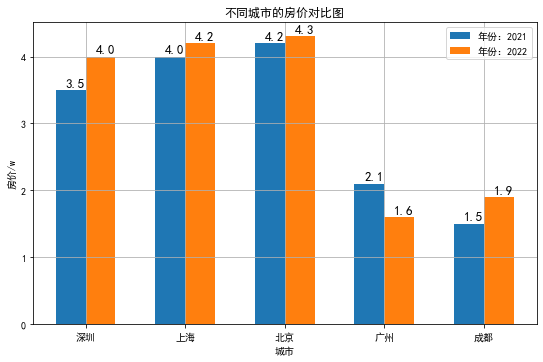
data\_2022 = data.loc[data['年份']==2022,:]

for i in range(len(data\_2022)):

plt.text(i+0.15-0.05, data\_2022.iloc[i,2]+0.05, data\_2022.iloc[i,2],fontsize=13) # 添加数据注释

plt.legend()

plt.show()



## 堆叠柱状图

## Matplotlib绘制堆叠柱状图：不同城市在不同年份的房价对比

data = pd.DataFrame({

'城市':['深圳', '上海', '北京', '广州', '成都', '深圳', '上海', '北京', '广州', '成都'],

'年份':[2021,2021,2021,2021,2021,2022,2022,2022,2022,2022],

'房价(w)':[3.5, 4.0, 4.2, 2.1, 1.5, 4.0, 4.2, 4.3, 1.6, 1.9]

})

tmp=data.set\_index(['城市','年份'])['房价(w)'].unstack()

data=tmp.rename\_axis(columns=None).reset\_index()

data.columns = ['城市','2021房价','2022房价']

print(data)

plt.figure(figsize=(10,6))

plt.bar(

data['城市'],

data['2021房价'],

width=0.6,

align='center',

orientation='vertical',

label='年份：2021'

)

plt.bar(

data['城市'],

data['2022房价'],

width=0.6,

align='center',

orientation='vertical',

bottom=data['2021房价'],

label='年份：2022'

)

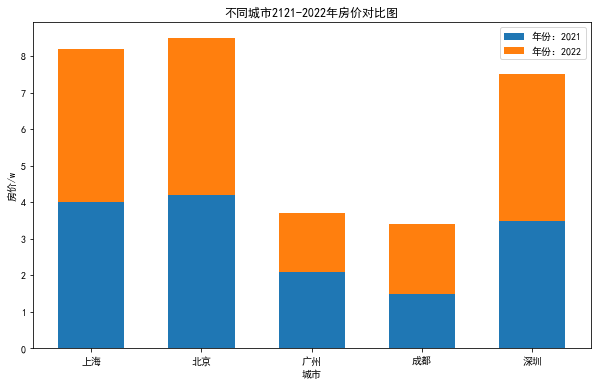
plt.title("不同城市2121-2022年房价对比图") # 在axes1设置标题

plt.xlabel("城市") # 在axes1中设置x标签

plt.ylabel("房价/w") # 在axes1中设置y标签

plt.legend()

plt.show()



## 百分比柱状图

## Matplotlib绘制百分比柱状图：不同城市在不同年份的房价对比

## Matplotlib绘制堆叠柱状图：不同城市在不同年份的房价对比

data = pd.DataFrame({

'城市':['深圳', '上海', '北京', '广州', '成都', '深圳', '上海', '北京', '广州', '成都'],

'年份':[2021,2021,2021,2021,2021,2022,2022,2022,2022,2022],

'房价(w)':[3.5, 4.0, 4.2, 2.1, 1.5, 4.0, 4.2, 4.3, 1.6, 1.9]

})

tmp=data.set\_index(['城市','年份'])['房价(w)'].unstack()

data=tmp.rename\_axis(columns=None).reset\_index()

data.columns = ['城市','2021房价','2022房价']

print(data)

plt.figure(figsize=(10,6))

plt.bar(

data['城市'],

data['2021房价']/(data['2021房价']+data['2022房价']),

width=0.4,

align='center',

orientation='vertical',

label='年份：2021'

)

plt.bar(

data['城市'],

data['2022房价']/(data['2021房价']+data['2022房价']),

width=0.4,

align='center',

orientation='vertical',

bottom=data['2021房价']/(data['2021房价']+data['2022房价']),

label='年份：2022'

)

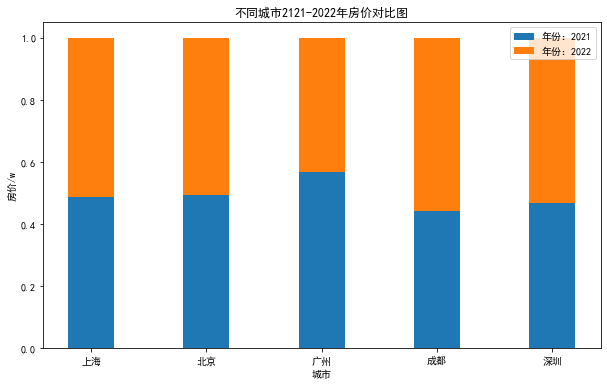
plt.title("不同城市2121-2022年房价对比图") # 设置标题

plt.xlabel("城市") # 在axes1中设置x标签

plt.ylabel("房价/w") # 在axes1中设置y标签

plt.legend()

plt.show()



## 5.火柴图：（棒棒糖图）

由于柱状图在表达数据的数值大小时使用的是不等高的长方形柱子，柱子会占据大量的绘图面积，因此当类别较多时，会出现画不下的情况。再者，柱子的宽度并没有表达什么信息，因此可以省略柱子或者将柱子替换为直线就可以节省大量的绘图空间，这样的图就是火柴图。

## 使用Matplotlib绘制火柴图（棒棒糖图）

data = pd.DataFrame({'city':['深圳', '上海', '北京', '广州', '成都'], 'house\_price(w)':[3.5, 4.0, 4.2, 2.1, 1.5]})

plt.figure(figsize=(10,6))

markerline, stemlines, baseline = plt.stem(data['city'],data['house\_price(w)'],bottom=0, label='城市')

plt.setp(markerline, color='red', marker='o',ms=8) # marker点：火柴头 ms=markersize

plt.setp(stemlines, color='#FF9900', lw=3, ls=':' ) # 火柴杆 lw=linewidth

plt.setp(baseline, color='white', linewidth=2, ls='-') # 基准线 ls=linestyle

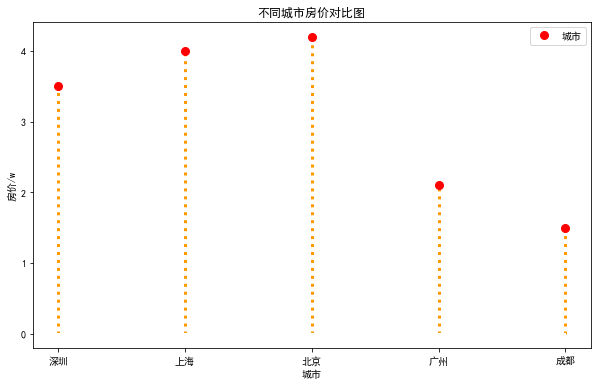
plt.title("不同城市房价对比图") # 设置标题

plt.xlabel("城市") # 在axes1中设置x标签

plt.ylabel("房价/w") # 在axes1中设置y标签

plt.legend()

plt.show()



## 6. 哑铃图：

由于火柴杆图只能展示一个纬度的数值对比情况，如果想像多系列柱状图一样对比两个或多个因素的数值变化情况，又不想像柱状图一样浪费许多绘图空间，那么哑铃图是个不错的选择。

## Matplotlib绘制哑铃图：对比不同城市2021-2022年的房价情况

data = pd.DataFrame({

'城市':['深圳', '上海', '北京', '广州', '成都', '深圳', '上海', '北京', '广州', '成都'],

'年份':[2021,2021,2021,2021,2021,2022,2022,2022,2022,2022],

'房价(w)':[3.5, 4.0, 4.2, 2.1, 1.5, 4.0, 4.2, 4.3, 1.6, 1.9]

})

tmp=data.set\_index(['城市','年份'])['房价(w)'].unstack()

data=tmp.rename\_axis(columns=None).reset\_index()

data.columns = ['城市','2021房价','2022房价']

print(data)

plt.figure(figsize=(10,6))

plt.vlines(

data.loc[data['2022房价'] - data['2021房价'] >= 0, '城市'],

ymin=data.loc[data['2022房价'] - data['2021房价'] >= 0, '2021房价'],

ymax=data.loc[data['2022房价'] - data['2021房价'] >= 0, '2022房价'],

color='red',

label='房价上涨',

zorder=1,

lw=3,

) # 绘制端点之间的连线

plt.vlines(

data.loc[data['2022房价'] - data['2021房价'] <= 0, '城市'],

ymin=data.loc[data['2022房价'] - data['2021房价'] <= 0, '2021房价'],

ymax=data.loc[data['2022房价'] - data['2021房价'] <= 0, '2022房价'],

color='yellow',

label='房价下跌',

zorder=1,

lw=3,

) # 绘制端点之间的连线

plt.scatter(x=data['城市'], y=data['2021房价'], color='#00589F', s=100, label='2021房价') # 绘制哑铃图的端点

plt.scatter(x=data['城市'], y=data['2022房价'], color='#F68F00', s=100, label='2022房价') # 绘制哑铃图的另一个端点

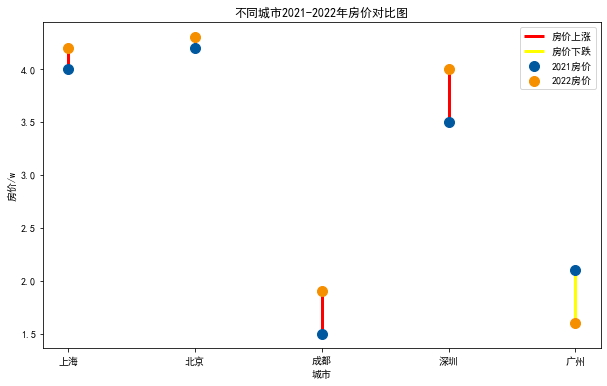
plt.title("不同城市2021-2022年房价对比图") # 设置标题

plt.xlabel("城市") # 设置x标签

plt.ylabel("房价/w") # 设置y标签

plt.legend()

plt.show()



## 7. 坡度图（感觉不是很好看）：

以上的柱状图、多系列柱状图以及火柴杆和哑铃图等等都是机遇柱状图的推广，坡度图可以看多折线图的推广。坡度图可以很好的比较各个类别在两个不同时间点或者两种不同状态下的数值数据的变化，表现的内容与哑铃图大同小异。

## Matplotlib绘制坡度图：对比不同城市2021-2022年的房价情况

data = pd.DataFrame({

'城市':['深圳', '上海', '北京', '广州', '成都', '深圳', '上海', '北京', '广州', '成都'],

'年份':[2021,2021,2021,2021,2021,2022,2022,2022,2022,2022],

'房价(w)':[3.5, 4.0, 4.2, 2.1, 1.5, 4.0, 4.2, 4.3, 1.6, 1.9]

})

tmp=data.set\_index(['城市','年份'])['房价(w)'].unstack()

data=tmp.rename\_axis(columns=None).reset\_index()

data.columns = ['城市','2021房价','2022房价']

print(data)

plt.figure(figsize=(10,6))

plt.vlines(x=[2021, 2022], ymin=0, ymax=len(data)+1, lw=3, color='gray', zorder=1)

for i in range(len(data)):

plt.plot([2021, 2022], data.iloc[i, 1:3].values, color='black', zorder=2)

plt.text(x=2021-0.4, y=data.iloc[i, 1], s=data.iloc[i, 0]+"房价: "+str(data.iloc[i, 1]))

plt.text(x=2022+0.1, y=data.iloc[i, 2], s=data.iloc[i, 0]+"房价: "+str(data.iloc[i, 2]))

plt.scatter(x=[2021]\*len(data), y=data['2021房价'], s=100, color='#00AFBB', zorder=3)

plt.scatter(x=[2022]\*len(data), y=data['2022房价'], s=100, color='#FC4E07', zorder=3)

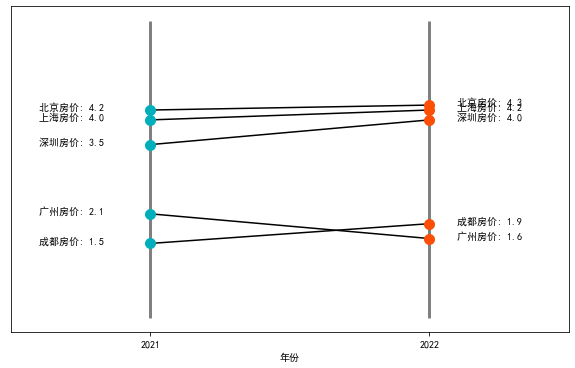
plt.xlim(2020.5,2022.5)

plt.xticks([2021,2022], ['2021', '2022'])

plt.yticks([]) # 不显示y轴

plt.xlabel("年份")

plt.show()



## 8.雷达图（可以对于origin来看效果）：

雷达图可以展示多个变量在不同属性上的数值对比，如：十个员工在职场中五种能力的分数对比、英雄联盟每个英雄在每个属性的数值分数对比等等。由于Plotnine没有实现极坐标，因此只能使用Matplotlib绘制雷达图。

# 使用Matplotlib绘制雷达图：英雄联盟几位英雄的能力对比

data = pd.DataFrame({

'属性': ['血量', '攻击力', '攻速', '物抗', '魔抗'],

'艾希':[3, 7, 8, 2, 2],

'诺手':[8, 6, 3, 6, 6]

})

plt.figure(figsize=(8,8))

theta = np.linspace(0, 2\*np.pi, len(data), endpoint=False) # 每个坐标点的位置

theta = np.append(theta, theta[0]) # 让数据封闭

aixi = np.append(data['艾希'].values,data['艾希'][0]) #让数据封闭

nuoshou = np.append(data['诺手'].values,data['诺手'][0]) # 让数据封闭

shuxing = np.append(data['属性'].values,data['属性'][0]) # 让数据封闭

plt.polar(theta, aixi, 'ro-', lw=2, label='艾希') # 画出雷达图的点和线

plt.fill(theta, aixi, facecolor='r', alpha=0.5) # 填充

plt.polar(theta, nuoshou, 'bo-', lw=2, label='诺手') # 画出雷达图的点和线

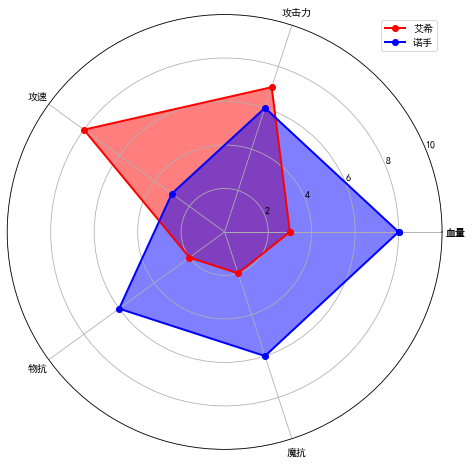
plt.fill(theta, nuoshou, facecolor='b', alpha=0.5) # 填充

plt.thetagrids(theta/(2\*np.pi)\*360, shuxing) # 为每个轴添加标签

plt.ylim(0,10)

plt.legend()

plt.show()



## 9. 散点图（描述两者关系）

x = np.random.randn(100)\*10

y1 = np.random.randn(100)\*10

y2 = 2 \* x + 1 + np.random.randn(100)

y3 = -2 \* x + 1 + np.random.randn(100)

y4 = x\*\*2 + 1 + np.random.randn(100)

plt.figure(figsize=(12, 12))

plt.subplot(2,2,1) #创建两行两列的子图，并绘制第一个子图

plt.scatter(x, y1, c='dodgerblue', marker=".", s=50)

plt.xlabel("x")

plt.ylabel("y1")

plt.title("y1与x不存在关联关系")

plt.subplot(2,2,2) #创建两行两列的子图，并绘制第二个子图

plt.scatter(x, y2, c='tomato', marker="o", s=10)

plt.xlabel("x")

plt.ylabel("y2")

plt.title("y2与x存在正相关")

plt.subplot(2,2,3) #创建两行两列的子图，并绘制第三个子图

plt.scatter(x, y3, c='magenta', marker="o", s=10)

plt.xlabel("x")

plt.ylabel("y3")

plt.title("y3与x存在负相关")

plt.subplot(2,2,4) #创建两行两列的子图，并绘制第四个子图

plt.scatter(x, y4, c='deeppink', marker="s", s=10)

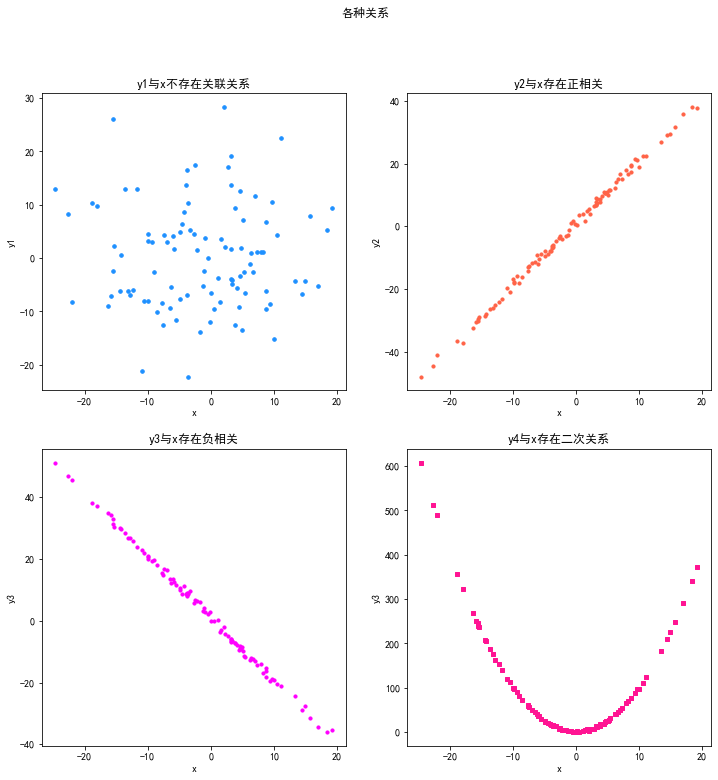
plt.xlabel("x")

plt.ylabel("y3")

plt.title("y4与x存在二次关系")

plt.suptitle('各种关系')

plt.show()

10

10. 带趋势线的散点图：

一般来说，靠肉眼观察到的趋势性规律往往不具备说服力，在统计学中能够解释趋势关系的方法是：回归。因此，如果在散点图的基础上添加回归的趋势线，将会使得我们的关系更加具备说服力。回归方法在统计学中常有：参数型回归和非参数型回归，参数回归的代表为线性回归、多项式回归、指数回归、对数回归等等，而非参数回归的代表有LOESS数据平滑方法、GAM模型、样条数据平滑方法。

## 10.带趋势线的散点图：

一般来说，靠肉眼观察到的趋势性规律往往不具备说服力，在统计学中能够解释趋势关系的方法是：回归。因此，如果在散点图的基础上添加回归的趋势线，将会使得我们的关系更加具备说服力。回归方法在统计学中常有：参数型回归和非参数型回归，参数回归的代表为线性回归、多项式回归、指数回归、对数回归等等，而非参数回归的代表有LOESS数据平滑方法、GAM模型、样条数据平滑方法。

# 使用Matplotlib绘制具备趋势线的散点图

from sklearn.linear\_model import LinearRegression #线性回归等参数回归

import statsmodels.api as sm

from sklearn.preprocessing import PolynomialFeatures # 构造多项式

x = np.linspace(-10, 10, 100)

y = np.square(x) + np.random.randn(100)\*100

x\_poly2 = PolynomialFeatures(degree=2).fit\_transform(x.reshape(-1, 1))

y\_linear\_pred = LinearRegression().fit(x.reshape(-1, 1), y).predict(x.reshape(-1, 1))

y\_poly\_pred = LinearRegression().fit(x\_poly2, y).predict(x\_poly2)

y\_exp\_pred = LinearRegression().fit(np.exp(x).reshape(-1, 1), y).predict(np.exp(x).reshape(-1, 1))

y\_loess\_pred = sm.nonparametric.lowess(x, y, frac=2/3)[:, 1] 非参数的数据拟合

plt.figure(figsize=(8, 8))

plt.subplot(2,2,1)

plt.scatter(x, y, c='tomato', marker="o", s=10)

plt.plot(x, y\_linear\_pred, c='dodgerblue')

plt.xlabel("x")

plt.ylabel("y")

plt.title("带线性趋势线的散点图")

plt.subplot(2,2,2)

plt.scatter(x, y, c='tomato', marker="o", s=10)

plt.plot(x, y\_poly\_pred, c='dodgerblue')

plt.xlabel("x")

plt.ylabel("y")

plt.title("带二次趋势线的散点图")

plt.subplot(2,2,3)

plt.scatter(x, y, c='tomato', marker="o", s=10)

plt.plot(x, y\_exp\_pred, c='dodgerblue')

plt.xlabel("x")

plt.ylabel("y")

plt.title("带指数趋势线的散点图")

plt.subplot(2,2,4)

plt.scatter(x, y, c='tomato', marker="o", s=10)

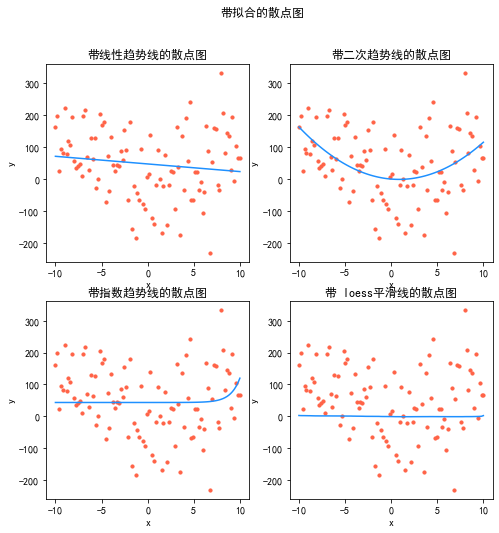
plt.plot(x, y\_loess\_pred, c='dodgerblue')

plt.xlabel("x")

plt.ylabel("y")

plt.title("带 loess平滑线的散点图")

plt.show()



## 11 QQ图和PP图

* Q-Q图：Q-Q图的横纵坐标分别是理论的分位数和样本的分位数，如果Q-Q图接近𝑦=𝑥y=x直线，则认为样本符合某个分布；
* P-P图：P-P图的横纵坐标是理论的累积分布函数和样本的累积分布，如果P-P图接近于𝑦=𝑥y=x那么认为样本符合某个分布。而当P-P图不是一条直线，有一定的规律，那么可以考虑对变量做一定的变换使得变换后的数据服从该分布。

# 使用Matplotlib绘制QQ图与PP图

from scipy import stats

sample\_norm = np.random.randn(100) #构造正态分布的样本

sample\_exp = np.random.exponential(1, 100) #构造指数分布的样本

sort\_sample\_norm = np.sort(sample\_norm) #假设为正态分布样本的分位数

sort\_sample\_exp = np.sort(sample\_exp) #假设为指数分布样本的分位数

theory\_norm = stats.norm.ppf(np.arange(100) / 100) #正态分布理论分位数

sample\_norm\_cdf = stats.norm.cdf(sort\_sample\_norm) # 正态分布理论累积分布

sample\_exp\_cdf = stats.norm.cdf(sort\_sample\_exp) # 正态分布理论累积分布

plt.figure(figsize=(12,12))

plt.subplot(2,2,1)

plt.scatter(theory\_norm, sort\_sample\_norm, c='tomato', marker="o", s=20)

plt.plot(theory\_norm, theory\_norm, c='dodgerblue')

plt.xlabel("理论分位数点")

plt.ylabel("样本分位数点")

plt.title("Q-Q图：样本基本服从标准正态分布")

plt.subplot(2,2,2)

plt.scatter(theory\_norm, sort\_sample\_exp, c='tomato', marker="o", s=20)

plt.plot(theory\_norm, theory\_norm, c='dodgerblue')

plt.xlabel("理论分位数点")

plt.ylabel("样本分位数点")

plt.title("Q-Q图：样本不服从标准正态分布")

plt.subplot(2,2,3)

stats.probplot(sample\_norm, plot=plt)

plt.xlabel("理论累积比例")

plt.ylabel("样本累积比例")

plt.title("P-P图：样本服从标准正态分布")

plt.subplot(2,2,4)

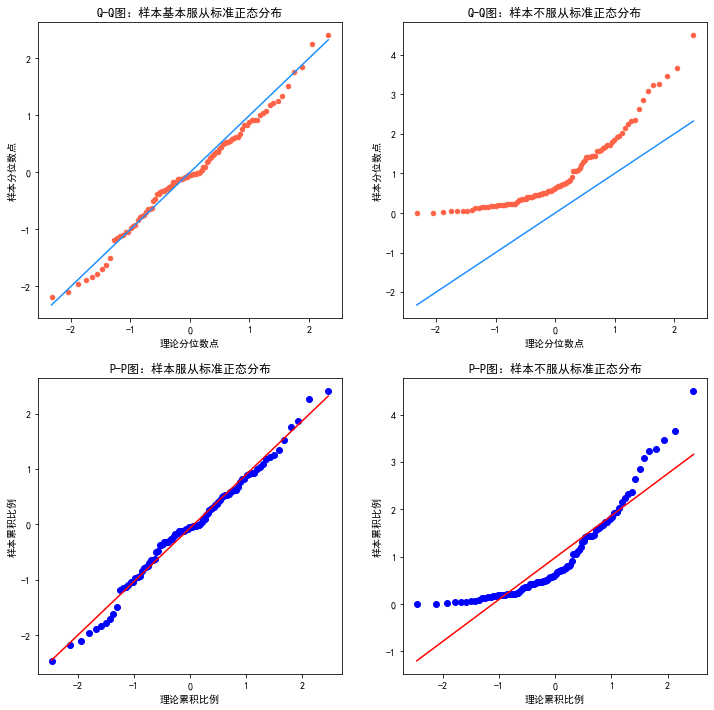
stats.probplot(sample\_exp, plot=plt)

plt.xlabel("理论累积比例")

plt.ylabel("样本累积比例")

plt.title("P-P图：样本不服从标准正态分布")

plt.show()



## 11.聚类散点图

# 使用Matplotlib绘制聚类散点图

from sklearn.datasets import load\_iris #家在鸢尾花数据集

iris = load\_iris()

X = iris.data

label = iris.target

feature = iris.feature\_names

df = pd.DataFrame(X, columns=feature)

df['label'] = label

label\_unique = np.unique(df['label']).tolist()

plt.figure(figsize=(10, 6))

for i in label\_unique:

df\_label = df.loc[df['label'] == i, :]

plt.scatter(x=df\_label['sepal length (cm)'], y=df\_label['sepal width (cm)'], s=20, label=i)

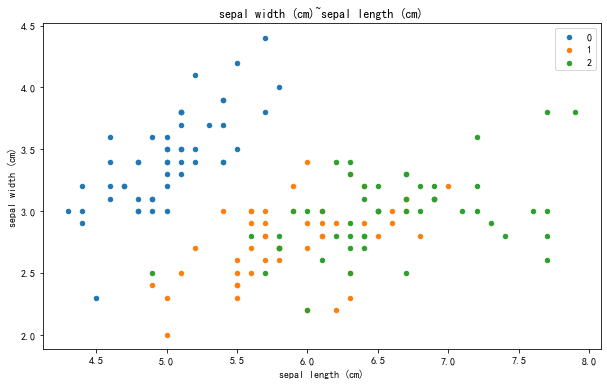
plt.xlabel('sepal length (cm)')

plt.ylabel('sepal width (cm)')

plt.title('sepal width (cm)~sepal length (cm)')

plt.legend()

plt.show()



## 12 相关系数矩阵图

### 12.1 使用plotpine

# 使用plotnine绘制相关系数矩阵图：

from plotnine.data import mtcars

corr\_mat = np.round(mtcars.corr(), 1).reset\_index() #计算相关系数矩阵

df = pd.melt(corr\_mat, id\_vars='index', var\_name='variable', value\_name='corr\_xy') #将矩阵宽数据变成长数据

df['abs\_corr'] = np.abs(df['corr\_xy'])

p1 = (

ggplot(df, aes(x='index', y='variable', fill='corr\_xy', size='abs\_corr'))+

geom\_point(shape='o', color='black')+

scale\_size\_area(max\_size=11, guide=False)+

scale\_fill\_cmap(name='RdYIBu\_r')+

coord\_equal()+

labs(x="Variable", y="Variable")+

theme(dpi=100, figure\_size=(4.5,4.55))

)

p2 = (

ggplot(df, aes(x='index', y='variable', fill='corr\_xy', size='abs\_corr'))+

geom\_point(shape='s', color='black')+

scale\_size\_area(max\_size=10, guide=False)+

scale\_fill\_cmap(name='RdYIBu\_r')+

coord\_equal()+

labs(x="Variable", y="Variable")+

theme(dpi=100, figure\_size=(4.5,4.55))

)

p3 = (

ggplot(df, aes(x='index', y='variable', fill='corr\_xy', label='corr\_xy'))+

geom\_tile(color='black')+

geom\_text(size=8, color='white')+

scale\_fill\_cmap(name='RdYIBu\_r')+

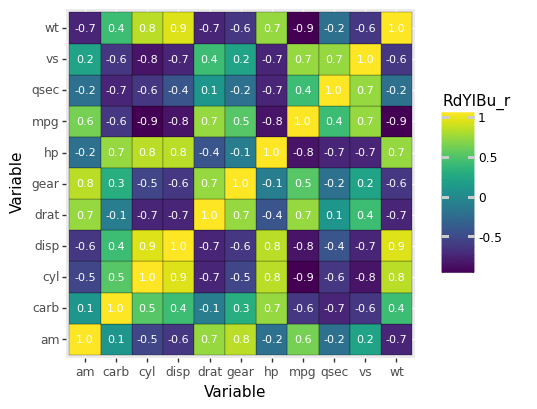
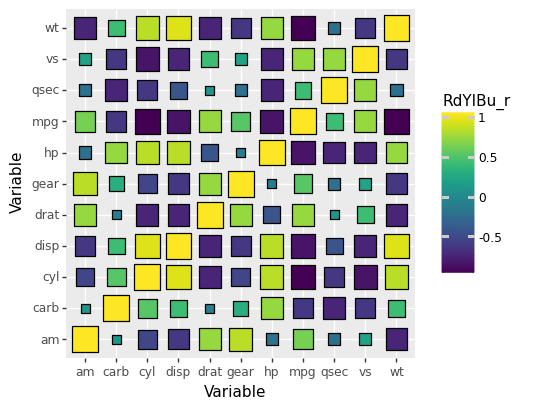
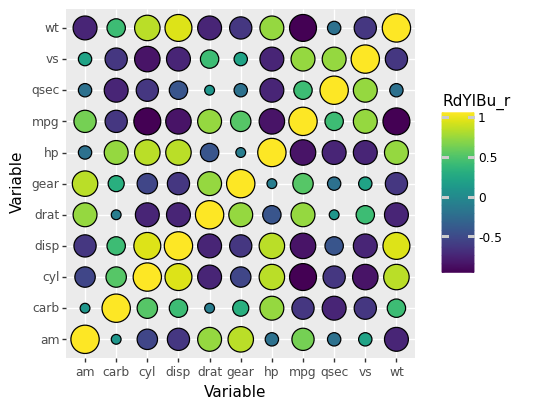
coord\_equal()+

labs(x="Variable", y="Variable")+

theme(dpi=100, figure\_size=(4.5,4.55))

)

print([p1, p2, p3])

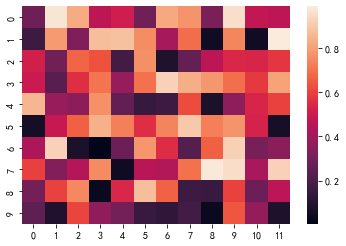


### 12.2 使用matplotlib

# 使用Matplotlib/Seaborn绘制相关系数矩阵图

uniform\_data = np.random.rand(10, 12)

sns.heatmap(uniform\_data)



## 13. 统计直方分布图：（统计直方图）

# 使用matplotlib绘制直方图：

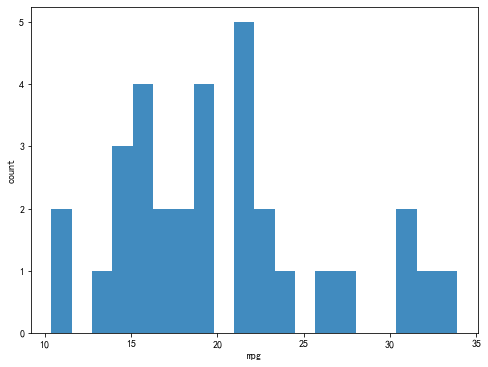
plt.figure(figsize=(8, 6))

plt.hist(mtcars['mpg'], bins=20, alpha=0.85)

plt.xlabel("mpg")

plt.ylabel("count")

plt.show()



## 14 核密度图：

直方图使用的是离散的区间内的频数来描述聚集和离散信息，能不能使用连续的曲线描述呢？实际上，使用连续曲线描述描述数据的聚集和离散信息的图称为：核密度图。

# 使用plotnine绘制核密度图：

from plotnine.data import mtcars

p1 = (

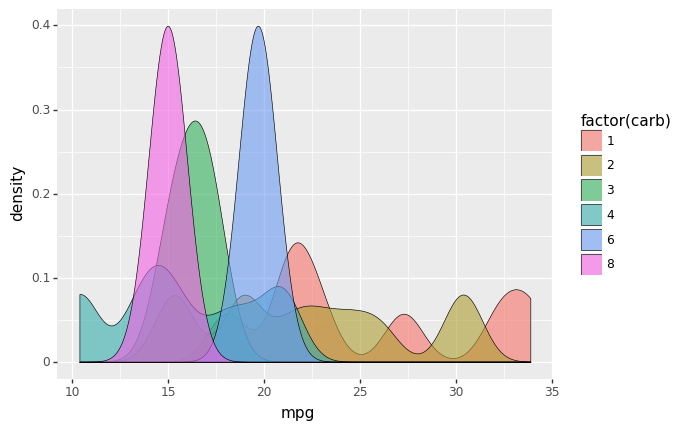
ggplot(mtcars, aes(x='mpg', fill='factor(carb)'))+

geom\_density(bw=1, alpha=0.6, color='black', size=0.25)+

scale\_fill\_hue(s=0.90, l=0.65, h=0.0417, color\_space='husl')

)

print(p1)



## 15. 箱线图：

描述某个变量的分布其实不仅仅使用密度函数和分布函数，也可以使用随机变量的数字特征，数字特征是反应某个随机变量的某方面特点的数值量。箱线图可以反应随机变量的分位数特征，通过箱线图的观察，可以知道变量的位置信息：

# 使用matplotlib绘制箱线图

import seaborn as sns

from plotnine.data import mtcars

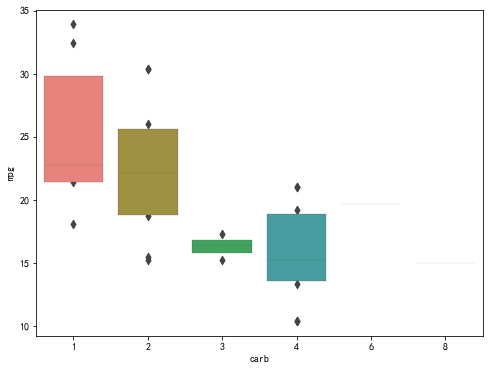
data = mtcars

data['carb'] = data['carb'].astype('category')

plt.figure(figsize=(8, 6))

sns.boxenplot(x='carb', y='mpg', data=mtcars, linewidth=0.2, palette=sns.husl\_palette(6, s=0.9, l=0.65, h=0.0417))

plt.show()



# 使用plotnine绘制的箱线图：

from plotnine.data import mtcars

p1 = (

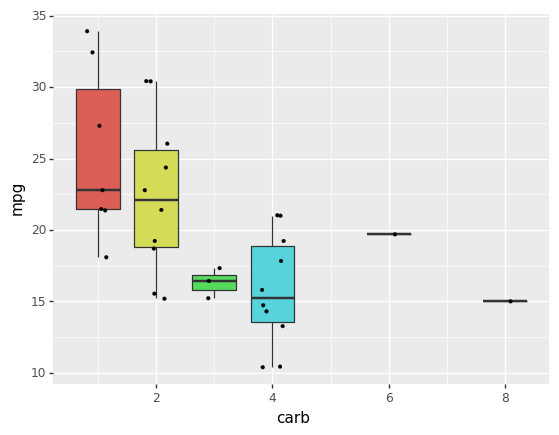
ggplot(mtcars, aes(x='carb', y='mpg', fill='factor(carb)'))+

geom\_boxplot(show\_legend=False)+

geom\_jitter(fill='black', shape='.', width=0.2, size=3, stroke=0.1, show\_legend=False)

)

print(p1)



## 16.提琴图：

之前的直方图只能反映数据的密度信息，箱线图只能反映数据的位置信息，有没有一种图可以将数据的密度信息和位置信息结合起来呢？有，这个图叫做：提琴图！

# 使用Plotnine绘制提琴图：

from plotnine.data import mtcars

p1 = (

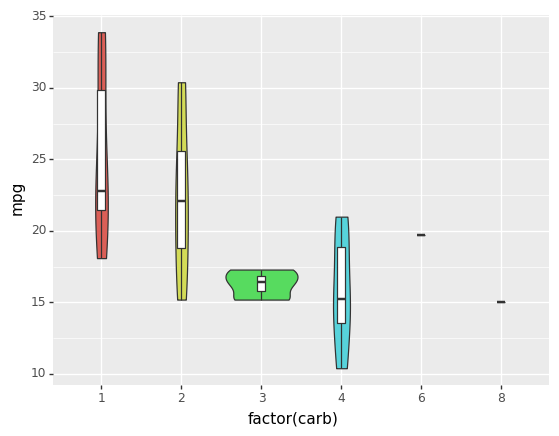
ggplot(mtcars, aes(x='factor(carb)', y='mpg', fill='factor(carb)'))+

geom\_violin(show\_legend=False)+

geom\_boxplot(fill='white', width=0.1, show\_legend=False)

)

print(p1)



## 17.饼状图

### 17.1 实心饼状图

# 使用Matplotlib绘制饼状图：

from matplotlib import cm, colors

df = pd.DataFrame({

'己方': ['寒冰', '布隆', '发条', '盲僧', '青钢影'],

'敌方': ['女警', '拉克丝', '辛德拉', '赵信', '剑姬'],

'己方输出': [26000, 5000, 23000, 4396, 21000],

'敌方输出': [25000, 12000, 21000, 10000, 18000]

})

df\_our = df[['己方', '己方输出']].sort\_values(by='己方输出', ascending=False).reset\_index()

df\_other = df[['敌方', '敌方输出']].sort\_values(by='敌方输出', ascending=False).reset\_index()

color\_list = [cm.Set3(i) for i in range(len(df))]

plt.figure(figsize=(16, 10))

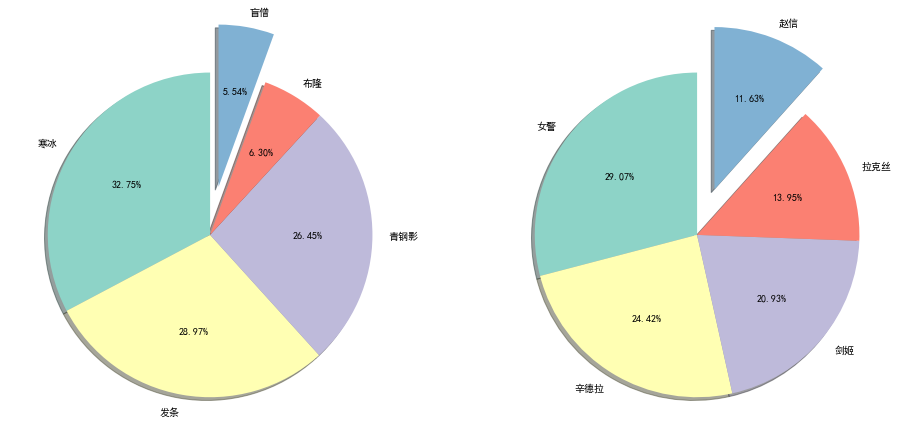
plt.subplot(1,2,1)

plt.pie(df\_our['己方输出'].values, startangle=90, shadow=True, colors=color\_list, labels=df\_our['己方'].tolist(), explode=(0,0,0,0,0.3), autopct='%.2f%%')

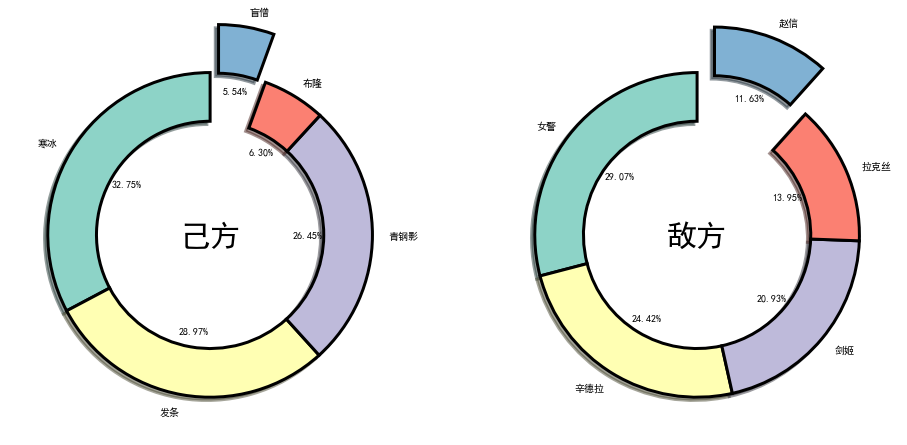
plt.subplot(1,2,2)

plt.pie(df\_other['敌方输出'].values, startangle=90, shadow=True, colors=color\_list, labels=df\_other['敌方'].tolist(), explode=(0,0,0,0,0.3), autopct='%.2f%%')

plt.show()



### 17.2 空心环状图（更好看！！！）：



## 18 时间序列线图

### 18.1 单时间序列折线图

df = pd.read\_csv(

'./data/AirPassengers.csv'

) # 航空数据1949-1960

df['date'] = pd.to\_datetime(df['date'])

# 使用Matplotlib绘制时间序列折线图

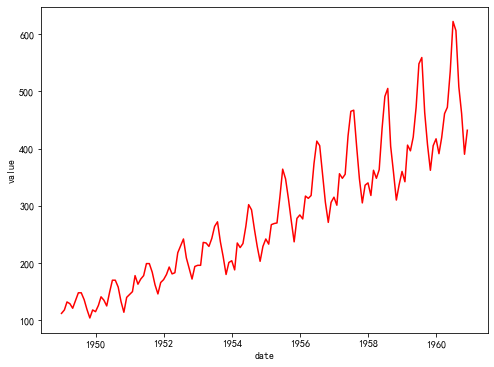
plt.figure(figsize=(8,6))

plt.plot(df['date'], df['value'], color='red')

plt.xlabel("date")

plt.ylabel("value")

plt.show()



### 18.2 多序列图

# Matplotlib 绘制多系列折线图

date\_list = pd.date\_range('2022-01-01', '2022-03-31').astype('str').tolist()

value\_list1 = np.random.randn(len(date\_list))

value\_list2 = np.random.randn(len(date\_list))

data = pd.DataFrame({

'date\_list': date\_list,

'value\_list1': value\_list1,

'value\_list2': value\_list2

})

data['date\_list'] = pd.to\_datetime(data['date\_list'])

plt.figure(figsize=(8, 6))

plt.plot(data['date\_list'], data['value\_list1'], color='red', alpha=0.86, label='value1')

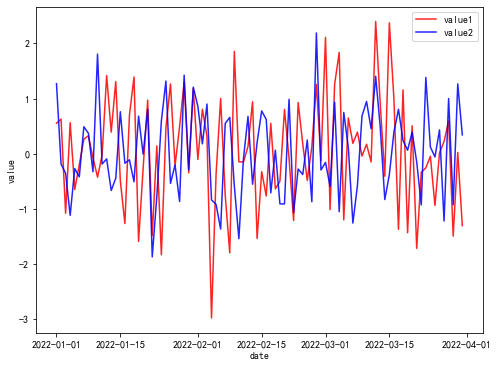
plt.plot(data['date\_list'], data['value\_list2'], color='blue', alpha=0.86, label='value2')

plt.legend()

plt.xlabel('date')

plt.ylabel('value')

plt.show()



## 19. 日历图：

说到时间，不可不说日历，我们刚刚使用的是线图表达每天的数值变化大小，但是这个并不利于发现周期性的信息，如：每周五晚上火车站客流量也许是个高峰，每个月月初的进货达到高峰等等。这时候，我们可以使用日历图：

# 使用Plotnine绘制日历图：

df = pd.DataFrame({

'date': pd.date\_range('2021-01-01', '2021-12-31'),

'value': np.abs(np.random.randn(len(pd.date\_range('2021-01-01', '2021-12-31')))\*10)

})

df['Year'] = df['date'].dt.year # 哪一年

df['Month'] = df['date'].dt.month # 哪个月

month\_list = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

df['Month\_label'] = df['Month'].replace(np.arange(1, 13, 1), month\_list) # 将月份转化成英文形式

df['Month\_label'] = pd.Categorical(df['Month\_label'], categories=month\_list, ordered=True) # 数据由字符串变成Category格式

df['Week'] = [int(t.strftime('%W')) for t in df['date']] # 第几周

df['Weekday'] = df['date'].dt.weekday + 1 # 星期几

weekday\_list = ['mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

df['Weekday\_label'] = df['Weekday'].replace(np.arange(1, 8, 1), weekday\_list) #将星期几换成英文

df['Weekday\_label'] = pd.Categorical(df['Weekday\_label'], categories=weekday\_list, ordered=True) # 数据由字符串变成Category格式

df['Day'] = df['date'].dt.day # 几号

df['month\_week'] = df.groupby(['Month\_label'])['Week'].apply(lambda x: x-x.min()+1) # 每个月的哪一周

p1 = (

ggplot(df, aes(x='Weekday\_label', y='month\_week', fill='value'))+

geom\_tile(colour='white', size=0.1)+ # 画出日历的方框

scale\_fill\_cmap(cmap\_name='Spectral')+

geom\_text(aes(label='Day'), size=8)+

facet\_wrap('~Month\_label', nrow=3)+ # 按照月份分面

scale\_y\_reverse()+ #将y周转向，因为越往下周数越大

xlab('Day')+ylab('value')+

theme(

strip\_text = element\_text(size=11,face="plain",color="black"),

axis\_title =element\_text(size=10,face="plain",color="black"),

axis\_text = element\_text(size=8,face="plain",color="black"),

legend\_position = 'right',

legend\_background = element\_blank(),

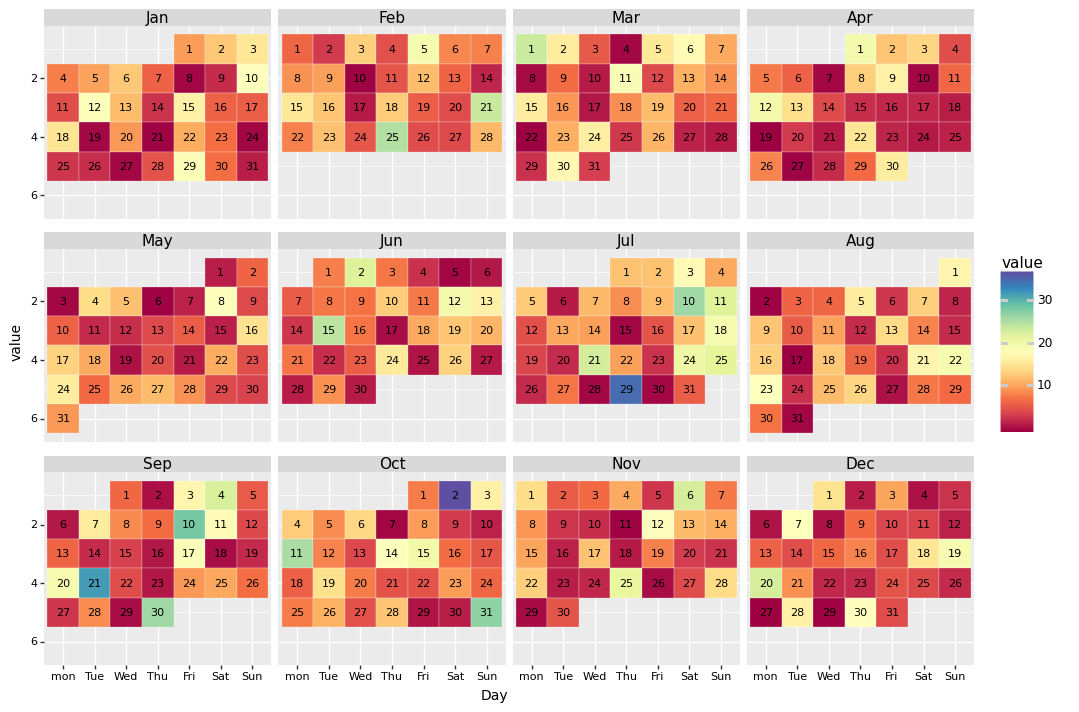
aspect\_ratio=0.85,

figure\_size=(12,12),

dpi=100)

)

print(p1)



## 20.空间分布图表（暂时不用看）

在介绍地图可视化之前，先来学习下关于地图数据的相关知识。绘制地理图表，首先得有空间位置数据，地图常见的数据格式有：SHP格式与Json格式。一般来说，在国家地理信息统计局提供的地图数据大多数以SHP格式，而大多数空间分析软件都能打开SHP格式的数据集并进行可视化；Json格式表示空间信息越来越常用，但是这种形式的数据体积更大。在Python中，我们可以使用GeoPandas包可以轻松读取这两种数据形式的地图数据。在geopandas中，有两种基本的数据格式：GeoSeries与GeoDataFrame两种基础数据结构，对应pandas的两种基本数据结构：Series和DataFrame。