数据可视化

Table of Contents

官方使用教程是非常重要的学习来源。

第一章　图形的构成

图形的构成可以参考 Matplotlib 官方网站上Anatomy of a figure的说明：

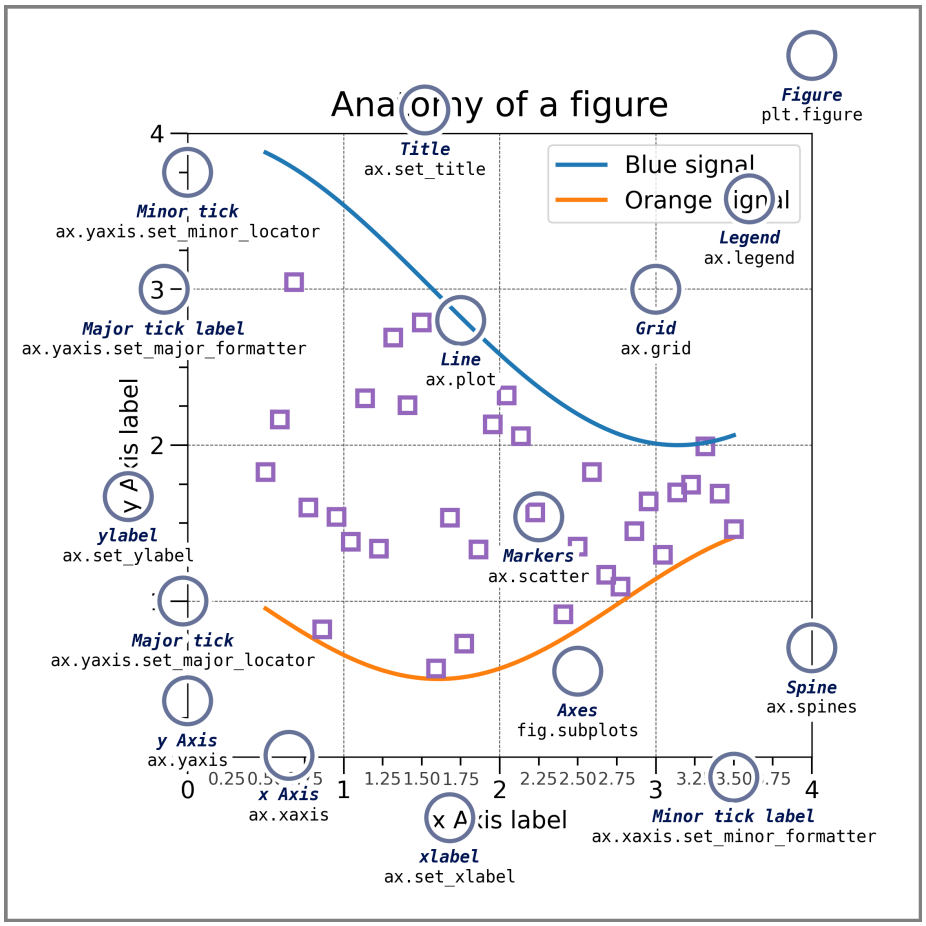


Figure 1.1: 图形解剖图

我们用一个例子来说明绘图的过程。下面的全球电影票房数据来自维基百科全球最高電影票房收入列表

```
import pandas as pd
import re
data = pd.read_excel("datasets/highest_gross_films.xlsx")
data['全球票房'] = data['全球票房'].apply(lambda ser: pd.to_numeric(re.sub(r'\D',''
df = data[:10]
df
```

	排名	峰值	影片名称	全球票房	年份
0	1	1	阿凡达	2923706026	2009
1	2	1	复仇者联盟：终局之战	2797501328	2019
2	3	3	阿凡达：水之道	2320250281	2022
3	4	1	泰坦尼克号	2257844554	1997
4	5	5	哪吒 2	2217080000	2025
5	6	3	星球大战：原力觉醒	2068223624	2015
6	7	4	复仇者联盟：无限战争	2048359754	2018
7	8	6	蜘蛛侠：英雄无归	1922598800	2021
8	9	8	头脑特工队 2	1698863816	2024
9	10	3	侏罗纪世界	1671537444	2015

第二章　绘图一般步骤

图形的种类非常多，应用 Python 绘图时可以大致分为几个步骤。

2.1　载入必要的库

除了基本的绘图工具Matplotlib外，Seaborn库也经常使用，在应用之前均应载入。

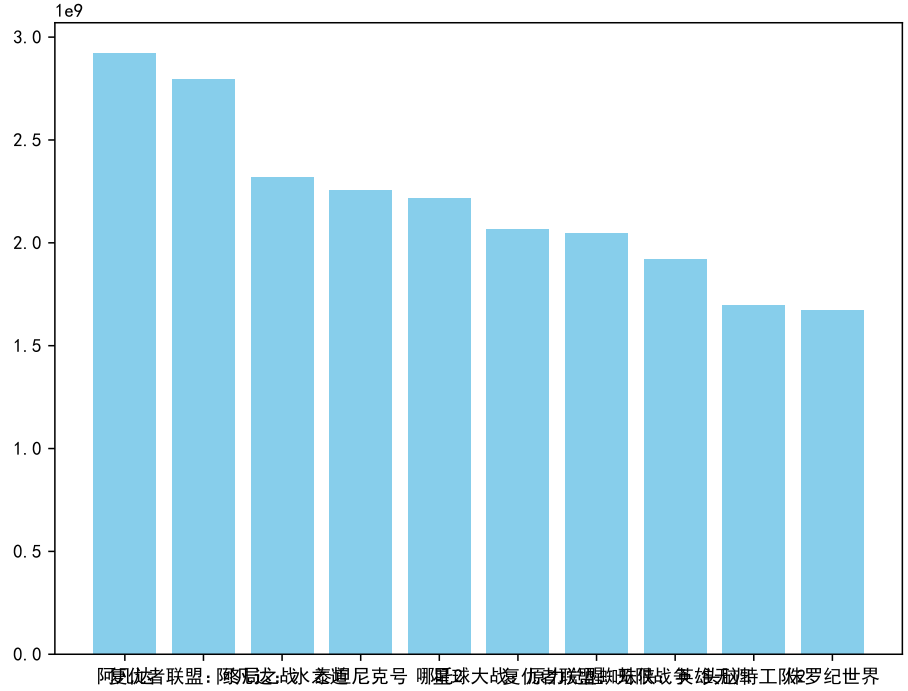
```
import matplotlib.pyplot as plt
import seaborn as sns
```

另外，Matplotlib 默认情况下不支持中文字符。如果你直接在图表标题、坐标轴标签或图例中使用中文，很可能会看到方框乱码或者问号。

```
plt.rcParams['font.sans-serif'] = ['SimHei', 'Heiti TC', 'WenQuanYi Zen Hei', 'Arial']

plt.rcParams["axes.unicode_minus"] = False
```

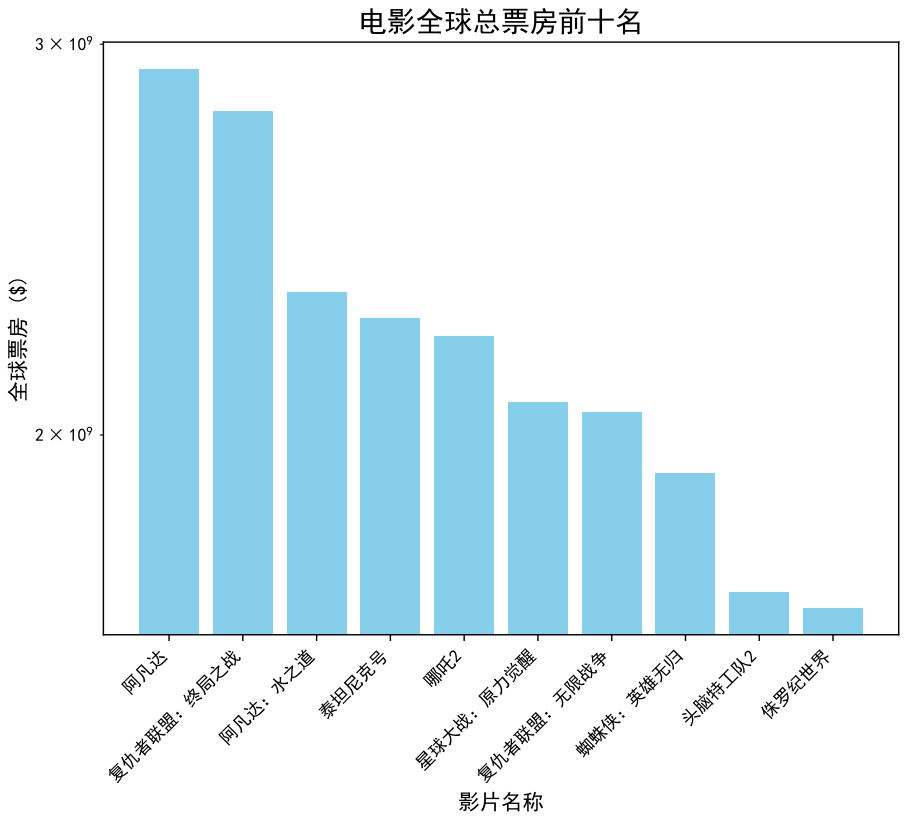
现在有了图形轴（Axes）的实例，就可以在上面绘制图形了。例如绘制一幅柱形图，使用.bar() 方法：

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(x=df['影片名称'], height=df['全球票房'], color='skyblue')
plt.show()
```



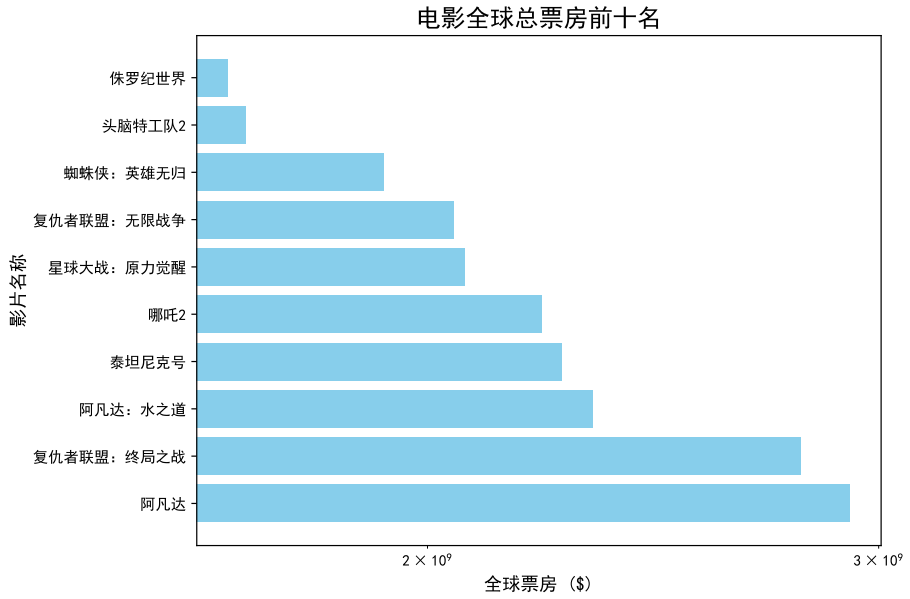
显然，图形还有改善的空间。比如横轴的标签，即电影名字挤在一起看不清楚，也可以设置纵轴标签、图形标题等：

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(x=df['影片名称'], height=df['全球票房'], color='skyblue')
ax.set_yscale("log")
ax.set_title('电影全球总票房前十名', fontsize=16, fontweight='bold')
ax.set_xlabel('影片名称', fontsize=12)
ax.set_ylabel('全球票房 ($)', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.show()
```



横向柱形图

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(y = df['影片名称'], width=df['全球票房'], color='skyblue')
ax.set_xscale("log")
ax.set_title('电影全球总票房前十名', fontsize=16, fontweight='bold')
ax.set_ylabel('影片名称', fontsize=12)
ax.set_xlabel('全球票房 ($)', fontsize=12)

plt.show()
```


电影全球总票房前十名
侏罗纪世界
头脑特工队2
蜘蛛侠：英雄无归
复仇者联盟：无限战争
星球大战：原力觉醒
哪吒2
泰坦尼克号
阿凡达：水之道
复仇者联盟：终局之战
阿凡达
影片名称
2 × 10⁹
3 × 10⁹
全球票房（$）

第三章　常用的图形

3.1　直方图

直方图可以被看作是估计概率密度函数（PDF）的一种基本而直观的方法，但严格来说，它估计的是概率质量函数（PMF），尤其是在处理离散数据时。当用于连续数据时，它更像是 PDF 的一个粗略估计。

直方图将数据分成一系列不重叠的 “箱子”（bins）。对于每个箱子，它统计落入该箱子中的数据点的数量，并以一个矩形柱的高度来表示这个数量，当然也可以用每个柱子的高度表示该箱子中数据点所占的比例或频率。

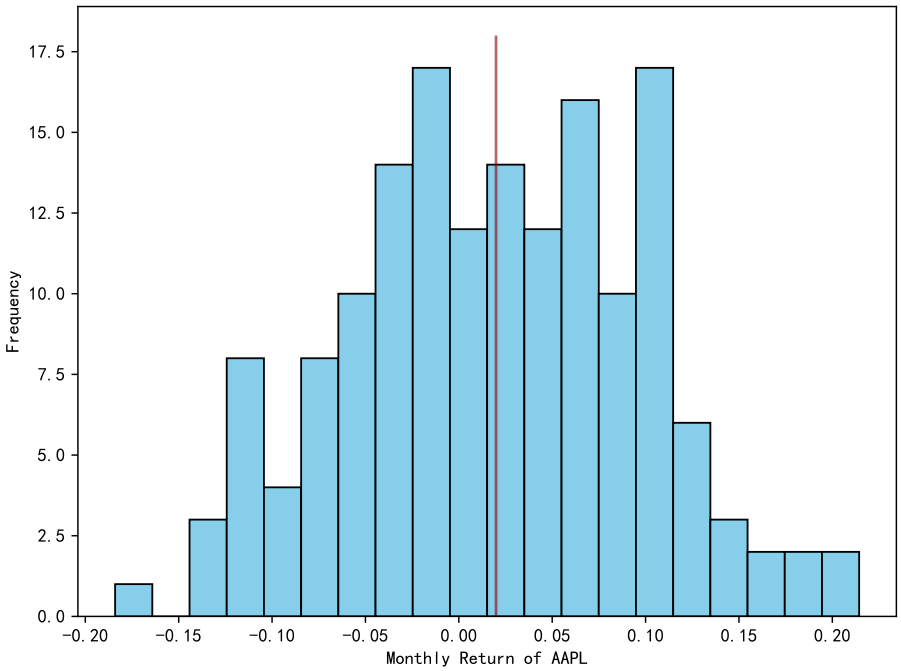
下面的例子自雅虎财经网站下载几支股票的月度（后复权调整）收盘价数据，然后使用 `df.pct_change()` 函数计算了简单收益率。

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
#import yfinance as yf
# stocks_list = ['AAPL','BA','MGM','AMZN','IBM','TSLA','GOOG','^GSPC']
# start_date = "2012-01-01"
# end_date = "2025-06-30"
# df = yf.download(tickers=stocks_list,
#          start=start_date,
#          end=end_date,
#          interval="1mo",
#          auto_adjust=True,
```

```
#          progress=False)['Close']

# df.to_csv("datasets/stocks_price_us.csv")

stocks_price_us = pd.read_csv("datasets/stocks_price_us.csv",
                    header=0, index_col=0, parse_dates=True )

returns = stocks_price_us.pct_change()
```

例如，绘制苹果公司股票收益率的直方图：

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.hist(x=returns['AAPL'], bins=20,
        color="skyblue", edgecolor="black")
ax.vlines(x=returns['AAPL'].mean(),
        ymin=0, ymax=18, colors="darkred",
        alpha=0.6)
ax.set_xlabel("Monthly Return of AAPL")
ax.set_ylabel("Frequency")
plt.show()
```

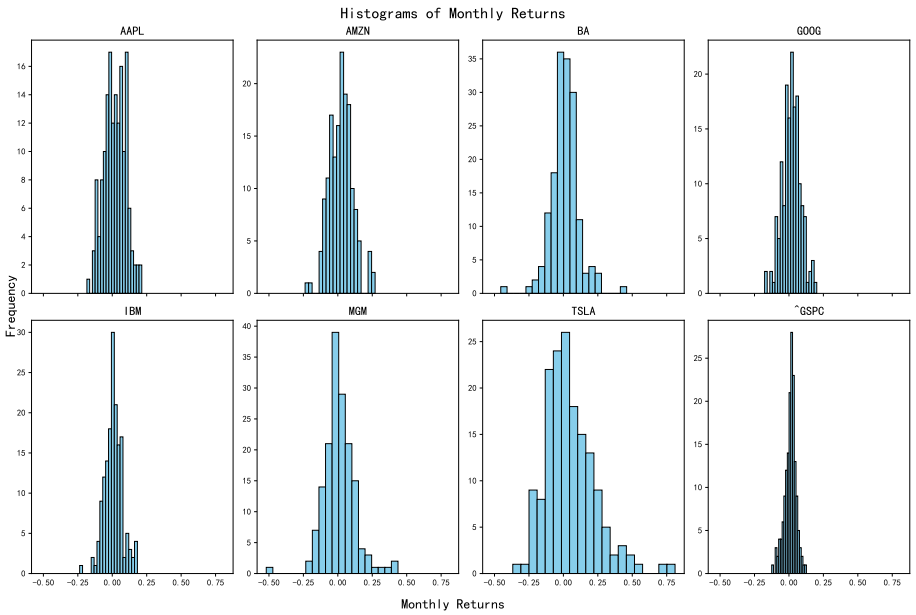


如果纵轴希望表示为概率密度，加上参数 `density=True`。

下载的数据包含 8 家 7 家企业以及标准普尔 500 指数（GSPC），下面将收益率为子图绘制直方图。为了可比，横轴使用了 `sharex=True` 参数：

```
fig, axes = plt.subplots(nrows=2,
                         ncols=4,
                         figsize=(15, 10),
                         sharex=True)
stock_index = 0
for row in range(2):
    for col in range(4):
        stock = returns.columns[stock_index]
        axes[row, col].hist(returns[stock], color="skyblue",
                edgecolor='black', bins=20)
        axes[row, col].set_title(f'{stock}', fontsize=14)
        stock_index += 1
fig.supxlabel("Monthly Returns", fontsize=16)
```

```
fig.supylabel("Frequency", fontsize=16)
fig.suptitle("Histograms of Monthly Returns", fontsize=18)
plt.tight_layout()
plt.show()
```



3.2 核密度图

对连续数据来讲，估计概率密度函数的更好的方法是核密度函数估计（Kernel Density Estimator，KDE）。

设 $(x_1, x_2, \cdots, x_n)$ 为从单变量分布中抽取的独立同分布样本，给定点 x 有未知的概率密度 $f(x)$，我们需要估计观察到的值的概率密度函数 $\hat{f}(x)$ 。

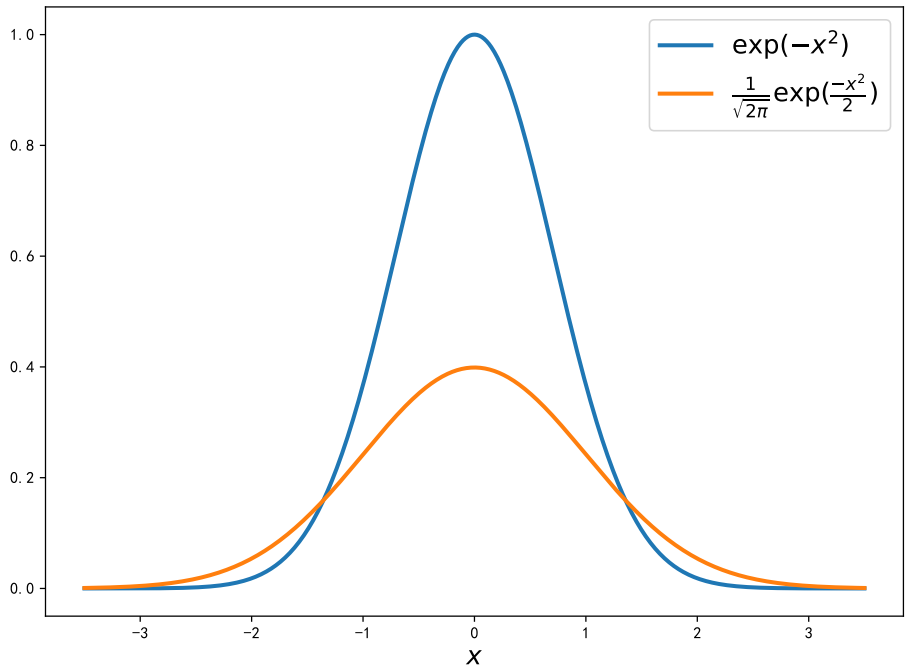
在 KDE 估计过程中处于核心地位的是**核函数**，我们逐步来看应用的过程。

假设有一个观测值，例如 $x = 0$，我们要估计观测值服从的概率密度函数，最合理的估计是使用一个 PDF 在该点取峰值，向两侧衰减。例如函数 $= \exp(-x^2)$ 满足这个条件。但是我们知道，PDF 下面积应为 1，因此进行

适当转换以满足该条件，记为：

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp(\frac{-x^2}{2})$$

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

def k(x):
    return np.exp(-x**2)
def K(x):
    return (1/np.sqrt(2*np.pi))*np.exp(-x**2/2)

x = np.linspace(-3.5, 3.5, 1000)
y1 = k(x)
y2 = K(x)

fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x, y1, linewidth = 2.5, label = r"$\exp(-x^{2})$")
ax.plot(x, y2, linewidth = 2.5, label = r"$\frac{1}{\sqrt{2\pi}}\exp(\frac{-x^{2}}{2})$")
ax.set_xlabel(r'$x$', fontsize=16)
ax.set_ylabel('')
ax.legend(fontsize=16)
plt.tight_layout()
plt.show()
```



这样我们就得到了一个非常常用的核函数 $K(x)$，零均值和单位方差的高斯（Gaussian）分布。

对观测值中的任意点 x_i，相当于沿着 x 轴平移曲线，用核函数表示为：

$$K(x - x_i)$$

要让曲线更宽或更窄，可以加入一个常数 h 在分母上，称为带宽。这样将核函数曲线下面积乘了 h ，为保持单位面积需要除以 h，即：

$$\frac{1}{h}K(\frac{x - x_i}{h}) \tag{5.1}$$

带宽的选择影响估计的密度函数，带宽越宽，曲线越平缓；带宽越小，曲线越陡峭。

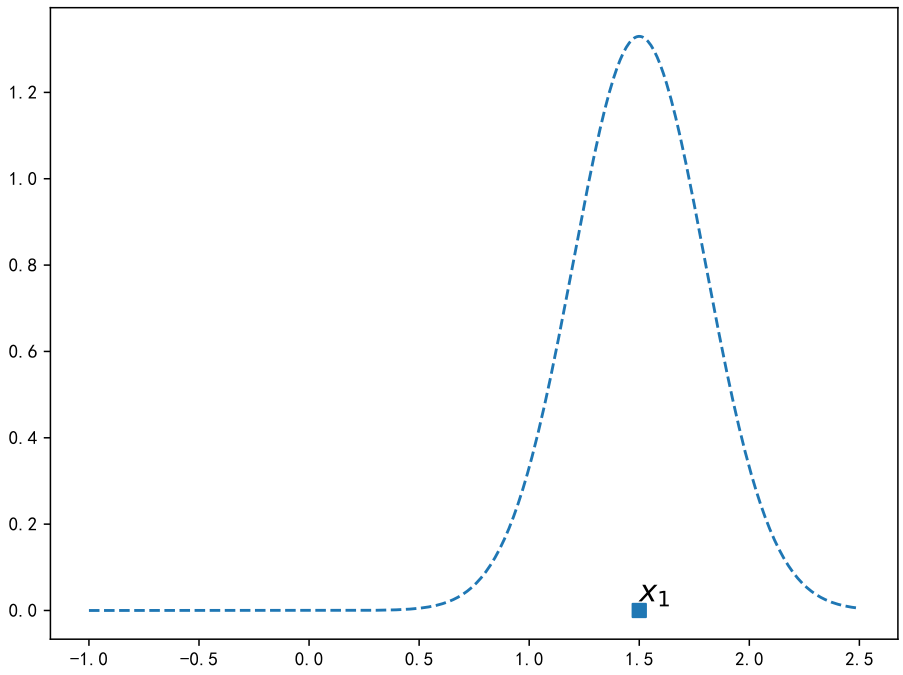
KDE 估计阐释性例子

下面利用上面的函数 5.1，从最简单的情况开始，逐步阐释 KDE 是如何进行的。

首先假设观测值只包含一个点，$x_1 = 1.33$，我们选择一个带宽，比如 $h = 0.3$，估计的 PDF 为：

$$\frac{1}{h}K(\frac{x - x_1}{h})$$

```
x1 = 1.5
h = 0.3

x_grid = np.linspace(-1, 2.5, num = 500)
fig, ax = plt.subplots(figsize=(8, 6))
f_grid = K((x_grid-x1)/h)/h
ax.plot(x_grid, f_grid, linestyle = "--")
ax.scatter(x1, 0, marker="s",s=50)
ax.annotate(r"$x_{1}$", xy=[x1, 0.02], fontsize=16)
plt.show()
```



现在假设第二个观测值为 $x_2 = 0.5$，可以同样的方式估计

$$\frac{1}{h}K(\frac{x - x_2}{h})$$

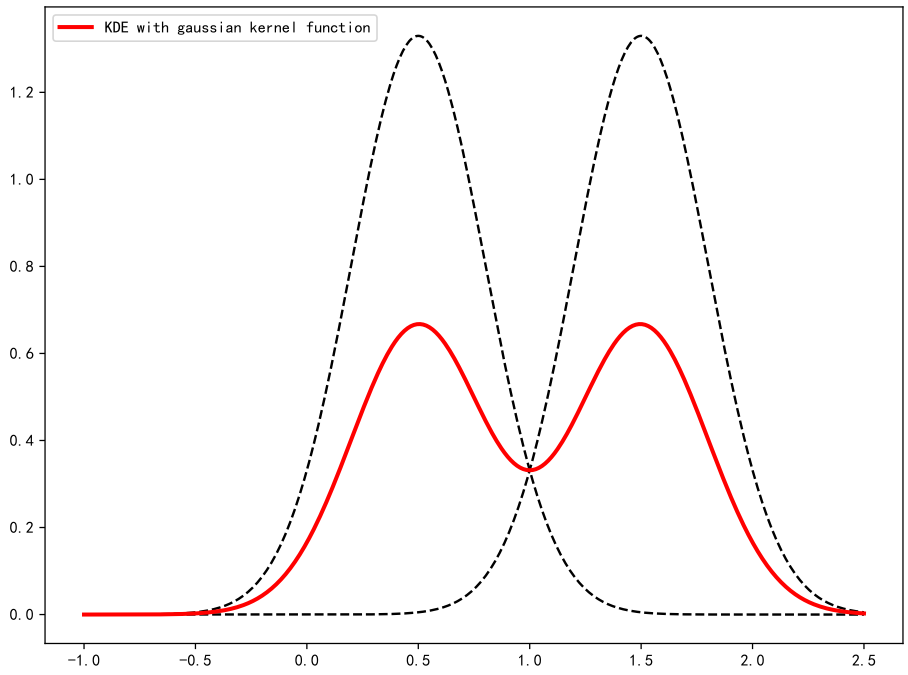
要得到一个概率密度函数，需要将两者加起来，然后除以 2：

$$f(x) = \frac{1}{2h}\left[K(\frac{x-x_1}{h}) + K(\frac{x-x_2}{h})\right] = \frac{1}{2h}\sum_{i=1}^{2} K(\frac{x-x_i}{h})$$

在 Scipy 库中，有常见的统计分布的概率密度函数，可以利用其正态分布的概率密度函数 `norm().pdf()` 非常简便的进行计算:

- 对每个数据点 xi，创建一个均值为 xi、标准差为 h 的正态分布对象；
- 对每个对象，计算 x_d 网格上的 PDF 值；
- 对所有 PDF 值进行求和；
-

```
from scipy.stats import norm
x = np.array([1.5, 0.5])
h = 0.3
x_grid = np.linspace(-1, 2.5, num = 500)

n = len(x)
density = sum(norm(loc=xi, scale=h).pdf(x_grid) for xi in x)/n

fig, ax = plt.subplots(figsize=(8, 6))
for xi in x:
    ax.plot(x_grid, norm(xi,h).pdf(x_grid), color = 'black', linestyle = "--")
ax.plot(x_grid, density, color='red',
            linewidth = 2.5, label = "KDE with gaussian kernel function")
ax.legend()
plt.tight_layout()
plt.show()
```

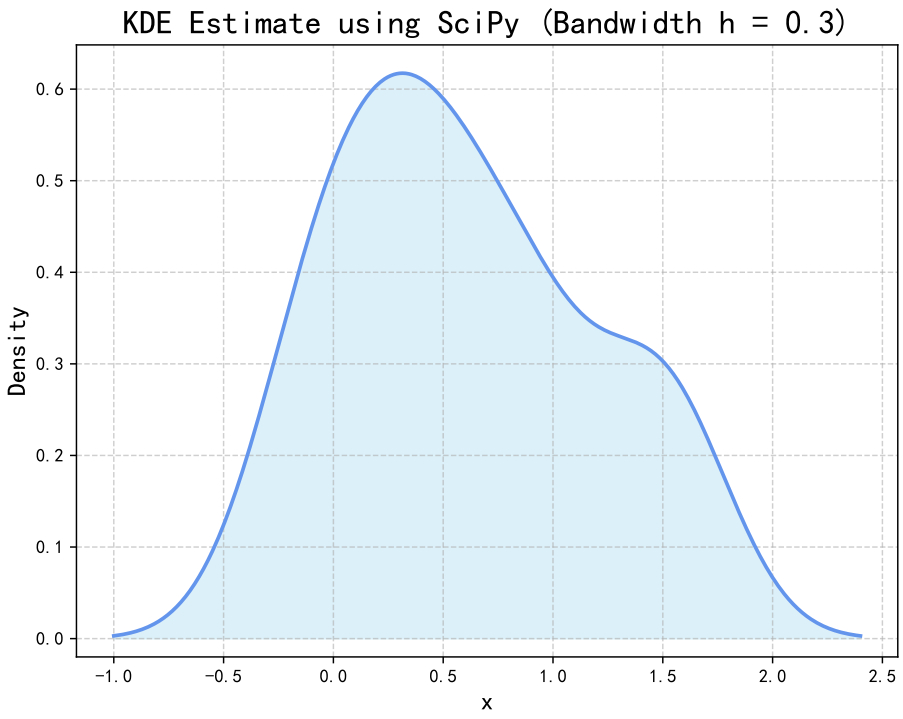


很容易将上述方法类推到 n 个观测值 $(x_1, x_2, \cdots, x_n)$ 的情况：

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{5.2}$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def plot_kde_scipy(data, h, n_points=500):
    data_min = np.min(data)
    data_max = np.max(data)
    x_min = data_min - 3 * h
    x_max = data_max + 3 * h
    n = len(data)
    x_grid = np.linspace(x_min, x_max, num=n_points)
    density = sum(norm(loc=xi, scale=h).pdf(x_grid) for xi in data)/n
```

```
    plt.figure(figsize=(8, 6))
    plt.fill_between(x_grid, density, alpha=0.3, color='skyblue')
    plt.plot(x_grid, density, color='cornflowerblue', linewidth=2)

    plt.title(f'KDE Estimate using SciPy (Bandwidth h = {h})', fontsize=18)
    plt.xlabel('x', fontsize=14)
    plt.ylabel('Density', fontsize=14)
    plt.ylim(bottom=-0.02)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()

data = np.array([1.5, 0.5,-0.1, 0.9, 0.23])
bandwidth1 = 0.3
plot_kde_scipy(data, h=bandwidth1)
```



Seaborn 中的 `kdeplot` 方法

Seaborn 库中有 `kdeplot` 函数，可以方便用来估计。带宽选择参数 `bw_method`:

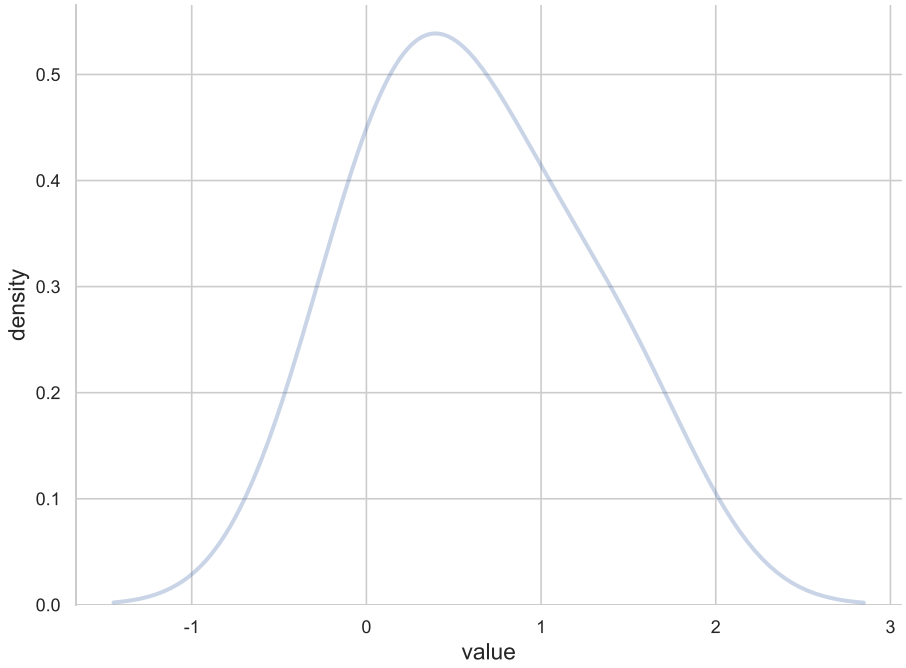
- Scott 方法 (Scott [1]):

$$h \approx 1.06 \cdot \hat{\sigma} n^{-1/5}$$

- Silvermans 方法 (Silverman [2])：

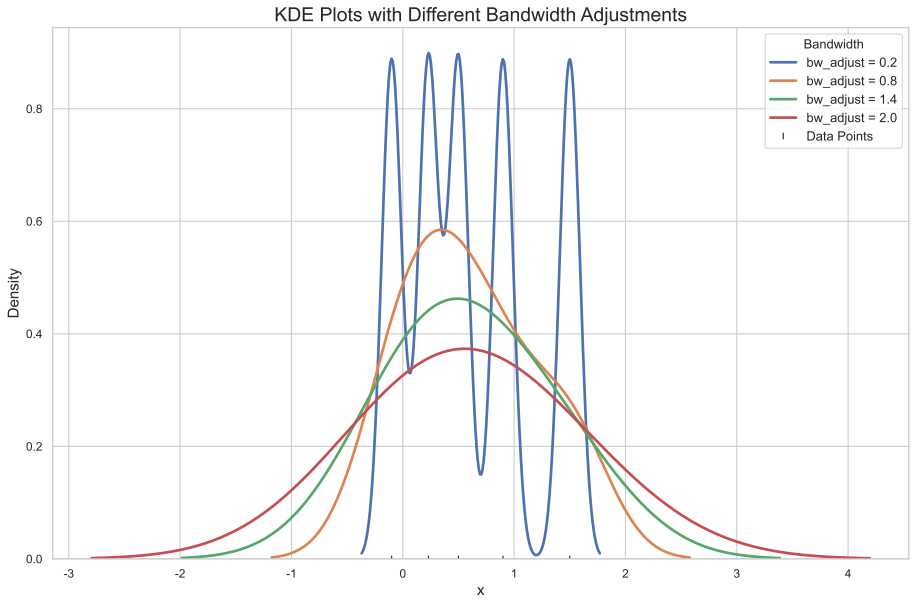
$$h = 0.9 \cdot \min(\hat{\sigma}, IQR/1.35) n^{-1/5}$$

```
data = np.array([1.5, 0.5,-0.1, 0.9, 0.23])
import seaborn as sns
sns.set_theme(style="whitegrid")
fig, ax = plt.subplots(figsize=(8, 6))
sns.kdeplot(data, bw_method="scott",
            alpha = 0.3,
            linewidth = 2.5,
            ax=ax)

ax.set_xlabel("value", fontsize=14)
ax.set_ylabel("density", fontsize=14)
sns.despine(left=False, bottom=True)
plt.tight_layout()
plt.show()
```



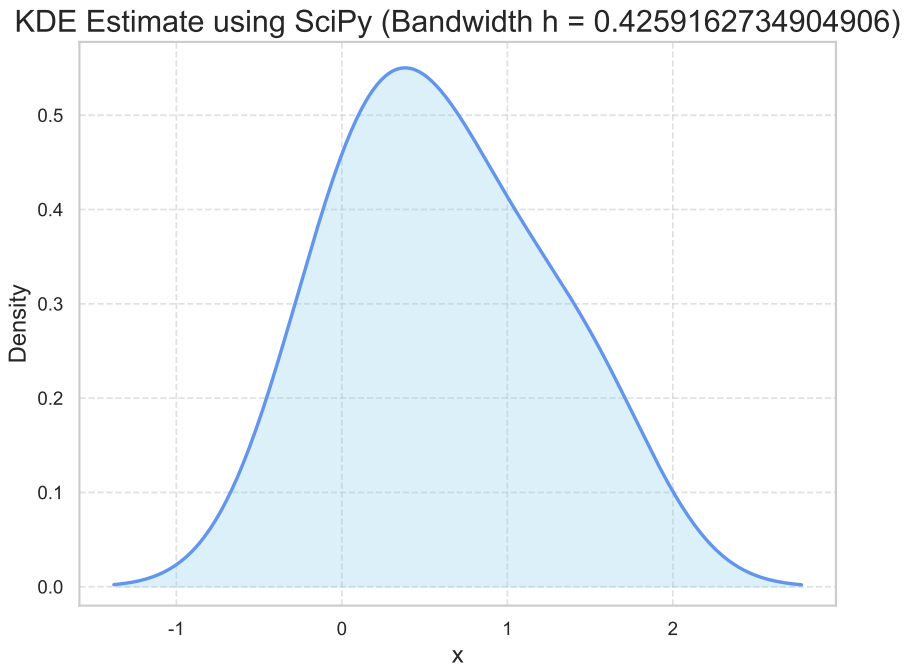
可以取不同带宽看其影响。注意这里 `bw_adjust` 是作为一个系数乘以 `scott` 方法得到的带宽值，详见说明文档：

```
bw_adjust_values = [0.2, 0.8, 1.4, 2.0]
fig, ax = plt.subplots(figsize=(12, 8))
for bw in bw_adjust_values:
    sns.kdeplot(data, bw_adjust=bw, label=f'bw_adjust = {bw}', linewidth=2.5)
ax.plot(data, np.zeros_like(data), '|k', markeredgewidth=1, label='Data Points')

ax.set_title('KDE Plots with Different Bandwidth Adjustments', fontsize=18)
ax.set_xlabel('x', fontsize=14)
ax.set_ylabel('Density', fontsize=14)
ax.legend(title='Bandwidth', fontsize=12)
plt.tight_layout()
plt.show()
```



我们可以定义一个函数计算 soctt 方法的带宽值，然后用自定义的函数来绘制 KDE 图：

```
def scott_method(x):
    h = 1.06*x.std()*len(x)**(-0.2)
    return h

data = np.array([1.5, 0.5,-0.1, 0.9, 0.23])
scott_value = scott_method(data)
plot_kde_scipy(data, h=scott_value)
```



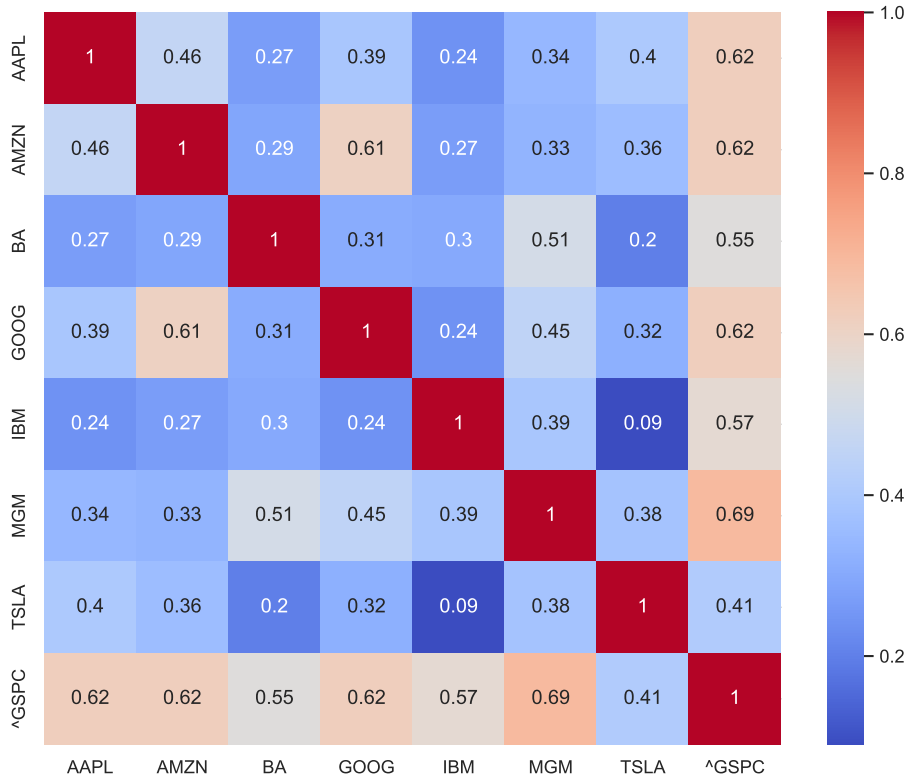
除了正态核函数以外，还有其他核函数，但总体来讲核函数的影响没有带宽影响的差异大，Seaborn 的 `kdeplot` 方法甚至取消了原本可以选择的设定。Scikit-Learn Kernel Density 算法提供了 6 种不同的核函数。另外，Pandas 库的 `s.plot.kde()` 方法也可以绘制核密度图。

3.3 热图

热图是一种数据可视化技术，用颜色深浅来表示数据集中不同数值的大小或密集程度。

```
corr = returns.corr().round(2)

fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr, annot=True,
            ax=ax,
            cmap='coolwarm')
```

```
plt.show()
```



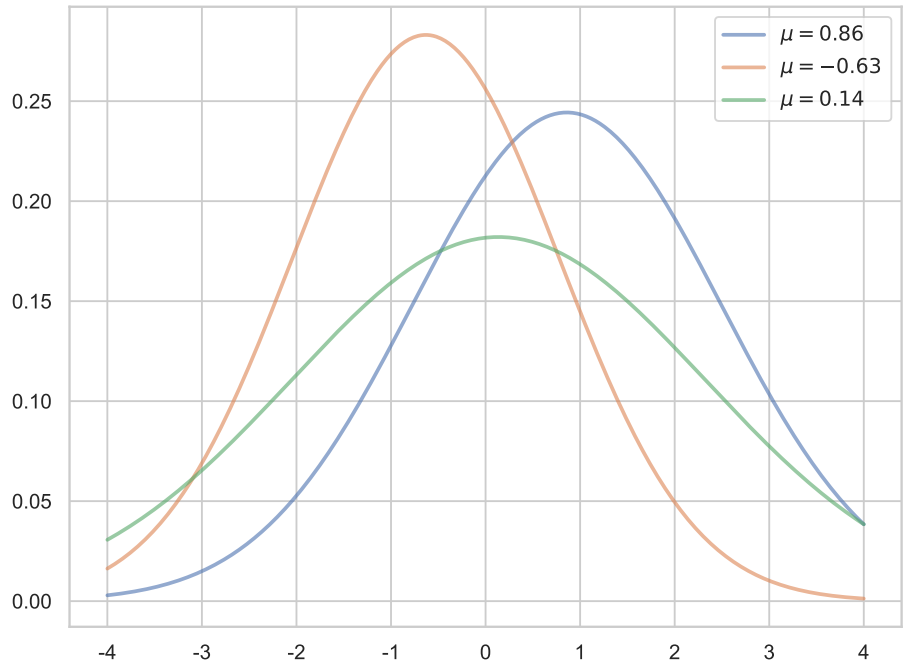
3.4 箱形图

3.5 散点图

第四章　多图和子图

当有多幅图形绘制时，可以放在一张图形上或用子图排列。### 多图

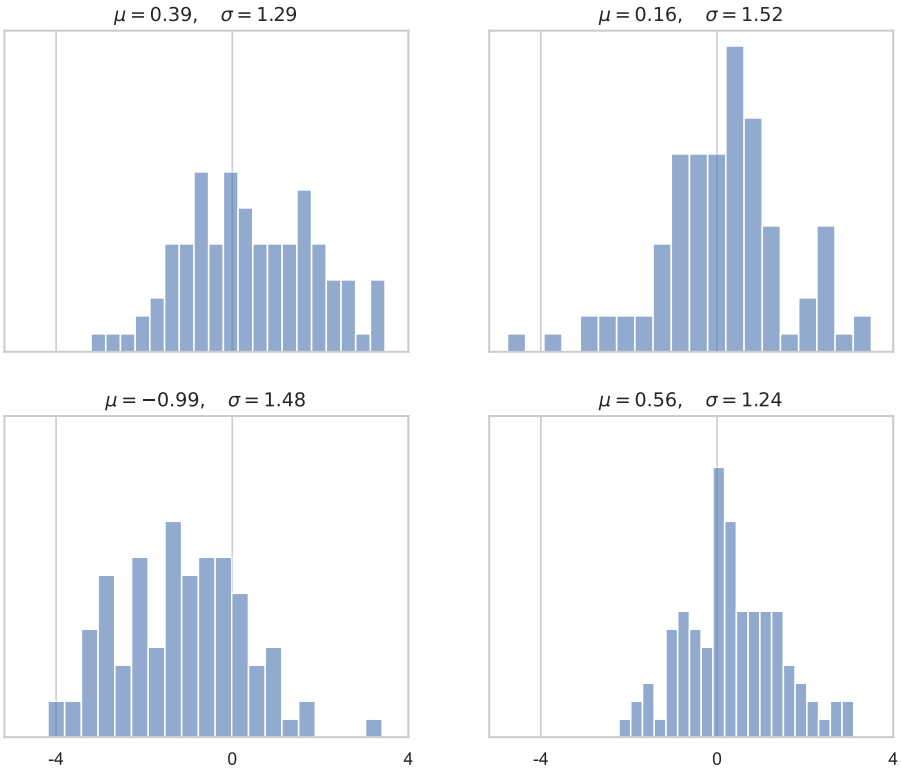
```
# 正态分布
import numpy as np
from scipy.stats import norm
np.random.seed(12345)

fig, ax = plt.subplots(figsize=(8, 6))
x = np.linspace(-4, 4, 500)
for i in range(3):
    mu, std = np.random.uniform(-1,1), np.random.uniform(1, 3)
    y = norm.pdf(x, loc = mu, scale = std)
    current_label = rf"$\mu = {mu:.2f}$"
    ax.plot(x, y, linewidth = 2, alpha = 0.6, label = current_label)
ax.legend()
plt.show()
```



4.1 子图

下面的例子绘制了 4 个子图，按照 2×2 的方式排列 `nrows=2, ncols=2`：

```
np.random.seed(123)
fig, axes = plt.subplots(nrows=2, ncols=2,figsize=(10,8),sharex=True,sharey=True)
for i in range(2):
    for j in range(2):
        m, s = np.random.uniform(-1, 1), np.random.uniform(1, 2)
        x = np.random.normal(m,s,100)
        axes[i, j].hist(x, alpha = 0.6, bins=20)
        title = rf"$\mu = {m:.2f},\quad \sigma = {s:.2f}$"
        axes[i, j].set(title = title, xticks = [-4, 0, 4],yticks = [])
plt.show()
```



4.2 图形风格

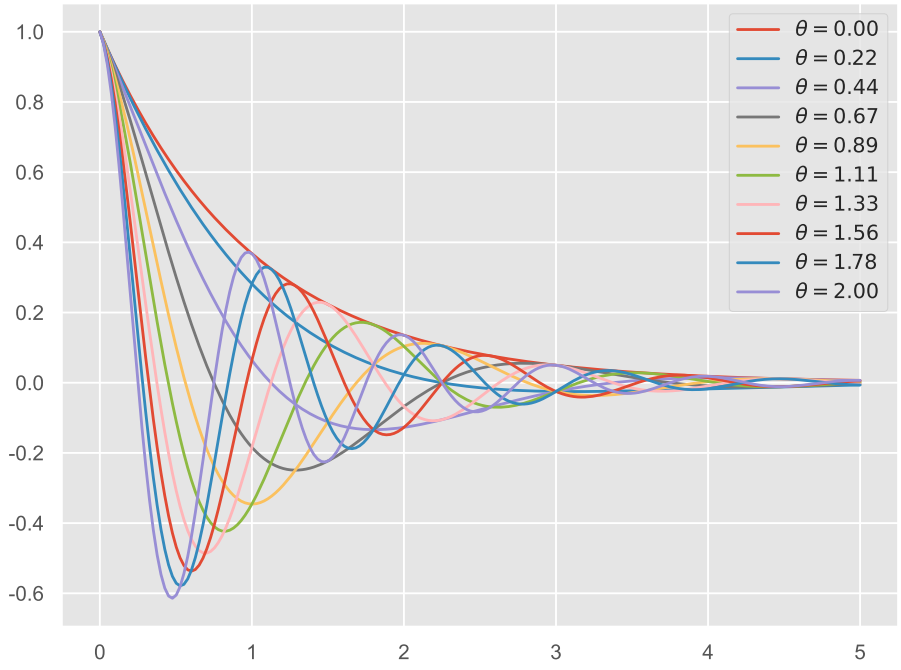
```
plt.style.available
```

```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
```

```
 'ggplot',
 'grayscale',
 'petroff10',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

```python
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")

def f(x, theta):
    return np.cos(np.pi * theta * x ) * np.exp(- x)

 _vals = np.linspace(0, 2, 10)
x = np.linspace(0, 5, 200)
fig, ax = plt.subplots(figsize=(8, 6))

for theta in  _vals:
    ax.plot(x, f(x, theta), label = rf"$\theta = {theta:.2f}$")
```

```
ax.legend()
plt.show()
```



第五章　应用：收益率的几个典型事实

这部分内容，

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
import scipy.stats as stats
import statsmodels.api as sm
```

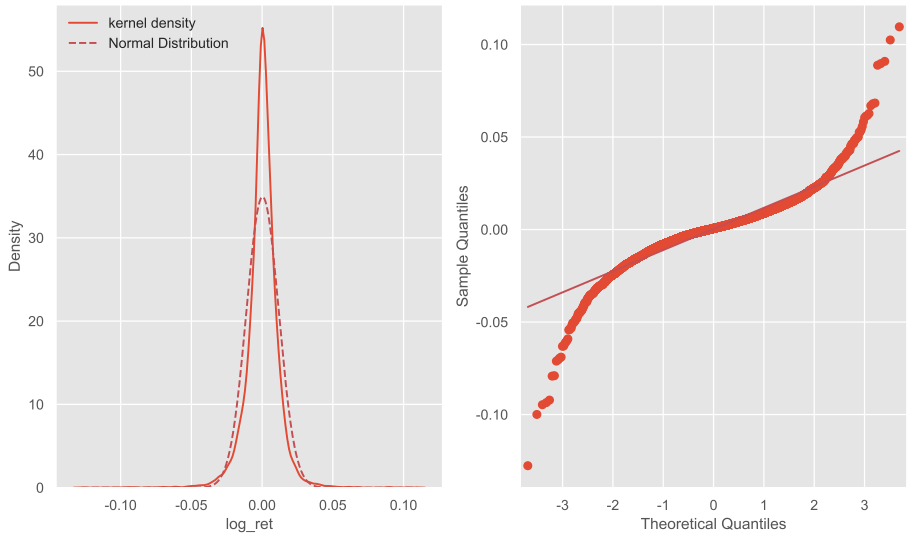
然后下载或读取数据：

```
# df = yf.download(['^GSPC', '^VIX'],
#                  start='1990-01-01',
#                  auto_adjust=True,
#                  progress=False)['Close']
# df.to_csv('datasets/sp500_vix.csv')

df = pd.read_csv('datasets/sp500_vix.csv',
                 header=0,
                 index_col=0,
                 parse_dates=True)
df.columns = ["SP500", "VIX"]
```

```
df['log_ret'] = np.log(df['SP500']/df['SP500'].shift(1))
df.dropna(inplace=True)
```

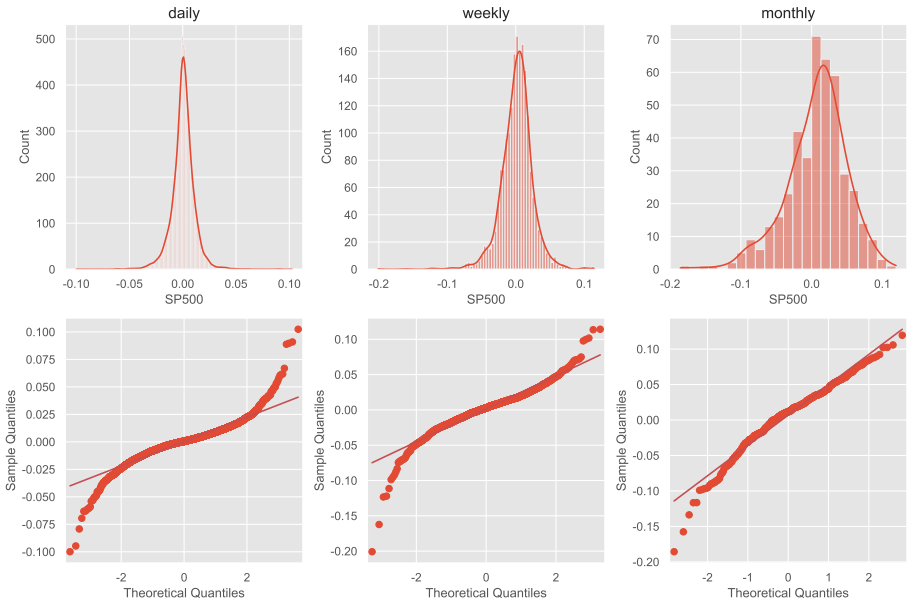
5.1 厚尾

```
fig,ax =plt.subplots(1,2,figsize=(10,6))
sns.kdeplot(df['log_ret'], fill=False,label='kernel density',ax=ax[0])
mu, sigma = stats.norm.fit(df['log_ret'])
x = np.linspace(df['log_ret'].min(), df['log_ret'].max(), 1000)
y = stats.norm.pdf(x, mu, sigma)
ax[0].plot(x, y, color='r', label='Normal Distribution',linestyle='--')
ax[0].legend(frameon=False)
sm.qqplot(df['log_ret'], line='s', ax=ax[1])
plt.tight_layout()
plt.show()
```



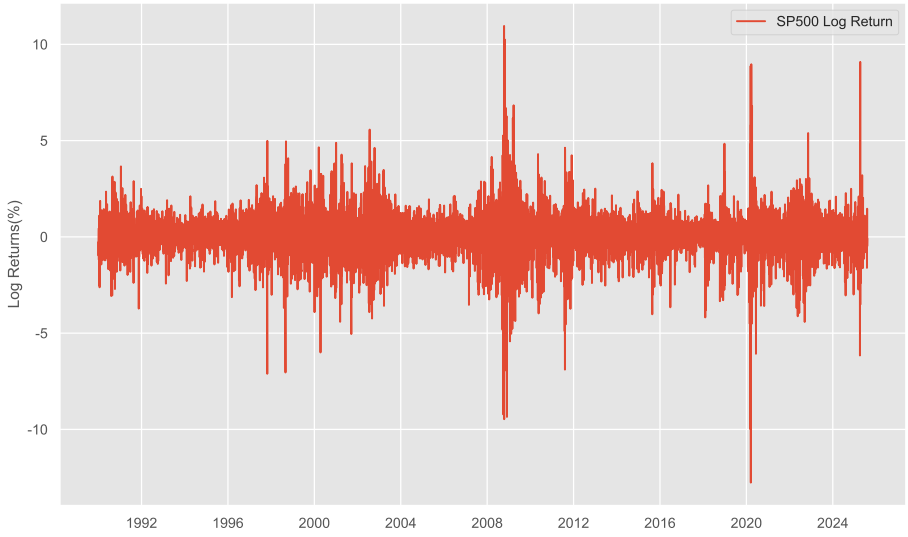
5.2 高斯性质

```
periods = ['D','W','ME']
frequency = ['daily','weekly','monthly']

fig, ax = plt.subplots(2,3,figsize=(12,8))
for i, p in enumerate(periods):
    df_resample = df.resample(p).last()
    log_return = np.log(df_resample['SP500']/df_resample['SP500'].shift(1)).dropna()
    sns.histplot(log_return, kde=True,label='Histogram',ax=ax[0][i])
    ax[0][i].set_title(frequency[i])
    sm.qqplot(log_return, line='s', ax=ax[1][i])
plt.tight_layout()
plt.show()
```



5.3 波动集聚性

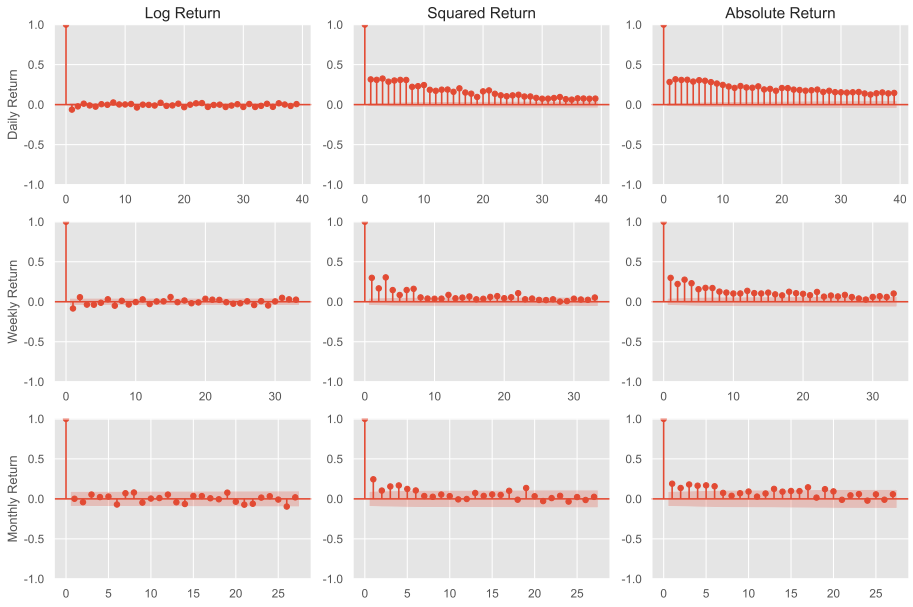
```
fig, ax = plt.subplots(dpi=300,figsize=(10,6))
ax.plot(df['log_ret']*100,
        label='SP500 Log Return')
ax.set_ylabel('Log Returns(%)')
ax.legend()
plt.tight_layout()
plt.show()
```



5.4 自相关

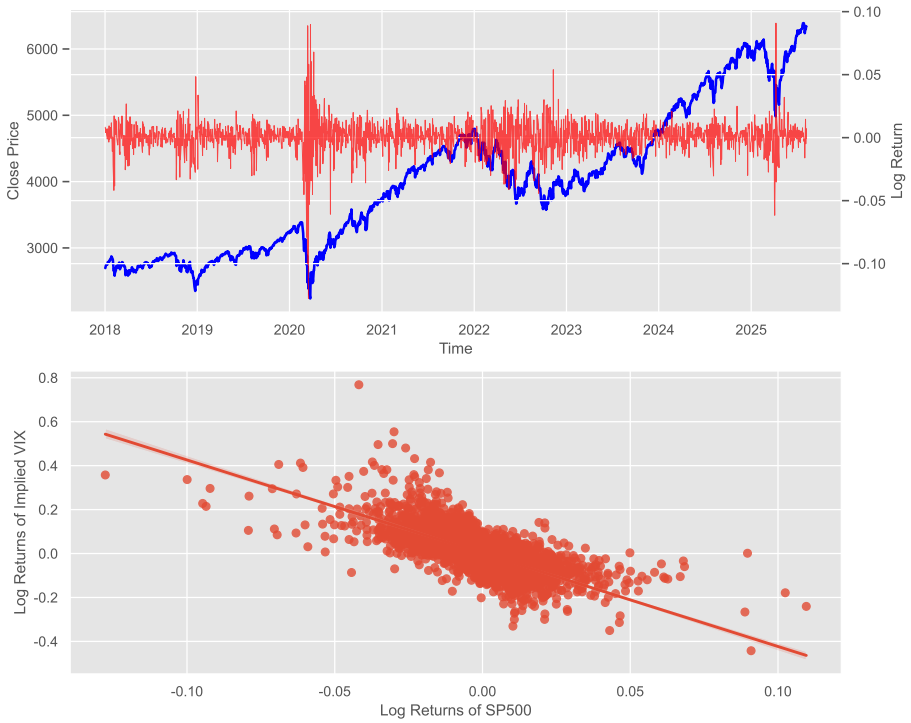
```
from statsmodels.graphics.tsaplots import plot_acf

periods = ['D','W','ME']
frequency = ['daily','weekly','monthly']

fig, ax = plt.subplots(3,3,figsize=(12, 8))
```

```
for i, p in enumerate(periods):
    df_resample = df.resample(p).last()
    log_return = np.log(df_resample['SP500']/df_resample['SP500'].shift(1)).dropna()
    plot_acf(log_return,ax=ax[i][0],title='')
    plot_acf(log_return**2,ax=ax[i][1],title='')
    plot_acf(np.abs(log_return),ax=ax[i][2],title='')
ax[0][0].set_ylabel('Daily Return')
ax[1][0].set_ylabel('Weekly Return')
ax[2][0].set_ylabel('Monthly Return')
ax[0][0].set_title('Log Return')
ax[0][1].set_title('Squared Return')
ax[0][2].set_title('Absolute Return')
plt.tight_layout()
plt.show()
```



5.5 杠杆效应

```
df = pd.read_csv('datasets/sp500_vix.csv',
                 header=0,
                 index_col=0,
                 parse_dates=True)
df.columns = ["SP500", "VIX"]
ret_sp = np.log(df/df.shift(1)).dropna()

fig, ax = plt.subplots(2,1,figsize=(10, 8))
ax[0].plot(df['SP500'].loc["2018":],
             color='blue',lw=2)
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Close Price')
ax2 = ax[0].twinx()
ax2.plot(ret_sp['SP500'].loc["2018":],
             color='red',alpha=0.7,lw=0.5)
ax2.set_ylabel('Log Return')
sns.regplot(x='SP500', y='VIX',
            data=ret_sp, ax=ax[1])
ax[1].set_xlabel('Log Returns of SP500')
ax[1].set_ylabel('Log Returns of Implied VIX')
plt.tight_layout()
plt.show()
```


Close Price
Log Return
Time
2018
2019
2020
2021
2022
2023
2024
2025
Log Returns of Implied VIX
Log Returns of SP500

Bibliography

[1] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[2] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.