Timeseries Data in Pandas

24 Pandas 时间序列数据

时间戳作为索引值，实现对时间序列数据的标记和运算

很难做出预测，尤其是对未来的预测。

It is difficult to make predictions, especially about the future.

—— 尼尔斯·玻尔 (Niels Bohr) | 丹麦物理学家 | 1885 ~ 1962

- `df.bfill()` 向后填充缺失值
- `df.ffill()` 向前填充缺失值
- `df.interpolate()` 插值法填充缺失值
- `df.rolling().corr()` 计算数据帧 `df` 的移动相关性
- `df.rolling().mean()` 计算数据帧 `df` 滚动均值
- `df.rolling().std()` 计算数据帧 `df` MA 平均值
- `joypy.joyplot()` 绘制山脊图
- `numpy.random.uniform()` 生成满足均匀分布的随机数
- `plotly.express.bar()` 绘制可交互条形图
- `plotly.express.histogram()` 绘制可交互直方图
- `plotly.express.imshow()` 绘制可交互热图
- `plotly.express.line()` 绘制可交互二维线图
- `plotly.express.scatter()` 绘制可交互散点图
- `seaborn.heatmap()` 绘制热图
- `statsmodels.api.tsa.seasonal_decompose()` 季节性调整
- `statsmodels.regression.rolling.RollingOLS()` 计算移动 OLS 线性回归系数

24.1 什么是时间序列?

时间序列 (timeseries) 是指按照时间顺序排列的一系列数据点或观测值，通常是等时间间隔下的测量值，如每天、每小时、每分钟等。时间序列数据通常用于研究时间相关的现象和趋势，例如股票价格、气象数据、经济指标等。图 1 (a) 所示为标普 500 (S&P 500) 数据。

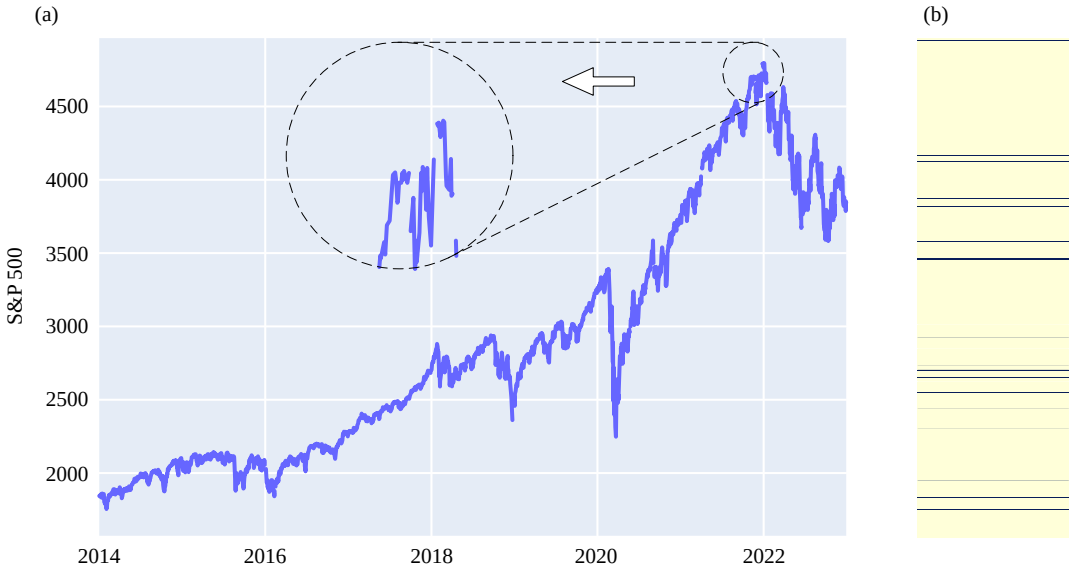

图 1. 标普 500 数据，含有缺失值

时间序列分析是一种重要的数据分析方法，它可以用于预测未来的趋势和变化，评估现有趋势的稳定性和可靠性，并发现异常点和异常趋势。时间序列分析通常包括以下几个步骤：

- 数据预处理：对数据进行清洗、去噪、填补缺失值等操作，以提高数据质量和可靠性。
- 时间序列的可视化：对数据进行绘图，以了解数据的分布、趋势和周期性。
- 时间序列的统计分析：对数据进行时间序列分解、平稳性检验、自相关性检验等统计分析，以评估数据的稳定性和相关性。
- 时间序列的建模和预测：根据统计分析的结果，建立合适的时间序列模型，进行未来趋势的预测和评估。

比如，在图 1 (a) 中被局部放大的曲线上，大家已经看到了缺失值。图 1 (b) 用热图可视化缺失值的位置。在本章配套的代码中，大家会看到经过计算缺失值的占比约为 3.5%。

本章仅仅采用“图解”介绍部分时间序列分析，《数据有道》一册将专门介绍时间序列相关话题。

```python
# 导入包
import pandas_datareader as pdr
# 需要安装 pip install pandas_datareader
import joypy
# 需要安装 conda install joypy
import pandas as pd
import datetime
import plotly.express as px
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
pd.options.mode.chained_assignment = None  # default='warn'

# 从FRED下载标普500 (S&P 500)
start_date = datetime.datetime(2014, 1, 1)
end_date   = datetime.datetime(2022, 12, 31)

ticker_list = ['SP500']
df = pdr.DataReader(ticker_list,
                    'fred',
                    start_date,
                    end_date)
# 双备份数据
df.to_csv('SP500_' + str(start_date.date()) + '_'
          + str(end_date.date()) + '.csv')
df.to_pickle('SP500_' + str(start_date.date()) + '_'
             + str(end_date.date()) + '.pkl')
# 从备份数据导入
# df = pd.read_csv('SP500_2014-01-01_2022-12-31.csv',
#                  index_col=0, parse_dates=True)
# df = pd.read_pickle('SP500_2014-01-01_2022-12-31.pkl')
```

图 2. 下载金融数据

```
# 含有缺失值的时间序列线图

fig = px.line(df)
fig.update_layout(xaxis_title = 'Date',
                  yaxis_title = 'S&P 500 index',
                  legend_title = 'Curve',
                  showlegend=False)
fig.show()

# 计算缺失值比例
percentag_missing = df.isnull().sum()*100/len(df)
print('Percentage of missing data')
print("%.3f%%" % (percentag_missing))

# 可视化缺失值
fig, ax = plt.subplots(figsize = (2,4))

sns.heatmap(df.isnull(), cbar = False,
            cmap = 'YlGnBu', yticklabels = [])
plt.show()
```

图 3. 可视化缺失值，使用时配合前文代码

Pandas 中的时间序列功能

在 Python 中，Pandas 库提供了强大的时间序列处理和分析功能，使得时间序列的处理和分析变得更加简单和高效。在 Pandas 中，时间序列分析的主要方法包括：

- 创建时间序列：可以通过 pandas.date_range() 方法创建一个时间范围，或者将字符串转换为时间序列对象。
- 时间序列索引：可以使用时间序列作为 DataFrame 的索引，从而方便地进行时间序列分析。
- 时间序列的切片和索引：可以使用时间序列的标签或位置进行切片和索引。
- 时间序列的重采样：可以将时间序列转换为不同的时间间隔，例如将日频率的数据转换为月频率的数据。
- 移动窗口函数：可以对时间序列数据进行滑动窗口操作，计算滑动窗口内的统计指标，例如均值、方差等。
- 时间序列的分组操作：可以将时间序列数据按照时间维度进行分组，从而进行聚合操作，例如计算每月的平均值、最大值等。
- 时间序列的聚合操作：可以对时间序列数据进行聚合操作，例如计算每周、每月、每季度的总和、平均值等。
- 时间序列的可视化：可以使用 Pandas、Matplotlib、Seaborn、Plotly 等库对时间序列数据进行可视化，例如绘制线形图、散点图、直方图等。

24.2 缺失值

缺失值 (missing value) 指的是数据集中的某些值缺失或未被记录的情况。它们可能是由于测量设备故障、记录错误、样本丢失或数据清洗不完整等原因导致的。缺失值可能在数据分析和建模中产生严重的影响，因为它们会导致数据样本的大小不一致，使得数据的统计分布和关系不准确或无法得出。另外，许多机器学习算法无法处理缺失值，必须对其进行处理或者删除。

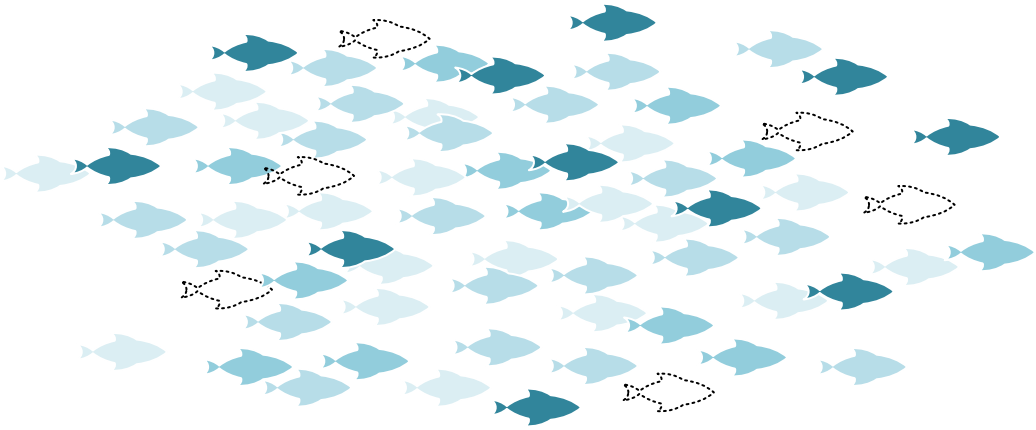

图 4. 缺失值

在数据处理中，通常需要对缺失值进行识别、处理或删除。一些处理缺失值的方法包括：

- 删除带有缺失值的样本或变量。
- 使用常量填充缺失值，例如用零、平均值、中位数等常量填充。
- 使用回归模型、插值方法等技术，对缺失值进行预测和填充。
- 对于分类变量，可以创建一个新的类别来表示缺失值。

在选择处理方法时，需要根据具体情况和数据分析的目的来决定。

图 1 中的缺失值则对应非营业日，比如周六日、节假日等。将这些缺失值删除之后，我们便得到图 5 所示的趋势。为了醒目地观察每年趋势，我们绘制了图 7。

鸢尾花书《数据有道》将介绍处理缺失值的各种方法。

图 5. 标普 500 数据，删除缺失值

```
# 删除NaN
a df_ = df.dropna()
b percentag_missing = df_.isnull().sum()*100/len(df)
# 再次确认缺失值比例
print('Percentage of missing data')
print("%.3f%%" % (percentag_missing))

# 删除缺失值的时间序列线图
c fig = px.line(df_, y = 'SP500', title = 'S&P 500 index')
fig.show()
```

图 6. 删除缺失值，使用时配合前文代码

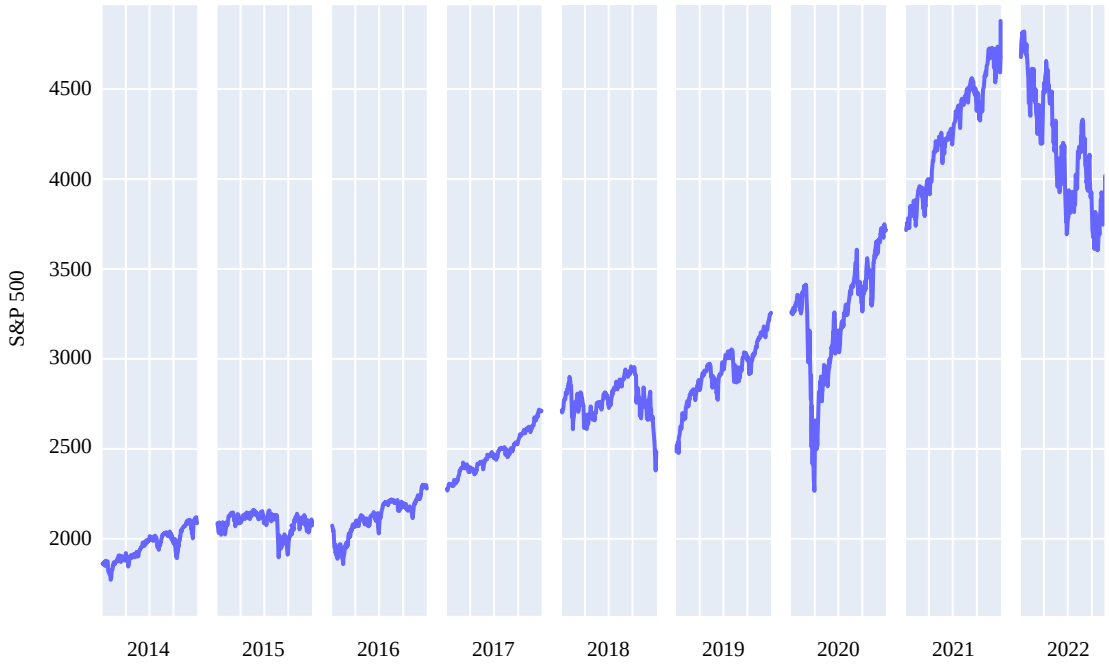

图 7. 标普 500 数据，按年观察趋势

```
# 按年度分图展示时间序列趋势
a df_['Year'] = pd.DatetimeIndex(df_.index).year
b fig = px.line(df_, y = 'SP500', title = 'S&P 500 index',
                facet_col='Year', facet_row=None)

c fig.update_layout(
    width=700,
    height=500,
    margin=dict(l=20, r=20, t=30, b=20),
    paper_bgcolor="white")

d fig.update_xaxes(matches=None)
fig.show()
```

图 8. 按年绘制标普 500 水平子图，使用时配合前文代码

什么是离群值?

在统计学和数据分析中，离群值 (outlier) 指的是在数据集中与其他数据值显著不同的异常值。它们可能是由于测量误差、实验异常、录入错误、样本损坏或数据处理错误等因素导致的。离群值具有比其他数据点更大或更小的数值，与其他数据点之间的差异通常非常显著。

离群值会对数据分析结果产生影响，比如对平均值、方差、相关性等统计指标的计算都会受到其影响。因此，在数据分析和建模中，需要对离群值进行识别、处理或删除。常见的方法包括使用箱线图或 3σ 准则等方法来识别离群值，并根据具体情况进行处理或删除。如果离群值确实是数据中真实存在的异常值，则可能需要对其进行单独分析或建立针对其的模型。

本章不会介绍如何处理离群值，相关内容请参考《数据有道》。

24.3 移动平均

时间序列的移动平均 (moving average, MA) 是一种常用的平滑技术，用于去除序列中的噪声和波动，以便更好地观察和分析序列的长期趋势。

移动平均通过计算序列中一段固定长度（通常称为窗口）内数据点的平均值来平滑序列。窗口的大小决定了平滑的程度，较大的窗口将平滑更多的波动，但可能会导致较长的滞后。

具体步骤如下：

- 1) 选择窗口的大小，例如 10 个数据点。
- 2) 从序列的起始位置开始，计算窗口内数据点的平均值。
- 3) 将该平均值作为移动平均的第一个数据点，记录下来。
- 4) 移动窗口向后滑动一个数据点的位置。
- 5) 重复步骤 2 至 4，计算新窗口内的平均值，并记录下来。
- 6) 继续滑动窗口直到到达序列的末尾，得到一系列移动平均值。

移动平均的计算可以使用简单移动平均 (Simple Moving Average, SMA) 或加权移动平均 (Weighted Moving Average, WMA) 来进行。简单移动平均对窗口内的每个数据点赋予相等的权重，而加权移动平均则可以根据需求赋予不同的权重，以更强调某些数据点的重要性。

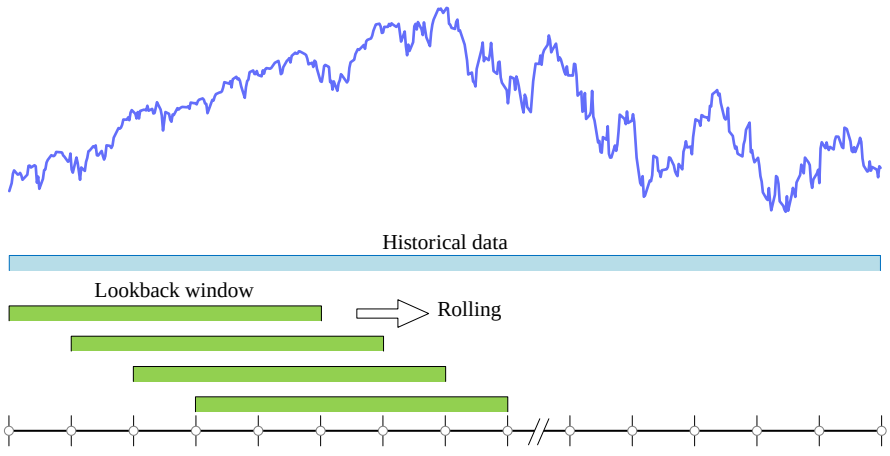

图 9. 移动窗口

通过计算移动平均，时间序列中的短期波动可以平滑，从而更容易观察到长期趋势和周期性变化。移动平均在金融分析、经济预测和数据分析等领域得到广泛应用。

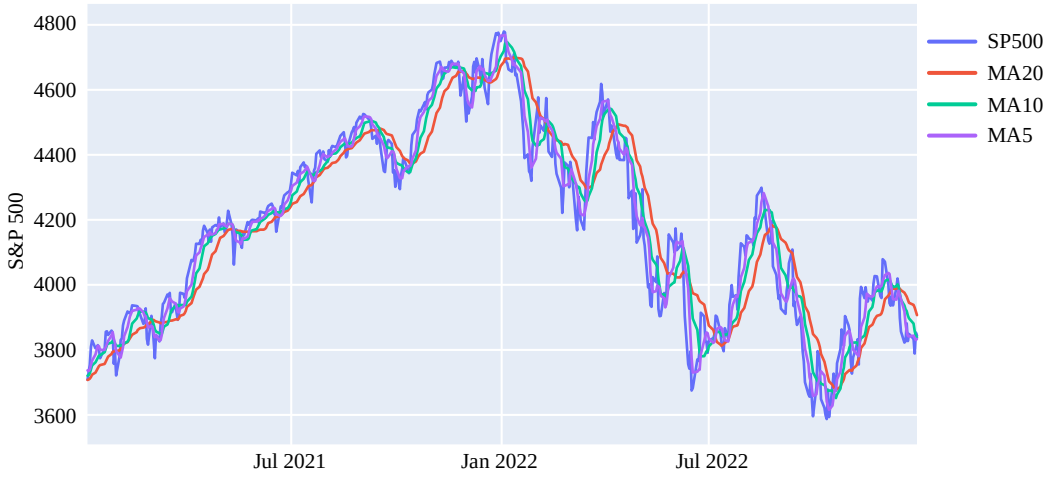

图 10. 标普 500 数据，移动平均

```
# 计算三种移动平均
a df_['MA20'] = df_['SP500'].rolling(20).mean()
b df_['MA10'] = df_['SP500'].rolling(10).mean()
c df_['MA5'] = df_['SP500'].rolling(5).mean()
d df_selected = df_[['SP500','MA20','MA10','MA5']]
e fig = px.line(df_selected.loc['20210101':'20221231'])
fig.update_layout(title = 'S&P 500 index and moving average',
                  xaxis_title = 'Date',
                  yaxis_title = 'S&P500',
                  legend_title = 'Curve')

d fig.show()
```

图 11. 绘制标普 500 移动平均，使用时配合前文代码

24.4 收益率

为了量化股票市场的每日涨跌，我们需要计算股票的日收益率。计算当日收益率时需要知道两个关键数据点：股票的当日收盘价、前一日收盘价。

日收益率的计算公式为：日收益率 = (当日收盘价 − 前一日收盘价) / 前一日收盘价。将这个公式应用于具体的股票数据，就可以计算出每个交易日的日收益率。图 12 所示为标普 500 的日收益率。

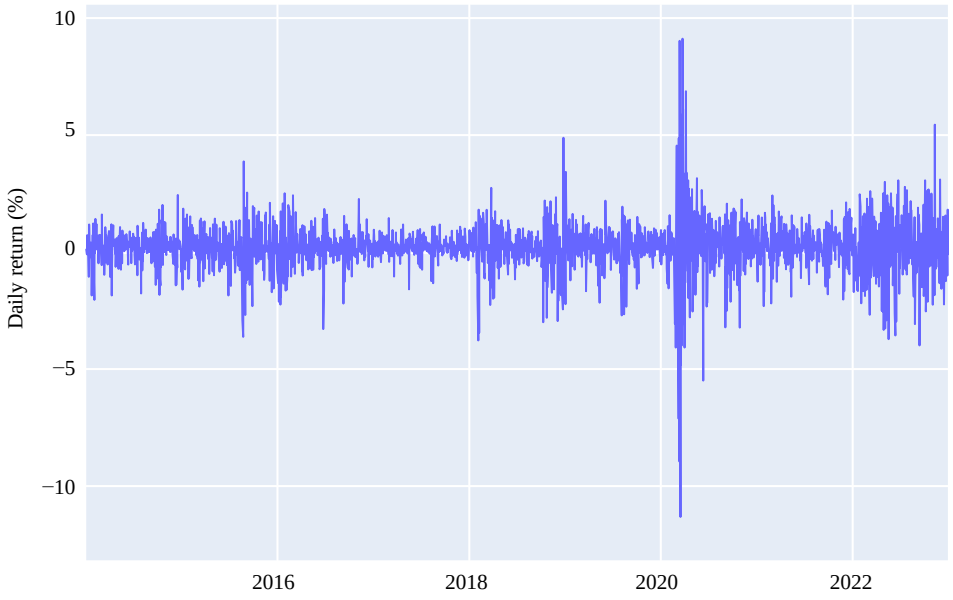

图 12. 标普 500 数据日收益率

```
a df_['daily_r'] = df_['SP500'].pct_change() * 100
  # 计算日收益率
  # 小数转为百分数

b fig = px.line(df_, y = 'daily_r')
c fig.update_layout(title = 'Daily relative return',
                    xaxis_title = 'Date',
                    yaxis_title = 'Daily return (%)',
                    legend_title = 'Curve')

  fig.show()
```

图 13. 绘制标普 500 日收益率，使用时配合前文代码

为了更方便观察每年涨跌情况，我们绘制了图 14。

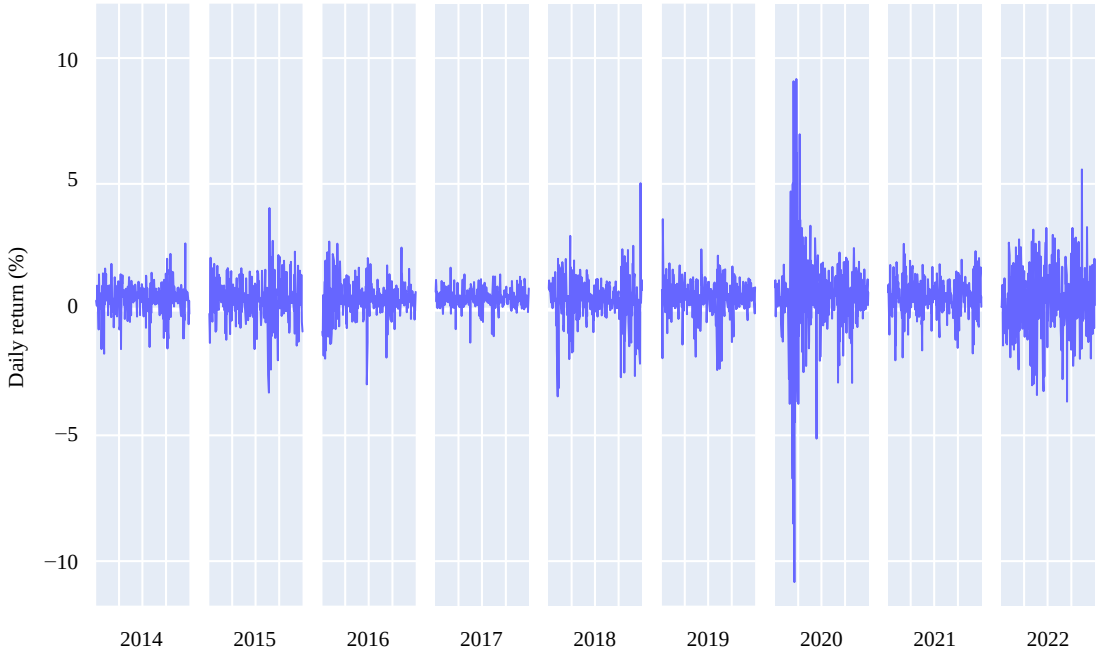

图 14. 标普 500 数据日收益率，按年观察趋势

```
# 日收益率，按年子图
fig = px.line(df_,y = ['daily_r'], title = 'S&P 500 index',
              facet_col='Year', facet_row=None)

fig.update_layout(
    width=700,
    height=500,
    margin=dict(l=20, r=20, t=30, b=20),
    paper_bgcolor="white")

fig.update_xaxes(matches=None)
```

图 15. 按年绘制标普 500 收益率子图，使用时配合前文代码

24.5 统计分析

市场涨跌越越剧烈，曲线波动越剧烈。图 14 这些曲线类似随机行走，为了发现规律，我们需要借助统计工具。

年度分布

图 16 所示为下载所有数据计算得到日收益率绘制的分布图。大家可以从分布中计算得到均值和标准差。这个任务交给大家自行完成。图 18 所示为年度日收益率分布变化情况。

为了更好的量化股票的波动情况，我们需要一个指标——波动率 (volatility)。波动率是衡量其价格变动幅度的指标，常用的量化方法为历史波动率 (historical volatility)。历史波动率本质上就是一定回望窗口内收益率样本数据的标准差。

图 20 所示为利用水平柱状图可视化日收益率的年度均值、波动率 (标准差)。

此外，我们还可以使用山脊图 (ridgeline plot) 可视化每年收益率的分布情况，具体如图 22 所示。Joypy 是一个 Python 库，用于创建山脊图。山脊图是一种可视化工具，用于展示多个连续变量在一个维度上的分布，并且能够显示不同组之间的比较。

山脊图的特点是将多个曲线图，通常是核密度估计曲线，沿着一个共享的垂直轴线堆叠显示，形成一座山脉状的图形。每个曲线代表一个组或类别，可以通过颜色或其他视觉属性进行区分。要使用 Joypy 绘制山脊图，需要首先安装 Joypy 库，并导入 joyplot 模块。

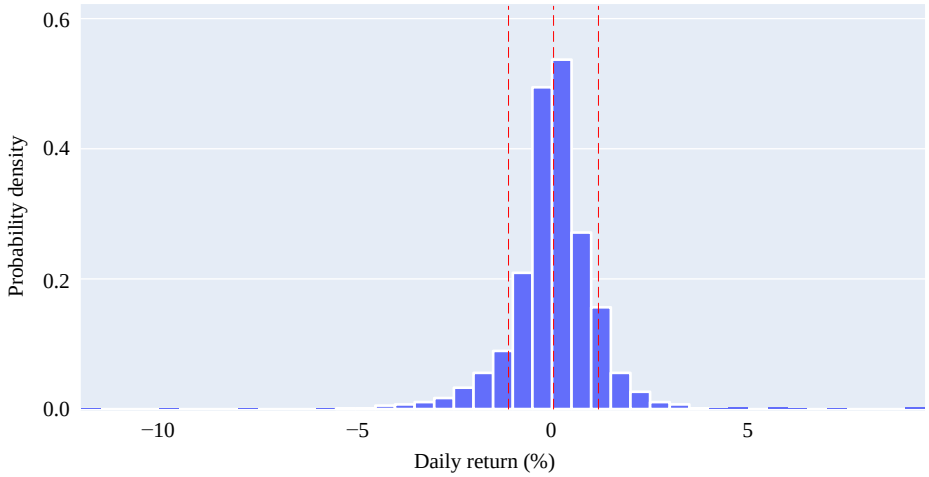

图 16. 所有下载历史数据日收益率分布

```python
# 计算均值和标准差
mean = np.mean(df_['daily_r'])
std = np.std(df_['daily_r'])

# 绘制直方图
fig = px.histogram(df_['daily_r'], nbins=50,
                   histnorm='probability density')

# 标注均值和均值加减标准差的位置
fig.add_shape(type='line', x0=mean, y0=0,
              x1=mean, y1=1,
              line=dict(color='red', dash='dash'),
              name='mean')

fig.add_shape(type='line', x0=mean+std, y0=0,
              x1=mean+std, y1=1,
              line=dict(color='red', dash='dash'),
              name='mean+std')

fig.add_shape(type='line', x0=mean-std, y0=0,
              x1=mean-std, y1=1,
              line=dict(color='red', dash='dash'),
              name='mean-std')

# 设置图形布局
fig.update_layout(showlegend=False,
                  xaxis_title = 'Daily return (%)',
                  yaxis_title = 'Probability density')

# 显示图形
fig.show()
```

图 17. 日收益率分布，使用时配合前文代码

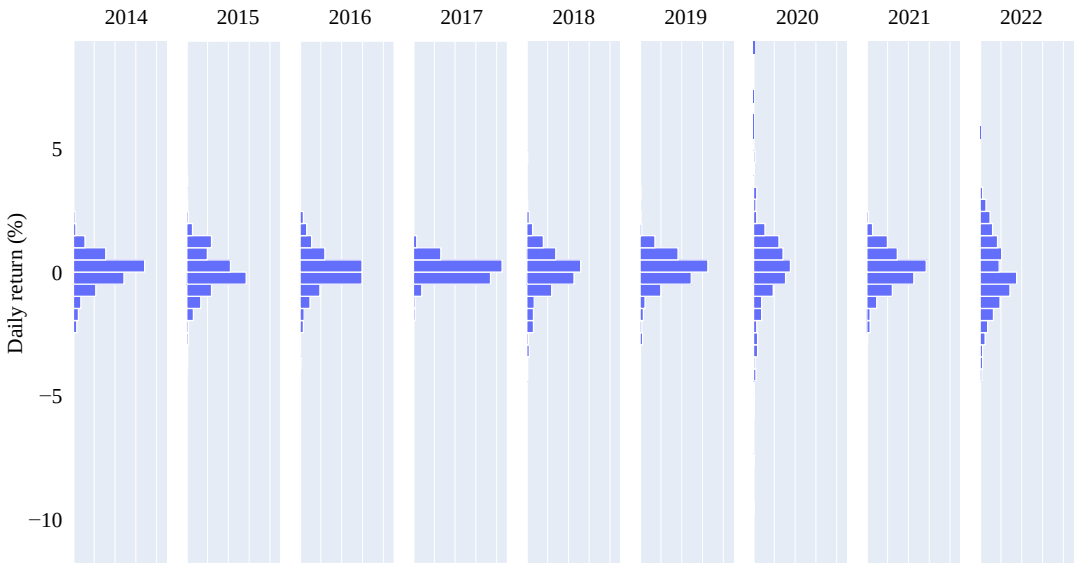

图 18. 日收益率分布，按年

```
# 每年收益率直方图
a fig = px.histogram(df_[['daily_r','Year']], nbins=80,
                     histnorm='probability density',
                     title = 'S&P 500 index',
                     orientation='h',
                     facet_col='Year', facet_row=None)

b fig.update_layout(
    width=1200,
    height=500,
    margin=dict(l=20, r=20, t=30, b=20),
    paper_bgcolor="white")

fig.update_xaxes(matches='x')
fig.show()
```

图 19. 按年日收益率直方图子图，使用时配合前文代码

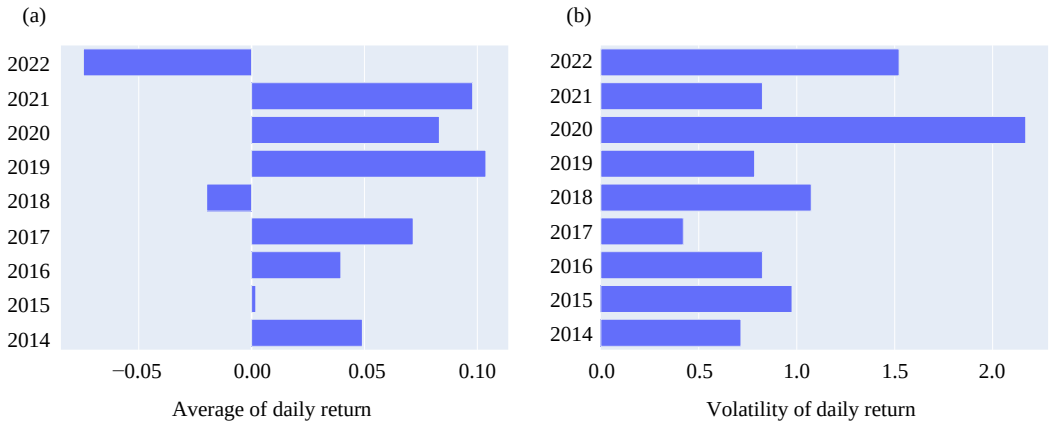

图 20. 水平柱状图可视化收益率均值、标准差 (波动率)，按年

```
# 年度统计数据
Yearly_stats_df = df_.groupby(['Year'],
         as_index=False).agg({'daily_r':['mean','std']})

# 使用plotly.express绘制条形图
fig = px.bar(y=Yearly_stats_df['Year'],
             x=Yearly_stats_df['daily_r']['mean'],
             title='Mean', orientation='h')

# 设置图形布局
fig.update_layout(showlegend=False,
                  xaxis_title = 'Mean of daily return (%)',
                  yaxis_title = 'Year')
# 显示图形
fig.show()

      # 使用plotly.express绘制条形图
fig = px.bar(y=Yearly_stats_df['Year'],
             x=Yearly_stats_df['daily_r']['std'],
             title='Daily vol',
             orientation='h')

# 设置图形布局
fig.update_layout(showlegend=False,
                  xaxis_title = 'Volatility of daily return (%)',
                  yaxis_title = 'Year')
# 显示图形
fig.show()
```

图 21. 日收益率年度统计数据柱状图，使用时配合前文代码

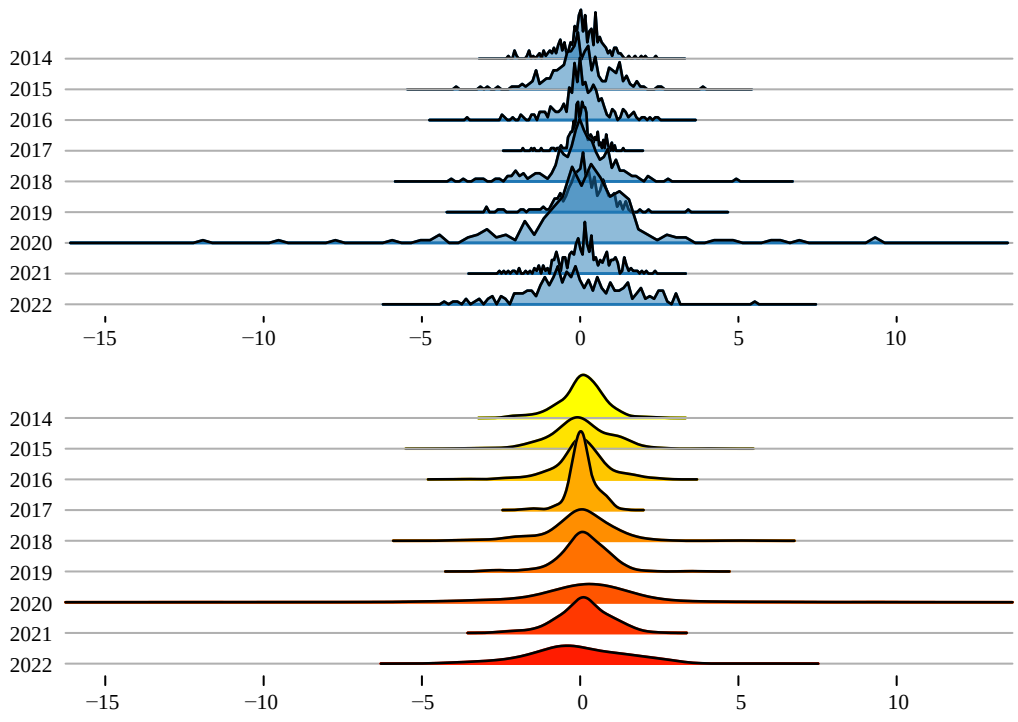

图 22. 山脊图，按年

```
a df_['Year'] = pd.DatetimeIndex(df_.index).year
  # 绘制山脊图
  from matplotlib import cm

  fig, ax = plt.subplots(figsize = (6,4))
  # 频率
b joypy.joyplot(df_, by="Year", ax = ax,
                column="daily_r", range_style='own',
                grid="y", linewidth=1, legend=False,
                fade=True,kind="counts", bins=100)
  plt.show()

  # 高斯概率密度估计
  fig, ax = plt.subplots(figsize = (6,4))
c joypy.joyplot(df_, by="Year", column="daily_r", ax = ax,
                range_style='own', grid="y",
                linewidth=1, legend=False,
                colormap=cm.autumn_r, fade=True)
  plt.show()
```

图 23. 绘制年度数据山脊图，使用时配合前文代码

季度分布

当然，我们也可以按季度分析收益率。图 24 所示为每个季度收益率的均值、标准差的柱状图。图 26 所示为每个季度收益率的山脊图。这幅图我们把纵轴的时间隐去。

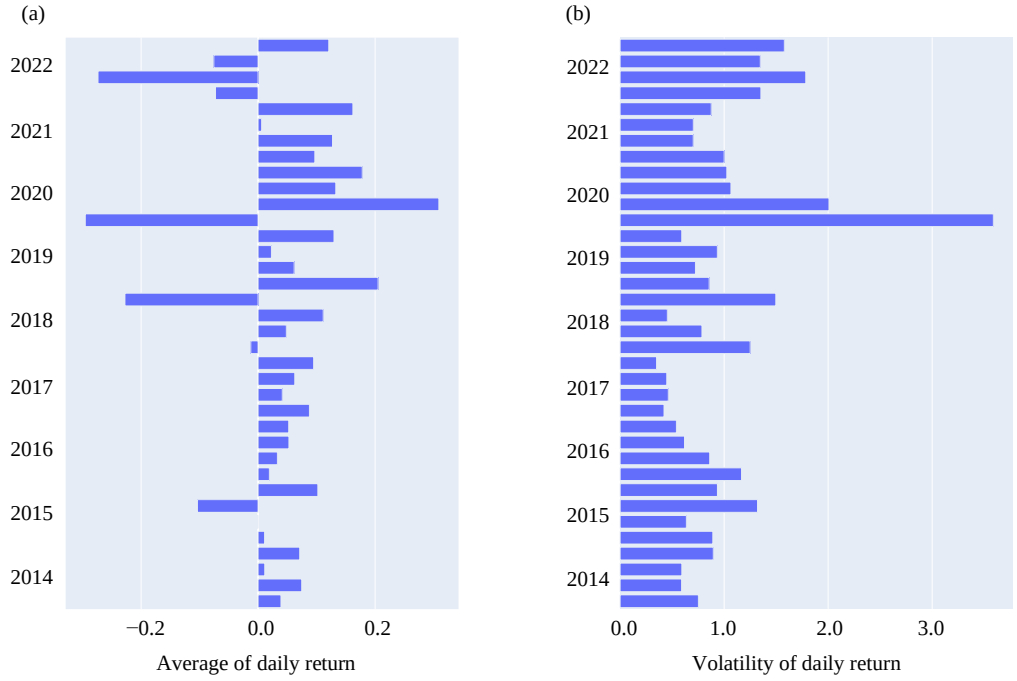

图 24. 水平柱状图可视化收益率均值、标准差 (波动率)，按季度

```
# 季度均值、标准差
df_['Quarter'] = pd.DatetimeIndex(df_.index).quarter

quarters = df_.index.quarter
years = df_.index.year

# 将季度和年份信息组合成字符串
quarters = [f"{year}, Q{quarter}"
            for year, quarter in zip(years, quarters)]

# 添加新列 "Quarter"
df_['Quarter'] = quarters

Qly_stats_df = df_.groupby(['Quarter'],
    as_index=False).agg({'daily_r':['mean','std']})

# 使用plotly.express绘制条形图
fig = px.bar(y=Qly_stats_df['Quarter'],
             x=Qly_stats_df['daily_r']['mean'],
             title='Mean',
             orientation='h')

# 设置图形布局
fig.update_layout(showlegend=False,
                  width = 600,
                  height = 800,
                  xaxis_title = 'Mean of daily return (%)',
                  yaxis_title = 'Quarter')
# 显示图形
fig.show()

# 使用plotly.express绘制条形图
fig = px.bar(y=Qly_stats_df['Quarter'],
             x=Qly_stats_df['daily_r']['std'],
             title='Volatility',
             orientation='h')

# 设置图形布局
fig.update_layout(showlegend=False,
                  width = 600,
                  height = 800,
                  xaxis_title = 'Vol of daily return (%)',
                  yaxis_title = 'Quarter')
# 显示图形
fig.show()
```

图 25. 绘制季度统计量柱状图，使用时配合前文代码

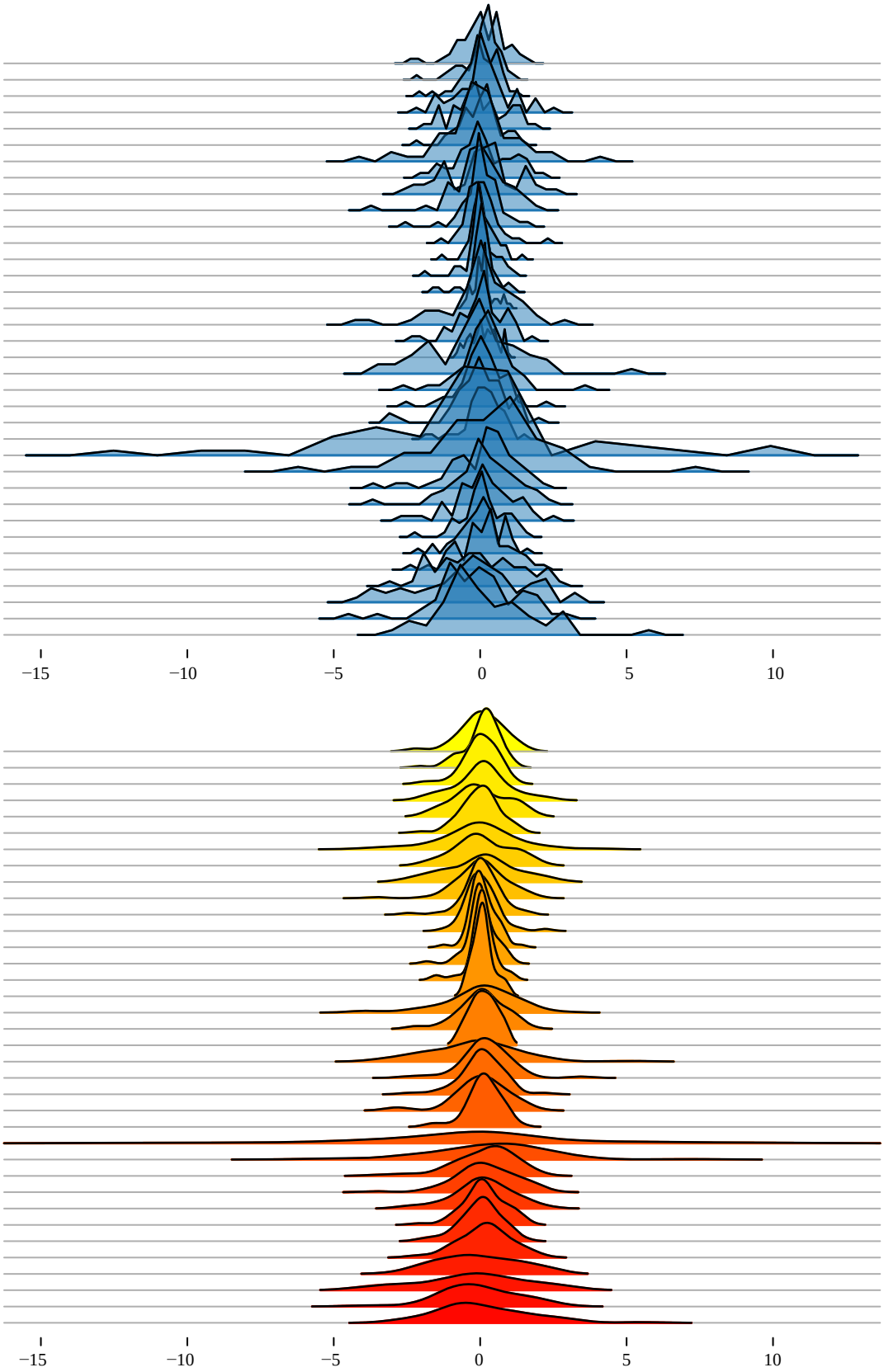

图 26. 山脊图，按季度

```
# 季度数据
# 默认效果山脊图
fig, ax = plt.subplots(figsize = (6,5))

joypy.joyplot(df_, by="Quarter", ax = ax,
              column="daily_r", range_style='own',
              grid="y", linewidth=1, legend=False,
              fade=True,kind="counts", bins=20)
plt.show()

# KDE
fig, ax = plt.subplots(figsize = (6,5))
joypy.joyplot(df_, by="Quarter", column="daily_r", ax = ax,
              range_style='own', grid="y",
              linewidth=1, legend=False,
              colormap=cm.autumn_r, fade=True)
plt.show()
```

图 27. 绘制季度数据山脊图，使用时配合前文代码

移动波动率

准确来说，历史波动率是根据过去一段时间内的股票价格数据计算得出的波动率。

可以选择一个时间窗口，例如 20 营业日 (一个月)、60 营业日 (一个季度)、125 或 126 营业日 (半年)、250 或 252 营业日 (一年)，计算每个交易日的收益率，然后求得其标准差，最终得到历史波动率。当这个回望窗口移动时，我们便得到移动波动率的时间序列数据。图 28 所示的移动波动率的回望窗口长度为 250 天营业日。请大家自己修改回望窗口长度 (营业日数量)，比较移动波动率曲线。

《数据有道》还会专门介绍指数加权移动平均 EWMA 方法计算的均值和波动率。

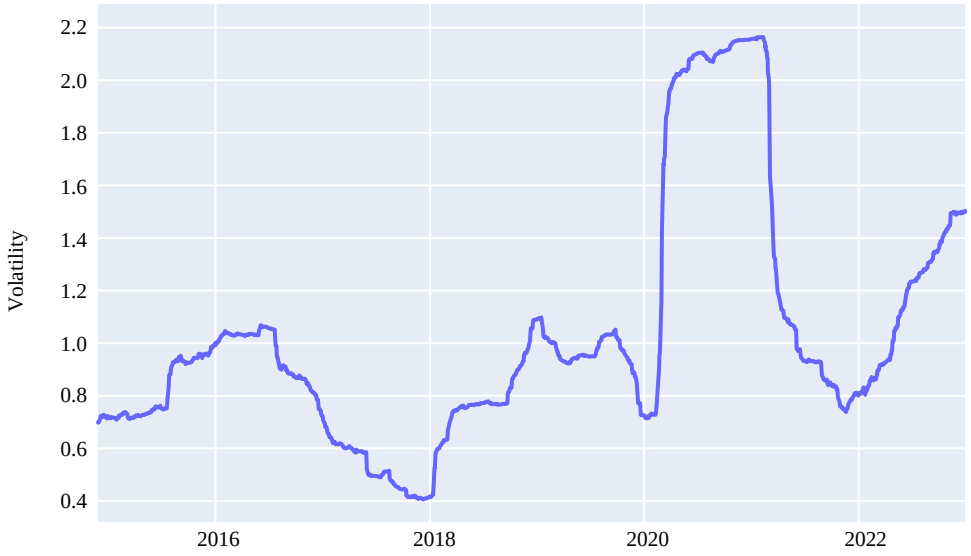

图 28. 移动波动率

```
# 滚动波动率
a df_vol = df_['daily_r'].rolling(250).std()

b fig = px.line(df_vol, y = 'daily_r')
fig.update_layout(title = 'Rolling vol',
                  xaxis_title = 'Date',
                  yaxis_title = 'Volatility')

fig.show()
```

图 29. 绘制滚动波动率，使用时配合前文代码

24.6 相关性

几个不同时间序列之间肯定也会存在相关性。图 30 所示为标普 500 日收益率和三个汇率收益率之间的相关性系数矩阵热图。

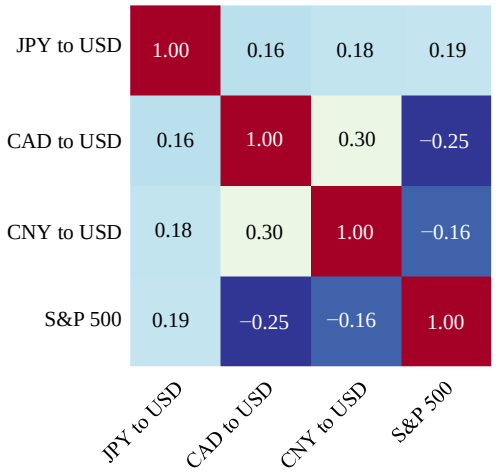

图 30. 相关性系数矩阵

相关性并不是一成不变的，也是随时间不断变化。如图 32 所示，当我们指定具体的移动窗口长度，也可以计算移动相关性。

```
# 下载更多数据
a ticker_list = ['DEXJPUS','DEXCAUS','DEXCHUS','SP500']
b df_FX_SP500 = pdr.DataReader(ticker_list,
                               'fred',
                               start_date,
                               end_date)

# 备份数据
df_FX_SP500.to_csv('FX_SP500_' + str(start_date.date()) + '_'
                   + str(end_date.date()) + '.csv')
df_FX_SP500.to_pickle('FX_SP500_' + str(start_date.date()) + '_'
                      + str(end_date.date()) + '.pkl')

# 修改column names
c df_FX_SP500 = df_FX_SP500.rename(columns={'DEXJPUS': 'JPY to USD',
                                            'DEXCAUS': 'CAD to USD',
                                            'DEXCHUS':'CNY to USD'})
d df_FX_SP500_return = df_FX_SP500.dropna().pct_change()

# 相关性矩阵热图
e fig = px.imshow(df_FX_SP500_return.corr(),
                  text_auto = '.2f',
                  color_continuous_scale = 'RdYlBu_r')
fig.show()
```

图 31. 可视化相关性系数矩阵，使用时配合前文代码

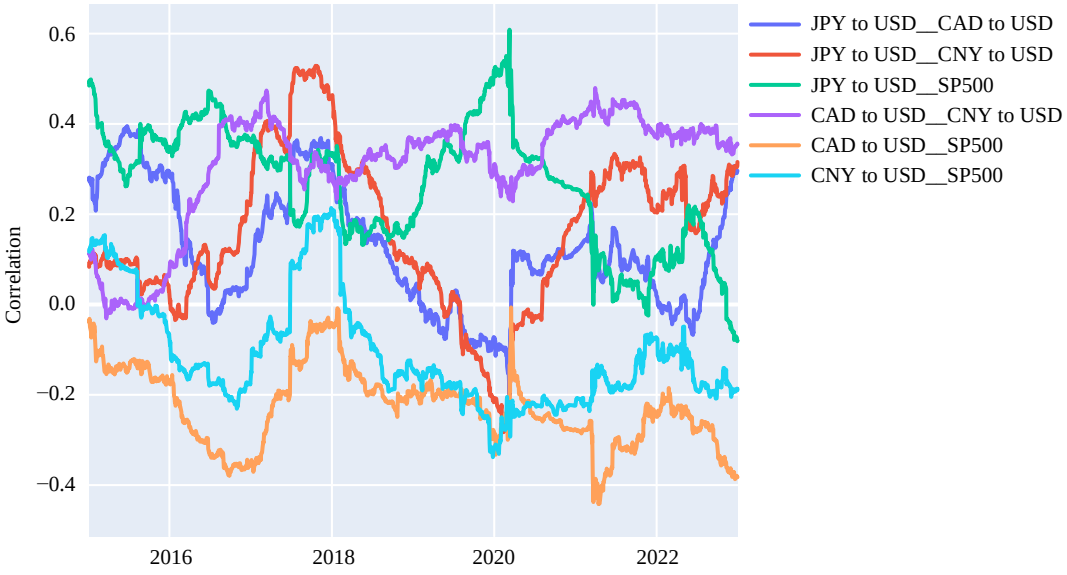

图 32. 移动相关性

```
# 计算滚动相关性系数
a df_roll_corr = df_FX_SP500_return.rolling(250).corr()

# 整理数据
b df_roll_corr_ = df_roll_corr.unstack()
c df_roll_corr_.columns=df_roll_corr_.columns.get_level_values(0)
d df_roll_corr_.columns = ['_'.join(col).strip()
                           for col in
                           df_roll_corr.unstack().columns.values]
e df_roll_corr_ = df_roll_corr_.dropna()

# 保留成对相关性数据
f from itertools import combinations
g list_tickers = list(df_FX_SP500_return.columns)

h pairs_kept = ['_'.join(combo)
                for combo in combinations(list_tickers,2)]
i df_roll_corr_ = df_roll_corr_[pairs_kept]

# 可视化
j fig = px.line(df_roll_corr_)

fig.update_layout(xaxis_title = 'Date',
                  yaxis_title = 'corr',
                  legend_title = 'Pair')

fig.show()
```

图 33. 可视化移动相关性系数，使用时配合前文代码

对时间序列历史数据完成分析后自然少不了预测这个环节。本书不会展开讲解，请大家参考《数据有道》。

请大家完成下面这道题目。

Q1. 请大家把本章配套代码中历史数据截止时间修改为最近日期，重新下载数据逐步完成本章前文时间序列分析。

* 本章不提供答案。

有关 Pandas 中时间序列更多用法，请大家参考：

https://pandas.pydata.org/docs/user_guide/timeseries.html

此外，Statsmodels 有大量时间序列分析工具：

https://www.statsmodels.org/stable/user-guide.html#time-series-analysis