

## 24

## Timeseries Data in Pandas

## Pandas 时间序列数据

时间戳为行索引值，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。

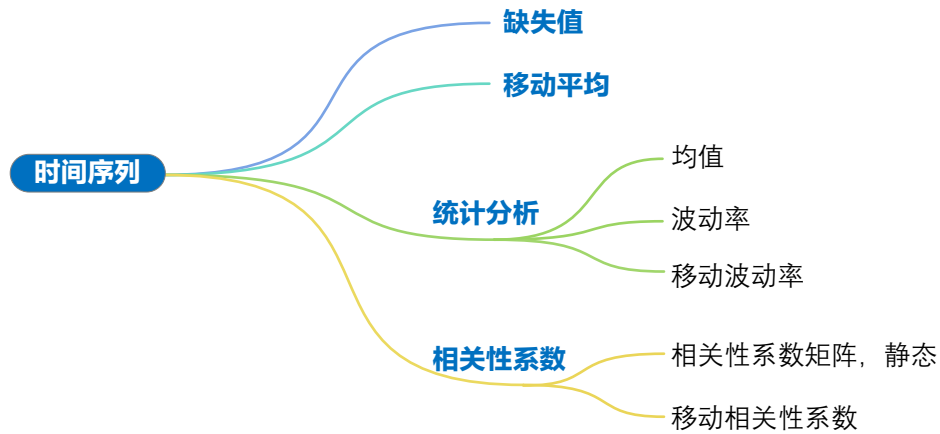
*It is difficult to make predictions, especially about the future.*

—— 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` 移动均值
- ◀ `df.rolling().std()` 计算数据帧 `df` MA 平均值
- ◀ `joyplot.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？

本书前文介绍过，**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，比如图 1（a）所示为标普 500（S&P 500）数据。

本章介绍 Pandas 中常用的时间序列处理和分析工具。

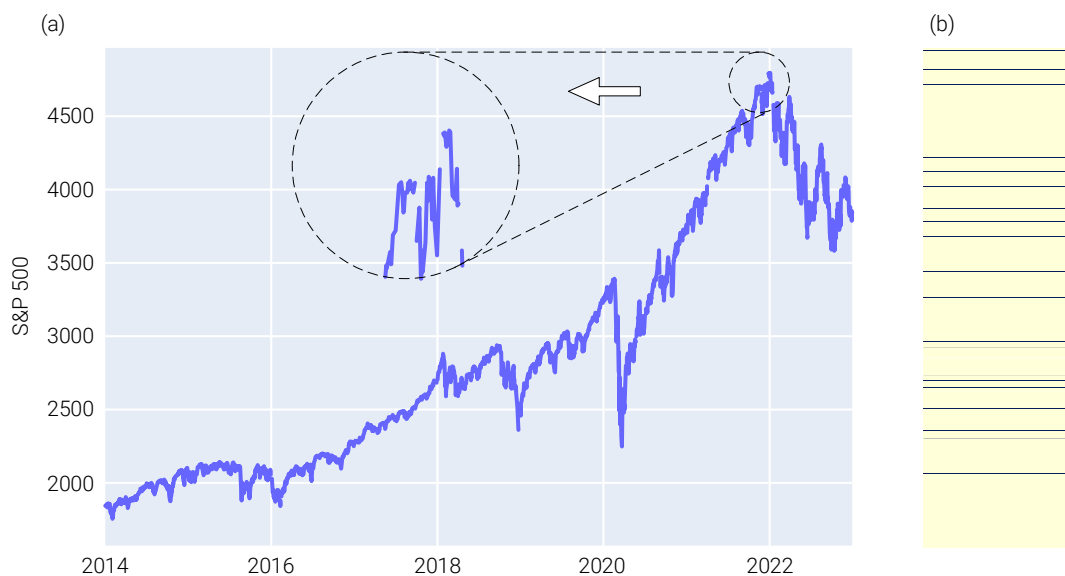


图 1. 标普 500 数据，含有缺失值

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**（missing values）。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%。

### 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。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。


版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

- ▶ 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。

 本章仅仅采用“图解”介绍最基本的时间序列分析方法，《数据有道》一册将专门介绍时间序列相关话题。

## 下载 + 可视化数据

代码 1 下载金融数据，大家应该对这部分代码很熟悉了，下面简单介绍关键语句，也请回顾本书前文讲过的相关内容。

**a** 将 `pandas_datareader` 导入，别名为 `pdr`。`pandas_datareader` 是一个用于从多种数据源中获取金融和经济数据的库。在安装 `Anaconda` 时，这个库并没有安装，请大家自行安装（`pip install pandas-datareader`）。此外，请大家注意这个库的更新情况。

 如果下载数据失败的话，请大家用 `pandas.read_csv()` 读入配套 CSV 数据。

**b** 将名为“`joypy`”的 Python 库导入。本章后文，我们将用 `joypy` 绘制山脊图（`ridgeline plot`）。这个库也需要大家自行安装（`conda install joypy`）。

 如果忘记怎么安装第三方 Python 库，请大家参考本书第 2 章。

**c** 导入 Python 的 `datetime` 库。`datetime` 模块提供了处理日期和时间的功能，包括创建日期时间对象、执行日期时间运算、格式化日期时间字符串等。

**d** 关闭链式赋值操作警告。

**e** 用 `datetime.datetime()` 创建日期对象。

**f** 指定下载数据的标识符 ID。

**g** 利用 `pdr.DataReader()` 从 FRED 下载数据。本书第 19 章还介绍过其他下载方法，请大家回顾。

**h** 将数据帧保存为 `.csv` 文件。

**i** 将数据帧保存为 `.pkl` 文件。如果读者无法从 FRED 官方下载数据，则可以用 **j** 或 **k** 从本章配套文件中读入数据。

```

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


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

f ticker_list = ['SP500']
g df = pdr.DataReader(ticker_list,
                      'fred',
                      start_date,
                      end_date)

# 双备份数据
h df.to_csv('SP500_' + str(start_date.date()) + '_' +
           + str(end_date.date()) + '.csv')
i df.to_pickle('SP500_' + str(start_date.date()) + '_' +
              + str(end_date.date()) + '.pkl')

# 从备份数据导入
j # df = pd.read_csv('SP500_2014-01-01_2022-12-31.csv',
#                   index_col=0, parse_dates=True)
k # df = pd.read_pickle('SP500_2014-01-01_2022-12-31.pkl')

```

代码 1. 下载金融数据;  Bk1\_Ch24\_01.ipynb

代码 2 中 **a** 利用 `plotly.express.line()` 绘制时间序列线图。

**b** 使用 `update_layout` 方法来自定义图形的布局和标题。

`axis_title='Date'` 设置了 X 轴的标题为 "Date"。

`axis_title='S&P 500 index'` 设置了 Y 轴的标题为 "S&P 500 index"。

`legend_title='Curve'` 设置了图例为 "Curve"。但是, `showlegend=False` 关闭图例。

**c** 计算缺失值比例。`isnull()` 方法用于检查 DataFrame 中的每个元素是否为缺失值 NaN。它返回一个与原始 DataFrame 相同形状的布尔值 DataFrame, 其中每个元素都是一个布尔值, 表示对应位置是否是缺失值。True (1) 代表缺失, False (0) 代表存在。方法 `sum()` 对每列求和, 得到每列中缺失值的总数。`len(df)` 则计算数据帧的总行数。

**d** 中 `% .3f` 表示要插入一个浮点数, 并保留三位小数。注意, `%%` 表示要插入一个百分号字符 `%`。注意, 因为 `%` 在格式化字符串中具有特殊含义; 所以如果要打印百分号字符本身, 需要用两个百分号 `%%` 来表示。



如果忘记怎么在字符串中插入数据, 请大家参考本书第 5 章。

**e** 创建了一个 Matplotlib 的 Figure 对象 `fig` 和一个 Axes 对象 `ax`，并设置了图形的宽、高为 (2, 4)，单位为英寸。

**f** 利用 `seaborn.heatmap()` 绘制热图展示缺失值，具体如图 1 (b) 所示。参数 `cbar=False` 表示不绘制颜色条 (colorbar)，即颜色映射和数值关系。

参数 `cmap='YlGnBu'` 指定了用于渲染热图的颜色映射为 'YlGnBu' 颜色映射，它表示黄色到蓝色的渐变，用于表示缺失值的程度。


参数 `yticklabels=[]` 用于指定 Y 轴上的刻度标签。通过将其设置为空列表 `[]`，我们不显示 Y 轴上的刻度标签。

```
# 含有缺失值的时间序列线图

a fig = px.line(df)
b 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
c percentag_missing = df.isnull().sum()*100/len(df)
print('Percentage of missing data')
d print("%.3f%%" % (percentag_missing))

# 可视化缺失值
e fig, ax = plt.subplots(figsize = (2,4))
# 使用seaborn.heatmap() 绘制热图
f sns.heatmap(df.isnull(), cbar = False,
              cmap = 'YlGnBu', yticklabels = [])
plt.show()
```

代码 2. 可视化缺失值，使用时配合前文代码；使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

## 24.2 缺失值：用 NaN 表示

**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。


 本书第 29 章将简述 Scikit-learn 中处理缺失值方法；此外，鸢尾花书《数据有道》将专门介绍处理缺失值的各种方法。

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 2 所示的趋势。



图 2. 标普 500 数据，删除缺失值


代码 3 中 **a** 利用 `dropna()` 方法删除包含缺失值 `NaN` 的行（默认）。如果希望删除包含缺失值的列，可以传递额外的参数 `axis=1`。

**b** 再次计算新数据帧的缺失值情况。

**c** 再次用 `plotly.express.line()` 绘制删除缺失值后的时间序列线图。

```
# 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()
```

代码 3. 删除缺失值，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

为了醒目地观察每年趋势，我们绘制了图 3。图 3 中每个子图代表一个年度的时间序列走势。

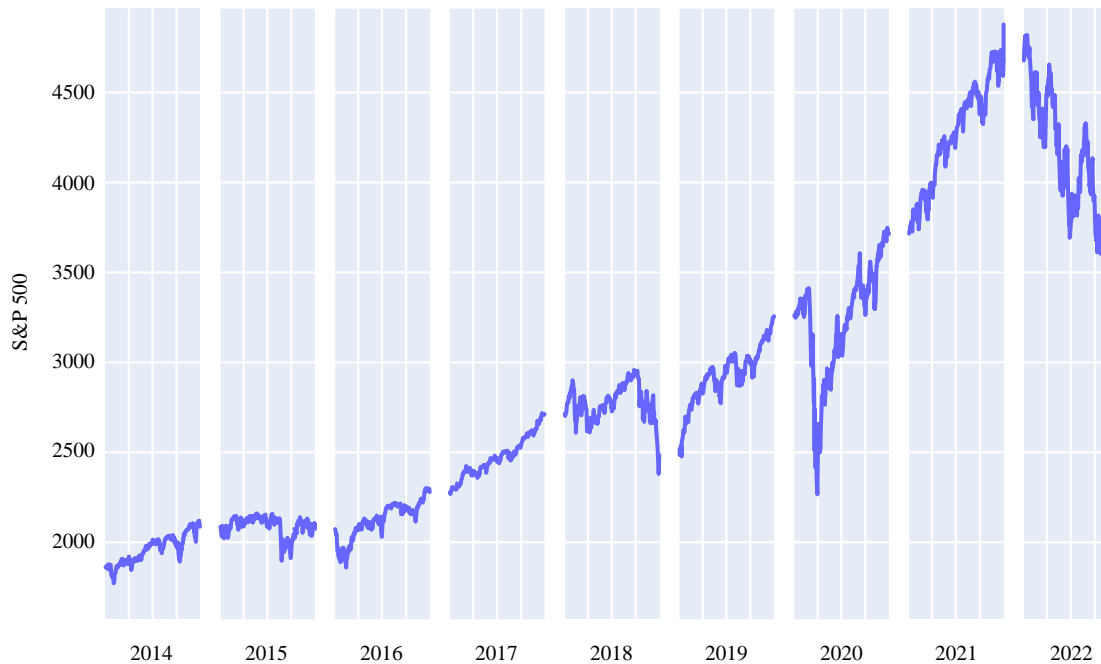


图 3. 标普 500 数据，按年观察趋势

代码 4 绘制图 3。a 在 `DataFrame` 中创建一个新的列 "Year"，该列包含了日期时间索引中的年份信息。

`pandas.DatetimeIndex()` 数据帧的时间索引转换为 `DatetimeIndex` 对象。

`DatetimeIndex` 是 Pandas 中用于处理日期时间索引的数据结构。

然后，利用 `.year` 这个 `DatetimeIndex` 对象的属性，用于从日期中提取年份信息。它会返回一个包含与索引中日期时间对应的年份的 Pandas Series。

b 也是用 `plotly.express.line()` 绘制线图，和本章前面不同的是参数 `facet_col='Year'` 启用了多列子图。

也就是说，我们根据 "Year" 列的不同值将数据分成不同的列绘制不同年份数据的子图。

`facet_row=None` 这个参数表示不启用按行分面绘制，即子图不会按行分割。

c 更新 Plotly 图表对象 `fig` 的布局参数。

`width=700` 设置图表的宽度为 700 像素。`height=500` 设置图表的高度为 500 像素。

参数 `margin=dict(l=20, r=20, t=30, b=20)` 设置图表的边距 (margin)。

`dict` 函数创建了一个包含左边距 (l)、右边距 (r)、顶部边距 (t)、底部边距 (b) 的字典。这些值表示图表内容与图表边界之间的像素间隔。例如，`l=20` 表示左边距为 20 像素。

`paper_bgcolor="white"` 设置图表的背景颜色为白色。



```

# 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")
fig.show()

```

代码 4. 按年绘制标普 500 水平子图，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

➔ 除了缺失值，样本数据中也难免会出现**离群值** (outlier)。本章不会介绍如何处理离群值，本书第 29 章将简述 Scikit-learn 中处理离群值的工具。



#### 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  $3\sigma$  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。

## 24.3 移动平均：一种平滑技术

时间序列的**移动平均** (moving average, MA)，也叫滚动平均，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**回望窗口** (lookback window)，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，但可能会导致移动平均数据出现明显滞后，跟踪效果变差。

具体步骤如下。

- ▶ 选择窗口的大小，例如 10 个数据点。
- ▶ 从序列的起始位置开始，计算窗口内数据点的平均值。
- ▶ 将该平均值作为移动平均的第一个数据点，记录下来。
- ▶ 移动窗口向后滑动一个数据点的位置。
- ▶ 重复步骤 2 至 4，计算新窗口内的平均值，并记录下来。
- ▶ 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。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

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。鸢尾花书《数据有道》将专门介绍加权移动平均。

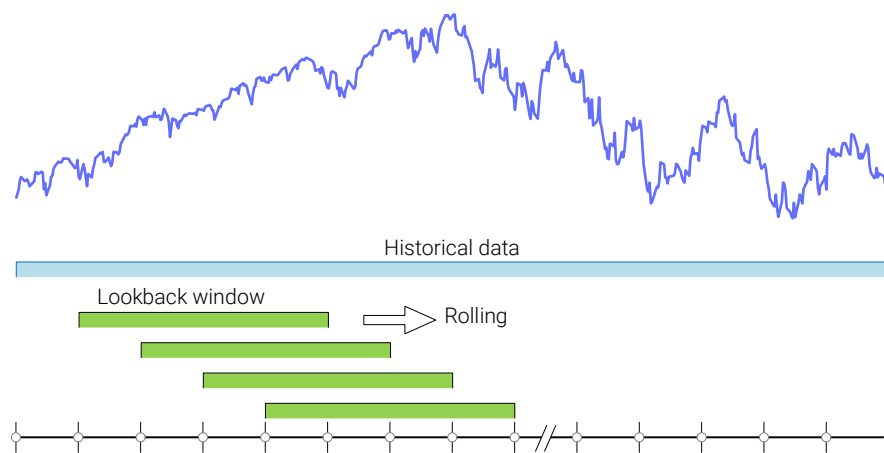


图 4. 移动窗口

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 5. 标普 500 数据，移动平均

代码 5 绘制图 5，下面让我们聊聊这段代码。

**a** 在原来的数据帧中创建一个新的列 "MA20"，该列包含了基于 "SP500" 列的移动均值。

方法 `.rolling(20)` 执行移动计算，例如移动平均。20 表示移动窗口长度，即在计算移动均值时要考虑的数据点的数量。

方法 `.mean()` 表示对移动窗口内的数据计算均值。

注意，在使用 `rolling` 方法时，默认 `center=False`。如图 6 (a) 所示，当设置 `center = False` 时，移动窗口的标签将被设置为窗口索引的右边缘；也就是说，窗口的标签与移动窗口的右边界对齐。这意味着移动窗口中的数据包括右边界，但不包括左边界。

如图 6 (b) 所示，当 `center=True` 时，移动窗口的标签将被设置为窗口索引的中心。也就是说，窗口的标签位于移动窗口的中间。这意味着移动窗口中的数据将包括左右两边的数据，并且标签位于窗口中央。

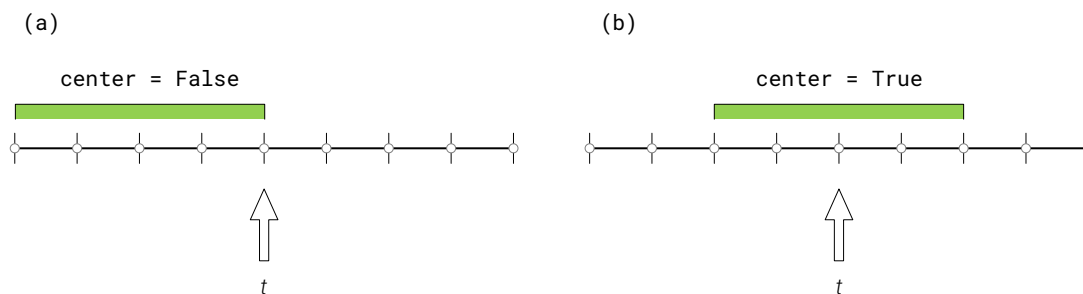



图 6. 移动窗口位置

同理，**b** 和 **c** 也计算移动平均，只不过窗口内历史数据不同。

**d** 只选取数据帧其中 4 列。

**e** 在利用 `plotly.express.line()` 绘制折线图时，利用 `.loc['20210101':'20221231']` 只绘制日期范围在 '20210101' 到 '20221231' 之间的数据。

```
# 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')
fig.show()
```

代码 5. 绘制标普 500 移动平均，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

## 24.4 收益率：相对涨跌

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。

本书前文提过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 7 所示为标普 500 的日收益率。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

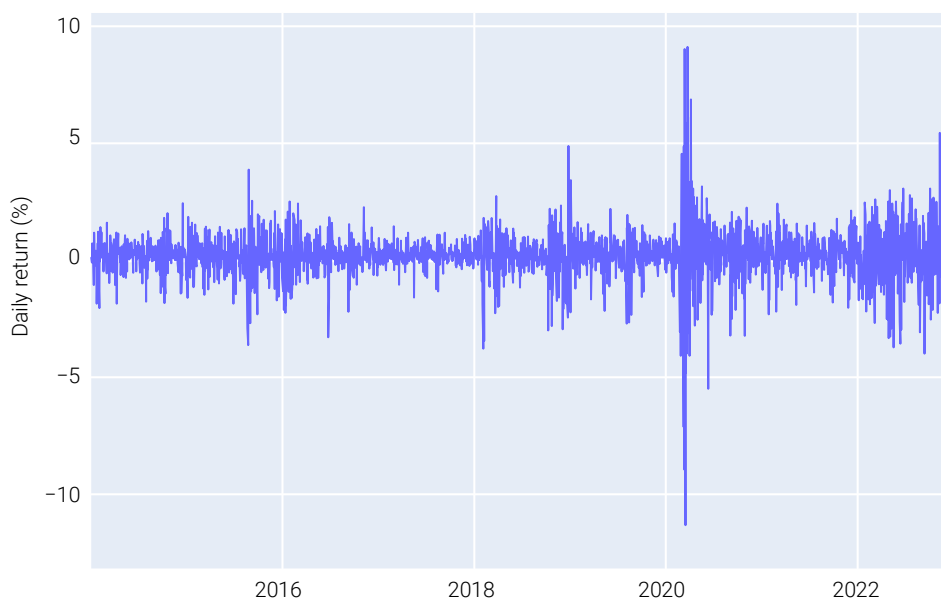


图 7. 标普 500 数据日收益率

代码 6 计算收益率并绘制图 7。


- a 中 `.pct_change()` 方法用于计算列中相邻两个值相对变化，即日收益率。
- b 还是用 `plot.express.line()` 绘制收益率百分比。
- c 修改 `fig` 对象的标题、横轴标题、纵轴标题、图例名称。

```

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()
```

代码 6. 绘制标普 500 日收益率，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

为了方便观察每年涨跌情况，我们绘制了图 8。

类似前文代码，代码 7 中 a 以绘制年份线图子图，b 调整 `fig` 对象设置。

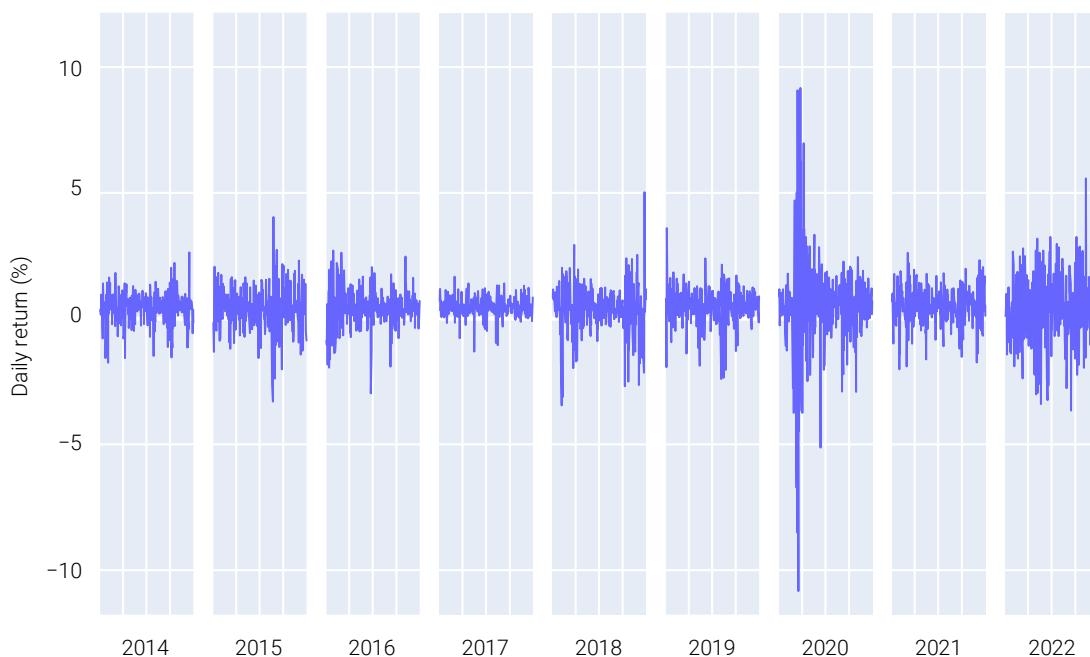


图 8. 标普 500 数据日收益率，按年观察趋势

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

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

代码 7. 按年绘制标普 500 收益率子图，使用时配合前文代码； Bk1\_Ch24\_01.ipynb

## 24.5 统计分析：均值、波动率等

市场涨跌越越剧烈，曲线波动越剧烈。图 8 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 9 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。

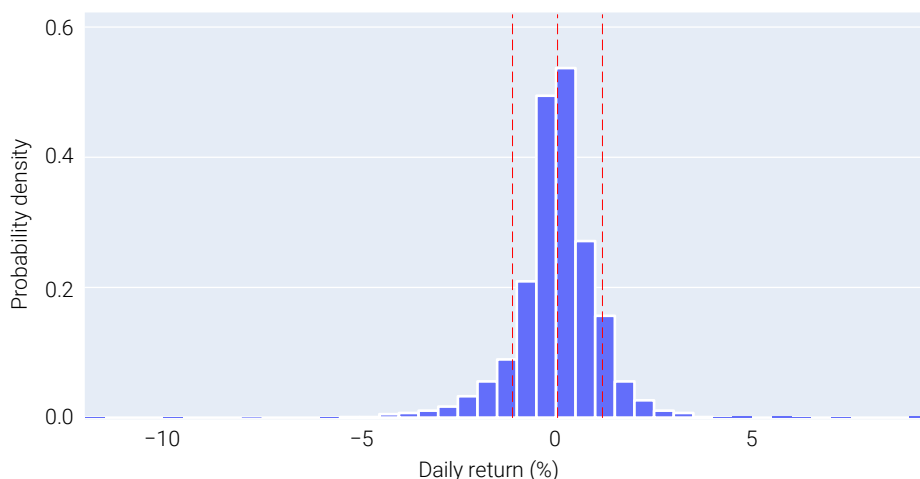


图 9. 所有下载历史数据日收益率分布

代码 8 中 **a** 用 `numpy.mean()` 计算均值。请大家修改代码，用 `pandas.DataFrame.mean()` 计算均值，并比较结果。

**b** 用 `numpy.std()` 计算均方差。

请大家修改代码，用 `pandas.DataFrame.std()` 计算均值，并比较结果解释为什么两个结果不一致。

**c** 使用 `plotly.express.histogram()` 绘制直方图。其中，`nbins=50` 指定了直方图的柱子 (bin) 的数量，用于显示不同收益率区间的频率。

参数 `histnorm='probability density'` 控制直方图的归一化方式。"probability density" 表示直方图的高度表示概率密度，这意味着每个柱子的高度表示对应收益率区间的概率密度而不是频数，这使得直方图的总面积等于 1。

**d** 这段代码用 `.add_shape()` 方法给 Plotly 图表中添加一条竖直虚线，以标识数据的平均值。

参数 `type='line'` 指定要添加的形状类型为一条线。

参数 `x0=mean` 设置线起点横坐标，即样本平均值。参数 `y0=0` 设置线起点纵坐标。

参数 `x1=mean` 设置线结束点横坐标，同样也是平均值。

参数 `y1=1` 设置线的结束点的纵坐标，表示线延伸到 Y 轴的 1 位置。

参数 `line=dict(color='red', dash='dash')` 设置线的样式。具体来说：`color='red'` 设置线的颜色为红色，`dash='dash'` 设置线的类型为虚线。

参数 `name='mean'` 给添加的形状一个名称，以便在图例中显示。

```

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

# 绘制直方图
c 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
d 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()

```


代码 8. 日收益率分布，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

图 10 所示为年度日收益率分布变化情况。代码 9 绘制图 10。

其中，**a** 也是绘制直方图，不同的是我们利用 `orientation='h'` 这个参数设置直方图的方向为水平方向，即柱子是水平放置的。如果设置 `orientation` 为 `'v'`，则柱子将垂直放置，这也是默认方向。

此外，`facet_col='Year'` 这个参数启用了子图绘制，根据不同年份将数据分成不同的列绘制子图。

**b** 也是修改图像默认设置。

**c** 则是利用 `.update_xaxes()` 方法通过将 `matches` 参数设置为 `'x'`，让所有子图横轴的属性都设置为相同的值，以使它们在图表中具有一致的外观和行为。

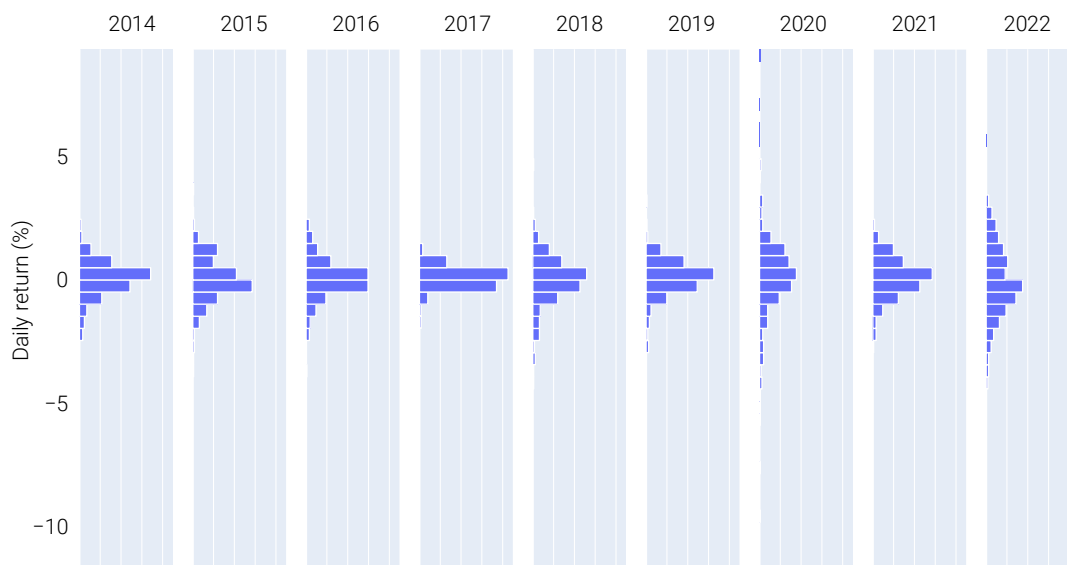


图 10. 日收益率分布, 按年

```

# 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")

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

```

代码 9. 按年日收益率直方图子图, 使用时配合前文代码; Bk1\_Ch24\_01.ipynb

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 11 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)。



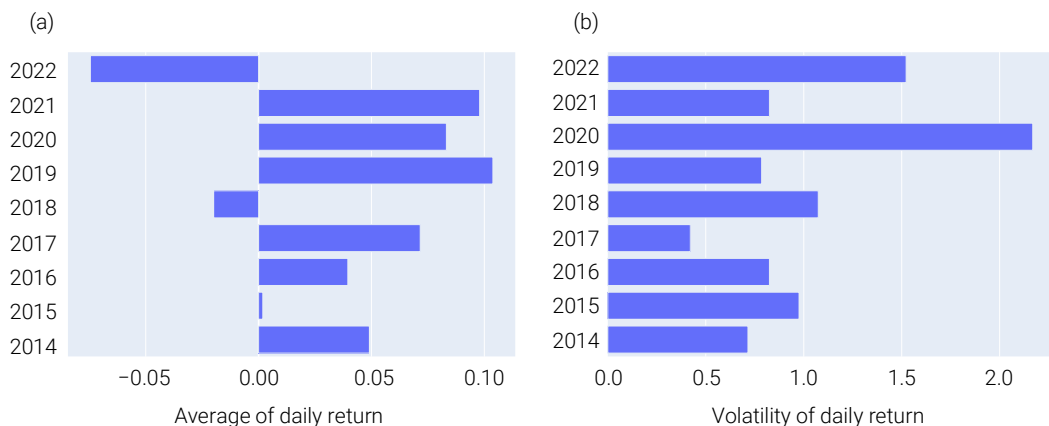


图 11. 水平柱状图可视化收益率均值、标准差（波动率），按年

代码 10 中 **a** 进行分组聚合。`groupby(['Year'], as_index=False)` 利用 `groupby()` 方法，它用于将数据按照指定的年度进行分组。

参数 `as_index=False` 指示不将 "Year" 列作为索引列，而保留其作为普通列。

方法 `.agg({'daily_r': ['mean', 'std']})` 指定对每个分组应用聚合函数的操作，'daily\_r' 给定列，列表 `['mean', 'std']` 指定聚合操作，包括**均值** (mean) 和**标准差** (std)。

**b** 利用 `plotly.express.bar()` 绘制水平柱状图。

参数 `y=Yearly_stats_df['Year']` 指定了纵轴上的数据，即年份数据。

参数 `x=Yearly_stats_df['daily_r']['mean']` 指定横轴上的数据，即每年平均收益率的数据。

参数 `orientation='h'` 指定了柱状图的方向为水平方向。

**c** 类似 **b**，不同的是横轴为收益率的波动率。

```

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

# 使用plotly.express绘制条形图
b 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
c 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()

```

代码 10. 日收益率年度统计数据柱状图，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

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 12 所示。

本章前文提过，Joypy 是一个第三方 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。

**⚠ 需要强调的是**，要使用 Joypy 绘制山脊图，需要首先安装 Joypy 库，并导入 joyplot 模块。安装 Joypy，请参考 <https://pypi.org/project/joypy/>。

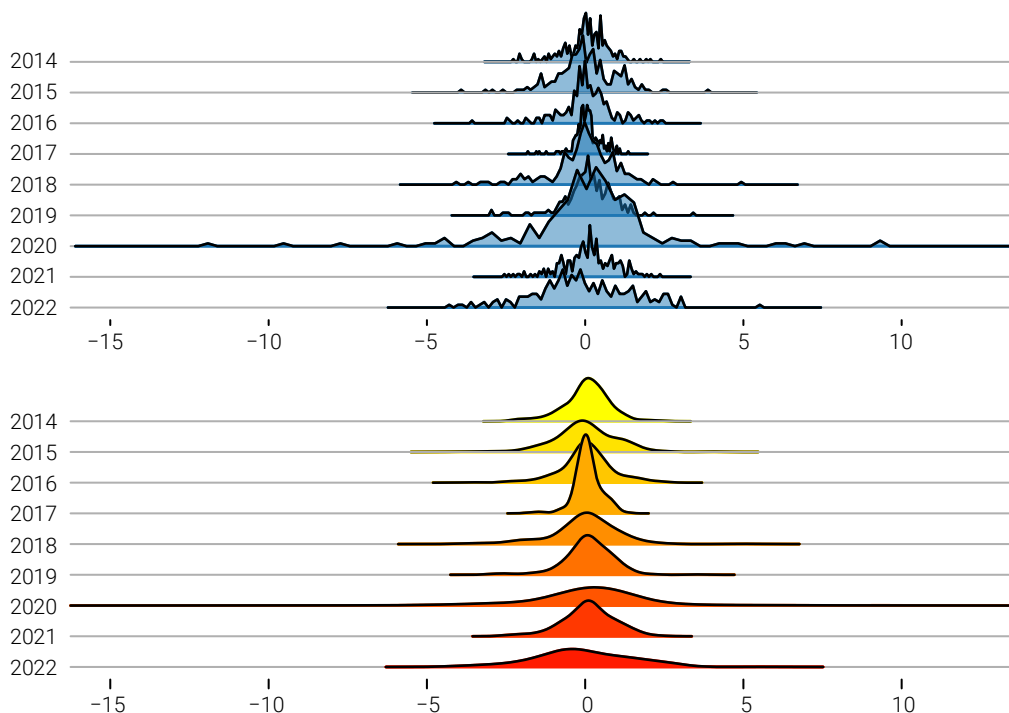


图 12. 山脊图, 按年

代码 11 绘制图 12 山脊图。

相信大家已经熟悉 [a](#) 这句，我们用它从日期中提取年份信息。

本章前文代码已经导入 `joypy`。[b](#) 中 `joypy.joyplot()` 绘制山脊图。

参数 `by="Year"` 指定了数据帧的一个列，用于将数据按照年份进行分组。也就是说，每个年份对应山脊图中一个分布。

参数 `ax=ax` 指定了用于绘制 `joyplot` 的坐标轴对象。变量 `ax` 对应上一句用 `plt.subplots(figsize = (6,4))` 生成的轴对象 `ax`。

`column="daily_r"` 这个参数指定显示每天收益率分布情况。

参数 `range_style='own'` 表示每个年份的分布图会根据数据的范围自动调整，以便更好地显示数据。

参数 `grid="y"` 表示只显示垂直网格线。

参数 `linewidth=1` 指定了分布线条的宽度，用于绘制分布图。

参数 `legend=False` 这个参数表示不显示图例。

参数 `fade=True` 给山脊图添加渐变效果。

参数 `kind="counts"` 指定了分布图的类型为计数。

参数 `bins=100` 指定了数据分布图中的直方图区间数。

[c](#) 类似 [b](#)，有几点不同需要指出。这个山脊图采用更为平滑的**核密度估计** (Kernel Density Estimation, KDE)。此外，`colormap=cm.autumn_r` 指定了用于渲染山脊图的颜色映射为

"autumn\_r" 颜色映射，该映射主题为“秋色”，红黄渐变。注意，\_r 代表翻转颜色映射；请大家修改代码尝试"autumn" 颜色映射，并对比效果。

➡ 本书第 27 章将简单介绍核密度估计 KDE 方法；鸢尾花书《统计至简》会专门深入介绍 KDE 的数学原理。

```

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()

```

代码 11. 绘制年度数据山脊图，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

## 季度分布

当然，我们也可以按季度分析收益率。图 13 所示为每一年四个季度收益率的均值、标准差的柱状图。

? 请思考如何利用上一章介绍的方法，分别设定两种钻取方法“年份 → 季度”和“季度 → 年份”，然后用柱状图可视化结果。

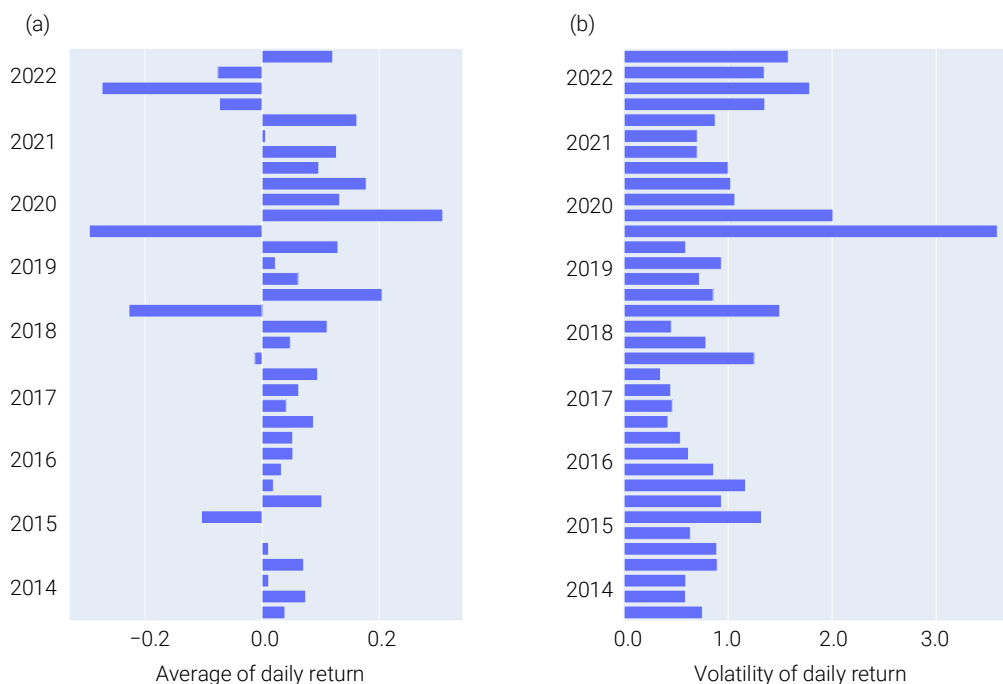


图 13. 水平柱状图可视化收益率均值、标准差（波动率），按季度；注意，图中省去季度标识

代码 12 计算绘制图 13。下面讲解其中关键语句。

- a** 用 `.index.quarter` 提取季度信息。
- b** 用 `.index.year` 提取年份信息。
- c** 用列表生成式创建一个名为 `quarter_yr` 的新列表。它将两个之前生成的 `years` 和 `quarters` 组合起来，创建一个包含字符串的列表 `quarter_yr`，每个字符串表示一个年份和季度的组合。

在列表生成式中，`f"{year}, Q{quarter}"` 表达式使用 `f-string` 迭代创建一组新的字符串。它将 `year` 和 `quarter` 变量的值插入到字符串中，其中 `year` 表示年份，`quarter` 表示季度，并在季度前面加上 "Q"。这样，它就形成了一个字符串，例如 "2018, Q2"，其中 2018 是年份，Q2 是季度。



如果大家忘了列表生成式和 `f-string`，请回顾本书第 7 章。

- d** 用列表在数据帧中创建新的一列。
- 本章前文用过类似 **e** 语句。方法 `.groupby(['quarter_yr'], as_index=False)` 是一个分组操作，它将数据帧 `df_` 按照 `'quarter_yr'` 列的独特值进行分组。  
`as_index=False` 参数表示不将 `'quarter_yr'` 列设置为索引，而保留它作为列。  
 方法 `.agg({'daily_r': ['mean', 'std']})` 是一个聚合操作，它对每个分组进行统计计算。  
 类似前文，**f** 和 **g** 用 `plotly.express.bar()` 绘制水平方向柱状图。

```

# 季度均值、标准差
a quarters = df_.index.quarter
b years = df_.index.year

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

# 添加新列
d df_['quarter_yr'] = quarter_yr

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

# 使用plotly.express绘制条形图
f fig = px.bar(y=Qly_stats_df['quarter_yr'],
               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')

fig.show()

# 使用plotly.express绘制条形图
g fig = px.bar(y=Qly_stats_df['quarter_yr'],
               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')

fig.show()

```

代码 12. 绘制季度统计量柱状图，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

图 14 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。请大家自行分析代码 13 语句。

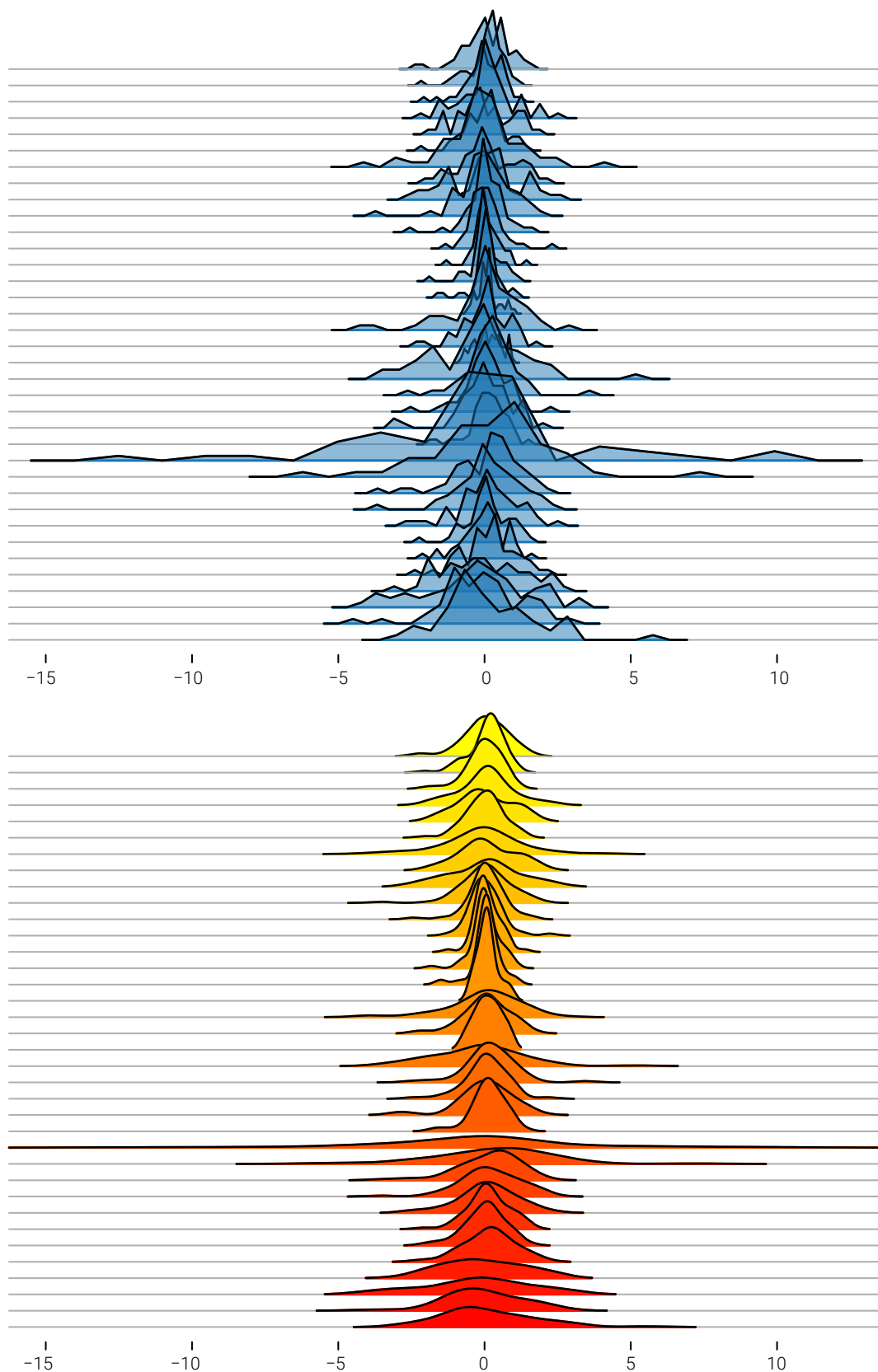


图 14. 收益率山脊图，按季度

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

```

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


a joypy.joyplot(df_, by="quarter_yr", 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))
b joypy.joyplot(df_, by="quarter_yr", column="daily_r", ax = ax,
                 range_style='own', grid="y",
                 linewidth=1, legend=False,
                 colormap=cm.autumn_r, fade=True)

plt.show()

```

代码 13. 绘制季度数据山脊图，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

## 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。

类似移动平均值，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 15 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。



鸢尾花书《数据有道》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。



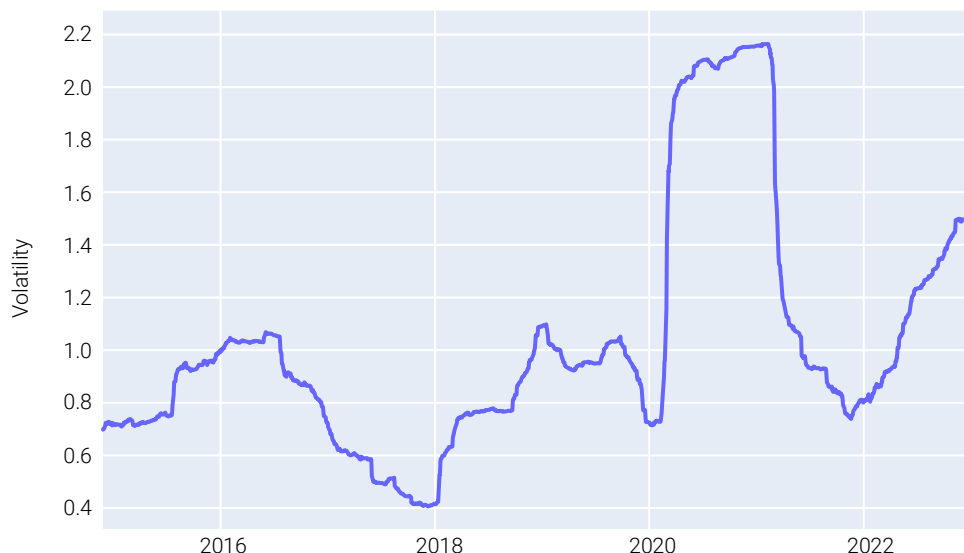


图 15. 移动波动率

代码 14 中 <sup>a</sup> 对数据帧 `df_` 中的 `'daily_r'` 列计算移动波动率（标准差）。

方法 `.rolling(250)` 设置移动窗口长度。这意味着在进行标准差计算时，每次都会考虑 250 个历史数据点。

方法 `.std()` 表示在移动窗口对象上计算标准差，即波动率。

<sup>b</sup> 也适用 `plotly.express.line()` 绘制移动波动率曲线。

请大家使用其他移动窗口长度计算不同的移动波动率，并绘制曲线比较。



请大家思考，移动窗口长度对移动波动率有怎样影响。

```
# 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()
```

代码 14. 绘制移动波动率，使用时配合前文代码； Bk1\_Ch24\_01.ipynb

## 24.6 相关性：也可以随时间变化

几个不同时间序列之间肯定也会存在相关性。图 16 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。注意，这个矩阵是给定历史时间序列数据的“静态”相关性系数。

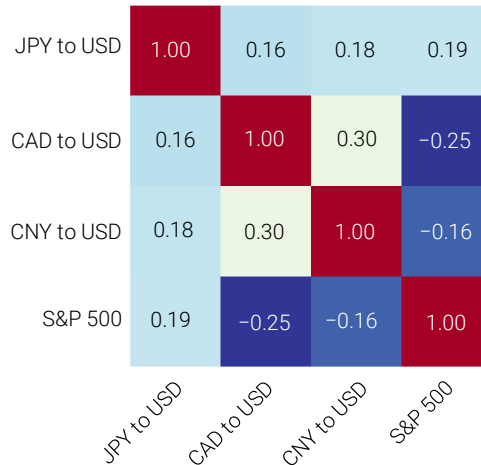


图 16. 相关性系数矩阵

代码 15 绘制图 16。下面聊聊其中关键语句。

- a** 用列表列出数据的标识符，其中前三个为汇率。**b** 从 FRED 下载数据。
- c** 修改数据帧列标签。
- d** 先删除 NaN，再计算收益率，即相对涨跌。
- e** 用 `plotly.express.imshow()` 绘制相关性矩阵热图。方法 `corr()` 计算数据帧各列之间的相关性矩阵。`text_auto='.2f'` 控制在热图上显示的文本标签的格式。`.2f` 表示将浮点数格式化为包含两位小数。`color_continuous_scale='RdYlBu_r'` 指定了用于着色的颜色映射。

```
# 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()
```


本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

代码 15. 可视化相关性系数矩阵, 使用时配合前文代码;  Bk1\_Ch24\_01.ipynb

相关性并不是一成不变的, 也是随时间不断变化。如图 17 所示, 当我们指定具体的移动窗口长度, 在不同的时间点上都可以计算得到相关性系数。因此, 我们也可以得到移动相关性时间序列, 这组数据就变成了“动态”数据。

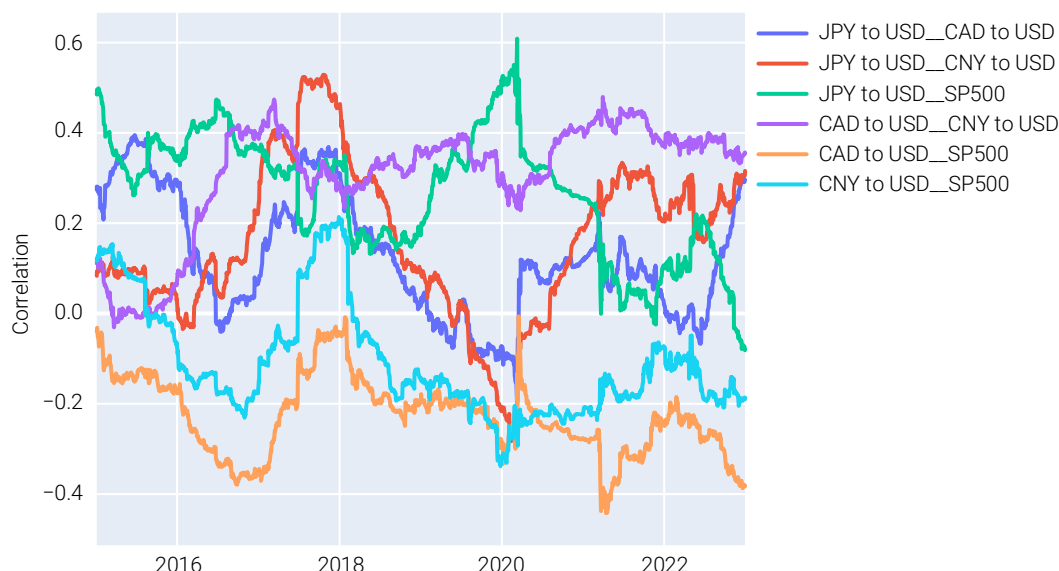


图 17. 移动相关性

代码 16 绘制图 17。

**a** 也是用 `.rolling()` 方法计算移动相关系数矩阵, 并删除缺失值行。

注意, 计算的结果是多级行标签数据帧。如表 1 所示, 每个日期都对应一个相关性系数矩阵。绘制图 17 时, 我们仅仅需要获取其中 6 个成对相关系数时间序列。

**b** 将多级行索引数据帧转换为一般数据帧, 默认将多级行索引中低级索引转化为列, 结果如表 2 所示。

**c** 这句话中, `df_rolling_corr.unstack().columns.values` 的结果为数组:

```
array([('JPY to USD', 'CAD to USD'), ('JPY to USD', 'CNY to USD'),
      ('JPY to USD', 'JPY to USD'), ('JPY to USD', 'SP500'),
      ('CAD to USD', 'CAD to USD'), ('CAD to USD', 'CNY to USD'),
      ('CAD to USD', 'JPY to USD'), ('CAD to USD', 'SP500'),
      ('CNY to USD', 'CAD to USD'), ('CNY to USD', 'CNY to USD'),
      ('CNY to USD', 'JPY to USD'), ('CNY to USD', 'SP500'),
      ('SP500', 'CAD to USD'), ('SP500', 'CNY to USD'),
      ('SP500', 'JPY to USD'), ('SP500', 'SP500')], dtype=object)
```

而数组中的每个元素为元组, 这个元组中的两个字符串就是计算相关性系数的成对标识符。

然后, 在列表生成式迭代时, `'_'.join(col)` 这一句用下划线 `_` 将元组中的两个字符串将列名 `col` 中的元素用空格连接起来。因为 `col` 是一个列表, 它可能包含多个元素, 这些元素会以空格分隔。然后列表将会替代数据帧现有列标签。

**d** 删除数据帧中存在 NaN 的行。

本 PDF 文件为作者草稿, 发布目的为方便读者在移动终端学习, 终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有, 请勿商用, 引用请注明出处。

代码及 PDF 文件下载: <https://github.com/Visualize-ML>


本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>



欢迎大家批评指教, 本书专属邮箱: [jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

 导入 Python 标准库中的 `itertools` 模块中的 `combinations` 函数。





本书第 7 章介绍过如何使用 `itertools.combinations()`，请大家回顾。

 用 `list()` 将 `DataFrame` 列标签转换为一个 Python 列表。

 类似 ，用列表生成式和给定的列标签列表 `list_tickers` 中的所有可能的两两组合（不考虑顺序）连接成一个长度为 6 元素字符串列表 `pairs_kept`。这个字符串列表代表要保留的相关性系数曲线。

也就是说，图 16 所示的相关性系数矩阵中，其实我们只关心其中 6 ( $C_4^2$ ) 个值。这 6 个值可以是（不含对角线）的下三角元素，或者（不含对角线）的上三角元素。

 选定数据帧中要保留的 6 列。

 用 `plotly.express.line()` 绘制 6 条移动相关性系数。



请大家思考如果在计算移动相关性系数时，采用的回望窗口宽度不是 250，而是 125 或 500，会对结果产生怎样影响？

表 1. 滚动相关性系数矩阵，多级行标签数据帧

DATE		JPY to USD	CAD to USD	CNY to USD	SP500
2015-01-05	JPY to USD	1.000	0.276	0.076	0.482
	CAD to USD	0.276	1.000	0.106	-0.046
	CNY to USD	0.076	0.106	1.000	0.110
	SP500	0.482	-0.046	0.110	1.000
2015-01-06	JPY to USD	1.000	0.269	0.085	0.488
	CAD to USD	0.269	1.000	0.104	-0.048
	CNY to USD	0.085	0.104	1.000	0.115
	SP500	0.488	-0.048	0.115	1.000
...	...	...	...	...	...
2022-12-30	JPY to USD	1.000	0.291	0.311	-0.089
	CAD to USD	0.291	1.000	0.350	-0.395
	CNY to USD	0.311	0.350	1.000	-0.196
	SP500	-0.089	-0.395	-0.196	1.000

表 2. 滚动相关性系数矩阵，宽格式

	JPY to USD				CAD to USD				CNY to USD				SP500			
	CAD to USD	CNY to USD	JPY to USD	SP500	CAD to USD	CNY to USD	JPY to USD	SP500	CAD to USD	CNY to USD	JPY to USD	SP500	CAD to USD	CNY to USD	JPY to USD	SP500
DATE																
1/5/2015	0.276	0.076	1.000	0.482	1.000	0.106	0.276	0.046	0.106	1.000	0.076	0.110	-0.046	0.110	0.482	1.000
1/6/2015	0.269	0.085	1.000	0.488	1.000	0.104	0.269	0.048	0.104	1.000	0.085	0.115	-0.048	0.115	0.488	1.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12/30/2022	0.291	0.311	1.000	-0.089	1.000	0.350	0.291	0.395	0.350	1.000	0.311	0.196	-0.395	-0.196	-0.089	1.000

```

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

# 整理数据
b df_roll_corr_ = df_roll_corr.unstack()

c df_roll_corr_.columns = ['_'.join(col)
                           for col in
                           df_roll_corr.unstack().columns.values]
d df_roll_corr_ = df_roll_corr_.dropna()

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


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


# 可视化
i fig = px.line(df_roll_corr_)

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

fig.show()

```

代码 16. 可视化移动相关性系数，使用时配合前文代码；  Bk1\_Ch24\_01.ipynb

 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](https://pandas.pydata.org/docs/user_guide/timeseries.html)

此外，Statsmodels 有大量时间序列分析工具：

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

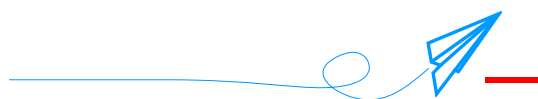
版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

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



这是专门介绍 Pandas 库的最后一章，我们特别介绍了时间序列数据帧基本分析，以及用 Plotly 完成可视化。

总的来说，Pandas 特别适合数据处理和分析。Seaborn 和 Plotly 这两库和 Pandas 数据帧结合的特别紧密。

本书专门讲解 Pandas 的内容到此为止。Pandas 的用法很灵活，希望大家一边实践应用、一边不断探索学习。下一板块将介绍三个常用 Python 数学库——SymPy、SciPy、Statsmodels。