algotrading

Release 0.0.1

May 25, 2021

Getting started


Learn & Have fun with
Algorithmic Trading
THE ALL-IN-ONE LEARNING REPO

CHAPTER 1

How to use this

1.1 What this is

Welcome to the **Algo Trading Learning Repo**! This is an educational resource produced by undergraduate students at the University of Hong Kong (HKU) to make it easier to learn about algorithmic trading.

The code repository could be accessed here. It contains a variety of helpful resources, including:

- a short introduction to algo trading terminology, basic theory and data science
- explanation and equation of common technical indicators
- code implementation of technical & fundamental analysis strategies in Python
- code examples of using machine learning for stock screening and trend analysis

We recommend beginners to start with the *Introduction*, and for coders who have a basic understanding of technical analysis & fundamental analysis could jump right into *Technical analysis*.

1.2 Why we built this

Algorithmic trading is a multidisciplinary field that involves knowledge in three domains, which are finance, programming and statistics. While one could easily look for online learning resources on the Internet nowadays, they still present various challenges to a beginner who would like to learn the field from scratch:

- There are lots of blog posts and online materials about algo trading, but they are scattered; and are usually just focusing on a few concepts or topics.

- There exist open-source repositories that feature the implementation of a wide selection of algo trading strategies, but they do not come along with explanations and assume that the user knows the financial logic behind.
- On the other hand, there exist good resources that provide detailed discussions about financial concepts and rationale of trading strategies Investopedia, but they are not accompanied by any code examples.
- Majority of the resources do not fit the local context (e.g. about trade execution in Hong Kong) and are merely focusing on the US market.

Personally, we also find this process inefficient and daunting. Being a group of students who are majoring in Computer Science and Finance at the university right now, we are eager to dedicate our efforts into creating learning resources that could align with the financial setting of Hong Kong, and could pique the interest of beginners.

1.3 Report bugs & Support history

If you find any bugs or mistakes, feel free to report and we'll repair them to ensure that the repository is working properly.

CHAPTER 2

Installation

The Algo Trading Learning Repo requires Python3 and other libraries which would be specified in the modules.

The repository is currently only supported on Linux and OSX. It may be possible to install on Windows, but this has not been extensively tested.

2.1 Installing Python

We recommend installing Python through Anaconda. The installation instructions for Anaconda could be found here.

CHAPTER 3

Introduction

Welcome to the first tutorial for algorithmic trading!

In this tutorial, you will learn:

- The basics of algorithmic trading
- Definition of technical analysis
- Definition of fundamental analysis
- Difference between technical analysis and fundamental analysis
- How machine learning could assist algo trading

3.1 What is algo trading?

Definition

Algorithmic trading (a.k.a. algo trading) is the use of computer codes and chart analysis to enter and exit trades automatically according to set parameters such as price movements or volatility levels.

In algo trading, one writes an algorithm and instruct a computer to buy or sell stocks for you when the defined conditions are met. These programs can trade at speed and frequency that is much higher than that of a human trader, and the process can be semi-automated or completely automated. This is why sometimes we could use the term **"automated trading"** interchangelably, but note that they are not necesssarily referring to the same thing.

A typical algorithmic trading system does the following things (in order):

- Inspect data, charts, quotes or news and generate trade signals as per your strategy
- Fill in order details when a trade signal is found
- Monitor and evaluate trades to see if they reached your target or went in the opposite direction
- Close positions to either book profits or cut losses
- Rinse and repeat

3.1.1 Automated Trading

Automated Trading is the absolute automation of the trading process. Trading orders are automatically created, submitted to the market and executed. Automated trading facilities are usually utilized by hedge funds that exploit proprietary execution algorithms and trade via Direct-Market Access (DMA) or sponsored access.

3.1.2 High-freqeuncy Trading

High-frequency Trading (HFT) is a subset of automated trading. Technology has enabled orders to be made within milliseconds. As execution speed is of a priority here, HFT makes use of Direct-Market Access (DMA) in order to reduce the time transactions. You can read more about HFT here.

Tip:

- **Algorithmic Trading**: execution process based on an algorithm
- **Automated Trading**: as its name implies, automate the trading process
- **High-frequency Trading**: ultra-fast automated trading

There are two major schools of methods for analysing the market, which are (1) technical analysis and (2) fundamental analysis. We'll go through both of these in detail respectively in the coming tutorials.

3.2 What is technical analysis?

Definition

Technical analysis is the study of market action, primarily through the use of charts, for the purpose of forecasting future price trends. - *John J. Murphy*

Here, the term *"market action"* (or price data, as some people call it) refers to to any combination of the open, high, low, close, volume, or open interest for a given security over a specific timeframe. The timeframe can be based on intraday (1-minute, 5-minute, 10-minute, 15-minute, 30-minute or hourly), daily, weekly or monthly price data and last a few hours or many years.

3.2.1 Rationale of technical analysis

1. Market Action Discounts Everything

This assumption states that any factor that could possibly affect the stock's price - fundamental, political, psychological, or otherwise - is fully reflected in the price of that stock. In other words, it is presumed that prices should reflect shifts in demand and supply. For example, if prices are rising, no matter the reasons behind, demand must exceed supply and thus the fundamentals must be bullish. Conversely, if prices fall, the fundamentals must be bearish.

Hence as a rule, technicians do not concern themselves with the reasons why prices rise or fall. If everything that affects the market prices is ultimately reflected in market price, then it makes sense that they study of that market price is all that is necessary. That's why this assumption serves as a cornerstone for technical analysis to work.

2. Prices Move in Trends

Unless one accepts that markets do in fact trend, there's no point in reading any further. Given the assumption that prices move in trends, we obtain the corollary that a trend in motion is likely to continue than to reverse. Just like Newton's first law of motion. The entire trend-following approach is predicated on riding an existing trend, until a sign of reversing is observed.

3. History repeats itself

Under this assumption, it is believed that the key to understanding the future lies in a study of the past. Charts tend to form shapes that have occurred in the past, and thus the analysis of historical patterns helps predict future market movements.

3.2.2 Pros and cons of technical analysis

Pros:

- Objectiveness
- Mathematical precision
- Emotional indifference
- Inexpensive

Cons:

- Self-fulfilling prophecy
- The future keeps running away
- Conflicting signals from different indicators
- Substantial movements might have taken place when a pattern is identified

3.3 What is fundamental analysis?

While technical analysis concentrates on the study of market action, fundamental analysis focuses on the economic forces of supply and demand that leads to price movements.

Definition

The **fundamental analysis** approach examines all of the relevant factors affecting the price of a stock in order to determine the intrinsic value of that stock. - *John J. Murphy*

The term *"intrinsic value"* here refers to what the stock is actually worth based on the law of supply and demand. If the intrinsic value is below the current market price, it means that the stock is overpriced and thus it should be sold. Conversely, if the intrinsic value is above the price, then the market is undervalued and that stock should be bought.

3.3.1 Rationale of fundamental analysis

One of the primary assumptions of fundamental analysis is that the current price from the stock market often does not fully reflect the value of a company. Thus, a fundamentalist would look at a company's publicly available information and other economic data in order to gain an understanding of the company's ability to create products and services, and to evaluate whether it is generating earnings in aproductive way.

This approach stems from the assumption that the reported financial information is legitimate and correct. A fundamentalist may also assume that a company's past performance and metrics may continue into the future.

Fundamental analysis consists of three main parts:

- **Economic analysis** - focuses on analysing various macroeconomic factors such as interest rates, inflation, and GDP levels
- **Industry analysis** - focuses on assessing specific prospects and potential opportunities within the identified industries and sectors
- **Company analysis** - focuses on analysing and selecting individual stocks within the most promising industries

We could conduct fundamental analysis either in a **top-down approach** or **bottom-up approach**. By following the former, the investor will start with analysing the health of the overall economy, and then to determine the industry trends, and thus filter out promising companies within the industry. With regards to latter, it will be the reverse - beginning with individual stock analysis first, to find out stocks that could outperform the industry.

3.3.2 Pros and cons of fundamental analysis

Pros:

- Seeks to understand the value of an asset
- Long-term view
- Comprehensive

Cons:

- Time-consuming
- Results not suitable for quick decisions
- Does not provide info about entry points*

With regards to the last point, some investors would try to complement fundamental analysis by making use of technical analysis to decide entry points in the identified stocks.

3.4 Technical analysis vs Fundamental analysis

Both of these approaches intend to solve the same problem - to determine the direction that prices are liekly to move. While technical analysis focuses more on answering the question of "when to buy", fundamental analysis helps us find an answer to the question of "what to buy".

Important:

The fundamentalist studies the cause of market movement, while the technician studies the effect. - *John J. Murphy*

Theoretically, the technician would, according to the assumptions, ignore the reasons that cause prices to change, and the fundamentalist would constantly be digging into the causes of price movements. However, in reality there is a lot of overlap between these two approaches, and they are not mutually exclusive. **The problem of using a combination of both, is that the technical indicators and fundamentals might come in conflict with each other**, and such descrepancies especially occur at the most critical moments.

Some people believe the fact that "market price tends to lead the known fundamentals" accounts for this phenomenon. Put it in another way, it implies that the market price acts as a leading indicator of the fundamentals. While the known fundamentals are already reflected "in the market", they are now reacting to the unknown fundamentals, and thus inducing the discrepancy.

In learning about the premises of technical analysis, one can see why technicians usually see their approach superior to fundamentalists'. Because, by definition, if the fundamentals are reflected in the market price, then technical approach includes the fundamental. Nevertheless, fundamental analysis does not include a study of market action. Therefore, whilee it is feasible to trade solely relying on technial approach, it is doubtful that anyone could trade off the fundamentals alone without any consideration of the technical side of the market.

3.5 The use of machine learning

(To-be edited)

3.5.1 Macroeconomic data

The stock prices are often moved by some macroeconomic indicators, and hence it is good to keep track of macroeconomic indicators.

Macroeconomic analysis is the study of relation between stock prices and macroeconomic indicators.

(To-be edited)

3.5.2 Sentiment analysis

Market sentiment refers to the overall attitude of investors toward a particular security. Investors profit by finding stocks that are overvalued based on market sentiment.

(To-be edited)

3.6 Conclusion

Algorithmic Trading has become increasingly popular in the recent decade, and it now accounts for the majority of trades in the market globally and has attributed to the success of some of the world's best-performing hedge funds. Indeed, 84% of trades that happened in New York Stock Exchange (NYSE), and 60% in London Stock Exchange (LSE) were all done using algorithmic trading. Therefore, whether one is interested in making money with algo trading or not, studying algo trading would definitely bring you insights on how technolgy has been applied in stock markets and learn how algorithms have been shaping our modern day world.

References

- Murphy, J. J. (1991). Technical analysis of the futures markets: A comprehensive guide to trading methods and applications. New York: New York Institute of Finance.
- CFI - Technical Analysis: A Beginner's Guide
- IG - Technical Analysis definition
- FBS - Pros and Cons of Technical Analysis
- CFI - What is Fundamental Analysis?

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 4

Data science basics

In this tutorial, you will learn to:

- Conduct Exploratory Data Analysis (EDA)
- Carry out resampling
- Visualise time series data
- Calculate and plot distribution of percentage change
- Use moving windows

Requirements:

- pandas
- matplotlib
- numpy

4.1 Exploratory Data Analysis

Definition

Exploratory Data Analysis (a.k.a. EDA) is the process of performing initial investigations on the data so as to discover patterns, spot anomalies and to check assumptions with the help of summary statistics and graphical representations.

For example, in the process of EDA, aim to find the answer to these questions:

- What kind of data does the dataframe store?

- What is the range of each column?
- What is the data type of each column?
- Are there null values in the data?

```
# import AAPL csv file
aapl = pd.read_csv('../../database/nasdaq_ticks_day/nasdaq_AAPL.csv', header=0, index_
↪col='Date', parse_dates=True)

# inspect first 5 rows (default)
aapl.head()

# inspect first 3 rows
aapl.head(3)
```

```
# check name of columns
aapl.columns

# output
# aapl.columns
```

```
# generate descriptive statistics, e.g. central tendency and dispersion of the dataset
# (excl. NaN values)
aapl.describe()

# prints a summary of the dataframe e.g. dtype, non-null values
aapl.info()
```

```
# print rows between two specific dates
print(aapl.loc[pd.Timestamp('2020-07-01'):pd.Timestamp('2020-07-17')])

# select only data between 2006 and 2019
aapl = aapl.loc[pd.Timestamp('2006-01-01'):pd.Timestamp('2019-12-31')]
aapl.head()
```

4.2 Resampling

It is easy to mix up **sampling** and **resampling**, which are indeed referring to two different concepts. We will first take a look at the definition of the former:

Definition

Sampling is the process of gathering observations with the intent to estimate a population variable.

Assume that each row of data represents an observation about something in the world. When working with data, we usually do not have access to all possible observations since they might be hard to gather, or it might be to costly to process them altogether. Thus, we use sampling as a solution - to select some part of the population to observe, so that we can infer something about the whole population.

For example, we can conduct **Simple Random Sampling**, which means that each row (or observation) is drawn with a uniform probability from the dataset.

```
# Take 10 rows from the dataframe randomly
sample = aapl.sample(10)
print(sample)
```

Now that we understand what sampling is, let's go back to resampling.

Definition

Resampling is a method that uses a data sample to improve the accuracy and quantify the uncertainty of a population parameter.

As a data sample might not accurately represent the popultion, it introduces the problems of **Selection Bias** and **Sampling Error**.

- **Selection Bias** occurs when the method of drawing observations skews the sample in some way.
- **Sampling Error** occurs when randomly drawing observations skews the sample in some way.

To address this problem, we want to know how accruately the data sample is estimating the population parameter (e.g. the mean, or the standard deviation).

If we only sample that data once, we will only have one single estimate of the population parameter, which makes it impossible to quantify the uncertainty of the estimate. (Assuming we do not have the population data) Therefore, we could try estimate the population parameter *multiple times* from our data sample - we call this action, **resampling**.

With regards to the *resample* function in pandas, it is used for changing the time interval of a dataset. Thus, we need a datetime type index or column in order to use the function.

```
# Resample to monthly level
monthly_aapl = aapl.resample('M').mean()
print(monthly_aapl)
```

As shown in the code above, there are two steps in calling the function:

1. Pass the **'Rule'** argument to the function, which determines by what interval the data will be resampled by. In the example above, 'M' means by month end frequency.
2. Decide how to **reduce the old datapoints** or fill in the new ones, by calling groupby aggregate functions including mean(), min(), max(), sum().

In the above example, as we are resampling that data to a wider time frame (from days to months), we are actually **"downsampling"** the data.

On the other hand, if we resample the data to a shorter time frame (from days to minutes), it will be called **"upsampling"**:

```
# Resample to minutely level
minutely_aapl = aapl.resample('T').ffill()
print(minutely_aapl)
```

As we end up having additional empty rows in the resulting table, we need to decide how to fill in them with numeric values:

- `ffill()` 'Forward filling' or `pad()` 'padding' — Use the last known value.

- `bfill()` or `backfill()` 'Backfilling' — Use the next known value.

4.3 Calculate percentage change

We can just directly use the `pct_change()` function to do this.

```
daily_close = aapl[['Close']]

# Calculate daily returns
daily_pct_change = daily_close.pct_change()

# Replace NA values with 0
daily_pct_change.fillna(0, inplace=True)

# Inspect daily returns
print(daily_pct_change.head())
```

```
# Calculate daily log returns
daily_log_returns = np.log(daily_close.pct_change()+1)

# Print daily log returns
print(daily_log_returns.head())
```

We can also combine with the operation of resampling to get the percentage change of different time intervals.

```
# Resample to business months, take last observation as value
monthly = aapl.resample('BM').apply(lambda x: x[-1])

# Calculate monthly percentage change
monthly.pct_change().tail()
```

This example takes the mean instead of the last observation in each bin as the value.

```
# Resample to quarters, take the mean as value per quarter
quarter = aapl.resample("4M").mean()

# Calculate quarterly percentage change
quarter.pct_change().tail()
```

It is also good to learn how to manually do the calculation, without using the `pct_change()` function.

```
# Daily returns
daily_pct_change = daily_close / daily_close.shift(1) - 1

# Print `daily_pct_change`
daily_pct_change.tail()
```

4.4 Visualise time series data

We will mainly use plotting functions provided by matplotlib. **Line plot** is the most common type of plot that we will use for analysis of stock data.

```
# Plot the closing prices for `aapl`
aapl['Close'].plot(grid=True)

# Show the line plot
plt.show()
```

Here is an example of plotting a histogram:

```
# Plot the distribution of `daily_pct_c`
daily_pct_change.hist(bins=50)

# Show the plot
plt.show()

# Pull up summary statistics
print(daily_pct_change.describe())
```

We can also create a new column to store the cumulative daily returns and plot the data in a graph.

```
# Calculate the cumulative daily returns
cum_daily_return = (1 + daily_pct_change).cumprod()

# Plot the cumulative daily returns
cum_daily_return.plot(figsize=(12,8))

# Show the plot
plt.show()
```

4.5 Moving windows

Moving windows (also called "rolling windows") are snapshots of a portion of a time series at an instant in time. It is common to use the moving window in a trading strategy, for example to calculate a moving average.

```
# Isolate the closing prices
close_px = aapl['Close']

# Calculate the moving average
moving_avg = close_px.rolling(window=40).mean()

# Inspect the result
moving_avg.tail()
```

We can now easily plot the short-term and long-term moving averages:

```
# Short moving window rolling mean
aapl['42'] = close_px.rolling(window=40).mean()

# Long moving window rolling mean
aapl['252'] = close_px.rolling(window=252).mean()

# Plot the adjusted closing price, the short and long windows of rolling means
aapl[['Close', '42', '252']].plot()

# Show plot
plt.show()
```

4.6 Summary

1. **Exploratory Data Analysis (EDA)**
    - `head()` and `tail()` - check first or last rows
    - `describe()` - mean, sd, range
    - `info()` - dtype, non-null, count
2. **Resampling**
    - `df.resample('M').mean` - downsample to months (small to big, reduce values)
    - `df.resample('T').ffill()` - upsample to minutes (big to small, add values)
3. **Percentage change**
    - `col.pct_change()`
4. **Visualise data**
    - `col.plot` - line plot
    - `col.hist(bins=50)` - histogram
5. **Moving window**
    - `close_px.rolling(window=40).mean()` - moving average

References

- Towards Data Science - What is Exploratory Data Analysis?
- Jason Brownlee - A Gentle Introduction to Statistical Sampling and Resampling
- Towards Data Science - Using the Pandas "Resample" Function
- Algorithmic trading explained
- DataCamp - Python for Finance: Algorithmic Trading

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 5

Technical analysis

In this tutorial, you will learn:

- The basics of technical analysis
- Technical analysis charts
- What are the common technical indicators
- How to implement technical indicators

5.1 Intro to technical analysis

As we have discussed in the first tutorial, technical analysis is a methodology for predicting price movement through the study of market action in the past. Technical analysis most commonly applies to price changes, but some technicians also track other data such as trading volume and open interest figures. Within the industry, there exist a tons of patterns and signals that have been developed by researchers in order to conduct technical analysis trading. Some technical indicators are primarily focused on identifying the current market trend, while some are aimed at spotting patterns in historical patterns.

In general, technicians consider the following types of indicators:

- Price trends
- Chart analysis
- Volume indicators
- Momentum indicators
- Oscillators
- Moving averages

In the upcoming tutorials for technical analysis, we will learn how to construct and analyse charts with the help of programming; and to build and implement different technical indicators widely used in the market.

Requirements:

- pandas
- matplotlib
- numpy

5.2 Chart analysis

Definition

Bullish signal means a signal to buy.
Bearish signal means a signal to sell.

5.2.1 Line chart

Line chart is the most basic type of chart used in finance, and we typically use it to plot a security's closing prices over time.

Using matplotlib, we could also plot the moving average and volume of a security:

```
plt.style.use('ggplot')

# Initialise the plot figure
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)

ax1 = plt.subplot2grid((6,1), (0,0), rowspan=5, colspan=1)
ax2 = plt.subplot2grid((6,1), (5,0), rowspan=1, colspan=1, sharex=ax1)

df['50ma'] = df['Close'].rolling(window=50, min_periods=0).mean()
df.dropna(inplace=True)

ax1.plot(df.index, df['Close'])
ax1.plot(df.index, df['50ma'])
ax2.bar(df.index, df['Volume'])

plt.show()
```

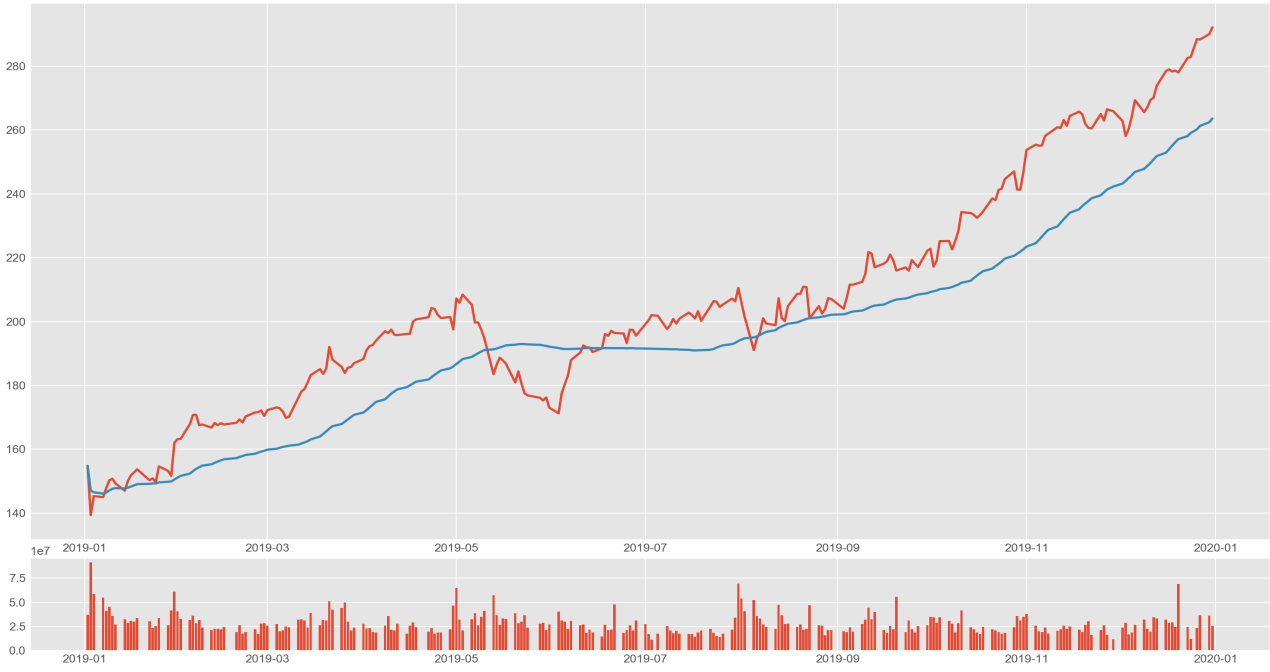


Fig. 1: Example of a line chart and a bar chart showing price and volume changes respectively.

5.2.2 Candlesticks chart

Candlestick charts originated in Japan over 100 years before the West developed the bar and point-and-figure charts. In the 1700s, a Japanese man named Homma discovered that, while there was a link between price and the supply and demand of rice, the markets were strongly influenced by the emotions of traders. Candlesticks show that emotion by visually representing the size of price moves with different colors. Traders use the candlesticks to make trading decisions based on regularly occurring patterns that help forecast the short-term direction of the price.

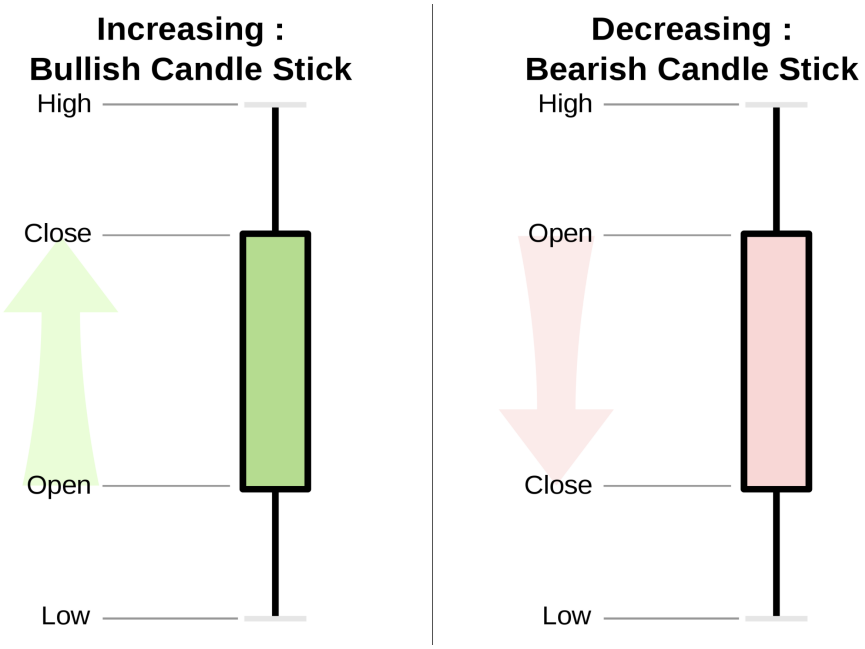


Fig. 2: Explanation of candlestick components.[1]

We could use `mpl_finance` to plot candlestick charts:

```
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)

ax1 = plt.subplot2grid((6,1), (0,0), rowspan=5, colspan=1)
ax2 = plt.subplot2grid((6,1), (5,0), rowspan=1, colspan=1, sharex=ax1)

# plot candlesticks
mpl_finance.candlestick_ohlc(ax1, data, width=0.7, colorup='g', colordown='r')

ax.grid() # show grids

############# x-axis locater settings #################

locator = mdates.AutoDateLocator() #  interval automically set
ax1.xaxis.set_major_locator(locator) # as as interval in a-axis
ax1.xaxis.set_minor_locator(mdates.DayLocator())
```

(continues on next page)

[1] By Probe-meteo.com - Probe-meteo.com, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=26048221

(continued from previous page)

```
############# x-axis locater settings #################

ax1.xaxis.set_major_formatter(mdates.AutoDateFormatter(locator)) # set x-axis label␣
↪as date format

fig.autofmt_xdate() # rotate date labels on x-axis

pos = df['Open'] - df['Close'] < 0
neg = df['Open'] - df['Close'] > 0
ax2.bar(df.index[pos],df['Volume'][pos],color='green',width=1,align='center')
ax2.bar(df.index[neg],df['Volume'][neg],color='red',width=1,align='center')

plt.show()
```

Fig. 3: Example of a candlestick chart.

5.2.3 Scaling

There are two types of scales for plotting charts - **arithmetic** or **semi-logarithmic**. As most of us who have studied science/mathematics should know, examples of logarithmic scales include growth of microbes, mortality rate due to epidemics and so on. The difference in scale can completely alter the shape of the chart even though it is plotted using the same set of data. Semi-logarithmic charts are sometimes more preferrable in order to overcome the weaknesses inherent in arithmetic charts.

Arithmetic scaling

In arithmetic or linear charts, both x and y axes scales are plotted at an equal distance.

Key points

- On a linear scale, as the distance in the axis increases the corresponding value also increases linearly.
- When the values of data fluctuate between extremely small values and very large values – the linear scale will **miss out the smaller values** thus conveying a wrong picture of the underlying phenomenon.

Semi-logarithmic scaling

A semi-log plot is a graph where the data in one axis is on logarithmic scale (either x axis or y axis), and data in the other axis is on normal scale (i.e. linear scale).

Key points

- On a logarithmic scale, as the distance in the axis increases the corresponding value increases exponentially.
- With logarithmic scale, **both smaller valued data and bigger valued data can be captured** in the plot more accurately to provide a holistic view.

Therefore, semi-logarithmic charts can be of immense help especially when plotting **long-term charts**, or when the price points show significant volatility even in short-term charts. The underlying chart patterns will be revealed more clearly in semi-logarithmic scale charts.

```
plt.style.use('ggplot')

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(18.5, 7.0)

### Subplot 1 - Semi-logarithmic ###
plt.subplot(121)
plt.grid(True, which="both")

# Linear X axis, Logarithmic Y axis
plt.semilogy(df.index, df['Close'], 'r')
plt.ylim([10,500])

plt.xlabel("Date")
```

(continues on next page)

(continued from previous page)

```
plt.title('Semi-logarithmic scale')
fig.autofmt_xdate()

### Subplot 2 - Arithmetic ###
plt.subplot(122)

plt.plot(df.index, df['Close'], 'b')

plt.xlabel("Date")
plt.title('Arithmetic scale')
fig.autofmt_xdate()

# show plot
plt.show()
```

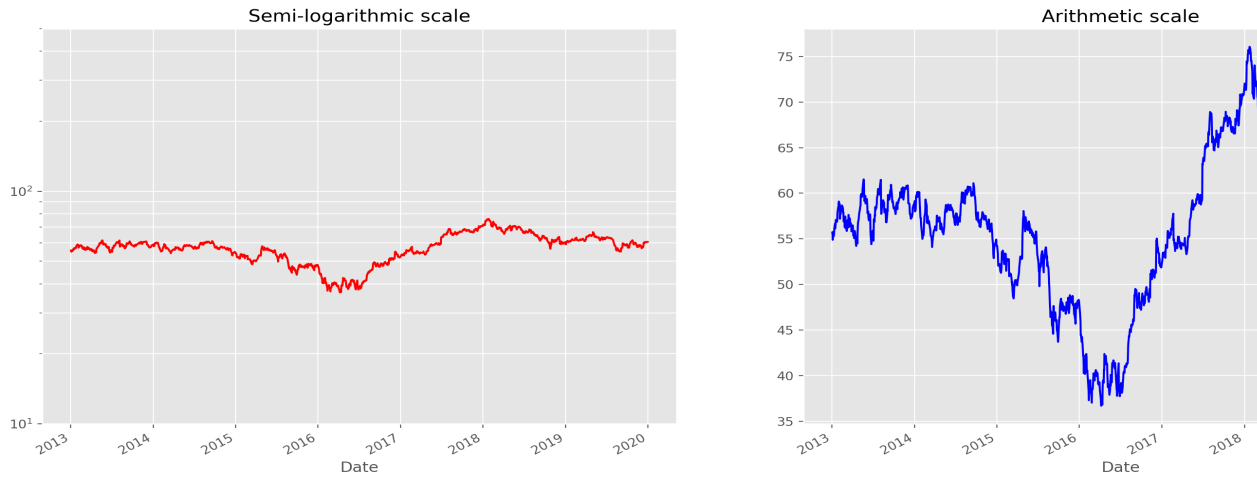


Fig. 4: The same data plotted with semi-logarithmic and arithmetic scales.

5.3 Technical indicators

The implementation of the below indicators could all be found in `code/technical-analysis_python/` in the repository.

In general, there are 2 categories of indicators:

- **Leading** - They give trade signals when the trend is about to started, hence they use shorter periods in their calculations. Examples are MACD and RSI.
- **Lagging** - They follow the price action, and thus gives a signal after a trend or a reversal started. Examples are Moving Averages and Bollinger Bands.

By calculating the technical indicators, we could create rules in order to generate entry points (i.e. buy and sell signals) in the market, and evaluate the performance of such a strategy using a **backtester**.

Definition

Backtesting is the process of applying a trading strategy to historical data in order to evaluate the performance of the strategy.

You could try running a strategy in the repository by:

```
# in terminal
cd code/technical-analysis_python
python main_macd_crossover.py # run macd in the backtester
```

The following provides the explanation and equations for all the example strategies featured in the repository.

5.3.1 Trend indicators

Trend indicators measure the direction and strength of a trend, using some form of price averaging to establish a baseline.

Definition

Exponential Moving Average (EMA) (a.k.a. exponentially weighted moving average) is a type of moving average (MA) that places a greater weight and significance on the most recent data points.

Moving Average Convergence Divergence (MACD)

The Moving Average Convergence Divergence (MACD) indicator is used to reveal changes in strength, direction, momentum and duration of a trend in a stock's price.

$$\text{MACD} = \text{12-Period EMA} \text{26-Period EMA}$$

One of the simplest strategy established with MACD, is to identify MACD **crossovers**. The rules are as follows.

Tip:

- **Buy signal**: MACD rises above the signal line
- **Sell signal**: MACD falls below the signal line

It is easy to calculate the EMA with pandas:

```
# Get adjusted close column
close = self.df['Close']

exp1 = close.ewm(span=12, adjust=False).mean()
exp2 = close.ewm(span=26, adjust=False).mean()
df['MACD'] = exp1 - exp2
```

`span` specifies the time span, and `adjust=False` means the exponentially weighted function is calculated recursively (as we do not need a decaying adjustment factor for beginning periods).

$$\text{Signal Line} = \text{9-day EMA of MACD Line}$$

To plot the signal line:

```
df['Signal line'] = self.df['MACD'].ewm(span=9, adjust=False).mean()
```

Moving Averages (MA)

Moving Averages (MA) are used to identify current trends and trend reversals, as well as to set up support and resistance levels.

We could estalish a simple trading strategy making use of two moving averages:

Tip:

- **Buy signal**: shorter-term MA crosses above the longer-term MA **(golden cross)**
- **Sell signal**: shorter-term MA crosses below the longer-term MA **(dead/death cross)**

Here is an example of how to plotting the two MAs:

```
# Create short simple moving average over the short window
signals['short_mavg'] = self.df['Close'].rolling(window=short_window, min_periods=1,␣
↪center=False).mean()

# Create long simple moving average over the long window
signals['long_mavg'] = self.df['Close'].rolling(window=long_window, min_periods=1,␣
↪center=False).mean()
```

We could define the `short_window` and `long_window` on our own, for example as setting `short_window = 40` and `long_window = 40`.

And then we could generate signals based on the two line plots:

```
# Generate signals
signals['signal'][short_window:] = np.where(signals['short_mavg'][short_window:]
                                            > signals['long_mavg'][self.short_
↪window:], 1.0, 0.0)

signals['positions'] = signals['signal'].diff()
```

Parabolic Stop and Reverse (Parabolic SAR)

The Parabolic Stop and Reverse (Parabolic SAR) indicator used to find potential reversals in the market price direction.

We need to calculate the **rising SAR** and **falling SAR** respectively:

(i) Rising SAR (Uptrend)

$$\text{Current SAR} = \text{Prior SAR} + \text{Prior AF} \times (\text{Prior EP} - \text{Prior SAR})$$

where:

- **Prior SAR**: The SAR value for previous period.
- **Extreme Point (EP)**: The highest high of the current uptrend.
- **Acceleration Factor (AF)**: Starting at 0.02, increases by 0.02 each time the extreme point makes a new high. AF can only reach a maximum of 0.2, no matter how long the uptrend extends.

Note that for rising SAR, the SAR can never be above the prior two periods' lows. Should SAR be above one of those lows, use the lowest of the two for SAR.

(ii) Falling SAR (Downtrend)

$$\text{Current SAR} = \text{Prior SAR} + \text{Prior AF} \times (\text{Prior EP} - \text{Prior SAR})$$

where:

- **Prior SAR:** The SAR value for previous period.
- **Extreme Point (EP):** The lowest low of the current downtrend.
- **Acceleration Factor (AF):** Starting at 0.02, increases by 0.02 each time the extreme point makes a new low. AF can only reach a maximum of 0.2, no matter how long the downtrend extends.

Note that for falling SAR, the SAR can never be below the prior two periods' highs. Should SAR be below one of those highs, use the highest of the two for SAR.

We generate signals based on the rising and falling SARs.

Tip:

- **Buy signal**: if falling SAR goes below the price
- **Sell signal**: if rising SAR goes above the price

In the code, we first need to extract the high / low / closing prices column from the dataframe, and initialise the arrays for storing the rising SAR and falling SAR:

```
array_high = list(df['High'])
array_low = list(df['Low'])
array_close = list(df['Close'])

psar = df['Close'].copy()
psarbull = [None] * len(df)
psarbear = [None] * len(df)

bull = True # flag to indicate saving value for rising SAR
af = initial_af # initialise acceleration factor
max_af = 0.2
ep = array_low[0] # extreme price
hp = array_high[0] # extreme high
lp = array_low[0] # extreme low
```

Then, traversing each row in the dataframe, we could calculate rising SAR and falling SAR at the same time:

```
for i in range(2, len(self.df)):
  if bull:
      # Rising SAR
      psar[i] = psar[i-1] + af * (hp - psar[i-1])
  else:
      # Falling SAR
      psar[i] = psar[i-1] + af * (lp - psar[i-1])

    reverse = False

    # Check reversion point
    if bull:
        if array_low[i] < psar[i]:
            bull = False
            reverse = True
            psar[i] = hp
            lp = array_low[i]
            af = initial_af
    else:
        if array_high[i] > psar[i]:
            bull = True
            reverse = True
            psar[i] = lp
            hp = array_high[i]
            af = initial_af
```

(continues on next page)

(continued from previous page)

```
    if not reverse:
        if bull:
            # Extreme high makes a new high
            if array_high[i] > hp:
                hp = array_high[i]
                af = min(af + initial_af, max_af)

            # Check if SAR goes abov prior two periods' lows.
            # If so, use the lowest of the two for SAR.
            if array_low[i-1] < psar[i]:
                psar[i] = array_low[i-1]
            if array_low[i-2] < psar[i]:
                psar[i] = array_low[i-2]

        else:
            # Extreme low makes a new low
            if array_low[i] < lp:
                lp = array_low[i]
                af = min(af + initial_af, max_af)

            # Check if SAR goes below prior two periods' highs.
            # If so, use the highest of the two for SAR.
            if array_high[i-1] > psar[i]:
                psar[i] = array_high[i-1]
            if array_high[i-2] > psar[i]:
                psar[i] = array_high[i-2]

    # Save rising SAR
    if bull:
        psarbull[i] = psar[i]
    # Save falling SAR
    else:
        psarbear[i] = psar[i]
```

5.3.2 Momentum indicators

Momentum indicators help identify the speed of price movement by comparing prices over time. Typically when there is a divergence between price and a momentum indicator, it can signal a change in future prices.

Commodity Channel Index (CCI)

The Commodity Channel Index (CCI) helps identify price reversals, price extremes, and trend strength.

Developed by Donald Lambert, CCI is a momentum-based oscillator used to help determine when an investment vehicle is reaching a condition of being **overbought or oversold**. It is also used to assess price trend direction and strength. This information allows traders to determine if they want to enter or exit a trade, refrain from taking a trade, or add to an existing position.

The formula for calculating CCI is given as follow.

$$\text{CCI} = \frac{(\text{Typical Price} - \text{x-period SMA of TP})}{(\text{Constant} \times \text{Mean Deviation})}$$

where:

- Typical Price (TP) = (High + Low + Close) / 3
- Constant = 0.015
- x = Window size (default set as 20)
- SMA: Simple Moving Average

We could first compute the the subcomponents of CCI:

```
signals['Typical price'] = (df['High'] + df['Low'] + df['Close']) / 3

signals['SMA'] = signals['Typical price'].rolling(
                 window=self.window_size, min_periods=1, center=False).mean()

signals['mean_deviation'] = signals['Typical price'].rolling(
                            window=20, min_periods=1, center=False).std()
```

Then calculate CCI using the formula:

```
signals['CCI'] = (signals['Typical price'] - signals['SMA']) /
                 (self.constant * signals['mean_deviation'])
```

A simple strategy formulated by using CCI is (the thresholds only serve as examples:

Tip:

- **Buy signal**: when CCI surges above +100
- **Sell signal**: when CCI plunges below -100

To implement this rule in code:

```
# Generate buy signal
signals.loc[signals['CCI'] > 100, 'signal'] = 1.0

# Generate sell signal
signals.loc[signals['CCI'] < -100, 'signal'] = -1.0
```

Relative Strength Index (RSI)

The Relative Strength Index (RSI) measures recent trading strength, velocity of change in the trend, and magnitude of the move.

$$\text{RSI} = 100 - \frac{100}{(1 + \text{RS})}$$

$$\text{RS} = \frac{\text{Average Gain}}{\text{Average Loss}}$$

where **Average Gain** and **Average Loss** are calculated as follows:

$$\text{First Average Gain} = \frac{\text{Sum of gains over the past 14 periods}}{14}$$

$$\text{First Average Loss} = \frac{\text{Sum of losses over the past 14 periods}}{14}$$

$$\text{Average Gain} = \frac{\text{Previous average gain} \times 13 + \text{Current gain}}{14}$$

$$\text{Average Loss} = \frac{\text{Previous average loss} \times 13 + \text{Current loss}}{14}$$

Note that the first calculations are just simple 14-period averages. Subsequent averages take the prior value plus the current value to compute the average. This is a smoothing technique similar to that used in calculating an exponential moving average. Thus, the RSI values become more accurate as the calculation period extends.

In the dataset, we need to extract gains and losses from the price column respectively:

```
# Get adjusted close column
close = df['Close']

# Get the difference in price from previous step
delta = close.diff()
# Get rid of the first row
delta = delta[1:]

# Make the positive gains (up) and negative gains (down) series
up, down = delta.copy(), delta.copy()
up[up < 0] = 0
down[down > 0] = 0
```

To calculate RS, as well as RSI:

```
# Calculate SMA using 'rolling' function
roll_up = up.rolling(window_size).mean()
roll_down = down.abs().rolling(window_size).mean()

# Calculate RSI based on SMA
RS = roll_up / roll_down
RSI = 100.0 - (100.0 / (1.0 + RS))
```

The output of the RSI is a number on a scale **from 0 to 100** and it is typically calculated on a 14-day basis. To generate the trading signals, it is common to **specify the low and high levels of the RSI** (e.g. at 30 and 70

respectively). The interpretation of the thresholds is that the lower one indicates that the asset is oversold, and the upper one that the asset is overbought.

Tip:

- **Oversold**: when RSI crosses the lower threshold (e.g. 30)
- **Overbought**: when RSI crosses the upper threshold (e.g. 70)

Rate of Change (ROC)

The Rate of Change (ROC) measures the strength of price momentum.

Positive values of the ROC indicates upward buying pressure or momentum, while **negative values** below zero indicate selling pressure or downward momentum. Increasing values in either direction, positive or negative, indicate increasing momentum, and decreasing values indicate waning momentum.

$$\text{ROC} = \frac{(\text{Closing price} - \text{Closing price n periods ago})}{(\text{Closing price n periods ago})} \times 100$$

As you could see from above, it's just the simple percentage change formula.

We could identify overbought and oversold conditions using ROC:

Tip:

- **Oversold**: when ROC crosses the lower threshold (e.g. -30)
- **Overbought**: when ROC crosses the upper threshold (e.g. +30)

And here is one of the possible ways to calculate ROC:

```
n = 12 # set time period

diff = df['Close'].diff(n - 1)

# Calculate closing price n periods ago
closing = self.df['Close'].shift(n - 1)

df['ROC'] = (diff / closing) * 100
```

Stochastic Oscillator (STC)

Stochastic Oscillators (STC) are used to predict price turning points by comparing the closing price to its price range.

Highest High = highest high for the look-back period

Note that in the formula %K is multiplied by 100 so as to move the decimal point by two places.

We could traverse the dataframe and store all highests highs, lowest lows in two separate arrays:

```
array_highest = [0] * length # store highest highs
for i in range(k - 1, length):
    highest = array_high[i]
    for j in range(i - 13, i + 1): # k-day lookback period
        if array_high[j] > highest:
            highest = array_high[j]
    array_highest[i] = highest

array_lowest = [0] * length # store lowest lows
for i in range(k - 1, length):
    lowest = array_low[i]
    for j in range(i - 13, i + 1): # k-day lookback period
        if array_low[j] < lowest:
            lowest = array_low[j]
    array_lowest[i] = lowest
```

Then, we can calculate %K and %D:

```
# find %K line values
kvalues = [0] * length

for i in range(self.k - 1, length):
    k = ((array_close[i] - array_lowest[i]) * 100) / (array_highest[i] - array_
↪lowest[i])
    kvalues[i] = k

df['%K'] = kvalues

# find %D line values
df['%D'] = df['%K'].rolling(window=3, min_periods=1, center=False).mean()
```

A strategy established with %K and %D is as follows:

Tip:

- **Buy signal**: when %K line crosses above the %D line
- **Sell signal**: when %K line crosses below the %D line

True Strength Index (TSI)

The True Strength Index (TSI) is a momentum oscillator based on a double smoothing of price changes. By smoothing price changes, it captures the ebbs and flows of price action with a steadier line that tries to filter out noises.

$$\text{TSI} = \frac{\text{Double Smoothed PC}}{\text{Double Smoothed Absolute PC}} \times 100$$

where:

(i) Double Smoothed Price Change (PC)

- PC = Current Price - Prior Price
- First Smoothing = 25-period EMA of PC
- Second Smoothing = 13-period EMA of 25-period EMA of PC

(ii) Double Smoothed Absolute Price Change (PC)

- Absolute Price Change | PC | = Absolute Value of Current Price minus Prior Price
- First Smoothing = 25-period EMA of | PC |
- Second Smoothing = 13-period EMA of 25-period EMA of | PC |

Based on the above formulae, the code is shown as follow:

```
df['Double Smoothed PC'] = pc.ewm(span=25, adjust=False).mean().ewm(
                                  span=13, adjust=False).mean()

df['Double Smoothed Abs PC'] = abs(pc).ewm(span=25, adjust=False).mean().ewm(
                                      span=13, adjust=False).mean()

df['TSI'] =  df['Double Smoothed PC'] / df['Double Smoothed Abs PC'] * 100
```

In order to interpret the TSI, we could define a signal line:

$$\text{Signal line} = \text{10-period EMA of TSI}$$

And we could observe signal line crossovers:

Tip:

- **Buy signal**: when TSI crosses above the signal line from below
- **Sell signal**: when TSI crosses below the signal line from above

---

Money Flow Index (MFI)

The Money Flow Index (MFI) is an oscillator that generates overbought or oversold signals using both prices and volume data.

$$\text{Money Flow Index} = 100 - \frac{100}{(1 + \text{Money Flow Ratio})}$$

$$\text{Raw Money Flow} = \text{Typical Price} \times \text{Volume}$$

$$\text{Typical Price} = \frac{(\text{High} + \text{Low} + \text{Close})}{3}$$

$$\text{Money Flow Ratio} = \frac{\text{14-period Positive Money Flow}}{\text{14-period Negative Money Flow}}$$

It is pretty straightforward to calculate typical price:

```
# Typical price
tp = (df['High'] + df['Low'] + df['Close']) / 3.0
```

With regards to the positive and negative money flows, we will first want to compute the raw money flow, then label which of them belong to positive / negative respectively:

```
# positive = 1, negative = -1
self.df['Sign'] = np.where(tp > tp.shift(1), 1, np.where(tp < tp.shift(1), -1, 0))

# Raw money flow
df['Money flow'] = tp * df['Volume'] * df['Sign']

# Positive money flow with n periods
n_positive_mf = df['Money flow'].rolling(n).apply
                (lambda x: np.sum(np.where(x >= 0.0, x, 0.0)), raw=True)

# Negative money flow with n periods
n_negative_mf = abs(df['Money flow'].rolling(self.n).apply
                (lambda x: np.sum(np.where(x < 0.0, x, 0.0)), raw=True))
```

With the money flows, it would be easy to compute the MFI:

```
mf_ratio = n_positive_mf / n_negative_mf
df['MFI'] = (100 - (100 / (1 + mf_ratio)))
```

By way of example, we could use MFI to identify overbought and oversold conditions:

Tip:

- **Oversold**: when MFI crosses the upper threshold
- **Overbought**: when MFI crosses the lower threshold

William %R

William %R reflects the level of the close relative to the highest high for the look-back period.

Highest High = highest high for the look-back period

Note that in the formula, %R is multiplied by -100 to correct the inversion and move the decimal. The default setting for the lookback period is 14, which can be days, weeks, months or an intraday timeframe.

The code for implementing %R is shown as follows:

```
lbp = 14 # set lookback period

hh = df['High'].rolling(lbp).max()  # highest high over lookback period
ll = df['Low'].rolling(lbp).min()  # lowest low over lookback period
df['%R'] = -100 * (hh - df['Close']) / (hh - ll)
```

Similarly, we could use %R to identify overbought and oversold conditions:

Tip:

- **Oversold**: when %R goes below -80
- **Overbought**: when %R goes above -20

5.3.3 Volatility indicators

These indicators measure the rate of price movement, regardless of direction. Generally, they are based on a change in the highest and lowest historical prices. They provide useful information about the range of buying and selling that takes place in a given market, and help traders determine points where the market may change direction.

Bollinger Bands

The Bollinger Bands indicator is used to measure the "highness" or "lowness" of the price, relative to previous trades.

The Bollinger Bands consist of a middle band with two outer bands:

$$\begin{aligned}\text{Middle Band} &= \text{20-day simple moving average (SMA)}\\ \text{Upper Band} &= \text{20-day SMA} + (\text{20-day standard deviation of price} \times 2)\\ \text{Lower Band} &= \text{20-day SMA} - (\text{20-day standard deviation of price} \times 2)\end{aligned}$$

We could first compute the middle band and the 20-day standard deviation:

```
window = 20

# Compute middle band
df['Middle band'] = self.df['Close'].rolling(window).mean()

# Compute 20-day s.d.
mstd = df['Close'].rolling(window).std(ddof=0)
```

Then we could get the outer bands:

```
# Computer upper and lower bands
df['Upper band'] = df['Middle band']  + mstd * 2
df['Lower band'] = df['Middle band']  - mstd * 2
```

The signals could be derived from observing the below conditions:

Tip:

- **Buy signal**: when price goes below lower band
- **Sell signal**: when price goes above upper band

Average True Range

Average True Range (ATR) is an indicator that tries to measure the degree of price volatility.

$$\text{Current ATR} = \frac{(\text{Prior ATR} \times 13) + \text{Current TR}}{n}$$

where:

- 1st True Range (TR) value = High - Low
- 1st n-day ATR = average of the daily TR values for the last n days

The True Range could be computed by:

```
array_high = list(df['High'])
array_low = list(df['Low'])

tr = [None] * len(df) # initialisation

for i in range(len(df)):
    tr[i] = array_high[i] - array_low[i]
```

We would first calculate the first n-day ATR:

```
atr = [None] * len(self.df) # initialisation
window = 14

atr[15] = sum(tr[0:15]) / window
```

Then for the following ATRs:

```
for i in range(16,len(self.df)):
    atr[i] = (atr[i-1] * (window-1) + tr[i]) / window
```

We could determine whether the stock is **highly volatile** by checking the following conditions:

$$\text{50-day EMA} > \text{200-day EMA}$$

AND

$$\frac{\text{250-day ATR}}{\text{20-day SMA}} \times 100 < 4$$

Tip: We could use ATR to filter out stocks that are **highly volatile**.

Standard Deviation

Standard Deviation measures expected risk and determines the significance of certain price movements.

As an example, we could set `window=21`:

```
window = 21
df['SD'] = df['Close'].rolling(window).std(ddof=0)
```

Tip: We could use Standard Deviation to measure the **expected risk** of stocks.

5.3.4 Volume indicators

Volume indicators measure the strength of a trend or confirm a trading direction based on some form of averaging or smoothing of raw volume. The strongest trends often occur while volume increases; in other words, some would assume that it is the increase in trading volume that can lead to large movements in price.

Chaikin Oscillator

The Chaikin Oscillator indicator monitors the flow of money in and out of the market. By comparing money flow to price action, it helps identify tops and bottoms in short and intermediate cycles.

$$\text{Chaikin Oscillator} = \text{3-day EMA of ADL} - \text{10-day EMA of ADL}$$

$$\begin{aligned}\text{Accumulation Distribution Line (ADL)} &= \text{Previous ADL} \\ &+ \text{Current Period's Money Flow Volume}\end{aligned}$$

$$\begin{aligned}\text{Money Flow Volume} &= \text{Money Flow Multiplier} \\ &\times \text{Volume for the Period}\end{aligned}$$

$$\text{Money Flow Multiplier} = \frac{(\text{Close} - \text{Low}) - (\text{High} - \text{Close})}{(\text{High} - \text{Low})}$$

We would first calculate the Money Flow Multiplier:

```
df['MFM'] = ((df['Close'] - df['Low']) - df['High'] - df['Close'])
            / (df['High'] - df['Low'])
```

Then use it to calculate Money Flow Volume:

```
df['MFV'] = df['MFM'] * df['Volume']
```

Following, we could compute ADL and the Chaikin Oscillator:

```
df['ADL'] = df['Close'].shift(1) + df['MFV']

short_w = 3
long_w = 10
ema_long = df['ADL'].ewm(ignore_na=False, min_periods=0, com=short_w, adjust=True).
↪mean()
ema_short = df['ADL'].ewm(ignore_na=False, min_periods=0, com=long_w, adjust=True).
↪mean()

df['Chaikin'] = ema_short - ema_long
```

We could establish the following simple strategy, as an example:

Tip:

- **Buy signal**: when the oscillator is positive
- **Sell signal**: when the oscillator is negative

On-Balance Volume (OBV)

The On Balance Volume indicator attempts to measure the level of accumulation or distribution by comparing volume to price movements.

The formula for OBC changes according to the following 3 cases:

1) If closing price > prior close price:

$$\text{Current OBV} = \text{Previous OBV} + \text{Current Volume}$$

2) If closing price < prior close price:

$$\text{Current OBV} = \text{Previous OBV} - \text{Current Volume}$$

3) If closing price = prior close price then:

$$\text{Current OBV} = \text{Previous OBV (no change)}$$

We could traverse the dataframe, and use if-else statements to capture the 3 conditions:

```
obv = [0] * len(self.df) # for storing the on-balance volume

array_close = list(df['Close'])
array_volume = list(df['Volume'])

for i in range(1, len(self.df)):
    if (array_close[i] > array_close[i-1]):
        obv[i] = obv[i-1] + array_volume[i]
    elif (array_close[i] < array_close[i-1]):
        obv[i] = obv[i-1] - array_volume[i]
    else:
        obv[i] = obv[i-1]
```

The absolute value of OBV is not important. We should instead focus on the characteristics of the OBV line and its the trend.

Tip:

- A rising OBV reflects positive volume pressure that can lead to **higher prices**
- A falling OBV reflects negative volume pressure that can foreshadow **lower prices**

Volume Rate of Change

The Volume Rate of Change (Volume ROC) highlights increases in volume, which normally occurs at most significant market tops, bottoms and breakouts.

$$\text{Volume Rate of Change} = \frac{\text{Volume (today)} - \text{Volume (n days ago)}}{\text{Volume (n days ago)}}$$

The way of calculating Volume ROC is similar to ROC:

```
n = 25 # example time period
df['Volume ROC'] = ((df['Close'] - df['Close'].shift(n)) /
                    df['Close'].shift(n))
```

Here is a simple example strategy based on Volume ROC:

Tip:

- **Buy signal**: if Volume ROC goes below zero
- **Sell signal**: if Volume ROC is negative

References

- Investopedia
- StockCharts
- Parabolic SAR

Image sources

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 6

Fundamental analysis

In this tutorial, you will learn:

- The basics of fundamental analysis
- What are financial ratios
- How to carry out stock screening

6.1 Intro to fundamental analysis

As we have discussed in the first tutorial, fundamental analysis is a methodology for predicting the security's intrinsic value, which may or may not be equal to its current market value. In general, fundamental analysis could be divided into **quantitative** and **qualitative**.

- **Quantitative** aspect concerns about studying numerical figures like the company's revenues, profits, assets, debts, etc
- **Qualitative** aspect focuses on intangible factors like the company's management background and brand recognition.

In addition, it is also crucial to look at economic factors like interest rates, inflation, GDP, etc. These can all affect the general situation of the company.

In this module, we'll focus on the quantitave aspect of a company. We could get the numerical figures we need from the company's income statements, cash flow statements and balance sheets.

6.2 Financial ratios

Instead of using the raw figures we obtained, we would use these figures to compute **financial ratios** that help us summarise the financial statements and the health of a company.

Here are four basic ratios that are often used to pick stocks for investment portfolios:

- Price-earnings (P/E) ratio
- Earnings per share (EPS) ratio
- Debt-to-equity ratio
- Return on equity (ROE) ratio

In the following, we'lll go through the major types of ratios and the equation for each ratio one by one.

6.2.1 Short-term solvency ratios

Definition

Short-term solvency ratios are used to judge the adequacy of liquid assets for meeting short-term obligations as they come due.

The **higher** the ratio result, the better a company's liquidity and financial health; the lower the ratio, the more likely the company will struggle with paying debts.

Current ratio

Current ratio measures a company's ability to pay short-term obligations or those due within one year.

$$\text{Current ratio} = \frac{\text{Current assets}}{\text{Current liabilities}}$$

Quick ratio

Quick ratio indicates a company's capacity to pay its current liabilities without needing to sell its inventory or get additional financing.

$$\text{Quick ratio} = \frac{\text{Current assets} - \text{Inventory}}{\text{Current liabilities}}$$

Cash ratio

Cash ratio calculates a company's ability to repay its short-term debt with cash or near-cash resources, such as easily marketable securities.

$$\text{Cash ratio} = \frac{\text{Cash equivalents} + \text{Cash}}{\text{Current liabilities}}$$

Networking capital to current liabilities

text{Networking capital to current liabilities} = frac{text{Current assets} + text{Current liabilities}}{text{Current liabilities}}

6.2.2 Turnover ratios

Definition

Turnover ratios represent the amount of assets or liabilities that a company replaces in relation to its sales. The concept is useful for determining the efficiency with which a business utilises its assets.

Inventory turnover ratios

Inventory turnover ratios is a ratio that measures the number of times inventory is sold or consumed in a given time period.

$$\text{Inventory turnover ratio} = \frac{\text{Cost of goods sold}}{\text{Average inventories}}$$

Receivable turnover

Receivable turnover is the number of times per year that a business collects its average accounts receivable. It evaluates the ability of a company to efficiently issue credit to its customers and collect funds from them in a timely manner.

$$\text{Receivable turnover} = \frac{\text{Sales}}{\text{Accounts receivable}}$$

Fixed asset turnover

Fixed asset turnover indicates how well the business is using its fixed assets to generate sales.

$$\text{Fixed asset turnover} = \frac{\text{Net sales}}{\text{Average fixed assets}}$$

Total asset turnover

Total asset turnover measures the efficiency with which a company uses its assets to produce sales.

$$\text{Total asset turnover} = \frac{\text{Net sales}}{\text{Average total assets}}$$

6.2.3 Financial leverage ratios

Definition

Financial leverage ratios indicates the level of debt incurred by a business entity against several other accounts in its balance sheet, income statement, or cash flow statement. They show how the company's assets and business operations are financed (using debt or equity).

Total debt ratio

Total debt ratio represents the proportion of a company's assets that are financed by debt.

$$\text{Total debt ratio} = \frac{\text{Total debts}}{\text{Total assets}}$$

Debt to equity ratio

Debt to equity ratio measures the degree to which a company is financing its operations through debt versus wholly-owned funds.

$$\text{Debt to equity ratio} = \frac{\text{Total debts}}{\text{Total equity}}$$

Equity ratio

Equity ratio represents the relative proportion of equity used to finance a company's assets.

$$\text{Equity ratio} = \frac{\text{Shareholder's equity}}{\text{Total asset}}$$

Long-term debt ratio

Long-term debt ratio shows the proportion of a company's assets it would have to liquidate to repay its long-term debt.

$$\text{Long-term debt ratio} = \frac{\text{Long-term liabilities}}{\text{Total equity}}$$

Times interest earned ratio

Times interest earned ratio measures a company's ability to meet its debt obligations based on its current income.

$$\text{Times interest earned ratio} = \frac{\text{Earnings before interest and tax (EBIT)}}{\text{Total interest expense}}$$

6.2.4 Profitability ratios

Definition

Profitability ratios measure the ability of a company to generate income (profit) relative to revenue, balance sheet assets, operating costs, and shareholders' equity during a specific period of time. They show how well a company utilises its assets to produce profit and value to shareholders.

Gross profit margin

Gross profit margin indicates how efficient a business is at managing its operations.

$$\text{Gross profit margin} = \frac{\text{Revenue} - \text{Cost of goods sold (COGS)}}{\text{Revenue}}$$

Net profit margin

Net profit margin indicates how much of each dollar in revenue collected by a company translates into profit.

$$\text{Net profit margin} = \frac{\text{Revenue} - \text{Cost}}{\text{Revenue}}$$

Return on assets (ROA)

Return on assets indicates how profitable a company is relative to its total assets.

$$\text{Return on assets} = \frac{\text{Net income}}{\text{Total assets}}$$

Return on equity (ROE)

Return on equity measures the profits made for each dollar from shareholders' equity.

$$\text{Return on equity} = \frac{\text{Net income}}{\text{Shareholder's equity}}$$

Earning per share (EPS)

Earning per share indicates how much money a company makes for each share of its stock.

$$\text{Earning per share} = \frac{\text{Net income} - \text{Preferred stock dividend}}{\text{Average outstanding shares}}$$

6.3 Ratio analysis for stock screening

The ratio analysis example could be found in :code: *code/fundamental-analysis/ratio-analysis.ipynb*.

With the equations above, it would be easy to compute the ratios just with column manipulations:

```
# take equity ratio as an example
equity_ratio = df["Total stockholders\' equity"].astype(int)
               / df['Total Assets'].astype(int) # Total stockholders' equity / Total␣
↪Assets

# store all the ratios in the `ratios` dataframe
ratios["Equity ratio"] = equity_ratio
```

After calculating these ratios, we would want to create screening masks to filter out stocks that have a relatively better performance.

```
# e.g. filter out stocks with profitability ratio greater than overall mean
mask1 = (ratios['EPS'] > ratios['EPS'].mean())
mask2 = (ratios['ROE'] > ratios['ROE'].mean())
mask3 = (ratios['ROA'] > ratios['ROA'].mean())
```

We could apply these masks on the ratios dataframe to get the filtered stocks eventually:

```
# apply the masks
ratios[(mask1) & (mask2) & (mask3)]
```

References

- NASDAQ - Short-term solvency ratios
- Accounting Tools

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 7

Evaluation metrics

This provides the explanation and equation of the evaluation metrics implemented in the backtester. The code could be found in `code/evalaute.py` in the repository.

7.1 Portfolio return

Definition

Portfolio return is the percentage change in total value of the portfolio.

$$\text{Portfolio return} = \frac{\text{Current portfolio value} - \text{Previous portfolio value}}{\text{Previous portfolio value}} \times 100$$

7.2 Sharpe ratio

Definition

Sharpe ratio computes the ratio of the return of an investment to its risk.

Mathematically, it is the average return earned in excess of the risk-free rate per unit of total volatility.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

$R_p =$ Portfolio return
$R_f =$ Risk-free rate$(assumed = 0 in the function)$
$\sigma_p =$ Portfolio risk, i.e. standard deviation

7.3 Maximum drawdown (MDD)

Definition

Maximum drawdown (MDD) measures the maximum observed loss from a peak to a trough of a portfolio. It is an indicator of downside risk over the given time period.

$$\text{MDD} = \frac{\text{P} - \text{T}}{\text{P}}$$

P = Peak value before largest drop
L = Lowest value before newest high established

7.4 Compound Annual Growth Rate (CAGR)

Definition

Compoumd Annual Growth Rate (CAGR) describes the rate at which an investment would have grown if it had grown the same rate every year and the profits were reinvested at the end of each year.

It is used to smooth returns so that they may be more easily understood when compared to alternative investments. Risk is not taken into account.

$$\text{CAGR} = \left(\frac{\text{Ending portfolio value}}{\text{Beginning portfolio value}}\right)^{252 \div \text{number of days}} - 1$$

Note that "number of days" refers to the time span of the portfolio, and 252 is the total number of trading days in one year.

7.5 Standard Deviation

Definition :class: myOwnStyle

Standard Deviation measures the dispersion of the historical portfolio values relative to its mean. A higher standard deviation infers higher volatility.

$$\text{SD} = \sqrt{\frac{\sum(r_i - r_{avg})^2}{n-1}}$$

$r_i =$ Portfolio daily return
$r_{avg} =$ Mean of portfolio daily returns

Note that sample SD is used, and thus the degree of freedom equals n-1 in the equation.

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 8

Introduction to Julia

Julia is a high-level, high-performance programming language that is designed for numerical analysis and computational science. Nature describes it as the "*best of both worlds*" - a language that combines the interactivity and syntax of 'scripting' languages, such as Python, Matlab and R, with the speed of 'compiled' languages such as Fortran and C.

8.1 Why use Julia

Julia is designed for speed, performance and scalability. Here are some highlights about the best features of Julia:

- It has a legible syntax and is easy to learn.
- It incorporates vector notations and DataFrames as part of the language.
- It compiles codes in advance and thus is designed to be fast.

Many financial firms including BlackRock and State Street are using Julia as their primary development language. With the belief that the community for Julia will continue to grow, here we would like to introduce some examples of using Julia to write algorithmic trading strategies.

8.2 Getting started

The official repo and installation instructions could be found here.

Here are some good tutorials for picking up the language:

- Julia By Example
- Julia Tutorials (from official website)

References

- Is Julia the best language for quantitave finance?
- Why Julia for Finance?

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 9

Data science basics

In this tutorial, you will learn to do the following in Julia:

- Load and output csv files
- Manipulate DataFrames

You could run the code for this tutorial in `code/technical-analysis_julia/data-science-basics.ipynb`. Make sure you have installed Julia and all the required dependencies (follow instructions here).

You would need to install and import the following libraries:

```
# import libraries
using CSV;
using Dates;
using DataFrames;
using Statistics;
using Plots;
using StatsPlots;
using RollingFunctions;
```

9.1 Read and output csv file

The CSV package is useful for loading and manipulating dataframes.

```
# load data
df = CSV.File("../../database/hkex_ticks_day/hkex_0001.csv") |> DataFrame
```

```
# save as csv
CSV.write("test.csv", df)
```

9.2 Data inspection

The `first` and `last` functions are similar to `head` and `tail` in pandas. Additionally, we also have `describe` that returns a summary of the dataframe.

```
first(df, 5) # show first 5 rows
last(df, 5) # last 5 rows
```

```
describe(df) # get summary of df
```

We can get the column names by:

```
names(df) # column names
```

9.3 Data selection

As we always select rows within a particular date range for stock price data, here is how to do it:

```
df[(df.Date .> Date(2017, 1)) .& (df.Date .< Date(2019, 1)), :]
```

Alteratively, we could generate a list of dates and check if date is in this range:

```
dates = [Date(2017, 1),Date(2018)];
yms = [yearmonth(d) for d in dates];
print(yms) # [(2017, 1), (2018, 1)]

df[in(yms).(yearmonth.(df.Date)), :], 10
```

We can select column(s) by the following way:

```
close = select(df, :Close) # select column "Close"
close = select(df, [:Close, :Volume]) # select columns "Close", "Volume"
```

References

- Towards Data Science - What is Exploratory Data Analysis?
- Jason Brownlee - A Gentle Introduction to Statistical Sampling and Resampling
- Towards Data Science - Using the Pandas "Resample" Function
- Algorithmic trading explained
- DataCamp - Python for Finance: Algorithmic Trading

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 10

Write a technical strategy

In this tutorial, you will learn to do the following in Julia:

- Write a trading strategy
- Backtest the strategy
- Plot the trading signals
- Evalaute strategy performance

You could run the code for this tutorial in `code/technical-analysis_julia/moving_average.jl` or `code/technical-analysis_julia/moving-average.ipynb`.

You would need to install and import the following libraries:

```
# import libraries
using CSV;
using Dates;
using DataFrames;
using Statistics;
using Plots;
using PyCall;
using RollingFunctions;

@pyimport matplotlib.pyplot as plt
```

First, load the file from the sample database:

```
# load data
df = CSV.File("../../database/microeconomic_data/hkex_ticks_day/hkex_0001.csv") |>␣
↪DataFrame

ticker = "0001.HK" # set this variable for plot title
```

10.1 Build the strategy

We will use Moving Average (MA) as the example here. You could refer to the section *Moving Averages (MA)* to read the formulae and conditions for generating buy/sell signals.

Calculate the short and long moving averages, and respectively append them to the `signals` dataframe:

```
# initialise signals dataframe
signals = df[:,[:Date, :Close]]

dates = Array(convert(Matrix, select(df, :Date))) # get dates
close = convert(Matrix, select(df, :Close)) # get closing price

# short MA
short_window = 40
short_mavg = runmean(vec(close), short_window)
insertcols!(signals, 1, :short_mavg => short_mavg)

# long MA
long_window = 100
long_mavg = runmean(vec(close), long_window)
insertcols!(signals, 1, :long_mavg => long_mavg)
```

Generate the buy and sell signals based on the `short_mavg` and `long_mavg` colums:

```
# Create signals
signal = Float64[]

for i in 1:length(short_mavg)
    if short_mavg[i] > long_mavg[i]
        x = 1.0 # buy signal
    else
        x = 0.0
    end
    push!(signal, x)
end

insertcols!(signals, 1, :signal => signal)
```

Generate the positions by taking the row differences in `signal`:

```
# Generate positions
function gen_pos(signal)
    positions = zeros(length(signal))
    positions[1] = 0
    for i in 2:length(signal)
        positions[i] = signal[i] - signal[i-1]
    end
    return positions
end

positions = gen_pos(signal)
insertcols!(signals, 1, :positions => positions)
```

We also generate temporary arrays for plotting buy and sell signals respectively:

```
# Generate tmp arrays to plot buy signals

buy_signals = DataFrame()
buy_dates = []
buy_prices = []

for i in 1:length(positions)
    if (positions[i] == 1.0)
      push!(buy_dates, dates[i])
      push!(buy_prices, close[i])
    end
end

insertcols!(buy_signals, 1, :Date => buy_dates)
insertcols!(buy_signals, 1, :Price => buy_prices)
#print(first(buy_signals,10))

# Generate tmp arrays to plot sell signals

sell_signals = DataFrame()
sell_dates = []
sell_prices = []

for i in 1:length(positions)
    if (positions[i] == -1.0)
      push!(sell_dates, dates[i])
      push!(sell_prices, close[i])
    end
end

insertcols!(sell_signals, 1, :Date => sell_dates)
insertcols!(sell_signals, 1, :Price => sell_prices)
```

10.2 Plotting graphs

As we make use of `matplotlib` to plot graphs, the functions are very similar to those we have used in Python.

```
fig = plt.figure() # Initialise the plot figure
ax1 = fig.add_subplot(111, ylabel="Price in \$") # Add a subplot and label for y-axis

# plot moving averages as line
plt.plot(signals.Date, signals.short_mavg, color="blue", linewidth=1.0, label="Short␣
↪MA")
plt.plot(signals.Date, signals.long_mavg, color="orange", linewidth=1.0, label="Long␣
↪MA")

# plot signals with colour markers
plt.plot(buy_signals.Date, buy_signals.Price, marker=10, markersize=7, color="m",␣
↪linestyle="None", label="Buy signal")
plt.plot(sell_signals.Date, sell_signals.Price, marker=11, markersize=7, color="k",␣
↪linestyle="None", label="Sell signal")

plt.title("MA crossover signals")
plt.show()
```

(continues on next page)

(continued from previous page)

```
# save fig
fig.savefig("./figures/moving-average-crossover_signals", dpi=100)
```

10.3 Backtesting

We could then backtest the strategy on the historical price data:

```
initial_capital = 100000.0

# Initialise the portfolio with value owned
portfolio = signals[:,[:Date, :Close, :positions]]
portfolio[:trade] = signals[:Close] .* (100 .* signals[:positions])

# Add `holdings` to portfolio
portfolio[:quantity] = cumsum(100 .* signals[:positions])
portfolio[:holdings] = portfolio[:Close] .* portfolio[:quantity]

# Add `cash` to portfolio
portfolio[:cash] = initial_capital .- cumsum(portfolio[:trade])

# Add `total` to portfolio
portfolio[:total] = portfolio[:cash] .+ portfolio[:holdings]

portfolio_total = Array(portfolio[:total])

# Generate returns
function gen_returns(portfolio_total)
    returns = zeros(length(portfolio_total))
    returns[1] = 0
    for i in 2:length(portfolio_total)
        returns[i] = (portfolio_total[i] - portfolio_total[i-1]) / portfolio_total[i-
↪1]
    end
    return returns
end

returns = gen_returns(portfolio_total)
insertcols!(portfolio, 1, :returns => returns)

# Print final portfolio value and total return in terminal

@printf("Final total value: %f\n", portfolio.total[size(portfolio,1)])

total_return = (portfolio.total[size(portfolio,1)] - portfolio.total[1]) / portfolio.
↪total[1]
@printf("Total return: %f\n", total_return)
```

10.4 Strategy evaluation

Note that all the evalation metric functions are designed to take the portfolio dataframe as argument. You could refer to the *Evaluation metrics* section the mathematical formulae for each evaluation metric.

10.4.1 Portfolio return

```
function portfolio_return(portfolio)
fig = plt.figure() # Initialise the plot figure
ax1 = fig.add_subplot(111, ylabel="Total in \$") # Add a subplot and label for y-axis

plt.plot(portfolio.Date, portfolio.returns, color="blue", linewidth=1.0, label=
↪"Returns")

plt.title("MA crossover portfolio return")
plt.show()

# save fig
fig.savefig("./figures/moving-average-crossover_returns", dpi=100)
end
```

```
# call function
portfolio_return(portfolio)
```

10.4.2 Sharpe ratio

```
function sharpe_ratio(portfolio)
# Annualised Sharpe ratio
sharpe_ratio = sqrt(252) * (mean(returns) / std(returns))

return sharpe_ratio
end
```

```
# Call function and print output
sharpe = sharpe_ratio(portfolio)
@printf("Sharpe ratio: %f\n", sharpe)
```

10.4.3 Compound Annual Growth Rate (CAGR)

```
function CAGR(portfolio)
# Get the number of days in df
format = DateFormat("y-m-d")
days = portfolio.Date[size(portfolio,1)] - portfolio.Date[1]

# Calculate the CAGR
cagr = ^((portfolio.total[size(portfolio,1)] / portfolio.total[1]), (252.0 / Dates.
↪value(days))) - 1

return cagr
end
```

```
# Call function and print output
cagr = CAGR(portfolio)
@printf("CAGR: %f\n", cagr)
```

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 11

Bankruptcy prediction

In this tutorial, you will learn to:

- Train a machine learning model for bankruptcy prediction
- Carry out inferencing
- Screen out stocks based on the results

11.1 Intro to bankruptcy prediction

In this problem setting, we assume that the bankruptcy of a company could be predicted several years ahead. We presume that certain financial ratios are significant for the prediction of bankruptcy, and that we would compute these ratios using the fundamentals data we have collected and train a machine learning model to guess if a company would go bankrupt in one year based on these financial ratios.

11.2 Bankruptcy prediction with machine learning

The workflow of the prediction model is outlined as follows:

1. Collect labelled data (bankrupted = 0, survived = 1)
2. Split the dataset into training set and test set a ratio of 7:3
3. Train the machine learning model(s) with the labelled data (training set)
4. Evaluate the accuracy of the training model with the test set
5. Make use of the models to predict our own set of data (i.e. inferencing)

We will go through these steps one by one, and try to predict the bankruptcy of a set of companies in Hong Kong.

11.2.1 Data collection

The labelled dataset featured in the repository is collected by the UCLA School of Law. It contains a total of about 200 records, which includes the fundamentals data of public companies in US as well as its bankruptcy status (bankrupted = 0, survived = 1). Note that in the suffix of the dataset file names, `t1` means 1 year, `t2` means 2 years, and so on.

Based on the training data and other references[1], the below ratios have been selected as input variables for the machine learning model:

$$
\begin{aligned}
X_1 &= \text{Working capital} \ / \ \text{Total assets} \\
X_2 &= \text{Retained earnings} \ / \ \text{Total assets} \\
X_3 &= \text{EBIT} \ / \ \text{Total assets} \\
X_4 &= \text{Total equity (book)} \ / \ \text{Total assets} \\
X_5 &= \text{Net income} \ / \ \text{Total assets} \\
X_6 &= \text{Total liabilities} \ / \ \text{Total assets}
\end{aligned}
$$

The code snippet for concatenating the dataset and creating the variables as shown below:

```
# Concatenate data
data_full = pd.concat([bankrupt_data, non_bankrupt_data], ignore_index=True)

# Add and scale variables
data_full["X1"] = preprocessing.scale(data_full["WoCap"] / data_full["ToAsset"])
data_full["X2"] = preprocessing.scale(data_full["CFOper"] / data_full["ToLia"])
data_full["X3"] = preprocessing.scale(data_full["EBIT"] / data_full["ToAsset"])
data_full["X4"] = preprocessing.scale(data_full["ToEqui"] / data_full["ToAsset"])
data_full["X5"] = preprocessing.scale(data_full["NetInc"] / data_full["ToAsset"])
data_full["X6"] = preprocessing.scale(data_full["ToLia"] / data_full["ToAsset"])
```

11.2.2 Train-test split

The dataset is then split into training set and test set (with a ratio of 7:3), so that we could achieve unbiased evaluation (i.e. the model is evaluated with fresh data).

```
# Split data for training and testing
X = data_full[["X1", "X2", "X3", "X4","X5","X6"]]
y = data_full['Status']
self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, test_
↪size=0.3, random_state=101)
```

In the example, three machine learning models typically used for classification have been chosen:

1. Support Vector Machine
2. Decision Tree

[1] SAF2002, https://ja.wikipedia.org/wiki/SAF2002

3. Random Forest
4. K-Nearest Neighbor (KNN)

They are defined as methods of the bankruptcy prediction model class in the example file. Thus, it would be easier for us to compare and evaluate the performance of different machine learning models.

11.2.3 Training the model

```
# after loading and pre-processing the data

# create an instance of the class
model = bankrupt_prediction(bankrupt_t1, non_bankrupt_t1, input_df, False)

# train with knn
t1_results = model.knn()
```

11.2.4 Evaluate accuracy

We can check the performance of the model by looking into the confusion matrix and classification report:

```
# ouput:

  Confusion Matrix using Decision Tree:

  [[7 2]
  [2 9]]

  Classification Report using Decision Tree:

                precision    recall  f1-score   support

            0       0.78      0.78      0.78         9
            1       0.82      0.82      0.82        11

    accuracy                           0.80        20
   macro avg       0.80      0.80      0.80        20
weighted avg       0.80      0.80      0.80        20
```

11.2.5 Inferencing

To carry out prediction with our own dataset, simply set the boolean of `inferencing` to `True` and call the method again.

```
model.inferencing = True # set Inferencing as True

t1_results = model.knn()
```

(continues on next page)

(continued from previous page)

```
# save results to a csv file
t1_results.to_csv("./results/t1_results-knn.csv", header='column_names', index=False)
```

To check what are the companies that survived in 1 year, load the results csv file and filter out companies with a label of `1`.

```
df_t1 = pd.read_csv("./results/t1_results-knn.csv", index_col=None, header=0)
mask = (df_t1['knn'] == 1)

# get the percentage of survival
len(df_t1[mask]) / len(df_t1)
```

And we could save the filtered list as another csv file.

```
df_filtered = df_t1[mask]
df_filtered.to_csv("survived-t1.csv", header='column_names', index=False)
```

Sources

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 12

Property price prediction

In this tutorial, you will learn:

- The basics in macroeconomic analysis
- The ways of analyzing macroeconomic indicators
- The ways of analyzing real estate market data
- How to build a property price prediction model

12.1 Intro to macroeconomic analysis

As we have discussed in the first tutorial, macroeconomic analysis is a way of investigating the macroeconomic indicators that influence the stock market.

In this module, we will first analyze the macroeconomic indicators and explore how the indicators affect the stock prices in Hong Kong.

Then, we will specifically analyze the Hong Kong real estate market, as we believe that it is one of the most important macroeconomic indicator that can reflect the Hong Kong's economy.

In addition, we will build a property price prediction model to predict the house price of Hong Kong.

12.2 Macroeconomic indicators in Hong Kong

Before proceeding, first import some necessary libraries needed for this module.

```
import random
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

After importing the libraries, let's have a look at the data. The data contains 8 different macroeconomic indicators collected from 2016 to 2020.

Use `df.info()` to print information of all columns.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 11 columns):
year                          60 non-null int64
month                         60 non-null int64
hsi                           60 non-null float64
house_price                   60 non-null int64
population                    54 non-null float64
unemployment_adjusted         60 non-null float64
unemployment_not_adjusted     60 non-null float64
imports                       60 non-null int64
total_exports                 60 non-null int64
gdp                           57 non-null float64
ccp_index                     60 non-null float64
dtypes: float64(6), int64(5)
memory usage: 5.3 KB
```

Fig. 1: The column information of macroeconomic data.

12.2.1 Univariate analysis

In univariate analysis, use `pandas.Dataframe.describe()` to examine the distribution of the numerical features. It returns the statistical summary such as mean, standard deviation, min, and max of a data frame.

For a better understanding of the statistics summary, use `seaborn.distplot()` to visualise the results with histograms.

```
# Statistical summary
print(df[feature_name].describe())

# Histogram
```

(continues on next page)

(continued from previous page)

```
plt.figure(figsize=(8,4))
sns.distplot(df[feature_name], axlabel=feature_name);
```

12.2.2 Bivariate analysis

In bivariate analysis, we are going to study the correlations between a macroeconomic indicator and the Hang Seng Index. Use `matplotlib.pyplot.scatter()` and `seaborn.regplot()` to visualize the relationship between two features.

```
x = df[feature_name]
y = df['hsi']

plt.scatter(x, y)
plt.xticks(rotation=45)
fig = sns.regplot(x=feature_name, y="hsi", data=df)
```

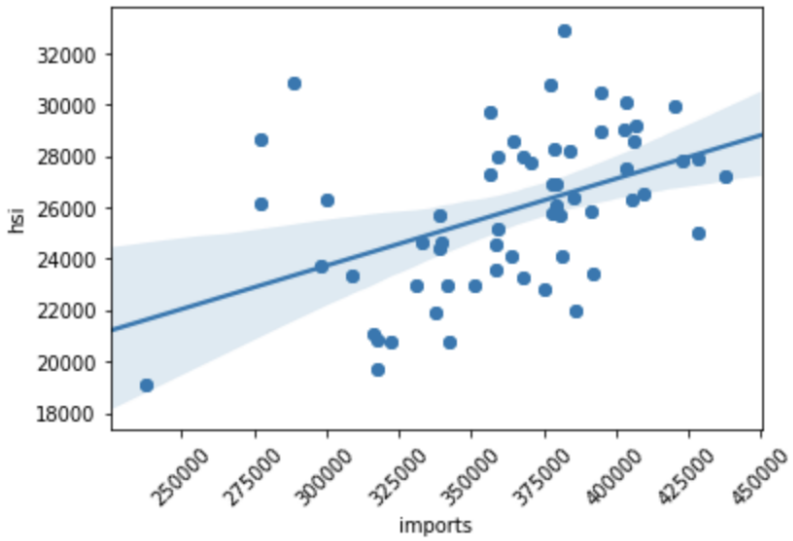


Fig. 2: An example of a scatter plot with a regression line.

Then, use `pandas.Dataframe.corr()` and `seaborn.heatmap()` to compute a pairwise correlation of features and visualize the correlation matrix.

```
fig, ax = plt.subplots(figsize=(10,10))
cols = df.corr().sort_values('hsi', ascending=False).index
cm = np.corrcoef(df[cols].values.T)
hm = sns.heatmap(cm, annot=True, square=True, annot_kws={'size':11}, yticklabels=cols.
↪values, xticklabels=cols.values)
plt.show()
```

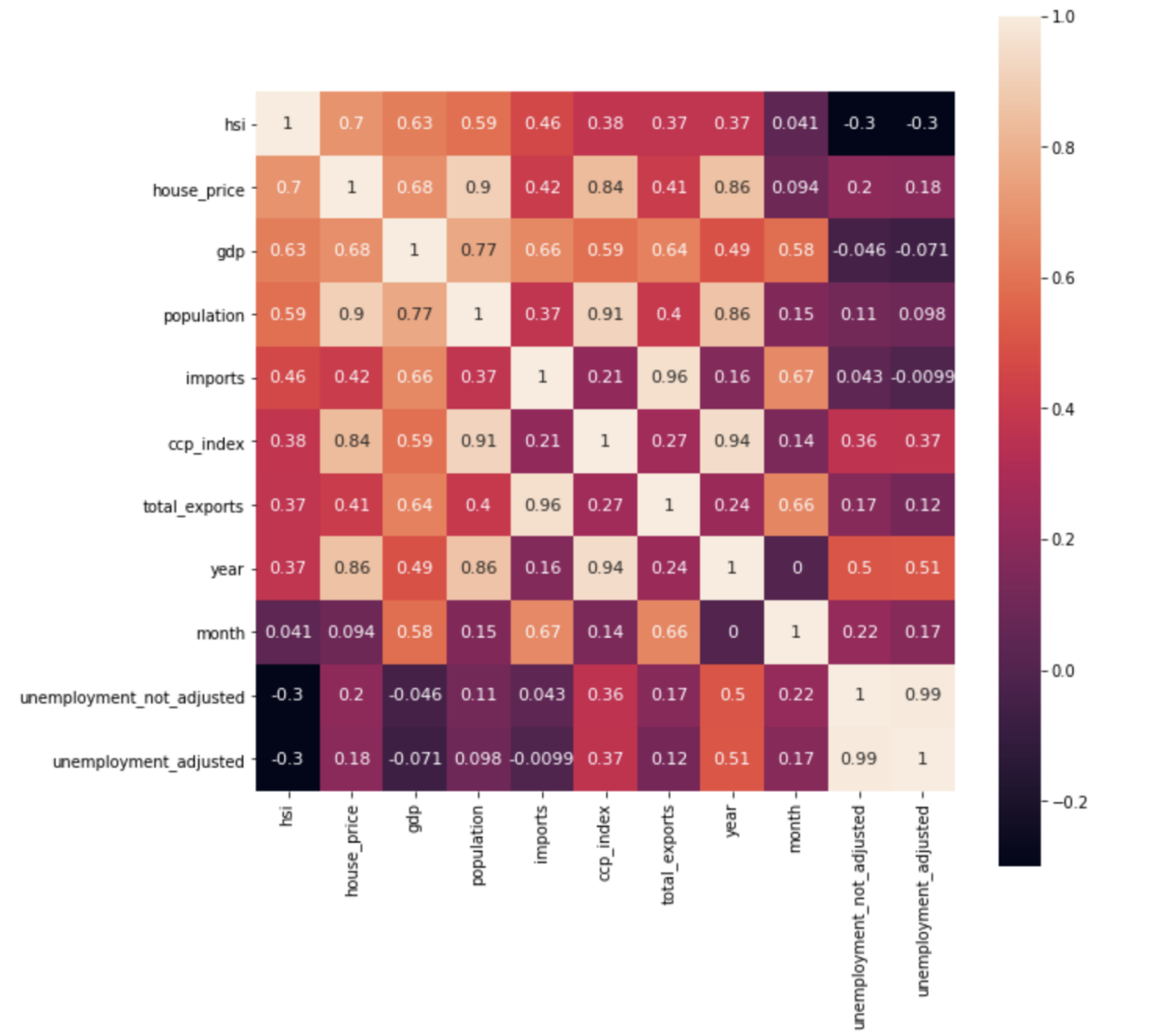


Fig. 3: Heatmap - macroeconomic indicators of Hong Kong.

According to the above figure, we can see that house price, GDP, population, imports, composite consumer price index , total exports, and year are positively correlated to the Hang Seng index, while both seasonally adjusted unemployment rate and not seasonally adjusted unemployment rate are negatively correlated to the Hang Seng index.

12.3 The Hong Kong real estate market

As shown above, the house price in Hong Kong has a strong positive correlation with the Hang Seng Index. In fact, the properties and construction sector accounts for over 10% of weighting in the Hang Seng Index (Hang Seng Indexes Company Limited, 2020), and thus the real estate market data is a source of volatility in the Hong Kong stock market.

While Hong Kong's real estate market is a constant topic of discussion, it will be worth analyzing the Hong Kong real estate market data. Using the same data analysis technique used for the above analysis, we will now analyze Hong Kong residential market transaction records.

12.3.1 Data pre-processing

Before analyzing the transaction records:

1. Derive some useful features from existing features.

```
# Add new features
df['month'] = pd.to_datetime(df['RegDate']).dt.month
df['year'] = pd.to_datetime(df['RegDate']).dt.year
```

2. Drop unmeaningful features and features with too many missing values

```
# Drop unnecessary columns
df = df.drop([feature_name], axis=1)
```

3. Handle missing values by replacing NAN with a mean value of a feature

```
# Handling missinig values
# Fill with mean
feature_name_mean = df[feature_name].mean()
df[feature_name] = df[feature_name].fillna(feature_name_mean)
```

4. Label encode categorical features

```
le = LabelEncoder()
le.fit(list(processed_df[feature_name].values))
processed_df[feature_name] = le.transform(list(processed_df[feature_name].values))
```

12.3.2 Economic indicator analysis

In economic indicator analysis, we will explore how the macroeconomic indicators affect the monthly average house price per saleable area in Hong Kong.

The transaction records from Centaline Property will be used for this analysis.

address	buildingAge	regDate	price	saleableArea	grossArea	upSaleableArea	upGrossArea	lasthold	gain
FLAT 7 26/F BLOCK A SMITHFIELD TERRACE	34	2020-12-23	5300000.0	262	357	20229	14846	3394	89
FLAT H 41/F THE FOREST HILLS	12	2020-12-23	7380000.0	439	627	16811	11770	2793	50
FLAT 5 (NO. 26) 1/F MAN YUE MANSION (BUILDING)	56	2020-12-23	3980000.0	480	-	8292	-	-	NaN
FLAT 12 30/F CHUN HONG HOUSE (BLOCK E) TIN MA ...	34	2020-12-23	4800000.0	-	461	-	10412	-	NaN
NO. 3 4/F GOLDEN PHOENIX BUILDING	55	2020-12-23	3350000.0	381	-	8793	-	7718	329

Fig. 4: The data structure of transaction record (Centaline Property).

Before analyzing the data, calculate the monthly average house price per saleable area. Then, join the data with economic indicators by year and month.

```
# calculate the monthly average house price
df = df.groupby(['year','month'],as_index=False).mean()
df = df.rename(columns={'UnitPricePerSaleableArea': 'AveragePricePerSaleableArea'})
```

Using the bivariate analysis method we learned, a pairwise correlation of features is computed and visualized. The result shows that population, year, composite consumer price index, GDP, imports, and total exports are positively correlated to the monthly average house price per saleable area in Hong Kong, while both unemployment rates are negatively correlated to the monthly average house price per saleable area in Hong Kong.

12.3.3 Transaction record analysis

In transaction record analysis, we will examine the relationship between features describing the house and the individual housing prices of Hong Kong.

The transaction records from Midland Realty will be used for this analysis.

In univariate analysis, the distribution of Hong Kong's house price is examined. The housing price of Hong Kong has a mean of 9 million HKD and a standard deviation of 13 million HKD. The skewness and kurtosis were 26.9 and 1526.4 respectively, showing that the housing price of Hong Kong is skewed positively to a very high degree.

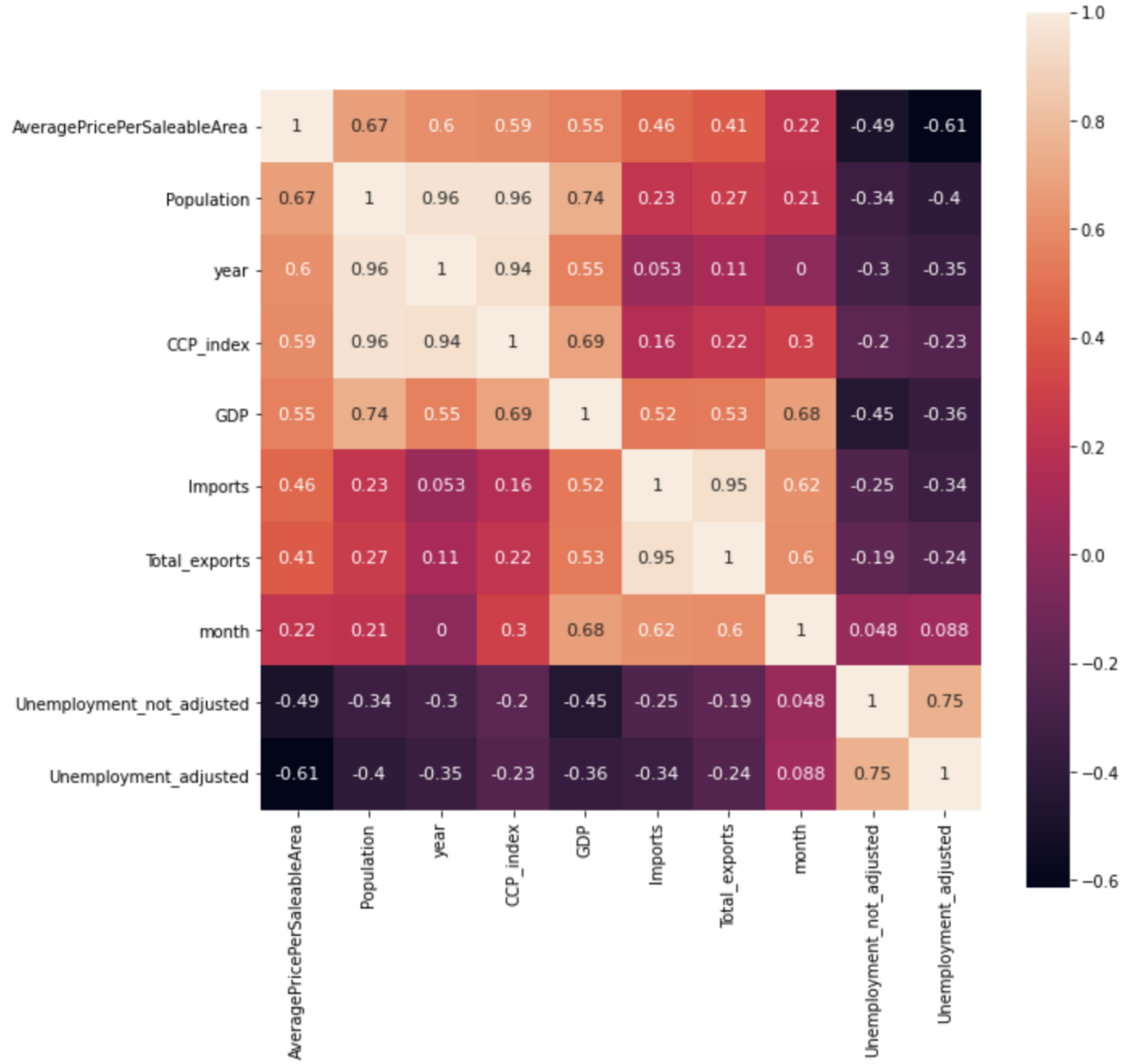


Fig. 5: Heatmap - economic indicators analysis.

region	subregion	district	estate	building	firstOpDate	floorL	bedroom	sittingroom	floor
Hong Kong Island	Eastern	Heng Fa Chuen (Chai Wan)	Heng Fa Chuen	Block 31	1988-03-19	L	2.0	NaN	NaN
New Territories	Sha Tin	Shatin	City One Shatin	Block 31	1983-08-13	M	2.0	1.0	NaN
Kowloon	Kowloon City	Hung Hom	Royal Peninsula	Block 1	2000-12-14	H	3.0	NaN	NaN
New Territories	Kwai Tsing	Tsing Yi	Cheung Fat Estate	Block 1 (King Fat House)	1989-09-01	H	NaN	NaN	NaN
New Territories	Yuen Long	Yuen Long	Grand Del Sol	Block 01	1997-12-05	H	3.0	NaN	NaN

Fig. 6: The data structure of transaction record (Midland Realty) - Part 1.

flat	grossArea	saleableArea	price	regDate	lastRegDate	lastPrice	gain	lat	lon
6	597.0	498.0	8250000.0	2021-01-07	2003-01-17	1700000.0	385.29	22.278636	114.239373
D	395.0	327.0	5400000.0	2021-01-07	NaT	NaN	0.00	22.386003	114.204891
B	1251.0	952.0	21000000.0	2021-01-07	2008-01-18	9680000.0	116.94	22.304435	114.184404
21	629.0	485.0	3950000.0	2021-01-07	NaT	NaN	0.00	22.361853	114.103990
E	883.0	740.0	7830000.0	2021-01-07	1997-08-06	4953900.0	58.06	22.440296	114.034537

Fig. 7: The data structure of transaction record (Midland Realty) - Part 2.

```
# Distribution
print(df['price'].describe())

# Skewness and kurtosis
print("Skewness: ", df['price'].skew())
print("Kurtosis: ", df['price'].kurt())
```

```
#output:

count    1.664090e+05
mean     9.133268e+06
std      1.310856e+07
min      5.500000e+05
25%      5.200000e+06
50%      6.830000e+06
75%      9.500000e+06
max      1.399000e+09
Name: price, dtype: float64

Skewness:  26.927207752922435
Kurtosis:  1526.4066673335874
```

In order to get a better result for the bivariate analysis, outliers are removed by using standard deviation.

```
# Calculate mean and standard deviation
data_mean, data_std = np.mean(df[feature_name]), np.std(df[feature_name])

# Calculate upper boundary
upper = data_mean + data_std * 3

# Remove outliers
df = df[df[feature_name] < upper]
```

In bivariate analysis, the correlation coefficient between the features describing the house and the house price is computed. 7 features with the highest correlation is selected and shown below.

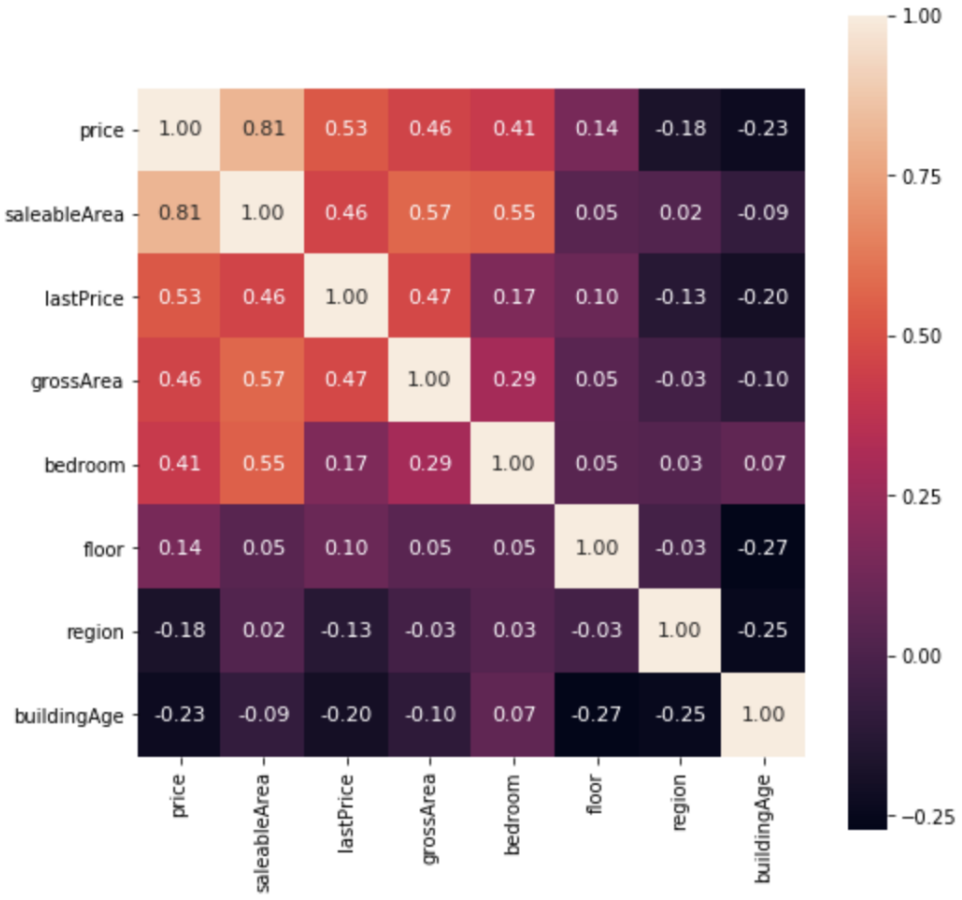


Fig. 8: Heatmap - transaction data analysis.

According to the above figure, the housing price in Hong Kong has (1) a strong positive correlation with saleable area; (2) a moderate positive correlation with last transaction price; (3) a moderate positive correlation with gross area; (4) a moderate positive correlation with number of bedrooms; (5) a weak positive correlation with floor; (6) a weak negative correlation with region; and (7) a weak negative correlation with building age.

The full implementation of the economic indicator analysis and transaction data analysis could be found in `code/macroeconomic-analysis/` in the repository.

12.4 Property price prediction with machine learning

Based on the transaction data analysis, let's build property price prediction models.

12.4.1 Train-test split

Use `sklearn.model_selection.train_test_split()` to split the data with the ratio of 8:2. The input variables are the top 7 features selected from the analysis, and the output feature is the house price.

```
feat_col = [ c for c in df.columns if c not in ['price'] ]
x_df, y_df = df[feat_col], df['price']

x_train, x_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_
↪state=RAND_SEED)
```

12.4.2 Log transformation

Before training the model, transform `y_train` using log function to normalise the highly skewed price data. In this way, the dynamic range of Hong Kong's property price can be reduced.

```
log_y_train= np.log1p(y_train)
```

12.4.3 Training the model

In total, 4 different types of predictive models will be built:

1. XGBoost
2. Lasso
3. Random Forest
4. Linear Regression

Train the models with `x_train` and `y_train`, and use the models to make the predictions.

```
import xgboost as xgb

# XGBoost
model_xgb = xgb.XGBRegressor(objective ='reg:squarederror',
                     learning_rate = 0.1, max_depth = 5, alpha = 10,
                     random_state=RAND_SEED, n_estimators = 1000)
model_xgb.fit(x_train, log_y_train)
xgb_train_pred = np.expm1(model_xgb.predict(x_train))
xgb_test_pred = np.expm1(model_xgb.predict(x_test))
```

12.4.4 Evaluate accuracy

Then, evaluate the performance of each model by root mean square log error (RMSLE). The reason why RMSLE is used is because the price values are too big, and RMSLE prevents penalising large differences between actual and predicted prices.

```
from sklearn.metrics import mean_squared_log_error

def rmsle(y, y_pred):
  return np.sqrt(mean_squared_log_error(y, y_pred))
```

```
#output:

    XGBoost RMSLE(train):  0.1626671056150446
    XGBoost RMSLE(test):  0.16849945199484243
```

The train model RMSLE and the test model RMSLE are 0.1627 and 0.1685 respectively. XGBoost uses a more accurate implementation of gradient boosting algorithm and optimised regularisation, and hence, it gives a better result than other models.

However, in this case, the result shows that the model is slightly overfitting the train data. The below figure shows the graph of actual and predicted property price for XGBoost.

```
plt.figure(figsize=(5,5))
plt.scatter(y_test,xgb_test_pred)
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.show()
```

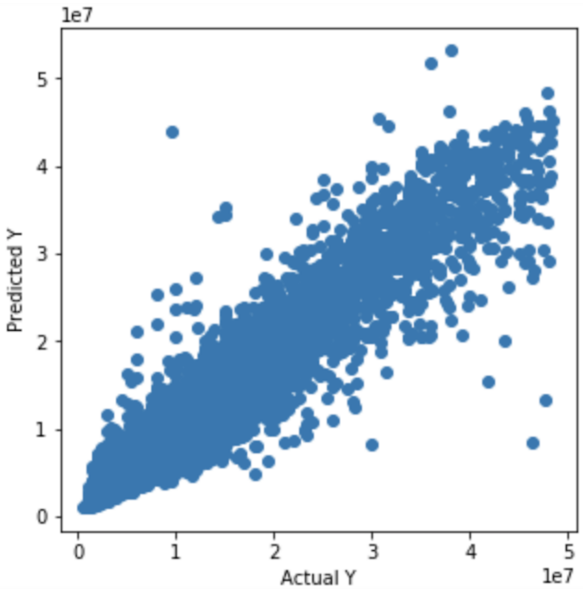
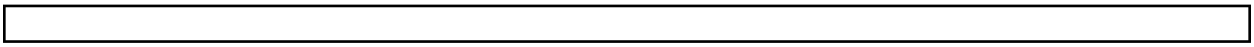


Fig. 9: The graph of actual and predicted house price for XGBoost.

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.



CHAPTER 13

Sentiment analysis

In this tutorial, you will learn:

- The basics of sentiment analysis
- How to collect tweets
- How to collect financial news headlines
- What are the common ways of analysing sentiment
- How to measure the accuracy of the sentiment prediction

13.1 Intro to sentiment analysis

As we have discussed in the Introduction part, sentiment analysis is a natural language processing technique that is used to determine whether a statement contains positive, negative or neutral sentiment. In this tutorial, we aim to analyse the daily sentiment of a stock with the use of relevant news headlines and tweets, and thus to find out the market sentiment.

13.2 Collection of tweets

Apply for developer account from Twitter use Tweepy

1. Click and apply for a developer account through this link: https://developer.twitter.com/en/apply-for-access
2. Create a new project and connect it with the developer App in the developer portal
3. Enable App permissions (*Read* and *Write*)
4. Navigate to the **'Keys and token'** page, save your API key, API secret, Access token and Access secret

Code example

```
import tweepy

# do not share the API key in any public platform (e.g github, public website)
consumer_key = API secret
consumer_secret = API secret
access_token = Access token
access_token_secret = Access secret


# authorisation of consumer key and consumer secret
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
```

Access the relevant tweets using the Twitter API

There are different types of API provided by Twitter with various limitations. Please visit this link for further information: https://developer.twitter.com/en/docs/twitter-api .In the following section, you will learn how to retrieve tweets from the Twitter timeline, hashtag/cashtag and also stream data that contains real time tweets.

13.2.1 Timeline tweets

Returns the 20 most recent tweets posted from the authenticated user. It is also possible to request another user's timeline via the the `id` parameter.

Pass the `user_id` or `screen_name` parameters to access the user-specified tweets. For more information regarding the parameters, please visit the official documentation: https://docs.tweepy.org/en/v3.5.0/api.html

Code example

```
# create an empty list
alltweets = []

# extract data from the API
timeline = api.user_timeline(user_id=userid, count=number_of_tweets)
alltweets.extend(timeline)

with open('%s_tweets.csv' % screen_name, 'a') as f:
    writer = csv.writer(f)
    for tweet in alltweets:
         tweet_text = tweet.text.encode("utf-8")
         dates=tweet.created_at
         writer.writerow([dates,tweet_text])
```

13.2.2 Hashtag/Cashtag tweets

Cashtag is a feature on Twitter that allows users retrieve tweets relevant to a particular ticker, say $GOOG, $AAPL or $FB. Use `tweepy.Cursor()` to access data from hashtag and cashtags.

Code example

```
# extract data from the API
hashtags = tweepy.Cursor(api.search, q=name, lang='en', tweet_mode='extended').
↪items(200)

with open('%s_tweets.csv' % screen_name, 'a') as f:
    writer = csv.writer(f)
    for status in hashtags:
         tweet_text = status.full_text
         dates = str(status.created_at)[:10]
         writer.writerow([dates,tweet_text])
```

If you want to collect tweets for a period of time, we could further amend the code snippet in the following way:

```
with open('%s_tweets.csv' % screen_name, 'a') as f:
    writer = csv.writer(f)
    for status in hashtags:
        # Add this line
        ** if (datetime.datetime.now() - status.created_at).days <= day_required: **
             tweet_text = status.full_text
             dates = str(status.created_at)[:10]
             writer.writerow([dates,tweet_text])
```

13.2.3 Stream tweets

The Twitter streaming API is used to download the tweets in real time. It is useful for obtaining a high volume of tweets, or for creating a live feed using a site stream. For more information with the API, please visit this link: https://docs.tweepy.org/en/v3.5.0/streaming_how_to.html.

1. Create a class inheriting from StreamListener

```
# override tweepy.StreamListener
class MyStreamListener(tweepy.StreamListener):
    # add logic to the on_staus method
    def on_status(self, status):
        if (self.tweet_count == self.max_tweets):
            return False
        # collect tweets
        else:
            tweet_text = status.text
            writer = csv.writer(self.output_file)
            writer.writerow([status.created_at,status.extended_tweet['full_text'].
↪encode("utf-8")])
            self.tweet_count += 1

    # add logic to the initialisation function
    def __init__(self, output_file=sys.stdout,input_name=sys.stdout):
        super(MyStreamListener,self).__init__()
        self.max_tweets = 200
        self.tweet_count = 100
        self.input_name = input_name
```

2. Create a stream

```
# add an output_file parameter to store the output tweets
myStreamListener = MyStreamListener(output_file=f, input_name=firm)
myStream = tweepy.Stream(auth=api.auth, tweet_mode='extended',␣
↪listener=myStreamListener, languages=["en"])
```

3. Start a stream

```
myStream.filter(track=target_firm)
```

13.3 Collect financial headlines

13.3.1 US news headlines

Finviz.com is a browser-based stock market research platform that allows visitors to read the latest financial news collected from different major newsagents such as Yahoo! Finance, Accesswire, and Newsfile.

Before the tutorial, it is important to take a look at the front-end code of the website.

1. Access the website of each ticker through the `urllib.request` module

```
allnews = []
finviz_url = 'https://finviz.com/quote.ashx?t='
url = finviz_url + ticker
req = Request(url=url, headers={'user-agent': 'my-app/0.0.1'})
```

2. Get the HTML document using Beautiful Soup

```
html = BeautifulSoup(resp, features="lxml")
```

3. Get the information of id='news-table' in the website

```
news_table = html.find(id='news-table')
news_tables[ticker] = news_table
```

4. Find all the news under the <tr> tag in the news-table

```
for info in df.findAll('tr'):
    text = info.a.get_text()
    date_scrape = info.td.text.split()
    if (len(date_scrape) == 1):
        time = date_scrape[0]
    else:
        date = date_scrape[0]
        time = date_scrape[1]
        news_time_st r= date + " " + time
```

5. Convert the date format to 'YYYY-MM-dd'

```
date_time_obj = datetime.datetime.strptime(news_time_str, '%b-%d-%y %I:%M%p')
date_time=date_time_obj.strftime('%Y-%m-%d')
```

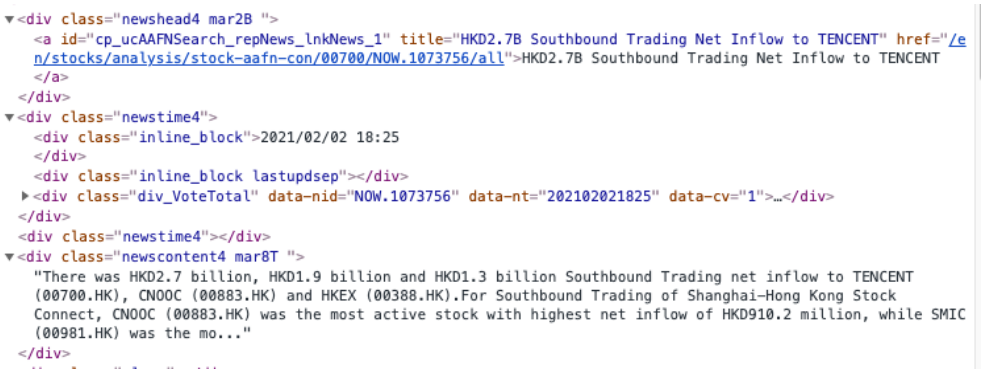
6. Append all the news together

```
allnews.append([date_time,text])
```

13.3.2 HK news headlines

We will also learn how to collect news headlines from aastock.com. The website one of the most popular financial information platforms in Hong Kong. It offers real-time international information relevant to Hong Kong shares, which are useful for analysing sentiment and trends in the local market.

Again, before writing code to scrape the news, we need to have a look of the front-end code of the website. Take tencent (00700.HK) as an example. Click 'inspect' and you can view the front-end code of the website. (Or visit this link: http://www.aastocks.com/en/stocks/analysis/stock-aafn/00700/0/all/1)



From the above snippet, we could know that the `date'` attribute is stored within the `<div class ='inline_block>` under the `<div class='newstime 4'>`, while the news headlines are stored within the `<div class='newscontent4 mar8T'>`.

The following steps are similar to that for collecting US news headlines.

1. Access the website of each ticker through `urllib.request` module

```
prefix_url = 'http://www.aastocks.com/en/stocks/analysis/stock-aafn/'
postfix_url = '/0/all/1'
url = prefix_url + fill_ticker + postfix_url
req = Request(url=url, headers={'user-agent': 'my-app/0.0.1'})
resp = urlopen(req)
```

2. Get the HTML document using Beautiful Soup

```
html = BeautifulSoup(resp, features="lxml")

# get the html code containing the dates and news
dates = html.findAll("div", {"class": "inline_block"})
news = html.findAll("div", {"class": "newshead4"})
```

3. Find all the news and corresponding dates from the html code from step 2

```
# track the index in the news list
idx = 0

with open('%s_tweets.csv' % screen_name, 'a') as f:
    writer = csv.writer(f)
    for i in dates:
        # as the dates are in yyyy/mm/dd format
        if "/" in str(i.get_text()):
            date = str(i.get_text())
            # the front-end code is not standardised and sometimes contains 'Release␣
↪Time' string
            if "Release Time" in date:
                date = date[13:23]
        else:
            date = str(date[:10])
            text = news[idx].get_text()
            date_time_obj = datetime.datetime.strptime(date, '%Y/%m/%d')

            # standardise the date format as 'YYYY-mm-dd'
            date_time = date_time_obj.strftime('%Y-%m-%d')

            # set the number of days you want to collect
            if (datetime.datetime.now()-date_time_obj).days <= day_required:
                writer.writerow([date_time,text])
                idx += 1
```

13.4 VADER sentiment prediction

After the collection of data, it is time for you to now carry out the analysis with the database.

VADER (Valence Aware Dictionary for Sentiment Reasoning) is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is available in the NLTK package and can be applied directly to unlabelled text data.

The sentiment labels are generated from the VADER Compound score according to the following rules:

- Positive sentiment (= 2): compound score > 0.01
- Neutral sentiment (= 1): 0.01 compound score 0.01
- Negative sentiment (= 0): compound score < 0.01

Note that 1% was set as the threshold value accounting for the average stock movement in the US market, feel free to set any value for your own analysis

1. Import these libraries

```
import pandas as pd
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import twitter_samples
```

2. VADER's `SentimentIntensityAnalyzer()` takes in a string and returns a dictionary of scores in each of four categories:

- negative
- neutral
- positive
- compound (computed by normalising the scores above, ranging from -1 to 1)

Let us analyse the data that we have collected through the sentimental analyser.

```
# pass in the path where you stored the csv file containing the data
def read_tweets_us_path(path):
    # could change to your own path
    path = os.path.join(dir_name,'train-data/'+path)
    # read in data as pandas dataframe
    df = pd.read_csv(path)
    cs = []

    for row in range(len(df)):
        cs.append(analyzer.polarity_scores(df['tweets'].iloc[row])['compound'])

    # create a new column for the calculated results
    df['compound_vader_score'] = cs
    print(df)

    return df
```

3. Label the sentiment for each tweet

Parameters:

- `grouped_data`: consolidated data with features including (dates, tweets, compound_vader_score)
- `file_name`: the output name after the label function
- `perc_change`: the threshold value for labelling the sentiment

Code example

```
def find_tweets_pred_label(grouped_data,file_name,perc_change):
    print('find_pred_label')
    tweets = grouped_data['tweets']

# group the tweets within the csv using ['dates','ticker'] index,
grouped_data = grouped_data.groupby(['dates','ticker'])['compound_vader_score'].
↪mean().reset_index()

final_label = []

for i in range(len(grouped_data)):
    if grouped_data['compound_vader_score'].iloc[i] > perc_change:
        final_label.append(2)
    elif grouped_data['compound_vader_score'].iloc[i] < -perc_change:
        final_label.append(0)
    elif ((grouped_data['compound_vader_score'].iloc[i] >= -perc_change) and (grouped_
↪data['compound_vader_score'].iloc[i] <= perc_change)):
        final_label.append(1)

# add the column of vader_label
grouped_data['vader_label'] = final_label
grouped_data['tweets'] = tweets

grouped_data.to_csv(file_name)
```

4. Merge all the data together

- actual label (= 2): price movement 0.01
- actual label (= 1): 0.01 price movement 0.01
- actual label (= 0): price movement 0.01

Parameters:

- `file_name`: consolidated data with features including (dates,tweets,compound_vader_score)
- `label_data`: the label data contains the actual label from yahoo finance

Code example

```
def merge_actual_label (file_name,label_data):

    vader_data = pd.read_csv(file_name)
    vader_data.set_index(keys=["dates","ticker"], inplace=True)

    label_data = pd.read_csv(label_data)
    label_data.set_index(keys=["dates","ticker"], inplace=True)

    # merge the actual label and the predicted label into a single pandas data frame
    merge = pd.merge(vader_data,label_data, how='inner', left_index=True, right_
↪index=True)
    merge.drop(columns=['Unnamed: 0_y'], axis=1)

    return merge
```

5. Validation using confusion matrix

Parameters:

- `df`: the final merged pandas dataframe
- `name`: the output csv file containing all the merged information with dates, tweets, vader label and actual label

Code illustration

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

def validation(df,name):
    pred_label = list(df['vader_label'])
    actual_label = list(df['label'])
    labels = [0,1,2]

    cm = confusion_matrix(actual_label, pred_label,labels)
    labels = ['True Neg','False Pos','False Neg','True Pos']
    categories = ['Negative','Neutral', 'Positive']

    make_confusion_matrix(cm, group_names=labels, categories=categories)

    df.to_csv(name)
```

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 14

Integrated trading strategy

In this tutorial, you will learn:

- How to make use of different features to write a strategy
- How to use machine learning models to predict trading signals

14.1 Putting it all together

In this part we'll look into how to put everything together in order to build a strategy that incorporates different features (technical indicator, macroeconomic trends and sentiment indicator) so as to comprehensively examine the market as a whole.

14.2 Simple approach

We'll first look into simplistic, rule-based methods to make use of the features in an algorithmic trading pipeline.

14.2.1 Baseline model

We have created the *Baseline model* example to demonstrate how we could generate signals using a technical analysis strategy, then filter it by macroeconomic data and sentiment labels. The implementation of the baseline model could be found in `integrated-strategy/baseline.py`.

The workflow of the baseline model is as follows:

- Generate the signals dataframe using a technical analysis strategy (e.g. MACD)
- Pass the signals to the macroeconomic filter

- Pass the signals to the sentiment filter
- Backtest with the filtered signals dataframe

As usual, we first select the ticker and time range to run the strategy:

```
# load price data
df_whole = pd.read_csv('../../database/microeconomic_data/hkex_ticks_day/hkex_0001.csv
↪', header=0, index_col='Date', parse_dates=True)

ticker = "0005.HK"

# select time range (for trading)
start_date = pd.Timestamp('2017-01-01')
end_date = pd.Timestamp('2021-01-01')

df = df_whole.loc[start_date:end_date]
```

Note that we'll also need this `filered_df` additionally to calculate the stock price's sensitivity to economic indicators.

```
# get filtered df for macro analysis
filtered_df = df_whole.loc[:end_date]
```

In the example, we choose to apply the MACD crossover strategy.

```
# apply MACD crossover strategy
macd_cross = macdCrossover(df)
macd_fig = macd_cross.plot_MACD()
plt.close() # hide figure

# get signals dataframe
signals = macd_cross.gen_signals()
print(signals.head())
signal_fig = macd_cross.plot_signals(signals)
plt.close()  # hide figure
```

After obtaining the signals dataframe by running the technical analysis strategy, we'll pass it to the macroeconomic filters and sentiment filter to eliminate signals that are contradicatory with the economic indicator or sentiment labels.

In this code snippet, we first apply the macroeconomic filter:

```
# get ticker's sensitivity to macro data
s_gdp, s_unemploy, s_property = GetSensitivity(filtered_df)

# append signals with macro data
signals = GetMacrodata(signals)
```

(continues on next page)

(continued from previous page)

```
# calculate adjusting factor
signals['macro_factor'] = s_gdp * signals['GDP'] + s_unemploy * signals['Unemployment␣
↪rate'] + s_property * signals['Property price']
signals['signal'] = signals['signal'] + signals['macro_factor']

# round off signals['signal'] to the nearest integer
signals['signal'] = signals['signal'].round(0)
```

We then apply the sentiment filter:

```
filtered_signals = SentimentFilter(ticker, signals)
```

With this filtered signals dataframe, we could pass it directly to the backtesting function in order to evaluate the portfolio performance.

```
portfolio, backtest_fig = Backtest(ticker, filtered_signals, df)
plt.close() # hide figure

# print stats
print("Final total value: {value:.4f} ".format(value=portfolio['total'][-1]))
print("Total return: {value:.4f}%".format(value=(((portfolio['total'][-1] - portfolio[
↪'total'][0])/portfolio['total'][-1]) * 100))) # for analysis
print("No. of trade: {value}".format(value=len(signals[signals.positions == 1])))
```

We could also make use of the evaluation metric functions:

```
# Evaluate strategy

# 1. Portfolio return
returns_fig = PortfolioReturn(portfolio)
returns_fig.suptitle('Baseline - Portfolio return')
#returns_fig.savefig('./figures/baseline_portfolo-return')
plt.show()

# 2. Sharpe ratio
sharpe_ratio = SharpeRatio(portfolio)
print("Sharpe ratio: {ratio:.4f} ".format(ratio = sharpe_ratio))

# 3. Maximum drawdown
maxDrawdown_fig, max_daily_drawdown, daily_drawdown = MaxDrawdown(df)
maxDrawdown_fig.suptitle('Baseline - Maximum drawdown', fontsize=14)
#maxDrawdown_fig.savefig('./figures/baseline_maximum-drawdown')
plt.show()

# 4. Compound Annual Growth Rate
```

(continues on next page)

(continued from previous page)

```
cagr = CAGR(portfolio)
print("CAGR: {cagr:.4f} ".format(cagr = cagr))
```

14.3 Machine learning approach

Moving on, we'll look into more advanced methods that pass the data features into a machine learning model in order to make sequential predictions.

14.3.1 Recurrent Neural Networks

A **recurrent neural network (RNN)** is a type of artificial neural network designed for sequential data or time series data. As opposed to traditional feedforward neural networks, they are networks with loops in them which allow information to persist. It has a lot of applications, ranging from language modelling to speech recognition. You can read more about the it in Andrej Karpathy's blog post - The Unreasonable Effectiveness of Recurrent Neural Networks.

However, one problem with RNNs is that they have a hard time in capturing long-term dependencies. When making a prediction at the current time step, RNNs would weigh more recent historical information to be more important. But sometimes contextual information could lie in the far past. This is When LSTM comes into play.

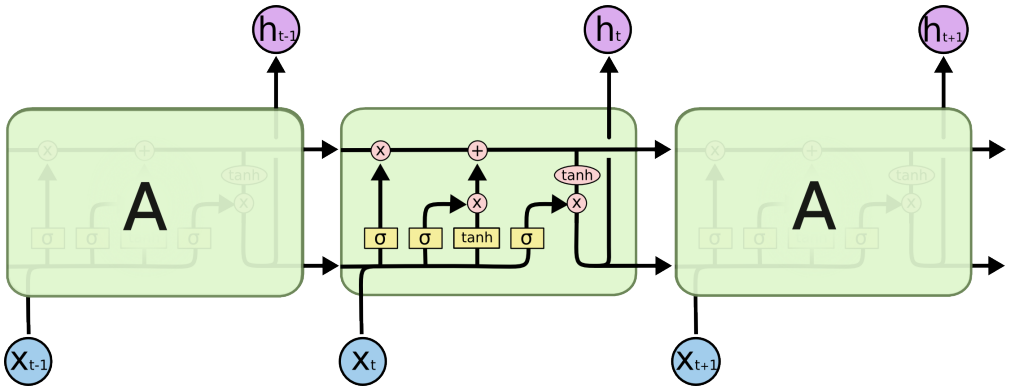


Fig. 1: The internal structure of an LSTM.[1]

Long Short Term Memory networks (LSTMs) is an improved version of RNN that is specifically designed to avoid the long-term dependency problem. Their default behaviour is to remember information for long periods of time.

If you want to know more about the mechanism and details of LSTMs, you could read this great blog post - Understanding LSTM Networks.

[1] By Christopher Olah - Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs

14.3.2 Single-feature LSTM model

The implememtation of the single-feature LSTM model could be found in `integrated-strategy/LSTM-train_price-only.py`.

We'll make use of the PyTorch library to build the LSTM model. You could install the library from here: https://pytorch.org/.

We first load the data for training and testing respectively.

```
data_dir = "../../database/microeconomic_data/hkex_ticks_day/"

# select date range
dates = pd.date_range('2010-01-02','2016-12-31',freq='B')
test_dates = pd.date_range('2017-01-03','2020-09-30',freq='B')

# select ticker
symbol = "0001"

# load data
df = read_data(data_dir, symbol, dates)
df_test = read_data(data_dir, symbol, test_dates)
```

The `MinMaxScaler` function from the `sklearn.preprocessing` is used to normalise the input features, i.e. they will be transformed into the range [-1,1] in the following code snippet.

```
scaler = MinMaxScaler(feature_range=(-1, 1))

df['Close'] = scaler.fit_transform(df['Close'].values.reshape(-1,1))
df_test['Close'] = scaler.fit_transform(df_test['Close'].values.reshape(-1,1))

look_back = 60 # choose sequence length
```

We can check the shapes of the train and test data:

```
x_train, y_train, x_test, y_test = load_data(df, look_back)
print('x_train.shape = ',x_train.shape)
print('y_train.shape = ',y_train.shape)
print('x_test.shape = ',x_test.shape)
print('y_test.shape = ',y_test.shape)
```

And then make the traing and testing sets in torch:

```
# make training and test sets in torch
x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test).type(torch.Tensor)
y_train = torch.from_numpy(y_train).type(torch.Tensor)
y_test = torch.from_numpy(y_test).type(torch.Tensor)
```

Moving on, let's set the hyperparameters.

```
# Hyperparameters
n_steps = look_back - 1
batch_size = 32
num_epochs = 100
input_dim = 1
hidden_dim = 32
num_layers = 2
output_dim = 1
torch.manual_seed(1) # set seed
```

We'll use mean squared error (MSE) as the loss function, and use Adam as the optimiser with a learning rate of 0.01.

```
train = torch.utils.data.TensorDataset(x_train,y_train)
test = torch.utils.data.TensorDataset(x_test,y_test)

train_loader = torch.utils.data.DataLoader(dataset=train,
                                           batch_size=batch_size,
                                           shuffle=False)

test_loader = torch.utils.data.DataLoader(dataset=test,
                                           batch_size=batch_size,
                                           shuffle=False)



model = LSTM(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim, num_
↪layers=num_layers)

loss_fn = torch.nn.MSELoss()
optimiser = torch.optim.Adam(model.parameters(), lr=0.01)
```

We'll write the training loop now:

```
# Initialise a list to store the losses
hist = np.zeros(num_epochs)

# Number of steps to unroll
seq_dim = look_back - 1

# Train model
for t in range(num_epochs):
    # Forward pass
    y_train_pred = model(x_train)
    loss = loss_fn(y_train_pred, y_train)

    if t % 10 == 0 and t !=0:
        print("Epoch ", t, "MSE: ", loss.item())

    hist[t] = loss.item()

    # Zero out gradient, else they will accumulate between epochs
    optimiser.zero_grad()

    # Backward pass
```

(continues on next page)

(continued from previous page)

```
    loss.backward()

    # Update parameters
    optimiser.step()
```

We could now make predictions on the test set to get the MSE:

```
# Make predictions
y_test_pred = model(x_test)

# Invert predictions
y_train_pred = scaler.inverse_transform(y_train_pred.detach().numpy())
y_train = scaler.inverse_transform(y_train.detach().numpy())
y_test_pred = scaler.inverse_transform(y_test_pred.detach().numpy())
y_test = scaler.inverse_transform(y_test.detach().numpy())

# Calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(y_train[:,0], y_train_pred[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(y_test[:,0], y_test_pred[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

Eventually, we'll carry out inferencing and save the output signals dataframe for backtesting:

```
# Inferencing
y_inf_pred, y_inf = predict_price(df_test, model, scaler)
signal = gen_signal(y_inf_pred, y_inf)

# Save signals as csv file
output_df = pd.DataFrame(index=df_test.index)
output_df['signal'] = signal
output_df.index.name = "Date"

output_filename = 'output/' + symbol + '_output.csv'
output_df.to_csv(output_filename)
```

With the signals csv files, we could simply run `output-backtester_wrapper.py` in the same directory that would load all the files in the output directory and run it with the backtester_wrapper to compute the evaluation metrics.

14.3.3 Multi-feature LSTM model

The implememtation of the single-feature LSTM model could be found in `integrated-strategy/LSTM-train_wrapper.py`.

We'll focus on looking at the `LSTM_predict` function, as the `main` function is simply a wrapper that calls the

`LSTM_predict` function with different ticker symbols.

The code structure of the multi-feature LSTM and single-feature LSTM is largely the name, except that we'll need the **merged dataframe** as the input and we'll need to change the input/output dimensions of the model.

```
# load file
dir_name = os.getcwd()
data_dir = os.path.join(dir_name,"database_real/machine_learning_data/")
sentiment_data_dir=os.path.join(dir_name,"database/sentiment_data/data-result/")

# Get merged df with stock tick and sentiment scores
df, scaled, scaler = merge_data(symbol, data_dir, sentiment_data_dir, strategy)

look_back = 60 # choose sequence length

x_train, y_train, x_test_df, y_test_df = load_data(scaled, look_back)

# make training and test sets in torch
x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test_df).type(torch.Tensor)
y_train = torch.from_numpy(y_train).type(torch.Tensor)
y_test = torch.from_numpy(y_test_df).type(torch.Tensor)

# Hyperparameters
num_epochs = 100
lr = 0.01
batch_size = 72
input_dim = 7
hidden_dim = 64
num_layers = 4
output_dim = 7
torch.manual_seed(1) # set seed

print("Hyperparameters:")
print("input_dim: ", input_dim, ", hidden_dim: ", hidden_dim, ", num_layers: ", num_
↪layers, ", output_dim", output_dim)
print("num_epochs: ", num_epochs, ", batch_size: ", batch_size, ", lr: ", lr)

train = torch.utils.data.TensorDataset(x_train,y_train)
test = torch.utils.data.TensorDataset(x_test,y_test)

train_loader = torch.utils.data.DataLoader(dataset=train,
                                           batch_size=batch_size,
                                           shuffle=False)

test_loader = torch.utils.data.DataLoader(dataset=test,
                                          batch_size=batch_size,
                                          shuffle=False)

model = LSTM(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim, num_
↪layers=num_layers)

loss_fn = torch.nn.MSELoss()
optimiser = torch.optim.Adam(model.parameters(), lr=lr)
```

(continues on next page)

(continued from previous page)

```
hist = np.zeros(num_epochs)

# Number of steps to unroll
seq_dim = look_back - 1

# Train model
for t in range(num_epochs):
    for i, (train_data, train_label) in enumerate(train_loader):
        # Forward pass
        train_pred = model(train_data)
        loss = loss_fn(train_pred, train_label)

        hist[t] = loss.item()

        # Zero out gradient, else they will accumulate between epochs
        optimiser.zero_grad()

        # Backward pass
        loss.backward()

        # Update parameters
        optimiser.step()

    if t % 10 == 0 and t != 0:
        y_train_pred = model(x_train)
        loss = loss_fn(y_train_pred, y_train)
        print("Epoch ", t, "MSE: ", loss.item())

# Make predictions
y_test_pred = model(x_test)

# Invert predictions
y_train_pred = scaler.inverse_transform(y_train_pred.detach().numpy())
y_train = scaler.inverse_transform(y_train.detach().numpy())
y_test_pred = scaler.inverse_transform(y_test_pred.detach().numpy())
y_test = scaler.inverse_transform(y_test.detach().numpy())

# Calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(y_train[:,0], y_train_pred[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(y_test[:,0], y_test_pred[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

visualise(df, y_test[:,0], y_test_pred[:,0], pred_filename)

signal_dataframe = gen_signal(y_test_pred[:,0], y_test[:,0], df[len(df)-len(y_test):].
↪index, by_trend=True)

# Save signals as csv file
output_filename = 'LSTM_output_trend/' + symbol + '_output.csv'
signal_dataframe.to_csv(output_filename,index=False)
```

Note that we'll need to pass the ticker symbol name and the name of the technical indicator (to be included in the merged dataframe) to the `LSTM_predict` function, for example by calling in this way:

```
LSTM_predict('0001', 'macd-crossover')
```

Similarly, we could run the `output-backtester_wrapper.py` file to backtest the output signals.

References

- Long Short Term Memory

Image sources

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 15

Paper Trading with Interactive Brokers

In this tutorial, you will learn:

- What is paper trading
- How to start paper trading with Interactive Brokers

15.1 Intro to paper trading

Definition

A **paper trade** is a simulated trade that allows an investor to practice buying and selling without risking real money.

In this module, we will first set up a connection to Interactive Brokers Trader Workstation (IB TWS).

Then, we will learn how to create basic contracts, request market data, manage orders, and request account summary.

15.2 Setup Interactive Brokers API

Before setting up a connection to IB TWS, there are few tasks to be completed:

1. Visit InteractiveBrokers website, and open an account
2. Download IB API software from InteractiveBrokers GitHub account

3. Download TWS software from InteractiveBrokers TWS
4. Choose an IDE that you code in
5. Subscribe to market data

For detailed instructions, please refer to InteractiveBrokers Initial Setup.

15.3 Connect to Interactive Brokers TWS

Once finish the setup, it's time to connect to IB TWS. Use `app.connect()` to establish an API connection.

```
class App(EWrapper, EClient):

    def __init__(self):
        EClient.__init__(self, self)

# Establish API connection
# app.connect(ipAddress, portNumber, clientId)
app = App()
app.connect('127.0.0.1', 7497, 0)
app.run()
```

If you are successfully connected to IB TWS, you will get the below output in your terminal.

```
ERROR -1 2104 Market data farm connection is OK:usfarm.nj
ERROR -1 2104 Market data farm connection is OK:cashfarm
ERROR -1 2104 Market data farm connection is OK:usfarm
ERROR -1 2106 HMDS data farm connection is OK:ushmds
ERROR -1 2158 Sec-def data farm connection is OK:secdefil
```

15.4 Create Basic Contracts

Then, let's create basic contract objects (trading instruments) such as stocks, or fx pairs.

```
from ibapi.contract import Contract

# Create contracts - stocks
tsla_contract = Contract()
tsla_contract.symbol = "TSLA"
tsla_contract.secType = "STK"
tsla_contract.exchange = "ISLAND"
tsla_contract.currency = "USD"
```

(continues on next page)

(continued from previous page)

```
# Create contracts - fx pairs
eurgbp_contract = Contract()
eurgbp_contract.symbol = "EUR"
eurgbp_contract.secType = "CASH"
eurgbp_contract.currency = "GBP"
eurgbp_contract.exchange = "IDEALPRO"
```

15.5 Request Market Data

Using the contract objects, we can request both streaming and historical market data.

15.5.1 Request Streaming Market Data

Use `app.reqMktData()` to request streaming market data.

```
class App(EWrapper, EClient):

    # Receive market data
    def tickPrice(self, tickerId, field, price, attribs):
        print("Tick Price. Ticker Id:", tickerId, ", TickType: ", TickTypeEnum.to_
↪str(field),
              ", Price: ", price, ", CanAutoExecute: ", attribs.canAutoExecute,
              ", PastLimit: ", attribs.pastLimit, ", PreOpen: ", attribs.preOpen)

# Request market data
# app.reqMktData(tickerId, contract, genericTickList, snapshot, regulatorySnaphsot,␣
↪mktDataOptions)
app.reqMktData(1, tsla_contract, '', False, False, None)
```

Note that if you haven't subscribed the market data, you will receive 10-15 minute delayed streaming data. Before getting the delayed streaming data, make sure you use `app.reqMarketDataType(3)` to switch market data type to delayed data.

```
# Switch market data type
# 3 for delayed data
app.reqMarketDataType(3)
```

15.5.2 Request Historical Market Data

Use `app.reqHistoricalData()` to request historical bar data.

```
class App(EWrapper, EClient):

    # Receive historical bar data
    def historicalData(self, reqId, bar):
        print("HistoricalData. ReqId:", reqId, "BarData.", bar)

# Request historical bar data
# app.reqHistoricalData(tickerId, contract, endDateTime, durationString,␣
↪barSizeSetting, whatToShow, useRTH, formatDate, keepUpToDate)
app.reqHistoricalData(1, eurgbp_contract, '', '1 M', '1 day', 'ASK', 1, 1, False,␣
↪None)
```

15.6 Manage Orders

Now, let's try to make an order!

First, write some methods in EWrapper that are required for receiving all relevant information on order opening, order status, and order execution.

```
class App(EWrapper, EClient):

    def nextValidId(self, orderId: int):
        super().nextValidId(orderId)
        self.nextorderId = orderId
        print('The next valid order id is: ', self.nextorderId)

    def orderStatus(self, orderId, status, filled, remaining, avgFillPrice, permId,␣
↪parentId,
                    lastFillPrice, clientId, whyHeld, mktCapPrice):
        print("OrderStatus. Id: ", orderId, ", Status: ", status, ", Filled: ",␣
↪filled,
              ", Remaining: ", remaining, ", AvgFillPrice: ", avgFillPrice,
              ", PermId: ", permId, ", ParentId: ", parentId, ", LastFillPrice: ",␣
↪lastFillPrice,
              ", ClientId: ", clientId, ", WhyHeld: ", whyHeld, ", MktCapPrice: ",␣
↪mktCapPrice)

    def openOrder(self, orderId, contract, order, orderState):
        print("OpenOrder. PermID: ", order.permId, ", ClientId: ", order.clientId,
              ", OrderId: ", orderId, ", Account: ", order.account, ", Symbol: ",␣
↪contract.symbol,
              ", SecType: ", contract.secType, " , Exchange: ", contract.exchange,
              ", Action: ", order.action, ", OrderType: ", order.orderType,
              ", TotalQty: ", order.totalQuantity, ", CashQty: ", order.cashQty,
              ", LmtPrice: ", order.lmtPrice, ", AuxPrice: ", order.auxPrice,
              ", Status: ", orderState.status)

    def execDetails(self, reqId, contract, execution):
        print("ExecDetails. ", reqId, " - ", contract.symbol, ", ", contract.secType,
              ", ", contract.currency, " - ", execution.execId, ", ", execution.
↪orderId,
              ", ", execution.shares , ", ", execution.lastLiquidity)
```

15.6.1 Place Orders

To place an order, use `app.placeOrder()` to submit an order.

```
# Place order
# app.placeOrder(orderId, contract, order)
app.placeOrder(app.nextorderId, eurgbp_contract, order)
```

15.6.2 Modify Orders

To modify the order, call `app.placeOrder()` again with the order id to be modified and the updated parameters.

```
# Modify order
order_id = 1
order.lmtPrice = '0.82'
app.placeOrder(order_id, eurgbp_contract, order)
```

15.6.3 Cancel Orders

To cancel an order by its order id, use `app.cancelOrder()`.

To cancel all open orders, use `app.reqGlobalCancel()`.

```
# Cancel order by order Id
app.cancelOrder(app.nextorderId)

# Cancel all open orders
app.reqGlobalCancel()
```

15.7 Request Account Summary

Lastly, use `app.reqAccountSummary()` to get the summarized account information.

```
class App(EWrapper, EClient):

    # Receive account summary
    def accountSummary(self, reqId:int, account:str, tag:str, value:str,␣
↪currency:str):
          print("Acct Summary. ReqId:" , reqId , "Acct:", account, "Tag: ", tag,
↪"Value:", value,
                "Currency:", currency)

# Request account summary in base currency
app.reqAccountSummary(9002, "All", "$LEDGER");
```

(continues on next page)

(continued from previous page)

```
# Request account summary in HKD
app.reqAccountSummary(9002, "All", "$LEDGER:HKD");
```

References

- Investopedia - Paper Trade
- Trader Workstation API

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.

CHAPTER 16

Resources

- Investopedia
- StockCharts

16.1 Acknowledgement

We would like to extend our gratitude to the contributions of many people who helped get this project off the ground, including people who coordinated the final year project, gave feedback on the material, or otherwise supported the project.

16.2 About the author

The Algo Trading Learning Repo was primarily developed by Angel Woo, Lee Kwanyoung and Snow Wu, who are undergraduate students majoring in Computer Science at the University of Hong Kong, under the supervision of Dr. Ruibang Luo.

Visit this website to learn more about the project.

Being a group of female undergraduate students , we advocate the initiative of , and we create resources in hopes of assisting other females to too get a tech career opportunity in finance companies.

Are you also a female who is interested in working on Tech & Finance? Here are some good articles to understand more and get yourself involved in the community:

- A day in the life of a female in tech at Morgan Stanley

- Google Women Techmakers
- *And more to be updated. . .*

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Attention:

All investments entail inherent risk. This repository seeks to solely educate people on methodologies to build and evaluate algorithmic trading strategies. All final investment decisions are yours and as a result you could make or lose money.