# Monash University (Clayton Campus)

# MTH 5530 – Computational methods in finance
# Semester 1

# Assignment 2

Lecturer: Oscar Tian Yu

Name: Ken SO
SID: 34856323
Coding/Computation method: Python (coding)/LaTeX (Formula)/ Excel

**Question 1: Bloomberg Market Concepts (BMC) certificate (10 marks)**
Complete the Bloomberg Market Concepts online course and take quiz questions. Please submit your certificate as the evidence of completion. Use your Monash email address to register and access: https://portal.bloombergforeducation.com/ (10 marks)


**< Attach 01 - Certificate_Bloomberg_mkt_concept_1>**

**Question 2: Measuring Portfolio Performance (30 marks)**
Assume that you are a portfolio manager. You set up a buy-and-hold portfolio with an initial investment of $10,000. You start your portfolio from your birthday in 2023 (if your birthday in 2023 is a weekend or public holiday, use the next business day instead), investing in the shares of the Big Four banks (CBA, NAB, Westpac and ANZ) with $2,500 in each of them on your birthday. ASX200 Index will be used as the market portfolio for benchmarking and let's assume there has been no dividend paid.

Methodology:
This question requires a data analytics model for real stocks' data, so, first thing first, I assume myself a portfolio manager as requested. From this assumption, I further assume that I am a fund manager holding $10,000 in my escrow account where the fund is sourced from clients. My investment strategy is long-only and my investment target is mainly in financial institution ("FI") and related index.

In practice, I will follow **my management framework** to build up the data analytic model for any further comparison, visualization and computation. **My management framework** is broken down as follows.

| **My management framework** |
| --- |
| *Planning* |
| - In planning phrase, I need to ensure my understanding on the question is theoretically correct |

- Given the background information above, I summarize some points as motivation below:

    1. I am a fund manager providing portfolio management service to my clients

    2. In my escrow account, I held $10,000 cash as the initial wealth

    3. My goal is to maximize the value of asset for my clients

    4. After summarizing risk appetite and preference from my clients, along with due diligence report presented a low third parties' risk, I sumarized 2 investment plans based on my insight and third parties' professional advice.

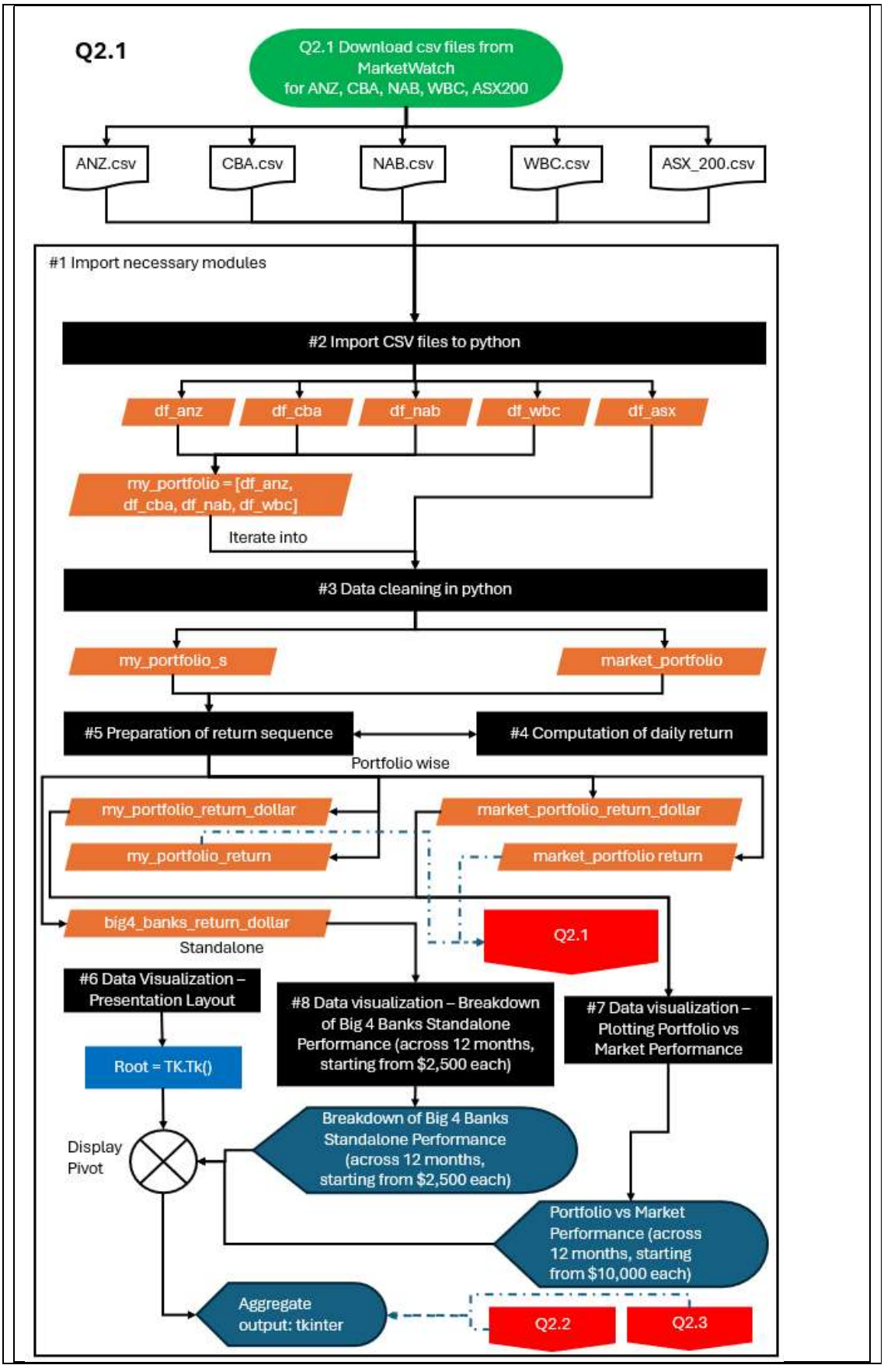
        - Buy-and-hold ANZ, CBA, NAB and WBC (i.e. the Big 4 banks) ("my portfolio"), or;
        - Buy-and-hold ASX200 Index ("market portfolio")

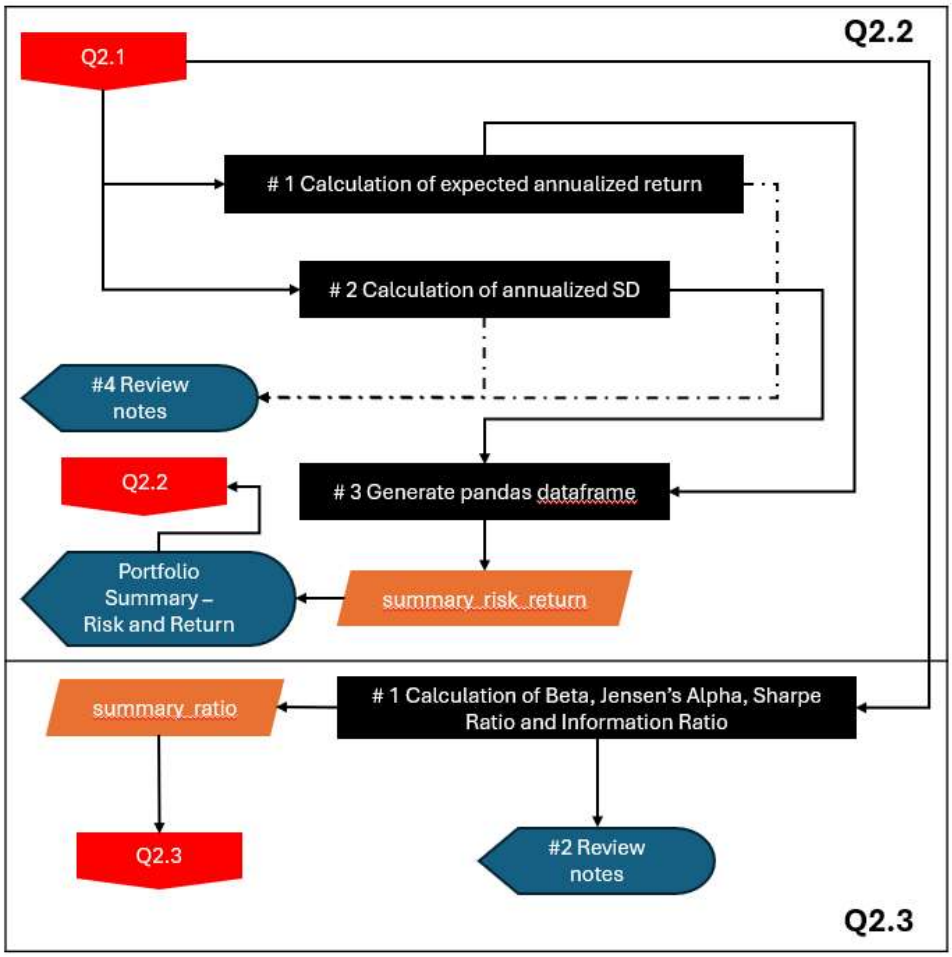
    5. To perform portfolio management practice, real stocks' data are required (as requested in Q2.1). After collecting the real stocks' data, I will plot the return curves for both standalone strategy as a mean of scenario analysis to compare those 2 strategies above

    6. As a professional, I would exercise **due models practice** cross-checking my computation as well as supporting my findings and methodologies. **Due models practice** refers to reperform existing model with same operations by another development platform, then cross-checking their output

       To construct these 2 models, I prefer using **Python** as the *"existing model"* and **Excel Spreadsheet** as the *"supporting model"*

    7. As such, a **floor plan** for data-analytics model development is required. Please find the **<extract of Attach 02 – ASM_Q2_floor plan>** below.


Q2.1
Q2.1 Download csv files from MarketWatch for ANZ, CBA, NAB, WBC, ASX200
ANZ.csv
CBA.csv
NAB.csv
WBC.csv
ASX_200.csv
#1 Import necessary modules
#2 Import CSV files to python
df_anz
df_cba
df_nab
df_wbc
df_asx
my_portfolio = [df_anz, df_cba, df_nab, df_wbc]
Iterate into
#3 Data cleaning in python
my_portfolio_s
market_portfolio
#5 Preparation of return sequence
#4 Computation of daily return
Portfolio wise
my_portfolio_return_dollar
market_portfolio_return_dollar
my_portfolio_return
market_portfolio return
big4_banks_return_dollar
Standalone
Q2.1
#6 Data Visualization – Presentation Layout
#8 Data visualization – Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each)
#7 Data visualization – Plotting Portfolio vs Market Performance
Root = TK.Tk()
Display Pivot
Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each)
Portfolio vs Market Performance (across 12 months, starting from $10,000 each)
Aggregate output: tkinter
Q2.2
Q2.3




*Organizing*

- As requested by Q2.1, stock data could be sourced from any stock watch platform. based on your guidance during the lecture. In other words, the source of data is flexible as long as it can help to achieve my goal. I picked MarketWatch as the data provider. Note that YahooFinance is not free in Australia nowadays

- As planned, the data will be cleaned in python model in order to maintain the applicability to different data. For excel spreadsheet model, the cleaning is flexible as this spreadsheet model is used for supporting purpose (extra work)

1. To compare the performance of both investment plans, some performance indicators are required to calculate. As requested by Q2.2 and Q2.3, they will be:

| | |
|---|---|
| Expected annualized return | Annualized standard deviation |
| Beta | Jensen's Alpha |
| Sharpe Ratio | Information Ratio |

For theoretical understanding, including the formulae required for each performance indicators will be discussed in the sections – Q2.2 and Q2.3

2. Proper visualizations are required. As requested by Q2.1, a data plot for comparison of my portfolio and market portfolio across 12 months in dollar, starting from $10,000, is required

3. As a decent professional, I would also plot the individual performance of the Big 4 banks in my portfolio in dollar, starting from $2,500, in the purpose of justifying my plotting of portfolio performance comparison between my own one and market one

4. Tabulate the any analysis of my portfolio and market portfolio and put them into the same output for convenience

*Executing*

- Build the algorithm/model in accordance with the floor plan for existing model first and then supporting model

- Walkthrough for the source code and spreadsheet model will be included in following sections accordingly

- Exercise and test the model with **concurrent control practice**. **Concurrent control practice** will be discussed in Controlling section

*Controlling*

**Concurrent control practice**

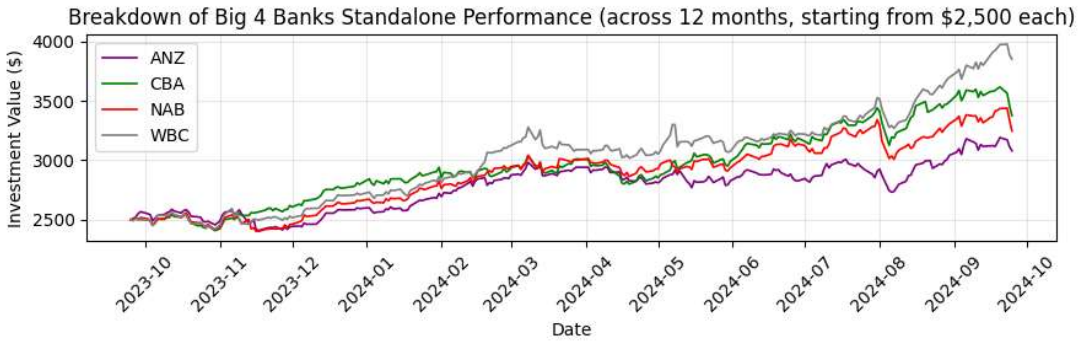
- This is a practice based on my personal experience, refers to monitor and regulate processes and operations in real time during model development

- As this is a personal assignment, I am considered as an end-to-end full stacker. Concurrent control practice is the most efficient way to resolve any issue arises during the model development

- Controlling items included (but not limited to):
  1. Bug fixing
  2. Model optimization
  3. Model Review
  4. Performance Indicators reasoning (i.e. fact check)
  5. Reporting and review
  6. Attachment preparation and review (e.g. floor plan)

The **walkthrough of source code** and **designated result by model operation** will be discussed in the following section.

**Question 2.1:** Download the daily share price (close price) of the Big Four Banks and the benchmark ASX200 Index from your birthday in 2023 for the next 12 months. Plot the daily return of 1) your portfolio ($2,500 in each of the Big Four banks at the beginning) and 2) the market portfolio of ASX200 Index ($10,000 at the beginning) with a buy-and-hold strategy over the 12-month period. (10 marks)


**<extract of Attach 04 – Q2 - plot of port vs mkt>**

Attached is the extracted of the summarized visualization. With a standalone buy-and-hold strategy for my portfolio and market portfolio (where the initial investment for both portfolios is $10,000), I obtained 2 historical return processes across 12 months, starting from my birthday 25 Sep 2023 to 25 Sep 2024 (see graph 1 – Portfolio vs Market Performance (across 12 months, starting from $10,000 each)).

As professional practice assumed, I also plotted the breakdown of big 4 banks individual performance to understand the portfolio wise return processes better. (see graph 2 – Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each).

Below is the elaboration of my source code.

**<Attach 05 - ASM_2_Q2_v5 (submission)>**

**#1: Import necessary modules**

1. Pandas: this is for data analytics as it works with dataframe type data **(line 3)**
2. Matplotlib.pyplot: this is for data visualization, for both table and chart **(line 4)**
3. I called FigureCanvasTKAgg from matplotlib.backends.backend_tkagg for integrating all visualization data in one image **(line 5)**

4. Numpy: this is for mathematical calculation. Mostly adopted in Q2.2 and Q2.3 **(line 6)**
5. Tkinter: this is for integrating all visualization data in one image, working with FigureCanvasTKAgg **(line 7)**

```python
""" import of necessary modules and csv file as well as denote global parameters """
# import necessary module
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
import tkinter as TK
```

**#2: Import CSV files to python**

1. Open and read CSV files for ANZ, CBA, NAB, WBC and ASX 200 index with in python environment and transform them to be dataframes with pd.read_csv() function. **(line 2 to line 6)**
2. Denote transformed dataframes corresponding to their name respectively. df_anz for ANZ; df_cba for CBA, df_nab for NAB, df_wbc for WBC and df_asx for ASX 200 index. **(line 2 to line 6)**

```python
# import csv file for stocks and index
df_anz = pd.read_csv('ANZ.csv')
df_cba = pd.read_csv('CBA.csv')
df_nab = pd.read_csv('NAB.csv')
df_wbc = pd.read_csv('WBC.csv')
df_asx = pd.read_csv('ASX_200.csv')
```

**#3: Data cleaning in python**

1. After importing csv files into python, I noticed that some of the data is not in the data type “datetime”, which is unexpected. As such, I constructed a function: convert_to_date(column) to centrally process all datetime transformation work for all CSV files, where the input required for the function is the column in a CSV file. It will be the ‘Date’ column in all CSV files.

2. I notice that the data is in reverse chronological order, that’s, the data is arranged from 25 Sep 2024 to 25 Sep 2025 (see the snapshot below) in default, which is unexpected. Considering this, I chose to operate the sorting process (in terms of ‘Date’) in chronological order under the iteration of my portfolio **(line 23)** with .sort_value[‘Date’], at the same time converting all ‘Date’ in dataframes to datetime **(line 24)**.

ANZ.csv > data

```
1  Date,Open,High,Low,Close,Volume
2  09/25/2024,31.05,31.09,30.58,30.78,"5,965,180"
3  09/24/2024,31.8,31.83,30.97,31.11,"6,341,290"
4  09/23/2024,31.73,31.83,31.52,31.7,"3,118,861"
5  09/20/2024,31.61,31.94,31.45,31.89,"11,733,090"
6  09/19/2024,31.54,31.68,31.34,31.5,"8,605,576"
7  09/18/2024,31.27,31.31,31.02,31.15,"4,199,100"
8  09/17/2024,31.4,31.44,31.09,31.21,"2,663,798"
9  09/16/2024,31.21,31.38,31.04,31.2,"4,254,118"
10 09/13/2024,31.39,31.47,31.01,31.15,"3,054,256"
```

**Snapshot of raw data with reverse chronological order**

3. Additionally, I notice that some data are not in numeric type, particularly the ASX 200 data, where the close price is more than 3 digits. It is because the comma is the delimiter of CSV file. All "," in the file is considered as delimiters and python read values comma by comma. As such, a central function: convert_to_float(column) for all data is constructed **(line 5 to line 6)** in order to numerate all comma separated numbers under 'Close' in ASX_200.csv. In this case, I only processed ASX 200 data **(line 9)**.

ASX_200.csv > data

```
1  Date,Open,High,Low,Close
2  09/25/2024,"8,156.00","8,179.10","8,115.00","8,126.40"
3  09/24/2024,"8,153.60","8,172.70","8,111.30","8,142.00"
4  09/23/2024,"8,192.80","8,193.40","8,131.50","8,152.90"
5  09/20/2024,"8,200.00","8,246.20","8,188.00","8,209.50"
6  09/19/2024,"8,158.60","8,200.30","8,133.10","8,191.90"
7  09/18/2024,"8,142.60","8,153.50","8,114.10","8,142.10"
8  09/17/2024,"8,128.60","8,150.50","8,128.60","8,140.90"
9  09/16/2024,"8,103.20","8,145.00","8,103.20","8,121.60"
```

4. To iterate the cleaned dataframes back to my portfolio, I chose to create a new list: my_portfolio_s for clean storage. **(line 20 to line 25)**

**Display of cleaned data (sorted by 'Date' in chronological order)**

```
4   2024-09-19  31.54  31.68  31.34  31.50   8,605,576
3   2024-09-20  31.61  31.94  31.45  31.89  11,733,090
2   2024-09-23  31.73  31.83  31.52  31.70   3,118,861
1   2024-09-24  31.80  31.83  30.97  31.11   6,341,290
0   2024-09-25  31.05  31.09  30.58  30.78   5,965,180

[255 rows x 6 columns],          Date    Open    High
```

5. Perform the same practice to ASX200 data **(line 27 to line 28)**

```python
""" data cleaning in ASX ("Notice that there's ',' in the numbers under
'Close' where the entire field is determined as object by python") """

# function to convert columns with commas to float
def convert_to_float(column):
    return pd.to_numeric(column.str.replace(',', ''), errors='coerce')

# Apply to the 'Close' column (and any other relevant columns)
df_asx['Close'] = convert_to_float(df_asx['Close'])

""" data cleaning in Big4 banks and market data ("Notice that 'Date' type
is not in the type 'datetime'") """

# function to convert raw date to datetime
def convert_to_date(column):
    return pd.to_datetime(column)

# Summarize my_portfolio and market_portfolio
my_portfolio = [df_anz, df_cba, df_nab, df_wbc]
my_portfolio_s = []

for df in my_portfolio:
    df['Date'] = convert_to_date(df['Date'])
    df = df.sort_values(['Date'])  # Sort in ascending order (earliest to latest)
    my_portfolio_s.append(df)

df_asx['Date'] = convert_to_date(df_asx['Date'])
market_portfolio = df_asx.sort_values(['Date'])  # Sort in ascending order (earliest to latest)
```

**#4 Computation of daily return**

1. Constructed a common function for daily return computation with operation .pct_change(), which is equivalent to:

$$return_t = \frac{close\ price_{t+1}}{close\ price_t} - 1$$

**(line 2 to line 5)**

```python
# common function to compute daily return for each stocks and index
def compute_daily_return(df):
    df = df.copy()  # Avoid modifying original dataframe
    df['Daily Return'] = df['Close'].pct_change()  # More direct percentage change calculation
    return df
```

**#5 Preparation of return sequence**

1. This is the most complicated part. My idea is that

    a. Iterate a new list: big4_banks_return to store all **(line 3 to line 11)**
    b. Calculate the return in percentage for my portfolio with function: compute_daily_return(df) **(line 6)**
    c. Drop columns other than 'Date' and 'Daily Return' **(line 8)**
    d. Notice that the first entries under 'Daily Return' is NaN. This error will distort ploting and computation in python. The initial daily return is fixed by iloc operation that all initial daily returns are set to be 0 **(line 10)**

| | Date | Open | High | Low | Close | Volume | Daily Return |
|---|---|---|---|---|---|---|---|
| 254 | 2023-09-25 | 20.93 | 21.12 | 20.86 | 21.12 | 4,448,333 | NaN |
| 253 | 2023-09-26 | 21.03 | 21.14 | 20.90 | 21.05 | 5,557,913 | -0.003314 |
| 252 | 2023-09-27 | 21.06 | 21.24 | 21.05 | 21.19 | 5,277,856 | 0.006651 |
| 251 | 2023-09-28 | 21.10 | 21.25 | 21.06 | 21.13 | 3,381,611 | -0.002832 |
| 250 | 2023-09-29 | 21.15 | 21.20 | 21.07 | 21.15 | 5,898,677 | 0.000947 |

e. Now, I aggreate all big 4 banks dataframe in term of 'Date' with new list: portfolio_returns **(line 15)** and make sure the first entry of 'Daily Return' is 0 **(line 16)**
f. To iterate the portfolio_returns series in % **(line 18 to 21)** for the use of further anaylsis (Q2.2 and Q2.3), I firstly merge all big 4 bank dataframe in term of 'Date' and indicate 'Date' as inner join field such that 'Daily Return' in each big 4 banks' dataframes will be excluded as 'Daily_return_temp' and sum up together (in a new total) as .merge() does **(line 19)**
g. To operate the summed up daily return pandas series 'Daily_return_temp' (in %), given the weighting for each big 4 banks' dataframe in my portfolio is \$2,500/\$10,000 = 1/4, I apply this weighting in 'Daily_return_temp' and accumalated back to the value into original field 'Daily_return' **(line 20)**. While ['Daily_Return_temp'] is redundant after the accumulation, .drop operation is applied for dropping this field **(line 21)**
h. The mainframe of this question is asking for performance comparison between my portfolio and market portfolio in dollar sense **<extract of Attach 04 – Q2 - plot of port vs mkt>**
i. . Hence, a pandas series of return in dollar should be established. I copied the 'Daily Return' from my_portfolio_return **(line 26)** and compute the return in dollar by multiplying \$10,000 as a common product **(line 27)**
j. As assumed that, I should maintain my professionalism. Hence, I also prepared an extra plotting for individual return series of the big 4 banks. I applied the same practice as a to h did on **(line 30 to 41)** where the main difference is that the individual return series is starting from \$2,500 per each (as the common product of 'Daily Return' in %) **(line 40)**. A dataframe: big4_banks_return_dollar is created for separate plotting
k. Apply the same practice as in a to h on the market portfolio **(line 44 to 55)**. The difference between market portfolio and my portfolio is that no iteration is required as market portfolio contains only one underlying asset – ASX200 index

```python
""" Preparation of return sequence """
# First calculate percentage returns for portfolio
big4_banks_return = []

for df in my_portfolio_s:
    df_r = compute_daily_return(df)
    # Keep only Date and Daily Return columns
    df_rr = df_r[['Date', 'Daily Return']].copy()
    # Set first return to 0 (no return on first day)
    df_rr.iloc[0, df_rr.columns.get_loc('Daily Return')] = 0
    big4_banks_return.append(df_rr)

# Create equal-weighted portfolio returns
# Merge all banks on Date and calculate average return
portfolio_returns = big4_banks_return[0][['Date']].copy()
portfolio_returns['Daily Return'] = 0

for df in big4_banks_return:
    portfolio_returns = portfolio_returns.merge(df, on='Date', how='inner', suffixes=('', '_temp'))
    portfolio_returns['Daily Return'] += portfolio_returns['Daily Return_temp'] / 4
    portfolio_returns = portfolio_returns.drop('Daily Return_temp', axis=1)

my_portfolio_return = portfolio_returns.sort_values(['Date'])

# Convert portfolio percentage returns to cumulative dollar values starting from $10,000
my_portfolio_return_dollar = my_portfolio_return.copy()
my_portfolio_return_dollar['Daily Return'] = (1 + my_portfolio_return_dollar['Daily Return']).cumprod() * 10000

# Process individual bank returns in dollar for graph 2
big4_banks_return_dollar = []

for df in my_portfolio_s:
    df_r = compute_daily_return(df)
    df_rr = df_r[['Date', 'Daily Return']].copy()

    # Set first return to 0
    df_rr.iloc[0, df_rr.columns.get_loc('Daily Return')] = 0

    # Convert to cumulative dollar value starting from $2500 per stock
    df_rr['Daily Return'] = (1 + df_rr['Daily Return']).cumprod() * 2500
    big4_banks_return_dollar.append(df_rr)

# Process market_portfolio_return (daily) in dollar
market_df = compute_daily_return(market_portfolio)
market_portfolio_return_dollar = market_df[['Date', 'Daily Return']].copy()

# Set first return to 0 and convert to cumulative dollar value starting from $10000
market_portfolio_return_dollar.iloc[0, market_portfolio_return_dollar.columns.get_loc('Daily Return')] = 0
market_portfolio_return_dollar['Daily Return'] = (1 + market_portfolio_return_dollar['Daily Return']).cumprod() * 10000
market_portfolio_return_dollar = market_portfolio_return_dollar.sort_values(['Date'])

# Process market_portfolio_return (daily) in percentage
market_portfolio_return = market_df[['Date', 'Daily Return']].copy()
market_portfolio_return.iloc[0, market_portfolio_return.columns.get_loc('Daily Return')] = 0
market_portfolio_return = market_portfolio_return.sort_values(by='Date')
```

**#6 Data visualization – Presentation Layout**

1. Import date formatting tools to better present the date **(line 3)**
2. As the graphs required for this question will be jointly presented with the tables required in Q2.2 and Q2.3, Tkinter window is created and root is set for integrating the presentation of both graphs **(line 6)**

```python
""" Data Visualization"""
# Import date formatting tools
import matplotlib.dates as mdates

# Create the Tkinter window
root = TK.Tk()
```

**#7 Data visualization – Plotting Portfolio vs Market Performance**

1. Create a figure for plotting and define its framework: 4 in 1 subplot with figure size 10 x 20 **(line 2)**
2. In order to show the return processes for both my portfolio and market portfolio on the same graph, I denoted the same axs index '0' for both plots. Plot my_portfolio return_dollar and market_portfolio_dollar in terms of 'Date' and 'Daily Return' in blue and orange lines respectively with proper formatting **(line 5 to line 9)**.
3. Set title for this graph: 'Portfolio vs Market Portfolio (across 12 months, starting from $10,000 each)' **(line 11)**. Denote proper labels for x and y-axis **(line 12 to line 13)**. Differentiate axs[0] by .legend() **(line 14)** and set grid for the graph **(line 15)**.
4. As the original plot presented a poor x-axis ('Date') labelling, in this case, I called the mdates modules to deal with the issue **(line 18 to line 20)**. Note that tick_params is an operation under matplotlib module that control the density of the x-axis. This densify the labels presented in x-axis and thus including more dates into the graph.

```
1  # Create a figure for plotting
2  fig, axs = plt.subplots(4, 1, figsize=(10, 12))
3
4  # Plot 1: Portfolio vs Market performance in dollars
5  axs[0].plot(my_portfolio_return_dollar['Date'], my_portfolio_return_dollar['Daily Return'],
6              label='Portfolio (Big 4 Banks)', color='blue', linewidth=1.2)
7
8  axs[0].plot(market_portfolio_return_dollar['Date'], market_portfolio_return_dollar['Daily Return'],
9              label='Market (ASX200)', color='orange', linewidth=1.2)
10
11 axs[0].set_title('Portfolio vs Market Performance (across 12 months, starting from $10,000 each)')
12 axs[0].set_xlabel('Date')
13 axs[0].set_ylabel('Portfolio Value ($)')
14 axs[0].legend()
15 axs[0].grid(True, alpha=0.3)
16
17 # Format x-axis to show months
18 axs[0].xaxis.set_major_locator(mdates.MonthLocator())
19 axs[0].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
20 axs[0].tick_params(axis='x', rotation=45)
```

**#8 Data visualization – Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each)**

1. As professional decency assumed, this is an extra work for better understanding the portfolio return series in dollar.
2. Same practice as **#7** where the dataframe: big4_banks_return_dollar was plotted

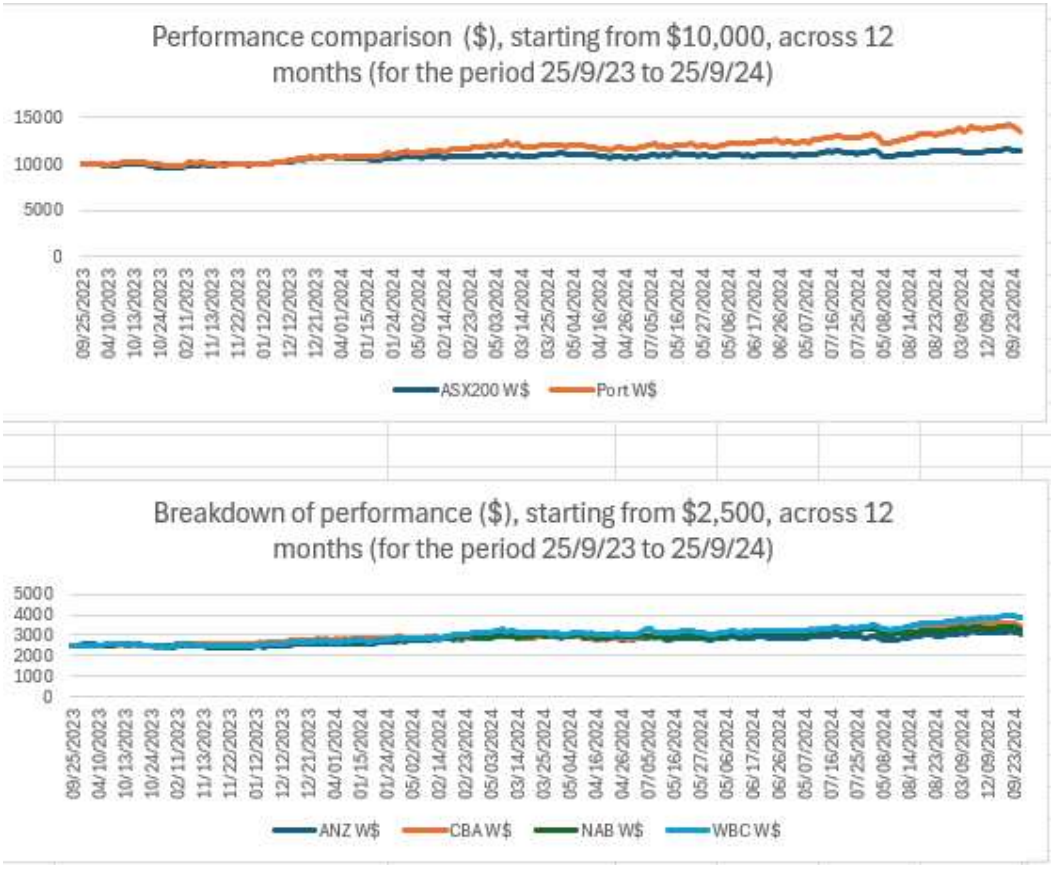
```python
# Plot 2: Individual bank performance breakdown
banks = ['ANZ', 'CBA', 'NAB', 'WBC']
colors = ['purple', 'green', 'red', 'grey']
for i, df in enumerate(big4_banks_return_dollar):
    axs[1].plot(df['Date'], df['Daily Return'], label=banks[i],
                color=colors[i], linewidth=1.2)

axs[1].set_title('Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each)')
axs[1].set_xlabel('Date')
axs[1].set_ylabel('Investment Value ($)')
axs[1].legend()
axs[1].grid(True, alpha=0.3)

# Format x-axis to show months for the second plot as well
axs[1].xaxis.set_major_locator(mdates.MonthLocator())
axs[1].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
axs[1].tick_params(axis='x', rotation=45)
```

**Concurrent control practice:**

Bug fixing

Performed simultaneously with model development. Please refer the comments box/ # “”” Notes in both supporting and existing model.

Spreadsheet model (supporting model)

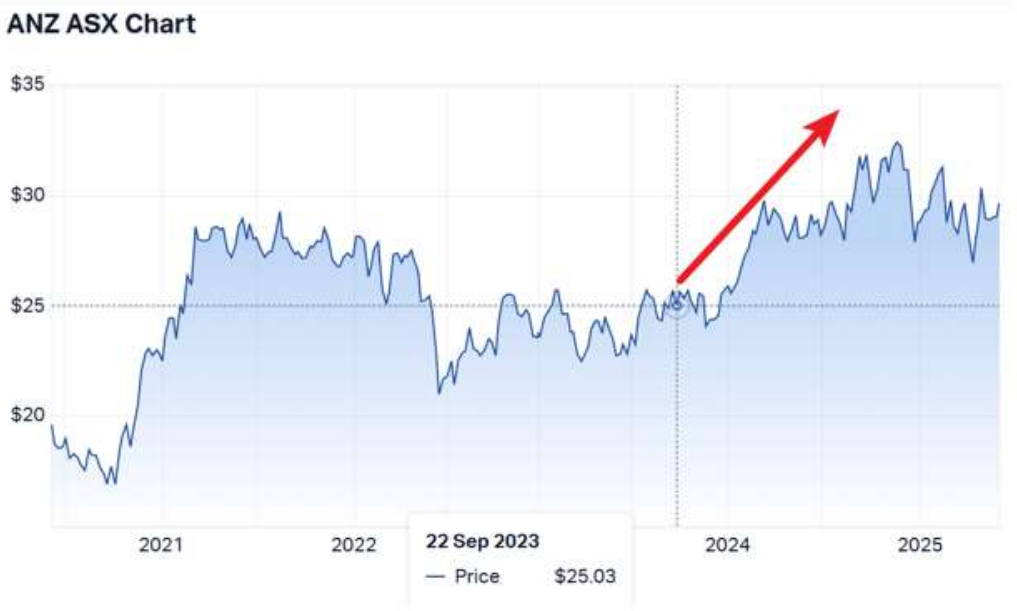



**<extract – Attach 07 – Spreadsheet model>**

As abovementioned, the raw data is in reverse chronological order and the date type is not unified as type ‘datetime’. In excel, I applied add-ins ‘Ablebits Data’ the flip the data horizontal to fix the first issue and perform data cleaning practice for ‘Date’ with IF() and TEXT() functions. The plots do align with python model.

Fact check


(Sourced from Market Index https://www.marketindex.com.au/asx200)




(Sourced from Market Index https://www.marketindex.com.au/asx/anz?src=search-all)

**CBA ASX Chart**


(Sourced from Market Index https://www.marketindex.com.au/asx/cba?src=search-all)

**NAB ASX Chart**


(Sourced from Market Index https://www.marketindex.com.au/asx/nab?src=search-all)


(Sourced from Market Index https://www.marketindex.com.au/asx/wbc?src=search-all)

As per the 5 years price chart for ANZ, CBA, NAB, WBC and ASX 200, we can observe that the price is keep moving upwards from September 2023 to September 2024, which implies that if we invest our wealth into either market index or portfolio of big 4 banks, our wealth will raise. Note that the WBC is the champion of the price boosting, which consenses with my plotting – Breakdown of Big 4 Banks Standalone Performance (across 12 months, starting from $2,500 each).

**Question 2.2:** Calculate the expected annualised return and annualised standard deviation of your portfolio and the market portfolio. (10 marks)

Portfolio Summary - Risk and Return

| Measures | Expected Annualized Return (%) | Annualized Std Dev (%) |
|---|---|---|
| Market Portfolio | 14.387 | 11.479 |
| My Portfolio | 31.353 | 15.3293 |

**<extract of Attach 04 – Q2 - plot of port vs mkt>**

Below is the elaboration for my source code in this section.

**< Attach 05 - ASM_2_Q2_v5 (submission)>**

**# 1 Calculation of expected annualized return**

1. Clean data to avoid misalignment issues as Q2.1 did **(line 5 to line 8)**
2. Compute expected annualized return as suggested by you (simply multiply 252 to mean of daily return for both my portfolio and market portfolio **(line 11 to line 12)**. Please note that T is a global parameter = 252.

```
""" Calculation of expected annualized return and annualised SD """
# Use percentage returns for calculations (excluding first day with 0 return)

# Reset index to avoid alignment issues
portfolio_daily_returns = my_portfolio_return['Daily Return'].iloc[1:].reset_index(drop=True)

# Reset index to avoid alignment issues
market_daily_returns = market_portfolio_return['Daily Return'].iloc[1:].reset_index(drop=True)

# Expected annualized return using simple multiplication by T (as per requirements)
expected_annualized_return_my_port = portfolio_daily_returns.mean() * T
expected_annualized_return_market = market_daily_returns.mean() * T
```

**# 2 Calculation of annualized SD**

1. Take the sampled SD for both my portfolio and market portfolio and multiply by $\sqrt{252}$ **(line 2 to line 3)**
2. Compute the annualized variance of market portfolio as well to prepare for calculation of Beta **(line 6)**

```
# Annualized standard deviation
annualized_sd_my_port = portfolio_daily_returns.std() * np.sqrt(T)
annualized_sd_market = market_daily_returns.std() * np.sqrt(T)

# Annualized variance for market (needed for beta calculation)
annualized_var_market = market_daily_returns.var() * T
```

**# 3 Generate pandas dataframe**

1. Generate pandas dataframe 'summary_risk_return' for coming visualization with proper labels and decimal places **(line 2 to line 6)**

```
# Generate summary dataframe
summary_risk_return = pd.DataFrame({
    'Measures': ['Market Portfolio', 'My Portfolio'],
    'Expected Annualized Return (%)': [expected_annualized_return_market * 100, expected_annualized_return_my_port * 100],
    'Annualized Std Dev (%)': [annualized_sd_market * 100, annualized_sd_my_port * 100]
})

```

**# 4 Prepare review note**

1. Print review notes "Summary Risk Return" and the dataframe for review purpose **(line 1 to line 2)**

```
print("Summary Risk Return:")
print(summary_risk_return)
```

As per the plotting from existing model (i.e. python model), I obtained the result at the beginning of this section. I considered this result is preliminarily reasonable, according to the plot "Portfolio vs Market Performance (across 12 months, starting from $10,000 each)". As we can observe that my portfolio outperformed market portfolio for approximately 1.35 times, graphically. As such, the expected return should be higher than market portfolio. I concluded that the result is fair in this sense of reasonableness.

**Concurrent control practice**

Bug fixing

Performed simultaneously with model development. Please refer the comments box/ # Notes in supporting and existing model.

Spreadsheet model

| Analysis of portfolio and market | | |
|---|---|---|
| | Port | Market |
| Expected annualized return | 0.312300924 | 0.143306 |
| Annualized SD | 0.152995767 | 0.114565 |

**<extract – Attach 07 – Spreadsheet model>**

By spreadsheet model (i.e. supporting model), the result is very similar to existing model. The difference is believed to be the discrepancy of backend programming method of python function and excel function. The supporting model provided a fair comparison to existing model.

Fact check

As per those price charts shown in Q2.1, I concluded that those figures computed by python model are fairly and truly supported by the empirical data.

**Question 2.3:** Assume that the market portfolio is ASX200 Index and the risk free return is 3%. Calculate Beta, Jenson's Alpha, Sharpe Ratio and Information Ratio of your portfolio of the Big Four banks with a buy-and-hold strategy and comment on your findings. (10 marks)

**Note:** you can simply multiply the expected daily return by the number of trading days (e.g., 252) in order to get the expected annualised return (similarly using 252 for annualised variance, covariance, etc.), under the assumption that the return series follows a normal distribution.

Portfolio Ratio Analysis

| Measures | Ratio |
|---|---|
| Beta | 1.0674 |
| Jensen Alpha | 0.162 |
| Sharpe Ratio | 1.8496 |
| Information Ratio | 1.8354 |

**<extract of Attach 04 – Q2 - plot of port vs mkt>**

Below is the elaboration for my source code in this part

**< Attach 05 - ASM_2_Q2_v5 (submission)>**

**# 1 Calculation of Beta, Jensen's Alpha, Sharpe Ratio and Information Ratio**

1. For Beta,
   a. The very first task is to prepare the annualized covariance first **(line 3)**
   b. Call annualized market variance prepared in Q2.2. Take the annualized covariance and the annualized market variance into calculation **(line 6)**
2. For Jensen's Alpha,
   a. The formula is based on the difference between LHS and RHS in CAPM model. Call expected_annualized_return_my_port and expected_annualized_return_market prepared in Q2.2, beta from 1, and Risk free rate (Rf = 0.03) from global **(line 9)**
3. For Sharpe Ratio,
   a. Called expected_annualized_return_my_port and expected_annualized_return_market prepared in Q2.2, beta from 1, and Risk free rate (Rf = 0.03) from global **(line 12)**
4. For Information Ratio,
   a. Prepare 'diff' pandas series by subtracting the pandas series of portfolio_daily_returns and market_daily_returns **(line 15)**
   b. Take the SD of the 'diff' by .std() and assign to Tracking_error **(line 16)**
   c. Call expected_annualized_return_my_port and expected_annualized_return_market prepared in Q2.2, working with Tracking_error to calculate Information Ratio 'IR_p' **(line 18)**
5. Summarize 4 Ratios into pandas dataframe with proper name and variables **(line 20 to line 23)**

```
1  """ Beta, Jensen's Alpha, Sharpe Ratio, Information Ratio"""
2  # Annualized covariance
3  covariance_annualized = np.cov(portfolio_daily_returns, market_daily_returns)[0, 1] * T
4
5  # Beta
6  Beta = covariance_annualized / annualized_var_market
7
8  # Jensen's Alpha (annualized)
9  Alpha = expected_annualized_return_my_port - (Rf + Beta * (expected_annualized_return_market - Rf))
10
11 # Sharpe Ratio
12 SR_p = (expected_annualized_return_my_port - Rf) / annualized_sd_my_port
13
14 # Information Ratio
15 diff = portfolio_daily_returns - market_daily_returns  # Now both series have aligned indices
16 Tracking_error = diff.std() * np.sqrt(T)  # Annualized tracking error
17
18 IR_p = (expected_annualized_return_my_port - expected_annualized_return_market) / Tracking_error
19
20 summary_ratio = pd.DataFrame({
21     'Measures': ['Beta', 'Jensen Alpha', 'Sharpe Ratio', 'Information Ratio'],
22     'Ratio': [Beta, Alpha, SR_p, IR_p]
23 })
24
```

**# 2 Prepare review note**

1. Print review notes “Summary Ratios” and the dataframe for review purpose **(line 1 to line 2)**

```
1  print("\nSummary Ratios:")
2  print(summary_ratio)
```

As per the result, below is my comments to each ratio:

| | |
|---|---|
| **Beta** | The portfolio-to-market beta is 1.0674. A greater than 1 beta indicates that the portfolio is more volatile than market. This figure aligns with the result in Q2.2 and the plot ‘Portfolio vs Market Performance (across 12 months, starting from $10,000 each)’. So, the portfolio implies a greater potential gain or loss than the market does. |
| **Jensen Alpha** | The Jensen Alpha is greater than 0 of my portfolio (= 0.162), which implies abnormal return and indicates that my portfolio outperformed the market for the selected 12 months period. This result aligns with Q2.2 and the plot ‘Portfolio vs Market Performance (across 12 months, starting from $10,000 each)’. So, if we invest wealth into my portfolio over the selected 12 months period, we beat the market down. |
| **Sharpe Ratio** | I observed a good Sharpe ratio 1.8496 from my portfolio. It implies that my portfolio generates 1.8496 times the level of risk taken for the selected 12 months period. This result aligns with Q2.2 and the plot ‘Portfolio vs Market Performance (across 12 months, starting from $10,000 each)’. |
| **Information Ratio** | The information ratio of my portfolio is 1.8354, which is considered as a good figure. This implied that the portfolio returns went beyond the market portfolio, in comparison to the level of risk for the selected 12 months period. This result aligns with Q2.2 and the plot ‘Portfolio vs Market Performance (across 12 months, starting from $10,000 each)’. |

**Concurrent control practice**

Bug fixing
Performed simultaneously with model development. Please refer the comments box/ # “”” Notes in both supporting and existing model.

Spreadsheet model

| Ratio analysis | |
|---|---|
| Cov(port, market) (daily) | 0.000055598 |
| Rf (given) | 0.03 |
| Tracking error (P-M) | 0.092257553 |
| | |
| Beta | 1.067468824 |
| Jensen's Alpha | 0.161349992 |
| Sharpe Ratio | 1.845155125 |
| Information Ratio | 1.831770176 |

The spreadsheet model consenses with python model. As discussed in Q2.2, the tiny discrepancy is believed to be the discrepancy of backend programming method of python function and excel function. The supporting model provided a fair comparison to existing model.

Fact check

According to the stock price graphs in Q2.1, these ratios are truly and fairly tied to the actual performance of each portfolio.

**Question 3: Target Date Capital Guarantee Fund (60 marks)**

You are an equity fund manager and want to launch a new equity fund product based on the ASX200 Index with a guarantee feature for retirees. The Initial purchase price is set at $1,000 per unit at the inception date. The fund will be closed to new customers post the inception date and only be paid out at the end of a 10-year term. No withdrawal is allowed after the inception date. Fund management fee is charged annually at each year end post the inception date, and there is no other fees or taxes. The performance (log-return) of the fund before will be based on the performance of the ASX200 Index at each year end, then less the fund management fee. A prominent feature of the fund is to pay out the maximum of year-end fund values (that is after fund management fee) at the end of 10-year term. Let's assume that the return of ASX200 Index (assuming no dividend payment) follows a Geometric Brownian Motion process with the below parameters: risk-free interest rate r=5% pa and volatility sigma=20% pa. To determine the annualised fund management fee, one possible approach is to assume that the expectation of present values of the final payoffs (the maximum of fund values at year ends less the terminal fund value) is equal to the expectation of present values of fund management fees charged annually, which can be solved numerically via optimisation. Let's set the tolerance level for the difference between the above two to $0.5 and assume that the discount rate for calculating present values is the risk-free interest rate. Please use Monte Carlo simulation (hint: use yearly time steps and no less than 10,000 simulations) and iteration methods (hint: set an iteration using a loop until the tolerance level $0.5 is met) to determine the annualised fund management fee (rounded to the nearest 0.01%).

Methodology:

Same practice as Q2. Below is the breakdown of **my management framework**.

| **My management framework** |
|---|
| *Planning* |

- In planning phrase, I need to ensure my understanding on the question is theoretically and practically correct

- Given the background information above, I summarized some points as motivation below:

    1. I am an equity fund manager, proposing to launch a new equity fund product based on the ASX200 index with a guarantee feature for retirees

    2. The product's features:

| | |
|---|---|
| Purchase price/unit | $1,000 |
| Risk-Free rate (= discount rate) | 5% |
| Span of fund | 10 years |
| Sigma (i.e. volatility) | 20% |

    3. My goal is to ensure the product will not expose to an extreme risk under the market dynamics

    4. As such, scenario testing is required. To most efficient way to test for the model is by Monte Carlo Simulation as requested by Q3

    5. To proper execute the Monte Carlo Simulation, below parameters are required:

| | |
|---|---|
| Tolerance | $0.5 |
| Simulation paths | 10,001 (as requested, no less than 10,000) |

    6. **Floor plan** of the algorithm will be presented and illustrated in section Q3.1

*Organizing*

- No supporting model is required

- Existing model is the python model

- Note that the iterative optimization could be supported by other optimization, after doing a few research, I decided to use scalar minimization ("Brent's method") as a supporting method to cross check the optimized data

- My theoretical understanding is below.

    1. Based on the theoretical understanding given, the *Fund Values Paths* can be modelled by GBM model below:

    $$dV = (r - f)Vdt + \sigma VdW$$

$$dlog(V) = \left(r - f - \frac{1}{2}\sigma^2\right)dt + \sigma dW$$

By discretisation, we have:

$$log\left(\frac{V_i+1}{V_i}\right) = \left(r - f - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\Delta W, i = 0, M-1$$
$$V_{i+1} = V_i e^{\left(r-f-\frac{1}{2}\sigma^2\right)\Delta t + \sigma\Delta}$$

where,
$V$: Fund value
$r$ : Risk-free rate
$f$ : fund total fee rate
$W$: Brownian Motion
$t$ : Time
σ : Volatility of fund value
$\Delta W = Z\sqrt{\Delta t}$ : Z is a standard normal random variable

2. The simulation should be performed on the *Fund Values Paths*. A proper simulation algorithm is required

3. Having the *Fund Values Paths*, we can step further to compute the present value of total fee and payoff in dollar:

$$PV_{fee}^{j} = \sum_{i=0}^{M} e^{-ir\Delta}\ fV_i^j \Delta t$$

$$PV_{payoff}^{j} = e^{-r}\ \left(max_{i\,=\,0,\ldots,M} V_i^j - V_m^j\right)$$

4. As a professional, I will perform **both numerical approach and iterative approach** for the optimization in the purpose of cross-checking. First, define the objective function below:

*Objective functions*
$$\left|\frac{1}{N}\sum_{j=1}^{N} PV_{fee}^{j} - \frac{1}{N}\sum_{j=1}^{N} PV_{payoff}^{j}\right| < Tolerance$$

Then, iterate those 2 means of present value until convergence. As a professional equity fund manager, I will present the iteration process including each the breakdown of each time of simulation and the number of iterations until convergence in purpose of further evaluating the tolerance level and the accuracy of fund fee optimization.

5. Display result for each time of iteration in the general function until convergence and enclose the result with the visualization output. The detail of optimization method in python will be introduced in section Q3.1

6. Visualize *Fund Values Paths* by plotting each simulation. As assumed, I am a professional equity fund manager, the *Fund Values Paths* should be added on some details such as a label on the graph showing features of fund paths and deterministic curves representing mean and SD of all *Fund Values Paths* for client pitch/presentation/further statistical analysis purpose

*Executing*

- Build the algorithm/model in accordance with the **floor plan** for the python model

- Walkthrough for the source code and spreadsheet model will be included in following sections accordingly

- Exercise and test the model with **concurrent control practice**. Concurrent control practice will be discussed in Controlling section

*Controlling*

**Concurrent control practice**

- This is a practice based on my personal experience, refers to monitor and regulate processes and operations in real time

- As this is a personal assignment, I am considered as an end-to-end full stacker. Concurrent control practice is the most efficient way to resolve any issue arises during the model development

- Controlling items included (but not limited to):
  1. Bug fixing
  2. Model optimization cross check
  3. Model Review
  4. Performance Indicators reasoning
  5. Reporting and review
  6. Attachment review (i.e. floor plan)

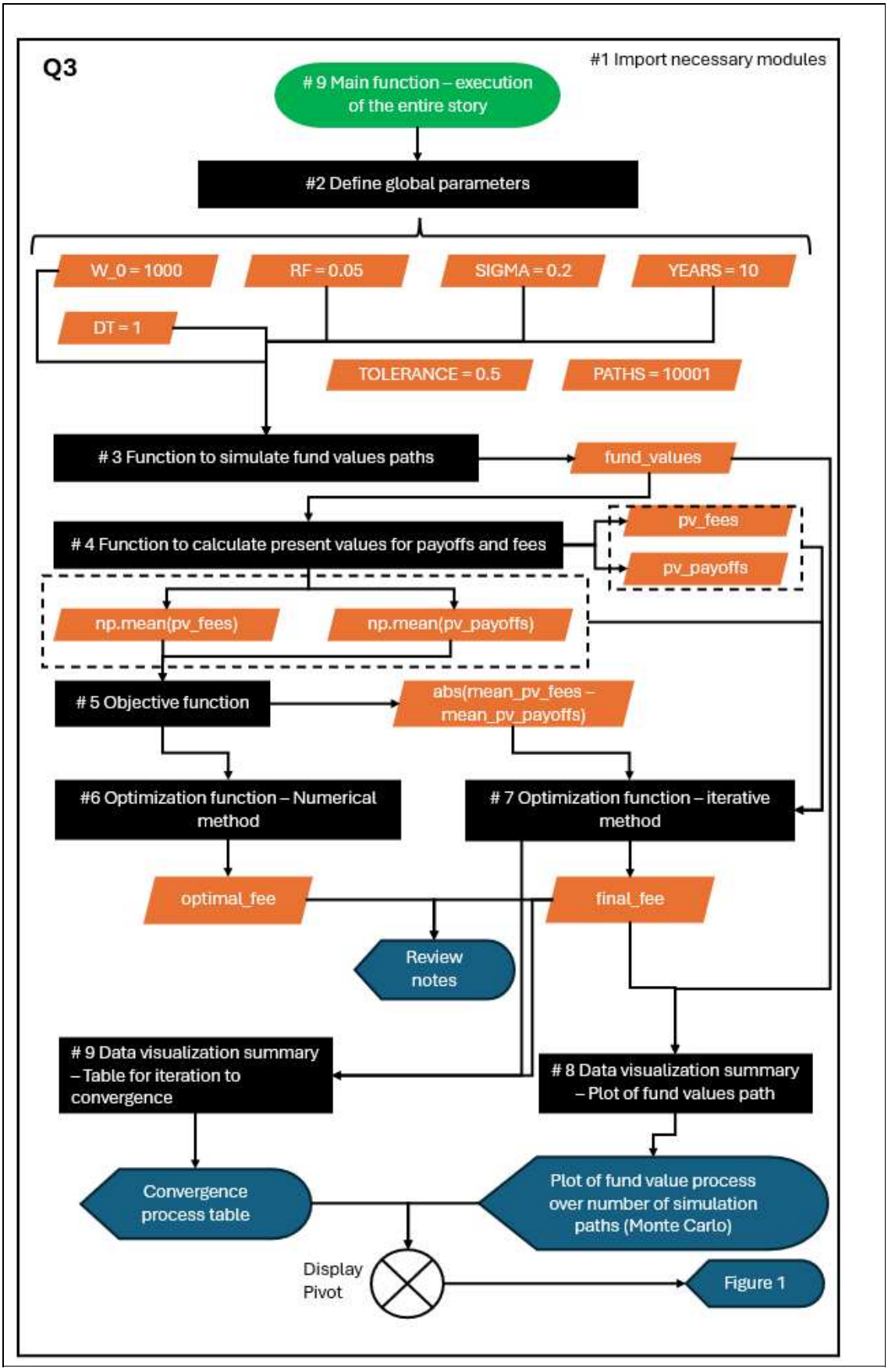
The **walkthrough of source code** and **designated result by model operation** will be discussed in the following section.

**Question 3.1:** Formulate the problem and present it in an algorithm or pseudo code. (20 marks)

According to my management framework, I summarized the theory behind this question and developed a floor plan for the design of algorithm. Below is my floor plan.

**Floor plan:**

As per the floor plan **<extract of Attach 03 - ASM_Q3_floor plan>**, we have the algorithm design below:


Q3
#1 Import necessary modules
# 9 Main function – execution of the entire story
#2 Define global parameters
W_0 = 1000
RF = 0.05
SIGMA = 0.2
YEARS = 10
DT = 1
TOLERANCE = 0.5
PATHS = 10001
# 3 Function to simulate fund values paths
fund_values
# 4 Function to calculate present values for payoffs and fees
pv_fees
pv_payoffs
np.mean(pv_fees)
np.mean(pv_payoffs)
# 5 Objective function
abs(mean_pv_fees – mean_pv_payoffs)
#6 Optimization function – Numerical method
# 7 Optimization function – iterative method
optimal_fee
final_fee
Review notes
# 9 Data visualization summary – Table for iteration to convergence
# 8 Data visualization summary – Plot of fund values path
Convergence process table
Plot of fund value process over number of simulation paths (Monte Carlo)
Display Pivot
Figure 1

As the same practice in Q2, below is the elaboration of my source code.

**<Attach 06 - ASM_2_Q3_v2 (submission)>**

**#1: Import necessary modules**

1. Import numpy for mathematical operation **(line 1)**
2. Import matplotlib.pyplot as plt for plotting the fund values paths **(line 2)**
3. Import pandas as pd for data analytics **(line 3)**
4. Call minimize_scalar function from the module scipy.optimize for optimization purpose **(line 4)**

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from scipy.optimize import minimize_scalar
```

**#2 Define global parameters**

1. W_0 = $1,000: Initial wealth **(line 2)**
2. RF = 5%: Risk-free rate (= discount rate) **(line 3)**
3. SIGMA = 20%: Given volatility **(line 4)**
4. YEARS = 10: Given fund span/ investment horizon to fund manager **(line 5)**
5. TOLERANCE = $0.5: Given tolerance level for iteration method **(line 6)**
6. PATHS = 10,001: Simulation paths required (> 10,000) **(line 7)**
7. DT = 1: Discretised time step **(line 8)**

```
1  # Global parameters - unified and optimized
2  W_0 = 1000              # Initial wealth
3  RF = 0.05               # Risk-free rate (unified with discount rate)
4  SIGMA = 0.20            # Fund SIGMA
5  YEARS = 10              # Investment horizon
6  TOLERANCE = 0.5         # Convergence tolerance
7  PATHS = 10001           # Number of Monte Carlo paths
8  DT = 1.0                # Time step (annual)
9
```

**# 3 Function to simulate fund values paths**

1. Define function: simulate_fund_paths() with parameters fee_rate and n_sims. Please note that the parameter **(line 2)**: fee_rate is set for "optimal_fee", where the optimal fee is generated by optimization function **#6**. Paths is set for simulation paths, which is globally defined to be 10,001

```
# Generate fund paths for visualization
sample_paths = simulate_fund_paths(optimal_fee, PATHS)
```

2. Locally, set seed for ensuing reproducibility **(line 4)**
3. Locally, define the random number generator $Z$ for discretised fund values path with size = (n_sims, YEARS) **(line 8)**

4. Locally, assigned a zero matrix to fund_values with the size (n_sims, YEARS +1) **(line 11)**, and assign all values under initial column to be initial wealth W_0 to ensure all simulation paths start at $1,000 **(line 12)**
5. Locally, iterate the fund paths based on
   a. $log\left(\frac{V_i+1}{V_i}\right) = \left(r - f - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\Delta W, i = 0, M-1$ **(line 16)**
   b. $V_{i+1} = V_i e^{\left(r-f-\frac{1}{2}\sigma^2\right)\Delta t + \sigma\Delta}$ **(line 17)**
6. Return fund_values paths **(line 19)**

```python
# Function to simulate fund values paths
def simulate_fund_paths(fee_rate, n_sims):

    np.random.seed(42)  # Ensure reproducibility

    # Generate independent standard normal random variables
    # Z_t ~ N(0,1) for t = 0, M-1 across N paths
    Z = np.random.standard_normal((n_sims, YEARS))

    # Initialize fund values matrix
    fund_values = np.zeros((n_sims, YEARS + 1))
    fund_values[:, 0] = W_0  # Initial value V_0 = W_0

    # Simulate paths using the discrete log-return model
    for t in range(YEARS):
        log_return = (RF - fee_rate - 0.5 * SIGMA**2) * DT + SIGMA * np.sqrt(DT) * Z[:, t]
        fund_values[:, t+1] = fund_values[:, t] * np.exp(log_return)

    return fund_values
```

**# 4 Function to calculate present values for payoffs and fees**

1. Define function: calculate_present_values() with parameters fee_rate. Please note that the parameter **(line 2)**: fee_rate is set for "optimal_fee", where the optimal fee is generated by optimization function **#6**. Paths is set for simulation paths, which is globally defined to be 10,001. See **#3**.
2. Call fund_values from **#2 (line 5)**, Call features of fund_values by .shape. Only n_sims is adopted in the local environment **(line 6)**.
3. Iterate the calculation of PV of fees **(line 8 to line 14)**. Denote a zero list for pv_fees with length = n_sims (line 9). Iterate the zero matrix with j for n_sims, i for YEARS and operate the formulae below inside the loop

$$PV_{fee}^{j} = \sum_{i=0}^{M} e^{-ir} \quad fV_i^j \Delta t$$

For discount factor: $e^{-ir\Delta t}$ **(line 12)**
For fee amount: $fV_i^j \Delta t$ **(line 13)**
For present value of fund fee j: $e^{-ir\Delta t} fV_i^j \Delta t$ **(line 14)**

4. Same practice as step 3. Iterate the calculation of PV of payoffs **(line 16 to line 22)** into zeros list pv_payoffs with length = n_sims **(line 17)**. Operation based on the formula:

$$PV_{payoff}^{j} = e^{-rT}\left(max_{i\,=\,0,\ldots,M} V_i^j - V_m^j\right)$$

5. Return mean of pv_fees and mean of pv_payoffs **(line 24)**

```python
# Function to calculate present_values for payoffs and fees
def calculate_present_values(fee_rate):

    # Call fund_values paths from simulate_fund_paths
    fund_values = simulate_fund_paths(fee_rate, PATHS)
    n_sims, _ = fund_values.shape

    # Calculate PV of fees
    pv_fees = np.zeros(n_sims)
    for j in range(n_sims):  # For each simulation path j
        for i in range(YEARS):  # For each time step i = 0 to M-1
            discount_factor = np.exp(-RF * i * DT)  # e^{-r·i·DT}
            fee_amount = fund_values[j, i] * fee_rate * DT  # V_i^j · f · DT
            pv_fees[j] += discount_factor * fee_amount

    # Calculate PV of payoffs
    pv_payoffs = np.zeros(n_sims)
    for j in range(n_sims):  # For each simulation path j
        max_value = np.max(fund_values[j, :])  # max_{i=0,...,M}V_i^j
        terminal_value = fund_values[j, -1]    # V_M^j
        payoff = max_value - terminal_value    # Guarantee benefit
        pv_payoffs[j] = np.exp(-RF * YEARS) * payoff  # e^{-r·T} · payoff

    return np.mean(pv_fees), np.mean(pv_payoffs), pv_fees, pv_payoffs
```

**# 5 Objective function**

1. Define function: objective_function() with parameters fee_rate. Please note that the parameter **(line 2)**: fee_rate is set for "optimal_fee", where the optimal fee is generated by optimization function **#6** (See **#3**).
2. Call mean_pv_fees and mean_pv_payoffs from **#4** **(line 5)**
3. Return the absolute differece between mean of pv_fees and mean of pv_payoffs **(line 6)**

```python
# Function to calculate the absolute difference between mean of pv_fees and mean of pv_payoffs
def objective_function(fee_rate):

    # Call mean of pv_fees and mean of pv_payoffs from function: calculate_present_values
    mean_pv_fees, mean_pv_payoffs, _, _ = calculate_present_values(fee_rate)
    return abs(mean_pv_fees - mean_pv_payoffs)
```

**#6 Optimization function – Numerical method**

1. Define function: find_optimal_fee_numerical() **(line 2)**
2. Locally, set up the display notes for review purpose **(line 5 to line 6)**
3. Locally, call the minimize_scalar() function from the module scipy.optimize and assign to local variable result with parameters **(line 8 to line 13)**:
   a. Objective funciton (parameter: callable function) **(line 9)**

b. bounds = (0.001, 0.1) (bonds: initial boundary for the roots searching) **(line 10)**
c. method = 'bounded' (as the 'bounds' is provided, method = 'bounded' refers to bounded method) **(line 11)**
d. options = {'xatol': 1e-6} (options is a dictionary parameters where xatol denotes the local tolerance and 1e-6 = 1 million denotes maximum number of iterations) **(line 12)**

4. Locally, assigned the root 'x' to 'result' **(line 15)**
5. Locally, assigned the minimum of objective function to 'min_difference' **(line 16)**
6. Locally, set up the display notes for 'Optimal fee' and 'Convergence achieved with difference' for review purpose **(line 18 to line 19)**
7. Return optimal_fee **(line 21)**

```
1  # Optimization function to perform numerical method with scalar minization
2  def find_optimal_fee_numerical():
3
4      # Display notes
5      print("=== NUMERICAL APPROACH ===")
6      print("Employing bounded scalar optimization method...")
7
8      result = minimize_scalar(
9          objective_function,
10         bounds=(0.001, 0.10),
11         method='bounded',
12         options={'xatol': 1e-6}
13     )
14
15     optimal_fee = result.x
16     min_difference = result.fun
17
18     print(f"Optimal fee rate: {optimal_fee:.4f} ({optimal_fee*100:.2f}%)")
19     print(f"Convergence achieved with difference: ${min_difference:.2f}")
20
21     return optimal_fee
```

**# 7 Optimization function – iterative method**

1. Define function: find_optimal_fee_iterative() **(line 2)**
2. Locally, call global covergence_data to local **(line 5)**. Note that this line of data is to fix the bug in function: visualization_summary()
3. Locally, set up the display notes for review purpose **(line 11 to line 12)**
4. Locally, set up bounds for iteration and maximum iteration time **(line 15 to line 17)**
5. Locally, reset covergence list locally for the own use of this function **(line 19)**
6. Locally, iterate the objective function until it reaches the tolerance with while-loop **(line 22 to line 48)**:
   a. Denote the initial parameters: $mid = \frac{(low+high)}{2}$ **(line 23)**
   b. Denote the difference to be objective function **(line 24)**
   c. Call mean_pv_fees and mean_pv_payoffs from #4 **(line 25)**

d. Store convergence data for visualization by .append with dictionary of the above defined local parameters and adding if/else logic as the condition for returning the 'Yes' and 'No' to the field: 'Convergence'. This is set for displaying whether the convergence is achieved for review purpose **(line 28 to line 35)**

e. Same fashion as bisection method, write up 2 if/else logic for the review notes **(line 37 to line 40)** and bisection method value update **(line 43 to line 48)**

f. Return final fee as the result of optimal fee by iterative approach **(line 50 to line 51)**

```python
# Optimization function to perform iterative method
def find_optimal_fee_iterative():

    # Call global empty list: covergence_data to local
    global convergence_data

    # Display notes
    print("\n=== ITERATIVE APPROACH ===")
    print("Implementing the bisection optimization method...")

    # Bounds for iteration and maximum iteration time
    low, high = 0.001, 0.10
    iteration = 0
    max_iterations = 50

    convergence_data = []  # Reset convergence list locally for the own use of this function

    # While loop to iterate the objective function until it reaches the tolerance
    while iteration < max_iterations:
        mid = (low + high) / 2
        difference = objective_function(mid)
        mean_pv_fees, mean_pv_payoffs, _, _ = calculate_present_values(mid)

        # Store convergence data for visualization
        convergence_data.append({
            'Iteration': iteration + 1,
            'Fee Rate (%)': mid * 100,
            'Difference ($)': difference,
            'PV Fees ($)': mean_pv_fees,
            'PV Payoffs ($)': mean_pv_payoffs,
            'Convergence': 'Yes' if difference <= TOLERANCE else 'No'
        })

        if difference <= TOLERANCE:
            print(f"Convergence achieved after {iteration + 1} iterations")
            print(f"Optimal fee rate: {mid:.4f} ({mid*100:.2f}%)")
            return mid

        # Bisection method value update
        if mean_pv_payoffs > mean_pv_fees:
            low = mid
        else:
            high = mid

        iteration += 1

    final_fee = (low + high) / 2
    return final_fee
```

**# 8 Data visualization summary – Plot of fund values path**

1. As planned, define function: visualization summary() with parameter 'optimal_fee' **(line 2)**
2. Locally, storing fund paths for visualization by calling **#3** into sample_paths **(line 5)**
3. Locally, create the main visualization 'fig' by assigning the plt.figure() with size 16 x 12 **(line 8)**
4. Locally, plot the fund values paths **(line 11)** and apply np.arrange to return evenly spaced values within a given interval **(line 13)**
5. Locally iterate the plotting of standalone sample_paths with proper parameters **(line 16 to line 17)**
6. As a professional, I prefer to locally plot the mean of fund values paths and standard deviation of fund values paths along with the individual fund values paths plotting for presentation purpose, say, presenting the confidence interval for the mean of fund values process to demonstrate the profitability of product to increae the convincing power **(line 19 to line 27)**
7. Locally, set title, labels and format the graphs **(line 23 to line 37)**
8. Locally, add statistical annotation by creating a little tag on the left top of the graph for easy understanding **(line 39 to line 43)**
9. It is noteworthy that the optimal fund fee rate, risk-free rate and sigma are presented simultaneously on top of the graph **(line 31 to line 34)**

```python
# Consolidated data visualization
def visulization_summary(optimal_fee):

    # Generate fund paths for visualization
    sample_paths = simulate_fund_paths(optimal_fee, PATHS)

    # Create the main visualization
    fig = plt.figure(figsize=(16, 12))

    # Main plot: Fund paths
    ax_main = plt.subplot2grid((3, 2), (0, 0), colspan=2, rowspan=2)

    years = np.arange(0, YEARS + 1)

    # Plot sample paths with better visibility
    for i in range(sample_paths.shape[0]):
        ax_main.plot(years, sample_paths[i, :], alpha=0.3, color='green', linewidth=1.2)

    # Plot average path and confidence intervals using dotted lines
    mean_path = np.mean(sample_paths, axis=0)
    std_path = np.std(sample_paths, axis=0)

    ax_main.plot(years, mean_path, color='red', linewidth=3, label='Expected Path', zorder=10)
    ax_main.plot(years, mean_path + std_path, color='red', linewidth=2, linestyle='--',
                 label='+1σ Confidence', alpha=0.8, zorder=9)
    ax_main.plot(years, mean_path - std_path, color='red', linewidth=2, linestyle='--',
                 label='-1σ Confidence', alpha=0.8, zorder=9)

    ax_main.set_xlabel('Years', fontsize=12, fontweight='bold')
    ax_main.set_ylabel('Fund Value ($)', fontsize=12, fontweight='bold')
    ax_main.set_title(f'ASX200 Fund Value Simulation Paths\n'
                      f'Management Fee: {optimal_fee*100:.2f}% p.a. | Risk-Free Rate: {RF*100:.1f}% | '
                      f'SIGMA: {SIGMA*100:.1f}%',
                      fontsize=14, fontweight='bold', pad=20)
    ax_main.legend(loc='upper left', fontsize=10, framealpha=0.9)
    ax_main.grid(True, alpha=0.3)
    ax_main.set_xlim(0, YEARS)

    # Add statistical annotations with better positioning
    final_stats = f'Initial: ${W_0:,}\nExpected Terminal: ${mean_path[-1]:,.0f}\nPaths: {sample_paths.shape[0]:,}'
    ax_main.text(0.02, 0.85, final_stats, transform=ax_main.transAxes,
                 verticalalignment='top', fontsize=9,
                 bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.9))
```

**# 9 Data visualization summary – Table for iteration to convergence**

1. This section is continuing the operation inside the function of visualization summary(), so, locally;
    a. Plot the table for presenting the convergence process **(line 2)**. As this is a table, I turned off the axis **(line 3).**
    b. Dataframe the global convergence data with pandas module as df_convergence **(line 6)**
    c. As iteration process can take so many steps (e.g. > 15), I set the if/else logic that if length of df_convergence is greater than 7, show only last 7 rows of the process, else show the full legth of rows and finally assigned the outcome to display_df **(line 9)**
    d. For the table formatting, iterate the headings for the table, setting of the decimal places and units for all underlying figures under particular headings to table_data **(line 11 to line 21)**
    e. Headers is a list contained all headers to presenting table **(line 23)**
    f. Construct table object in python with calling objects from d and e **(line 25 to line 29)**
    g. Formatting for table font size and scale **(line 31 to line 33)**
    h. Set title for the table ‘Covergence process’ with proper formatting **(line 35 to 36)**
    i. Control the layout and show the outcome **(line 38 to line 39)**

j. Return a dictionary type data for ‘optimal fee’ and ‘convergence_iterations’ **(line 41 to line 44)**

```
# Convergence table
ax_table = plt.subplot2grid((3, 2), (2, 0), colspan=2)
ax_table.axis('off')

# Create convergence DataFrame
df_convergence = pd.DataFrame(convergence_data)

# Display last 10 iterations for clarity
display_df = df_convergence.tail(7) if len(df_convergence) > 7 else df_convergence

# Format the table
table_data = []
for _, row in display_df.iterrows():
    table_data.append([
        f"{int(row['Iteration'])}",
        f"{row['Fee Rate (%)']:.3f}%",
        f"${row['Difference ($)']:.2f}",
        f"${row['PV Fees ($)']:.2f}",
        f"${row['PV Payoffs ($)']:.2f}",
        row['Convergence']
    ])

headers = ['Iter_num', 'Fee Rate', 'Abs Difference', 'PV Fees', 'PV Payoffs', 'Converged']

table = ax_table.table(cellText=table_data,
                       colLabels=headers,
                       cellLoc='center',
                       loc='center',
                       colWidths=[0.08, 0.12, 0.15, 0.15, 0.15, 0.12])

table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1, 2)

ax_table.set_title('Convergence process',
                   fontsize=11, fontweight='bold', pad=10)

plt.tight_layout()
plt.show()

return {
    'optimal_fee': optimal_fee,
    'convergence_iterations': len(convergence_data)
}
```

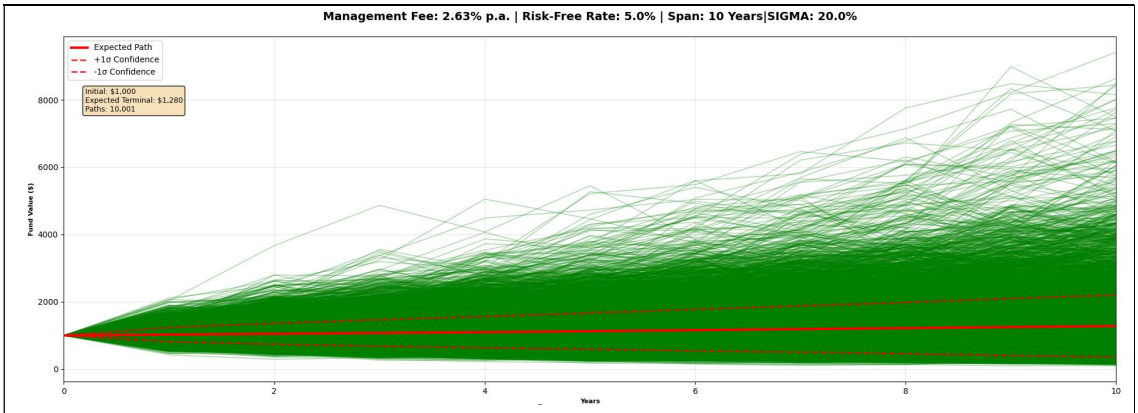
**# 9 Main function – execution of the entire story**

1. To execute all of the functions above to figure out the optimal management fee and plot the graph for presentation purpose, function: main() is defined (line 1)
2. Additionally, review notes are created for improving readability of output **(line 5 to line 14; line 21 to line 24; line 28; line 33 to line 34)**
3. To execute numerical and iterative optimization for fund fee calculation, call find_optimal_fee_numerical() and find_optimal_fee_iterative() and assign them to corresponding local variables **(line 17 to line 18)**

4. I also work on the result comparison for these 2 optimization method **(line 21 to line 24)** and summarize the final fee for optimal fund fee as in iterative method as requested in notes **(line 28)**
5. Then, put the final_fee as parameter into visualization_summary() the obtain the local variable "results" **(line 31)**
6. Final review notes for 'Optimization completed successfully' **(line 33)** and show the times used to achieve convergence **(line 34)**
7. For future development, I would safeguard the main() by if__name__=="__main__": such that any other .py file would not execute main function when calling this .py file **(line 36 to line 37)**

```python
def main():
    """
    Main execution function implementing the complete fee optimization workflow.
    """
    print("ASX200 EQUITY FUND MANAGEMENT FEE OPTIMIZATION")
    print("=" * 60)

    print(f"\nModel Parameters:")
    print(f"• Initial Fund Value: ${W_0:,}")
    print(f"• Risk-Free Rate: {RF*100:.1f}% p.a.")
    print(f"• SIGMA: {SIGMA*100:.1f}% p.a.")
    print(f"• Investment Horizon: {YEARS} years")
    print(f"• Convergence Tolerance: ${TOLERANCE}")
    print(f"• Monte Carlo Paths: {PATHS:,}")

    # Execute both optimization approaches
    optimal_fee_numerical = find_optimal_fee_numerical()
    optimal_fee_iterative = find_optimal_fee_iterative()

    # Results comparison and validation
    print(f"\n=== RESULTS ANALYSIS ===")
    print(f"Numerical method: {optimal_fee_numerical*100:.3f}% p.a.")
    print(f"Iterative method: {optimal_fee_iterative*100:.3f}% p.a.")
    print(f"Method Convergence: {abs(optimal_fee_numerical - optimal_fee_iterative)*100:.4f}%")

    # Final result selection
    final_fee = round(optimal_fee_numerical * 100, 2) / 100
    print(f"\nOPTIMAL MANAGEMENT FEE: {final_fee*100:.2f}% per annum")

    # Generate comprehensive visualization
    results = visulization_summary(final_fee)

    print(f"\nOptimization completed successfully.")
    print(f"Convergence achieved in {results['convergence_iterations']} iterations.")

if __name__ == "__main__":
    main()
```

**Question 3.2:** Find the annualised fund management fee (rounded to the nearest 0.01%). (10 marks)




**<extract of Attach 08 - Plotting for fund values path, convergence process & fund fee>**

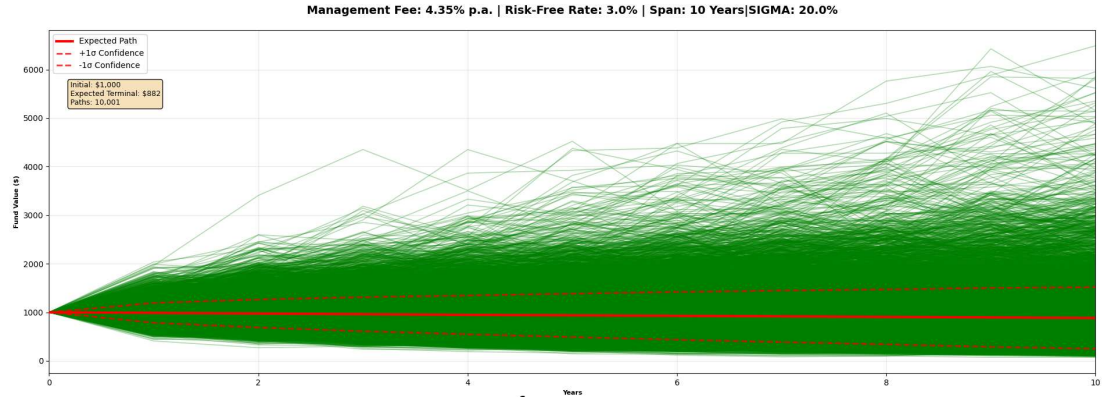
Taking all the given (original) parameters below, the fund fee converges to 2.63% by iterative method after 9 times of iteration. Numerical method results in 2.64%, supported the figure from iterative method. If the initial wealth is $1,000, the management fee for the fund is $\$1{,}000(0.0264) = \$26.4$ per unit.

**Convergence process**

| Iter_num | Fee Rate | Abs Difference | PV Fees | PV Payoffs | Converged |
|---|---|---|---|---|---|
| 3 | 3.813% | $69.96 | $324.14 | $254.18 | No |
| 4 | 3.194% | $34.31 | $278.72 | $244.41 | No |
| 5 | 2.884% | $15.46 | $255.06 | $239.59 | No |
| 6 | 2.730% | $5.78 | $242.98 | $237.20 | No |
| 7 | 2.652% | $0.87 | $236.88 | $236.02 | No |
| 8 | 2.614% | $1.61 | $233.81 | $235.42 | No |
| 9 | 2.633% | $0.37 | $235.35 | $235.72 | Yes |

**<extract of Attach 08 - Plotting for fund values path, convergence process & fund fee>**

**Question 3.3:** If the risk-free interest rate (and the discount rate) decreases to 3% pa, what is the annualised fund management fee assuming all other parameters are unchanged? (10 marks)




**<extract of Attach 09 – Q3.3>**

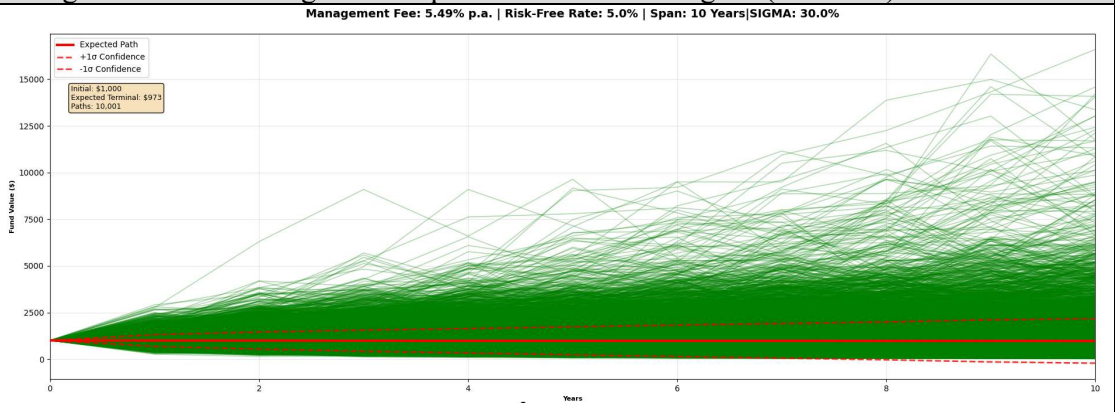
If the risk-free rate changes to 3% p.a., assuming all other parameters unchanged, the fund fee will increase to 4.35% by iterative method after 7 times of iteration. Numerical method results in 4.36%, supported the figure from iterative method. If the initial wealth is $1,000, the management fee for the fund is $\$1{,}000(0.0436) = \$43.6$ per unit.

**Convergence process**

| Iter_num | Fee Rate | Abs Difference | PV Fees | PV Payoffs | Converged |
|---|---|---|---|---|---|
| 1 | 5.050% | $31.20 | $407.89 | $376.69 | No |
| 2 | 2.575% | $94.75 | $230.74 | $325.49 | No |
| 3 | 3.813% | $26.49 | $324.14 | $350.63 | No |
| 4 | 4.431% | $3.53 | $367.15 | $363.62 | No |
| 5 | 4.122% | $11.17 | $345.94 | $357.11 | No |
| 6 | 4.277% | $3.75 | $356.62 | $360.36 | No |
| 7 | 4.354% | $0.09 | $361.90 | $361.99 | Yes |

**<extract of Attach 09 – Q3.3>**

**Question 3.4:** If the assumed volatility increases to 30% pa, what is the annualised fund management fee assuming all other parameters are unchanged? (10 marks)




**<extract of Attach 10 – Q3.4>**

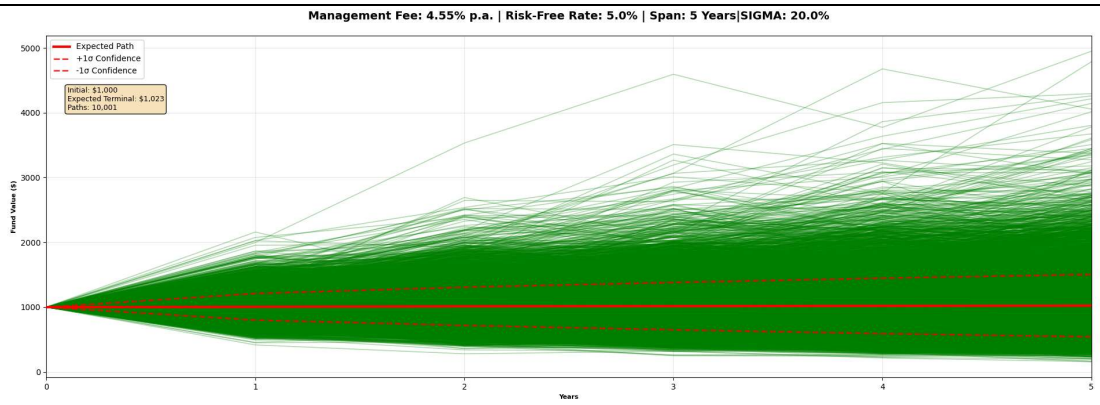
If the volatility changes to 30% p.a., assuming all other parameters unchanged, the fund fee will increase to 5.49% by iterative method after 10 times of iteration. Numerical method results in the same figure, supported the figure from iterative method. If the initial wealth is \$1,000, the management fee for the fund is $\$1{,}000(0.0549) = \$54.9$ per unit.

**Convergence process**

| Iter_num | Fee Rate | Abs Difference | PV Fees | PV Payoffs | Converged |
|---|---|---|---|---|---|
| 4 | 5.669% | $9.65 | $448.21 | $438.56 | No |
| 5 | 5.359% | $6.71 | $429.11 | $435.82 | No |
| 6 | 5.514% | $1.54 | $438.72 | $437.19 | No |
| 7 | 5.437% | $2.57 | $433.93 | $436.50 | No |
| 8 | 5.475% | $0.51 | $436.33 | $436.85 | No |
| 9 | 5.495% | $0.51 | $437.53 | $437.02 | No |
| 10 | 5.485% | $0.00 | $436.93 | $436.93 | Yes |

**<extract of Attach 10 – Q3.4>**

**Question 3.5:** If the term of the fund decreases to 5 years, what is the annualised fund management fee assuming all other parameters are unchanged? (10 marks)




**<extract of Attach 11 – Q3.5>**

If the total span of fund decreases to 5 years, assuming all other parameters unchanged, the fund fee will increase to 4.55% by iterative method after 8 times of iteration. Numerical method results in the 4.56%, supported the figure from iterative method. If the initial wealth is \$1,000, the management fee for the fund is $\$1{,}000(0.0455) = \$45.5$ per unit.

**Convergence process**

| Iter_num | Fee Rate | Abs Difference | PV Fees | PV Payoffs | Converged |
|---|---|---|---|---|---|
| 2 | 2.575% | $59.52 | $122.54 | $182.06 | No |
| 3 | 3.813% | $21.37 | $177.13 | $198.50 | No |
| 4 | 4.431% | $3.56 | $203.45 | $207.01 | No |
| 5 | 4.741% | $5.06 | $216.37 | $211.31 | No |
| 6 | 4.586% | $0.77 | $209.93 | $209.16 | No |
| 7 | 4.509% | $1.39 | $206.69 | $208.08 | No |
| 8 | 4.547% | $0.31 | $208.31 | $208.62 | Yes |

**<extract of Attach 11 – Q3.5>**

## List of Attachments

**<Attach 01 - Certificate_Bloomberg_mkt_concept_1>**
**< Attach 02 - ASM_Q2_floor plan>**
**<Attach 03 - ASM_Q3_floor plan>**
**< Attach 04 - Q2 - plot of port vs mkt>**
**< Attach 05 - ASM_2_Q2_v5 (submission)>**
**< Attach 06 - ASM_2_Q3_v2 (submission)>**
**< Attach 07 - Spreadsheet model>**
**< Attach 08 - Plotting for fund values path, convergence process & fund fee>**
**< Attach 09 - Q3.3>**
**< Attach 10 - Q3.4>**
**< Attach 11 - Q3.5>**

**Source data for Q2:**

1. ANZ.csv
2. CBA.csv
3. NAB.csv
4. WBC.csv
5. ASX_200.csv