In [1]:
```python
#These are the libraries you can use.  You may add any libraries directy related to threading if this is a direction
#you wish to go (this is not from the course, so it's entirely on you if you wish to use threading).  Any
#further libraries you wish to use you must email me, james@uwaterloo.ca, for permission.

from IPython.display import display, Math, Latex

import pandas as pd
import numpy as np
import numpy_financial as npf
import yfinance as yf
import matplotlib.pyplot as plt
import random
from datetime import datetime
```

Group Assignment

Team Number: 09

Team Member Names: Jacob, William, Michael

Team Strategy Chosen: Market Beat

In [2]:
```python
# Constants
start_date = "2023-10-01"
end_date = "2024-09-30"

# Load tickers from CSV
tickers = pd.read_csv('Tickers_Example.csv', header=None)
tickers_list = tickers[0].tolist()

# Function to validate tickers
def validate_tickers(tickers, start_date, end_date):

    # Download data for all tickers
    all_data = yf.download(tickers, start=start_date, end=end_date, group_by='ticker', auto_adjust=True)

    valid_tickers = []

    for ticker in tickers:
        try:
            # Access individual ticker data
            ticker_data = all_data[ticker]
            ticker_data.index = pd.to_datetime(ticker_data.index)

            # Resample to monthly and count trading days
            month_day_count = ticker_data['Volume'].resample('ME').count()
            valid_months = month_day_count[month_day_count >= 18]  # Months with at least 18 trading days

            # Filter the data for valid months
            ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]

            # Check currency and average volume
            ticker_obj = yf.Ticker(ticker)
            average_volume = ticker_data['Volume'].mean()
            if ticker_obj.fast_info['currency'] in ['CAD', 'USD'] and average_volume >= 100000:
                valid_tickers.append(ticker)
        except Exception as e:
            print(f"Error processing ticker {ticker}: {e}")

    return valid_tickers

# Validate tickers
valid_tickers = validate_tickers(tickers_list, start_date, end_date)

# Printing the DataFrame
valid_tickers
```

```
[*********************100%***********************]  41 of 41 completed

4 Failed downloads:
['RTN', 'AGN', 'CELG', 'MON']: YFTzMissingError('$%ticker%: possibly delisted; no timezone found')
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
$AGN: possibly delisted; no price data found  (period=5d) (Yahoo error = "No data found, symbol may be delisted")
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
Error processing ticker AGN: 'currency'
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
$CELG: possibly delisted; no price data found  (period=5d) (Yahoo error = "No data found, symbol may be delisted")
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
Error processing ticker CELG: 'currency'
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
$MON: possibly delisted; no price data found  (period=5d) (Yahoo error = "No data found, symbol may be delisted")
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone information.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
Error processing ticker MON: 'currency'
```

```
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
$RTN: possibly delisted; no price data found  (period=5d) (Yahoo error = "No data found, symbol may be delisted")
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
Error processing ticker RTN: 'currency'
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
C:\Users\jimic\AppData\Local\Temp\ipykernel_9652\2265284816.py:28: UserWarning: Converting to PeriodArray/Index representation will drop timezone in
formation.
  ticker_data = ticker_data[ticker_data.index.to_period('M').isin(valid_months.index.to_period('M'))]
```

Out[2]:

```
['AAPL',
 'ABBV',
 'ABT',
 'ACN',
 'AIG',
 'AMZN',
 'AXP',
 'BA',
 'BAC',
 'BB.TO',
 'BIIB',
 'BK',
 'BLK',
 'BMY',
 'C',
 'CAT',
 'CL',
 'KO',
 'LLY',
 'LMT',
 'MO',
 'MRK',
 'PEP',
 'PFE',
 'PG',
 'PM',
 'PYPL',
 'QCOM',
 'RY.TO',
 'SHOP.TO',
 'T.TO',
 'TD.TO',
 'TXN',
 'UNH',
 'UNP',
 'UPS',
 'USB']
```

In [3]:

```
rating_list = []

for ticker in valid_tickers:
    try:
        # Get the ticker object and fetch data
        Ticker = yf.Ticker(ticker)
```

```
        info = Ticker.info

        # Getting Beta
        beta = info.get('beta')

        # Get the historical data
        hist = Ticker.history(period="1y", interval="1d")

        # Drop rows with missing 'Close' prices
        if 'Close' in hist.columns:
            hist = hist.dropna(subset=['Close'])
        else:
            raise ValueError(f"Missing 'Close' column for ticker {ticker}")

        # Check if the cleaned data is still non-empty
        if not hist.empty:

            # Calculate standard deviation, Sharpe ratio
            std = hist['Close'].std()
            avg_daily_returns = hist['Close'].pct_change().mean()
            expected_return = (1 + avg_daily_returns) ** 252 - 1
            sharpe = ((expected_return - 0.04) / std) * 100

        else:
            # If no data remains after dropping missing rows, set to None
            std = None
            sharpe = None

        # Append results to the beta list
        rating_list.append([ticker, beta, std, sharpe])
    except Exception as e:
        print(f"Error processing ticker {ticker}: {e}")
        continue

# Create a DataFrame
calculation_df = pd.DataFrame(rating_list, columns=['Ticker', 'Beta', 'Standard Deviation', 'Sharpe']).set_index('Ticker')

# Remove rows with missing values
calculation_df = calculation_df.dropna()

# Compute the sum of Beta, Standard Deviation, and Sharpe
beta_sum = calculation_df['Beta'].sum()
std_sum = calculation_df['Standard Deviation'].sum()
sharpe_sum = calculation_df['Sharpe'].sum()

# Calculate the Rating
calculation_df['Rating'] = (
    (calculation_df['Beta'] / beta_sum) * 0.4 +
    (calculation_df['Standard Deviation'] / std_sum) * 0.2 +
    (calculation_df['Sharpe'] / sharpe_sum) * 0.4)

# Sort by Rating in descending order and keep only the top 12 rows
sorted_df = calculation_df.sort_values(by='Rating', ascending=False).head(12)

# Printing the sorted DataFrame
sorted_df
```

Out[3]:

	Beta	Standard Deviation	Sharpe	Rating
Ticker				
BAC	1.325	4.063192	15.583447	0.164350
USB	1.040	3.505797	14.273904	0.148420
C	1.426	5.844395	10.697581	0.119810
BK	1.060	8.758190	8.316577	0.093769
MO	0.670	5.649239	8.595692	0.090904
PYPL	1.436	7.959492	7.332063	0.088677
SHOP.TO	2.365	13.943511	5.103519	0.080295
BMY	0.441	4.507665	5.717106	0.060621
AIG	1.069	4.168921	4.354337	0.055065
AXP	1.214	32.386183	2.682409	0.049002
BLK	1.311	85.989397	0.520496	0.044968
RY.TO	0.842	16.602041	3.012270	0.043218

The overall strategy exhibited by the portfolio is based on that of momentum investing, in which stocks which have increased in value over a recent period are invested in. This approach is based on the premise that stocks which have recently performed well are likely to continue their upwards trajectory. Despite its inherent risk, the strategy is frequently used by existing algorithmic trading models.

After filtering out stocks that don't meet the given requirements, key properties on each one are computed. First, the beta of the stock, which measures volatility relative to the market, is retrieved, as well as the stock's standard deviation. Sharpe Ratio, a comparison of return relative to risk, is also calculated. For calculating

Sharpe ratio, risk-free returns were estimated at 4%; this was based on the benchmark yield of a 10-year Canadian government bond, which as of November 2024 was around 3.3%. The 4% figure was used to prioritize stocks with high expected returns.

A few strict requirements are imposed on stocks in order for them to be considered for the final portfolio. First, tickers with a beta value below 1 were eliminated, in order to eliminate low-volatility stocks that might not align with the goal of maximizing returns. Second, tickers with a negative Sharpe ratio are removed, effectively eliminating any stock with an expected return below 4%. This ensures that all selected stocks trend upwards and exhibit positive momentum.

For stocks which meet these requirements, Beta, standard deviation, and Sharpe ratio are all weighed at 40%, 20%, and 40% respectively and summed up to provide an overall rating for each stock. To achieve this, all three properties are standardized relative to the rest of the stocks in portfolio in order to provide for a balanced rating.

In [4]:
```
# Fetch market cap data using yfinance
market_caps = []
for ticker in sorted_df.index:
    market_caps.append(yf.Ticker(ticker).info['marketCap'])
sorted_df['Market Cap'] = market_caps



# Normalize the ratings to a 0-1 scale
sorted_df['Normalized Rating'] = (sorted_df['Rating'] - sorted_df['Rating'].min()) / (sorted_df['Rating'].max() - sorted_df['Rating'].min())

# Normalize market cap to a 0-1 scale
sorted_df['Normalized Market Cap'] = (sorted_df['Market Cap'] - sorted_df['Market Cap'].min()) / (sorted_df['Market Cap'].max() - sorted_df['Market

# Calculate the combined weight: 70% rating, 30% market cap
sorted_df['Ultimate Rating'] = 0.7 * sorted_df['Normalized Rating'] + 0.3 * sorted_df['Normalized Market Cap']


# Calculate initial weighting based on our ultimate rating (disregarding constraints)
sorted_df['Initial Weight'] = sorted_df['Ultimate Rating']/sorted_df['Ultimate Rating'].sum()


# The constraints as listed in assignment
min_weight = 100/(2*len(sorted_df))/100
max_weight = 0.15

# Redistributing weights until they satisfy
while True:

    # All tickers that are above weigh limit, are set to the upper weight limit, all that are below, are set to the lower weight limit
    sorted_df['Adjusted Weight'] = np.clip(sorted_df['Initial Weight'], min_weight, max_weight)


    # Check if total weight sums up to 1
    total_weight = sorted_df['Adjusted Weight'].sum()
    remaining_weight = 1 - total_weight

    # If remaining weight is 0 then stop because all tickers are finished weighing
    if abs(remaining_weight) == 0:
        break

    # Get all tickers that are below upper weght limit
    below_max = sorted_df['Adjusted Weight'] < max_weight
    total_below_max = sorted_df.loc[below_max, 'Adjusted Weight'].sum()


    # Check if there are any stocks with Adjusted Weight below the maximum allowed weight
    if total_below_max > 0:

    # Calculate how much weight to redistribute to each stock with weight below max_weight
    # Proportionally distribute the remaining weight based on the current adjusted weight of each stock
        redistribution = remaining_weight * (sorted_df.loc[below_max, 'Adjusted Weight'] / total_below_max)

    # Add the calculated redistribution amount to the Initial Weight of each stock
    # This increases the weight of each eligible stock proportionally
        sorted_df.loc[below_max, 'Initial Weight'] += redistribution
    else:
        break  # No stocks available to redistribute

# Ensure weights sum to 1
sorted_df['Final Weight'] = sorted_df['Adjusted Weight'] / sorted_df['Adjusted Weight'].sum()

# Display results
print(f"The summed up weight is {sorted_df['Final Weight'].sum()}")
sorted_df[['Rating', 'Final Weight']]
```

```
The summed up weight is 1.0
```

Out[4]:

Ticker	Rating	Final Weight
BAC	0.164350	0.150000
USB	0.148420	0.150000
C	0.119810	0.137149
BK	0.093769	0.079166
MO	0.090904	0.084394
PYPL	0.088677	0.078624
SHOP.TO	0.080295	0.092745
BMY	0.060621	0.044396
AIG	0.055065	0.041667
AXP	0.049002	0.049978
BLK	0.044968	0.041667
RY.TO	0.043218	0.050215

One last variable is introduced in order to create a final rating. Market capitalization, the total value of a company's outstanding shares on the stock market, is standardized to be in the range of 0 to 1 and weighted at 30% of the final rating (thereby reducing the actual weights of the previous variables). Generally, higher market-cap companies enjoy greater financial security and are less speculative in nature, making their historical performance a more reliable metric. This factor was introduced in order to add a measure of stability to the portfolio without sacrificing significant profitability.

Although it may seem precarious or incomprehensive to base each stock's rating on only four criteria, the strength of the formula is ultimately derived from its simplicity. By focusing on a few critical metrics rather than overfitting the model with too many variables, the formula ensures that weight is only given to impactful factors and statistical noise is averted.

After each stock has been given its final rating, its weight in the portolfio is based on the proportion of its rating to the combined rating of the top twelve stocks, although the final weight also takes into account the upper and lower limits on the proportion of each stock in the portfolio.

The decision to only include the top twelve stocks wasn't taken lightly. However, prioritizing concentration over diversification was necessary in order to ensure that the porfolio's capital was allocated solely to high-potential investments. Although diversified portfolios experience less risk, gains are often diluted by low-performing stocks. This occurence was observable in assignment 3, in which the inclusion of sectors outside of the tech industry (which is known for high risk and growth potential) greatly reduced the overall returns of a portfolio. The benefits of concetrated portfolios have been emphasized by renowned investors such as Warren Buffet, who explained his rationale with the following analogy: "If you have Lebron James on your team, don't take him out of the game just to make room for someone else... It's crazy to put money into your 20th choice rather than your first choice." Transaction fees are also lowered, although this benefit is more marginal.

In [13]:

```
buy_date = '2024-11-22'

# Creating the Final DataFrame
Portfolio_Final = pd.DataFrame()
Portfolio_Final['Ticker'] = sorted_df.index
Portfolio_Final.index = Portfolio_Final.index + 1 # Adjusting the index

# Empty arrays to add to the Final Portfolio later on
currency = []
price = []
weight = []
money = 1000000
shares = []

# Getting the exchange rate
exchangeRate = yf.Ticker('USDCAD=x').history(start=buy_date)['Close']

for ticker in Portfolio_Final['Ticker']:
    ticker_obj = yf.Ticker(ticker)

    # Adjusting stock prices to CAD
    if ticker_obj.fast_info['currency'] == 'USD':
        ticker_price = ticker_obj.history(start=buy_date)['Close'].iloc[0] * exchangeRate.iloc[0]
    else:
        ticker_price = ticker_obj.history(start=buy_date)['Close'].iloc[0]

    # Price
    price.append(ticker_price)

    # Adjusting for Fees
    moneyAllocated = sorted_df.loc[ticker, 'Final Weight'] * 1000000
    buy_price = ticker_price
    quantity = moneyAllocated / buy_price
    if quantity > 3950:
        fee = 3.95
    else:
        fee = quantity * 0.001

    # Number of Shares
    actualQuantity = (moneyAllocated - fee) / buy_price

    # Shares
```

```python
    shares.append(actualQuantity)

    # Currency
    currency.append(ticker_obj.fast_info['currency'])

# Adding everything to the Final Portfolio
Portfolio_Final['Price'] = price
Portfolio_Final['Currency'] = currency
Portfolio_Final['Shares'] = shares
Portfolio_Final['Value'] = Portfolio_Final['Shares']*Portfolio_Final['Price']
Portfolio_Final['Weight'] = sorted_df['Final Weight'].values

print("The final portfolio's value is $" + str(Portfolio_Final['Value'].sum()) + " after fees.")
Portfolio_Final
```

```
The final portfolio's value is $999989.8604467035 after fees.
```

Out[13]:

	Ticker	Price	Currency	Shares	Value	Weight
1	BAC	65.689080	USD	2283.449815	149997.716515	0.150000
2	USB	73.362125	USD	2044.623905	149997.955348	0.150000
3	C	97.611172	USD	1405.035289	137147.140966	0.137149
4	BK	112.006868	USD	706.791578	79165.510925	0.079166
5	MO	79.288116	USD	1064.386559	84393.204975	0.084394
6	PYPL	121.273217	USD	648.316452	78623.421867	0.078624
7	SHOP.TO	149.479996	CAD	620.443876	92743.948003	0.092745
8	BMY	82.279065	USD	539.572035	44395.482376	0.044396
9	AIG	106.304494	USD	391.952147	41666.274711	0.041667
10	AXP	421.108912	USD	118.681024	49977.636993	0.049978
11	BLK	1448.597889	USD	28.763426	41666.637903	0.041667
12	RY.TO	174.710007	CAD	287.418739	50214.929864	0.050215

In [6]:

```python
# Outputting the Final Portfolio to CSV
Stocks_Final = pd.concat([Portfolio_Final['Ticker'], Portfolio_Final['Shares']], axis=1)
Stocks_Final.to_csv('Stocks_Group_09.csv', index=False)
```

Contribution Declaration

The following team members made a meaningful contribution to this assignment:

Insert Names Here: Jacob, William, Michael