Programming for Financial Data Science Assignment 4

Author: Gavin Connolly
Student Number: 18308483

Module: Programming for Financial Data Science

Academic Year - 2022/23
Submitted: 2nd May, 2023

Question 1:

```python
import yahooquery as yq
import pandas as pd
import numpy as np
from splinter import Browser
import datetime as dt
import time
from scipy.stats import norm

# Set input variables
download_folder = r'C:\Users\Gavin\Downloads'
# Set path for browser driver
executable_path = {'executable_path': r'C:\Users\Gavin\Desktop\geckodriver-v0.33.0-win64\geckodriver.exe'}

# Link to FRED database
url = 'https://fred.stlouisfed.org/series/DTB4WK'

start_date = '2018-04-24'
end_date = '2023-04-21'

num_trading_days = 252
ma_lag = 10


# Download SPY data using yahooquery
# (yfinance package has been lacking some functionality due to changes to the yahoofinance site)
spy = yq.Ticker("SPY")
df = spy.history(start=start_date, end=end_date, interval='1d')
df['returns'] = np.log(df['adjclose'].pct_change()+1)

df.reset_index(drop=False, inplace=True)
df = df[['date', 'adjclose', 'returns']]

df['ma'] = df['adjclose'].rolling(window = ma_lag, min_periods = ma_lag).mean()

# Comment out lines between dashes to run file using pre-downloaded file
# ------------------------------------------------------------
# Download the FRED data for the given dates from website using the splinter package.
with Browser('firefox', **executable_path, headless=True) as browser:
    browser.visit(url)
    browser.find_by_id('input-cosd').click()
    time.sleep(1)
    browser.find_by_id('input-cosd').fill(start_date)
    browser.find_by_id('input-coed').click()
    time.sleep(1)
    browser.find_by_id('input-coed').fill(end_date)
    browser.find_by_id('download-button').click()
    browser.find_by_id('download-data-csv').click()
# ------------------------------------------------------------

df_rf = pd.read_csv(download_folder+"\\DTB4WK.csv")

# Transform the risk-free rate to daily rate, & fill in '.' values with previous value.
df_rf['DTB4WK'].loc[df_rf['DTB4WK']=='.'] = np.nan
df_rf['DTB4WK'] = df_rf['DTB4WK'].fillna(method='ffill')

df_rf['DTB4WK'] = df_rf['DTB4WK'].astype(float)/100
df_rf['DTB4WK'] = (1+df_rf['DTB4WK'])**(1/num_trading_days)-1

df_rf.rename(columns={"DATE":"date","DTB4WK":"rf"}, inplace=True)
df_rf['date'] = pd.to_datetime(df_rf['date']).dt.date

# Join risk-free rate & SPY data
df = df.merge(df_rf, how='left', on='date')
```

We download the SPY data using the yahooquery package, and get the risk-free rate data from the FRED website based on the given dates.
We then perform some data cleaning as there are some days which have null values for the risk-free rate. We fill these values in using the previous day's value.
The annual risk-free rate figures can then be converted to daily figures using the formula:

$$R_{daily} = (1 + R_{annual})^{\frac{1}{252}}$$

We then merge our two dataframes based on date.

Question 2:

```
# Calculate strategy position
df['position']=-1
df.loc[df['adjclose']>df['ma'], 'position']=1
df['position'] = df['position'].shift(1)

df['excess_return']=0
df.loc[df['position']==1, 'excess_return']=df['returns']-df['rf']

# Calculate summary statistics of returns
mean_return     = df['excess_return'].mean()
vol_returns     = df['excess_return'].std()
sharpe_returns  = mean_return/vol_returns

annualized_return = (1+mean_return)**num_trading_days-1
annualized_vol = vol_returns*np.sqrt(num_trading_days)
annualized_sharpe = annualized_return/annualized_vol

summary = pd.DataFrame(np.stack([[mean_return, vol_returns, sharpe_returns],
                                 [annualized_return, annualized_vol, annualized_sharpe]]))

summary.index = ['Daily', 'Annual']
summary.columns = ['Mean', 'Volatility', 'Sharpe Ratio']
print(summary)
```

We decide whether to hold/sell the stock each day based on whether the adjusted close was greater than the 10-day moving average or not on the previous day. Our excess returns on the portfolio can then be calculated using the following formula:

$$\tilde{R}_t = \begin{cases} R_t - R_t^f, & \text{if } P_{t-1} \geq A_{t-1,10} \\ 0, & \text{otherwise} \end{cases}$$

Calculating the mean, volatility & corresponding Sharpe ratio of the excess returns, we get the following table of results:

	Mean	Volatility	Sharpe Ratio
Daily	0.000327	0.007854	0.041594
Annual	0.085789	0.124675	0.688100

Question 3:

```
# Perform Pearsan-Timmermann test on the predicted sign of returns
df_pt_test = df.dropna()[['returns','position']] # Drop observations in the MA lead-in
n = len(df_pt_test)

df_pt_test['Z'] = np.sign(df_pt_test['returns']*df_pt_test['position'])
df_pt_test.loc[df_pt_test['Z']<0, 'Z']=0 # convert negative values to 0

P_hat = df_pt_test['Z'].mean()
P_x = len(df_pt_test.loc[df_pt_test['returns']>0])/n
P_y = len(df_pt_test.loc[df_pt_test['position']>0])/n
P_star = P_x*P_y+(1-P_x)*(1-P_y)

var_P_hat = n**-1*P_star*(1-P_star)
var_P_star = n**-1*(2*P_y-1)**2*P_x*(1-P_x) + n**-1*(2*P_x-1)**2*P_y*(1-P_y) + 4*n**-2*P_y*P_x*(1-P_y)*(1-P_x)

S_n = (P_hat-P_star)/(var_P_hat-var_P_star)**(1/2)
p_val = norm.cdf(-abs(S_n))

print(f'\nPearsan-Timmermann Test Statistic:\t{S_n:.4f}\nP-value of Test:\t{p_val:.4f}')
```

We then perform the Pearsan-Timmerman test on the predicted direction of stock price movements, in order to test the accuracy of our forecasts. We obtain a test-statistic of 0.6736, which has a corresponding p-value of 0.2503. This implies that under the null hypothesis, that the MA-forecast has no predictive power, we would see predictions as good as those provided by the MA-forecast just 25% of the time. This gives us an indication that the model is quite useful in predicting the direction of movements in the S&P.