Shea-FA692-HW3

April 5, 2023

# 1 FA692 Homework 3

# 2 Due: Wednesday, April 5 @ 11:59PM

Name: Ryan Shea

Date: 2023-04-05

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set seed of random number generator
CWID = 10445281 #Place here your Campus wide ID number, this will personalize
#your results, but still maintain the reproduceable nature of using seeds.
#If you ever need to reset the seed in this assignment, use this as your seed
#Papers that use -1 as this CWID variable will earn 0's so make sure you change
#this value before you submit your work.
personal = CWID % 10000
np.random.seed(personal)
```

## 2.1 Question 1 (20pt)

### 2.1.1 Question 1.1

Use the `yfinance` package (or other method of your choice) to obtain the daily adjusted close prices for the S&P500 (`SPY`) from January 1, 2023 to March 15, 2023. You should inspect the dates for your data to make sure you are including everything appropriately. Create a data frame (or array) of the daily log returns of this stock; you may concatenate this to your price data. Use the `print` command to display your data.

[ ]:
```
import yfinance as yf

data = yf.download('SPY', start='2023-01-01', end='2023-03-15')

data['log_ret'] = np.log(data['Adj Close']/data['Adj Close'].shift(1))
print(data.head())
```

```
[*********************100%***********************]  1 of 1 completed
                  Open        High         Low       Close   Adj Close  \
Date
2023-01-03  384.369995  386.429993  377.829987  380.820007  379.372131
2023-01-04  383.179993  385.880005  380.000000  383.760010  382.300964
2023-01-05  381.720001  381.839996  378.760010  379.380005  377.937592
2023-01-06  382.609985  389.250000  379.410004  388.079987  386.604492
2023-01-09  390.369995  393.700012  387.670013  387.859985  386.385345

               Volume   log_ret
Date
2023-01-03   74850700       NaN
2023-01-04   85934100  0.007691
2023-01-05   76970500 -0.011479
2023-01-06  104189600  0.022673
2023-01-09   73978100 -0.000567

                  Open        High         Low       Close   Adj Close  \
Date
2023-01-03  384.369995  386.429993  377.829987  380.820007  379.372131
2023-01-04  383.179993  385.880005  380.000000  383.760010  382.300964
2023-01-05  381.720001  381.839996  378.760010  379.380005  377.937592
2023-01-06  382.609985  389.250000  379.410004  388.079987  386.604492
2023-01-09  390.369995  393.700012  387.670013  387.859985  386.385345

               Volume   log_ret
Date
2023-01-03   74850700       NaN
2023-01-04   85934100  0.007691
2023-01-05   76970500 -0.011479
2023-01-06  104189600  0.022673
2023-01-09   73978100 -0.000567
```

### 2.1.2 Question 1.2

Scrape data from the Bloomberg `@business` Twitter account from January 1, 2023 to March 15, 2023. Save this data to a Data Frame with time stamps. Additionally, save all the collected data to a text file with time stamps. You will need to submit the text file along with your work (-5 points if not submitted).

Note: Bloomberg tweets sometimes include the pipe "|". I recomment using tilde "~" as a delimiter instead.

Hint: Because saving the tweets can take a long time, you can comment that code out before exporting to pdf.

[ ]:
```
# import snscrape.modules.twitter as tw

# f = open('business3.txt', 'w', encoding='utf-8')
```

```
# for tweet in tw.TwitterSearchScraper(query="(from:business) since:2023-01-01␣
↪until:2023-03-15").get_items():
#     date_str = tweet.date.strftime("%Y-%m-%d %H:%M:%S%z")
#     date_str = date_str[:-2] + ":" + date_str[-2:]
#     #f.write(date_str + "|" + tweet.content + "\n")
#     f.write(date_str + "~" + tweet.rawContent + "\n")
# f.close()
```

[ ]:

```
from datetime import datetime as dt
import pytz

business = []
dates = []
f = open('business3.txt', 'r', encoding='utf-8')

for l in f:
    line = l.split('~')
    date_str = line[0]
    try:
        date_time = dt.fromisoformat(date_str)
        date_time = date_time.astimezone(pytz.timezone("US/Eastern"))
        line[0] = date_time
        line[1] = line[1][:-1]
        business.append(line)
        dates.append(date_time.date())
    except:
        business[-1][1] += " "+l[:-1]
f.close()

business = pd.DataFrame(business, columns=['Time', 'Tweet'])
business['Date'] = dates

business
```

[ ]:

```
                            Time  \
0      2023-03-14 19:40:29-04:00
1      2023-03-14 19:40:29-04:00
2      2023-03-14 19:35:41-04:00
3      2023-03-14 19:31:07-04:00
4      2023-03-14 19:25:09-04:00
…                            …
26809 2022-12-31 19:00:09-05:00
26810 2022-12-31 19:00:09-05:00
26811 2022-12-31 19:00:09-05:00
26812 2022-12-31 19:00:09-05:00
26813 2022-12-31 19:00:08-05:00
```

```
                                                    Tweet        Date
0      One Japanese fintech firm is making it compuls…  2023-03-14
1      An unlikely startup guru has emerged in Japan,…  2023-03-14
2      Some US cities are late in making financial di…  2023-03-14
3      The shipping industry is looking to rethink ev…  2023-03-14
4      A biotech wants to cut fashion waste by using …  2023-03-14
…                                                   …         …
26809  Toymakers have found a new group of customers:…  2022-12-31
26810  Belarusian hackers and dissidents determined t…  2022-12-31
26811  Landlords are taking out millions in loans to …  2022-12-31
26812  It took a pandemic to make a dent in US inequa…  2022-12-31
26813  A planned train line in Mexico is billions ove…  2022-12-31

[26814 rows x 3 columns]
```

### 2.1.3 Question 1.3

Using your favorite sentiment analyzer (e.g., `vaderSentiment`), find the average sentiment for the headlines on each date that data was collected. Concatenate this sentiment score to your data frame of log returns. Use the `print` command to display your data.

[ ]:
```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

sentiment = []
analyzer = SentimentIntensityAnalyzer()
for tweet in business.Tweet:
    vs = analyzer.polarity_scores(tweet)
    sentiment.append(vs["compound"])

business['Sentiment'] = sentiment

data['sent'] = business.pivot_table(index='Date', values='Sentiment',␣
 ↪aggfunc='mean')
print(data.head())
```

```
                  Open        High         Low       Close   Adj Close  \
Date
2023-01-03  384.369995  386.429993  377.829987  380.820007  379.372131
2023-01-04  383.179993  385.880005  380.000000  383.760010  382.300964
2023-01-05  381.720001  381.839996  378.760010  379.380005  377.937592
2023-01-06  382.609985  389.250000  379.410004  388.079987  386.604492
2023-01-09  390.369995  393.700012  387.670013  387.859985  386.385345

              Volume   log_ret      sent
Date
2023-01-03  74850700       NaN -0.007160
2023-01-04  85934100  0.007691 -0.010883
```

```
2023-01-05   76970500 -0.011479 -0.032084
2023-01-06  104189600  0.022673 -0.001990
2023-01-09   73978100 -0.000567  0.048762
```

## 2.2 Question 2 (20pt)

### 2.2.1 Question 2.1

Linearly regress SPY returns as a function of the lagged returns (2 lags). This should be of the form $r_t = \beta_0 + \beta_1 r_{t-1} + \beta_2 r_{t-2}$. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```python
# from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import warnings
from pandas.core.common import SettingWithCopyWarning
warnings.filterwarnings('ignore', category=SettingWithCopyWarning)


df = data[['sent', 'Adj Close', 'log_ret']]

for i in range(1, 3):
    df[f"log_ret_{i}"] = df['log_ret'].shift(i)
    df[f"sent_{i}"] = df['sent'].shift(i)

df = df.dropna()

y = df['log_ret']
X = df[['log_ret_1', 'log_ret_2']]

# X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X, y, test_size=0.
↪2, random_state=personal)

model1 = LinearRegression()
model1.fit(X, y)

y_pred_1 = model1.predict(X)
mse_1 = mean_squared_error(y, y_pred_1)

print("MSE for model 1:", mse_1)

for coef, var in zip(model1.coef_, X.columns):
    print(f"{var:>5} : {coef}")

print(f"Intercept : {model1.intercept_}")
```

```
MSE for model 1: 0.00011083584269359484
log_ret_1 : 0.05614851635772728
```

```
log_ret_2 : -0.12133150745716827
Intercept : 0.0007263457920817708
```

### 2.2.2 Question 2.2

Linearly regress SPY returns as a function of the lagged sentiment (2 lags). This should be of the form $r_t = \beta_0 + \beta_1 s_{t-1} + \beta_2 s_{t-2}$. Evaluate the performance of this model with the mean squared error of the training data.

```
[ ]: y = df['log_ret']
     X = df[['sent_1', 'sent_2']]

     # X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X, y, test_size=0.
      ↪2, random_state=personal)

     model2 = LinearRegression()
     model2.fit(X, y)

     y_pred_2 = model2.predict(X)
     mse_2 = mean_squared_error(y, y_pred_2)

     print("MSE for model 2:", mse_2)

     for coef, var in zip(model2.coef_, X.columns):
         print(f"{var:>9} : {coef}")

     print(f"Intercept : {model2.intercept_}")
```

```
MSE for model 2: 0.00010145279900828163
   sent_1 : -0.06312613243734158
   sent_2 : -0.07328922397704475
Intercept : 0.0015461034838210735
```

### 2.2.3 Question 2.3

Linearly regress SPY returns as a function of the lagged returns and sentiment (2 lags each). This should be of the form $r_t = \beta_0 + \beta_{1,r} r_{t-1} + \beta_{2,r} r_{t-2} + \beta_{1,s} s_{t-1} + \beta_{2,s} s_{t-2}$. Evaluate the performance of this model with the mean squared error of the training data.

```
[ ]: y = df['log_ret']
     X = df[['log_ret_1', 'log_ret_2', 'sent_1', 'sent_2']]

     # X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(X, y, test_size=0.
      ↪2, random_state=personal)

     model3 = LinearRegression()
     model3.fit(X, y)
```

```python
y_pred_3 = model3.predict(X)
mse_3 = mean_squared_error(y, y_pred_3)

print("MSE for model 3:", mse_3)

for coef, var in zip(model3.coef_, X.columns):
    print(f"{var:>9} : {coef}")

print(f"Intercept : {model3.intercept_}")
```

```
MSE for model 3: 0.00010082831355879292
log_ret_1 : 0.030041028903574258
log_ret_2 : -0.07178435657940134
   sent_1 : -0.058105002975061194
   sent_2 : -0.07250676753721168
Intercept : 0.0015323903400410355
```

### 2.2.4 Question 2.4

Compare the performance of these 3 linear regressions.

It seems like the linear regression does the best when it is just the sentiment being analyzed. That is where the MSE is the lowest and also that the lagged return is not as solid of a predictor as the sentiment. The MSE is slighly better when just sentiment, but it is close.

[ ]:
```python
y = df['log_ret']
X = df[['log_ret_1', 'log_ret_2', 'sent_1', 'sent_2']]

# X_train_4, X_test_4, y_train_4, y_test_4 = train_test_split(X, y, test_size=0.
↪2, random_state=personal)

from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.00001)
lasso.fit(X, y)

y_pred_4 = lasso.predict(X)
mse_4 = mean_squared_error(y, y_pred_4)

print("MSE for LASSO:", mse_4)

for coef, var in zip(lasso.coef_, X.columns):
    print(f"{var:>9} : {coef}")

print(f"Intercept : {lasso.intercept_}")
```

```
MSE for LASSO: 0.00010161782815104636
log_ret_1 : 0.0
```

```
log_ret_2 : -0.0
   sent_1 : -0.05606639040958814
   sent_2 : -0.06384601609841364
Intercept : 0.0014422150856061768
```

When using LASSO, you can see that the log returns get penalized a lot more than sentiment, meaning that the algorithm also believes that sentiment is more important as well.

## 2.3 Question 3 (20pt)

### 2.3.1 Question 3.1

Regress `SPY` returns with a random forest as a function of the lagged returns (2 lags). This should be of the form $r_t = f(r_{t-1}, r_{t-2})$. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```python
from sklearn.ensemble import RandomForestRegressor

def rf(X, y):
    # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=personal)
    rf = RandomForestRegressor(random_state=personal)

    rf.fit(X, y)

    y_pred = rf.predict(X)
    mse = mean_squared_error(y, y_pred)
    print("MSE for Random Forest:", mse)

    return rf

y = df['log_ret']
X = df[['log_ret_1', 'log_ret_2']]

rf1 = rf(X, y)
```

```
MSE for Random Forest: 2.467667811997739e-05
```

### 2.3.2 Question 3.2

Regress `SPY` returns with a random forest as a function of the lagged sentiments (2 lags). This should be of the form $r_t = f(s_{t-1}, s_{t-2})$. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```python
y = df['log_ret']
X = df[['sent_1', 'sent_2']]

rf2 = rf(X, y)
```

```
MSE for Random Forest: 1.9185302109313623e-05
```

### 2.3.3 Question 3.3

Regress SPY returns with a random forest as a function of the lagged returns and sentiment (2 lags each). This should be of the form $r_t = f(r_{t-1}, r_{t-2}, s_{t-1}, s_{t-2})$. Evaluate the performance of this model with the mean squared error of the training data.

```
[ ]: y = df['log_ret']
     X = df[['log_ret_1', 'log_ret_2', 'sent_1', 'sent_2']]
     rf3 = rf(X, y)
```

```
MSE for Random Forest: 2.0071930255675254e-05
```

### 2.3.4 Question 3.4

Compare the performance of these 3 random forest regressions.

```
[ ]: for var, imp in zip(X.columns, rf3.feature_importances_):
         print(f"{var:>9} : {imp}")
```

```
log_ret_1 : 0.15087334302566752
log_ret_2 : 0.19491711197686978
   sent_1 : 0.4154345731427369
   sent_2 : 0.23877497185472585
```

The same thing happens here similar to the linear regression: The random forest does the best when it is just the sentiment being analyzed. The MSE is lowest and it seems like having the lagged returns makes it worse. The feature importances of the random forest also show a stronger importance of sentiment than the lagged returns.

## 2.4 Question 4 (10pt)

### 2.4.1 Question 4.1

Compare the performance of the various regressions utilized. Do you find the text data to be a useful feature in your analysis. Explain why or why not.

Both of these regressions said the same thing: sentiment is the most important features in the model. Not only was the MSE the lowest in both regressions that only used sentiment, but also, LASSO dropped out the log returns and the random forest showed the sentiment feature importances were higher than log returns. For this reason, text data is a useful feature.