Omar Sadek

MAIS 202: Deliverable 3

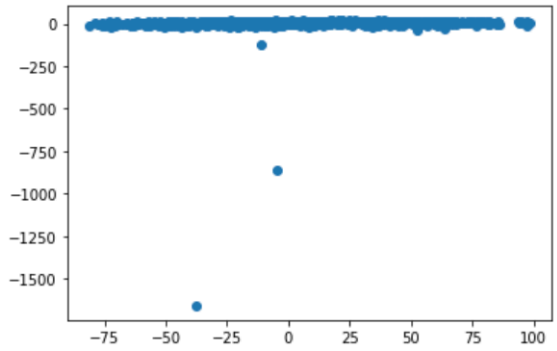
i- As discussed in the previous deliverables, my initial project idea was to use NLP on new articles to predict how the S&P 500 index would perform the same day. However, after a lot of data preprocessing and attempting to use different models, such as SVM and Random Forest Regressor, the model would continue to output the same result regardless of the new headline that is given to it as input. Therefore, it became apparent to me that the model is unable to detect any significant relationship between the words in such news headlines, and that is why it simply outputs a constant value. It was for that reason that I had to completely change my project.

My new project idea was to use a dataset from Kaggle that listed the financials of many companies and how their stock performed the following year. I then decided to make this another regression task where the aim would be allowing a user to input a company's financials, and then output a prediction for how that company's stock would perform the following year. In doing so, I attempted to use the following models: Linear Regression, Random Forest Regressor, SVM, Decision Tree Regressor, MLP Regressor, and a CNN using Sequential.

Each model had its pros and cons, and they mostly had very different Mean squared errors. The performance of each of the model is outlined as follows:

Linear Regression:

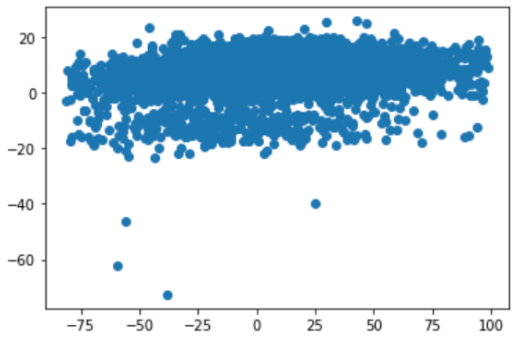
```
[22] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[23] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 3489.608238946783
```

```
[18] plt.scatter(y_train, y_train_predict)
     plt.show()
```

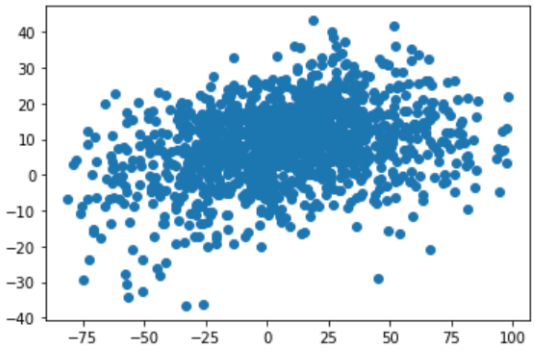



```
[19] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 997.8712244500134
```

Random Forest Regressor:

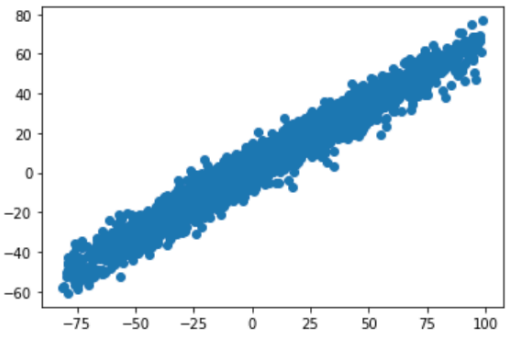
```
[30] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[31] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 972.3507960869266
```

```
[26] plt.scatter(y_train, y_train_predict)
     plt.show()
```

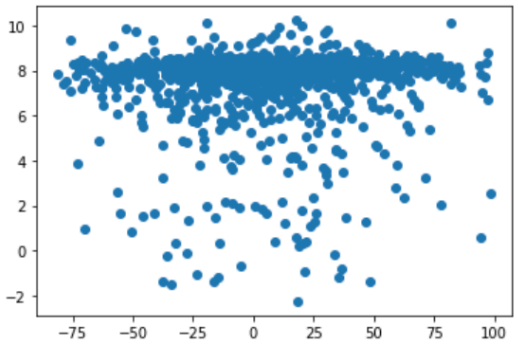



```
[27] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 139.54554436147004
```

SVM:

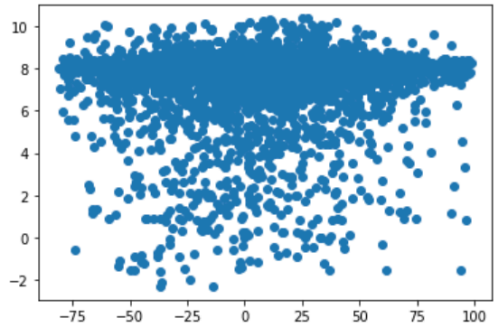
```
[38] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[39] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 1070.2417638253644
```

```
[34] plt.scatter(y_train, y_train_predict)
     plt.show()
```

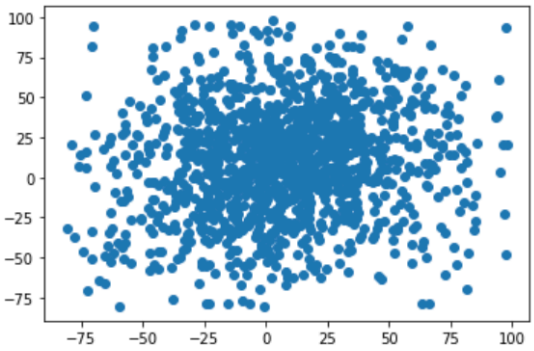



```
[35] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 1046.4331811020847
```

Decision Tree Regressor:

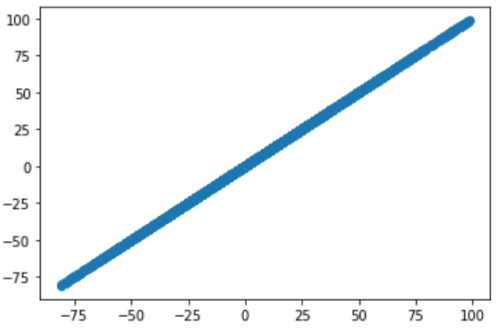
```
[46] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[47] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 1930.5566065463297
```

```
[42] plt.scatter(y_train, y_train_predict)
     plt.show()
```

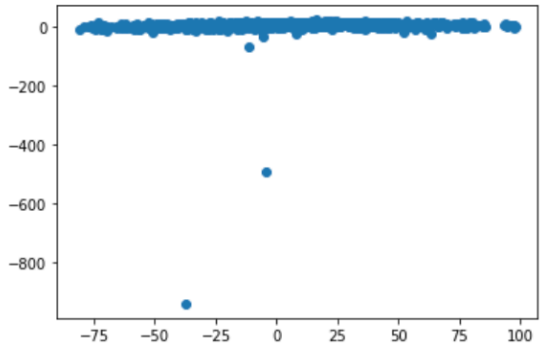



```
[43] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 0.0
```

MLP Regressor:

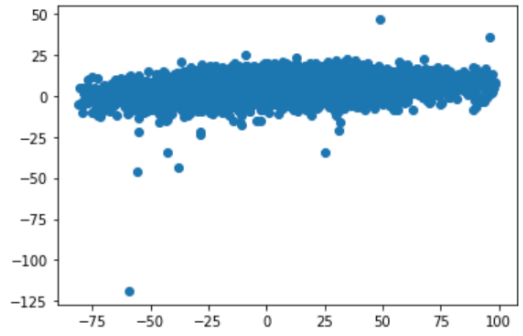
```
[54] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[55] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 1778.3980739901374
```

```
[50] plt.scatter(y_train, y_train_predict)
     plt.show()
```

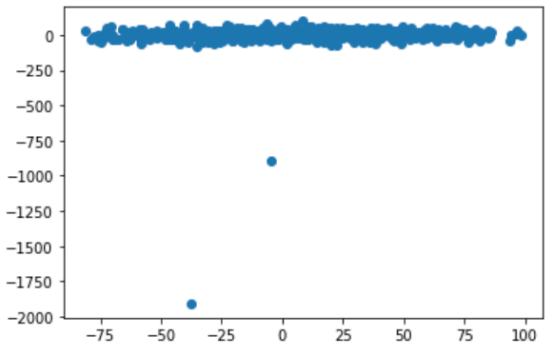



```
[51] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 983.3209969902357
```

CNN:

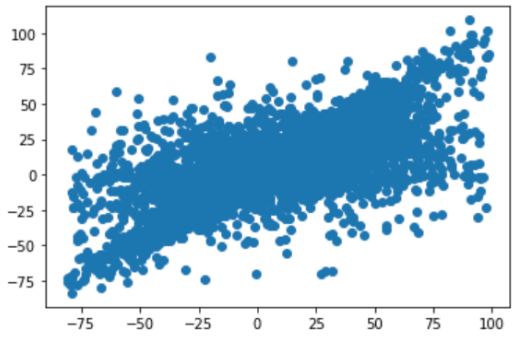
```
[65] plt.scatter(y_test, y_test_predict)
     plt.show()
```




```
[66] #MSE OF THE TESTING SET
     mse_test = np.mean(np.square(np.subtract(y_test, y_test_predict)))
     print("Testing set Mean Squared Error: {}".format(mse_test))
```

```
Testing set Mean Squared Error: 4793.319664623099
```

```
[61] plt.scatter(y_train, y_train_predict)
     plt.show()
```




```
[62] #MSE OF THE TRAINING SET
     mse_train = np.mean(np.square(np.subtract(y_train, y_train_predict)))
     print("Training set Mean Squared Error: {}".format(mse_train))
```

```
Training set Mean Squared Error: 1645.569586064599
```

As shown above, each model provided me with a slightly different way to model the data, and having this variation was good as it allowed me to carefully outweigh the pros and cons of each model, when deciding which model was most appropriate for this context. After some analysis, I was between the Random Forest Regressor, MLP Regressor, and the CNN models. The MLP Regressor and CNN models had relatively low Mean Squared Errors, but I chose not to go with them because when they were fit on the test set, they would occasionally output values that seemed absurdly large (in absolute value) and would not make any logical sense, considering the context. For example, a data point produced by the CNN model predicted a return of almost -2000% return for a stock. I chose to go with the Random Forest Regressor, because it

achieved the lowest Mean Squared Error on the test set and did so without being overfit on the train set. Furthermore, all of the predicted values it output after being run on the test set were between the range of about -40% to 40%, which is a very logical range given the context of the problem.

ii- As proposed in the previous deliverables, I am still planning on deploying this project through a website. I have already saved the relevant weights from the Random Forest Model using Pickle. I then added three methods that are used to vectorize the input data and created an HTML form that will take in the needed inputs from the user to create a vector that will then be fit using the model. In deploying this, I have used Flask and Heroku and this is also hosted on my personal website. This is the link: https://www.omarwagih.com/stockPredictor