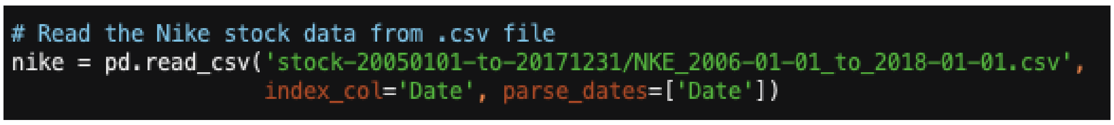
<u>Feature Selection</u>

In this section feature selection for the project will be explained, The developer loaded the dataset provided and carried out a few basic data inspections on the fie, such as observing the amount of features, looking at the first few lines of loaded data and a summary of the data to get an overall feel for the dataset, the developer then began to search of there was any missing data in the dataset. Once the developer realized there was missing data in the columns Open and Low, the developer got the average from both columns and divided it by two. The reason the developer did this is if they replaced all the values with 0 then it could have a negative impact n the dataset.

The developer used this value to fill in the missing data within the dataset. Once the developer was happy with the dataset they visualized the dataset using a time series to showcase each feature.

For this project the developer used from the sklearn library to perform feature selection. The developer performed Univariate feature extraction, PCA and feature importance to try and select the best feature. The developer also evaluated 6 different algorithms against the dataset. The algorithms are the following:
(1) Logistic Regression (LR)
(2) Linear Discriminant Analysis (LDA)
(3) K-Nearest Neighbors (KNN).
(4) Classification and Regression Trees (CART).
(5) Gaussian Naive Bayes (NB).
(6) Support Vector Machines (SVM).

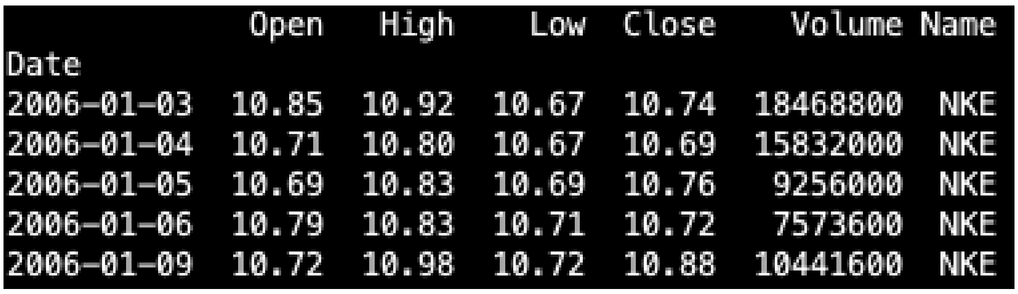
Below are some figures to help give the reader an idea of how the developer completed the feature selection part.



```
# Read the Nike stock data from .csv file
nike = pd.read_csv('stock-20050101-to-20171231/NKE_2006-01-01_to_2018-01-01.csv',
                   index_col='Date', parse_dates=['Date'])
```

Figure 1

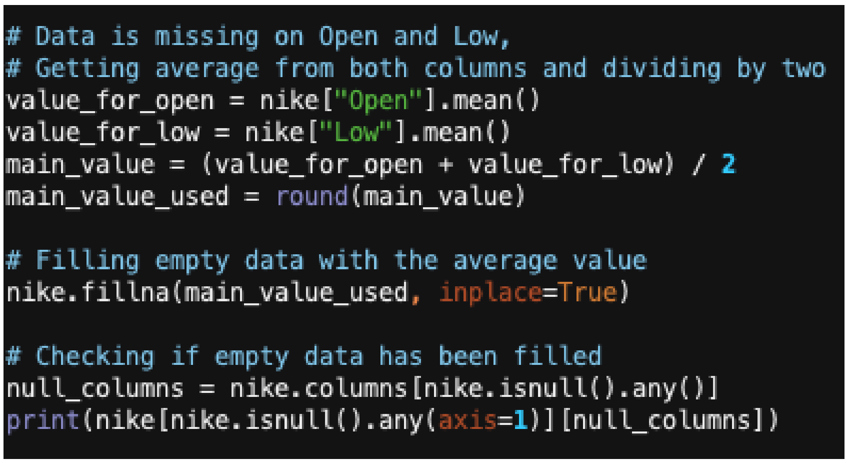
Above in figure 1 displays how the developer read in the dataset. Using pandas to read in the csv file the developer then added two parameters to the file. The required time series column is imported as a datetime column using parse_dates parameter and is also selected as index of the dataframe using index_col parameter.



```
             Open   High    Low  Close    Volume Name
Date
2006-01-03  10.85  10.92  10.67  10.74  18468800  NKE
2006-01-04  10.71  10.80  10.67  10.69  15832000  NKE
2006-01-05  10.69  10.83  10.69  10.76   9256000  NKE
2006-01-06  10.79  10.83  10.71  10.72   7573600  NKE
2006-01-09  10.72  10.98  10.72  10.88  10441600  NKE
```

Figure 2

Figure two shows the first few lines of the loaded data set NKE_20006-01-01_to_2018-01-01. It allows the developer to see how the data is presented along with what features are being used.



```
# Data is missing on Open and Low,
# Getting average from both columns and dividing by two
value_for_open = nike["Open"].mean()
value_for_low = nike["Low"].mean()
main_value = (value_for_open + value_for_low) / 2
main_value_used = round(main_value)

# Filling empty data with the average value
nike.fillna(main_value_used, inplace=True)

# Checking if empty data has been filled
null_columns = nike.columns[nike.isnull().any()]
print(nike[nike.isnull().any(axis=1)][null_columns])
```

Figure 3

Figure 3 shows how the developer searched the data set to see if there was any null values as well as how the developer replaced the values with the average of the column.

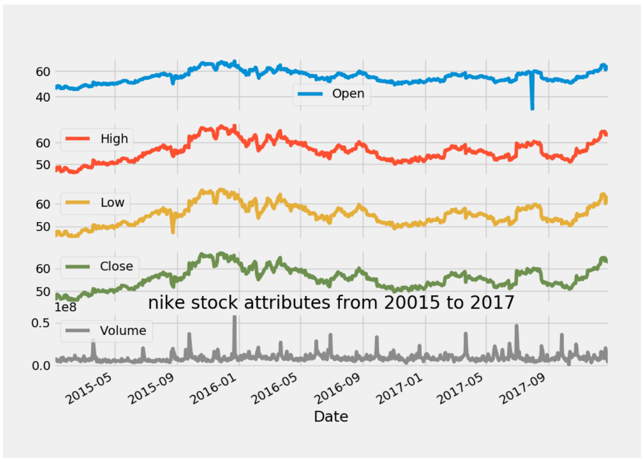
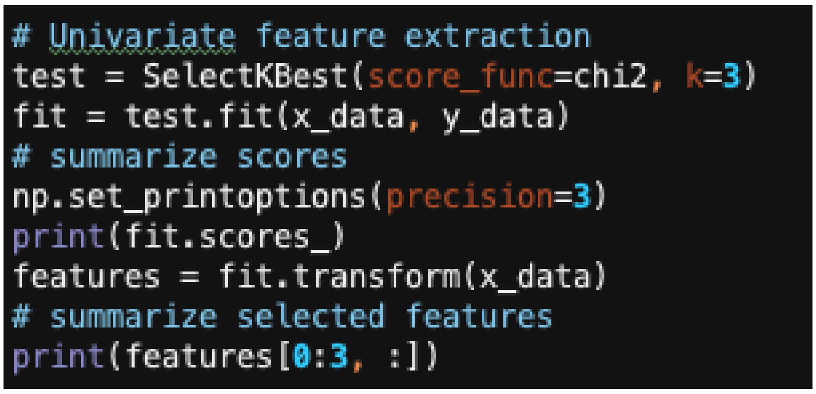


Figure 4

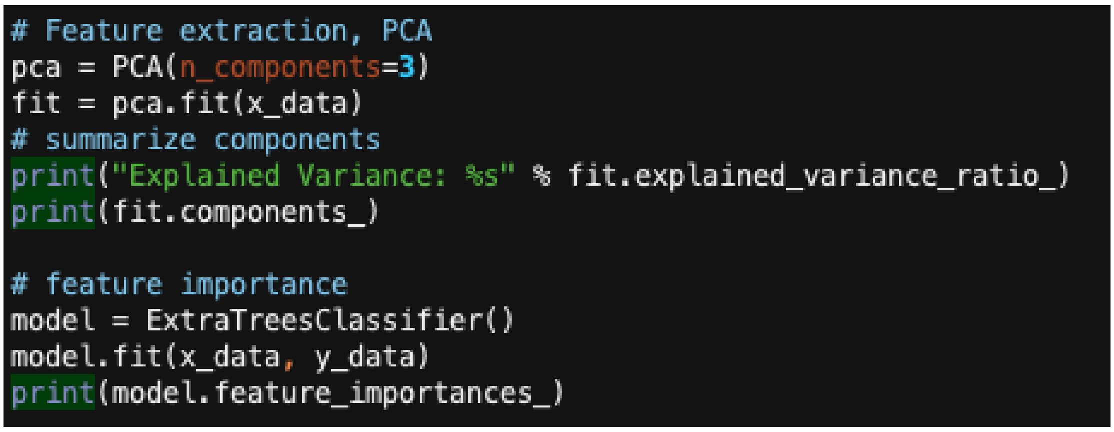
Figure 4 displays the time series of each feature in the dataset and how each feature performs over a series of time.



```
# Univariate feature extraction
test = SelectKBest(score_func=chi2, k=3)
fit = test.fit(x_data, y_data)
# summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(x_data)
# summarize selected features
print(features[0:3, :])
```

Figure 5

Figure 5 displays the feature selection method for “Univariate Feature Extraction” that was used on the dataset. The algorithm selects three of the best features from the dataset.

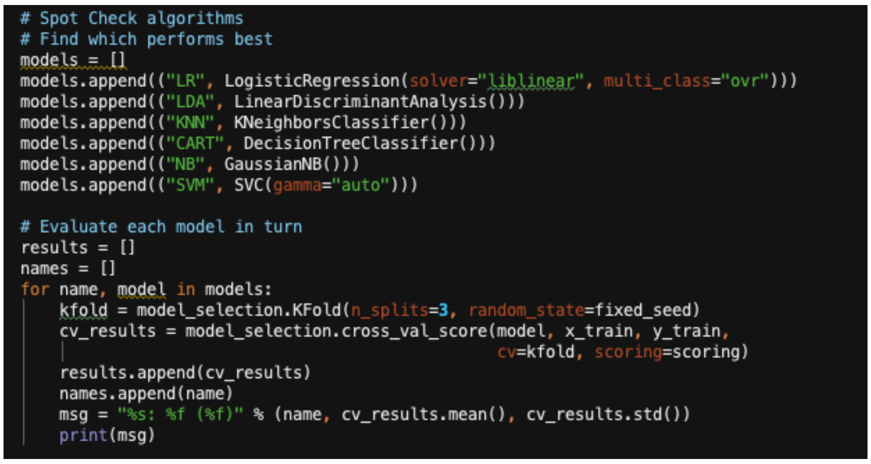


```python
# Feature extraction, PCA
pca = PCA(n_components=3)
fit = pca.fit(x_data)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)

# feature importance
model = ExtraTreesClassifier()
model.fit(x_data, y_data)
print(model.feature_importances_)
```

Figure 6

Figure 6 shows PCA and feature importance techniques being applied throughout a series of algorithms, PCA (data reduction) it was used, and 3 principal components were selected. Feature importance was also used to find which feature is the most important within the dataset, feature with higher scores will show more importance and vice versa.



```python
# Spot Check algorithms
# Find which performs best
models = []
models.append(("LR", LogisticRegression(solver="liblinear", multi_class="ovr")))
models.append(("LDA", LinearDiscriminantAnalysis()))
models.append(("KNN", KNeighborsClassifier()))
models.append(("CART", DecisionTreeClassifier()))
models.append(("NB", GaussianNB()))
models.append(("SVM", SVC(gamma="auto")))

# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=3, random_state=fixed_seed)
    cv_results = model_selection.cross_val_score(model, x_train, y_train,
                                                 cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

Figure 7

Figure 7 shows the process the developer took in order to find out with feature selection algorithm performed best against the dataset. The developer took all the algorithms and placed them in an array, next the developer ran each model before evaluating them using cross fold validation. Once complete the developer displayed the results in a boxplot as seen below in figure 8.

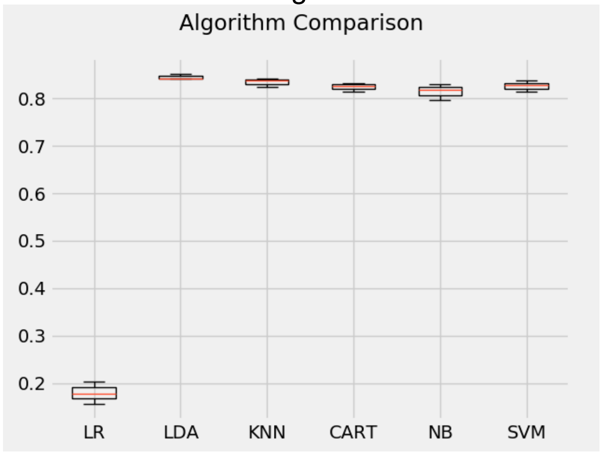


Figure 8

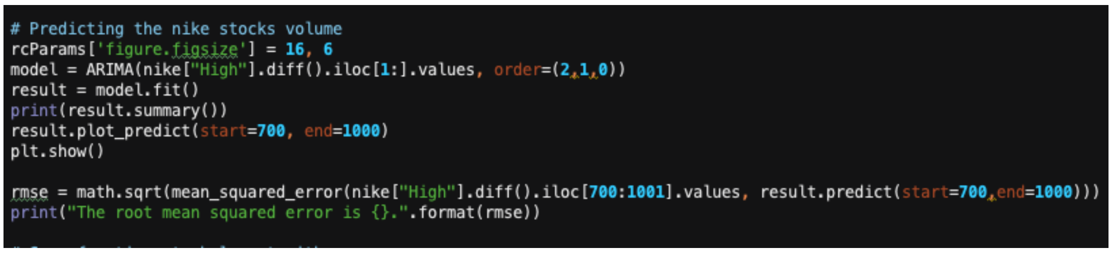
The developer can see that most algorithms are achieving over 80% accuracy how logistic regression is only achieving 20% accuracy, which is weird for logistic regression.

<u>Range of machine learning methods</u>

In this section machine learning methods used in the project will be looked at. The developer has performed feature selection on the dataset and now will chooses to evaluate three different machine-learning methods against said dataset; the models are as follow:

1. ARIMA model
2. LSTM model
3. GRU model

The developer used each model for a different reason. For ARIMA, the model is a simplification of the autoregressive moving average (ARMA) model. ARIMA models are implemented in some cases where data show evidence of non-stationarity, where an initial differencing step can be applied one or more times to eliminate the non-stationarity. How the developer implemented the ARIMA model can be seen in figure 9, with figure 10 displaying the resuts and figure 11 showing the root mean squared error rate.



```
# Predicting the nike stocks volume
rcParams['figure.figsize'] = 16, 6
model = ARIMA(nike["High"].diff().iloc[1:].values, order=(2,1,0))
result = model.fit()
print(result.summary())
result.plot_predict(start=700, end=1000)
plt.show()

rmse = math.sqrt(mean_squared_error(nike["High"].diff().iloc[700:1001].values, result.predict(start=700,end=1000)))
print("The root mean squared error is {}.".format(rmse))
```

Figure 9

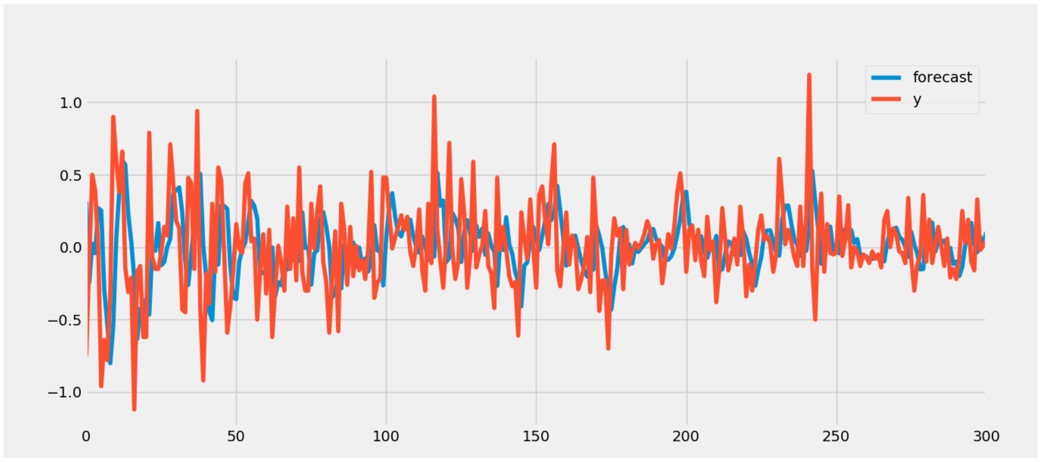
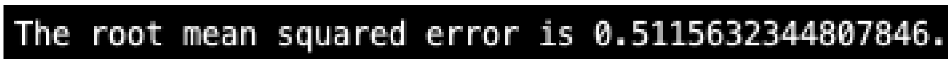


Figure 10

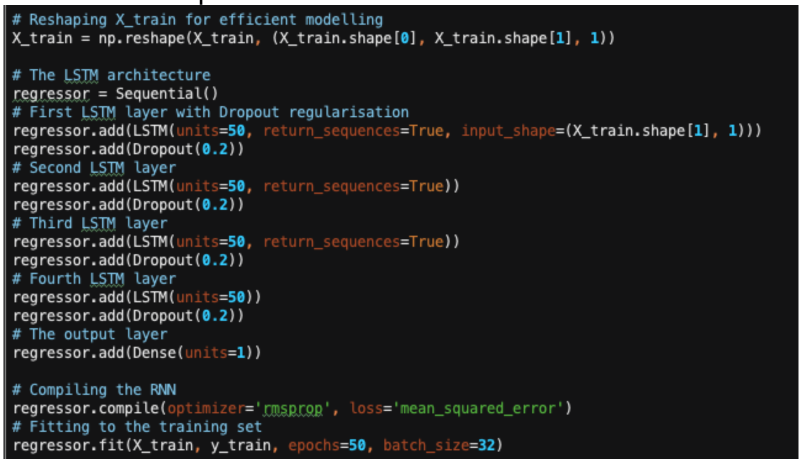


```
The root mean squared error is 0.5115632344807846.
```

Figure 11

The model is fine but there is a slight lag.

For Long Short Term Memory (LSTM), the model is best suited for the short-term memory, which can last for a long period of time. An LSTM is suited to classify, process and predict time series given time lags of unknown size and duration between important events. Below in figure 12 is how the developer implemented the LSTM model, with figure 13 displaying the results and figure 14 showing the developer the root mean squared error rate.



```
# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop', loss='mean_squared_error')
# Fitting to the training set
regressor.fit(X_train, y_train, epochs=50, batch_size=32)
```

Figure 12

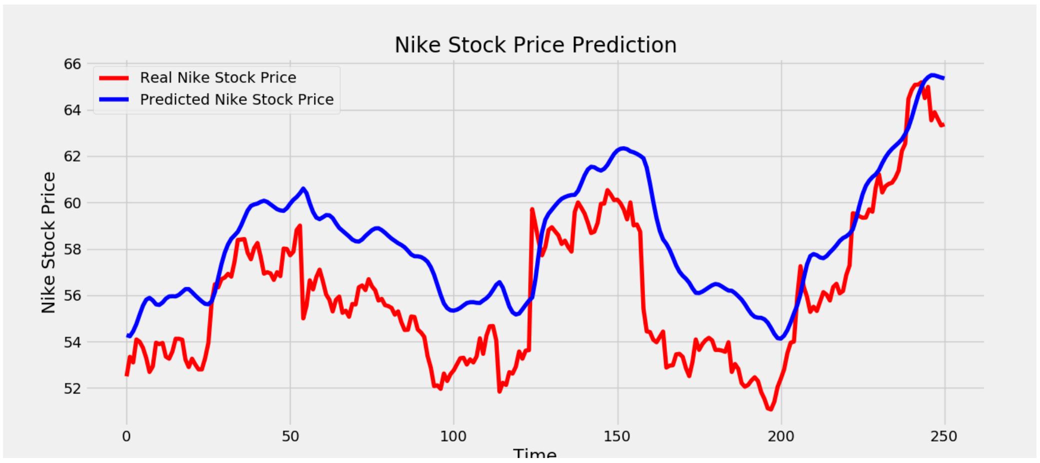
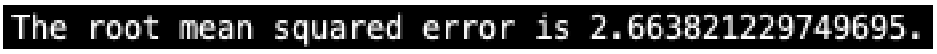


Figure 13

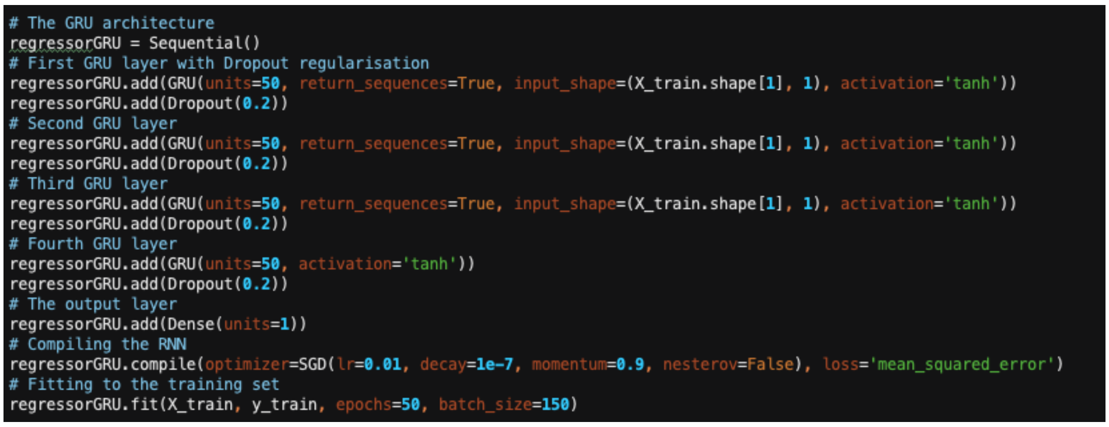


```
The root mean squared error is 2.663821229749695.
```

Figure 14

The model predicted too high for the dataset, but in the developer's opinion it was still a good result.

For GRU, the model, unlike the LSTM model, does not have to use a memory unit to control the flow of information. It can make use of the hidden states without any control. GRUs have fewer parameters and train that bit faster compared to LSTM. Below in figure 15 is how the developer implemented the LSTM model, with figure 16 displaying the results and figure 17 showing the developer the root mean squared error rate.



```
# The GRU architecture
regressorGRU = Sequential()
# First GRU layer with Dropout regularisation
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Second GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Third GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Fourth GRU layer
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dropout(0.2))
# The output layer
regressorGRU.add(Dense(units=1))
# Compiling the RNN
regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False), loss='mean_squared_error')
# Fitting to the training set
regressorGRU.fit(X_train, y_train, epochs=50, batch_size=150)
```

Figure 15

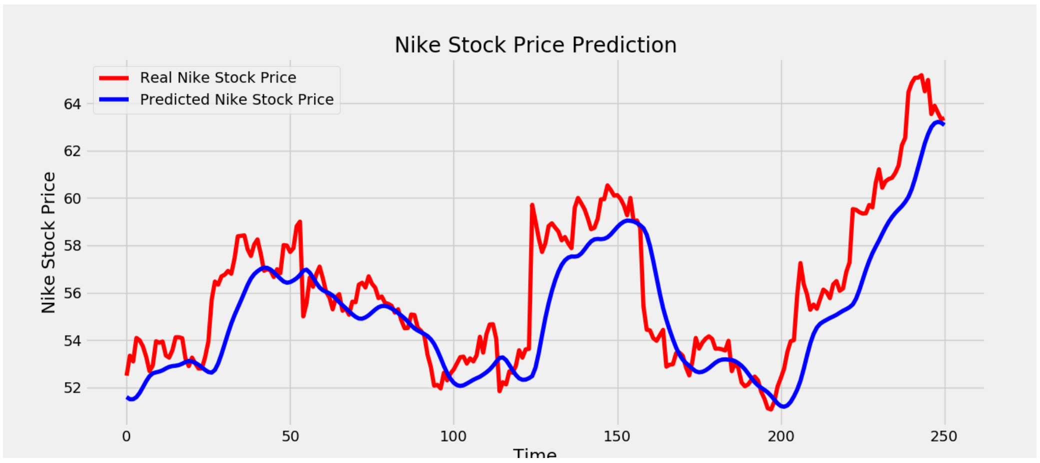
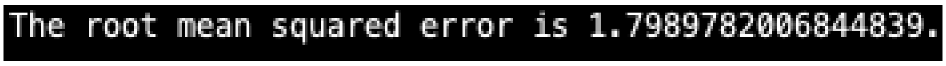


Figure 16



```
The root mean squared error is 1.7989782006844839.
```

Figure 17

The GRU model is very good at predicting the stock while lower than the actual dataset. After evaluating each model the developer decides it is best to use the machine-learning model GRU to predicate stock prices.

<u>Evaluating the other Datasets.</u>

Next the developer performs the GRU model on fellow datasets IBM, XOM and MMM. The developer simply took the GRU model and loaded the required dataset. Once that was accomplished the developer ran the model on the dataset. Below, in each figure, is the result of each dataset when the GRU model is performed on them and the root mean squared error rate.

IBM dataset

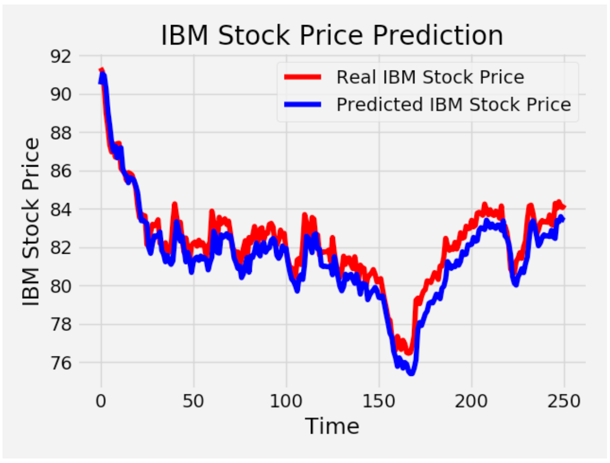
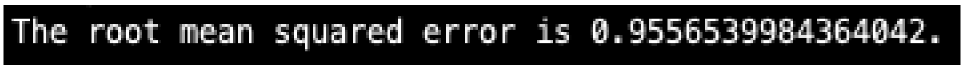


Figure 18



```
The root mean squared error is 0.9556539984364042.
```

Figure 19

MMM dataset

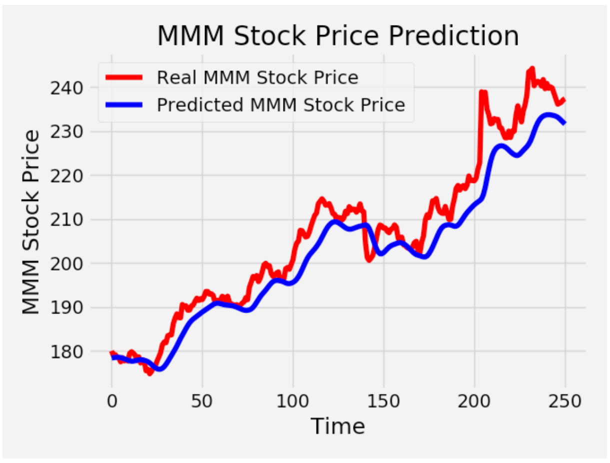
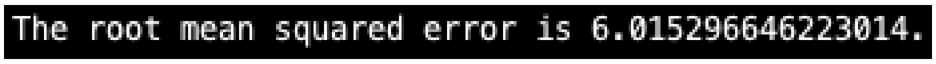


Figure 20



```
The root mean squared error is 6.015296646223014.
```

Figure 21

UTX Dataset

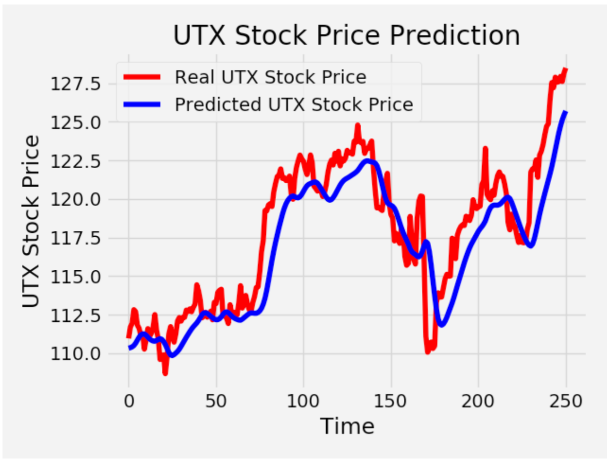
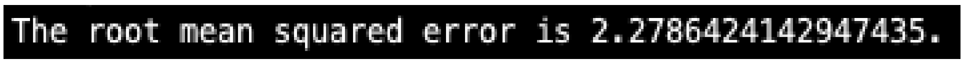


Figure 22



```
The root mean squared error is 2.2786424142947435.
```

Figure 23

Conclusion

Overall, the developer believes they need to gain a better understanding of trading algorithms and how to achieve feature selection on their dataset, the developer also feels the GRU models results still aren’t achieving what is being asked of them however it is a good start. In the end, the developer is happy with the results they have achieved. The developer believes they have improved on their python skills as well as deep learning skills. The project also allowed the developer to gain a bigger interest n machine learning and AI development.