Investigating the Efficient Market Hypothesis with Machine Learning

Shayne LaRochelle (301380121, BUS 315)

INTRODUCTION

In this paper, I will investigate the Efficient Market Hypothesis (EMH) using the 10 year historical stock data of Tesla, AMD and Google. Specifically, I will be investigating the weak form of the EMH. The weak form EMH states that all information that can be derived from analyzing the historical data of a stock is already reflected in its price (Bodie et.al, 2019). In other words, we should not be able to predict tomorrow's price change or the direction of tomorrow's price change given the stock's historical data. Through the use of both a regression based Recurrent Neural Network (RNN) and a classification based neural network, I will attempt to predict both a stock's future price change and the direction of tomorrow's price change.

Analysis of each stock is split into two sections. In the first section, I attempt to predict tomorrow's exact percentage price change using historical price change data. In the second section, I will attempt to predict the direction of the stock's price change tomorrow (up or down). If the model is able to predict either of these data points with accuracy, we will have strong evidence against the weak form of the EMH.

PROCESS

Data was retrieved from www.kaggle.com. For each company, I begin by cleaning and analyzing the data. All analysis was done using Python in my Jupyter Notebook and the following libraries were used: Pandas, NumPy, Seaborn, Sklearn and Tensorflow. All graphs included in this paper were created by myself.

After the data has been cleaned, I engineered the following additional features:

- *Percentage change in price* - The difference between closing price today and closing price yesterday.
- *1, 10, 30 and 90 day autocorrelation* - Computed as the autocorrelation amongst the stocks closing prices at 1, 10, 30 and 90 days of lag respectively.
- *1 day price* - The closing price of the stock in 1 day from today eg. tomorrow.
- *Price up* - Binary variable that is true if tomorrow's closing price is higher than today's closing price. (1 = Price is greater than today, 0 = Price is the same or lower tomorrow). This is the dependent variable for the second portion of the paper.

With each stock, I will first attempt to predict the percentage change in price between today and tomorrow. I will then attempt to predict whether the stock will be higher or lower than it is today.

Predicting Tomorrow's Price Change

Data is split into a 75:25 training:test split. Using a univariate RNN, I will attempt to predict the stock's real price action in 1 day, using the previous 30 days of "Percentage change in price" figures as input. After training the model, I will feed in my test set, which the model has never seen, to see if it is able to accurately predict tomorrow's percentage change in price.

To help visualize the output, the model will randomly sample three, 30+1 day periods from my test set, showing tomorrow's predicted and real percentage change in price. In machine learning terms, this is a regression problem. For practical reasons, we will run 10 epochs. The model uses the adaptive moment estimation optimization (ADAM) and is evaluated on the basis of the Mean Absolute Percentage Error (MAPE) loss function as well as whether or not the loss function is converging/ has converged. Loss function convergence is important because it suggests the model is no longer learning or improving as it iterates through the training set. Loss function convergence at a non-zero error provides us with evidence that that error value is as good as the model can do. In contrast, the loss function continuing to converge towards zero would occur only if historical data is helping to predict the future, which according to the EMH, should not be the case.

Predicting Tomorrow's Price Direction
Engineered variables with the exception of "Price up" are removed, as they contain future data that is unknowable at the date in question. Data is then split into a 75:25 training:test split. Utilizing ADAM optimization I attempt to predict whether the price of the stock will be higher tomorrow than it is today. In machine learning terms, this is a classification problem. The model is evaluated using total accuracy, precision, recall and f1 score. The model's performance metrics will be compared to the skew of the dependent variable. A cross validation matrix is then generated to visualize the success of the model. The skew of the dependent variable is very important to acknowledge and with classification models; precision, recall and f1 score are usually superior gauges of performance than absolute accuracy (Shung, 2020).

HYPOTHESIS

The null hypothesis is that asset prices are efficient, we should see no predictive pattern in the stock's day-to-day price change. If I am able to build a model that can predict the change in price or direction of a stocks change with statistical significance, this will be evidence against the EMH.

RESULTS

1. TESLA
The first company I will analyze is Tesla. Visually, the stock's price action exhibits no obvious pattern and appears to follow a random walk.

Predicting Tomorrow's Price Change
The model's MAPE is 100.05% when running on the validation set. After running 10 epochs, there is no evidence that the loss function is moving towards zero. For this reason, we are given neither evidence that the model is learning anything from the training set nor any evidence that running more iterations will improve the outcome of the model. From this, we can conclude that the current model has no predictive power.

Predicting Tomorrows' Price Direction
After cleaning, there remains 2325 days of historical data. In this period, the stock priced higher tomorrow than today 1179 times or 50.7% of the time.

The model's accuracy is 0.48135398 or 48.1%, which is worse than guessing at random. Furthermore, the model's precision is 51.7% and the model's recall is 50.8%. These combine for an f1 score of 51.2%. Considering the distribution of up and down days is ~51:49; these numbers are in line with randomly guessing. From this, we can conclude the model has no ability to predict the direction of the stock price tomorrow given historical data.

2. AMD
The second company I will analyze is AMD. Visually, the stock's price action exhibits no obvious pattern and appears to follow a random walk.

Predicting Tomorrow's Price Change
The model's MAPE is 99.6% and the loss function appears to have converged at roughly 100% error through 10 epochs.

Predicting Tomorrow's Price Direction
After data clearing, there remains 2333 days of historical data, of which 1184 or 50.8% were up days.
The model's accuracy is 0.5032679 or 50.3%. This is almost exactly in line with the skew of the dependent variable. The model's precision is 46.2%. The model's recall is very poor at 25.4%, which given the fairly

even distribution of up and down days, is significantly worse than guessing. The metrics combine for an f1 score of 32.8%. The model's performance metrics suggest that it is actually worse than randomly guessing.

3. Google
The final stock I will analyze is Google. Again, the daily percentage price change does not visually exhibit any pattern.

Predicting Tomorrow's Price Change
The model's MAPE is 100.27% and the loss function appears to have converged at roughly 100% error through 10 epochs.

Predicting Tomorrow's Price Direction
After data cleaning, there remains 2333 days of observation of which 1226 are up days or 52.6%

The model's accuracy is 51.9%, which is roughly akin to guessing and in line with what we would expect if the market is efficient. It is also in line with the skew of the dependent variable. The model's precision is 48.7%. The model's recall is poor, at 34.7%. These metrics combine for an f1 score of 40.0% Given the slight positive skew to the dependent variable, this result is significantly worse than guessing. Again, the outcome of these metrics do not suggest that the model is any better than guessing at random.

CONCLUSION

When trying to predict tomorrow's actual percentage price change, the models for each stock were off by more than 100%, hardly an actionable result. Furthermore, for all data sets, the loss functions had converged and were fluctuating around the mean MAPE for the total number of iterations. This means the model was not actually learning anything new that was beneficial as it iterated through the data sets.

When attempting to classify whether the stock would be higher or lower tomorrow than it is today, we again saw very poor results. For all datasets, the model's accuracy was roughly 50%. Only the Google model was able to attain more than 51% accuracy, but that data set had stronger skew in the dependent variable. Again, this level of accuracy is in line with what we would expect if it were guessing at random. Finally, the precision, recall and in turn, f1 of the classification models was poor for all data sets. It was especially poor for the AMD and Google data sets. More research and/or further testing would be required to identify why this was the case, but for both of these data sets, you would do better to simply guess the market will be up every other day.

In conclusion, this analysis has not provided us with any way to predict the price change or direction of movement of a stock in the coming days. We are also left with no reason to think any of the models would be improved upon given more data or more epochs. For this reason, the null hypothesis stands that the price of these stocks is efficient.

Further Extensions of the Model/Areas of Further Interest
To improve the model, we could add additional features based on other common financial metrics such as option execution volume, put/call ratios, executive option execution dates or executive equity transactions as well as company fundamentals. This would give us a way to test the semi-strong form of the EMH.

We could also transcribe executive press conferences and use a sentiment analysis model to look for possible words or combinations of words that may indicate that an executive is not telling the truth or exaggerating the truth in their press conference. If we are able to predict if a company was likely exaggerating the truth, we could assume there would be price corrections when the truth becomes public. We could then rerun our existing models to see if these features provide us with any predictive ability.

APPENDIX

Code Snippets of Interest

Features and graphs were computed for all stocks. All graphs have been included but for the sake of space, only code related to the generation of Tesla features are included.

Full list of libraries used

```
import pandas as pd
from pandas.plotting import autocorrelation_plot
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics, linear_model, datasets, feature_selection
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neural_network import MLPClassifier
import tensorflow as tf
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix
```

Original Data Sets

```
tesla.head(5) #showing first 5 rows of the original database
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

```
google.head(5)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2009-05-22	198.528534	199.524521	196.196198	196.946945	196.946945	3433700
1	2009-05-26	196.171173	202.702698	195.195190	202.382385	202.382385	6202700
2	2009-05-27	203.023026	206.136139	202.607605	202.982986	202.982986	6062500
3	2009-05-28	204.544540	206.016022	202.507507	205.405411	205.405411	5332200
4	2009-05-29	206.261261	208.823822	205.555557	208.823822	208.823822	5291100

```
amd.head(5)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2009-05-22	4.43	4.43	4.25	4.26	4.26	8274300
1	2009-05-26	4.26	4.57	4.23	4.53	4.53	16094300
2	2009-05-27	4.57	4.80	4.55	4.71	4.71	21512600
3	2009-05-28	4.75	4.84	4.54	4.70	4.70	18383900
4	2009-05-29	4.71	4.78	4.38	4.54	4.54	24539700

Feature Engineering

```
tesla['Percentage change in price'] = (tesla['Close']/tesla['Close'].shift(1))-1 #Change in price
"today" = price today - price yesterday
tesla['tesla_1_day_autocorr'] = tesla_auto_corr.autocorr(lag=1) #autocorrelation todays price
change with yesterdays
tesla['tesla_10_day_autocorr'] = tesla_auto_corr.autocorr(lag=10) #autocorrelation todays price
change with price 10 days ago
tesla['tesla_30_day_autocorr'] = tesla_auto_corr.autocorr(lag=30) #30 days ago
tesla['Tesla_90_day_autocorr'] = tesla_auto_corr.autocorr(lag=90) #90 days ago
tesla['1_day_close'] = tesla['Close'].shift(-1) #price 1 day in the future from "today"
tesla['Price Up Tomorrow'] = np.where(tesla['Close'].shift(-1)>tesla['Close'], 1, 0) #Price Up
Tomorrow Tomorrow is True (1) if tomorrows
#closing price is higher than todays. This is our dependent variable in our classification model
```

Autocorrelation within Percentage Price Change

Tesla

```
1 day lag: 0.008628711791364287.
10 day lag: 0.014093612910607414
30 day lag: 0.014708342742235087
90 day lag: 0.0018570185923042605
```

Google

```
1 day lag: 0.02977616713430228
10 day lag: 0.011715200721515733
30 day lag: -0.00723938611257335
90 day lag: -0.016023588821673814
```

AMD

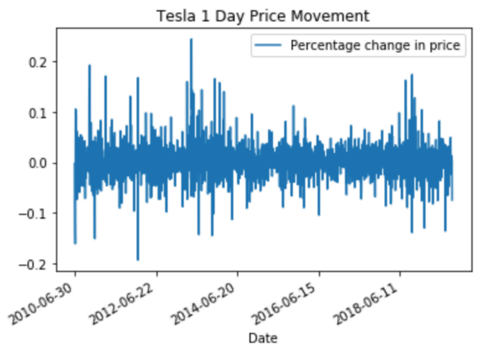
```
1 day lag: 0.004431876351059422
10 day lag: 0.04945694510526541
30 day lag: -0.024917746546279724
90 day lag: -0.02975236121221433
```

Percentage Change in Price Visuals and Summary Statistics

TESLA

```
tesla['Percentage change in price'].describe()
```

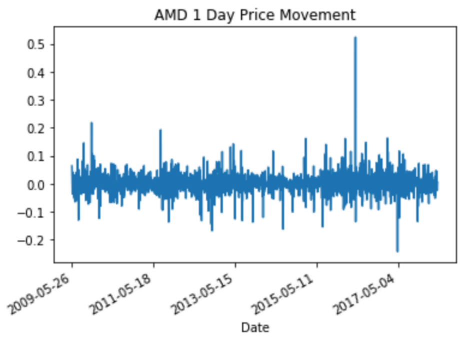
```
count    2415.000000
mean        0.001975
std         0.032762
min        -0.193274
25%        -0.014178
50%         0.000870
75%         0.018115
max         0.243951
Name: Percentage change in price, dtype: float64
```



AMD

```
amd['Percentage change in price'].describe()
```

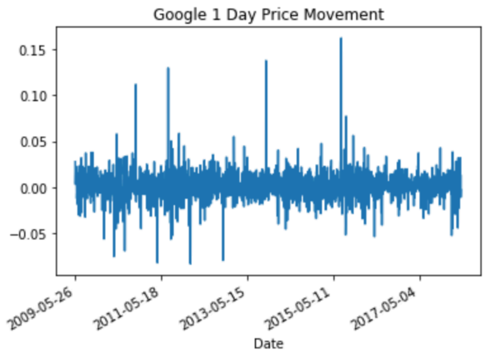
```
count    2334.000000
mean        0.001393
std         0.035922
min        -0.242291
25%        -0.016622
50%         0.000000
75%         0.018172
max         0.522901
Name: Percentage change in price, dtype: float64
```



Google

```
google['Percentage change in price'].describe()
```

```
count    2334.000000
mean        0.000909
std         0.015005
min        -0.083775
25%        -0.006118
50%         0.000681
75%         0.008511
max         0.162584
Name: Percentage change in price, dtype: float64
```



Price Action Prediction: Neural Network Hyper Parameters and Outputs of Loss Function

*Loss function is "val_loss: ##". Notice the values fluctuate around the mean and do not improve for any of the models.

TESLA

```
tesla_lstm_model = tf.keras.models.Sequential([tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),tf.keras.layers.Dense(1)])
tesla_lstm_model.add(tf.keras.layers.Dropout(0.2))
tesla_lstm_model.add(tf.keras.layers.Activation('linear'))
tesla_lstm_model.compile(optimizer='adam', loss='mape')
```

```
tesla_lstm_model.fit(train_univariate, epochs=10,
                      steps_per_epoch=500,
                      validation_data=tesla_univariate, validation_steps=50)
```

```
Epoch 1/10
500/500 [==============================] - 4s 9ms/step - loss: 109.2159 - val_loss: 99.9798
Epoch 2/10
500/500 [==============================] - 4s 8ms/step - loss: 100.7624 - val_loss: 100.1537
Epoch 3/10
500/500 [==============================] - 4s 9ms/step - loss: 100.4818 - val_loss: 99.8471
Epoch 4/10
500/500 [==============================] - 5s 9ms/step - loss: 100.1133 - val_loss: 101.3772
Epoch 5/10
500/500 [==============================] - 5s 9ms/step - loss: 100.2897 - val_loss: 100.0445
Epoch 6/10
500/500 [==============================] - 5s 9ms/step - loss: 99.9091 - val_loss: 100.6293
Epoch 7/10
500/500 [==============================] - 5s 10ms/step - loss: 99.9254 - val_loss: 100.2486
Epoch 8/10
500/500 [==============================] - 5s 10ms/step - loss: 99.9752 - val_loss: 99.6717
Epoch 9/10
500/500 [==============================] - 5s 10ms/step - loss: 99.8323 - val_loss: 102.2051
Epoch 10/10
500/500 [==============================] - 5s 10ms/step - loss: 99.9787 - val_loss: 100.0787
```

AMD

```
amd_lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),
    tf.keras.layers.Dense(1)
])
amd_lstm_model.add(tf.keras.layers.Dropout(0.2))
amd_lstm_model.add(tf.keras.layers.Activation('linear'))

amd_lstm_model.compile(optimizer='adam', loss='mape')
```

```
amd_lstm_model.fit(train_univariate, epochs=10,
                      steps_per_epoch=500,
                      validation_data=val_univariate, validation_steps=50)
```

```
Epoch 1/10
500/500 [==============================] - 4s 8ms/step - loss: 109.9867 - val_loss: 100.0050
Epoch 2/10
500/500 [==============================] - 4s 8ms/step - loss: 102.2896 - val_loss: 101.1916
Epoch 3/10
500/500 [==============================] - 4s 8ms/step - loss: 101.3090 - val_loss: 101.5589
Epoch 4/10
500/500 [==============================] - 4s 8ms/step - loss: 101.1818 - val_loss: 101.0481
Epoch 5/10
500/500 [==============================] - 4s 8ms/step - loss: 101.7388 - val_loss: 100.9510
Epoch 6/10
500/500 [==============================] - 4s 8ms/step - loss: 100.5984 - val_loss: 101.5657
Epoch 7/10
500/500 [==============================] - 4s 8ms/step - loss: 100.7585 - val_loss: 102.1222
Epoch 8/10
500/500 [==============================] - 4s 8ms/step - loss: 100.3366 - val_loss: 101.0926
Epoch 9/10
500/500 [==============================] - 4s 8ms/step - loss: 100.5671 - val_loss: 104.3902
Epoch 10/10
500/500 [==============================] - 4s 8ms/step - loss: 100.6035 - val_loss: 100.4426
```

Google

```
BATCH_SIZE = 32


train_univariate = tf.data.Dataset.from_tensor_slices((x_train_uni, y_train_uni))
train_univariate = train_univariate.cache().shuffle(TRAIN_SPLIT).batch(BATCH_SIZE).repeat()

val_univariate = tf.data.Dataset.from_tensor_slices((x_val_uni, y_val_uni))
val_univariate = val_univariate.batch(BATCH_SIZE).repeat()
```

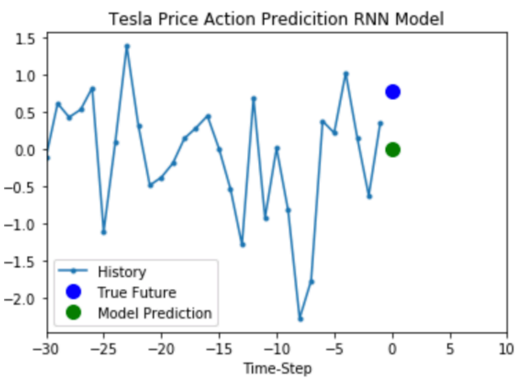
```
google_lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),
    tf.keras.layers.Dense(1)
])
google_lstm_model.add(tf.keras.layers.Dropout(0.2))
google_lstm_model.add(tf.keras.layers.Activation('linear'))

google_lstm_model.compile(optimizer='adam', loss='mape')
```

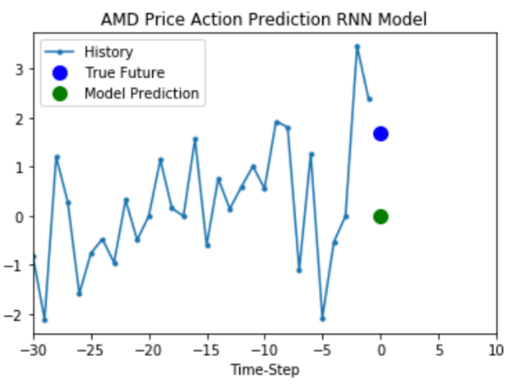
```
google_lstm_model.fit(train_univariate, epochs=10,
                      steps_per_epoch=500,
                      validation_data=val_univariate, validation_steps=50)
```

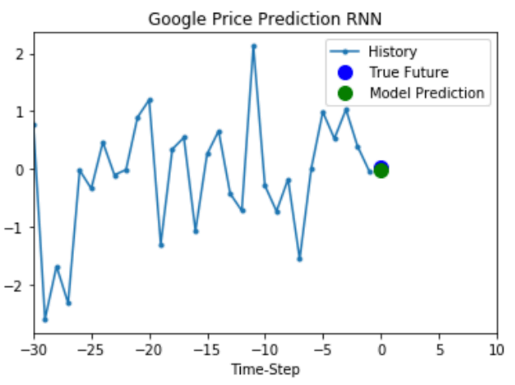
```
Epoch 1/10
500/500 [==============================] - 4s 8ms/step - loss: 101.9237 - val_loss: 100.7562
Epoch 2/10
500/500 [==============================] - 4s 8ms/step - loss: 100.9055 - val_loss: 100.2207
Epoch 3/10
500/500 [==============================] - 4s 8ms/step - loss: 100.5280 - val_loss: 99.2681
Epoch 4/10
500/500 [==============================] - 4s 8ms/step - loss: 100.7360 - val_loss: 100.7274
Epoch 5/10
500/500 [==============================] - 4s 8ms/step - loss: 100.5798 - val_loss: 101.1086
Epoch 6/10
500/500 [==============================] - 4s 8ms/step - loss: 100.2854 - val_loss: 100.1766
Epoch 7/10
500/500 [==============================] - 4s 8ms/step - loss: 100.1270 - val_loss: 100.9516
Epoch 8/10
500/500 [==============================] - 4s 8ms/step - loss: 100.1891 - val_loss: 100.3053
Epoch 9/10
500/500 [==============================] - 4s 8ms/step - loss: 99.9429 - val_loss: 99.4980
Epoch 10/10
500/500 [==============================] - 4s 8ms/step - loss: 100.2863 - val_loss: 99.7715
```

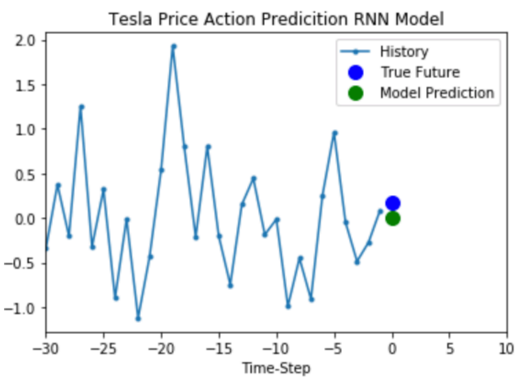
Price Action Prediction: Visualizing Prediction Accuracy
*Three randomly retrieved samples from out test showing 30 days of historical data and the models attempt to predict the action after point zero (Tomorrow)

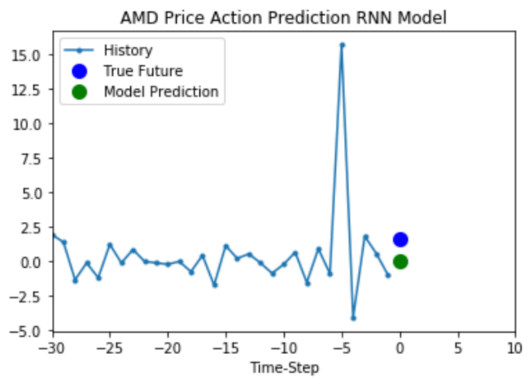
TESLA **AMD** **Google**

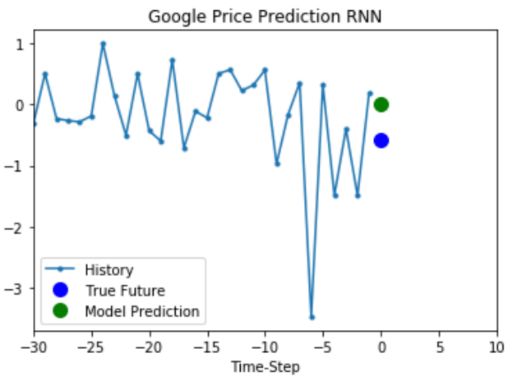


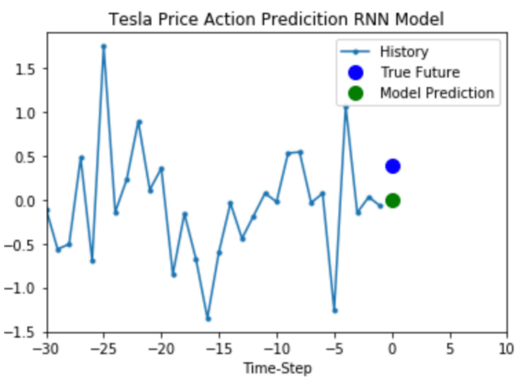


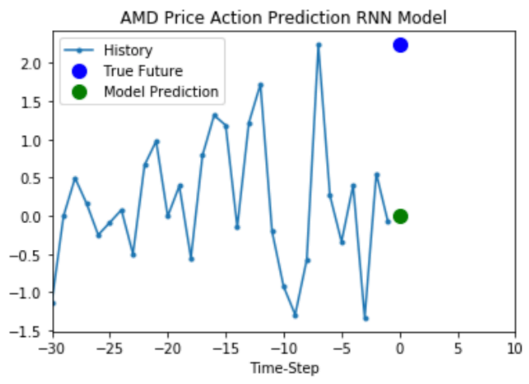


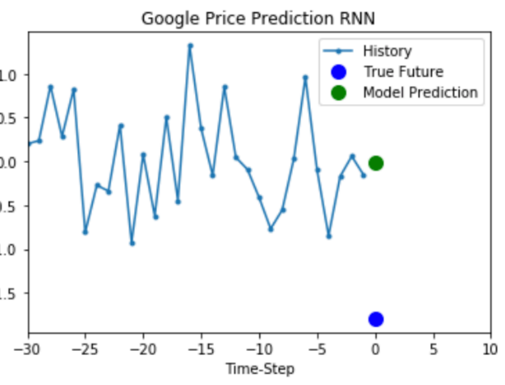






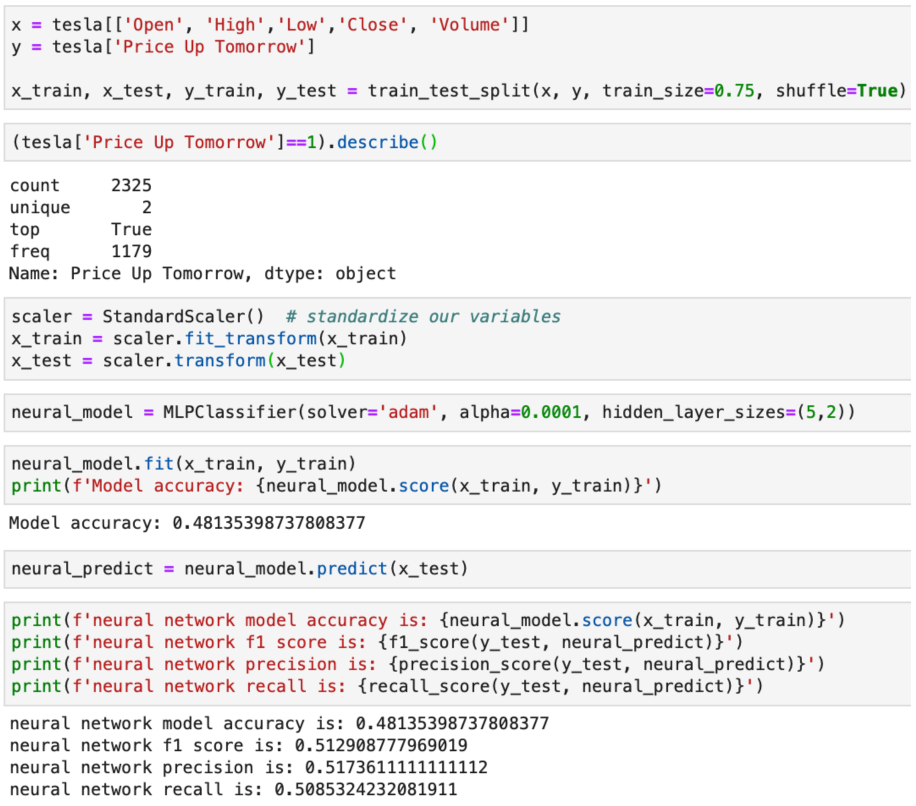






Price Direction Neural Network: Hyper Parameters and Output

1. TESLA



```
x = tesla[['Open', 'High','Low','Close', 'Volume']]
y = tesla['Price Up Tomorrow']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.75, shuffle=True)
```

```
(tesla['Price Up Tomorrow']==1).describe()
```

```
count      2325
unique        2
top        True
freq       1179
Name: Price Up Tomorrow, dtype: object
```

```
scaler = StandardScaler()  # standardize our variables
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
neural_model = MLPClassifier(solver='adam', alpha=0.0001, hidden_layer_sizes=(5,2))
```

```
neural_model.fit(x_train, y_train)
print(f'Model accuracy: {neural_model.score(x_train, y_train)}')
```

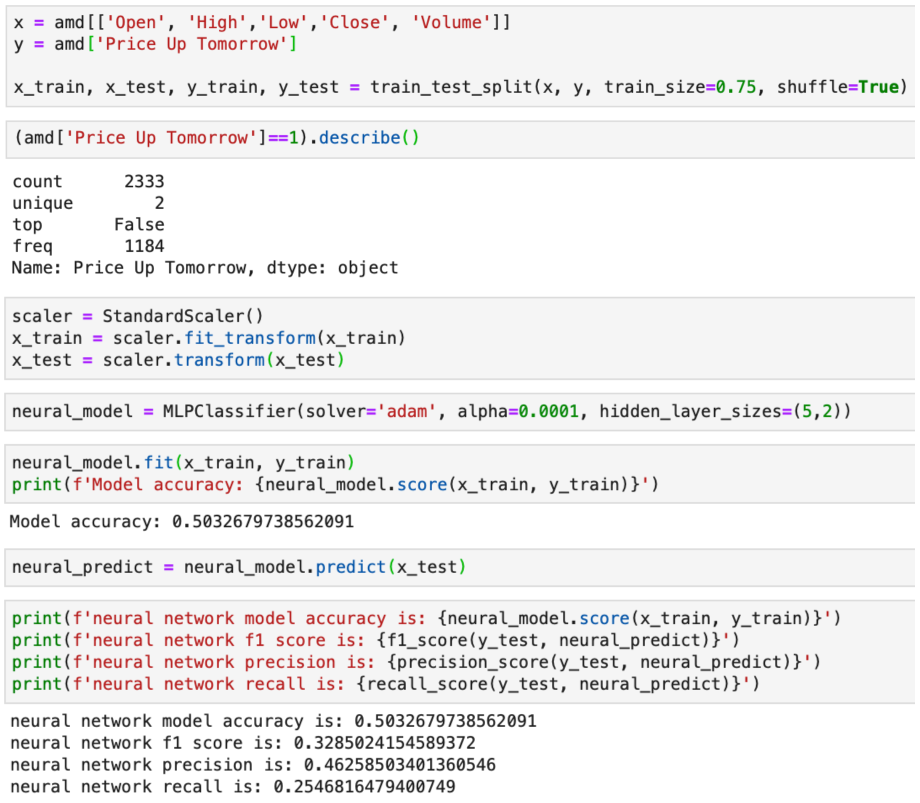
```
Model accuracy: 0.48135398737808377
```

```
neural_predict = neural_model.predict(x_test)
```

```
print(f'neural network model accuracy is: {neural_model.score(x_train, y_train)}')
print(f'neural network f1 score is: {f1_score(y_test, neural_predict)}')
print(f'neural network precision is: {precision_score(y_test, neural_predict)}')
print(f'neural network recall is: {recall_score(y_test, neural_predict)}')
```

```
neural network model accuracy is: 0.48135398737808377
neural network f1 score is: 0.512908777969019
neural network precision is: 0.5173611111111112
neural network recall is: 0.5085324232081911
```

2. AMD



```
x = amd[['Open', 'High','Low','Close', 'Volume']]
y = amd['Price Up Tomorrow']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.75, shuffle=True)
```

```
(amd['Price Up Tomorrow']==1).describe()
```

```
count       2333
unique         2
top        False
freq        1184
Name: Price Up Tomorrow, dtype: object
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
neural_model = MLPClassifier(solver='adam', alpha=0.0001, hidden_layer_sizes=(5,2))
```

```
neural_model.fit(x_train, y_train)
print(f'Model accuracy: {neural_model.score(x_train, y_train)}')
```

```
Model accuracy: 0.5032679738562091
```

```
neural_predict = neural_model.predict(x_test)
```

```
print(f'neural network model accuracy is: {neural_model.score(x_train, y_train)}')
print(f'neural network f1 score is: {f1_score(y_test, neural_predict)}')
print(f'neural network precision is: {precision_score(y_test, neural_predict)}')
print(f'neural network recall is: {recall_score(y_test, neural_predict)}')
```

```
neural network model accuracy is: 0.5032679738562091
neural network f1 score is: 0.3285024154589372
neural network precision is: 0.46258503401360546
neural network recall is: 0.2546816479400749
```

3. Google

```
x = google[['Open', 'High','Low','Close', 'Volume']]
y = google['Price Up Tomorrow']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.75, shuffle=True)
```

```
(google['Price Up Tomorrow']==1).describe()
```

```
count     2333
unique       2
top       True
freq      1226
Name: Price Up Tomorrow, dtype: object
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
neural_model = MLPClassifier(solver='adam', alpha=0.0001, hidden_layer_sizes=(5,2))
```

```
neural_model.fit(x_train, y_train)
print(f'Model accuracy: {neural_model.score(x_train, y_train)}')
```

```
Model accuracy: 0.5187165775401069
```

```
neural_predict = neural_model.predict(x_test)
```

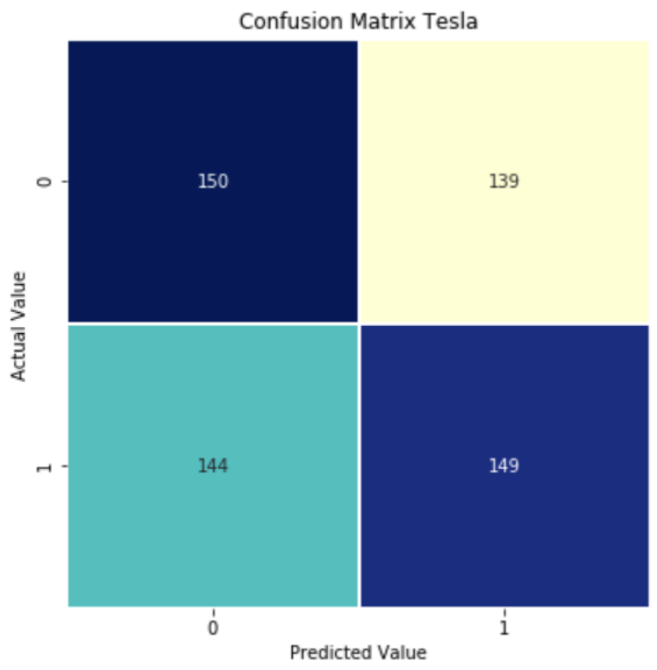
```
print(f'Google classification model accuracy is: {neural_model.score(x_train, y_train)}')
print(f'Google classification f1 score is: {f1_score(y_test, neural_predict)}')
print(f'Google classification precision is: {precision_score(y_test, neural_predict)}')
print(f'Google classification recall is: {recall_score(y_test, neural_predict)}')
```

```
Google classification model accuracy is: 0.5187165775401069
Google classification f1 score is: 0.4008714596949891
Google classification precision is: 0.48677248677248675
Google classification recall is: 0.34074074074074073
```

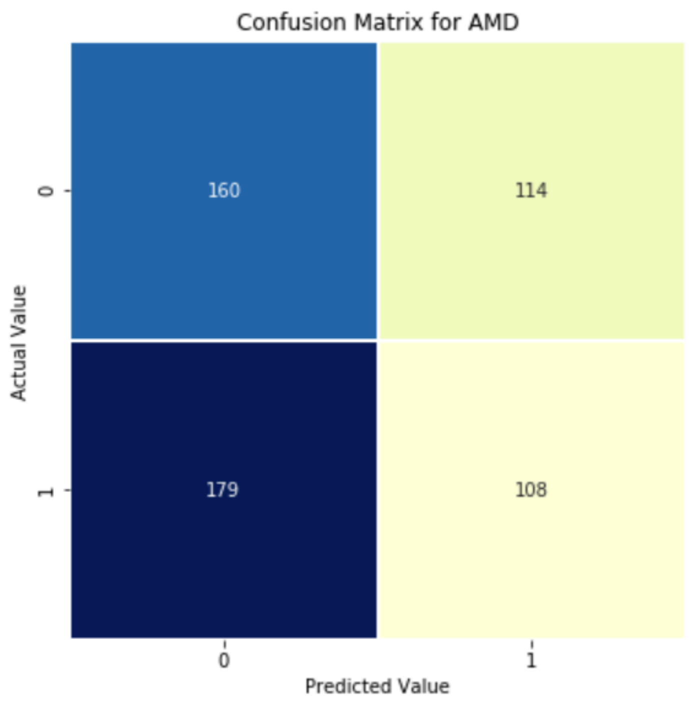
Classification Confusion Matrices

Interpreting the graphs: the Y axis shows the actual value that occurred. The X axis shows the value that was predicted by the model. Values in the top left and bottom right corner represent correctly predicted occurrences. The top right and bottom left represent false positives and false negatives respectively

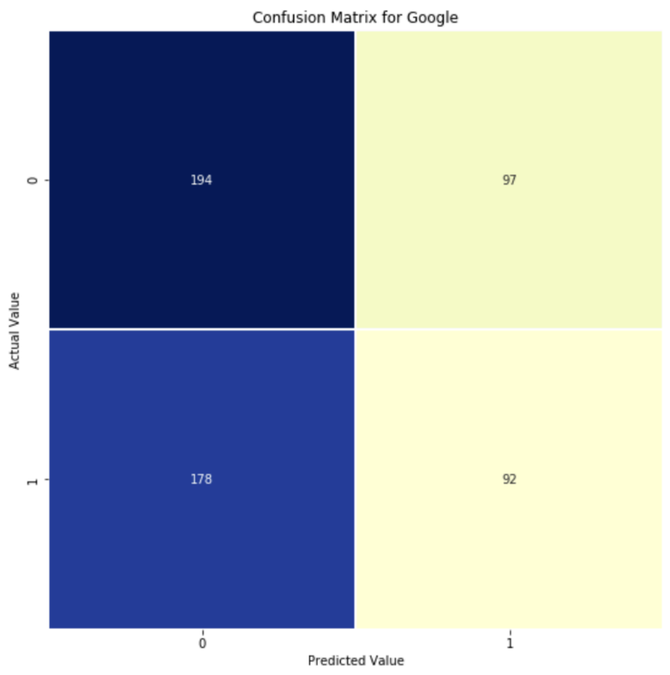
1. TESLA



2. AMD



3. Google



SOURCES

Bodie, Z., Kane, A., Marcus, A. J., Switzer, L., Stapleton, M., Boyko, D., & Panasian, C. (2019). *Investments* (p. 337). Whitby, Ontario: McGraw-Hill Ryerson.

Shung, K. (2020, April 10). Accuracy, Precision, Recall or F1? Retrieved July 28, 2020, from https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9