Project Overview:

The project aims to use the 20 variables over the course of 30 days to predict the return of shares over the course of 30 days. This will allow us to create a tool to facilitate users in getting better entry points for trades. The 20 variables are as follows:

1) 'Book_Value_Per_Share_*_USD',
2) 'Cap_Spending_USD_Mil',
3) 'Earnings_Per_Share_USD',
4) 'Free_Cash_Flow_Per_Share_*_USD',
5) 'Free_Cash_Flow_USD_Mil',
6) 'Gross_Margin_%',
7) 'Net_Income_USD_Mil',
8) 'Operating_Cash_Flow_USD_Mil',
9) 'Operating_Income_USD_Mil',
10) 'Operating_Margin_%',
11) 'Payout_Ratio_%_*',
12) 'Revenue_USD_Mil',
13) 'Shares_Mil',
14) 'Working_Capital_USD_Mil',
15) 'close',
16) 'dividend_amount',
17) 'high', ‘
18) ‘low',
19) 'open',
20) 'volume'

Given that we have chosen 131 tickers which made up the Dow Jones U.S. Technology Index, we expect them to be helpful in explaining the moves of one another. For example, an increase in Apple’s earnings per share will have a predictive impact on Google’s returns.

For our input data, I used 2 data source specifically and spliced it together to come up with the complete data set. The first data set is from AlphaVantage that provides me with the data on the equity prices such as high, low, open, share splits, volume and dividend amount. The second set of data is picked up from Morningstar. The data consist of the key ratios such as earnings per share, operating income, payout ratios, revenue and working capital. The companies report these data on a quarterly basis but Morningstar complies them on annual basis which is useful as it removes the seasonality factor of the data. Furthermore, some of the data by the companies have yet been reported and thus, I decided to forward fill the information i.e. filling forward the information such that the most recent readings will be used for the yet-reported data.

Initially, there was a plan to utilise Twitter data but I failed to properly classify the sentiment used and hence decided that it will be much more useful to utilise clean data rather than having a data point that is flawed. Furthermore, I am fortunate enough to obtain a lot more information from Morningstar and thus decided on removing the Twitter dependency but introduce more variables that I have sourced from Morningstar.

Problem Statement:

I will look to build a stock price predictor that takes daily trading data over a certain date range as input and output buy or sell signals for the given query date.

The best proxy for the future is the recent past is the basis of the strategy to solve this problem. I believe that all shares exhibit similar behaviours which allows us to utilise what we have learned from the share price movements in Apple to predict the future movements of Facebook or Google. On this basis it will be incredibly useful for each factor to learn from one another and thus help us to maximise the dataset we have. Furthermore, a rise in Apple revenue also could drive other related companies like Micron or Intel and thus, we can utilise such cross company information to help refine the expected return of the shares.

One has to note that a 30 day return model is used as well and this is fundamental because it helps us avoid volatility which is intrinsic in the market. The expected solution will be one where we provide a complete set of information of all the related data and will be able to confidently tell us if the share price of our target company will rise or fall. This expected solution will be a dense neural network comprising of 3 layers. Between each of these layers, I look to drop out 20% of the point so as to ensure that we do not overfit the model to the data we have. It is extremely important to do that given that a lot of the information we have are correlated and thus resulting in us failing to see the wood for the trees.

Metrics:

Originally the proposed metric to use the area under the Receiver Operating Characteristic Curve where the curve is plotted by using the True Positive Rate (True Positive divided by the sum of False Negative and True Positive) and False Positive Rate (False Positive divided by the sum of False Positive and True Negative) with different thresholds for the logistic regression. However, this was scrapped after I ran the model once as I realised that 30 days return is often highly skewed. For example, the last 30 values of the 30 days return for AMD is all positive and there is only one class present in the y true value making it impossible for us to calculate True Positive and False Positive Rates and thus unable to utilise the ROC AUC score. Thus I decided to change to utilise accuracy score which is a simple singular metrics to use which make the model output and that of the benchmark model easily comparable.

Data Exploration:

Looking at the data, I first spliced the various data sets into a single HDF. Thereafter, I took a first look at the first 5 rows of the dataframe. The first thing that jumped out was that there are various measures that has data that are empty. For example, the volume information of WP, Z and VSM is coming in as NaN.

```
In [2]: pivot= data.pivot_table(index=data.index,columns=data.ticker)
        pivot.columns.names = ['data_types','tickers']
        pivot.head()
```

Out[2]:

data_types	Book_Value_Per_Share_*_USD										...	volume							
tickers	AABA	AAPL	ACIW	ADBE	ADI	ADSK	AKAM	AMAT	AMD	ANET	...	VRSN	VSAT	VSM	WDAY	WDC	WP	XLNX	Z
date																			
2015-07-09	29.86	22.53	5.03	13.74	16.23	9.99	17.41	6.88	-0.42	10.61	...	587436.0	326919.0	NaN	1293608.0	2106557.0	NaN	3417044.0	NaN
2015-07-10	29.86	22.53	5.03	13.74	16.23	9.99	17.41	6.88	-0.42	10.61	...	400388.0	160234.0	NaN	1382053.0	1960616.0	NaN	2506968.0	NaN
2015-07-13	29.86	22.53	5.03	13.74	16.23	9.99	17.41	6.88	-0.42	10.61	...	423769.0	140892.0	NaN	1192943.0	3047139.0	NaN	2201036.0	NaN
2015-07-14	29.86	22.53	5.03	13.74	16.23	9.99	17.41	6.88	-0.42	10.61	...	568524.0	234940.0	NaN	1974882.0	3203009.0	NaN	2164689.0	NaN
2015-07-15	29.86	22.53	5.03	13.74	16.23	9.99	17.41	6.88	-0.42	10.61	...	744753.0	149075.0	NaN	2187854.0	3341257.0	NaN	1732800.0	NaN

5 rows × 2849 columns

The data frame has 750 rows and 2849 columns where the rows are dates starting from 9th July 2015 to 28th June 2018. The columns have 2 layers namely data_types and tickers. Thus there are 23 different data types for each of the 131 unique tickers.

I created a new "missing_data" dataframe using a summary of "total" and "percent" dataframes. I then sorted it and noticed that out of the 2849 columns, there are 393 columns that have some sort of data that is missing in the columns. Each ticker is supposed to have 23 * 750 = 17,250 data points. Given that I will need to split the data into training and testing dataset, I decided to remove the tickers which has more than 15% of 17,250 data points missing. Thus, out of the 131 tickers, 4 of the tickers have more than 15% of their data missing as below:

```
In [5]: data_points_per_ticker = len(pivot)* len(pivot.columns.get_level_values("data_types").unique())
        print("Each ticker would have ", data_points_per_ticker, "data points")
        print("Thus, if we have data missing for more than 15% of the data, we will remove those tickers")

        tickers_to_remove = missing_data["Total"].groupby("tickers").sum().sort_values(ascending= False)
        tickers_to_remove=tickers_to_remove.where(tickers_to_remove>=0.15*data_points_per_ticker).dropna()
        tickers_to_remove

        Each ticker would have  17250 data points
        Thus, if we have data missing for more than 15% of the data, we will remove those tickers

Out[5]: tickers
        WP      12046.0
        DXC      9316.0
        VSM      7550.0
        DVMT     6737.0
        Name: Total, dtype: float64
```

Thus we are left with 393-(4*23) = 301 columns that has some sort of missing data. Looking at the top 20 names that drive the highest percentage of missing data, I noticed that all the data that are missing are data from my Morningstar source

```
In [8]: missing_data  = ReportFunctions.check_missing(pivot)
        missing_data[missing_data['Total']>0].head(20)
```

Out[8]:

		Total	Percent
data_types	tickers		
Payout_Ratio_%_*	DBD	627	83.600000
Dividends_USD	IAC	627	83.600000
Payout_Ratio_%_*	IAC	627	83.600000
Free_Cash_Flow_Per_Share_*_USD	VMW	626	83.466667
Book_Value_Per_Share_*_USD	DY	626	83.466667
Working_Capital_USD_Mil	AABA	375	50.000000
Book_Value_Per_Share_*_USD	VMW	375	50.000000
Dividends_USD	CTSH	375	50.000000
Free_Cash_Flow_Per_Share_*_USD	AABA	375	50.000000
Earnings_Per_Share_USD	AABA	375	50.000000

Looking at the column with the highest ratio of missing data, I noticed that DBD has missing payout ratio due to the fact that Payout Ratio is computed as Dividends divided by Earnings. However, given that the company made a loss in 2016, 2017 and 2018. It resulted in a negative Payout Ratio which the data omits. Thus, decide that I will replace all the Payout Ratio by dividing the Dividends by the Earnings and accept that there will be negative Payout Ratios.

```
In [9]: pivot.xs("DBD",axis=1,level=1).xs([r"Payout_Ratio_%_*","Earnings_Per_Share_USD","Dividends_USD"],axis=1).loc["2015-12-30":"2016-1
```

Out[9]:

data_types	Payout_Ratio_%_*	Earnings_Per_Share_USD	Dividends_USD
date			
2015-12-30	105.5	1.12	1.15
2015-12-31	105.5	1.12	1.15
2016-01-04	NaN	-0.48	0.96
2016-01-05	NaN	-0.48	0.96
2016-01-06	NaN	-0.48	0.96

```
In [10]: newPayoutRatio = (pivot.loc[:,"Dividends_USD"]/pivot.loc[:,"Earnings_Per_Share_USD"]).copy()
         newPayoutRatio = newPayoutRatio.dropna(axis =1,how='all').fillna(0)
         pivot.drop(r"Payout_Ratio_%_*",axis=1,inplace=True)
         tickers = newPayoutRatio.columns
         newPayoutRatio.columns = pd.MultiIndex.from_tuples([(r"Payout_Ratio_%_*",ticker) for ticker in tickers], names=['data_types', 'se
         pivot = pd.concat([pivot,newPayoutRatio],axis=1)
         display(pivot.loc[:,r"Payout_Ratio_%_*"].xs(["DBD"],axis=1).loc["2015-12-30":"2016-1-6"])
```

tickers	DBD
date	
2015-12-30	1.026786
2015-12-31	1.026786
2016-01-04	-2.000000
2016-01-05	-2.000000
2016-01-06	-2.000000

After this cleaning, we note that we have significantly less dirty data and we note that they are generally concentrated in the following 5 fields

```
In [21]: missing_data  = ReportFunctions.check_missing(pivot)
         missing_data[missing_data['Total']>0].groupby("data_types").sum().sort_values("Total",ascending=False)
```

Out[21]:

	Total	Percent
data_types		
Free_Cash_Flow_Per_Share_*_USD	14001	1866.800000
Working_Capital_USD_Mil	13522	1802.933333
Book_Value_Per_Share_*_USD	1353	180.400000
Dividends_USD	1249	166.533333
Earnings_Per_Share_USD	499	66.533333
split_coefficient	124	16.533333

Given that we have 2 measures that are identical to each other namely

1. Free_Cash_Flow_Per_Share_*_USD
2. 'Free_Cash_Flow_Per_Share_*_USD'.

Thus, we can safely ignore the issues with the measure Free_Cash_Flow_Per_Share_*_USD

Also, I note that there are shares that do not distribute dividends like IAC in recent years hence, it is fair to fill the NaN values in as zero. The same applies to that of Book_Value_Per_Share_*_USD and Working_Capital_USD_Mil.

```
In [14]: missing_data  = ReportFunctions.check_missing(pivot)
         missing_data[missing_data.index.get_level_values("data_types")!="Free_Cash_Flow_Per_Share_*_USD"]
```

Out[14]:

		Total	Percent
data_types	tickers		
Earnings_Per_Share_USD	AABA	375	50.000000
Free_Cash_Flow_USD_Mil	HPE	71	9.466667
Operating_Income_USD_Mil	HPE	71	9.466667
volume	HPE	71	9.466667
split_coefficient	HPE	71	9.466667
dividend_amount	HPE	71	9.466667
Earnings_Per_Share_USD	HPE	71	9.466667
open	HPE	71	9.466667
Revenue_USD_Mil	HPE	71	9.466667
low	HPE	71	9.466667

```
In [15]: print("For the remaining details, we will provide a forward fill so as to ensure that there will be no gaps in due to data i.e. u
         pivot= pivot.fillna(method="ffill")
         pivot= pivot.fillna(0)
```

```
For the remaining details, we will provide a forward fill so as to ensure that there will be no gaps in due to data i.e. using
the latest data and using it for data that we have yet to received. We will then replace all NA values with zero.
```

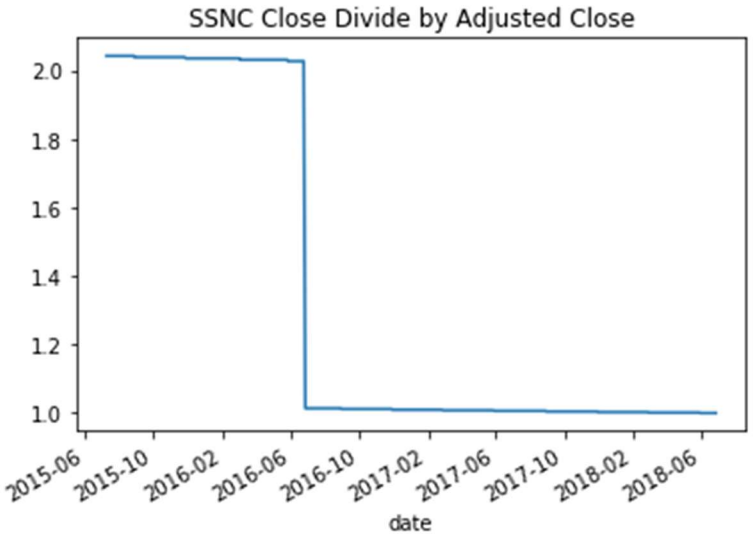
Hence we are left with just some odd data points that are empty. AABA has incomplete information but given that we are in a situation where we are unable to get the completeness of the Earnings per share of AABA, I decide that I will forward fill so as to ensure that there will be no gaps in due to data i.e. using the latest data and using it for data that we have yet to received and replace all the remaining NA values with zero for simiplicity sake given that they are minimal.

Next, I sorted and printed all the tickers that has their stock split and when they were split.

```
In [17]: print("Below we see all the stock splits for all the shares for our data set:")
         stock_splits = pivot.xs('split_coefficient',axis=1).stack().sort_values(ascending=False)
         stock_splits = stock_splits[stock_splits!=1].swaplevel()
         stock_splits =  stock_splits[stock_splits.values>0]
         print(stock_splits)
         ticker_sample = stock_splits.index.get_level_values("tickers")[0]
```

```
Below we see all the stock splits for all the shares for our data set:
tickers  date
SSNC     2016-06-27    2.0000
HPE      2017-04-03    1.3348
CTXS     2017-02-01    1.2558
dtype: float64
```

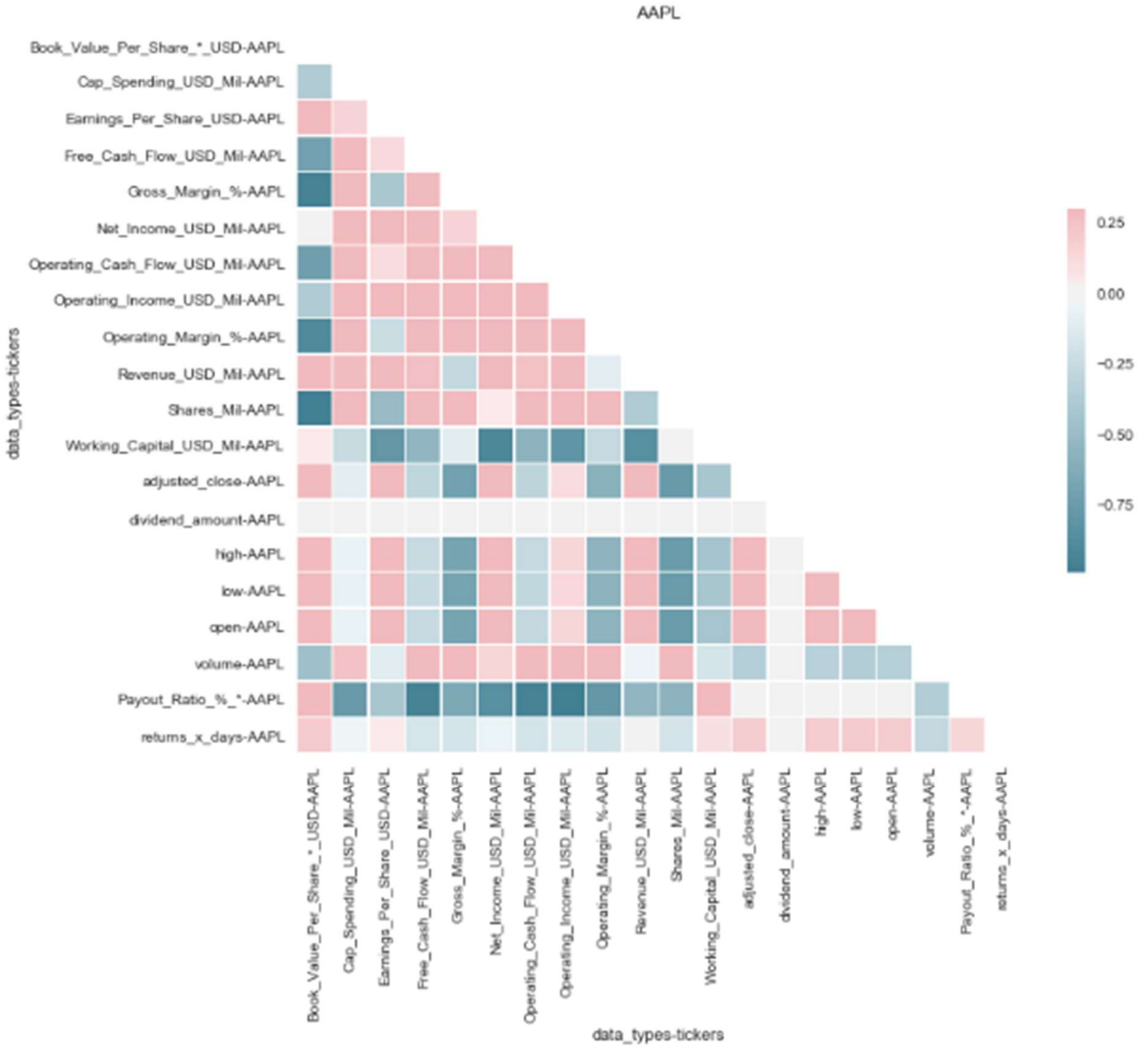
We note that 3 tickers namely SSNC, HPE and CTXS has had stock splits over the years and hence their data would be skewed and we can see the impact of a share split on the closing price of stock prices as seen below.



This is particularly evident in the data we have downloaded from Alpha Vantage. Thus, I decided to divide all the values before the stock split by the stock split ratio and hence making it consistent. We can easily see that upon the adjustment, the 4 variables namely ['high', 'low', 'open', 'volume'] are much smoother as seen in Appendix 1.

Visualisation:

Given the multitude of variables and cross variables available, I decide that I will be focusing on a single ticker to start off the visualisation.



First and foremost, we noticed something that is rather unique. The data seems to negatively correlate with one another more than they are positively correlated with one another as seen by the skew in the colour scale where positive correlations goes up to 0.25 but the negative correlations goes as low as -0.75.

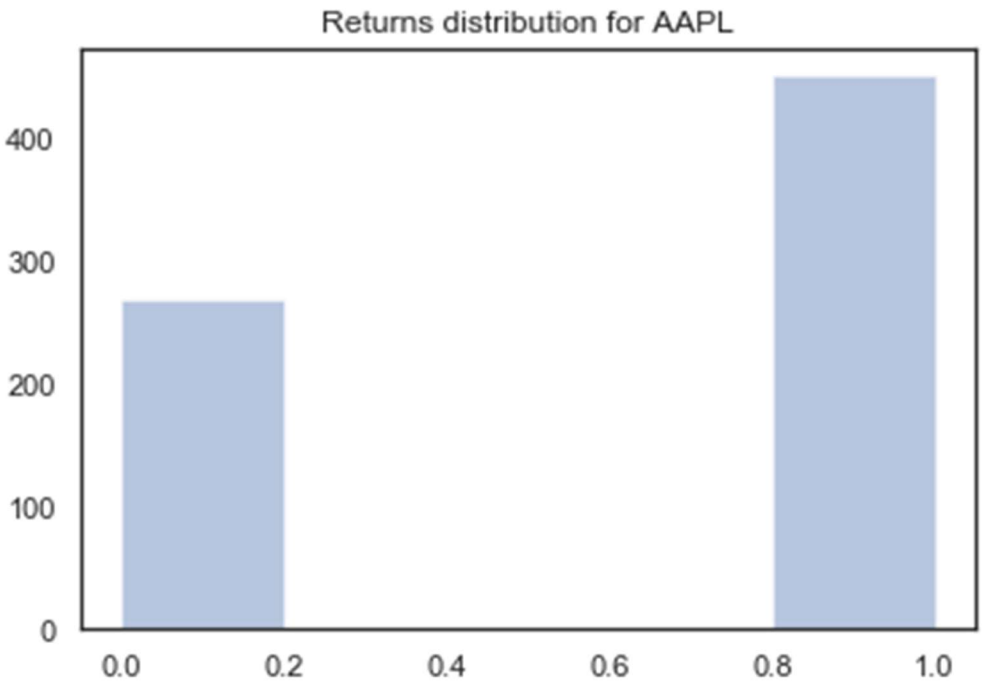
Secondly, we can see that Payout Ratio and Working Capital correlates negatively and strongly with the other variables and dividend amount has almost no correlation with any other variables. Furthermore, we see that our target variable “returns_x_days” does not seem to have very strong correlation with any other variables. This is confirmed by taking a specific look at the output below where the strongest negative correlation is with volume and the strongest positive correlation is with Book Value and daily low prices.

```
In [65]: aapl.corr()['returns_x_days'].sort_values("AAPL")
```

Out[65]:

	tickers	AAPL
data_types	tickers	
volume	AAPL	-0.28
Operating_Margin_%	AAPL	-0.19
Gross_Margin_%	AAPL	-0.18
Shares_Mil	AAPL	-0.18
Operating_Cash_Flow_USD_Mil	AAPL	-0.18
Free_Cash_Flow_USD_Mil	AAPL	-0.17
Operating_Income_USD_Mil	AAPL	-0.12
Net_Income_USD_Mil	AAPL	-0.06
Cap_Spending_USD_Mil	AAPL	-0.04
dividend_amount	AAPL	0.01
Revenue_USD_Mil	AAPL	0.02
Earnings_Per_Share_USD	AAPL	0.04
Working_Capital_USD_Mil	AAPL	0.10
Payout_Ratio_%_*	AAPL	0.15
high	AAPL	0.19
open	AAPL	0.19
adjusted_close	AAPL	0.19
Book_Value_Per_Share_*_USD	AAPL	0.20
low	AAPL	0.20
returns_x_days	AAPL	1.00

We can see that the distribution between positive returns and negative/ zero returns 30 days are generally distributed in a 60:40 split for Apple which is rather balanced.



Algorithm

I chose to use a neural network to help predict the returns. Thus first and foremost, I would need to normalise the data. This is done via the min-max scalar where all the values are scaled between 1 and 0 where 1 is the max of the number in the series and 0 is the smallest number of the series. I chose this method of scaling over some other options like standardising the data using the standard deviation of the sample as I did not wish to make any assumption about the shape and structure of each variable.

Given that the data are all highly correlated, I decided that it is a lot more useful to construct the layers of the neural network as dense layers. This allows all the information provided by each node to impact the end result without any bias. I included a layer of dropout after each dense layer so as to avoid overfitting and improve the robustness of the data. However, I have to flatten each input layer. This also means that the information containing in the time dimension gets diluted/ lost.

We can see that the average accuracy we had on the random forest classifier comes in at about 70.55% but using a simple 2 layered dense neural network, we can easily achieve a much higher accuracy of 86.67%.

However, due to flattening of the data, I believe there is data that is lost in the temporal dimension which means that I am not properly utilising our full data set. Thus, I created a second model using the Long Short Term Memory layer which is a kind of recurrent neural network that attempts to model time and series dependent behaviour (given that more recent information should always be given higher priority compared to a flattened Dense layer where all nodes are “equal”) . This resulted in a model that trained significantly longer, same degree of accuracy but slightly higher mean squared error.

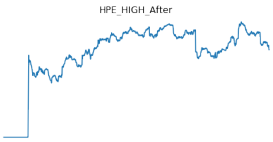
Similarly, I use a 20% dropout rate for the LSTM model so as to ensure that we do not overfit the data and thus lowering its performance in real-world scenarios.

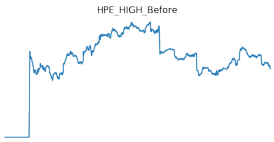
All in all, after all the training and testing we can see that both LSTM and a simple Dense neural network outperforms that of a random forest classifier.

Appendix 1


SSNC_HIGH_After


SSNC_HIGH_Before


HPE_HIGH_After


HPE_HIGH_Before


CTXS_HIGH_After


CTXS_HIGH_Before


SSNC_LOW_After


SSNC_LOW_Before


HPE_LOW_After


HPE_LOW_Before


CTXS_LOW_After


CTXS_LOW_Before


SSNC_OPEN_After

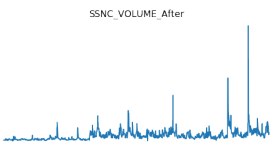

SSNC_OPEN_Before

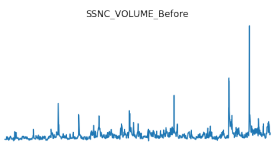

HPE_OPEN_After

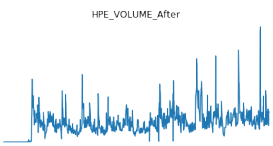

HPE_OPEN_Before

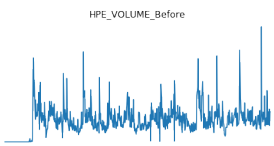

CTXS_OPEN_After

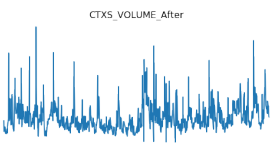

CTXS_OPEN_Before

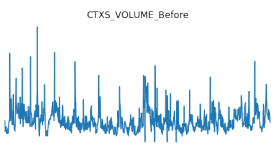

SSNC_VOLUME_After


SSNC_VOLUME_Before


HPE_VOLUME_After


HPE_VOLUME_Before


CTXS_VOLUME_After


CTXS_VOLUME_Before

References

https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib

https://stackoverflow.com/questions/18770504/resize-ipython-notebook-output-window

https://www.somebits.com/~nelson/pandas-multiindex-slice-demo.html

https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/

https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-lstms-807fa6ca7f

https://stackoverflow.com/questions/39674713/neural-network-lstm-input-shape-from-dataframe