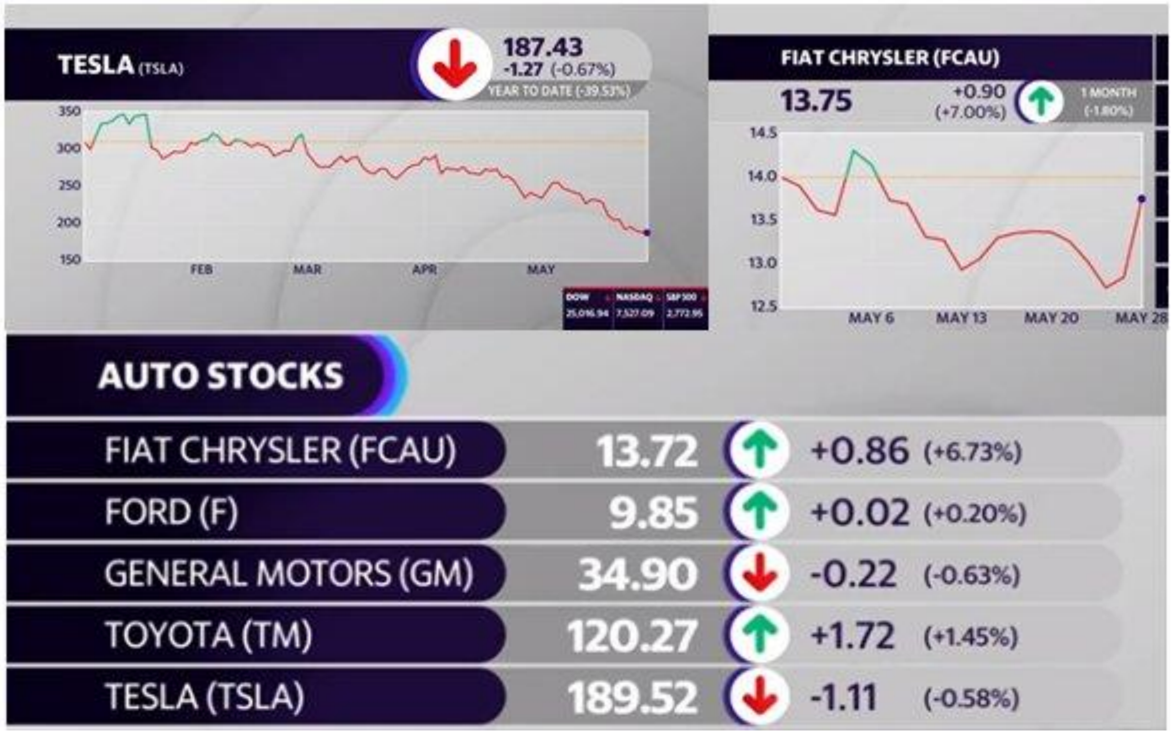
Financial Data Analysis & Prediction of Stock Prices using Recurrent Neural Network

Rini Christy

May 30, 2019



FIAT CHRYSLER (FCAU)	13.72	+0.86	(+6.73%)
FORD (F)	9.85	+0.02	(+0.20%)
GENERAL MOTORS (GM)	34.90	-0.22	(-0.63%)
TOYOTA (TM)	120.27	+1.72	(+1.45%)
TESLA (TSLA)	189.52	-1.11	(-0.58%)

Table of Contents

Introduction:

Brownian motion, a phenomenon, named after the English botanist Robert Brown, was originally observed as the irregular random motion exhibited by tiny particles when they are completely immersed in a gas or liquid. Louis Jean-Baptiste Alphonse Bachelier a French mathematician at the turn of the 20th century introduced for the first time the idea of an advanced mathematical model of Brownian motion and its use for valuing stock options by trying to model stock prices on the Paris stock exchange way back in 1900 using the idea of a Brownian motion as part of his PhD thesis, "The Theory of Speculation" (Théorie de la spéculation, published 1900). Bachelier applied the concept of Brownian motion which had been used throughout the physical sciences and engineering, into a financial context to get more understanding about stock price modelling and hence he is considered as the forefather of mathematical finance and a pioneer in the study of stochastic processes. In 1905, Albert Einstein first explained the Brownian phenomenon as due to the bombardment of immersed pollen by the molecules in the surrounding liquid medium. It is also called the Weiner process, as the mathematical theory behind this motion was first developed by Norbert Weiner in the 1920s. Since then it has been applied effectively in such areas as statistical goodness of fit, analyzing stock price fluctuations caused by bombardment of internal and external variables. According to the geometric Brownian motion model the future price of financial stocks has a lognormal probability distribution and therefore their future value can be estimated with a certain level of confidence.

The goal of this project is to predict these Brownian movements of 6 future automobile stock prices. The focus is on exploratory data analysis of past stock prices and prediction of trend in future stock prices using Recurrent Neural Network (abbreviated as RNN). This project is just meant to show the application of recurrent neural network on financial analysis without the use of hardcore mathematical functions. It is not meant to be a robust financial analysis or be taken as financial advice.

Data Collection:

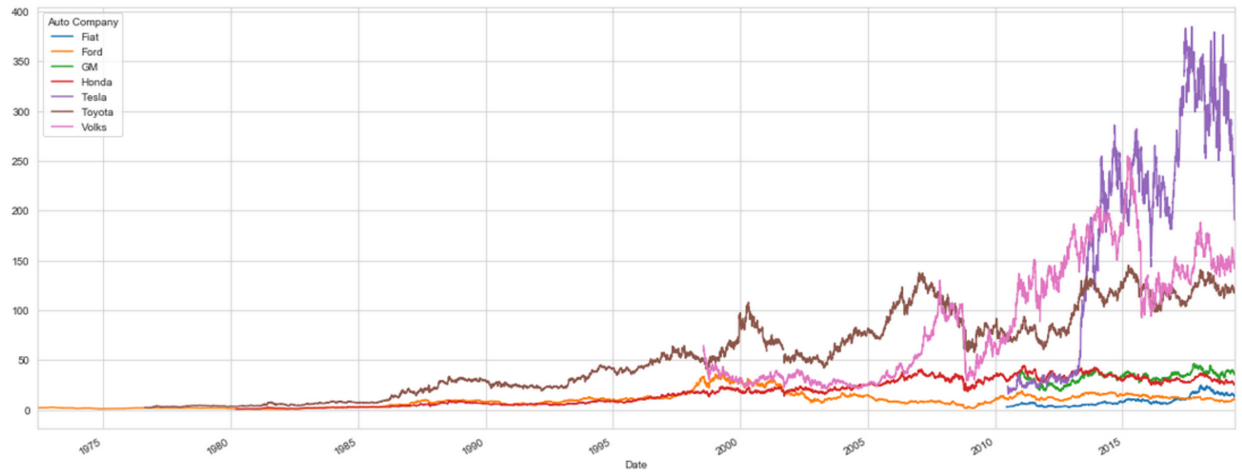
Seven automobile stocks are selected to see how they progressed throughout the financial ups and downs all the way till May 24, 2019. Data is collected directly from Yahoo finance using pandas-datareader to read stock information directly from the internet. The stock information for the following car companies is gathered:

Fiat Chrysler Automobiles N.V. (FCAU)
Ford Motor Company (F)
General Motors Company (GM)
Honda Motor Co., Ltd. (HMC)
Tesla, Inc. (TSLA)
Toyota Motor Corporation (TM)
Volkswagen AG (VOW3.DE)

```
auto_stocks.tail()
```

Auto Ticker	Fiat						Ford				...	Toyota				Volks		
Stock Info	**High**	**Low**	**Open**	**Close**	**Volume**	**Adj Close**	**High**	**Low**	**Open**	**Close**	**...**	**Open**	**Close**	**Volume**	**Adj Close**	**High**	**Low**	**Open**
Date																		
2019-05-20	13.40	13.14	13.21	13.37	4897900.0	13.37	10.30	10.20	10.30	10.28	...	118.120003	118.110001	156300.0	118.110001	147.339996	143.699997	146.2
2019-05-21	13.34	13.18	13.27	13.26	3815100.0	13.26	10.31	10.15	10.31	10.24	...	117.849998	118.250000	124300.0	118.250000	146.860001	143.800003	145.6
2019-05-22	13.26	13.02	13.22	13.03	3364300.0	13.03	10.21	9.93	10.17	9.97	...	117.779999	117.320000	107600.0	117.320000	145.160004	142.279999	144.3
2019-05-23	12.81	12.61	12.75	12.73	5029000.0	12.73	9.85	9.67	9.85	9.85	...	117.379997	117.470001	116500.0	117.470001	142.500000	140.619995	142.1
2019-05-24	12.93	12.82	12.90	12.85	3965400.0	12.85	9.95	9.80	9.92	9.83	...	118.839996	118.550003	121200.0	118.550003	145.100006	142.460007	144.4

5 rows × 42 columns



The plot shows how the price of a financial auto stocks varies over time. The horizontal axis represents the date in years. The vertical axis is the daily closing price of auto stocks from their respective initial public offering (IPO) till date. The price behavior shows the same behavior as a stochastic process called “Brownian motion”.

Exploratory Data Analysis

The statistical analysis of all indicator prices for each auto stock throughout the time period is calculated.

Auto Company	Fiat	Ford	GM	Honda	Tesla	Toyota	Volks
count	2256.000000	11849.000000	2142.000000	9883.000000	2242.000000	10786.000000	5288.000000
mean	8.731031	9.250980	32.896844	17.873856	177.336958	54.718070	88.701663
std	5.262561	7.205074	5.613092	12.526928	114.930065	41.313122	58.740566
min	2.802632	0.643736	18.799999	0.878125	15.800000	2.246658	20.924101
25%	4.868421	2.349382	29.940001	5.593750	33.500001	18.552876	33.100899
50%	6.920000	8.758861	33.955000	18.015625	206.844994	50.250000	73.845848
75%	10.791513	12.499624	36.610001	29.879999	262.699997	83.618750	138.762501
max	24.809999	36.647751	46.480000	44.490002	385.000000	145.320007	255.199997

Returns:

The practical way to estimate empirically the volatility of a stock is to observe the historical data at fixed intervals of time, for example the daily closing price. The volatility is a constant characteristic of a stock, expressed as daily percentage or 30 day averages. It gives an idea about the stability of stock price. Relatively high volatility means that the stock price varies continuously within relatively large interval. The most usual method of measuring the stock volatility is the standard deviation of the price returns.

Pandas pct_change() method on the Close column is applied to create a column representing the return value. This is calculated employing the following formula:

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1$$

Date	Fiat Returns	Ford Returns	GM Returns	Honda Returns	Tesla Returns	Toyota Returns	Volks Returns
2019-05-20	-0.098449	-0.000972	-0.000811	-0.003902	-0.026868	-0.002449	-0.015767
2019-05-21	-0.008227	-0.003891	0.004328	0.007050	-0.001363	0.001185	-0.000829
2019-05-22	-0.017345	-0.026367	-0.042553	-0.006612	-0.060220	-0.007865	-0.002764
2019-05-23	-0.023024	-0.012036	-0.011814	-0.010180	0.014321	0.001279	-0.015939
2019-05-24	0.009427	-0.002031	-0.000285	0.006329	-0.024861	0.009194	0.006056

Using this 'returns' DataFrame, figure out on what dates each auto stock had the best and worst single day returns.

```
# Worst Drop
returns.idxmin()
```

```
Fiat Returns      2011-11-07
Ford Returns      2008-11-19
GM Returns        2011-11-09
Honda Returns     1987-10-19
Tesla Returns     2012-01-13
Toyota Returns    2008-11-06
Volks Returns     2015-09-22
```

```
# Best Single Day Gain
returns.idxmax()
```

```
Fiat Returns      2011-10-07
Ford Returns      2008-11-26
GM Returns        2018-05-31
Honda Returns     2008-10-28
Tesla Returns     2013-05-09
Toyota Returns    1987-10-21
Volks Returns     2008-10-28
```

Notice that two of the auto companies, Fiat & GM share have the worst one-day percentage drops couple of days apart. Similarly, Ford and Toyota had their worst drops in the month of November, 2008. This was in accordance to the automotive industry crisis of 2008–2010, which itself was a part of a great global financial downturn with October sales of cars and light trucks in the United States dropping swiftly in 2008 with Ford falling 30%, and Toyota falling 23% (Wikipedia, "Global financial crisis in November 2008"). Much of the falloff in sales was attributable to customers being unable to arrange financing due to unemployment, continued declines in housing prices, and the aftershocks of the Wall Street financial crisis. Ironically, Honda & Volkswagen had their best single day gain on 28th October 2008, just before the start of great recession in November 2008.

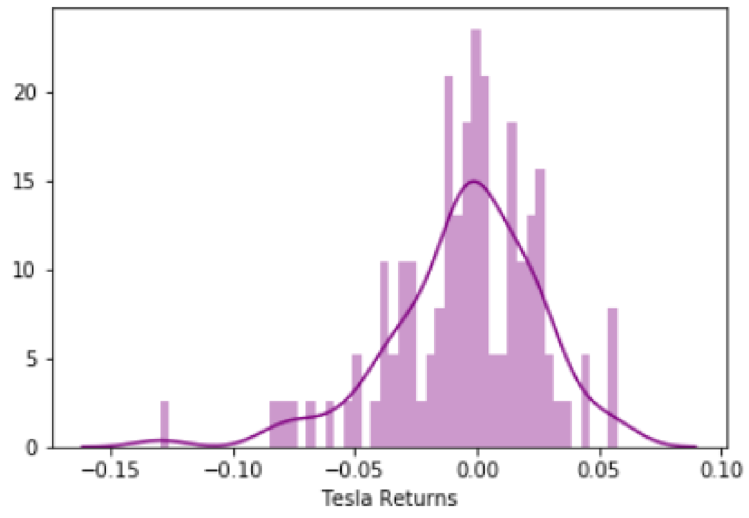
	Fiat Returns	Ford Returns	GM Returns	Honda Returns	Tesla Returns	Toyota Returns	Volks Returns
count	2313.000000	11983.000000	2197.000000	10017.000000	2299.000000	10920.000000	5379.000000
mean	0.001021	0.000365	0.000169	0.000518	0.001416	0.000518	0.000434
std	0.027443	0.021881	0.017737	0.019214	0.032153	0.018324	0.023878
min	-0.157143	-0.250000	-0.109026	-0.163952	-0.193274	-0.165236	-0.198185
25%	-0.008596	-0.010705	-0.008977	-0.009740	-0.014411	-0.008563	-0.011243
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.009793	0.010774	0.009156	0.010000	0.017379	0.008798	0.012301
max	0.178947	0.295181	0.128734	0.196109	0.243951	0.193548	0.196886

Take a look at the standard deviation of the returns, Tesla stock should be classified as the riskiest over the entire time period, with the highest standard deviation of 0.032153. This risky behavior of Tesla profile is continued in the current year too, as shown below:

```
Statistical analysis of auto stocks returns since January, 2019
```

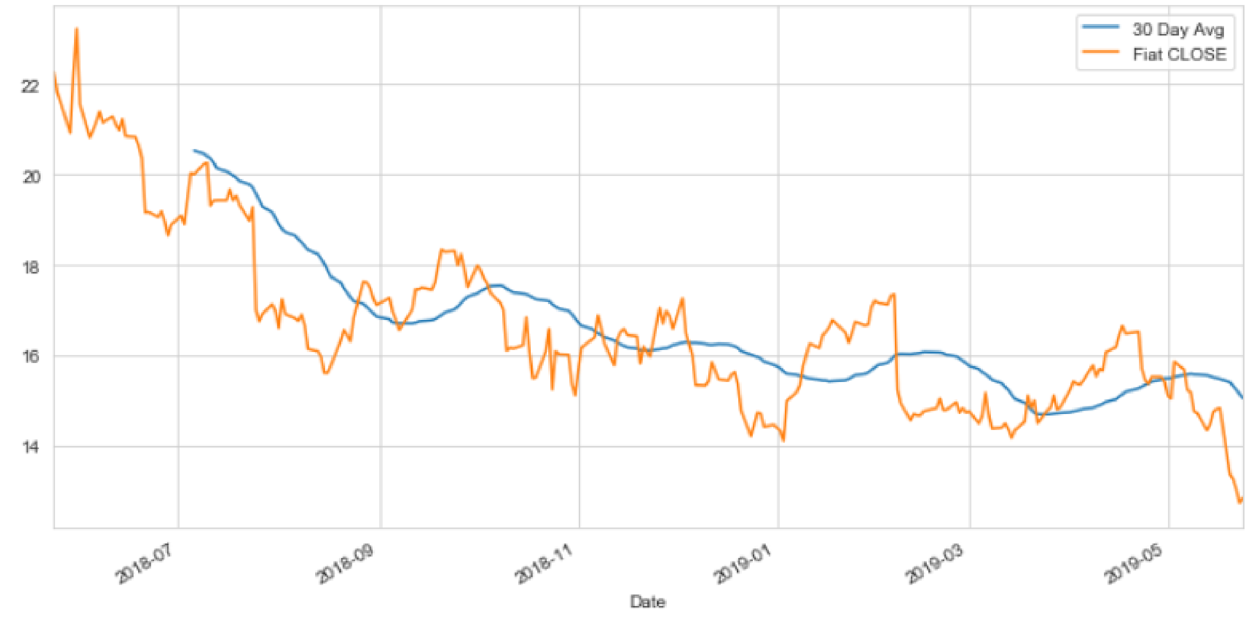
	Fiat Returns	Ford Returns	GM Returns	Honda Returns	Tesla Returns	Toyota Returns	Volks Returns
count	102.000000	102.000000	102.000000	102.000000	102.000000	102.000000	102.000000
mean	-0.000875	0.002654	0.000594	-0.000307	-0.004962	0.000254	0.000396
std	0.023485	0.019910	0.015323	0.012279	0.030933	0.009852	0.015717
min	-0.122190	-0.062217	-0.042553	-0.031260	-0.129711	-0.024288	-0.047389
25%	-0.011065	-0.008272	-0.005897	-0.006211	-0.019794	-0.006003	-0.010919
50%	0.001371	0.001165	0.001280	0.001292	-0.001886	0.000492	-0.000406
75%	0.011635	0.012921	0.009493	0.006826	0.014188	0.006294	0.010444
max	0.062367	0.107447	0.070544	0.045559	0.057697	0.044309	0.042446

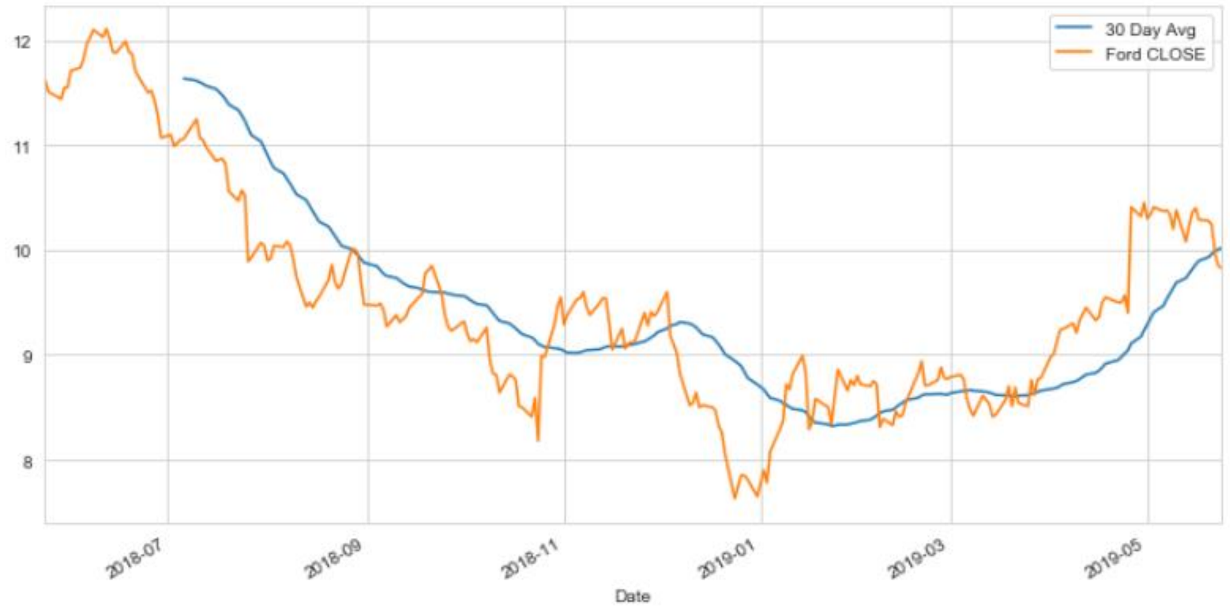
The risky profile is displayed using seaborn distplot of the 2019 returns for Tesla.

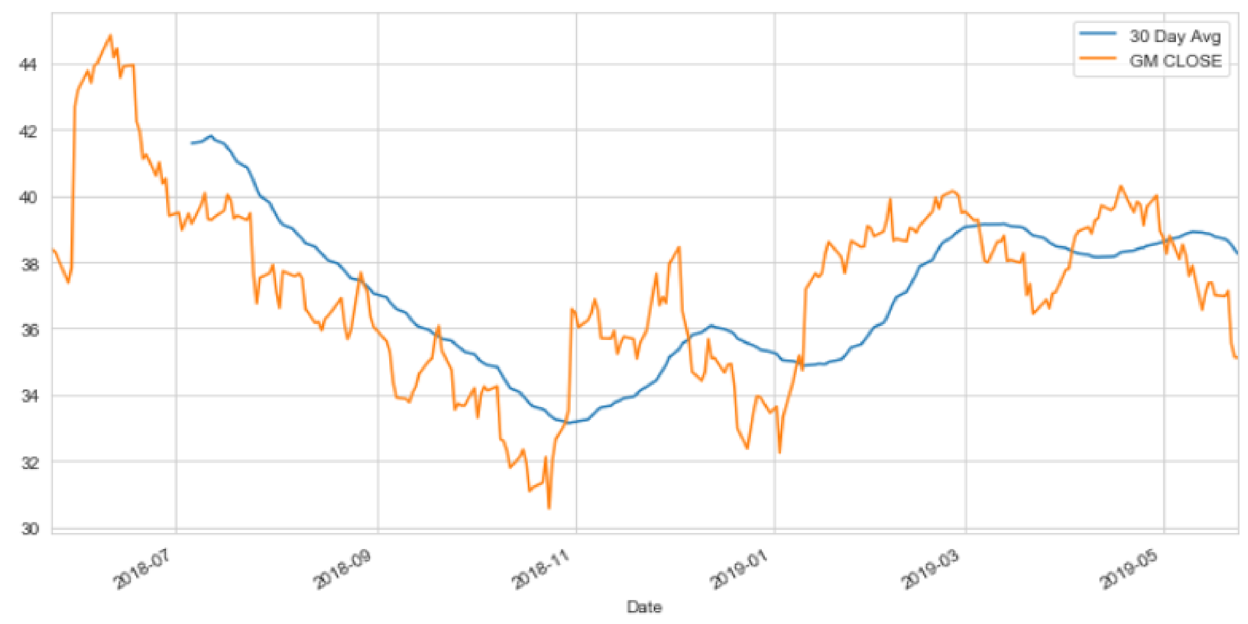


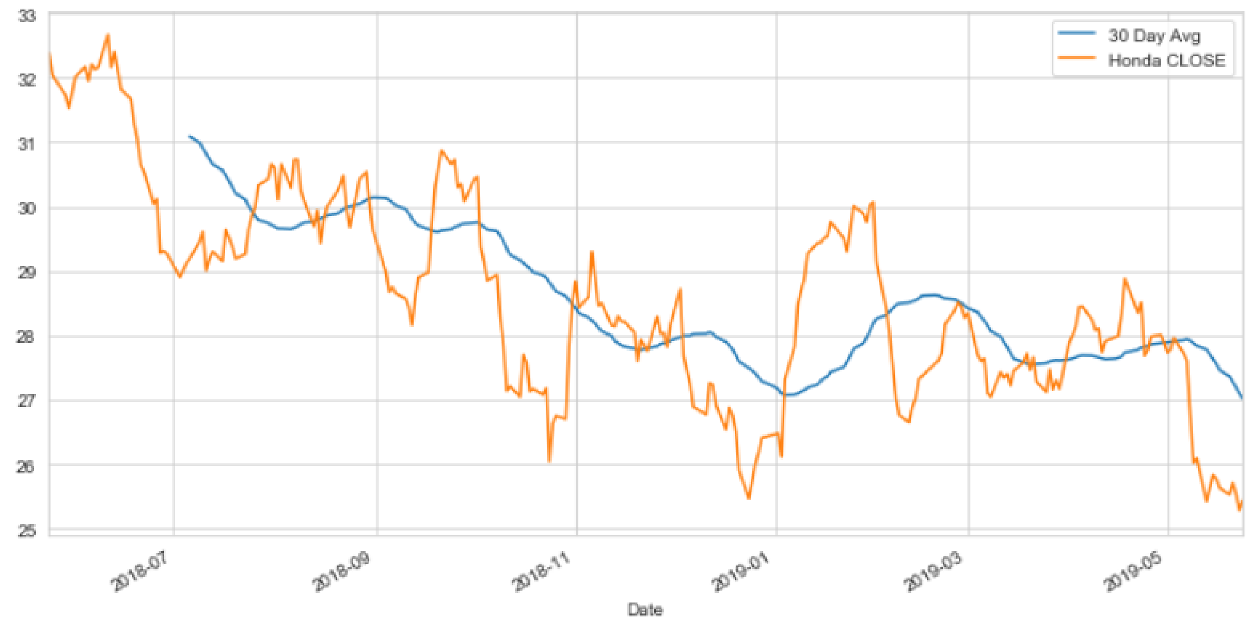
Moving Averages

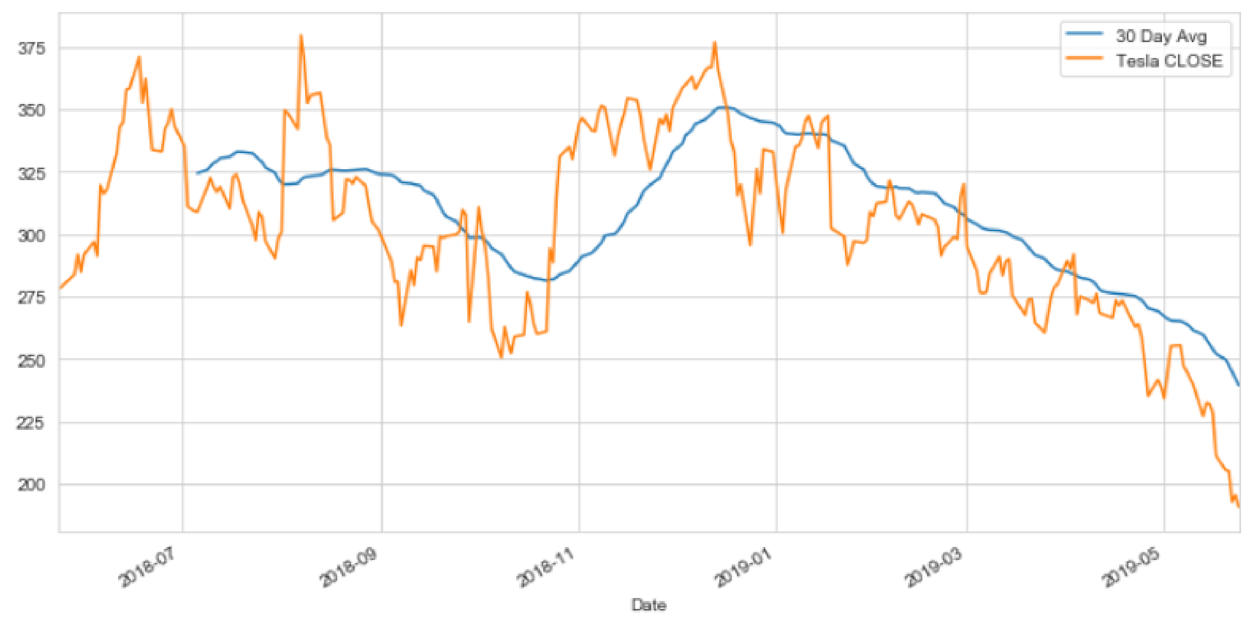
To analyze the moving averages for these stocks the rolling 30 day average against the Close Price for auto stock for the past one year are plotted.

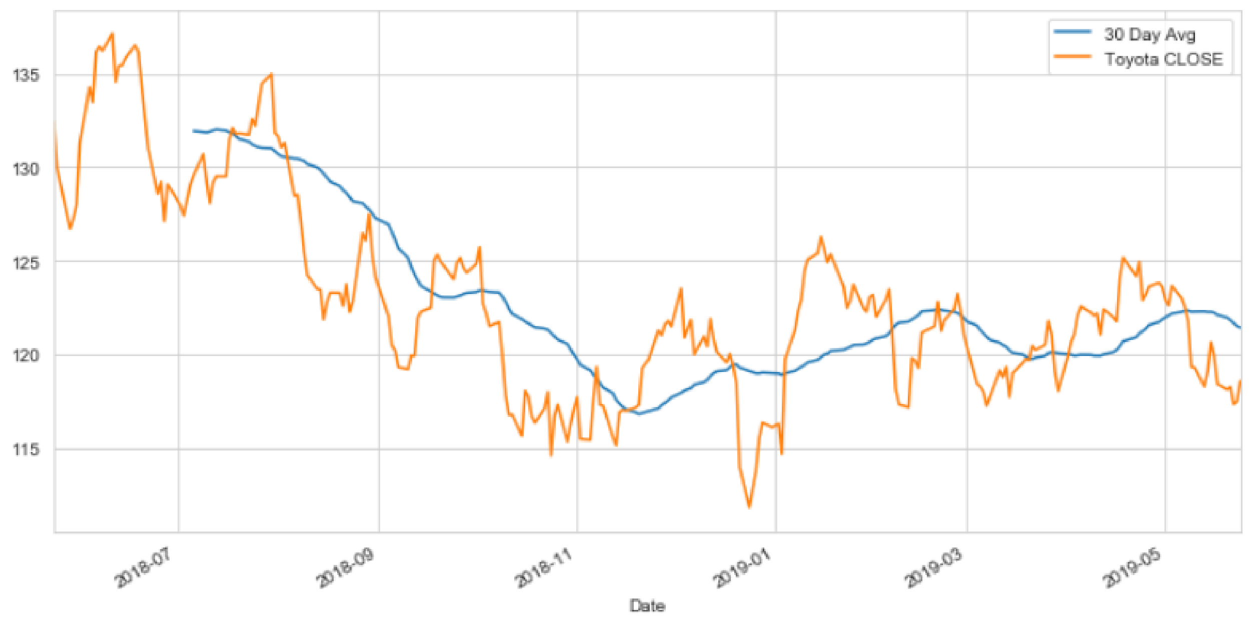


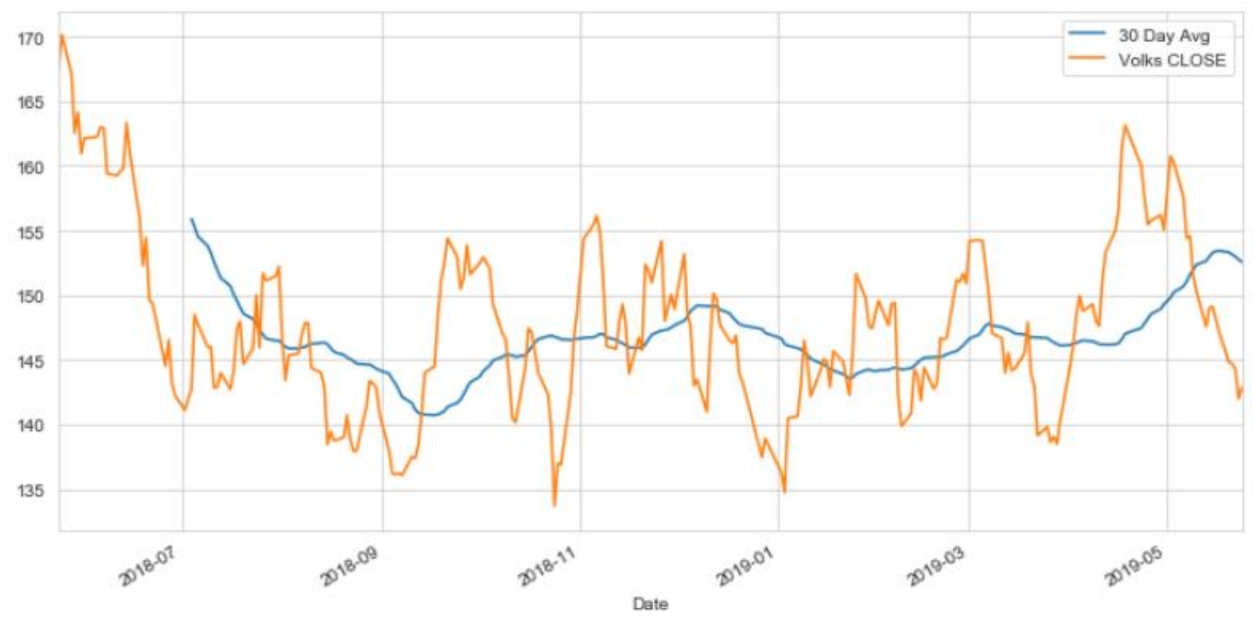



30 Day Avg
GM CLOSE
44
42
40
38
36
34
32
30
2018-07
2018-09
2018-11
2019-01
2019-03
2019-05
Date

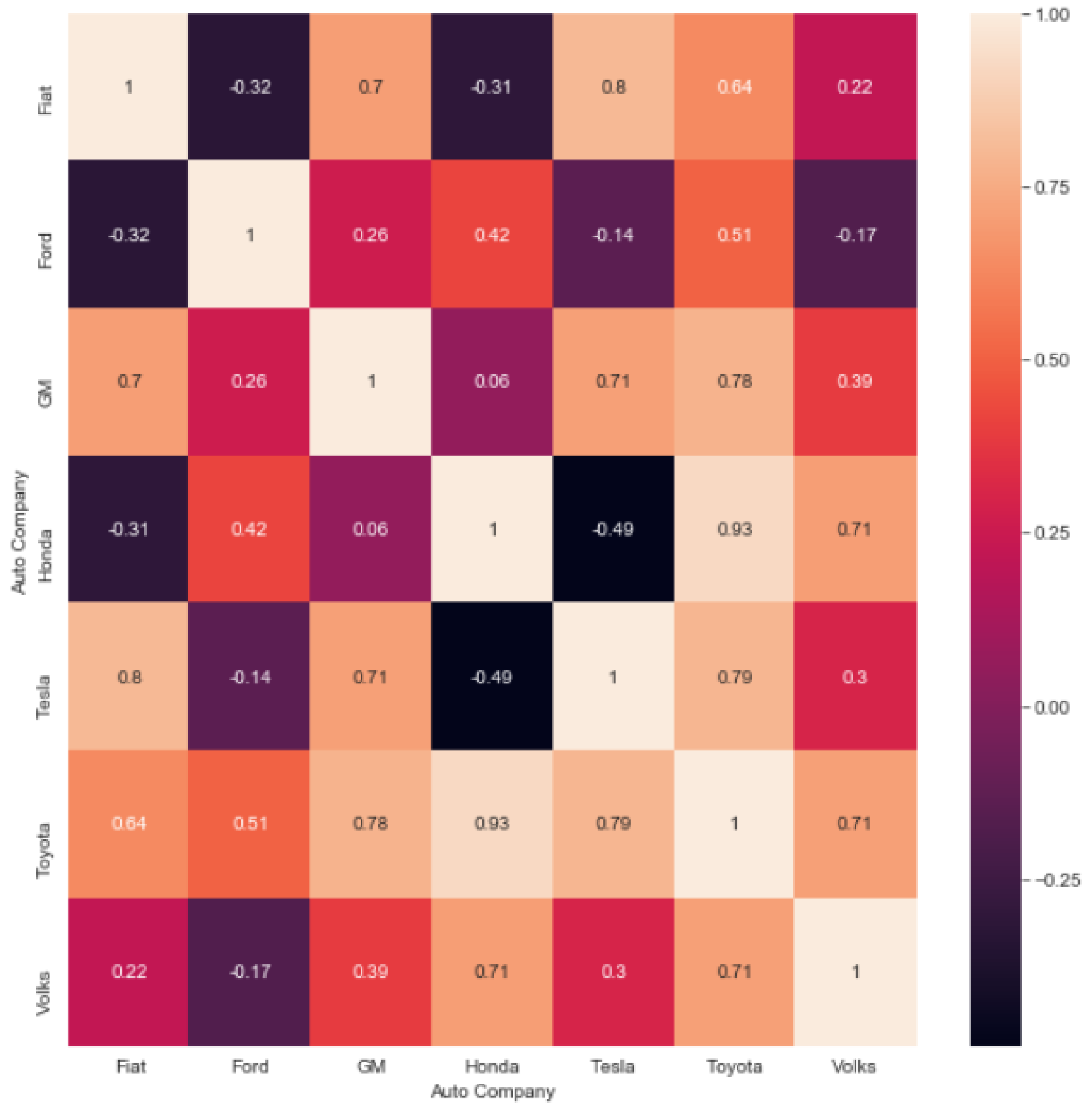

30 Day Avg
Honda CLOSE
33
32
31
30
29
28
27
26
25
2018-07
2018-09
2018-11
2019-01
2019-03
2019-05
Date


30 Day Avg
Tesla CLOSE
375
350
325
300
275
250
225
200
2018-07
2018-09
2018-11
2019-01
2019-03
2019-05
Date


30 Day Avg
Toyota CLOSE
135
130
125
120
115
2018-07
2018-09
2018-11
2019-01
2019-03
2019-05
Date



Heat Map
A heatmap of the correlation between the Close Price stock values is also generated.



	Fiat	Ford	GM	Honda	Tesla	Toyota	Volks
Fiat	1	-0.32	0.7	-0.31	0.8	0.64	0.22
Ford	-0.32	1	0.26	0.42	-0.14	0.51	-0.17
GM	0.7	0.26	1	0.06	0.71	0.78	0.39
Honda	-0.31	0.42	0.06	1	-0.49	0.93	0.71
Tesla	0.8	-0.14	0.71	-0.49	1	0.79	0.3
Toyota	0.64	0.51	0.78	0.93	0.79	1	0.71
Volks	0.22	-0.17	0.39	0.71	0.3	0.71	1

Honda and Toyota both Japanese automobile manufactures shows a very high significant positive correlation.

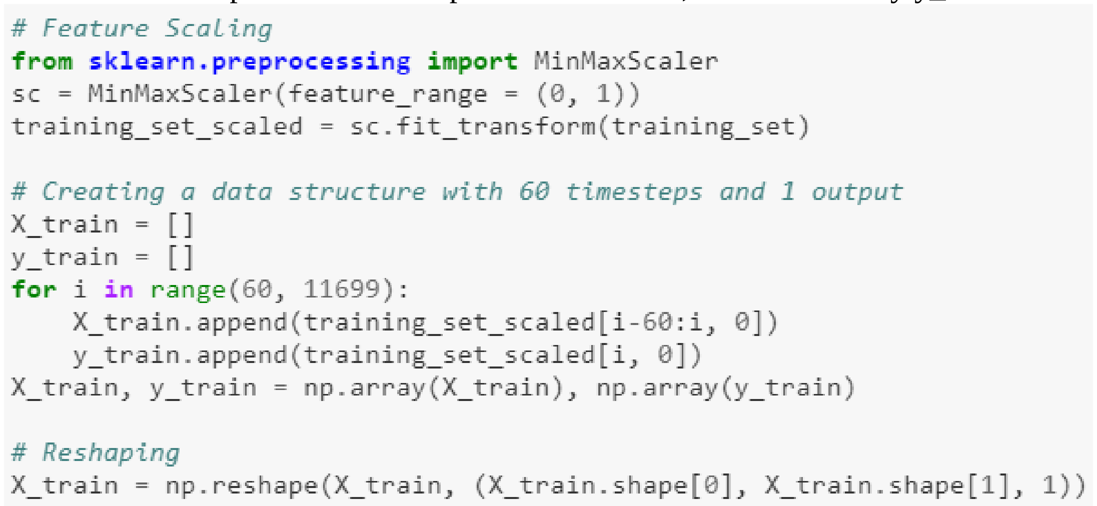
Prediction of Stock prices trend using Recurrent Neural Network

The Brownian motion describes the motion exhibited by particles immersed in a gas or liquid. The particles were essentially being bombarded by molecules present within the matter causing displacement or movement in an unpredictable random direction. In the real world of financial markets, investors and financial analysts are generally more interested in the profit or loss of the stock over a period of time i.e. the increase or decrease in the price, than in the price self. Therefore modelling a behavior of a stock price can be made through its relative change in the time.

The training set on which the RNN will be trained will be containing the input data of the neural network. iloc method is used to get the right index of the column which is the close stock price to be analyzed. Feature scaling by normalization is applied whenever an RNN is built with a sigmoid function as the activation function in the output layer of neural network.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Time steps: The heart of the Recurrent Neural Network is to create a special data structure with for example 60 time steps and one output. At each time ‘t’, the RNN is going to look at the 60 stock prices before time ‘t’, that is the stock prices between 60 days before time ‘t’ and time ‘t’, and based on the trends, it will try to predict the next output. So 60 time steps of the past information from which the RNN try to learn and understand some correlations, or some trends, and based on its understanding, it's going to try to predict the next output labelled as y_train , the stock price at time ‘t+1’. Thus, X_train contains all the stock prices of 60 previous stock prices before t equal 60, and based on all the stock prices of each line, the Recurrent Neural Network will be trained to predict the stock price at time ‘t+1’, which is exactly y_train.



```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)

# Creating a data structure with 60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, 11699):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

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

The architecture of RNN is structured with five Long Short - Term Memory (LSTM) layers using LSTM class and some Dropout regularization to avoid overfitting. To construct a high dimensional LSTM model to capture the trends of a pretty complex stock price, the model needs to have a large number of neurons in each of the multiple LSTM layers. And therefore, number of units is set to be equal to 50. The parameter return_sequences should be set to be equal to True if another layer is following, else it should be equal to the default False value. The final argument, input_shape, contains the two dimensions corresponding to the time steps and the indicators; x_train.shape[1] corresponding to the time steps, and one corresponding to the predictors, the indicators, that could help predict the stock price trends.

Dropout rate, which is the rate of neurons to be ignored in the layers to do the regularization, is taken as 0.2, corresponding to exactly 20% of the neurons of the LSTM layer during the forward propagation and back propagation happening in each iteration of the training.

Four more LSTM layers are added further so as to make a big, stacked LSTM model.

The model is then compiled with ‘adam’ optimizer and ‘mean squared error’ loss function to deal with a regression problem of prediction of a continuous value. This completes the construction of an intelligent, trained, neural network, to be able to predict the upward and downward trends of the various automobile company stock prices. The model is then fit to the training set over the chosen number of epochs and batch sizes for each stock depending on the convergence of the maximum accuracy or minimum loss.

```
# Part 2 - Building the RNN

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

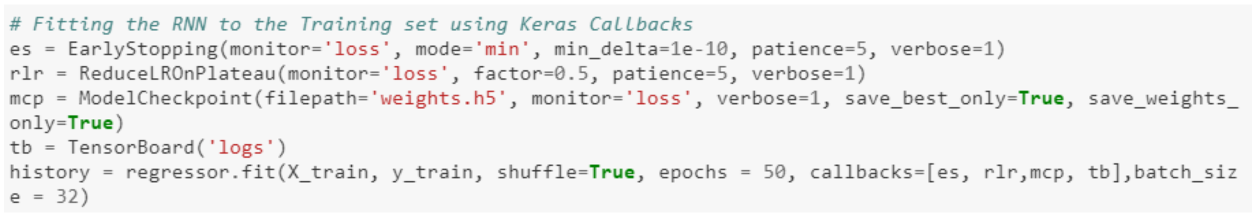
Learning rate: Keras provides the ReduceLROnPlateau that will adjust the learning rate when a plateau in model performance is detected, e.g. no change for a given number of training epochs. This callback is designed to reduce the learning rate after the model stops improving with the hope of fine-tuning model weights. The *ReduceLROnPlateau* requires the user to specify the metric to monitor during training via the "*monitor*" argument, the value that the learning rate will be multiplied by via the "*factor*" argument and the "*patience*" argument that specifies the number of training epochs to wait before triggering the change in learning rate.

EarlyStopping: To reduce overfitting of the training dataset led by excess number of epochs, and under-fitting by too few, early stopping is configured into the model. It is a method that allows the user to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a validation dataset.

Arguments

- monitor: quantity to be monitored. The "monitor" allows the user to specify the performance measure to monitor in order to end training.

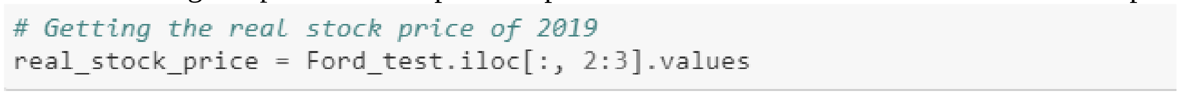
- min_delta: minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement. A change in improvement in a specific increment, such as 1 unit for mean squared error or 1% for accuracy.
- patience: number of epochs with no improvement after which training will be stopped.
- verbose: verbosity mode
- mode: one of {auto, min, max}. In min mode, training will stop when the quantity monitored has stopped decreasing; in max mode it will stop when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity. a minimum for validation loss and for validation mean squared error, whereas a maximum for validation accuracy. mode can be set to 'auto' to minimize loss or maximize accuracy
- baseline: Baseline value for the monitored quantity to reach. Training will stop if the model doesn't show improvement over the baseline.
- restore_best_weights: whether to restore model weights from the epoch with the best value of the monitored quantity. If False, the model weights obtained at the last step of training are used.



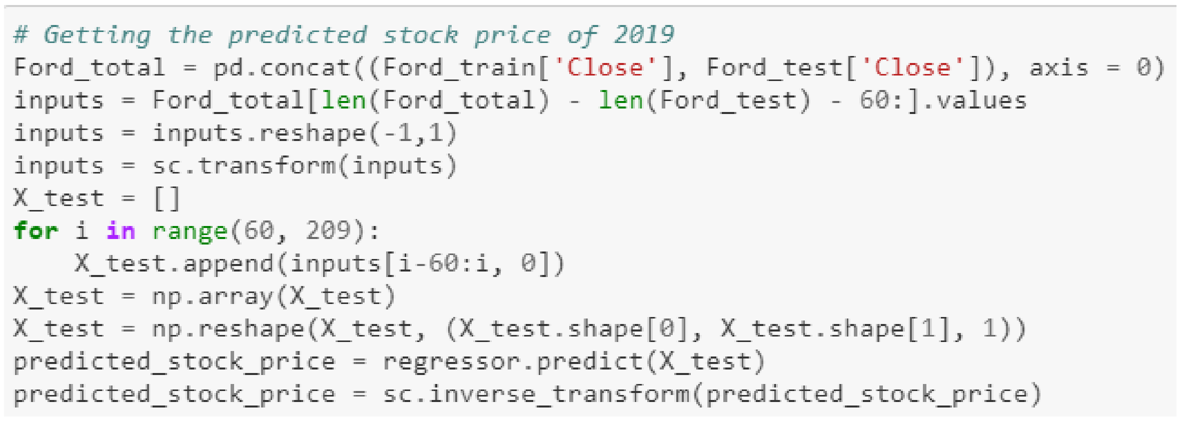
```
# Fitting the RNN to the Training set using Keras Callbacks
es = EarlyStopping(monitor='loss', mode='min', min_delta=1e-10, patience=5, verbose=1)
rlr = ReduceLROnPlateau(monitor='loss', factor=0.5, patience=5, verbose=1)
mcp = ModelCheckpoint(filepath='weights.h5', monitor='loss', verbose=1, save_best_only=True, save_weights_
only=True)
tb = TensorBoard('logs')
history = regressor.fit(X_train, y_train, shuffle=True, epochs = 50, callbacks=[es, rlr,mcp, tb],batch_siz
e = 32)
```

The model is thus trained using 50 epochs and batch sizes of 32 with reduce learning rate and early stopping call backs applied.

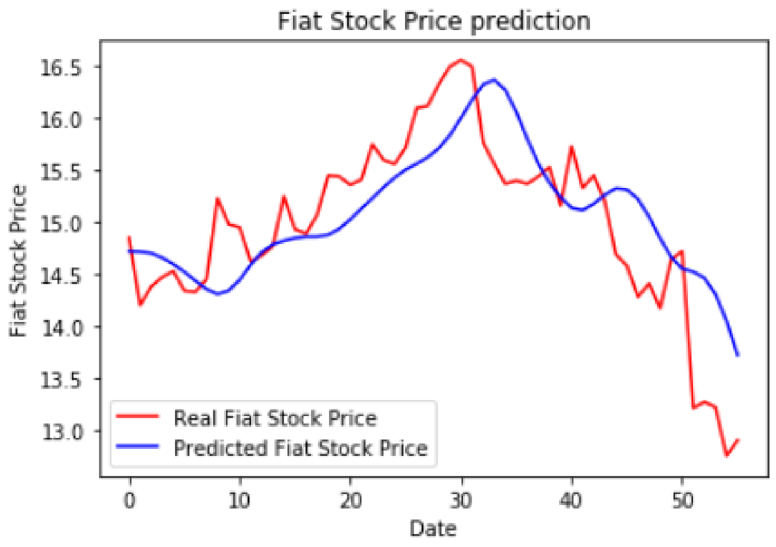
The model is now ready to predict the future stock prices. For testing the accuracy, the trained model is run on test set generated from recent stock values up until May 24th 2019. The results can be visualized using Matplotlib to compare the predicted values with the real values of stock prices.



```
# Getting the real stock price of 2019
real_stock_price = Ford_test.iloc[:, 2:3].values
```



```
# Getting the predicted stock price of 2019
Ford_total = pd.concat((Ford_train['Close'], Ford_test['Close']), axis = 0)
inputs = Ford_total[len(Ford_total) - len(Ford_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 209):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

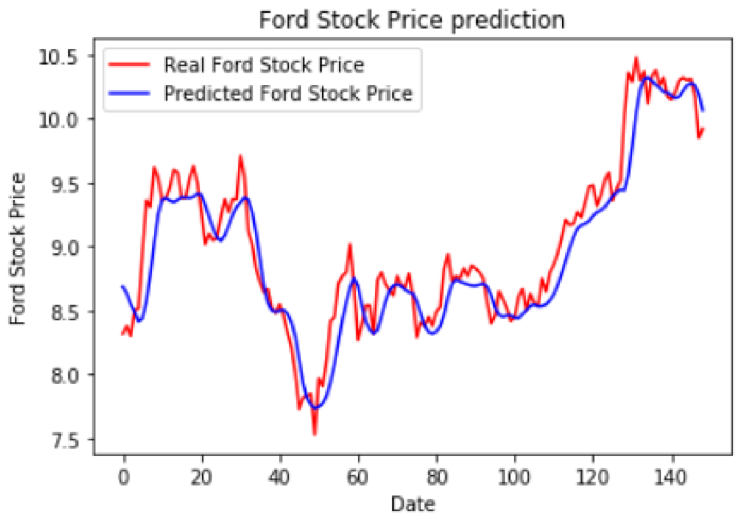


```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
0.5593707640464152
```

```
min_val=min(real_stock_price)
max_val=max(real_stock_price)
print ("relative error: ", rmse/(max_val-min_val))
```

```
relative error:  [0.14681649]
```

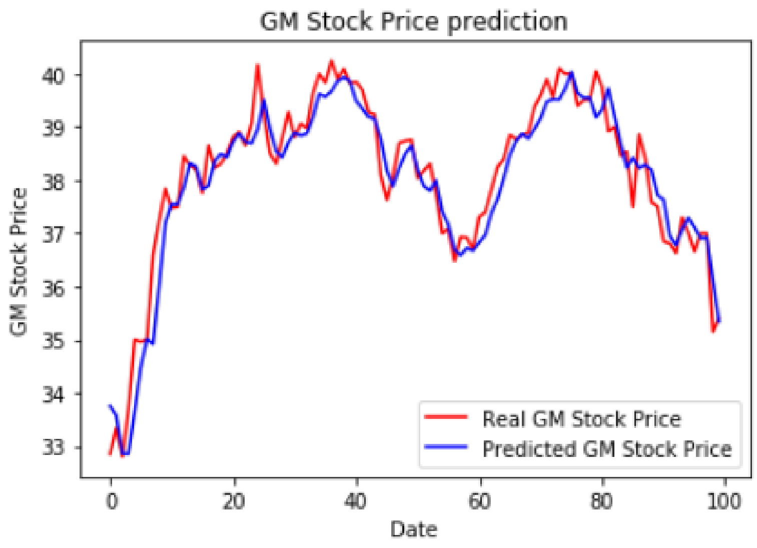


```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
0.2289859325589702
```

```
min_val=min(real_stock_price)
max_val=max(real_stock_price)
print ("relative error: ", rmse/(max_val-min_val))
```

```
relative error:  [0.07762237]
```

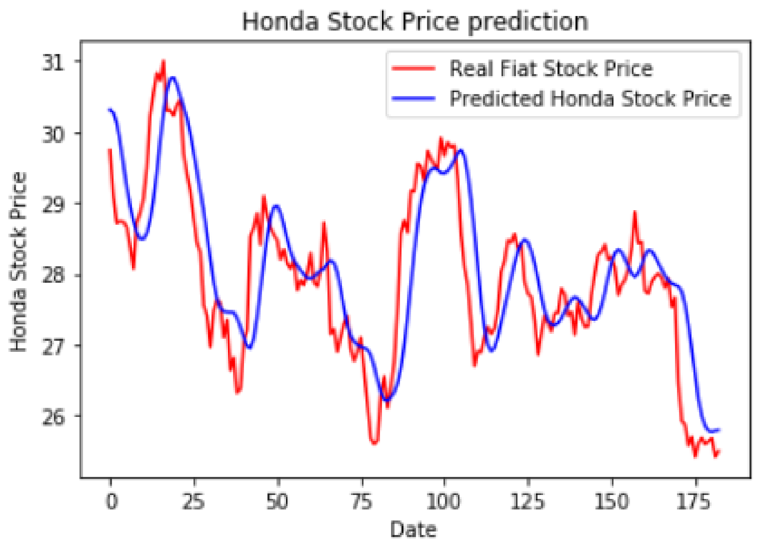


```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
0.47540999945021745
```

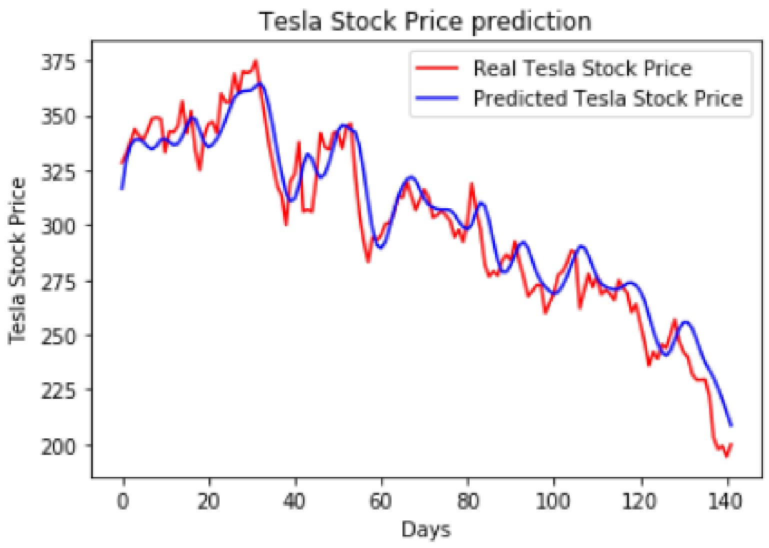
```
min_val=min(real_stock_price)
max_val=max(real_stock_price)
print ("relative error: ", rmse/(max_val-min_val))
```

```
relative error:  [0.06381342]
```



```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
0.7455506685943594
```

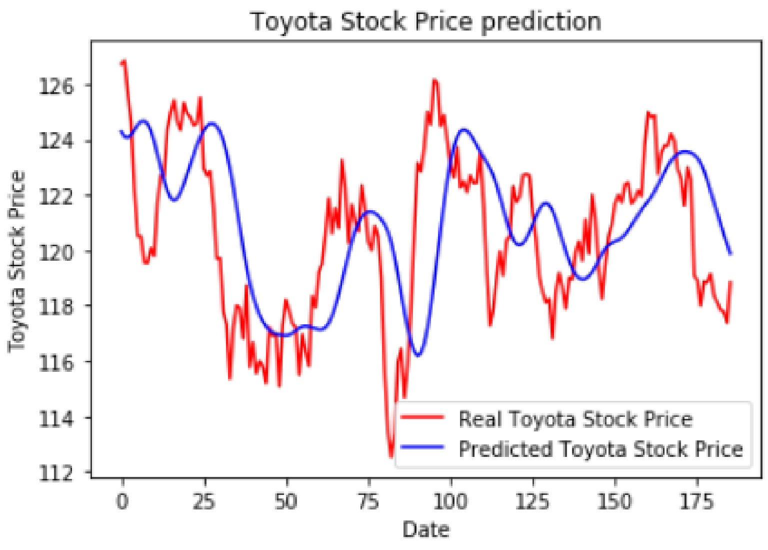


```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
12.343664092368371
```

```
min_val=min(real_stock_price)
max_val=max(real_stock_price)
print ("relative error: ", rmse/(max_val-min_val))
```

```
relative error:  [0.06832538]
```

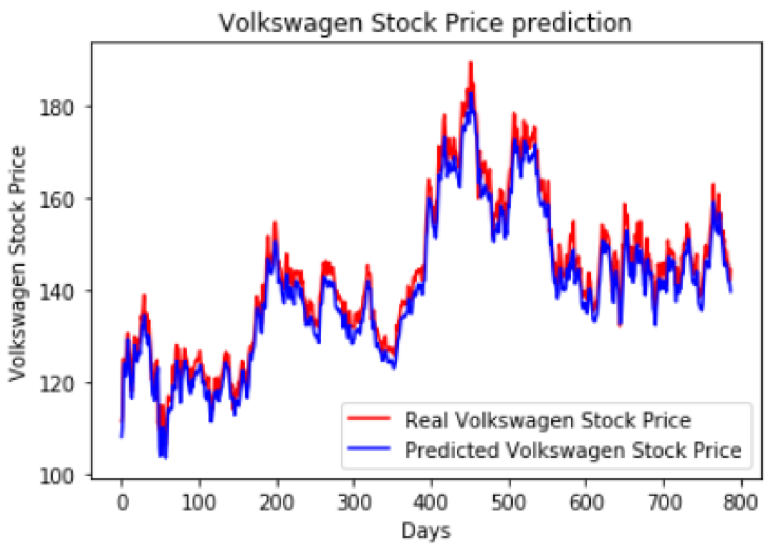


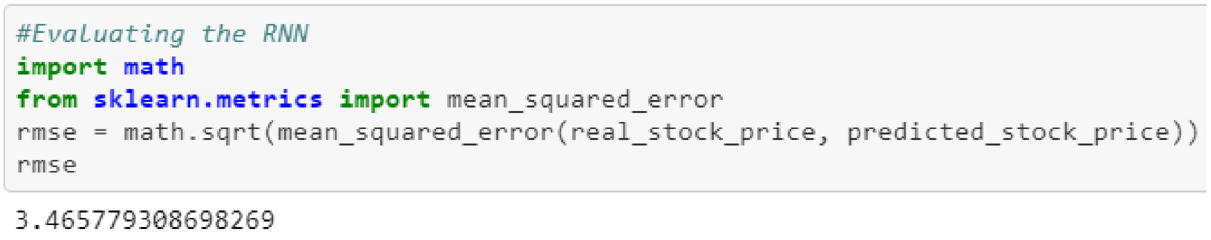
```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
2.9797294162647763
```

```
min_val=min(real_stock_price)
max_val=max(real_stock_price)
print ("relative error: ", rmse/(max_val-min_val))
```

```
relative error:  [0.20822711]
```





```
#Evaluating the RNN
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
rmse
```

```
3.465779308698269
```

Conclusion:

The RNN model has performed well and predicted accurately the trends in stock values. However, for Ford stock, the model just lags behind because it cannot react properly to fast, nonlinear changes. Those minute changes can be ignored, because, according to the Brownian Motion Mathematical Concept in financial engineering, the future variations of the stock price are independent from the past. Like the particles being bombarded in the Brownian motion, stock prices deviate from a steady state as a result of being jolted by mutual interactions, various external / internal factors, indicators, predictors, and /or trades. And therefore, the future variation that is seen here around the spike is a variation that is indeed totally independent from the previous stock prices. Thus some characteristics of the stock price motion can be seen as parallel to those of the Brownian motion process initiated by bombardment of external variables which I have not considered here. Even without those external indicators, the model responds perfectly well to smooth changes. The root mean square error and relative error values of all stocks studied here indicate a near perfect model to predict future trends in auto stocks prices.

Summary:

High dimensional stacked Long Short - Term Memory (LSTM) model of recurrent neural network (RNN) is used to predict the upward and downward trends of future auto stock prices. For this auto stock prices obtained from Yahoo finance is used. Based on the training on price values, the correlations identified or captured by the LSTM of the auto stock price, the model predicted accurately the trend of recent closing stock prices.

References:

Brownian motion and its applications in financial mathematics: https://scholarworks.rit.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=5989&context=theses

Financial Engineering and Risk Management Part I From the course by Columbia University: https://www.coursera.org/lecture/financial-engineering-1/introduction-to-brownian-motion-H6k29

Wikipedia: Louis Bachelier: https://en.wikipedia.org/wiki/Louis_Bachelier

Stock price modelling: Theory and Practice: https://beta.vu.nl/nl/Images/werkstuk-dmouj_tcm235-91341.pdf

Wikipedia, “Global financial crisis in November 2008”: https://en.wikipedia.org/wiki/Global_financial_crisis_in_November_2008

The Ultimate Guide to Recurrent Neural Networks (RNN): https://www.superdatascience.com/blogs/the-ultimate-guide-to-recurrent-neural-networks-rnn

Understand the Impact of Learning Rate on Model Performance With Deep Learning Neural Networks: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/