PRML Bonus Project Report

Bitcoin Price Prediction

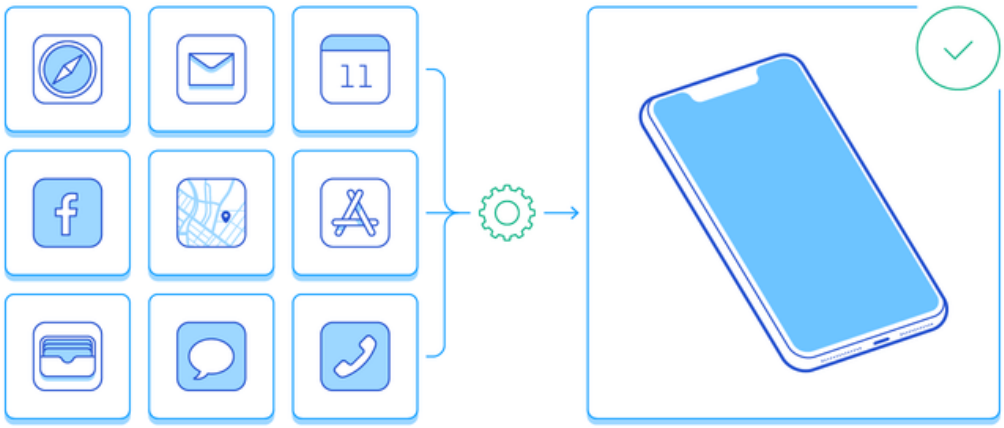
Kartik Choudhary [B20CS025]
Computer Science and Engineering

The Dataset: Bitcoin Prediction

Time Series Data Exploration

In this Dataset, we are supposed to predict the Closing price of Bitcoin and do the exploratory analysis on the same. The given dataset has 7 columns and 1556 rows, with each row having the Date-stamp corresponding to the prices. This specific report is intended to provide various insights on the Bitcoin Dataset and give knowledge about varied statistical methods of forecasting that could be used to gain insight into the bitcoin dataset provided in the "Bitcoin Price Prediction. This is based on Time-series data. Time series forecasting. Time series data are useful when you are forecasting something that is changing over time (e.g., stock prices, sales figures, profits, etc.). So here we are going to use various forecasting techniques and other models which we have used in this course. We have used LSTM, autoregressive model, RNNs, and a few ensemble learning algorithms for the prediction

Analysis: Ensemble Techniques



Methodolgy

Ensemble Learning Techniques such as Decision Tree Regressor, Random Forest Regressor, Linear Regression, and additionally KNN and SVR have been used to do the prediction. The dataset is split into 30%-70% test and training data and the results were obtained as follows.

01 Decision Tree Regressor

The Default Decision Tree Regressor is used for the prediction and the prediction comes out to be

```
dtr = DecisionTreeRegressor()
```

```
(dtr.score(X_test, y_test)))
```

```
Decision Tree Regressor: 0.9961676594055343
```

02 AdaBoost Regressor

```
abdtr = AdaBoostRegressor(DecisionTreeRegressor(max_depth = 4), n_estimators = 100)
abdtr_score = abdtr.score(X_test, y_test)
```

```
Ada Boost - Decision Tree Regressor: 0.9963730893903499
```

03 Random Forest Regressor

```
rfr = RandomForestRegressor(n_estimators = 300)
(rfr.score(X_test, y_test)))
```

```
Random Forest Regressor: 0.9979960259015486
```

04 Linear Regressor

```
linreg = LinearRegression()
linreg_score = linreg.score(X_test, y_test)
```

```
Linear Regression: 0.9982009246570771
```

05 SVR Using Different kernels

```
for k in ['linear','poly','rbf','sigmoid']:
    svr = SVR(kernel=k)
svr_score = svr.score(X_test, y_test)
```

```
SVR linear: 0.9975951826589186
SVR poly: 0.7097205657146175
SVR rbf: 0.250519745742395
SVR sigmoid: 0.6188342457723466
```

06 KNN Regressor

```
knn = KNeighborsRegressor()
knn_score = knn.score(X_test, y_test)
```

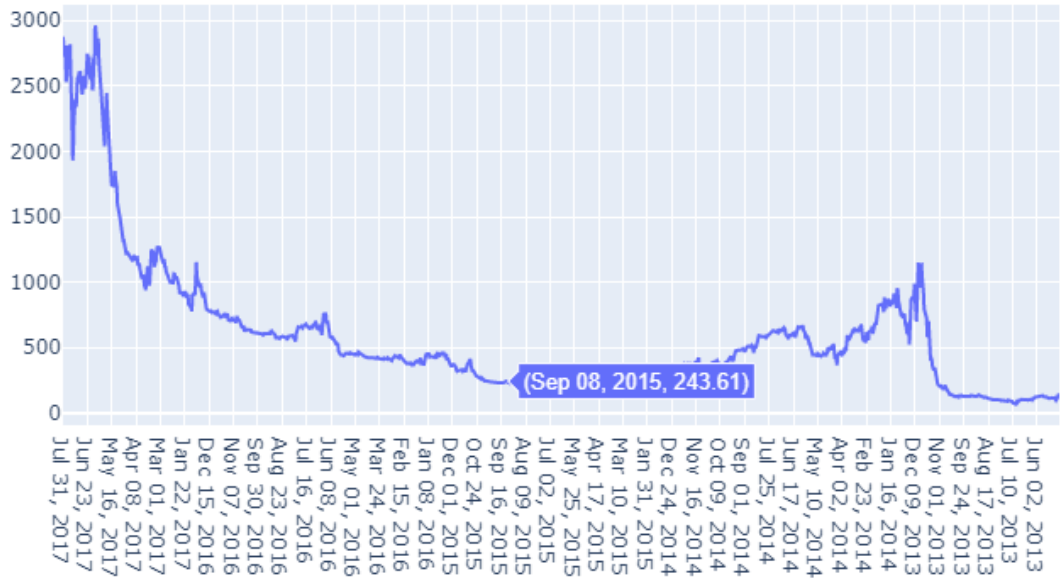
```
KNN Regressor: 0.9979633245677225
```

Using LSTM model

Long short-term memory is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points, but also entire sequences of data.

Since the closing price is explored and predicted.

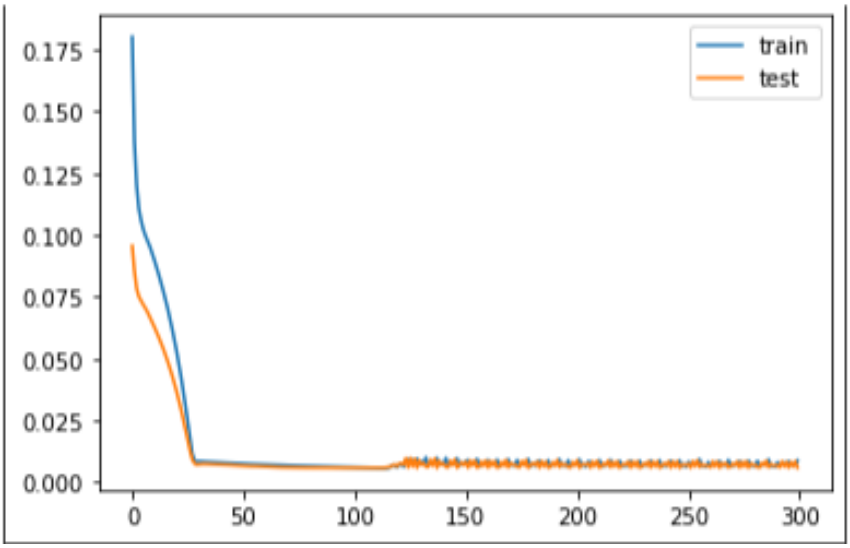
- Plotline graph based on Close Price



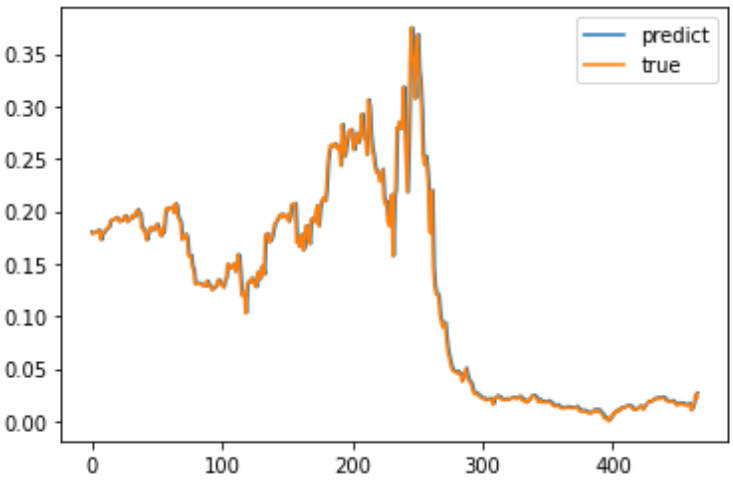
The above graph shows the trend of closing prices with the date

We have run the LSTM model for 300 epochs.

- Plot line graph to show amount loss according the epoch.

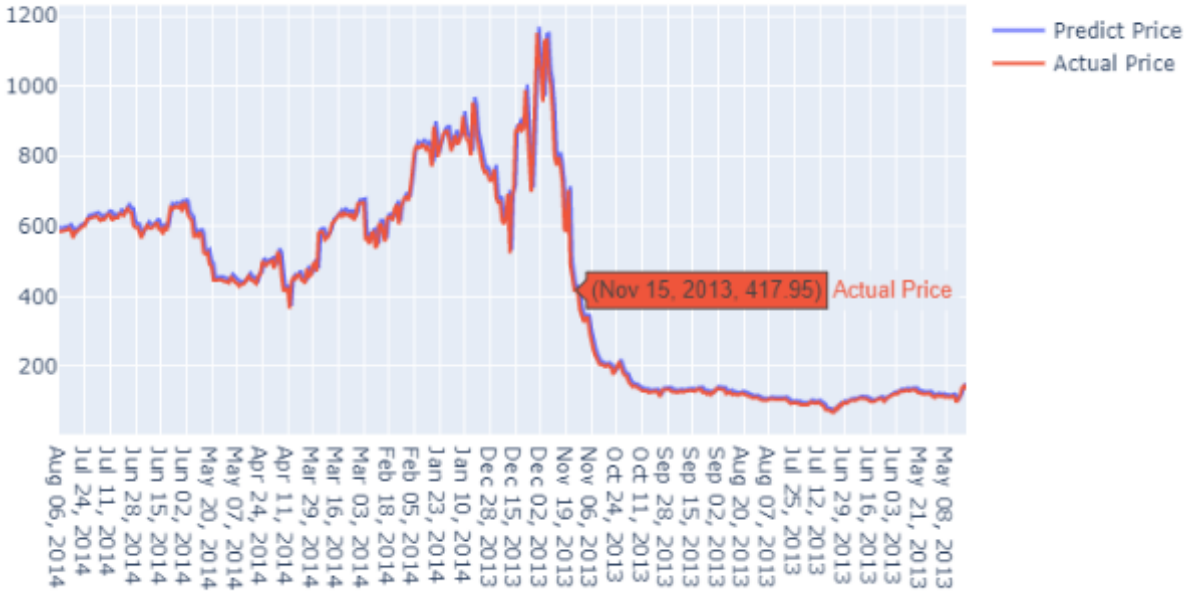


- Now prediction is made using textX and plotting line graph against testY

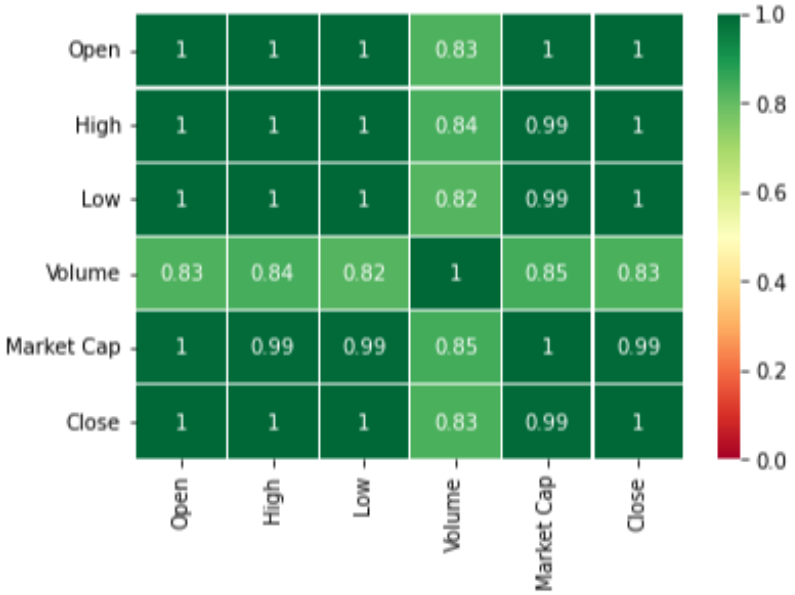


- The RMSE comes out to be:

- Plot predicted and actual line graph with X=dates, Y=Volume



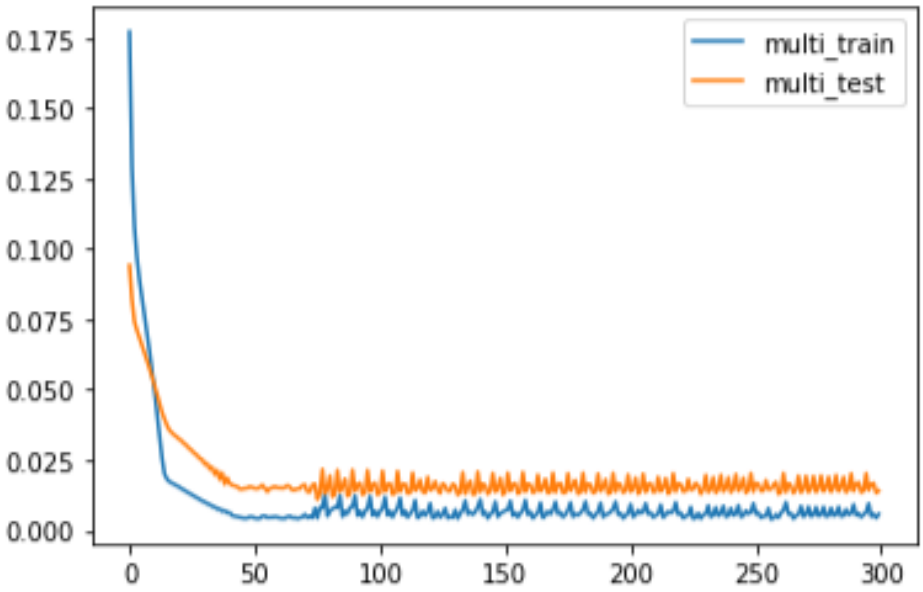
- Now we have used additional features for model training, the correlation matrix is found out to be as:



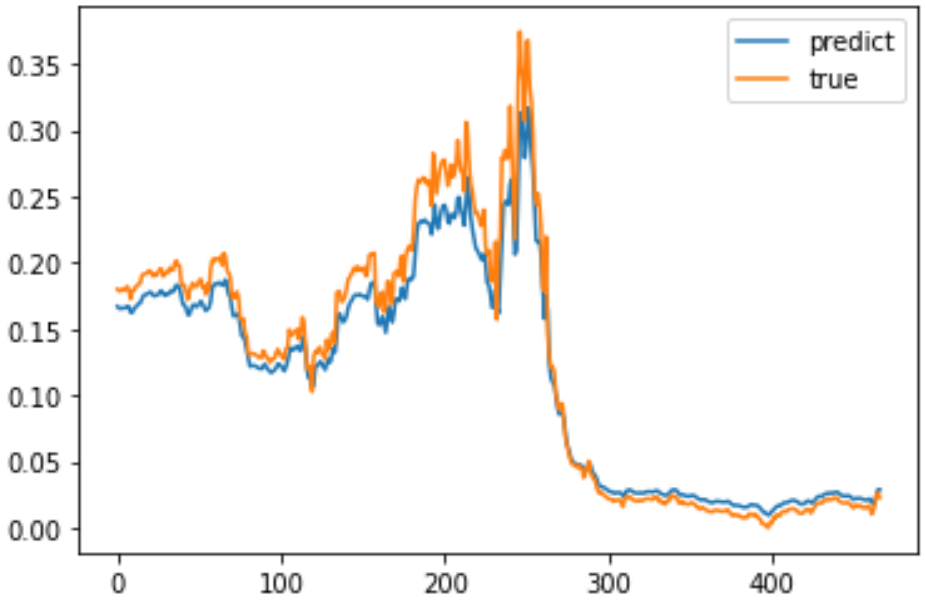
- Observation: Volume is correlated to Close Price. Open, High, Low, Market Cap are directly related to Close Price.

LSTM model is run again with additional features to do the prediction:

- Plot line graph to show amount loss according the epoch.



- Now prediction is made using textX and plotting line graph against testY



- From this the RMSE comes out to be:

```
Test RMSE: 52.949
```

Plotline graph with the actual price, predicted price with feature Close Price, predicted price with features Volume and Close Price

- LSTM with single feature of Weighted Price have RMSE of 34.409
- LSTM with features of Volume and Close Price have RMSE of 52.949

Using a Simple RNN Model

We have used a simple RNN model here to predict the closing price of Bitcoin
.The preprocessing is done and the data is standardized before feeding into the model . The hyper-parameters for the model are chosen after drawing insights from the lag plot and the autoregressive model.

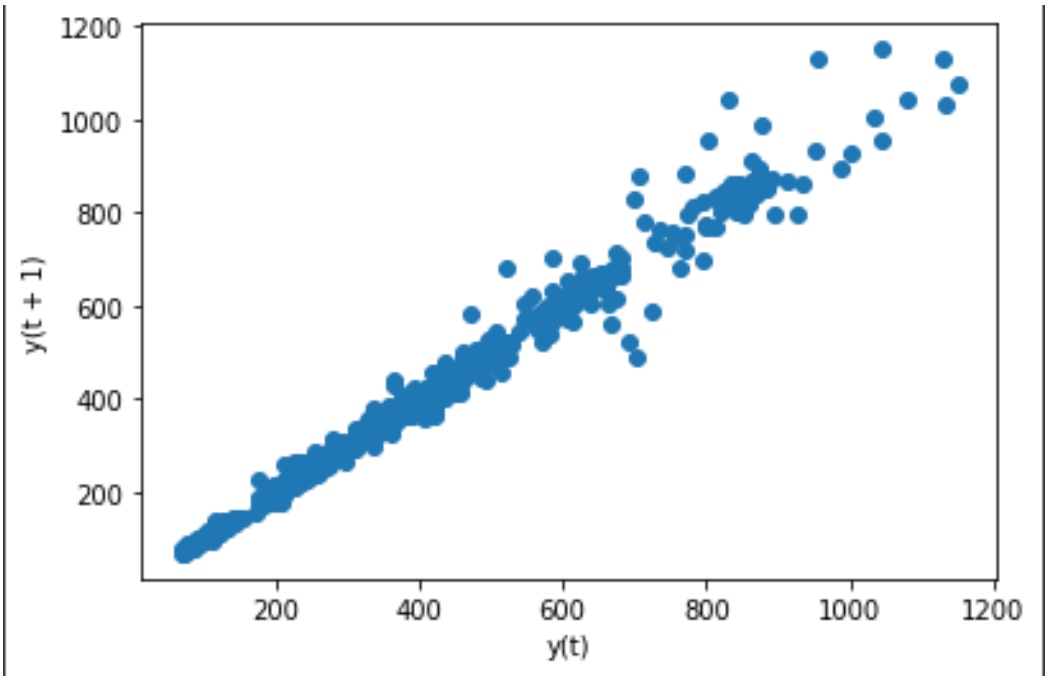
Lag Plot (check whether the time series is random or not)

In the graph below, the first axis represents the t(lag), the second axis represents t+1

Ex: if data is, [1,4,5,3,2], then y(t):= [1,4,5,3,2], y(t+1): = [4,5,3,2]

As we see the graph below, this suggests a non-random pattern (the graph is positively linear).

Non-randomness in the data reveals that we could use an autoregressive model.

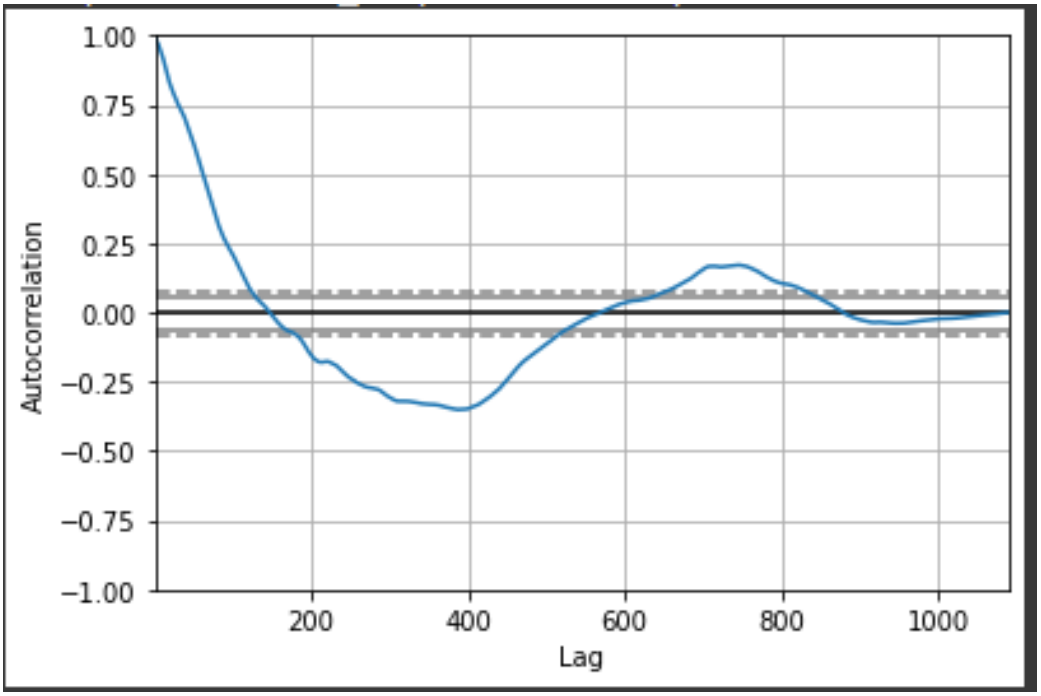


Autocorrelation

A lag plot above shows the structure of the data. In order to quantify the correlation between the point at t and point at t+1 with respect to expectation, autocorrelation is used.

A black line in the graph below shows the expectation for random data(thus 0 correlation) and two dash lines above and below it represent the confidence interval with each 95% and 99%.

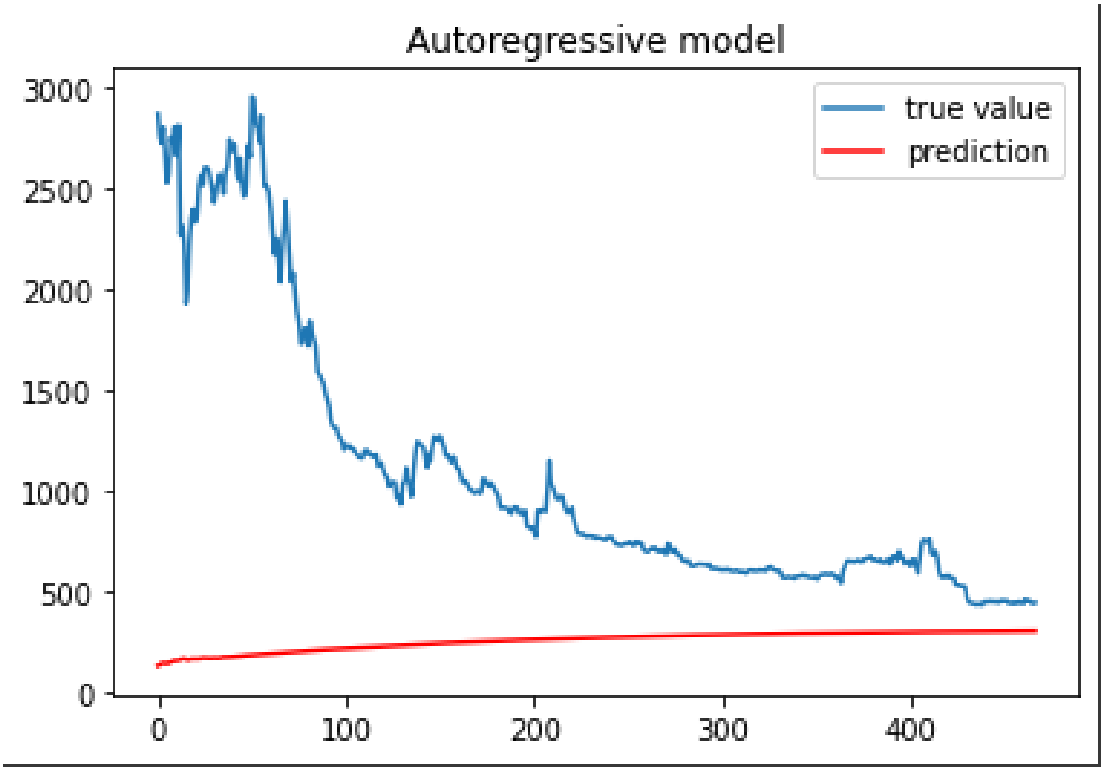
The graph shows a strong correlation for lags of < 100 days. (lag 0 is always 1 correlation)



Autoregression

lag plot and autocorrelation revealed that we could use autoregression for fitting data we will predict test data (closing price for 7 days) using training data

22 lags are used to train the model (22 previous points are used to predict the next point)
MSE is high.



Hyperparameters:

lookback: how many points (days) should be used as inputs to predict the future value
delay: how many points should be predicted
Ex: In this example, we use previous prices for 24 days as input to predict the prices for future 7 days

lookback: 22 is the value chosen by the autoregressive model
delay: test data has 7 points(daily values) so prediction and true values will be possible

- RNN runs for 300 epoch

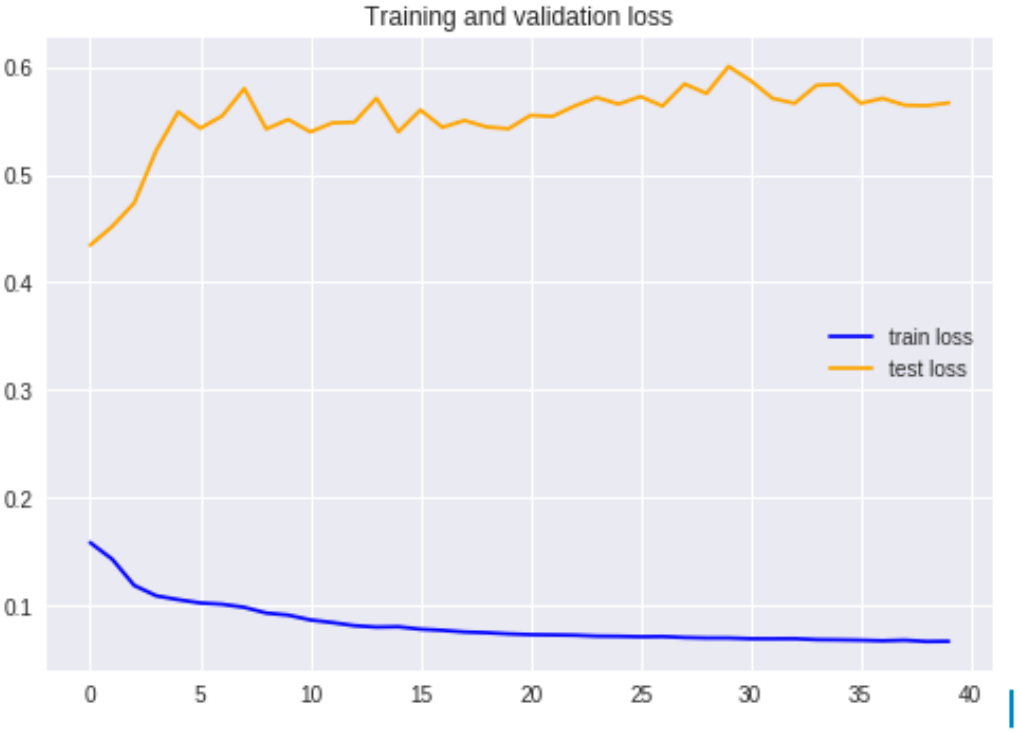
```
model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.2,
                     recurrent_dropout=0.2,
                     input_shape=(None, train_n.shape[-1])))
model.add(layers.Dense(1))
model.compile(optimizer=RMSprop(), loss='mae')
model.summary()
```

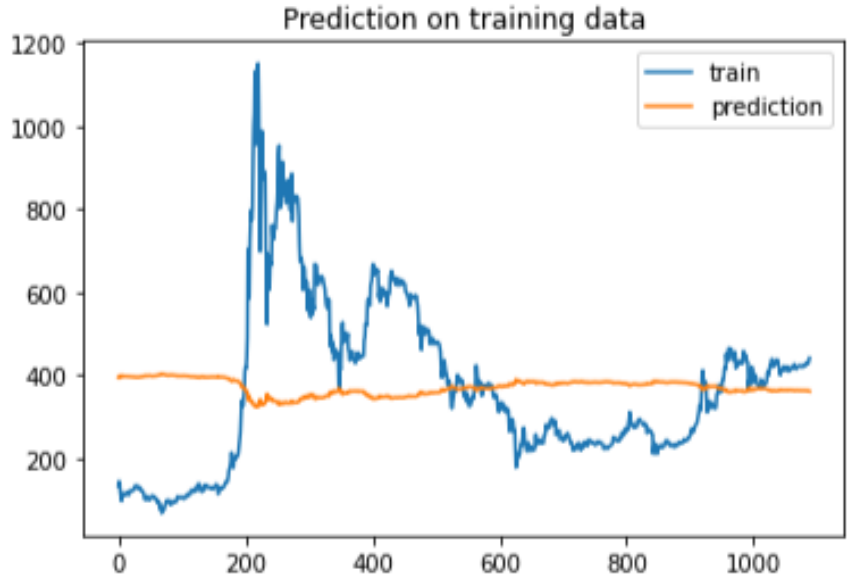
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 gru_1 (GRU)                  (None, 32)                3360

 dense_3 (Dense)              (None, 1)                 33

=================================================================
Total params: 3,393
Trainable params: 3,393
Non-trainable params: 0
_________________________________________________________________
```

- RESULTS





Conclusion:

The simple RNN performs poorly on the training data.
The best model for the prediction comes out to be the LSTM model