**Integer Programming**

Integer Programming (IP) problems are optimization problems where all of the variables are constrained to be integers. IP mathematical programming technique to obtain the best result or outcome, like maximum profit or least cost, in a mathematical model whose requirements are represented by linear relationships. Generally, an organization or a company has mainly two objectives, the first one is minimization and the other is maximization. Minimization means to minimize the total cost of production while maximization means to maximize their profit.

The **purpose** of using Integer Programming to find optimal weights/lot size of each strategies based on past performance of each 15 GL strategies data to maximize the profit in the trading. Strategy weights/lot size varies from 0 to 5 and sum of all strategies weights/lots size equal to the number of strategy. Based on performance of all 15 strategies in the past, Integer Programming assign weights/lots size to each strategies. Such that for particular strategy if we get weight/lots size 0 then that means no signal(zero Lot size) produce by strategy whereas 1-5 weights/lots size means signal produced.

**Portfolio Rebalancing** is the process of buying and selling portions of your portfolio in order to set the weight of each asset class back to its original state.

**PuLP**

PuLP is an open-source [linear programming](https://en.wikipedia.org/wiki/Linear_programming) (LP) package, it is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems.

1. **Define Problem**

Define the problem by giving a suitable name to your problem. Also, specify your aim for the objective function of whether to LpMaximize or LpMinimize.

1. **Define Integer variables for each strategy (Decision Variables)**

Define LpVariable to hold the variables of the objective functions. The next argument specifies the lower bound and the upper bound of the defined variable. After that select type of category as per your requirement which can be LpContinuous or LpBinary or LpInteger.

1. **Create the objective function**

We start building the IP problem by adding the main objective function. We are defining objective function using decision variables and return values (which is difference between first and last value of return value of each strategies) of each strategies. By taking sum of product decision variables and return values, we created our objective function.

s1\*pnl1 + s2\*pnl2+ s3\* pnl3 + s4\* pnl4+..............................+s15\* pnl15

1. **Create constraints**

The constraints are added in a similar way to the objective.

1. **Run problem**

Run the problem using default solver.

1. **Getting Solution**

Now get the solution using variable.value() function and the objective value using problem.objective.value()

**Code Go Through**

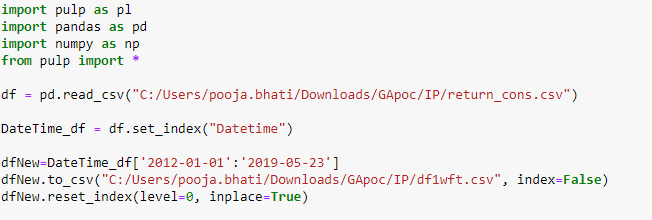
There are 3 different files, whereas we sort dataset high to low based on SG strategy return value for creating constraints:

Top1: It will take top first row.

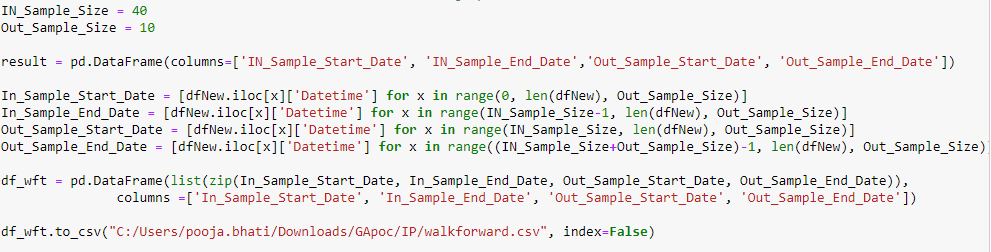
Top2: It will take top two rows.

Top3: It will take top three rows.

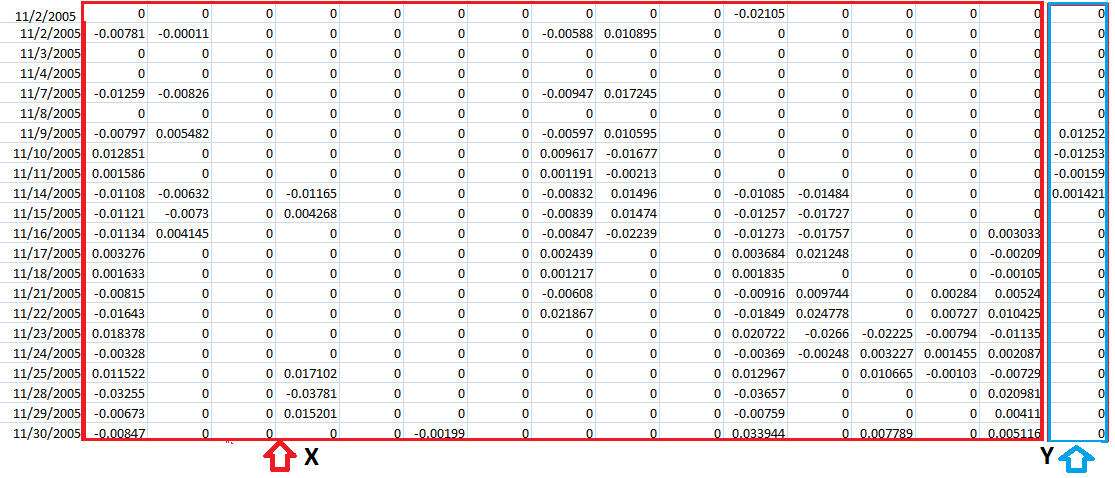
Define library and the necessary packages required, read the return consolidated file from 2012 to 2019.



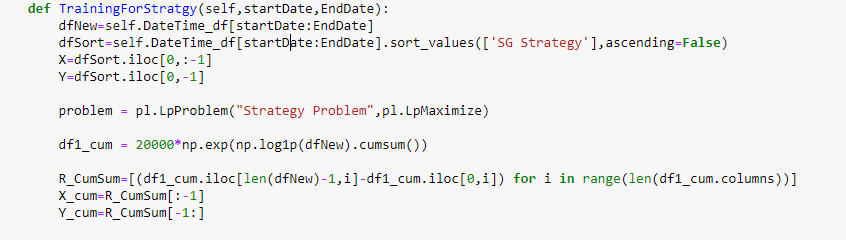
Using Walk forward testing create different iteration to run the problem statement multiple time of the sequenced data. In\_Sample\_Size is for training whereas Out\_Sample\_Size is of testing our problem statement.



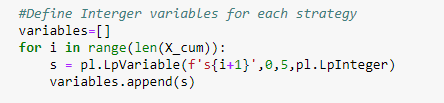
For particular iteration created by Walk forward testing, consider all 15 strategies return value as X and SG strategy as Y. We are problem as "problem" and the objective function as LpMaximize.



Now take cumulative sum of X and multiply be 20000 and after that take difference between first and last value for each strategies. Store that in X\_cum for all 15 strategies which is also called PnL which we further use for creating objective function.



We defined variables as s1, s2, s3, s4,.....,s15, bound between 0 to 5 and Category as LpInteger. LpInteger forces all of the variables to assume only integer values.



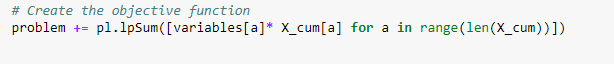
Create objective function using decision variables and X\_cum defined above. Here, in our case objective function is sum of product of PnL and weights/lot size for all strategies.

s1\*PnL1 + s2\* PnL2+ s3\* PnL3+ s4\* PnL4+..............................+s15\*PnL15

Where:

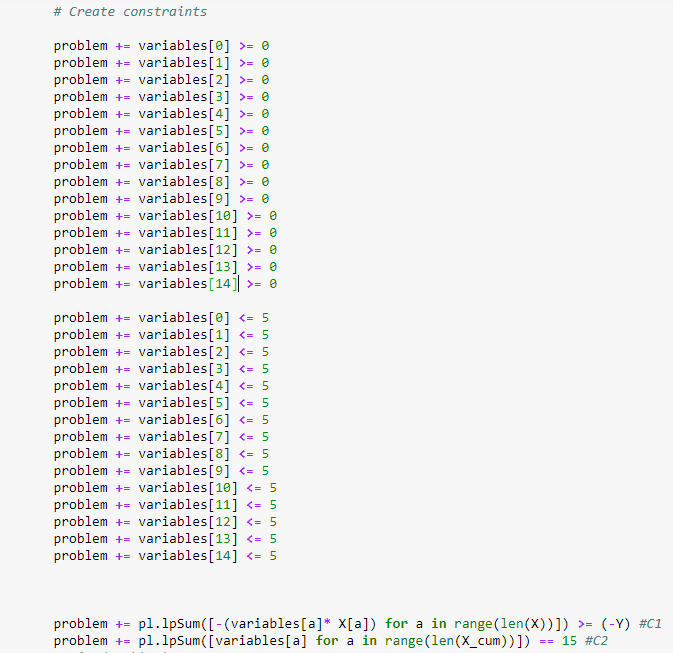
Cumulative equity over period = initial investment\*(1+rreturn) , this is calculated over returns for a given time frame

PnL = Cumulative equity at the end of the period – Cumulative equity at the starting of the period



We defined following constraints:

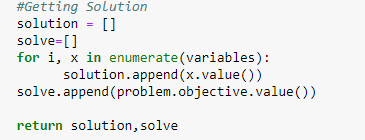
* Hard constraints for all the variables lies between 0 to 5.
* Main objective function value must be less than equal to SG Strategy return value(Sort values High to Low based on SG Strategy and use highest value to create constraint).
* Sum of all weights/objective value must be equal to 15.



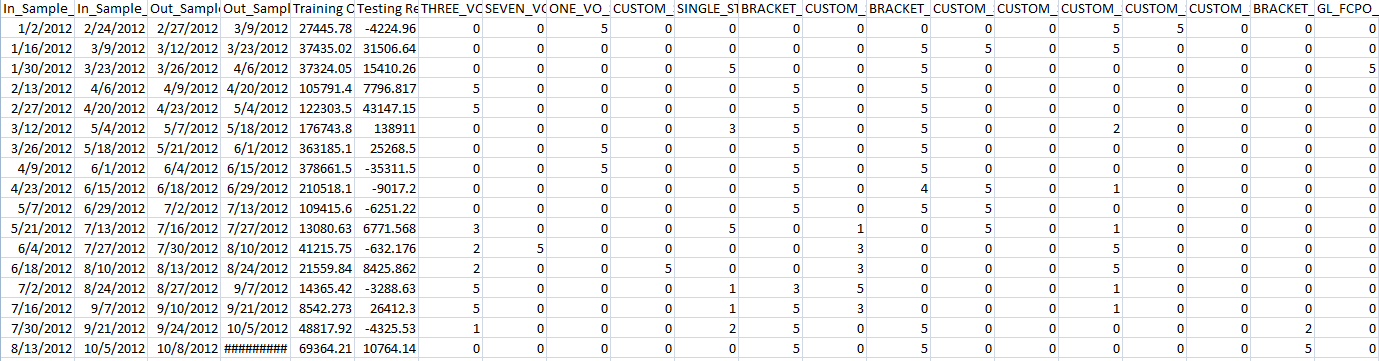
Here we solve the problem,



Getting the solution/weight for each strategies and the objective value.



Below is the output file generated by the integer programming. First two columns show the training sample start, end date. 3rd and 4th columns show the start and end date for test dataset. 5th and 6th columns describe the optimal PnL defined using objective function over training dataset and testing dataset respectively. Rest of the columns are weights/lot size for each strategies whose value varies between 0 to 5 and sum of all strategies lot size equal to the number of strategies.



**Trend Classification**

A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease". A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. Here we are try to classify trend as Up, Down or No Tread which is a multi class classification problem.

1. Multi-Class Logistic Regression

Multi-Class Logistic Regression is a [classification algorithm](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2?utm_source=blog&utm_medium=SimpleGuidetoLogisticRegressionarticle). It is used to predict value more than two outcomes given a set of independent variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

1. k-NN

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm.

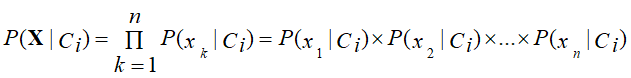
1. Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an [ensemble](https://en.wikipedia.org/wiki/Ensemble_learning). Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don’t constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

1. Naive Bayes

Naive Bayes is a probabilistic classifier inspired by the Bayes theorem under a simple assumption which is the attributes are conditionally independent.



The classification is conducted by deriving the maximum posterior which is the maximal P(Ci|**X**) with the above assumption applying to Bayes theorem. This assumption greatly reduces the computational cost by only counting the class distribution. Even though the assumption is not valid in most cases since the attributes are dependent, surprisingly Naive Bayes has able to perform impressively.

1. XG-Boost

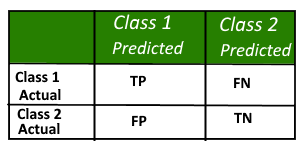
[XGBoost](https://xgboost.ai/)is a decision-tree-based ensemble Machine Learning algorithm that uses a [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting) framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

1. LSTM (Long Short Term Memory)

Long-Short-Term Memory(LSTM) models are a type of Recurrent Neural Networks(RNNs) which has the ability to learn and remember over long sequences of input data through the use of **“gates”**which regulate the information flow of the network.

**Confusion Matrix**

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. A confusion matrix is a summary of prediction results on a classification problem.

****

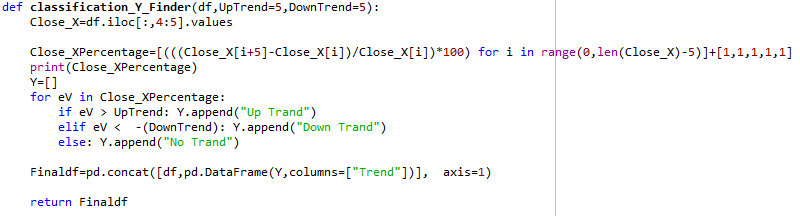
• True Positive (TP) : Observation is positive, and is predicted to be positive.  
• False Negative (FN) : Observation is positive, but is predicted negative.  
• True Negative (TN) : Observation is negative, and is predicted to be negative.  
• False Positive (FP) : Observation is negative, but is predicted positive.

**Code Overview**

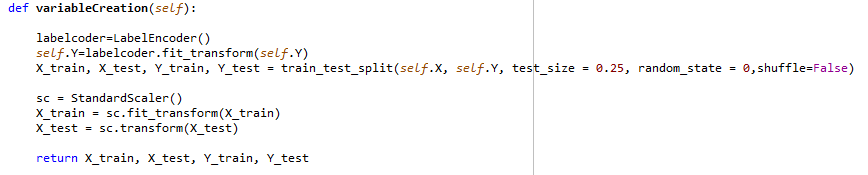
This function used to find Y based on close price of the OHLCV dataset whereas need to pass uptrend and down trend cutoff value which is default set to 5 for both but can be pass manually also.

Suppose we didn't pass any value while calling function then close price:

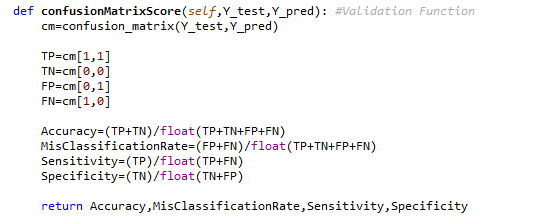
* Less than - 5 consider as Down Trend
* Greater than 5 consider as Up Trend
* Between -5 to 5 consider as No Trend



Below function will preprocess your data which further used by algorithms. Split data into Training and testing dataset then scale X values for standardization.

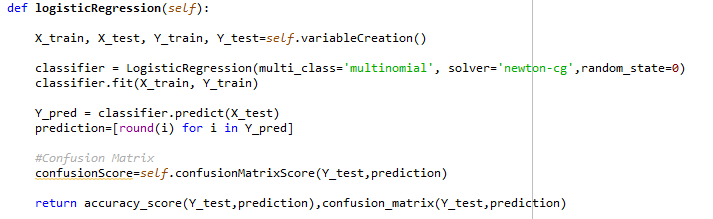
****

In this function we pass actual and classified values from which it create confusion matrix to evaluate model.

****

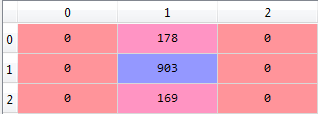
1. Logistic Regression

Split dataset into train and test then using SKlearn library call Logistic model and fit into the train dataset. Predict values using testing dataset then create confusion matrix for evaluating the model.



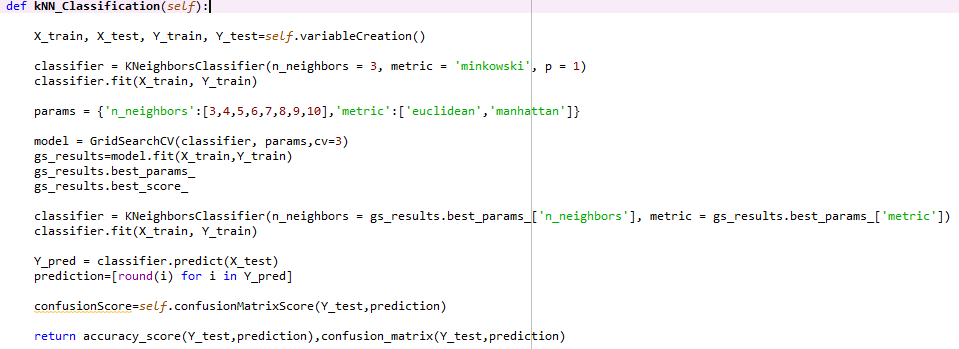
Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



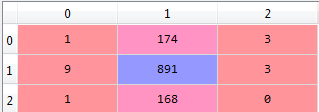
1. k-NN

Same way split dataset, train and fit. Now we used Grid Search optimization technique to get optimize hyper-parameters. Again fit model using optimize hyper-parameters, predict values and generate confusion matrix.



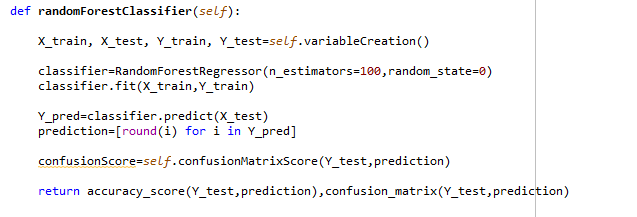
Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



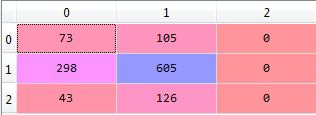
1. Random Forest

Same way split dataset, train, fit, predict values and generate confusion matrix using SKlearn Library.



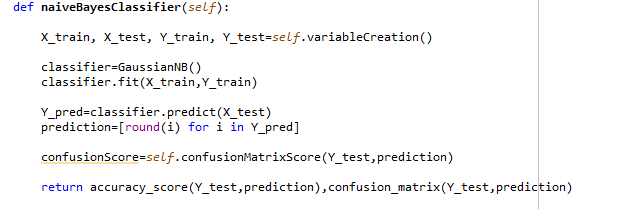
Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



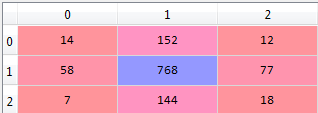
1. Naive Bayes

Same way split dataset, train, fit, predict values and generate confusion matrix using SKlearn Library.



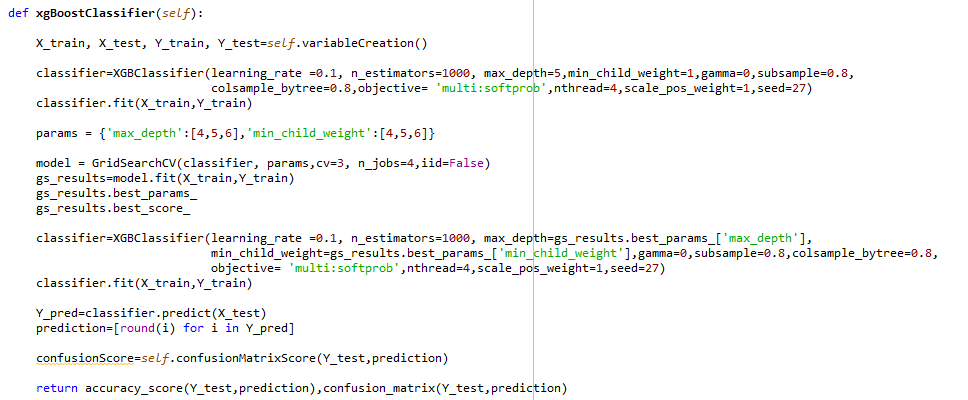
Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



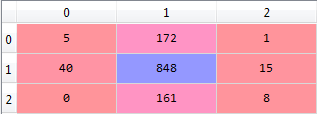
1. XG-Boost

Same way split dataset, train and fit. Now we used Grid Search optimization technique to get optimize hyper-parameters. Again fit model using optimize hyper-parameters, predict values and generate confusion matrix.



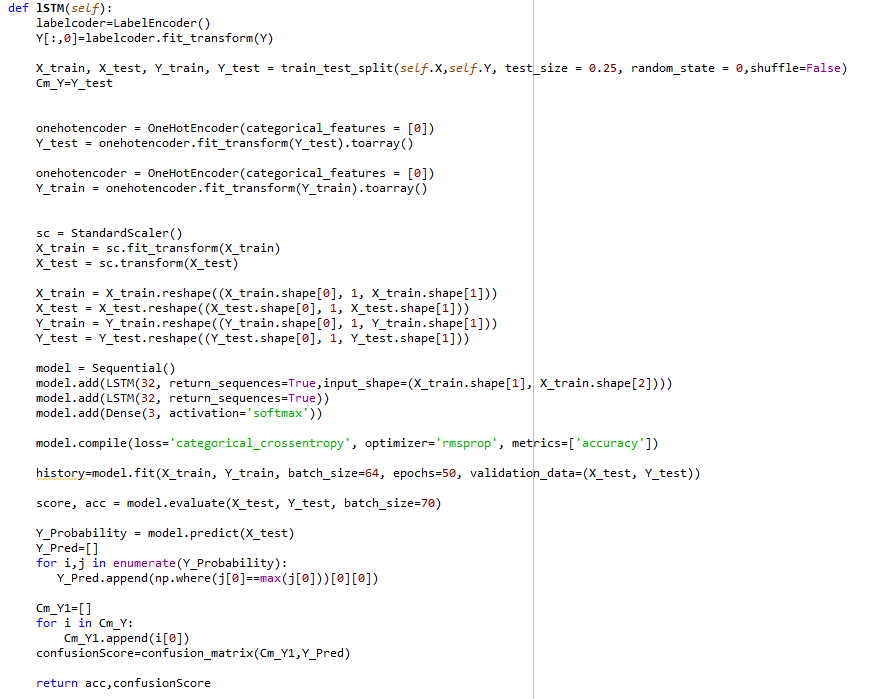
Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



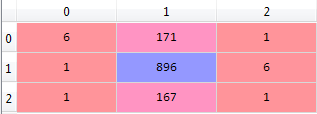
1. LSTM

Firstlysplit dataset then using OneHotEncoder convert categorical to discrete data. Now reshape you dataset to 3-dimensional as accepted by LSTM. Define layers and fit your model. Finally create confusion matrix to get the accuracy.



Confusion matrix shown below:

Here, 0 -> Down Trend, 1-> No Trend, 2-> Up Trend



**Trend Prediction:**

1. **Linear Regression:**

**Background:**

The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable.

The equation for linear regression can be written as:

Here, x1, x2,….xn represent the independent variables while the coefficients θ1, θ2, …. θn represent the weights.

**Trained Models:**

**1 day Model:** Regression model for 1 day prediction

**2 day Model:** Regression model for 2 day prediction

**3 day Model:** Regression model for 3 day prediction

**4 day Model:** Regression model for 4 day prediction

**5 day Model:** Regression model for 5 day prediction

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Create features: we use the date column to extract features like – day, month, year, mon/fri etc. and then fit a linear regression model with the help of the library fastai.

This creates features such as:

‘Year’, ‘Month’, ‘Week’, ‘Day’, ‘Dayofweek’, ‘Dayofyear’, ‘Is\_month\_end’, ‘Is\_month\_start’, ‘Is\_quarter\_end’, ‘Is\_quarter\_start’, ‘Is\_year\_end’, and ‘Is\_year\_start’.

3. Traning and testing data split: We splited the data in training and testing in 80:20 ratio.

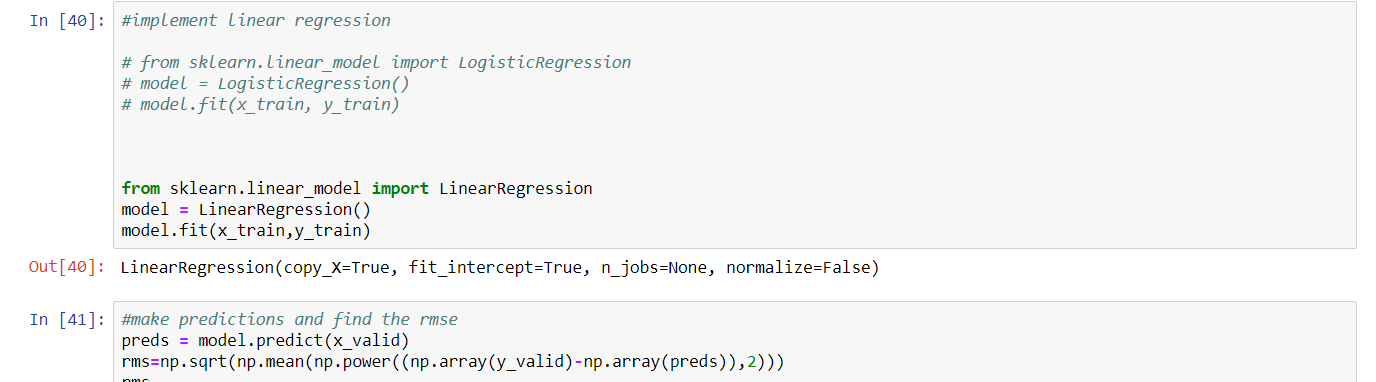
4. Applied Regression Model: After importing the Linear Regression from linear model, we have created the object of Linear regressor and then fit the traing data into regressor model through fit method.

5. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

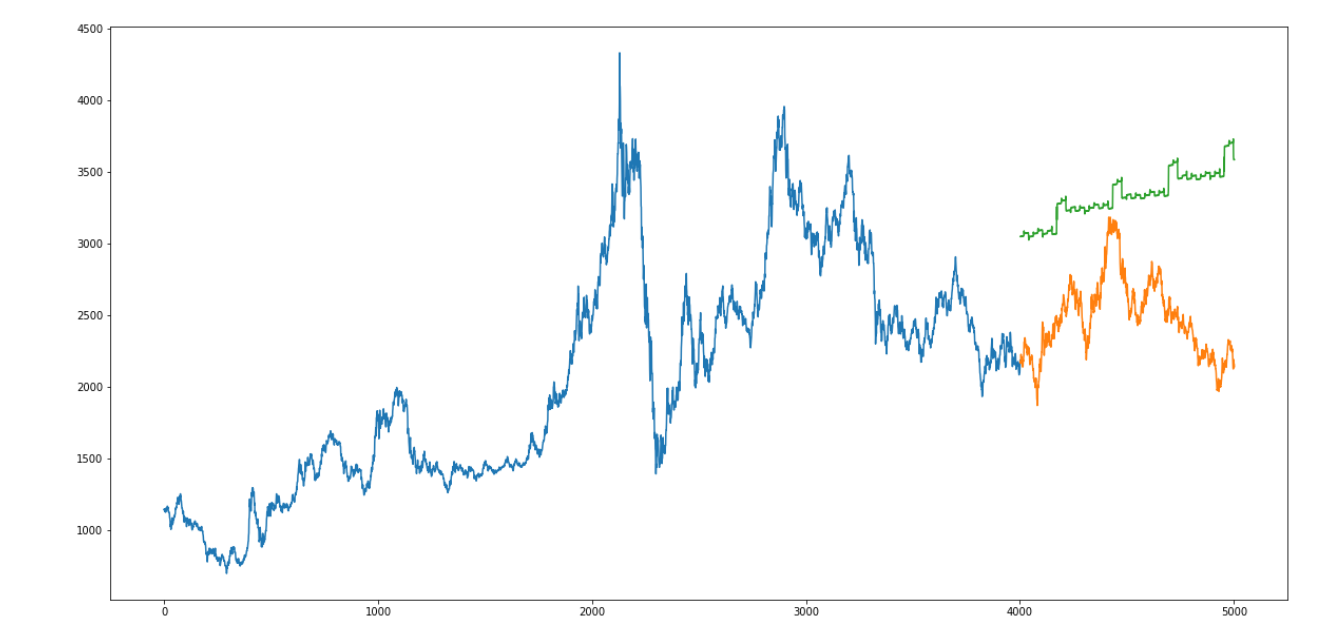
**RMSE Table:**

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| **1 day** | **906.24** |
| **2 day** | **908.51** |
| **3 day** | **667.75** |
| **4 day** | **1045.24** |
| **5 day** | **910.27** |

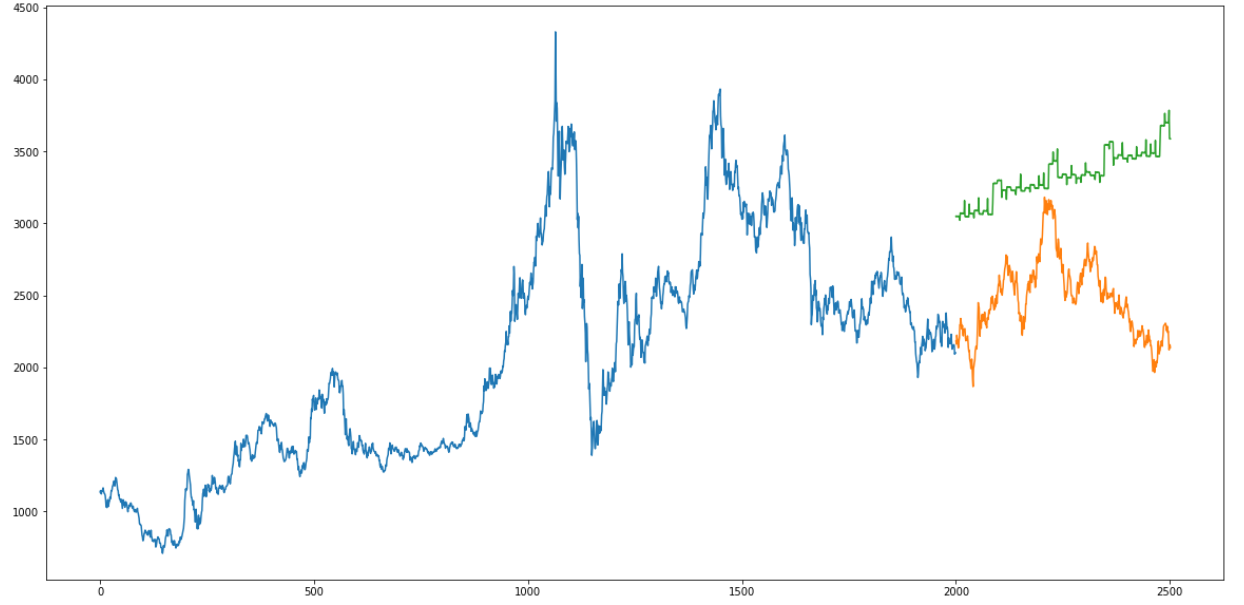
**Code Snippet:**



**Predicted vs actual: 1-day**



**Predicted vs actual: 2 Day :**



1. **k-Nearest Neighbors:**

**background:**

Based on the independent variables, kNN finds the similarity between new data points and old data points.

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

**Trained Models:**

**1 day Model:** KNN model for 1 day prediction

**2 day Model:** KNN model for 2 day prediction

**3 day Model:** KNN model for 3 day prediction

**4 day Model:** KNN model for 4 day prediction

**5 day Model:** KNN model for 5 day prediction

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Applied K Nearest Neighbors Model: After importing Nearest Neighbors from model, we have created the object of Nearest Neighbors and then fit the training data into Nearest Neighbors model through fit method.

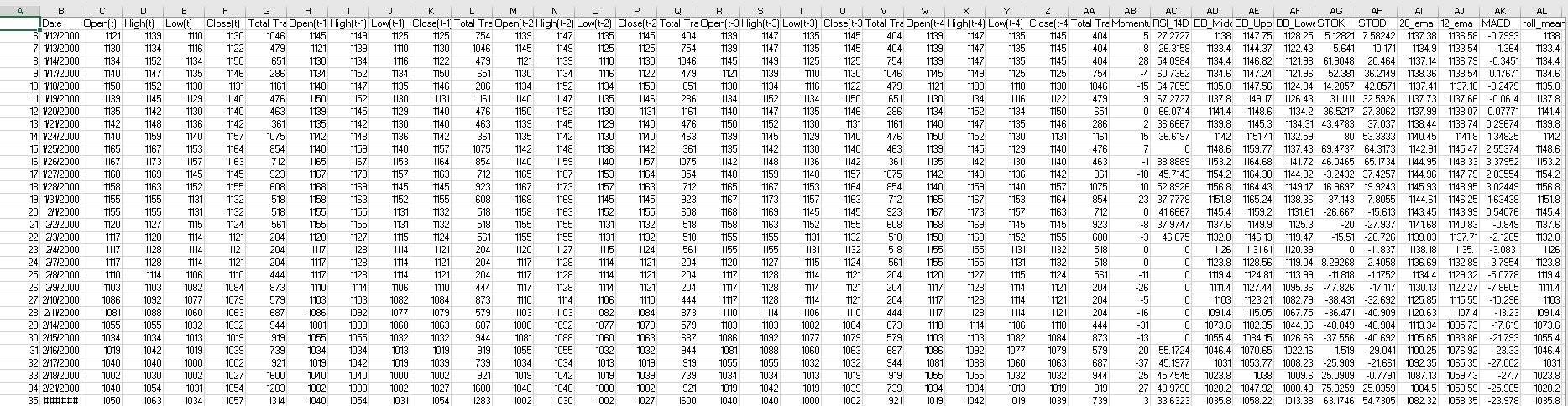
5. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| **1 day** | **846.16** |
| **2 day** | **803.02** |
| **3 day** | **1314.36** |
| **4 day** | **875.97** |
| **5 day** | **837.54** |

**List of Dependent and Independent variables:**

**Dependent: Close Price**

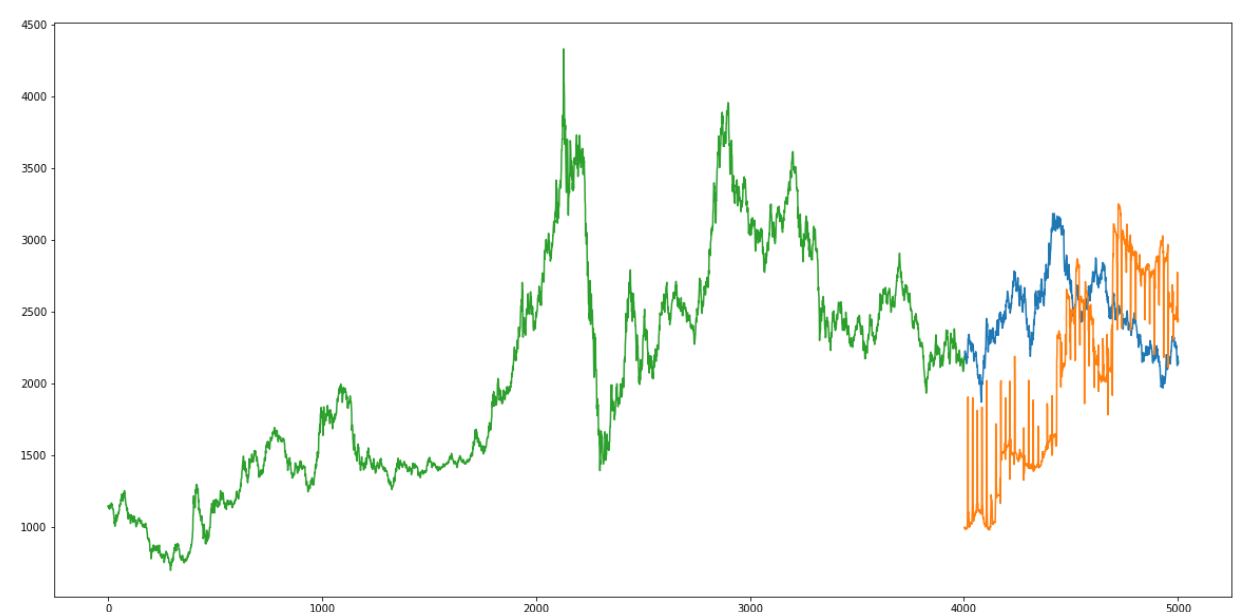
**Independent: Except Close Price**



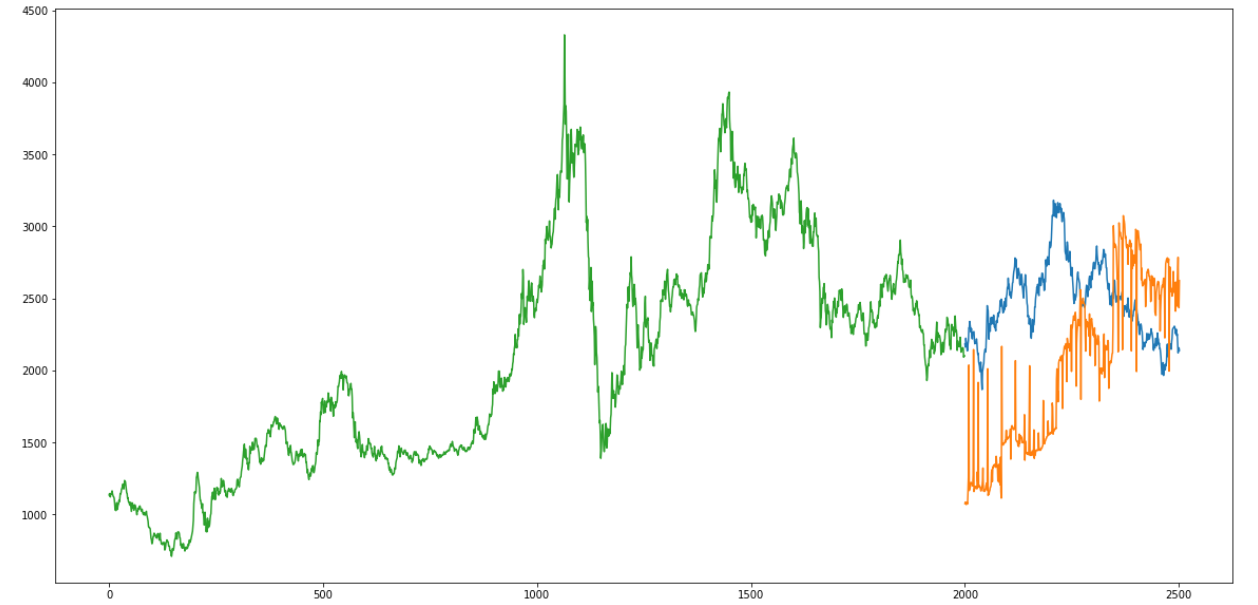
**Code Snippet:**



**Predicted vs actual 1 Day:**



**Predicted vs actual 2 Day:**



1. **Auto ARIMA:**

**Background:**

ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:

p (past values used for forecasting the next value)

q (past forecast errors used to predict the future values)

d (order of differencing)

Parameter tuning for ARIMA consumes a lot of time**. So we will use auto ARIMA which automatically selects the best combination of (p,q,d) that provides the least error**.

**Value Used for p, d,q is 1**

**Trained Models:**

**1 day Model:** Auto ARIMA model for 1 day prediction

**2 day Model:** Auto ARIMA model for 2 day prediction

**3 day Model:** Auto ARIMA model for 3 day prediction

**4 day Model:** Auto ARIMA model for 4 day prediction

**5 day Model:** Auto ARIMA model for 5 day prediction

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

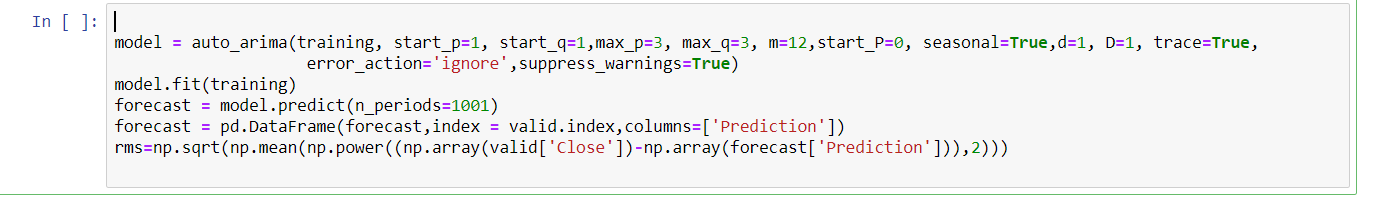
3. Applied Auto ARIMA Model: After importing the Auto ARIMA from Pyramid ARIMA, we have created the object of Auto ARIMA and then fit the training data into Auto model through fit method.

4. Predictions and RMSE calculation: After fitting the model, we have predicted the values for test data and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

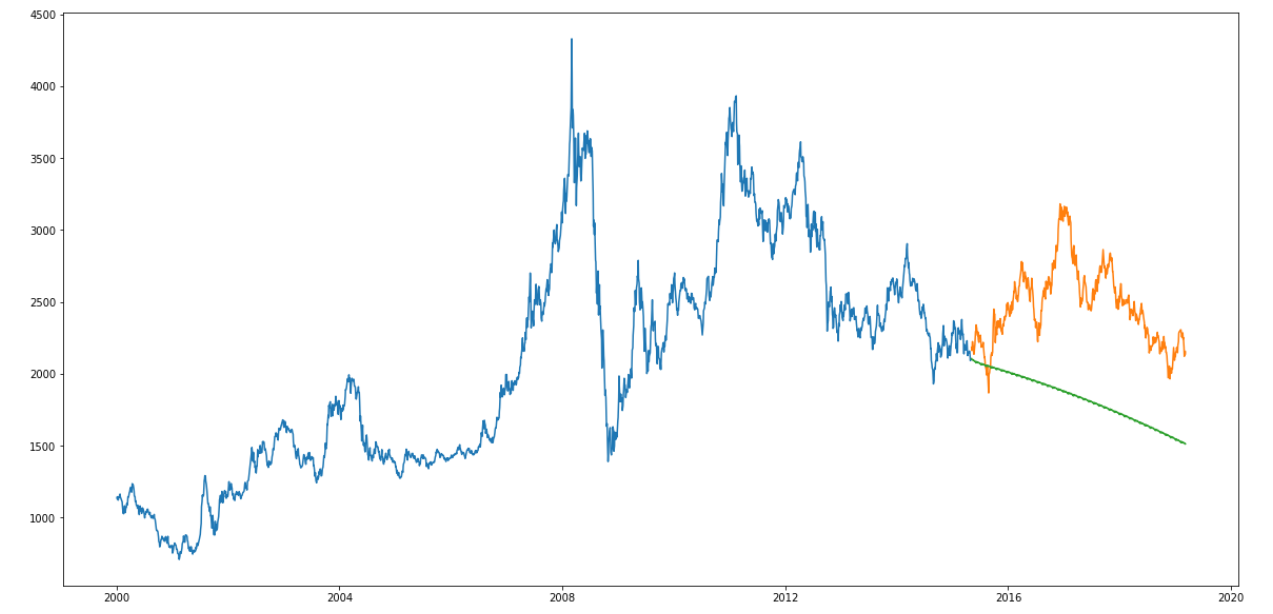
RMSE Table:

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| **1 day** | **710.49** |
| **2 day** | **790.51** |
| **3 day** | **610.49** |
| **4 day** | **574.39** |
| **5 day** | **698.62** |

**Code Snippet:**



**Predicted vs actual:**



1. **Prophet:**

**Background:**

There are a number of time series techniques that can be implemented on the stock prediction dataset, but most of these techniques require a lot of data preprocessing before fitting the model. Prophet, designed and pioneered by Facebook, is a time series forecasting library that requires no data preprocessing and is extremely simple to implement. The input for Prophet is a data frame with two columns: date and target (ds and y).

Prophet tries to capture the seasonality in the past data and works well when the dataset is large.

**Trained Models:**

**1 day Model:** Prophet Model for 1 day prediction

**2 day Model:** Prophet Model for 2 day prediction

**3 day Model:** Prophet Model for 3 day prediction

**4 day Model:** Prophet Model for 4 day prediction

**5 day Model:** Prophet Model for 5 day prediction

**Implementation:**

1. Data Preprocessing: We have created a separate dataset with Date and Close price so that any new feature created does not affect the original data and then rename Close price as y and Date as ds.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

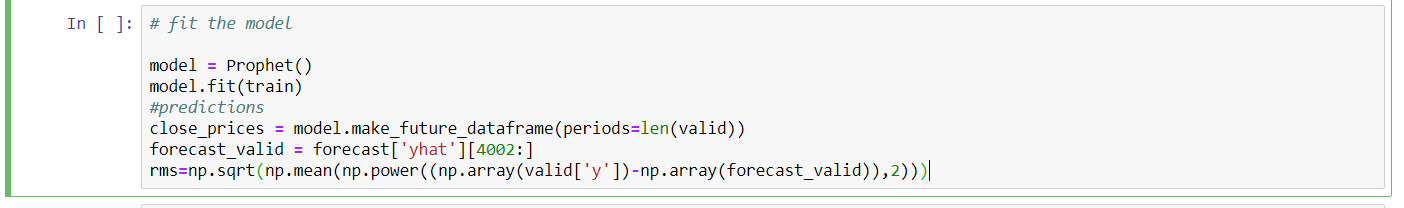
3. Applied Prophet Model: After importing the Prophet machine learning Model from fbprophet library, we have created the object of Prophet and then fit the training data into Prophet model through fit method.

4. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

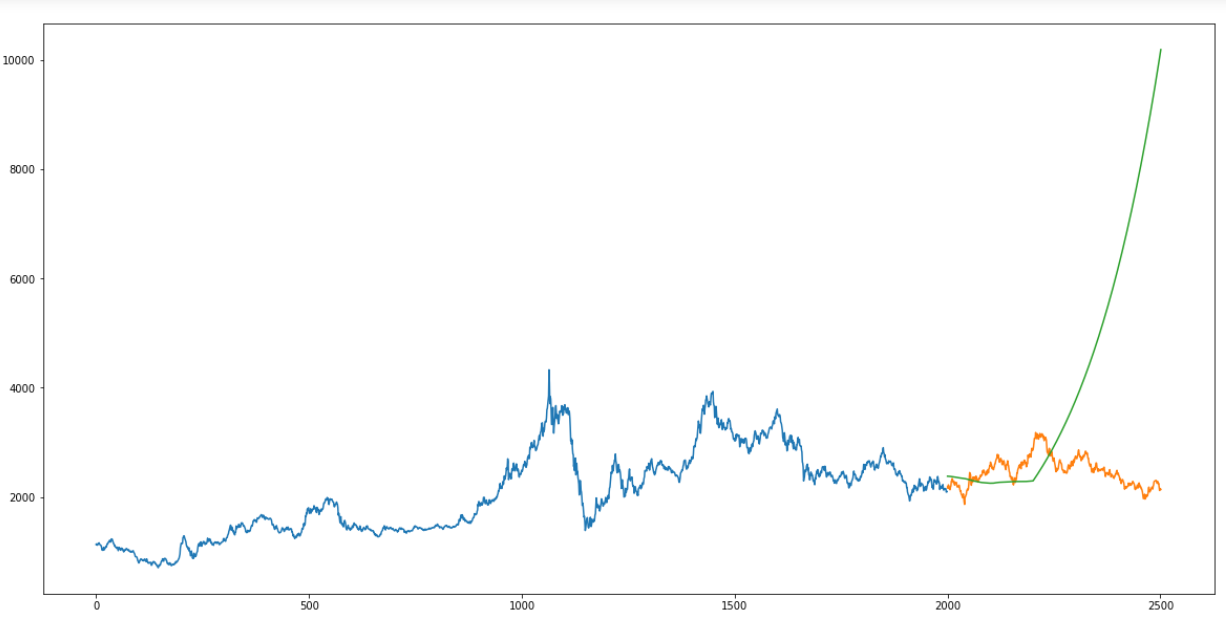
RMSE Table:

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| **1 day** | **676.32** |
| **2 day** | **823.88** |
| **3 day** | **765.90** |
| **4 day** | **845.18** |
| **5 day** | **628.10** |

**Code Snippet:**



**Predicted vs actual:**



1. **Long Short Term Memory (LSTM):**

**Background:**

LSTMs are widely used for sequence prediction problems and have proven to be extremely effective. The reason they work so well is because LSTM is able to store past information that is important, and forget the information that is not. LSTM has three gates:

The input gate: The input gate adds information to the cell state

The forget gate: It removes the information that is no longer required by the model

The output gate: Output Gate at LSTM selects the information to be shown as output

**Key Terms:**

**Vanishing Gradient:** Vanishing Gradient problem arises while training an Artificial Neural Network. This mainly occurs when the network parameters and hyper parameters are not properly set. Parameters could be weights and biases while hyper parameters could be learning rate, the number of epochs, the number of batches, etc.

**Activation function:** In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs.

**Sigmoid function:** A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve.

**Libraries:** Keras sequential, Drop, Dropout, LSTM

**Trained Models:**

**1 day Model:** LSTM Model for 1 day prediction

**2 day Model:** LSTM Model for 2 day prediction

**3 day Model:** LSTM Model for 3 day prediction

**4 day Model:** LSTM Model for 4 day prediction

**5 day Model:** LSTM Model for 5 day prediction

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Add LSTM Model: After importing the LSTM from Sequential model, we have created the object of Sequential and add Sequential object in Model.

4. Compile the Model: Compilation is prerequisite prior to fit the model so we have compiled the model.

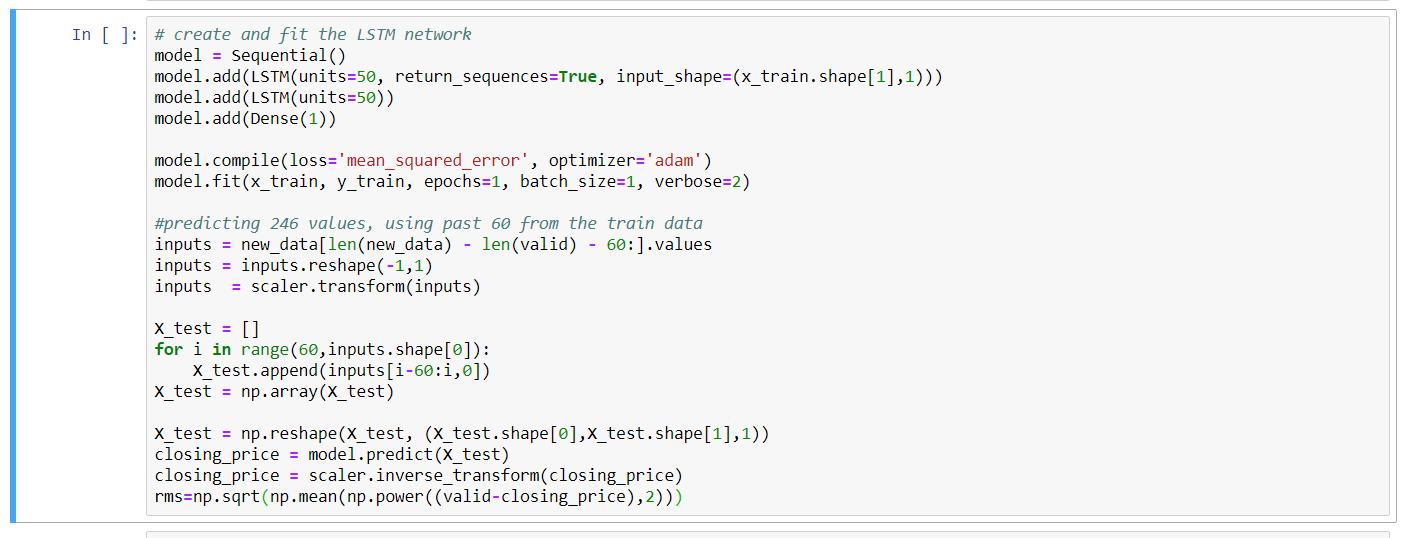
5. Fit the Model: After compilation model is ready for fitting. we have fitted the model with training data and default parameters epochs=1, batch\_size=1, verbose=2

6. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

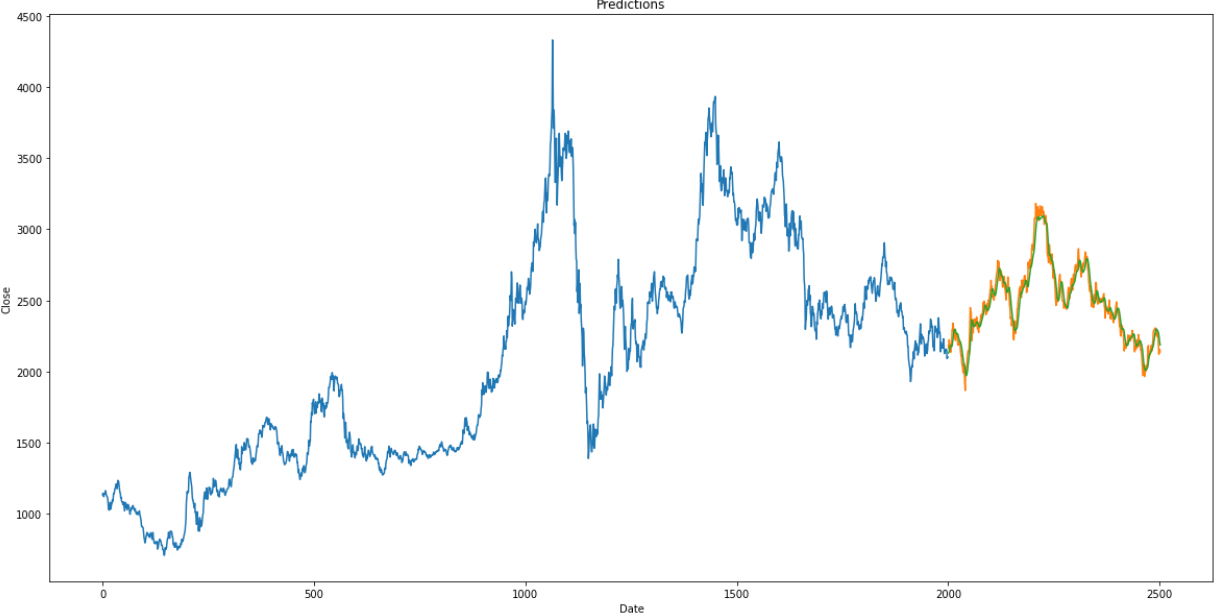
**RMSE Table**

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| **1 day** | **66.83** |
|  |  |
|  |  |
|  |  |
|  |  |

**Code Snippet:**



**Predicted vs actual:**



**Code Flow:**

1. Put data files under data directory/ directory from where running the jupyter file.
2. Run price prediction Jupiter files