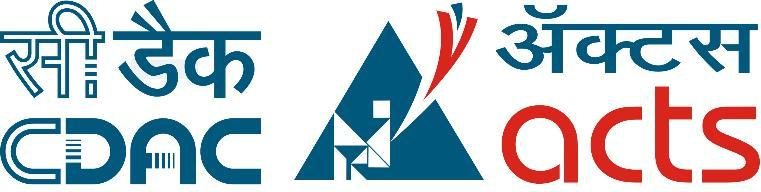
A

Project Report on

COMMODITY PRICE PREDICTION

Submitted in partial fulfilment for the award of

**PG DIPLOMA IN BIG DATA ANALYTICS**



CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING (C-DAC)

ADVANCED COMPUTING TRAINING SCHOOL (ACTS)

Knowledge Park, Bangalore - 560 038

Submitted By

ANISH KOTHARI PRN: 170850125004

VINEET SAJWAN PRN: 170850125044

MUKESH JHA PRN: 170850125018

BARGE SAURABH PRN: 170850125008

Guided by

SWAPNIL SRIVASTAVA

**CERTIFICATE**

This is to certify that, the project report entitled

**COMMODITY PRICE PREDICTION**

Submitted By

ANISH KOTHARI PRN: 170850125004

VINEET SAJWAN PRN: 170850125044

MUKESH JHA PRN: 170850125018

BARGE SAURABH PRN: 170850125008

is the record of bonafide work carried out by them in partial fulfilment of the requirement for the award of **PG Diploma in Big Data Analytics** prescribed by **Centre For Development Of Advanced Computing (C-DAC)**.

Ms. SWAPNIL SHRIVASTAVA Ms. M. Savitri

Project Guide Course Co-ordinator

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to all the people without whom this project would have been highly impossible.

We would like to devote our first vote of thanks to our guide Ms.Swapnil Shrivastava for her constant support and encouragement. She has a great hand in the firm foundation of this project. We are deeply in debt for his valuable suggestions, scholarly guidance and constructive criticisms along with constant encouragement at each and every step for successful completion of the project.

We would also like to thank our Project Co-ordinator Mrs. Janaki for inspiring us towards completion of this project.

Last but not the least we would like to thank all those who assisted us directly or indirectly for their valuable time and help.

**ABSTRACT**

Commodity price prediction serves as an important quantitative basis for commodity production planning-in particular for capacity planning. High level decision and planning in commodity production relies heavily on temperature and past commodity price. Many researches have shown that commodity price is subject to great volatile now then has been the case in the past. Many past predictive models for commodity price models have mixed performance due to unanticipated events and circumstances in the forecasts.

The goal of this analysis is to provide a realistic forecast based on latest available data to reflect the current price of the commodity, supported by information in the study providing an adequate justification for the commodity production pricing and price of commodity.

The aim here is to develop a model that can accurately predict the price of commodity monthly, daily and weekly for a particular district using the dataset available.

**TABLE OF CONTENTS**

1. INTRODUCTION
2. TOOLS TO BE USED #
3. DESIGN APPROACH FLOWCHART #
4. DATA SET DESCRIPTION #
5. INITIAL HYPOTHESIS #
6. RECURRENT NEURAL NETWORK #
7. IMPLEMENTATION OF THE ARIMA MODEL #
8. IDENTIFYING THE POSSIBLE MODEL #
9. COMPARING THE MODELS #
10. CONCLUSION

**`1. Introduction**

This document outlines the software requirements for analysing and forecasting a time series commodities price data. It will cover the overall description of the system, specific requirements as well as modelling requirements, diagrams and a description of a prototype to be built to demonstrate the system’s functionality

**1.1 Document Purpose**

This document is intended to inform those who have a vested interest in the time series data analysis of its purpose and design. It should be useful to the customer(s) as well as any members of whole sale marketers who are tasked to supply daily basis commodity to the society at efficient price. Users of the system need not concern themselves with the information in this document unless they desire a deeper understanding of the system’s architecture. However, this document does not focus on ways in which the government role to increase/decrease the price of the commodity.

**1.2 Scope**

This document concerns time series data analysis and how do we manage to perform that. This analysis will be contained statistical inferences between the data and it’s modelling in order to better assist how the price can be predicted. It is intended to focus on daily price time series data along with rainfall and temperature. The model will provide forecasting of price for upcoming days, week or month. The model also shows how daily rainfall and temperature could affect the price for upcoming days. The system will receive data in the form of dates and predicts the price for upcoming days. It also improve itself as we store more data. We also represent the model in such way that any user can easily understand how the price changes with respective to the date.

**2. Tools to be used.**

**a. Programming Language**

• Python

• R programming

**b. Libraries For Python:-**

• Keras (Tensorflow in backend)

• sklearn

**For R:-**

• TTR

• forecast

• tseries

• lubridate

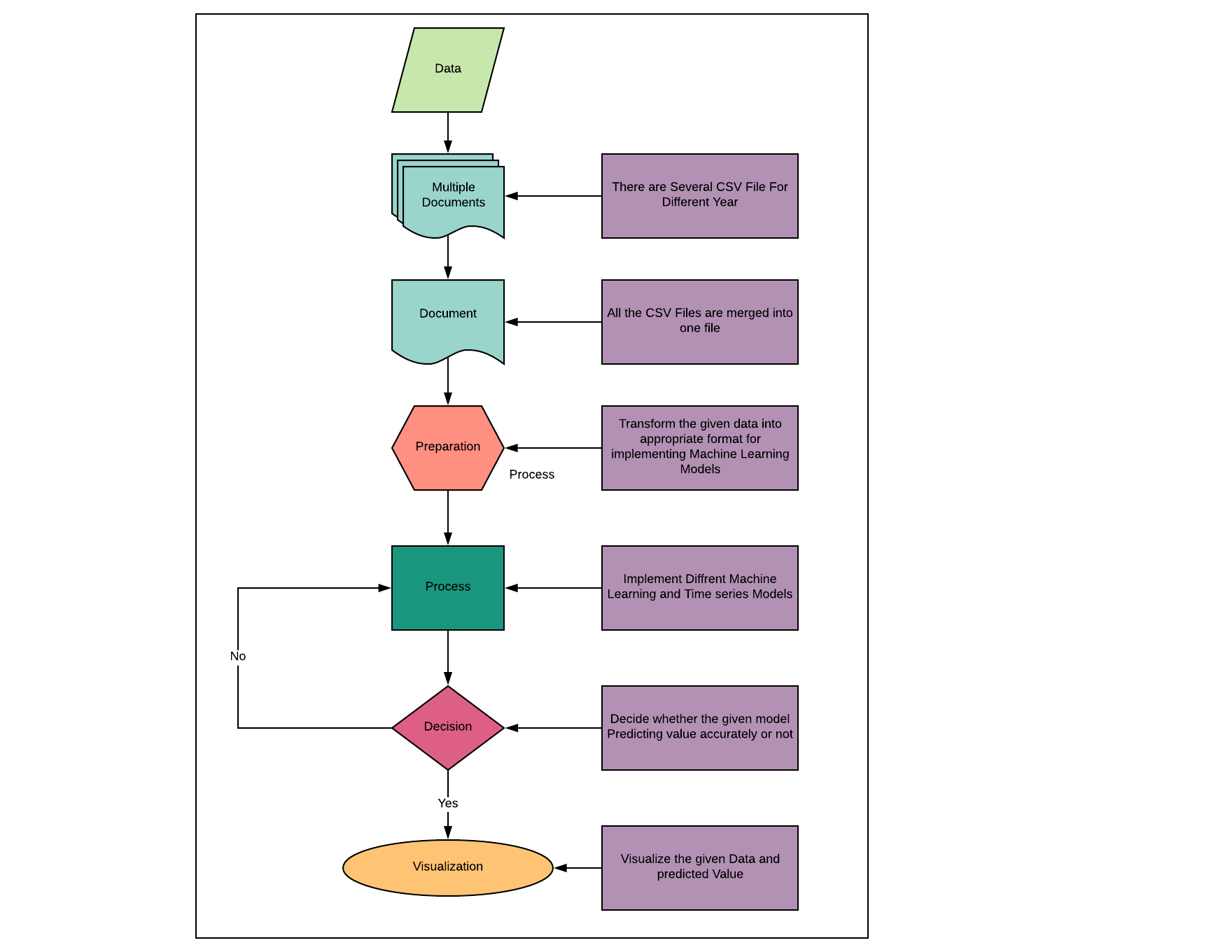
• inferr

• stats

**c. Visualisation Tool**

• Tableau

**3.Design Approach Flowchart**

****

**4. Dataset Description (Name, Size etc.)**

**Columns**: The Data set contains 9 columns; The Initial 4 Columns are Commodity Name, State, District and Market for that Commodity

Remaining 7 Columns are Arrival Date, Min\_Price, Max\_price, Modal\_price, Arrival Quantity,Rainfall,Temp.

**Size**: The total Size of Data is around 50MB and it Contain prices of Commodity for last 5 Years with Different market.

**Dimensions**: No of rows = 935522, No of columns = 16,Time Period = 2012 to 2017

**Data Dictionary**

**1. Comm\_name**: Name of the Commodity, there are 5 Commodity in The Data

**2. State**: Describe the State for which prediction going to be done

**3. District\_name**: Each state have several no of districts

**4. Market\_centre\_name**: Name of the market

**5. Arrival\_date**: Date of arrival of commodity in that Market

**6. MinPrice**: Minimum price for which given commodity has been sold in the market

**7. MaxPrice**: Maximum price for which given commodity has been sold in the market

**8. ModalPrice**: Price between minimum and maximum price.

**9. ArrivalQuantity**: Amount of Quantity of commodity arrived in the market.

**10. Rainfall**: Record of the rain.

**11. Temp**: Daily recorded temperature.

**5. Initial Hypothesis:**

Initially the Data-Set received is a non-continuous Time series Data. As the target column is the price of the commodity which is dependent on many socio-economic factors as well as market condition at that time. And even other Factors such as Temperature and Rainfall at that time.

**Brief idea about the models used:**

**ARIMA :**

Several research studies on stock or commodity price predictions have been conducted with various solution techniques proposed over the years. The prominent techniques fall into two broad categories, namely, statistical and soft computing techniques. Statistical techniques include, among others, exponential smoothing, autoregressive integrated moving average (ARIMA), and generalized autoregressive conditional heteroscedasticity (GARCH) volatility. The ARIMA model, also known as the Box-Jenkins model or methodology, is commonly used in analysis and forecasting. It is widely regarded as the most efficient forecasting technique in social science and is used extensively for time series. The use of ARIMA for forecasting time series is essential with uncertainty as it does not assume knowledge of any underlying model or relationships as in some other methods. ARIMA essentially relies on past values of the series as well as previous error terms for forecasting. However, ARIMA models are relatively more robust and efficient than more complex structural models in relation to short-run forecasting .

**RNN :**

Artificial neural networks (ANNs) as a soft computing technique are the most accurate and widely used as forecasting models in many areas including social, engineering, economic, business, finance, foreign exchange, and stock problems . Its wide usage is due to the several distinguishing features of ANNs that make them attractive to both researchers and industrial practitioners. As stated in , ANNs are data-driven, self-adaptive methods with few prior assumptions. They are also good predictor with the ability to make generalized observations from the results learnt from original data, thereby permitting correct inference of the latent part of the population. Furthermore, ANNs are universal approximator as a network can efficiently approximate a continuous function to the desired level of accuracy. Finally, ANNs have been found to be very efficient in solving nonlinear problems including those in real world. This is in contrast to many traditional techniques for time series predictions, such as ARIMA, which assume that the series are generated from linear processes and as a result might be inappropriate for most real-world problems that are nonlinear. There is growing need to solve highly nonlinear, time-variant problems as many applications such as stock markets are nonlinear with uncertain behaviour that changes with time . ANNs are known to provide competitive results to various traditional time series models such as ARIMA model. In this paper, the performance of ANN and ARIMA models is studied and compared for a case of stock prediction, which also further clarify and/or confirm contradictory opinions reported in literature about superiority of each of the model over one another.

Data Preparation

In data preparation we followed the following steps:-

1) Date handling

2) Make the data continuous and replace NAs

3) Merge the weather data and the continuous data

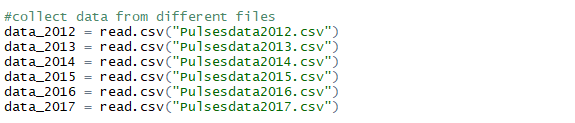
4) Statistical Inference

5) Conclusion

Date Handling:-

Collected data from different files into single file.

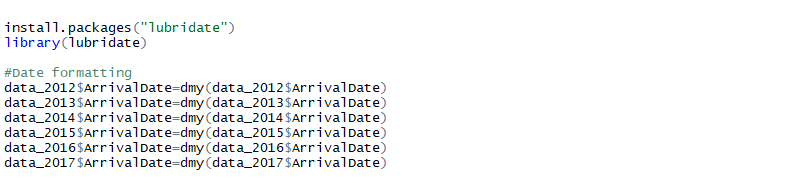
We used read.csv for collection data from different sources.



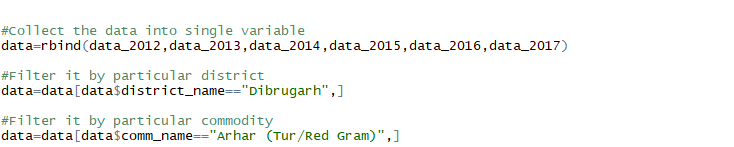
We used rbind function in R to append all data into single variable

.

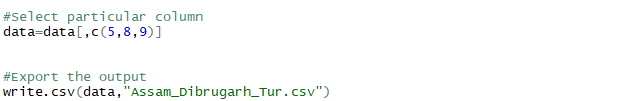
For date handling we used lubridate package. In lubridate package we have dmy function. Since our data is in the form of date-month-year, we used dmy function and pass the Arrival Date variable inside it.



We append the data by using rbind and apply filter according to our requirement. Since we need data for Dibrugarh district and Tur commodity, so we applied the filter accordingly.



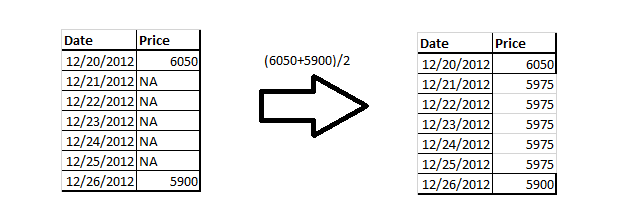
We selected column according to our requirement and export the file by using write.csv function.



Make the data continuous and replace NAs

Our data isn’t continuous due to which we won’t able to do proper forecasting. So we make it continuous by replacing the null values with the average of consecutive non-null values where it is fallen between.

Please check the below image for more clarification.



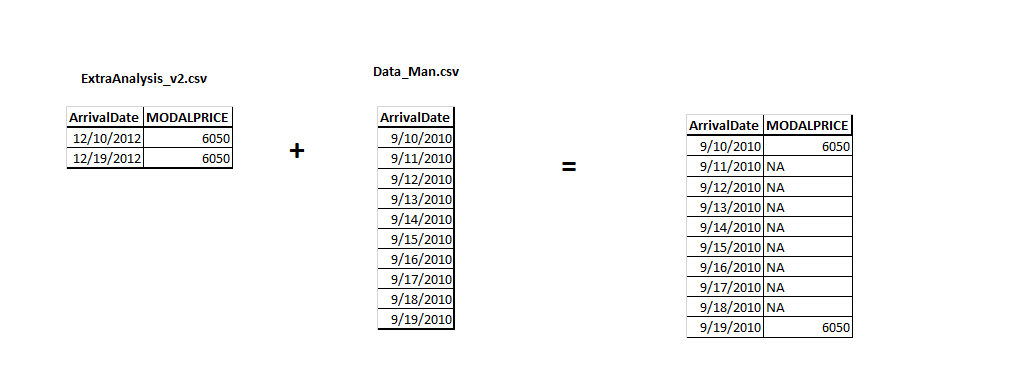
We have taken two files one with continuous date only (Data\_Man) and the discontinuous weather commodity data (ExtraAnalysis\_v2).We also formatted the date column.



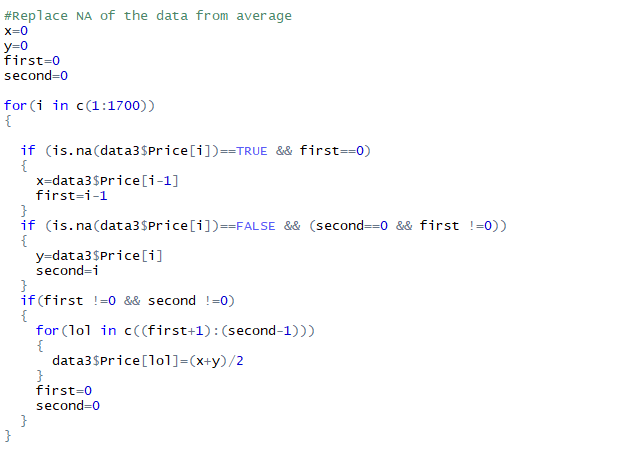
We merge the two files



So the final file be merged like below diagram:-

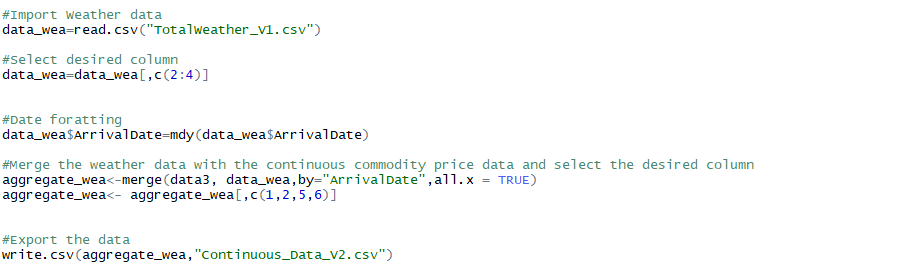


Now we replaces the NAs with the average of consecutive non NA values. As we shown above.



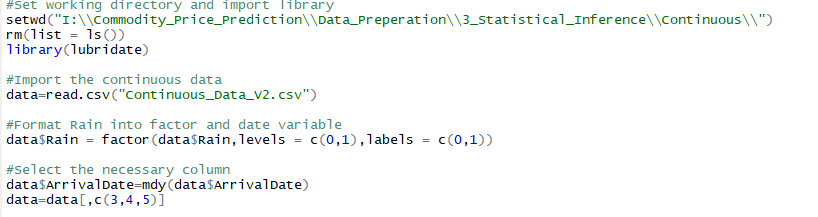
Merge the weather data and the continuous data

Now we import the weather data and merge the continuous data with the weather data and export the output.



**Statistical Inference**

Import the continuous data and formatted it (as the below code is self-explanatory) and select the column which is used for statistical inference.



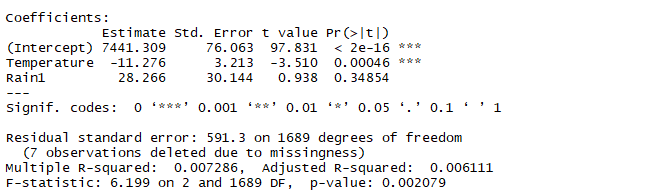
We checked the correlation between Price, rainfall and temperature.

We use lm function for checking the correlation.

In this model the Null hypothesis is that all the variable is independent from each other which means Temperature and Rain doesn’t make any impact on Price. Alternative hypothesis is that Temperature and Rain impact price.



Output:-



Below are following takeaways from the above output:-

1) Since the p-value is so small from significance value we have to adapt alternative hypothesis which is Temperature makes impact on the price.

2) The Estimate coefficient is negative which means temperature make negative impact on the price.

3) The p-value of Rain is higher than we adapt the hypothesis that the Rain doesn’t make any impact on the price.

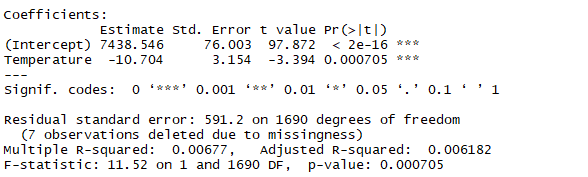
4) The value of Adjusted R-square is 0.00611.

5) We will remove Rain and check whether our model is improve or not.

Eliminated Rain and run the model again



**Output:-**

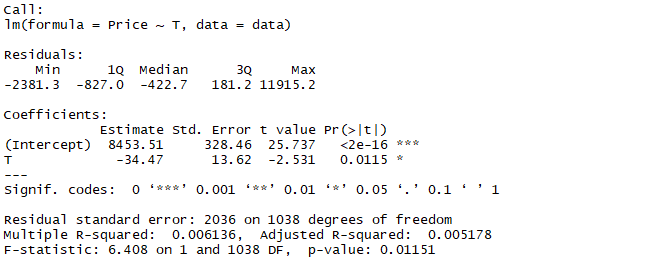


Hence our R-squared value is improved so we came to final conclusion that Rain doesn’t make any impact on the Price.

We applied same thing with discontinuous data as well to check whether our continuous data is better than from discontinuous data or not.

Hence our R-squared value is lower than continuous data R-squared value which is 0.006. It proved that our decision to make data continuous is correct.

**Output**:-



Conclusion:

1.Continuous data make our data and forecasting more robust.

2.Temperature is negatively correlated with the price. To be precise, if the temperature is decreased by 1 unit the price will increase by 35 unit.

3.Rain doesn’t make any impact on the Price.

**Recurrent Neural Network**

Time series prediction problems are a difficult type of predictive modelling problem.

Unlike regression predictive modelling, time series also adds the complexity of a sequence dependence among the input variables.

A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.

During this project, we learned how to develop LSTM networks in Python using the Keras deep learning library to address a demonstration time-series prediction problem.

After Reading this part you will know how to implement and develop LSTM networks for your own time series prediction problems and other more general sequence problems.

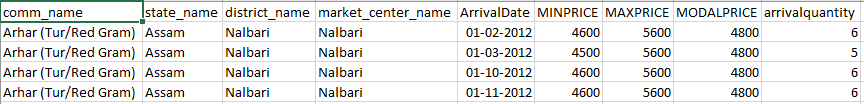
**Problem Description**

The problem we are going to look at in this project is the Commodity Price Prediction problem.

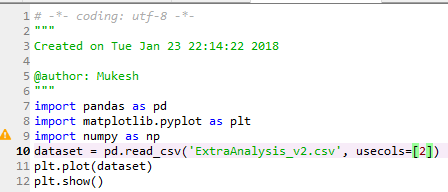
This is a problem where, a commodity name and price of the commodity for last five years per quintals for each is given, the task is to predict the Price of commodity for the very next day precisely. The data ranges from January 2012 to December 2017, or 5 years, with several missing observations.

The dataset is provided by the Centre of development of advanced computing (Bengaluru)

Below is a sample of the first few lines of the file:

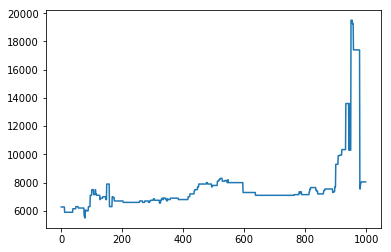


We can load this dataset easily using the Pandas library. We are not interested in the date, Because Recurrent Neural network doesn’t take date as parameter when we load the dataset we can exclude the date column.



You can see an upward trend in the dataset over time.

You can also see there is no periodicity to the dataset that probably corresponds time.



We are going to keep things simple and work with the data as-is.

Normally, it is a good idea to investigate various data preparation techniques to rescale the data and to make it stationary.

## Long Short-Term Memory Network

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem.

As such, it can be used to create large recurrent networks that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results.

Instead of neurons, LSTM networks have memory blocks that are connected through layers.

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block’s state and output. A block operates upon an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional.

There are three types of gates within a unit:

* **Forget Gate**: conditionally decides what information to throw away from the block.
* **Input Gate**: conditionally decides which values from the input to update the memory state.
* **Output Gate**: conditionally decides what to output based on input and the memory of the block.

Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

You can see how you may achieve sophisticated learning and memory from a layer of LSTMs, and it is not hard to imagine how higher-order abstractions may be layered with multiple such layers.

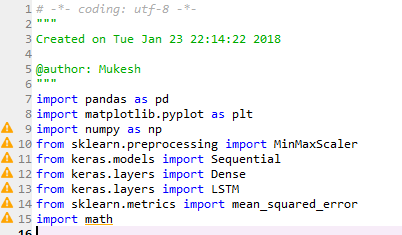
## LSTM Network for Regression

We can phrase the problem as a regression problem.

That is, given the price of commodity this day, what is the price of commodity next day in that market?

We can write a simple function to convert our single column of data into a two-column dataset: the first column containing this day’s (t) commodity price and the second column containing next day’s (t+1) commodity price, to be predicted.

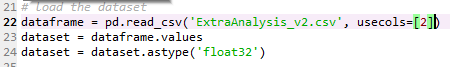
Before we get started, let’s first import all of the functions and classes we intend to use.



Before we do anything, it is a good idea to fix the random number seed to ensure our results are reproducible.



We can also use the code from the previous section to load the dataset as a Pandas DataFrame. We can then extract the NumPy array from the DataFrame and convert the integer values to floating point values, which are more suitable for modeling with a neural network.

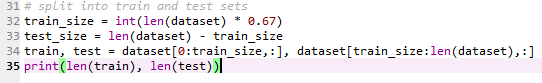


LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. We can easily normalize the dataset using the **MinMaxScaler** pre-processing class from the scikit-learn library.



After we model our data and estimate the skill of our model on the training dataset, we need to get an idea of the skill of the model on new unseen data. For a normal classification or regression problem, we would do this using cross validation.

With time series data, the sequence of values is important. A simple method that we can use is to split the ordered dataset into train and test datasets. The code below calculates the index of the split point and separates the data into the training datasets with 67% of the observations that we can use to train our model, leaving the remaining 33% for testing the model.

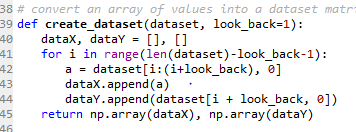




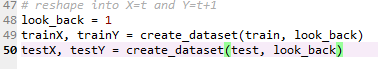
Now we can define a function to create a new dataset, as described above.

The function takes two arguments: the **dataset**, which is a NumPy array that we want to convert into a dataset, and the **look\_back**, which is the number of previous time steps to use as input variables to predict the next time period — in this case defaulted to 1.

This default will create a dataset where X is the Price of commodity at a given time (t) and Y is the price of commodity at the next time (t + 1).



Let’s use this function to prepare the train and test datasets for modeling.



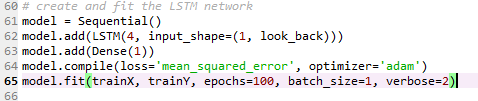
The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: [samples, time steps, features].

Currently, our data is in the form: [samples, features] and we are framing the problem as one-time step for each sample. We can transform the prepared train and test input data into the expected structure using **numpy.reshape()** as follows:



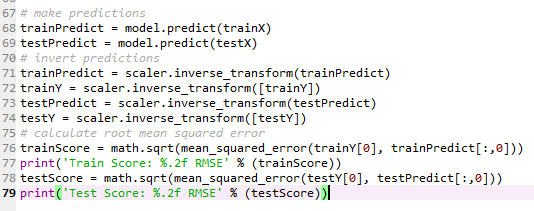
We are now ready to design and fit our LSTM network for this problem.

The network has a visible layer with 1 input, a hidden layer with 4 LSTM blocks or neurons, and an output layer that makes a single value prediction. The default sigmoid activation function is used for the LSTM blocks. The network is trained for 100 epochs and a batch size of 1 is used.



Once the model is fit, we can estimate the performance of the model on the train and test datasets. This will give us a point of comparison for new models.

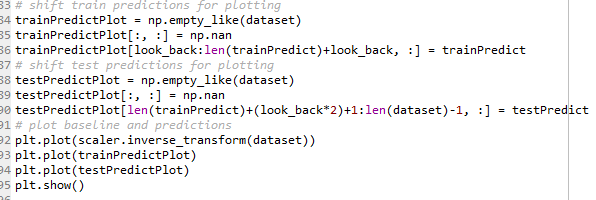
Note that we invert the predictions before calculating error scores to ensure that performance is reported in the same units as the original.



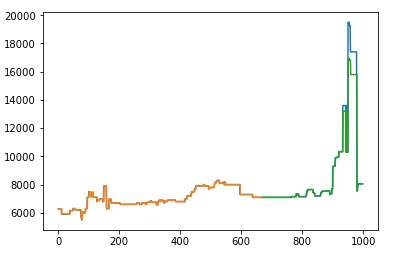


Finally, we can generate predictions using the model for both the train and test dataset to get a visual indication of the skill of the model.

Because of how the dataset was prepared, we must shift the predictions so that they align on the x-axis with the original dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in green, and the predictions on the unseen test dataset in Green.



We can see that the model did an excellent job of fitting both the training and the test datasets.



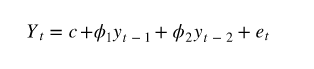
**Implementation of ARIMA:**

**(Auto Regressive Integrated Moving Average)**

Several research studies on stock or commodity price predictions have been conducted with various solution techniques proposed over the years. The prominent techniques fall into two broad categories, namely, statistical and soft computing techniques. Statistical techniques include, among others, exponential smoothing, autoregressive integrated moving average (ARIMA), and generalized autoregressive conditional heteroscedasticity (GARCH) volatility. The ARIMA model, also known as the Box-Jenkins model or methodology, is commonly used in analysis and forecasting. It is widely regarded as the most efficient forecasting technique in social science and is used extensively for time series. The use of ARIMA for forecasting time series is essential with uncertainty as it does not assume knowledge of any underlying model or relationships as in some other methods. ARIMA essentially relies on past values of the series as well as previous error terms for forecasting. However, ARIMA models are relatively more robust and efficient than more complex structural models in relation to short-run forecasting .

ARIMA stands for auto-regressive integrated moving average and is specified by these three order parameters: (p, d, q). The process of fitting an ARIMA model is sometimes referred to as the Box-Jenkins method.

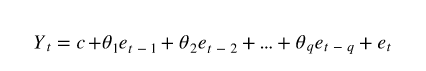
An auto regressive (AR(p)) component is referring to the use of past values in the regression equation for the series Y. The auto-regressive parameter p specifies the number of lags used in the model. For example, AR(2) or, equivalently, ARIMA(2,0,0), is represented as



Where φ1, φ2 are parameters for the model.

The d represents the degree of differencing in the integrated (I(d)) component. Differencing a series involves simply subtracting its current and previous values d times. Often, differencing is used to stabilize the series when the stationarity assumption is not met.

A moving average (MA(q)) component represents the error of the model as a combination of previous error terms et. The order q determines the number of terms to include in the model



Differencing, autoregressive, and moving average components make up a non-seasonal ARIMA model which can be written as a linear equation:



where yd is Y differenced d times and c is a constant.

Note that the model above assumes non-seasonal series, which means you might need to de-seasonalize the series before modelling.

ARIMA models can be also specified through a seasonal structure. In this case, the model is specified by two sets of order parameters: (p, d, q) as described above (P, D, Q) parameters describing the seasonal component of m periods.

ARIMA methodology does have its limitations. These models directly rely on past values, and therefore work best on long and stable series. Also note that ARIMA simply approximates historical patterns and therefore does not aim to explain the structure of the underlying data mechanism.

* ARIMA has Basically 3 parameters ARIMA(p,d,q)
* p = order of autoregressive model
* d = degree of differencing.
* q = order of moving average model.
* It is available in Forecast package of R.
* And is highly Flexible for Any kind of Data Seasonal or Stationary

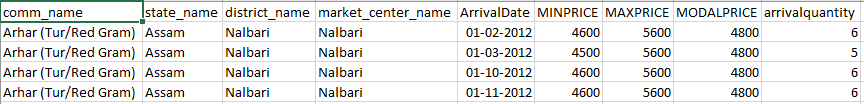
**PROBLEM STATEMENT:**

The problem we are going to look at in this project is the Commodity Price Prediction problem.

This is a problem where, a commodity name and price of the commodity for last five years per quintals for each is given, the task is to predict the Price of commodity for the very next day precisely. The data ranges from January 2012 to December 2017, or 5 years, with several missing observations.

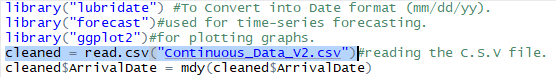
The dataset is provided by the Centre of development of advanced computing (Bengaluru)

Below is a sample of the first few lines of the file:



The data can be loaded into R using file I/O operations and the the necessary packages must be loaded such as:

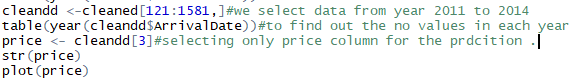
* lubridate: To convert Dates into Desired format.
* Forecast: for ARIMA modelling and forecasting.
* ggplot: for plotting the Graph.



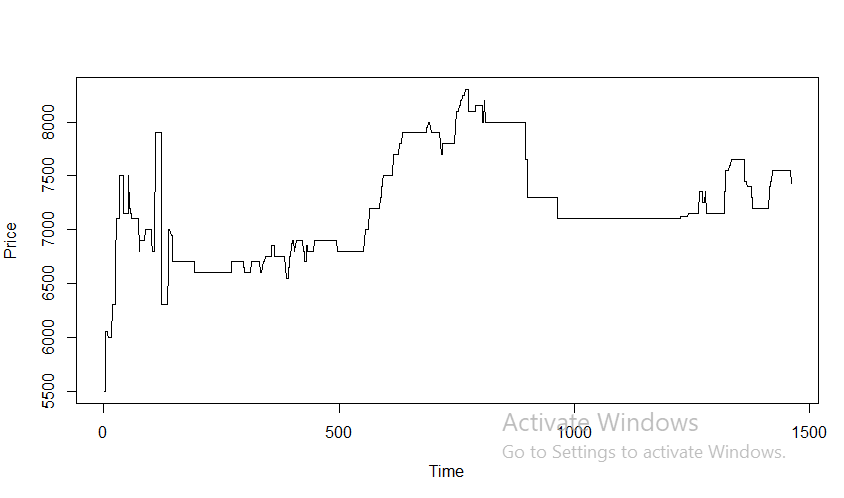
As the Data obtained was dis-continuous and had many missing values Data cleaning had to be done. And the Data was to be made into continuous form before loading into R.

|  |
| --- |
| table(year(cleaned$ArrivalDate))  2010 2011 2012 2013 2014 2015  120 365 366 365 365 118 |
|  |
| |  | | --- | |  | |

is used to know the Distribution of Data according to years .



Plot shows the Distribution of Value of prices from year 2012 to 2014:



**ADF test** :

The Augmented Dickey Fuller Test (ADF) is unit root test for stationarity. Unit roots can cause unpredictable results in your time series analysis.

The Augmented Dickey-Fuller test can be used with serial correlation. The ADF test can handle more complex models than the Dickey-Fuller test, and it is also more powerful. That said, it should be used with caution because — like most unit root tests — it has a relatively high Type I error rate.

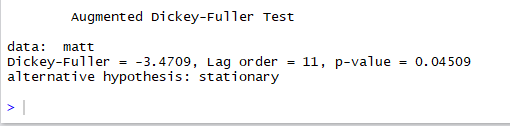
Hypotheses:

The hypotheses for the test:

The null hypothesis for this test is that there is a unit root.

The alternate hypothesis differs slightly according to which equation you’re using. The basic alternate is that the time series is stationary (or trend-stationary).

**When we use the ADF test on price Data we get :**



Here the p value is Less than 5% in general, a [p-value](http://www.statisticshowto.com/p-value/) of less than 5% means you can [reject the null hypothesis](http://www.statisticshowto.com/support-or-reject-null-hypothesis/) that there is a unit root. You can also compare the calculated DFTstatistic with a tabulated [critical value](http://www.statisticshowto.com/probability-and-statistics/find-critical-values/). If the DFT statistic is more negative than the table value, reject the null hypothesis of a unit root.

Hence the Dickey-fuller-test states that it has no unit root .

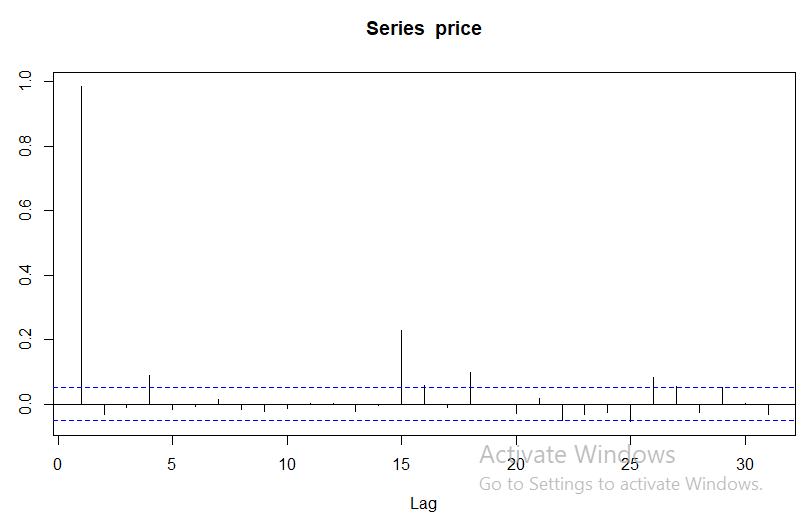
ACF and PACF plots: After a time series has been stationarized by differencing, the next step in fitting an ARIMA model is to determine whether AR or MA terms are needed to correct any autocorrelation that remains in the differenced series. Of course, with software like Stat-graphics, you could just try some different combinations of terms and see what works best. But there is a more systematic way to do this. By looking at the autocorrelation function (ACF) and partial autocorrelation (PACF) plots of the differenced series, you can tentatively identify the numbers of AR and/or MA terms that are needed. You are already familiar with the ACF plot: it is merely a bar chart of the coefficients of correlation between a time series and lags of itself. The PACF plot is a plot of the partial correlation coefficients between the series and lags of itself.

In general, the "partial" correlation between two variables is the amount of correlation between them which is not explained by their mutual correlations with a specified set of other variables. For example, if we are regressing a variable Y on other variables X1, X2, and X3, the partial correlation between Y and X3 is the amount of correlation between Y and X3 that is not explained by their common correlations with X1 and X2. This partial correlation can be computed as the square root of the reduction in variance that is achieved by adding X3 to the regression of Y on X1 and X2.

**The ACF plot:**



**PACF plot:**



We have seen that for MA(q) models, the ACF will be insignificant for lags > q. Therefore, the ACF is provides considerable information for specifying an MA(q) process. Unfortunately, if the process is ARMA or AR, the ACF alone yields little information about the orders of dependence (p, q). There is however, another function which can act like the ACF for an MA process, but for AR models; the partial autocorrelation function (PACF).

An easy example of a PACF can be explained using a linear regression where we predict y from x1, x2, and x3. Basically, in a PACF we want to, for example, correlate the “parts” of y and x3 that are not predicted by x1 and x2. However, what happens normally (non-partial) is that the linear dependency between x3 and y has accounted accounted for the dependency between y and x1 and x2.

Similarly for time series, For a time series, the PACF is the conditional autocorrelation between xs and xt, with the linear effect of everything in between those two points removed. Consider and AR(1) model, whereby the correlation between xt and xt - 2 is not zero, as it would be for an MA(1), because xt is dependent on xt - 2 through xt - 1. Thus, a PACF would break this chain of dependence by literally subtracting out (or partial out) the effect of xt - 1.

For an AR model, the theoretical PACF essentially “shuts off” the past order of the model. Thus, exactly the same as the MA(q) model selection using an ACF, we identify the order of the model by the number of non-zero partial autocorrelations. An AR(1) model will have one significant (non-zero) autocorrelation at lag 1 in the PACF.

**Identifying a possible model**

Three items should be considered to determine a first guess at the form of an ARIMA model:

1. Stationarity
2. ACF
3. PACF

**Time series plot**

What to look for: **possible trends**, **seasonality**, **outliers**, **constant/non-constant variance**.

* This is the most obvious first step to begin to understand the data; though you will not be able to spot a model, you could be informed of some possible next steps.
* If there is an obvious upward or downward linear trend, a first difference may be needed, or second differences for a quadratic term. Yet, *over differencing* can cause us to introduce unnecessary levels of dependency.
* For data with a curved upward trend and increasing variance, we can consider transforming the series with either logarithm or square root.

Note: non-constant variance in a series with no trend may indicate something like and ARCH model which is designed for modelling changing variance over time, we will cover this later.

**ACF and PACF**

The ACF and PACF should be considered together, as a few combining patterns will stand out, especially with experience.

* AR models have theoretical PACFs with non-zero values at the AR terms in the model, and zero values elsewhere. The ACF will taper to zero in some way.
* An AR (2) has a sinusoidal ACF that converges to zero.
* MA models have theoretical ACFs with non-zero values at the MA terms in the model, and zero terms elsewhere.
* ARMA models have ACFs and PACFs that both tail off to zero. These are difficult because the order will not be obvious. The best way to do it is guise a few terms and test their model estimates.
* If the ACF and PACF do not tail off, and have values close to 1 over many lags, the series is -non-stationary and differences is needed. Try first differencing then investigate ACF and PACFs.
* If all the autocorrelations are insignificant, then the series is random (white noise). Your work is easy and done in this case.
* If you take first difference and all the autocorrelations are insignificant, then the series is called a random walk, and you are done. The data are dependent and not identically distributed; both the mean and variance increase through time.

**Using auto.ARIMA ():**

Returns best ARIMA model according to either AIC, AICc or BIC value. The function conducts a search over possible model within the order constraints provided**.**

**Usage:**

auto.arima(y, d = NA, D = NA, max.p = 5, max.q = 5, max.P = 2,

max.Q = 2, max.order = 5, max.d = 2, max.D = 1, start.p = 2,

start.q = 2, start.P = 1, start.Q = 1, stationary = FALSE,

seasonal = TRUE, ic = c("aicc", "aic", "bic"), stepwise = TRUE,

trace = FALSE, approximation = (length(x) >150 | frequency(x) >12),

truncate = NULL, xreg = NULL, test = c("kpss", "adf", "pp"),

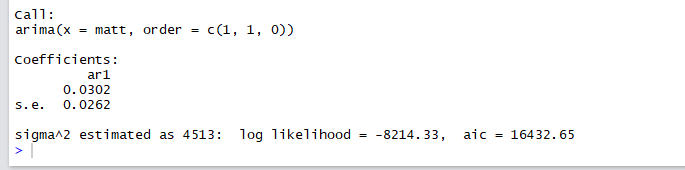
seasonal.test = c("ocsb", "ch"), allowdrift = TRUE, allowmean = TRUE,

lambda = NULL, biasadj = FALSE, parallel = FALSE, num.cores = 2,

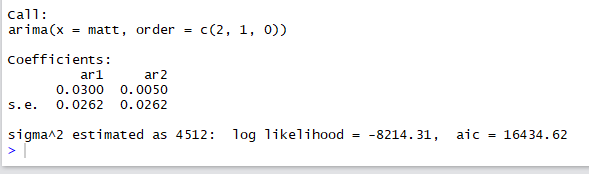
x = y, **...**).

If we use arima() according to ACF and PACF values , the Modell we come out with would be a zero or one AR term and requires a Differencing .

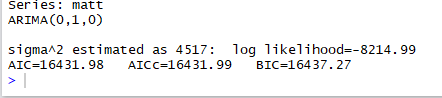
* arima(matt,order = c(1,1,0))



* arima(matt,order = c(2,1,0))



* arima(matt,order = c(0,1,0))



**In our code as we used auto. arima() for the Daily prediction :**

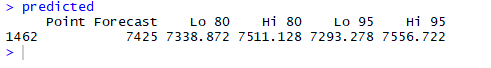
* Order of the auto - regressive Model = 0
* Degree of Differencing = 1
* Order of Moving Average Model = 0
* AIC = 16431.98
* AICc =16431.99
* BIC = 16437.27

**For getting the value of forecast for next day we have to fit the arima model and then forecast :**



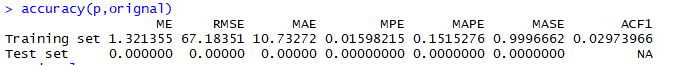
**OUT-PUT predicted price :**

**For date 1/1/2015:**

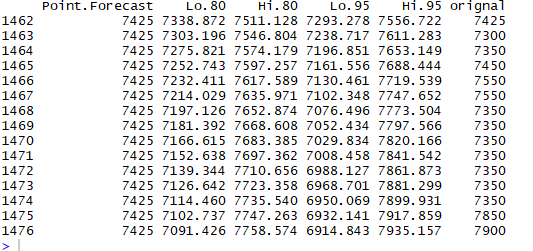
**Orignal Value:**

Date 1/1/2015: 7425

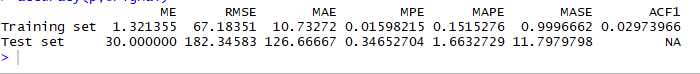
Accuracy measure:



**Predicted and original values for next 15 days:**



**Accuracy:**



As we can see the predicted value is exactly equal to original value but it is only for one day.

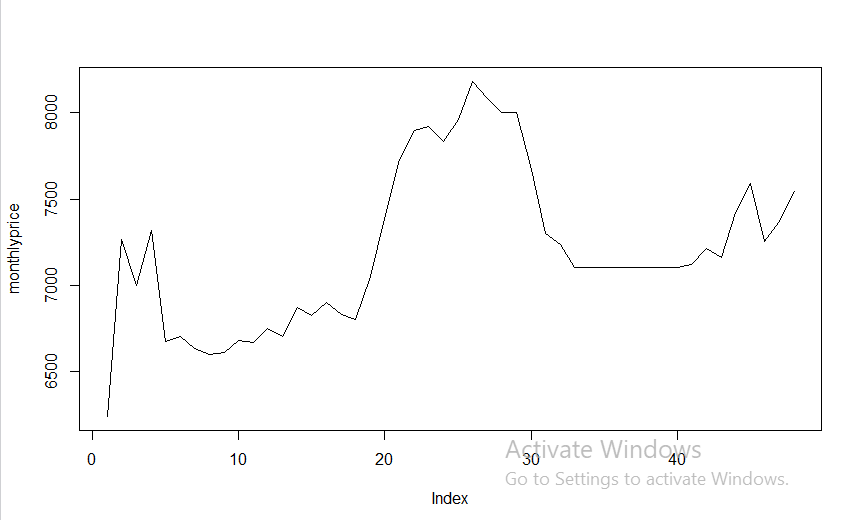
But the trend of Hi.95 is kind of correlating with the original value trend.

In similar way we calculate weekly average and Monthly average of the given Data.

The predicted value for the Monthly average value of January 2015 is:

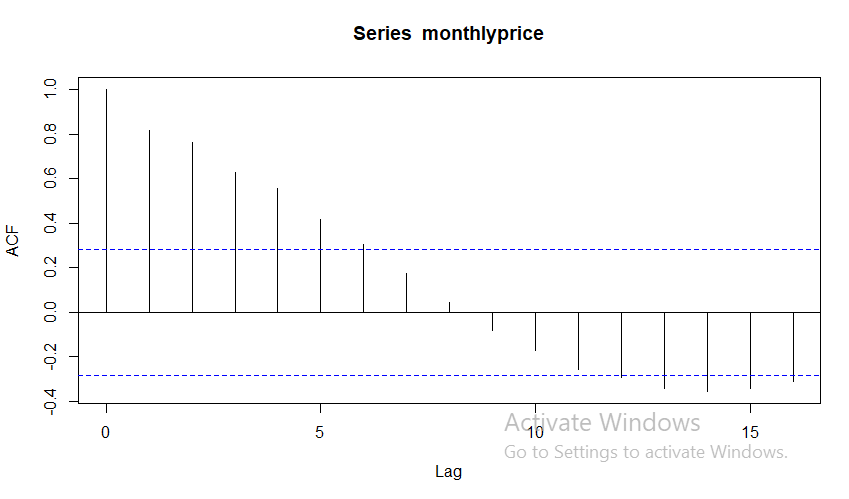
**For Monthly (Avg) model using Auto.ARIMA() :**

**Price plot for the Monthly Average from 2012-2014 is :**

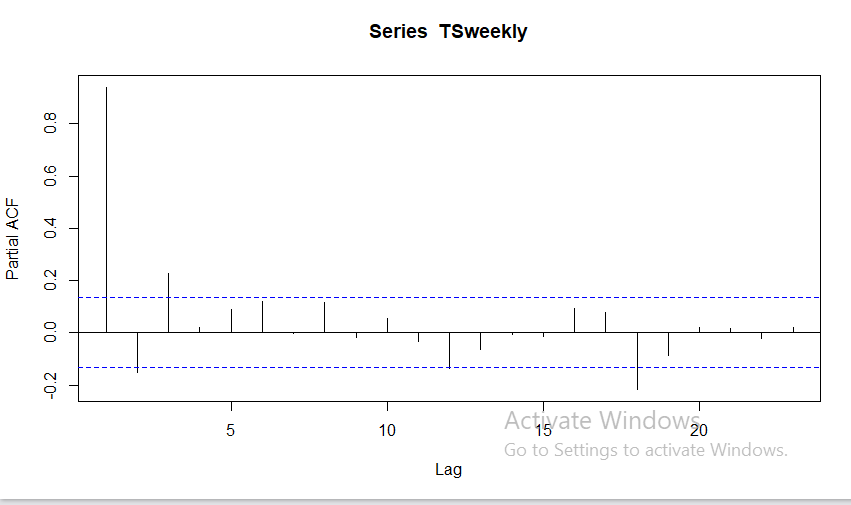


The plot shows non-seasonality and less Stationarity

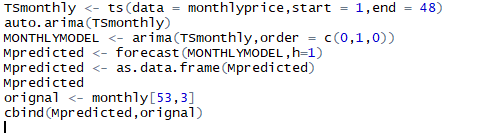
**The ACF and PACF plots are:**



**PACF:**



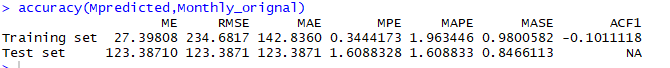
**CODE:**



**OUTPUT with Original value:**

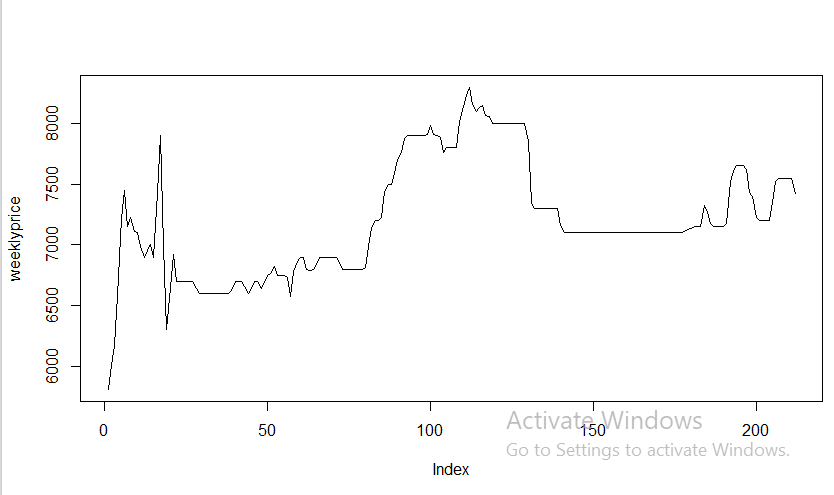


**Accuracy :**

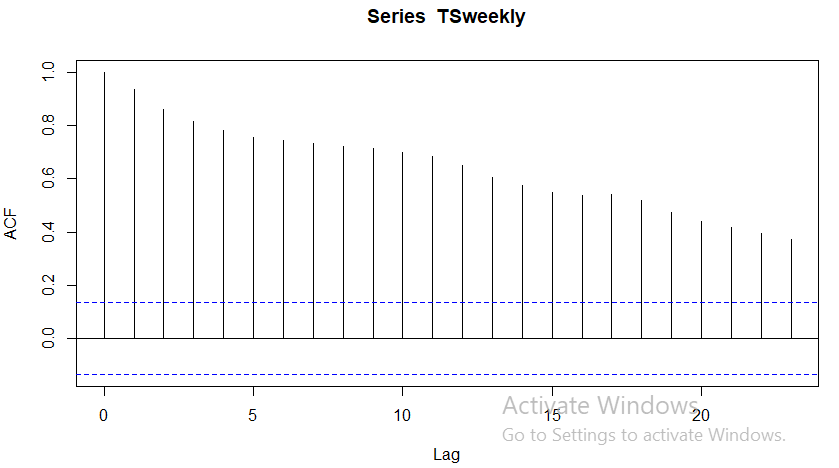


**For Weekly (avg) using auto.ARIMA model:**

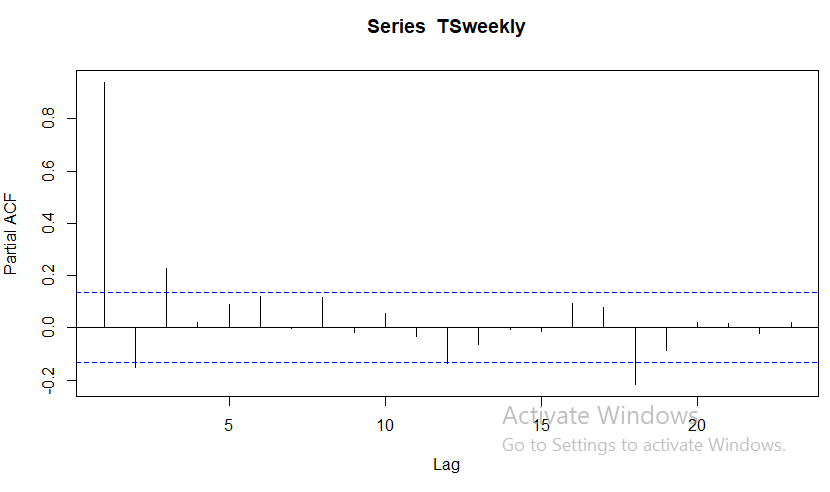
**Price Plot for Weekly (avg): 2012-2014**



**ACF and PACF plots for Weekly (avg): 2012-2014**

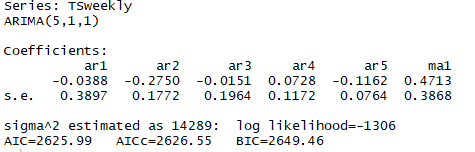


**PACF plot :**

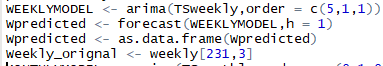


**Using auto.ARIMA for Weekly (AVG) we get c (p,d,q) as**

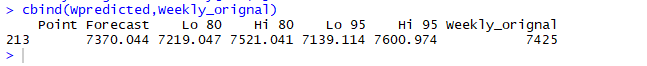
**ARIMA (5,1,1)**



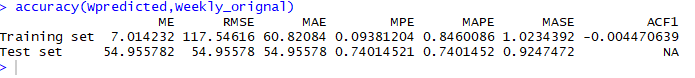
**CODE :**



**OUTPUT with Original value:**



**Accuracy:**

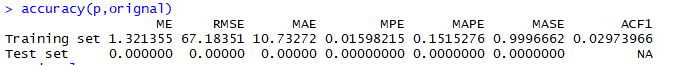


**Comparing Models:**

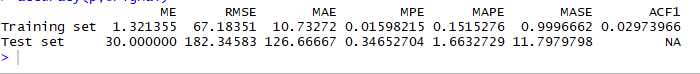
As we have applied both the models, we compare both the models:

**ARIMA models:**

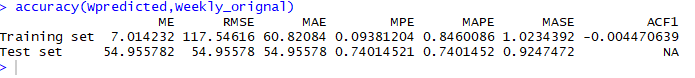
**For Daily price prediction:**



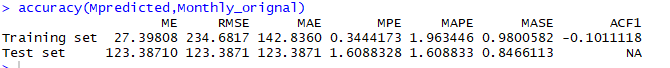
**For Daily price predication for 15 days :**



**For Weekly price prediction (Avg):**

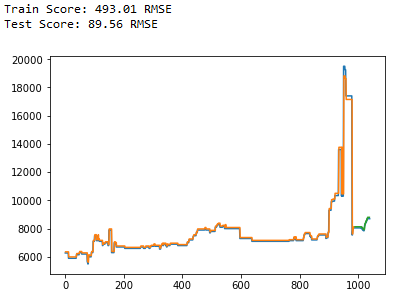


**For Monthly price prediction (Avg):**

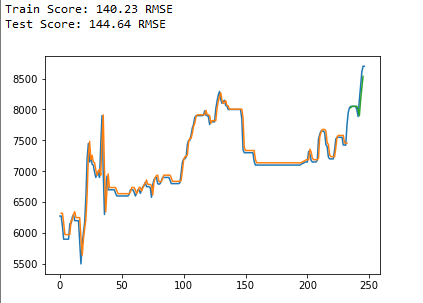


**RNN (Recurrent Neural Network) :**

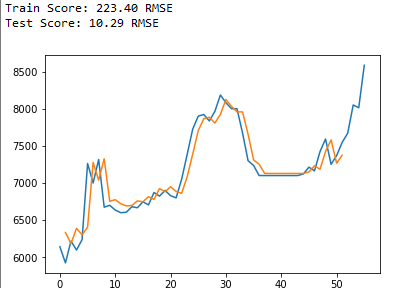
**For Daily Price prediction:**

****

**For weekly price prediction :**

****

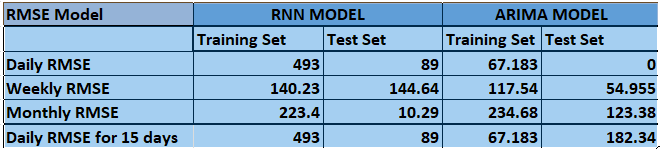
**For Monthly price prediction(Avg):**

****

**CONCLUSION:**

By comparing both the models there is a common accuracy

Measure: RMSE (Root mean square error)



RNN model will be used for monthly price prediction.

For a single day and single week (AVG) price prediction ARIMA will be used.

Whereas for continuous Daily data prediction RNN is preferred.