AN INDUSTRIAL TRAINING REPORT

on

Time Series Analysis

Submitted by

Abhijeet Katiyar
161500012

Department of Computer Engineering & Applications
Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA
August, 2018

Abstract

This report is simply introducing the univariate time series analysis and demonstrate the specific experiment on NIFTY50 data integrated with python and jupyter notebook. The Report contains introduction to time series, Internal structure of time series, methods to detecting trend and Seasonality and methods to removing them, forecasting the time series with the accuracy of 97% of predictions.

The dataset that is used in report is NIFTY50 data. The NIFTY 50 is the flagship index on the National Stock Exchange of India Ltd. (NSE). The Index tracks the behavior of a portfolio of blue-chip companies, the largest and most liquid Indian securities. It includes 50 of the approximately 1600 companies listed on the NSE, captures approximately 65% of its float-adjusted market capitalization and is a true reflection of the Indian stock market.

The NIFTY 50 covers major sectors of the Indian economy and offers investment managers exposure to the Indian market in one efficient portfolio. The Index has been trading since April 1996 and is well suited for benchmarking, index funds and index-based derivatives.

The NIFTY 50 is owned and managed by NSE Indices Limited (formerly known as India Index Services & Products Limited-IISL), India's first specialized company focused on an index as a core product.

Certificate

M-67, New Civil Line
Near Gurdev Talkies
Kanpur-208024
TEL: +91 9884480211
EMAIL: hello@civicloop.com

DATAMATIC
Data Distribution APIs

SUB: INTERNSHIP COMPLETION LETTER

We are glad to inform you that **Mr. Abhijeet Katiyar** from **GLA University, Mathura,** has successfully completed his internship at Datamatic IT Solutions from **1st June, 2018 - 30th July, 2018.**

During his internship, he was exposed to the various activities in **Time series Analysis.**

We found him extremely inquisitive and hardworking. He was very much interested to learn the functions of our core division and also willing to put his best efforts and get into the depth of the subject to understand it better.

His association with us was very fruitful and we wish him all the best in his future endeavors.

For Datamatic IT Solutions Date: 05-08-2018

Authorized Signatory

Founder – DATAMATIC IT Solutions

GSTIN: 09BVPK9003G1ZH

Acknowledgement

I am extremely grateful of Mr. Smrati Katiyar for his revered guidance and encouragement, which led to the completion of project under the summer training. Without his constant appraisal and efforts, this task would have been a mere dream. He was always there to help me throughout this training and project that I did under the training. He provided me with all the necessary recourses and guidance during the project which helped me to complete the projects successfully. Finally, I deem it a great pleasure to thank one and all, which helped me directly or indirectly in carrying out these project works and training. I am also thankful to our teachers and friends for their support. On a personal note, I owe everything to the almighty God and our parents. I would also like to thank our parents & other family members whose constant support & motivation helped us to complete the project successfully.

Table of content

Introduction

1.1 Overview: Data Mining is an approach where analytically designed to discover data (market related or business oriented) and in search of reliable patterns and/or logical relationships between
variables. Appropriate validation is used to find the patterns with the detected patterns among the data set. It is also called as data discovery or knowledge discovery. Data mining enhance the revenue and reduce the cost incurred for the exploration of data.

The general research associated with stock market is highly focusing on neither buy nor sell. But it fails to address the dimensionality and expectancy of a naive investor. The general trend towards stock market among the society is that it is highly risky for investment or not suitable for trade. The seasonal variance and steady flow of any index will help both for existing and naïve investor to understand and make a decision to invest in the stock market.

1.2 Objective: To solve these types of problems, the time series analysis will be the best tool for forecast and also to predict the trend. The trend chart will provide adequate guideline for the investor. Some time it may not address or forecast the variations or steady flow of the market. It may forecast only on particular season, but it is not adequate for long term decision making. The investors are very much interested to know the past trend or flow, seasonal growth, or variations of the stock. A general view or expectation is that it must give a holistic view of the stock market.

In this project I focused on real world problem in the stock market. The seasonal trend and flow is the highlight of the stock market. Eventually investors as well the stock broking company will also observe and capture the variations, constant

growth of the index. This will aid new investor as well as existing people will make a strategic decision. It can be achieved by experience and the constant observations by the investors. In order to overcome the above said issues, I have used ARIMA algorithm in three steps,

Step 1: Model identification

Step 2: Model estimation

Step 3: Forecasting

Company's profile

2.1 About company: About Us:

DataMatic IT Solutions was founded in 2018. We provide information technology related solutions to Institutional client in India and Abroad. Our primary customers are startups who require more cost-effective solutions to their IT needs.

We also provide opportunities to upcoming generation of software developers from different academic domains whether it is Engineering, Science, Commerce or Art. Our approach while training young software developers is that we help them acquire new skills while working on IT problems relevant to our clients and we closely monitor their progress while working on these problems and assist them in case they need expert help during the process.

2.2 Company's vision: At DataMatic our mission is to give cost effective solutions to our clients and at the same time develop a new generation of software developers who are industry ready from the very first day they complete their college.

Project Implementation

3.1 Different Types of Dataset:

1. Cross-sectional data: Cross sectional data can obtain when there is multiple observations is taken from multiple individuals at same point time. In case of cross-sectional data Time does not play any important role in analysis. Analysis of cross-sectional data starts with visualization of basic statistical properties i.e. central tendency, dispersion, skewness, kurtosis.

2. Time series data: Timeseries data can obtained when there is multiple observations is taken form same source at different points of time. Time series data can be described by trend, seasonality, stationarity, autocorrelation, and so on.

3. Panel data: Panel data is collection of multiple entities over multiple points in time. Panel data also known as longitudinal data.

3.2 Introduction to dataset: We have Nifty50 data from Apr-2010 to Mar-2018
https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm

First we need to explore the data to check the presence of null values

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1987 entries, 2010-04-01 to 2018-03-28
Data columns (total 6 columns):
Open                1987 non-null float64
High                1987 non-null float64
Low                 1987 non-null float64
Close               1987 non-null float64
Shares Traded       1987 non-null int64
Turnover (Rs. Cr)   1987 non-null float64
dtypes: float64(5), int64(1)
memory usage: 108.7 KB
```

Figure 3.2.1: Description of NULL values

This is time series data with the frequency of business days.

Let’s understand the arguments one by one:

parse_dates: This specifies the column which contains the date-time information. As we say above, the column name is 'date'.

index_col: This argument tells pandas to use the ‘date’ column as index.

In this dataset we have DateTime series as index and 6 columns, 1987 entries for each. 5 of them are float and one is integer type also we don't have any Null values.

3.3 Internal structure of Timeseries: A time series is combination of trend, seasonal, cyclical, and irregular components.

General trend: A general trend can be identified by its upward or downward movement in a long run. General trend can be seen with plotting the data, here we are using matplotlib library to plot Close prices.

```
import matplotlib.pyplot as plt

plt.plot(Nifty_data.Close)

plt.title('Nifty50 data for Close prices')

plt.show()
```

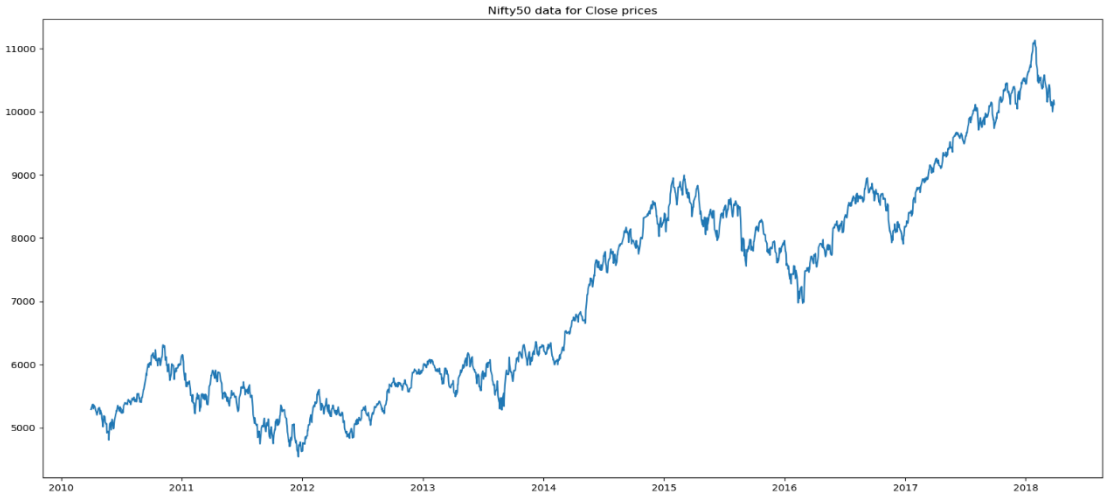


Figure 3.3.1: Trend in NIFTY50 data

The graph is showing movement in upward direction, this is clear sign of presence of trend component.

General trend might not capable of being noticed during short run of time series.

Seasonality: If in time series data, there are patterns that repeat over known periods of time. For example, the consumption of ice-cream during summer is more than winter and hence an ice-cream dealer's sales would be higher in summer months. Mostly, presence of seasonality can be reveals by exploratory data analysis.

Cyclical movements: If there are movements observe after every few units of time, but they are not as frequent as seasonal components, are known as Cyclic components. cyclic components do not have fixed periods of variations.

Unexpected variations: These are sudden changes occurring in a time series which are unlikely to be repeated. They are components of a time series which cannot be explained by trends, seasonal or cyclic movements. These variations are sometimes called residual or random components. These variations, though accidental in nature, can cause a continual change in the trends, seasonal and cyclical oscillations during the forthcoming period.

The objective of time series analysis is to decompose timeseries into it's elements and develop mathematical models to predict future values.

3.4 Stationary time series: When a time series free from general trend and seasonality, it become stationary. Statistical properties like mean, variance, autocorrelation etc. of a stationary time series remains constant over time. It is important to gain stationarity before forecasting because most statistical forecasting methods are applicable on stationary time series.

we can check stationarity using the following:

Plotting rolling statistics: we can plot moving mean and moving variance and see if these terms are varying with time. let's plot the moving average for business week

days, business month days, Quarterly and yearly. we have rolling() method in pyhon to calculate rolling statistics.

- Weekly

```
plt.plot(Nifty_data.Close,label='Close Prices',color='green')
plt.plot(Nifty_data['Close'].rolling(window=5).mean(),label='moving avg',color='orange')
plt.legend()
plt.title('Weekly rolling statistics')
plt.show()
```

Figure 3.4.1: Weekly moving average of time series

- Monthly

```
plt.plot(Nifty_data.Close,label='Close Prices',color='green')
plt.plot(Nifty_data['Close'].rolling(window=21).mean(),label='moving avg',color='orange')
plt.legend()
plt.title('Monthly rolling statistics')
plt.show()
```

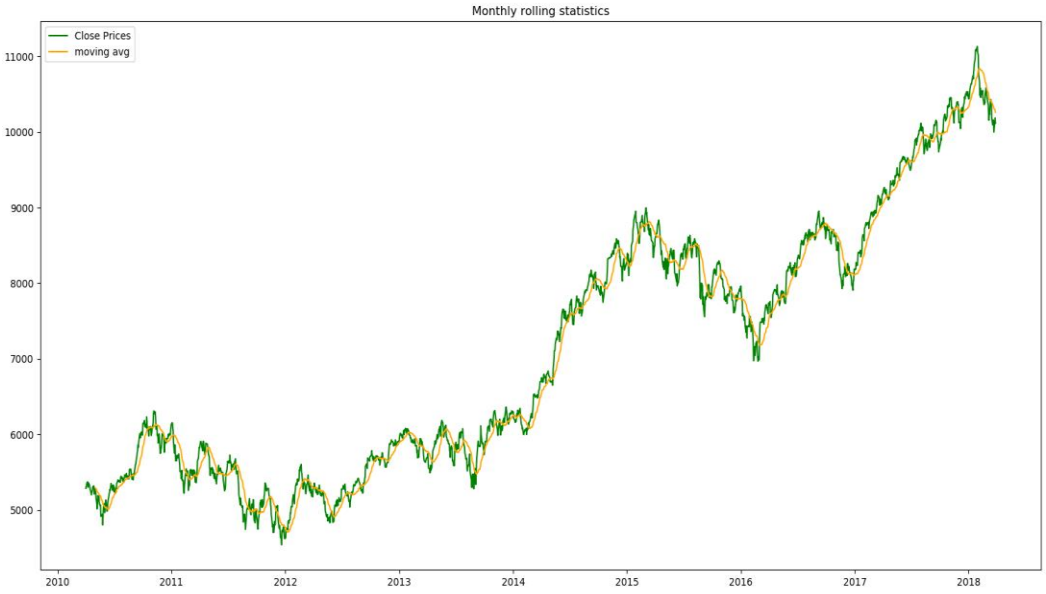


Figure 3.4.2: Monthly moving average of time series

- Quarterly

```
plt.plot(Nifty_data.Close,label='Close Prices',color='green')
plt.plot(Nifty_data['Close'].rolling(window=63).mean(),label='moving avg',color='orange')
plt.legend()
plt.title('Quarterly rolling statistics')
plt.show()
```

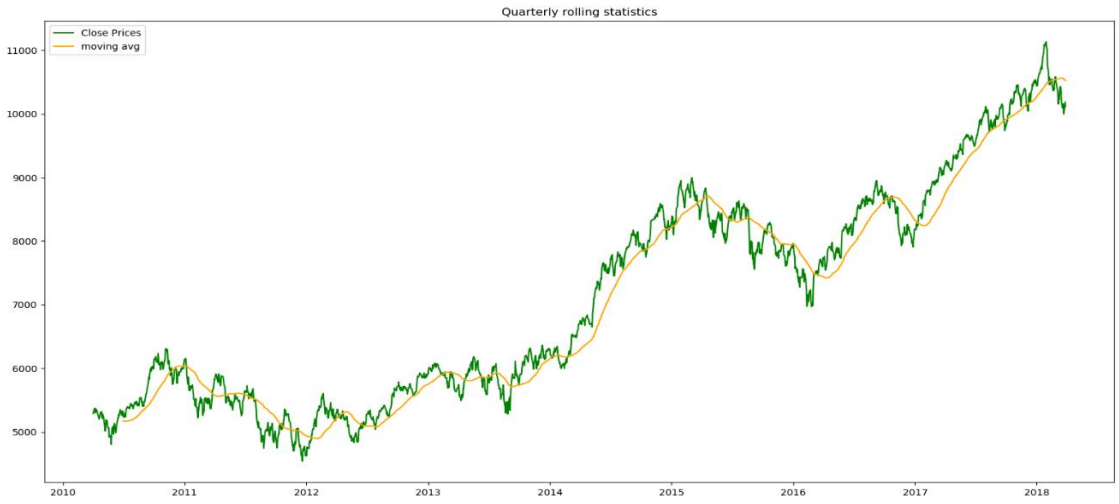


Figure 3.4.3: quarterly moving average of time series

- Yearly

```
plt.plot(Nifty_data.Close,label='Close Prices',color='green')

plt.plot(Nifty_data['Close'].rolling(window=252).mean(),label='moving avg',color='orange')

plt.legend()

plt.title('Yearly rolling statistics')

plt.show()
```

Figure 3.4.4: yearly moving average of time series

As we can see moving average is changing with time so, Time series is not stationary.

Augmented Dickey Fuller test: This is one of the unit root tests for checking stationarity. In this this test we check for null hypothiesis, where H_0 stats that time series is stationary whereas H_A stats that time series is not Stationary. In python we have stattools.adfuller function to check the stationarity of time series.

```
from statsmodels.tsa.stattools import adfuller

stnry_test=adfuller(Nifty_data['Close'],autolag='AIC')

stnry_rslt = pd.Series(stnry_test[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in stnry_test[4].items():

 stnry_rslt['Critical Value (%s)'%key] = value

print(stnry_rslt)

if(stnry_test[1]>0.05):

 print("Time Series is not stationary")

else:

 print("Time series is stationary")
```

```
Test Statistic                  -0.371803
p-value                          0.914701
#Lags Used                       1.000000
Number of Observations Used    1985.000000
Critical Value (1%)             -3.433649
Critical Value (5%)             -2.862997
Critical Value (10%)            -2.567546
dtype: float64
Time Series is not stationary
```

Figure 3.4.5: ADFuller test on original data

Note - if p-value is greater than 0.05 reject null hypothesis.

Here, p-value is 0.914701 which is greater than 0.05. Hence it is confirmed that our time series is not stationary.

To get stationary time series we need to remove trend and seasonality.

3.5 Methods to detrending data:

A time series can be detrended using following methods -

- Differencing
- Regression
- using functions

Method to detrending timeseries using Differencing: Differencing is the process of taking difference between successive occurrence of time series $\Delta x_t = x_t - x_{t-1}$.

Where, Δx_t is stationary time series.

x_t is original time series.

x_{t-1} is time series with lag 1.

In python we have .shift() method to create a series with lag.

```
diff=Nifty_data['Close']-Nifty_data['Close'].shift(1)
```

There will be null values because of lag. It is important to remove null values otherwise adfuller test function will show an error i.e. "SVD did not converge"

```
diff.dropna(inplace=True)

plotting rolling statistics

plt.plot(diff,label='differenced timeseries',color='grey')

plt.plot(diff.rolling(window=252).mean(),label='Moving average',color='red')

plt.title('Yearly rolling statistics on diffrenced time series')

plt.axhline(y=0,color='green')

plt.legend()

plt.show()
```

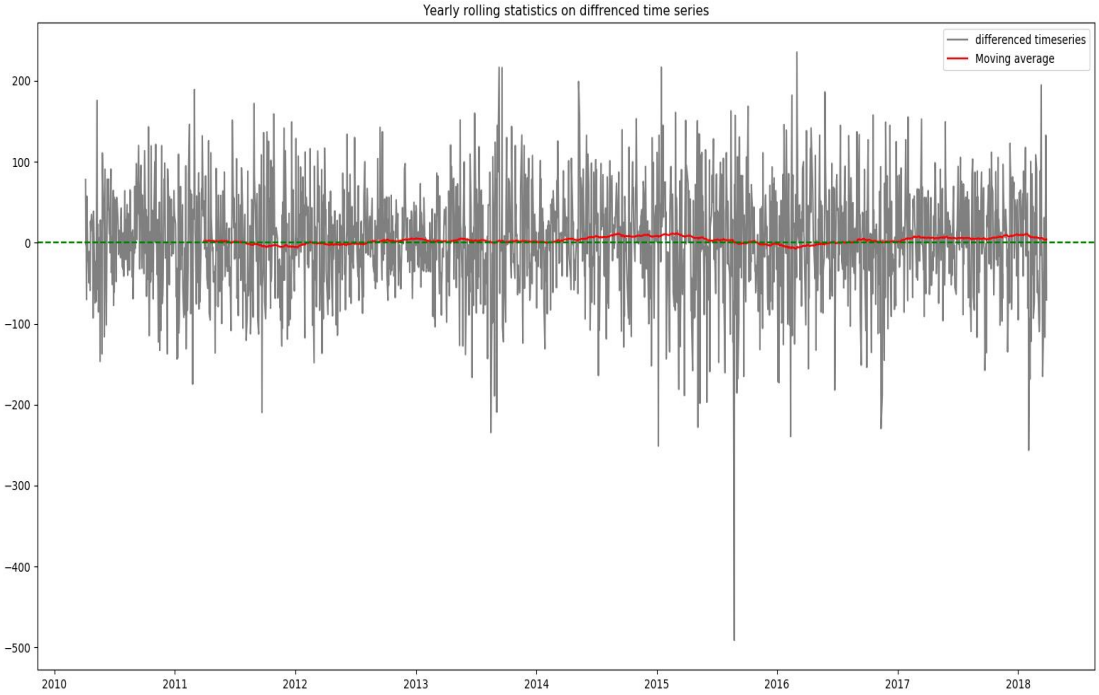


Figure 3.5.1: Yearly moving average on differenced data by lag 1

Now, the moving average is constant.

We can confirm the stationarity by apply Augmented DickeyFullerTest on `diff` time series –

```
stnry_test_diff=adfuller(diff,autolag='AIC')

stnry_rslt_diff = pd.Series(stnry_test_diff[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in stnry_test[4].items():

   stnry_rslt_diff['Critical Value (%s)'%key] = value

print(stnry_rslt_diff)

if(stnry_test_diff[1]>0.05):

   print("Time Series is not stationary")

else:

   print("Time series is stationary")
```

```
Test Statistic                  -41.047846
p-value                           0.000000
#Lags Used                        0.000000
Number of Observations Used    1985.000000
Critical Value (1%)              -3.433649
Critical Value (5%)              -2.862997
Critical Value (10%)             -2.567546
dtype: float64
Time series is stationary
```

Figure 3.5.2: ADFuller test on differenced data

p-value is 0.000000 which is less than 0.05.

So, the time series is now stationary.

The method of differencing discussed above is first order differencing. After applying first order differencing It is possible that the time series may not become stationary then we have to perform second order differencing. To perform second order differencing, take difference another time.

$$x'_t - x'_{t-1} = (x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} - x_{t-2}$$

This was the first method of making a time series stationary.

Method to detrending timeseries using Regression: Regression is another way to detrending timeseries data. Objective of choosing regression over differencing is to get trend line. Trend line can later be used as prediction of long run movement of time series.

Regression analysis is a form of predictive modeling technique which looks into the relationship between a target and predictor variables. Regression analysis is used for forecasting, time series modeling.

Here we fit a trend line to the training data, in such a manner that the distance between data points and trend line is minimum.

let's perform linear regression, but first we have to separate training data and testing data.

```
# separating training and testing data

train = Nifty_data['Close'].iloc[:1750]

test = Nifty_data['Close'].iloc[1751:]

# building linear model

import numpy as np

from sklearn import linear_model

lm = linear_model.LinearRegression(normalize=True, fit_intercept=True)
```

Details of parameters:

Normalize: This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression. By default, this is set to False.

fit_intercept: whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations. By default, this is set to True.

For more information about parameters you can see LinearRegression documentation.

Now we have to fit linear_model to Nifty_data timeseries

```
lm.fit(np.arange(np.array(len(train.index))).reshape((-1,1)), train)
```

Now lets check the accuracy of fitted model

```
lm.score(np.arange(np.array(len(train.index))).reshape((-1,1)), train)
```

```
In [110]: lm.score(np.arange(np.array(len(train.index))).reshape((-1,1)), train)
Out[110]: 0.7783615453922795
```

Now, we are going to plot close prices with trend line

```
plt.plot(Nifty_data['Close'],label='Original data')

plt.plot(pd.Series(lm.predict(np.arange(np.array(len(Nifty_data.index))).reshape((
-1,1))),index=Nifty_data.index),label='Trend line')

plt.title('Close prices with trend line')

plt.legend()

plt.show()
```

Output will be –

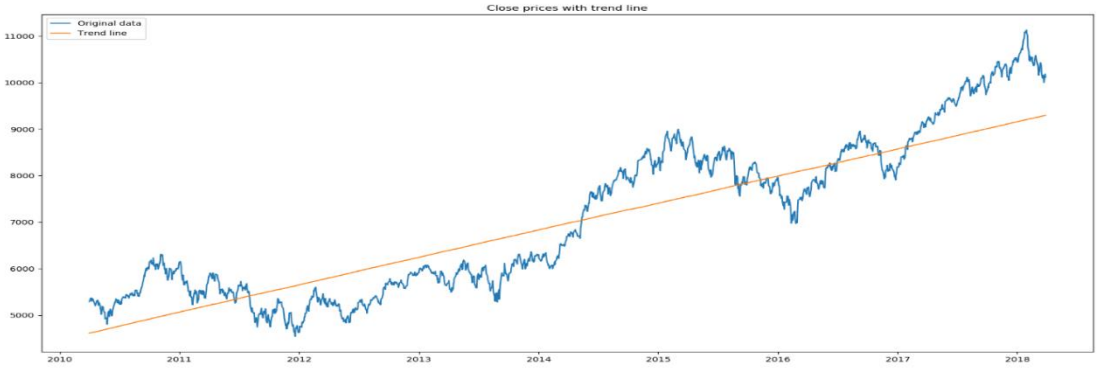


Figure 3.5.3: trend line in data using regression

Now we are going to calculate residuals which will be used later in forecasting

```
Residuals=Nifty_data['Close']-
lm.predict(np.arange(np.array(len(Nifty_data.index))).reshape((-1,1)))

plt.plot(pd.Series(Residuals,index=Nifty_data.index),label=Residuals)

plt.title('Residuals for Close prices')

plt.show()
```

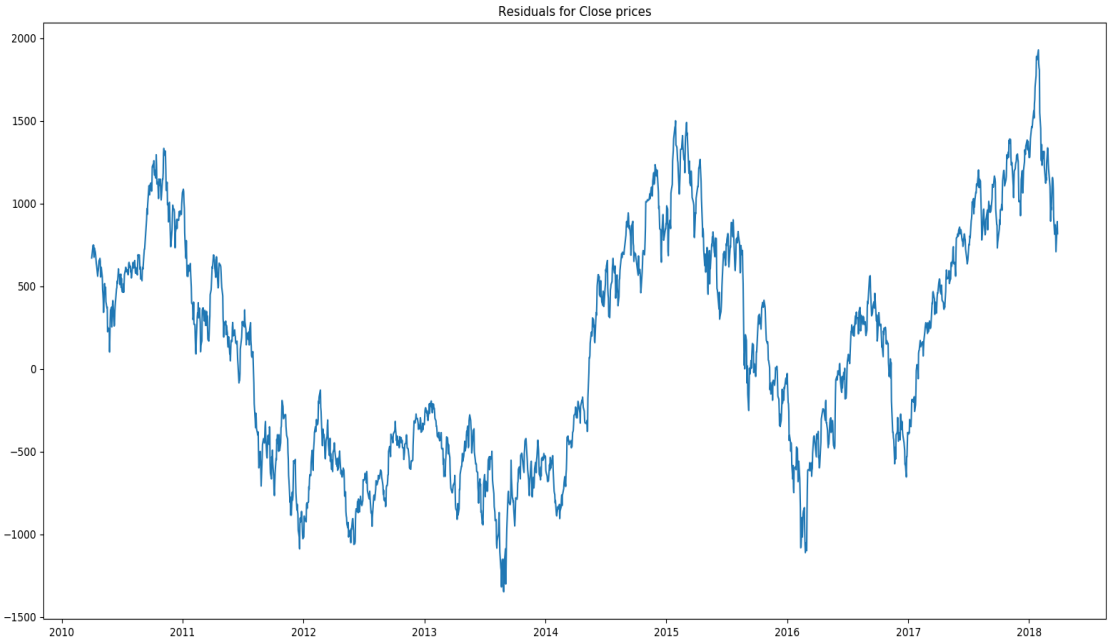


Figure 3.5.4: Residuals of data after removing trend

Method to detrending timeseries using Functions: There are many libraries in python which are specially made to perform Statistical operations, one of them is statsmodels.

```
from statsmodels.tsa import seasonal

decompose = seasonal.seasonal_decompose(Nifty_data['Close'],freq=252)
```

seasonal_decompose() method returns,

- Observed data (i.e. Close prices from Nifty_data dataset)
- Trend components
- Seasonal components
- Residuals

We can access Trend, seasonal components and Residuals by

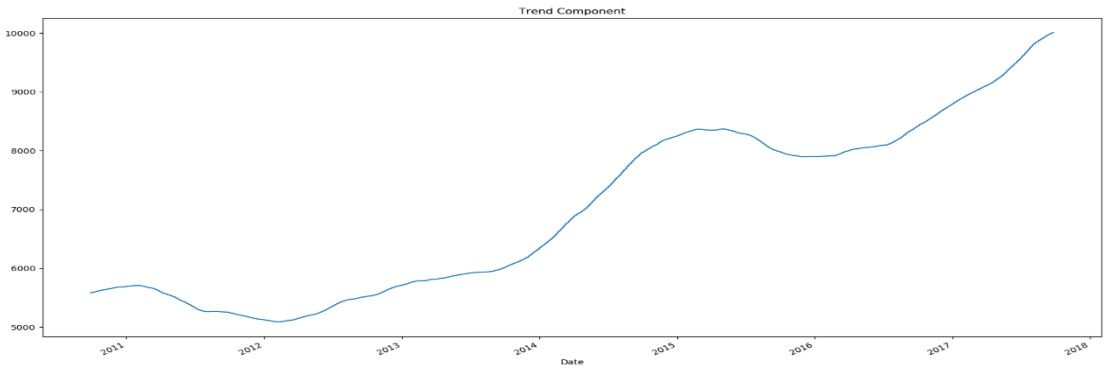
decompose.trend, decompose.seasonal, decompose.resid respectively.

Note: Because we use 252 as frequency so we will have 252 null values in trend component and in residuals

```
decompose.trend.dropna(inplace=True)

decompose.resid.dropna(inplace=True)
```

Now, we are going to plot trend component.

```
decompose.trend.plot()

plt.title('Trend Component')

plt.show()
```



We detrended our time series now it's time to do work on seasonality.

3.6 Estimating seasonality: while removing trend by using differencing, seasonality was also removed so if we perform differencing method trend and seasonality both will removed while applying seasonal_decompose method we already get the seasonal component, so there is no need to perform any other operation to remove seasonality if we follow differencing or use statsmodels package. If we use regression then we don't get anything related to seasonality. So we have to perform exploratory data analysis through following plots:

- Run sequence plot
- Seasonal subseries plot
- Multiple box plot

To remove Seasonality, simply take average of detrended data for specific season.

In these plots we will use Residuals –

```
Residuals=Nifty_data['Close']-
lm.predict(np.arange(np.array(len(Nifty_data.index))).reshape((-1,1)))
```

Run sequence plot: A simple time series plot wit time on x-axis itself reveals following properties:

1. Movement in mean of series
2. Shifts in variance
3. presence of outliers

let's visualize:

- Monthly

```
plt.plot(Residuals.rolling(window=21).mean(),label='residual mean')

plt.plot(Residuals.rolling(window=21).std(),label='residual std')

plt.plot(Residuals,label='Residuals')

plt.xlabel('year')

plt.ylabel('Residuals')

plt.title('Residuals,Monthly mean and Monthly Std ')

plt.legend()

plt.show()
```

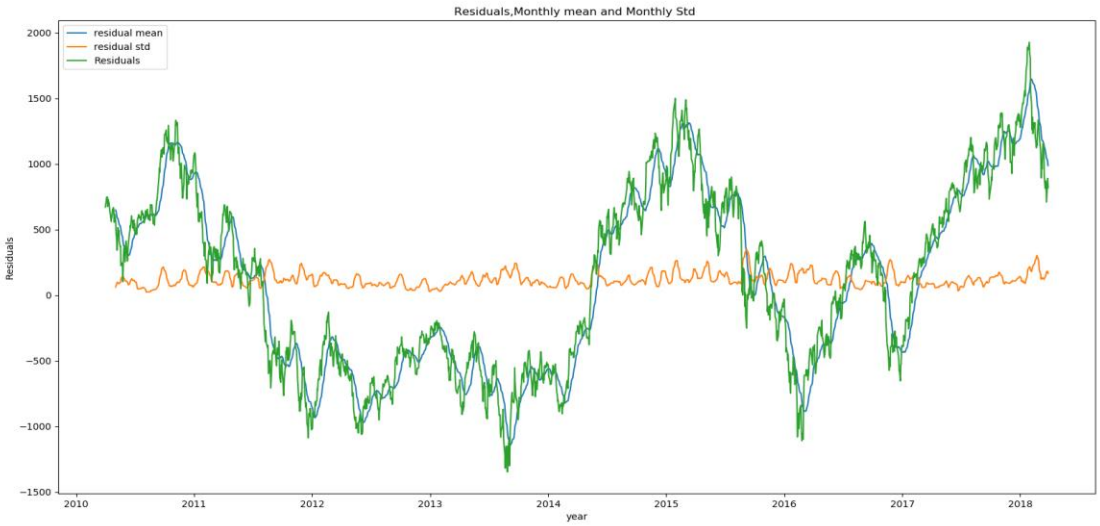


Figure 3.6.1: Monthly mean and standard deviation of residuals

- Quarterly

```
plt.plot(Residuals.rolling(window=63).mean(),label='residual mean')

plt.plot(Residuals.rolling(window=63).std(),label='residual std')

plt.plot(Residuals,label='Residuals')

plt.xlabel('year')

plt.ylabel('Residuals')

plt.title('Residuals,Quarterly mean and Quarterly Std ')

plt.legend()

plt.show()
```

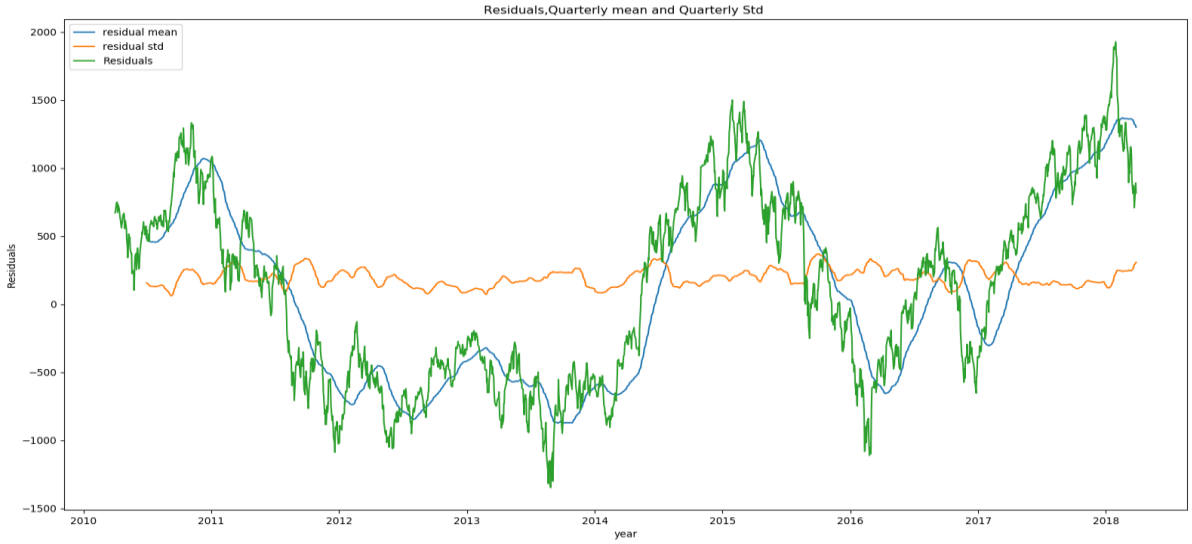


Figure 3.6.2: Quarterly mean and standard deviation of residuals

- Yearly.

```
plt.plot(Residuals.rolling(window=252).mean(),label='residual mean')

plt.plot(Residuals.rolling(window=252).std(),label='residual std')

plt.plot(Residuals,label='Residuals')

plt.xlabel('year')

plt.ylabel('Residuals')

plt.title('Residuals,Yearly mean and Yearly Std ')

plt.legend()

plt.show()
```

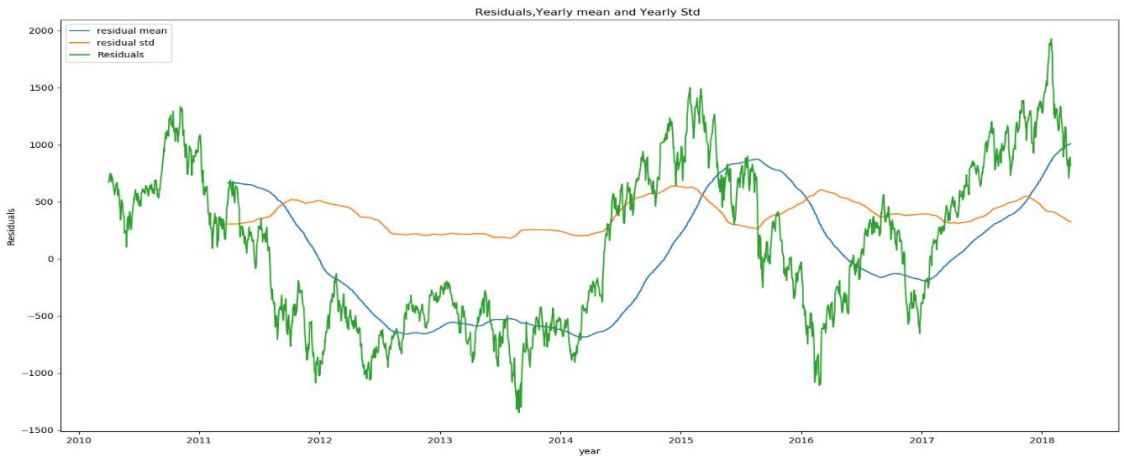


Figure 3.6.3: Yearly mean and standard deviation of residuals

Seasonal sub series plot: We can identify seasonality by groping time periods like month, quarter or year. this method is only useful when time period of seasonality is known.

before plotting lets add columns for Residuals, months, quarter and years.

- Adding column "Residuals"

```
Nifty_data['Residuals']=Residuals
```

- Adding column "month"

```
Nifty_data['month']=Nifty_data.index.map(lambda x:x.month)
```

- Adding column "year"

```
Nifty_data['year']=Nifty_data.index.map(lambda x:x.year)
```

- Adding column "quarter"

```
month_quarter_map = {1: 'Q1', 2: 'Q1', 3: 'Q1',
                     4: 'Q2', 5: 'Q2', 6: 'Q2',
                     7: 'Q3', 8: 'Q3', 9: 'Q3',
                     10: 'Q4', 11: 'Q4', 12: 'Q4'
                    }
Nifty_data['quarter'] = Nifty_data['month'].map(lambda m: month_quarter_map.get(m))
```

Now we are going to calculate mean and standard deviation for residuals and making a new subseries. we already calculated residuals while detrending data using regression.

- Quarterly

```
# Creating new subseries
sub_series_quarterly = Nifty_data.groupby(by=['year', 'quarter'])['Residuals'].aggregate([np.mean, np.std])
sub_series_quarterly.columns = ['Quarterly Mean Close', 'Quarterly Standard Deviation Close']
#Create row indices of seasonal_sub_series_data using Year & Quarter
sub_series_quarterly.reset_index(inplace=True)
sub_series_quarterly.index = sub_series_quarterly['year'].astype(str) + '-' + sub_series_quarterly['quarter']
```

Let’s have a look to our sub series:

```
sub_series_quarterly.head()
```

	year	quarter	Quarterly Mean Close	Quarterly Standard Deviation Close
2010-Q2	2010	Q2	485.421131	157.670891
2010-Q3	2010	Q3	696.596141	193.226523
2010-Q4	2010	Q4	1045.849666	155.593209
2011-Q1	2011	Q1	430.527603	234.236798
2011-Q2	2011	Q2	297.627121	219.394727

Figure 3.6.4: sample of subseries to detect seasonality

Now we are going to plot it –

```
sub_series_quarterly['Quarterly Mean Close'].plot(color='b')
plt.title('Quarterly Mean of Residuals')
plt.ylabel('Residuals')
plt.show()
sub_series_quarterly['Quarterly Mean Close'].plot()
plt.title('Quarterly standard Deviation of Residuals')
plt.show()
```

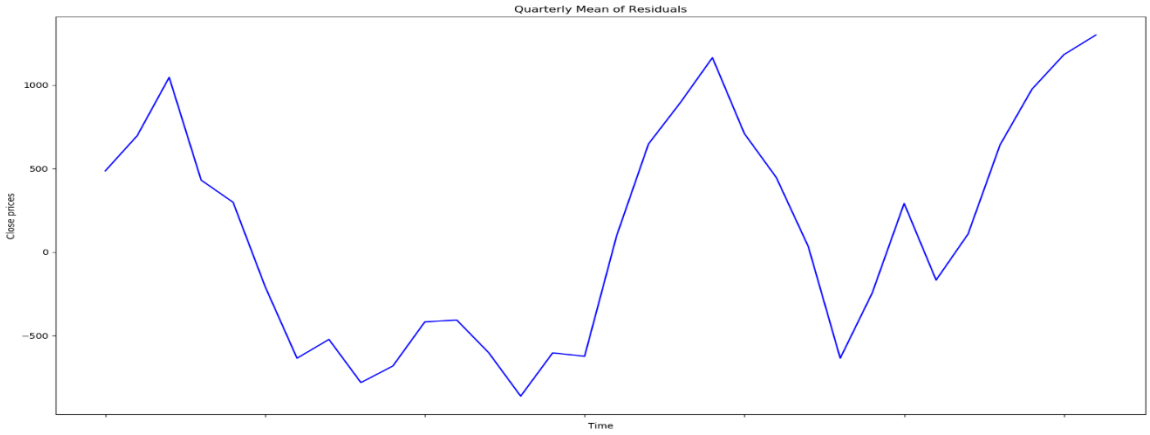


Figure 3.6.5: Quarterly Mean of Residuals

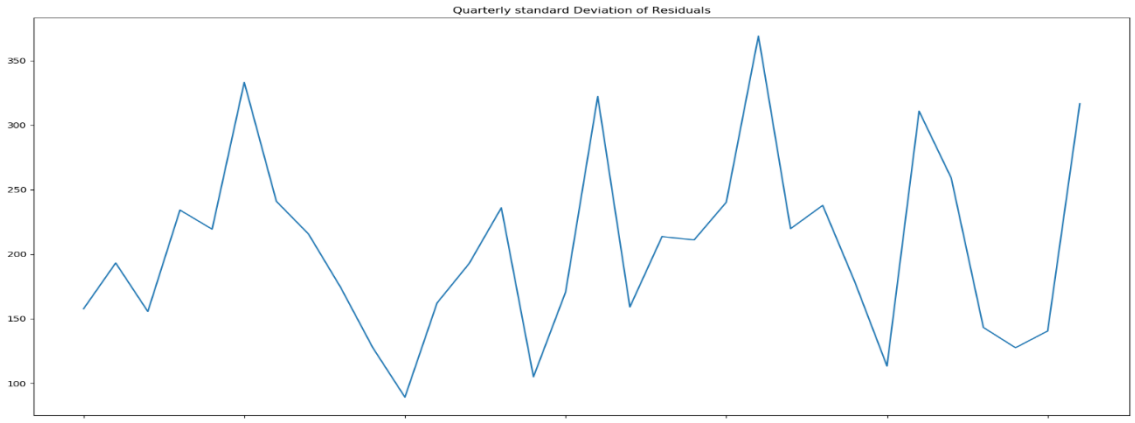


Figure 3.6.6: Quarterly standard Deviation of Residuals

- Monthly

```
# Creating new subseries
sub_series_Monthly=Nifty_data.groupby(by=['month'])['Residuals'].aggregate
([np.mean, np.std])
sub_series_Monthly.columns= ['monthly Mean', 'monthly Standard Deviation']
# plotting sub series
sub_series_Monthly['monthly Mean'].plot()
plt.title('Monthly mean of residuals')
plt.show()
sub_series_Monthly['monthly Standard Deviation'].plot()
plt.title('Monthly Standard Deviation of residuals')
plt.show()
```

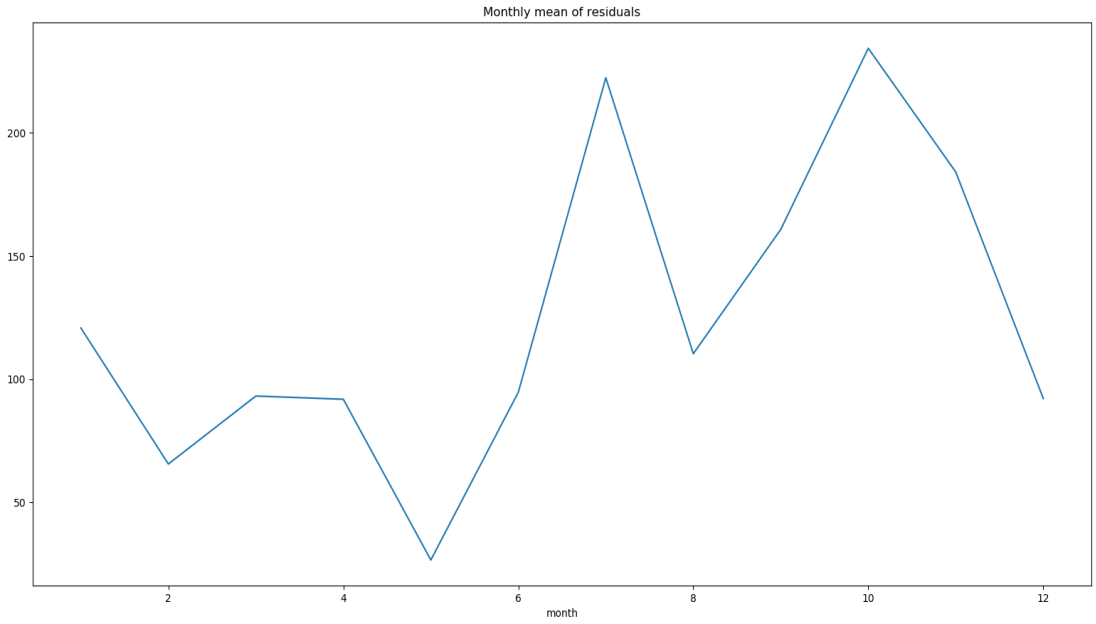


Figure 3.6.7: Monthly mean of residuals

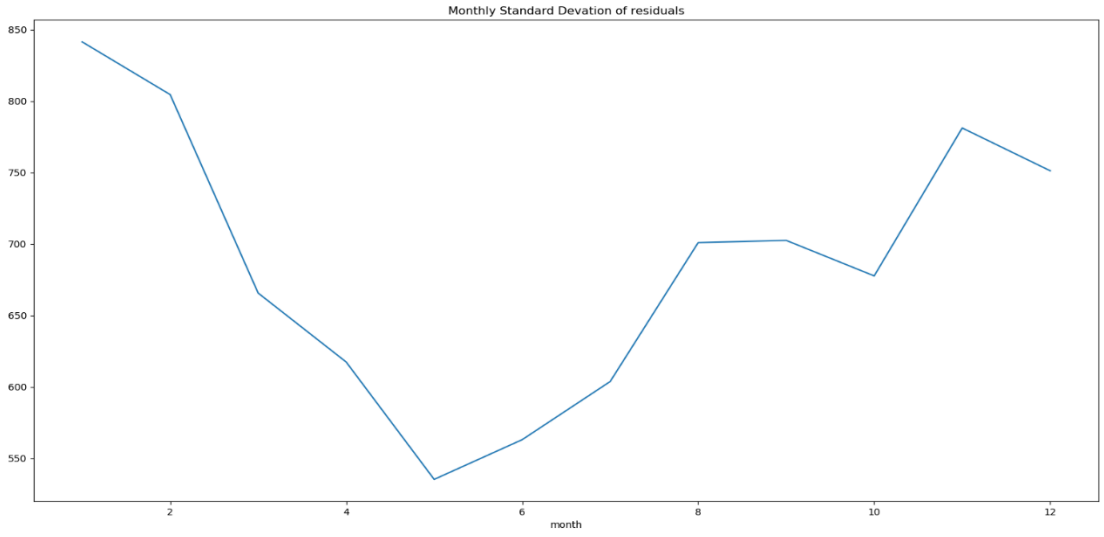


Figure 3.6.8: Monthly standard deviation of residuals

- Yearly

```
# Creating new subseries
sub_series_yearly =
Nifty_data.groupby(by=['year'])['Residuals'].aggregate([np.mean, np.std])
sub_series_yearly.columns = ['yearly Mean', 'yearly Standard Deviation']
# plotting sub series
sub_series_yearly['yearly Mean'].plot()
plt.title('yearly mean of residuals')
plt.show()
sub_series_yearly['yearly Standard Deviation'].plot()
plt.title('Yearly Standard Devation of residuals')
plt.show()
```

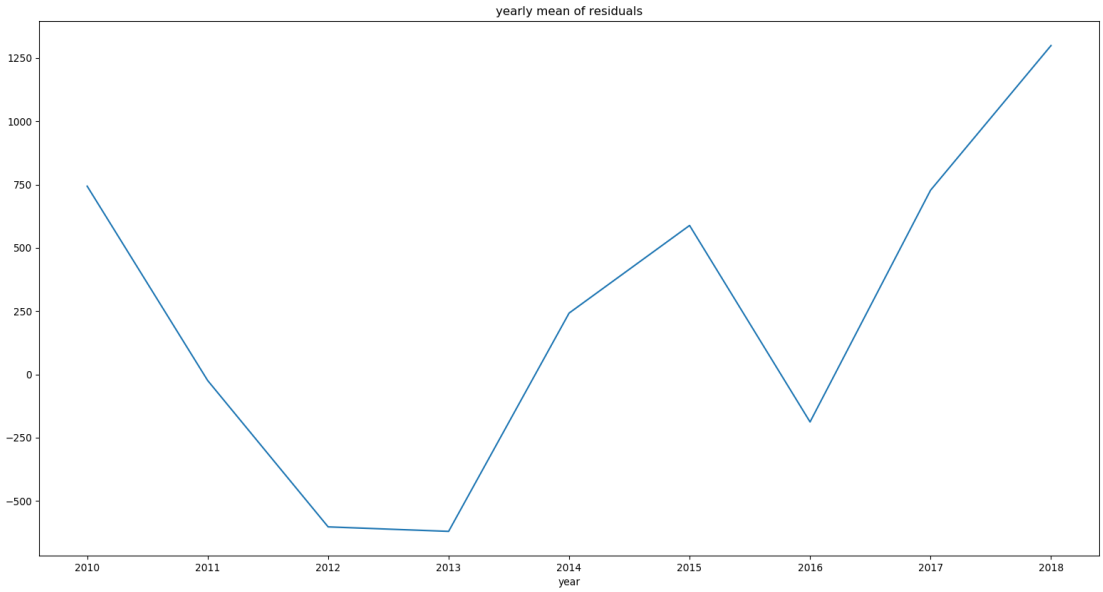


Figure 3.6.9: Yearly mean of residuals

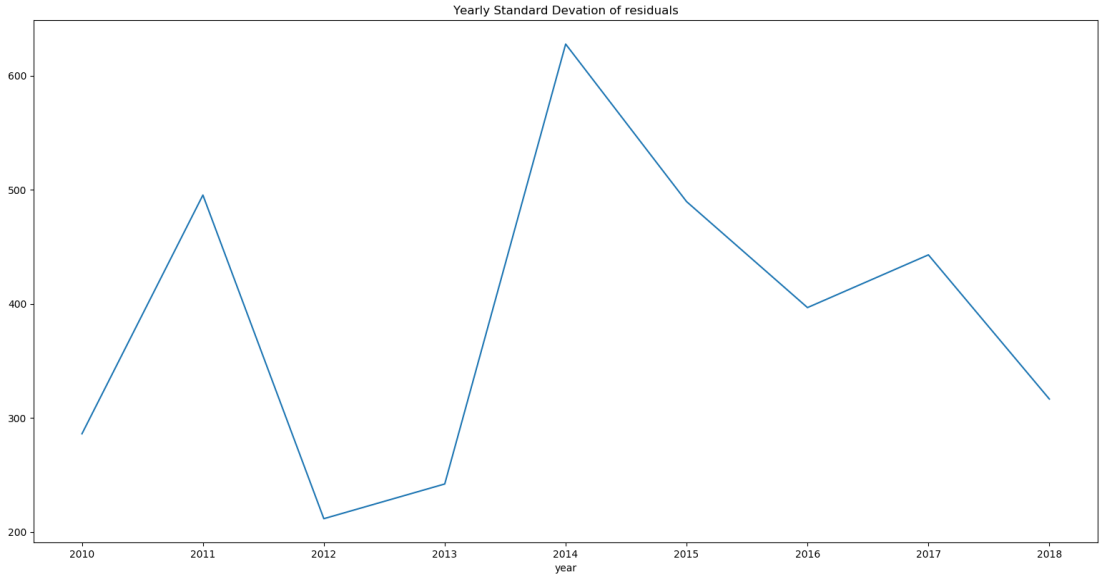


Figure 3.6.10: Yearly standard Deviation of Residuals

Multiple box plots: Multiple box plots are basically the boxplots of seasonal subseries plots. Box plots are more informative than any simple plot. Box plot revels information like mean, inter quartile range, minimum and maximum value and outliers, central tendency and dispersion.

For box plots we are going to use seaborn.

- Quarterly

```
# Multiple Boxplot(Quarterly)
import seaborn as sns
plt.figure(figsize=(5.5, 5.5))
g = sns.boxplot(data=Nifty_data[['Residuals','quarter']], y=Nifty_data['Residuals'],
x=Nifty_data['quarter'])
g.set_title('Quarterly box plot of Residuals')
g.set_xlabel('Time')
g.set_ylabel('Residuals')
plt.show()
```

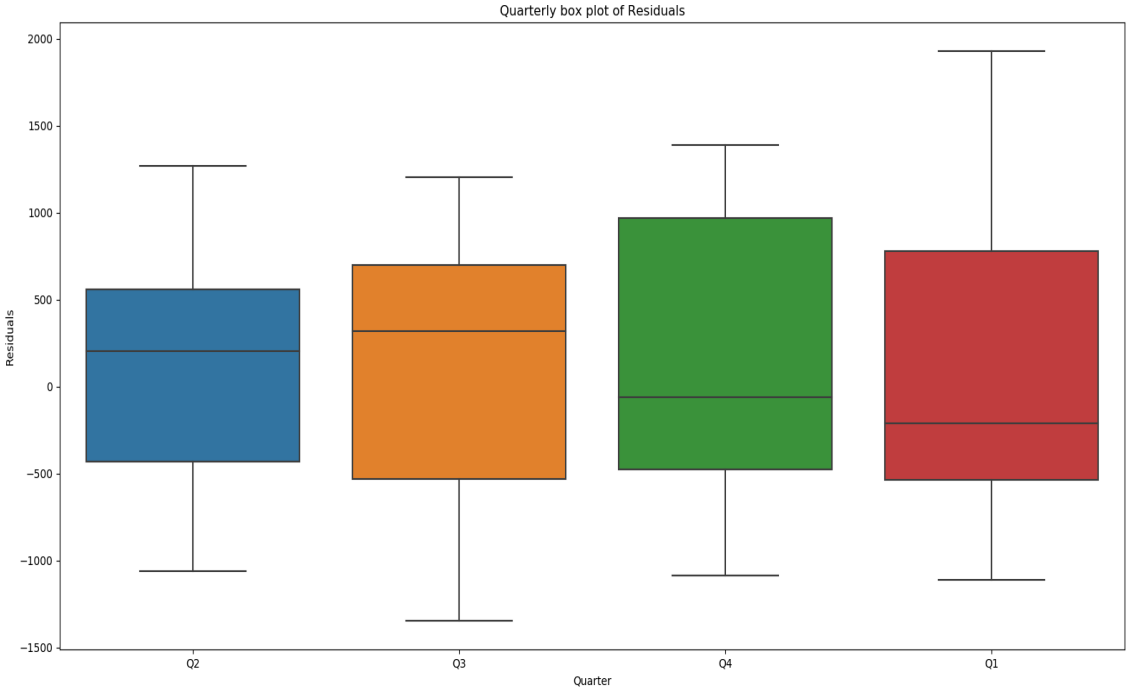


Figure 3.6.11: Quarterly box plots of residuals

- Monthly

```
# Multiple Boxplot(Monthly)
import seaborn as sns
plt.figure(figsize=(5.5, 5.5))
g = sns.boxplot(data=Nifty_data[['Residuals','month']], y=Nifty_data['Residuals'], x=Nifty_data['month'])
g.set_title('Monthly box plot of Residuals')
g.set_xlabel('Time')
g.set_ylabel('Residuals')
plt.show()
```

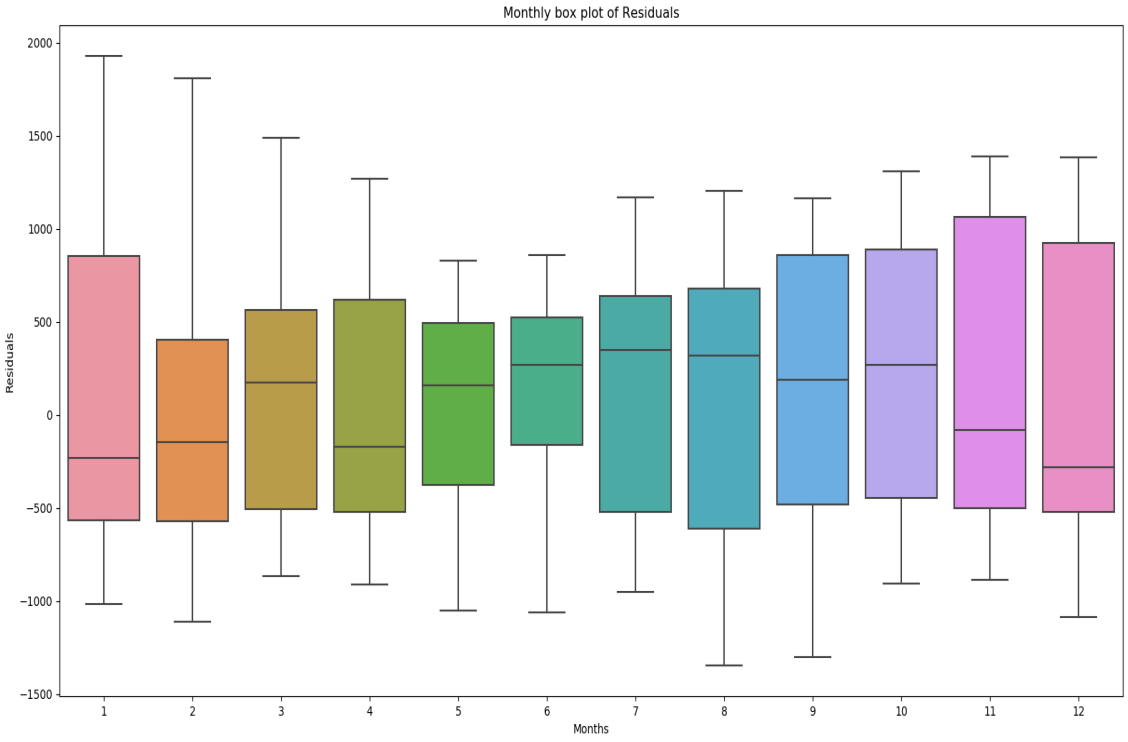


Figure 3.6.12: Monthly box plots of residuals

- Yearly

```
# Multiple Boxplot(Yearly)
import seaborn as sns
plt.figure(figsize=(5.5, 5.5))
g = sns.boxplot(data=Nifty_data[['Residuals','year']], y=Nifty_data['Residuals'], x=Nifty_data['year'])
g.set_title('Yearly box plot of Residuals')
g.set_xlabel('Time')
g.set_ylabel('Residuals')
plt.show()
```

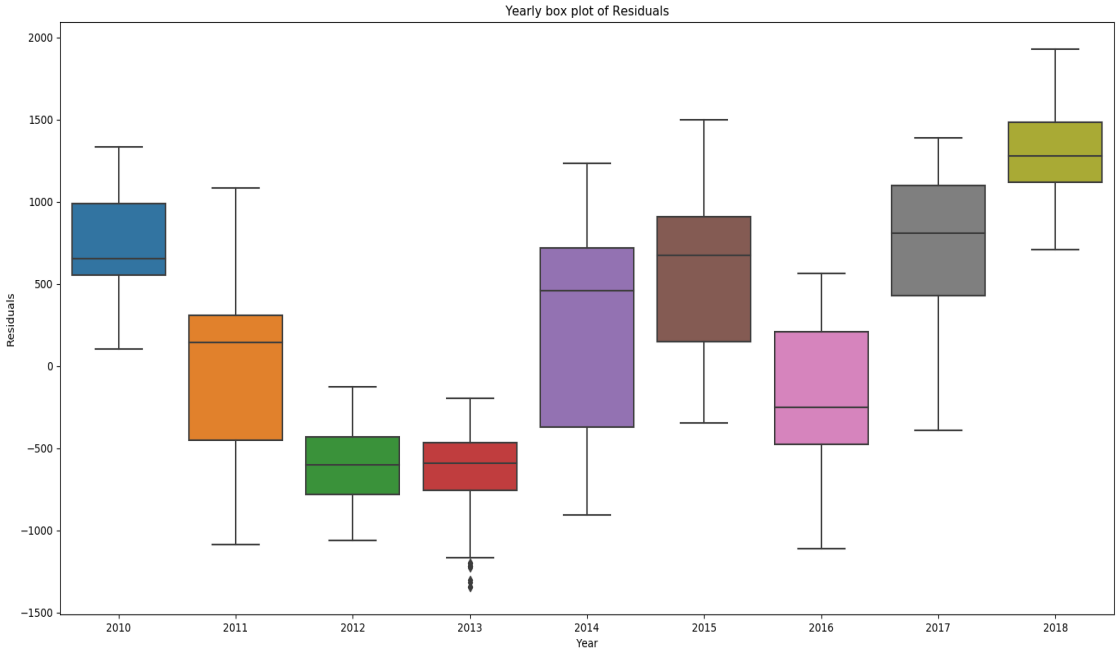


Figure 3.6.12: Yearly box plots of residuals

These are the methods to estimating seasonality.

3.7 Forecasting: For Time series forecasting ARIMA model is popular and widely used statistical method. ARIMA is an acronym of AutoRegressive Integrated Moving Average. It is generalization of AutoRegressive Moving Average and differencing.

AR (AutoRegression): This technique can be used on time series where output variable depends on its own previous values.

I (integrated): This is order of differencing to make time series stationary.

MA (Moving Average): A model that analyze data points by creating series of averages of subsets of data.

The ARIMA(p,d,q) represents the order of AR term, Differencing order and MA term respectively

To find the value of p and q we will plot autocorrelation and partial autocorrelation functions.

```
from statsmodels.tsa.stattools import acf, pacf
# ACF plot
lag_acf=acf(diff,nlags=20)
lag_pacf=pacf(diff,nlags=20,method='ols')
# plot ACF
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
plt.show()
# plot PACF
plt.plot(lag_pacf)
```

```
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.show()
```

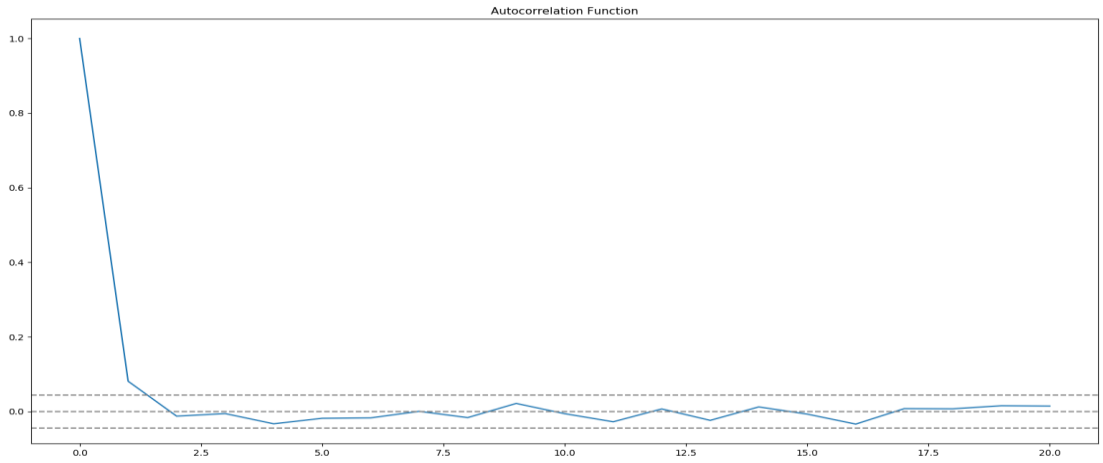


Figure 3.7.1: Auto correlation plot of differenced series

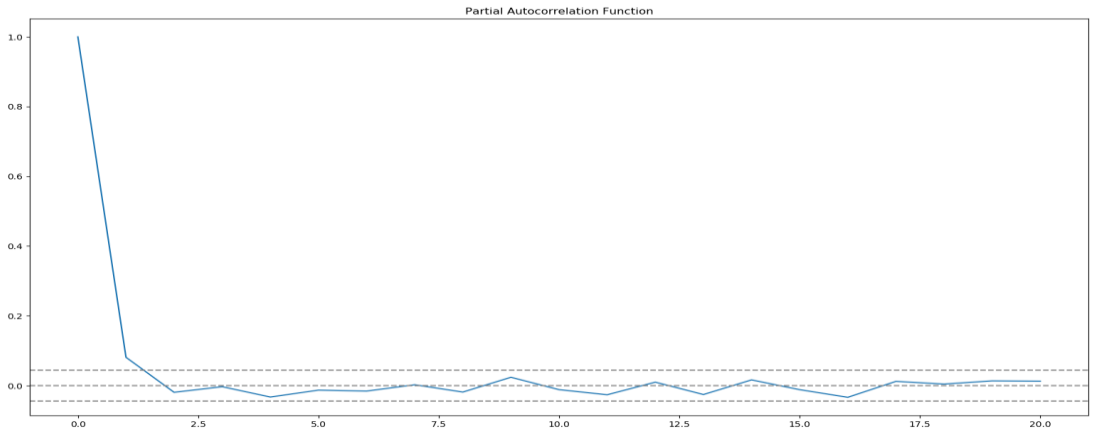


Figure 3.7.2: Partial auto correlation plot of differenced series

In these plots we draw the dotted line at y=0 and confidence intervals (dotted lines).

1. The value of p will be the lag value where the PACF chart crosses the upper confidence interval for the first time. In our plot it is 2.
2. The value of q will be the lag value where the ACF chart crosses the upper confidence interval for the first time. In our plot it is also 2.

Time series forecasting can be done in two ways:

1. We can calculate fitted values from model and take it back to original time series scale
2. Use .forecast() function on ARIMAresults, which performs one step forcast using the model.

Forecasting without using .forecast() function:

```
from statsmodels.tsa.arima_model import ARIMA
# seperating training and testing data
train=Nifty_data['Close'].iloc[:1750]
test=Nifty_data['Close'].iloc[1751:]
# Building ARIMA model
model = ARIMA(Nifty_data['Close'], order=(2, 1, 2))
results_ARIMA = model.fit()

# taking it back to original scale
predict_Arima=pd.Series(results_ARIMA.fittedvalues, copy=True)
predict_Arima.head()
```

```
Date
2010-04-05    2.429683
2010-04-06    8.567037
2010-04-07    0.673000
2010-04-08    3.941245
2010-04-09   -4.477808
dtype: float64
```

The first element of our original time series is lost because we took a lag by 1. to convert the differencing to log scale we will first determine the cumulative sum at index and then add it to the base number.

```
predict_Arima_cumsum=predict_Arima.cumsum()
predict_Arima_data=pd.Series(Nifty_data['Close'].iloc[0],
index=Nifty_data.index)
predict_Arima_data=predict_Arima_data.add(predict_Arima_cumsum,fill_value=
0)
predict_Arima_data.head()
```

```
Date
2010-04-01    5290.500000
2010-04-05    5292.929683
2010-04-06    5301.496720
2010-04-07    5302.169720
2010-04-08    5306.110965
dtype: float64
```

Now the series is on its original scale.

```
plt.plot(Nifty_data['Close'],label='Close prices')
plt.plot(predict_Arima_data,label='predicted values')
plt.legend()
plt.show()
```

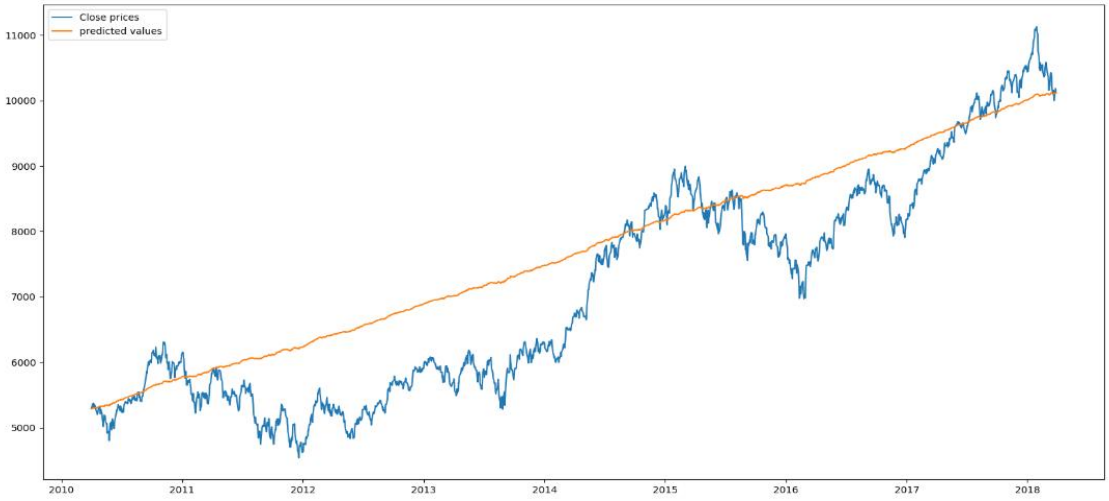


Figure 3.7.3: Predicted values vs original values

- Accuracy

```
import sklearn.metrics
sklearn.metrics.r2_score(Nifty_data['Close'],predict_Arima_data)
```

The accuracy is: 0.7398187397952602

Forecasting using forecast() function

```
training_set = [x for x in train]
predictions = []
for i in range(len(test)):
    model = ARIMA(training_set, order=(2,1,2))
    model_fit = model.fit()
    output = model_fit.forecast()
    predictions.append(output[0])
    obs = test[i]
    training_set.append(obs)
    print('predicted=%f, expected=%f' % (output[0], obs))
```

Sample of output is:

```
predicted=9141.223915, expected=9103.500000
predicted=9102.972637, expected=9136.400000
predicted=9141.926903, expected=9119.400000
predicted=9118.675974, expected=9217.950000
predicted=9228.694665, expected=9306.600000
predicted=9311.536421, expected=9351.850000
predicted=9357.132425, expected=9342.150000
predicted=9342.207983, expected=9304.050000
predicted=9304.528437, expected=9313.800000
predicted=9317.016738, expected=9311.950000
predicted=9313.615780, expected=9359.900000
predicted=9365.893142, expected=9285.300000
predicted=9280.411972, expected=9314.050000
predicted=9321.322902, expected=9316.850000
predicted=9316.589199, expected=9407.300000
predicted=9417.740810, expected=9422.400000
predicted=9422.073492, expected=9400.900000
predicted=9403.372917, expected=9445.400000
```

Plot of predicted values with test data:

```
pred=pd.Series(predictions,index=test.index)
plt.plot(test,label='Test data')
plt.plot(pred, color='red',label='Predicted values')
plt.title('Test data VS. Predicted values')
plt.legend()
plt.show()
```

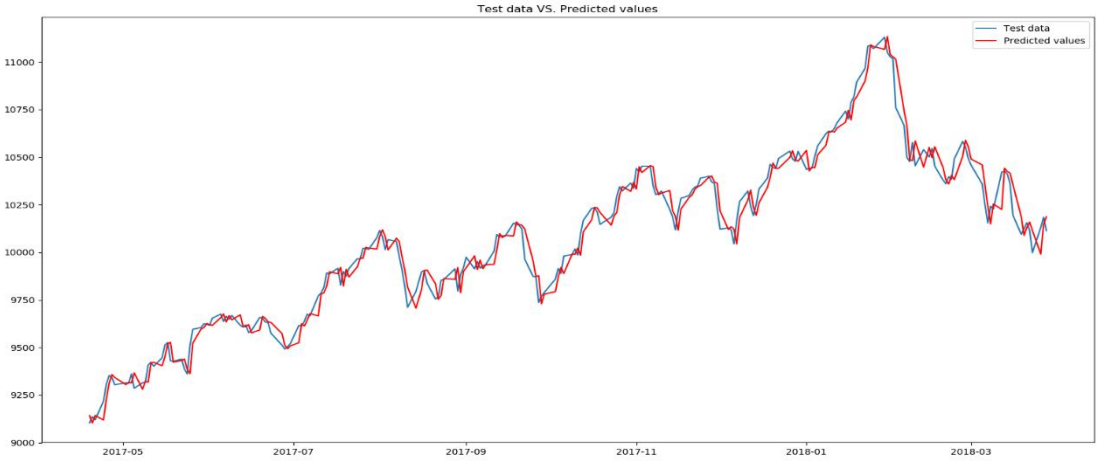


Figure 3.7.4: predicted values vs test values

- Accuracy

```
sklearn.metrics.r2_score(test,pred)
```

The accuracy is: 0.9777090172770078

Conclusion

It was a great opportunity that I got and worked such an interesting project. I felt very small hurdle in the project thanks to the supportive trainer guiding throughout training session. And I learnt a lot of new things and got confidence doing any project and entire training made me capable to tackle any hurdle as we programmer usually feel.