Data Science PROJECT  
 Client: NewX Services  
 Category: Forecasting Spare parts inventory  
 Project Ref: PM-PR-0027

**TEAM MEMBERS**:

1. **Ishan Pradip Borker**
2. **Kingston Teffy Roy**
3. **Ravichandra Reddy**
4. **Ayyappa Thalawar**
5. **Shilpa V.**

**BUSINESS CASE:**

The case business case is on the inventory management. Keeping  
Inventory of spare in various service centre to the market demand is  
always a challenge as most service centres spends significant amount  
in spare parts inventory costs. In spite of this, availability of spare  
parts is been one of the problem areas.

**PROJECT GOAL:**

Create Predictive model for inventory forecasting so that  
service centre achieve JIT standards.

**FEATURE DETAILS:**

Range Index: 28484 entries, 0 to 28483  
Data columns (total 7 columns):  
Invoice Date 28482 non-null datetime64[ns]  
Job Card Date 28482 non-null datetime64[ns]  
Business Partner Name 28482 non-null object  
Vehicle No. 28484 non-null object  
Vehicle Model 28482 non-null object  
Current KM Reading 28482 non-null float64  
INVOICE LINE TEXT 28449 non-null object

**Assumptions**

* Used two fields for the project. i.e. Job\_Card\_Date and INVOICE\_LINE\_TEXT.
* Created a new field named orders on demand from INVOICE\_LINE\_TEXT for using it in time series forecasting along with Job\_Card\_Date.
* Plotted graphs for finding the trends and getting the insights of the spare service parts dataset.
* Did not use Invoice\_Date for prediction.
* Used more data in train dataset than the test dataset.

**Approach**

1. Import all the necessary packages like

* **import pandas as pd**
* **import numpy as np**
* **from datetime import datetime**
* **import matplotlib.pyplot as plt**
* **from matplotlib import pyplot**
* **from matplotlib import rcParams**
* **%matplotlib inline**
* **from scipy import stats**
* **from collections import Counter**
* **import warnings**
* **warnings.filterwarnings('ignore')**

1. Define the function named parser.

It will parse the data in yyyy-mm-dd format.

**def** parser(x):

**return** datetime.strptime(x,'%d-%m-%y').date()

1. Load the dataset.

**data=pd.read\_excel("C:\\Users\DELL\Desktop\Rubixe projects\FEB2020\service-data.xlsx",header=0,date\_parser='Job\_Card\_Date')**

**data.head()**

1. Describe the data

**data.describe()**

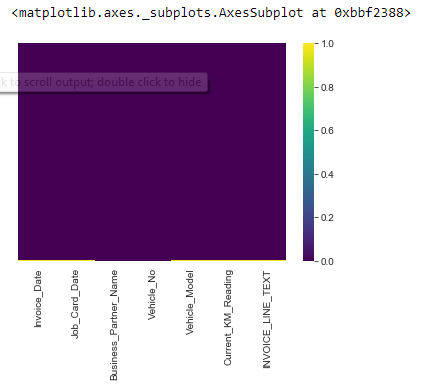
**data.info()**

1. Checking for the outliers

**import seaborn as sns**

**sns.set\_style('whitegrid')**

**sns.heatmap(data.isnull(),cbar=True,yticklabels=False,cmap='viridis')**



**Fig 1: heatmap representation**

1. Create Orders on demand column in the DataFrame

**data.loc[data.INVOICE\_LINE\_TEXT!=0,'Orders\_on\_Demand']=1**

**data.head(4)**

1. Find difference of days between job card date and invoice date.

**data['date\_difference']=data['Invoice\_Date']-data['Job\_Card\_Date']**

Use the counter to count the date difference between job card date and invoice date.

**Counter(data.date\_difference)**

To check for the day difference of 19 days use the following code:

**data[data.date\_difference =='19 days']**

1. Check for the EDA Steps

data.shape 🡪 For checking the shape of the data

Counter(data.INVOICE\_LINE\_TEXT) 🡪 To count the number of each spare part

data.isnull().sum().to\_frame() 🡪 To check for the null values in the data

data.dropna(axis=0,inplace=True) 🡪 To drop all the null values present in the data

1. Convert argument to datetime

**data['Job\_Card\_Date']=pd.to\_datetime(data['Job\_Card\_Date'])**

**type(data['Job\_Card\_Date'].iloc[0]) 🡪 Check the type**

**data['Job\_Card\_Date'] 🡪 Display the details of Job Card Date**

1. Split the data using DatetimeIndex function

* **data['Year'] = pd.DatetimeIndex(data['Job\_Card\_Date']).year** 🡪

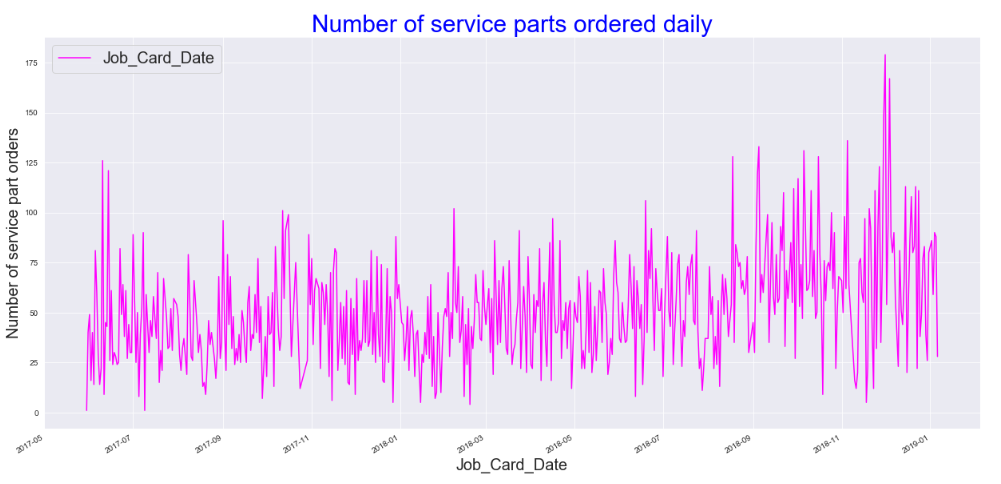
**To separate year from the job card date**

* **data['Month'] = pd.DatetimeIndex(data['Job\_Card\_Date']).month** 🡪

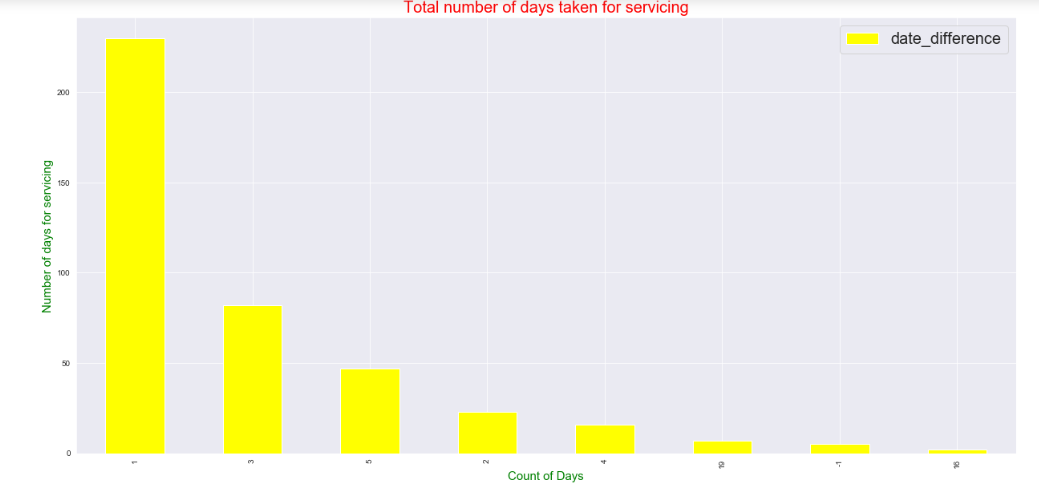
**To separate month from the job card dates**

* **data.groupby('INVOICE\_LINE\_TEXT').count() 🡪 Count the value of each spare part and group them**

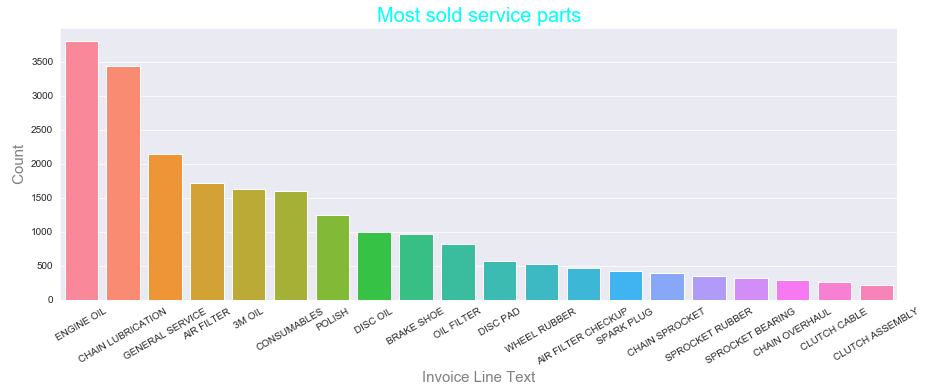
1. Finding the trends



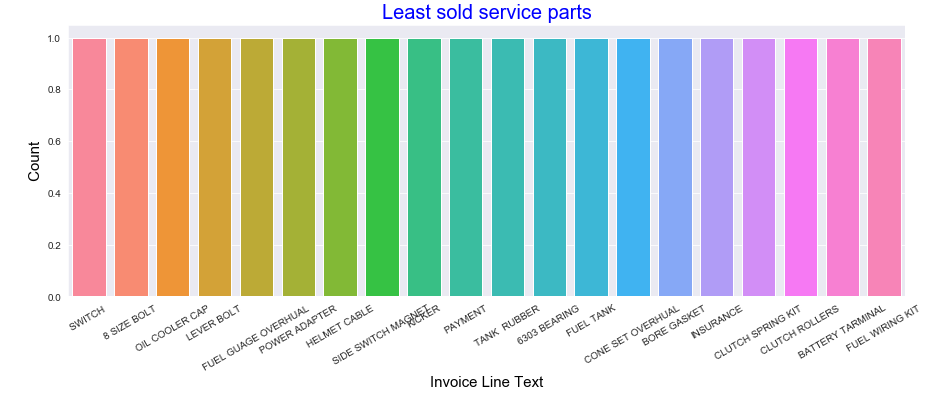
**Fig 2: Line plot for number of service parts that are ordered on daily basis**

****

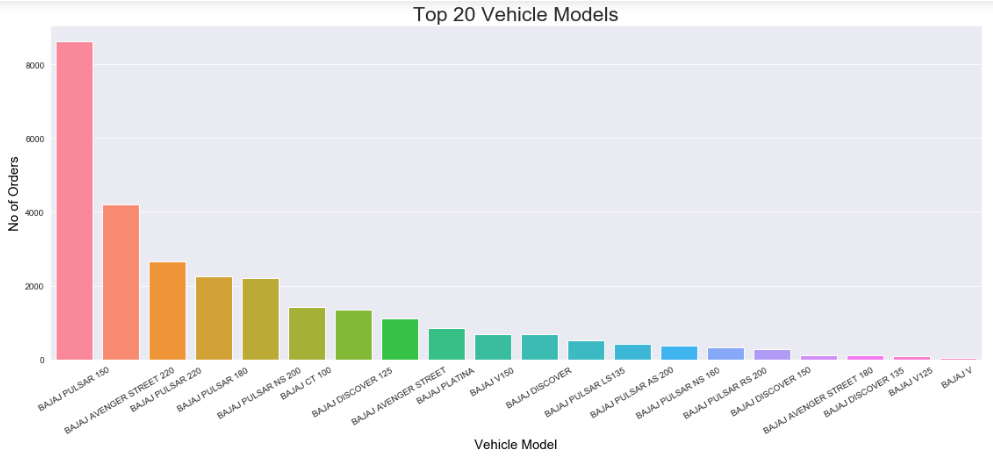
**Fig 3: Bar plot representation for total number of days taken for servicing of a vehicle**

****

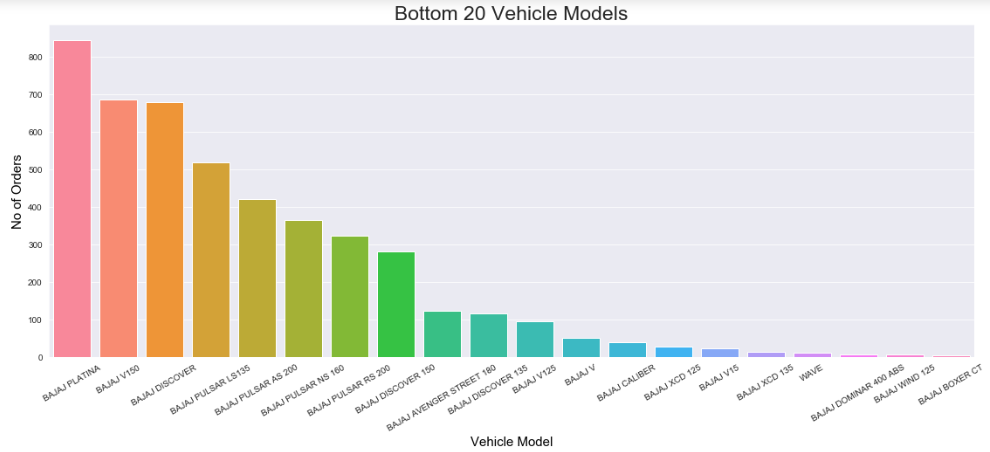
**Fig 4: Bar plot representing most sold service parts**

****

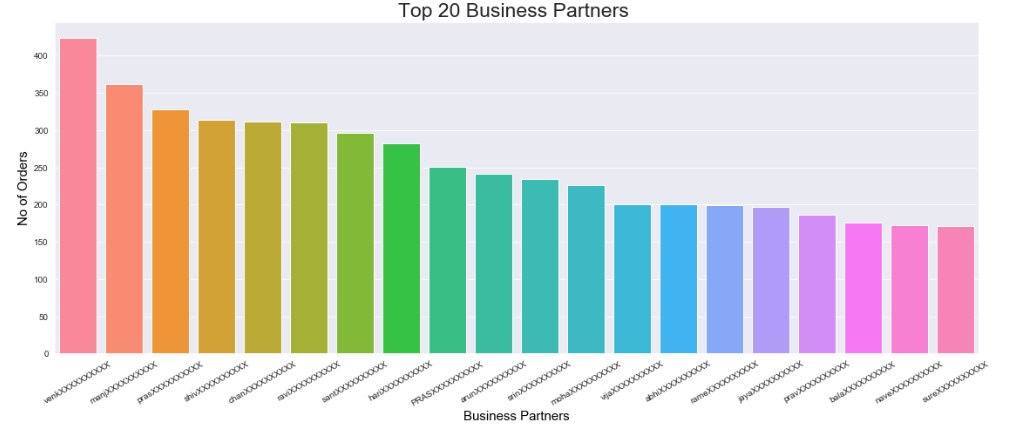
**Fig 5: Bar plot representing least sold service parts**

****

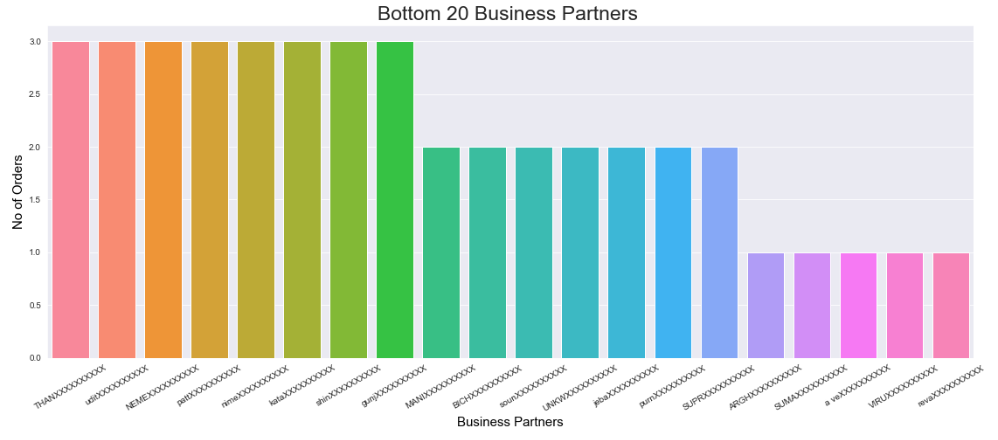
**Fig 6: Bar plot representing Top 20 Vehicle Models**

****

**Fig 7: Bar plot representing Bottom 20 Vehicle Models**

****

**Fig 8: Bar plot representing Top 20 Business Partners**

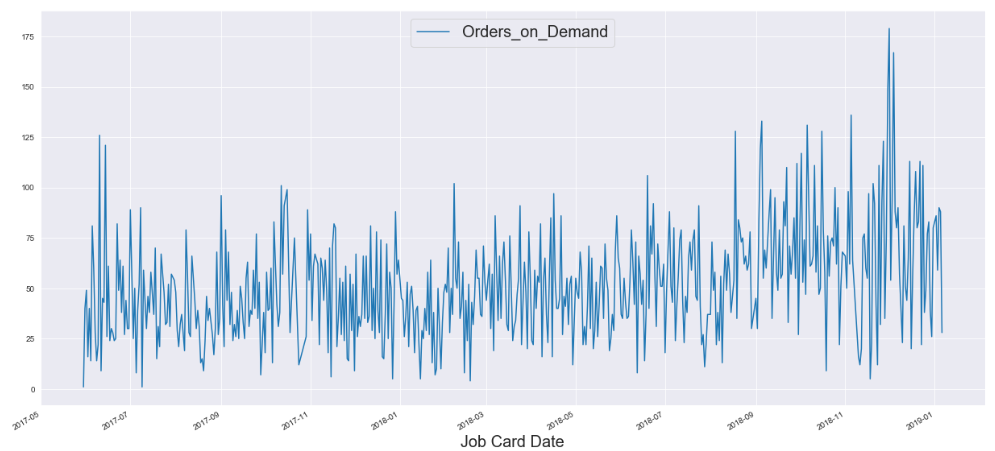
****

**Fig 9: Bar plot representing Bottom 20 Business Partners**

1. Creating a new Dataframe containing Invoice Line Text and Orders on Demand

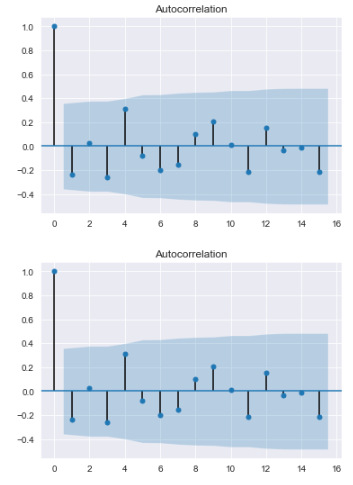
**data\_new=data.groupby(data.Job\_Card\_Date).sum()**

1. Timeseries Forecasting



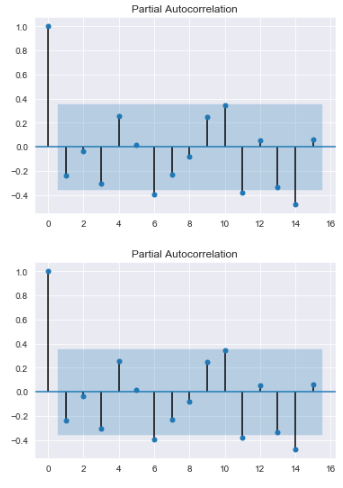
**Fig 10: Line plot representing orders on demand with respect to Job Card Date for data\_new dataframe**

1. Plot the autocorrelation function(acf)



**Fig 11: Auto correlation function (ACF)**

1. Plot the partial autocorrelation function(pacf)



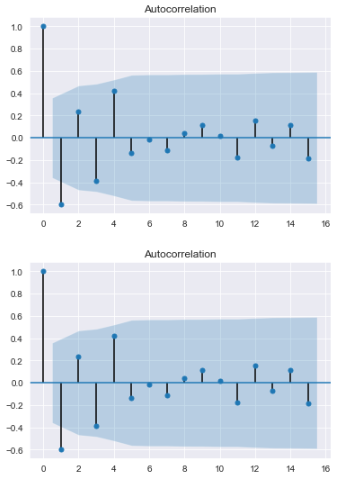
**Fig 12: Partial Auto correlation function (PACF)**

1. Converting data to stationary

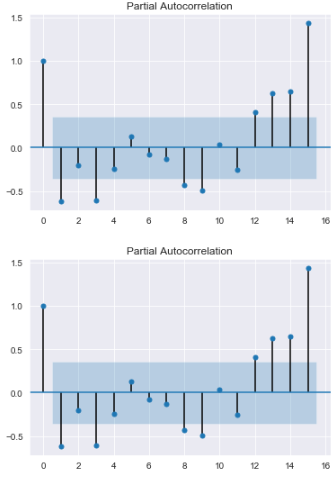
**data\_diff=data\_new.diff(periods=1)**

**data\_diff=data\_diff[1:]**

1. Plot acf and pacf for the new data

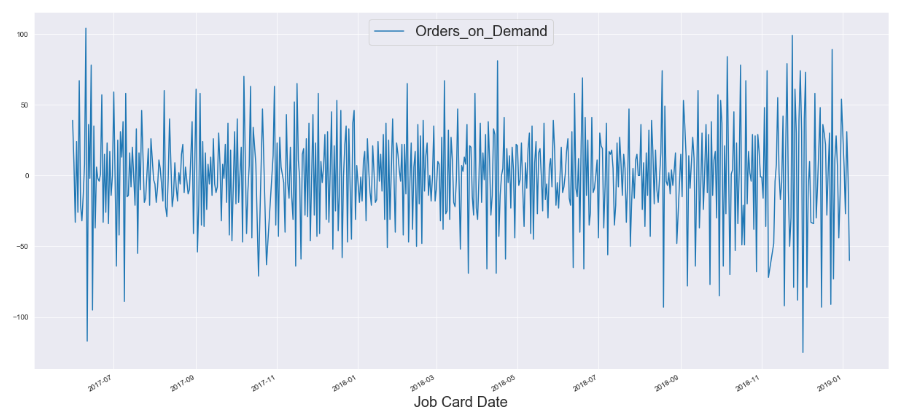


**Fig 13: Auto correlation function (ACF)**



**Fig 14: Partial Auto correlation function (PACF)**

1. Plot the line plot for new data



**Fig 15: Line plot representing orders on demand with respect to Job Card Date for data\_diff dataframe**

Next, different kinds of time series forecasting algorithms were being used. They are as follows:

1. **AutoRegressive(AR) Model**
2. Initialize test and train data and declare predictions

**X=data\_diff.values**

**train=X[0:450]**

**test=X[451:]**

**predictions=[]**

1. Import the necessary packages for AR model and define the model

**from statsmodels.tsa.ar\_model import AR**

**from sklearn.metrics import mean\_squared\_error**

**model\_ar=AR(train)**

**model\_ar\_fit=model\_ar.fit()**

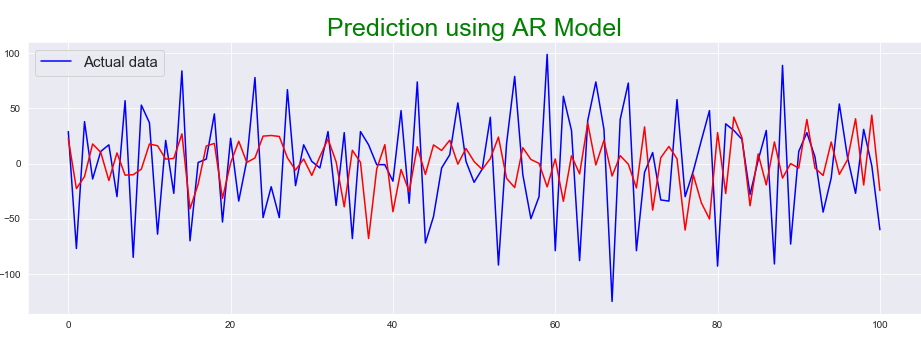
**print(model\_ar\_fit.aic)**

1. Define the value of predictions

**predictions=model\_ar\_fit.predict(start=50,end=150)**

**predictions**

1. Plot the line graph for AR model consisting of actual data and predicted data .



**Fig 16: Prediction using AR Model**

1. Find out the mean squared error

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

1. **Moving Average (MA) Model**
2. Initialize test and train data and declare predictions

**X=data\_diff.values**

**train=X[0:450]**

**test=X[451:]**

**predictions=[]**

1. Import the necessary packages for MA model and define the model

**from statsmodels.tsa.arima\_model import ARMA**

**from sklearn.metrics import mean\_squared\_error**

**model\_ma=ARMA(train, order=(0, 1))**

**model\_ma\_fit=model\_ma.fit()**

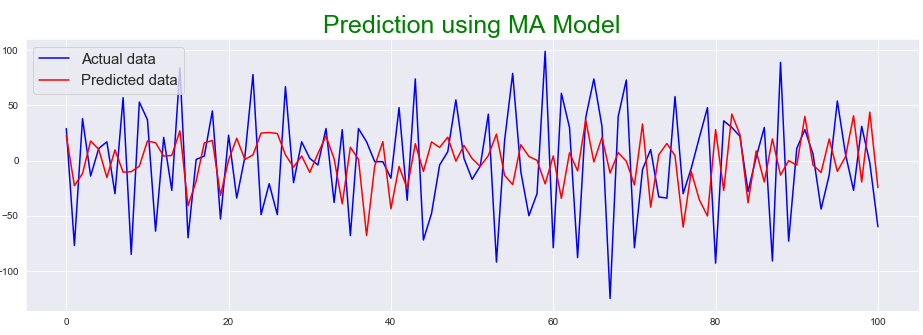
**print(model\_ma\_fit.aic)**

1. Define the value of predictions

**predictions=model\_ar\_fit.predict(start=50,end=150)**

**predictions**

1. Plot the line graph for MA model consisting of actual data and predicted data.

****

**Fig 17: Prediction using MA Model**

1. Find out the mean squared error

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

1. **Auto Regressive Moving Average (ARMA) Model**
2. Initialize test and train data and declare predictions

**X=data\_diff.values**

**train=X[0:450]**

**test=X[451:]**

**predictions=[]**

1. Import the necessary packages for ARMA model and define the model

**from statsmodels.tsa.arima\_model import ARMA**

**from sklearn.metrics import mean\_squared\_error**

**model\_arma=ARMA(train,order=(1,1))**

**model\_arma\_fit=model\_arma.fit()**

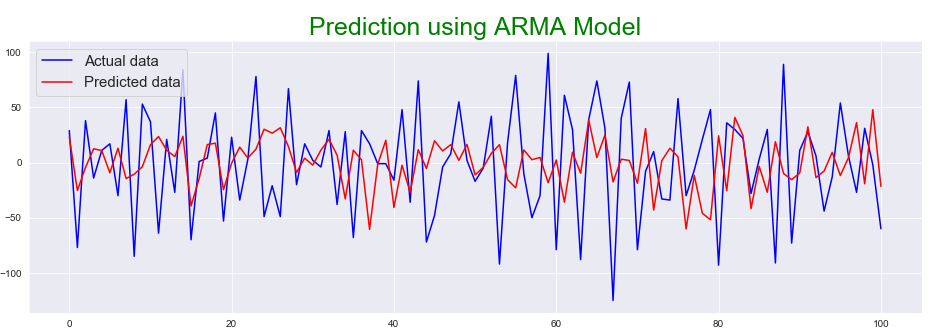
**print(model\_arma\_fit.aic)**

1. Define the value of predictions

**predictions=model\_arma\_fit.predict(start=50,end=150)**

**predictions**

1. Plot the line graph for ARMA model consisting of actual data and predicted data.

****

**Fig 18: Prediction using ARMA Model**

1. Find out the mean squared error

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

1. **Auto Regressive Integrated Moving Average (ARIMA) Model**
2. Initialize test and train data and declare predictions

**X=data\_diff.values**

**train=X[0:450]**

**test=X[451:]**

**predictions=[]**

1. Import the necessary packages for ARIMA model and define the model

**from statsmodels.tsa.arima\_model import ARIMA**

**model\_arima=ARIMA(test, order=(2,1,0))**

**model\_arima\_fit=model\_arima.fit()**

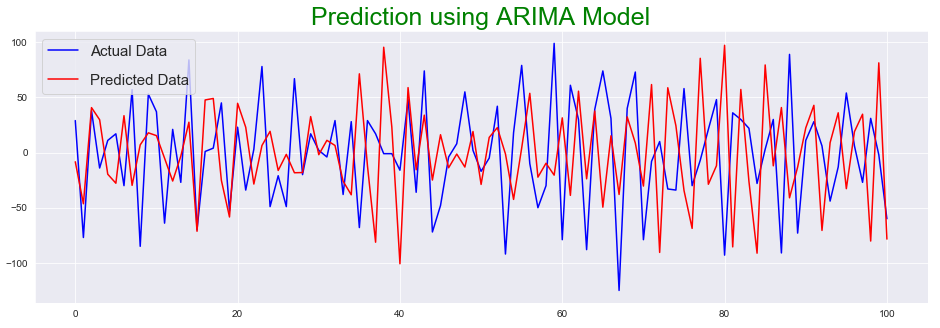
**print(model\_arima\_fit.aic) # Akaike Information Criteria**

1. Define the value of predictions

**predictions=model\_arima\_fit.predict(start=50,end=100)**

**predictions**

1. Plot the line graph for ARIMA model consisting of actual data and predicted data.

****

**Fig 19: Prediction using ARIMA Model**

1. Find out the mean squared error.

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

1. **Seasonal Auto Regressive Integrated Moving Average (SARIMA) Model**
2. Initialize test and train data and declare predictions.

**X=data\_diff.values**

**train=X[0:500]**

**test=X[501:]**

**predictions=[]**

1. Import the necessary packages for SARIMA model and define the model

**from statsmodels.tsa.statespace.sarimax import SARIMAX**

**from sklearn.metrics import mean\_squared\_error**

**model\_sarima = SARIMAX(test, order=(10,2,2), seasonal\_order=(1,1,1,1))**

**model\_sarima\_fit = model\_sarima.fit()**

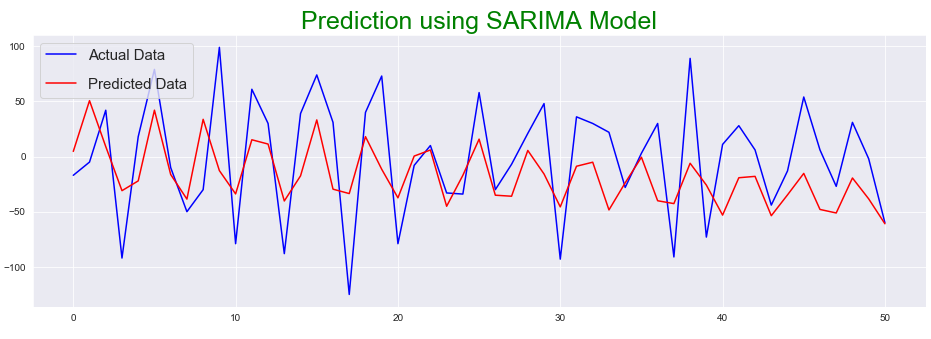
**print(model\_sarima\_fit.aic)**

1. Define the value of predictions.

**predictions=model\_sarima\_fit.predict(start=50,end=100)**

**predictions**

1. Plot the line graph for SARIMA model consisting of actual data and predicted data.

****

**Fig 20: Prediction using SARIMA Model**

1. Find out the mean squared error

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

# Seasonal Auto Regressive Integrated Moving Average with Exogenous Regressors

# (SARIMAX) Model

1. Initialize test and train data and declare predictions.

**X=data\_diff.values**

**train=X[0:400]**

**test=X[401:]**

**predictions=[]**

1. Import the necessary packages for SARIMAX model and define the model

**from statsmodels.tsa.statespace.sarimax import SARIMAX**

**from sklearn.metrics import mean\_squared\_error**

**model\_sarimax = SARIMAX(train,exdog=test,order=(3,2,2), seasonal\_order=(1,1,1,1))**

**model\_sarimax\_fit = model\_sarimax.fit()**

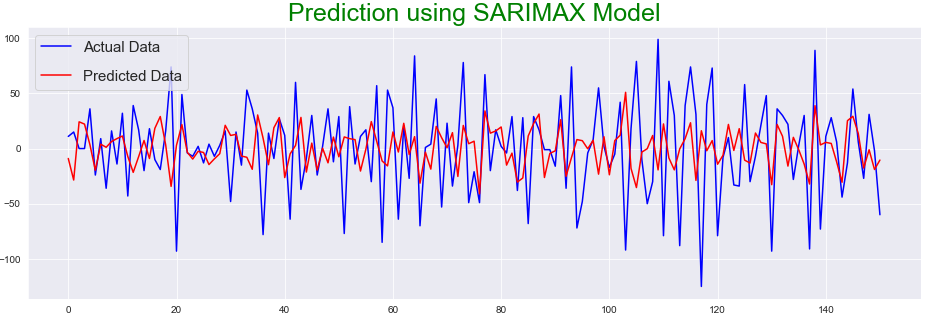
**print(model\_sarimax\_fit.aic)**

1. Define the value of predictions.

**predictions=model\_sarimax\_fit.predict(start=50,end=200)**

**predictions**

1. Plot the line graph for SARIMAX model consisting of actual data and predicted data.

****

**Fig 21: Prediction using SARIMAX Model**

1. Find out the mean squared error

**mean\_squared\_error(test,predictions)**

**np.sqrt(mean\_squared\_error(test,predictions))**

**Conclusion**

In this project, different time series forecasting algorithms were used for finding the better predictions.

The algorithms used were Auto Regression (AR), Moving Average (MA), Auto Regressive Integrated Moving Average (ARIMA), Seasonal Auto Regressive Integrated Moving Average (SARIMA), and Seasonal Auto Regressive Integrated Moving Average with Exogenous Regressors (SARIMAX).

As per the graph, ARIMA model was giving better prediction as compared to all the other models.