Internship Report on

Predictive Analysis Using Machine Learning

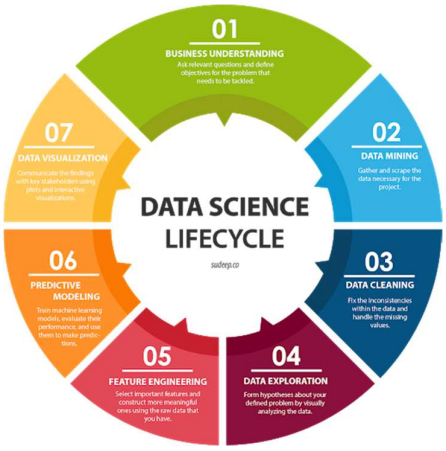
-By Shraddha Reddy

Date: 20-07-2022

Data Science

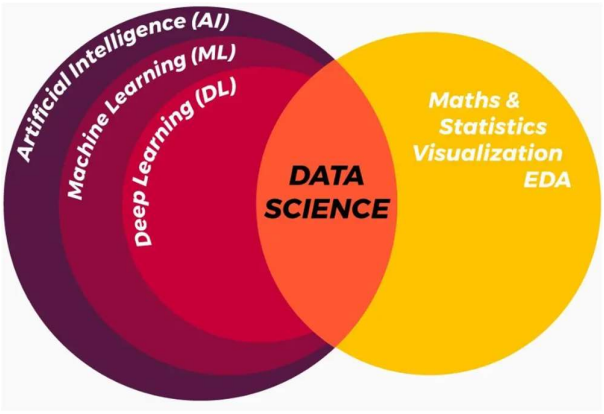
What is Data Science?

Data Science is analysing data using scientific methods, processes, algorithms, and systems for extracting insights and information from structured or unstructured data. Data science lifecycle consists of the following: Business Understanding, Data Mining, Data Cleaning, Data Exploration, Feature Engineering, Predictive Modelling and Data Visualization.



Machine Learning

Machine Learning is a specific part of Artificial Intelligence which focuses on construction and study of system which can learn from data. It involves parsing data, learning from data by figuring out patterns and applying statistic, produce predications and make an informed decision.



Predictive Analysis

Applying various Machine Learning Algorithms on SAP Data, analysing it, and predicting outcomes is called Predictive Analysis. Predictive Analysis uses historical data to predict future events. We use various Machine Learning and Deep Learning algorithms for these processes. We build a model that captures info and important trends and use it for prediction or suggest actions to take for optimal outcomes.

Time Series

Time Series Data

Time series data is sequence of data points, collected at different points in time and indexed in time order. They consist of information of successive measurements made from the same source over a time interval and are used to track change over time. Time series forecasting is the process of analysing time series data using statistics and modelling to make predictions and informed strategic decision making. Time series analysis involves developing models to gain an understanding of the data to understand the underlying causes. We use different forecasting models and techniques for time series analysis.

Machine Learning Models for Time series

Few of the popular machine learning models for timeseries are:

1) ARIMA family – ARIMA, AR, MA, ARMA, ARIMAX and SARIMAX
2) Multivariate Time Series
3) Sectional Motif
4) PyCaret etc.
5)

Components of Timeseries Forecasting methods

There are 4 major components that a Timeseries forecasting model is comprised of:

1) **Trend:** Increase or decrease in the series of data over longer a period.
2) **Seasonality:** Fluctuations in the pattern due to seasonal determinants over a period such as a day, week, month, season.
3) **Cyclical variations:** Occurs when data exhibit rises and falls at irregular intervals.
4) **Random or irregular variations:** Instability due to random factors that do not repeat in the pattern.

Dataset

Understanding Time Series Data

Given Dataset is the information about the number of tickets and types of tickets raised in the time period between 27-02-2020 and 06-05-2022. Our dataset contains the following columns

- Zone
- Dealer Code
- Dealer Name
- Transaction No.
- Description
- Status
- Posting Date
- CATEGORY
- IRTSTATUS
- MPT
- MPTSTATUS
- Changed On
- TRANSACTIONTYPE
- Created By
- Reported By
- Object GUID

Exploratory Data Analysis (EDA)

EDA involves the pre-processing of our data. It performs the processes that are required initially, so that we can discover patterns in our data and make sure if we have the required information and get our data into proper shape. It involves loading dataset, data cleaning, removing, or filling of null values and missing values, drawing insights, perform hypothesis testing and display the statistics and info of our data.

The following snips show the data pre-processing of our data:

```
[77] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sb
     import datetime
     import time
```

```
[78] import io
     data = pd.read_excel('Maindata.xlsx')
     data.head()
```

	Zone	Dealer Code	Dealer Name	Transaction No.	Description	Status	Posting Date	IRT	CATEGORY	IRTSTATUS	MPT	MPTSTATUS	Changed On	TRANSACTIONTYPE	Created By	Reported By	Object GUID
0	NaN	NaN	NaN	2000095635	Medium	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	ANANDSHARMA	NaN	0
1	NaN	NaN	NaN	1000003085	Very High	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	S309PRM0001	NaN	0
2	NaN	NaN	NaN	1000003086	Medium	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	WW14PRM0001	NaN	0
3	NaN	NaN	NaN	2000095644	Very High	Open	2022-05-06	0	After Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DW02HSR0001	NaN	0
4	NaN	NaN	NaN	2000095646	High	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DE04CRE0001	NaN	0

```
[121] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97125 entries, 0 to 97124
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   zone              95080 non-null  object
 1   dealer_code       95080 non-null  object
 2   dealer_name       95056 non-null  object
 3   trans_no.         97125 non-null  int64
 4   description       97118 non-null  object
 5   status            97125 non-null  object
 6   posting_date      97125 non-null  datetime64[ns]
 7   IRT               97125 non-null  int64
 8   category          92095 non-null  object
 9   MPT               97125 non-null  int64
 10  changed_on        97125 non-null  datetime64[ns]
 11  transaction_type  97124 non-null  object
 12  created_by        97125 non-null  object
 13  reported_by       90933 non-null  object
dtypes: datetime64[ns](2), int64(3), object(9)
memory usage: 10.4+ MB
```

```
data.isnull().sum()

Zone               2045
Dealer Code        2045
Dealer Name        2069
Transaction No.       0
Description           7
Status                0
Posting Date          0
IRT                   0
CATEGORY           5030
IRTSTATUS         93335
MPT                   0
MPTSTATUS         86515
Changed On            0
TRANSACTIONTYPE       1
Created By            0
Reported By        6192
Object GUID           0
dtype: int64
```

```
[81] data.drop(columns=["IRTSTATUS","MPTSTATUS","Object GUID"],axis=1,inplace=True)

[82] data.columns=["zone","dealer_code","dealer_name","trans_no.","description","status","posting_date","IRT","category","MPT","changed_on","transaction_type","created_by","reported_by"]
     data.head()
```

	zone	dealer_code	dealer_name	trans_no.	description	status	posting_date	IRT	category	MPT	changed_on	transaction_type	created_by	reported_by
0	NaN	NaN	NaN	2000095635	Medium	Open	2022-05-06	0	Sales	0	2022-05-06	MG_MOTORS_Incident	ANANDSHARMA	NaN
1	NaN	NaN	NaN	1000003085	Very High	Open	2022-05-06	0	NaN	0	2022-05-06	MG Parts Support	S309PRM0001	NaN
2	NaN	NaN	NaN	1000003086	Medium	Open	2022-05-06	0	NaN	0	2022-05-06	MG Parts Support	WW14PRM0001	NaN
3	NaN	NaN	NaN	2000095644	Very High	Open	2022-05-06	0	After Sales	0	2022-05-06	MG_MOTORS_Incident	DW02HSR0001	NaN
4	NaN	NaN	NaN	2000095646	High	Open	2022-05-06	0	Sales	0	2022-05-06	MG_MOTORS_Incident	DE04CRE0001	NaN

```
data.nunique()
```

```
zone                   7
dealer_code          138
dealer_name          135
trans_no           97124
description            6
status                 9
posting_date         794
IRT                  433
category              11
MPT                  869
changed_on           788
transaction_type       2
created_by           400
reported_by          411
dtype: int64
```

```
data.posting_date.max()
```

```
Timestamp('2022-05-06 00:00:00')
```

```
[85] data.posting_date.min()
```

```
Timestamp('2020-02-27 00:00:00')
```

```
[86] date_tickets = data.groupby(["posting_date"]).agg({"description":"count"})
     date_tickets.reset_index(inplace=True)
     date_tickets.head()
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

```
[87] date_tickets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794 entries, 0 to 793
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   posting_date  794 non-null    datetime64[ns]
 1   description   794 non-null    int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 12.5 KB
```

```
[88] date_tickets.description.max()
```

```
321
```

```
[89] date_tickets.description.min()
```

```
1
```

Machine Learning Models and Codes

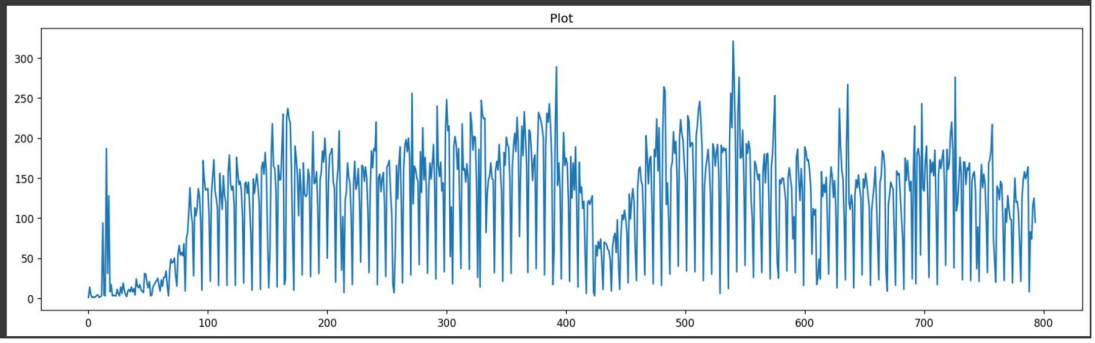
a. Arima Model

Code file name: arima_maindata.ipynb

ARIMA is Auto Regressive Time Series Model.

Plotting the graph of our data will give us the following graph.

```
[122] #Creating Time Series Plot
      plt.figure(figsize=(18,5))
      plt.plot(date_tickets["description"])
      plt.title("Plot")
      plt.show()
```




```
#No Trend
#No Seasonality
#No Cyclic component
#Manual Analysis: Non-Stationary
#Statistical Test - Hypothesis Testinf adfuller test
```

```
[92] from statsmodels.tsa.stattools import adfuller
     result= adfuller(date_tickets["description"])
     print("ADF Statistic is: ",result[0])
     print("p-value is: ",result[1])

     ADF Statistic is:  -2.5630610747387443
     p-value is:  0.10087461824113392
```

```
#Null-Hypothesis - does not follow
#Alternate Hypothesis follow - Time Series is Non Stationary
result
```

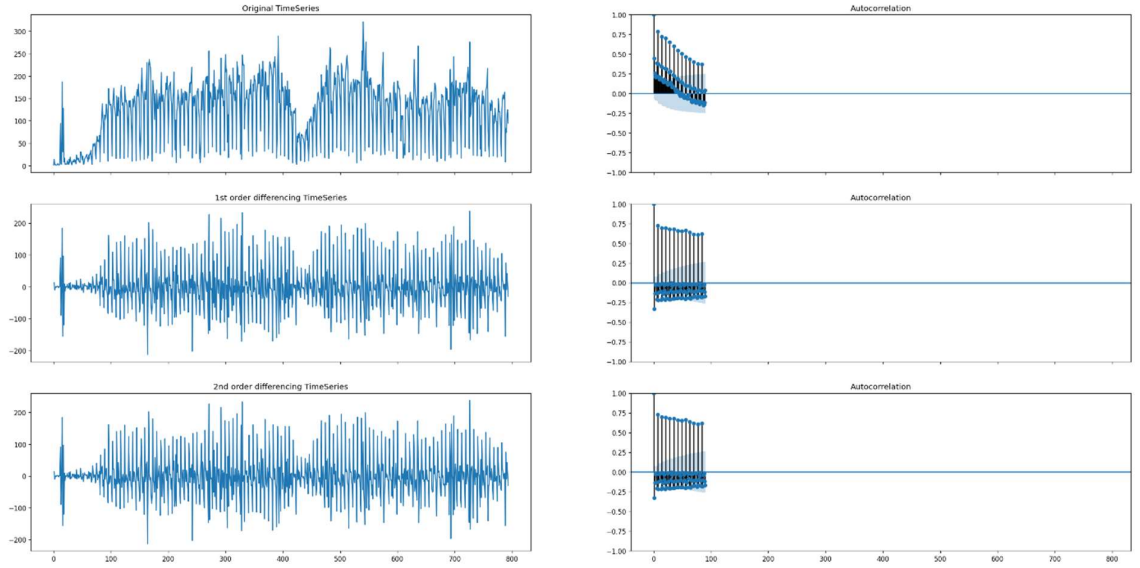
```
(-2.5630610747387443,
 0.10087461824113392,
 21,
 772,
 {'1%': -3.438848933683600 3,
  '10%': -2.568767459327767,
  '5%': -2.865291103159288},
 7720.633022456812)
```

There is No Seasonality, No Trend nor Cyclic component observed in our data.

We apply differencing to convert out Non Stationary data into Stationary data.

```
[94] #Applying Differencing - Convert Non Stationery to Stationery Time Series
     #Autocorrelation - Determining  order of difference
     #p-value>0.05 - timeseries is not stationary
```

```
[123] # importing plot_acf from statsmodel

      #figure & axis
      from statsmodels.graphics.tsaplots import plot_acf
      plt.rcParams.update({'figure.figsize':(30,15),'figure.dpi':120})
      fig,axes=plt.subplots(3,2,sharex=True)

      #original timeseries plot
      axes[0,0].plot(date_tickets["description"])
      axes[0,0].set_title("Original TimeSeries")
      plot_acf(date_tickets["description"],lags=90,ax=axes[0,1])

      #1st order differencing
      axes[1,0].plot(date_tickets["description"].diff())
      axes[1,0].set_title("1st order differencing TimeSeries")
      plot_acf(date_tickets["description"].diff().dropna(),lags=90,ax=axes[1,1])


      #2d order differencing
      axes[2,0].plot(date_tickets["description"].diff())
      axes[2,0].set_title("2nd order differencing TimeSeries")
      plot_acf(date_tickets["description"].diff().dropna(),lags=90,ax=axes[2,1])

      plt.show()
```



Importing pmdarima

```
[96] !pip install pmdarima

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1.8.5)
     Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (0.29.30)
     Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.3.5)
     Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (0.13.2)
     Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.21.6)
     Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.24.3)
     Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.1.0)
     Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.7.3)
     Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (1.0.2)
     Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.7/dist-packages (from pmdarima) (57.4.0)
     Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
     Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.19->pmdarima) (2022.1)
     Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.19->pmdarima) (1.15.0)
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22->pmdarima) (3.1.0)
     Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (from statsmodels!=0.12.0,>=0.11->pmdarima) (0.5.2)
     Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels!=0.12.0,>=0.11->pmdarima) (21.3)
     Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=21.3->statsmodels!=0.12.0,>=0.11->pmdarima) (3.0.9)
```

ACF Plot

```
   from pmdarima.arima.utils import ndiffs

[98] #Satistical measure of Order of difference
     #Verify difference - 3 methods: kpss, adf and pp

[99] y = date_tickets["description"]
     ndiffs(y,test = "kpss")

     1

[100] ndiffs(y,test="adf")

      0

[101] ndiffs(y,test="pp")

      0

[102] #Determine model Parameters
      from pmdarima import auto_arima
```

```
model = auto_arima(date_tickets["description"], start_p=1, start_q=1,
                   test='adf',           #use adf test to find optimal d
                   max_p=5, max_q=5,     #maximum p and q
                   m=1,                  #frequency of series
                   d=None,
                   seasonal=False,
                   start_P=0,
                   D=0,
                   trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=8700.689, Time=0.25 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=10103.925, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=8994.063, Time=0.09 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=9547.734, Time=0.23 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=8682.050, Time=0.54 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=8929.946, Time=0.07 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=8665.990, Time=0.77 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=8880.925, Time=0.11 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=8649.855, Time=1.39 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=8852.779, Time=0.40 sec
 ARIMA(5,0,1)(0,0,0)[0]             : AIC=8612.620, Time=2.87 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=8826.586, Time=0.82 sec
 ARIMA(5,0,2)(0,0,0)[0]             : AIC=8652.263, Time=3.57 sec
 ARIMA(4,0,2)(0,0,0)[0]             : AIC=8708.943, Time=3.07 sec
 ARIMA(5,0,1)(0,0,0)[0] intercept   : AIC=8611.728, Time=4.28 sec
 ARIMA(4,0,1)(0,0,0)[0] intercept   : AIC=8650.777, Time=4.47 sec
 ARIMA(5,0,0)(0,0,0)[0] intercept   : AIC=8773.548, Time=0.68 sec
 ARIMA(5,0,2)(0,0,0)[0] intercept   : AIC=8624.132, Time=5.05 sec
 ARIMA(4,0,0)(0,0,0)[0] intercept   : AIC=8780.366, Time=0.53 sec
 ARIMA(4,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=2.51 sec

Best model:  ARIMA(5,0,1)(0,0,0)[0] intercept
Total fit time: 31.782 seconds
```

```
[105] print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  794
Model:               SARIMAX(5, 0, 1)   Log Likelihood               -4297.864
Date:                Sun, 17 Jul 2022   AIC                          8611.728
Time:                        16:49:48   BIC                          8649.145
Sample:                             0   HQIC                         8626.107
                                  - 794
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      1.1284      1.031      1.094      0.274      -0.892       3.149
ar.L1          1.0430      0.040     25.813      0.000       0.964       1.122
ar.L2         -0.2571      0.053     -4.837      0.000      -0.361      -0.153
ar.L3          0.0420      0.062      0.676      0.499      -0.080       0.164
ar.L4         -0.0757      0.064     -1.183      0.237      -0.201       0.050
ar.L5          0.2370      0.045      5.323      0.000       0.150       0.324
ma.L1         -0.8343      0.030    -28.185      0.000      -0.892      -0.776
sigma2      2967.4370    158.844     18.681      0.000    2656.108    3278.766
===================================================================================
Ljung-Box (L1) (Q):                   3.81   Jarque-Bera (JB):                38.22
Prob(Q):                              0.05   Prob(JB):                         0.00
Heteroskedasticity (H):               1.41   Skew:                            -0.50
Prob(H) (two-sided):                  0.01   Kurtosis:                         3.37
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
#p=5, d=0, q=1
#split data
#perform split keeping time series intact
train = date_tickets.iloc[:670]
test = date_tickets.iloc[670:]
train.head()
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

```
[107] train.shape
```

```
(670, 2)
```

Dividing our data into train and test data in the ratio of 85%:15%

Total data length = 794

Train : Test = 670:124

```
model = auto_arima(train["description"], start_p=1, start_q=1,
                   test='adf',
                   max_p=5, max_q=5,
                   m=1,
                   d=None,
                   seasonal=False,
                   start_P=0,
                   D=0,
                   trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=7318.306, Time=0.39 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=7610.052, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=7543.548, Time=0.09 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=7339.119, Time=0.22 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=7608.052, Time=0.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=7303.359, Time=0.54 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=7495.276, Time=0.13 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=7288.860, Time=0.69 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=7467.793, Time=0.12 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=7255.554, Time=0.90 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=7441.507, Time=0.20 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=7168.007, Time=1.24 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=7346.778, Time=0.60 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=7048.818, Time=2.28 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=7249.896, Time=1.24 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=7038.402, Time=2.35 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=7202.459, Time=1.96 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=7024.260, Time=2.82 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=7129.700, Time=2.59 sec
 ARIMA(5,1,5)(0,0,0)[0] intercept   : AIC=6983.941, Time=3.38 sec
 ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=3.63 sec
 ARIMA(5,1,5)(0,0,0)[0]             : AIC=inf, Time=2.57 sec

Best model:  ARIMA(5,1,5)(0,0,0)[0] intercept
Total fit time: 28.035 seconds
```

```
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  670
Model:               SARIMAX(5, 1, 5)   Log Likelihood               -3479.971
Date:                Sun, 17 Jul 2022   AIC                           6983.941
Time:                        16:50:18   BIC                           7038.011
Sample:                             0   HQIC                          7004.886
                                  - 670
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.4288      1.189      0.361      0.718      -1.902       2.760
ar.L1          0.5649      0.093      6.078      0.000       0.383       0.747
ar.L2         -1.2526      0.075    -16.728      0.000      -1.399      -1.106
ar.L3          0.4717      0.131      3.601      0.000       0.215       0.728
ar.L4         -0.8016      0.075    -10.622      0.000      -0.949      -0.654
ar.L5         -0.2232      0.088     -2.522      0.012      -0.397      -0.050
ma.L1         -1.4789      0.079    -18.679      0.000      -1.634      -1.324
ma.L2          1.8855      0.100     18.915      0.000       1.690       2.081
ma.L3         -1.7764      0.114    -15.601      0.000      -2.000      -1.553
ma.L4          1.4128      0.099     14.295      0.000       1.219       1.607
ma.L5         -0.6641      0.079     -8.368      0.000      -0.820      -0.509
sigma2      3128.9455    229.693     13.622      0.000    2678.756    3579.135
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                85.32
Prob(Q):                              0.87   Prob(JB):                         0.00
Heteroskedasticity (H):               1.25   Skew:                            -0.23
Prob(H) (two-sided):                  0.10   Kurtosis:                         4.69
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
model.fit(train["description"])
```

```
ARIMA(order=(5, 1, 5), scoring_args={}, suppress_warnings=True)
```

```
[111] test.shape
```

```
(124, 2)
```

```
[112] forecast, confidence_interval = model.predict(n_periods=124, return_conf_int=True, alpha=0.05)
```

```
[113] forecast
```

```
array([109.8696988 , 161.31184493, 141.23750441, 146.34300323,
       151.6149646 ,  75.41079143,  33.20383287, 108.11508034,
       162.42056334, 139.69384535, 145.43713268, 152.56649282,
        78.85964882,  37.5249282 , 110.75768312, 162.56918313,
       138.53106365, 144.60912846, 153.5446065 ,  82.19391049,
        41.69367721, 113.32288194, 162.77215616, 137.51000466,
       143.90428835, 154.53646599,  85.42222261,  45.71771805,
       115.81516836, 163.02671375, 136.62301605, 143.3158237 ,
       155.54121027,  88.55020934,  49.60485616, 118.2386506 ,
       163.33025807, 135.86285013, 142.83732001, 156.55803335,
        91.58319676,  53.36247363, 120.59720685, 163.68032336,
       135.22264894, 142.46271486, 157.58618116,  94.52622855,
        56.99755222, 122.89449809, 164.07456927, 134.69592336,
       142.18627861, 158.62494825,  97.384081  ,  60.51669501,
       125.13398025, 164.51077445, 134.27653322, 142.00259614,
       159.67367478, 100.1612713 ,  63.92614703, 127.31891566,
       164.98683057, 133.9586686 , 141.90654962, 160.73174356,
       102.86210012,  67.2318146 , 129.4523839 , 165.50073654,
       133.73683209, 141.8933022 , 161.79857741, 105.49060603,
        70.43928375, 131.53729197, 166.05059309, 133.60582197,
       141.95828259, 162.87363663, 108.0506358 ,  73.55383759,
       133.57638399, 166.6345976 , 133.56071636, 142.0971705 ,
       163.95641656, 110.54582668,  76.58047275, 135.57225028,
       167.2510392 , 133.59685812, 142.3058829 , 165.04644543,
       112.97962298,  79.52391488, 137.52733598, 167.89829407,
       133.70984072, 142.58056097, 166.1432822 , 115.35528635,
        82.38863343, 139.44394914, 168.57482108, 133.89549469,
       142.91755783, 167.24651466, 117.6759054 ,  85.17885545,
       141.32426841, 169.27915754, 134.14987491, 143.31342687,
       168.35575754, 119.94440491,  87.89857884, 143.1703503 ,
       170.00991522, 134.46924857, 143.76491084, 169.47065083])
```

```
confidence_interval
```

```
array([[ 2.35236176e-01,  2.19504161e+02],
       [ 5.12719917e+01,  2.71351698e+02],
       [ 2.89763308e+01,  2.53498678e+02],
       [ 3.34520432e+01,  2.59233963e+02],
       [ 3.82982911e+01,  2.64931638e+02],
       [-3.84029128e+01,  1.89224496e+02],
       [-8.25229877e+01,  1.48930653e+02],
       [-1.14876036e+01,  2.27717764e+02],
       [ 3.98415417e+01,  2.84999585e+02],
       [ 1.57911360e+01,  2.63596555e+02],
       [ 2.08333912e+01,  2.70040874e+02],
       [ 2.75494639e+01,  2.77583522e+02],
       [-4.66413722e+01,  2.04360670e+02],
       [-8.96880949e+01,  1.64737951e+02],
       [-1.98728780e+01,  2.41388244e+02],
       [ 2.93036641e+01,  2.95834702e+02],
       [ 4.05816958e+00,  2.73003958e+02],
       [ 9.45569121e+00,  2.79762566e+02],
       [ 1.79698283e+01,  2.89119385e+02],
       [-5.38536026e+01,  2.18241424e+02],
       [-9.59140383e+01,  1.79301393e+02],
       [-2.73474307e+01,  2.53993195e+02],
       [ 1.97341207e+01,  3.05810192e+02],
       [-6.64390451e+00,  2.81663914e+02],
       [-9.15759493e-01,  2.88724336e+02],
       [ 9.28591537e+00,  2.99787017e+02],
       [-6.02935517e+01,  2.31137997e+02],
       [-1.01437579e+02,  1.92873015e+02],
       [-3.41180189e+01,  2.65748356e+02],
       [ 1.09419879e+01,  3.15111440e+02],
       [-1.65032263e+01,  2.89749258e+02],
       [-1.04660369e+01,  2.97097684e+02],
       [ 1.31929604e+00,  3.09763124e+02],
       [-3.12572516e+01,  3.52720739e+02],
       [-8.95760077e+01,  2.95300208e+02],
       [-1.26230517e+02,  2.60694146e+02],
       [-6.58032210e+01,  3.24707989e+02],
       [-3.11497506e+01,  3.62151224e+02],
       [-6.36899274e+01,  3.31163592e+02],
       [-5.61588516e+01,  3.39945456e+02],
       [-3.67443747e+01,  3.60341530e+02],
       [-9.35003779e+01,  3.04481590e+02],
       [-1.29533451e+02,  2.70412019e+02],
       [-7.01291971e+01,  3.33203781e+02],
       [-3.69338643e+01,  3.69035051e+02],
       [-7.01270880e+01,  3.37338732e+02],
       [-6.23970280e+01,  3.46313593e+02],
       [-4.19789931e+01,  3.67726266e+02],
       [-9.72490293e+01,  3.13350301e+02],
       [-1.32689394e+02,  2.79797069e+02],
       [-7.42716598e+01,  3.41424428e+02],
       [-4.24626706e+01,  3.75731866e+02],
       [-7.62598978e+01,  3.43381331e+02],
       [-6.83430051e+01,  3.52537346e+02],
       [-4.69869949e+01,  3.74899828e+02],
       [-1.00843653e+02,  3.21935306e+02],
       [-1.35718028e+02,  2.88878973e+02],
       [-7.82511300e+01,  3.49395631e+02],
       [-4.77597961e+01,  3.82261874e+02],
       [-8.21145339e+01,  3.49308250e+02],
       [-7.40223169e+01,  3.58634083e+02],
       [-5.17903147e+01,  3.81883206e+02],
       [-1.04302228e+02,  3.30261474e+02],
       [-1.38635534e+02,  2.97683363e+02],
       [-8.20848013e+01,  3.57139471e+02],
       [-5.28455340e+01,  3.88642122e+02],
       [-8.77137522e+01,  3.55133434e+02],
       [-7.94571219e+01,  3.64618244e+02],
       [-5.64077107e+01,  3.88694275e+02],
       [-1.07639783e+02,  3.38350356e+02],
       [-1.41455312e+02,  3.06232578e+02],
       [-8.57871993e+01,  3.64675098e+02],
       [-5.77374633e+01,  3.94887105e+02],
       [-9.30774983e+01,  3.60868488e+02],
       [-8.46668075e+01,  3.70501923e+02],
       [-6.08553570e+01,  3.95348386e+02],
       [-1.10868973e+02,  3.46220784e+02],
       [-1.44188532e+02,  3.14546243e+02],
       [-8.93706864e+01,  3.72019223e+02],
       [-6.24509276e+01,  4.01009243e+02],
       [-9.82233814e+01,  3.66523131e+02],
       [-8.96684590e+01,  3.76295313e+02],
       [-6.51473018e+01,  4.01858817e+02],
       [-1.14000523e+02,  3.53889333e+02],
       [-1.46844562e+02,  3.22641720e+02],
       [-9.28458545e+01,  3.79186555e+02],
       [-6.69994100e+01,  4.07019240e+02],
       [-1.03167048e+02,  3.72105546e+02],
       [-9.44772305e+01,  3.82007052e+02],
       [-6.92958272e+01,  4.08237129e+02]])
```

```
[115] test["forecast"] = forecast
      test["lower_range"] = confidence_interval[:,0]
      test["upper_range"] = confidence_interval[:,1]
      test
```

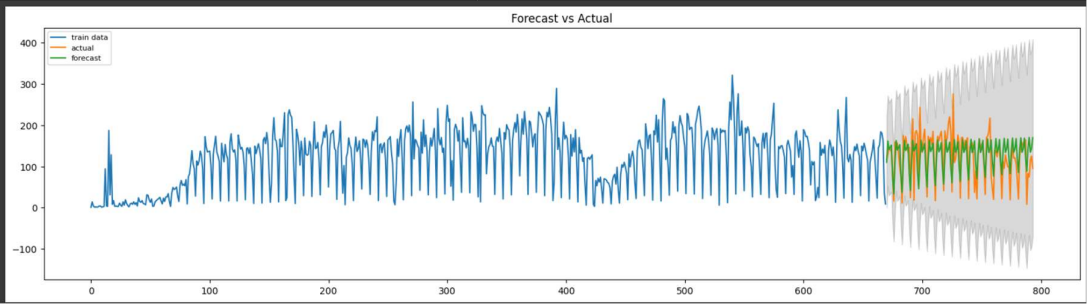
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

	posting_date	description	forecast	lower_range	upper_range
670	2022-01-03	117	109.869699	0.235236	219.504161
671	2022-01-04	129	161.311845	51.271992	271.351698

	posting_date	description	forecast	lower_range	upper_range
670	2022-01-03	117	109.869699	0.235236	219.504161
671	2022-01-04	129	161.311845	51.271992	271.351698
672	2022-01-05	145	141.237504	28.976331	253.498678
673	2022-01-06	140	146.343003	33.452043	259.233963
674	2022-01-07	136	151.614965	38.298291	264.931638
...	...	...	...	...	...
789	2022-05-02	83	143.170350	-92.845854	379.186555
790	2022-05-03	74	170.009915	-66.999410	407.019240
791	2022-05-04	116	134.469249	-103.167048	372.105546
792	2022-05-05	125	143.764911	-94.477231	382.007052
793	2022-05-06	95	169.470651	-69.295827	408.237129

124 rows × 5 columns

```
plt.figure(figsize=(20,5), dpi=100)
plt.plot(train["description"], label='train data')
plt.plot(test["description"], label='actual')
plt.plot(test["forecast"], label='forecast')
plt.fill_between(test.index, test["lower_range"], test["upper_range"], color='k', alpha=0.15)
plt.title('Forecast vs Actual')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```




```
[124] #MAPE = Mean Absolute Percentage Error
      from pmdarima.metrics import smape

[125] mpe = smape(test["description"], test["forecast"])
      print("MPE is", mpe)

      MPE is 33.97085924722436

[126] #model is 66% accurate
```

Model Accuracy = 66%

ARIMA Model including all the missing dates.

As we can observe from our pre-processed data that few dates are missing due to no ticket raised, we consider those dates too with 0 as number of tickets and work with ARIMA model.

Code file name: arima_all_dates.ipynb

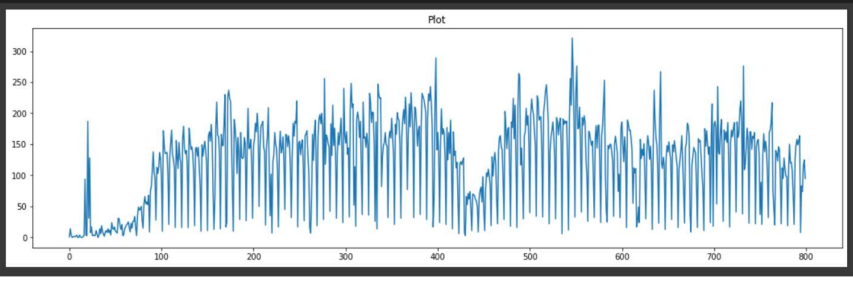
```
[ ] idx = pd.date_range('2020-02-27','2022-05-06')
    dt.index = pd.DatetimeIndex(dt.index)
    dt = dt.reindex(idx, fill_value=0)
    print(dt)
```

```
            description
2020-02-27            1
2020-02-28           14
2020-02-29            4
2020-03-01            0
2020-03-02            1
...                 ...
2022-05-02           83
2022-05-03           74
2022-05-04          116
2022-05-05          125
2022-05-06           95

[800 rows x 1 columns]
```

```
dt.reset_index(inplace=True)
dt.columns=["posting_date","description"]
dt.head(10)
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-01	0
4	2020-03-02	1
5	2020-03-03	2
6	2020-03-04	1
7	2020-03-05	2
8	2020-03-06	4
9	2020-03-07	0

```
plt.figure(figsize=(18,5))
plt.plot(dt["description"])
plt.title("Plot")
plt.show()
```



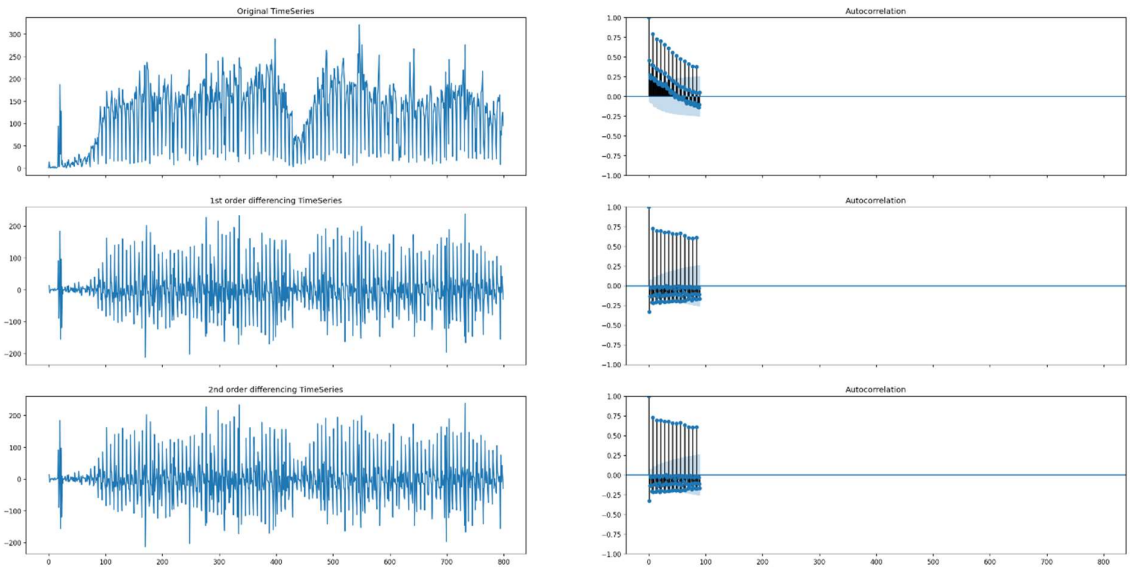

```
from statsmodels.tsa.stattools import adfuller
result= adfuller(dt["description"])
print("ADF Statistic is: ",result[0])
print("p-value is: ",result[1])
```

```
ADF Statistic is:  -2.4891028354323024
p-value is:  0.11814828625975088
```

```
[ ] result
```

```
(-2.4891028354323024,
 0.11814828625975088,
 21,
 778,
 {'1%': -3.438783171038672,
  '10%': -2.568752018688748,
  '5%': -2.865262118650577},
 7788.404724729095)
```

```
#figure & axis
from statsmodels.graphics.tsaplots import plot_acf
plt.rcParams.update({'figure.figsize':(30,15),'figure.dpi':120})
fig,axes=plt.subplots(3,2,sharex=True)

#original timeseries plot
axes[0,0].plot(dt["description"])
axes[0,0].set_title("Original TimeSeries")
plot_acf(dt["description"],lags=90,ax=axes[0,1])

#1st order differencing
axes[1,0].plot(dt["description"].diff())
axes[1,0].set_title("1st order differencing TimeSeries")
plot_acf(dt["description"].diff().dropna(),lags=90,ax=axes[1,1])


#2d order differencing
axes[2,0].plot(dt["description"].diff())
axes[2,0].set_title("2nd order differencing TimeSeries")
plot_acf(dt["description"].diff().dropna(),lags=90,ax=axes[2,1])

plt.show()
```



```
[ ] #Determine model Parameters
    from pmdarima import auto_arima
```

```
[ ] model = auto_arima(dt["description"], start_p=1, start_q=1,
                       test='adf',           #use adf test to find optimal d
                       max_p=5, max_q=5,     #maximum p and q
                       m=1,                  #frequency of series
                       d=None,
                       seasonal=False,
                       start_P=0,
                       D=0,
                       trace=True,
                       error_action='ignore',
                       suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=8729.805, Time=0.31 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=9098.465, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=9008.284, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=8748.741, Time=0.18 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=9096.468, Time=0.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=8713.725, Time=0.44 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=8946.842, Time=0.09 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=8697.579, Time=0.61 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=8912.858, Time=0.11 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=8660.212, Time=0.91 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=8882.792, Time=0.13 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=8561.505, Time=1.06 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=8779.620, Time=0.19 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=8416.657, Time=2.01 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=8651.129, Time=1.40 sec
```

```
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=8651.129, Time=1.40 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=8409.469, Time=2.03 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=8515.286, Time=1.75 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=8403.824, Time=2.34 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=8518.253, Time=2.16 sec
 ARIMA(5,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=3.29 sec
 ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=2.78 sec
 ARIMA(5,1,4)(0,0,0)[0]             : AIC=8401.319, Time=1.33 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=inf, Time=1.54 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=8407.021, Time=1.02 sec
 ARIMA(5,1,5)(0,0,0)[0]             : AIC=inf, Time=1.82 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=8588.179, Time=1.41 sec
 ARIMA(4,1,5)(0,0,0)[0]             : AIC=8400.309, Time=1.86 sec
 ARIMA(3,1,5)(0,0,0)[0]             : AIC=8488.116, Time=1.83 sec
 ARIMA(3,1,4)(0,0,0)[0]             : AIC=inf, Time=1.62 sec

Best model:  ARIMA(4,1,5)(0,0,0)[0]
Total fit time: 34.389 seconds
```

```
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  800
Model:               SARIMAX(4, 1, 5)   Log Likelihood               -4190.155
Date:                Tue, 19 Jul 2022   AIC                           8400.309
Time:                        05:43:39   BIC                           8447.143
Sample:                             0   HQIC                          8418.302
                                  - 800
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8056      0.015     55.296      0.000       0.777       0.834
ar.L2         -1.4366      0.020    -73.523      0.000      -1.475      -1.398
ar.L3          0.7902      0.019     42.021      0.000       0.753       0.827
ar.L4         -0.9860      0.014    -70.352      0.000      -1.013      -0.959
ma.L1         -1.6232      0.061    -26.728      0.000      -1.742      -1.504
ma.L2          2.0714      0.098     21.193      0.000       1.880       2.263
ma.L3         -1.8644      0.125    -14.865      0.000      -2.110      -1.619
ma.L4          1.4906      0.096     15.489      0.000       1.302       1.679
ma.L5         -0.7097      0.059    -11.932      0.000      -0.826      -0.593
sigma2      3509.2225    246.435     14.240      0.000    3026.218    3992.227
===================================================================================
Ljung-Box (L1) (Q):                  15.65   Jarque-Bera (JB):                87.24
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.29   Skew:                            -0.27
Prob(H) (two-sided):                  0.04   Kurtosis:                         4.53
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
#p=4, d=1, q=5
#Split data
#perform split keeping time series intact
train = dt.iloc[:680]
test = dt.iloc[680:]
train.head()
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-01	0
4	2020-03-02	1

```
[ ] train.shape
```

```
(680, 2)
```

```
model = auto_arima(train["description"], start_p=1, start_q=1,
                   test='adf',
                   max_p=5, max_q=5,
                   m=1,
                   d=None,
                   seasonal=False,
                   start_P=0,
                   D=0,
                   trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=7418.161, Time=0.28 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=7716.096, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=7648.900, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=7439.376, Time=0.17 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=7714.102, Time=0.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=7402.911, Time=0.37 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=7599.224, Time=0.09 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=7388.204, Time=0.56 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=7570.880, Time=0.09 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=7354.262, Time=0.78 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=7543.491, Time=0.13 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=7265.884, Time=0.95 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=7446.552, Time=0.47 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=7142.668, Time=1.81 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=7347.937, Time=1.06 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=7347.937, Time=1.06 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=7136.409, Time=1.81 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=7300.272, Time=1.42 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=7126.850, Time=2.13 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=1.86 sec
 ARIMA(5,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=3.69 sec
 ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=7235.216, Time=2.42 sec
 ARIMA(5,1,4)(0,0,0)[0]             : AIC=7126.234, Time=1.17 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=7218.382, Time=1.36 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=7131.995, Time=0.90 sec
 ARIMA(5,1,5)(0,0,0)[0]             : AIC=inf, Time=1.81 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=7298.456, Time=0.68 sec
 ARIMA(4,1,5)(0,0,0)[0]             : AIC=7233.225, Time=1.84 sec

Best model:  ARIMA(5,1,4)(0,0,0)[0]
Total fit time: 28.025 seconds
```

```
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  680
Model:               SARIMAX(5, 1, 4)   Log Likelihood               -3553.117
Date:                Tue, 19 Jul 2022   AIC                           7126.234
Time:                        05:44:07   BIC                           7171.440
Sample:                             0   HQIC                          7143.733
                                  - 680
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1402      0.051      2.755      0.006       0.040       0.240
ar.L2         -0.9025      0.042    -21.573      0.000      -0.984      -0.820
ar.L3         -0.1600      0.070     -2.295      0.022      -0.297      -0.023
ar.L4         -0.4578      0.038    -11.923      0.000      -0.533      -0.383
ar.L5         -0.6570      0.046    -14.238      0.000      -0.747      -0.567
ma.L1         -0.8207      0.050    -16.290      0.000      -0.919      -0.722
ma.L2          1.2707      0.059     21.381      0.000       1.154       1.387
ma.L3         -0.7371      0.063    -11.656      0.000      -0.861      -0.613
ma.L4          0.7951      0.051     15.704      0.000       0.696       0.894
sigma2      3193.2524    217.653     14.671      0.000    2766.659    3619.845
===================================================================================
Ljung-Box (L1) (Q):                  13.50   Jarque-Bera (JB):               107.43
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.27   Skew:                            -0.11
Prob(H) (two-sided):                  0.07   Kurtosis:                         4.94
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
model.fit(train["description"])
```

```
ARIMA(order=(5, 1, 4), scoring_args={}, suppress_warnings=True,
      with_intercept=False)
```

```
test.shape
```

```
(120, 2)
```

```
forecast, confidence_interval = model.predict(n_periods=120, return_conf_int=True, alpha=0.05)
```

```
test["forecast"] = forecast
test["lower_range"] = confidence_interval[:,0]
test["upper_range"] = confidence_interval[:,1]
test
```

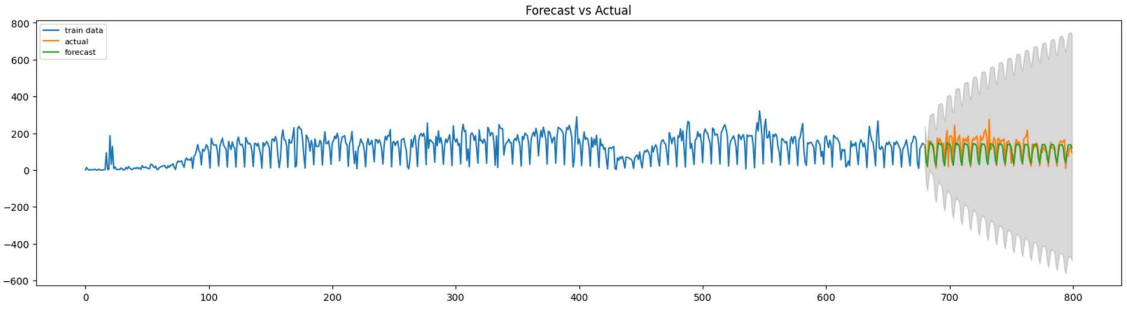
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

	posting_date	description	forecast	lower_range	upper_range
680	2022-01-07	136	127.656305	16.900956	238.411655
681	2022-01-08	79	44.152027	-72.118983	160.423037
682	2022-01-09	16	22.345503	-111.155400	155.846406
683	2022-01-10	159	98.752914	-40.189752	237.695581
684	2022-01-11	154	151.437308	1.312003	301.562613
...	...	...	...	...	...
795	2022-05-02	83	89.830722	-514.322825	693.984269
796	2022-05-03	74	134.903898	-471.293697	741.101493
797	2022-05-04	116	136.824149	-471.269205	744.917502
798	2022-05-05	125	137.886977	-472.851615	748.625570
799	2022-05-06	95	122.331145	-492.041661	736.703951

120 rows × 5 columns

```
plt.figure(figsize=(20,5), dpi=100)
plt.plot(train["description"], label='train data')
plt.plot(test["description"], label='actual')
plt.plot(test["forecast"], label='forecast')
plt.fill_between(test.index, test["lower_range"], test["upper_range"], color='k', alpha=0.15)
plt.title('Forecast vs Actual')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```




```
[ ] #MAPE = Mean Absolute Percentage Error
    from pmdarima.metrics import smape

[ ] mpe = smape(test["description"], test["forecast"])
    print("MPE is", mpe)

    MPE is 35.40874754981367

[ ] #model is 65% accurate
```

Model Accuracy = 65%

There is not much difference in the accuracy between the above 2 versions of ARIMA Model.

b. Linear Regression

Code file name: simple_linear_reg.ipynb

Pre-processing of the data remains similar for simple linear regression. We just need to do one more pre-processing step, which is convert our datetime type to an int or float for computing linearly.

Taking a reference data as Jan 1st 2020, we calculate the number of days from the reference date to the posting date giving us an integral date value and this as a column named “dates”.

```
[19] # date_tickets['posting_date'] = date_tickets["posting_date"].astype('datetime64[D]')
     date_tickets['posting_date'] = pd.to_datetime(date_tickets['posting_date'])
     date_tickets.head()
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

```
[20] date_tickets["dates"] = date_tickets["posting_date"] - pd.to_datetime('1/1/2020')
     date_tickets["dates"] = date_tickets["dates"].dt.days
```

```
date_tickets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794 entries, 0 to 793
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   posting_date  794 non-null    datetime64[ns]
 1   description   794 non-null    int64         
 2   dates         794 non-null    int64         
dtypes: datetime64[ns](1), int64(2)
memory usage: 18.7 KB
```

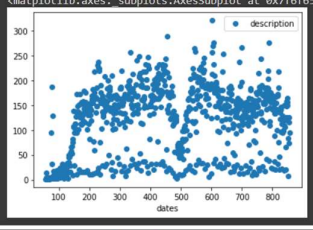
+ Code + Text

```
date_tickets.head()
```

	posting_date	description	dates
0	2020-02-27	1	57
1	2020-02-28	14	58
2	2020-02-29	4	59
3	2020-03-02	1	61
4	2020-03-03	2	62

```
[34] date_tickets.plot(x='dates', y='description', style='o')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f65c2cc10>
```




```
x = date_tickets["dates"]
y = date_tickets["description"]
```

```
import statsmodels.api as sm
model = sm.OLS(y, x).fit()
predictions = model.predict(x)

model.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing inst
  import pandas.util.testing as tm
                            OLS Regression Results
==============================================================================
Dep. Variable:            description   R-squared (uncentered):                   0.697
Model:                            OLS   Adj. R-squared (uncentered):              0.697
Method:                 Least Squares   F-statistic:                              1825.
Date:                Tue, 19 Jul 2022   Prob (F-statistic):                    7.48e-208
Time:                        18:55:44   Log-Likelihood:                         -4576.8
No. Observations:                 794   AIC:                                      9156.
Df Residuals:                     793   BIC:                                      9160.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
dates          0.2278      0.005     42.719      0.000       0.217       0.238
==============================================================================
Omnibus:                       21.702   Durbin-Watson:                   0.869
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               15.774
Skew:                          -0.238   Prob(JB):                     0.000376
Kurtosis:                       2.501   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

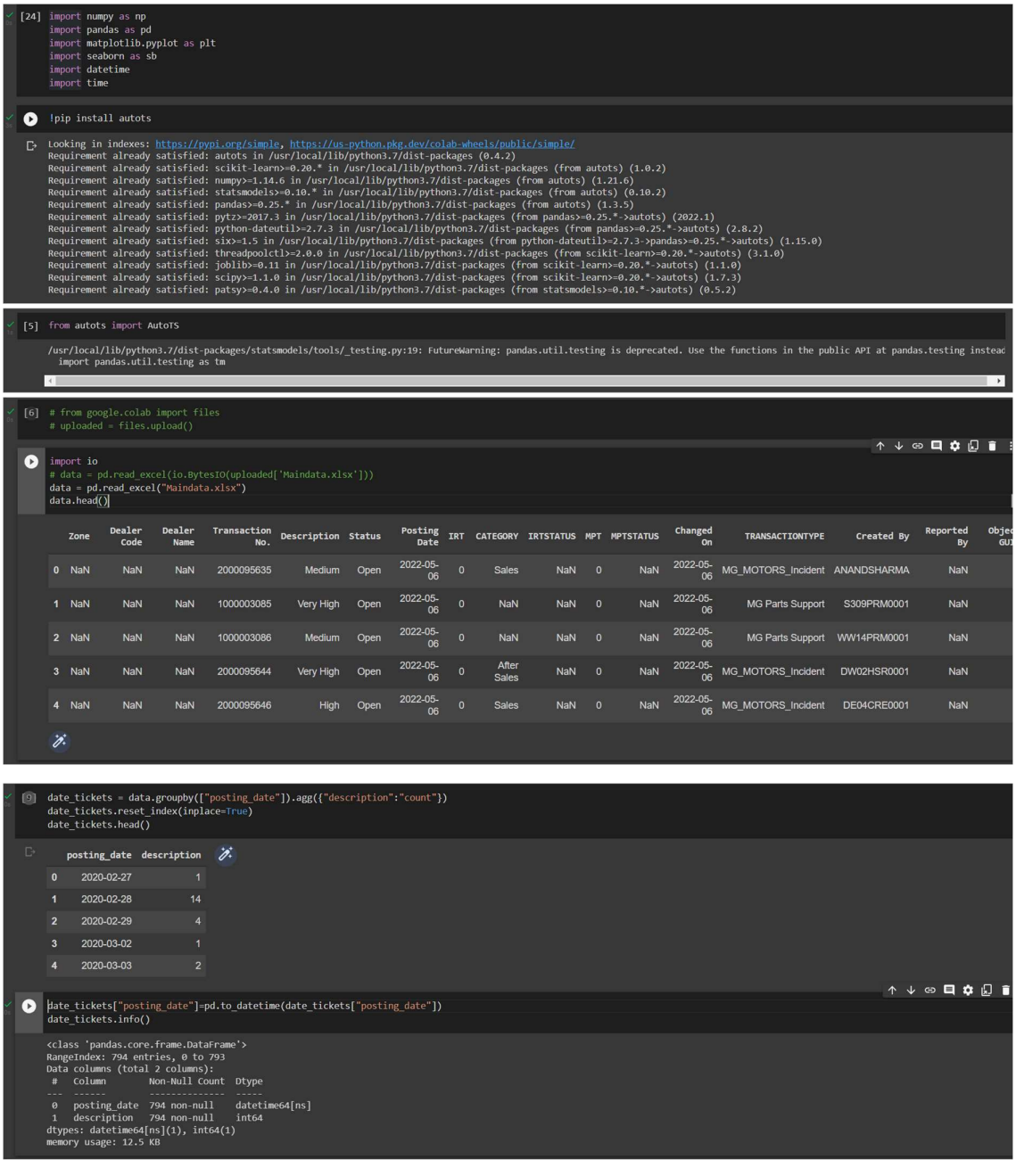
```
#R-squared = R2 =0.697
#Model accuracy = 69.7%
```

Model Accuracy = 69%

c. Auto Time Series

Code file name: autots.ipynb

Auto Time Series trains high-accuracy models within a short time and gives the summary of different models. Most Common models involve PyFlux, Non-Seasonal ARIMA, Seasonal ARIMA, SeactionalMotif and Basic Machine Learning Model



```
[24] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sb
     import datetime
     import time
```

```
!pip install autots
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: autots in /usr/local/lib/python3.7/dist-packages (0.4.2)
Requirement already satisfied: scikit-learn>=0.20.* in /usr/local/lib/python3.7/dist-packages (from autots) (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from autots) (1.21.6)
Requirement already satisfied: statsmodels>=0.10.* in /usr/local/lib/python3.7/dist-packages (from autots) (0.10.2)
Requirement already satisfied: pandas>=0.25.* in /usr/local/lib/python3.7/dist-packages (from autots) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25.*->autots) (2022.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25.*->autots) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.25.*->autots) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.*->autots) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.*->autots) (1.1.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.*->autots) (1.7.3)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from statsmodels>=0.10.*->autots) (0.5.2)
```

```
[5] from autots import AutoTS
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead
  import pandas.util.testing as tm
```

```
[6] # from google.colab import files
    # uploaded = files.upload()
```

```
import io
# data = pd.read_excel(io.BytesIO(uploaded['Maindata.xlsx']))
data = pd.read_excel("Maindata.xlsx")
data.head()
```

	Zone	Dealer Code	Dealer Name	Transaction No.	Description	Status	Posting Date	IRT	CATEGORY	IRTSTATUS	MPT	MPTSTATUS	Changed On	TRANSACTIONTYPE	Created By	Reported By	Objec GU
0	NaN	NaN	NaN	2000095635	Medium	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	ANANDSHARMA	NaN	
1	NaN	NaN	NaN	1000003085	Very High	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	S309PRM0001	NaN	
2	NaN	NaN	NaN	1000003086	Medium	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	WW14PRM0001	NaN	
3	NaN	NaN	NaN	2000095644	Very High	Open	2022-05-06	0	After Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DW02HSR0001	NaN	
4	NaN	NaN	NaN	2000095646	High	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DE04CRE0001	NaN	

```
date_tickets = data.groupby(["posting_date"]).agg({"description":"count"})
date_tickets.reset_index(inplace=True)
date_tickets.head()
```

	posting_date	description
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

```
date_tickets["posting_date"]=pd.to_datetime(date_tickets["posting_date"])
date_tickets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794 entries, 0 to 793
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   posting_date  794 non-null    datetime64[ns]
 1   description   794 non-null    int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 12.5 KB
```

```
[22] mod = AutoTS(forecast_length=20, frequency='infer',  ensemble='simple')

[23] train = date_tickets.iloc[:670]
     test = date_tickets.iloc[670:]
     train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 670 entries, 0 to 669
     Data columns (total 2 columns):
      #   Column        Non-Null Count  Dtype
     ---  ------        --------------  -----
      0   posting_date  670 non-null    datetime64[ns]
      1   description   670 non-null    int64
     dtypes: datetime64[ns](1), int64(1)
     memory usage: 10.6 KB

     mod = mod.fit(train, date_col='posting_date', value_col='description', id_col=None)
     print(mod)

     Inferred frequency is: D
     Model Number: 1 with model AverageValueNaive in generation 0 of 10
     Model Number: 2 with model AverageValueNaive in generation 0 of 10
     Model Number: 3 with model AverageValueNaive in generation 0 of 10
     Model Number: 4 with model DatepartRegression in generation 0 of 10
     Model Number: 5 with model DatepartRegression in generation 0 of 10
     Model Number: 6 with model DatepartRegression in generation 0 of 10
     /usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
       ConvergenceWarning,
     /usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:549: ConvergenceWarning: lbfgs failed to converge (status=1):
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
       self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
     Model Number: 7 with model DatepartRegression in generation 0 of 10
     Epoch 1/50
     21/21 [==============================] - 6s 7ms/step - loss: 0.3862
     Epoch 2/50
     21/21 [==============================] - 0s 6ms/step - loss: 0.3577
     Epoch 3/50
     21/21 [==============================] - 0s 6ms/step - loss: 0.3296
     Epoch 4/50
     21/21 [==============================] - 0s 6ms/step - loss: 0.3259
     Epoch 5/50
     21/21 [==============================] - 0s 5ms/step - loss: 0.3177
     Epoch 6/50
     21/21 [==============================] - 0s 6ms/step - loss: 0.3129
     Epoch 7/50
     21/21 [==============================] - 0s 6ms/step - loss: 0.3098
     155 - GLM with avg smape 51.58:
     Model Number: 156 of 161 with model UnivariateRegression for Validation 3
     156 - UnivariateRegression with avg smape 45.1:
     Model Number: 157 of 161 with model UnivariateRegression for Validation 3
     157 - UnivariateRegression with avg smape 36.93:
     Model Number: 158 of 161 with model UnivariateRegression for Validation 3
     158 - UnivariateRegression with avg smape 37.56:
     Model Number: 159 of 161 with model UnivariateRegression for Validation 3
     159 - UnivariateRegression with avg smape 109.08:
     Model Number: 160 of 161 with model UnivariateRegression for Validation 3
     160 - UnivariateRegression with avg smape 46.67:
     Model Number: 161 of 161 with model UnivariateRegression for Validation 3
     161 - UnivariateRegression with avg smape 152.63:
     Initiated AutoTS object with best model:
     Ensemble
     {}
     {'model_name': 'BestN', 'model_count': 3, 'model_metric': 'mixed_metric', 'models': {'d66e9ac4b268b56d3bb96305619d1cc8': {'Model': 'SectionalMotif', 'ModelParameters': '{"window": 10
     SMAPE: 14.298462916330626, 18.00438909376736, 30.787481931604873, 25.02841973456162
     MAE: 13.573046947750552, 21.634503731080663, 28.4584376819456, 26.74068120861383
     SPL: 0.13566619944030597, 0.2748818614576765, 0.39392809202248885, 0.2943755698979252
```

```
Initiated AutoTS object with best model:
Ensemble
{}
{'model_name': 'BestN', 'model_count': 3, 'model_metric': 'mixed_metric', 'models':
{'d66e9ac4b268b56d3bb96305619d1cc8': {'Model': 'SectionalMotif', 'ModelParameters': '{"window": 10,
"point_method": "weighted_mean", "distance_metric": "braycurtis", "include_differenced": true, "k": 3,
"stride_size": 1, "regression_type": null}', 'TransformationParameters': '{"fillna": "rolling_mean",
"transformations": {"0": "bkfilter", "1": "QuantileTransformer", "2": "PowerTransformer"},
"transformation_params": {"0": {}, "1": {"output_distribution": "uniform", "n_quantiles": 218}, "2":
{}}}'}, '1c5bb85c9adc7f66292bfb4e456c62a8': {'Model': 'MultivariateMotif', 'ModelParameters':
'{"window": 28, "point_method": "mean", "distance_metric": "euclidean", "k": 5, "max_windows": 10000}',
'TransformationParameters': '{"fillna": "rolling_mean", "transformations": {"0": "bkfilter", "1":
"QuantileTransformer", "2": "PowerTransformer"}, "transformation_params": {"0": {}, "1":
{"output_distribution": "uniform", "n_quantiles": 218}, "2": {}}}'},
'ba3a77c9e74389731d5e6f1744c848ea': {'Model': 'MultivariateMotif', 'ModelParameters': '{"window": 28,
"point_method": "mean", "distance_metric": "euclidean", "k": 5, "max_windows": 10000}',
'TransformationParameters': '{"fillna": "ffill_mean_biased", "transformations": {"0": "bkfilter", "1":
"Detrend", "2": "MaxAbsScaler", "3": "QuantileTransformer", "4": "MinMaxScaler"},
"transformation_params": {"0": {}, "1": {"model": "GLS", "phi": 0.99, "window": null}, "2": {}, "3":
{"output_distribution": "uniform", "n_quantiles": 218}, "4": {}}}'}}, 'model_weights': {},
'point_method': 'mean'}
SMAPE: 14.298462916330626, 18.00438909376736, 30.787481931604873, 25.02841973456162
MAE: 13.573046947750552, 21.634503731080663, 28.4584376819456, 26.74068120861383
SPL: 0.13566619944030597, 0.2748818614576765, 0.39392809202248885, 0.2943755698979252
```

```
print(forecast)
```

```
            description
2022-01-03   131.395344
2022-01-04   150.861604
2022-01-05   150.106313
2022-01-06   147.444888
2022-01-07   137.840698
2022-01-08    94.760295
2022-01-09    16.303346
2022-01-10   148.382480
2022-01-11   147.419383
2022-01-12   150.742583
2022-01-13   151.895153
2022-01-14   149.069985
2022-01-15    86.218972
2022-01-16    17.415368
2022-01-17   153.262302
2022-01-18   159.577455
2022-01-19   161.386525
2022-01-20   150.676546
2022-01-21   147.581152
2022-01-22    85.019869
```

```
print(validation)
```

```
                                 ID              Model  \
0    006d670bb178038daa411be4dd42d9c9  DatepartRegression
1    0181278de69621582f7c03c262606646               ETS
2    01b5131416d6f8e785376ae9865a43c6  DatepartRegression
3    0256dff8b92a6b86aad6fb6bd46a4362               ETS
4    0275c19262b8f4f6dd4b05c5128a9e67     LastValueNaive
..                                ...              ...
978  fef3b025928a631d356180a81c9933e7      ConstantNaive
979  ff36218e54bf50976a6173d48f07592d    UnivariateMotif
980  ff5b6f2972140826986a8606aae57e47     SectionalMotif
981  ff60f06f5561a901febca5e4fa1c2325               NVAR
982  ffba89a2e955c3e7582eb5480babfa25  AverageValueNaive

                                ModelParameters  \
0    {"regression_model": {"model": "KerasRNN", "mo...
1    {"damped_trend": false, "trend": null, "season...
2    {"regression_model": {"model": "KerasRNN", "mo...
3    {"damped_trend": false, "trend": null, "season...
4                                           {}
..                                             ...
978                          {"constant": 1}
979  {"window": 60, "point_method": "median", "dist...
980  {"window": 10, "point_method": "midhinge", "di...
981  {"k": 1, "ridge_param": 2e-07, "warmup_pts": 5...
982                {"method": "Mean", "window": 98}
```

```
print(validation)
```

```
979  {"window": 60, "point_method": "median", "dist...
980  {"window": 10, "point_method": "midhinge", "di...
981  {"k": 1, "ridge_param": 2e-07, "warmup_pts": 5...
982                {"method": "Mean", "window": 98}

                          TransformationParameters  Ensemble  Runs  \
0    {"fillna": "ffill", "transformations": {"0": "...         0     4
1    {"fillna": "zero", "transformations": {"0": "M...         0     1
2    {"fillna": "rolling_mean", "transformations": ...         0     1
3    {"fillna": "ffill", "transformations": {"0": "...         0     1
4    {"fillna": "pad", "transformations": {"0": "Mi...         0     1
..                                                 ...       ...   ...
978  {"fillna": "fake_date", "transformations": {"0...         0     1
979  {"fillna": "median", "transformations": {"0": ...         0     1
980  {"fillna": "median", "transformations": {"0": ...         0     1
981  {"fillna": "akima", "transformations": {"0": "...         0     1
982  {"fillna": "ffill", "transformations": {"0": "...         0     1

          smape        mae       rmse      made  ...  mle_weighted  \
0     23.630947  23.634833  34.142871  0.486651  ...     11.595317
1     33.911209  34.792222  41.419958  0.577323  ...     10.621817
2    108.875959  83.841615  92.909956  0.849459  ...     81.637525
3     33.172238  25.844056  30.418866  0.498314  ...      9.578967
4     29.031356  23.119749  29.074995  0.459721  ...      6.088955
..          ...        ...        ...       ...  ...           ...
978   46.495894  41.425000  53.711614  0.765179  ...     24.285351
979   27.217443  25.458919  31.719658  0.579635  ...      7.170081
980   53.015293  54.885330  69.623560  1.119342  ...     16.378319
981   24.301448  22.174041  29.652537  0.535637  ...      5.943167
982   43.601549  40.726742  54.728738  0.783192  ...     16.497236
```

```
     imle_weighted  spl_weighted  maxe_weighted  oda_weighted  mqae_weighted  \
0        14.763802      0.381691     131.048730        0.8125      15.435754
1        27.467446      0.372251      81.163975        0.5500      28.819746
2         6.449329      0.486731     145.202480        0.7000      74.628728
3        19.345694      0.371464      67.898919        0.5500      21.430396
4        19.956749      0.196848      80.196551        0.6000      17.220089
..             ...           ...            ...           ...           ...
978      20.386575      0.816947     105.000000        0.7000      31.823529
979      21.211891      0.225357      75.488264        0.5500      19.925925
980      42.253792      0.314164     139.789282        0.5000      40.289254
981      18.944396      0.405387      83.435330        0.7000      15.630511
982      27.470659      0.549053     117.732577        0.7000      28.372771

     containment_weighted  contour_weighted  TotalRuntimeSeconds      Score
0                    0.95              0.70                 18.0  28.103811
1                    1.00              0.75                  1.0  33.926847
2                    0.90              0.40                 17.0  76.660325
3                    0.80              0.75                  1.0  30.972145
4                    0.70              0.75                  1.0  24.917440
..                    ...               ...                  ...        ...
978                  0.00              0.60                  1.0  51.445688
979                  0.85              0.70                  1.0  25.894583
980                  0.60              0.40                  1.0  46.942388
981                  0.15              0.60                  1.0  28.460983
982                  0.35              0.60                  1.0  44.332026

[983 rows x 34 columns]
```

```
# 980  ff5b6f2972140826986a8606aae57e47      SectionalMotif
# 980   53.015293  54.885330  69.623560  1.119342  ...     16.378319
# RMSE value of SectionalMotif comes out to be 69.6
# Accuracy of SectionalMotif = 69.6%
```

Acknowledgment:

I have undertaken this Internship and completed the Internship Report under the guidance of **Mr. Vishwanath Srirangam** (Project Head), **Mr. Srinivasulu Ekambaram** (Project Manager) and **Mrs. Swati Banka** (Project Mentor) and am thankful to them for their guidance and assistance. I am grateful to **NTT DATA Business Solutions** for giving me this opportunity and assistance during my internship at the company. It was an excellent learning experience for me.

GitHub link:

https://github.com/shraddhareddyt/PredictiveAnalysis_MachineLearning