Ticket Prediction ML Model

Internship Project Report
by

SS Madhuvani Petla

24-Jul-22

—

Mentor: Swati Banka

—

NTT DATA

Content

1.Introduction

Use of Machine Learning for the Project?

1. You'll gain valuable insight for future Ticket Predictions

Forecasting gets you into the habit of looking at past and real-time data to predict future demand. And in doing so, you'll be able to anticipate demand fluctuations more effectively. But more than that, it'll give your insight into the company's health and provide you with an opportunity to course-correct or make adjustments.

2. You'll learn from past data

You don't start from scratch after each forecast. Even if your prediction was nowhere close to what ended up coming to pass, it gives you a starting point. It's common to review where and why things didn't happen the way you predicted. Your forecasts should eventually improve. But more than that, you'll get into the habit of reflecting upon past performance as a whole. And self-reflection can be a powerful driver of company growth.

3. It can decrease costs of labor

When done right, anticipating demand will help you tweak your processes to increase efficiency all along the supply chain. Because you're better able to predict what customers will want and when they'll want it, you may also be able to decrease excess inventory levels, thus increasing overall profitability.

Benefits for the current Problem statement or Use case

Since Ticket Prediction is a task that can consume an ample amount of the cost of labor this prediction Model can help perform contingency planning during the challenging financial times.

This makes a better future prediction of the work planning and the cost of time and cost of labor. By categorizing the predicted tickets, we can achieve the requirements.

2.Abstract

This abstract describes about the use case that we've worked on. In the given problem statement the task was to predict the total number of tickets that can be recorded in a particular day of the year or on a particular date. This can be achieved by applying a time series machine learning model on the past data. The time series model mainly takes an overall analysis of the seasonality, trend, irregularity and cyclic nature of the data. There are many time series models that can work on the past data to give future predictions. This report discusses of the different types of models that were used in the analysis and prediction of the number of tickets. The predicted data is then tested with the actual data and the accuracy is then calculated. Out of all the time series models, XGBoost has given the best accuracy which we will be discussing in the report. The given data has been a univariate data with almost no seasonality, trend, irregularity or cyclic nature which made the task tough. The data was first analyzed and the required operations on the data are performed to achieve the best results. After the prediction of data, the predicted value can be further divided or segregated to different categories to give complete analysis for the company about the requirements of the ticket resolution. References are attached in the end of the of the report with provided GitHub link.

3.Problem Definition and Algorithms

3.1. Task Definition

The use case given for the project is to predict the total number of tickets on a given date and further categorize the tickets according to the priority.

Ticketing System

Ticketing tool in SAP are **generally used to handle production support environment**. However, they are some third-party tools that we use to track the issues coming from the end users. Depending on the company and client the tools can vary. SAP Event Ticketing offers us a scalable and innovative platform to provide our customers with the best service possible.

These tickets are then resolved by the technical experts.

The ticketing process normally in SAP is 3 levels.

- High Priority
- Medium Priority
- Low priority

If it is High Priority, you should solve the issues within **2 hrs. or 3 hrs**. If it is Medium Priority, you should solve the issues within **4 to 6 hrs. or 8 hrs**. If it is Low Priority, you can solve the issues within **8 hrs**. or **next day** itself. The time dependencies may depend upon the client to client.

We should now build a machine learning model that predicts the total no of tickets and categorize according to the priority. The past data is used to train the model we are now provided with data that contains 97125 records of data that describes the details of every ticket that has been recorded for the past 2 years roughly from **27-02-2020** to **06-05-2022.**

3.2. Algorithm Definition

Time-series Analysis Machine Learning model

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly.

Time series analysis typically requires a large number of data points to ensure consistency and reliability. An extensive data set ensures you have a representative sample size and that analysis can cut through noisy data. It also ensures that any trends or patterns discovered are not outliers and can account for seasonal variance. Additionally, time series data can be used for forecasting—predicting future data based on historical data.

Why should we use time-series analysis for this use-case?

Time series forecasting is part of Predictive analysis. It can show likely changes in the data, like seasonality or cyclic behavior, which provides a better understanding of data variables and helps forecast better.

In time series data, variations can occur sporadically throughout the data:

- **Functional analysis** can pick out the patterns and relationships within the data to identify notable events.
- **Trend analysis** means determining consistent movement in a certain direction. There are two types of trends: deterministic, where we can find the underlying cause, and stochastic, which is random and unexplainable.
- **Seasonal variation** describes events that occur at specific and regular intervals during the course of a year. Serial dependence occurs when data points close together in time tend to be related.

Time series analysis is used for non-stationary data—things that are constantly fluctuating over time or are affected by time. Industries like finance, retail, and economics frequently use time series analysis because currency and sales are always

changing. Stock market analysis is an excellent example of time series analysis in action, especially with automated trading algorithms.

Time series analysis and forecasting models must define the types of data relevant to answering the business question. Once analysts have chosen the relevant data they want to analyze, they choose what types of analysis and techniques are the best fit.

Models of time series analysis include:

Classification: Identifies and assigns categories to the data.

Curve fitting: Plots the data along a curve to study the relationships of variables within the data.

Descriptive analysis: Identifies patterns in time series data, like trends, cycles, or seasonal variation.

Explanative analysis: Attempts to understand the data and the relationships within it, as well as cause and effect.

Exploratory analysis: Highlights the main characteristics of the time series data, usually in a visual format.

Forecasting: Predicts future data. This type is based on historical trends. It uses the historical data as a model for future data, predicting scenarios that could happen along future plot points.

Intervention analysis: Studies how an event can change the data.

Segmentation: Splits the data into segments to show the underlying properties of the source information.

This current use case uses **Forecasting.**

4.Statistical Analysis

In [2]:
```
df = pd.read_csv("Main_data.csv")
df.head(10)
```

Out[2]:

	Zone	Dealer Code	Dealer Name	Transaction No.	Description	Status	Posting Date	IRT	CATEGORY	IRTSTATUS	MPT	MPTSTATUS	Changed On	TRANSACTIONTYPE
0	NaN	NaN	NaN	2000095635	Medium	Open	06-05-2022	0	Sales	NaN	0	NaN	06-05-2022	MG_MOTORS_Incident
1	NaN	NaN	NaN	1000003085	Very High	Open	06-05-2022	0	NaN	NaN	0	NaN	06-05-2022	MG Parts Support
2	NaN	NaN	NaN	1000003086	Medium	Open	06-05-2022	0	NaN	NaN	0	NaN	06-05-2022	MG Parts Support
3	NaN	NaN	NaN	2000095644	Very High	Open	06-05-2022	0	After Sales	NaN	0	NaN	06-05-2022	MG_MOTORS_Incident
4	NaN	NaN	NaN	2000095646	High	Open	06-05-2022	0	Sales	NaN	0	NaN	06-05-2022	MG_MOTORS_Incident
5	NaN	NaN	NaN	1000000756	Medium	Open	28-09-2021	999	NaN	IRT Exceeded	999	MPT Exceeded	21-10-2021	MG Parts Support
6	NaN	NaN	NaN	1000000761	Medium	Open	29-09-2021	999	NaN	IRT Exceeded	999	MPT Exceeded	21-10-2021	MG Parts Support

This is the given data to work on.

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97125 entries, 0 to 97124
Data columns (total 17 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   Transaction No.  97125 non-null  int64
 4   Description      97118 non-null  object
 5   Status           97125 non-null  object
 6   Posting Date     97125 non-null  object
 7   IRT              97125 non-null  int64
 8   CATEGORY         92095 non-null  object
 9   IRTSTATUS        3790 non-null   object
 10  MPT              97125 non-null  int64
 11  MPTSTATUS        10610 non-null  object
 12  Changed On       97125 non-null  object
 13  TRANSACTIONTYPE  97124 non-null  object
 14  Created By       97125 non-null  object
 15  Reported By      90933 non-null  object
 16  Object GUID      97125 non-null  int64
dtypes: int64(4), object(13)
memory usage: 12.6+ MB
```

```
cdf = df[["Description","Status","Posting Date","IRT","MPT","TRANSACTIONTYPE"]]
cdf.head(15)
```

	Description	Status	Posting Date	IRT	MPT	TRANSACTIONTYPE
0	Medium	Open	5/6/2022	0	0	MG_MOTORS_Incident
1	Very High	Open	5/6/2022	0	0	MG Parts Support
2	Medium	Open	5/6/2022	0	0	MG Parts Support
3	Very High	Open	5/6/2022	0	0	MG_MOTORS_Incident
4	High	Open	5/6/2022	0	0	MG_MOTORS_Incident
5	Medium	Open	9/28/2021	999	999	MG Parts Support
6	Medium	Open	9/29/2021	999	999	MG Parts Support
7	Very High	Open	5/6/2022	0	0	MG_MOTORS_Incident
8	Low	In Process	6/15/2020	0	0	NaN
9	Change Request	In Process	6/4/2020	0	999	MG_MOTORS_Incident
10	Service Request	In Process	5/3/2022	0	9	MG_MOTORS_Incident

These are the Features that can be used for the classification of the tickets

```
In [5]: df["Description"].unique()
Out[5]: array(['Medium', 'Very High', 'High', 'Low', 'Change Request',
               'Service Request', nan], dtype=object)

In [6]: df["Status"].unique()
Out[6]: array(['Open', 'In Process', 'Awaiting For User Action',
               'Proposed Solution', 'Closed', 'Acknowledged', 'Withdrawn',
               'On Hold', 'Change Request'], dtype=object)

In [7]: df["Zone"].unique()
Out[7]: array([nan, 'ITELLI', 'MG', 'SOUTH', 'EAST', 'NORTH', 'NCR', 'WEST'],
              dtype=object)

In [8]: df["TRANSACTIONTYPE"].unique()
Out[8]: array(['MG_MOTORS_Incident', 'MG Parts Support', nan], dtype=object)

In [9]: df["MPT"].unique().shape
Out[9]: (869,)
```

```
In [10]:
1 df["Posting Date"] = pd.to_datetime(df["Posting Date"])
2 df["Changed On"] = pd.to_datetime(df["Changed On"])
```

Changing the string formatted dates into actual date-time format using the function to_datetime in pandas' library

```
In [12]:
1 date_ticket_count = df.groupby(["Posting Date"]).agg({"Status" : "count"})
2 date_ticket_count
```

Grouping the whole data into another data-frame consisting of "Posting Date" as the index for that data-frame

Out[12]:

Posting Date	Status
2020-02-27	1
2020-02-28	14
2020-02-29	4
2020-03-02	1
2020-03-03	2
...	...
2022-05-02	83
2022-05-03	74
2022-05-04	116
2022-05-05	125
2022-05-06	95

794 rows × 1 columns

Resulting to the Above data

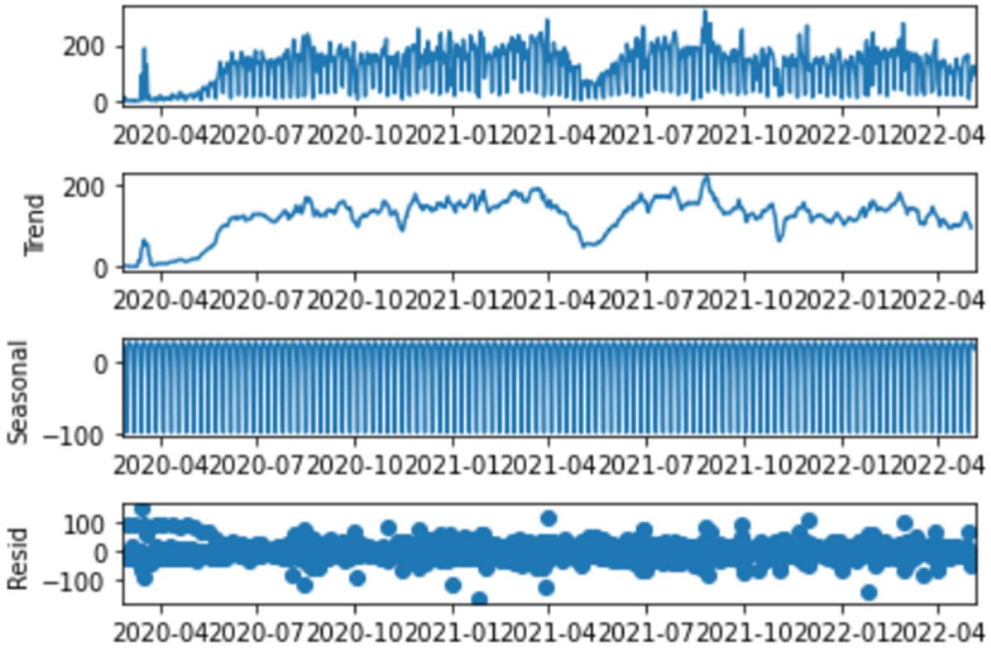
```
In [13]:  1  date_ticket_count.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 794 entries, 2020-02-27 to 2022-05-06
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Status  794 non-null    int64
dtypes: int64(1)
memory usage: 12.4 KB
```

Info of the group by data

```
In [15]:  1  from statsmodels.tsa.seasonal import seasonal_decompose
          2  result = seasonal_decompose(date_ticket_count)
          3  result.plot()
```

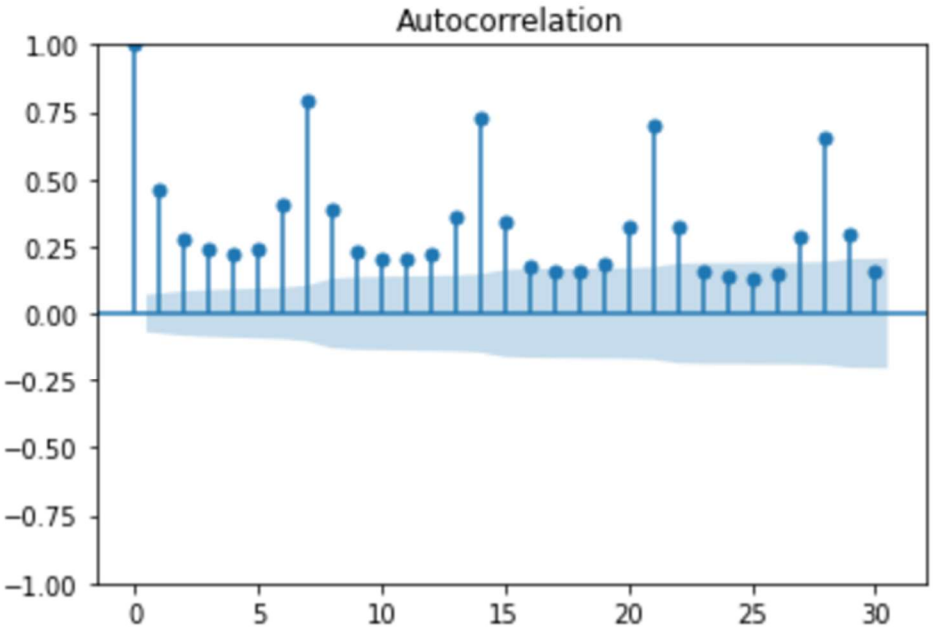
To find the Seasonality, trend, cyclic nature we import a library called statsmodels There is a function **seasonal_decompose** from the class of **statsmodels.tsa.seasonal** that is used to plot the visualization of the mentioned components.



Auto correlation of the data is plotted as below

In [16]:
```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(date_ticket_count)
```

Out[16]:



In [17]:
```
Description = df.groupby('Posting Date')['Description'].value_counts().unstack().fillna(0)
```

In [18]:
```
Description.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 794 entries, 2020-02-27 to 2022-05-06
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Change Request   794 non-null    float64
 1   High             794 non-null    float64
 2   Low              794 non-null    float64
 3   Medium           794 non-null    float64
 4   Service Request  794 non-null    float64
 5   Very High        794 non-null    float64
dtypes: float64(6)
memory usage: 43.4 KB
```

Finding the count of every individual and unique values of the “Description”, “Zone”, “Status”, “Transaction Type” in the data

```
In [19]: 1 Description = Description.astype(int)
```

```
In [20]: 1 Description.head(20)
```

Out[20]:

Description Posting Date	Change Request	High	Low	Medium	Service Request	Very High
2020-02-27	0	0	0	1	0	0
2020-02-28	0	1	0	13	0	0
2020-02-29	0	0	0	3	0	1
2020-03-02	0	0	0	1	0	0
2020-03-03	0	1	0	0	0	1
2020-03-04	0	1	0	0	0	0
2020-03-05	0	1	0	1	0	0
2020-03-06	0	0	0	2	0	2
2020-03-09	0	1	0	0	0	3
2020-03-10	0	1	0	0	0	0
2020-03-13	0	0	0	2	0	0

```
In [137]: 1 Zone = df.groupby('Posting Date')['Zone'].value_counts().unstack().fillna(0).astype(int)
```

```
In [138]: 1 Zone.head(10)
```

Out[138]:

Zone Posting Date	EAST	ITELLI	MG	NCR	NORTH	SOUTH	WEST
2020-02-27	0	1	0	0	0	0	0
2020-02-28	0	7	0	0	0	7	0
2020-02-29	0	0	0	0	0	4	0
2020-03-02	0	0	0	0	0	1	0
2020-03-03	0	0	0	0	0	2	0
2020-03-04	0	0	0	0	0	1	0
2020-03-05	0	0	0	0	0	2	0
2020-03-06	0	0	0	0	0	4	0
2020-03-09	0	0	0	0	0	4	0
2020-03-10	0	0	0	0	0	1	0

In [136]:
```
1 Zone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 794 entries, 2020-02-27 to 2022-05-06
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   EAST    684 non-null    float64
 1   ITELLI  697 non-null    float64
 2   MG      622 non-null    float64
 3   NCR     728 non-null    float64
 4   NORTH   745 non-null    float64
 5   SOUTH   726 non-null    float64
 6   WEST    740 non-null    float64
dtypes: float64(7)
memory usage: 49.6 KB
```

In [24]:
```
1 TransactionType = df.groupby('Posting Date')['TRANSACTIONTYPE'].value_counts().unstack().fillna(0).astype(int)
2 TransactionType.head()
```

Out[24]:

TRANSACTIONTYPE	MG Parts Support	MG_MOTORS_Incident
Posting Date		
2020-02-27	0	1
2020-02-28	0	14
2020-02-29	0	4
2020-03-02	0	1
2020-03-03	0	2

In [25]:
```
1 TransactionType.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 794 entries, 2020-02-27 to 2022-05-06
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   MG Parts Support    794 non-null    int32
 1   MG_MOTORS_Incident  794 non-null    int32
dtypes: int32(2)
memory usage: 12.4 KB
```

```
In [30]: 1 Refined_data = pd.concat([Description, Zone, TransactionType, date_ticket_count], axis = 1, join = "inner")

In [31]: 1 Refined_data.reset_index(inplace = True)

In [32]: 1 Refined_data
```

Out[32]:

	Posting Date	Change Request	High	Low	Medium	Service Request	Very High	EAST	ITELLI	MG	NCR	NORTH	SOUTH	WEST	MG Parts Support	MG_MOTORS_Incident	Status
0	2020-02-27	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1
1	2020-02-28	0	1	0	13	0	0	0	7	0	0	0	7	0	0	14	14
2	2020-02-29	0	0	0	3	0	1	0	0	0	0	0	4	0	0	4	4
3	2020-03-02	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1
4	2020-03-03	0	1	0	0	0	1	0	0	0	0	0	2	0	0	2	2

Made a data with division of data according to the categories summing up to the total tickets.

```
In [33]: 1 Refined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794 entries, 0 to 793
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Posting Date         794 non-null    datetime64[ns]
 1   Change Request       794 non-null    int32
 2   High                 794 non-null    int32
 3   Low                  794 non-null    int32
 4   Medium               794 non-null    int32
 5   Service Request      794 non-null    int32
 6   Very High            794 non-null    int32
 7   EAST                 794 non-null    int32
 8   ITELLI               794 non-null    int32
 9   MG                   794 non-null    int32
 10  NCR                  794 non-null    int32
 11  NORTH                794 non-null    int32
 12  SOUTH                794 non-null    int32
 13  WEST                 794 non-null    int32
 14  MG Parts Support     794 non-null    int32
 15  MG_MOTORS_Incident   794 non-null    int32
 16  Status               794 non-null    int64
dtypes: datetime64[ns](1), int32(15), int64(1)
memory usage: 59.1 KB
```

5.Machine Learning Models Used

TIME SERIES MODELS

11 different classical time series forecasting methods are:

1. Autoregression (AR)
2. Moving Average (MA)
3. Autoregressive Moving Average (ARMA)
4. Autoregressive Integrated Moving Average (ARIMA)
5. Seasonal Autoregressive Integrated Moving-Average (SARIMA)
6. Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
7. Vector Autoregression (VAR)
8. Vector Autoregression Moving-Average (VARMA)
9. Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
10. Simple Exponential Smoothing (SES)
11. Holt Winter's Exponential Smoothing (HWES)

These come under the classic Time series model apart from this we have few other prediction models like:

- Prophet
- XGBoost
- LSTM
- TBATS
- Classic Linear Regression

In the current Project we will be working on 3 different models

1. ARIMA
2. Linear Regression
3. XGBOOST

In the further analysis we will find out the accuracy, errors like Root mean squared error, mean absolute error, R^2 score that help in the working and performance of the model.

5.1. ARIMA MODEL

In [34]:
```
from sklearn import linear_model

X = date_ticket_count.iloc[:,0]
Y = date_ticket_count.iloc[:,-1]
```

In [35]:
```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

Let us first split the data that grouped using the “Posting Date”. This splitting of data can be done by the function “train_test_split” from the library sklearn.model_selection

In [38]:
```
from statsmodels.tsa.ar_model import AutoReg

model = AutoReg(X_train, lags=1)
model_fit = model.fit()
```

```
C:\Users\ndbsguest\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarnin
g: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\ndbsguest\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarnin
g: A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

In [39]:
```
from pmdarima.arima.utils import ndiffs
```

In [40]:
```
z=date_ticket_count["Status"]
print(ndiffs(z,test='kpss'))
print(ndiffs(z,test='adf'))
print(ndiffs(z,test='pp'))
```

```
1
1
0
```

In [41]:
```
from pmdarima import auto_arima
```

In [42]:
```
from inspect import trace
model=auto_arima(date_ticket_count["Status"],start_p=1,start_q=1,test="adf",
                 max_p=10,max_q=10,
                 m=1,
                 d=None,
                 seasonal=True,
                 start_P=0,
                 D=0,
                 trace=True,
                 error_action='ignore',
                 suppress_warnings=True,
                 stepwise=True
                )
```

By installing the pmdarima using the command “pip install pmdarima” we can use the model.
Automatically discover the optimal order for an ARIMA model. The auto-ARIMA process seeks to identify the most optimal parameters for an ARIMA model, settling on a single fitted ARIMA model. This process is based on the commonly-used R function, forecast: auto_arima

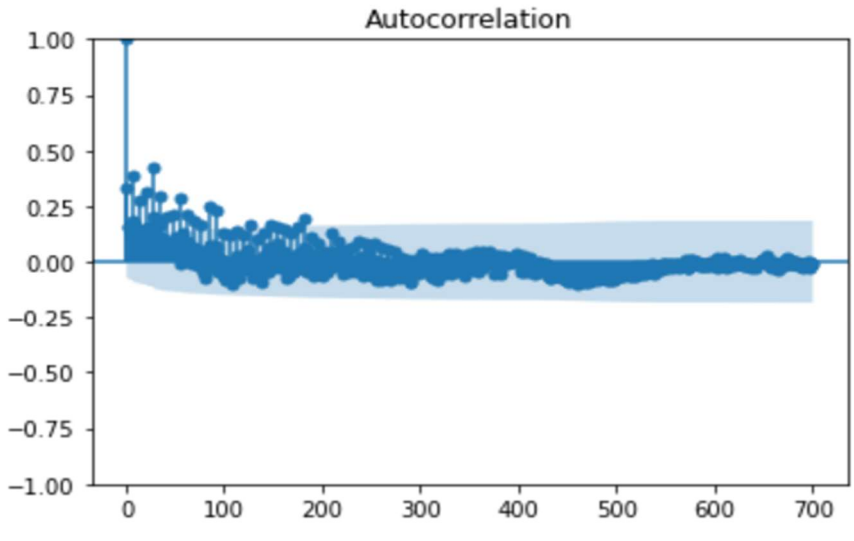
```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=8730.028, Time=0.32 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=9098.601, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=9008.461, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=8749.003, Time=0.13 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=9096.603, Time=0.01 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=8713.958, Time=0.35 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=8947.068, Time=0.07 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=8697.796, Time=0.48 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=8913.063, Time=0.10 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=8660.458, Time=0.80 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=8882.978, Time=0.11 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=8561.826, Time=0.91 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=8779.812, Time=0.17 sec
 ARIMA(6,1,1)(0,0,0)[0] intercept   : AIC=8160.422, Time=1.80 sec
 ARIMA(6,1,0)(0,0,0)[0] intercept   : AIC=8178.239, Time=1.09 sec
 ARIMA(7,1,1)(0,0,0)[0] intercept   : AIC=8156.010, Time=2.29 sec
 ARIMA(7,1,0)(0,0,0)[0] intercept   : AIC=8156.546, Time=0.94 sec
 ARIMA(8,1,1)(0,0,0)[0] intercept   : AIC=8157.206, Time=2.69 sec
 ARIMA(7,1,2)(0,0,0)[0] intercept   : AIC=8161.949, Time=2.29 sec
 ARIMA(6,1,2)(0,0,0)[0] intercept   : AIC=8157.193, Time=2.23 sec
 ARIMA(8,1,0)(0,0,0)[0] intercept   : AIC=8155.659, Time=1.37 sec
 ARIMA(9,1,0)(0,0,0)[0] intercept   : AIC=8157.478, Time=1.74 sec
 ARIMA(9,1,1)(0,0,0)[0] intercept   : AIC=8151.766, Time=2.82 sec
 ARIMA(10,1,1)(0,0,0)[0] intercept   : AIC=8149.184, Time=3.38 sec
 ARIMA(10,1,0)(0,0,0)[0] intercept   : AIC=8157.780, Time=1.73 sec
 ARIMA(10,1,2)(0,0,0)[0] intercept   : AIC=8150.229, Time=3.97 sec
 ARIMA(9,1,2)(0,0,0)[0] intercept   : AIC=8150.712, Time=3.34 sec
 ARIMA(10,1,1)(0,0,0)[0]             : AIC=8147.387, Time=1.99 sec
 ARIMA(9,1,1)(0,0,0)[0]             : AIC=8150.012, Time=1.62 sec
 ARIMA(10,1,0)(0,0,0)[0]             : AIC=8155.920, Time=0.99 sec
 ARIMA(10,1,2)(0,0,0)[0]             : AIC=8133.978, Time=2.48 sec
 ARIMA(9,1,2)(0,0,0)[0]             : AIC=8147.924, Time=2.12 sec
 ARIMA(10,1,3)(0,0,0)[0]             : AIC=8150.331, Time=2.48 sec
 ARIMA(9,1,3)(0,0,0)[0]             : AIC=8120.735, Time=2.64 sec
 ARIMA(8,1,3)(0,0,0)[0]             : AIC=8135.418, Time=1.90 sec
 ARIMA(9,1,4)(0,0,0)[0]             : AIC=8150.544, Time=1.93 sec
 ARIMA(8,1,2)(0,0,0)[0]             : AIC=8128.898, Time=1.71 sec
 ARIMA(8,1,4)(0,0,0)[0]             : AIC=8155.419, Time=2.23 sec
 ARIMA(10,1,4)(0,0,0)[0]             : AIC=8112.164, Time=4.74 sec
 ARIMA(10,1,5)(0,0,0)[0]             : AIC=8067.212, Time=3.66 sec
 ARIMA(9,1,5)(0,0,0)[0]             : AIC=8078.540, Time=2.67 sec
 ARIMA(10,1,6)(0,0,0)[0]             : AIC=8085.637, Time=3.10 sec
 ARIMA(9,1,6)(0,0,0)[0]             : AIC=8077.054, Time=2.59 sec
 ARIMA(10,1,5)(0,0,0)[0] intercept   : AIC=8076.790, Time=4.49 sec

Best model:  ARIMA(10,1,5)(0,0,0)[0]
Total fit time: 78.574 seconds
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  794
Model:                 SARIMAX(3, 0, 2)   Log Likelihood               -4401.325
Date:                  Thu, 23 Jun 2022   AIC                           8814.650
Time:                          23:44:02   BIC                           8842.712
Sample:                               0   HQIC                          8825.434
                                  - 794
Covariance Type:                    opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
ar.L1          0.3271      0.064      5.150      0.000       0.203       0.452
ar.L2          0.7756      0.083      9.297      0.000       0.612       0.939
ar.L3         -0.1033      0.045     -2.314      0.021      -0.191      -0.016
ma.L1         -0.0640      0.054     -1.186      0.236      -0.170       0.042
ma.L2         -0.8565      0.054    -15.930      0.000      -0.962      -0.751
sigma2      3805.5136    224.109     16.981      0.000    3366.269    4244.759
==========================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                22.61
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               0.97   Skew:                            -0.39
Prob(H) (two-sided):                  0.79   Kurtosis:                         2.74
==========================================================================================

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

In [31]:

```
1 from statsmodels.graphics.tsaplots import plot_acf
2 plot_acf(dfss["T"],lags=700)
```

Out[31]:



Out[47]:

	date	T	forecast	lower	higher
635	2021-11-29	196	107.892828	-15.216611	231.002268
636	2021-11-30	268	132.323758	4.082879	260.564636
637	2021-12-01	160	124.030539	-4.269712	252.330789
638	2021-12-02	161	130.888333	1.804054	259.972612
639	2021-12-03	173	125.217514	-3.869052	254.304081
...	...	...	...	...	...
789	2022-11-04	112	127.784293	-21.266387	276.834973
790	2022-12-01	156	127.784293	-21.387472	276.956059
791	2022-12-02	115	127.784293	-21.508459	277.077046
792	2022-12-03	169	127.784293	-21.629348	277.197935
793	2022-12-04	95	127.784293	-21.750140	277.318727

159 rows × 5 columns

In [48]:
```
1 from pmdarima.metrics import smape
```

In [49]:
```
1
2 mpe=smape(testes["T"],testes["forecast"])
3 print(mpe)
```

```
42.06238389235066
```

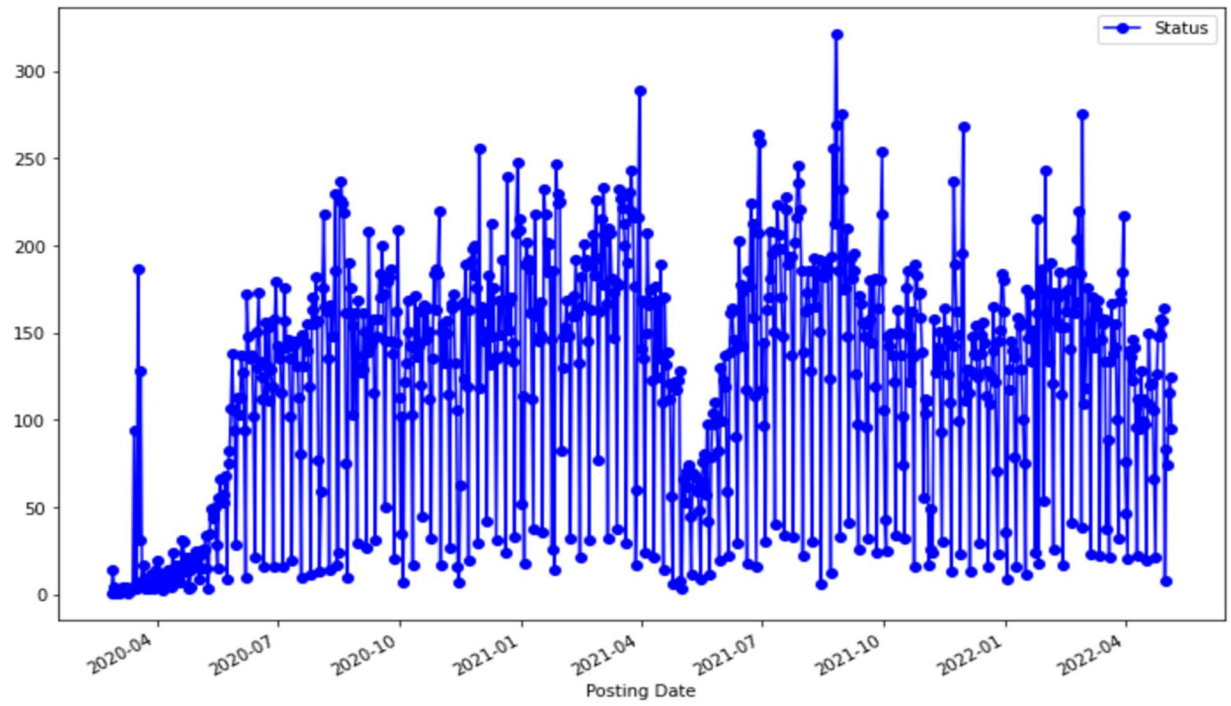
5.2. Linear Regression

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

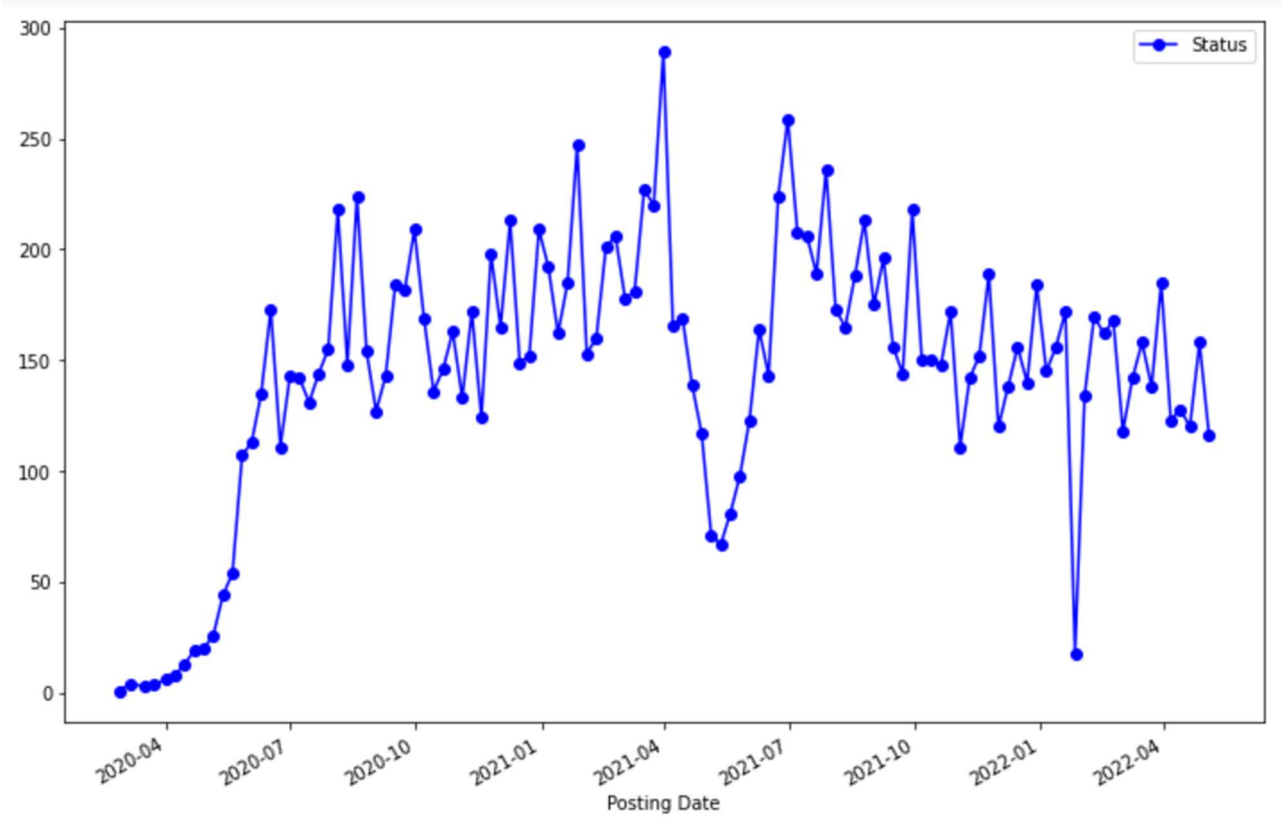
In [66]:
```
d_ticket_count['Lag'] = d_ticket_count['Status'].shift(1)
d_ticket_count = d_ticket_count.reindex(columns=['Status', 'Lag'])
```

In [50]:
```
# SES example

import matplotlib.pyplot as plt
#from statsmodels.tsa.holtwinters import SimpleExpSmoothing

#es = SimpleExpSmoothing(dt_ticket_count)
#es.fit(smoothing_level=0.2,optimized=False)

x = ['blue', 'black', 'red', 'green', 'yellow', 'purple', 'brown']

dt_ticket_count.plot(marker='o', color='blue', figsize=(12,8), legend=True)
for i in range(7):
    dt_ticket_count.iloc[i::7,:].plot(marker='o', color = x[i], figsize=(12,8), legend=True)


```

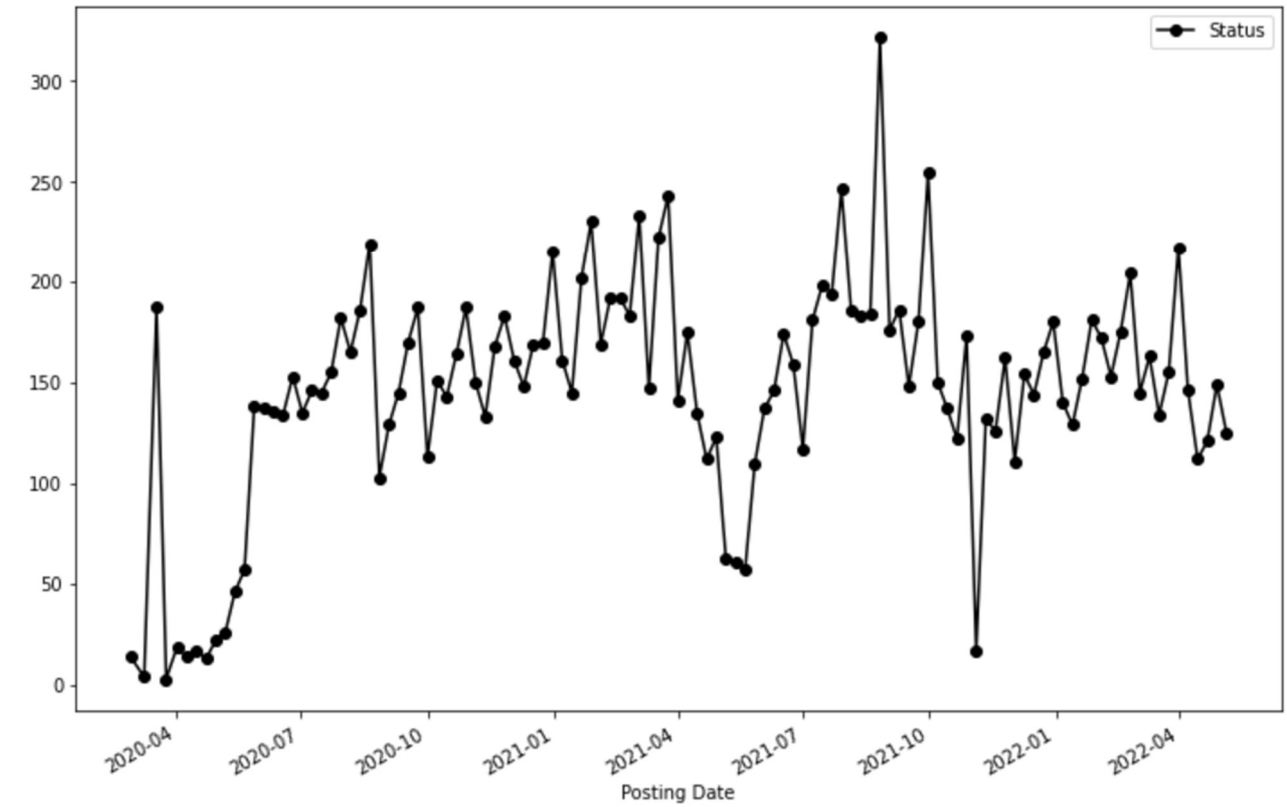
The below visualization is the entire visualization of total number of tickets



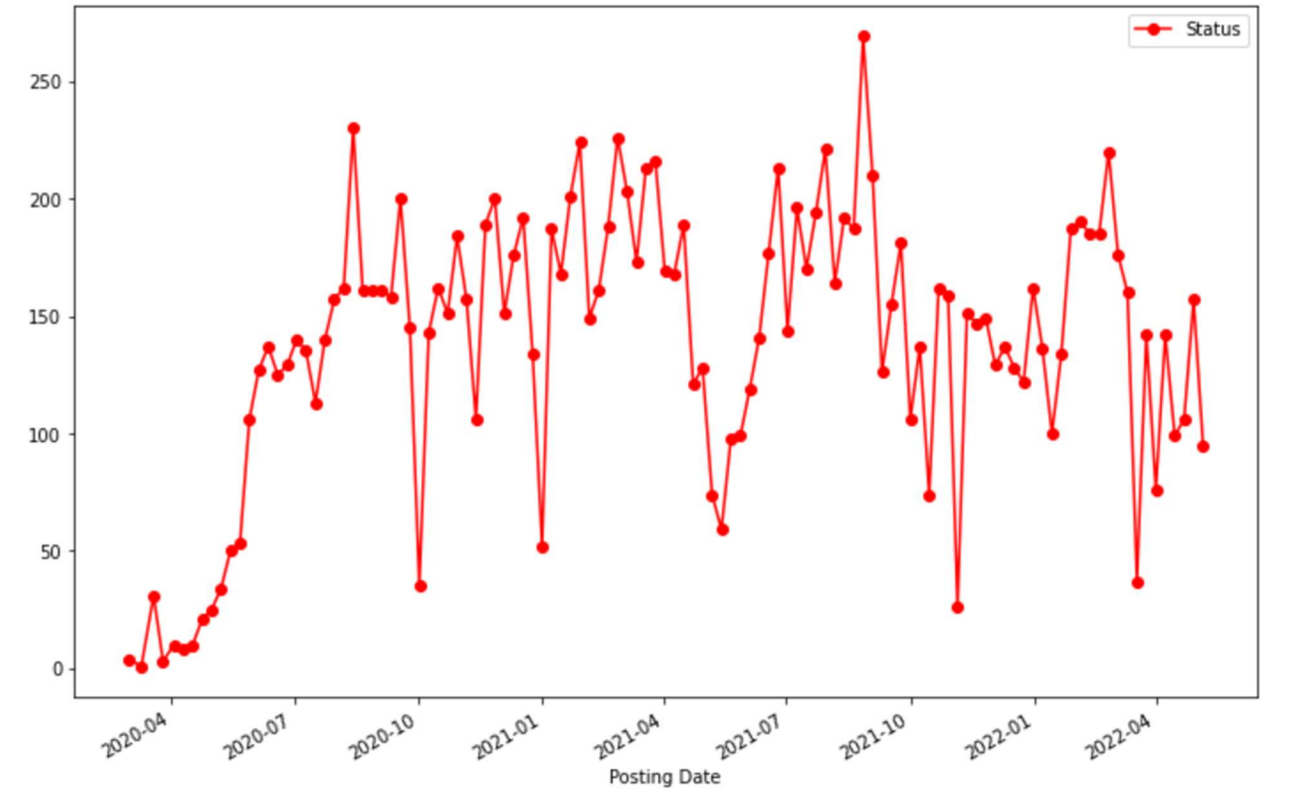
Wednesday analysis



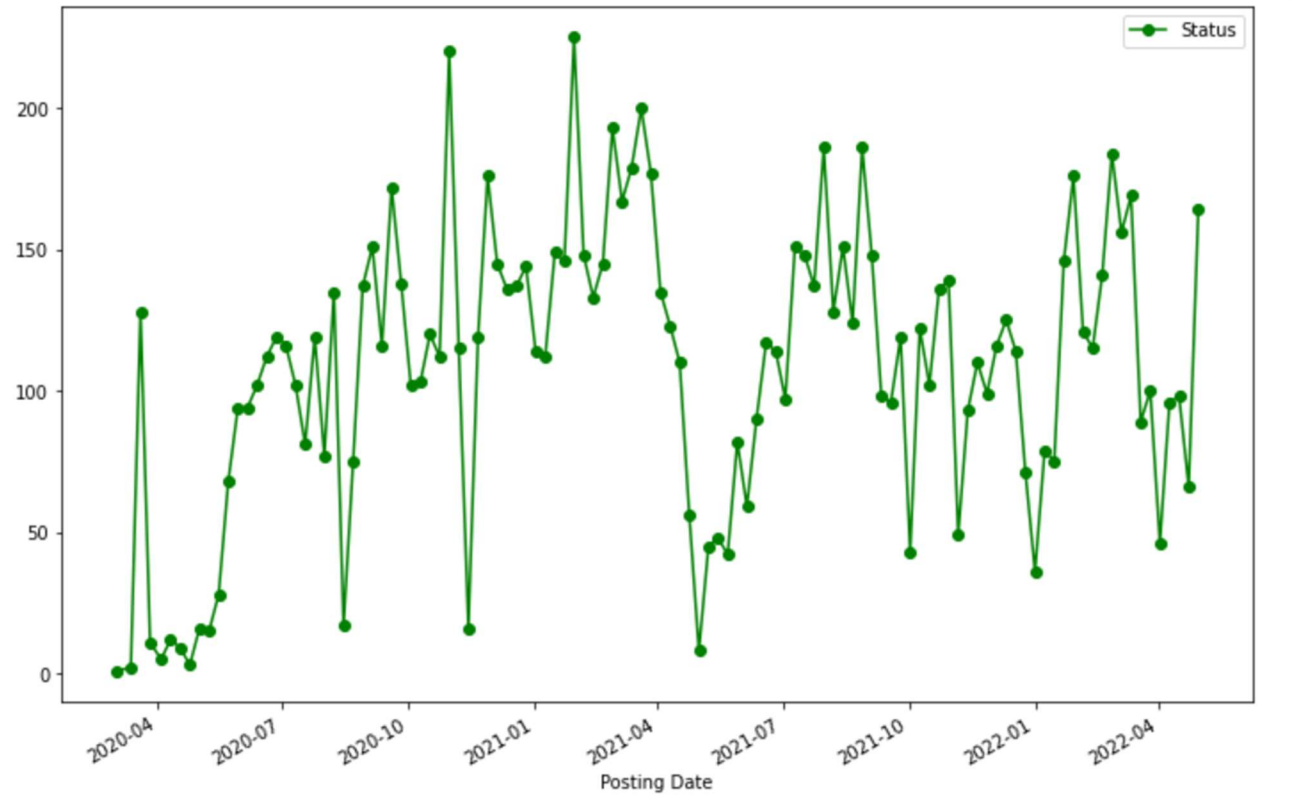
Thursday analysis



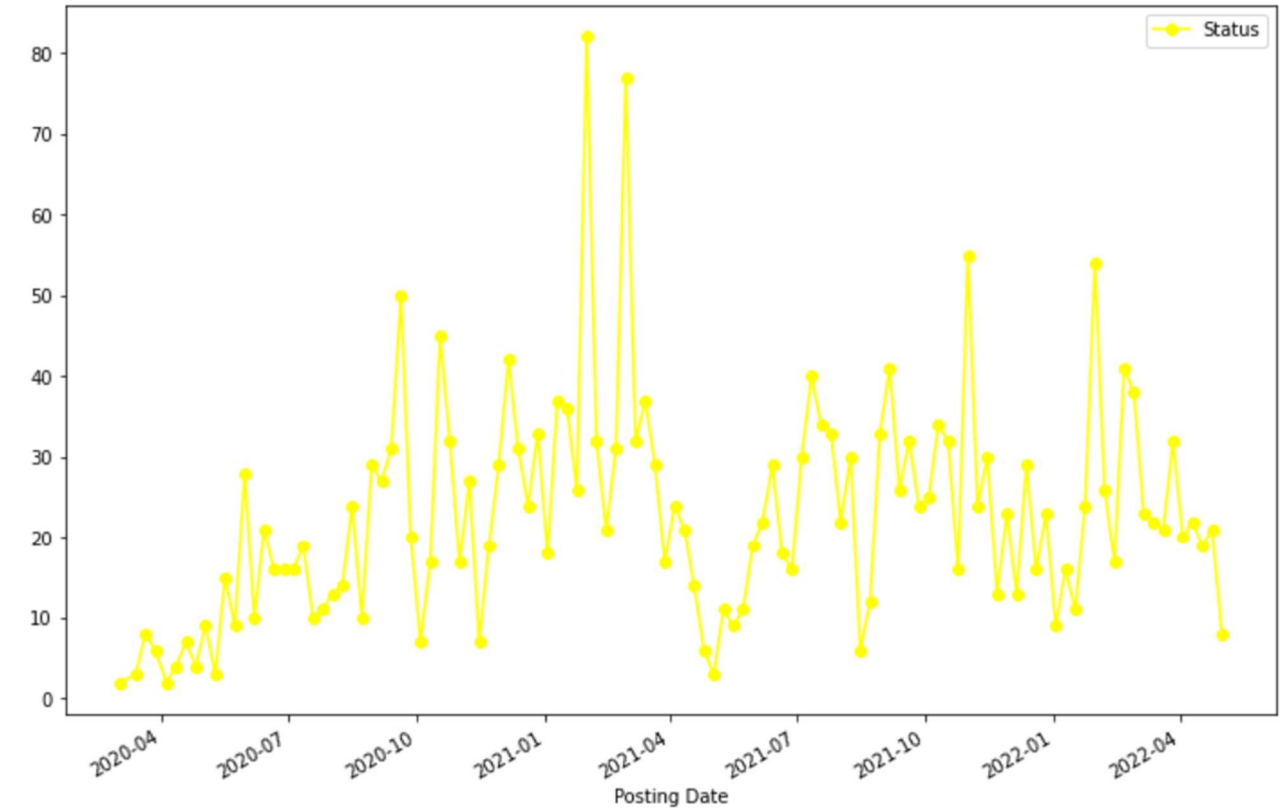
Friday analysis



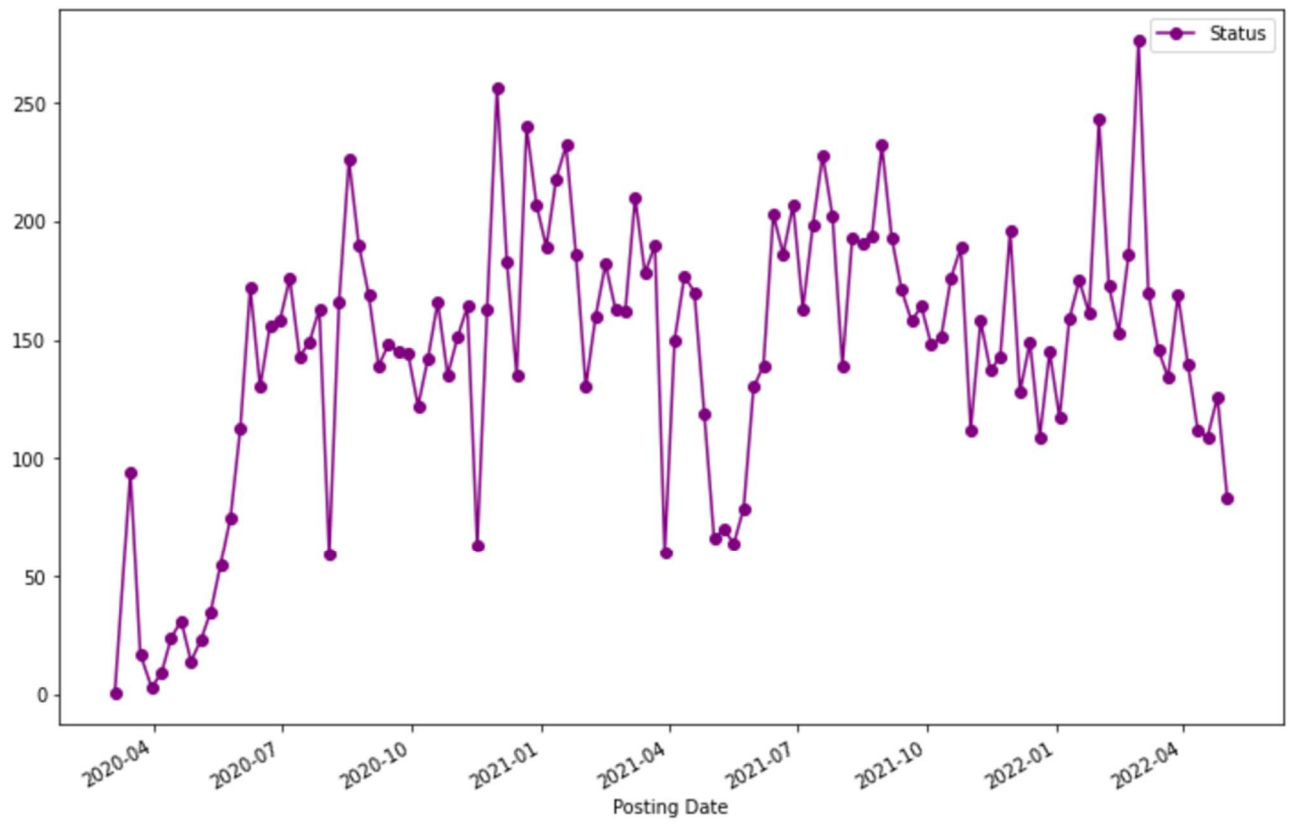
Saturday analysis



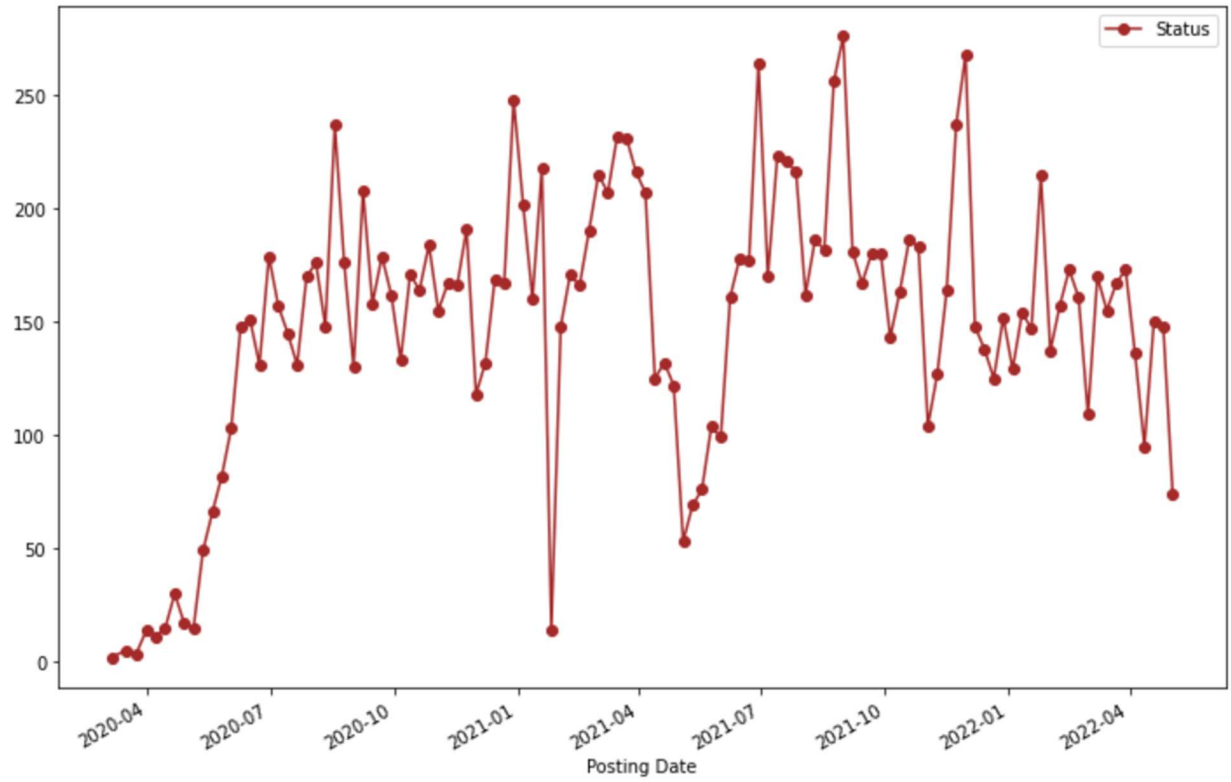
Sunday analysis



Monday analysis



Tuesday analysis



After the day wise analysis, we can see that Sundays have the least number of tickets.

In [51]:
```
for i in range(7):
    print(dt_ticket_count.iloc[i::7,:].mean(),'\n')
```

```
Status    145.842105
dtype: float64

Status    146.078947
dtype: float64

Status    138.22807
dtype: float64

Status    107.628319
dtype: float64

Status    22.699115
dtype: float64

Status    145.778761
dtype: float64

Status    149.451327
dtype: float64
```

```
In [52]: 1 d_ticket_count = dt_ticket_count.drop(dt_ticket_count.index[4::7])
```

```
In [54]: 1 d_ticket_count
```

Out[54]:

Posting Date	Status
2020-02-27	1
2020-02-28	14
2020-02-29	4
2020-03-02	1
2020-03-04	1
...	...
2022-05-02	83
2022-05-03	74
2022-05-04	116
2022-05-05	125
2022-05-06	95

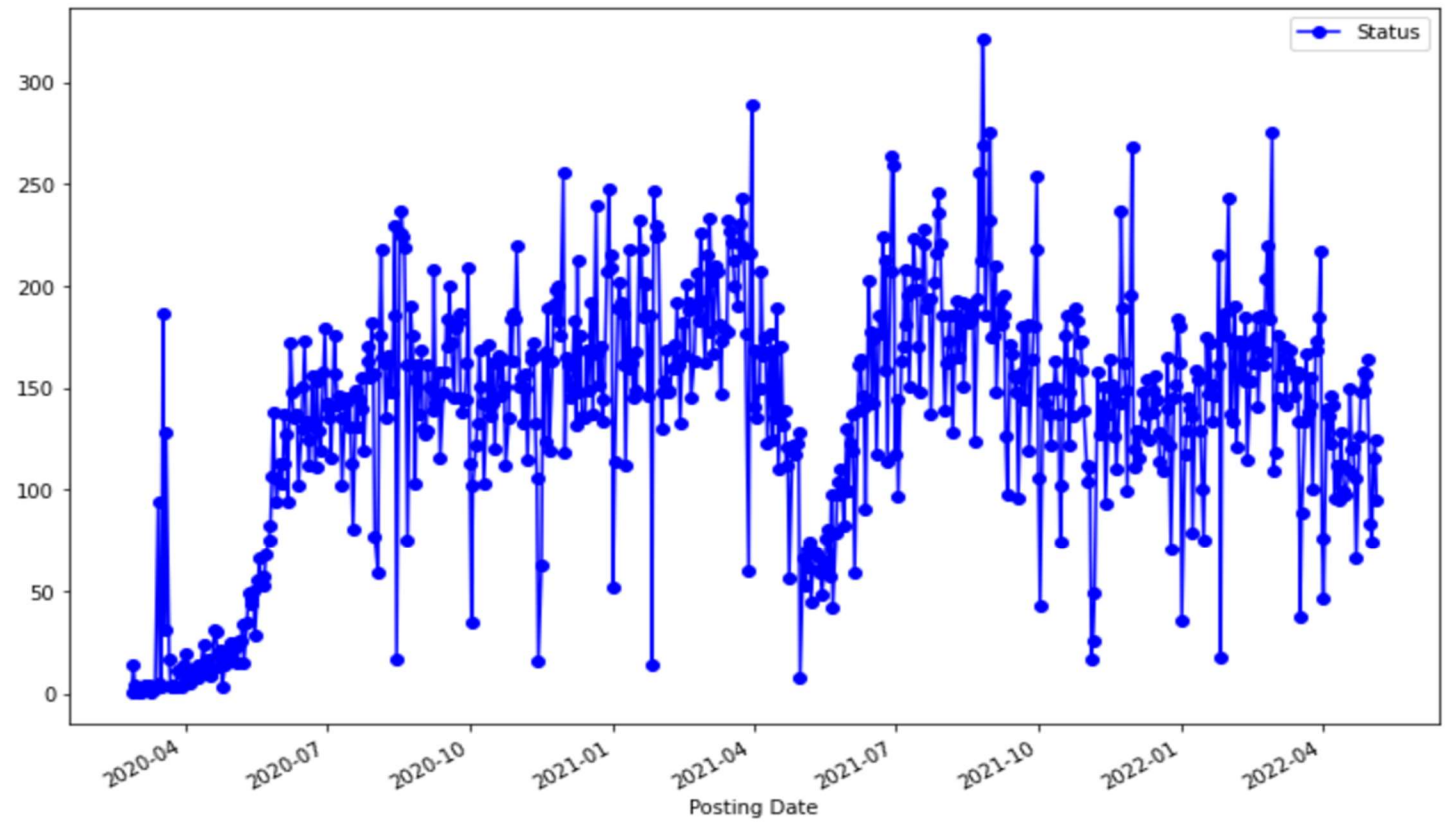
681 rows × 1 columns

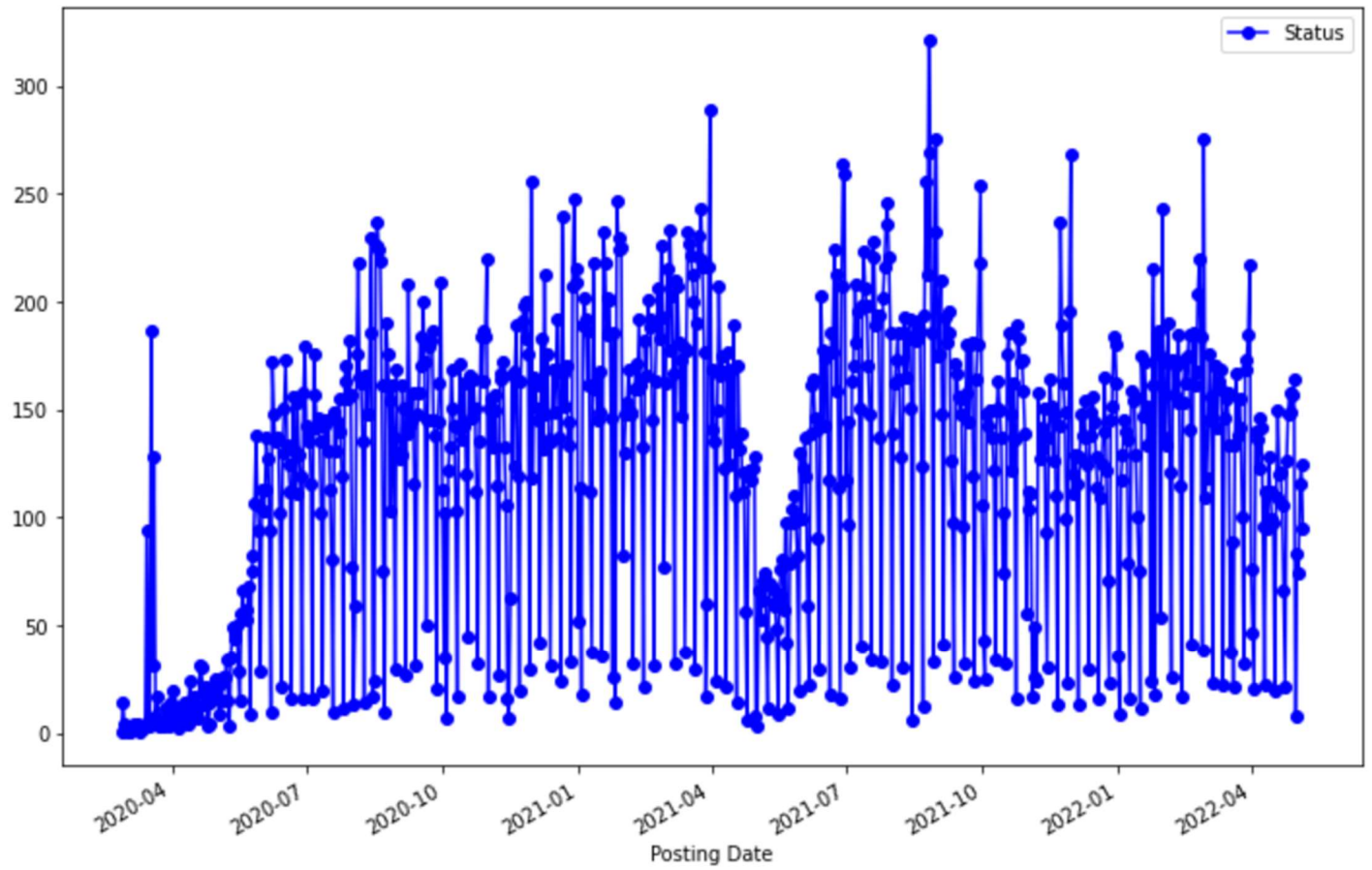
```
In [56]: 1 dt_ticket_count
```

Out[56]:

Posting Date	Status
2020-02-27	1
2020-02-28	14
2020-02-29	4
2020-03-02	1
2020-03-03	2
...	...
2022-05-02	83
2022-05-03	74
2022-05-04	116
2022-05-05	125
2022-05-06	95

794 rows × 1 columns





Now we will work on the data that do not contain Sunday

In [66]:
```
1 d_ticket_count['Lag'] = d_ticket_count['Status'].shift(1)
2 d_ticket_count = d_ticket_count.reindex(columns=['Status', 'Lag'])
```

In [67]:
```
1 d_ticket_count
```

Out[67]:

Posting Date	Status	Lag
2020-02-27	1	NaN
2020-02-28	14	1.0
2020-02-29	4	14.0
2020-03-02	1	4.0
2020-03-04	1	1.0
...	...	...
2022-05-02	83	164.0
2022-05-03	74	83.0
2022-05-04	116	74.0
2022-05-05	125	116.0
2022-05-06	95	125.0

681 rows × 2 columns

In [68]:
```
from sklearn.linear_model import LinearRegression

X = d_ticket_count.loc[:, ['Lag']]
X.dropna(inplace=True)  # drop missing values in the feature set
y = d_ticket_count.loc[:, 'Status']  # create the target
y, X = y.align(X, join='inner')  # drop corresponding values in target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = pd.Series(model.predict(X_test), index=X_test.index)
```

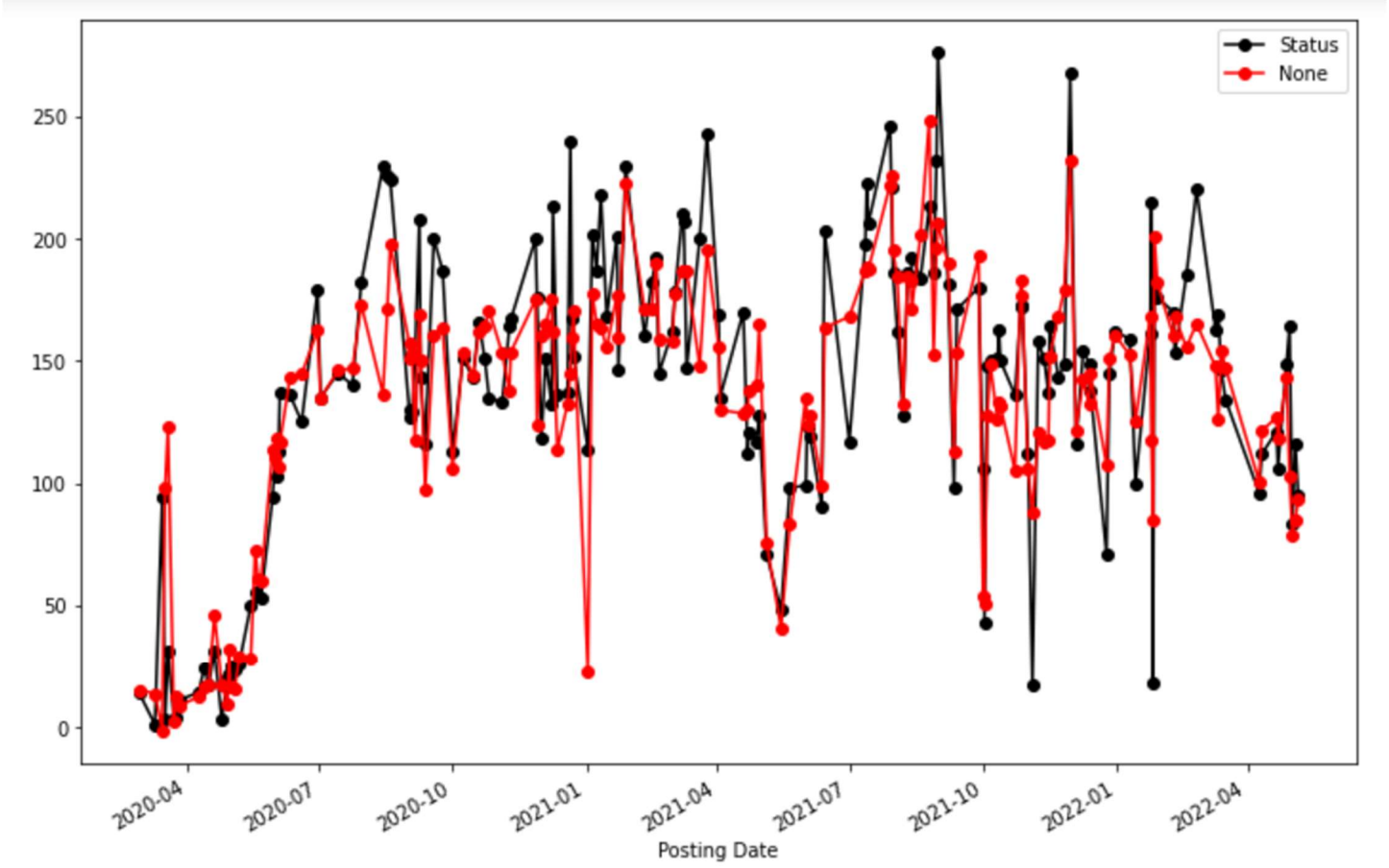
In [69]:
```
from sklearn.metrics import r2_score

r2_score(y_test, y_pred)
```

Out[69]: 0.5353954227813097

We got an accuracy of 53.53% with the linear regression model

In [102]:
```
y_test.plot(marker='o', color='black', figsize=(12,8), legend=True)
y_pred.plot(marker='o', color='red', figsize=(12,8), legend=True)
```

Out[102]: <AxesSubplot:xlabel='Posting Date'>



```
In [72]:  1 y_test
```

```
Out[72]: Posting Date
         2020-07-15    131
         2022-01-24    161
         2021-02-25    183
         2020-09-04    161
         2021-03-05    203
                       ...
         2020-06-04    137
         2020-06-20    112
         2021-08-04    173
         2020-05-21     57
         2022-02-10    153
         Name: Status, Length: 170, dtype: int64
```

```
In [73]:  1 y_pred.astype(int)
```

```
Out[73]: Posting Date
         2020-07-15    143
         2022-01-24    144
         2021-02-25    183
         2020-09-04    133
         2021-03-05    200
                       ...
         2020-06-04    122
         2020-06-20    130
         2021-08-04    154
         2020-05-21     84
         2022-02-10    159
         Length: 170, dtype: int32
```

They above represented data is the actual and predicted data for the linear regression model.

```
In [74]:  1 from sklearn import metrics
          2
          3 print(metrics.mean_absolute_error(y_test, y_pred))
          4 print(metrics.mean_squared_error(y_test, y_pred))
          5
```

```
30.842787793844025
1720.7137563258402
```

5.3. XG Boost

In machine learning, **gradient boosting** is an algorithm that helps in performing regression and classification tasks. Using the ensemble of weak prediction models, gradient boosting helps us in making predictions. Examples of weak models can be decision trees. Ensembled models using weak tree learners can be considered gradient-boosted trees. Gradient-boosted trees are comparable to the random forest even though they can perform better than random forests if finely tuned. It helps in generalizing the other model by optimizing the arbitrary differential loss function.

XGBoost is a library that can help us regularize gradient boosting in different languages like python, R, Julia, c++, and Java. XGBoost stands for extreme gradient boosting machine. As software, the main focus of XGBoost is to speed up and increase the performance of gradient boosted decision trees. This software can provide us with scalable, portable, and distributed gradient boosting. Using this library, we can utilize the functionality in single and distributed processing machines.

Gradient boosting is majorly focused on improving the performance of the machine learning models and with gradient boosting using XGBoost we can speed up the procedure as well as we can get better results. When we talk about the field of time series analysis and forecasting, we use traditional models like ARIMA (autoregressive integrated moving average) models where the main focus or the model is on regression analysis and if we can perform this regression with such software and technique, we can also achieve a state-of-the-art performance in the time series modelling. Ensemble of weak machine learning models to regularized gradient boosting can help us in improving the results in every section of data science. The section can also be time series. In this article, we will see how we can make XGBoost perform in time series modelling.

XGBoost is faster than the LSTM method with equal precision in the correct tuning parameters. The drawback is its feature-importance is not so accuracy as LSTM+SHAP combination.

So now we are using this model since it gave the best prediction values with a highest accuracy out of all the models in the past with an accuracy of 82.82% where the expected was around 70% and this can be concluded as the best

In [20]:
```
def create_features(df, label=None):
    df['date'] = df.index
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear

    X = df[['hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear']]
    if label:
        y = df[label]
        return X, y
    return X
```

We are first creating features from the posting date, since it is the only predefined feature

In [21]:
```
X, y = create_features(dt_ticket_count, label='Status')

```

```
C:\Users\MADHUV~1\AppData\Local\Temp/ipykernel_3452/752915949.py:10: FutureWarning:

Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().week instead.
```

In [22]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 20)
```

In [23]:
```
import xgboost as xgb
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

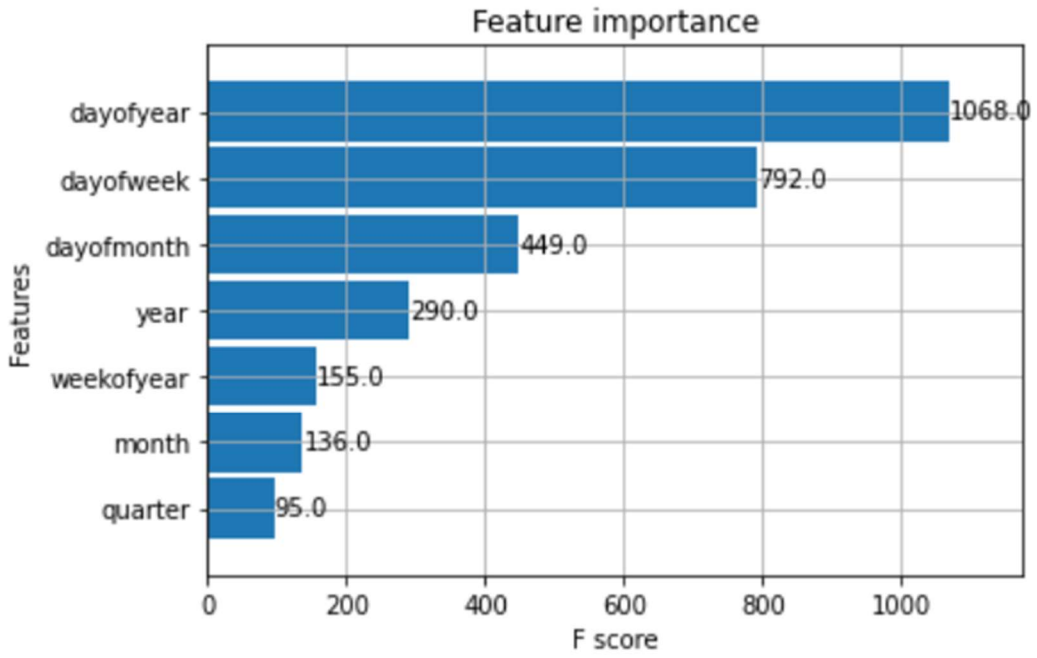
In [24]:
```
reg = xgb.XGBRegressor(n_estimators=1000)
```

In [25]:
```
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        early_stopping_rounds=50,
       verbose=False)
```

```
c:\users\madhuvani petla\appdata\local\programs\python\python39\lib\site-packages\xgboost\sklearn.py:793: UserWarning:

`early_stopping_rounds` in `fit` method is deprecated for better compatibility with scikit-learn, use `early_stopping_rounds` i
n constructor or`set_params` instead.
```

Out[25]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=1000,
             n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
             reg_alpha=0, reg_lambda=1, ...)
```

In [26]:
```
1 _ = plot_importance(reg, height=0.9)
```



In [27]:
```
1 y_pred = reg.predict(X_test)
```

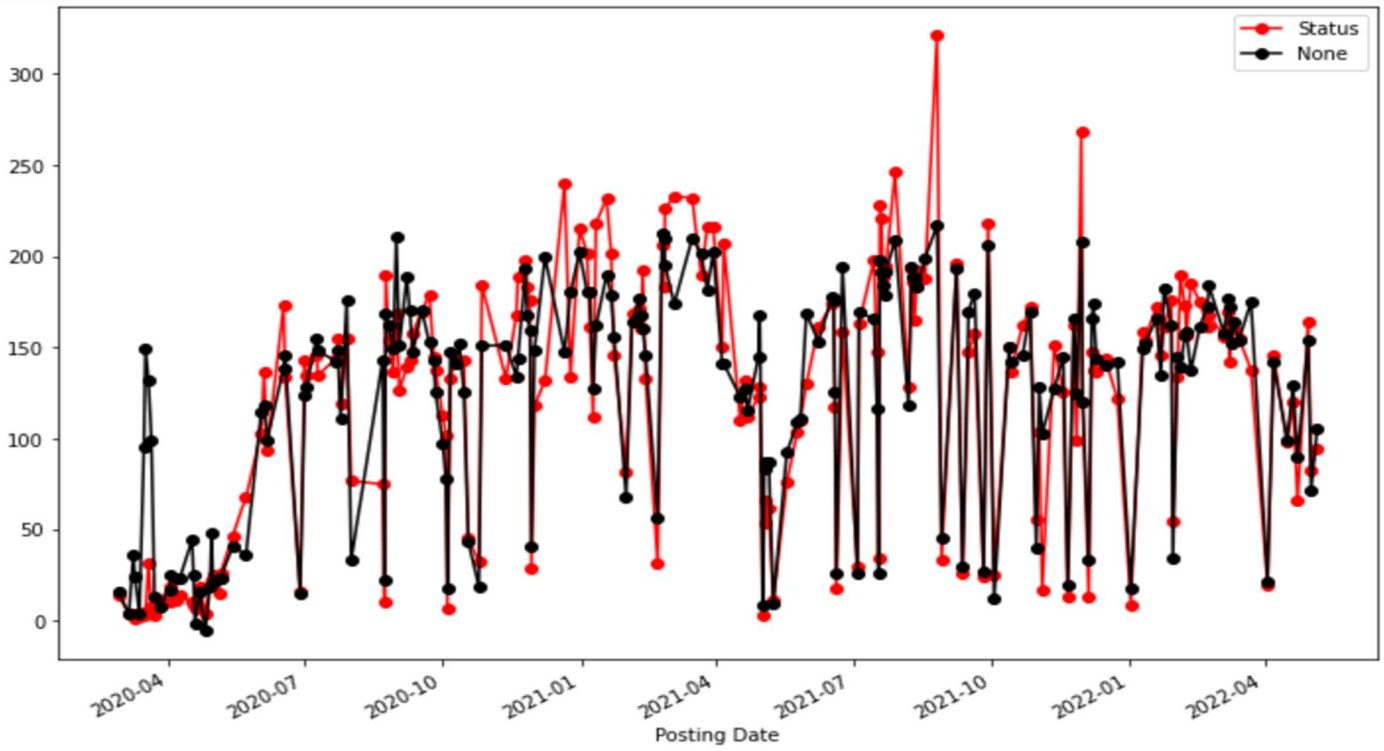
In [28]:
```
1 y_pred = pd.Series(reg.predict(X_test), index=X_test.index)
```

```
Posting Date
2022-03-15    154
2020-03-23     12
2021-07-19    197
2021-10-22    146
2022-05-02     71
             ...
2020-05-14     40
2021-04-17    122
2020-07-01    123
2022-05-06    105
2021-07-17    116
Length: 199, dtype: int32
```

```
Posting Date
2022-03-15    155
2020-03-23      3
2021-07-19    228
2021-10-22    162
2022-05-02     83
              ...
2020-05-14     46
2021-04-17    110
2020-07-01    143
2022-05-06     95
2021-07-17    148
Name: Status, Length: 199, dtype: int64
```

In [30]:
```
y_test.plot(marker='o', color='red', figsize=(12,8), legend=True)
y_pred.plot(marker='o', color='black', figsize=(12,8), legend=True)
```

Out[30]: <AxesSubplot:xlabel='Posting Date'>



In [31]:
```
from sklearn.metrics import r2_score

#r2_score(y_test, y_pred.apply(np.ceil).astype(int))

r2_score(y_test, y_pred)
```

Out[31]: 0.8282147160462968

In [32]:
```
from sklearn.metrics import mean_squared_error

rms = mean_squared_error(y_test, y_pred)

print(rms)
print(rms**0.5)
```

```
842.6140924557054
29.027815840254075
```

In [33]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[33]: 19.525026366339258

In [103]:
```
1 Test_data
```

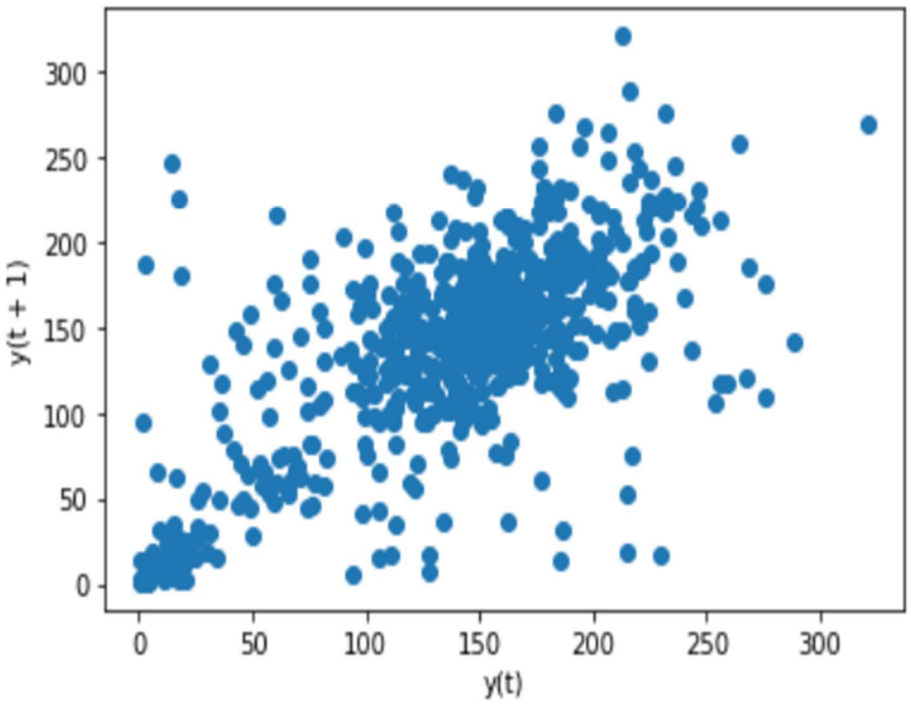
Out[103]:

Posting Date	True_values	Predicted_values	MG_MOTORS_Incident	MG Parts Support	Very High	Service Request	Medium	Low	High	Change Request	Difference
2022-03-15	155	154.167587	140	15	20	45	34	5	48	3	0.832413
2020-03-23	3	12.863723	3	0	0	0	3	0	0	0	-9.863723
2021-07-19	228	197.598846	228	0	16	97	68	7	38	2	30.401154
2021-10-22	162	146.139328	150	12	8	69	38	17	27	3	15.860672
2022-05-02	83	71.782173	76	7	3	28	21	0	31	0	11.217827
...	...	...	...	...	...	...	...	...	...	...	...
2020-05-14	46	40.314873	46	0	2	6	22	8	6	2	5.685127
2021-04-17	110	122.926315	110	0	4	60	26	0	20	0	-12.926315
2020-07-01	143	123.733322	143	0	10	14	69	3	46	1	19.266678
2022-05-06	95	105.196663	84	11	7	35	20	2	31	0	-10.196663
2021-07-17	148	116.648834	148	0	5	50	41	3	49	0	31.351166

199 rows × 11 columns

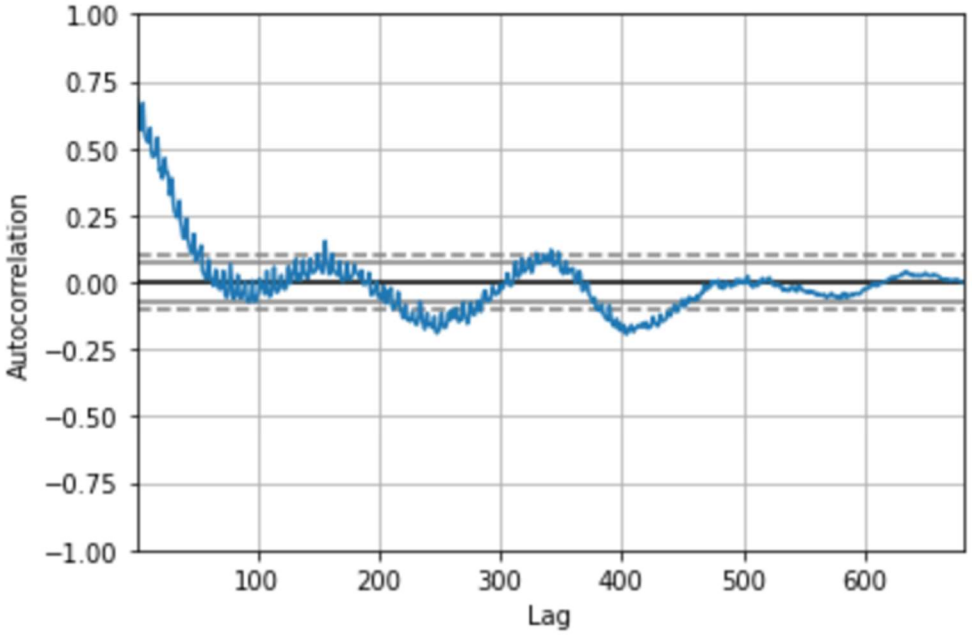
In [91]:
```
1 from pandas.plotting import lag_plot
2
3 lag_plot(_ticket_count["Status"])
```

Out[91]: <AxesSubplot:xlabel='y(t)', ylabel='y(t + 1)'>



In [92]:
```
1 from pandas.plotting import autocorrelation_plot
2
3 autocorrelation_plot(_ticket_count["Status"])
```

Out[92]: <AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>



Posting Date	Status	Very High	Service Request	Medium	Low	High	Change Request
2020-02-27	1	0	0	1	0	0	0
2020-02-28	14	0	0	13	0	1	0
2020-02-29	4	1	0	3	0	0	0
2020-03-02	1	0	0	1	0	0	0
2020-03-03	2	1	0	0	0	1	0
...	...	...	...	...	...	...	...
2022-05-02	83	3	28	21	0	31	0
2022-05-03	74	6	27	19	1	21	0
2022-05-04	116	8	35	24	1	48	0
2022-05-05	125	7	51	36	2	28	1
2022-05-06	95	7	35	20	2	31	0

794 rows × 7 columns

In [36]:
```
X_tr, X_test_distributed, y_tr, y_te = train_test_split(distributed_x, y, test_size = 0.25, random_state = 20)
```

In [37]:
```
X_test_distributed.drop("Status", axis = 1, inplace = True)
```

In [38]:
```
data =pd.concat({"True_values": y_test, "Predicted_values": y_pred.apply(np.ceil).astype(int)}, axis = 1)

Test_data = pd.concat([data,X_test_distributed], axis = 1)
```

In [39]:
```
Test_data["Difference"] = abs(Test_data["True_values"] - Test_data["Predicted_values"])
```

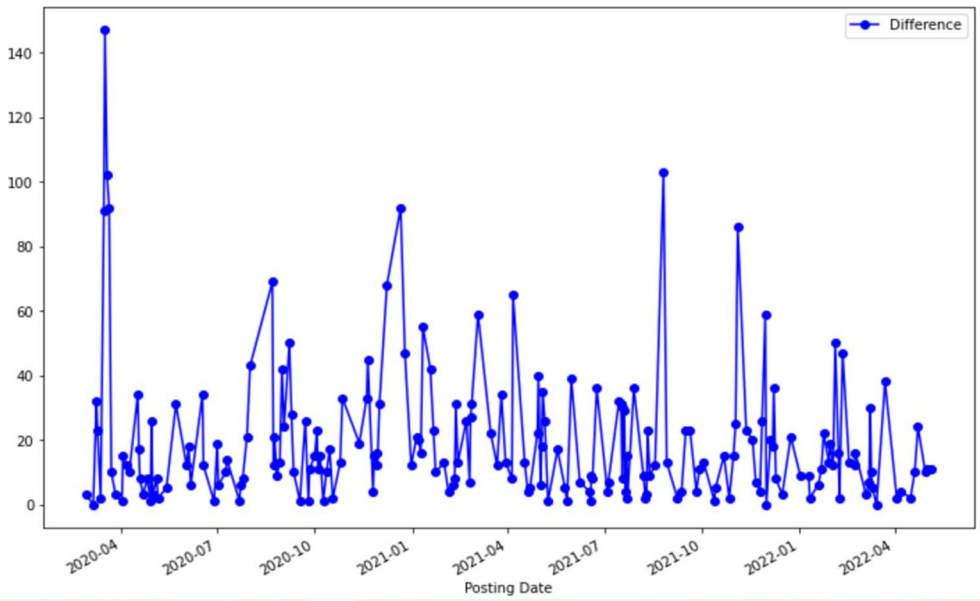
In [40]:
```
Test_data.head(30)
```

Out[40]:

Posting Date	True_values	Predicted_values	Very High	Service Request	Medium	Low	High	Change Request	Difference
2022-03-15	155	155	20	45	34	5	48	3	0
2020-03-23	3	13	0	0	3	0	0	0	10
2021-07-19	228	198	16	97	68	7	38	2	30
2021-10-22	162	147	8	69	38	17	27	3	15
2022-05-02	83	72	3	28	21	0	31	0	11
2022-04-23	66	90	5	32	18	2	9	0	24
2021-08-13	192	183	18	61	64	8	40	1	9
2021-04-30	128	168	7	79	24	1	16	1	40
2022-04-16	98	100	6	28	33	3	28	0	2
2020-06-02	103	115	10	6	52	9	23	3	12
2020-11-19	168	135	5	49	61	7	45	1	33
2021-12-08	138	174	7	36	53	7	31	4	36
2020-09-11	158	148	9	21	65	9	50	4	10
2022-01-22	146	135	16	53	48	6	23	0	11
2021-03-22	190	202	4	99	46	3	37	1	12
2022-03-11	160	165	23	47	28	3	58	1	5

In [41]:
```
Test_data["Difference"].plot(marker='o', color='blue', figsize=(12,8), legend=True)
```

Out[41]: <AxesSubplot:xlabel='Posting Date'>



Now let us predict the multilabel prediction or the multi-target prediction

In [43]:
```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Ridge
```

In [44]:
```
Refined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 794 entries, 2020-02-27 to 2022-05-06
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Change Request   794 non-null    int32
 1   High             794 non-null    int32
 2   Low              794 non-null    int32
 3   Medium           794 non-null    int32
 4   Service Request  794 non-null    int32
 5   Very High        794 non-null    int32
 6   Status           794 non-null    int64
dtypes: int32(6), int64(1)
memory usage: 63.3 KB
```

In [45]:
```
#second Prediction using MultioutputRegressor

MultiOutputReg_data = Refined_data

X,y = MultiOutputReg_data.iloc[:,-1],MultiOutputReg_data.iloc[:,:-1]
```

In [46]:
```
y.drop(["Change Request","Service Request"], inplace = True, axis = 1)
```

In [47]:
```
X
```

Out[47]:
```
Posting Date
2020-02-27      1
2020-02-28     14
2020-02-29      4
2020-03-02      1
2020-03-03      2
              ...
2022-05-02     83
2022-05-03     74
2022-05-04    116
2022-05-05    125
2022-05-06     95
Name: Status, Length: 794, dtype: int64
```

In [48]:
```
X_train_catg, X_test_catg, y_train_catg, y_test_catg = train_test_split(X, y, test_size = 0.25, random_state = 20)
```

In [49]:
```
regr = MultiOutputRegressor(Ridge(random_state=123)).fit(np.array(X_train_catg).reshape(595,1), y_train_catg)
```

In [50]:
```
y_pred_catg = regr.predict(np.array(X_test_catg).reshape(199,1))
```

In [51]:
```
y_pred = pd.DataFrame(y_pred_catg)
```

In [52]:
```
#y_pred.rename(columns = {'0' : 'First Name', 'age' : 'Age'}, inplace = True)
y_pred.astype(int)
```

Out[52]:

	0	1	2	3
0	36	5	49	9
1	0	0	5	0
2	53	8	70	13
3	38	5	51	9
4	19	3	28	4
...	...	...	...	...
194	10	1	17	2
195	25	4	36	6
196	33	5	45	8

Now after the multivariate prediction let us predict the categorical values for very high and high priority

In [54]:
```
#Categories = df.groupby(['Posting Date','Description'])['CATEGORY'].value_counts().unstack().fillna(0).astype(int)
Categories = df[df['Description'] == "High"]
Categories2 = df[df['Description'] == "Very High"]
```

In [55]:
```
Categories = pd.concat([Categories, Categories2])
```

In [56]:
```
Categories["Posting Date"] = pd.to_datetime(df["Posting Date"])
```

In [57]:
```
Categories = Categories.groupby('Posting Date')['CATEGORY'].value_counts().unstack().fillna(0).astype(int)
```

In [58]:
```
Categories["Total_tickets"] = Categories.iloc[:,:-1].sum(axis = 1)
```

In [59]:
```
Categories
```

Out[59]:

CATEGORY	After Sales	C4C	Dashboard	Finance	HCM	Hybris Marketing	Others	Parts	Sales	Success Factors	Warranty	Total_tickets
Posting Date												
2020-02-28	1	0	0	0	0	0	0	0	0	0	0	1
2020-02-29	1	0	0	0	0	0	0	0	0	0	0	1
2020-03-03	0	1	0	0	0	0	0	0	1	0	0	2
2020-03-04	0	0	0	0	0	0	0	0	1	0	0	1
2020-03-05	1	0	0	0	0	0	0	0	0	0	0	1

In [62]:
```
X,y = Categories.iloc[:,-1],Categories.iloc[:,:-1]
```

In [63]:
```
X_train_catg, X_test_catg, y_train_catg, y_test_catg = train_test_split(X, y, test_size = 0.25, random_state = 20)
```

In [65]:
```
regr = MultiOutputRegressor(Ridge(random_state=123)).fit(np.array(X_train_catg).reshape(581,1), y_train_catg)
```

In [67]:
```
y_pred_catg = regr.predict(np.array(X_test_catg).reshape(194,1))
```

In [88]:
```
1 y_pred.astype(int)
```

Out[88]:

	After Sales	C4C	Dashboard	Finance	HCM	Hybris Marketing	Others	Parts	Sales	Success Factors	Warranty
0	6	3	0	0	0	0	0	2	5	0	1
1	-1	1	0	0	0	0	0	0	0	0	0
2	1	2	0	0	0	0	0	1	2	0	0
3	19	6	0	0	0	0	1	4	11	0	2
4	14	5	0	0	0	0	1	3	8	0	1
...	...	...	...	...	...	...	...	...	...	...	...
189	4	2	0	0	0	0	0	1	3	0	1
190	24	7	0	0	0	0	2	5	14	0	2
191	22	7	0	0	0	0	1	4	13	0	2
192	31	9	0	0	0	0	2	6	18	0	3
193	27	8	0	0	0	0	2	5	16	0	2

194 rows × 11 columns

In [69]:
```
1 y_test_catg
```

Out[69]:

CATEGORY	After Sales	C4C	Dashboard	Finance	HCM	Hybris Marketing	Others	Parts	Sales	Success Factors	Warranty
Posting Date											
2021-06-18	9	4	0	0	0	0	0	1	5	0	0
2020-03-21	0	1	0	0	0	0	0	0	1	0	0
2022-01-26	3	0	1	0	0	0	0	0	4	0	0
2021-07-15	34	6	0	0	0	0	0	1	4	0	0
2021-09-27	17	7	0	0	0	0	0	5	5	0	1
...	...	...	...	...	...	...	...	...	...	...	...
2021-06-01	9	1	0	1	0	0	0	0	2	0	4
2022-02-23	12	3	1	1	1	0	0	12	25	0	0
2021-01-16	16	8	1	0	1	0	2	12	12	0	3
2021-02-26	26	7	2	0	0	0	5	7	22	1	6
2022-03-17	33	1	1	0	1	1	2	0	22	1	0

194 rows × 11 columns

From the above prediction we can find the distributed data we get an accuracy of 90% since the mean squared error is around 10.

In [91]:
```
1 mse = mean_squared_error(y_test_catg, y_pred)
2 mse
```

Out[91]: 10.627696489215259

5.4. Prediction for the next 6 months

Let's create a data frame with next 6-month dates as the index

In [70]:
```
#Prediction for the next 6 months


import datetime as dt
x_future_date = pd.date_range(start ="2022-08-01", end = "2023-01-31")

x_future_dates = pd.DataFrame()

x_future_dates["Dates"] = pd.to_datetime(x_future_date)

x_future_dates.index = x_future_dates["Dates"]
```

In [71]:
```
x_future_dates
```

Out[71]:

	Dates
Dates	
2022-08-01	2022-08-01
2022-08-02	2022-08-02
2022-08-03	2022-08-03
2022-08-04	2022-08-04
2022-08-05	2022-08-05
...	...
2023-01-27	2023-01-27
2023-01-28	2023-01-28
2023-01-29	2023-01-29
2023-01-30	2023-01-30
2023-01-31	2023-01-31

184 rows × 1 columns

In [72]:
```
X, y = create_features(x_future_dates, label='Dates')


y_future_total_tickets = reg.predict(X)
```

In [73]:
```
y_future_total_tickets
```

Out[73]:
```
array([111.88491 , 129.1587  , 138.18745 , 167.38808 , 177.86617 ,
       116.085724,  36.383804, 195.31863 , 201.70264 , 193.2268  ,
       189.04019 , 187.40605 , 114.27581 ,  43.051395, 180.3235  ,
       191.04047 , 186.54132 , 171.15892 , 162.50656 , 108.15466 ,
        25.0365  , 204.68837 , 234.83354 , 233.93391 , 226.04561 ,
       226.83661 , 162.38672 ,  57.343914, 214.25177 , 247.09554 ,
       275.24667 , 148.89235 , 156.60928 , 122.899796,  50.100464,
       182.4814  , 195.27632 , 198.01999 , 191.25403 , 177.51747 ,
        90.054436,  29.0747  , 167.1919  , 173.5759  , 171.10782 ,
       142.65302 , 152.08586 ,  82.61998 ,  26.817186, 178.44127 ,
       189.63751 , 177.52924 , 164.03973 , 184.29146 , 117.6034  ,
        35.083836, 172.72476 , 177.00705 , 198.83342 , 213.41676 ,
       216.33145 ,  20.584621,  21.139273, 103.01157 , 148.11726 ,
       146.05261 , 144.26056 , 138.69159 , 105.71514 ,  37.698944,
       153.64125 , 158.2726  , 154.30414 , 142.04362 , 125.43165 ,
        59.957924,  30.381006, 147.10237 , 162.86177 , 157.63315 ,
       150.77104 , 144.92062 , 108.082   ,  32.770027, 173.24847 ,
       184.8802  , 182.30325 , 175.57788 , 183.12984 , 116.29748 ,
        29.554626, 193.69331 ,  97.96141 , 110.04493 , 106.24459 ,
        87.74967 ,  23.032099,  19.021296, 150.64133 , 158.68393 ,
       155.98697 , 145.68211 , 128.99881 ,  88.45435 ,  35.095184,
       142.15521 , 154.69423 , 155.78813 , 137.52194 , 133.75327 ,
       107.0574  ,  27.625795, 145.8464  , 164.65414 , 205.04309 ,
       180.32283 , 170.88734 , 118.22506 ,  34.184456, 187.19244 ,
       219.83942 , 209.17966 , 102.59353 , 111.55629 , 109.99961 ,
        40.864452, 128.48245 , 154.43822 , 177.32582 , 161.28053 ,
       160.323   , 118.663864,  40.75333 , 146.39905 , 162.77037 ,
       155.26181 , 129.62372 , 132.91066 ,  91.2067  ,  23.925295,
       113.51228 , 132.88275 , 147.85637 , 143.53027 , 170.16338 ,
```

```
In [74]: 1 x_future_dates["Predicted Tickets"] = y_future_total_tickets
```

```
In [75]: 1 x_future_dates.drop("Dates", inplace = True, axis = 1)
```

```
In [76]: 1 x_future_dates
```

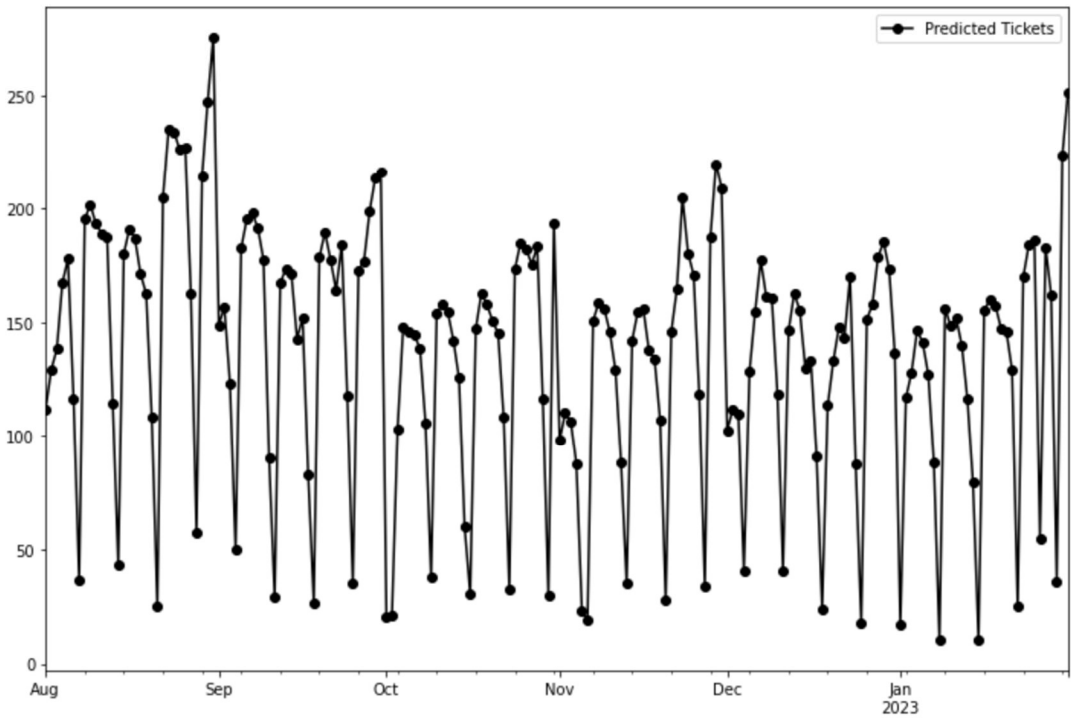
Out[76]:

Dates	date	hour	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear	Predicted Tickets
2022-08-01	2022-08-01	0	0	3	8	2022	213	1	31	111.884911
2022-08-02	2022-08-02	0	1	3	8	2022	214	2	31	129.158707
2022-08-03	2022-08-03	0	2	3	8	2022	215	3	31	138.187454
2022-08-04	2022-08-04	0	3	3	8	2022	216	4	31	167.388077
2022-08-05	2022-08-05	0	4	3	8	2022	217	5	31	177.866165
...	...	...	...	...	...	...	...	...	...	...
2023-01-27	2023-01-27	0	4	1	1	2023	27	27	4	182.958755
2023-01-28	2023-01-28	0	5	1	1	2023	28	28	4	162.003235
2023-01-29	2023-01-29	0	6	1	1	2023	29	29	4	36.078869
2023-01-30	2023-01-30	0	0	1	1	2023	30	30	5	223.491013
2023-01-31	2023-01-31	0	1	1	1	2023	31	31	5	251.209915

184 rows × 10 columns

```
In [77]: 1 x_future_dates["Predicted Tickets"].plot(marker='o', color='black', figsize=(12,8), legend=True)
```

Out[77]: <AxesSubplot:xlabel='Dates'>



In [78]:
```
y_future_prediction = regr.predict(np.array(x_future_dates["Predicted Tickets"]).reshape(184,1))
```

In [79]:
```
y_future_prediction
```

Out[79]:
```
array([[ 51.02539185,  14.01891906,   0.54315462, ...,  29.49740538,
          0.79580858,   4.88833464],
       [ 59.22172761,  16.00718823,   0.62011465, ...,  34.05122347,
          0.91649187,   5.56374162],
       [ 63.50582626,  17.04642595,   0.66034048, ...,  36.43143416,
          0.97957116,   5.91676645],
       ...,
       [ 15.05577823,   5.29340123,   0.20541556, ...,   9.51297795,
          0.26619007,   1.92431163],
       [103.98196727,  26.86513832,   1.04039387, ...,  58.91965189,
          1.57554403,   9.25214321],
       [117.13445741,  30.05567265,   1.16389005, ...,  66.22706941,
          1.76920199,  10.33595478]])
```

In [82]:
```
y_future_prediction.rename(columns = {0:'After Sales',
                                      1:'C4C',
                                      2:'Dashboard',
                                      3:'Finance',
                                      4:'HCM',
                                      5:'Hybris Marketing',
                                      6:'Others',
                                      7:'Parts',
                                      8:'Sales',
                                      9:'Success Factors',
                                      10:'Warranty'}, inplace = True)

```

In [92]:
```
y_future_prediction.head()
```

Out[92]:

Dates	After Sales	C4C	Dashboard	Finance	HCM	Hybris Marketing	Others	Parts	Sales	Success Factors	Warranty
2022-08-01	51.025392	14.018919	0.543155	0.548967	1.300661	0.445843	3.907292	9.801146	29.497405	0.795809	4.888335
2022-08-02	59.221728	16.007188	0.620115	0.632807	1.476870	0.513991	4.485742	11.232158	34.051223	0.916492	5.563742
2022-08-03	63.505826	17.046426	0.660340	0.676629	1.568972	0.549610	4.788090	11.980126	36.431434	0.979571	5.916766
2022-08-04	77.361386	20.407511	0.790438	0.818357	1.866846	0.664812	5.765936	14.399190	44.129471	1.183581	7.058513
2022-08-05	82.333190	21.613572	0.837121	0.869213	1.973733	0.706150	6.116818	15.267226	46.891766	1.256786	7.468208

In [84]:
```
y_future_prediction.index = x_future_dates.index
```

In [85]:
```
future_tickets_prediction =pd.concat([x_future_dates["Predicted Tickets"], y_future_prediction],axis = 1)
```

In [93]:
```
future_tickets_prediction.astype(int).head()
```

Out[93]:

Dates	Predicted Tickets	After Sales	C4C	Dashboard	Finance	HCM	Hybris Marketing	Others	Parts	Sales	Success Factors	Warranty
2022-08-01	111	51	14	0	0	1	0	3	9	29	0	4
2022-08-02	129	59	16	0	0	1	0	4	11	34	0	5
2022-08-03	138	63	17	0	0	1	0	4	11	36	0	5
2022-08-04	167	77	20	0	0	1	0	5	14	44	1	7
2022-08-05	177	82	21	0	0	1	0	6	15	46	1	7

6.Related Work

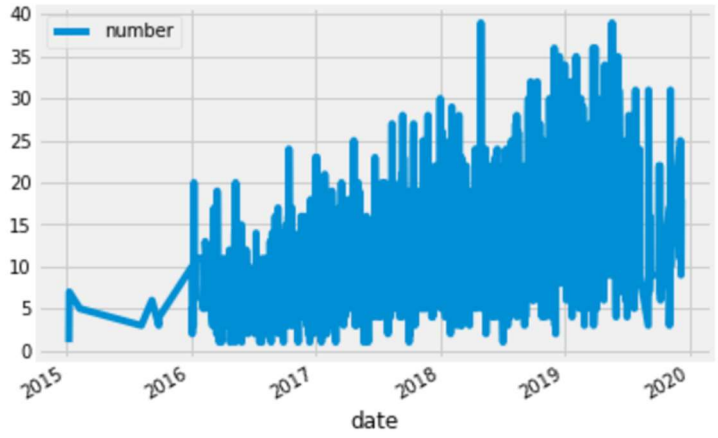
1.Food Analysis:

Description:

The order values are moving around the range between 5 to 15 and our model is also predicting between this range. Using those predictions, we can tell the restaurant to prepare around this range only so that we can reduce the wastage of food and maximize profit.

date	number
2015-01-09	1
2015-01-10	7
2015-02-10	5
2015-08-09	3
2015-09-09	6
...	...
2019-12-03	11
2019-12-04	25
2019-12-05	23
2019-12-06	9
2019-12-07	18

1207 rows × 1 columns



date	hour	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
2015-01-09	0	4	1	1	2015	9	9	2
2015-01-10	0	5	1	1	2015	10	10	2
2015-02-10	0	1	1	2	2015	41	10	7
2015-08-09	0	6	3	8	2015	221	9	32
2015-09-09	0	2	3	9	2015	252	9	37
...	...	...	...	...	...	...	...	...
2018-12-28	0	4	4	12	2018	362	28	52
2018-12-29	0	5	4	12	2018	363	29	52
2018-12-30	0	6	4	12	2018	364	30	52
2018-12-31	0	0	4	12	2018	365	31	1
2019-01-01	0	1	1	1	2019	1	1	1

993 rows × 8 columns

7.Worklog

	Model Name	Done by	No of Records	Size of Train/Test Data	Obstacles/ Show Stoppers	Expected Outcome	Accuracy
1	Linear Regession	Madhuvani	681	580	1.The data is not linear. 2.Linear Regrssion is not for Time series.	Above 70%	53.5.%
		Shraddha	794	670/124	1. Linear Regression isn't the best fit for Time Series. 2. Converting datetime[ns] type to float or int		
2	ARIMA Model	Shraddha	794	670/124	1. Faced an issue while importing ndiffs and auto_arima from pmdarima. This happens only few times during the runtime.	Expected accuracy greater than 80 as ARIMA model is considered one of	66%
			800	680/120		Expected considerable difference when	65%
		Madhuvani	794	523/171	1. arima was giving results that are way too far form th actual alue that could lead to accuracy problems	Above 70%	63.00%
3	XG Boost	Madhuvani	794	523/171	1.pretty much easy to handle but the in-depth details are to be studied	Expected accuracy is around 70 since it is not a suggested model for the time series model	82.80%

This is the worklog that has been created during the process of evaluation on the report

Link for the Worklog: [WORKLOG]

8.Future Work

Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity.

The future work on the developed model can be as follows:

- Split the given data into categories and make predictions for further use
- Use this model as the base model for further predictions
- As we analyze the data, we can add better features into the data like
    - Adding the Boolean value of recent software updating period if it is less than a week add the value as true else false
    - This can help in using a multivariate prediction
- Then this prediction is to updated in the platform for the better business solutions

9. References

Websites and Blogs referred for this project

- Time series
- Arima
- Linear regression
- XGBoost
- Multi target Regression
- How to use XGBoost for time-series analysis?
- 11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)
- Time Series Analysis: Definition, Types & Techniques | Tableau
- Univariate-Time-Series-using-LSTM
- Understanding input shape parameter in LSTM with Keras - Cross Validated
- Deep Learning Models for Univariate Time Series Forecasting
- time-series-analysis
- GitHub - stxupengyu/time-series-analysis: 使用经典的AR、MA、ARMA、ARIMA、ARCH、GARCH时间序列模型进行模型的检验和拟合。The classic AR, MA, ARMA, ARIMA, ARCH, GARCH time series models are used to test and predict the model.
- auto-regressive-time-series-model/AR-model-notebook.ipynb at master · bhattbhavesh91/auto-regressive-time-series-model · GitHub
- statsmodels.tsa.ar_model. AutoReg.predict — stats models
- Autoregression Models for Time Series Forecasting with Python
- What Is Time Series Forecasting?
- Time Series Forecasting with Prophet in Python
- ARIMA vs Prophet vs LSTM for Time Series Prediction - neptune.ai

- A Gentle Introduction to the Box-Jenkins Method for Time Series Forecasting
- 6.4.4. Univariate Time Series Models
- Time Series Forecasting with Regression and LSTM | Paper space Blog
- Python | ARIMA Model for Time Series Forecasting - GeeksforGeeks
- A Gentle Introduction to Exponential Smoothing for Time Series Forecasting in Python
- statsmodels.tsa.holtwinters.SimpleExpSmoothing — stats models
- Support vector machine models in drug design
- Support Vector Machine (SVM) Algorithm - Javatpoint
- How to Select a Model for Your Time Series Prediction Task [Guide] - neptune.ai
- Intro to Time Series Forecasting | Kaggle
- Matplotlib documentation — Matplotlib 3.5.2 documentation
- Logistic Regression in Python – Real Python
- sklearn.multioutput.MultiOutputRegressor — scikit-learn 1.1.1 documentation

These are references that are considered in the entire project

Videos/courses:

- Univariate Time Series Models || Forecasting || Data Science
- Time Series Forecasting Theory | AR, MA, ARMA, ARIMA | Data Science
- Auto Regressive Time Series Model in Python
- Autoregressive Models | Auto Regression | Machine Learning for Beginners | Edureka
- Time Series Analysis | Time Series Forecasting | Time Series Analysis In Excel | Simplilearn
- Time Series Analysis in Python | Time Series Forecasting | Data Science with Python | Edureka
- The Box Jenkins Models

10.Technologies used

Scripting Language: Python

- NumPy
- Pandas
- Scikit Learn
- Matplotlib
- Arima Time series model
- XGBoost Time series model
- Seaborn

Jupyter Notebook is used for the entire execution of the model

10.Conclusions

According to the analysis:

- The given data is a univariate data with date as the important feature
- The data consists of classification of tickets into different categories
- Categories involved in the data are
    - Description
    - Status
    - Category
    - Zone
    - Transaction type
- Time-series model can be applied on such data for future predictions
    - ARIMA
    - AR
    - MA
    - XGBOOST
    - LSTM
- **ARIMA** model gave us an accuracy of **58%**
- **Linear Regression** gave us an accuracy of **53.5%**
- **XGBOOST** gave us an accuracy of **82.8%**
- Since XGBOOST gave us highest accuracy we can make further predictions using the XGBOOST.
- For the further predictions MULTI TARGET REGRESSION is used
- Multi output regressor has given us an accuracy of **90%** which is pretty much a higher accuracy
- GitHub: Ticket Prediction