

RESTAURANT VISITORS TIME SERIES ANALYSIS PROJECT

Objective :

To forecast the number of visitors in restaurant for a daily data

importing required libraries

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import warnings
6 warnings.filterwarnings('ignore')
```

Loading the data

In [2]:

```
1 df=pd.read_csv('https://raw.githubusercontent.com/ubaid-shah/Time_Series_Analysis_IT')
```

In [3]:

1 df.head(30)

Out[3]:

	weekday	holiday	holiday_name	rest1	rest2	rest3	rest4	total
date								
2016-01-01	Friday	1	New Year's Day	65.0	25.0	67.0	139.0	296.0
2016-01-02	Saturday	0	na	24.0	39.0	43.0	85.0	191.0
2016-01-03	Sunday	0	na	24.0	31.0	66.0	81.0	202.0
2016-01-04	Monday	0	na	23.0	18.0	32.0	32.0	105.0
2016-01-05	Tuesday	0	na	2.0	15.0	38.0	43.0	98.0
2016-01-06	Wednesday	0	na	9.0	11.0	22.0	41.0	83.0
2016-01-07	Thursday	0	na	15.0	6.0	18.0	30.0	69.0
2016-01-08	Friday	0	na	79.0	32.0	22.0	16.0	149.0
2016-01-09	Saturday	0	na	44.0	44.0	47.0	99.0	234.0
2016-01-10	Sunday	0	na	26.0	43.0	49.0	94.0	212.0
2016-01-11	Monday	0	na	9.0	22.0	33.0	37.0	101.0
2016-01-12	Tuesday	0	na	6.0	10.0	21.0	20.0	57.0
2016-01-13	Wednesday	0	na	1.0	29.0	11.0	24.0	65.0
2016-01-14	Thursday	0	na	7.0	22.0	20.0	57.0	106.0
2016-01-15	Friday	0	na	32.0	21.0	21.0	21.0	95.0
2016-01-16	Saturday	0	na	49.0	52.0	50.0	86.0	237.0
2016-01-17	Sunday	0	na	60.0	25.0	62.0	50.0	197.0
2016-01-18	Monday	1	Martin Luther King Day	10.0	19.0	19.0	84.0	132.0
2016-01-19	Tuesday	0	na	12.0	27.0	20.0	41.0	100.0
2016-01-20	Wednesday	0	na	24.0	19.0	17.0	47.0	107.0
2016-01-21	Thursday	0	na	37.0	28.0	13.0	28.0	106.0
2016-01-22	Friday	0	na	39.0	55.0	23.0	44.0	161.0
2016-01-23	Saturday	0	na	42.0	48.0	51.0	47.0	188.0
2016-01-24	Sunday	0	na	59.0	32.0	64.0	50.0	205.0
2016-01-25	Monday	0	na	34.0	33.0	19.0	9.0	95.0
2016-01-26	Tuesday	0	na	9.0	30.0	14.0	23.0	76.0
2016-01-27	Wednesday	0	na	5.0	42.0	20.0	38.0	105.0
2016-01-28	Thursday	0	na	16.0	20.0	22.0	37.0	95.0
2016-01-29	Friday	0	na	57.0	53.0	24.0	16.0	150.0
2016-01-30	Saturday	0	na	46.0	50.0	45.0	84.0	225.0

We want to analyze the total number of visitors

In [4]:

```
1 data=df["total"]
```

In [5]:

```
1 data
```

Out[5]:

```
date
2016-01-01    296.0
2016-01-02    191.0
2016-01-03    202.0
2016-01-04    105.0
2016-01-05     98.0
...
2017-05-27     NaN
2017-05-28     NaN
2017-05-29     NaN
2017-05-30     NaN
2017-05-31     NaN
Name: total, Length: 517, dtype: float64
```

In [6]:

```
1 data.info()
```

```
<class 'pandas.core.series.Series'>
DatetimeIndex: 517 entries, 2016-01-01 to 2017-05-31
Series name: total
Non-Null Count  Dtype
-----
478 non-null    float64
dtypes: float64(1)
memory usage: 8.1 KB
```

In [7]:

```
1 data[data.isna()]
```

Out[7]:

date

2017-04-23	NaN
2017-04-24	NaN
2017-04-25	NaN
2017-04-26	NaN
2017-04-27	NaN
2017-04-28	NaN
2017-04-29	NaN
2017-04-30	NaN
2017-05-01	NaN
2017-05-02	NaN
2017-05-03	NaN
2017-05-04	NaN
2017-05-05	NaN
2017-05-06	NaN
2017-05-07	NaN
2017-05-08	NaN
2017-05-09	NaN
2017-05-10	NaN
2017-05-11	NaN
2017-05-12	NaN
2017-05-13	NaN
2017-05-14	NaN
2017-05-15	NaN
2017-05-16	NaN
2017-05-17	NaN
2017-05-18	NaN
2017-05-19	NaN
2017-05-20	NaN
2017-05-21	NaN
2017-05-22	NaN
2017-05-23	NaN
2017-05-24	NaN
2017-05-25	NaN
2017-05-26	NaN
2017-05-27	NaN
2017-05-28	NaN
2017-05-29	NaN
2017-05-30	NaN
2017-05-31	NaN

Name: total, dtype: float64

In [8]:

```
1 data.tail(45)
```

Out[8]:

```
date
2017-04-17    140.0
2017-04-18     91.0
2017-04-19     79.0
2017-04-20     90.0
2017-04-21    165.0
2017-04-22    226.0
2017-04-23      NaN
2017-04-24      NaN
2017-04-25      NaN
2017-04-26      NaN
2017-04-27      NaN
2017-04-28      NaN
2017-04-29      NaN
2017-04-30      NaN
2017-05-01      NaN
2017-05-02      NaN
2017-05-03      NaN
2017-05-04      NaN
2017-05-05      NaN
2017-05-06      NaN
2017-05-07      NaN
2017-05-08      NaN
2017-05-09      NaN
2017-05-10      NaN
2017-05-11      NaN
2017-05-12      NaN
2017-05-13      NaN
2017-05-14      NaN
2017-05-15      NaN
2017-05-16      NaN
2017-05-17      NaN
2017-05-18      NaN
2017-05-19      NaN
2017-05-20      NaN
2017-05-21      NaN
2017-05-22      NaN
2017-05-23      NaN
2017-05-24      NaN
2017-05-25      NaN
2017-05-26      NaN
2017-05-27      NaN
2017-05-28      NaN
2017-05-29      NaN
2017-05-30      NaN
2017-05-31      NaN
Name: total, dtype: float64
```

we can observe that the observations are recorded only till 24 April so we can remove the null values

In [9]:

```
1 data.dropna(inplace=True)
```

In [10]:

```
1 data.info()
```

```
<class 'pandas.core.series.Series'>
DatetimeIndex: 478 entries, 2016-01-01 to 2017-04-22
Series name: total
Non-Null Count  Dtype
-----
478 non-null    float64
dtypes: float64(1)
memory usage: 7.5 KB
```

In [11]:

```
1 data.index
```

Out[11]:

```
DatetimeIndex(['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04',
               '2016-01-05', '2016-01-06', '2016-01-07', '2016-01-08',
               '2016-01-09', '2016-01-10',
               ...,
               '2017-04-13', '2017-04-14', '2017-04-15', '2017-04-16',
               '2017-04-17', '2017-04-18', '2017-04-19', '2017-04-20',
               '2017-04-21', '2017-04-22'],
              dtype='datetime64[ns]', name='date', length=478, freq=None)
```

Since the data is on daily basis we will convert index frequency as daily

In [12]:

```
1 data.index.freq='d'
2
```

In [13]:

```
1 data.index
```

Out[13]:

```
DatetimeIndex(['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04',
               '2016-01-05', '2016-01-06', '2016-01-07', '2016-01-08',
               '2016-01-09', '2016-01-10',
               ...,
               '2017-04-13', '2017-04-14', '2017-04-15', '2017-04-16',
               '2017-04-17', '2017-04-18', '2017-04-19', '2017-04-20',
               '2017-04-21', '2017-04-22'],
              dtype='datetime64[ns]', name='date', length=478, freq='D')
```

In [14]:

```
1 tsa=pd.DataFrame(data)
```

In [15]:

```
1 tsa
```

Out[15]:

total	
date	
2016-01-01	296.0
2016-01-02	191.0
2016-01-03	202.0
2016-01-04	105.0
2016-01-05	98.0
...	...
2017-04-18	91.0
2017-04-19	79.0
2017-04-20	90.0
2017-04-21	165.0
2017-04-22	226.0

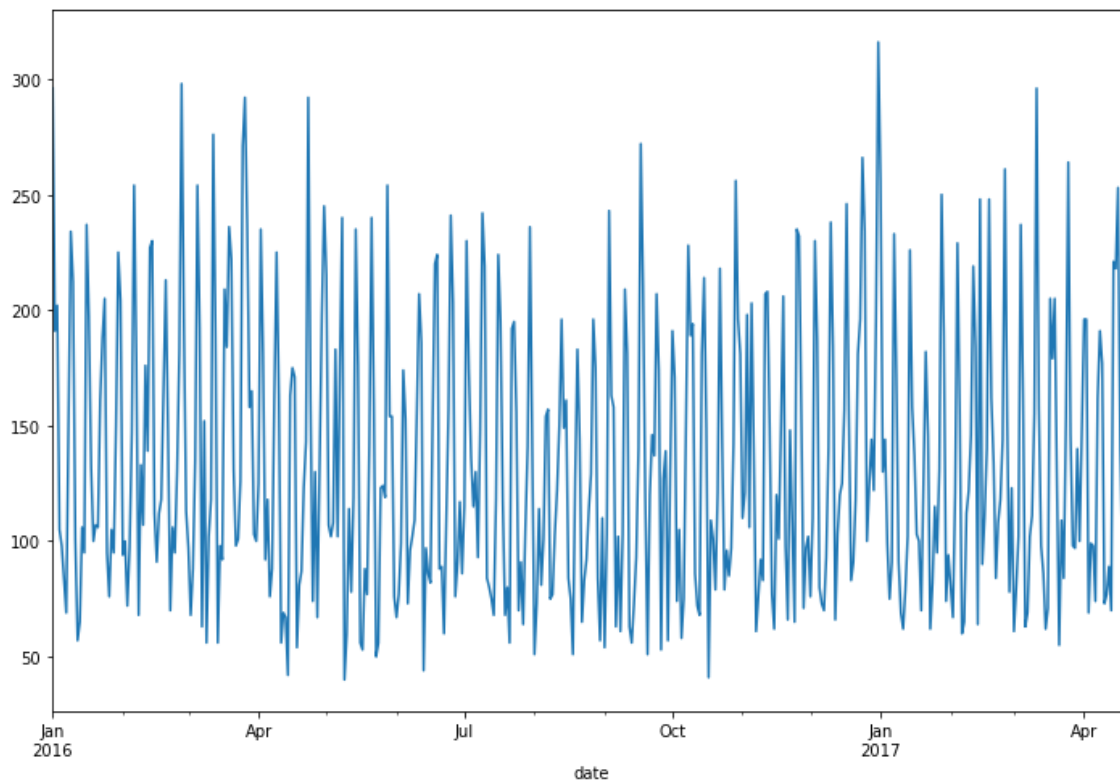
478 rows × 1 columns

In [16]:

```
1 tsa["total"].plot(figsize=(12,8))
```

Out[16]:

<AxesSubplot:xlabel='date'>



Checking for the STATIONARITY in series

This can be done in 2 ways:

- 1.PLOTTING GRAPH: ETS decomposition
- 2.STATISTICAL TEST :Augmented Dickey Fuller Test

In [17]:

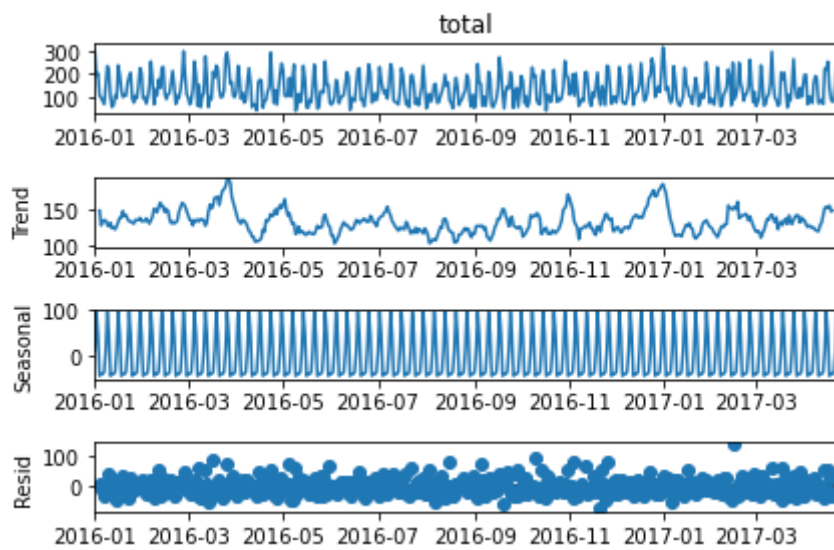
```
1 from statsmodels.tsa.seasonal import seasonal_decompose
```

In [18]:

```
1 x=seasonal_decompose(tsa["total"])
```


In [19]:

```
1 x.plot();
```

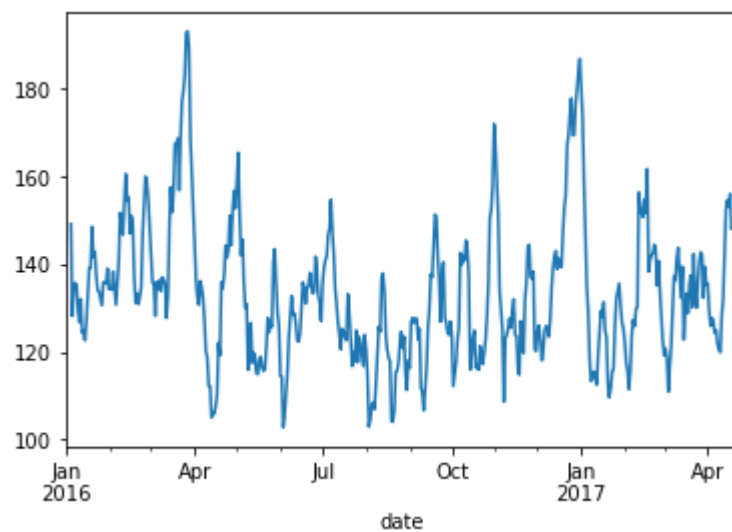


In [20]:

```
1 x.trend.plot()
```

Out[20]:

<AxesSubplot:xlabel='date'>

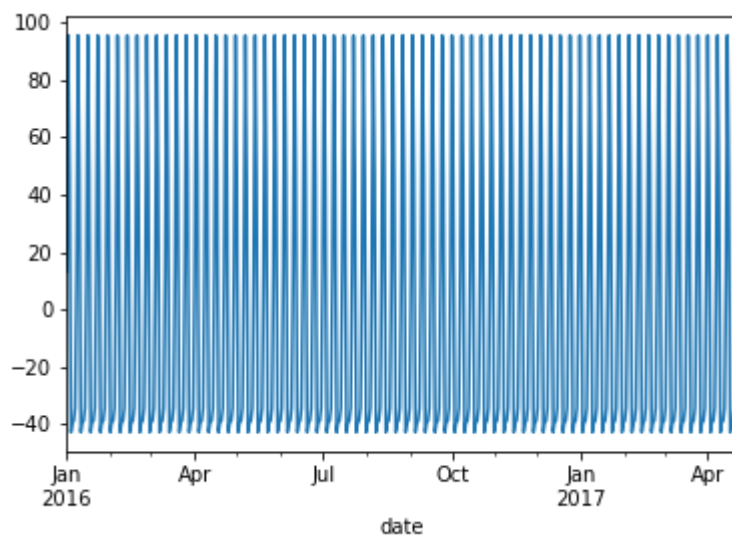


In [21]:

```
1 x.seasonal.plot()
```

Out[21]:

<AxesSubplot:xlabel='date'>



By the plot above, the data seems to be STATIONARY

Checking stationarity with Statistical test

In [22]:

```

1  #ADF TEST
2  from statsmodels.tsa.stattools import adfuller
3
4  def adf_test(tsa_data,col):
5      print(f"AUGMENTED DICKEY FULLER TEST FOR {col.upper()}")
6      print("\nH0: Data has UNIT ROOT and is NON-STATIONARY\nH1: Data has NO UNIT ROOT")
7      print("Reference p-value:0.05")
8      res=adfuller(tsa_data.dropna(),autolag="AIC")
9      index=["ADF test statistic","P value","No. of lags used","No of observations"]
10     output=pd.Series(res[:4],index=index)
11     print()
12     print(output)
13     print("---"*15)
14     print("\nResults of ADF TEST:\n")
15
16     ''' for p value less than 0.05 we reject null hypothesis i.e data is stationary
17         else we do not reject H0
18
19     '''
20     if res[1]<0.05:
21         print("Strong evidence against null hypothesis\nRejet the null hypothesis")
22         print("Data has NO UNIT ROOT and is STATIONARY ")
23     else:
24         print("Weak evidence against null hypothesis")
25         print("Do not Reject H0\nData has UNIT ROOT and is NON-STATIONARY ")

```

In [23]:

```
1 adf_test(tsa.total,"total")
```

AUGMENTED DICKEY FULLER TEST FOR TOTAL

H0: Data has UNIT ROOT and is NON-STATIONARY

H1: Data has NO UNIT ROOT and is STATIONARY

Reference p-value:0.05

ADF test statistic -5.592497

P value 0.000001

No. of lags used 18.000000

No of observations 459.000000

dtype: float64

Results of ADF TEST:

Strong evidence against null hypothesis

Rejet the null hypothesis

Data has NO UNIT ROOT and is STATIONARY

Splitting data into train test dataset

In [24]:

```
1 train=tsa[:436]
```

In [25]:

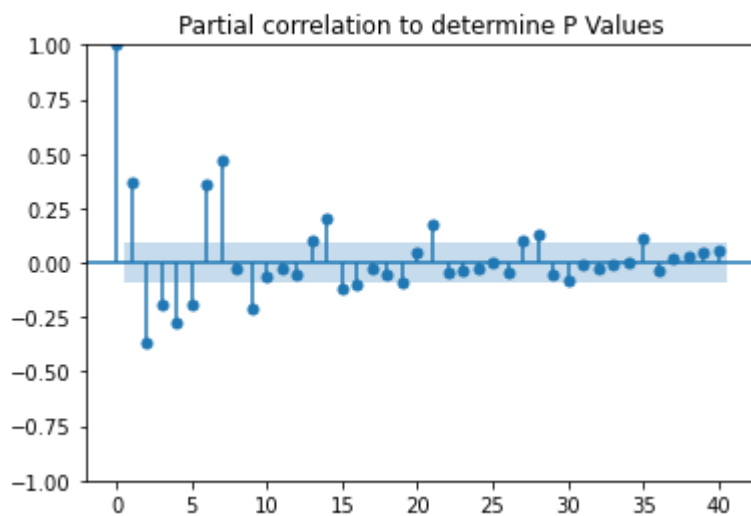
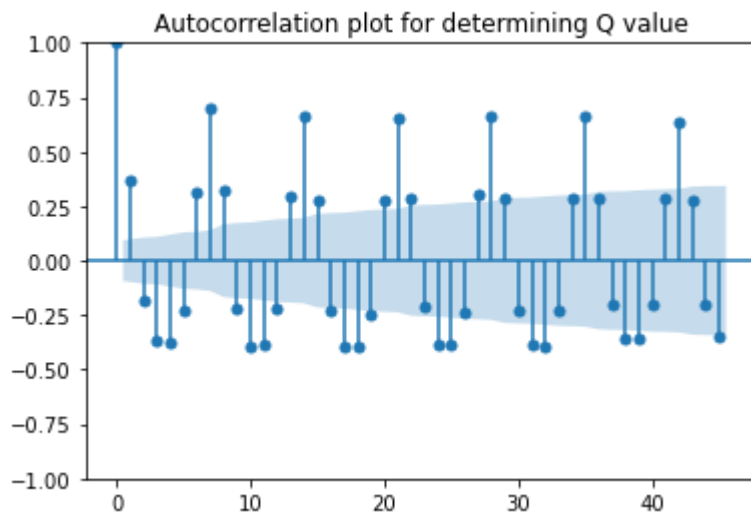
```
1 test=tsa[436:]
```

In [26]:

```
1 from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [27]:

```
1 title="Autocorrelation plot for determining Q value"  
2 lags=45  
3 plot_acf(train,title=title,lags=lags);  
4 title='Partial correlation to determine P Values'  
5 lags=40  
6 plot_pacf(train,title=title,lags=lags,method='ywm');
```



In [28]:

```
1 # !pip install pmdarima
```

In [29]:

```
1 from pmdarima.arima import auto_arima
```

In [30]:

```
1 arima_model=auto_arima(train,seasonal=True,m=7,stationary=True,stepwise=False,trace=
```

```

ARIMA(0,0,0)(0,0,0)[7] intercept : AIC=4809.926, Time=0.37 sec
ARIMA(0,0,0)(0,0,1)[7] intercept : AIC=4648.327, Time=0.60 sec
ARIMA(0,0,0)(0,0,2)[7] intercept : AIC=4578.361, Time=0.98 sec
ARIMA(0,0,0)(1,0,0)[7] intercept : AIC=4499.838, Time=1.03 sec
ARIMA(0,0,0)(1,0,1)[7] intercept : AIC=4342.331, Time=2.17 sec
ARIMA(0,0,0)(1,0,2)[7] intercept : AIC=4561.333, Time=3.50 sec
ARIMA(0,0,0)(2,0,0)[7] intercept : AIC=4442.352, Time=3.68 sec
ARIMA(0,0,0)(2,0,1)[7] intercept : AIC=inf, Time=4.29 sec
ARIMA(0,0,0)(2,0,2)[7] intercept : AIC=4662.621, Time=5.08 sec
ARIMA(0,0,1)(0,0,0)[7] intercept : AIC=4711.793, Time=0.56 sec
ARIMA(0,0,1)(0,0,1)[7] intercept : AIC=4606.228, Time=0.94 sec
ARIMA(0,0,1)(0,0,2)[7] intercept : AIC=4547.071, Time=1.41 sec
ARIMA(0,0,1)(1,0,0)[7] intercept : AIC=4489.132, Time=1.21 sec
ARIMA(0,0,1)(1,0,1)[7] intercept : AIC=4403.309, Time=2.03 sec
ARIMA(0,0,1)(1,0,2)[7] intercept : AIC=4460.001, Time=2.80 sec
ARIMA(0,0,1)(2,0,0)[7] intercept : AIC=4428.498, Time=2.75 sec
ARIMA(0,0,1)(2,0,1)[7] intercept : AIC=inf, Time=4.32 sec
ARIMA(0,0,1)(2,0,2)[7] intercept : AIC=inf, Time=3.92 sec
ARIMA(0,0,2)(0,0,0)[7] intercept : AIC=4713.153, Time=0.75 sec
ARIMA(0,0,2)(0,0,1)[7] intercept : AIC=4608.050, Time=1.29 sec
ARIMA(0,0,2)(0,0,2)[7] intercept : AIC=4548.843, Time=1.63 sec
ARIMA(0,0,2)(1,0,0)[7] intercept : AIC=4489.218, Time=1.47 sec
ARIMA(0,0,2)(1,0,1)[7] intercept : AIC=4551.629, Time=3.52 sec
ARIMA(0,0,2)(1,0,2)[7] intercept : AIC=4544.175, Time=5.25 sec
ARIMA(0,0,2)(2,0,0)[7] intercept : AIC=4427.435, Time=2.76 sec
ARIMA(0,0,2)(2,0,1)[7] intercept : AIC=inf, Time=4.53 sec
ARIMA(0,0,3)(0,0,0)[7] intercept : AIC=4675.935, Time=1.37 sec
ARIMA(0,0,3)(0,0,1)[7] intercept : AIC=4592.070, Time=1.68 sec
ARIMA(0,0,3)(0,0,2)[7] intercept : AIC=4544.173, Time=2.30 sec
ARIMA(0,0,3)(1,0,0)[7] intercept : AIC=4491.440, Time=3.86 sec
ARIMA(0,0,3)(1,0,1)[7] intercept : AIC=inf, Time=3.84 sec
ARIMA(0,0,3)(2,0,0)[7] intercept : AIC=4648.146, Time=3.49 sec
ARIMA(0,0,4)(0,0,0)[7] intercept : AIC=4674.535, Time=0.85 sec
ARIMA(0,0,4)(0,0,1)[7] intercept : AIC=4591.395, Time=2.03 sec
ARIMA(0,0,4)(1,0,0)[7] intercept : AIC=4598.356, Time=2.91 sec
ARIMA(0,0,5)(0,0,0)[7] intercept : AIC=4665.365, Time=1.66 sec
ARIMA(1,0,0)(0,0,0)[7] intercept : AIC=4747.438, Time=0.24 sec
ARIMA(1,0,0)(0,0,1)[7] intercept : AIC=4614.530, Time=1.45 sec
ARIMA(1,0,0)(0,0,2)[7] intercept : AIC=4550.811, Time=1.93 sec
ARIMA(1,0,0)(1,0,0)[7] intercept : AIC=4487.895, Time=1.46 sec
ARIMA(1,0,0)(1,0,1)[7] intercept : AIC=4385.420, Time=2.30 sec
ARIMA(1,0,0)(1,0,2)[7] intercept : AIC=4504.496, Time=3.53 sec
ARIMA(1,0,0)(2,0,0)[7] intercept : AIC=4426.117, Time=3.11 sec
ARIMA(1,0,0)(2,0,1)[7] intercept : AIC=4790.769, Time=3.53 sec
ARIMA(1,0,0)(2,0,2)[7] intercept : AIC=inf, Time=4.98 sec
ARIMA(1,0,1)(0,0,0)[7] intercept : AIC=4713.533, Time=0.95 sec
ARIMA(1,0,1)(0,0,1)[7] intercept : AIC=4608.148, Time=1.33 sec
ARIMA(1,0,1)(0,0,2)[7] intercept : AIC=4549.394, Time=1.39 sec
ARIMA(1,0,1)(1,0,0)[7] intercept : AIC=inf, Time=1.40 sec
ARIMA(1,0,1)(1,0,1)[7] intercept : AIC=inf, Time=2.80 sec
ARIMA(1,0,1)(1,0,2)[7] intercept : AIC=4508.729, Time=3.44 sec
ARIMA(1,0,1)(2,0,0)[7] intercept : AIC=inf, Time=3.08 sec
ARIMA(1,0,1)(2,0,1)[7] intercept : AIC=inf, Time=2.20 sec
ARIMA(1,0,2)(0,0,0)[7] intercept : AIC=4710.599, Time=0.78 sec
ARIMA(1,0,2)(0,0,1)[7] intercept : AIC=4593.505, Time=2.35 sec
ARIMA(1,0,2)(0,0,2)[7] intercept : AIC=4549.738, Time=1.53 sec
ARIMA(1,0,2)(1,0,0)[7] intercept : AIC=4481.617, Time=2.13 sec
ARIMA(1,0,2)(1,0,1)[7] intercept : AIC=4545.472, Time=3.63 sec
ARIMA(1,0,2)(2,0,0)[7] intercept : AIC=inf, Time=3.52 sec
ARIMA(1,0,3)(0,0,0)[7] intercept : AIC=4674.986, Time=1.76 sec
ARIMA(1,0,3)(0,0,1)[7] intercept : AIC=4588.563, Time=2.84 sec

```

```
ARIMA(1,0,3)(1,0,0)[7] intercept : AIC=4493.119, Time=1.86 sec
ARIMA(1,0,4)(0,0,0)[7] intercept : AIC=4679.071, Time=1.18 sec
ARIMA(2,0,0)(0,0,0)[7] intercept : AIC=4681.929, Time=0.33 sec
ARIMA(2,0,0)(0,0,1)[7] intercept : AIC=4597.795, Time=0.70 sec
ARIMA(2,0,0)(0,0,2)[7] intercept : AIC=4544.937, Time=2.23 sec
ARIMA(2,0,0)(1,0,0)[7] intercept : AIC=4489.608, Time=2.12 sec
ARIMA(2,0,0)(1,0,1)[7] intercept : AIC=4583.776, Time=2.97 sec
ARIMA(2,0,0)(1,0,2)[7] intercept : AIC=4541.858, Time=3.86 sec
ARIMA(2,0,0)(2,0,0)[7] intercept : AIC=4427.545, Time=4.02 sec
ARIMA(2,0,0)(2,0,1)[7] intercept : AIC=inf, Time=5.12 sec
ARIMA(2,0,1)(0,0,0)[7] intercept : AIC=4653.459, Time=1.29 sec
ARIMA(2,0,1)(0,0,1)[7] intercept : AIC=4577.374, Time=2.51 sec
ARIMA(2,0,1)(0,0,2)[7] intercept : AIC=4535.518, Time=2.93 sec
ARIMA(2,0,1)(1,0,0)[7] intercept : AIC=inf, Time=2.47 sec
ARIMA(2,0,1)(1,0,1)[7] intercept : AIC=inf, Time=2.58 sec
ARIMA(2,0,1)(2,0,0)[7] intercept : AIC=4449.163, Time=4.20 sec
ARIMA(2,0,2)(0,0,0)[7] intercept : AIC=4702.467, Time=1.05 sec
ARIMA(2,0,2)(0,0,1)[7] intercept : AIC=4608.095, Time=1.62 sec
ARIMA(2,0,2)(1,0,0)[7] intercept : AIC=inf, Time=2.54 sec
ARIMA(2,0,3)(0,0,0)[7] intercept : AIC=inf, Time=1.92 sec
ARIMA(3,0,0)(0,0,0)[7] intercept : AIC=4668.975, Time=0.74 sec
ARIMA(3,0,0)(0,0,1)[7] intercept : AIC=4584.808, Time=0.80 sec
ARIMA(3,0,0)(0,0,2)[7] intercept : AIC=4538.406, Time=2.38 sec
ARIMA(3,0,0)(1,0,0)[7] intercept : AIC=4499.682, Time=2.83 sec
ARIMA(3,0,0)(1,0,1)[7] intercept : AIC=4595.054, Time=3.26 sec
ARIMA(3,0,0)(2,0,0)[7] intercept : AIC=4588.708, Time=5.36 sec
ARIMA(3,0,1)(0,0,0)[7] intercept : AIC=4652.984, Time=1.62 sec
ARIMA(3,0,1)(0,0,1)[7] intercept : AIC=4575.298, Time=2.49 sec
ARIMA(3,0,1)(1,0,0)[7] intercept : AIC=4520.624, Time=3.18 sec
ARIMA(3,0,2)(0,0,0)[7] intercept : AIC=inf, Time=1.75 sec
ARIMA(4,0,0)(0,0,0)[7] intercept : AIC=4632.989, Time=0.89 sec
ARIMA(4,0,0)(0,0,1)[7] intercept : AIC=4569.184, Time=1.92 sec
ARIMA(4,0,0)(1,0,0)[7] intercept : AIC=4499.483, Time=3.29 sec
ARIMA(4,0,1)(0,0,0)[7] intercept : AIC=4632.418, Time=0.60 sec
ARIMA(5,0,0)(0,0,0)[7] intercept : AIC=4618.102, Time=0.96 sec
```

Best model: ARIMA(0,0,0)(1,0,1)[7] intercept

Total fit time: 224.970 seconds

In [31]:

```
1 arima_model.summary()
```

Out[31]:

SARIMAX Results

Dep. Variable:	y	No. Observations:	436			
Model:	SARIMAX(1, 0, [1], 7)	Log Likelihood	-2167.166			
Date:	Sun, 16 Jul 2023	AIC	4342.331			
Time:	10:07:45	BIC	4358.642			
Sample:	01-01-2016	HQIC	4348.768			
	- 03-11-2017					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
intercept	1.6404	0.916	1.791	0.073	-0.155	3.436
ar.S.L7	0.9863	0.007	136.449	0.000	0.972	1.000
ma.S.L7	-0.8432	0.041	-20.551	0.000	-0.924	-0.763
sigma2	1183.8244	69.038	17.147	0.000	1048.512	1319.137
Ljung-Box (L1) (Q):	18.47	Jarque-Bera (JB):	83.40			
Prob(Q):	0.00	Prob(JB):	0.00			
Heteroskedasticity (H):	0.89	Skew:	0.84			
Prob(H) (two-sided):	0.50	Kurtosis:	4.34			

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

ARIMA MODEL

In [32]:

```
1 from statsmodels.tsa.arima.model import ARIMA,ARIMAResults
```

In [33]:

```

1 arimamodel=ARIMA(train,order=(0,0,0))
2 results=arimamodel.fit()
3 results.summary()

```

Out[33]:

SARIMAX Results

Dep. Variable:	total	No. Observations:	436
Model:	ARIMA	Log Likelihood	-2402.963
Date:	Sun, 16 Jul 2023	AIC	4809.926
Time:	10:07:46	BIC	4818.082
Sample:	01-01-2016	HQIC	4813.145
	- 03-11-2017		
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
const	133.7477	3.409	39.229	0.000	127.065	140.430
sigma2	3586.4067	321.120	11.168	0.000	2957.022	4215.791

Ljung-Box (L1) (Q):	58.45	Jarque-Bera (JB):	37.02
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.87	Skew:	0.69
Prob(H) (two-sided):	0.41	Kurtosis:	2.62

Warnings:

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

In [34]:

```

1 prediction=results.get_forecast(steps=len(test))      #steps should be no.of periods

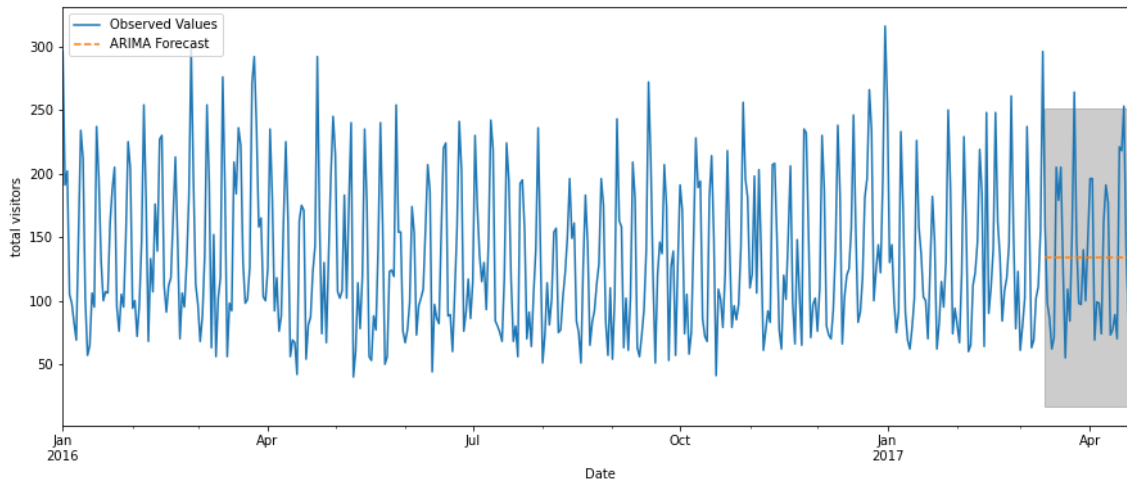
```

In [35]:

```

1 ax1=data["2016"].plot(label='Observed Values')
2 prediction.predicted_mean.plot(ax=ax1,label='ARIMA Forecast',figsize=(15,6),linestyle='--')
3 pred_ci=prediction.conf_int()
4 ax1.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color='k',alpha=0.1)
5 ax1.set_xlabel('Date')
6 ax1.set_ylabel('total visitors')
7 plt.legend(loc='upper left')
8 plt.show()

```

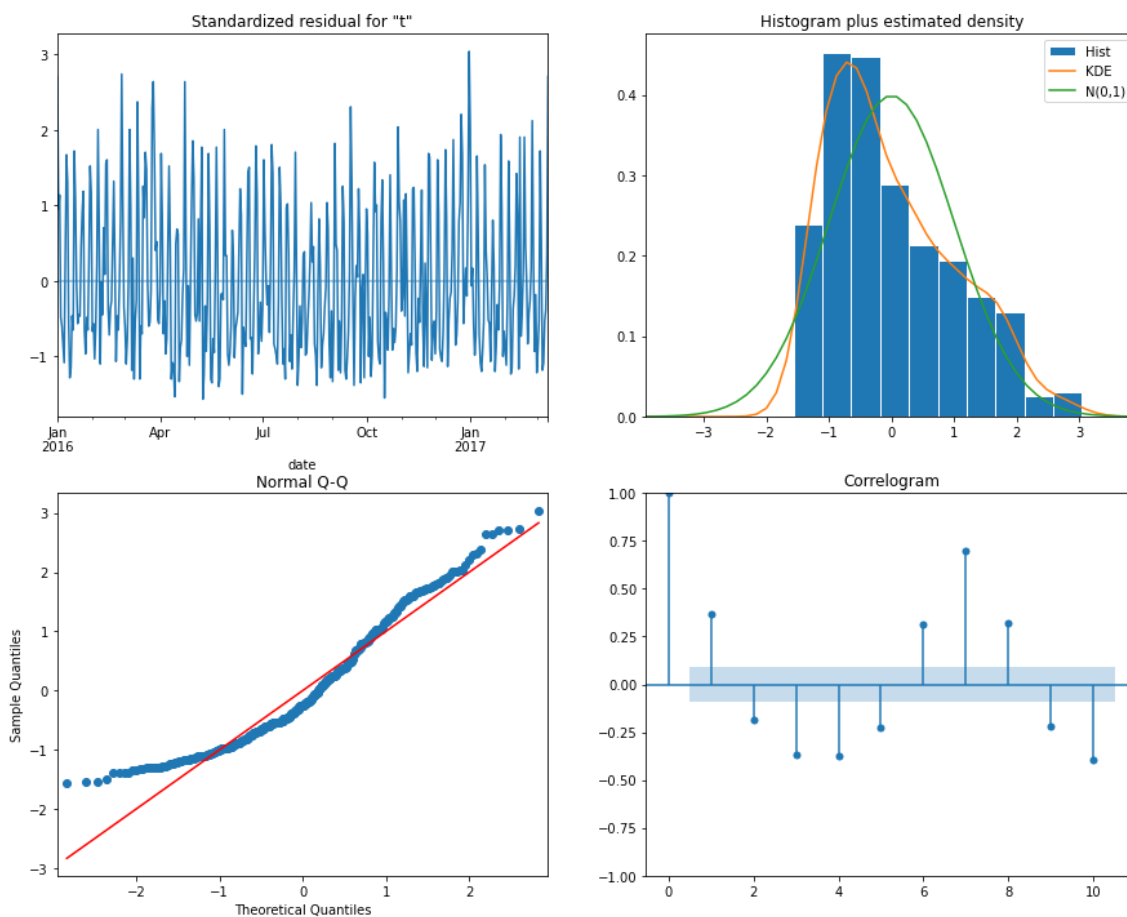


In [36]:

```

1 results.plot_diagnostics(figsize = (15, 12))
2 plt.show()

```



In [37]:

```
1 # fig1, ax2 = plt.subplots(figsize=(15, 6))
2 # test.plot(ax=ax2, label='Actual y value')
3 # prediction.predicted_mean.plot(ax=ax2, label='Predicted Y Values')
4 # ax2.set(title="Actual vs Predicted value[ARIMA]", xlabel="date", ylabel="Visitors")
5 # plt.legend()
6 # plt.show()
7
```

SARIMAX

In [38]:

```
1 from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [39]:

```
1 model1=SARIMAX(train,order=(0,0,0),seasonal_order=(1,0,1,7),enforce_stationarity=False)
2 fitted_model=model1.fit()
3 fitted_model.summary()
4
```

Out[39]:

SARIMAX Results

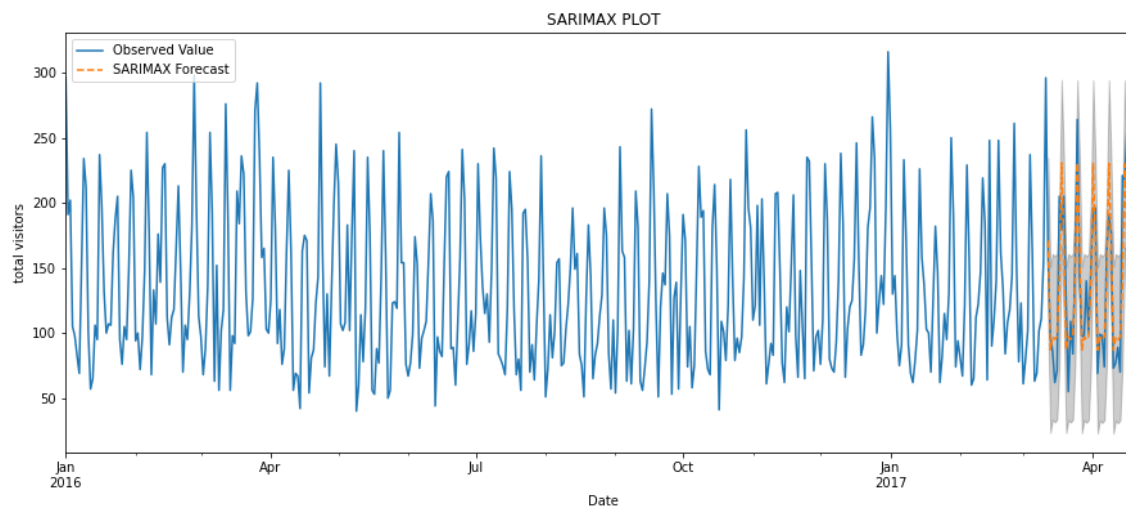
Dep. Variable:	total	No. Observations:	436			
Model:	SARIMAX(1, 0, [1], 7)	Log Likelihood	-2105.589			
Date:	Sun, 16 Jul 2023	AIC	4217.178			
Time:	10:07:50	BIC	4229.355			
Sample:	01-01-2016	HQIC	4221.987			
	- 03-11-2017					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.S.L7	0.9992	0.001	1085.998	0.000	0.997	1.001
ma.S.L7	-1.0522	0.026	-39.774	0.000	-1.104	-1.000
sigma2	954.6938	67.169	14.213	0.000	823.046	1086.342
Ljung-Box (L1) (Q):	19.23	Jarque-Bera (JB):	112.25			
Prob(Q):	0.00	Prob(JB):	0.00			
Heteroskedasticity (H):	1.09	Skew:	0.91			
Prob(H) (two-sided):	0.59	Kurtosis:	4.74			

Warnings:

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

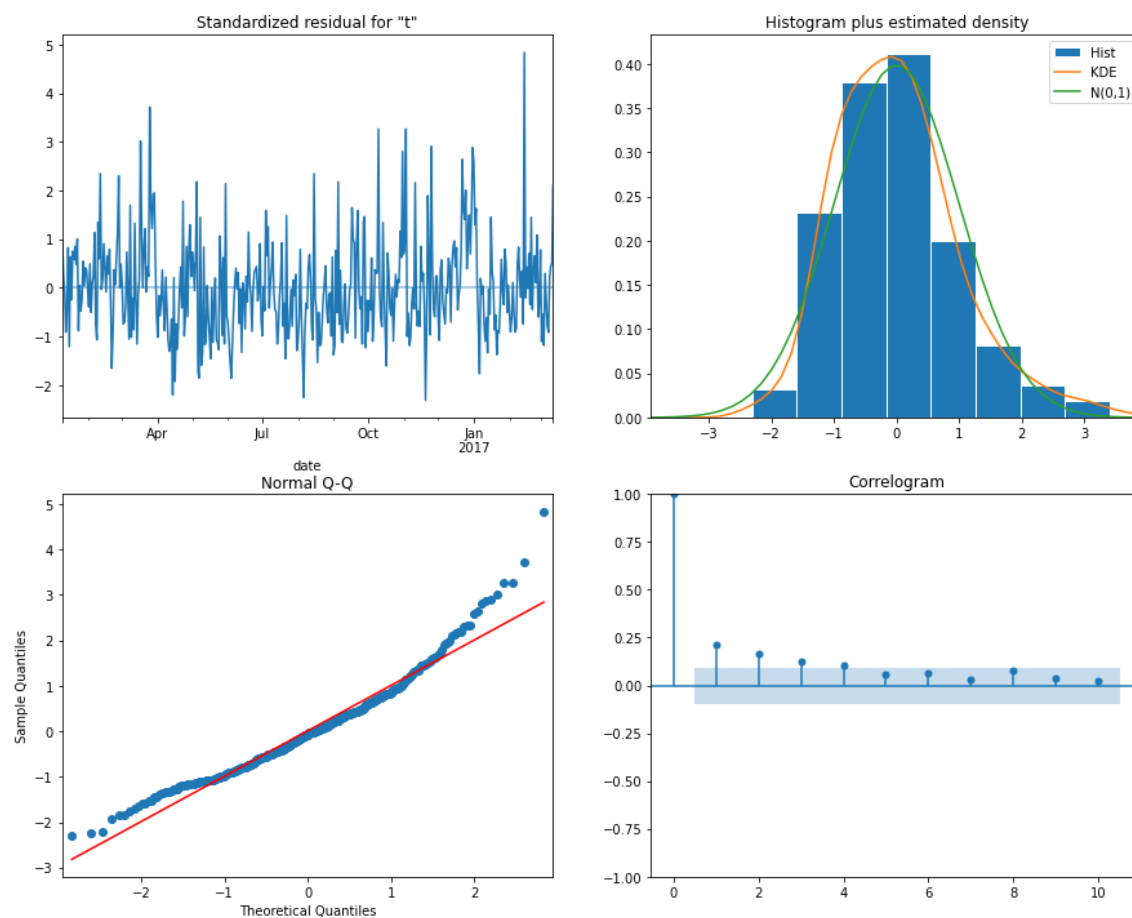
In [40]:

```
1 pred25=fitted_model.get_forecast(steps=len(test))
2 ax1=data['2016:'].plot(label='Observed Value')
3 pred25.predicted_mean.plot(ax=ax1,label='SARIMAX Forecast',figsize=(15,6),linestyle='--')
4 pred_ci=pred25.conf_int()
5 ax1.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color='k',alpha=0.1)
6 ax1.set_xlabel('Date')
7 plt.title("SARIMAX PLOT")
8 ax1.set_ylabel('total visitors')
9 plt.legend(loc='upper left')
10 plt.show()
```



In [41]:

```
1 fitted_model.plot_diagnostics(figsize = (15, 12))  
2 plt.show()
```



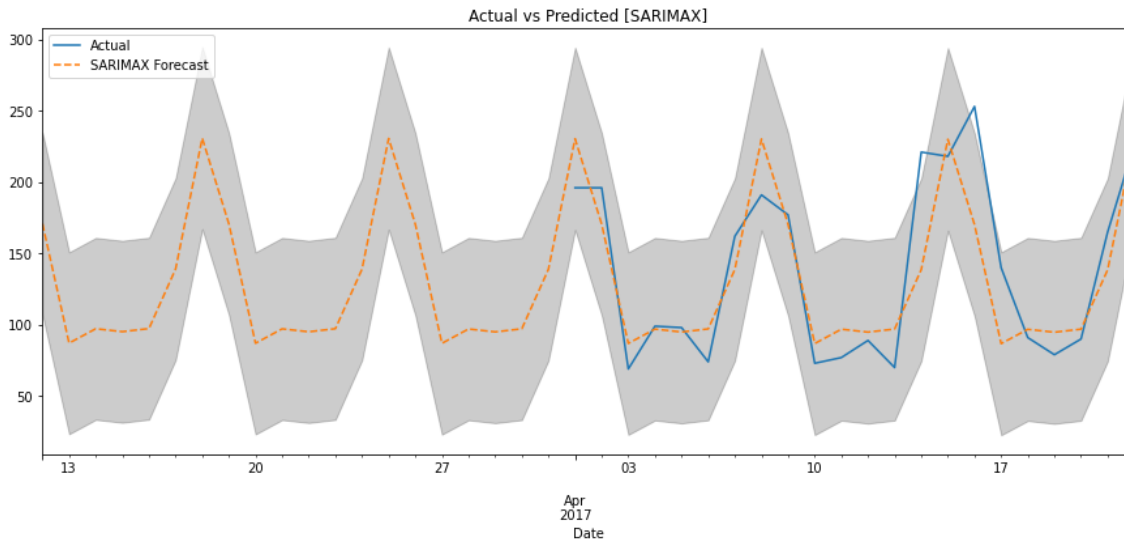
Actual vs Predicted

In [42]:

```

1 ax3=data["2017-04-01"].plot(label="Actual")
2 pred25.predicted_mean.plot(ax=ax3,label='SARIMAX Forecast',figsize=(15,6),linestyle=
3 ax3.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color="k",alpha=0
4 ax3.set_xlabel('Date')
5 plt.title("Actual vs Predicted [SARIMAX]")
6 ax1.set_ylabel('total visitors')
7 plt.legend(loc='upper left')
8 plt.show()

```



Evaluation of Model

In [43]:

```

1 sarimax_forecast=pred25.predicted_mean
2 actual_y=test["total"]
3 mse_sarimax=((sarimax_forecast-actual_y)**2).mean()
4 print(f"Mean Squared Error of Sarimax model is {round(mse_sarimax,2)}")
5 print(f"Root Mean Squared Error of Sarimax model is {round(np.sqrt(mse_sarimax),2)}")

```

Mean Squared Error of Sarimax model is 1000.45

Root Mean Squared Error of Sarimax model is 31.63

LSTM MODEL

In [44]:

```

1 from sklearn.preprocessing import StandardScaler

```

In [45]:

```
1 sc=StandardScaler()
2 scaled_train=sc.fit_transform(np.array(train).reshape(-1,1))
3 scaled_train
```

[-0.44664272],
[-0.46334108],
[0.4550685],
[0.90592411],
[1.18979616],
[-0.64702299],
[-0.96429175],
[-0.48003943],
[-0.64702299],
[0.27138658],
[1.52376328],
[1.1730978],
[-0.66372135],
[-0.56353121],
[-1.03108518],
[-0.61362628],
[0.22129151],
[2.0080156],
[0.72224219],
[-1.0978786]

In [46]:

```
1 from keras.preprocessing.sequence import TimeseriesGenerator
2 window_size=50      #no of inputs from trained data
3 generator=TimeseriesGenerator(scaled_train,scaled_train,length=window_size,batch_size=1)
```

In [47]:

```
1 len(generator)
```

Out[47]:

386

In [48]:

```
1 x,y=generator[0]
```


In [49]:

```
1 print(x,y)
```

```
[[[ 2.70934655]
 [ 0.95601917]
 [ 1.13970109]
 [-0.48003943]
 [-0.59692792]
 [-0.84740326]
 [-1.08118025]
 [ 0.25468823]
 [ 1.67404848]
 [ 1.30668465]
 [-0.54683286]
 [-1.28156052]
 [-1.14797367]
 [-0.46334108]
 [-0.64702299]
 [ 1.72414355]
 [ 1.05620931]
 [-0.02918382]
 [-0.56353121]
 [-0.44664272]
 [-0.46334108]
 [ 0.4550685 ]
 [ 0.90592411]
 [ 1.18979616]
 [-0.64702299]
 [-0.96429175]
 [-0.48003943]
 [-0.64702299]
 [ 0.27138658]
 [ 1.52376328]
 [ 1.1730978 ]
 [-0.66372135]
 [-0.56353121]
 [-1.03108518]
 [-0.61362628]
 [ 0.22129151]
 [ 2.0080156 ]
 [ 0.72224219]
 [-1.0978786 ]
 [-0.01248547]
 [-0.44664272]
 [ 0.70554384]
 [ 0.08770467]
 [ 1.55715999]
 [ 1.60725506]
 [-0.36315094]
 [-0.71381642]
 [-0.36315094]
 [-0.26296081]
 [ 0.55525863]]] [[1.323383]]
```

In [50]:

```

1 import tensorflow as tf
2 from tensorflow.keras.layers import LSTM,Dense
3 model=tf.keras.models.Sequential()
4 model.add(LSTM(256,activation="relu",input_shape=(window_size,1),return_sequences=True))
5 model.add(LSTM(124,activation="relu"))
6 model.add(Dense(1))

```

In [51]:

```

1 model.compile(loss="mse",optimizer="adam")
2

```

In [52]:

```

1 model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 256)	264192
lstm_1 (LSTM)	(None, 124)	188976
dense (Dense)	(None, 1)	125
Total params: 453,293		
Trainable params: 453,293		
Non-trainable params: 0		

In [53]:

```

1 from tensorflow.keras.callbacks import ModelCheckpoint
2 checkpoint = ModelCheckpoint(r"total.h5",
3                             monitor = 'loss', save_best_only = True)

```

In [54]:

```

1 import random as rd
2 rd.seed(25)
3 np.random.seed(13)
4 tf.random.set_seed(13)

```

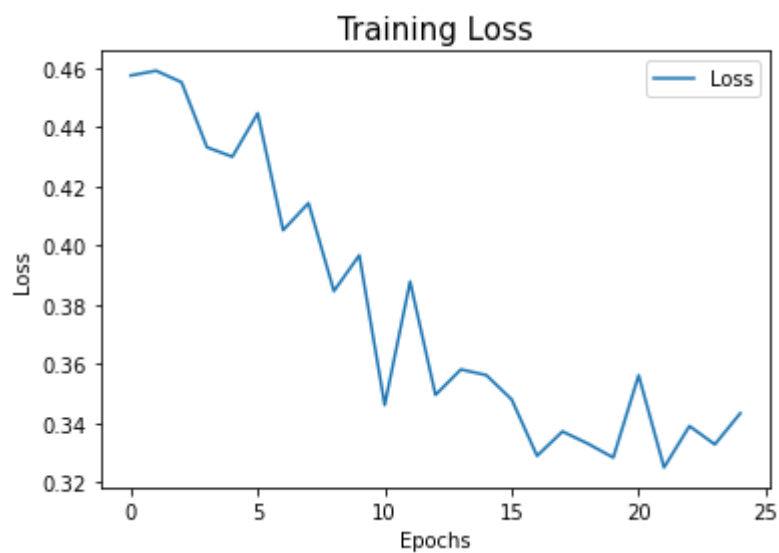
In [63]:

```
1 history=model.fit(generator,epochs=25,callbacks=[checkpoint])
```

```
Epoch 1/25
386/386 [=====] - 59s 138ms/step - loss: 0.4575
Epoch 2/25
386/386 [=====] - 49s 126ms/step - loss: 0.4591
Epoch 3/25
386/386 [=====] - 53s 137ms/step - loss: 0.4552
Epoch 4/25
386/386 [=====] - 52s 136ms/step - loss: 0.4332
Epoch 5/25
386/386 [=====] - 54s 140ms/step - loss: 0.4300
Epoch 6/25
386/386 [=====] - 50s 128ms/step - loss: 0.4447
Epoch 7/25
386/386 [=====] - 49s 127ms/step - loss: 0.4052
Epoch 8/25
386/386 [=====] - 49s 127ms/step - loss: 0.4144
Epoch 9/25
386/386 [=====] - 48s 125ms/step - loss: 0.3847
Epoch 10/25
386/386 [=====] - 49s 127ms/step - loss: 0.3968
Epoch 11/25
386/386 [=====] - 49s 126ms/step - loss: 0.3462
Epoch 12/25
386/386 [=====] - 50s 129ms/step - loss: 0.3879
Epoch 13/25
386/386 [=====] - 49s 127ms/step - loss: 0.3496
Epoch 14/25
386/386 [=====] - 55s 143ms/step - loss: 0.3582
Epoch 15/25
386/386 [=====] - 51s 133ms/step - loss: 0.3563
Epoch 16/25
386/386 [=====] - 51s 133ms/step - loss: 0.3481
Epoch 17/25
386/386 [=====] - 52s 134ms/step - loss: 0.3290
Epoch 18/25
386/386 [=====] - 57s 148ms/step - loss: 0.3373
Epoch 19/25
386/386 [=====] - 60s 156ms/step - loss: 0.3332
Epoch 20/25
386/386 [=====] - 52s 134ms/step - loss: 0.3285
Epoch 21/25
386/386 [=====] - 62s 160ms/step - loss: 0.3563
Epoch 22/25
386/386 [=====] - 57s 147ms/step - loss: 0.3251
Epoch 23/25
386/386 [=====] - 53s 136ms/step - loss: 0.3391
Epoch 24/25
386/386 [=====] - 58s 150ms/step - loss: 0.3329
Epoch 25/25
386/386 [=====] - 52s 134ms/step - loss: 0.3434
```

In [64]:

```
1 plt.plot(history.history["loss"])
2 plt.xlabel("Epochs", fontsize = 10)
3 plt.ylabel("Loss", fontsize = 10)
4 plt.legend(["Loss"])
5 plt.title("Training Loss", fontsize = 15)
6 plt.show()
```



In [65]:

```
1 from tensorflow.keras.models import load_model
2 model=load_model(r'total.h5')
```

In [66]:

```
1  #Creating an empty forecasts List:
2  lstm_predictions_scaled = []
3
4  #Creating a batch of the latest data points based on the window size for forecast:
5  batch = scaled_train[-window_size:]
6  #Reshaping the batch as per model requirements:
7  current_batch = batch.reshape((1, window_size, 1))
8
9
10 for i in range(len(test)):
11     lstm_pred = model.predict(current_batch)[0]
12     #Appending the next month forecast to the forecasts list:
13     lstm_predictions_scaled.append(lstm_pred)
14     #removing the earliest data point in its place to preserve the window size:
15     current_batch = np.append(current_batch[:, 1:, :], [[lstm_pred]], axis = 1)
16
17 #Since the original values were scaled before training the model, we need to
18 #inverse scale the forecast in order to get the forecast for the original data.
19 lstm_predictions = sc.inverse_transform(lstm_predictions_scaled)
```

```
1/1 [=====] - 2s 2s/step
1/1 [=====] - 0s 130ms/step
1/1 [=====] - 0s 108ms/step
1/1 [=====] - 0s 107ms/step
1/1 [=====] - 0s 113ms/step
1/1 [=====] - 0s 94ms/step
1/1 [=====] - 0s 222ms/step
1/1 [=====] - 0s 101ms/step
1/1 [=====] - 0s 104ms/step
1/1 [=====] - 0s 78ms/step
1/1 [=====] - 0s 84ms/step
1/1 [=====] - 0s 65ms/step
1/1 [=====] - 0s 52ms/step
1/1 [=====] - 0s 98ms/step
1/1 [=====] - 0s 96ms/step
1/1 [=====] - 0s 48ms/step
1/1 [=====] - 0s 62ms/step
1/1 [=====] - 0s 85ms/step
1/1 [=====] - 0s 63ms/step
1/1 [=====] - 0s 71ms/step
1/1 [=====] - 0s 78ms/step
1/1 [=====] - 0s 70ms/step
1/1 [=====] - 0s 89ms/step
1/1 [=====] - 0s 113ms/step
1/1 [=====] - 0s 102ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 95ms/step
1/1 [=====] - 0s 52ms/step
1/1 [=====] - 0s 91ms/step
1/1 [=====] - 0s 76ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 63ms/step
1/1 [=====] - 0s 96ms/step
1/1 [=====] - 0s 64ms/step
1/1 [=====] - 0s 120ms/step
1/1 [=====] - 0s 100ms/step
1/1 [=====] - 0s 82ms/step
1/1 [=====] - 0s 65ms/step
1/1 [=====] - 0s 100ms/step
1/1 [=====] - 0s 79ms/step
1/1 [=====] - 0s 117ms/step
1/1 [=====] - 0s 67ms/step
```

In [67]:


```
1 for i in range(0,len(lstm_predictions)):
2     pred_value=lstm_predictions[i][0]
3
4 pred_value
5
```

Out[67]:

213.13553173982254

In [68]:

```
1 lstm_pred = pd.DataFrame(data = [lstm_predictions[i][0] for i in range(0, len(lstm_p
2
3 lstm_pred
```



Out[68]:

LSTM Forecast

date	
2017-03-12	178.573792
2017-03-13	122.755204
2017-03-14	110.975403
2017-03-15	97.337437
2017-03-16	106.743558
2017-03-17	161.055784
2017-03-18	225.038395
2017-03-19	172.845750
2017-03-20	111.363322
2017-03-21	104.815496
2017-03-22	97.174890
2017-03-23	107.328748
2017-03-24	155.172658
2017-03-25	214.844290
2017-03-26	169.252078
2017-03-27	109.579725
2017-03-28	103.017184
2017-03-29	95.937100
2017-03-30	107.181300
2017-03-31	156.856641
2017-04-01	212.792696
2017-04-02	170.055049
2017-04-03	110.551237
2017-04-04	102.328318
2017-04-05	95.468408
2017-04-06	107.268060
2017-04-07	157.977711
2017-04-08	213.977261
2017-04-09	168.617080
2017-04-10	109.630083
2017-04-11	102.872841
2017-04-12	95.496875
2017-04-13	107.866776
2017-04-14	159.189982
2017-04-15	213.272672
2017-04-16	165.203557

LSTM Forecast**date**

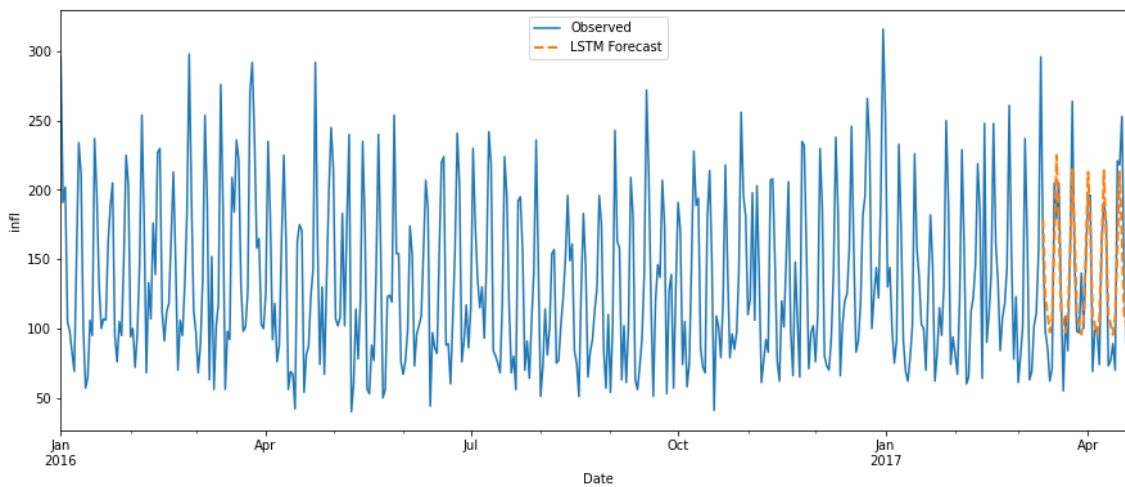
2017-04-17	108.364923
2017-04-18	102.874933
2017-04-19	95.048775
2017-04-20	107.673466
2017-04-21	158.389102

In [69]:

```

2017-04-22 213.135532
1 ax35 = data['2016':].plot(label = 'Observed')
2
3 lstm_pred.plot(ax = ax35, label = 'LSTM Forecast', figsize = (15, 6), linewidth = 2,
4 ax35.set_xlabel('Date')
5 ax35.set_ylabel('infl')
6 plt.legend()
7 plt.show()

```



In [70]:

```

1 y_forecasted_LSTM = lstm_pred['LSTM Forecast']
2 y_truth = test["total"]
3 mse_LSTM = ((y_forecasted_LSTM - y_truth) ** 2).mean()
4 print('The Mean Squared Error of LSTM forecast is {}'.format(round(mse_LSTM, 2)))
5 print('The Root Mean Squared Error of LSTM forecast is {}'.format(round(np.sqrt(mse_

```

The Mean Squared Error of LSTM forecast is 932.98

The Root Mean Squared Error of LSTM forecast is 30.54

In []:

1