Problem Statement : To Optimize the ad placements for my clients by forecasting the number of daily views in each language

In [1]:

```
import numpy as np
import pandas as pd
import regex as re
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
plt.rcParams['figure.figsize'] = (20, 6)
from statsmodels.tsa.statespace.sarimax import SARIMAX
import warnings
warnings.filterwarnings("ignore")
from prophet import Prophet
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm
```

In [2]:

```
from sklearn.metrics import (
    mean_squared_error as mse,
    mean_absolute_error as mae,
    mean_absolute_percentage_error as mape
)

# Creating a function to print values of all these metrics.
def performance(actual, predicted):
    print('MAE :', round(mae(actual, predicted), 3))
    print('RMSE :', round(mse(actual, predicted)**0.5, 3))
    print('MAPE:', round(mape(actual, predicted), 3))
```

In [3]:

```
def performance_mape(actual, predicted):
    return(round(mape(actual, predicted), 3))
```

In [4]:

```
df1=pd.read_csv('train_1.csv')
df2=pd.read_csv('Exog_Campaign_eng.csv')
# Reading data
```

In [5]:

```
df1['Page'].nunique()
# There are a total of 145063 unique pages
```

Out[5]:

```
145063
```

Function for extraction

In [6]:

```
def language(x):
    if re.search('_[a-z][a-z]\.', x) is None:
        return 'commons'
    ans=re.search('_[a-z][a-z]\.', x).group(0)[1:3]
    return ans
```

In [7]:

```
def title(x):
    if re.search('_[a-z][a-z]\.', x) is None:
        if x.find('com')==-1:
            ind=x.index('www')-1
            return x[:ind]
        ind=x.index('com')-1
        return x[:ind]
    ind=re.search('_[a-z][a-z]\.', x).span()[0]
    ans=x[:ind]
    return ans
```

In [8]:

```
def access(x):
    ans=x.rfind('.org_')+5
    return x[ans:]
```

In [9]:

```
def access_orgin(x):
    ans=re.search('\.[a-z]{3,}\.', x).group(0)[1:-1]
    return ans
```

Extracting Language, Title, Access , Website and Access_orgin

In [10]:

```
df1['language']=df1['Page'].apply(lambda x: language(x))
df1['title']=df1['Page'].apply(lambda x: title(x))
df1['access']=df1['Page'].apply(lambda x: access(x))
df1['website']=df1['Page'].apply(lambda x: access_orgin(x))
```

In [11]:

```
df1.drop('Page',axis=1,inplace=True)
df1=pd.concat([df1,pd.DataFrame(df1['access'].str.split('_').to_list(),columns=['Access','Access_orgin'])],axis=1)
df1.drop('access',inplace=True,axis=1)
```

In [12]:

```
df1
```

Out[12]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	language	
0	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	...	20.0	22.0	19.0	18.0	20.0	zh	
1	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	...	30.0	52.0	45.0	26.0	20.0	zh	
2	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	...	4.0	6.0	3.0	4.0	17.0	zh	
3	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	...	11.0	17.0	19.0	10.0	11.0	zh	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	11.0	27.0	13.0	36.0	10.0	zh	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145058	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	12.0	13.0	3.0	5.0	10.0	es	Underworld_(
145059	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	es	Resident_E
145060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	es	Enamorá
145061	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	es	Hasta
145062	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	es	Francisco_el_matemático_(serie

145063 rows × 555 columns

Moving columns

In [13]:

```
first_column=df1.pop('title')
df1.insert(0, 'title_', first_column)
first_column=df1.pop('Access')
df1.insert(0, 'access_', first_column)
first_column=df1.pop('Access_orgin')
df1.insert(0, 'access_orgin_', first_column)
first_column=df1.pop('language')
df1.insert(0, 'language_', first_column)
first_column=df1.pop('website')
df1.insert(0, 'website_', first_column)
```

In [14]:

```
df1
```

Out[14]:

	website_	language_	access_orgin_	access_	title_	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	...	2016-12-22	2016-12-23	2
0	wikipedia	zh	spider	all-access	2NE1	18.0	11.0	5.0	13.0	14.0	...	32.0	63.0	
1	wikipedia	zh	spider	all-access	2PM	11.0	14.0	15.0	18.0	11.0	...	17.0	42.0	
2	wikipedia	zh	spider	all-access	3C	1.0	0.0	1.0	1.0	0.0	...	3.0	1.0	
3	wikipedia	zh	spider	all-access	4minute	35.0	13.0	10.0	94.0	4.0	...	32.0	10.0	
4	wikipedia	zh	spider	all-access	52_Hz_I_Love_You	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145058	wikipedia	es	spider	all-access	Underworld_(serie_de_películas)	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145059	wikipedia	es	spider	all-access	Resident_Evil:_Capítulo_Final	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145060	wikipedia	es	spider	all-access	Enamorándome_de_Ramón	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145061	wikipedia	es	spider	all-access	Hasta_el_último_hombre	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145062	wikipedia	es	spider	all-access	Francisco_el_matemático_(serie_de_televisión_d...	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

145063 rows × 555 columns

In [15]:

```
df1.describe()
```

Out[15]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...
count	1.243230e+05	1.242470e+05	1.245190e+05	1.244090e+05	1.244040e+05	1.245800e+05	1.243990e+05	1.247690e+05	1.248190e+05	1.247210e+05	...
mean	1.195857e+03	1.204004e+03	1.133676e+03	1.170437e+03	1.217769e+03	1.290273e+03	1.239137e+03	1.193092e+03	1.197992e+03	1.189651e+03	...
std	7.275352e+04	7.421515e+04	6.961022e+04	7.257351e+04	7.379612e+04	8.054448e+04	7.576288e+04	6.820002e+04	7.149717e+04	7.214536e+04	...
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
25%	1.300000e+01	1.300000e+01	1.200000e+01	1.300000e+01	1.400000e+01	1.100000e+01	1.300000e+01	1.300000e+01	1.400000e+01	1.400000e+01	...
50%	1.090000e+02	1.080000e+02	1.050000e+02	1.050000e+02	1.130000e+02	1.130000e+02	1.150000e+02	1.170000e+02	1.150000e+02	1.130000e+02	...
75%	5.240000e+02	5.190000e+02	5.040000e+02	4.870000e+02	5.400000e+02	5.550000e+02	5.510000e+02	5.540000e+02	5.490000e+02	5.450000e+02	...
max	2.038124e+07	2.075219e+07	1.957397e+07	2.043964e+07	2.077211e+07	2.254467e+07	2.121089e+07	1.910791e+07	1.999385e+07	2.020182e+07	...

8 rows × 550 columns

In [16]:

```
df1.describe(include=object)
# There are 12 unique websites
# 2 unique access origin
# 3 unique access type
```

Out[16]:

	website_	language_	access_orgin_	access_	title_
count	145063	145063	145063	145063	145063
unique	12	17	2	3	49152
top	wikipedia	en	all-agents	all-access	Special:Search
freq	127208	23980	110150	74315	35

In [17]:

```
df1['website_'].value_counts()
# Wikipedia and wikimedia has the highest number of user visits
```

Out[17]:

```
wikipedia        127208
wikimedia         10537
mediawiki          7299
webmhd                4
theora                4
airlines              3
org                   3
mains                 1
with                  1
vattenkastare         1
geobae                1
welding               1
Name: website_, dtype: int64
```

In [18]:

```
df1['language_'].value_counts(normalize=True)
# 'en' and 'ja' has the highest number user visits
```

Out[18]:

```
en         0.165307
ja         0.140842
de         0.127827
commons    0.122816
fr         0.122719
zh         0.118769
ru         0.103555
es         0.097013
vs         0.001006
up         0.000034
in         0.000028
sk         0.000021
hq         0.000021
ca         0.000021
ft         0.000007
it         0.000007
bw         0.000007
Name: language_, dtype: float64
```

In [19]:

```
df1['access_orgin_'].value_counts(normalize=True)
# There are more count for 'all-agents'
```

Out[19]:

```
all-agents    0.759325
spider        0.240675
Name: access_orgin_, dtype: float64
```

In [20]:

```
df1['access_'].value_counts(normalize=True)
# There are more count for 'all-access'
```

Out[20]:

```
all-access    0.512295
mobile-web    0.247748
desktop       0.239958
Name: access_, dtype: float64
```

In [21]:

```
df1.isnull().sum()
# There are null values for each day.
# This signifies that this page was not active or this page did not exist in the server for this date
```

Out[21]:

```
website_            0
language_           0
access_orgin_       0
access_             0
title_              0
                   ...
2016-12-27       3701
2016-12-28       3822
2016-12-29       3826
2016-12-30       3635
2016-12-31       3465
Length: 555, dtype: int64
```

In [22]:

```
df1
```

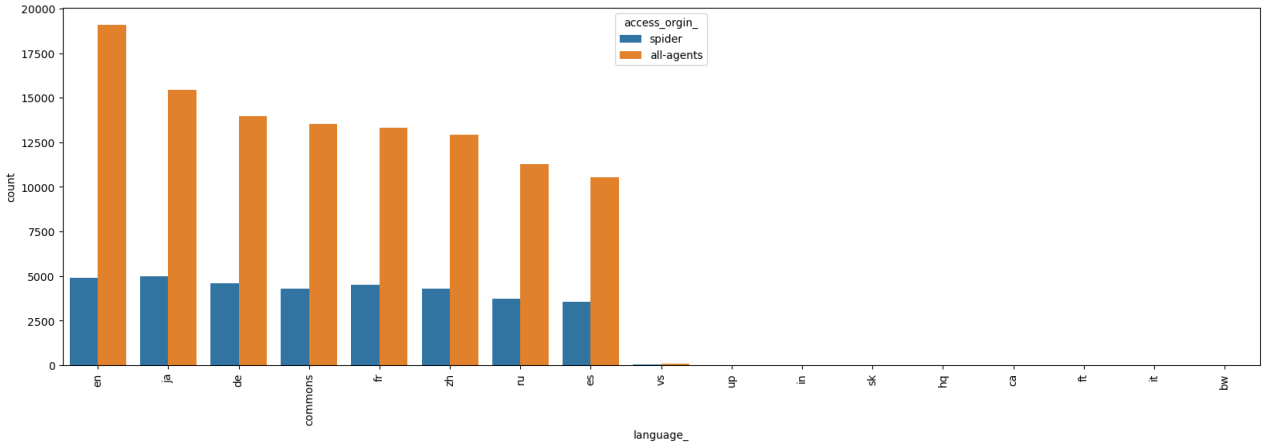
Out[22]:

	website_	language_	access_orgin_	access_	title_	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	...	2016-12-22	2016-12-23	2
0	wikipedia	zh	spider	all-access	2NE1	18.0	11.0	5.0	13.0	14.0	...	32.0	63.0	
1	wikipedia	zh	spider	all-access	2PM	11.0	14.0	15.0	18.0	11.0	...	17.0	42.0	
2	wikipedia	zh	spider	all-access	3C	1.0	0.0	1.0	1.0	0.0	...	3.0	1.0	
3	wikipedia	zh	spider	all-access	4minute	35.0	13.0	10.0	94.0	4.0	...	32.0	10.0	
4	wikipedia	zh	spider	all-access	52_Hz_I_Love_You	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145058	wikipedia	es	spider	all-access	Underworld_(serie_de_películas)	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145059	wikipedia	es	spider	all-access	Resident_Evil:_Capítulo_Final	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145060	wikipedia	es	spider	all-access	Enamorándome_de_Ramón	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145061	wikipedia	es	spider	all-access	Hasta_el_último_hombre	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
145062	wikipedia	es	spider	all-access	Francisco_el_matemático_(serie_de_televisión_d...	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

145063 rows × 555 columns

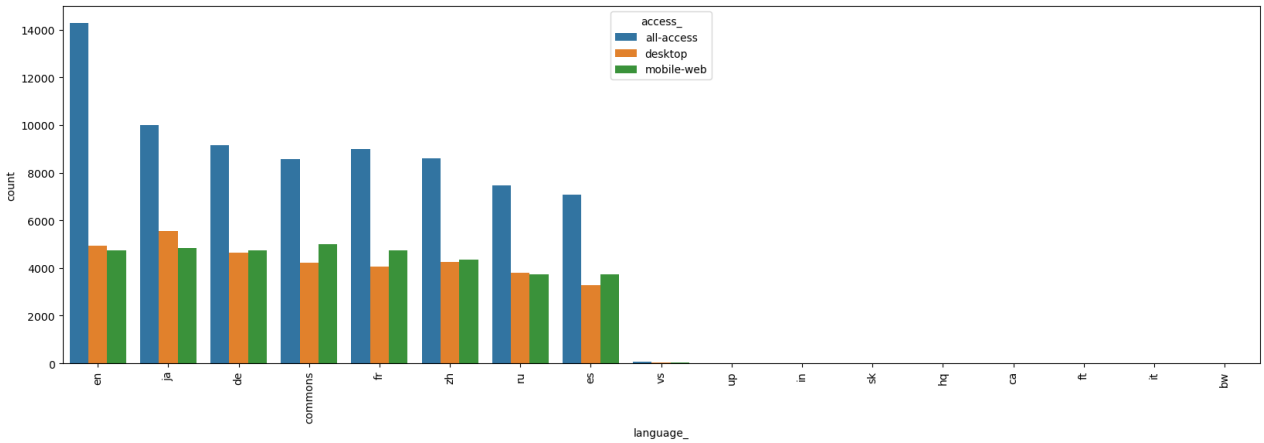
In [23]:

```
sns.countplot(data=df1, x='language_',order=df1['language_'].value_counts().index,hue='access_orgin_')
plt.xticks(rotation = 90)
plt.show()
# Below is the count plot for each languages
# 'en','ja','de' has the highest number visiting users
```



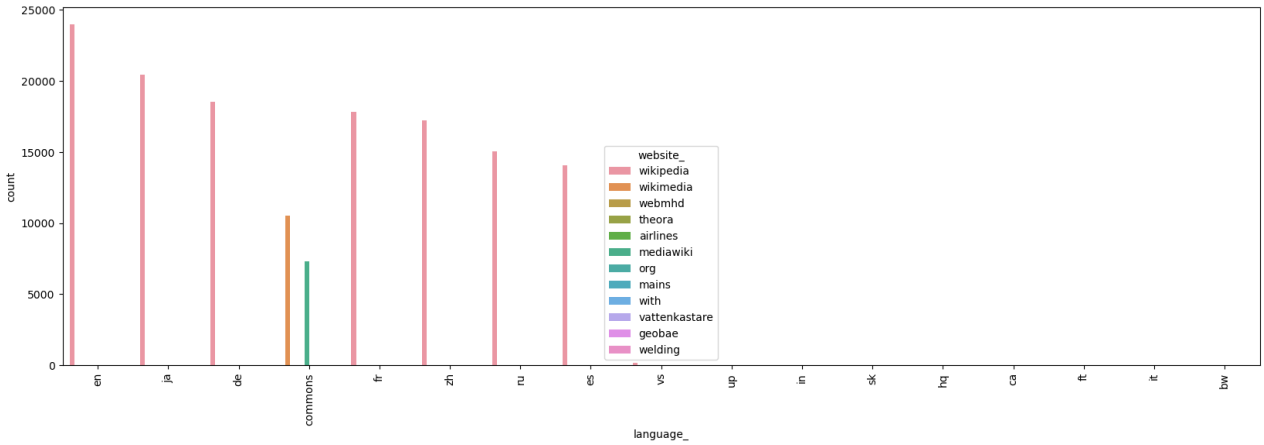
In [24]:

```
sns.countplot(data=df1, x='language_',order=df1['language_'].value_counts().index,hue='access_')
plt.xticks(rotation = 90)
plt.show()
# For all languages the count for 'all_access' is the most
# For 'de','commons','fr','zh', and 'es'
```



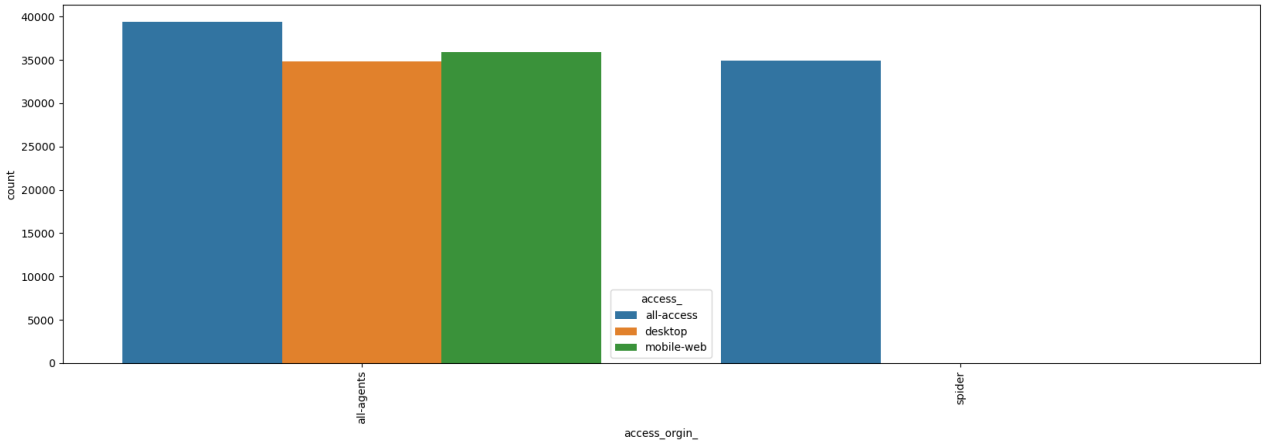
In [25]:

```
sns.countplot(data=df1, x='language_',order=df1['language_'].value_counts().index,hue='website_')
plt.xticks(rotation = 90)
plt.show()
# Maximum data that was collected is from 'wikipedia'
```



In [26]:

```
sns.countplot(data=df1, x='access_orgin_',order=df1['access_orgin_'].value_counts().index,hue='access_')
plt.xticks(rotation = 90)
plt.show()
# All access orgin from spider has 'all_access' and no desktop and mobile_web
```



In [27]:

```
df1['language_'].value_counts(normalize=True)
```

Out[27]:

```
en         0.165307
ja         0.140842
de         0.127827
commons    0.122816
fr         0.122719
zh         0.118769
ru         0.103555
es         0.097013
vs         0.001006
up         0.000034
in         0.000028
sk         0.000021
hq         0.000021
ca         0.000021
ft         0.000007
it         0.000007
bw         0.000007
Name: language_, dtype: float64
```

In [28]:

```
lang=str(input('Please enter the language '))
# Taking input from user for language
```

```
Please enter the language en
```

In [29]:

```
df1['language_'].value_counts()<150
# Checking for the languages that have count >150
```

Out[29]:

```
en         False
ja         False
de         False
commons    False
fr         False
zh         False
ru         False
es         False
vs          True
up          True
in          True
sk          True
hq          True
ca          True
ft          True
it          True
bw          True
Name: language_, dtype: bool
```

In [30]:

```
df3=df1.drop(['access_orgin_','access_','title_'],axis=1)
```

In [31]:

```
df3
```

Out[31]:

	website_	language_	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	20 12-
0	wikipedia	zh	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	...	32.0	63.0	15.0	26.0	14.0	20.0	22.0	19.0	18.0	2(
1	wikipedia	zh	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	...	17.0	42.0	28.0	15.0	9.0	30.0	52.0	45.0	26.0	2(
2	wikipedia	zh	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	...	3.0	1.0	1.0	7.0	4.0	4.0	6.0	3.0	4.0	1
3	wikipedia	zh	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	...	32.0	10.0	26.0	27.0	16.0	11.0	17.0	19.0	10.0	1
4	wikipedia	zh	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	25.0	13.0	3.0	11.0	27.0	13.0	36.0	1(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145058	wikipedia	es	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	13.0	12.0	13.0	3.0	5.0	1(
145059	wikipedia	es	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
145060	wikipedia	es	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
145061	wikipedia	es	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
145062	wikipedia	es	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

145063 rows × 552 columns

In [32]:

```
df4=df3.groupby('language_')[df3.columns[1:]].sum()
# Grouping all languages count
```

In [33]:

```
df4
```

Out[33]:

language_	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-22	20
bw	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	
ca	16.0	21.0	31.0	18.0	17.0	15.0	37.0	18.0	18.0	23.0	...	28.0	
commons	1490238.0	1561397.0	1475895.0	1259804.0	1396213.0	1601588.0	1694743.0	1768986.0	1543205.0	1559327.0	...	2348006.0	2
de	13260519.0	13079896.0	12554042.0	11520379.0	13392347.0	14741758.0	14296292.0	14505194.0	13964778.0	13014478.0	...	15370469.0	15
en	84708340.0	84433721.0	80163562.0	83458453.0	86194013.0	92804376.0	87833426.0	82875830.0	84794314.0	84314194.0	...	120400327.0	112
es	15278578.0	14601051.0	13427651.0	12606551.0	13710366.0	15625598.0	15230684.0	14781919.0	14502940.0	13184506.0	...	13837838.0	13
fr	8458638.0	8512952.0	8186030.0	8749842.0	8590493.0	8949799.0	8650800.0	8491533.0	8403646.0	7930703.0	...	11606988.0	11
ft	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
hq	20.0	39.0	29.0	18.0	17.0	17.0	47.0	24.0	25.0	33.0	...	42.0	
in	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.0	
it	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
ja	11863200.0	13620792.0	12305383.0	15456239.0	14827204.0	12920547.0	12568828.0	12492787.0	12178258.0	12652904.0	...	13793066.0	19
ru	9463854.0	9627643.0	8923463.0	8393214.0	8938528.0	9628896.0	9408180.0	9364117.0	9592309.0	10984872.0	...	13465000.0	13
sk	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
up	38.0	42.0	34.0	24.0	17.0	36.0	26.0	61.0	22.0	26.0	...	89.0	
vs	4047.0	5097.0	4419.0	4947.0	4871.0	5314.0	4889.0	4600.0	4777.0	5531.0	...	91517.0	
zh	4144988.0	4151189.0	4123659.0	4163448.0	4441286.0	4464290.0	4459421.0	4575842.0	4547843.0	4727889.0	...	5946850.0	5

17 rows × 550 columns

In [34]:

```
df4.T.isnull().sum()
# There are 0 null values in the dataframe
```

Out[34]:

```
language_
bw         0
ca         0
commons    0
de         0
en         0
es         0
fr         0
ft         0
hq         0
in         0
it         0
ja         0
ru         0
sk         0
up         0
vs         0
zh         0
dtype: int64
```

In [35]:

```
df5=df4.T.reset_index()
df5['index']=pd.to_datetime(df5['index'])
df5.set_index('index',inplace=True)
# Creating transpose of the dataframe and setting index to date
```

In [36]:

```
df5
```

Out[36]:

language_	bw	ca	commons	de	en	es	fr	ft	hq	in	it	ja	ru	sk	up	vs	
index																	
2015-07-01	0.0	16.0	1490238.0	13260519.0	84708340.0	15278578.0	8458638.0	0.0	20.0	0.0	0.0	11863200.0	9463854.0	0.0	38.0	4047.0	41449
2015-07-02	0.0	21.0	1561397.0	13079896.0	84433721.0	14601051.0	8512952.0	0.0	39.0	0.0	0.0	13620792.0	9627643.0	0.0	42.0	5097.0	41511
2015-07-03	0.0	31.0	1475895.0	12554042.0	80163562.0	13427651.0	8186030.0	0.0	29.0	0.0	0.0	12305383.0	8923463.0	0.0	34.0	4419.0	41236
2015-07-04	0.0	18.0	1259804.0	11520379.0	83458453.0	12606551.0	8749842.0	0.0	18.0	0.0	0.0	15456239.0	8393214.0	0.0	24.0	4947.0	41634
2015-07-05	0.0	17.0	1396213.0	13392347.0	86194013.0	13710366.0	8590493.0	0.0	17.0	0.0	0.0	14827204.0	8938528.0	0.0	17.0	4871.0	44412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2016-12-27	1.0	23.0	2625082.0	20125098.0	145535310.0	15945368.0	15281470.0	0.0	22.0	0.0	1.0	16123301.0	15040168.0	0.0	130.0	93693.0	64784
2016-12-28	2.0	17.0	3328588.0	19152196.0	141182641.0	16577375.0	13781521.0	0.0	31.0	3.0	0.0	16150715.0	14000319.0	0.0	94.0	96034.0	65134
2016-12-29	1.0	21.0	2629709.0	18447747.0	150457377.0	15647135.0	13399796.0	0.0	16.0	3.0	1.0	17682688.0	13478977.0	0.0	123.0	100430.0	60425
2016-12-30	2.0	25.0	2937528.0	17605880.0	125291741.0	11560095.0	12471074.0	0.0	17.0	6.0	0.0	19450687.0	12066750.0	0.0	80.0	113208.0	61112
2016-12-31	0.0	22.0	2569559.0	16562614.0	123380772.0	11077924.0	11504691.0	0.0	24.0	3.0	0.0	24460799.0	13223033.0	0.0	104.0	243684.0	62985

550 rows × 17 columns

In [37]:

```
data=pd.DataFrame(df5[lang])
# Extracting specific language which user had entered
```

In [38]:

```
data
```

Out[38]:

index	en
2015-07-01	84708340.0
2015-07-02	84433721.0
2015-07-03	80163562.0
2015-07-04	83458453.0
2015-07-05	86194013.0
...	...
2016-12-27	145535310.0
2016-12-28	141182641.0
2016-12-29	150457377.0
2016-12-30	125291741.0
2016-12-31	123380772.0

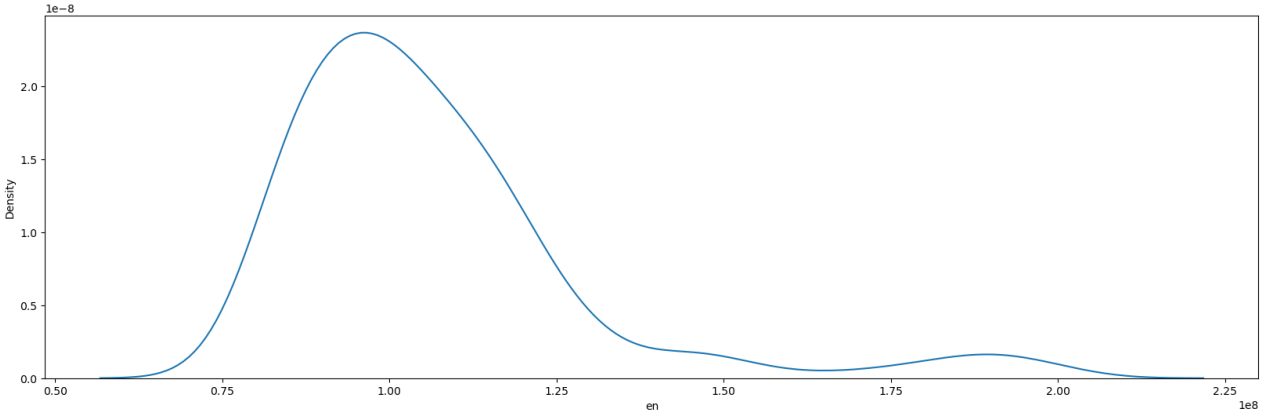
550 rows × 1 columns

In [39]:

```
sns.kdeplot(data[data.columns[0]])
# Below is the kde plot for the specific language
```

Out[39]:

```
<AxesSubplot:xlabel='en', ylabel='Density'>
```

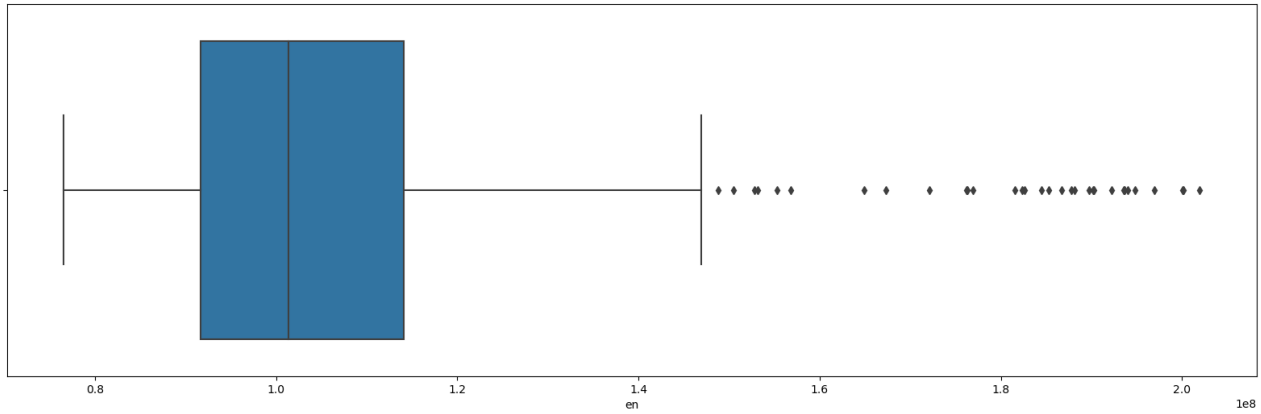


In [40]:

```
sns.boxplot(data[data.columns[0]])
# Boxplot shows there are few outliers
```

Out[40]:

```
<AxesSubplot:xlabel='en'>
```



In [41]:

```
df2.index=data.index
```

In [42]:

```
df2
# Setting index for exogenous dataframe
```

Out[42]:

index	Exog
2015-07-01	0
2015-07-02	0
2015-07-03	0
2015-07-04	0
2015-07-05	0
...	...
2016-12-27	1
2016-12-28	1
2016-12-29	1
2016-12-30	0
2016-12-31	0

550 rows × 1 columns

Augmented Dickey-Fuller test

H0: The time series is non-stationary. In other words, it has some time-dependent structure and does not have constant variance over time.

HA: The time series is stationary.

In [43]:

```
def test_full(x):
    p_value=adfuller(x)[1]
    sign_value=0.05
    if p_value>sign_value:
        return 'Time series is non-stationary'
    else:
        return 'Time series is stationary'
```

In [44]:

```
test_full(data)
# Orginal data is not stationary
```

Out[44]:

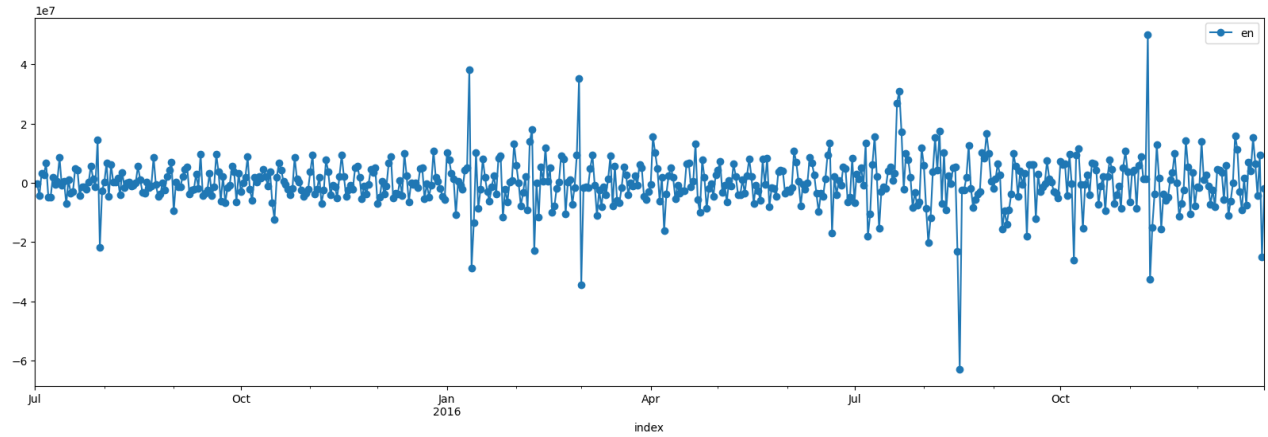
```
'Time series is non-stationary'
```

In [45]:

```
data.diff(1).plot(style='-o')
# Differencing once the data to make the data stationary
```

Out[45]:

```
<AxesSubplot:xlabel='index'>
```



In [46]:

```
test_full(data.diff(1).dropna())
# After differentiating the data is stationary
```

Out[46]:

```
'Time series is stationary'
```

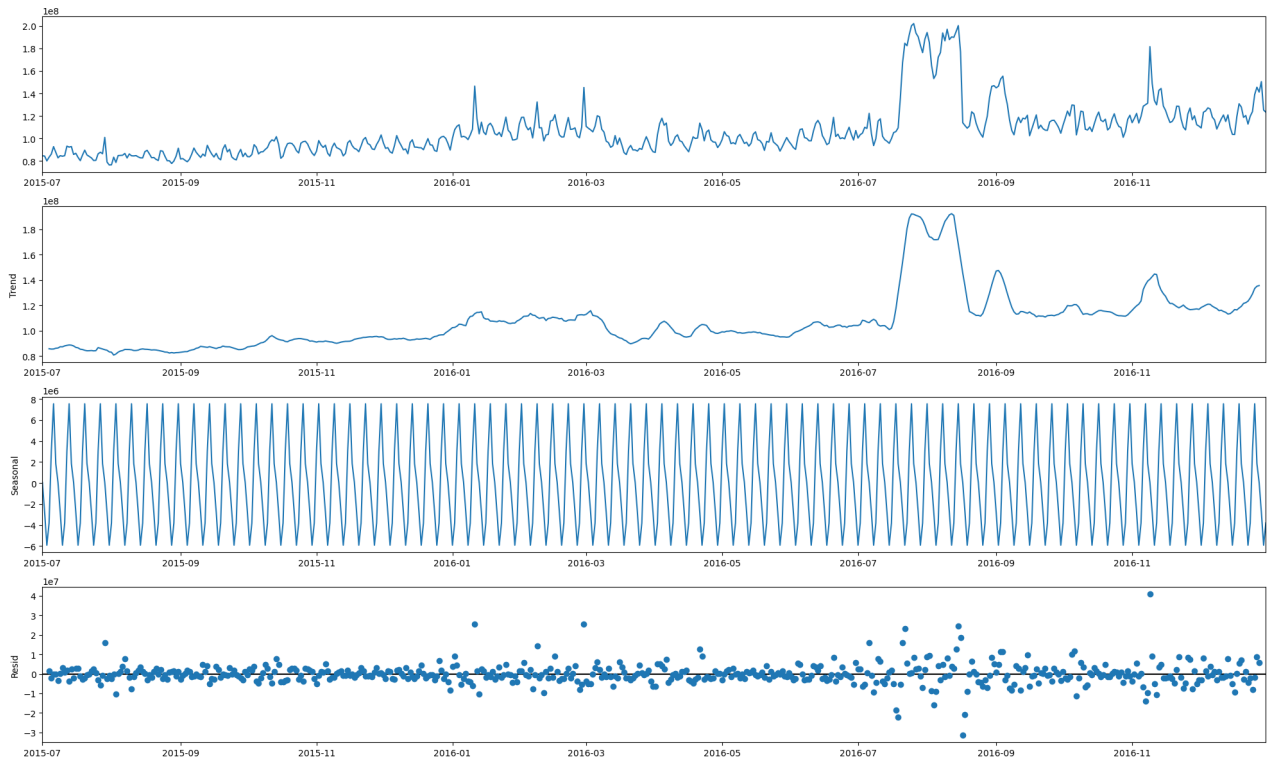
Decomposition Additive

In [47]:

```
model = sm.tsa.seasonal_decompose(data, model='additive')
```

In [48]:

```
plt.rcParams['figure.figsize'] = (20, 12)
model.plot();
# Data has seasonality and Trend
```



In [49]:

```
((model.resid**2).mean())**0.5
# RMSE of the residuals
```

Out[49]:

```
5406094.114679265
```

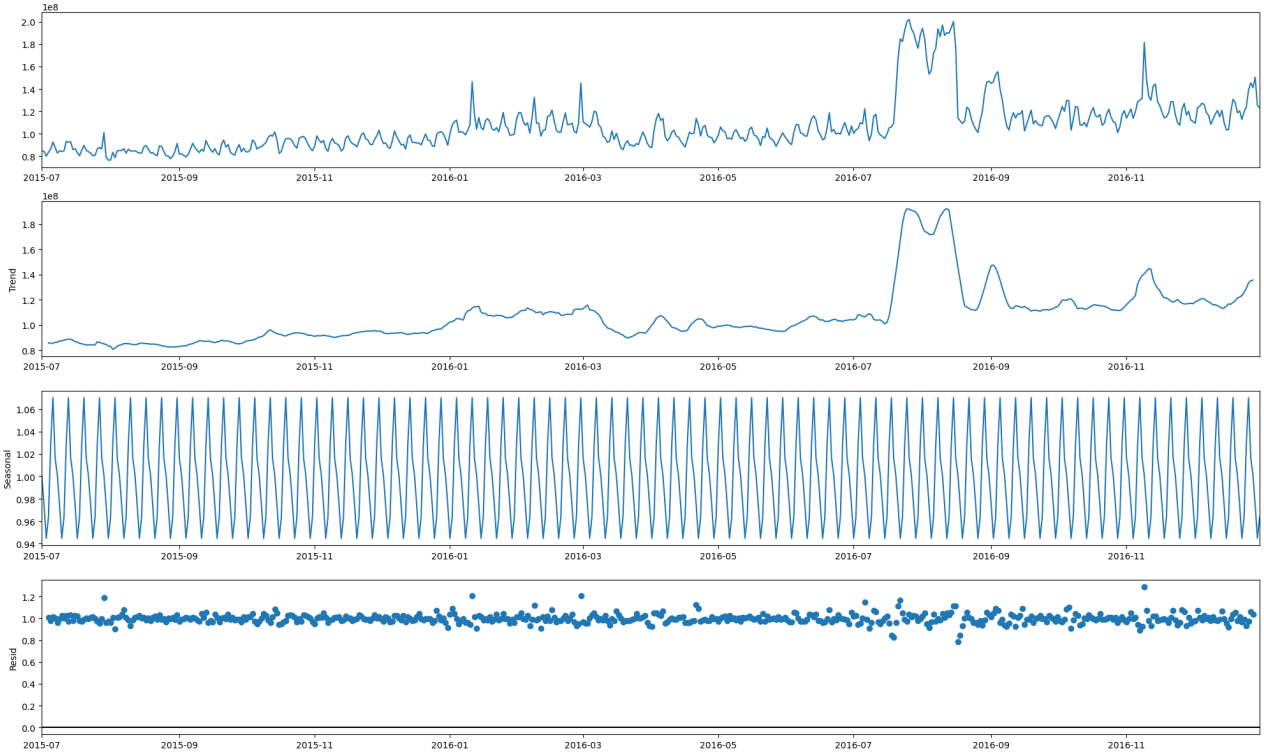
Decomposition Multiplicative

In [50]:

```
model = sm.tsa.seasonal_decompose(data, model='multiplicative')
```

In [51]:

```
plt.rcParams['figure.figsize'] = (20, 12)
model.plot();
# Data has seasonality and Trend
```



In [52]:

```
((model.resid**2).mean())**0.5
```

Out[52]:

```
1.0003498570678142
```

In [53]:

```
test_full(model.resid.dropna())
```

Out[53]:

```
'Time series is stationary'
```

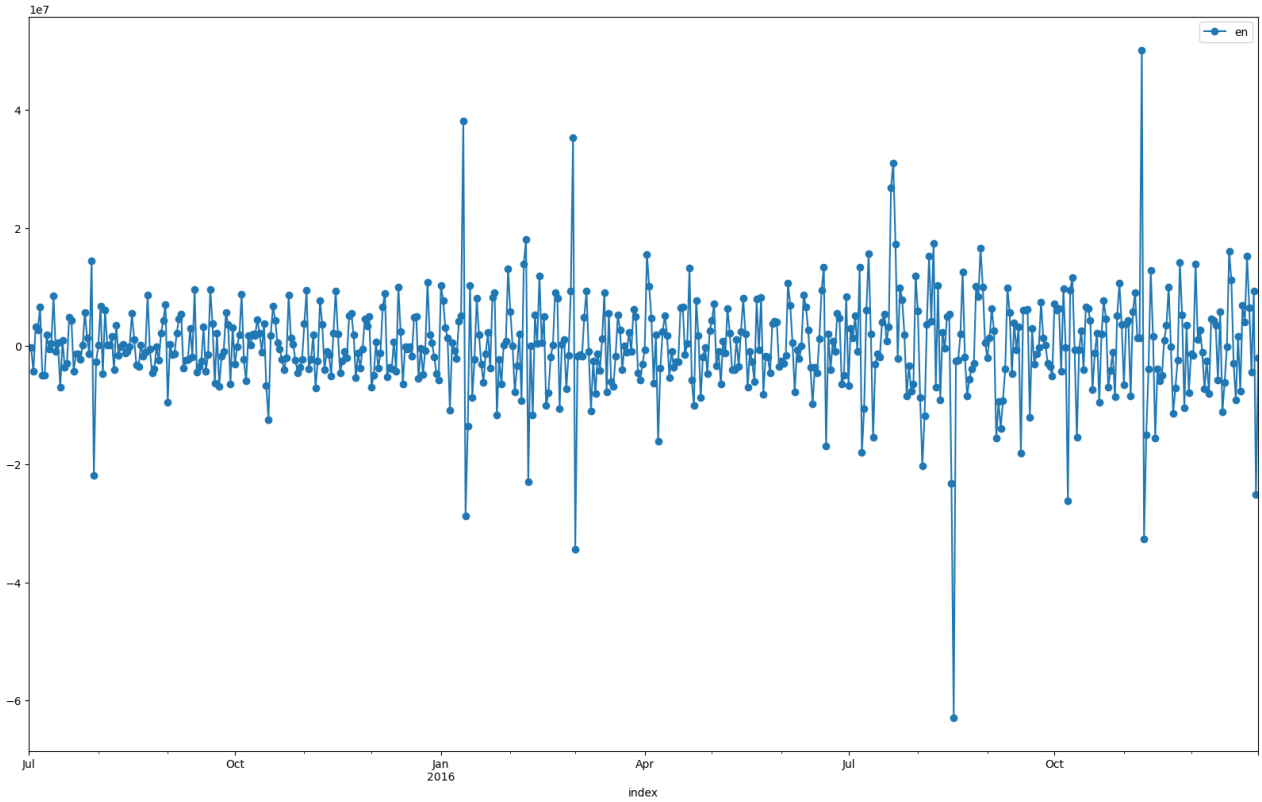
Finding d

In [54]:

```
data.diff(1).plot(style='-o')
```

Out[54]:

```
<AxesSubplot:xlabel='index'>
```



In [55]:

```
test_full(data.diff(1).dropna())
# Differentiating once will make the data stationary
```

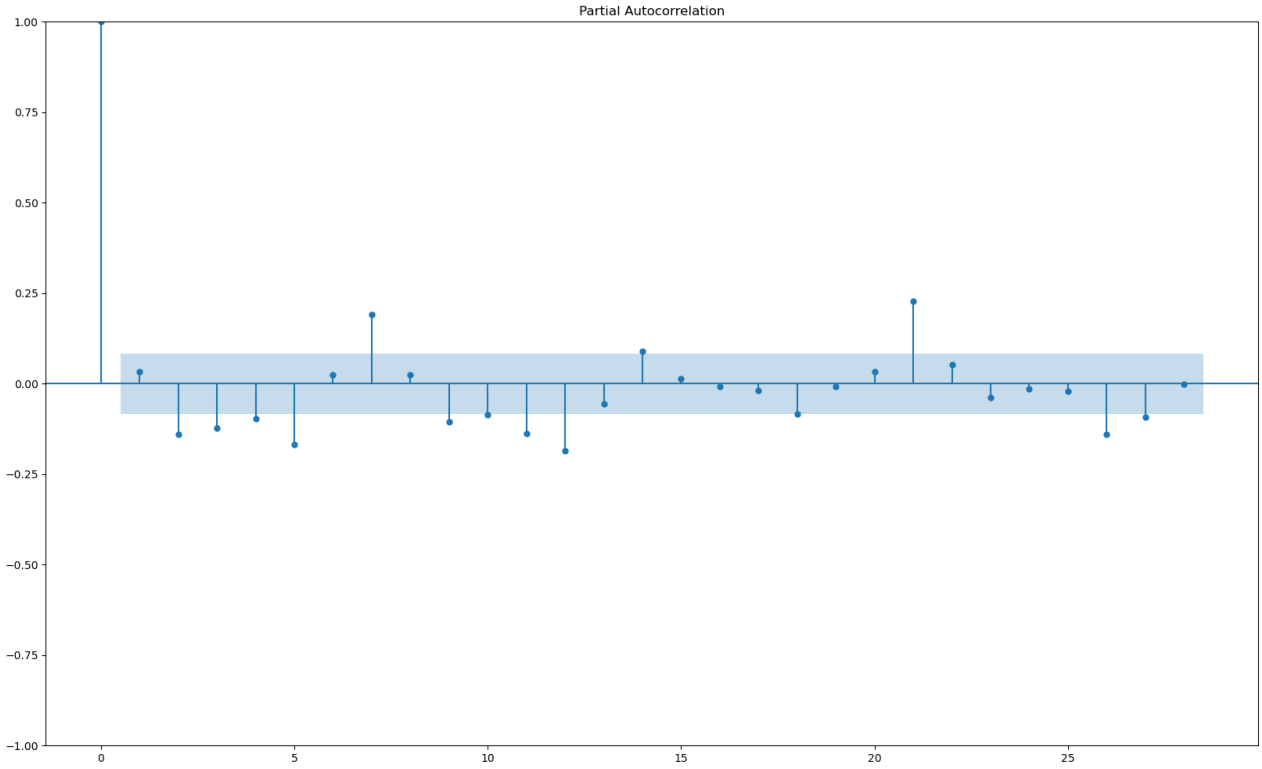
Out[55]:

```
'Time series is stationary'
```

Finding p from pacf plot

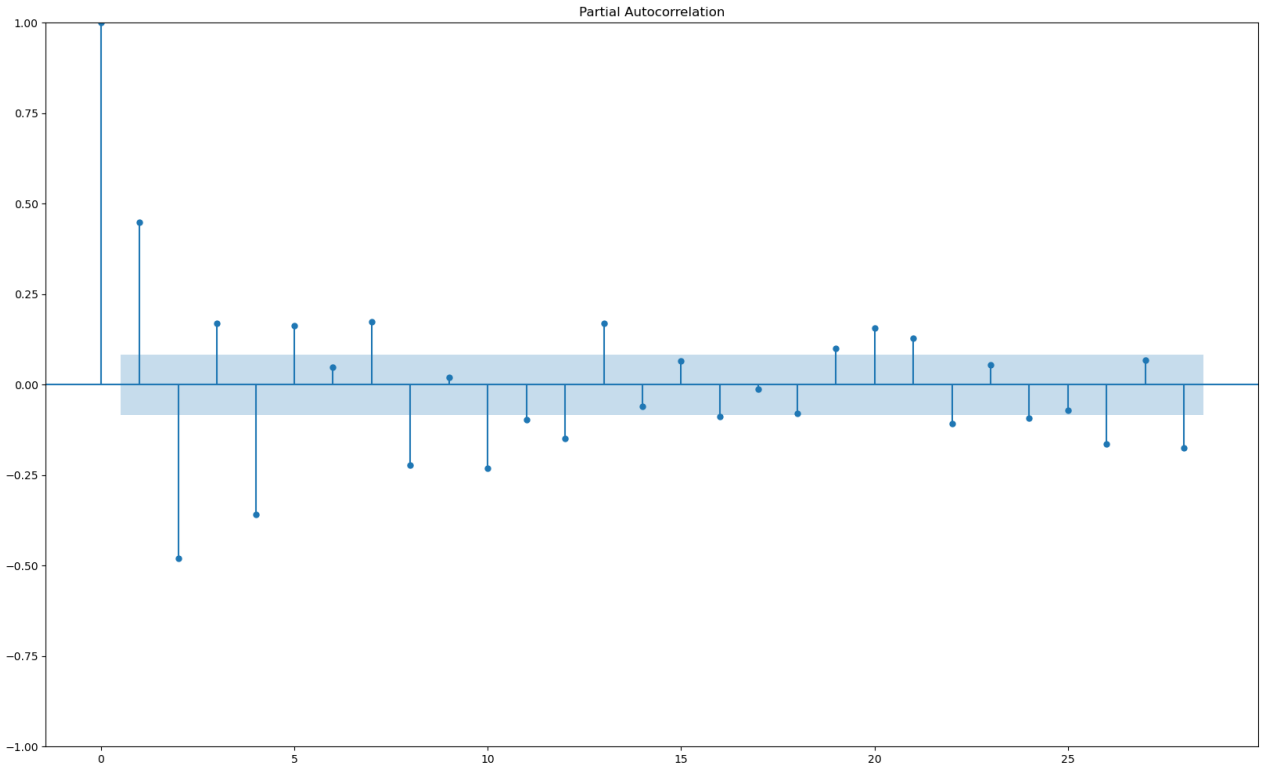
In [56]:

```
plot_pacf(data.diff(1).dropna());
# After 1 shift the change in insignificant
```



In [57]:

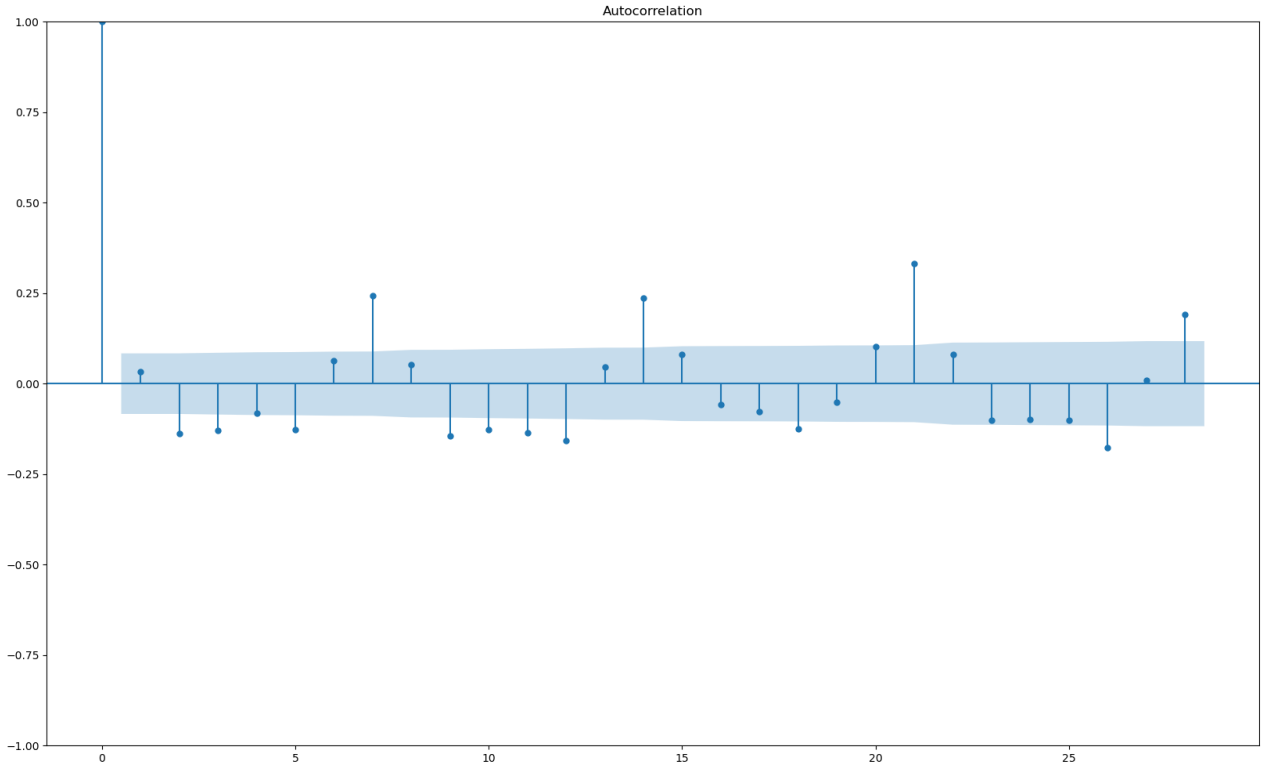
```
plot_pacf(data.diff(2).dropna());
# After 5 shift the change in insignificant when we differentiate data 2 times
```



Finding q from acf plot

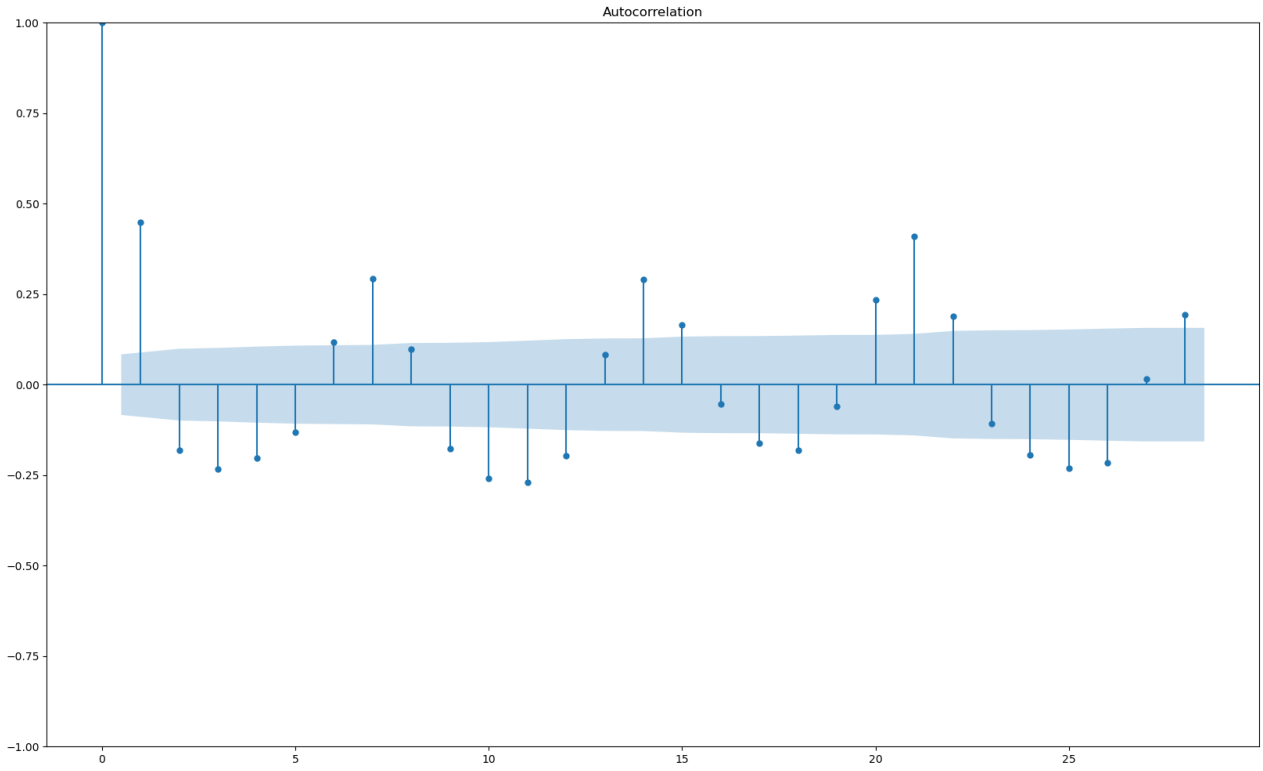
In [58]:

```
plot_acf(data.diff(1).dropna());
# After 1 shift the change in insignificant
```



In [59]:

```
plot_acf(data.diff(2).dropna());
# After 7 shift the change in insignificant when we differentiate data 2 times
```



ARIMA

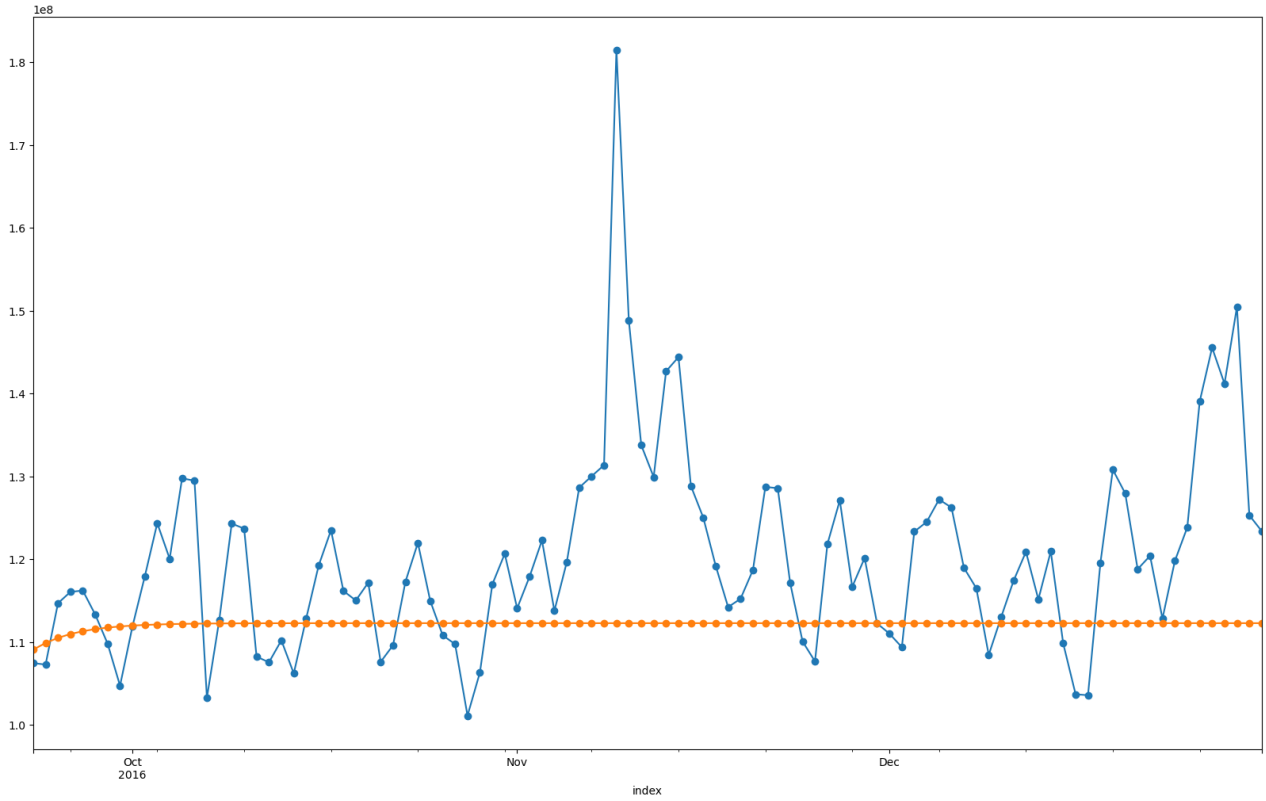
In [60]:

```
data_train=data[:-100]
data_test=data[-100:]
data_test.columns=['val']
model = SARIMAX(data_train, order=(2, 1, 2))
model = model.fit(disp=False)

data_test['pred'] = model.forecast(steps=len(data_test))

data_test['val'].plot(style='-o')
data_test['pred'].plot(style='-o')
performance(data_test['pred'], data_test['val'])
# MAPE error is 8.8%
```

```
MAE : 9885595.293
RMSE : 14157777.564
MAPE: 0.088
```



Grid Search

In [61]:

```
p=[1,2,3,4,5,6,7,8]
d=[1,2,3]
q=[1,2,3,4,5,6,7,8]
val_dict=dict()
for i in p:
    for j in d:
        for k in q:
            data_train=data[:-100]
            data_test=data[-100:]
            data_test.columns=['val']
            model = SARIMAX(data_train, order=(i,j,k))
            model = model.fit(disp=False)
            data_test['pred'] = model.forecast(steps=len(data_test))
            val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k)
print(sorted(val_dict.items(),key=lambda x: x[0])[0])
# Best order is (4,2,7)
```

```
(0.061, (4, 2, 7))
```

SARIMAX

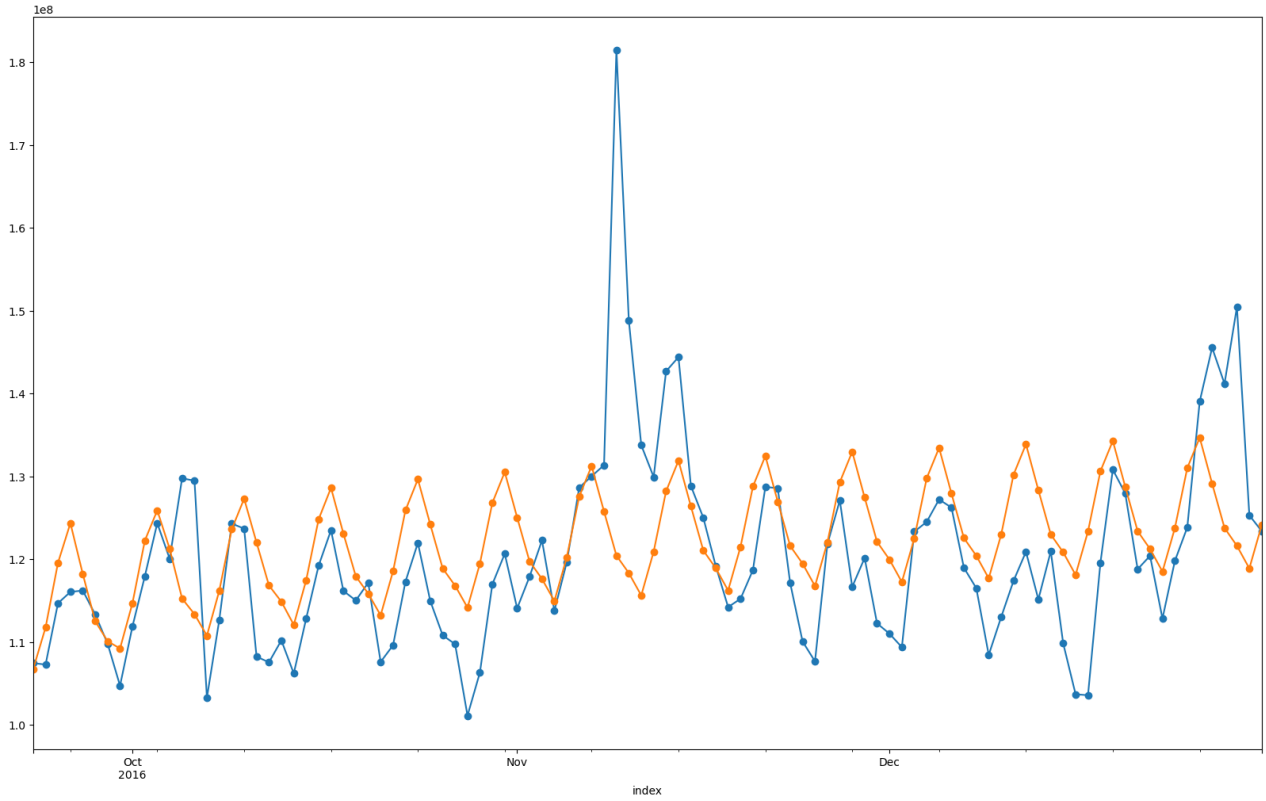
In [62]:

```
data_train=data[:-100]
data_test=data[-100:]
data_test.columns=['val']
model = SARIMAX(data_train, order=(2, 1, 2),seasonal_order=(2,1,1,7))
model = model.fit(disp=False)

data_test['pred'] = model.forecast(steps=len(data_test))

data_test['val'].plot(style='-o')
data_test['pred'].plot(style='-o')
performance(data_test['pred'], data_test['val'])
# Sarima has a MAPE error of 6.2%
```

```
MAE : 7607606.684
RMSE : 10939234.987
MAPE: 0.062
```



In [63]:

```
p=[1,2,3]
d=[1,2]
q=[1,2,3]
P=[1,2]
D=[1]
Q=[1,2]
s=[7]
val_dict=dict()
for i in p:
    for j in d:
        for k in q:
            for l in P:
                for m in D:
                    for n in Q:
                        for o in s:
                            data_train=data[:-100]
                            data_test=data[-100:]
                            data_test.columns=['val']
                            model = SARIMAX(data_train, order=(i,j,k),seasonal_order=(l,m,n,o))
                            model = model.fit(disp=False)
                            data_test['pred'] = model.forecast(steps=len(data_test))
                            val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k,l,m,n,o)
```

In [64]:

```
sorted(val_dict.items(),key=lambda x: x[0])[0][0],sorted(val_dict.items(),key=lambda x: x[0])[0][1]
# MAPE error for SARIMAX is 0.062 and its order is (2,1,2) and seasonal_order is (2,1,1,7)
```

Out[64]:

```
(0.062, (2, 1, 2, 2, 1, 1, 7))
```

In [65]:

```
p,d,q,P,D,Q,s=sorted(val_dict.items(),key=lambda x: x[0])[0][1]
```

SARIMAX with Exogenous

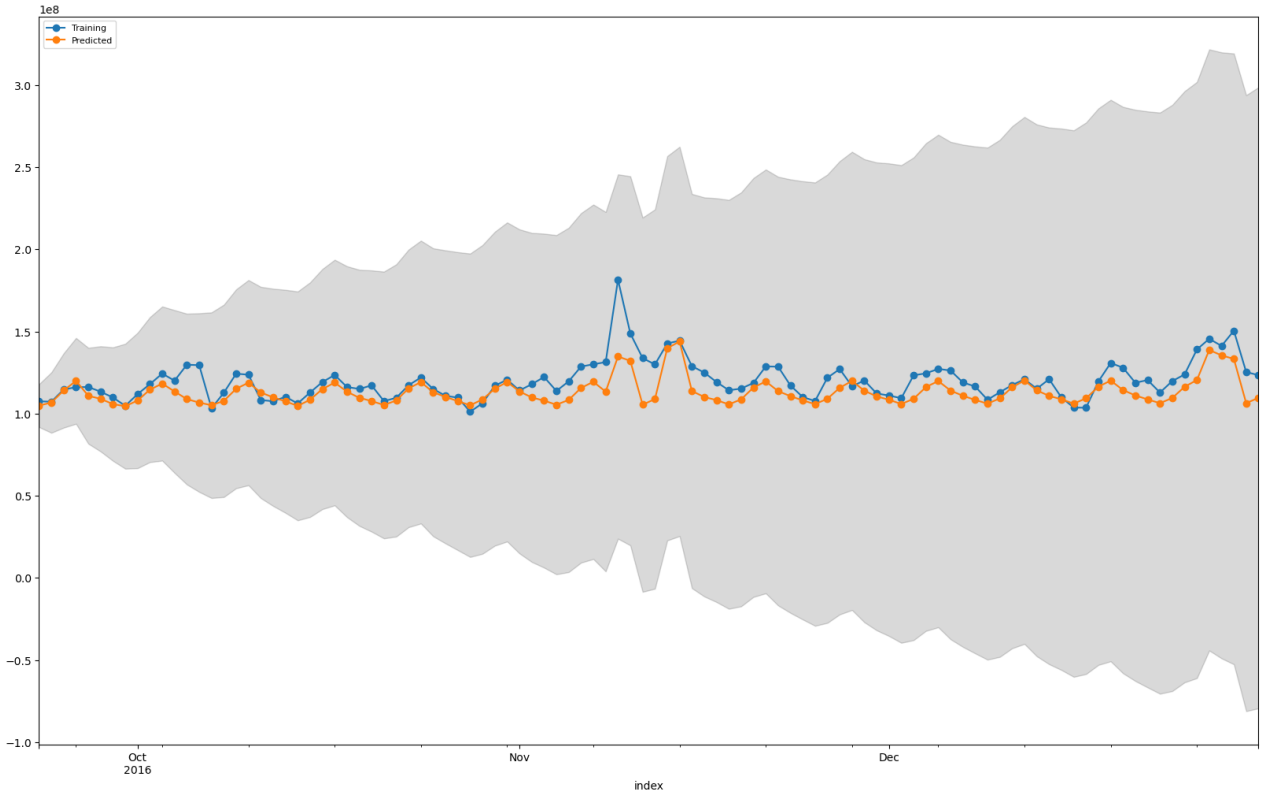
In [66]:

```
data_train=data[:-100]
data_test=data[-100:]
data_test.columns=['val']
model = SARIMAX(data_train, order=(p,d,q),seasonal_order=(P,D,Q,s),exog=df2[:-100])
model = model.fit(disp=False)
data_test[['lower', 'upper']] = model.get_forecast(steps=len(data_test), exog=df2[-100:]).conf_int(alpha=0.1).values

data_test['pred'] = model.forecast(steps=len(data_test),exog=df2[-100:])

data_test['val'].plot(style='-o',label='Training')
data_test['pred'].plot(style='-o',label='Predicted')
performance(data_test['pred'], data_test['val'])
plt.fill_between(data_test.index, data_test['lower'], data_test['upper'], color='k', alpha=.15)
plt.legend(loc='upper left', fontsize=8)
plt.show()
# MAPE error is 6.6%
# Upper
```

```
MAE : 7463956.35
RMSE : 10267077.282
MAPE: 0.066
```



Grid search SARIMAX with Exogenous

In [67]:

```
p=[2,3,4]
d=[1,2]
q=[2,3,4]
P=[1,2]
D=[1]
Q=[1,2]
s=[7]
val_dict=dict()
for i in p:
    for j in d:
        for k in q:
            for l in P:
                for m in D:
                    for n in Q:
                        for o in s:
                            data_train=data[:-100]
                            data_test=data[-100:]
                            data_test.columns=['val']
                            model = SARIMAX(data_train, order=(i,j,k),seasonal_order=(l,m,n,o),exog=df2[:-100])
                            model = model.fit(disp=False)
                            data_test['pred'] = model.forecast(steps=len(data_test),exog=df2[-100:])
                            val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k,l,m,n,o)
```

In [68]:

```
sorted(val_dict.items(),key=lambda x: x[0])[0][0],sorted(val_dict.items(),key=lambda x: x[0])[0][1]
# MAPE error with SARIMAX is 4.4 %
```

Out[68]:

```
(0.044, (3, 1, 3, 1, 1, 1, 7))
```

Prophet

In [69]:

```
df=pd.DataFrame()
```

In [70]:

```
df['ds'] = pd.to_datetime(data.index)
df['y'] = data[data.columns[0]].values
df['holiday']=df2['Exog'].values
df = df[['ds', 'y', 'holiday']]
df.head()
# Creating Data Frame
```

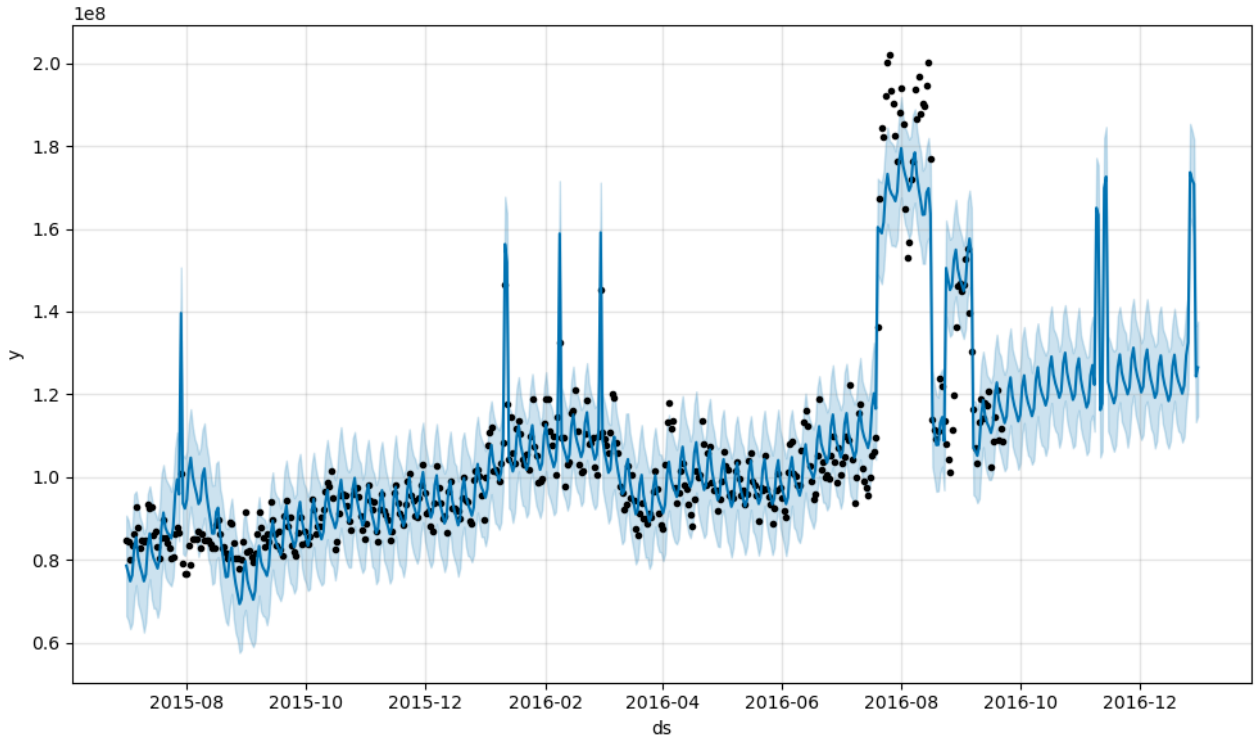
Out[70]:

	ds	y	holiday
0	2015-07-01	84708340.0	0
1	2015-07-02	84433721.0	0
2	2015-07-03	80163562.0	0
3	2015-07-04	83458453.0	0
4	2015-07-05	86194013.0	0

In [71]:

```
model2=Prophet(yearly_seasonality=True, weekly_seasonality=True)
model2.add_regressor('holiday') #adding holidays data in the model3
model2.fit(df[:-100])
forecast2 = model2.predict(df)
fig = model2.plot(forecast2)
# Future prediction
```

```
08:33:48 - cmdstanpy - INFO - Chain [1] start processing
08:33:49 - cmdstanpy - INFO - Chain [1] done processing
```



In [72]:

```
forecast2
```

Out[72]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	extra_regr
0	2015-07-01	8.064953e+07	6.644574e+07	9.071795e+07	8.064953e+07	8.064953e+07	-2.040235e+06	-2.040235e+06	-2.040235e+06	
1	2015-07-02	8.072336e+07	6.538028e+07	8.957595e+07	8.072336e+07	8.072336e+07	-3.850562e+06	-3.850562e+06	-3.850562e+06	
2	2015-07-03	8.079720e+07	6.332324e+07	8.708967e+07	8.079720e+07	8.079720e+07	-5.958478e+06	-5.958478e+06	-5.958478e+06	
3	2015-07-04	8.087103e+07	6.469113e+07	8.757865e+07	8.087103e+07	8.087103e+07	-4.595126e+06	-4.595126e+06	-4.595126e+06	
4	2015-07-05	8.094487e+07	7.080792e+07	9.589162e+07	8.094487e+07	8.094487e+07	2.003475e+06	2.003475e+06	2.003475e+06	
...	...	...	...	...	...	...	...	...	...	
545	2016-12-27	1.253585e+08	1.625958e+08	1.853716e+08	1.252852e+08	1.254374e+08	4.828223e+07	4.828223e+07	4.828223e+07	
546	2016-12-28	1.254410e+08	1.609629e+08	1.837384e+08	1.253666e+08	1.255212e+08	4.649192e+07	4.649192e+07	4.649192e+07	
547	2016-12-29	1.255235e+08	1.585636e+08	1.816774e+08	1.254477e+08	1.256049e+08	4.531434e+07	4.531434e+07	4.531434e+07	
548	2016-12-30	1.256060e+08	1.133208e+08	1.365430e+08	1.255283e+08	1.256888e+08	-1.265342e+06	-1.265342e+06	-1.265342e+06	
549	2016-12-31	1.256885e+08	1.147716e+08	1.376477e+08	1.256102e+08	1.257724e+08	7.957431e+05	7.957431e+05	7.957431e+05	

550 rows × 25 columns

In [73]:

```
performance(df['y'][:-100],forecast2['yhat'][:-100])
# Error for train data
```

```
MAE : 5915510.238
RMSE : 9170962.209
MAPE: 0.055
```

In [74]:

```
performance(df['y'][-100:],forecast2['yhat'][-100:])
# Error for test data
```

```
MAE : 8101037.547
RMSE : 10214134.131
MAPE: 0.067
```

In [ ]:

```

```

Pipeline for multiple series

In [75]:

```
lang=str(input('Please enter the language '))
```

```
Please enter the language de
```

In [76]:

```
def kde(data):
    print('-----------------------------------------')
    sns.kdeplot(data[data.columns[0]])
    plt.show()
    print('-----------------------------------------')
```

In [77]:

```
def test(data):
    print('-----------------------------------------')
    print('Dickey-Fuller Test ')
    print(test_full(data))
    print('-----------------------------------------')
```

In [78]:

```
def test(data):
    print('-----------------------------------------')
    print('Dickey-Fuller Test ')
    print(test_full(data))
    print('-----------------------------------------')
```

In [79]:

```
def diff_1(data):
    print('Diff 1 order')
    print('-----------------------------------------')
    data.diff(1).plot(style='-o')
    plt.show()
    test(data.diff(1).dropna())
    print('-----------------------------------------')
```

In [80]:

```
def dec_mul(data):
    print('-----------------------------------------')
    print('Decomposition (Multiplicaative)')
    model = sm.tsa.seasonal_decompose(data, model='multiplicative')
    plt.rcParams['figure.figsize'] = (20, 12)
    model.plot();
    plt.show()
    test(model.resid.dropna())
    print('-----------------------------------------')
```

In [81]:

```
def pacf(data):
    print('pacf diff 1 ----------------------------------------')
    plot_pacf(data.diff(1).dropna());
    plt.show()
    print('----------------------------------------')
    print('pacf diff 2 ----------------------------------------')
    plot_pacf(data.diff(2).dropna());
    plt.show()
    print('----------------------------------------')
```

In [82]:

```
def acf(data):
    print('acf diff 1 ----------------------------------------')
    plot_acf(data.diff(1).dropna());
    plt.show()
    print('----------------------------------------')
    print('acf diff 2 ----------------------------------------')
    plot_acf(data.diff(2).dropna());
    plt.show()
    print('----------------------------------------')
```

In [83]:

```
def arima(data):
    print('ARIMA----------------------------------------')
    data_train=data[:-100]
    data_test=data[-100:]
    data_test.columns=['val']
    model = SARIMAX(data_train, order=(2, 1, 2))
    model = model.fit(disp=False)

    data_test['pred'] = model.forecast(steps=len(data_test))

    data_test['val'].plot(style='-o')
    data_test['pred'].plot(style='-o')
    plt.show()
    performance(data_test['pred'], data_test['val'])
    print('----------------------------------------')
```

In [84]:

```
def arima_gridsearch(data):
    print('ARIMA grid search----------------------------------------')
    p=[1,2,3,4,5,6,7,8]
    d=[1,2,3]
    q=[1,2,3,4,5,6,7,8]
    val_dict=dict()
    for i in p:
        for j in d:
            for k in q:
                data_train=data[:-100]
                data_test=data[-100:]
                data_test.columns=['val']
                model = SARIMAX(data_train, order=(i,j,k))
                model = model.fit(disp=False)
                data_test['pred'] = model.forecast(steps=len(data_test))
                val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k)
    print(sorted(val_dict.items(),key=lambda x: x[0])[0])
    print('----------------------------------------')
```

In [85]:

```
def sarimax(data):
    print('SARIMAX grid ----------------------------------------')
    data_train=data[:-100]
    data_test=data[-100:]
    data_test.columns=['val']
    model = SARIMAX(data_train, order=(2, 1, 2),seasonal_order=(2,1,1,7))
    model = model.fit(disp=False)

    data_test['pred'] = model.forecast(steps=len(data_test))

    data_test['val'].plot(style='-o')
    data_test['pred'].plot(style='-o')
    plt.show()
    performance(data_test['pred'], data_test['val'])
    print('----------------------------------------')
```

In [86]:

```
def sarimax_grid(data):
    print('SARIMAX grid search----------------------------------------')
    p=[1,2,3]
    d=[1,2]
    q=[1,2,3]
    P=[1,2]
    D=[1]
    Q=[1,2]
    s=[7]
    val_dict=dict()
    for i in p:
        for j in d:
            for k in q:
                for l in P:
                    for m in D:
                        for n in Q:
                            for o in s:
                                data_train=data[:-100]
                                data_test=data[-100:]
                                data_test.columns=['val']
                                model = SARIMAX(data_train, order=(i,j,k),seasonal_order=(l,m,n,o))
                                model = model.fit(disp=False)
                                data_test['pred'] = model.forecast(steps=len(data_test))
                                val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k,l,m,n,o)
    print(sorted(val_dict.items(),key=lambda x: x[0])[0][1])
    print('----------------------------------------')
    p,d,q,P,D,Q,s=sorted(val_dict.items(),key=lambda x: x[0])[0][1]
    return p,d,q,P,D,Q,s
```

In [87]:

```
def sari_exo(data,p,d,q,P,D,Q,s):
    print('sari_exo----------------------------------------')
    data_train=data[:-100]
    data_test=data[-100:]
    data_test.columns=['val']
    model = SARIMAX(data_train, order=(p,d,q),seasonal_order=(P,D,Q,s),exog=df2[:-100])
    model = model.fit(disp=False)
    data_test[['lower', 'upper']] = model.get_forecast(steps=len(data_test), exog=df2[-100:]).conf_int(alpha=0.1).values

    data_test['pred'] = model.forecast(steps=len(data_test),exog=df2[-100:])

    data_test['val'].plot(style='-o',label='Training')
    data_test['pred'].plot(style='-o',label='Predicted')
    performance(data_test['pred'], data_test['val'])
    plt.fill_between(data_test.index, data_test['lower'], data_test['upper'], color='k', alpha=.15)
    plt.legend(loc='upper left', fontsize=8)
    plt.show()
    print('----------------------------------------')
```

In [88]:

```
def sari_exo_grid(data):
    print('SARIMAX EXO GRID----------------------------------------')
    p=[1,2,3]
    d=[1,2]
    q=[1,2,3]
    P=[1,2]
    D=[1]
    Q=[1,2]
    s=[7]
    val_dict=dict()
    for i in p:
        for j in d:
            for k in q:
                for l in P:
                    for m in D:
                        for n in Q:
                            for o in s:
                                data_train=data[:-100]
                                data_test=data[-100:]
                                data_test.columns=['val']
                                model = SARIMAX(data_train, order=(i,j,k),seasonal_order=(l,m,n,o),exog=df2[:-100])
                                model = model.fit(disp=False)
                                data_test['pred'] = model.forecast(steps=len(data_test),exog=df2[-100:])
                                val_dict[performance_mape(data_test['pred'], data_test['val'])]=(i,j,k,l,m,n,o)
    print('----------------------------------------------------------')
    print('MAPE and parameters is')
    print(sorted(val_dict.items(),key=lambda x: x[0])[0][0])
    print(sorted(val_dict.items(),key=lambda x: x[0])[0][1])
```

In [89]:

```
def pro(data):
    print('PROPHET-----------------------------------------')
    df=pd.DataFrame()
    df['ds'] = pd.to_datetime(data.index)
    df['y'] = data[data.columns[0]].values
    df['holiday']=df2['Exog'].values
    df = df[['ds', 'y', 'holiday']]
    model2=Prophet(yearly_seasonality=True, weekly_seasonality=True)
    model2.add_regressor('holiday') #adding holidays data in the model3
    model2.fit(df[:-100])
    forecast2 = model2.predict(df)
    fig = model2.plot(forecast2)
    plt.show()
    print('Error with train data is')
    performance(df['y'][:-100],forecast2['yhat'][:-100])
    print('Error with test data is')
    performance(df['y'][-100:],forecast2['yhat'][-100:])
    print('-----------------------------------------')
```

In [90]:

```
def pipeline(lang):
    data=pd.DataFrame(df5[lang])
    kde(data)
    test(data)
    diff_1(data)
    dec_mul(data)
    pacf(data)
    acf(data)
    arima(data)
    arima_gridsearch(data)
    sarimax(data)
    sari_exo_grid(data)
    pro(data)
```

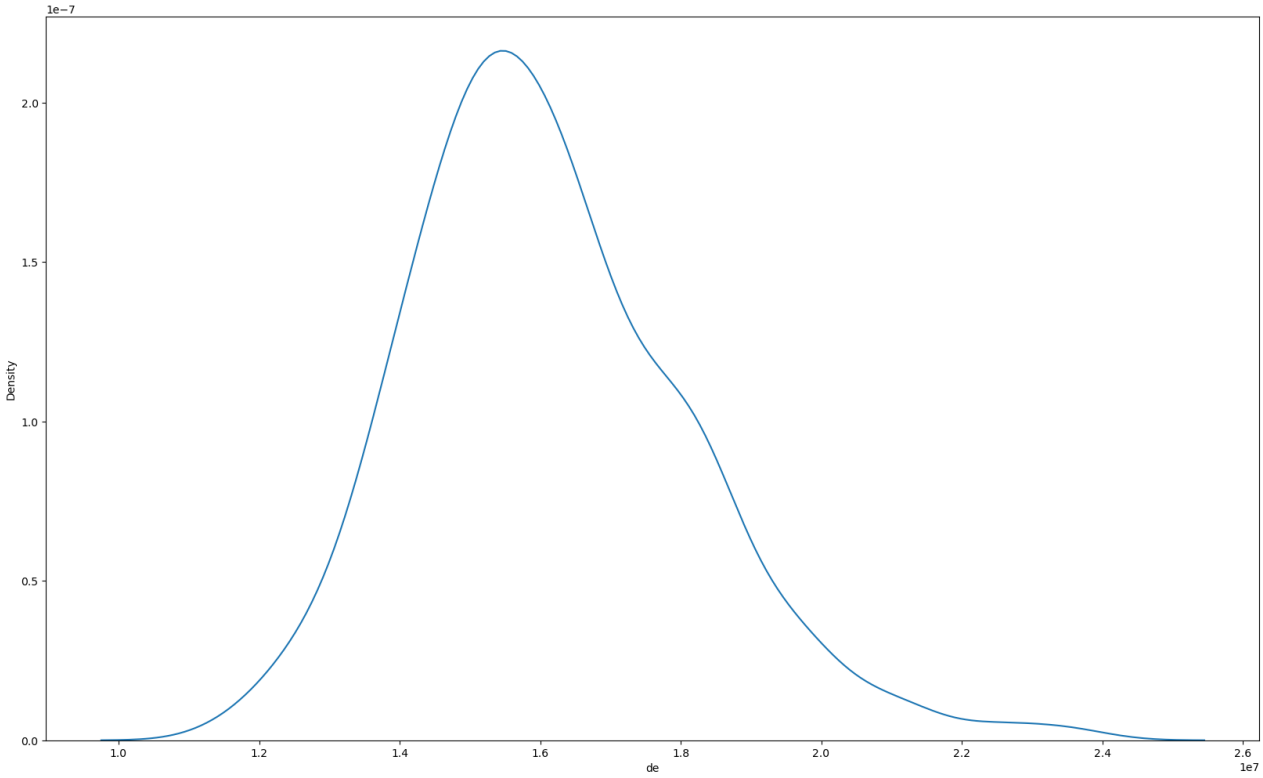
MAPE error for 'de' language is 6.1% and its parameters are (3, 2, 3) (2, 1, 2, 7) with exo

PROPHET has a MAPE error of 8%

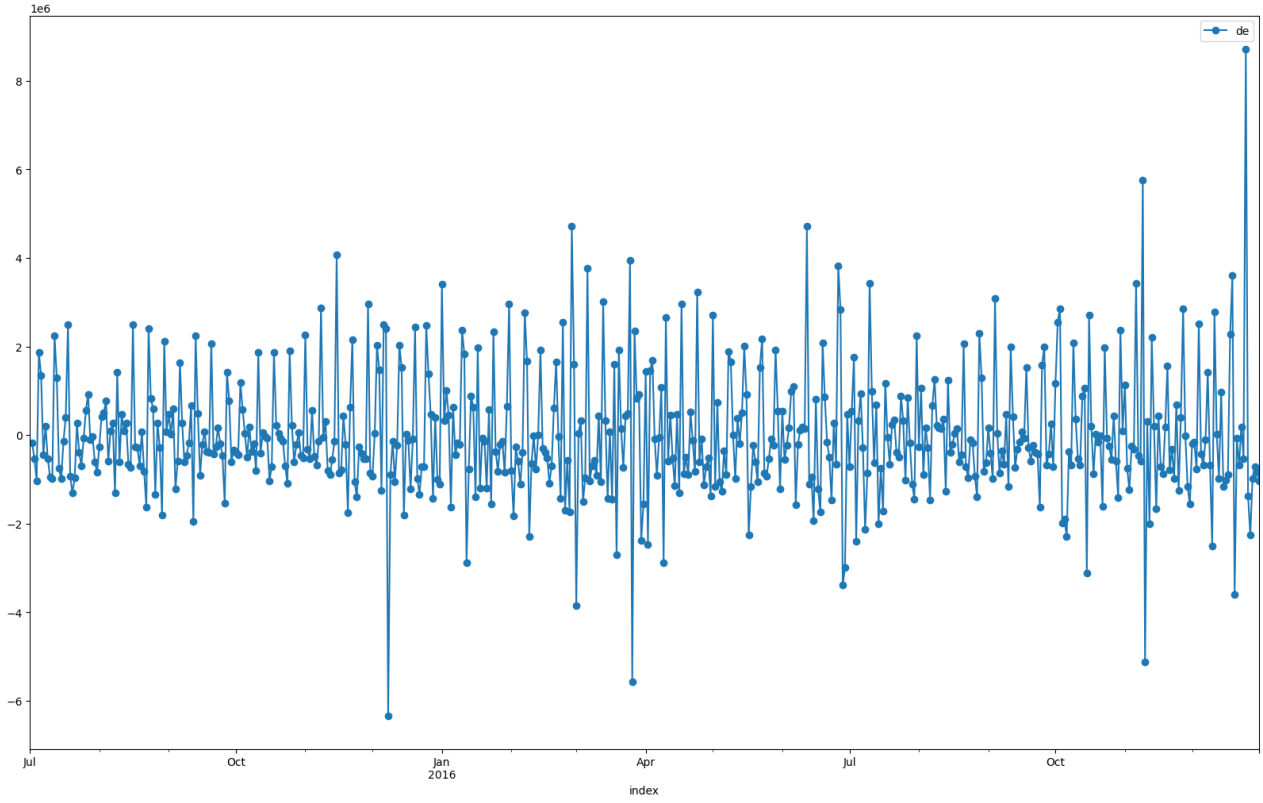
In [91]:

```
pipeline(lang)
```

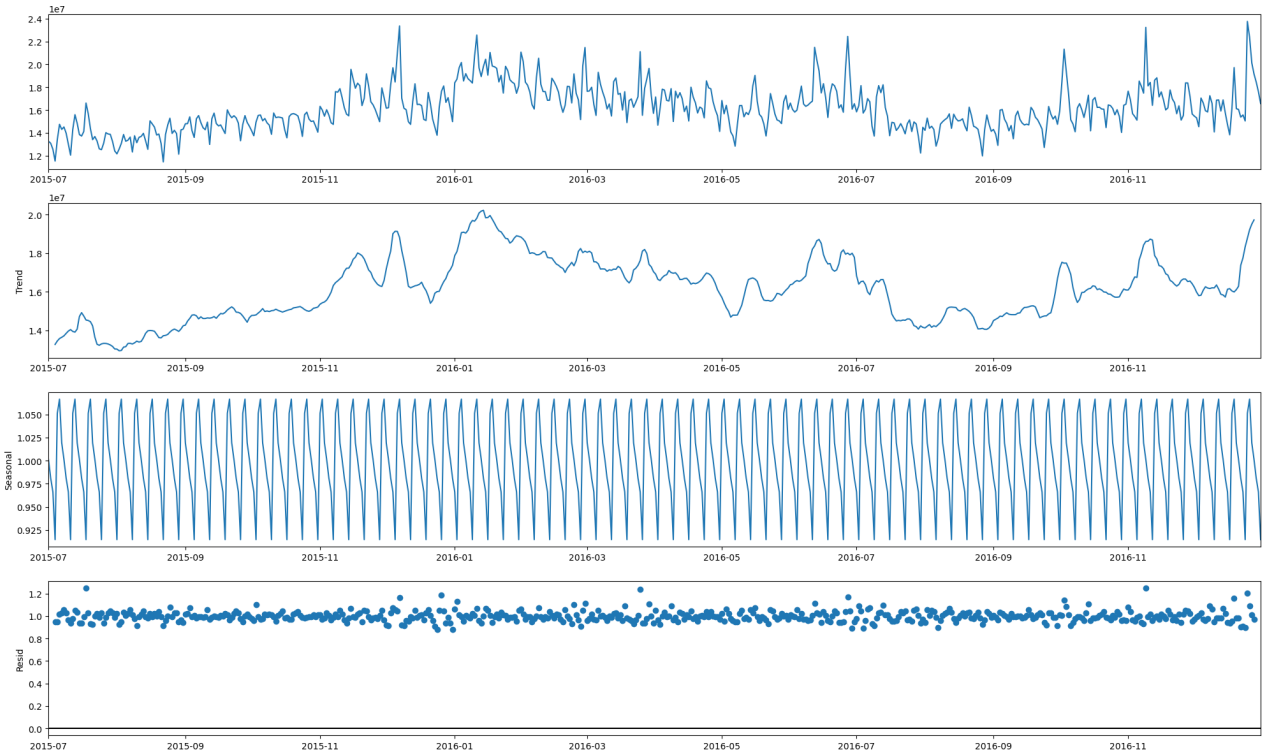
```
-----------------------------------------
```



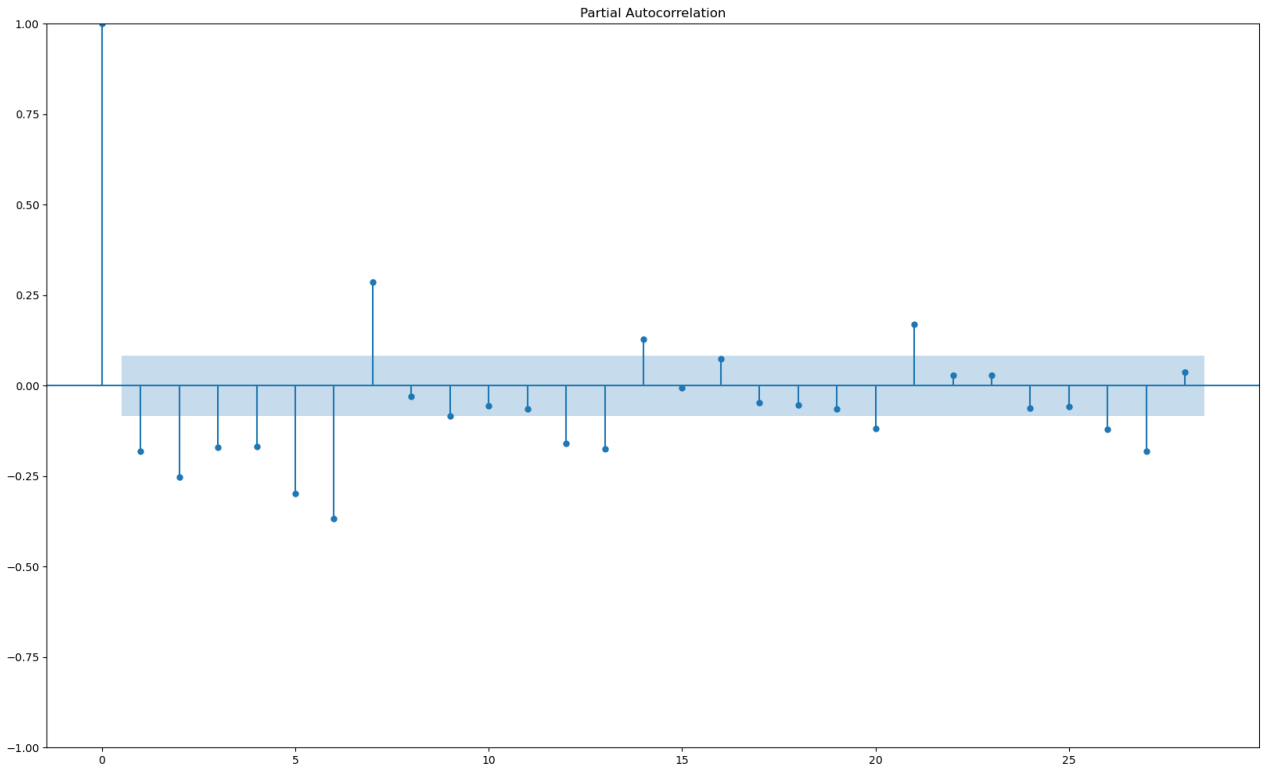

```
-----------------------------------------
-----------------------------------------
Dickey-Fuller Test
Time series is non-stationary
-----------------------------------------
Diff 1 order
-----------------------------------------
```



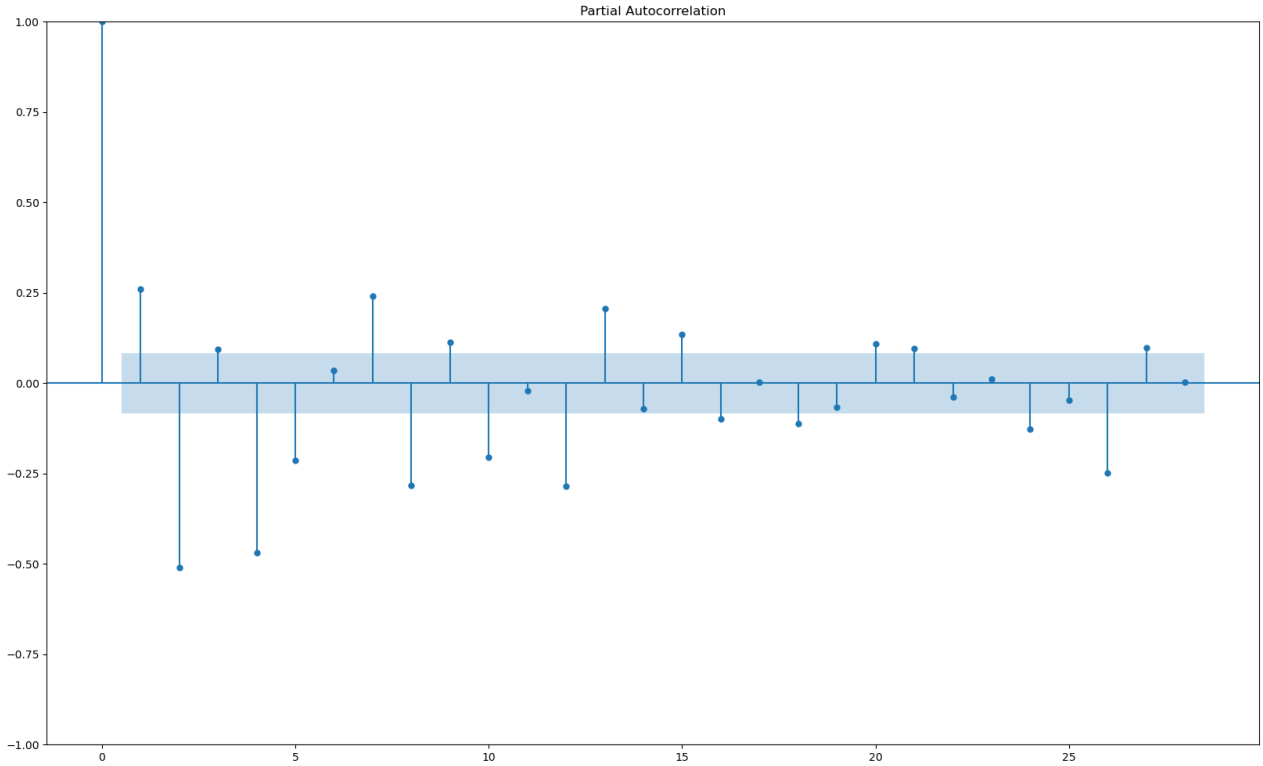
```
------------------------------------------
Dickey-Fuller Test
Time series is stationary
------------------------------------------
------------------------------------------
------------------------------------------
Decomposition (Multiplicaative)
```



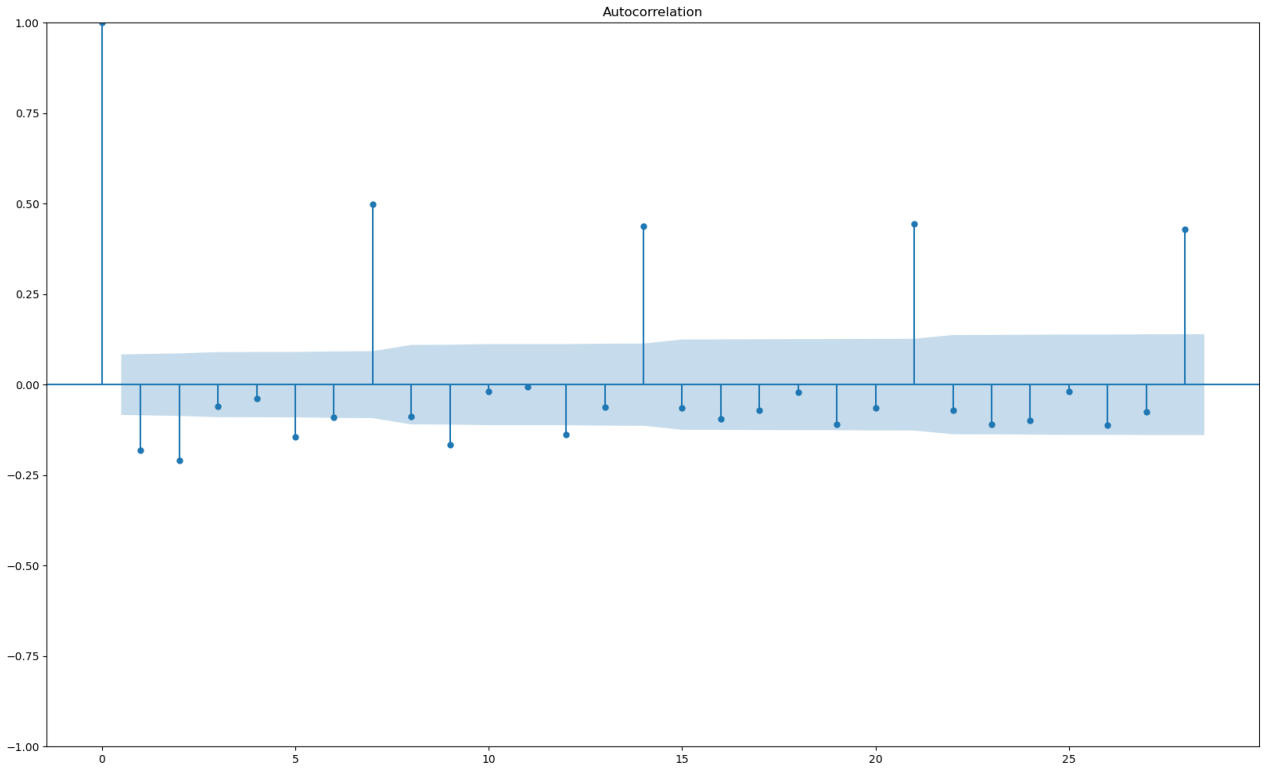

```
------------------------------------------
Dickey-Fuller Test
Time series is stationary
------------------------------------------
------------------------------------------
pacf diff 1 ----------------------------------------
```



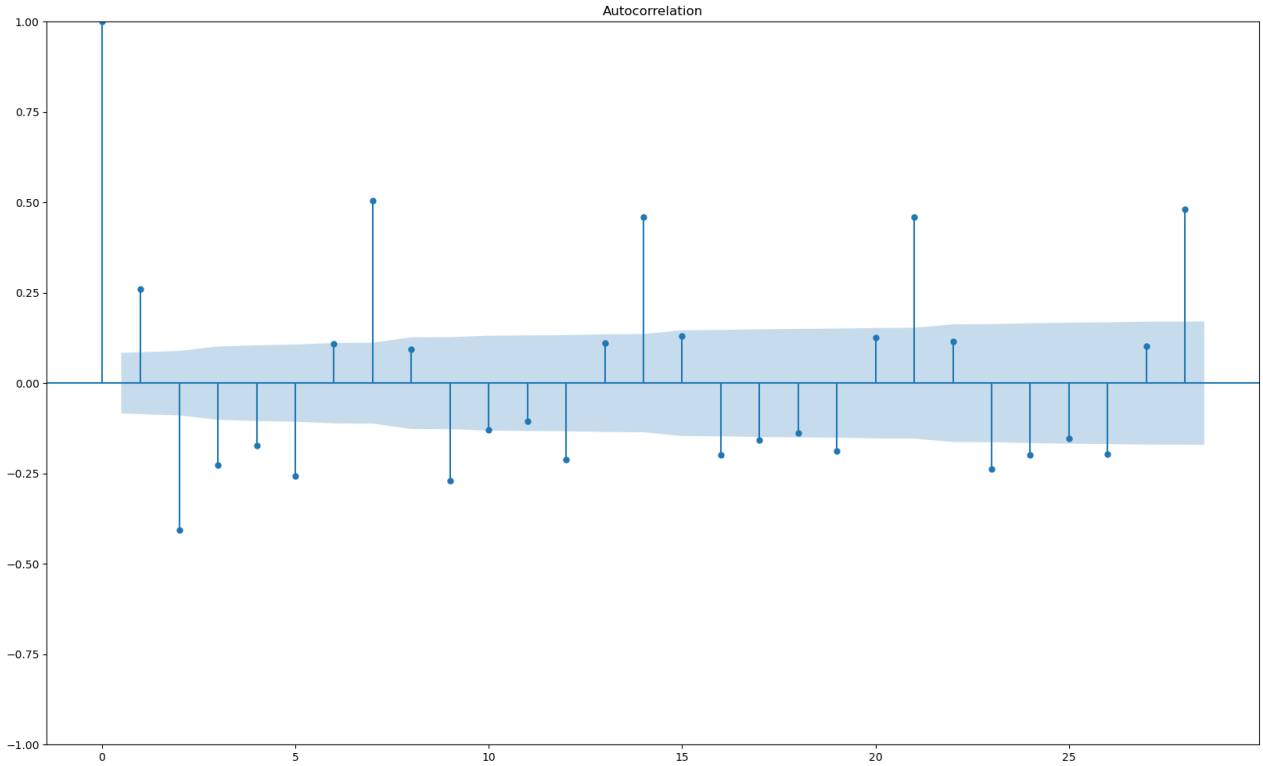

```
------------------------------------------
pacf diff 2 ----------------------------------------
```



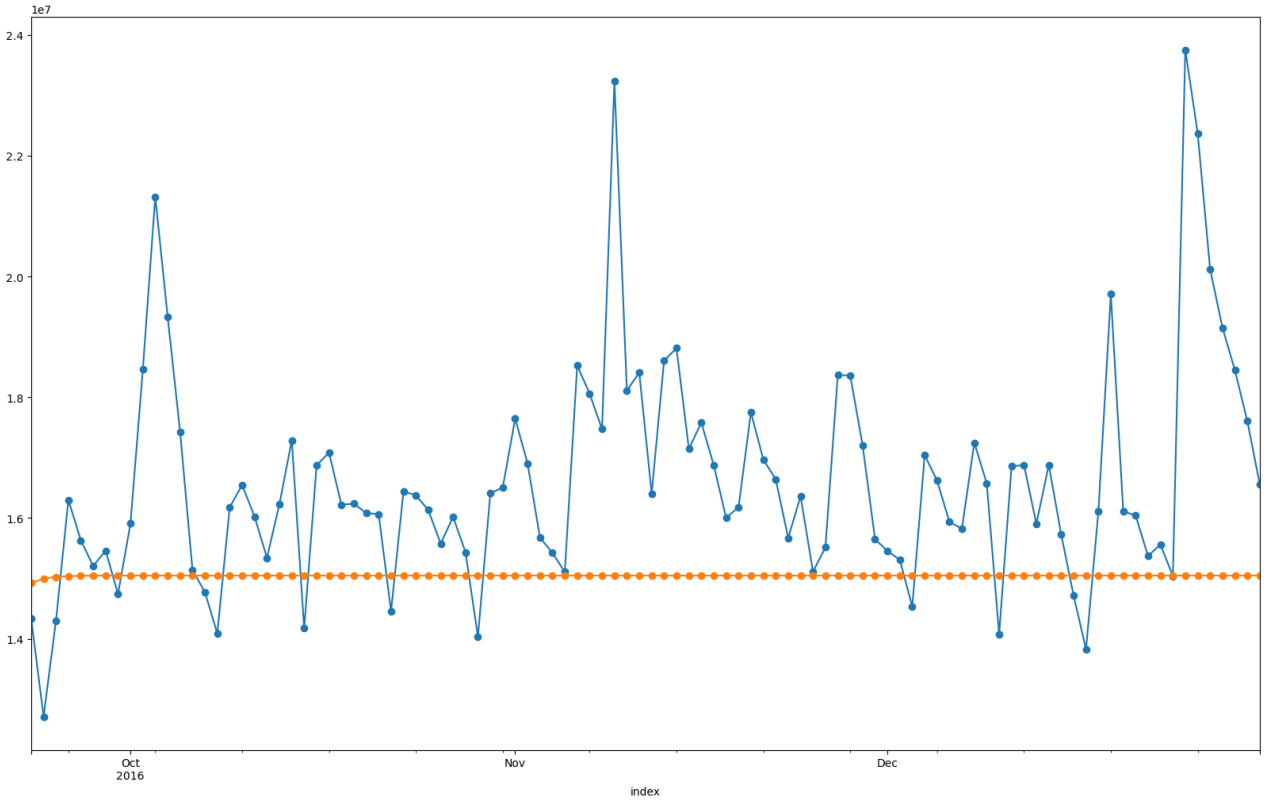

```
----------------------------------------
acf diff 1 ----------------------------------------
```



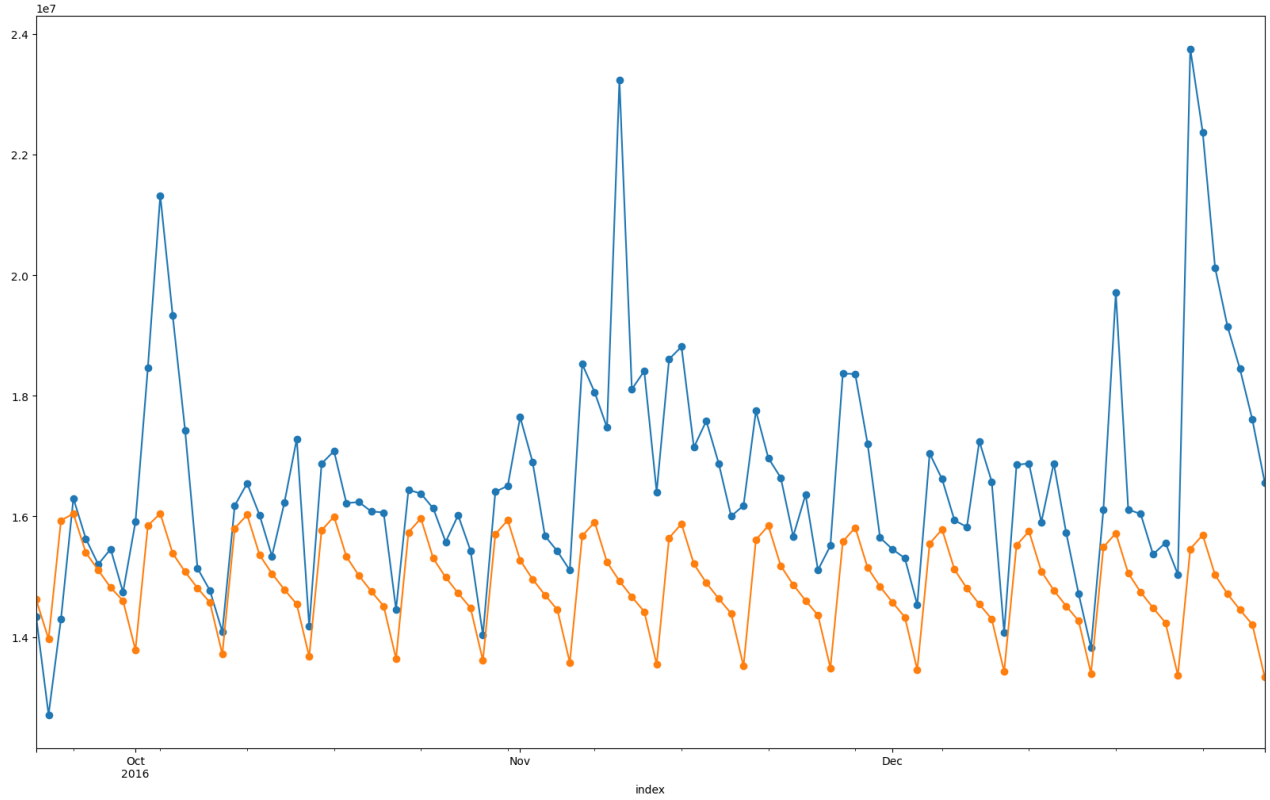

```
----------------------------------------
acf diff 2 ----------------------------------------
```




```
----------------------------------------
ARIMA----------------------------------------
```

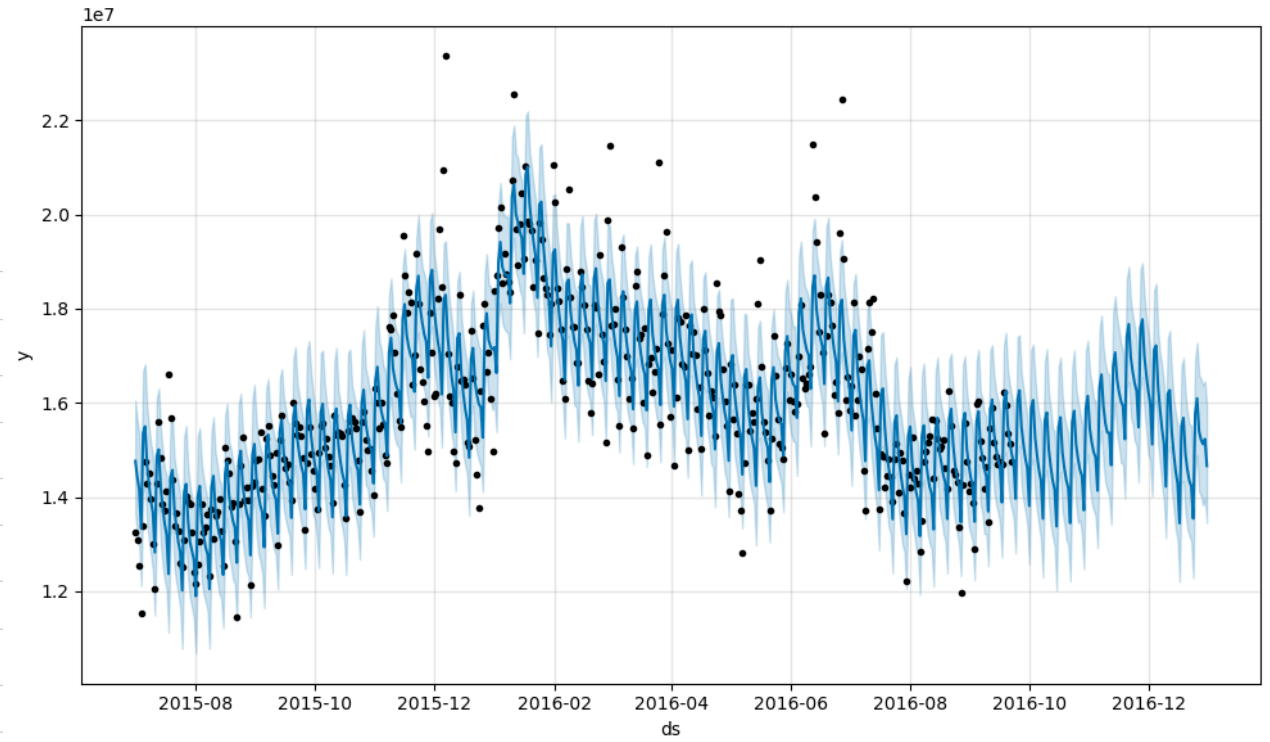



```
MAE : 1769499.402
RMSE : 2412988.766
MAPE: 0.118
----------------------------------------
ARIMA grid search----------------------------------------
(0.069, (8, 2, 8))
----------------------------------------
SARIMAX grid ----------------------------------------
```




```
MAE : 1779324.344
RMSE : 2362988.152
MAPE: 0.119
----------------------------------------
SARIMAX EXO GRID----------------------------------------
------------------------------------------------------------
MAPE and parameters is
0.061
(3, 2, 3, 2, 1, 2, 7)
PROPHET----------------------------------------

08:41:00 - cmdstanpy - INFO - Chain [1] start processing
08:41:00 - cmdstanpy - INFO - Chain [1] done processing
```



In [ ]:

In [ ]:
```
Error with train data is
MAE : 683297.385
RMSE : 949856.37
MAPE: 0.042
```

In [ ]:
```
Error with test data is
MAE : 1425749.412
RMSE : 2048414.596
MAPE: 0.08
----------------------------------------
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: