AdEase

May 12, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv('train_1.csv')
df.head()
```

[2]:

	Page	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	…	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	2016-12-31
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	…	32.0	63.0	15.0	26.0	14.0	20.0	22.0	19.0	18.0	20.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	…	17.0	42.0	28.0	15.0	9.0	30.0	52.0	45.0	26.0	20.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	…	3.0	1.0	1.0	7.0	4.0	4.0	6.0	3.0	4.0	17.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	…	32.0	10.0	26.0	27.0	16.0	11.0	17.0	19.0	10.0	11.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…	NaN	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	10.0

[5 rows x 551 columns]

[3]:
```
exog_data = pd.read_csv("Exog_Campaign_eng")
exog_data.head()
```

[3]:
```
   Exog
0     0
1     0
2     0
3     0
4     0
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 609.8+ MB
```

[5]:
```
exog_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Exog    550 non-null    int64
dtypes: int64(1)
memory usage: 4.4 KB
```

[6]:
```
df.isna().sum()
```

[6]:
```
Page            0
2015-07-01    20740
2015-07-02    20816
2015-07-03    20544
2015-07-04    20654
              ...
2016-12-27     3701
2016-12-28     3822
2016-12-29     3826
2016-12-30     3635
2016-12-31     3465
Length: 551, dtype: int64
```

We can see that with larger value of date, the number of NA values are decreasing hence our first guess to why there would be so many NA values is that those webpages must NOT be present on wikipedia on those dates. That is, the origin of that webpage would be later to that date.

[7]:
```
# SPECIFIC NAME _ LANGUAGE.wikipedia.org _ ACCESS TYPE _ ACCESS ORIGIN
def parse_page_string(webpage):
    webpage_reversed = webpage[::-1]
    first_split = webpage_reversed.split('_', maxsplit=1)
    reversed_access_origin = first_split[0]
    second_split = first_split[1].split('_', maxsplit=1)
    reversed_access_type = second_split[0]
    name_language_org = second_split[1][::-1]

    page_org = name_language_org.split('.')[-2] + '.' + name_language_org.
 ↪split('.')[-1]
    name_language = ''
    for elem in name_language_org.split('.')[:-2]:
        name_language = name_language + '.' + elem
    language_name = name_language[-1:0:-1]
    reverse_page_language, reverse_page_name = language_name.split('_',␣
 ↪maxsplit=1)
    return [reverse_page_name[::-1], reverse_page_language[::-1], page_org,␣
 ↪reversed_access_type[::-1], reversed_access_origin[::-1]]

print(parse_page_string('4minute_zh.wikipedia.org_all-access_spider'))
```

```
['4minute', 'zh', 'wikipedia.org', 'all-access', 'spider']
```

[8]:
```
exog_data['Exog']
```

[8]:
```
0      0
1      0
2      0
3      0
4      0
      ..
545    1
546    1
547    1
548    0
549    0
Name: Exog, Length: 550, dtype: int64
```

[9]:
```
parsed_pagename_list = []
for page in df['Page']:
    parsed_pagename_list.append(parse_page_string(page))
parsed_pagename_list = np.asarray(parsed_pagename_list)

df['page_name'] = parsed_pagename_list[:, 0]
df['page_language'] = parsed_pagename_list[:, 1]
df['page_org'] = parsed_pagename_list[:, 2]
```

```
df['page_access_type'] = parsed_pagename_list[:, 3]
df['page_access_origin'] = parsed_pagename_list[:, 4]
```

[10]:
```
cols = df.columns[1:551]

def compute_page_origin_date(x):
    origin_date = '2017-01-01'
    for col in cols:
        if not(pd.isna(x[col])):
            origin_date = col
            break
    return origin_date

origin_dates = []
for index, row in df.iterrows():
    page_origin_date = compute_page_origin_date(row)
    origin_dates.append(page_origin_date)

df['page_origin_date'] = origin_dates
```

[11]:
```
df.head()
```

[11]:

	Page	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	…	2016-12-28	2016-12-29	2016-12-30	2016-12-31	page_name	page_language	page_org	page_access_type
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	…	22.0	19.0	18.0	20.0	2NE1	zh	wikipedia.org	all-access
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	…	52.0	45.0	26.0	20.0	2PM	zh	wikipedia.org	all-access
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	…	6.0	3.0	4.0	17.0	3C	zh	wikipedia.org	all-access
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	…	17.0	19.0	10.0	11.0				
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	27.0	13.0	36.0	10.0				

```
3          4minute             zh  wikipedia.org        all-access
4  52_Hz_I_Love_You            zh  wikipedia.org        all-access

   page_access_origin  page_origin_date
0              spider        2015-07-01
1              spider        2015-07-01
2              spider        2015-07-01
3              spider        2015-07-01
4              spider        2016-04-17

[5 rows x 557 columns]
```

[12]:
```
data_melt = pd.melt(df, id_vars= ['Page' ,'page_name', 'page_language',␣
 ↪'page_org', 'page_access_type',
      'page_access_origin', 'page_origin_date'], var_name='Date',␣
 ↪value_name='Views')
data_melt
```

[12]:
```
                                                     Page  \
0                   2NE1_zh.wikipedia.org_all-access_spider
1                    2PM_zh.wikipedia.org_all-access_spider
2                     3C_zh.wikipedia.org_all-access_spider
3                 4minute_zh.wikipedia.org_all-access_spider
4           52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…
…                                                         …
79784645  Underworld_(serie_de_películas)_es.wikipedia.o…
79784646  Resident_Evil:_Capítulo_Final_es.wikipedia.org…
79784647  Enamorándome_de_Ramón_es.wikipedia.org_all-acc…
79784648  Hasta_el_último_hombre_es.wikipedia.org_all-ac…
79784649  Francisco_el_matemático_(serie_de_televisión_d…

                                          page_name page_language  \
0                                              2NE1            zh
1                                               2PM            zh
2                                                3C            zh
3                                           4minute            zh
4                                     52_Hz_I_Love_You            zh
…                                                 …             …
79784645              Underworld_(serie_de_películas)            es
79784646                Resident_Evil:_Capítulo_Final            es
79784647                       Enamorándome_de_Ramón            es
79784648                     Hasta_el_último_hombre             es
79784649  Francisco_el_matemático_(serie_de_televisión_d…           es

              page_org page_access_type page_access_origin page_origin_date  \
0         wikipedia.org       all-access             spider       2015-07-01
1         wikipedia.org       all-access             spider       2015-07-01
```

```
2         wikipedia.org       all-access              spider       2015-07-01
3         wikipedia.org       all-access              spider       2015-07-01
4         wikipedia.org       all-access              spider       2016-04-17
…                   …                  …                   …                  …
79784645  wikipedia.org       all-access              spider       2016-12-26
79784646  wikipedia.org       all-access              spider       2017-01-01
79784647  wikipedia.org       all-access              spider       2017-01-01
79784648  wikipedia.org       all-access              spider       2017-01-01
79784649  wikipedia.org       all-access              spider       2017-01-01

                Date  Views
0         2015-07-01   18.0
1         2015-07-01   11.0
2         2015-07-01    1.0
3         2015-07-01   35.0
4         2015-07-01    NaN
…                …    …
79784645  2016-12-31   10.0
79784646  2016-12-31    NaN
79784647  2016-12-31    NaN
79784648  2016-12-31    NaN
79784649  2016-12-31    NaN

[79784650 rows x 9 columns]
```

[13]:
```
data_melt['Date'] = pd.to_datetime(data_melt['Date'])
exog_data['Date'] = df.columns[1:551]
exog_data['Date'] = pd.to_datetime(exog_data['Date'])
exog_data.set_index('Date', inplace=True)
```

[14]:
```
temp_data = data_melt[data_melt['Page'] == '52_Hz_I_Love_You_zh.wikipedia.
 ↪org_all-access_spider']
temp_data[temp_data['Date'] >= temp_data['page_origin_date']]
```

[14]:
```
                                                      Page          page_name  \
42213337  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
42358400  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
42503463  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
42648526  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
42793589  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
…                                                      …                …
79059339  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
79204402  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
79349465  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
79494528  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
79639591  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s…  52_Hz_I_Love_You
```

```
         page_language       page_org page_access_type page_access_origin  \
42213337            zh  wikipedia.org       all-access             spider
42358400            zh  wikipedia.org       all-access             spider
42503463            zh  wikipedia.org       all-access             spider
42648526            zh  wikipedia.org       all-access             spider
42793589            zh  wikipedia.org       all-access             spider
…                  …            …                …                  …
79059339            zh  wikipedia.org       all-access             spider
79204402            zh  wikipedia.org       all-access             spider
79349465            zh  wikipedia.org       all-access             spider
79494528            zh  wikipedia.org       all-access             spider
79639591            zh  wikipedia.org       all-access             spider

         page_origin_date       Date  Views
42213337       2016-04-17 2016-04-17   38.0
42358400       2016-04-17 2016-04-18  159.0
42503463       2016-04-17 2016-04-19    9.0
42648526       2016-04-17 2016-04-20    4.0
42793589       2016-04-17 2016-04-21    1.0
…                   …        …    …
79059339       2016-04-17 2016-12-27   11.0
79204402       2016-04-17 2016-12-28   27.0
79349465       2016-04-17 2016-12-29   13.0
79494528       2016-04-17 2016-12-30   36.0
79639591       2016-04-17 2016-12-31   10.0

[259 rows x 9 columns]
```

[15]:
```
# Filter the data so that all info before page origin date is removed
data_melt['page_origin_date'] = pd.to_datetime(data_melt['page_origin_date'])
filtered_data = data_melt[data_melt['Date'] >= data_melt['page_origin_date']]
filtered_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 75062881 entries, 0 to 79784645
Data columns (total 9 columns):
 #   Column              Dtype
---  ------              -----
 0   Page                object
 1   page_name           object
 2   page_language       object
 3   page_org            object
 4   page_access_type    object
 5   page_access_origin  object
 6   page_origin_date    datetime64[ns]
 7   Date                datetime64[ns]
 8   Views               float64
dtypes: datetime64[ns](2), float64(1), object(6)
```

```
memory usage: 5.6+ GB
```

[16]:
```
filtered_data.isna().sum()
```

[16]:
```
Page                        0
page_name                   0
page_language               0
page_org                    0
page_access_type            0
page_access_origin          0
page_origin_date            0
Date                        0
Views                 1471162
dtype: int64
```

[17]:
```
data_melt.isna().sum()
```

[17]:
```
Page                        0
page_name                   0
page_language               0
page_org                    0
page_access_type            0
page_access_origin          0
page_origin_date            0
Date                        0
Views                 6192931
dtype: int64
```

Hence, we have removed 4721769 nan values without loss of any information

[18]:
```
# using the method below, we can create a new dataframe in which missing values␣
 ↪are interpolated using inbuilt methods for each page after grouping them by␣
 ↪page and then applying interpolation and the resetting the index
# We wont apply this for us as this is a very cpu intensive task and it is␣
 ↪causing the python kernel to break

# filtered_data_groups_with_interpolated_info = filtered_data.groupby('Page').
 ↪apply(lambda group: group.Views.interpolate(method='linear'))
# updated_filtered_data = filtered_data_groups_with_interpolated_info.
 ↪reset_index()
```

[19]:
```
# Lets apply our methods to the time series of a particular page. We apply the␣
 ↪method so that we can get results for any page that we want
total_views_data = filtered_data.groupby("Date").agg(Views = pd.
 ↪NamedAgg(column="Views", aggfunc="sum"))
sample_data_for_prediction = total_views_data
sample_data_for_prediction.head()
```

[19]:
```
                  Views
Date
2015-07-01  148672476.0
2015-07-02  149593840.0
2015-07-03  141164198.0
2015-07-04  145612937.0
2015-07-05  151495372.0
```

[20]:
```
## linear interpolation code
sample_data_for_prediction.isna().sum()
```

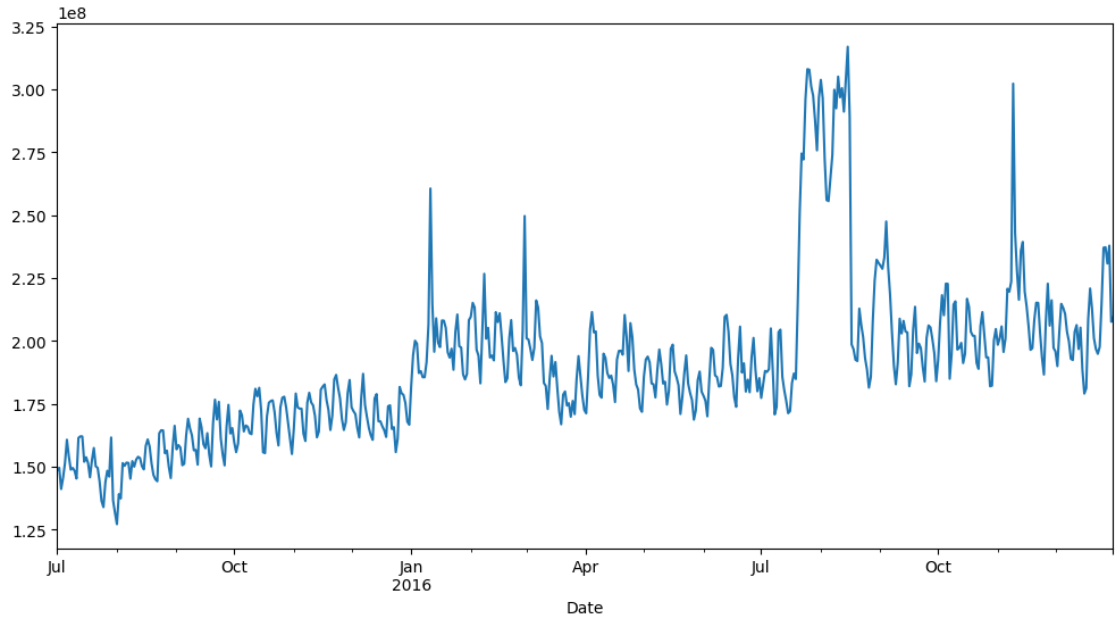
[20]:
```
Views    0
dtype: int64
```

[21]:
```
sample_data_for_prediction.Views = sample_data_for_prediction.Views.
 ↪interpolate(method='linear')
sample_data_for_prediction.isna().sum()
```

[21]:
```
Views    0
dtype: int64
```

[22]:
```
# sample_data_for_prediction.set_index('Date', inplace=True)
sample_data_for_prediction.Views.plot(figsize=(12,6))
```

[22]:
```
<Axes: xlabel='Date'>
```



[23]:
```
sample_data_for_prediction.head()
```

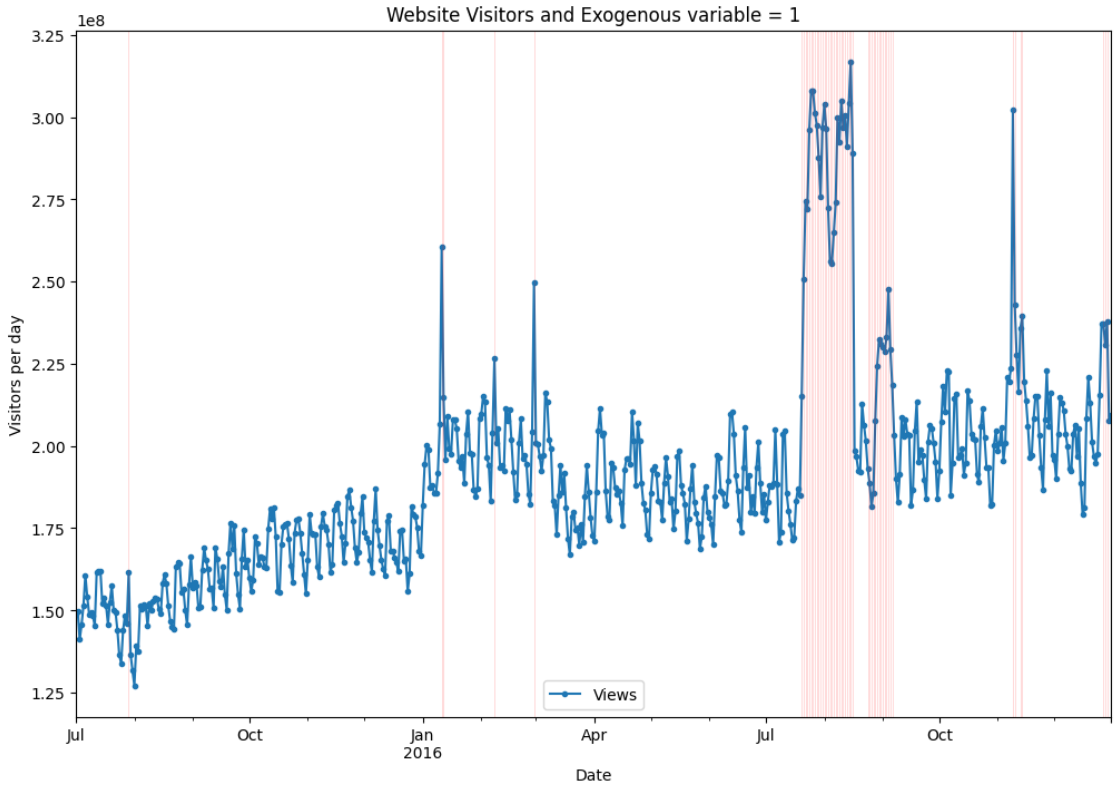
[23]:
```
                  Views
Date
2015-07-01  148672476.0
2015-07-02  149593840.0
2015-07-03  141164198.0
2015-07-04  145612937.0
2015-07-05  151495372.0
```

[24]:
```
exog_value_array = []
for index, row in sample_data_for_prediction.iterrows():
    exog_value_array.append(exog_data.loc[index]['Exog'])

sample_data_for_prediction['exogenous_variable'] = exog_value_array
```

[25]:
```
sample_data_for_prediction.head()
```

[25]:
```
                  Views  exogenous_variable
Date
2015-07-01  148672476.0                   0
2015-07-02  149593840.0                   0
2015-07-03  141164198.0                   0
2015-07-04  145612937.0                   0
2015-07-05  151495372.0                   0
```

[26]:
```
title='Website Visitors and Exogenous variable = 1'
ylabel='Visitors per day'
xlabel='Date'
ax = sample_data_for_prediction['Views'].
  ↪plot(legend=True,figsize=(12,8),title=title, style = ".-")
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in sample_data_for_prediction.query('exogenous_variable==1').index:
    ax.axvline(x=x, color='red', alpha = 0.2, linewidth = 0.5)
```



0.0.1 Dickey Fuller Test -

[27]:
```
# AD Fuller test -
import statsmodels.api as sm
def adf_test(dataset):
   pvalue = sm.tsa.stattools.adfuller(dataset)[1]
   print(pvalue)
   if pvalue <= 0.05:
      print('Sequence is stationary')
   else:
      print('Sequence is not stationary')

adf_test(sample_data_for_prediction['Views'])
```

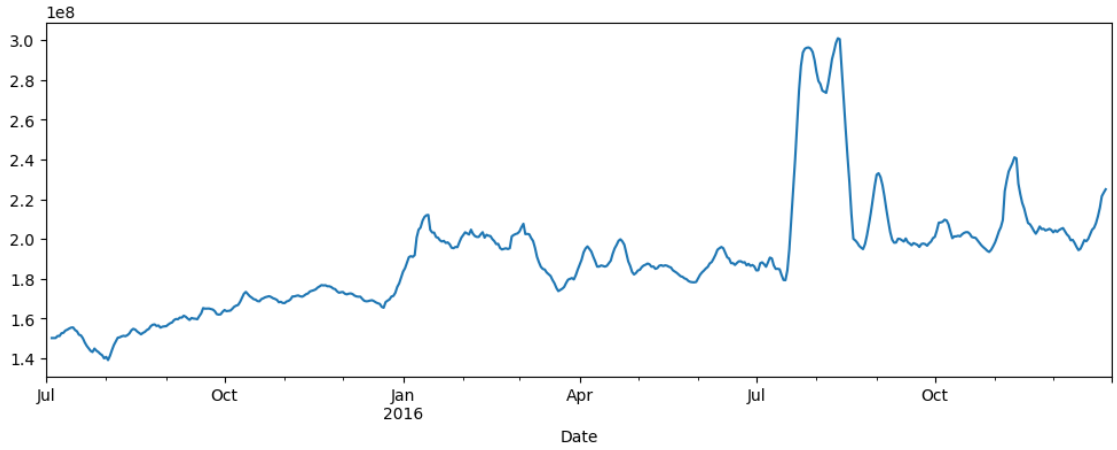
```
0.1316628509099551
Sequence is not stationary
```

1 Decomposition of series

[28]:
```
model = sm.tsa.seasonal_decompose(sample_data_for_prediction.Views,␣
↪model='additive')
```

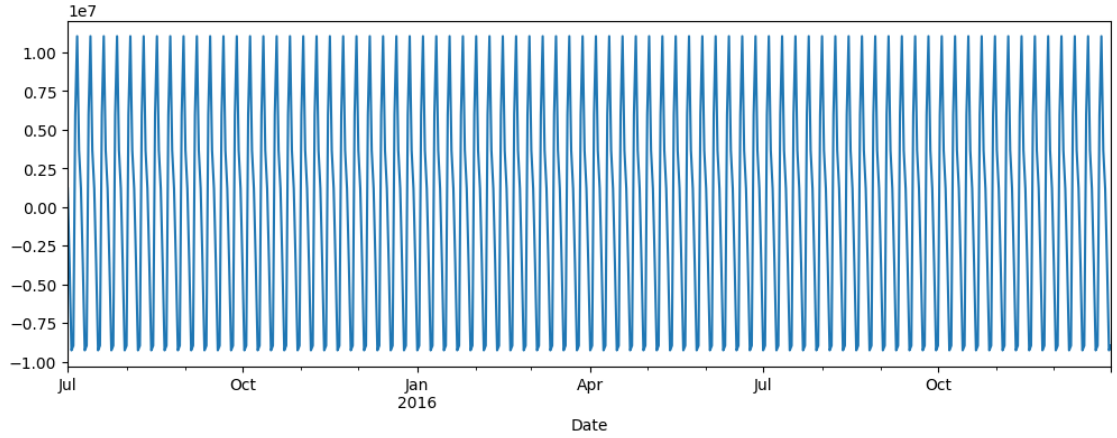
[29]:
```
model.trend.plot(figsize=(12,4))
```

[29]: <Axes: xlabel='Date'>



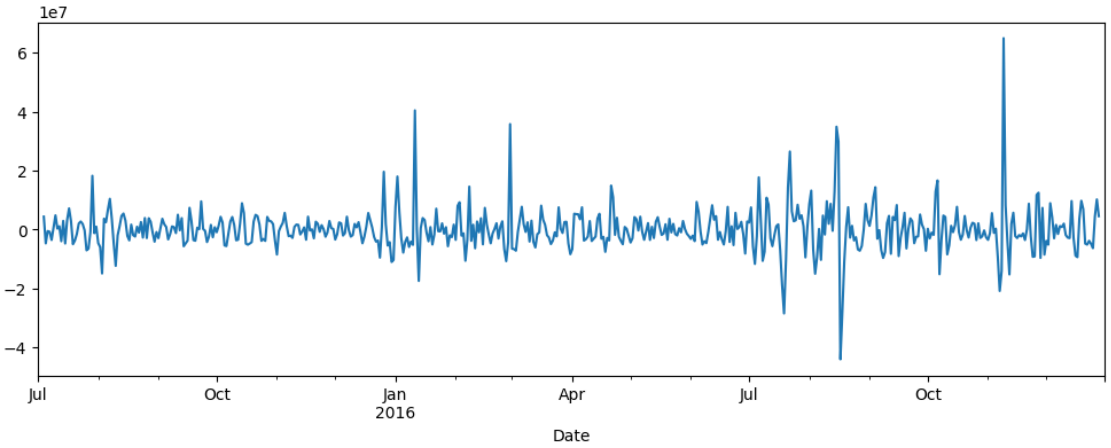
[30]:
```
model.seasonal.plot(figsize=(12,4))
```

[30]: <Axes: xlabel='Date'>



[31]:
```
model.resid.plot(figsize=(12,4))
```

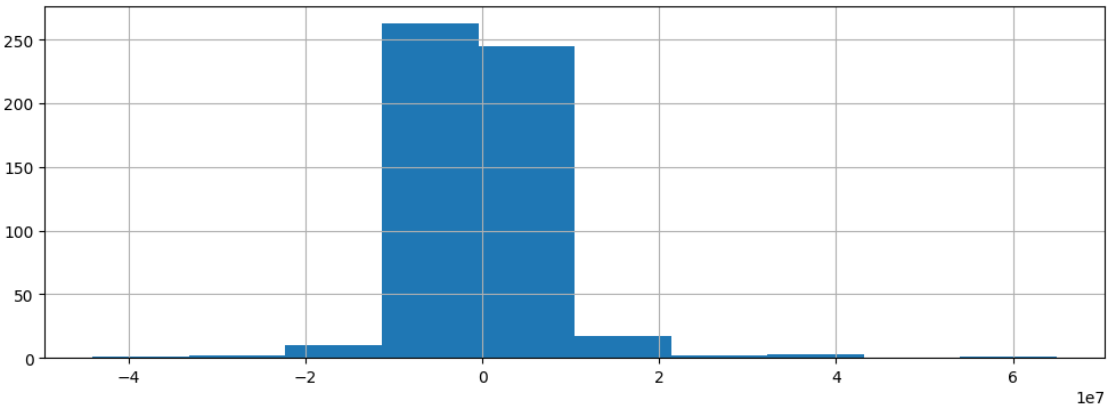
[31]: <Axes: xlabel='Date'>



[32]:
```
print(model.resid.mean())
model.resid.hist(figsize=(12,4))
```

```
1152.1685117724187
```

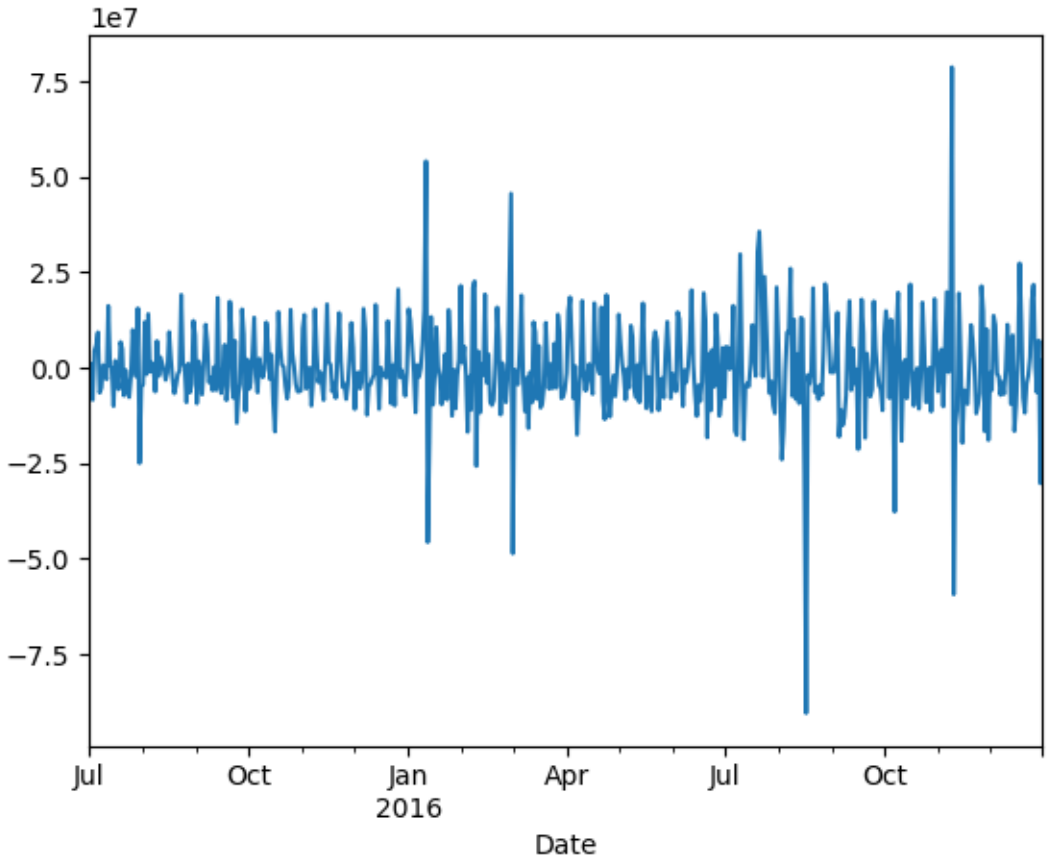
[32]: <Axes: >



2 Differencing of Series

[33]:
```
detrend = sample_data_for_prediction.Views.diff(1)
detrend.plot()
```

[33]: <Axes: xlabel='Date'>

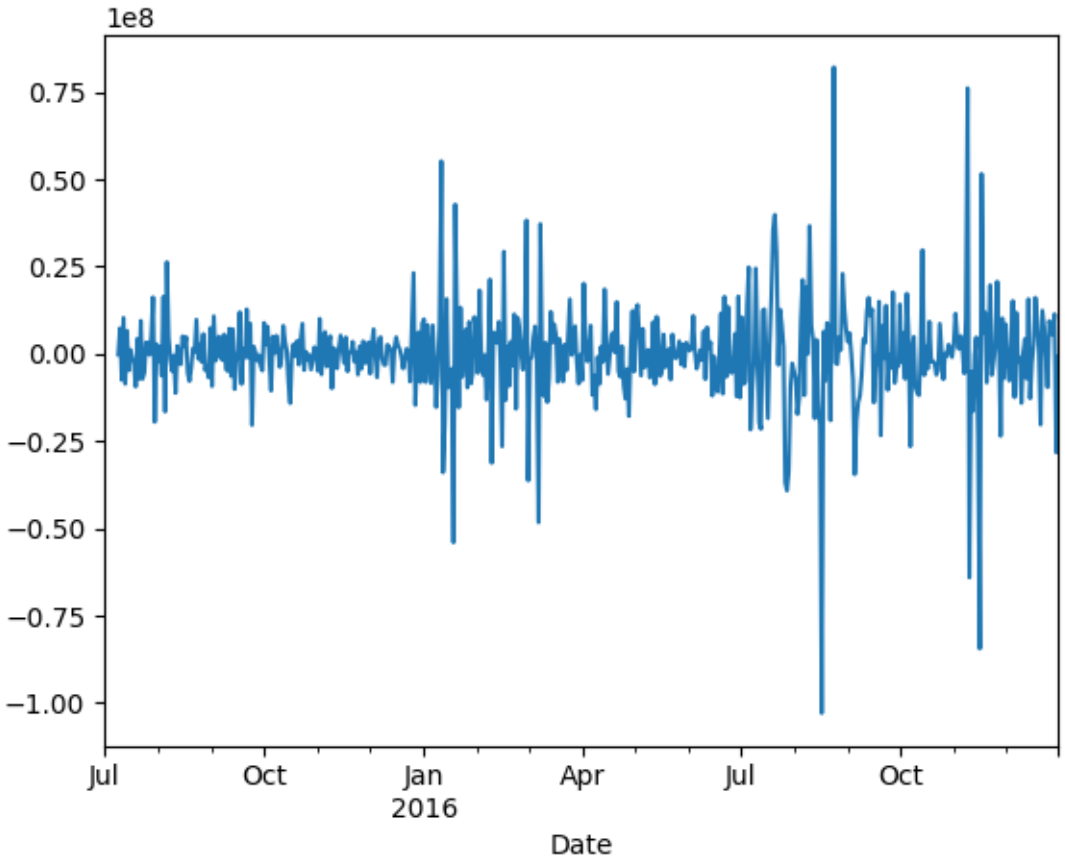


[34]:
```
adf_test(detrend.dropna())
```

```
4.367449082413303e-12
Sequence is stationary
```

[35]:
```
### Let's remove both trend and seasonality
stationary = sample_data_for_prediction.Views.diff(1).diff(7)
stationary.plot()
```

[35]:
```
<Axes: xlabel='Date'>
```



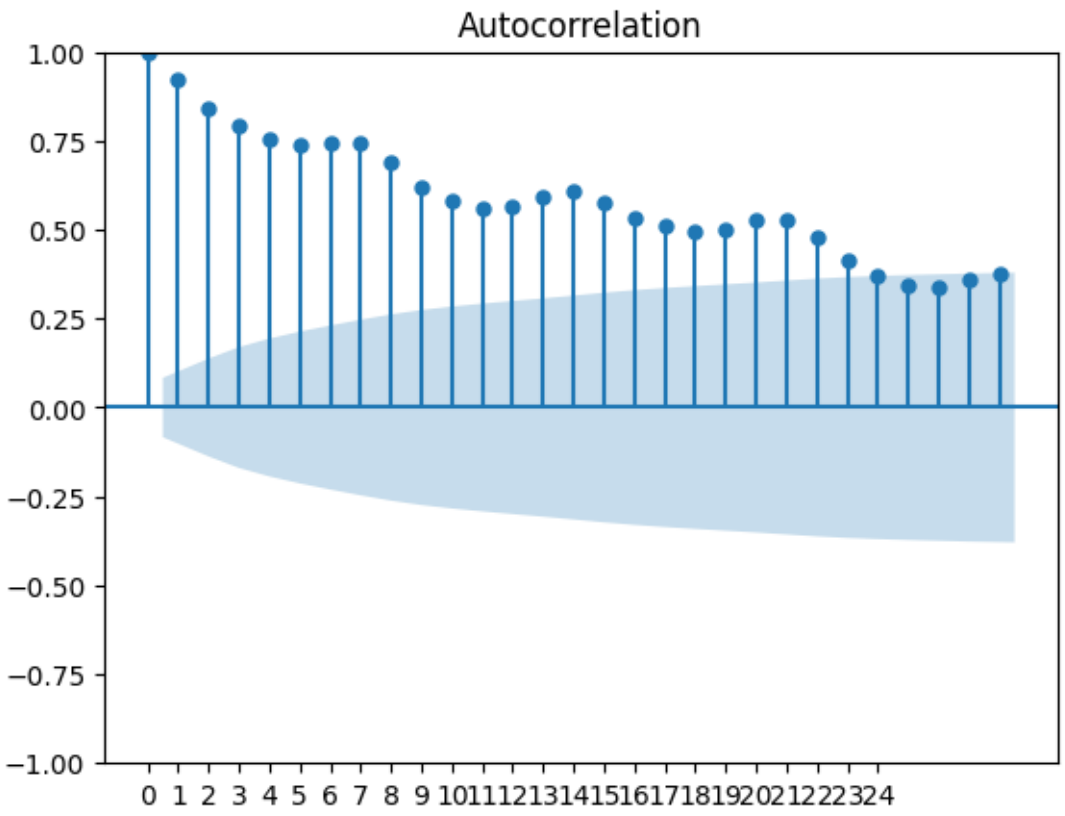
[36]:
```
adf_test(stationary.dropna())
```

```
4.710334879118502e-08
Sequence is stationary
```

3 ACF and PACF

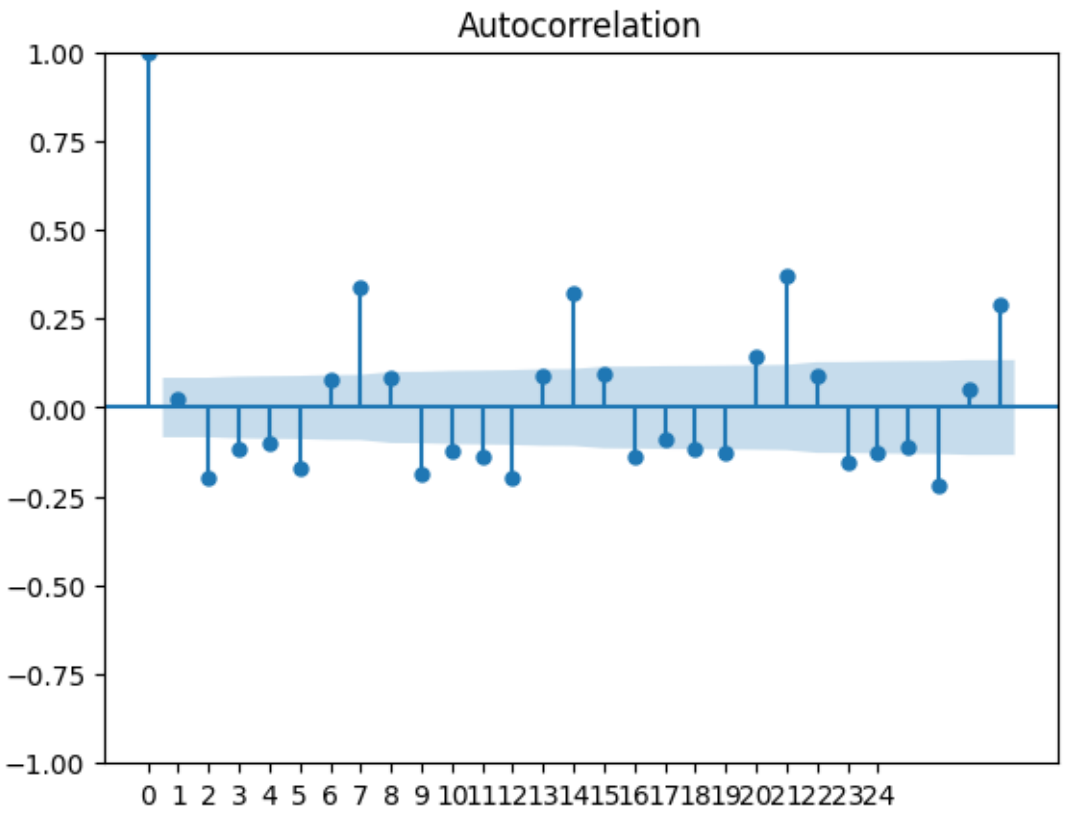
3.0.1 ACF

[37]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(sample_data_for_prediction.Views)
plt.xticks(range(25))
plt.show()
```



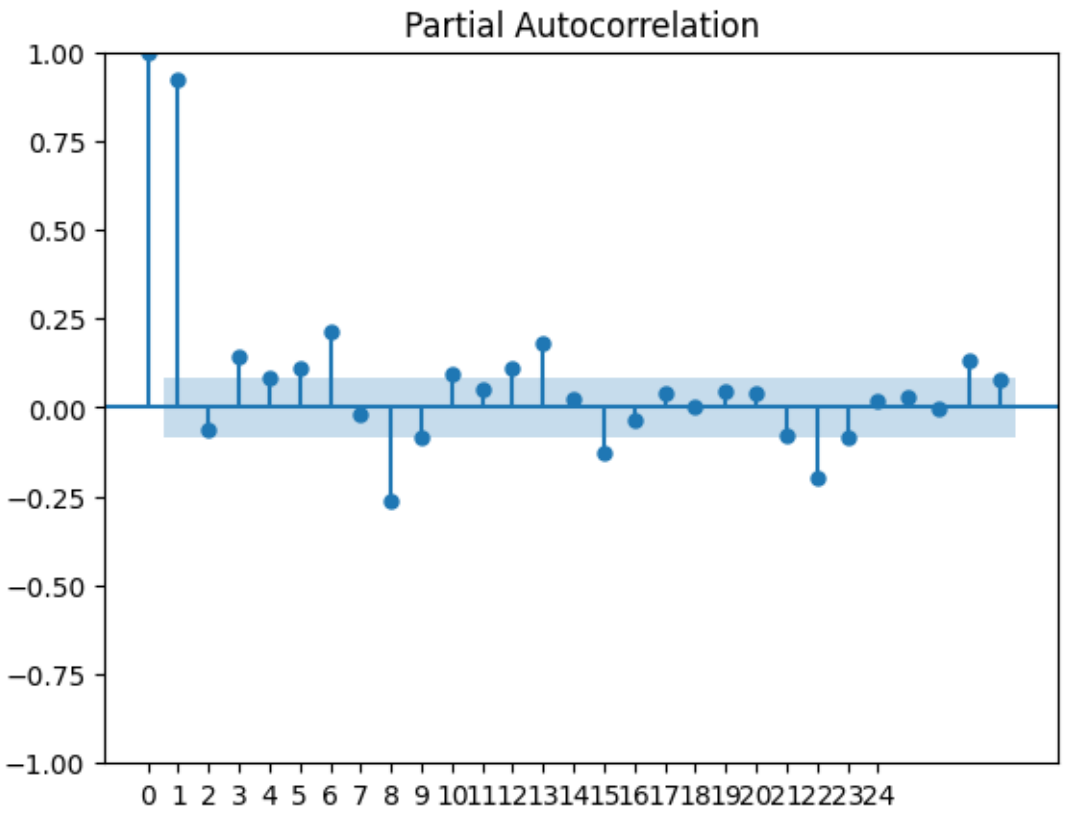
Let's check acf for detrended series

[38]:
```
### Only Seasonality and residuals: stationary
plot_acf(sample_data_for_prediction.Views.diff().dropna())
plt.xticks(range(25))
plt.show()
```

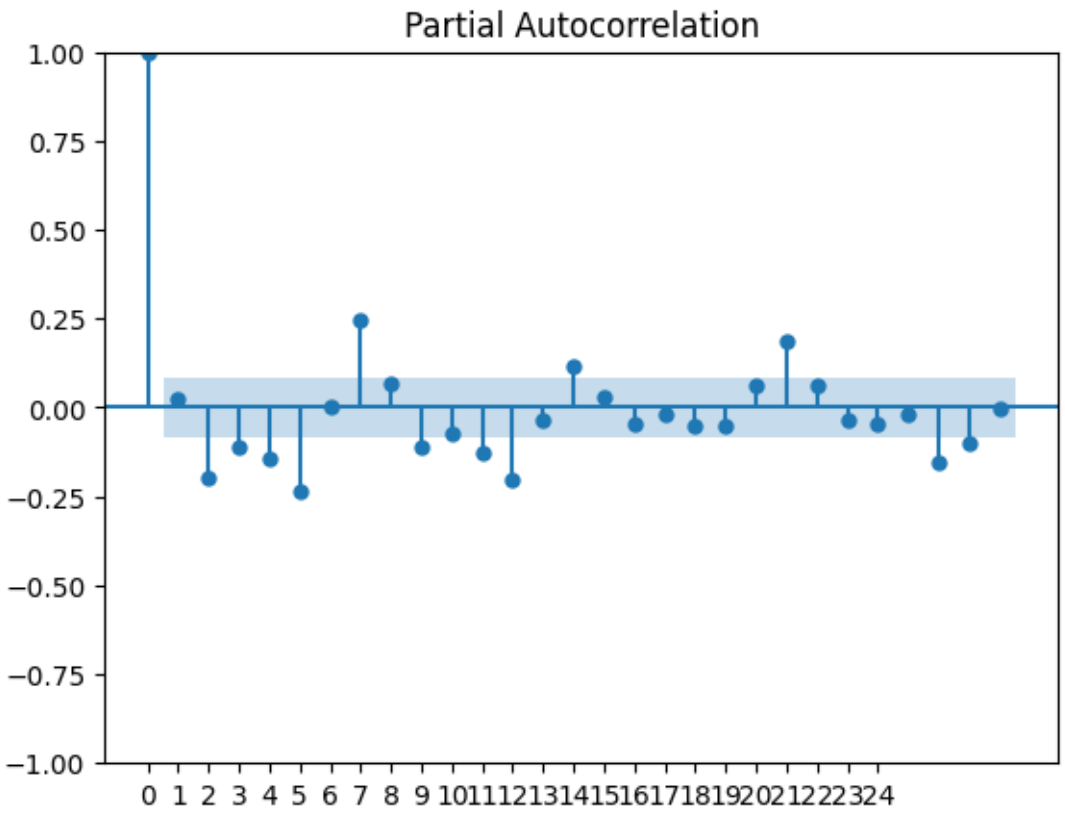


3.0.2 PACF

[39]:
```
plot_pacf(sample_data_for_prediction.Views)
plt.xticks(range(25))
plt.show()
```



[40]:
```
### Only Seasonality and residuals
plot_pacf(sample_data_for_prediction.Views.diff().dropna())
plt.xticks(range(25))
plt.show()
```



3.1 Test train split

[41]:
```
train_max_date = sample_data_for_prediction.index[-13]
train_max_date
```

[41]:
```
Timestamp('2016-12-19 00:00:00')
```

[42]:
```
train_x = sample_data_for_prediction.iloc[:-13].copy()
test_x = sample_data_for_prediction.iloc[-13:-1].copy()


test_x
```

[42]:

Date	Views	exogenous_variable
2016-12-19	220854276.0	0
2016-12-20	212933744.0	0
2016-12-21	201119631.0	0
2016-12-22	196860226.0	0
2016-12-23	194884736.0	0
2016-12-24	197644557.0	0

```
2016-12-25  215333402.0                  0
2016-12-26  237068067.0                  0
2016-12-27  237248109.0                  1
2016-12-28  230782936.0                  1
2016-12-29  237886569.0                  1
2016-12-30  207608296.0                  0
```

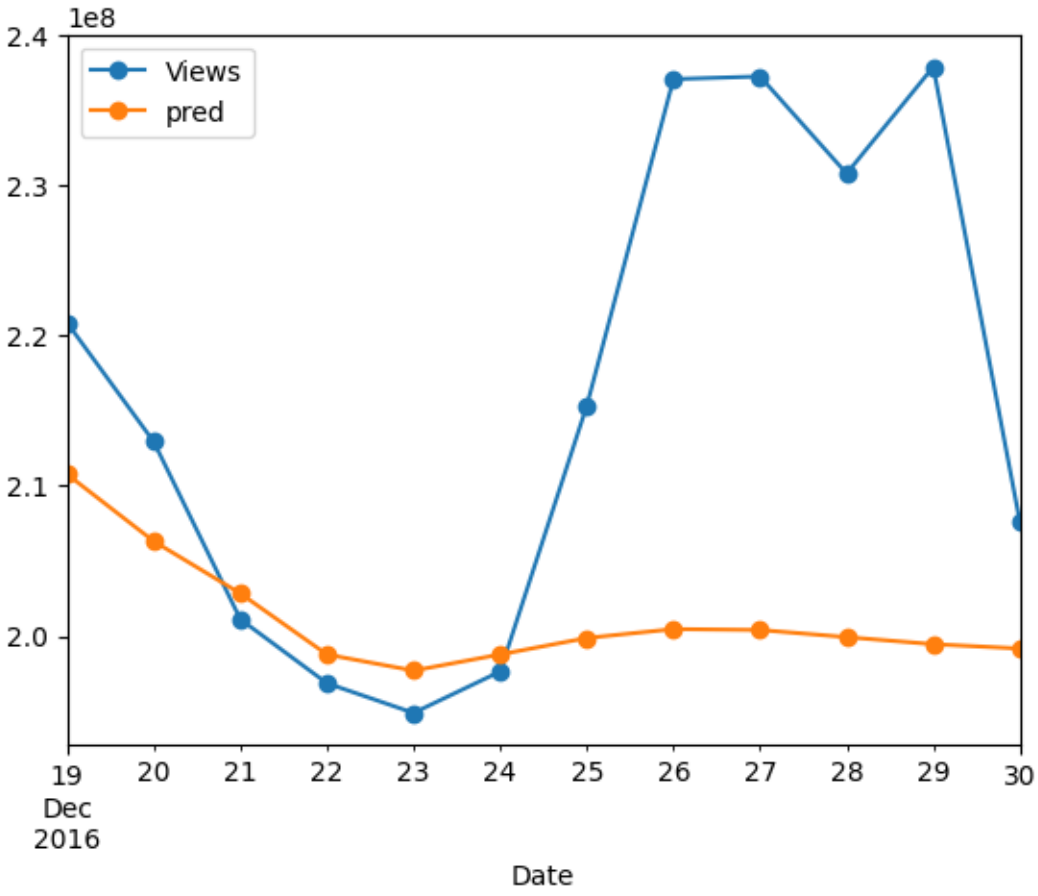
[43]:
```
from sklearn.metrics import (
    mean_squared_error as mse,
    mean_absolute_error as mae,
    mean_absolute_percentage_error as mape
)

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))
```

3.2 ARIMA

[44]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
p = 5
q = 1
d = 0

model = SARIMAX(train_x.Views, order=(p, d, q))
model = model.fit(disp=False)
test_x['pred'] = model.forecast(steps=48)
test_x[['Views','pred']].plot(style='-o')
performance(test_x['Views'], test_x['pred'])
```

```
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
MAE : 15909042.236
RMSE : 21596849.447
MAPE: 0.069
```

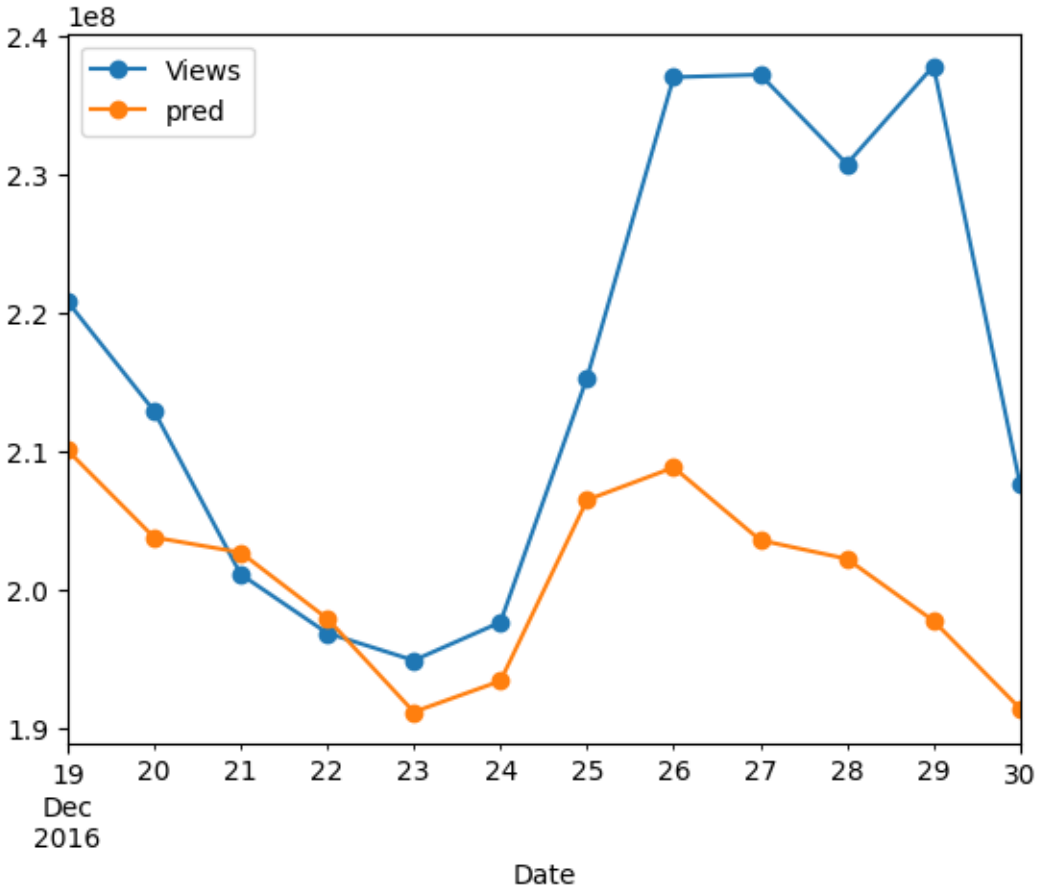


3.3 SARIMA

[45]:
```
# Standard Parameters
p = 5
d = 1
q = 0

## Seasonal Parameters
P = 3
D = 0
Q = 1
S = 7

model = SARIMAX(train_x.Views, order=(p, d, q), seasonal_order=(P,D,Q,S))
model = model.fit()
test_x['pred'] = model.forecast(steps=24)
test_x[['Views','pred']].plot(style='-o')
performance(test_x['Views'], test_x['pred'])
```

```
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           10     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.76626D+01    |proj g|=  2.54802D-01

At iterate    5    f=  1.75756D+01    |proj g|=  2.19224D-02

At iterate   10    f=  1.75577D+01    |proj g|=  2.03925D-02

At iterate   15    f=  1.75524D+01    |proj g|=  4.20298D-03

At iterate   20    f=  1.75522D+01    |proj g|=  1.77254D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
   10     22     25      1     0     0   1.059D-04   1.755D+01
  F =   17.552196290748146

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
MAE : 15513167.39
RMSE : 20275212.111
```

```
MAPE: 0.068
```



[46]:
```
# Standard Parameters
p = 5
d = 1
q = 0

## Seasonal Parameters
P = 3
D = 0
Q = 1
S = 7

start=len(train_x)
end=len(train_x)+len(test_x)-1

model = SARIMAX(train_x.Views, exog=train_x['exogenous_variable'], order=(p, d,␣
↪q), seasonal_order=(P,D,Q,S))
```

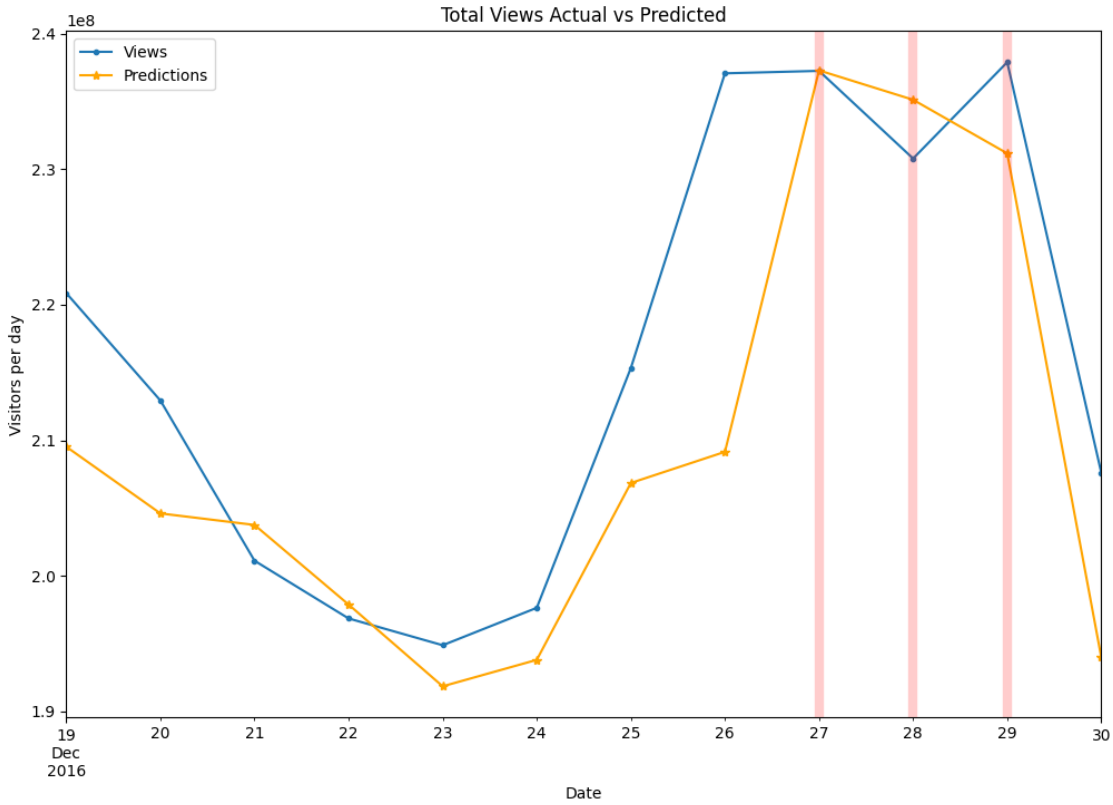
```
model = model.fit()


exog_forecast = test_x[['exogenous_variable']]
predictions = model.predict(start=start,
                            end=end,
                            exog=exog_forecast).rename('Predictions')
```

```
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           11     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.75741D+01    |proj g|=  2.86528D-01

At iterate    5    f=  1.74493D+01    |proj g|=  2.35042D-02

At iterate   10    f=  1.74240D+01    |proj g|=  3.71417D-02

At iterate   15    f=  1.74114D+01    |proj g|=  7.08839D-03

At iterate   20    f=  1.74104D+01    |proj g|=  2.33329D-03

At iterate   25    f=  1.74103D+01    |proj g|=  3.01615D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value
```

```
          * * *

   N    Tit      Tnf   Tnint  Skip  Nact     Projg        F
   11     25      28       1     0     0   3.016D-05   1.741D+01
  F =   17.410345239025130

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

[47]:
```
performance(test_x['Views'], predictions)
# Plot predictions against known values
title='Total Views Actual vs Predicted'
ylabel='Visitors per day'
xlabel='Date'

ax = test_x['Views'].plot(legend=True,figsize=(12,8),title=title, style = ".-")
predictions.plot(legend=True, color = 'orange', style = '-*')
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in test_x.query('exogenous_variable==1').index:
    ax.axvline(x=x, color='red', alpha = 0.2, linewidth = 6)
```

```
MAE : 7608480.02
RMSE : 10529680.704
MAPE: 0.034
```



Lets forecast for a specific language

[62]:
```
filtered_data_1 = filtered_data[filtered_data['page_language'] == 'es']
data_for_one_language = filtered_data.groupby("Date").agg(Views = pd.
 ↪NamedAgg(column="Views", aggfunc="sum"))
```

[65]:
```
exog_value_array = []
for index, row in data_for_one_language.iterrows():
    exog_value_array.append(exog_data.loc[index]['Exog'])

data_for_one_language['exogenous_variable'] = exog_value_array
```

3.3.1 Modelling data for a single language ('en' in our case)

[66]:
```
train_x = data_for_one_language.iloc[:-13].copy()
test_x = data_for_one_language.iloc[-13:-1].copy()
```

[67]:
```
# Standard Parameters
p = 5
d = 1
q = 0

## Seasonal Parameters
P = 3
D = 0
Q = 1
S = 7

start=len(train_x)
end=len(train_x)+len(test_x)-1

model = SARIMAX(train_x.Views, exog=train_x['exogenous_variable'], order=(p, d,␣
 ↪q), seasonal_order=(P,D,Q,S))
model = model.fit()


exog_forecast = test_x[['exogenous_variable']]
predictions = model.predict(start=start,
                            end=end,
                            exog=exog_forecast).rename('Predictions')
```

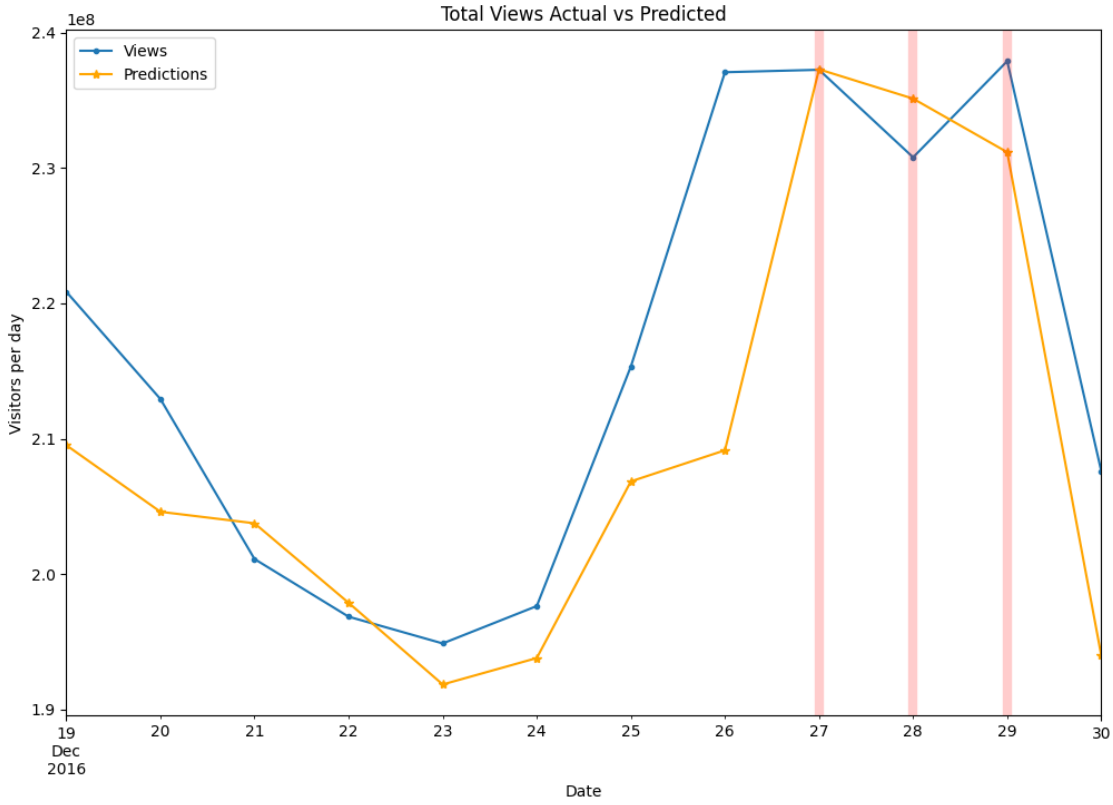
```
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
```

```
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           11     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.75741D+01    |proj g|=  2.86528D-01

At iterate    5    f=  1.74493D+01    |proj g|=  2.35042D-02

At iterate   10    f=  1.74240D+01    |proj g|=  3.71417D-02

At iterate   15    f=  1.74114D+01    |proj g|=  7.08839D-03

At iterate   20    f=  1.74104D+01    |proj g|=  2.33329D-03

At iterate   25    f=  1.74103D+01    |proj g|=  3.01615D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
   11     25     28      1     0     0   3.016D-05   1.741D+01
  F =   17.410345239025130

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

[68]:
```
performance(test_x['Views'], predictions)
# Plot predictions against known values
title='Total Views Actual vs Predicted'
ylabel='Visitors per day'
```

```
xlabel='Date'

ax = test_x['Views'].plot(legend=True,figsize=(12,8),title=title, style = ".-")
predictions.plot(legend=True, color = 'orange', style = '-*')
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
for x in test_x.query('exogenous_variable==1').index:
    ax.axvline(x=x, color='red', alpha = 0.2, linewidth = 6)
```

```
MAE : 7608480.02
RMSE : 10529680.704
MAPE: 0.034
```



3.4 Facebook Prophet

[71]:
```
from prophet import Prophet
sample_data_for_prediction.info()
```

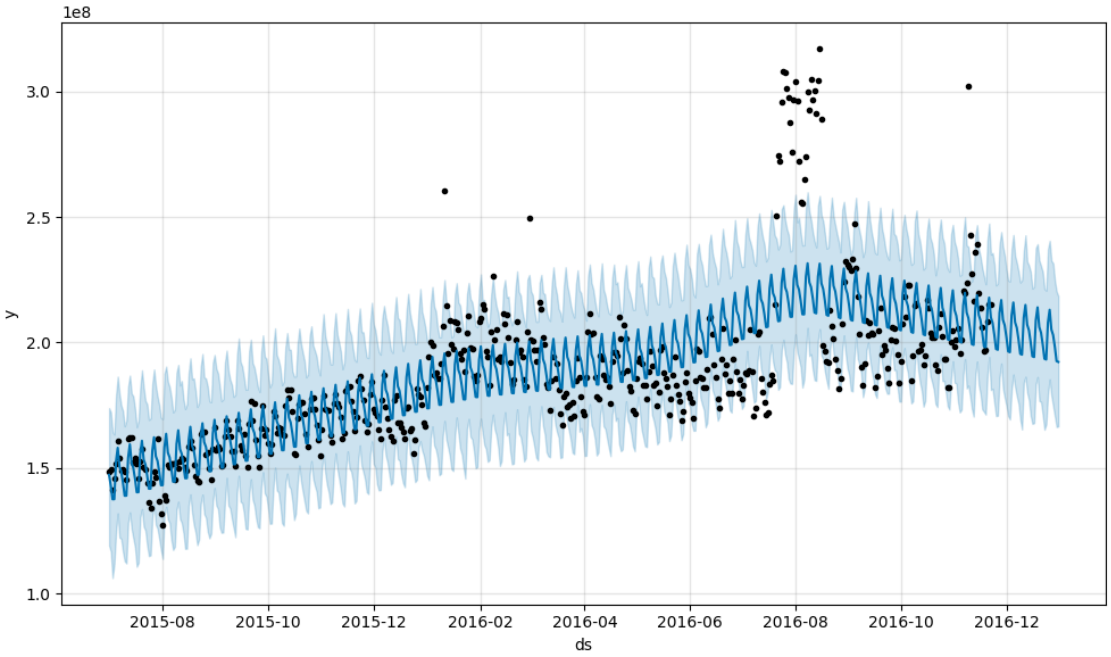
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 550 entries, 2015-07-01 to 2016-12-31
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype
```

```
---  ------              --------------  -----
 0   Views               550 non-null    float64
 1   exogenous_variable  550 non-null    int64
dtypes: float64(1), int64(1)
memory usage: 12.9 KB
```

[73]:
```
sample_data_for_prediction['ds'] = pd.to_datetime(sample_data_for_prediction.
 ↪index)
sample_data_for_prediction['y'] = sample_data_for_prediction['Views']
sample_data_for_prediction = sample_data_for_prediction[['ds','y',␣
 ↪'exogenous_variable']]
```

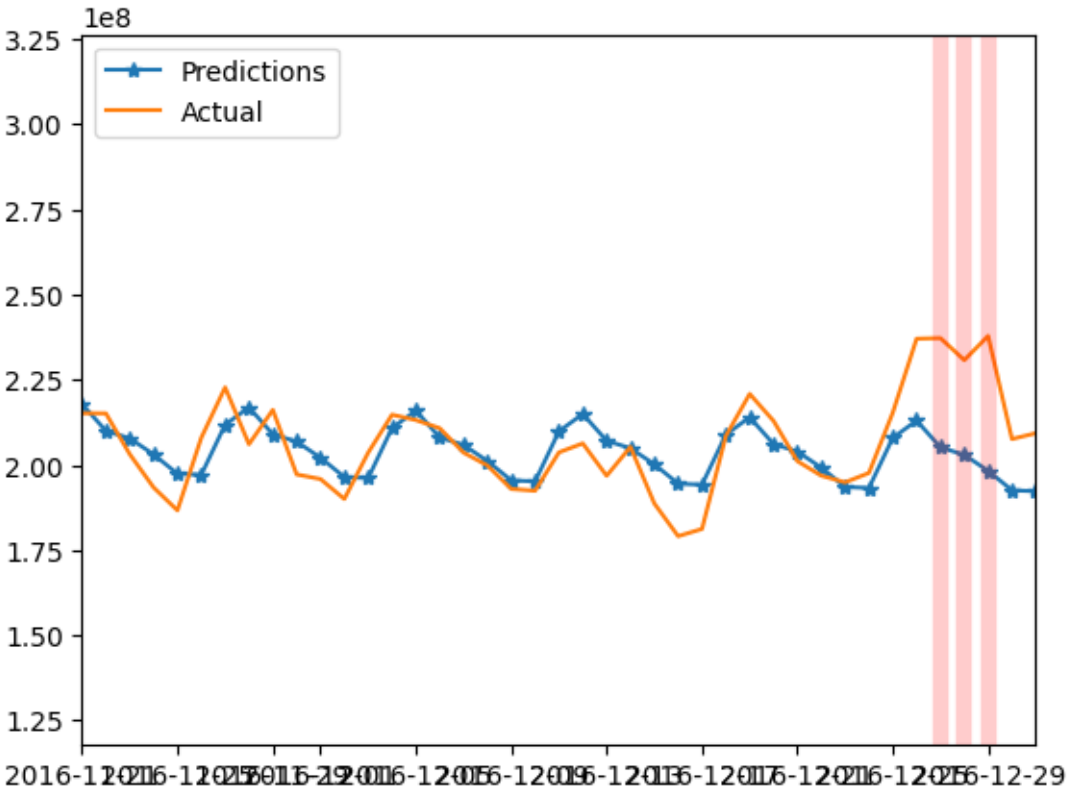
[75]:
```
# create model
m = Prophet()


# fit Model
m.fit(sample_data_for_prediction[['ds', 'y']][:-39]) #here we are leaving last␣
 ↪39 observations because we will predict it in 'future'

# predict Future
# future Dataframe
future = m.make_future_dataframe(periods=39,freq="D")

# prediction
forecast = m.predict(future)
fig = m.plot(forecast)
```

```
17:54:41 - cmdstanpy - INFO - Chain [1] start processing
17:54:42 - cmdstanpy - INFO - Chain [1] done processing
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:72: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:73: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```



[80]:
```
plt.plot(forecast['ds'], forecast['yhat'],'-*', label = 'Predictions')
plt.plot(sample_data_for_prediction['ds'], sample_data_for_prediction['y'],␣
 ↪label = 'Actual')
for x in sample_data_for_prediction.query('exogenous_variable==1')['ds']:
    plt.axvline(x=x, color='red', alpha = 0.2, linewidth = 6);
plt.xlim(pd.to_datetime('2016-11-21'), pd.to_datetime('2016-12-31'))
plt.legend()
```

[80]:
```
<matplotlib.legend.Legend at 0x150151a50>
```



[81]:
```
performance(sample_data_for_prediction['y'][:-39],forecast['yhat'][:-39])
```

```
MAE : 13875695.203
RMSE : 21363660.413
MAPE: 0.067
```

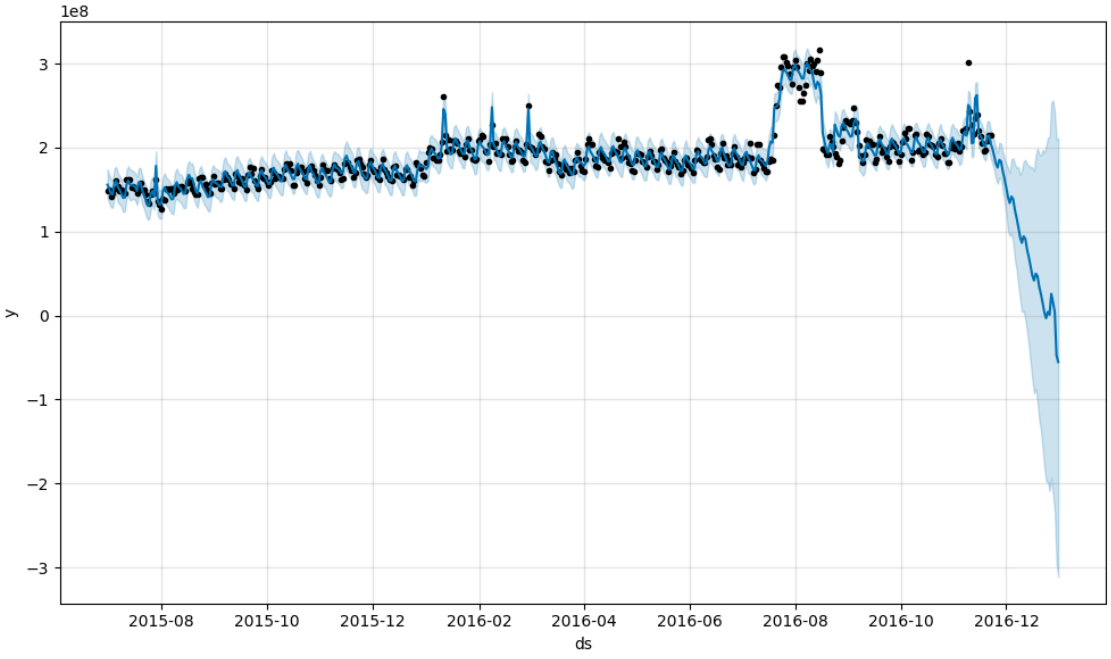
3.4.1 Exogeneous and Change Point scale Parameters

- higher Value of Change Point scale Parameters will make your model more flexting for Trend Changes

[82]:
```
model2=Prophet(interval_width=0.95,
               yearly_seasonality=True,
               weekly_seasonality=True,
               changepoint_prior_scale=4) # default = 0.05

# Add Exogeneous Variable
model2.add_regressor('exogenous_variable') #adding holidays data in the model3

# train Model
model2.fit(sample_data_for_prediction[:-39])
```

```
# predict
forecast2 = model2.predict(sample_data_for_prediction)

# plot
fig = model2.plot(forecast2)
```

```
18:00:32 - cmdstanpy - INFO - Chain [1] start processing
18:00:33 - cmdstanpy - INFO - Chain [1] done processing
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:72: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:73: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```

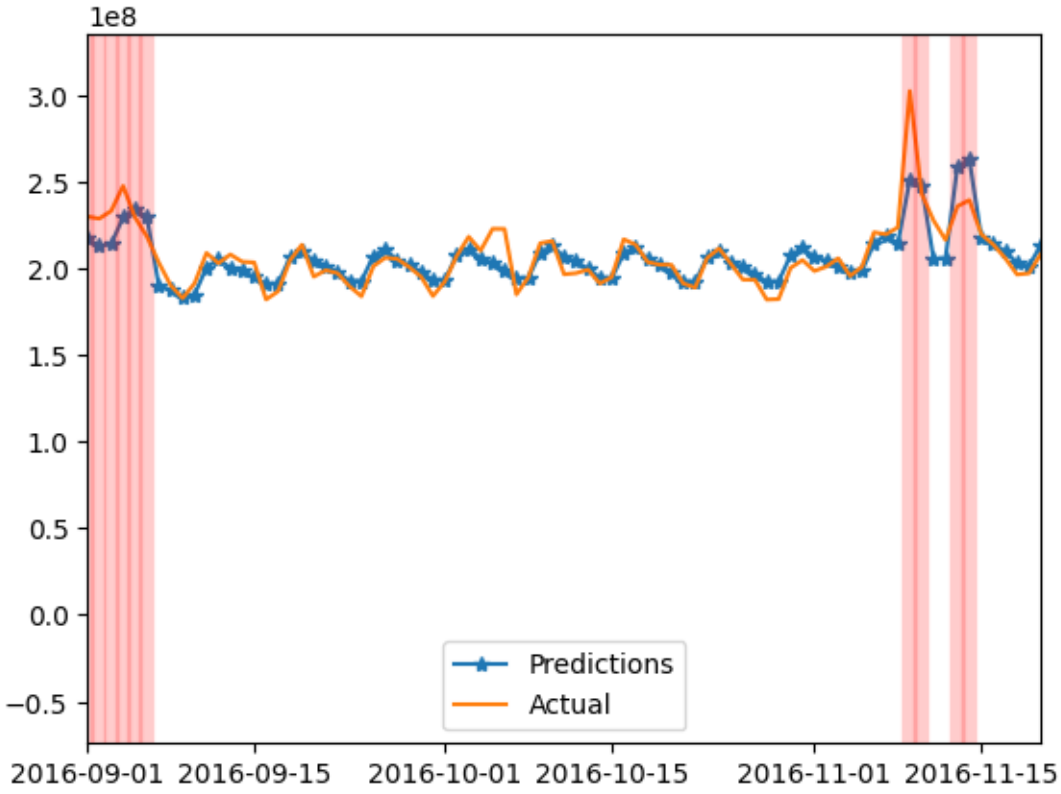


[84]:
```
performance(sample_data_for_prediction['y'][:-39],forecast2['yhat'][:-39])
```

```
MAE : 5904034.595
RMSE : 8716671.03
MAPE: 0.03
```

[89]:
```
plt.plot(forecast2['ds'], forecast2['yhat'],'-*', label = 'Predictions')
plt.plot(sample_data_for_prediction['ds'], sample_data_for_prediction['y'],␣
 ↪label = 'Actual')
plt.xlim(pd.to_datetime('2016-09-01'), pd.to_datetime('2016-11-20'))
for x in sample_data_for_prediction.query('exogenous_variable==1')['ds']:
    plt.axvline(x=x, color='red', alpha = 0.2, linewidth = 6);
plt.legend()
```

[89]: <matplotlib.legend.Legend at 0x136921a50>



[90]:
```
from prophet.plot import add_changepoints_to_plot
fig = m.plot(forecast2)
a = add_changepoints_to_plot(fig.gca(), m, forecast2)
```

```
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:72: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
/opt/homebrew/lib/python3.11/site-packages/prophet/plot.py:73: FutureWarning:
The behavior of DatetimeProperties.to_pydatetime is deprecated, in a future
version this will return a Series containing python datetime objects instead of
```

```
an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```