Problem statement

XYZ is an advertisements and marketing based company helping businesses get maximum clicks at minimum cost. The Data Science team of XYZ trying to understand the per page daily view report for 145k wikipedia pages for 550 days, and forecasting the number of views so that we can predict and optimize the ad placement for your clients. The clients belong to different regions and we need to forecast how their ads will perform on pages in different languages.

There are two csv files given:

- train_1.csv: In the csv file, each row corresponds to a particular article page and each column to a date. The values are the number of visits on that date. The page name contains data in this format: SPECIFIC NAME _ LANGUAGE.wikipedia.org _ ACCESS TYPE _ ACCESS ORIGIN having information about the page name, the main domain, the device type used to access the page, and also the request origin(spider or browser agent)
- Exog_Campaign_eng: This file contains data just for the pages in English for the dates which had a campaign or significant event that could affect the views for that day. There's 1 for dates with campaigns and 0 for remaining dates. It is to be treated as an exogenous variable for models when training and forecasting data for pages in English.

Where can this and modifications of this be used?

We can use this to forecast number of views in different languages and target ads in those language pages with maximum views so that their clients can get maximum views at an economically reasonable price per ad. It can be used to display advertisements on frequently visited webpages, leading online news websites, OTT platforms, social media websites such as Youtube, LinkedIn, etc.

In [1]:

```
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt
df = pd.read_csv('train_1.csv')
df.head(10)
```

Out[1]:

	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	...	2016-12-2
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.(
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.(
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.(
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.(
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.(
5	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0	17.0	24.0	...	16.(
6	91Days_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.(
7	A'N'D_zh.wikipedia.org_all-access_spider	118.0	26.0	30.0	24.0	29.0	127.0	53.0	37.0	20.0	...	64.(
8	AKB48_zh.wikipedia.org_all-access_spider	5.0	23.0	14.0	12.0	9.0	9.0	35.0	15.0	14.0	...	34.(
9	ASCII_zh.wikipedia.org_all-access_spider	6.0	3.0	5.0	12.0	6.0	5.0	4.0	13.0	9.0	...	25.(

10 rows × 551 columns

We can see there are many NaN values in some rows.

In [2]:
```
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
```

The date columns range from 2015-07-01 to 2016-12-31, so 17 months and 550 days.

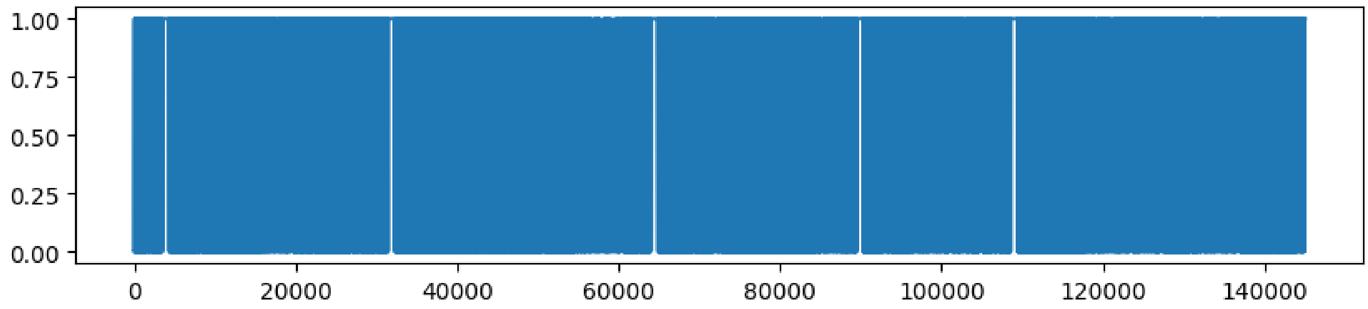
In [3]:
```
df.isnull().sum()
```

Out[3]:
```
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
```

In [4]:
```
nullrows = df.isna().any(axis=1)
plt.figure(figsize =(10,2))
```

```python
plt.plot(nullrows)
```

Out[4]: [<matplotlib.lines.Line2D at 0x2675acc3dd0>]



Reading the exogenous variable file

In [5]:
```python
contents = []
with open("Exog_Campaign_eng", "r") as file:
    record = []  # Initialize an empty list to store lines for a single record
    for line in file:
        line = line.strip()  # Remove leading/trailing whitespace, including '\n'

        # Check if the line is the end of a record (EOD)
        if line == "\\n":
            # Process the accumulated record
            if record:
                contents.append(record)
            # Reset the record list for the next record
            record = []
        else:
            # Add the line to the current record
            record.append(line)

# Ensure that the last record is processed if it doesn't end with "\\n"
if record:
    contents.append(record)
```

In [6]:
```python
contents = np.array(contents).reshape(-1,1)
contents = contents[1:]
contents = contents.astype(int)
contents[:5]
```

Out[6]:
```
array([[0],
       [0],
       [0],
       [0],
       [0]])
```

In [7]:
```python
contents.shape  # 550 exogenous values for 550 days
```

Out[7]: (550, 1)

In [8]:
```python
np.unique(contents, return_counts=True)
```

Out[8]: (array([0, 1]), array([496, 54], dtype=int64))

Understanding the page name format and splitting it to get language information.

In [9]:
```python
df['language'] = df['Page'].apply(lambda x:x.split('wikipedia.org')[0][-3:-1])
```

```
df['language'][:5]
```

Out[9]:
```
0    zh
1    zh
2    zh
3    zh
4    zh
Name: language, dtype: object
```

Converting the data to a format that can be fed to the Arima model (Pivoting etc)

In [10]:
```
df.drop('Page', axis=1, inplace=True)
df.head()
```

Out[10]:

	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	...	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016 12-2
0	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	...	63.0	15.0	26.0	14.0	20.
1	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	...	42.0	28.0	15.0	9.0	30.
2	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	...	1.0	1.0	7.0	4.0	4.
3	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	...	10.0	26.0	27.0	16.0	11.
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	9.0	25.0	13.0	3.0	11.

5 rows × 551 columns

In [11]:
```
df2 = df.groupby('language').sum()
df2
```

Out[11]:

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-
de	13299837.0	13142154.0	12615201.0	11573244.0	13470112.0	14814542.0	14347745.0	14611691.0	1401976
en	84712190.0	84438545.0	80167728.0	83463204.0	86198637.0	92809436.0	87838054.0	82880196.0	8479891
es	15278553.0	14601013.0	13427632.0	12606538.0	13710356.0	15625554.0	15230654.0	14781870.0	1450290
fr	8458638.0	8512952.0	8186030.0	8749842.0	8590493.0	8949799.0	8650800.0	8491533.0	840364
ja	11863200.0	13620792.0	12305383.0	15456239.0	14827204.0	12920547.0	12568828.0	12492787.0	1217825
nt	1451216.0	1499552.0	1415102.0	1207208.0	1318756.0	1529170.0	1643691.0	1662875.0	148849
ru	9463854.0	9627643.0	8923463.0	8393214.0	8938528.0	9628896.0	9408180.0	9364117.0	959230
zh	4144988.0	4151189.0	4123659.0	4163448.0	4441286.0	4464290.0	4459421.0	4575842.0	454784

8 rows × 550 columns

In [12]:
```
df2 = df2.transpose()
```

In [13]:
```
df2.head()
```

Out[13]:

language	de	en	es	fr	ja	nt	ru	zh
2015-07-01	13299837.0	84712190.0	15278553.0	8458638.0	11863200.0	1451216.0	9463854.0	4144988.0
2015-07-02	13142154.0	84438545.0	14601013.0	8512952.0	13620792.0	1499552.0	9627643.0	4151189.0
2015-07-03	12615201.0	80167728.0	13427632.0	8186030.0	12305383.0	1415102.0	8923463.0	4123659.0
2015-07-04	11573244.0	83463204.0	12606538.0	8749842.0	15456239.0	1207208.0	8393214.0	4163448.0
2015-07-05	13470112.0	86198637.0	13710356.0	8590493.0	14827204.0	1318756.0	8938528.0	4441286.0

In [14]:

```
df2.index.names  = ['date']
df2.index = pd.to_datetime(df2.index)
df2.head()
```

Out[14]:

language	de	en	es	fr	ja	nt	ru	zh
date								
2015-07-01	13299837.0	84712190.0	15278553.0	8458638.0	11863200.0	1451216.0	9463854.0	4144988.0
2015-07-02	13142154.0	84438545.0	14601013.0	8512952.0	13620792.0	1499552.0	9627643.0	4151189.0
2015-07-03	12615201.0	80167728.0	13427632.0	8186030.0	12305383.0	1415102.0	8923463.0	4123659.0
2015-07-04	11573244.0	83463204.0	12606538.0	8749842.0	15456239.0	1207208.0	8393214.0	4163448.0
2015-07-05	13470112.0	86198637.0	13710356.0	8590493.0	14827204.0	1318756.0	8938528.0	4441286.0

In [15]:

```
df2.columns
```

Out[15]:

```
Index(['de', 'en', 'es', 'fr', 'ja', 'nt', 'ru', 'zh'], dtype='object', name='language')
```

In [16]:

```
df2.columns = df2.columns.values
```

In [17]:

```
df2.columns
```

Out[17]:

```
Index(['de', 'en', 'es', 'fr', 'ja', 'nt', 'ru', 'zh'], dtype='object')
```

In [18]:

```
df2.head()
```

Out[18]:

	de	en	es	fr	ja	nt	ru	zh
date								
2015-07-01	13299837.0	84712190.0	15278553.0	8458638.0	11863200.0	1451216.0	9463854.0	4144988.0
2015-07-02	13142154.0	84438545.0	14601013.0	8512952.0	13620792.0	1499552.0	9627643.0	4151189.0
2015-07-03	12615201.0	80167728.0	13427632.0	8186030.0	12305383.0	1415102.0	8923463.0	4123659.0
2015-07-04	11573244.0	83463204.0	12606538.0	8749842.0	15456239.0	1207208.0	8393214.0	4163448.0
2015-07-05	13470112.0	86198637.0	13710356.0	8590493.0	14827204.0	1318756.0	8938528.0	4441286.0

In [19]:

```
df2.isnull().sum()
```

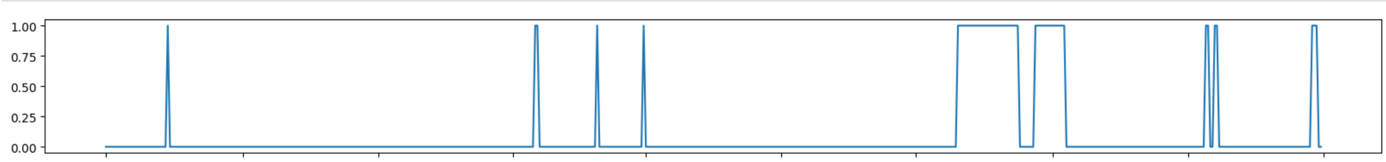
```
Out[19]: de    0
         en    0
         es    0
         fr    0
         ja    0
         nt    0
         ru    0
         zh    0
         dtype: int64
```

Data visualization

```
In [20]: #plt.figure(figsize=(20,10))
         df2.plot(figsize =(20,10), style = 'o-' )
         plt.legend()
         #plt.show(style = 'o-')
```

```
Out[20]: <matplotlib.legend.Legend at 0x26754a87910>
```


```
In [21]: plt.figure(figsize=(20,2))
         plt.plot(df2.index, contents)  #exogenous variable
         plt.show()
```

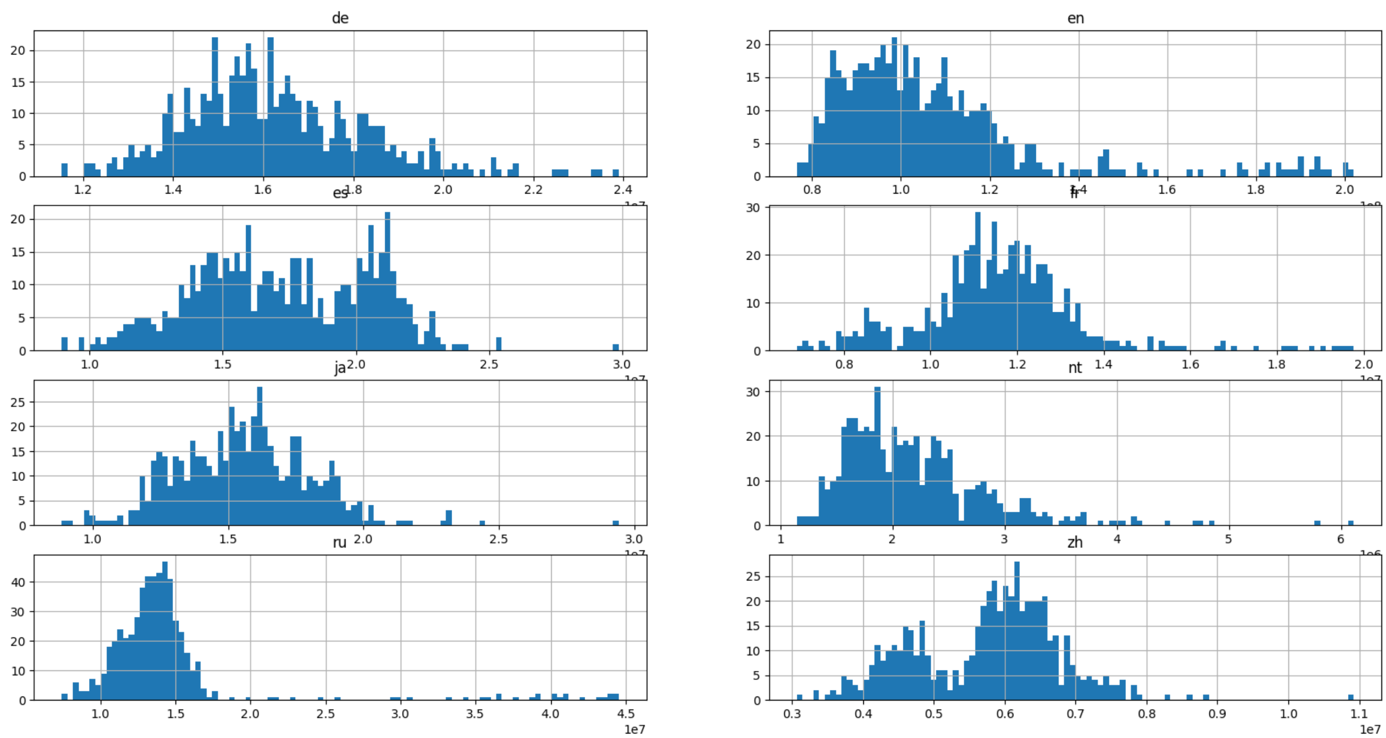


Inferences from the data visualizations:

1. English language webpages have much higher number of views compared to any other language websites.
2. While some time series have varying mean such as Spanish (es) and Japanese(ja), English time series is clearly having an increasing trend.

3. There are some events of sudden rises in number of views on certain days in English(en), Russian (ru), etc. that coincide with the exogenous variable plotted above.

In [22]:
```
plt.figure(figsize=(20,10))
for i,col in enumerate(df2.columns):
    plt.subplot(4,2,i+1)
    df2[col].hist(bins=100).plot()
    plt.title(col)
```



In [23]:
```
# df3 = df2.copy(deep=True)  #making new dataframe with outliers removed

# df3[col] = df3[col].clip(upper=df3[col].quantile(0.99), lower=df3[col].quantile(0.01))
```

In [24]:
```
# mn = df3[col].mean()
# df3[col].fillna(mn).plot(label='imputed')
# df3[col].plot(label='original')
# plt.legend()
```

In [25]:
```
# df3[col].interpolate(method='linear').plot(label='imputed')
# df3[col].plot(label='original')
# plt.legend()
```

Checking if the data is stationary using Augmented Dickey-Fuller test

The Augmented Dickey-Fuller (ADF) test is a statistical hypothesis test used to determine whether a time series is stationary or non-stationary. A stationary time series has constant statistical properties, such as mean, variance, and autocorrelation. Null hypothesis is that it is non-stationary, alternative hypothesis is it is stationary. If p-value < 0.05 we say null hypothesis is rejected.

Creating a pipeline for working with multiple series

In [26]:
```
import statsmodels.api as sm
for col in df2.columns:
    p_value = sm.tsa.stattools.adfuller(df2[col])[1]
```

```
    print("the p-value for {} is {}".format(col,p_value))
    # Check the result
    if p_value < 0.05:  # You can choose a significance level, e.g., 0.05
        print("Reject the null hypothesis. The time series is likely stationary.")
    else:
        print("Fail to reject the null hypothesis. The time series may be non-stationary.")
```

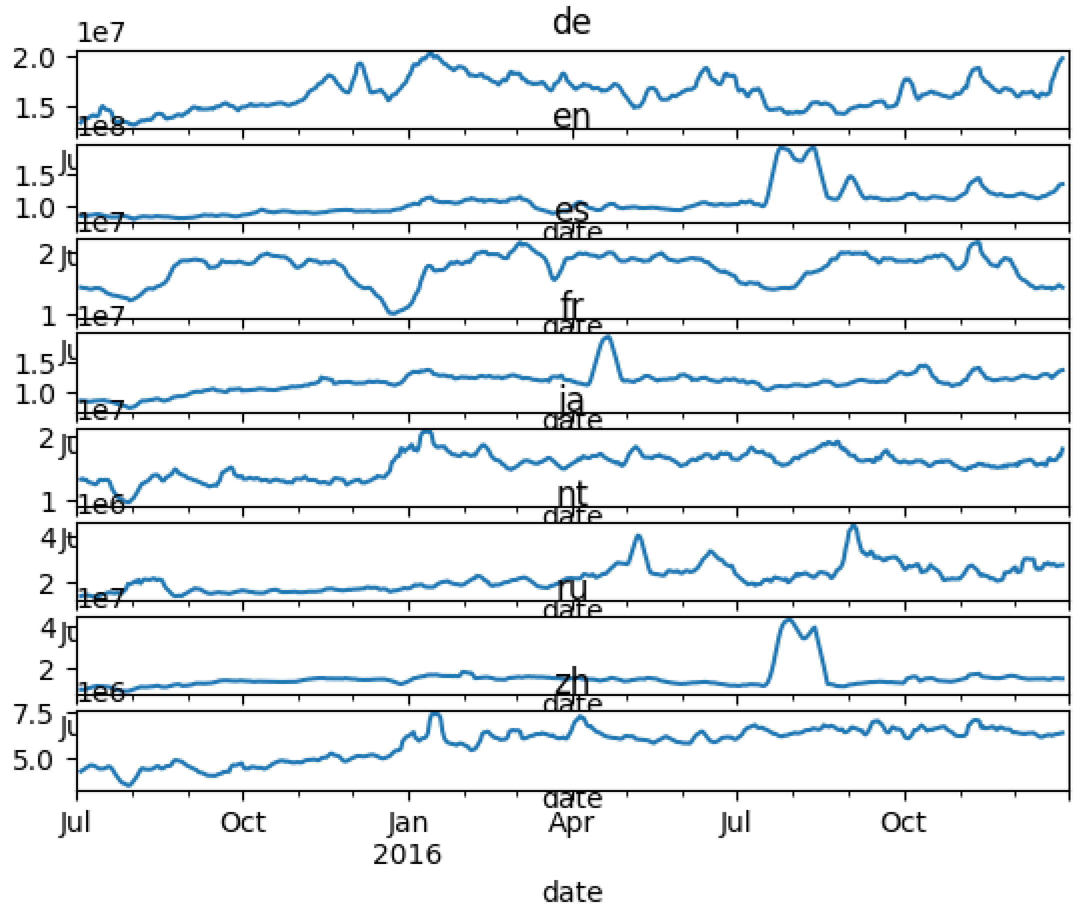
```
the p-value for de is 0.14125875658926407
Fail to reject the null hypothesis. The time series may be non-stationary.
the p-value for en is 0.18953359279992849
Fail to reject the null hypothesis. The time series may be non-stationary.
the p-value for es is 0.03358859084479074
Reject the null hypothesis. The time series is likely stationary.
the p-value for fr is 0.051495021952453604
Fail to reject the null hypothesis. The time series may be non-stationary.
the p-value for ja is 0.10257133898558674
Fail to reject the null hypothesis. The time series may be non-stationary.
the p-value for nt is 0.01332801076387422
Reject the null hypothesis. The time series is likely stationary.
the p-value for ru is 0.0018649376536621126
Reject the null hypothesis. The time series is likely stationary.
the p-value for zh is 0.4474457922931181
Fail to reject the null hypothesis. The time series may be non-stationary.
```

The time series of number of views for languages es, nt and ru are stationary, and those for languages de, en, fr, ja, zh are non-stationary. So we need to model them differently which means apply differencing to the non-stationary ones.

Decomposition of series

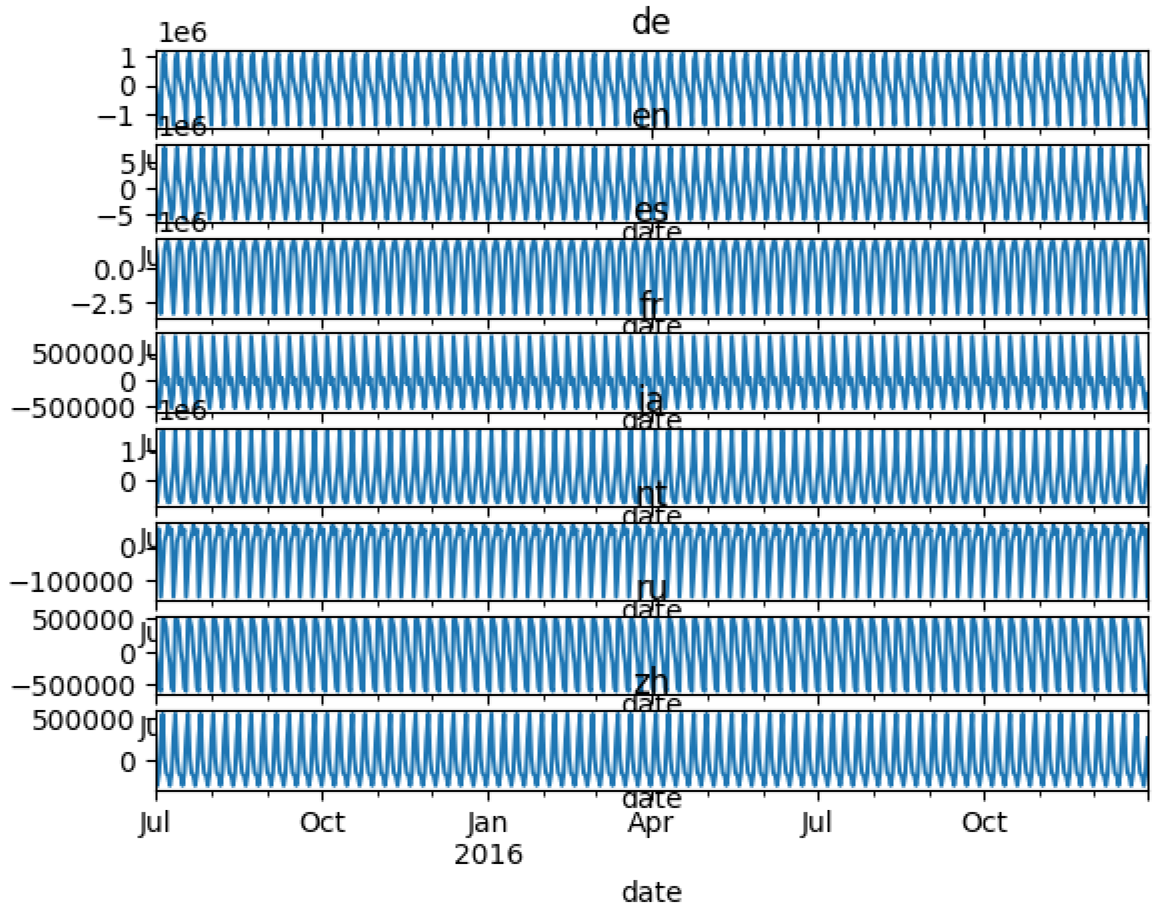
Time series decomposition involves separating of a series into level, trend, seasonality, and noise components.

In [27]:
```
import statsmodels.api as sm
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='additive')
    result.trend.plot()
    plt.title(col)
plt.show()
```



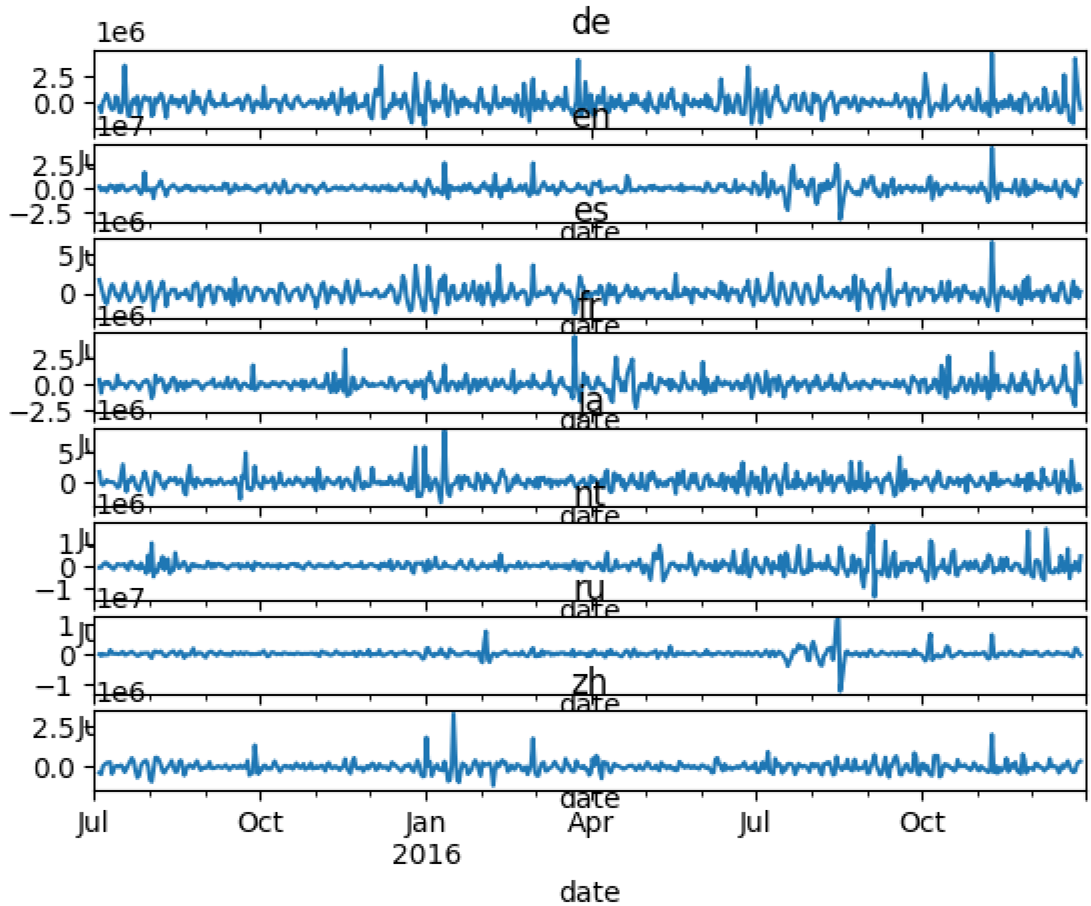
In [28]:

```
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='additive')
    result.seasonal.plot()
    plt.title(col)
plt.show()
```

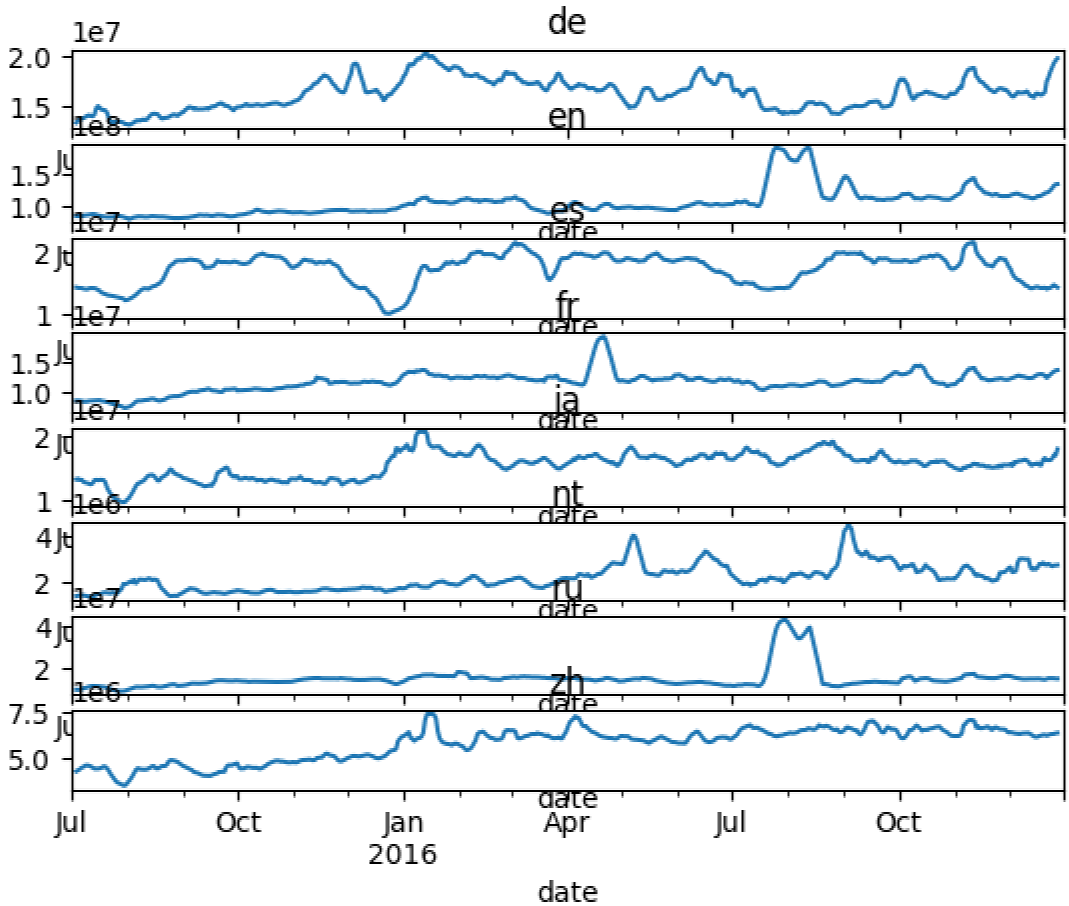


In [29]:

```
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='additive')
    result.resid.plot()
    plt.title(col)
plt.show()
```

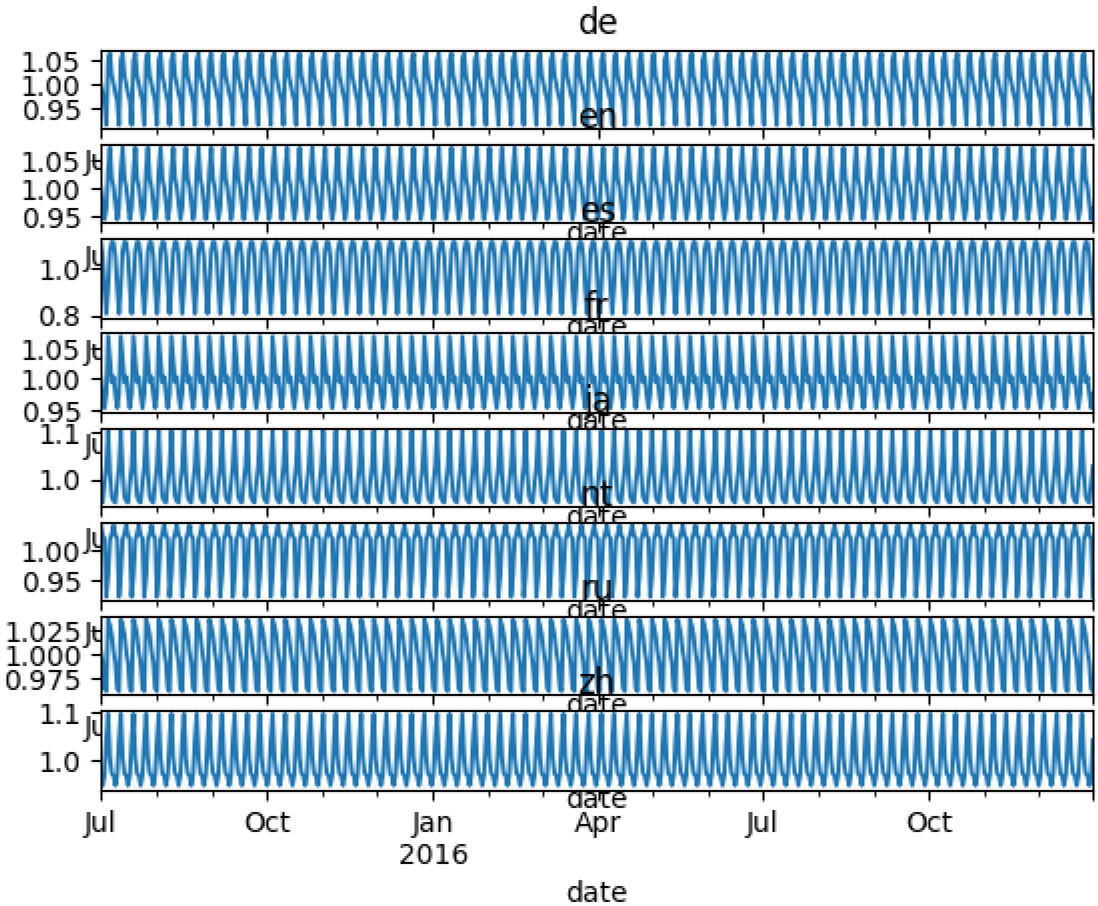


In [30]:
```
import statsmodels.api as sm
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='multiplicative')
    result.trend.plot()
    plt.title(col)
plt.show()
```



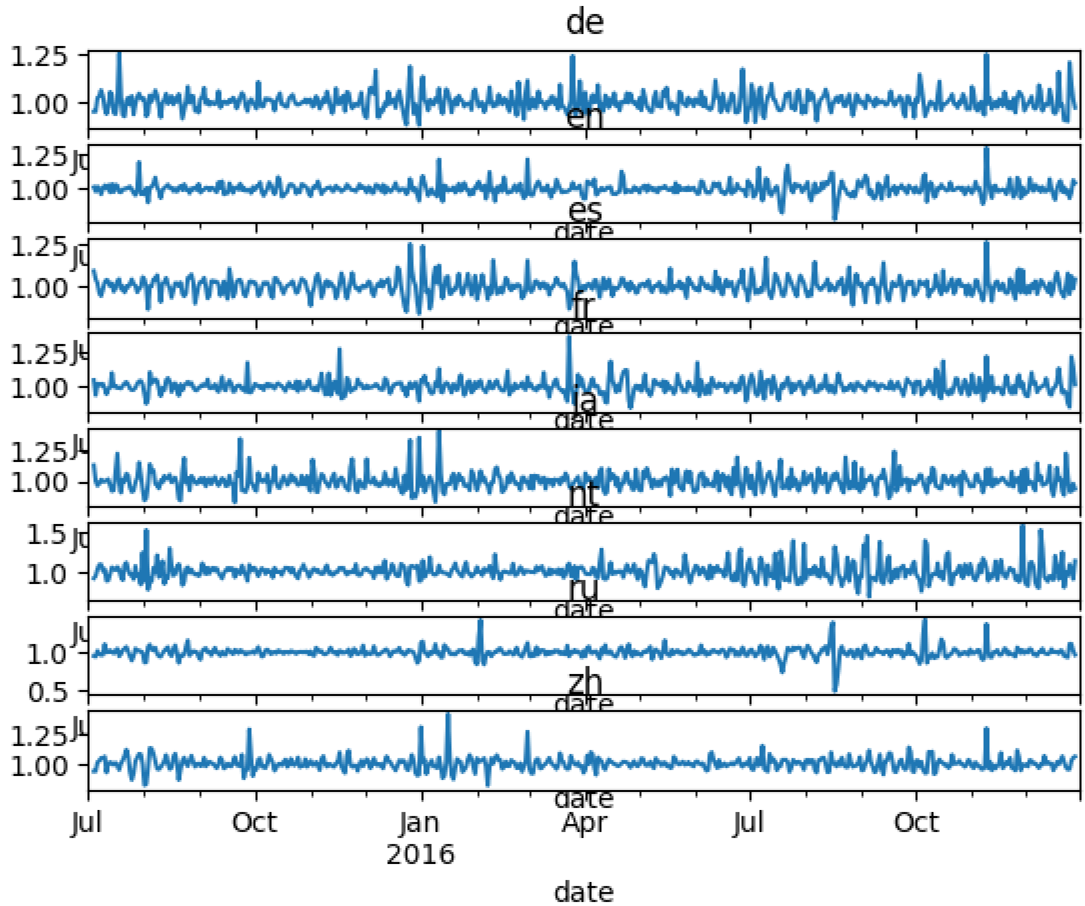
In [31]:

```
import statsmodels.api as sm
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='multiplicative')
    result.seasonal.plot()
    plt.title(col)
plt.show()
```



In [32]:

```
import statsmodels.api as sm
for i,col in enumerate(df2.columns):
    plt.subplot(8,1,i+1)
    result = sm.tsa.seasonal_decompose(df2[col], model='multiplicative')
    result.resid.plot()
    plt.title(col)
plt.show()
```



Time series differencing

In [33]:
```
df3 = df2.copy()
def adf_test(data, significance_level=0.05):
    pvalue = sm.tsa.stattools.adfuller(data)[1]
    if pvalue <= significance_level:
        print('Sequence is stationary')
    else:
        print('Sequence is not stationary')

for col in df2.columns:
    print(col)
    adf_test(df2[col])
    print('{} after differencing:'.format(col))
    adf_test(df2[col].diff(1).dropna())
    df3[col] = df2[col].diff(1).dropna()
```

```
de
Sequence is not stationary
de after differencing:
Sequence is stationary
en
Sequence is not stationary
en after differencing:
Sequence is stationary
es
Sequence is stationary
es after differencing:
Sequence is stationary
fr
Sequence is not stationary
fr after differencing:
Sequence is stationary
ja
Sequence is not stationary
ja after differencing:
Sequence is stationary
nt
Sequence is stationary
nt after differencing:
Sequence is stationary
ru
Sequence is stationary
ru after differencing:
Sequence is stationary
zh
Sequence is not stationary
zh after differencing:
Sequence is stationary
```

- What level of differencing gave you a stationary series?

Level 1 is sufficient so d=1.

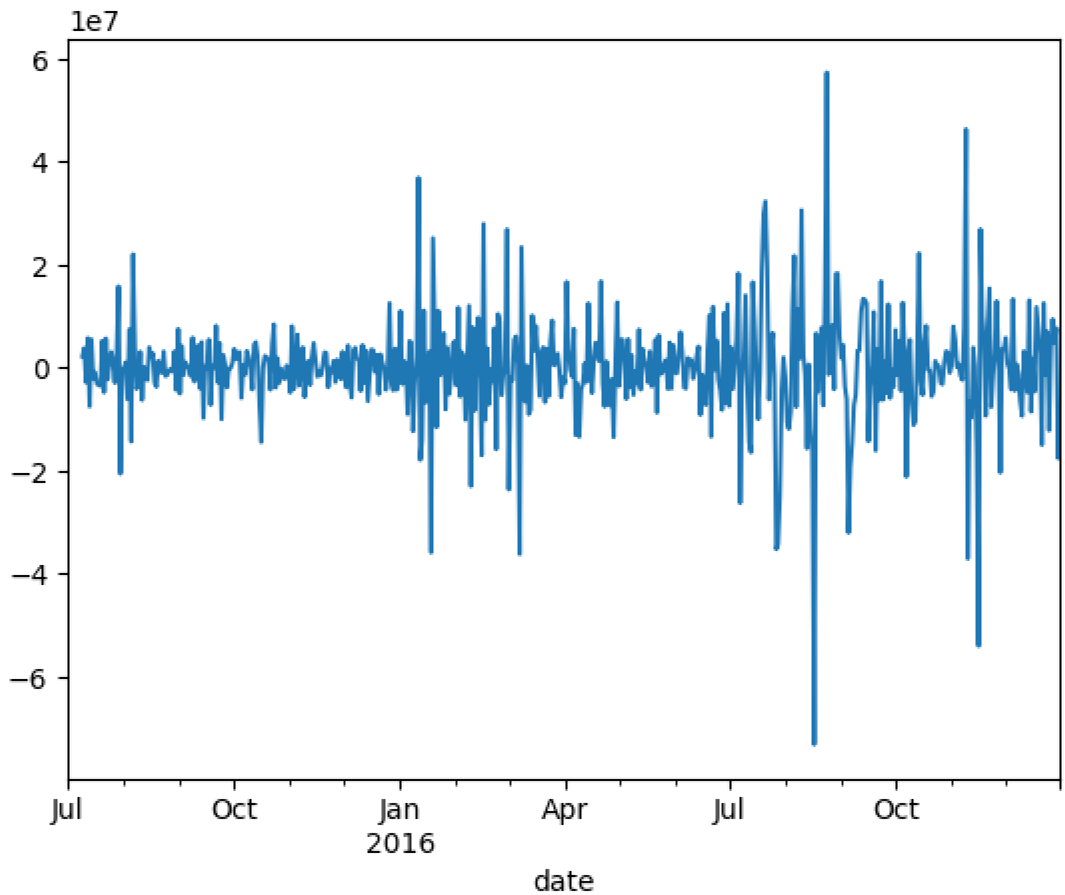
In [34]:
```
# Remove seasonality by differencing with a lag equal to the seasonal period (e.g., 7 days for we
seasonal_lag = 7
for col in df2.columns:
    print(col)
    adf_test(df3[col].diff(periods=seasonal_lag).dropna())
    df3[col] = df3[col].diff(periods=seasonal_lag).dropna()
```

```
de
Sequence is stationary
en
Sequence is stationary
es
Sequence is stationary
fr
Sequence is stationary
ja
Sequence is stationary
nt
Sequence is stationary
ru
Sequence is stationary
zh
Sequence is stationary
```

D=1 seasonal differencing.

In [35]:
```
df3.en.plot()
```

Out[35]: <AxesSubplot: xlabel='date'>

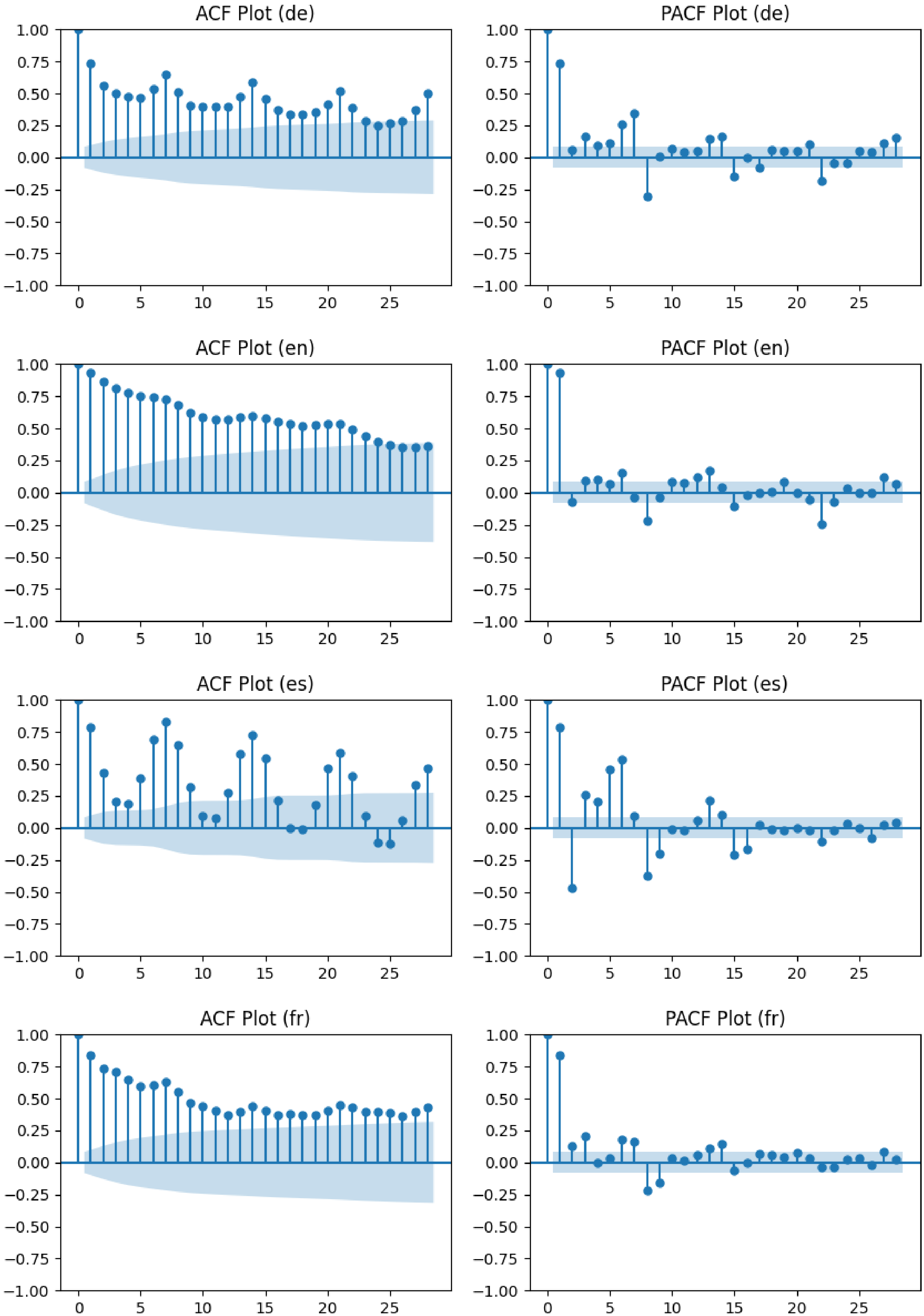


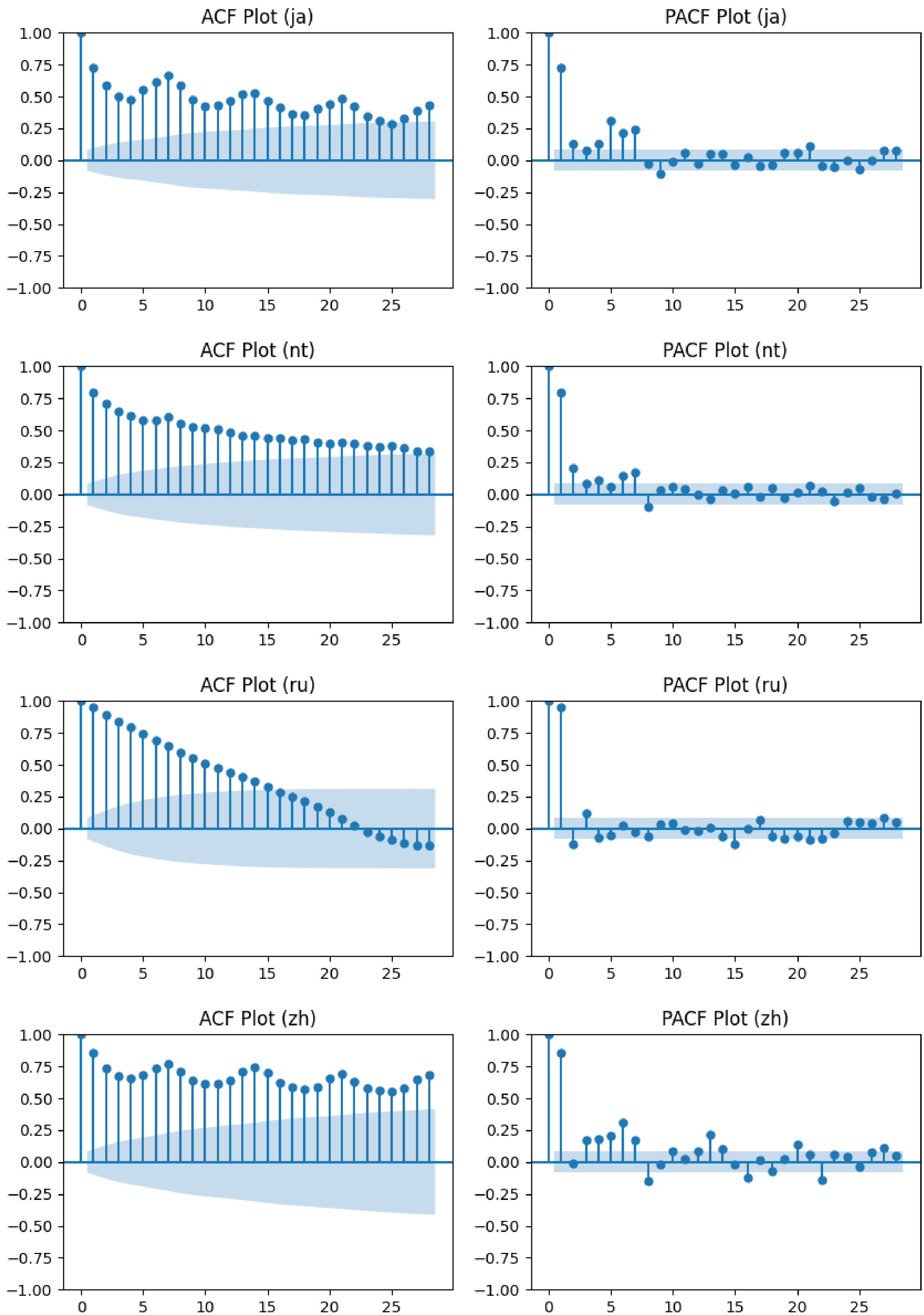
Plotting the ACF and PACF plots

The ACF plots the correlation between the data and its lagged values, while the PACF plots the correlation between the data and its lagged values after removing the effect of the intermediate values.

In [36]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plt.figure(figsize=(20,5))
for i,col in enumerate(df2.columns):
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10, 3))
    plot_acf(df2[col], ax=ax1)
    ax1.set_title('ACF Plot ({})'.format(col))

    plot_pacf(df2[col], ax=ax2)
    ax2.set_title('PACF Plot ({})'.format(col))

    plt.show()
```

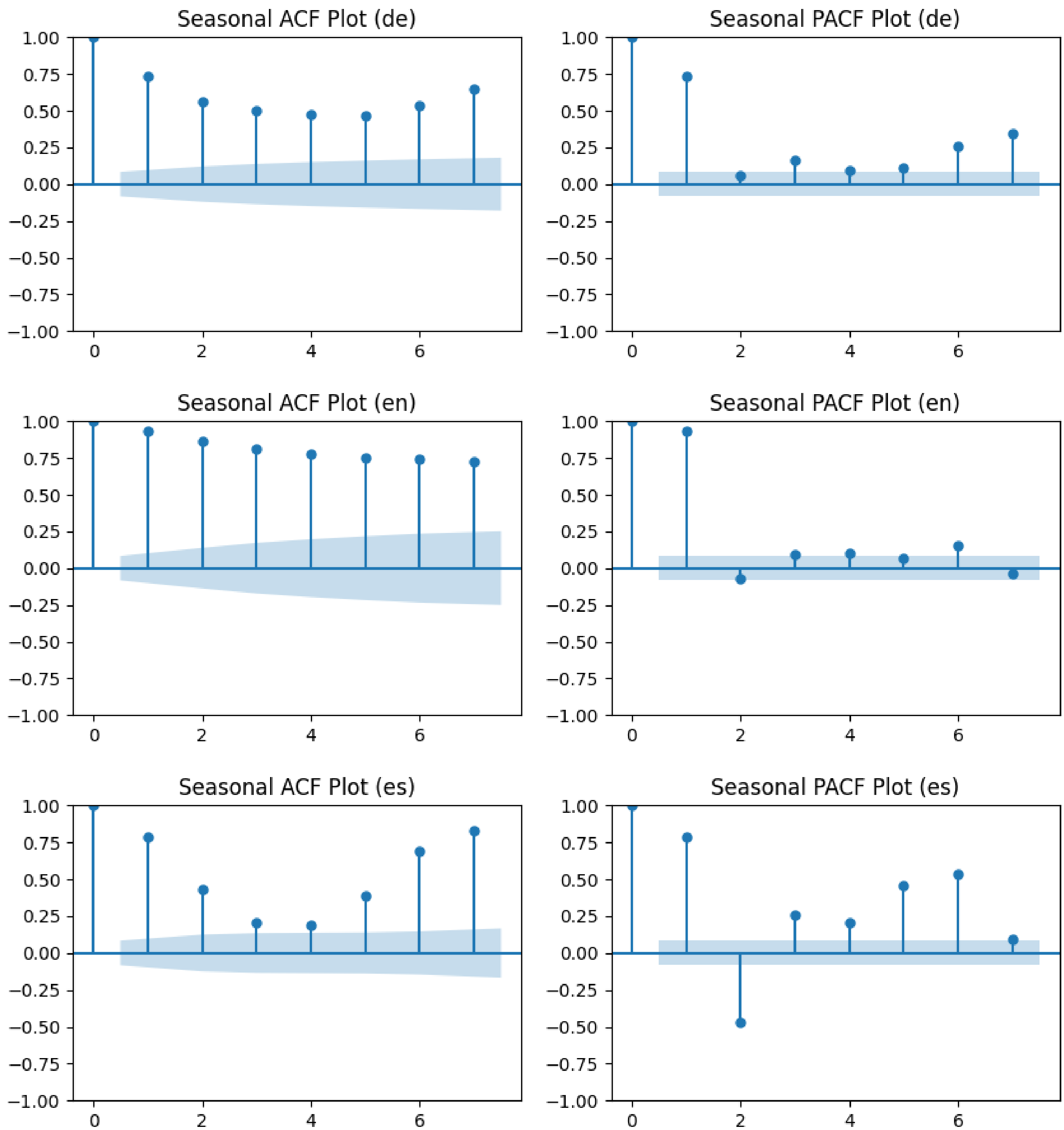
```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\statsmodels\graphics\ts
aplots.py:348: FutureWarning: The default method 'yw' can produce PACF values outside of the [-
1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can use
this method now by setting method='ywm'.
  warnings.warn(
<Figure size 2000x500 with 0 Axes>
```

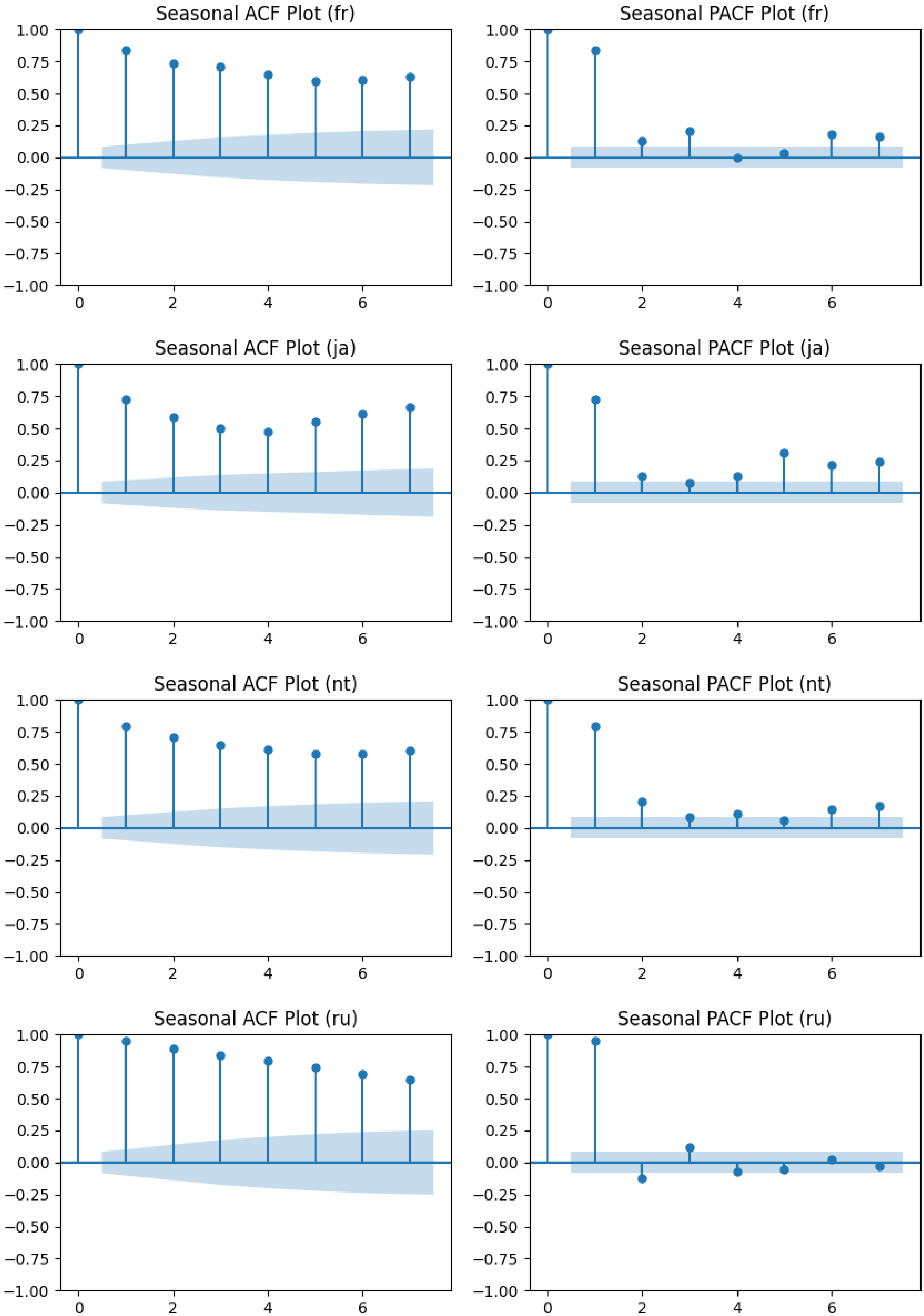

ACF Plot (de)
PACF Plot (de)
ACF Plot (en)
PACF Plot (en)
ACF Plot (es)
PACF Plot (es)
ACF Plot (fr)
PACF Plot (fr)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

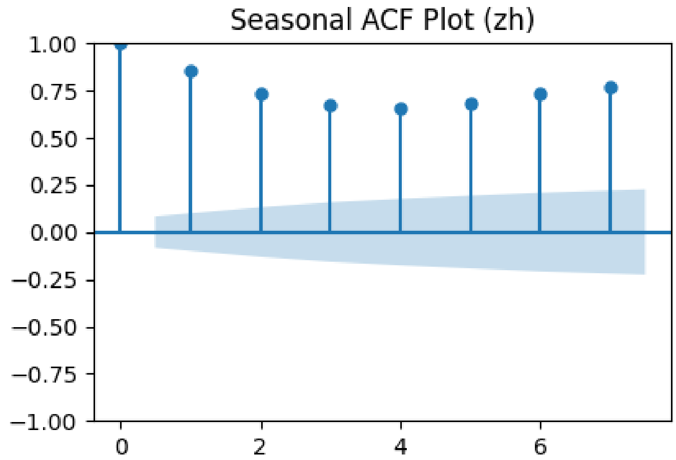


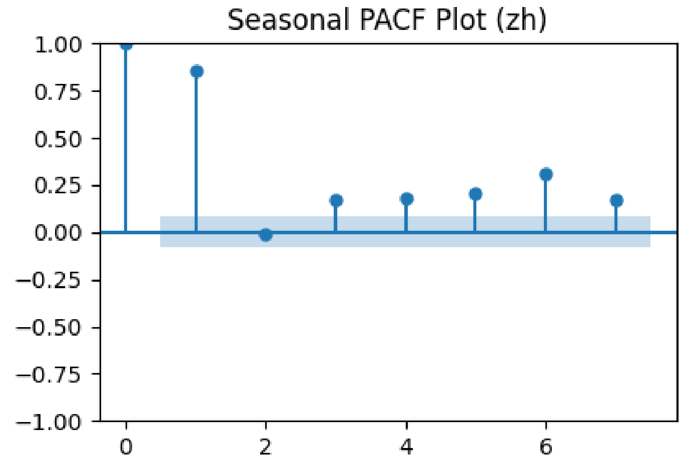
We should use p=1 Autoregressive model components as PACF plots show significant correlation between components only till lag 1 then it shuts off. We should use q=3 Moving Average model components as ACF plots show significant moving average components only till lag 3 in some series then it stays same or increases.

In [37]:
```
for i,col in enumerate(df2.columns):
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10, 3))
    plot_acf(df2[col], lags=7, ax=ax1)
    ax1.set_title('Seasonal ACF Plot ({})'.format(col))

    plot_pacf(df2[col], lags=7, ax=ax2)
    ax2.set_title('Seasonal PACF Plot ({})'.format(col))

    plt.show()
```




Seasonal ACF Plot (fr)
Seasonal PACF Plot (fr)
Seasonal ACF Plot (ja)
Seasonal PACF Plot (ja)
Seasonal ACF Plot (nt)
Seasonal PACF Plot (nt)
Seasonal ACF Plot (ru)
Seasonal PACF Plot (ru)





We should use P=1 Autoregressive model components as PACF plots show significant correlation between components only till lag 1 then it shuts off. We should use Q=3 Moving Average model components as ACF plots show significant moving average components only till lag 3 in some series then it stays same or increases.

ARIMA and SARIMAX model forecasting

- Difference between arima, sarima & sarimax.

ARIMA (AutoRegressive Integrated Moving Average) is used for a time series with a varying mean or trend (I component), and some dependence of data points on previous values (AR and MA components). SARIMA is used on data sets that have seasonal cycles. SARIMAX is used on data with seasonality and exogenous factors.

In [38]:
```
df2.shape
```
Out[38]:
```
(550, 8)
```

In [39]:
```
np.unique(contents, return_counts=True)
```
Out[39]:
```
(array([0, 1]), array([496,  54], dtype=int64))
```

In [40]:
```
exog = pd.Series(contents.flatten(), index=df2.index)
exog.unique()
```
Out[40]:
```
array([0, 1])
```

In [41]:
```
df4 =df2.copy()
df4['exog'] = exog
df4.shape
```
Out[41]:
```
(550, 9)
```

In [42]:
```
df4.isnull().sum()
```

Out[42]:
```
de       0
en       0
es       0
fr       0
ja       0
nt       0
ru       0
zh       0
exog     0
dtype: int64
```

In [43]:
```
# df4 = df4[8:]
df4.head(10)
```

Out[43]:

date	de	en	es	fr	ja	nt	ru	zh	exog
2015-07-01	13299837.0	84712190.0	15278553.0	8458638.0	11863200.0	1451216.0	9463854.0	4144988.0	0
2015-07-02	13142154.0	84438545.0	14601013.0	8512952.0	13620792.0	1499552.0	9627643.0	4151189.0	0
2015-07-03	12615201.0	80167728.0	13427632.0	8186030.0	12305383.0	1415102.0	8923463.0	4123659.0	0
2015-07-04	11573244.0	83463204.0	12606538.0	8749842.0	15456239.0	1207208.0	8393214.0	4163448.0	0
2015-07-05	13470112.0	86198637.0	13710356.0	8590493.0	14827204.0	1318756.0	8938528.0	4441286.0	0
2015-07-06	14814542.0	92809436.0	15625554.0	8949799.0	12920547.0	1529170.0	9628896.0	4464290.0	0
2015-07-07	14347745.0	87838054.0	15230654.0	8650800.0	12568828.0	1643691.0	9408180.0	4459421.0	0
2015-07-08	14611691.0	82880196.0	14781870.0	8491533.0	12492787.0	1662875.0	9364117.0	4575842.0	0
2015-07-09	14019764.0	84798911.0	14502906.0	8403646.0	12178258.0	1488498.0	9592309.0	4547843.0	0
2015-07-10	13074713.0	84319456.0	13184481.0	7930703.0	12652904.0	1499469.0	10984872.0	4727889.0	0

In [44]:
```
df4.shape
```

Out[44]:
```
(550, 9)
```

In [45]:
```
print(550-500)
print((550-500)/550)    # 9% kept in test dataset
```

```
50
0.09090909090909091
```

In [46]:
```
# splitting data into train and test datasets
train = df4.iloc[:500]
test = df4.iloc[500:]
```

ARIMA

In [47]:
```python
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)*100, 3))

title='Number of views- Actual vs Predicted'
ylabel='No of views'
xlabel='days'
```

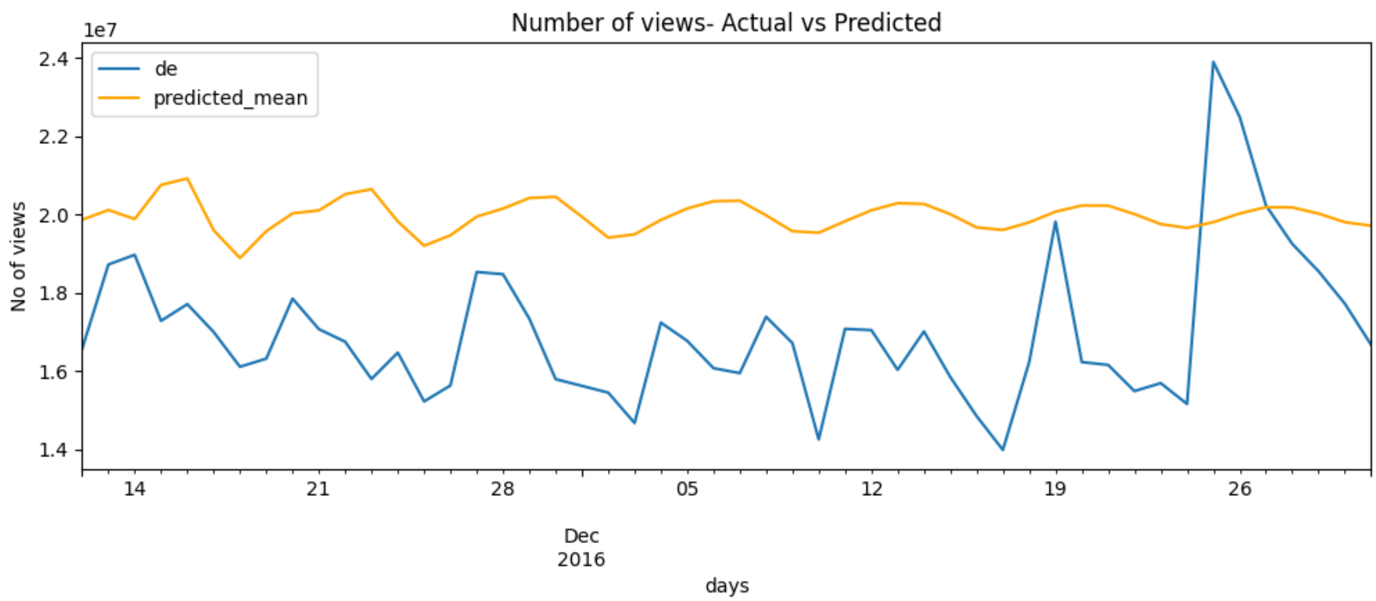
Trying out grid-search

In [53]:
```python
from itertools import product
import warnings
warnings.filterwarnings('ignore')

import statsmodels.api as sm
p_values = [1, 2, 3, 4, 5]  # AR order
q_values = [1, 2, 3, 4, 5]  # MA order
best_params = []

# Create a list of all possible parameter combinations
parameter_combinations = list(product(p_values, q_values))

# Perform grid search
for col in df3.columns:
    # Initialize variables to keep track of the best model and its performance
    best_model = None
    best_aic = np.inf  # Initialize with a high value (AIC should be minimized)
    bestp = 0
    bestq = 0
    for p, q in parameter_combinations:
        # Fit ARIMA model
        model = sm.tsa.ARIMA(train[col], order=(p, 1, q))
        results = model.fit()

        # Calculate AIC (Akaike Information Criterion)
        aic = results.aic
        # Update best model if the current model has a lower AIC
        if aic < best_aic:
            best_model = model
            best_aic = aic
            bestp = p
            bestq = q
    print("Best ARIMA Model for {}: p={} and q={}".format(col, bestp, bestq))
    best_params.append([p,q])
```

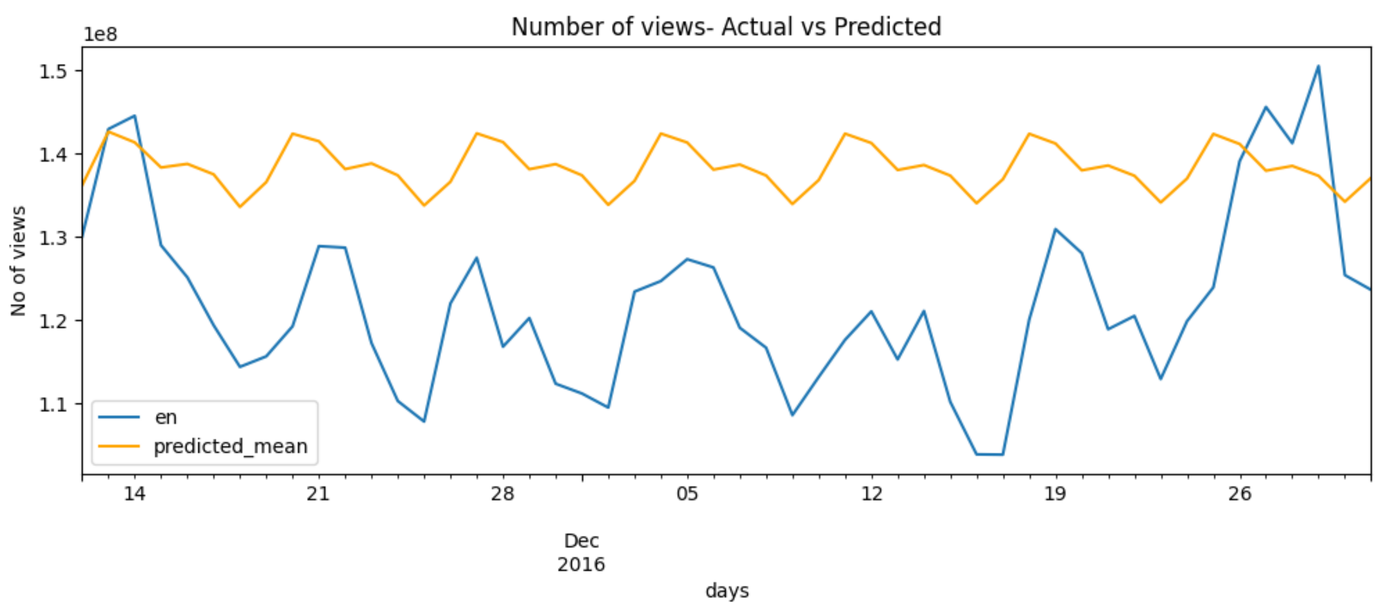
```
Best ARIMA Model for de: p=4 and q=5
Best ARIMA Model for en: p=4 and q=4
Best ARIMA Model for es: p=4 and q=5
Best ARIMA Model for fr: p=4 and q=5
Best ARIMA Model for ja: p=5 and q=5
Best ARIMA Model for nt: p=2 and q=5
Best ARIMA Model for ru: p=4 and q=4
Best ARIMA Model for zh: p=5 and q=5
```

In [56]:
```
for i,col in enumerate(df3.columns):
    model = sm.tsa.ARIMA(train[col], order=(best_params[i][0],1,best_params[i][1]))
    results = model.fit()
    predictions = results.forecast(50);
    print('Time series for: {}'.format(col))
    performance(test[col], predictions)
    # Plot predictions against known values
    ax = test[col].plot(legend=True,figsize=(12,4),title=title)
    predictions.plot(legend=True, color = 'orange')
    ax.autoscale(axis='x',tight=True)
    ax.set(xlabel=xlabel, ylabel=ylabel)
    plt.show()
```

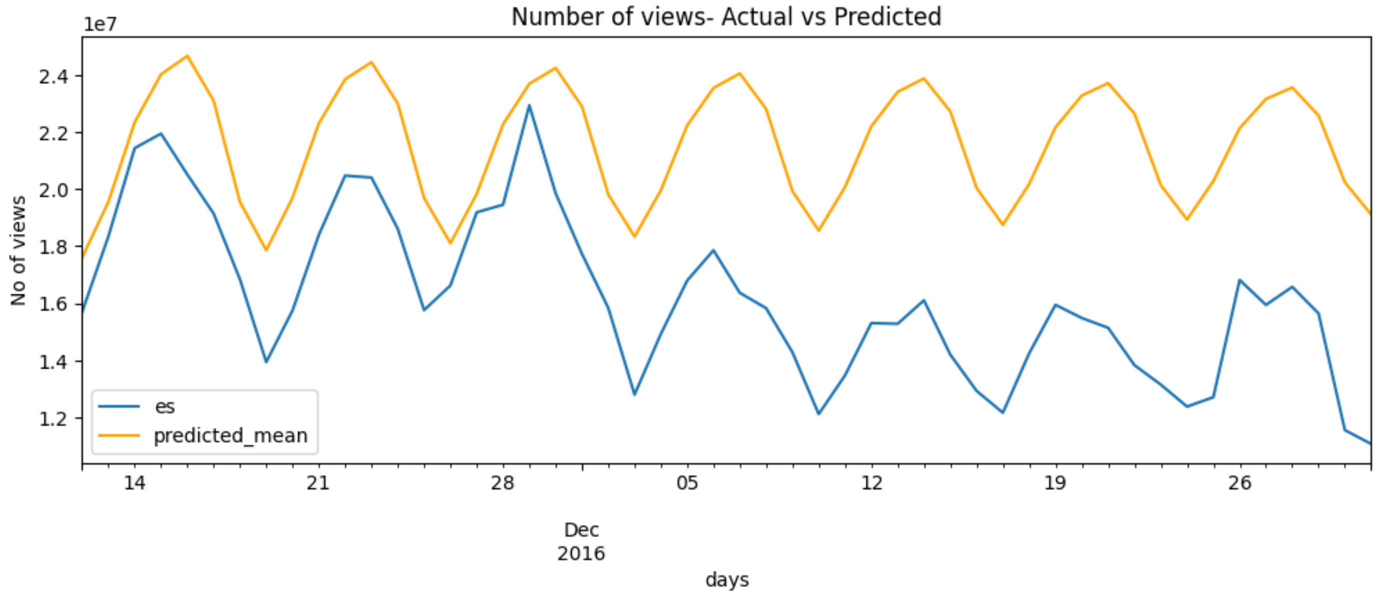
```
Time series for: de
MAE : 3246204.786
RMSE : 3483506.214
MAPE: 19.875
```



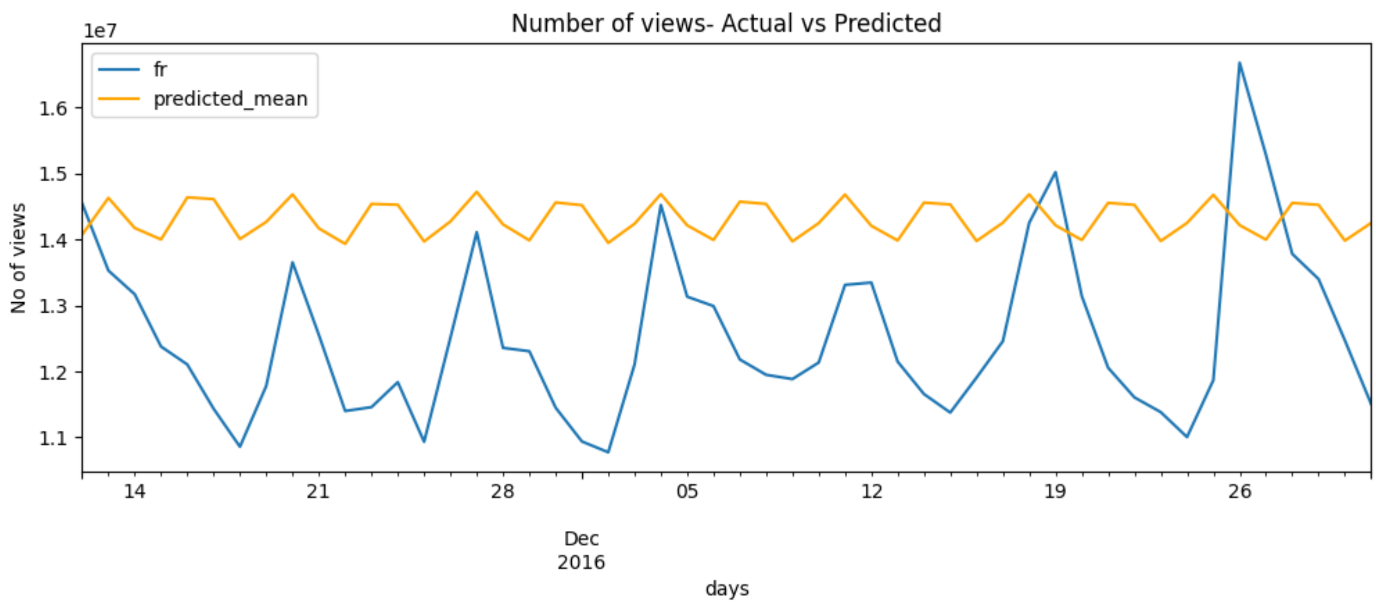

```
Time series for: en
MAE : 17330701.146
RMSE : 18918306.093
MAPE: 14.768
```



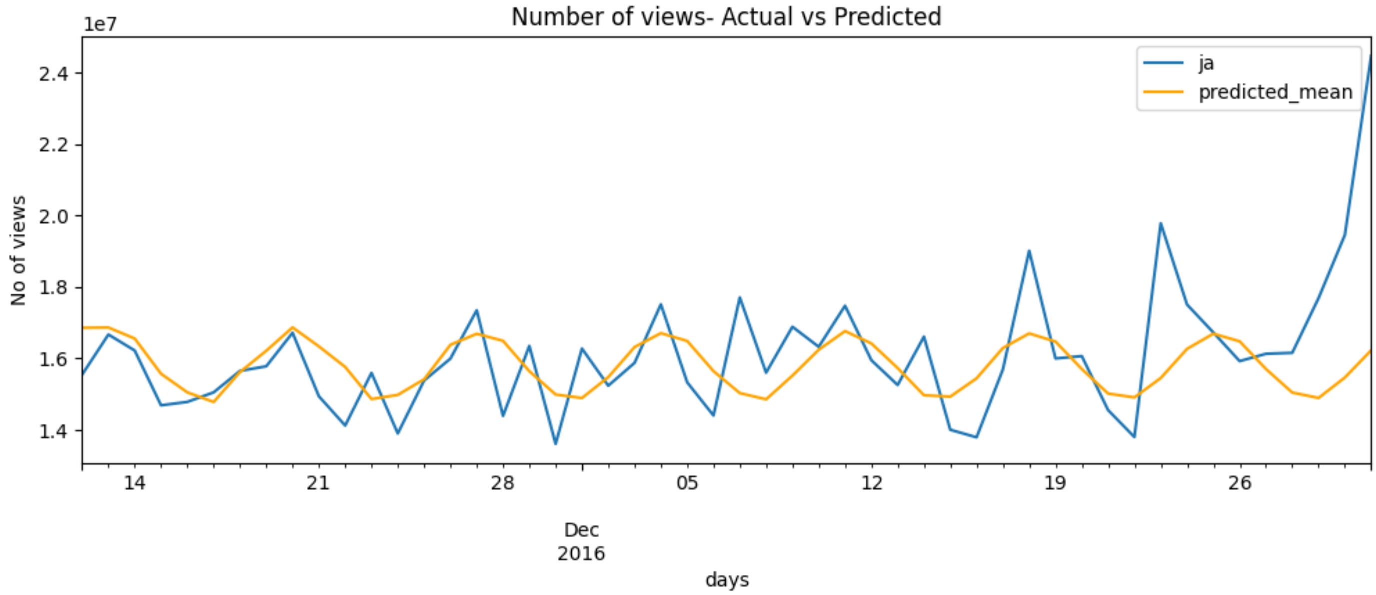

```
Time series for: es
MAE : 5303646.632
RMSE : 5771010.084
MAPE: 35.423
```



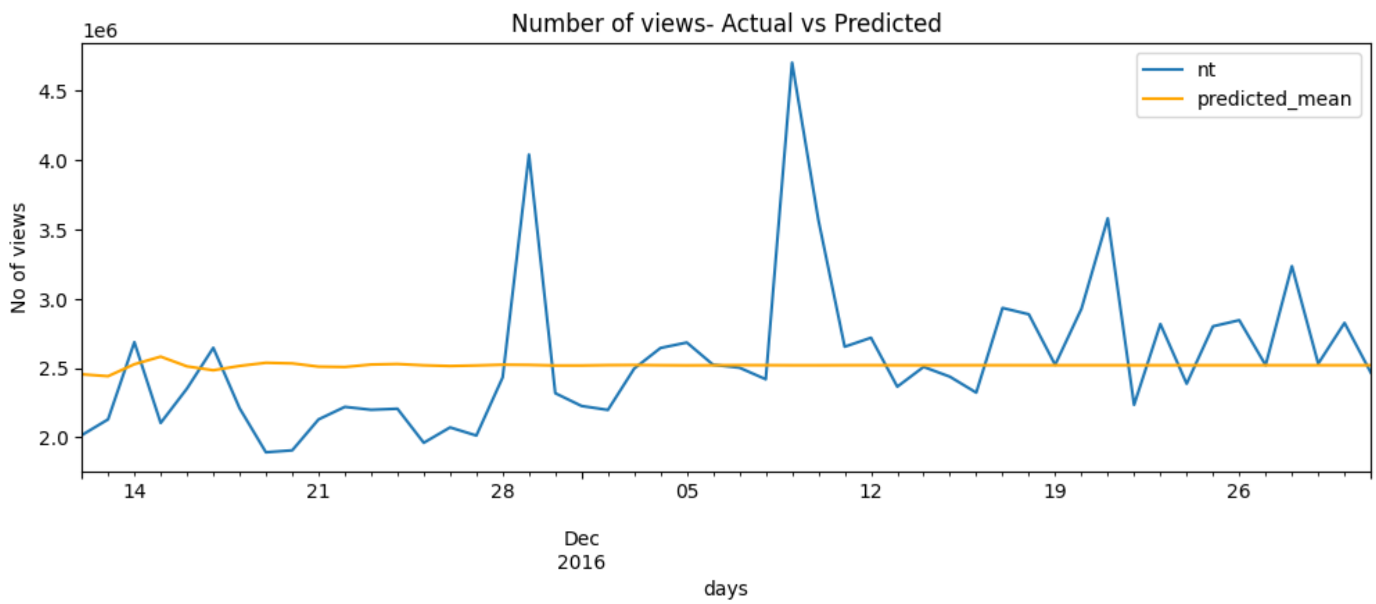
Time series for: fr
MAE : 1977939.42
RMSE : 2175677.965
MAPE: 16.487



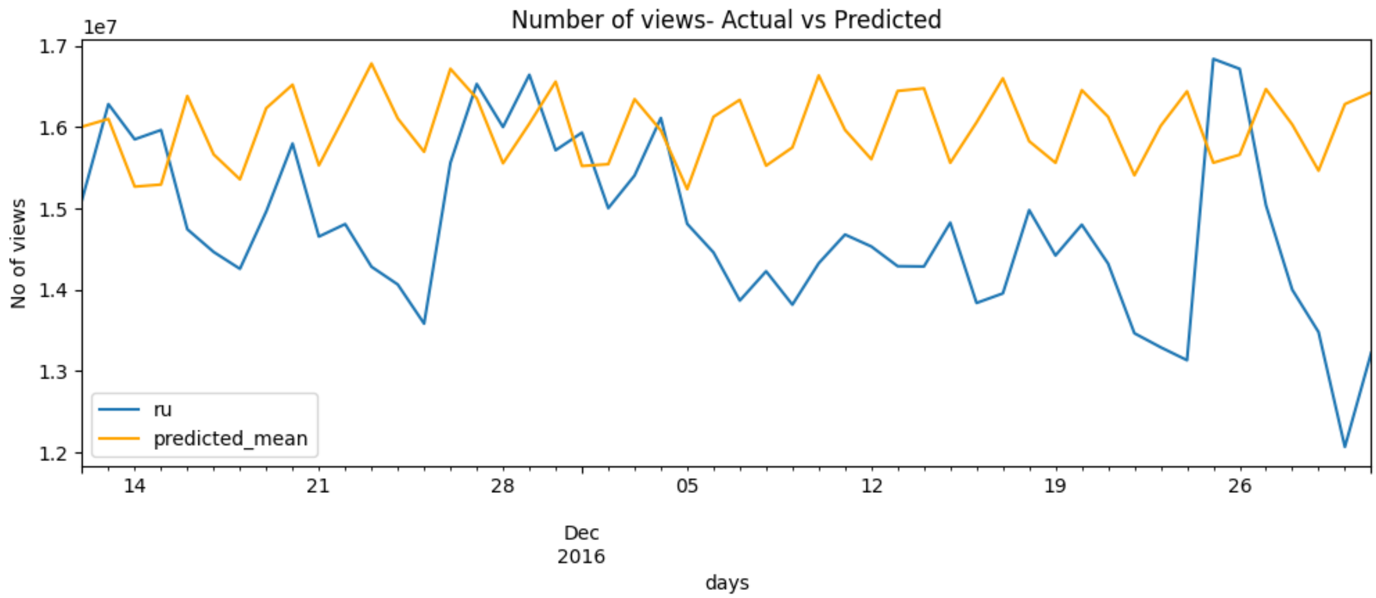
Time series for: ja
MAE : 1161126.502
RMSE : 1791398.916
MAPE: 6.79



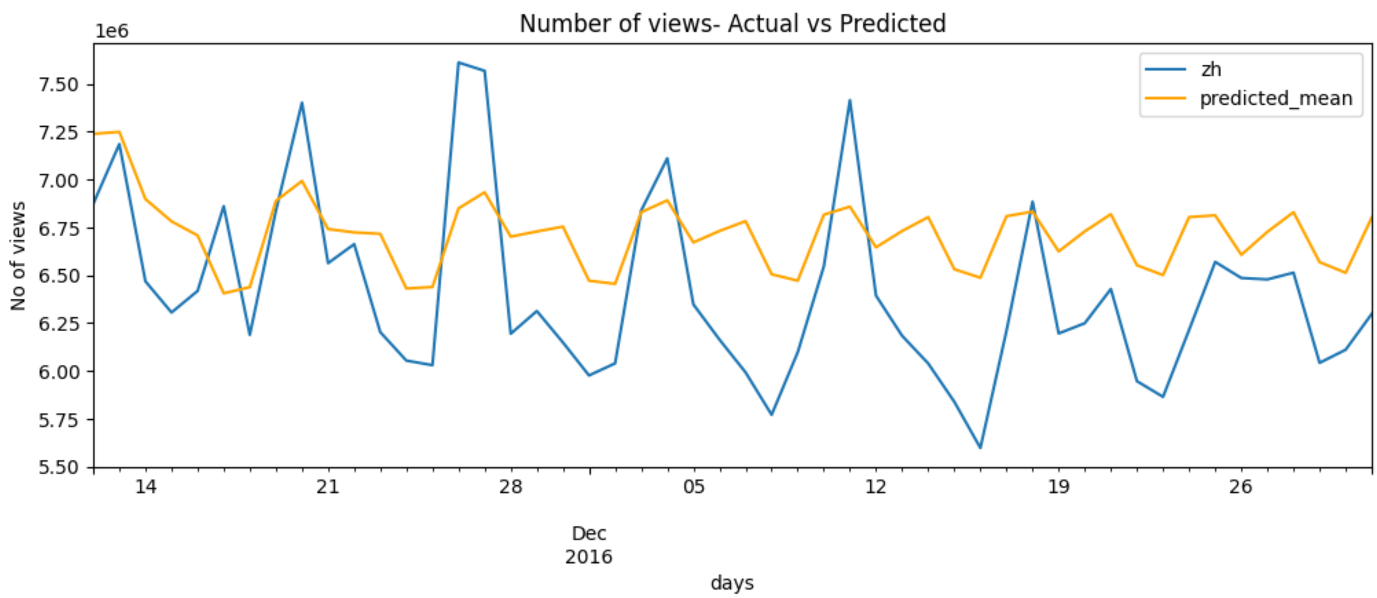

```
Time series for: nt
MAE : 353022.067
RMSE : 526766.791
MAPE: 13.046
```




```
Time series for: ru
MAE : 1468981.14
RMSE : 1709890.621
MAPE: 10.372
```




```
Time series for: zh
MAE : 425746.443
RMSE : 473758.775
MAPE: 6.766
```



Using ARIMA, we got MAPE 19.87% for German, 14.7% for English, 35.42% for Spanish, 16.5% for French, 6.79% for Japanese, 13% for 'nt', 10% for Russian, 6.77% for Mandarin.

Alternate method for grid-search is auto-ARIMA method.

In [66]:

```
import pmdarima as pm
for col in df2.columns:
    # Perform auto ARIMA model selection
    auto_model = pm.auto_arima(df2[col], seasonal=True, m=7)  # Set seasonal=True if your data h
    # Print the summary of the selected model
    print('for {} language: aic is {}'.format(col,auto_model.aic))
```

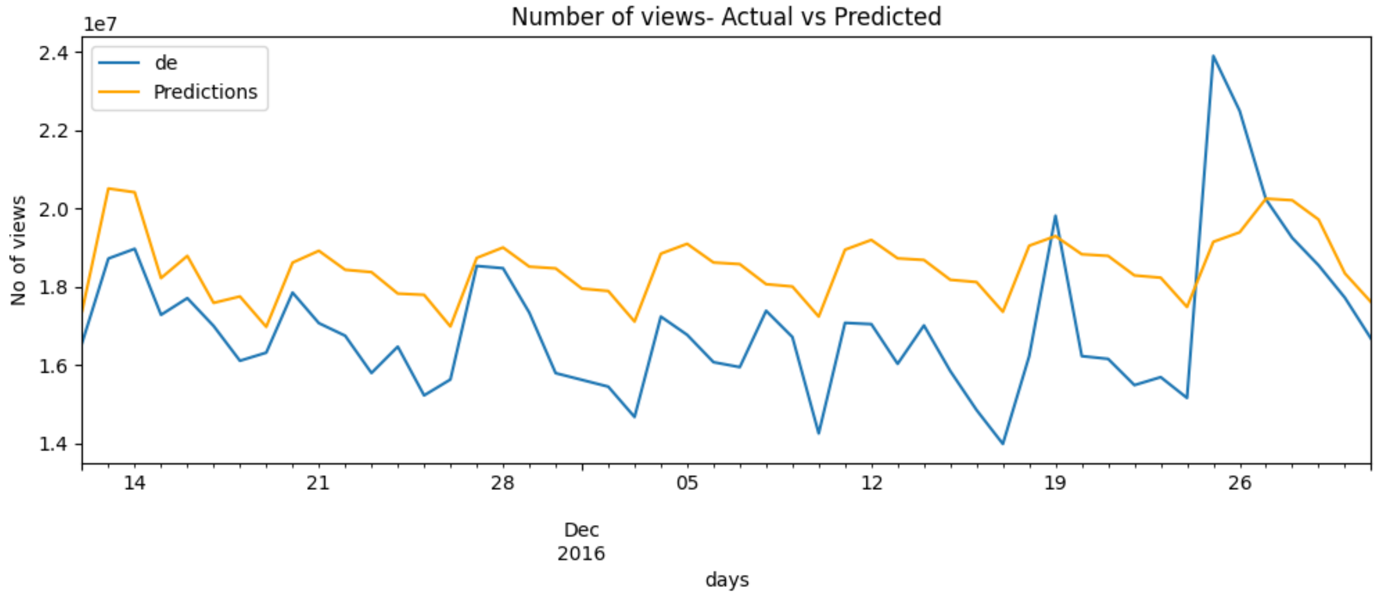
```
for de language: aic is <bound method ARIMA.aic of ARIMA(order=(2, 1, 2), scoring_args={}, seaso
nal_order=(1, 0, 1, 7),
      suppress_warnings=True)>
for en language: aic is <bound method ARIMA.aic of ARIMA(order=(1, 1, 1), scoring_args={}, seaso
nal_order=(2, 0, 1, 7),
      suppress_warnings=True)>
for es language: aic is <bound method ARIMA.aic of ARIMA(order=(1, 1, 2), scoring_args={}, seaso
nal_order=(2, 0, 1, 7),
      suppress_warnings=True)>
for fr language: aic is <bound method ARIMA.aic of ARIMA(order=(2, 1, 3), scoring_args={}, seaso
nal_order=(1, 0, 2, 7),
      suppress_warnings=True)>
for ja language: aic is <bound method ARIMA.aic of ARIMA(order=(3, 1, 5), scoring_args={}, seaso
nal_order=(0, 0, 0, 7),
      suppress_warnings=True, with_intercept=False)>
for nt language: aic is <bound method ARIMA.aic of ARIMA(order=(2, 1, 2), scoring_args={}, seaso
nal_order=(1, 0, 0, 7),
      suppress_warnings=True, with_intercept=False)>
for ru language: aic is <bound method ARIMA.aic of ARIMA(order=(0, 1, 2), scoring_args={}, seaso
nal_order=(1, 0, 1, 7),
      suppress_warnings=True)>
for zh language: aic is <bound method ARIMA.aic of ARIMA(order=(5, 1, 2), scoring_args={}, seaso
nal_order=(1, 0, 1, 7),
      suppress_warnings=True)>
```

SARIMAX

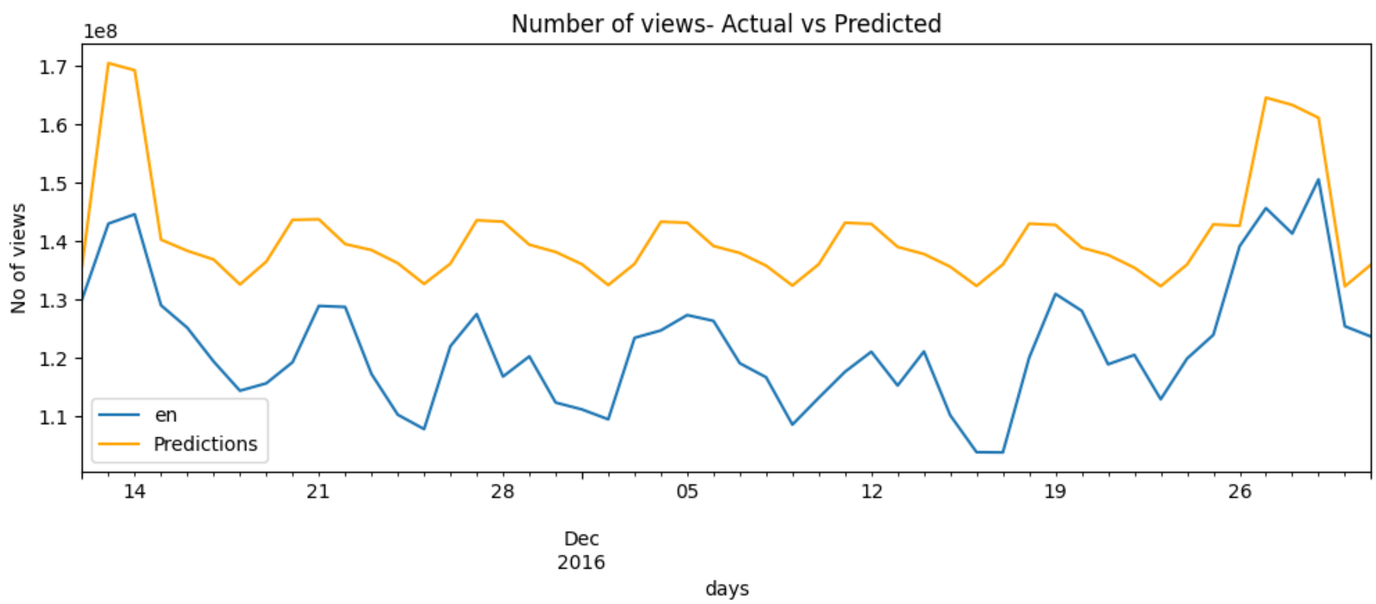
In [57]:

```python
from statsmodels.tsa.statespace.sarimax import SARIMAX
exog_forecast = test['exog']
for col in df3.columns:
    model = SARIMAX(train[col], exog=train['exog'], order=(best_params[i][0],1,best_params[i][1]
    results = model.fit(disp=False)
    predictions = results.predict(start=500, end=549, exog=exog_forecast).rename('Predictions')
    print('Time series for: {}'.format(col))
    print(results.aic, results.bic)
    performance(test[col], predictions)
    # Plot predictions against known values
    ax = test[col].plot(legend=True,figsize=(12,4),title=title)
    predictions.plot(legend=True, color = 'orange')
    ax.autoscale(axis='x',tight=True)
    ax.set(xlabel=xlabel, ylabel=ylabel)
    plt.show()
```

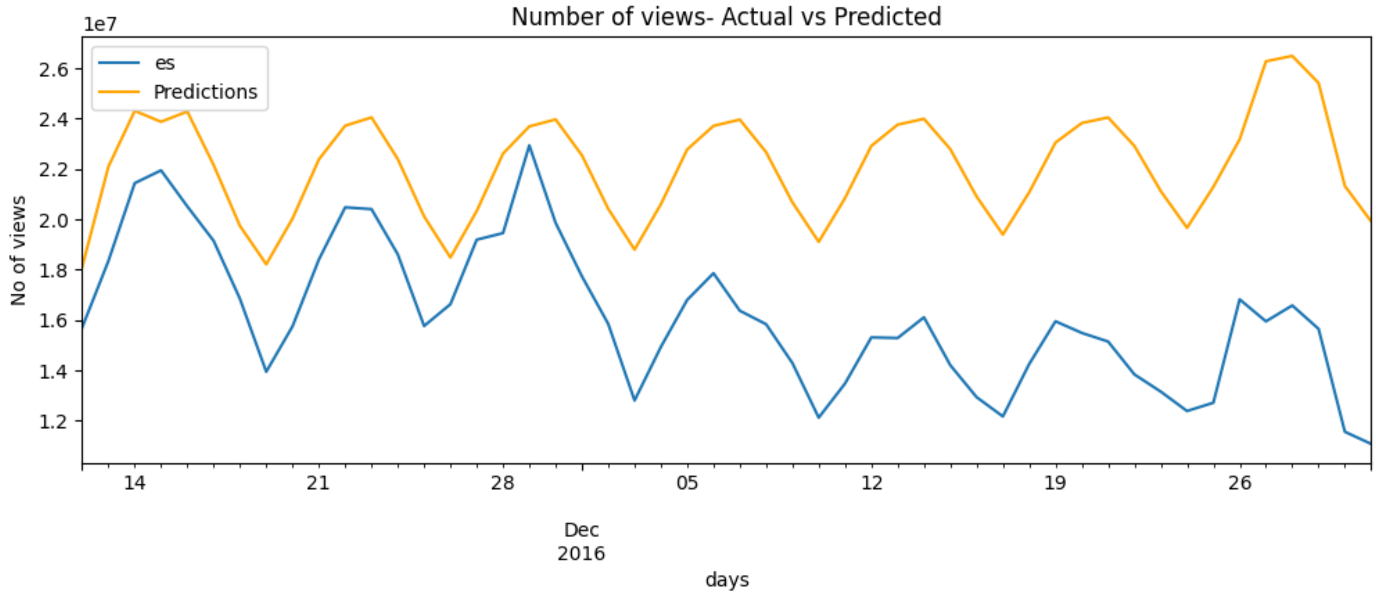
```
Time series for: de
15294.027339876164 15353.003825216685
MAE : 1843029.61
RMSE : 2083259.583
MAPE: 11.142
```



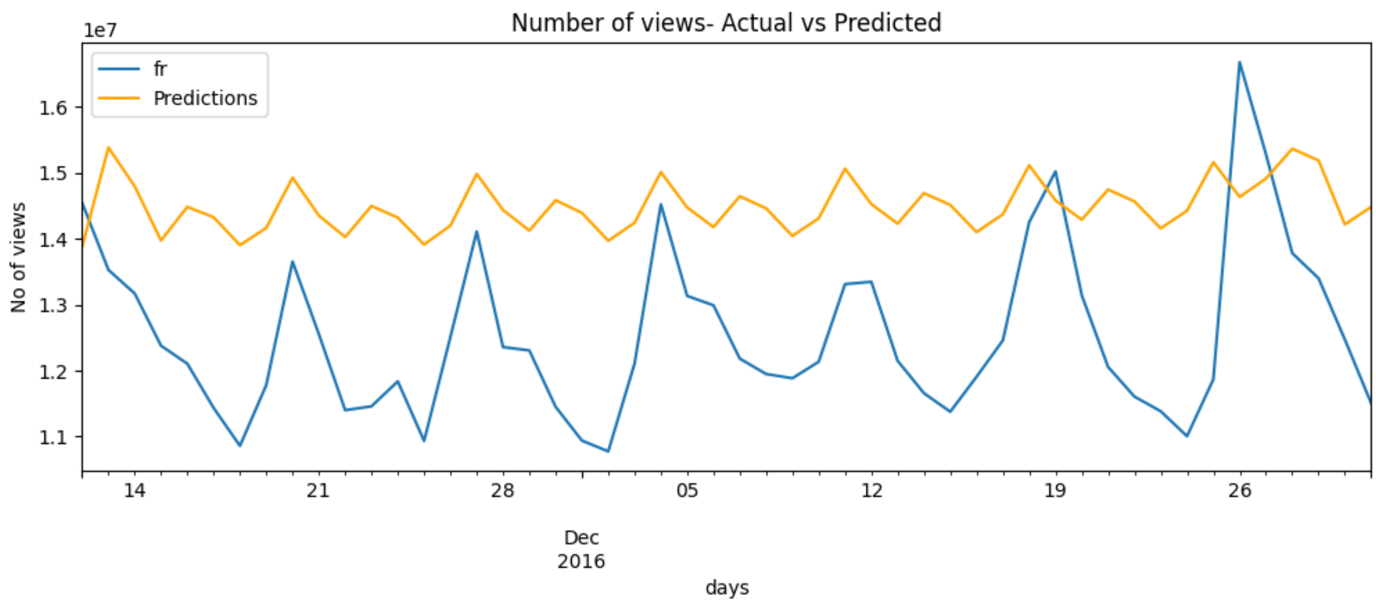

```
Time series for: en
17071.753318586456 17130.729803926977
MAE : 18728083.897
RMSE : 19740284.584
MAPE: 15.689
```



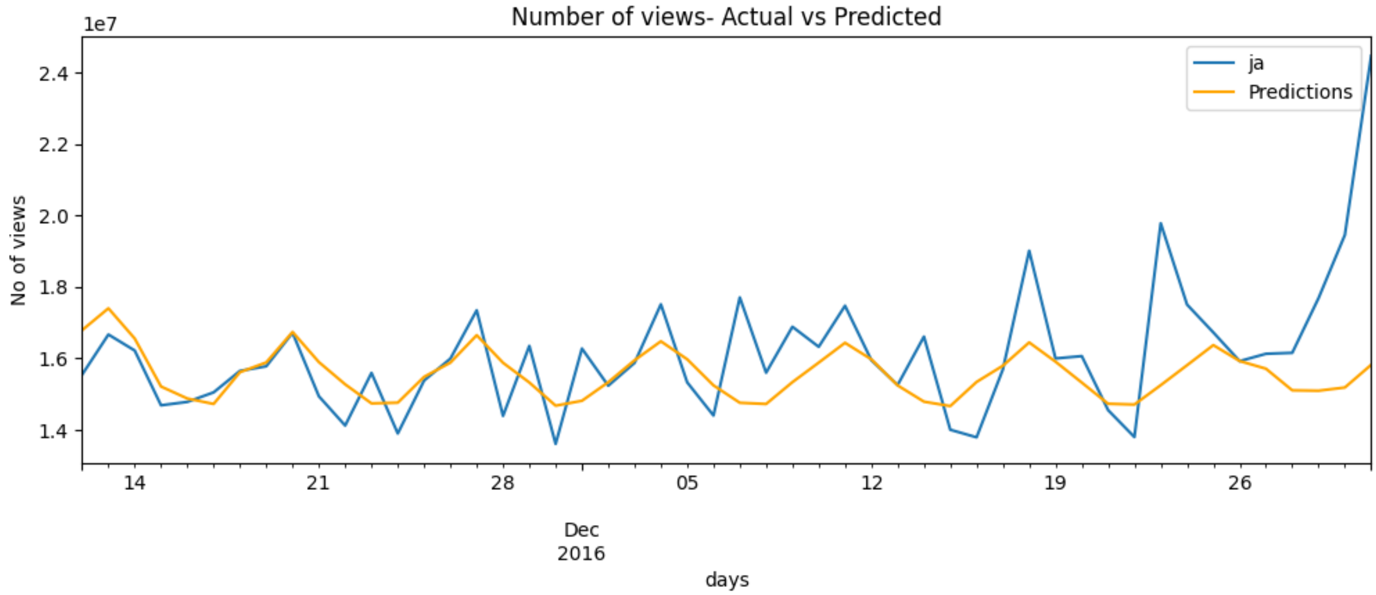

```
Time series for: es
15373.645199565859 15432.62168490638
MAE : 5878765.327
RMSE : 6399887.899
MAPE: 39.294
```



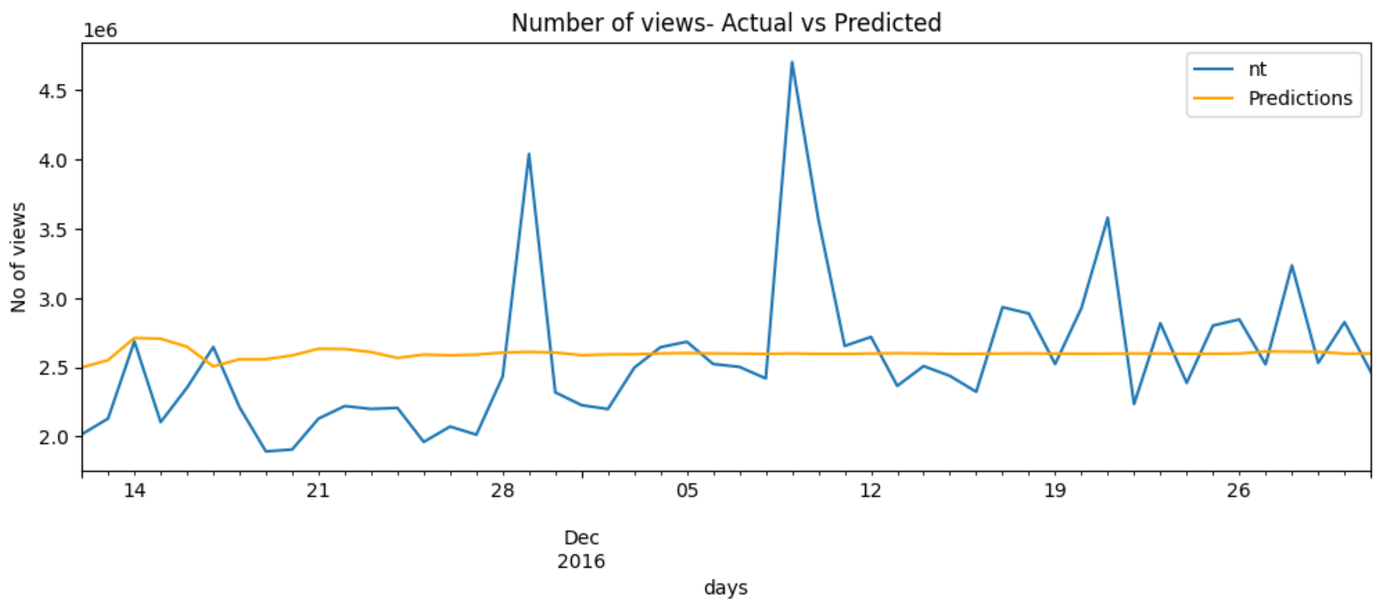

```
Time series for: fr
15126.095546498267 15185.072031838788
MAE : 2094349.037
RMSE : 2252579.185
MAPE: 17.405
```



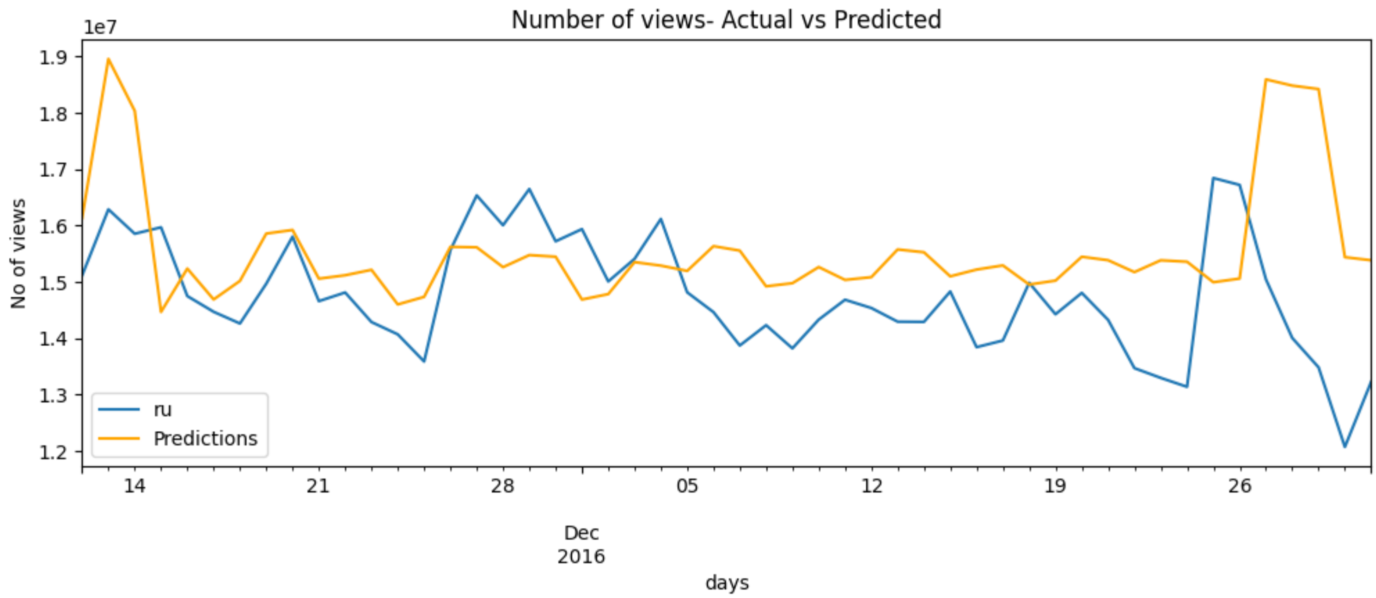

```
Time series for: ja
15619.10709532402 15678.083580664541
MAE : 1099424.635
RMSE : 1829910.421
MAPE: 6.286
```



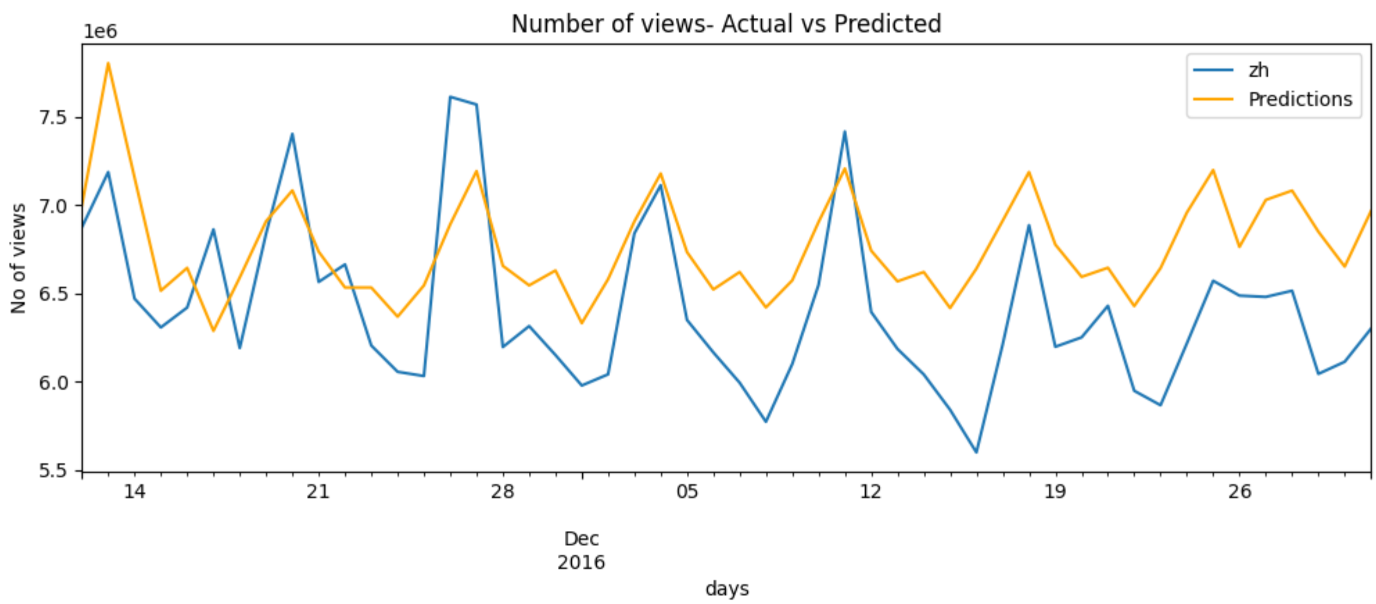

```
Time series for: nt
14136.538532624432 14195.515017964954
MAE : 374376.028
RMSE : 524741.304
MAPE: 14.29
```




```
Time series for: ru
15784.31714950137 15843.293634841892
MAE : 1228793.277
RMSE : 1627357.848
MAPE: 8.545
```




```
Time series for: zh
14389.772211154377 14448.748696494898
MAE : 442296.963
RMSE : 491662.712
MAPE: 7.03
```



Using SARIMAX forecasting the Mean Absolute Percentage Error (MAPE) for number of ad views was 11% on German websites, 15.7% for English, 39.3% for Spanish, 17.4% for French, 6.3% for Japanese, 14.3% for 'nt', 8.54% for Russian, 7% for Chinese websites.

Forecasting with Facebook prophet

In [62]:

```
df5 = df4.copy()
df5.reset_index(inplace=True)
df5.head()
```

Out[62]:

	date	de	en	es	fr	ja	nt	ru	zh	exog
0	2015-07-01	13299837.0	84712190.0	15278553.0	8458638.0	11863200.0	1451216.0	9463854.0	4144988.0	0
1	2015-07-02	13142154.0	84438545.0	14601013.0	8512952.0	13620792.0	1499552.0	9627643.0	4151189.0	0
2	2015-07-03	12615201.0	80167728.0	13427632.0	8186030.0	12305383.0	1415102.0	8923463.0	4123659.0	0
3	2015-07-04	11573244.0	83463204.0	12606538.0	8749842.0	15456239.0	1207208.0	8393214.0	4163448.0	0
4	2015-07-05	13470112.0	86198637.0	13710356.0	8590493.0	14827204.0	1318756.0	8938528.0	4441286.0	0

In [64]:

```
langs=df5.columns[1:-1]
langs
```

Out[64]:

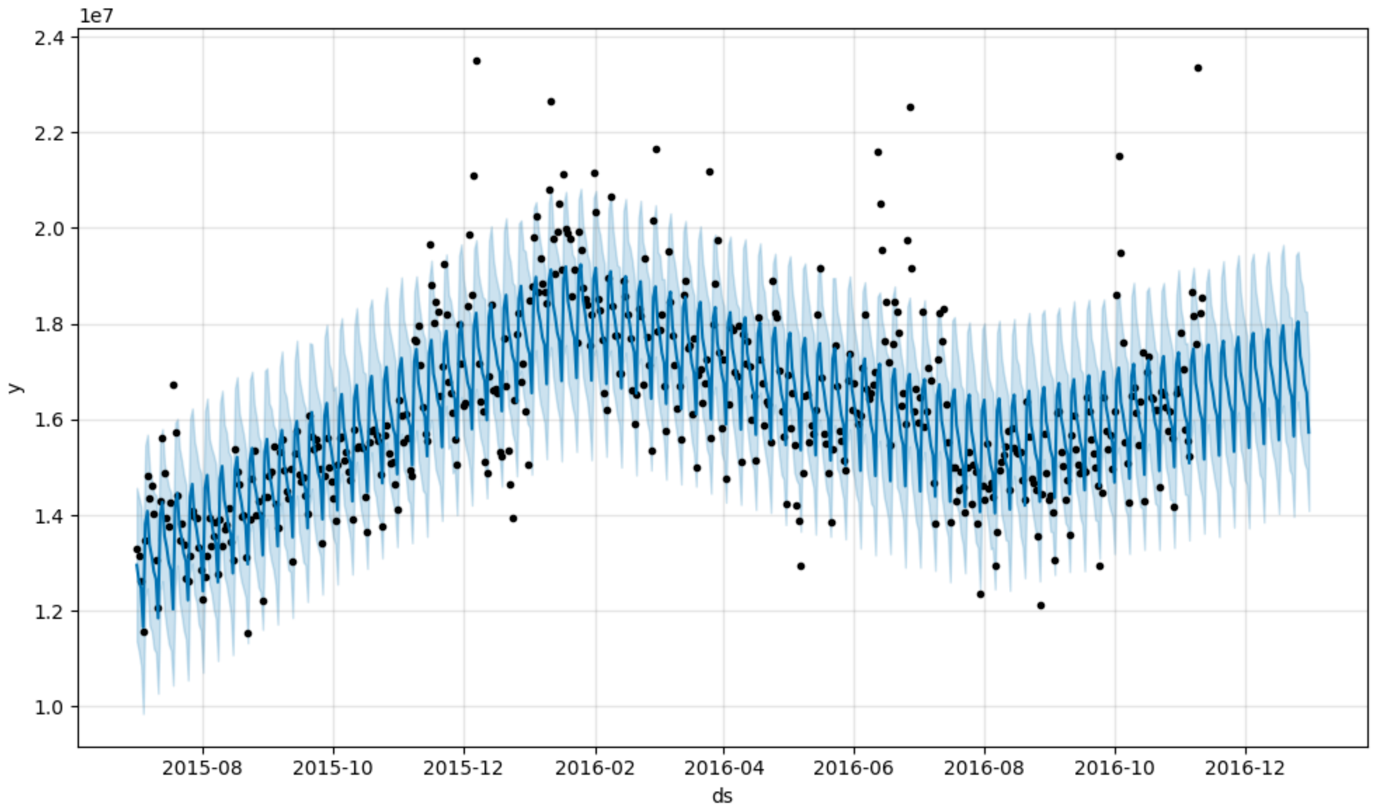
```
Index(['de', 'en', 'es', 'fr', 'ja', 'nt', 'ru', 'zh'], dtype='object')
```

In [73]:

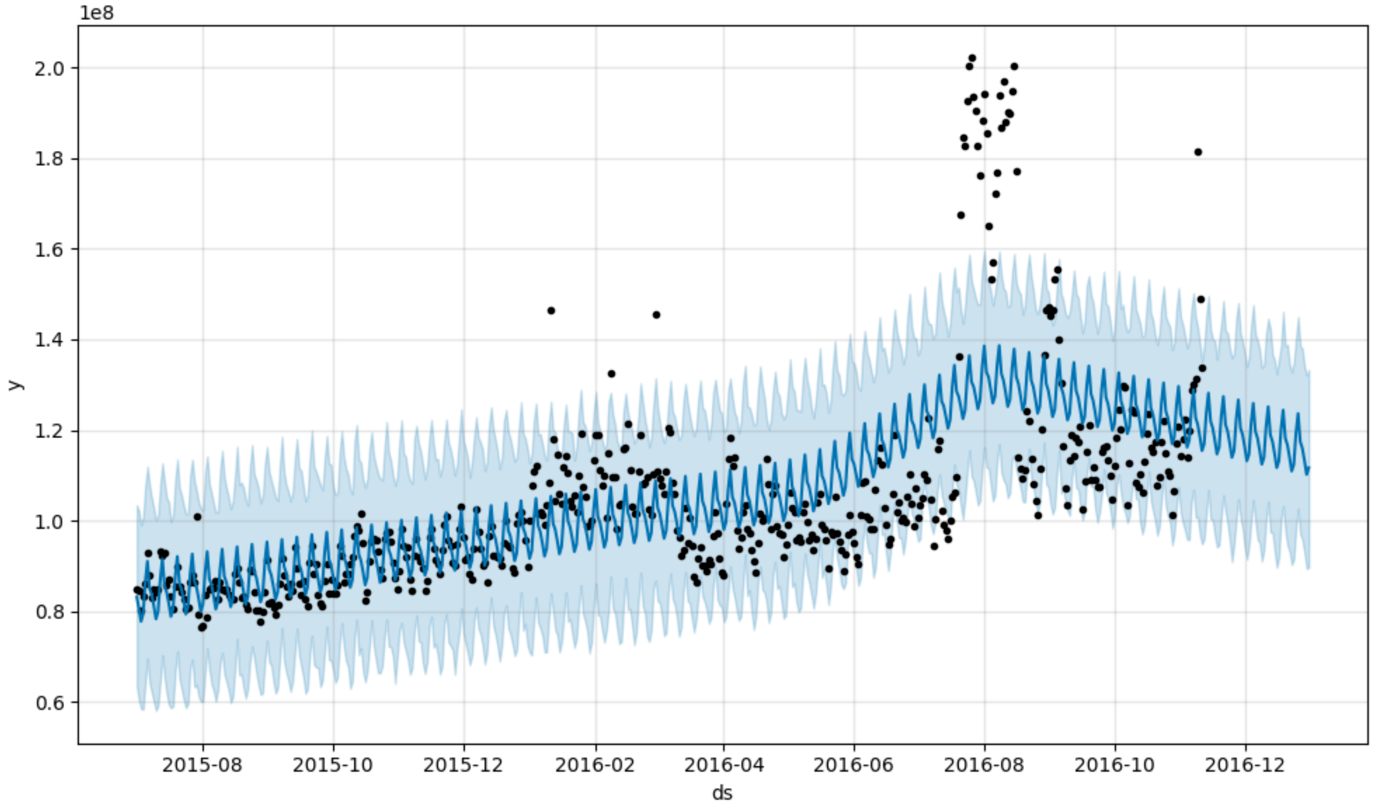
```
from prophet import Prophet

for col in langs:
    plt.figure(figsize=(20,5))
    m = Prophet()
    dummy_df = df5[['date',col]]
    dummy_df.rename(columns= {'date':'ds', col:'y'}, inplace = True)
    m.fit(dummy_df[['ds', 'y']][:-50])
    future = m.make_future_dataframe(periods=50, freq='D')
    forecast = m.predict(future)
    print('For language: {}'.format(col))
    performance(df4[col][-50:],forecast['yhat'][-50:])
    m.plot(forecast);
    # f.yhat.plot()
    # f.yhat_lower.plot()
    # f.yhat_upper.plot()
    # f.trend.plot()
```

```
23:38:56 - cmdstanpy - INFO - Chain [1] start processing
23:38:56 - cmdstanpy - INFO - Chain [1] done processing
For language: de
MAE : 1118765.962
RMSE : 1512660.469
MAPE: 6.295
23:38:56 - cmdstanpy - INFO - Chain [1] start processing
23:38:56 - cmdstanpy - INFO - Chain [1] done processing
For language: en
MAE : 7279217.898
RMSE : 10373993.398
MAPE: 5.676
23:38:57 - cmdstanpy - INFO - Chain [1] start processing
23:38:57 - cmdstanpy - INFO - Chain [1] done processing
For language: es
MAE : 4112913.762
RMSE : 4756848.243
MAPE: 27.887
23:38:58 - cmdstanpy - INFO - Chain [1] start processing
23:38:58 - cmdstanpy - INFO - Chain [1] done processing
```

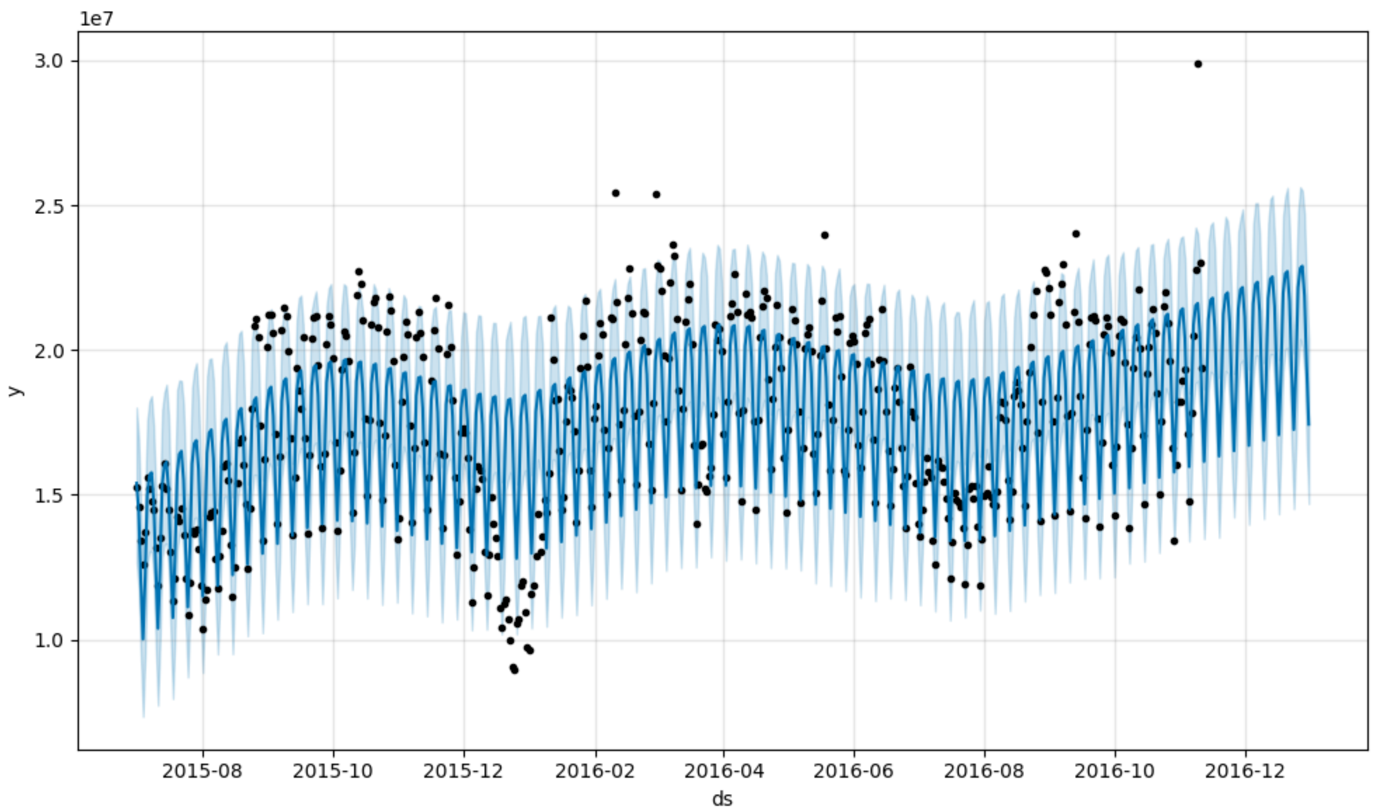
```
For language: fr
MAE : 820267.424
RMSE : 1079073.837
MAPE: 6.45
23:38:58 - cmdstanpy - INFO - Chain [1] start processing
23:38:58 - cmdstanpy - INFO - Chain [1] done processing
For language: ja
MAE : 1129359.027
RMSE : 1760466.83
MAPE: 6.625
23:38:59 - cmdstanpy - INFO - Chain [1] start processing
23:38:59 - cmdstanpy - INFO - Chain [1] done processing
For language: nt
MAE : 447514.024
RMSE : 560462.953
MAPE: 17.996
23:39:00 - cmdstanpy - INFO - Chain [1] start processing
23:39:00 - cmdstanpy - INFO - Chain [1] done processing
For language: ru
MAE : 637303.788
RMSE : 823136.102
MAPE: 4.21
23:39:00 - cmdstanpy - INFO - Chain [1] start processing
23:39:00 - cmdstanpy - INFO - Chain [1] done processing
For language: zh
MAE : 378493.072
RMSE : 441781.684
MAPE: 6.051
<Figure size 2000x500 with 0 Axes>
```



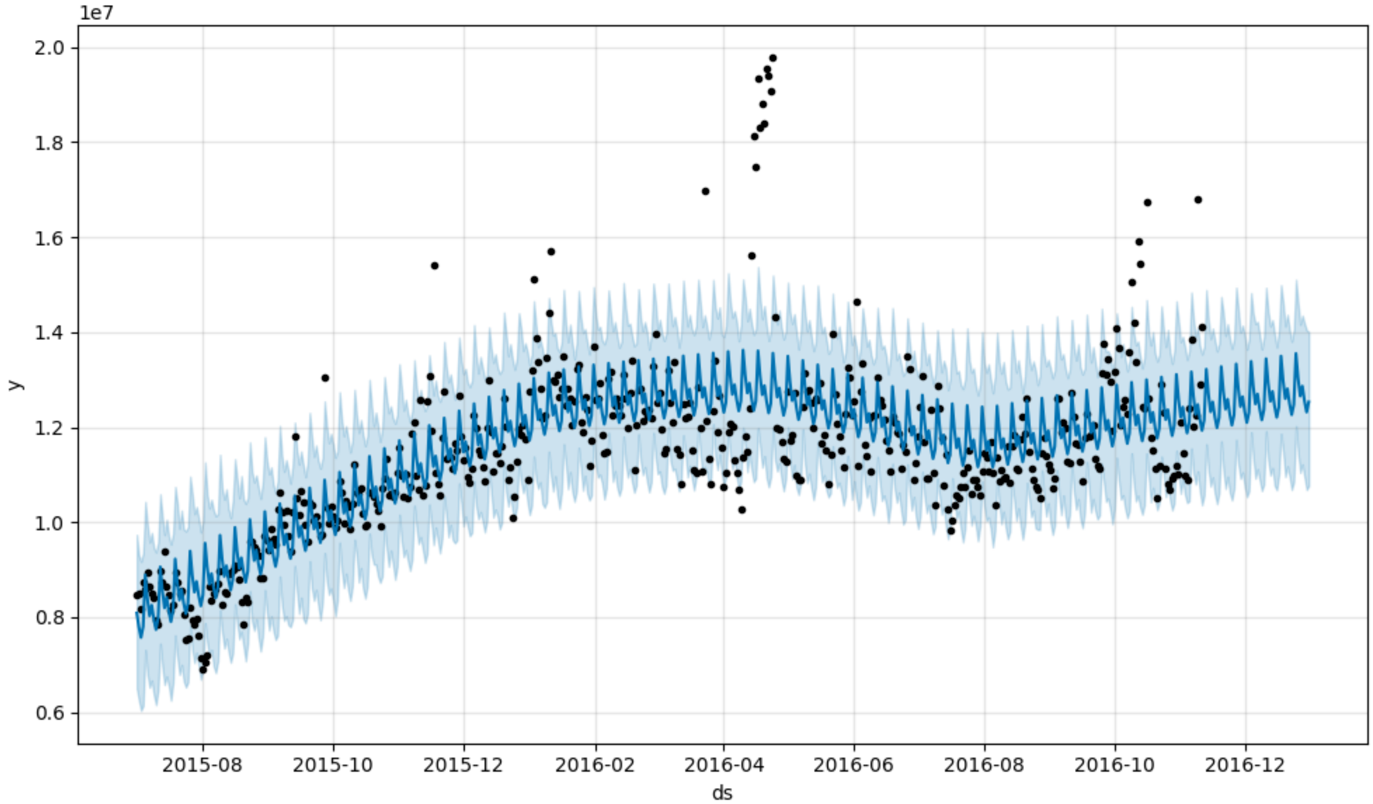

```
<Figure size 2000x500 with 0 Axes>
```



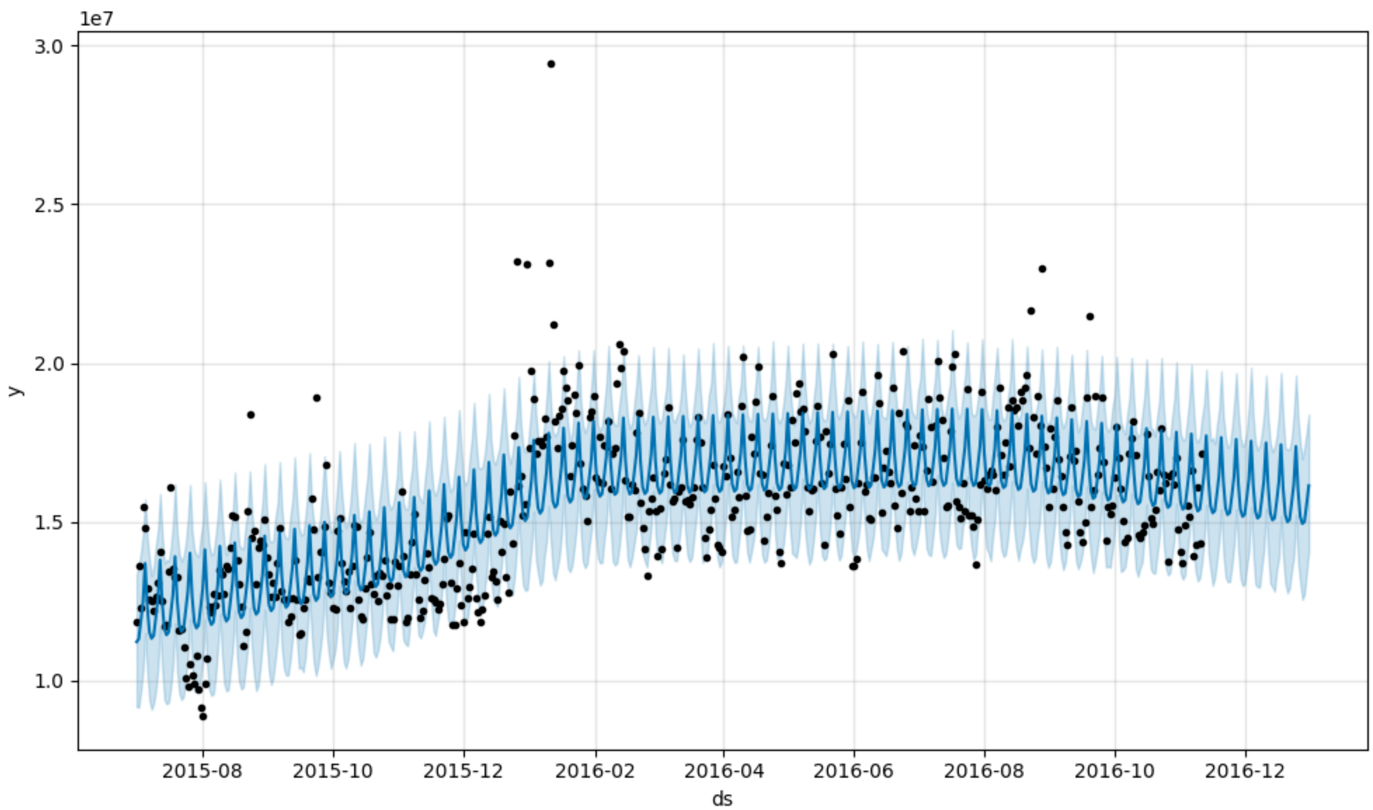
<Figure size 2000x500 with 0 Axes>



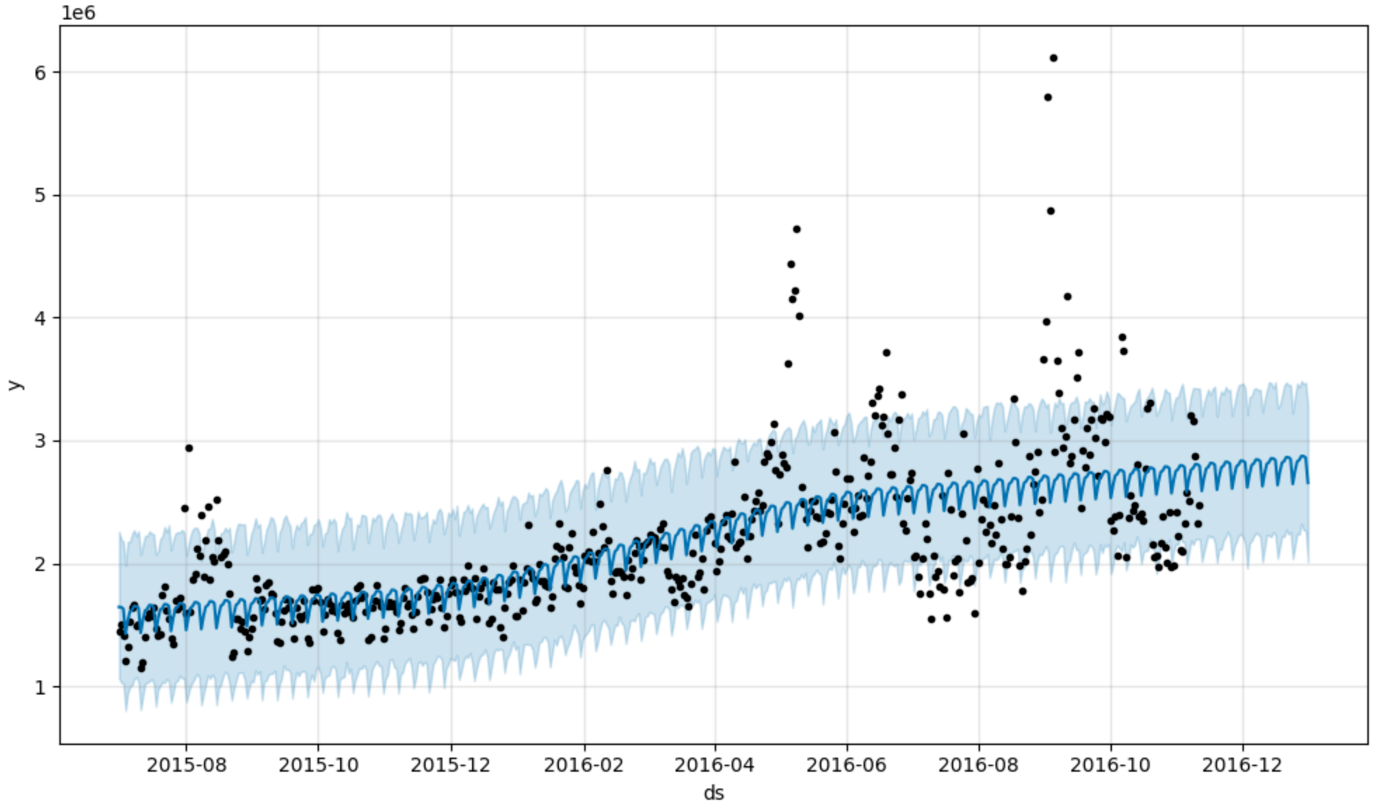
<Figure size 2000x500 with 0 Axes>



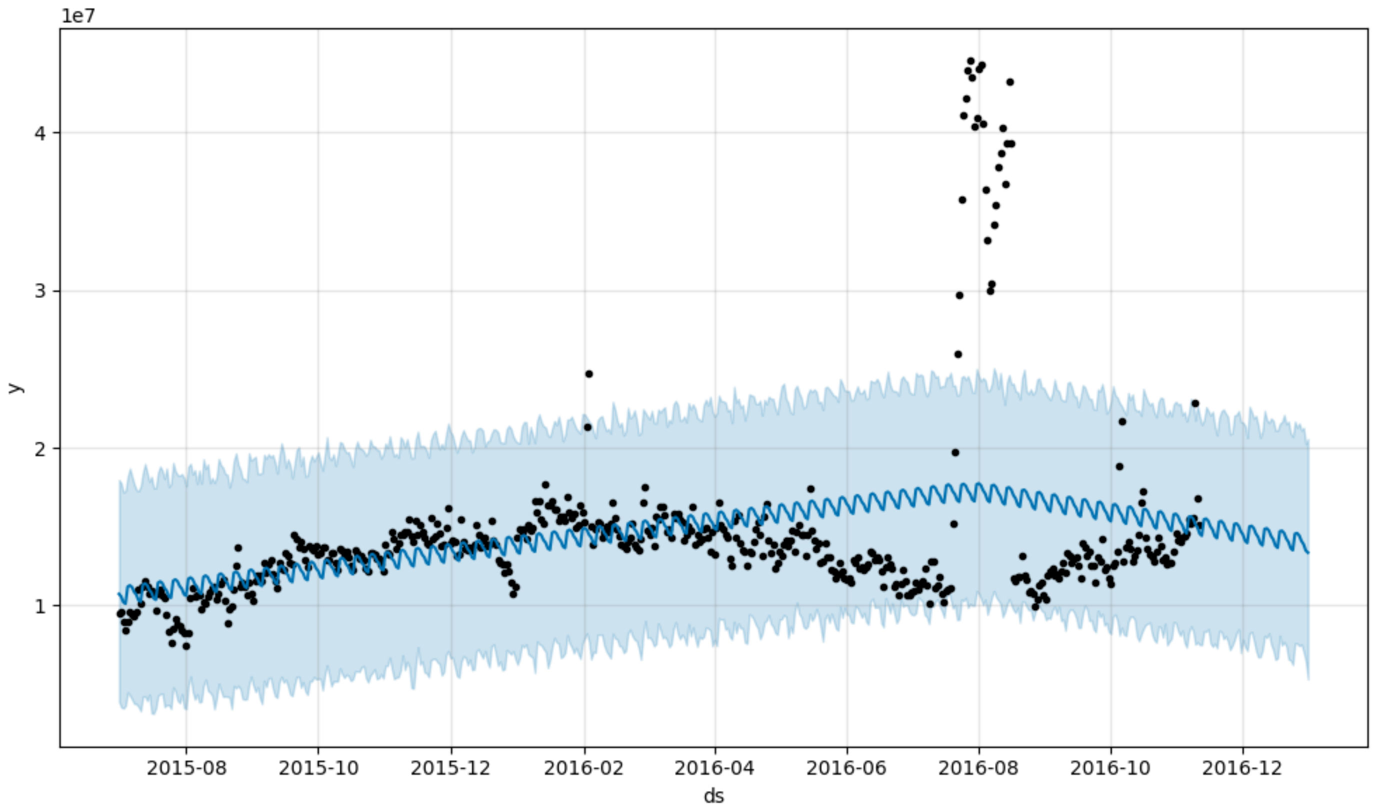
<Figure size 2000x500 with 0 Axes>



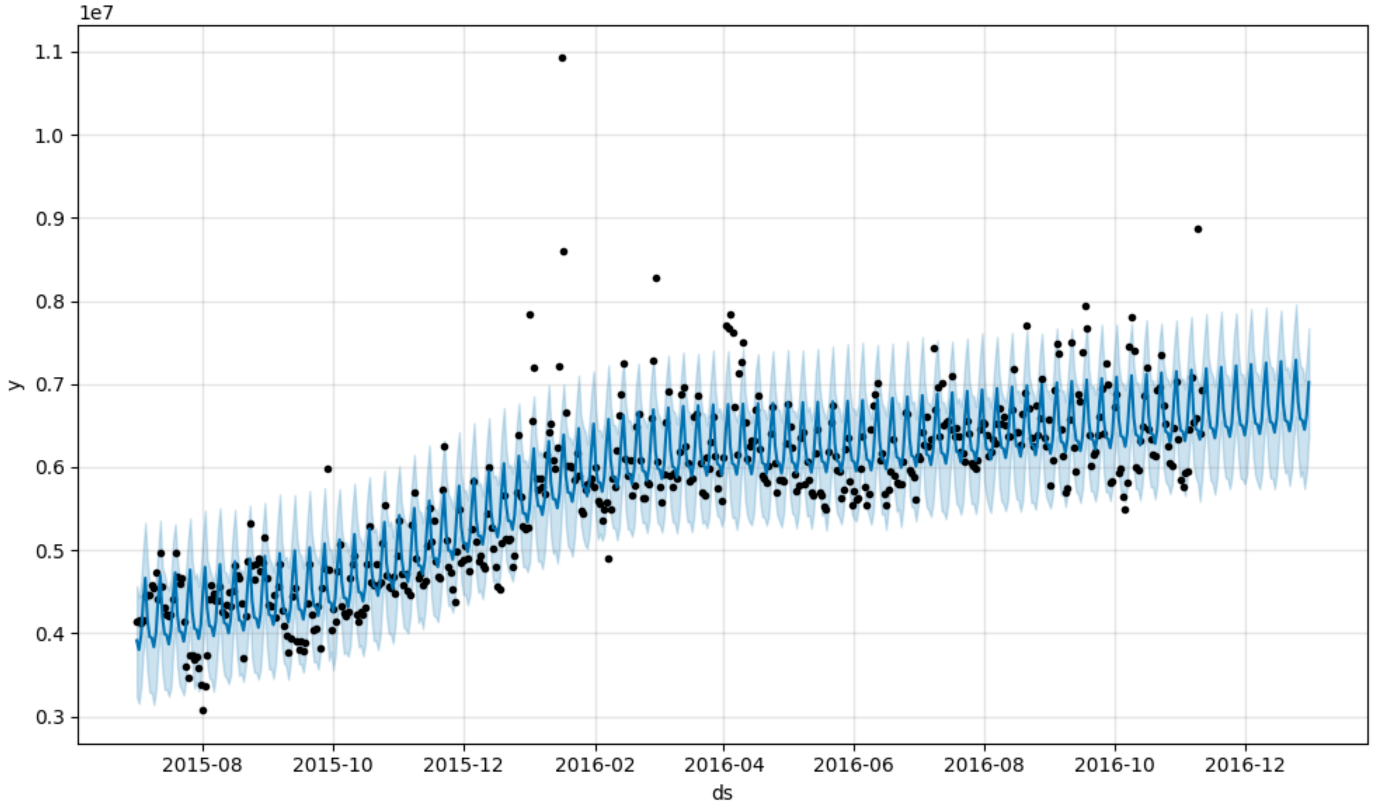
<Figure size 2000x500 with 0 Axes>



<Figure size 2000x500 with 0 Axes>



<Figure size 2000x500 with 0 Axes>



Using Prophet, MAPE for language: German 6.29%, English 5.67%, Spanish 27.89%, French 6.45%, Japanese 6.63%, 'nt' 18%, Russian 4.21%, and Chinese 6.05%.

Inferences:

- What other methods other than grid search would be suitable to get the model for all languages? Using auto-arima function.
- Compare the number of views in different languages.

1. English language webpages have much higher number of views compared to any other language websites.
2. While some time series have varying mean such as Spanish (es) and Japanese(ja), English time series is clearly having an increasing trend.
3. There are some events of sudden rises in number of views on certain days in English(en), Russian (ru), etc. that coincide with the exogenous variable.

Using ARIMA, we got MAPE in forecasting number of views as 19.87% for German websites, 14.7% for English, 35.42% for Spanish, 16.5% for French, 6.79% for Japanese, 13% for 'nt', 10% for Russian, 6.77% for Chinese websites.

Using SARIMAX forecasting the Mean Absolute Percentage Error (MAPE) for number of ad views was 11% on German websites, 15.7% for English, 39.3% for Spanish, 17.4% for French, 6.3% for Japanese, 14.3% for 'nt', 8.54% for Russian, 7% for Chinese websites.

Using Prophet, MAPE for language: German 6.29%, English 5.67%, Spanish 27.89%, French 6.45%, Japanese 6.63%, 'nt' 18%, Russian 4.21%, and Chinese 6.05%.

In [ ]: