AdEase Case Study

Introduction

- AdEase is an ads and marketing-based company helping businesses elicit maximum clicks @ minimum cost.
- AdEase is an ad infrastructure to help businesses promote themselves easily, effectively, and economically
- AdEase is trying to understand the per page view report for different wikipedia

pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients.

- By leveraging data science and time series, Ad Ease can forecast page visits for different languages.

What is expected?

- You are working in the Data Science team of Ad ease trying to understand the per page view report for different wikipedia pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients. You are provided with the data of 145k wikipedia pages and daily view count for each of them. Your clients belong to different regions and need data on how their ads will perform on pages in different languages.

1. Data

The analysis was done on the data located at - https://drive.google.com/drive/folders/1mdgQscjqnCtdg7LGItomyK0abN6lcHBb

2. Libraries

Below are the libraries required

In [1]:
```
# libraries to analyze data
import numpy as np
import pandas as pd
```

```python
# libraries to visualize data
import matplotlib.pyplot as plt
import seaborn as sns

import re

import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

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


from statsmodels.tsa.arima.model import ARIMA
```

3. Data Loading

Loading the data into Pandas dataframe for easily handling of data

In [2]:

```python
# read the file into a pandas dataframe
df = pd.read_csv('train_1.csv')
# look at the datatypes of the columns
print('*************************************************')
print(df.info())
print('*************************************************\n')
print('*************************************************')
print(f'Shape of the dataset is {df.shape}')
print('*************************************************\n')
print('*************************************************')
print(f'Number of nan/null values in each column: \n{df.isna().sum()}')
print('*************************************************\n')
print('*************************************************')
print(f'Number of unique values in each column: \n{df.nunique()}')
print('*************************************************\n')
print('*************************************************')
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
```

```
************************************************
<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
None
************************************************

************************************************
Shape of the dataset is (145063, 551)
************************************************

************************************************
Number of nan/null values in each column:
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
************************************************

************************************************
Number of unique values in each column:
Page          145063
2015-07-01      6898
2015-07-02      6823
2015-07-03      6707
2015-07-04      6995
               ...
2016-12-27      8938
2016-12-28      8819
2016-12-29      8761
2016-12-30      8733
2016-12-31      8826
Length: 551, dtype: int64
************************************************

************************************************
Duplicate entries:
False    145063
Name: count, dtype: int64
```

In [3]:

```
# look at the top 20 rows
df.head(5)
```

Out[3]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 551 columns

In [4]:
```
df.describe()
```

Out[4]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-(
count	1.243230e+05	1.242470e+05	1.245190e+05	1.244090e+05	1.244040e+05	1.245800e+(
mean	1.195857e+03	1.204004e+03	1.133676e+03	1.170437e+03	1.217769e+03	1.290273e+(
std	7.275352e+04	7.421515e+04	6.961022e+04	7.257351e+04	7.379612e+04	8.054448e+(
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+(
25%	1.300000e+01	1.300000e+01	1.200000e+01	1.300000e+01	1.400000e+01	1.100000e+(
50%	1.090000e+02	1.080000e+02	1.050000e+02	1.050000e+02	1.130000e+02	1.130000e+(
75%	5.240000e+02	5.190000e+02	5.040000e+02	4.870000e+02	5.400000e+02	5.550000e+(
max	2.038124e+07	2.075219e+07	1.957397e+07	2.043964e+07	2.077211e+07	2.254467e+(

8 rows × 550 columns

In [5]:
```
df.describe(include='object')
```

Out[5]:

	Page
count	145063
unique	145063
top	2NE1_zh.wikipedia.org_all-access_spider
freq	1

Insight

- There are **145063** entries with 551 columns, i.e. 145063 wikipedia pages with views for 550 days
- There are null/missing values in each of the dates
- There are no **duplicates**
- There are **145063** unique wikipedia pages

In [6]:
```
# read the file containing flag for each date indicating if those dates had a campa
exog_en = pd.read_csv('Exog_Campaign_eng')
# look at the datatypes of the columns
print('**************************************************')
print(exog_en.info())
print('**************************************************\n')
print('**************************************************')
print(f'Shape of the dataset is {exog_en.shape}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of nan/null values in each column: \n{exog_en.isna().sum()}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of unique values in each column: \n{exog_en.nunique()}')
print('**************************************************\n')
print('**************************************************')
print(f'Duplicate entries: \n{exog_en.duplicated().value_counts()}')
```

```
************************************************
<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
None
************************************************

************************************************
Shape of the dataset is (550, 1)
************************************************

************************************************
Number of nan/null values in each column:
Exog    0
dtype: int64
************************************************

************************************************
Number of unique values in each column:
Exog    2
dtype: int64
************************************************

************************************************
Duplicate entries:
True     548
False      2
Name: count, dtype: int64
```

In [7]:

```
exog_en.head()
```

Out[7]:

	Exog
0	0
1	0
2	0
3	0
4	0

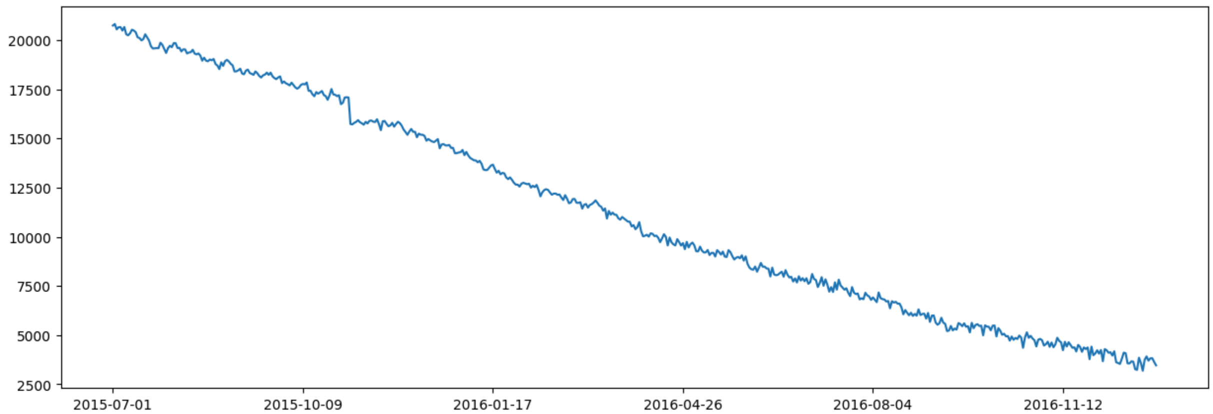
Insight

- There are **550** entries corresponding to 550 days in the previous dataset
- There are **no** null/missing values
- There are **2** unique values - 1 ans 0

4. Exploratory Data Analysis

4.1. Analysing date columns

In [8]:
```
date_columns = df.columns[1:]
df[date_columns].isna().sum().plot(figsize=(15,5))
plt.show()
```



Insight

- It can be observed that the null values keep decreasing with dates, indicating that there were no views for these dates
- We can infer that the webpages which were lauched recently will not have view data prior to launch and hence can be filled with 0

In [9]:
```
df[date_columns] = df.loc[:,date_columns].fillna(0)
```

In [10]:
```
df.isna().sum()
```

Out[10]:

	0
Page	0
2015-07-01	0
2015-07-02	0
2015-07-03	0
2015-07-04	0
...	...
2016-12-27	0
2016-12-28	0
2016-12-29	0
2016-12-30	0
2016-12-31	0

551 rows × 1 columns

dtype: int64

4.2. Extracting information from Page column

In [11]:

```
df['Page'].sample(10)
```

Out[11]:

	Page
72525	César_Gaviria_es.wikipedia.org_desktop_all-agents
122381	マギ_(漫画)_ja.wikipedia.org_all-access_all-agents
108907	超時空要塞Δ_zh.wikipedia.org_mobile-web_all-agents
52610	Paul_Touvier_fr.wikipedia.org_mobile-web_all-a...
42899	Talk:Wikimedia_Discovery_www.mediawiki.org_des...
119774	夏目漱石_ja.wikipedia.org_all-access_all-agents
18841	Латинский_язык_ru.wikipedia.org_mobile-web_all...
55049	Salvador_Dalí_fr.wikipedia.org_mobile-web_all-...
117187	Schtonk!_de.wikipedia.org_mobile-web_all-agents
88608	デッドプール_ja.wikipedia.org_desktop_all-agents

dtype: object

The page name contains data in the below format:

SPECIFIC NAME _ LANGUAGE.wikipedia.org _ ACCESS TYPE _ ACCESS ORIGIN

having information about page name, thn domain, device type used to access t e page, aso the request origin(spider or browser age 2.

4.2.1. Extracting name

In [12]:
```
def extract_name(page):
    pattern = r'(.{0,})_(.{2}).wikipedia.org_'
    result = re.findall(pattern, page)
    if len(result) == 1:
        return result[0][0]
    else:
        return 'unknown'
df['name'] = df['Page'].apply(extract_name)
```

```
<ipython-input-12-206822d5b8b3>:8: PerformanceWarning: DataFrame is highly fragmente
d.  This is usually the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
  df['name'] = df['Page'].apply(extract_name)
```

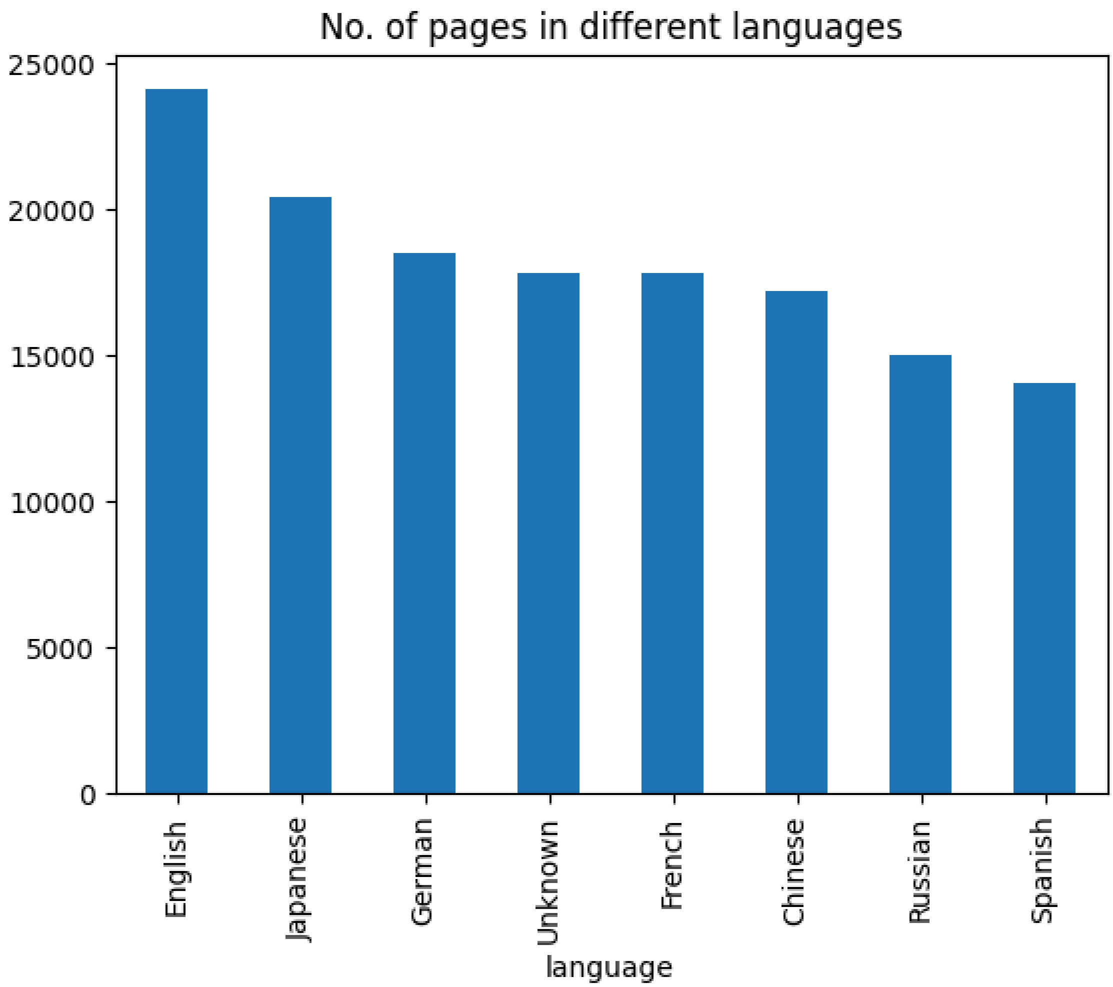
4.2.2. Extracting language

In [13]:
```
def extract_lang(page):
    pattern = r'(.{0,})_(.{2}).wikipedia.org_'
    result = re.findall(pattern, page)
    if len(result) == 1:
        return result[0][1]
    else:
        return 'un'
df['language'] = df['Page'].apply(extract_lang)
print(df['language'].unique())
```

```
['zh' 'fr' 'en' 'un' 'ru' 'de' 'ja' 'es']
```

```
<ipython-input-13-e92100694c85>:8: PerformanceWarning: DataFrame is highly fragmente
d.  This is usually the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
  df['language'] = df['Page'].apply(extract_lang)
```

In [14]:
```
lang_name_mapping={'zh':'Chinese', 'fr':'French', 'en':'English',
                   'un':'Unknown', 'ru':'Russian', 'de':'German',
                   'ja':'Japanese', 'es':'Spanish'}
df['language'] = df['language'].map(lang_name_mapping)
df['language'].value_counts().plot(kind='bar', title='No. of pages in different lan
plt.show()
print("% of pages in different languages")
round(df['language'].value_counts(normalize=True)*100,2)
```




```
% of pages in different languages
```

Out[14]:

	proportion
language	
English	16.62
Japanese	14.08
German	12.79
Unknown	12.31
French	12.27
Chinese	11.88
Russian	10.36
Spanish	9.70

dtype: float64

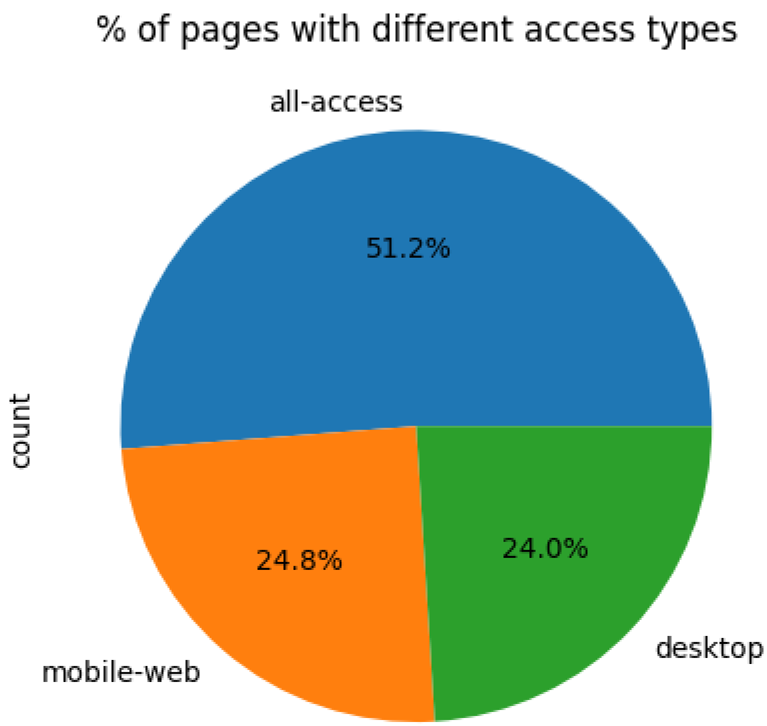
Insight

- Maximum number of pages, **16.62%**, are in **English** language

4.2.3. Extracting access type

In [15]:
```
df['access_type'] = df['Page'].str.findall(r'all-access|mobile-web|desktop').apply(
df['access_type'].value_counts().plot(kind='pie', autopct='%1.1f%%', title='% of pa
plt.show()
```

```
<ipython-input-15-e4ddf095414f>:1: PerformanceWarning: DataFrame is highly fragmente
d.  This is usually the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
  df['access_type'] = df['Page'].str.findall(r'all-access|mobile-web|desktop').apply
(lambda x: x[0])
```



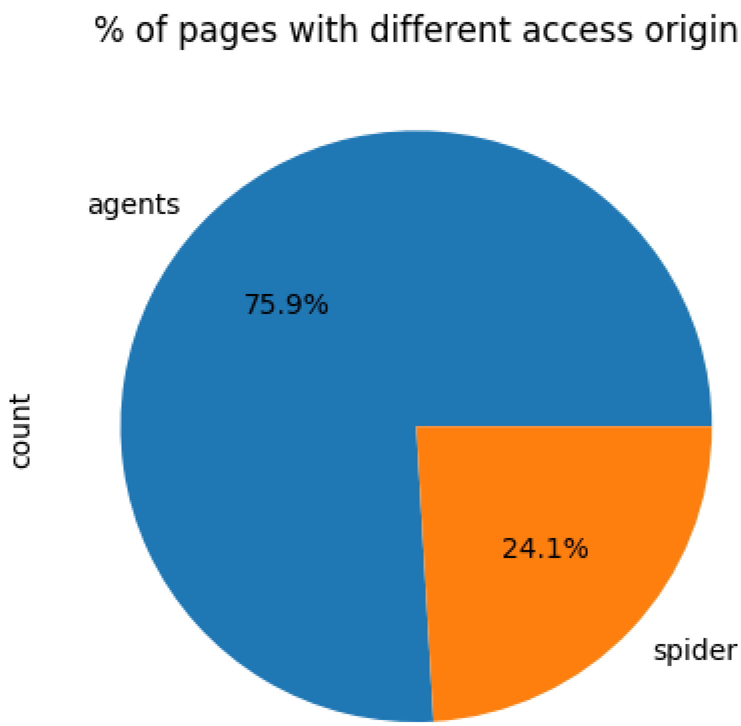
Insight

- Maximum number of pages, **51.2%**, have **all-access** access type

4.2.4. Extracting access origin

In [16]:
```
df['access_origin'] = df['Page'].str.findall(r'spider|agents').apply(lambda x: x[0]
df['access_origin'].value_counts().plot(kind='pie', autopct='%1.1f%%', title='% of
plt.show()
```

```
<ipython-input-16-86c9e6feaf3c>:1: PerformanceWarning: DataFrame is highly fragmente
d.  This is usually the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
  df['access_origin'] = df['Page'].str.findall(r'spider|agents').apply(lambda x: x
[0])
```



Insight

- Maximum number of pages, **75.9%**, have **agents** access origin

5. Aggregate and Pivoting

In [17]:
```
df.head()
```

Out[17]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 555 columns

Aggregating on language by taking average views per language for each date

In [18]:

```
df_agg = df.drop(columns=['Page', 'name', 'access_type', 'access_origin']).groupby(
df_agg['index'] = pd.to_datetime(df_agg['index'])
df_agg = df_agg.set_index('index')
df_agg.head()
```

Out[18]:

language	Chinese	English	French	German	Japanese	Russian	Sp
index							
2015-07-01	240.582042	3513.862203	475.150994	714.968405	580.647056	629.999601	1085.97
2015-07-02	240.941958	3502.511407	478.202000	705.229741	666.672801	640.902876	1037.81
2015-07-03	239.344071	3325.357889	459.837659	676.877231	602.289805	594.026295	954.41
2015-07-04	241.653491	3462.054256	491.508932	621.145145	756.509177	558.728132	896.05
2015-07-05	257.779674	3575.520035	482.557746	722.076185	725.720914	595.029157	974.50

5.1. Time series plots for all languages

In [19]:

```
df_agg.plot(figsize=(13,6))
plt.xlabel('Date')
plt.ylabel('No. of visits')
plt.show()
```

Insight

- **English** pages are the **most visited** pages follwed by Spanish
- **English** pages have an **upward trend** in terms of visits
- There is an **unusual peak** from **mid of July to end of August 2016**

6. Stationarity, Detrending, ACF and PACF

6.1. Stationarity test

Using Augmented Dickey-Fuller test to check for stationarity

- H0: The series is not stationary
- H1: The series is stationary

In [20]:
```
def adfuller_test(time_series):
    p_value = sm.tsa.stattools.adfuller(time_series)[1]
    if(p_value < 0.05):
        print('The time series is stationary')
    else:
        print('The time series is not stationary')
```

In [21]:
```
for lang in df_agg.columns:
    print(lang)
    adfuller_test(df_agg[lang])
    print()
```

```
Chinese
The time series is not stationary

English
The time series is not stationary

French
The time series is not stationary

German
The time series is not stationary

Japanese
The time series is not stationary

Russian
The time series is stationary

Spanish
The time series is stationary

Unknown
The time series is stationary
```

Insight

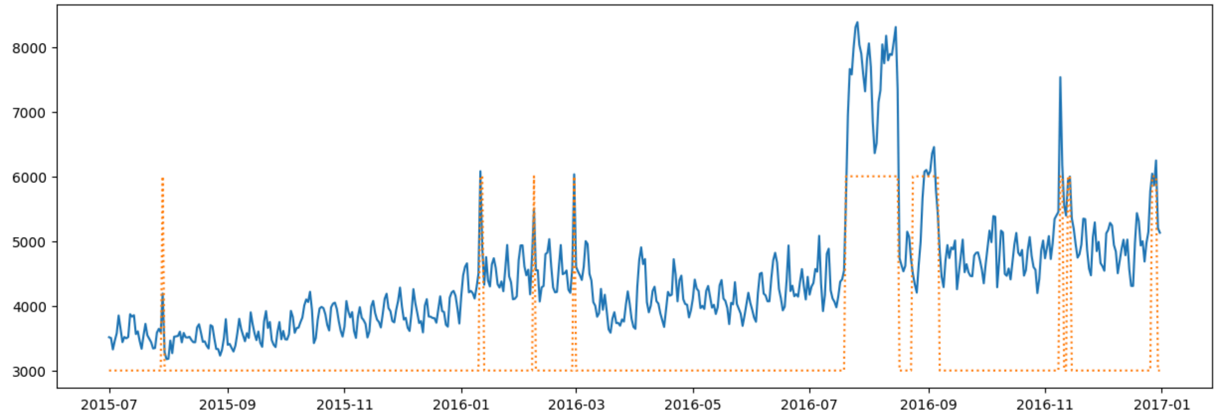
- Based on the Augmented Dickey-Fuller test, the time series corresponding to **Russian** and **Spanish** language page visits are **stationary**
- The time series corresponding to **Chinese, English, French, German** and **Japanese** language page visits are **not stationary**

From now on, we will work only on the English language page visit time series

In [22]:
```
ts_english = df_agg['English']
```

Let us look at the English time series along with its exog flag

In [23]:
```
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(ts_english.index, ts_english)
ax.plot(ts_english.index, (exog_en+1)*3000, ':')
plt.show()
```



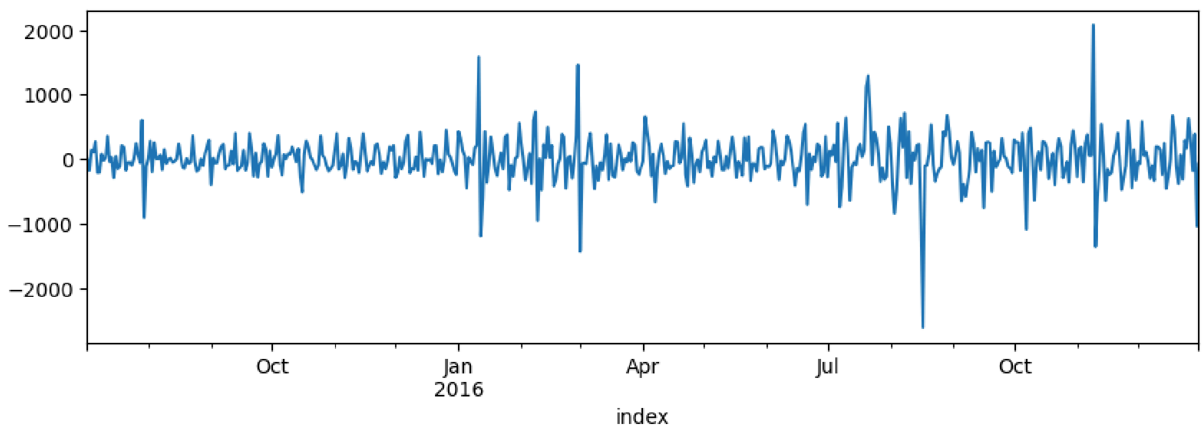
Insight

- It is very clear from the above plot that the time series looks like an additive time series with linear up trend and linear sesonality
- The unusual spikes in the visits are due to the special events marked by the orange peaks

6.2. De-trending and De-seasoning

As the trend is linear, differencing with the previous value should de-trend the time series

In [24]:
```
ts_english.diff(1).dropna().plot(figsize=(10,3))
plt.show()
```



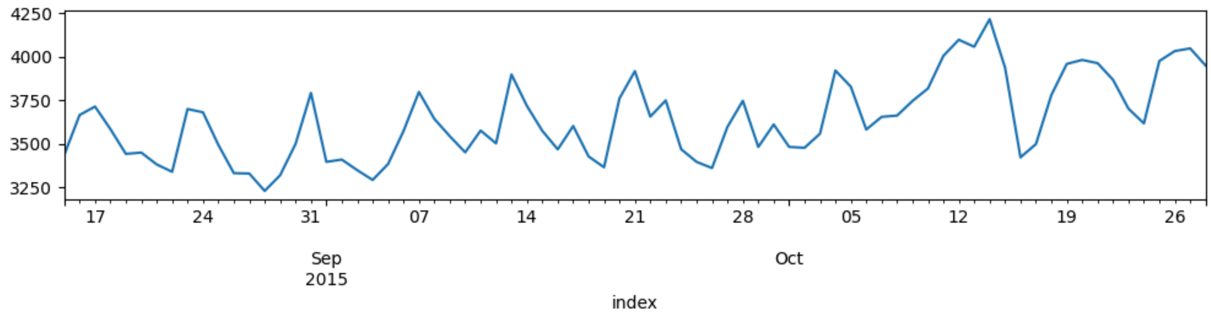
In [25]:
```
adfuller_test(ts_english.diff(1).dropna())
```

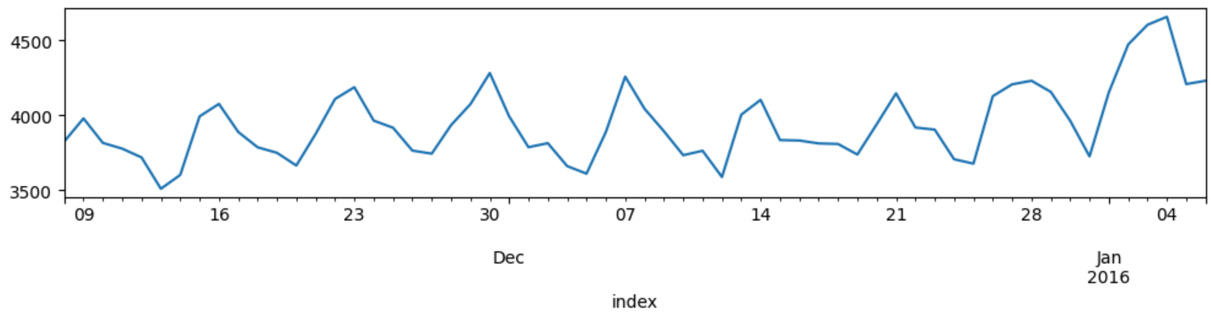
```
The time series is stationary
```

The time series became stationary by just doing first-order differencing, hence **d=1**
Let's now look at the seasonality

In [26]:
```
ts_english[45:120].plot(figsize=(12,2))
plt.show()
```

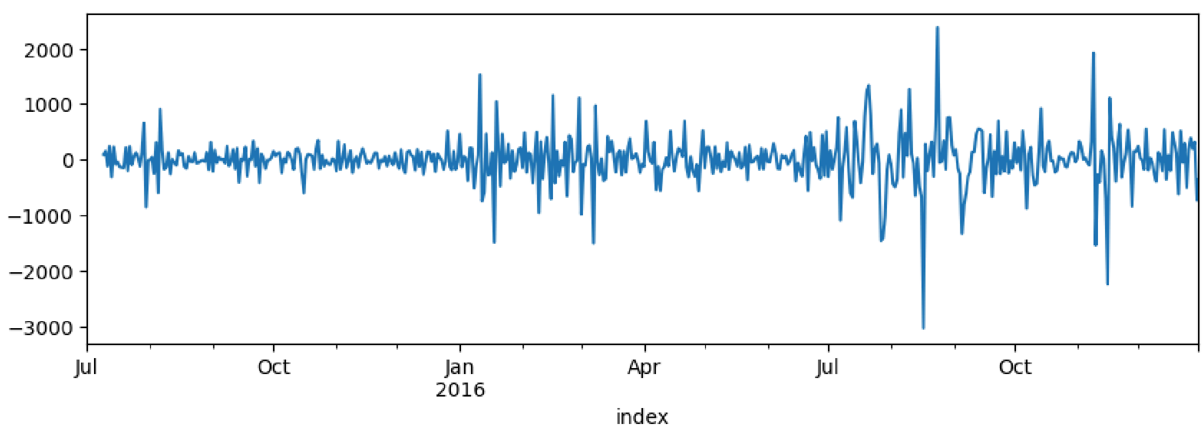
```
ts_english[130:190].plot(figsize=(12,2))
plt.show()
```





- Observing the above two plots, we can conclude that there is a **seasonality** of **7 days**. So **s=7**
- The peaks and troughs repeat every 7 days

In [27]:
```
ts_english.diff(1).diff(7).plot(figsize=(10,3))
plt.show()
```



In [28]:
```
adfuller_test(ts_english.diff(1).diff(7).dropna())
```

```
The time series is stationary
```

After **removing** the **trend**(and if required, **seasonality**) manually, the Augmented Dickey-Fuller test says that the **time series is stationary**

6.3. Auto de-composition

We had done manual decomposition above but there is a statsmodel library to decompose time series

In [29]:
```
decom = seasonal_decompose(ts_english)
ts_english_trend = decom.trend
ts_english_seas = decom.seasonal
ts_english_res = decom.resid
plt.figure(figsize=(15,8))
plt.subplot(411)
plt.plot(ts_english, label='actual')
plt.legend()
plt.subplot(412)
plt.plot(ts_english_trend, label='trend')
plt.legend()
plt.subplot(413)
plt.plot(ts_english_seas, label='seasonal')
plt.legend()
plt.subplot(414)
plt.plot(ts_english_res, label='residual')
plt.legend()
plt.tight_layout()
plt.show()
```

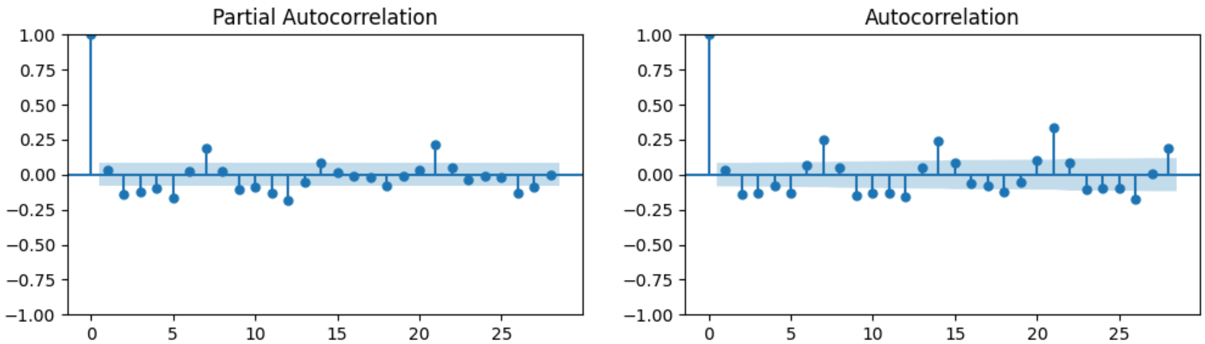
6.4. ACF and PACF plots

- The ACF plot shows the correlation of a time series with itself at different lags, while the PACF plot shows the correlation of a time series with itself at different lags, after removing the effects of the previous lags
- The ACF plot can be used to identify the order of an AR model. The order of an AR model is the number of lags that are included in the model. The ACF plot will show spikes at the lags that are included in the model.

- The PACF plot can be used to identify the order of an MA model. The order of an MA model is the number of lags that are included in the model. The PACF plot will show spikes at the lags that are included in the model \

Note: Stationary data needs to be provided to the ACF and PACF plots

In [30]:
```
fig, axs = plt.subplots(1,2, figsize=(12, 3))
plot_pacf(ax=axs[0], x=ts_english.diff(1).dropna())
plot_acf(ax=axs[1], x=ts_english.diff(1).dropna())
plt.show()
```



- From the PACF plot, we can see that there are 3 significant lags, at 5, 7 and 21. So **P=1,2 or 3**
- From the ACF plot, we can see that there are 3 significant lags, at 7, 14 and 21. So **Q=1,2 or 3**
- From the PACF plot, the cut-off is right from lag 0 and same for ACF plot. hence, **p** and **q = 0 or 1**

7. Model building and Evaluation

In [31]:
```
# Creating a function to print values of all these metrics.
def performance(actual, predicted, print_metrics=True):
    MAE = round(mae(actual, predicted), 3)
    RMSE = round(mse(actual, predicted)**0.5, 3)
    MAPE = round(mape(actual, predicted), 3)
    if(print_metrics==True):
        print('MAE :', MAE)
        print('RMSE :', RMSE)
        print('MAPE:', MAPE)
    return MAE, RMSE, MAPE
```

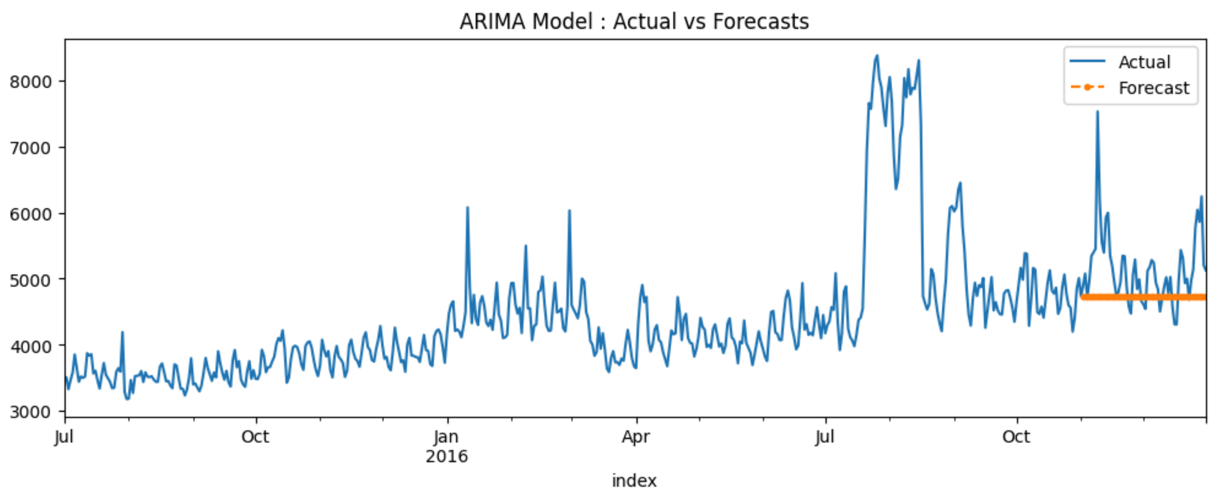
7.1. ARIMA model

In [32]:
```
TS = ts_english.copy(deep=True)
```

In [33]:

```
n_forecast = 60
model = ARIMA(TS[:-n_forecast], order = (0,1,0))
model = model.fit()
predicted = model.forecast(steps= n_forecast, alpha = 0.05)
plt.figure(figsize=(12,4))
TS.plot(label = 'Actual')
predicted.plot(label = 'Forecast', linestyle='dashed', marker='.')
plt.legend(loc="upper right")
plt.title('ARIMA Model : Actual vs Forecasts')
plt.show()
(_,_,_) = performance(TS.values[-n_forecast:], predicted.values, print_metrics=True
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/representation.p
y:374: FutureWarning: Unknown keyword arguments: dict_keys(['alpha']).Passing unknow
n keyword arguments will raise a TypeError beginning in version 0.15.
  warnings.warn(msg, FutureWarning)
```




```
MAE : 477.636
RMSE : 672.778
MAPE: 0.086
```

Insight

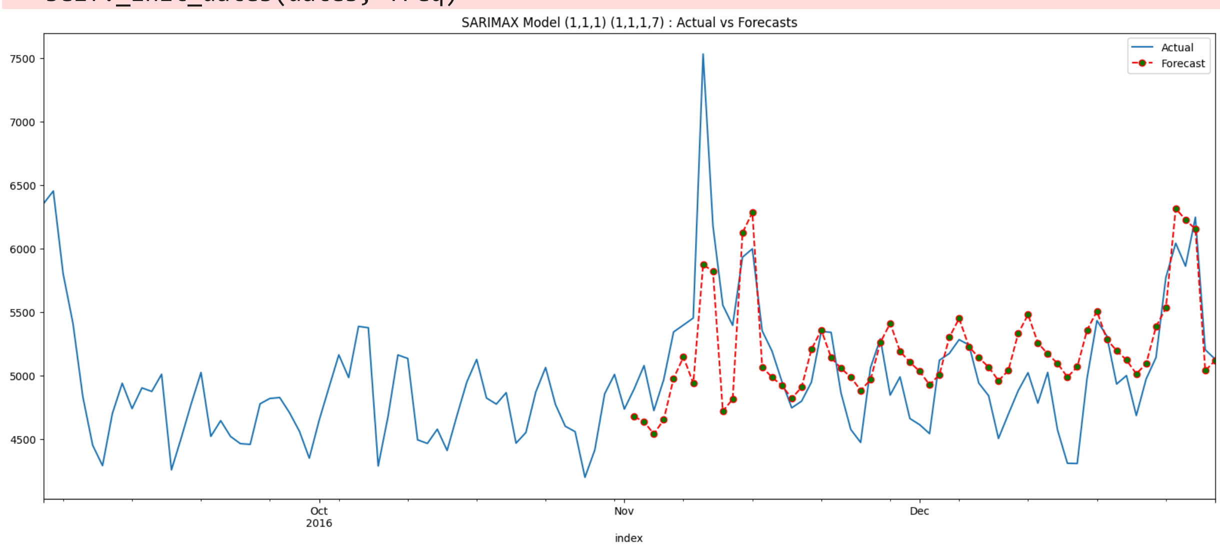
- The model is not doing a good job, even for different combinations of p and q

7.2. SARIMAX model

In [34]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [35]:
```
exog = exog_en['Exog'].to_numpy()
p,d,q,P,D,Q,s = 1,1,1,1,1,1,7
n_forecast = 60
model = SARIMAX(TS[:-n_forecast], order =(p,d,q), seasonal_order=(P, D, Q, s), exog
model_fit = model.fit()
#Creating forecast for last n-values
model_forecast = model_fit.forecast(n_forecast, dynamic = True, exog = pd.DataFrame

plt.figure(figsize = (20,8))
TS[-120:].plot(label = 'Actual')
model_forecast[-120:].plot(label = 'Forecast', color = 'red', linestyle='dashed', m
plt.legend(loc="upper right")
plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Forecasts')
plt.show()

(_,_,_) = performance(TS.values[-n_forecast:], model_forecast.values, print_metrics
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
```




```
MAE : 306.417
RMSE : 399.016
MAPE: 0.06
```

Insight

- SARIMAX model is doing a significantly better job. We need to search for the right order values

In [36]:
```
def SARIMAX_search(TS, forecast, p_list, d_list, q_list, P_list, D_list, Q_list, s_
    counter = 0
    #perf_df = pd.DataFrame(columns=['serial', 'pdq', 'PDQs', 'mape', 'rmse', 'aic'
    perf_df = pd.DataFrame(columns=['serial', 'pdq', 'PDQs', 'mape', 'rmse'])

    for p in p_list:
        for d in d_list:
            for q in q_list:
                for P in P_list:
                    for D in D_list:
                        for Q in Q_list:
                            for s in s_list:
                                try:
                                    model = SARIMAX(TS[:-n_forecast], order =(p,d,q
                                    model_fit = model.fit()
                                    model_forecast = model_fit.forecast(n_forecast,
                                    MAE, RMSE, MAPE = performance(TS.values[-n_fore
                                    counter += 1
                                    #list_row = [counter, (p,d,q), (P,D,Q,s), MAPE,
                                    list_row = [counter, (p,d,q), (P,D,Q,s), MAPE,
                                    perf_df.loc[len(perf_df)] = list_row
                                    print(f'Combination {counter} out of {(len(p_li
                                except:
                                    continue
    return perf_df
```

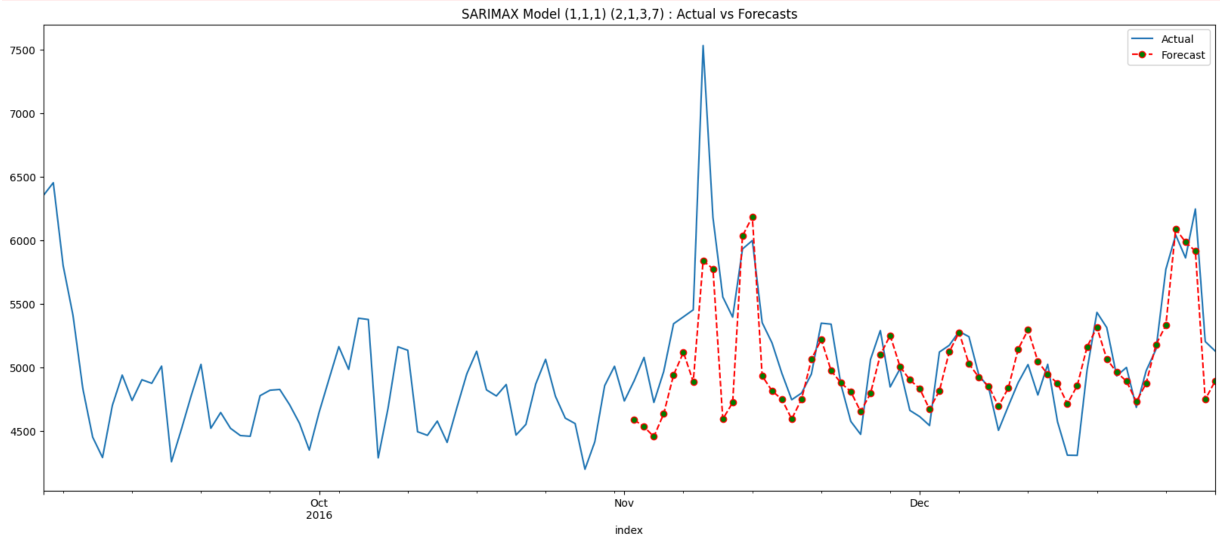
In [37]:
```
if 0:
    TS = ts_english.copy(deep=True)
    n_forecast = 60
    p_list = [0,1]
    d_list = [1]
    q_list = [0,1]
    P_list = [2,3]
    D_list = [1]
    Q_list = [2,3]
    s_list = [7]
    exog = exog_en['Exog'].to_numpy()
    perf_df = SARIMAX_search(TS, n_forecast, p_list, d_list, q_list, P_list, D_list
    perf_df.sort_values(['mape', 'rmse'])
```

After the above experiment, p,d,q,P,D,Q,s = 1,1,1,2,1,3,7 were found to be best values with low mape

In [38]:
```
exog = exog_en['Exog'].to_numpy()
p,d,q,P,D,Q,s = 1,1,1,2,1,3,7
n_forecast = 60
model = SARIMAX(TS[:-n_forecast], order =(p,d,q), seasonal_order=(P, D, Q, s), exog
model_fit = model.fit()
#Creating forecast for last n-values
model_forecast = model_fit.forecast(n_forecast, dynamic = True, exog = pd.DataFrame

plt.figure(figsize = (20,8))
TS[-120:].plot(label = 'Actual')
model_forecast[-120:].plot(label = 'Forecast', color = 'red', linestyle='dashed', m
```

```
plt.legend(loc="upper right")
plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Forecasts')
plt.show()

(_,_,_) = performance(TS.values[-n_forecast:], model_forecast.values, print_metrics
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWa
rning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```




```
MAE : 266.87
RMSE : 372.381
MAPE: 0.05
```

Insight

- There is good improvement in the SARIMAX model after tuning the parameters

7.4. Facebook Prophet

In [39]:
```
#!pip install pystan~=2.14
```

In [40]:
```
#!pip install prophet
```

In [41]:
```
TS = ts_english.copy(deep=True).reset_index()
TS = TS[['index', 'English']]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
exog = exog_en['Exog']
```

```
TS['exog'] = exog.values
TS.tail()
```

Out[41]:

	ds	y	exog
545	2016-12-27	6040.680728	1
546	2016-12-28	5860.227559	1
547	2016-12-29	6245.127510	1
548	2016-12-30	5201.783018	0
549	2016-12-31	5127.916418	0

In [42]:

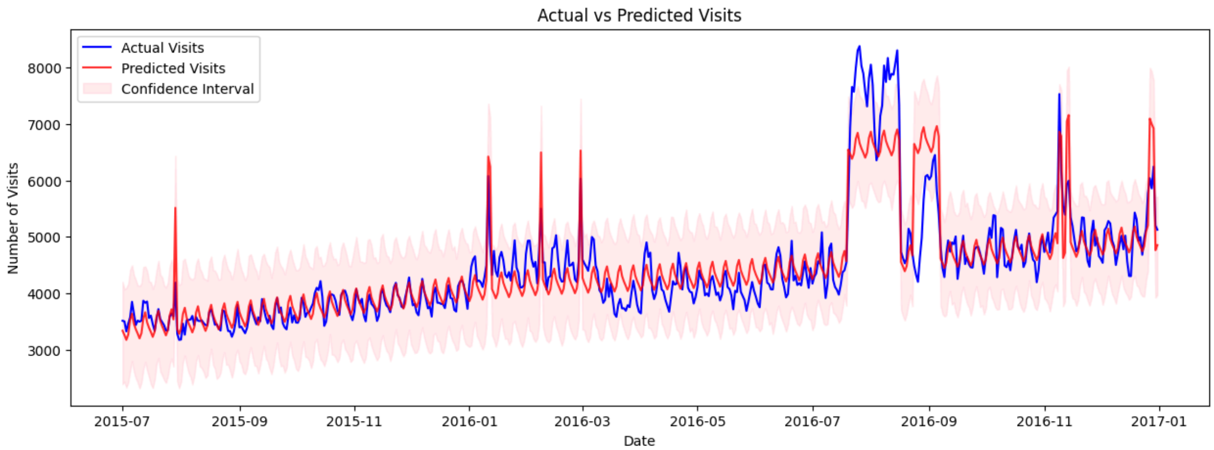
```
from prophet import Prophet
my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.add_regressor('exog')
n_forecast = 60
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
future_dates['exog'] = TS['exog']
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/tq2xkj16.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/lkpvxtce.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=32749', 'data', 'file=/tmp/tmptt5xnbws/tq
2xkj16.json', 'init=/tmp/tmptt5xnbws/lkpvxtce.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_modelcmajo3m6/prophet_model-20240916133212.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:12 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:12 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 287.417
RMSE : 441.959
MAPE: 0.06
```

In [43]:

```
# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
```

```
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```



Insight

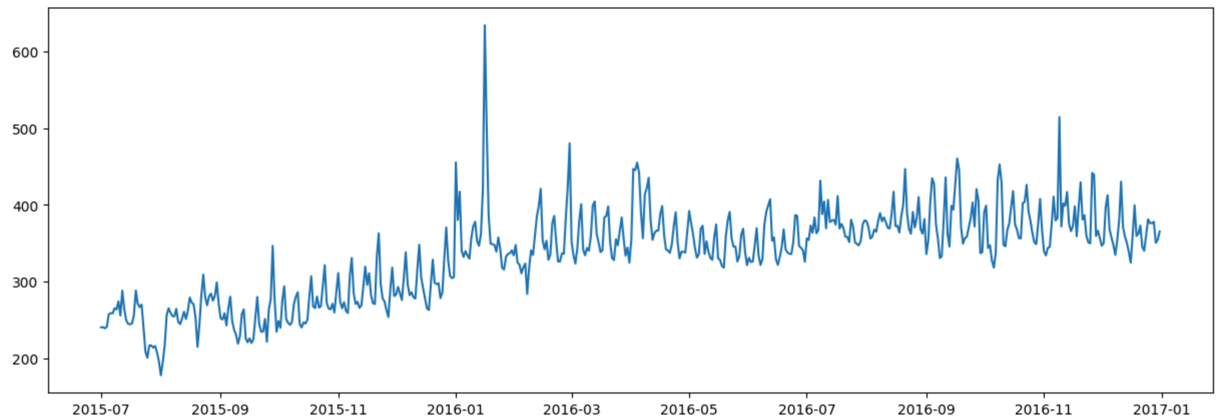
- Phropet is doing an incredible job capturing the trend and unusual peaks. It is also capturing the seasonality very well

7.5. Comparison

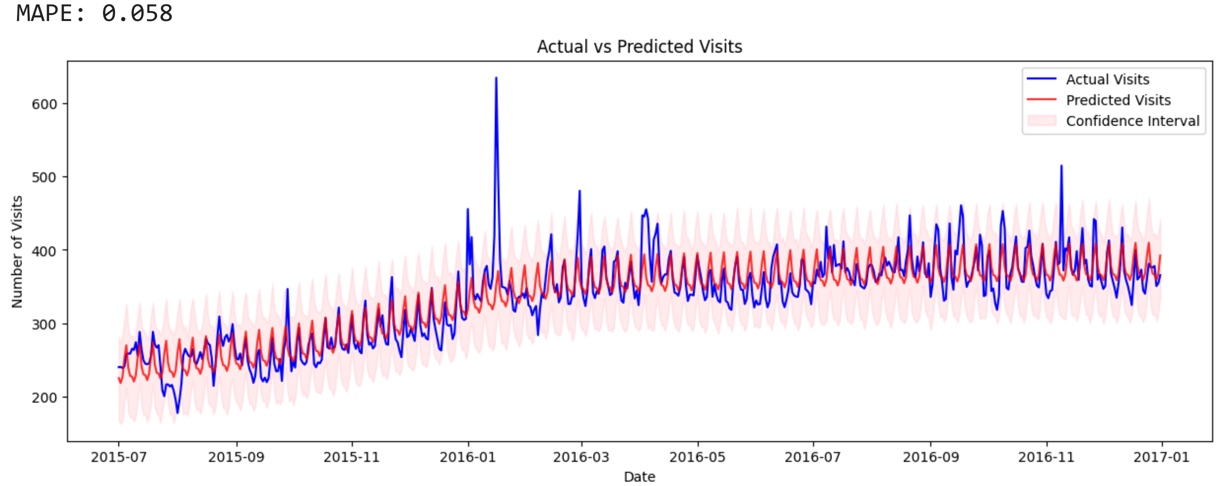
7.5.1 Chinese

In [44]:
```
lang = 'Chinese'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)
```

```python
# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```



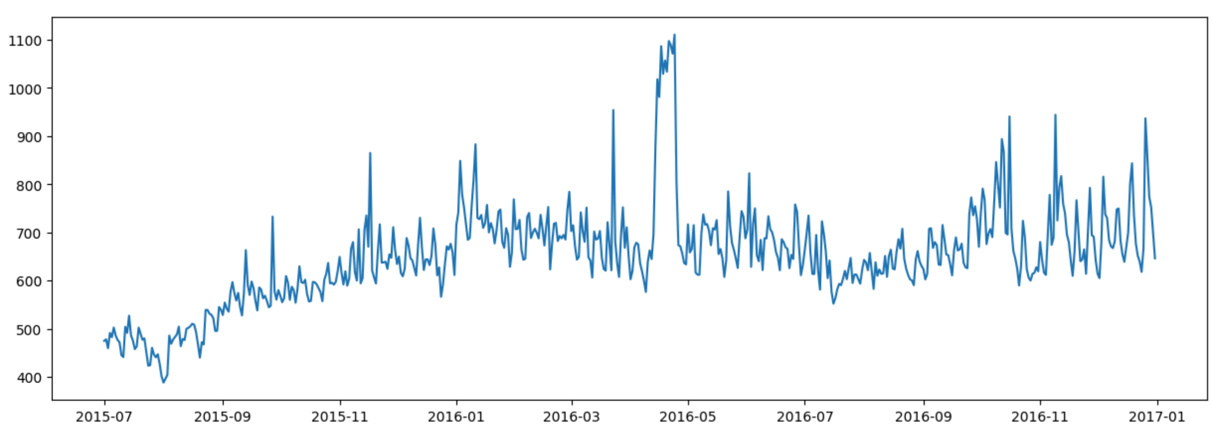
```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/gqhqlikl.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/l7rm6yk0.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=64283', 'data', 'file=/tmp/tmptt5xnbws/gq
hqlikl.json', 'init=/tmp/tmptt5xnbws/l7rm6yk0.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_modelg0inn_nu/prophet_model-20240916133213.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:13 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:13 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 19.353
RMSE : 28.703
MAPE: 0.058
```



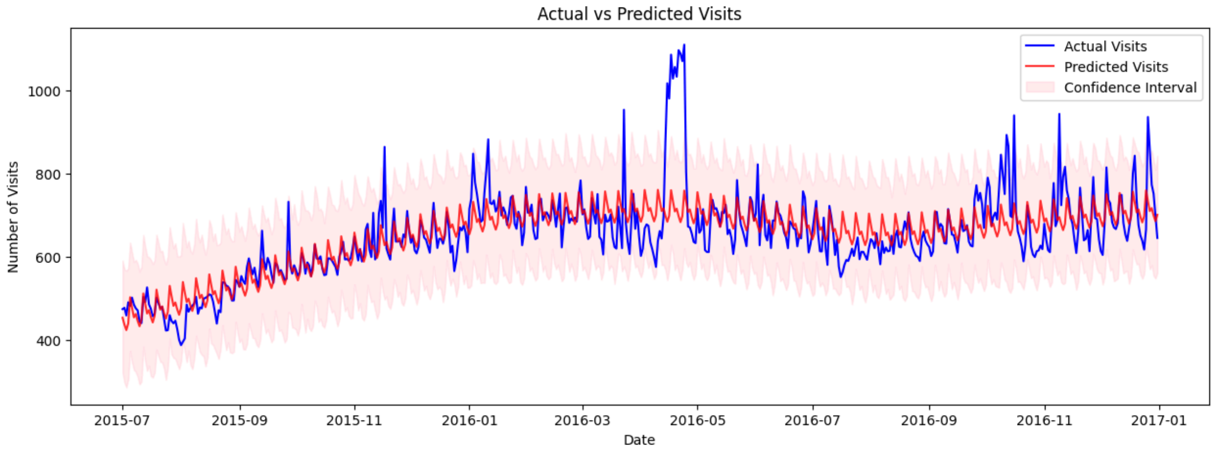
7.5.2 French

In [45]:

```
lang = 'French'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)

# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```



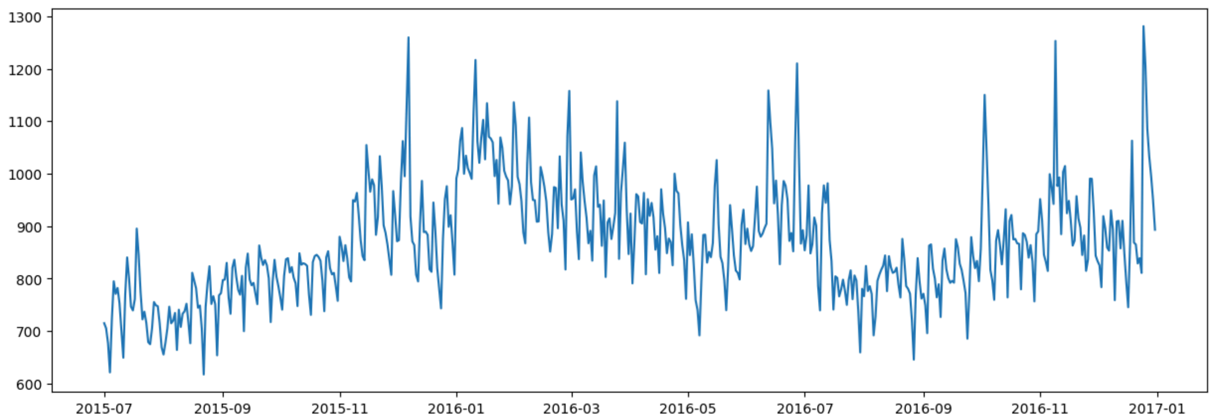
```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/z88r8z7u.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/ky4hva7e.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=91346', 'data', 'file=/tmp/tmptt5xnbws/z8
8r8z7u.json', 'init=/tmp/tmptt5xnbws/ky4hva7e.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_modelcpdfnxr8/prophet_model-20240916133214.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:14 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:14 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 42.038
RMSE : 68.864
MAPE: 0.061
```



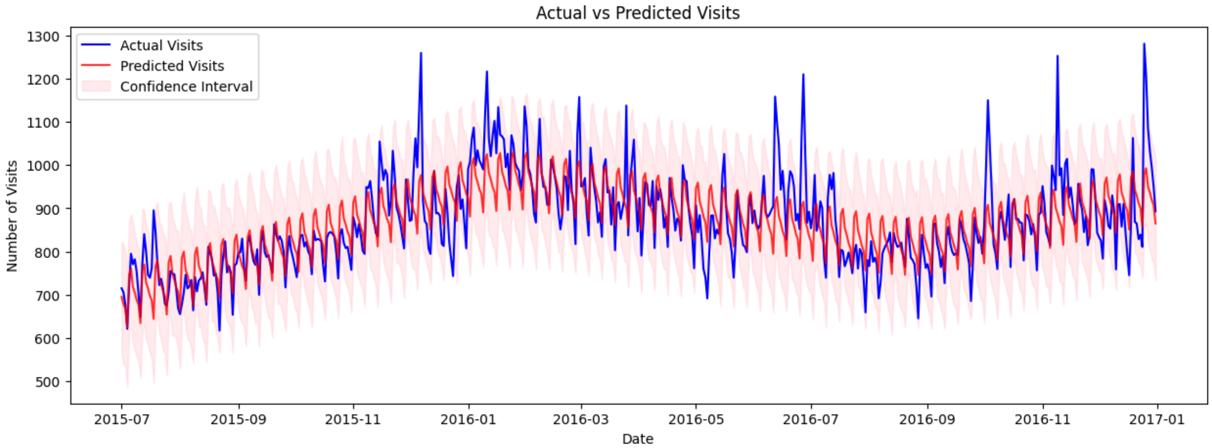
7.5.3 German

In [46]:
```python
lang = 'German'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval
```

```
(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)

# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```



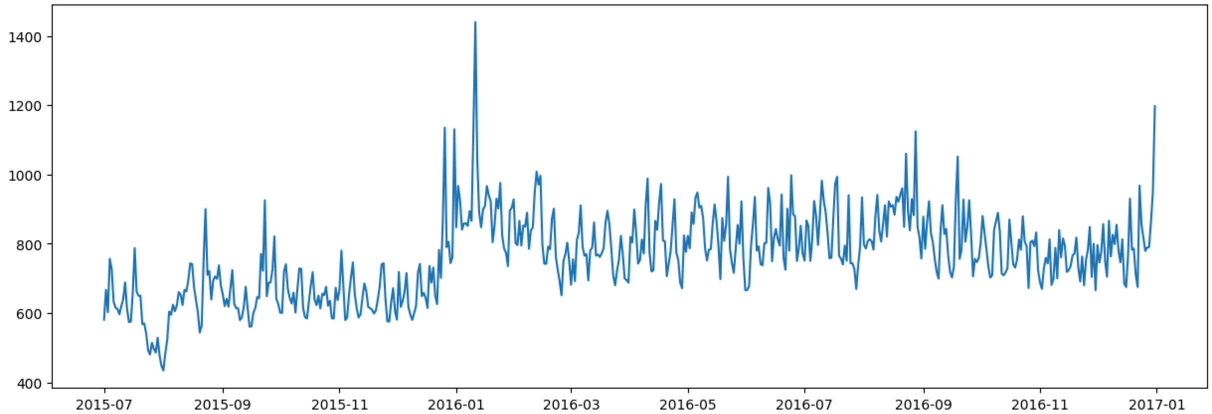

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/snrrw617.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/3c_dsv48.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=5360', 'data', 'file=/tmp/tmptt5xnbws/snr
rw617.json', 'init=/tmp/tmptt5xnbws/3c_dsv48.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_model1owoir90/prophet_model-20240916133215.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:15 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:15 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 49.262
RMSE : 68.189
MAPE: 0.055
```



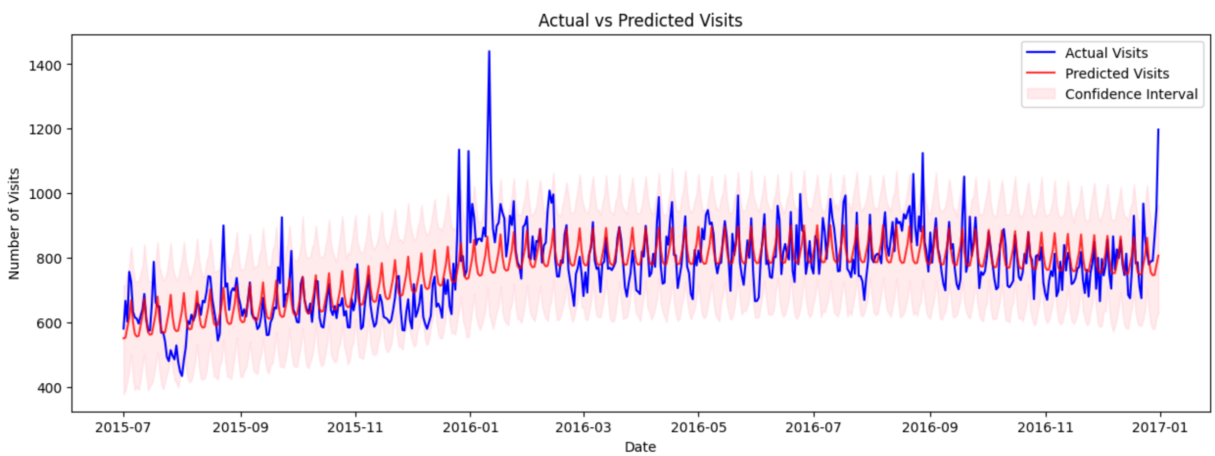
7.5.4 Japanese

In [47]:

```
lang = 'Japanese'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)

# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```




```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/4os3itn9.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/m7e6b26l.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=46678', 'data', 'file=/tmp/tmptt5xnbws/4o
s3itn9.json', 'init=/tmp/tmptt5xnbws/m7e6b26l.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_model9q_d_qml/prophet_model-20240916133218.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:18 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:18 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 61.153
RMSE : 84.062
MAPE: 0.08
```

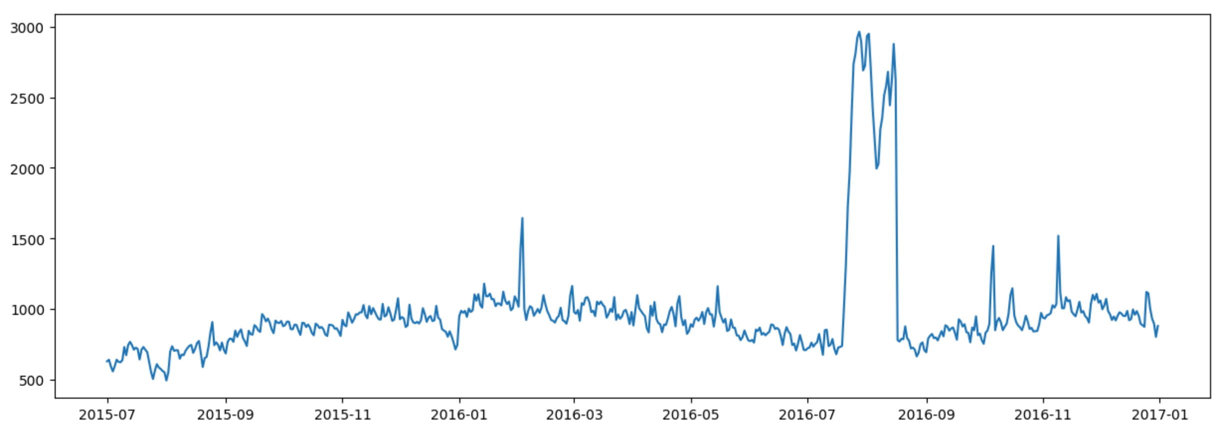


7.5.5 Russian

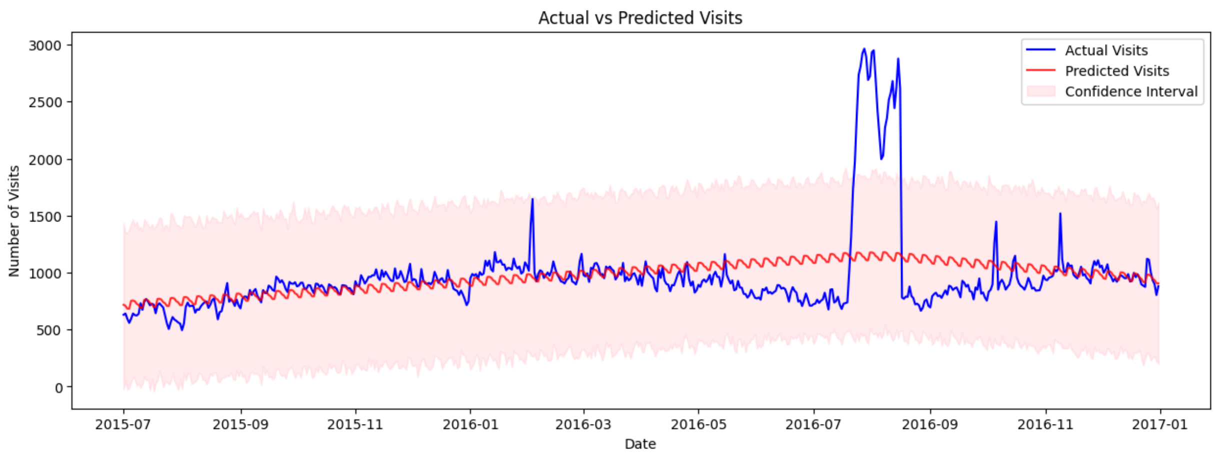
In [48]:

```
lang = 'Russian'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
```

```python
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)

# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```

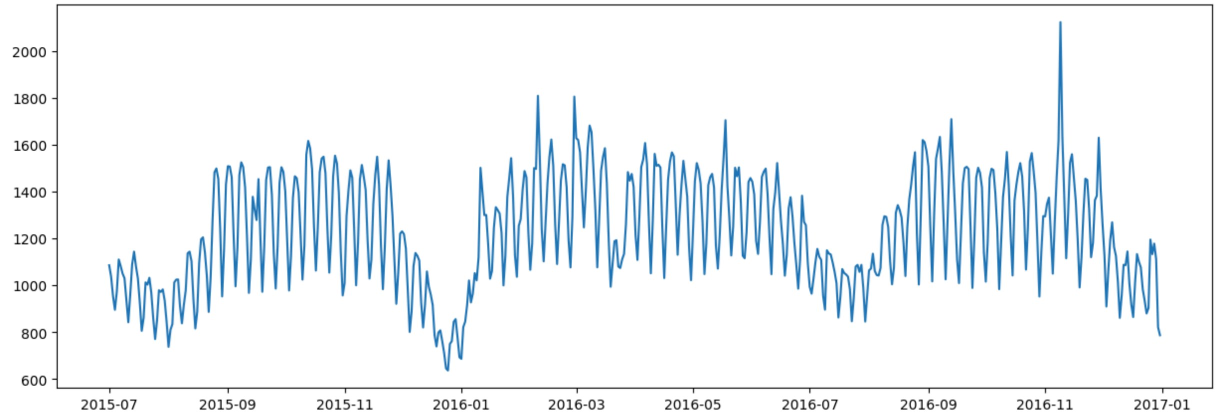



```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/idc2vmuh.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/te3xux83.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=70522', 'data', 'file=/tmp/tmptt5xnbws/id
c2vmuh.json', 'init=/tmp/tmptt5xnbws/te3xux83.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_modelermau_px/prophet_model-20240916133221.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:21 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:21 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 185.326
RMSE : 353.315
MAPE: 0.169
```

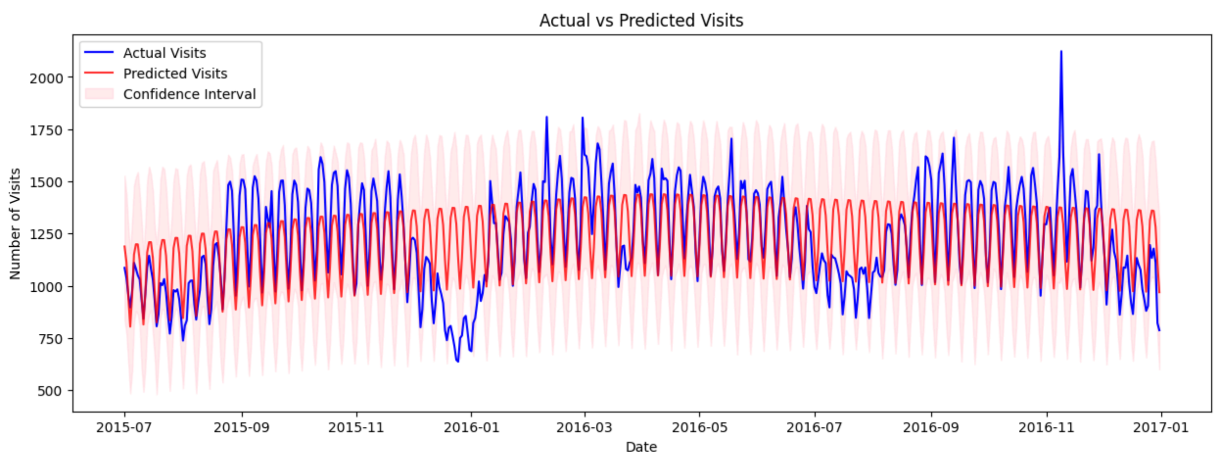


7.5.6 Spanish

In [49]:
```
lang = 'Spanish'
TS = df_agg[lang].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(TS.index, TS)
plt.show()

TS = TS.reset_index()
TS = TS[['index', lang]]
TS.columns = ['ds', 'y']
TS['ds'] = pd.to_datetime(TS['ds'])
TS.tail()

my_model = Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality
my_model.fit(TS)
future_dates = my_model.make_future_dataframe(periods=0)
forecast = my_model.predict(future_dates)

# Step 6: Merge Predictions with Actual Data
TS['yhat'] = forecast['yhat']  # The predicted values (yhat) from Prophet
TS['yhat_upper'] = forecast['yhat_upper']  # Upper bound of the confidence interval
TS['yhat_lower'] = forecast['yhat_lower']  # Lower bound of the confidence interval

(_,_,_) = performance(TS['y'], TS['yhat'], print_metrics=True)

# Plot actual vs predicted visits
plt.figure(figsize=(15, 5))
plt.plot(TS['ds'], TS['y'], label='Actual Visits', color='blue')
plt.plot(TS['ds'], TS['yhat'], label='Predicted Visits', color='red', alpha=0.8)
plt.fill_between(TS['ds'], TS['yhat_lower'], TS['yhat_upper'], color='pink', alpha=

plt.xlabel('Date')
plt.ylabel('Number of Visits')
plt.title('Actual vs Predicted Visits')
plt.legend()
plt.show()
```




```
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/l0f0a33u.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmptt5xnbws/tofbyvb9.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=1809', 'data', 'file=/tmp/tmptt5xnbws/l0f
0a33u.json', 'init=/tmp/tmptt5xnbws/tofbyvb9.json', 'output', 'file=/tmp/tmptt5xnbw
s/prophet_modelkgup0u65/prophet_model-20240916133223.csv', 'method=optimize', 'algor
ithm=lbfgs', 'iter=10000']
13:32:23 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
13:32:23 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
MAE : 134.492
RMSE : 173.774
MAPE: 0.118
```



In [49]:

In [49]:

8. Result Interpretations

- There are **7 known language** pages in the dataset - **English, Japanese, German, French, Chinese, Russian** and **Spanish**

- **English** has the maximum number of pages, **16.62%**. This is excpected as the maximum people speak English
- **Decomposition** helps in understanding the underlying **trend**, **seasonality** and **error**(residual) in the time series data.
- As per the analysis done on English language time series data, a differencing of 1 gives a stationary series. This is also tested using Augmented Dickey-Fuller test
- As per the exogenous variable given, the visits to the English page has an **unusual peak** whenever the **exogenous variable is 1**
- The performance of AdEase will be effected by events or campaings. AdEase can use the **Prophet model** along with **exogenous** variable to **improve** their **predictions**
- Without the exogenous variable, it becomes impossible to make accurate predictions. This is demonstarted by the plots of other languages which do not have exogneous variable
- The main difference between ARIMA, SARIMA and SARIMAX is:
  - ARIMA is used for modeling non-seasonal time series data with trends and autocorrelations
  - SARIMA extends ARIMA by explicitly modeling seasonality in the time series. It’s useful when the time series exhibits seasonal patterns (e.g., monthly or yearly cycles).
  - SARIMAX is an extension of SARIMA that includes exogenous variables (external factors) in the model. It models both seasonal and non-seasonal components, while also incorporating the effect of other external (exogenous) variables that may influence the time series.