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 Ingestion

- Read data from gdrive

In [30]:
```
import os
import gdown
import zipfile

# file_id = "1AbCDEfGhIJklMNopQRstuVWxyz12345"
# output_path = "train_1.csv"          # rename if needed
# gdown.download(f"https://drive.google.com/uc?id={file_id}", output_path, quiet=Fa

zip_id = "11uLnI8MB1BSMzzI4ox1jK7jbAwsxdbqo"
zip_path = "train_1.zip"

# Download the zip only if it doesn't already exist
if not os.path.exists(zip_path):
    gdown.download(f"https://drive.google.com/uc?id={zip_id}", zip_path, quiet=Fals
else:
```

```
    print(f"{zip_path} already exists. Skipping download.")

# Extract directly into current working directory (no subfolder)
# Skip extraction if the expected main file already exists
expected_file = "train_1.csv"
if not os.path.exists(expected_file):
    with zipfile.ZipFile(zip_path) as z:
        z.extractall(path=".")
    print("Extraction complete to current directory.")
else:
    print(f"{expected_file} already present. Skipping extraction.")
```

```
Downloading...
From (original): https://drive.google.com/uc?id=11uLnI8MB1BSMzzI4ox1jK7jbAwsxdbqo
From (redirected): https://drive.google.com/uc?id=11uLnI8MB1BSMzzI4ox1jK7jbAwsxdbqo&
confirm=t&uuid=a75cf7e2-d6e9-470a-a5c3-02bf603e7af1
To: c:\Users\FPK1COB\Documents\Learning\TimeSeries\AdEase_CaseStudy\train_1.zip
100%|██████████| 101M/101M [05:47<00:00, 292kB/s]
Extraction complete to current directory.
```

2.Libraries

Required Libraries

In [31]:
```
# libraries to analyze data
import numpy as np
import pandas as pd

# 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. Import Data

In [32]:
```
# 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')
```

```
************************************************
<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
************************************************
```

In [33]:

```
print(f'Number of unique values in each column: \n{df.nunique()}')
print('************************************************\n')
print('************************************************')
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
```

```
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 [34]:
```
df.head(20)
```

Out[34]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-0
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaI
5	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.
6	91Days_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaI
7	A'N'D_zh.wikipedia.org_all-access_spider	118.0	26.0	30.0	24.0	29.0	127.
8	AKB48_zh.wikipedia.org_all-access_spider	5.0	23.0	14.0	12.0	9.0	9.
9	ASCII_zh.wikipedia.org_all-access_spider	6.0	3.0	5.0	12.0	6.0	5.
10	ASTRO_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	1.
11	Ahq_e-Sports_Club_zh.wikipedia.org_all-access_...	2.0	1.0	4.0	4.0	2.0	6.
12	All_your_base_are_belong_to_us_zh.wikipedia.or...	2.0	5.0	5.0	1.0	3.0	3.
13	AlphaGo_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaI
14	Android_zh.wikipedia.org_all-access_spider	8.0	27.0	9.0	25.0	25.0	10.
15	Angelababy_zh.wikipedia.org_all-access_spider	40.0	17.0	25.0	42.0	41.0	7.
16	Apink_zh.wikipedia.org_all-access_spider	61.0	33.0	21.0	10.0	26.0	11.
17	Apple_II_zh.wikipedia.org_all-access_spider	4.0	8.0	4.0	9.0	7.0	4.
18	As_One_zh.wikipedia.org_all-access_spider	13.0	7.0	14.0	11.0	20.0	5.
19	B-PROJECT_zh.wikipedia.org_all-access_spider	NaN	NaN	NaN	NaN	NaN	NaI

20 rows × 551 columns

In [35]:
```
df.describe()
```

Out[35]:

	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 [36]:

```
df.describe(include='object')
```

Out[36]:

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

Observation

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

reading Exog_Campaign_eng file containing flag for each date indicating if those dates had a campaign/significant event which could have influenced the page views

In [37]:

```
file_id = "1GvWoXIxe1RaMWMSp1nNOw46Nxh_7vdzE"
output_path = "Exog_Campaign_eng"          # rename if needed
gdown.download(f"https://drive.google.com/uc?id={file_id}", output_path, quiet=Fals
```

```
Downloading...
From: https://drive.google.com/uc?id=1GvWoXIxe1RaMWMSp1nNOw46Nxh_7vdzE
To: c:\Users\FPK1COB\Documents\Learning\TimeSeries\AdEase_CaseStudy\Exog_Campaign_en
g
100%|██████████| 1.10k/1.10k [00:00<00:00, 1.15MB/s]
```

Out[37]: 'Exog_Campaign_eng'

In [38]:
```
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 [39]:
```
exog_en.head()
```

Out[39]:

	Exog
0	0
1	0
2	0
3	0
4	0

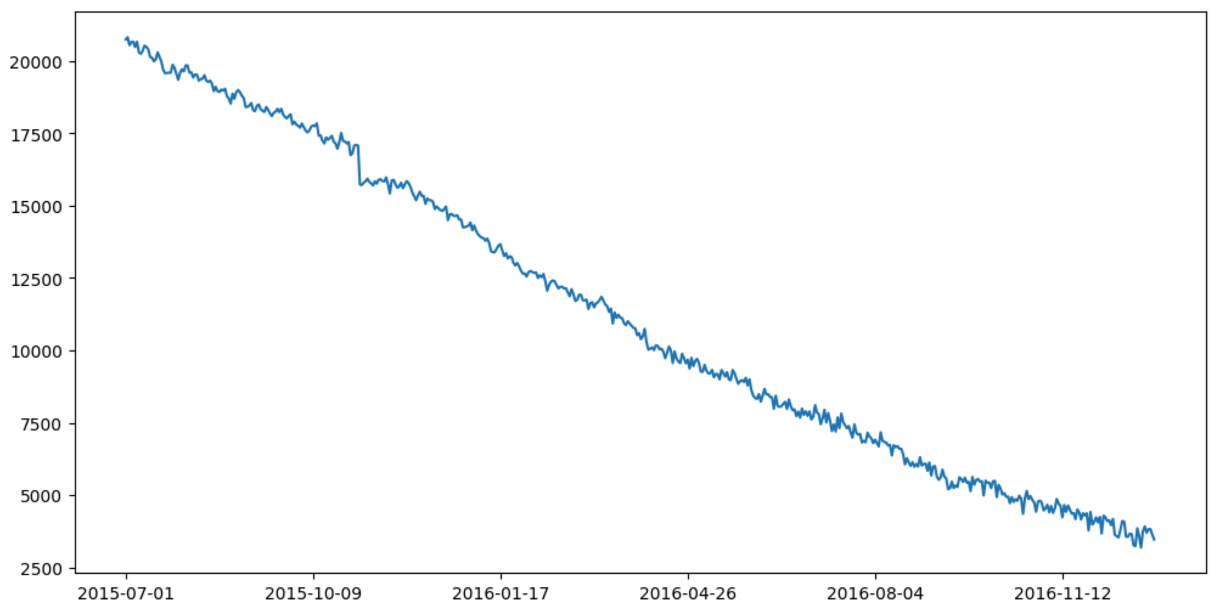
Observation

- For every **550** entries in **Exog_Campaign_eng** there are corresponding 550 days in the **train_1.csv** dataset
- **No** null/missing values
- **2** unique values - 1 ans 0

4. EDA

4.1 Date Columns

In [40]:
```
data_columns = df.columns[1:]
df[data_columns].isna().sum().plot(figsize=(12,6))
plt.show()
```



Observation

- The null values are keep decreasing with dates(time)
- We can infer that pages which are launched recently will not have views prior to launch

- We can fill those values with zeros.

In [41]:
```
df[data_columns] = df[data_columns].fillna(0)
```

In [42]:
```
df.isna().sum()
```

Out[42]:
```
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
Length: 551, dtype: int64
```

4.2 Extract information from page column

like

- page name
- Language
- domain
- Device type used to access data
- access origin

4.2.Extracting Page name from page column

In [43]:
```
df.Page.sample(10)
```

Out[43]:
```
100627      Мухаммед_ru.wikipedia.org_all-access_all-agents
16317     Руни,_Уэйн_ru.wikipedia.org_mobile-web_all-agents
26176     Liste_de_noms_de_couleur_fr.wikipedia.org_all-...
106447        萊文斯基醜聞_zh.wikipedia.org_mobile-web_all-agents
140469    Jutta_Winkelmann_de.wikipedia.org_all-access_a...
120089         田中みな実_ja.wikipedia.org_all-access_all-agents
78688     File:Roger_Waters_18_May_2008_London_O2_Arena....
59075           武田梨奈_ja.wikipedia.org_mobile-web_all-agents
105261            藜麥_zh.wikipedia.org_mobile-web_all-agents
133263             松たか子_ja.wikipedia.org_all-access_spider
Name: Page, dtype: object
```

The page column contains data in the below format:

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

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

In [44]:
```
# 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)
# df[['Page', 'name']].head(10)
```

Why we commented above code?

- The above code findall tries to scan entire page name and lists with similar format
- But re.search only returns the first entry which would be sufficient and fast

In [45]:
```
def extract_page_name(page):
    try:
        return re.search(r'^(.*?)_', page).group(1)
    except:
        return page

df['name'] = df.Page.apply(extract_page_name)
df[['Page', 'name']].head(10)
```

```
C:\Users\FPK1COB\AppData\Local\Temp\ipykernel_27144\4170752469.py:7: PerformanceWarning: DataFrame is highly fragmented.  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_page_name)
```

Out[45]:

	Page	name
0	2NE1_zh.wikipedia.org_all-access_spider	2NE1
1	2PM_zh.wikipedia.org_all-access_spider	2PM
2	3C_zh.wikipedia.org_all-access_spider	3C
3	4minute_zh.wikipedia.org_all-access_spider	4minute
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	52
5	5566_zh.wikipedia.org_all-access_spider	5566
6	91Days_zh.wikipedia.org_all-access_spider	91Days
7	A'N'D_zh.wikipedia.org_all-access_spider	A'N'D
8	AKB48_zh.wikipedia.org_all-access_spider	AKB48
9	ASCII_zh.wikipedia.org_all-access_spider	ASCII

4.2.2 Extracting Language from Page column

re.search(r'_\w{2}.wikipedia.org')

In [46]:
```
def extract_langugage(page):
    try:
        return re.search(r'_(\w{2})\.wikipedia\.org', page).group(1)
    except:
        return 'un'
df['language'] = df.Page.apply(extract_langugage)
print(df['language'].unique())
```

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

```
C:\Users\FPK1COB\AppData\Local\Temp\ipykernel_27144\3534192211.py:6: PerformanceWarning: DataFrame is highly fragmented.  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_langugage)
```

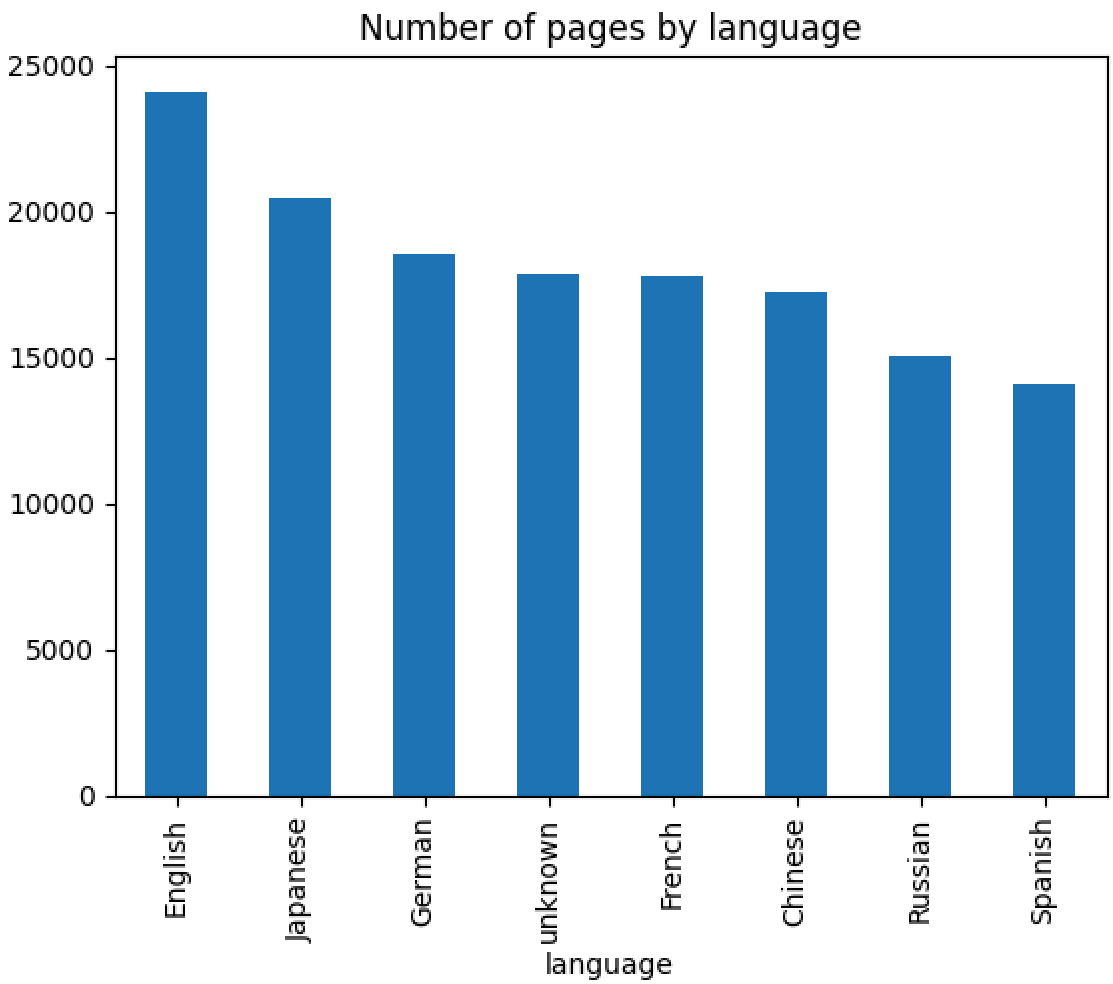
In [47]:
```
df.head(10)
```

Out[47]:

	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	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0
6	91Days_zh.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	A'N'D_zh.wikipedia.org_all-access_spider	118.0	26.0	30.0	24.0	29.0	127.0	53.0
8	AKB48_zh.wikipedia.org_all-access_spider	5.0	23.0	14.0	12.0	9.0	9.0	35.0
9	ASCII_zh.wikipedia.org_all-access_spider	6.0	3.0	5.0	12.0	6.0	5.0	4.0

10 rows × 553 columns

In [48]:
```
language_name_mapping ={
    'zh': 'Chinese',
    'fr': 'French',
    'en': 'English',
    'un': 'unknown',
    'ru': 'Russian',
    'de': 'German',
    'ja': 'Japanese',
    'es': 'Spanish'
}
df['language'] = df['language'].map(language_name_mapping)
df['language'].value_counts().plot(kind='bar', title='Number of pages by language')
plt.show()
```



In [49]:
```
## % pages of different languages
round(df['language'].value_counts(normalize=True)*100, 2)
```

Out[49]:
```
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
Name: proportion, dtype: float64
```

Observation

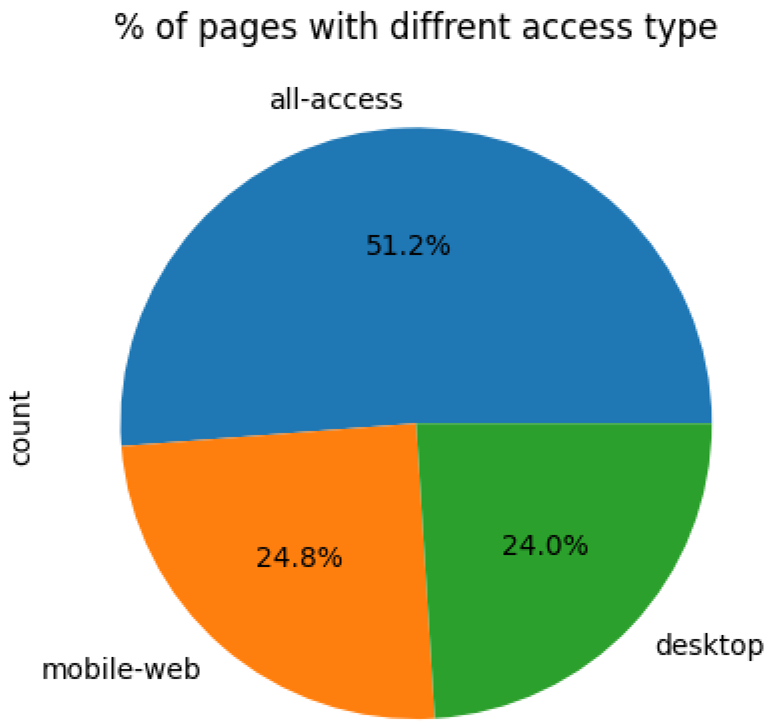
- Maximum number of pages are in English with 16.62%
- Followed by Japanese with 14.08%

4.2.3 Extracting access type

In [50]:
```
def extract_accessType(page):
    try:
        pattern = r'all-access|mobile-web|desktop'
        return re.search(pattern, page).group(0)
    except:
        return 'un'
df['access_type'] =df.Page.apply(extract_accessType)
df['access_type'].value_counts().plot(kind='pie', autopct='%1.1f%%', title='% of pa
```

```
C:\Users\FPK1COB\AppData\Local\Temp\ipykernel_27144\3091566948.py:7: PerformanceWarn
ing: DataFrame is highly fragmented.  This is usually the result of calling `frame.i
nsert` 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 = fram
e.copy()`
  df['access_type'] =df.Page.apply(extract_accessType)
```

Out[50]:
```
<Axes: title={'center': '% of pages with diffrent access type'}, ylabel='count'>
```



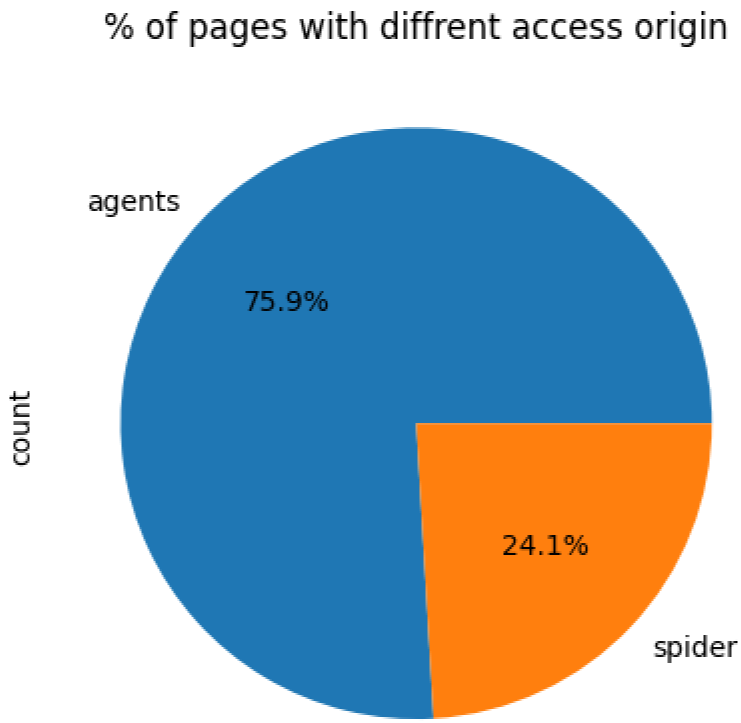
Observation

- Nearly half of the pages have all access
- Rest half are either accessible on mobile or desktop with almost equal percentage

4.2.4 Extracting access origin

In [51]:
```
df['access_origin'] = df['Page'].str.findall('spider|agents').apply(lambda x: x[0])
df['access_origin'].value_counts().plot(kind='pie', autopct='%1.1f%%', title='% of
plt.show()
```

```
C:\Users\FPK1COB\AppData\Local\Temp\ipykernel_27144\1560931515.py:1: PerformanceWarn
ing: DataFrame is highly fragmented.  This is usually the result of calling `frame.i
nsert` 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 = fram
e.copy()`
  df['access_origin'] = df['Page'].str.findall('spider|agents').apply(lambda x: x
[0])
```



Observations

- Most pages(75.9%) have **agents** as access origin

5.Aggregating and Pivoting

In [52]:
```
df.head(10)
```

Out[52]:

	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	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0
6	91Days_zh.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	A'N'D_zh.wikipedia.org_all-access_spider	118.0	26.0	30.0	24.0	29.0	127.0	53.0
8	AKB48_zh.wikipedia.org_all-access_spider	5.0	23.0	14.0	12.0	9.0	9.0	35.0
9	ASCII_zh.wikipedia.org_all-access_spider	6.0	3.0	5.0	12.0	6.0	5.0	4.0

10 rows × 555 columns

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

In [53]:

```
df_agg = df.drop(columns=['Page', 'name', 'access_type', 'access_origin'])
df_agg = df_agg.groupby(['language']).mean().T.reset_index()
df_agg['index'] = pd.to_datetime(df_agg['index'])
df_agg =df_agg.set_index('index')
df_agg.head(10)
```

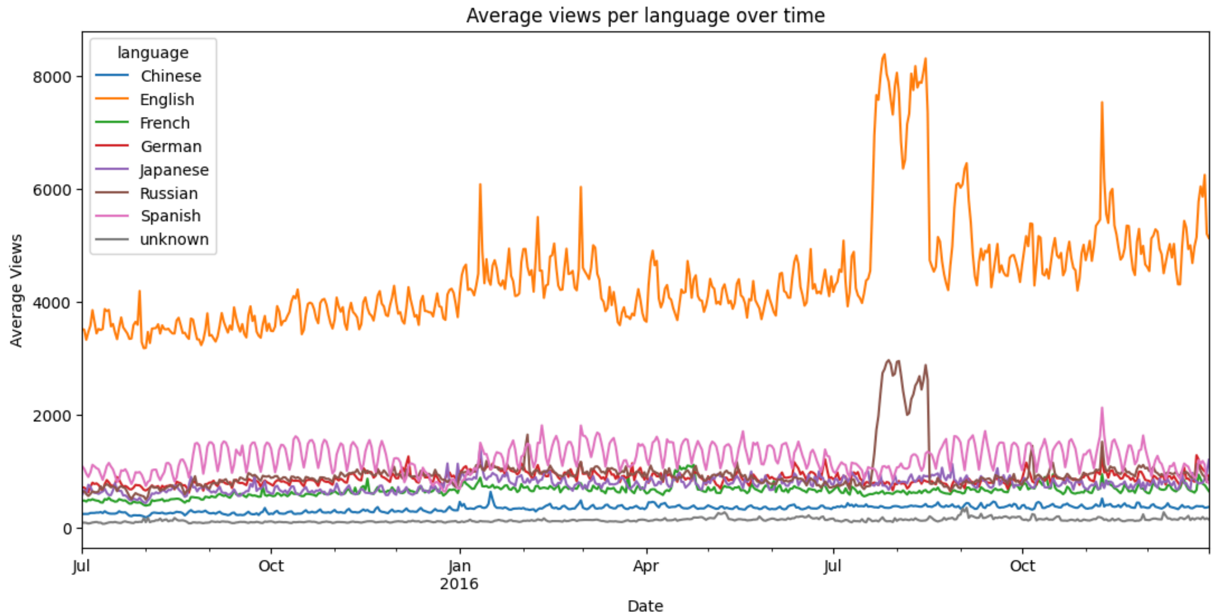
Out[53]:

language index	Chinese	English	French	German	Japanese	Russian	Sp
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
2015-07-06	259.114864	3849.736021	502.741209	794.832480	632.399148	640.986287	1110.63
2015-07-07	258.832260	3643.523063	485.945399	770.814256	615.184181	626.293436	1082.56
2015-07-08	265.589529	3437.871080	476.998820	782.077641	611.462337	623.360205	1050.66
2015-07-09	263.964420	3517.459391	472.061903	752.939990	596.067642	638.550726	1030.84
2015-07-10	274.414592	3497.571594	445.495057	701.702593	619.299300	731.252297	937.12

5.1 Time Series plot for all languages

In [54]:

```python
df_agg.plot(figsize=(13,6), title='Average views per language over time')
plt.xlabel('Date')
plt.ylabel('Average Views')
plt.show()
```



Observations:

- English pages are the most visited pages
- Followed by Spanish
- English pages have **upward trend**
- There is an **unusual peak** from **mid of July to end of August 2016** for **English** and **Russian** pages

6 Stationarity, Detrending, ACF and PACF plots

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 [55]:
```
def adfuller_test(timeseries):
    p_value = sm.tsa.stattools.adfuller(timeseries)[1]
    if p_value <= 0.05:
        print("Time series is stationary")
    else:
        print("Time series is non-stationary")
```

In [56]:
```
for language in df_agg.columns:
    print(f'ADF test for {language}:')
    adfuller_test(df_agg[language])
    print('-------------------------')
```

```
ADF test for Chinese:
Time series is non-stationary
-------------------------
ADF test for English:
Time series is non-stationary
-------------------------
ADF test for French:
Time series is non-stationary
-------------------------
ADF test for German:
Time series is non-stationary
-------------------------
ADF test for Japanese:
Time series is non-stationary
-------------------------
ADF test for Russian:
Time series is stationary
-------------------------
ADF test for Spanish:
Time series is stationary
-------------------------
ADF test for unknown:
Time series is stationary
-------------------------
```

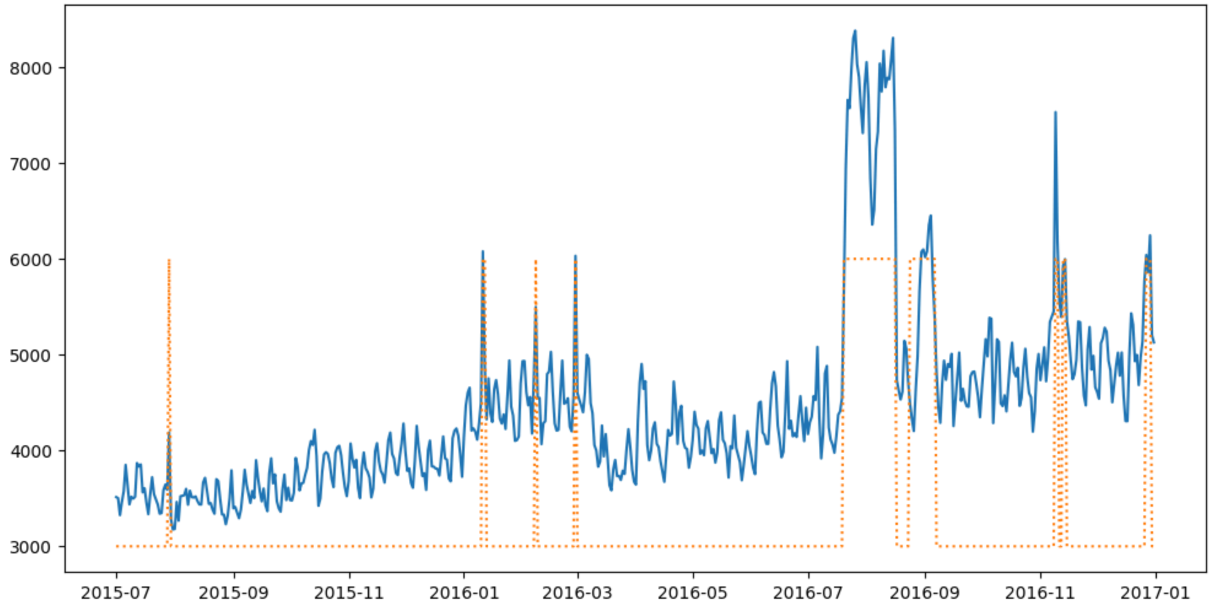
Observations:

- Only **Spanish, Russian** page visits are **stationary**
- **Chinese, English, French, German and Japanese** page visits are **not stationary**.

Starting with **English**

In [57]:
```
english_ts = df_agg['English']
```

In [58]:
```
fig, ax = plt.subplots(figsize=(12,6))
ax.plot(english_ts.index, english_ts)
ax.plot(english_ts.index, (exog_en + 1)*3000, ":") ## As english pages min mean is
plt.show()
```

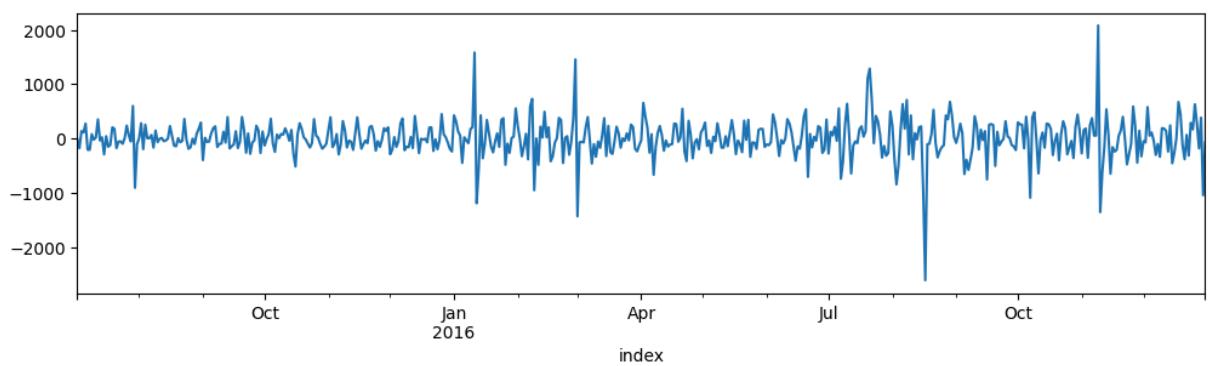


Observation:

- From above plot the ts looks like linear upward trend and linear seasonality
- Unusual spikes in page visits during the special events marked with orange peaks

6.2 De-Trending and De-seasoning

In [59]:
```
english_ts.diff(1).dropna().plot(figsize=(12, 3))
plt.show()
```

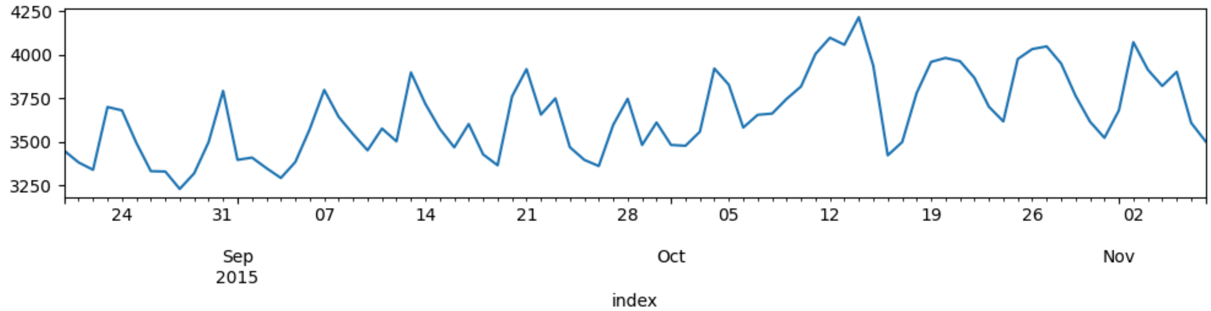


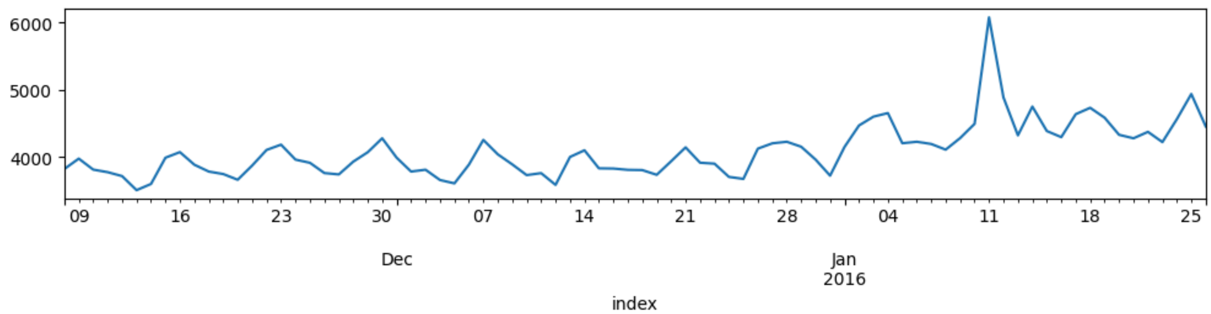
In [60]:
```
adfuller_test(english_ts.diff(1).dropna())
```

```
Time series is stationary
```

Series become stationary by doing first order diffrencing => **d = 1**

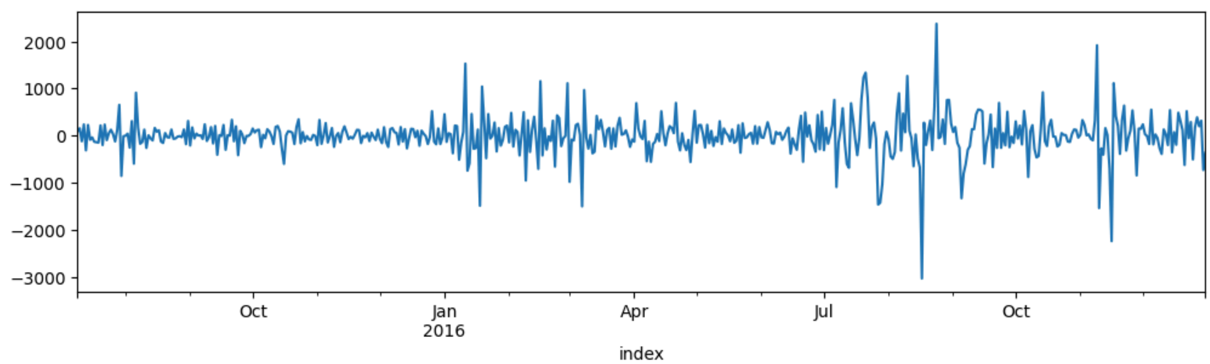
In [61]:
```
## Deseasoning
## check any small part of series
english_ts[50:130].plot(figsize=(12,2))
plt.show()
english_ts[130:210].plot(figsize=(12,2))
plt.show()
```





Seasonality is observed for every **7 days** ==> **s=7**

In [62]:
```
english_ts.diff(1).diff(7).dropna().plot(figsize=(12,3))
plt.show()
```



In [63]:
```
adfuller_test(english_ts.diff(1).diff(7).dropna())
```

```
Time series is stationary
```

As **Trend** and **Seasonality** are removed manually, ADF test gives **time series is stationary**

6.3. Auto de-composition

Auto decomposition using statsmodel library to decompose time series

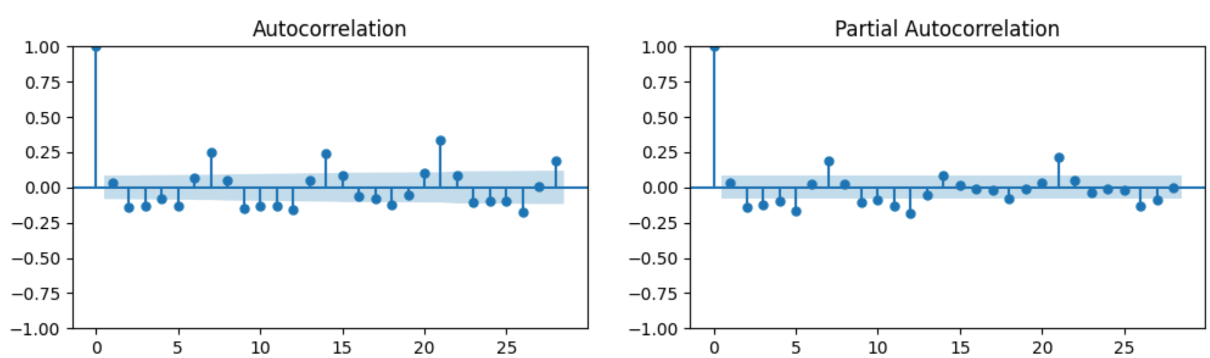
In [64]:
```
decom = seasonal_decompose(english_ts)
english_ts_trend = decom.trend
english_ts_seasonal = decom.seasonal
english_ts_res = decom.resid
plt.figure(figsize=(15,8))
plt.subplot(411)
plt.plot(english_ts, label = 'actual')
```

```
plt.legend()
plt.subplot(412)
plt.plot(english_ts_trend, label = 'trend')
plt.legend()
plt.subplot(413)
plt.plot(english_ts_seasonal, label = 'seasonal')
plt.legend()
plt.subplot(414)
plt.plot(english_ts_res, label = 'residual')
plt.legend()
```

Out[64]: `<matplotlib.legend.Legend at 0x297a5937510>`

6.4 ACF and PACF plots

In [65]:
```
fig, ax = plt.subplots(1,2,figsize=(12,3))
plot_acf(ax=ax[0], x=english_ts.diff(1).dropna())
plot_pacf(ax=ax[1], x=english_ts.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

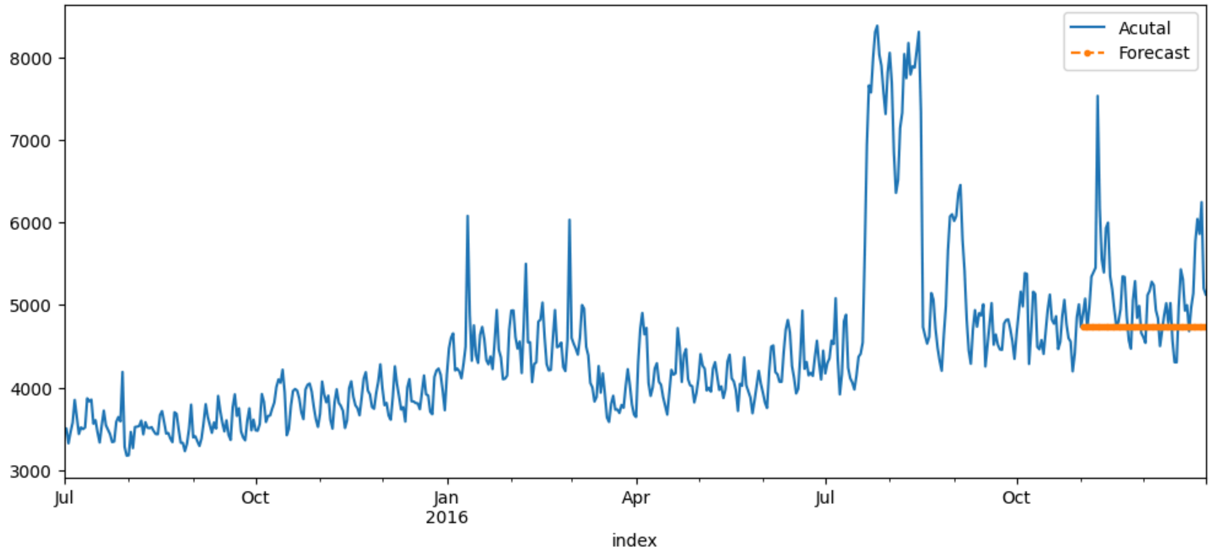
In [66]:
```
# Creating a function to print values of all these metrics.
def performance(actual, predicted, print_metrics=True):
    MAE = round(mae(actual, predicted), 3)
    RMSE = round(mse(actual, predicted)**0.5, 3)
    MAPE = round(mape(actual, predicted), 3)
    if(print_metrics==True):
        print('MAE :', MAE)
        print('RMSE :', RMSE)
        print('MAPE:', MAPE)
    return MAE, RMSE, MAPE
```

7.1 ARIMA model

In [67]:
```
timeSeries = english_ts.copy(deep=True)
```

In [68]:
```
n_forecast = 60
model = ARIMA(timeSeries[:-n_forecast], order=(0,1,0))
model = model.fit()
predicted = model.forecast(steps=n_forecast, alpha=0.05)
plt.figure(figsize=(12,5))
timeSeries.plot(label='Acutal')
predicted.plot(label='Forecast', linestyle='dashed', marker='.')
plt.legend(loc='upper right')
plt.show()

(_,_,_) = performance(timeSeries.values[-n_forecast:], predicted.values, print_metr
```

```
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
erred frequency D will be used.
  self._init_dates(dates, freq)
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
erred frequency D will be used.
  self._init_dates(dates, freq)
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
erred frequency D will be used.
  self._init_dates(dates, freq)
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\statespace\representation.py:374: FutureWarning: Unknown keyword arguments: dict_k
eys(['alpha']).Passing unknown keyword arguments will raise a TypeError beginning in
version 0.15.
  warnings.warn(msg, FutureWarning)
```




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

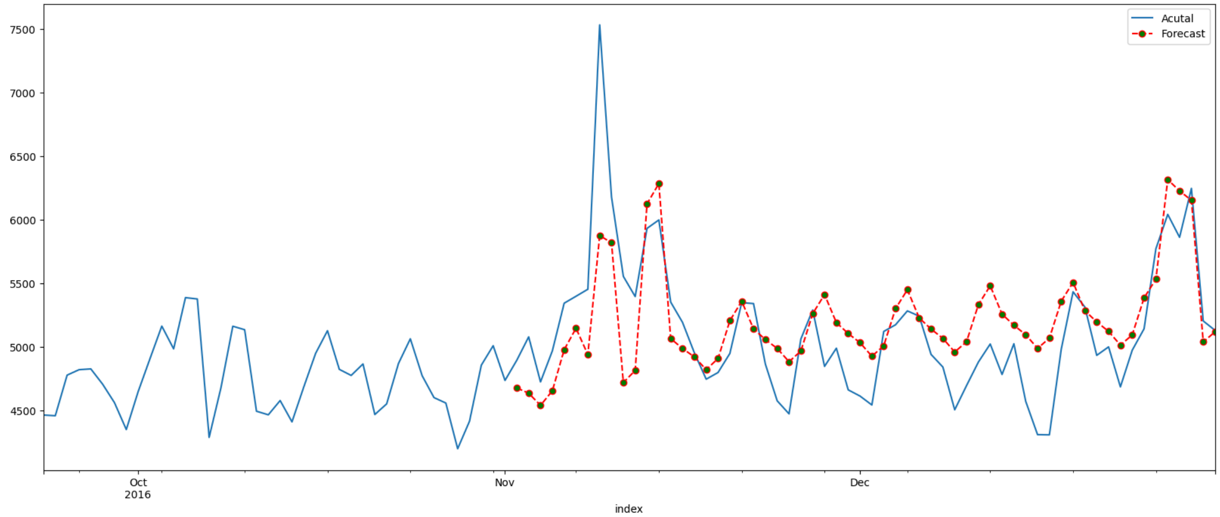
model is not doing good job even for diff comb of p and q

7.2 SARIMAX model

In [69]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [70]:
```
## let's try to include exogenous model
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(timeSeries[:-n_forecast],
                order=(p,d,q),
                seasonal_order=(P, D, Q, S),
                exog= exog[:-n_forecast],
                initialization='approximate_diffuse'
                )
model = model.fit()
moder_forecast = model.forecast(steps=n_forecast, dynamic = True, exog = pd.DataFra

plt.figure(figsize=(20,8))
timeSeries[-100:].plot(label='Acutal')
moder_forecast[-100:].plot(label = 'Forecast', color = 'red', linestyle='dashed', m
plt.legend(loc='upper right')
plt.show()

(_,_,_) = performance(timeSeries.values[-n_forecast:], predicted.values, print_metr
```

```
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
erred frequency D will be used.
  self._init_dates(dates, freq)
c:\Users\FPK1COB\Documents\Learning\TimeSeries\venv\Lib\site-packages\statsmodels\ts
a\base\tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
erred frequency D will be used.
  self._init_dates(dates, freq)
```




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

Observation

- SARIMAX model results are better, we need to do grid search to find the best params

In [71]:
```
def SARIMAX_search(timeSeries, forecast, p_list, d_list, q_list, P_list, D_list, Q_
    counter = 0
    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(timeSeries[:-n_forecast],
                                                    order =(p,d,q),
                                                    seasonal_order=(P, D, Q, s),
                                                    exog = exog[:-n_forecast],
                                                    initialization='approximate_dif
                                                    )
                                    model = model.fit()
                                    model_forecast = model.forecast(n_forecast, dyn
                                    MAE, RMSE, MAPE = performance(timeSeries.values
                                    counter += 1
                                    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 [72]:
```
import warnings
warnings.filterwarnings("ignore")

timeSeries = english_ts.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(timeSeries, n_forecast, p_list, d_list, q_list, P_list, D_
perf_df.sort_values(['mape', 'rmse'])
```

```
Combination 1 out of 16
Combination 2 out of 16
Combination 3 out of 16
Combination 4 out of 16
Combination 5 out of 16
Combination 6 out of 16
Combination 7 out of 16
Combination 8 out of 16
Combination 9 out of 16
Combination 10 out of 16
Combination 11 out of 16
Combination 12 out of 16
Combination 13 out of 16
Combination 14 out of 16
Combination 15 out of 16
Combination 16 out of 16
```

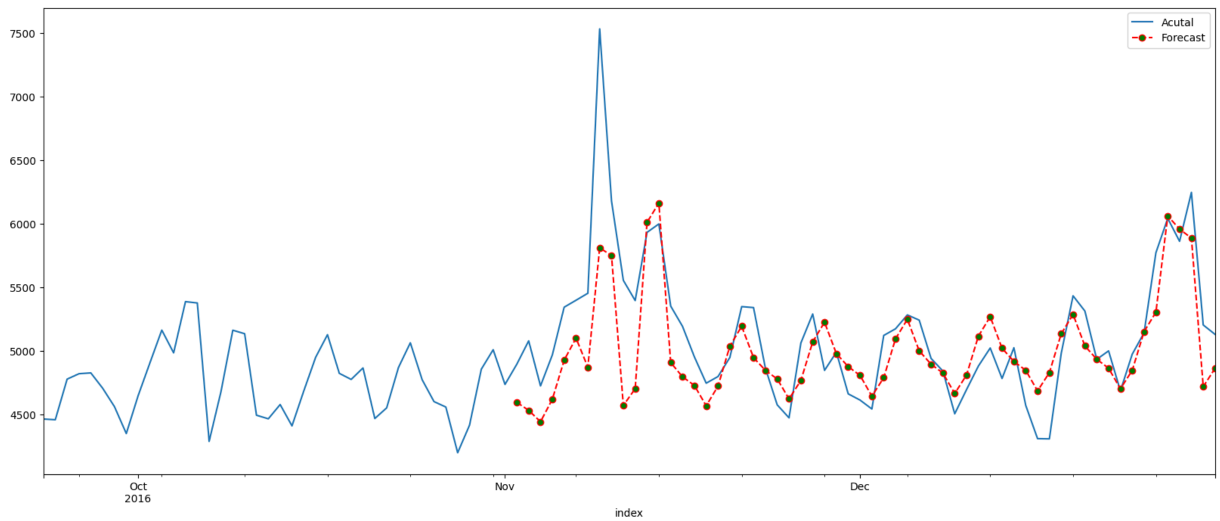
Out[72]:

	serial	pdq	PDQs	mape	rmse
13	14	(1, 1, 1)	(2, 1, 3, 7)	0.051	378.666
12	13	(1, 1, 1)	(2, 1, 2, 7)	0.054	372.375
11	12	(1, 1, 0)	(3, 1, 3, 7)	0.056	411.755
9	10	(1, 1, 0)	(2, 1, 3, 7)	0.056	412.042
15	16	(1, 1, 1)	(3, 1, 3, 7)	0.057	384.258
5	6	(0, 1, 1)	(2, 1, 3, 7)	0.057	416.966
14	15	(1, 1, 1)	(3, 1, 2, 7)	0.059	392.151
7	8	(0, 1, 1)	(3, 1, 3, 7)	0.061	437.273
3	4	(0, 1, 0)	(3, 1, 3, 7)	0.061	437.976
10	11	(1, 1, 0)	(3, 1, 2, 7)	0.062	444.530
6	7	(0, 1, 1)	(3, 1, 2, 7)	0.062	444.976
2	3	(0, 1, 0)	(3, 1, 2, 7)	0.062	447.552
1	2	(0, 1, 0)	(2, 1, 3, 7)	0.063	448.904
4	5	(0, 1, 1)	(2, 1, 2, 7)	0.064	456.425
8	9	(1, 1, 0)	(2, 1, 2, 7)	0.064	456.481
0	1	(0, 1, 0)	(2, 1, 2, 7)	0.064	458.305

p,d,q,P,D,Q,s = 1,1,1,2,1,3,7 gives lowest mape

In [73]:

```
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(timeSeries[:-n_forecast],
                order=(p,d,q),
                seasonal_order=(P, D, Q, S),
                exog= exog[:-n_forecast],
                initialization='approximate_diffuse'
                )
model = model.fit()
moder_forecast = model.forecast(steps=n_forecast, dynamic = True, exog = pd.DataFra

plt.figure(figsize=(20,8))
timeSeries[-100:].plot(label='Acutal')
moder_forecast[-100:].plot(label = 'Forecast', color = 'red', linestyle='dashed', m
plt.legend(loc='upper right')
plt.show()

(_,_,_) = performance(timeSeries.values[-n_forecast:], moder_forecast.values, print
```




```
MAE : 269.539
RMSE : 378.666
MAPE: 0.051
```

Observation

- SARIMAX model has shown best results after tuning the parameters

7.4 Facebook Prophet

In [74]:
```
# Install required dependencies for Prophet
# %pip install cython
# %pip install prophet
```

In [75]:
```
timeSeries = english_ts.copy(deep=True).reset_index()
timeSeries = timeSeries[['index', 'English']]
timeSeries.columns = ['ds', 'y']
timeSeries['ds'] = pd.to_datetime(timeSeries['ds'])
exog = exog_en['Exog']
timeSeries['exog'] = exog.values
timeSeries.tail()
```

Out[75]:

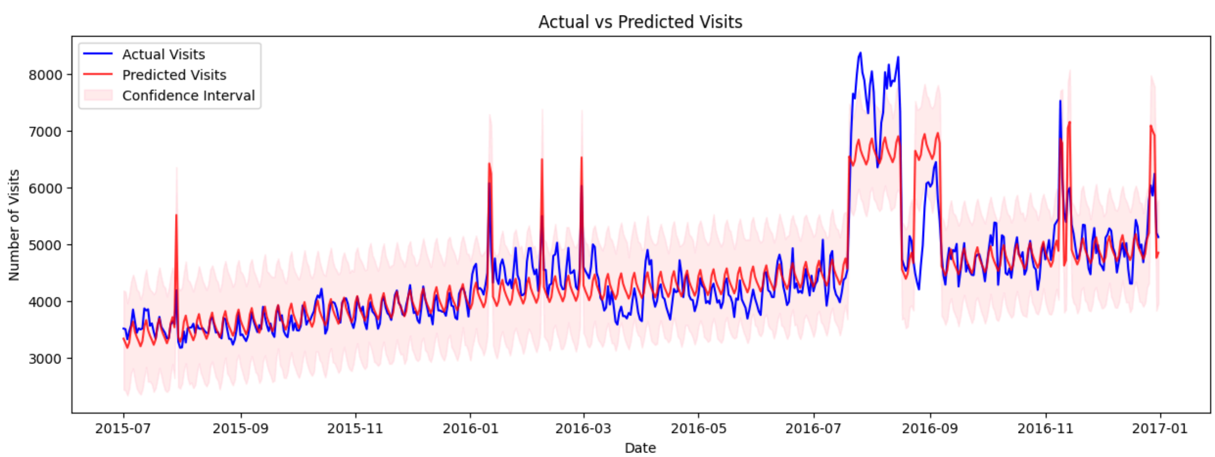
	ds	y	exog
545	2016-12-27	6040.680728	1
546	2016-12-28	5860.227559	1
547	2016-12-29	6245.127510	1
548	2016-12-30	5201.783018	0
549	2016-12-31	5127.916418	0

In [76]:
```
from prophet import Prophet
model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.add_regressor('exog')
n_forecast = 60
```

```
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast_dates['exog'] = timeSeries['exog']
forecast = model.predict(forecast_dates)


timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']

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

```
Importing plotly failed. Interactive plots will not work.
14:45:22 - cmdstanpy - INFO - Chain [1] start processing
14:45:22 - cmdstanpy - INFO - Chain [1] done processing
MAE : 287.499
RMSE : 441.92
MAPE: 0.06
```

In [77]:
```
plt.figure(figsize=(15, 5))
plt.plot(timeSeries['ds'], timeSeries['y'], label='Actual Visits', color='blue')
plt.plot(timeSeries['ds'], timeSeries['yhat'], label='Predicted Visits', color='red
plt.fill_between(timeSeries['ds'], timeSeries['yhat_lower'], timeSeries['yhat_upper

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



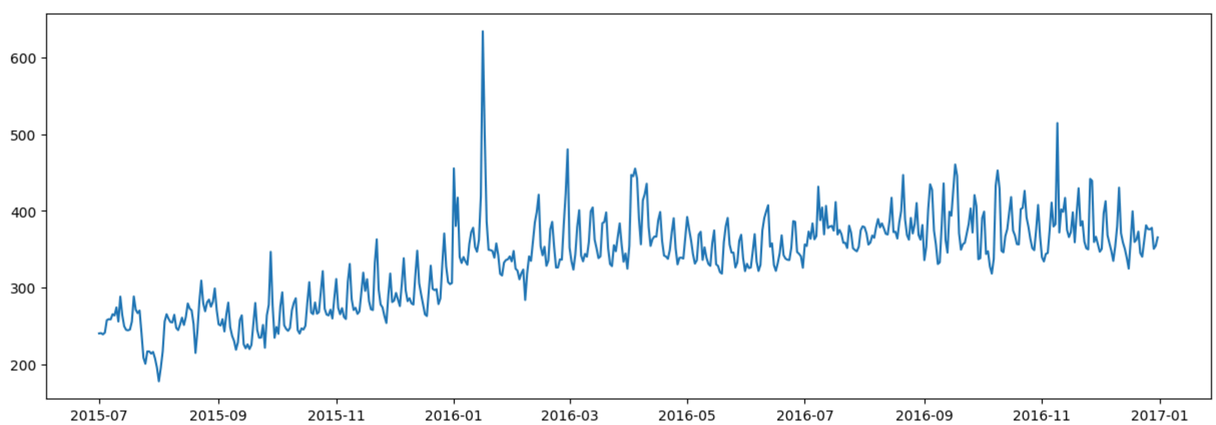
Observation

- Prophet capturing more efficiently trend and unusual peak
- Even seasonality capturing is very well

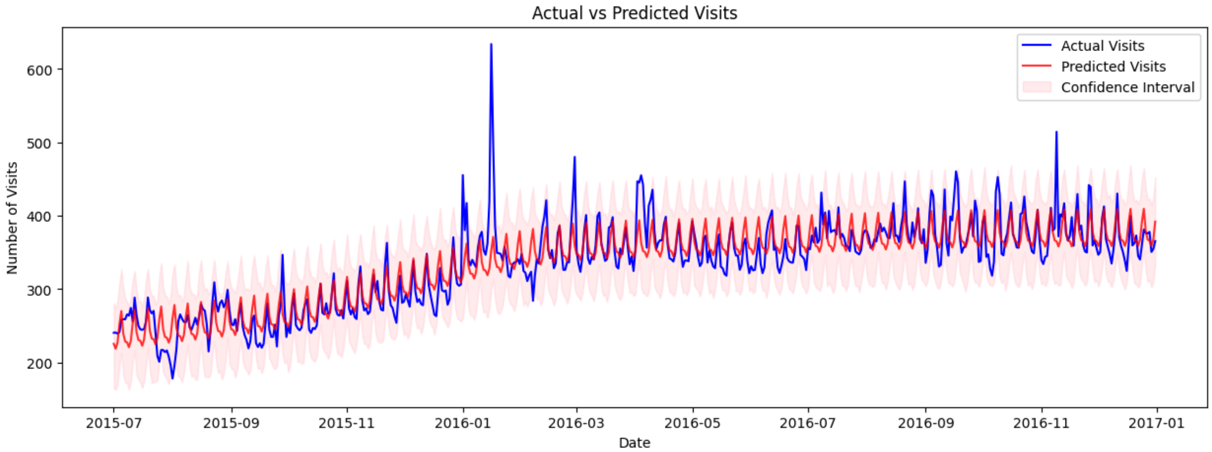
7.5 Other Langugages

7.5.1 Chinese

In [78]:
```
timeSeries = df_agg['Chinese'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

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

model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']

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

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

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

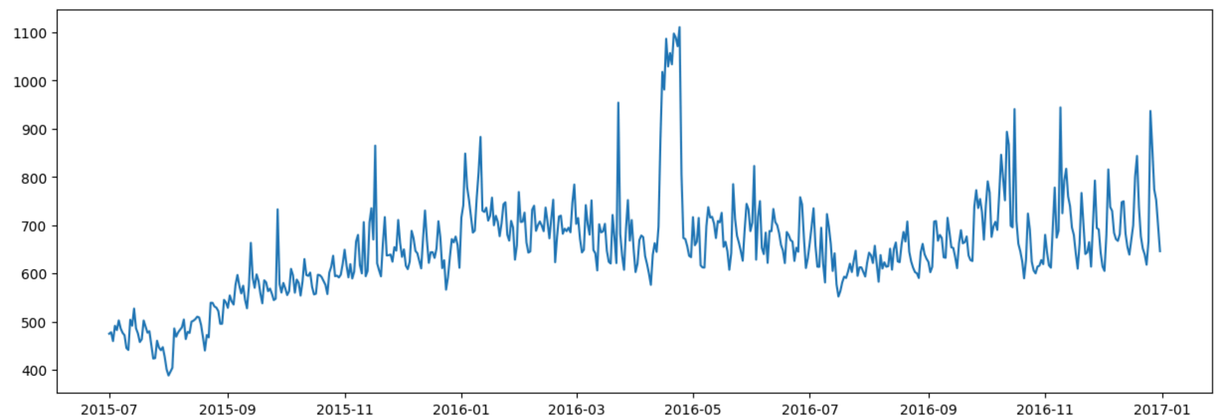



```
14:45:23 - cmdstanpy - INFO - Chain [1] start processing
14:45:23 - cmdstanpy - INFO - Chain [1] done processing
MAE : 19.352
RMSE : 28.702
MAPE: 0.058
```

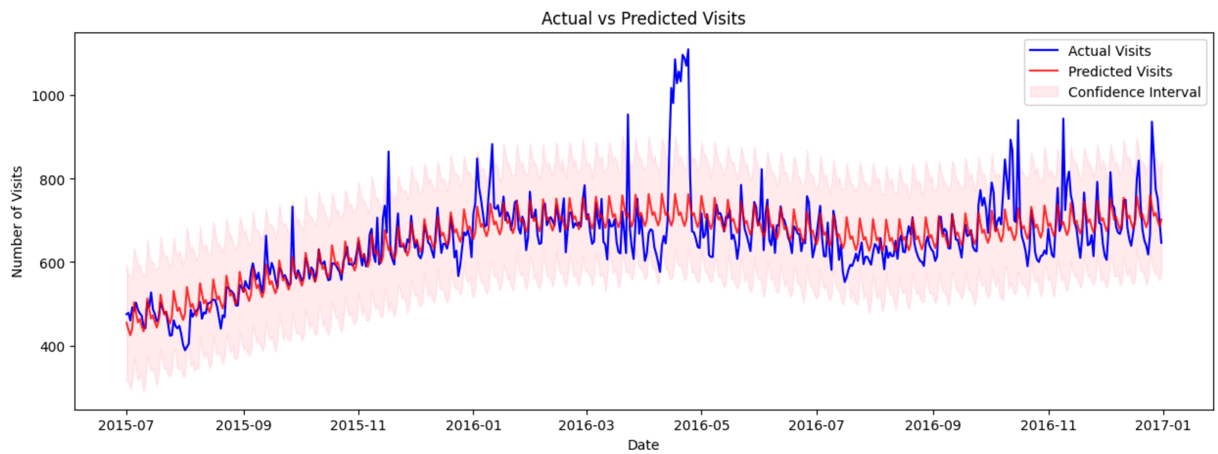


7.5.2 French

In [79]:
```
timeSeries = df_agg['French'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

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

model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']

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

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

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



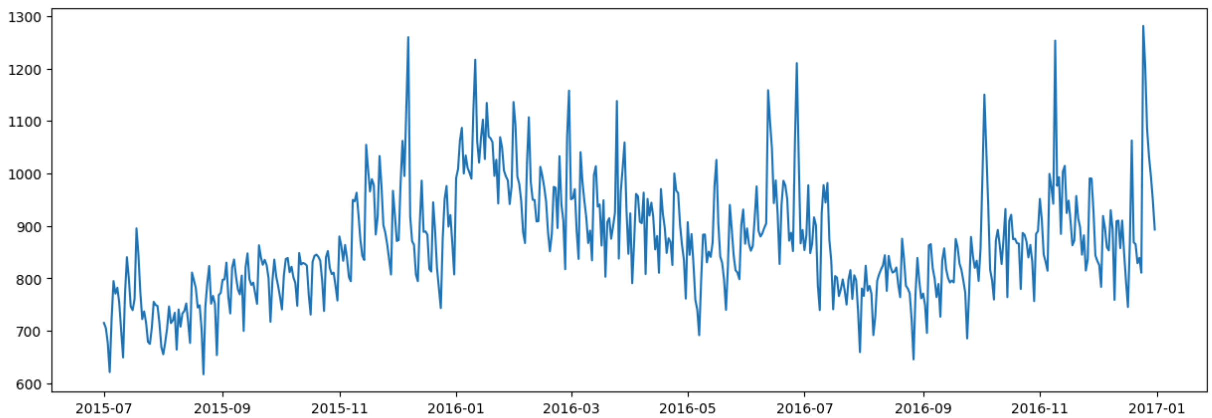
```
14:45:23 - cmdstanpy - INFO - Chain [1] start processing
14:45:23 - cmdstanpy - INFO - Chain [1] done processing
MAE : 42.004
RMSE : 68.664
MAPE: 0.061
```



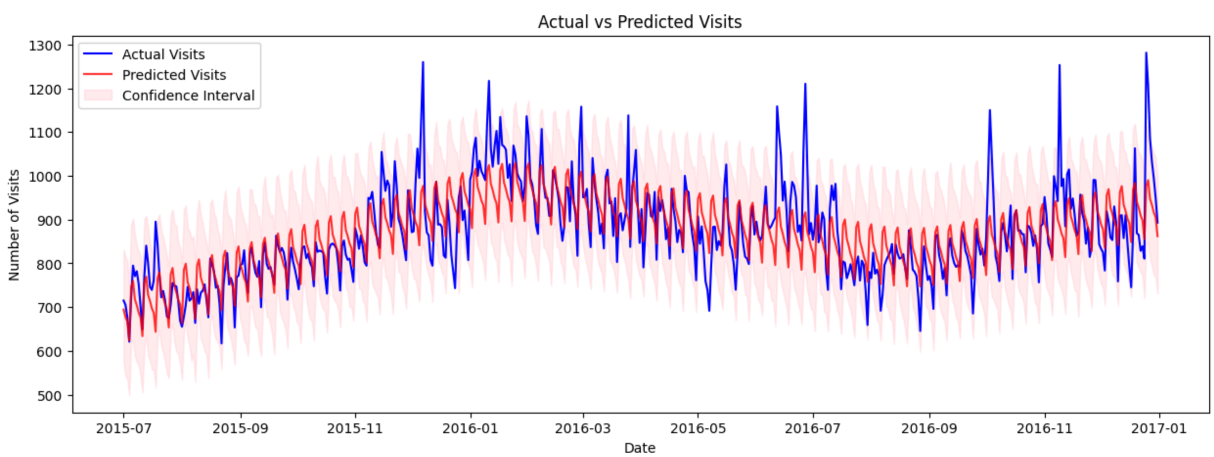
7.5.3 German

In [80]:
```python
timeSeries = df_agg['German'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

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

model  = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']
```

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

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

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



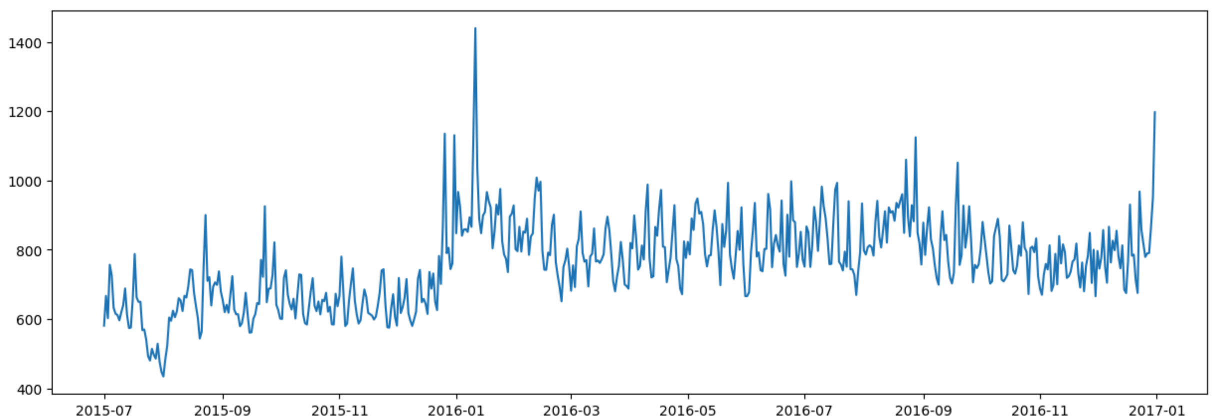
```
14:45:24 - cmdstanpy - INFO - Chain [1] start processing
14:45:24 - cmdstanpy - INFO - Chain [1] done processing
MAE : 49.367
RMSE : 68.284
MAPE: 0.055
```



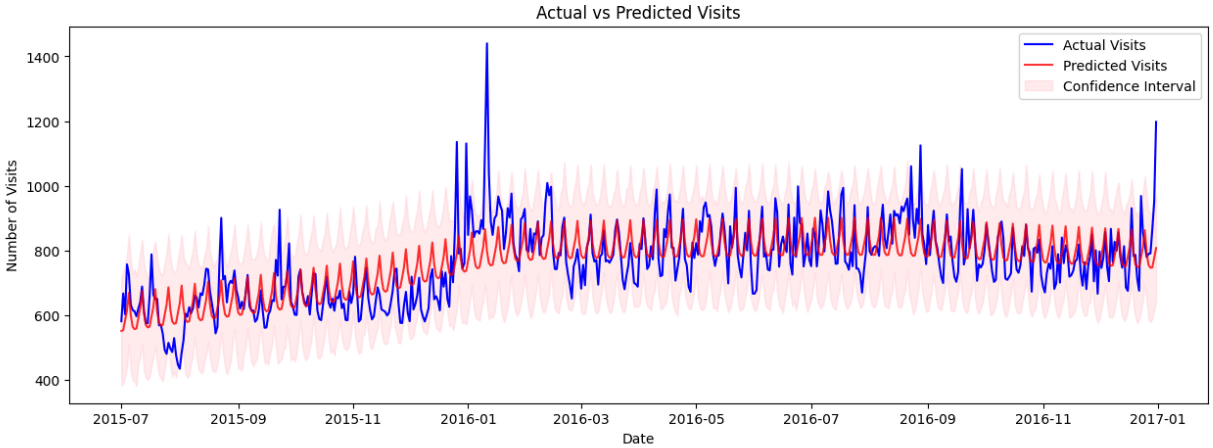
7.5.4 Japanese

In [81]:
```
timeSeries = df_agg['Japanese'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

timeSeries = timeSeries.reset_index()
timeSeries = timeSeries[['index', 'Japanese']]
```

```
timeSeries.columns = ['ds', 'y']
timeSeries['ds'] = pd.to_datetime(timeSeries['ds'])
timeSeries.tail()

model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']

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

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

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

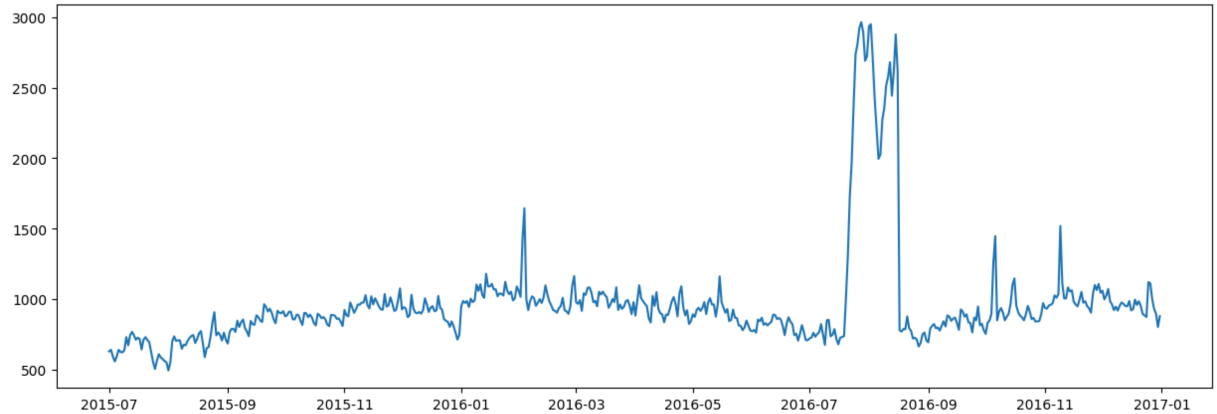



```
14:45:24 - cmdstanpy - INFO - Chain [1] start processing
14:45:24 - cmdstanpy - INFO - Chain [1] done processing
MAE : 61.17
RMSE : 84.08
MAPE: 0.08
```

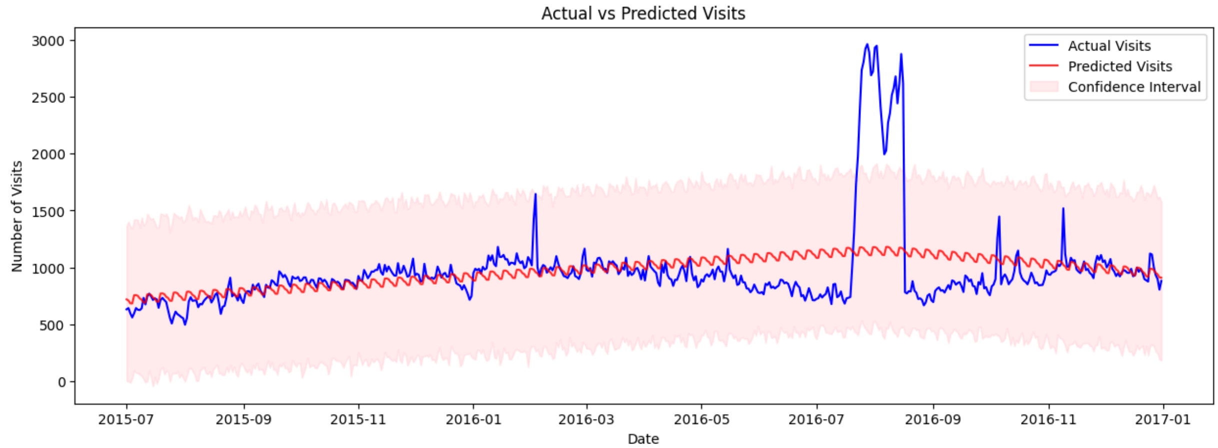


7.5.5 Russian

In [82]:
```
timeSeries = df_agg['Russian'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

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

model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']

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

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

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



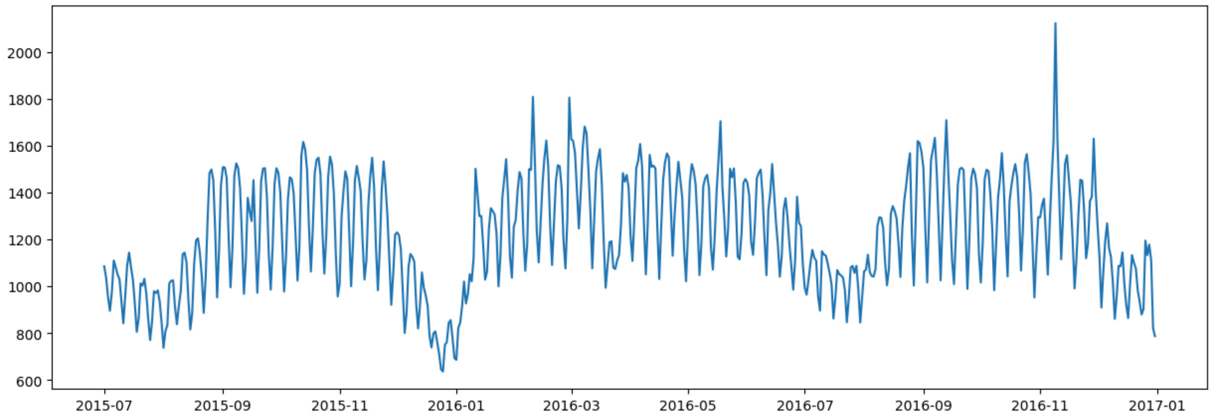

```
14:45:25 - cmdstanpy - INFO - Chain [1] start processing
14:45:25 - cmdstanpy - INFO - Chain [1] done processing
MAE : 185.548
RMSE : 353.401
MAPE: 0.169
```



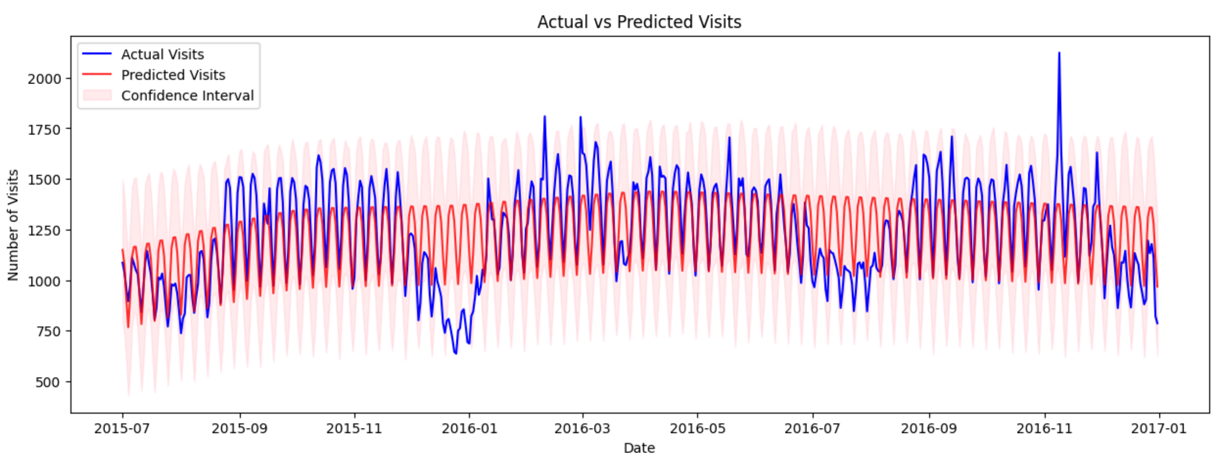
7.5.6 Spanish

In [83]:
```
timeSeries = df_agg['Spanish'].copy(deep=True)
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(timeSeries.index, timeSeries)
plt.show()

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

model = Prophet(interval_width=0.95, weekly_seasonality=True)
model.fit(timeSeries)
forecast_dates = model.make_future_dataframe(periods=0)
forecast = model.predict(forecast_dates)

timeSeries['yhat'] = forecast['yhat']
timeSeries['yhat_upper'] = forecast['yhat_upper']
timeSeries['yhat_lower'] = forecast['yhat_lower']
```

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

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

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




```
14:45:25 - cmdstanpy - INFO - Chain [1] start processing
14:45:25 - cmdstanpy - INFO - Chain [1] done processing
MAE : 131.112
RMSE : 170.643
MAPE: 0.115
```



In [ ]: