In [ ]:

```

```

Problem Statement :

Ad Ease 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. The interplay of 3 AI modules - Design, Dispense, and Decipher, come together to make it this an end-to-end 3 step process digital advertising solution for all.

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.

Data Dictionary:

- There are two csv files given
- **train_1.csv:**
  - In the csv file, each row corresponds to a particular article and each column corresponds to a particular date. The values are the number of visits on that date.
  - The page name contains data in this format:

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

  - having information about the page name, the main domain, the device type used to access the page, and also the request origin(spider or browser agent)
- **Exog_Campaign_eng:**
  - This file contains data for the dates which had a campaign or significant event that could affect the views for that day. The data is just for pages in English.
  - There's 1 for dates with campaigns and 0 for remaining dates.
  - It is to be treated as an exogenous variable for models when training and forecasting data for pages in English

In [ ]:

```

```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('display.max_rows', 5000)
pd.set_option('display.max_columns', 5000)
pd.set_option('display.width', 1000)
pd.options.display.max_colwidth = 1000
sns.set(style = 'darkgrid')
```

In [ ]:

```

```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- Importing the dataset and doing usual exploratory analysis steps like checking the structure & characteristics of the dataset

In [ ]:

```
df = pd.read_csv("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Exploration-/ad_e
```

In [ ]:

```
df.shape
```

Out[4]:

```
(145063, 551)
```

In [ ]:

```
Exog_Campaign_eng = pd.read_csv("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Ex
```

In [ ]:

```
Exog_Campaign_eng.shape
```

Out[6]:

```
(550, 1)
```

In [ ]:

```
df.Page.sample(20)
```

Out[7]:

```
35416                List_of_people_who_died_by_hanging_en.wikipedia.org_all-access_spider
130531                                 Unibail-Rodamco_fr.wikipedia.org_all-access_spider
31928                                      仙剑云之凡_zh.wikipedia.org_all-access_all-agents
126973                        Бердыев,_Курбан_Бекиевич_ru.wikipedia.org_all-access_spider
27096                                      Théâtre_fr.wikipedia.org_all-access_all-agents
3525                                     王凯_(大陆演员)_zh.wikipedia.org_all-access_spider
98224                 Зейналова,_Ирада_Автандиловна_ru.wikipedia.org_all-access_all-agents
108061                                       酷玩樂團_zh.wikipedia.org_mobile-web_all-agents
29493                    臺灣對東日本大震災之援助及各界反應_zh.wikipedia.org_all-access_all-agents
101437                              Quest_Pistols_Show_ru.wikipedia.org_desktop_all-agents
98903                                   Фелпс,_Майкл_ru.wikipedia.org_all-access_all-agents
118281                        Französische_Revolution_de.wikipedia.org_mobile-web_all-agents
26720                                  Aïd_el-Fitr_fr.wikipedia.org_all-access_all-agents
68640                                     Pets_(2016)_de.wikipedia.org_desktop_all-agents
70411                                       Naturaleza_es.wikipedia.org_desktop_all-agents
8837                              Colony_(TV_series)_en.wikipedia.org_desktop_all-agents
135070                                          羽田圭介_ja.wikipedia.org_all-access_spider
123675                              A_LIFE～愛しき人～_ja.wikipedia.org_all-access_all-agents
136151                                        新宿スワン_ja.wikipedia.org_all-access_spider
82066     File:Bahnhof_VIE_-_Zugang_Ost_2014.JPG_commons.wikimedia.org_desktop_all-agents
Name: Page, dtype: object
```

In [ ]:

```
df.Page.str.split("_").apply(lambda x:x[3]).head(20)
```

Out[8]:

```
0               spider
1               spider
2               spider
3               spider
4                 Love
5               spider
6               spider
7               spider
8               spider
9               spider
10              spider
11    zh.wikipedia.org
12                 are
13              spider
14              spider
15              spider
16              spider
17          all-access
18          all-access
19              spider
Name: Page, dtype: object
```

In [ ]:

```

```

In [ ]:

```
data = df.copy()
```

In [ ]:

```
data.duplicated().sum()
# No duplicate data
```

Out[10]:

```
0
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
# data.sample(100).head(10)
```

In [ ]:

```
data.dtypes.sample(10)
```

Out[12]:

```
2016-08-22    float64
2016-03-03    float64
2016-10-04    float64
2016-05-28    float64
2016-08-18    float64
2015-11-19    float64
2016-12-14    float64
2015-11-25    float64
2016-12-18    float64
2016-05-27    float64
dtype: object
```

In [ ]:

```
indexes = data.head(2).columns[1:][range(0,549,20)].values
indexes
```

Out[13]:

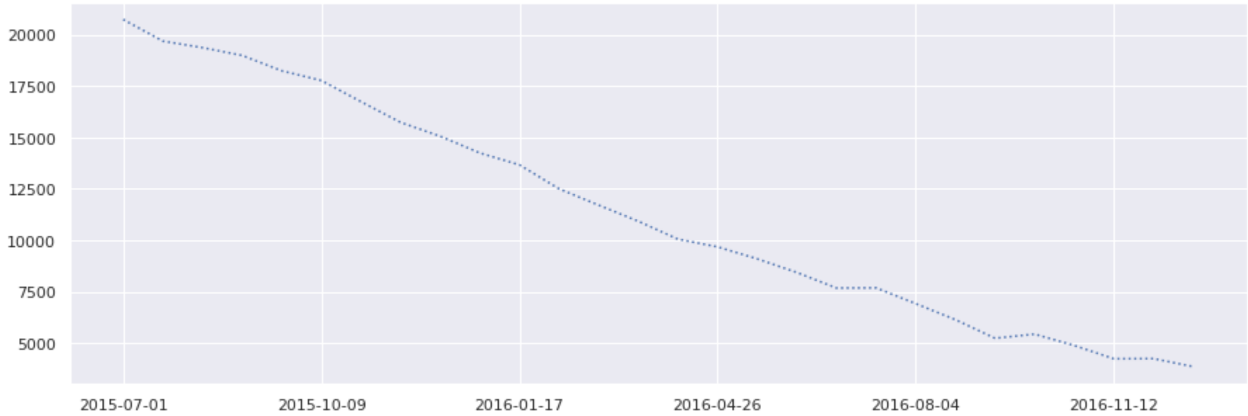
```
array(['2015-07-01', '2015-07-21', '2015-08-10', '2015-08-30',
       '2015-09-19', '2015-10-09', '2015-10-29', '2015-11-18',
       '2015-12-08', '2015-12-28', '2016-01-17', '2016-02-06',
       '2016-02-26', '2016-03-17', '2016-04-06', '2016-04-26',
       '2016-05-16', '2016-06-05', '2016-06-25', '2016-07-15',
       '2016-08-04', '2016-08-24', '2016-09-13', '2016-10-03',
       '2016-10-23', '2016-11-12', '2016-12-02', '2016-12-22'],
      dtype=object)
```

In [ ]:

```
plt.figure(figsize=(15, 5))

data.isna().sum()[indexes].plot(linestyle='dotted')
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a548638b0>
```



- from above plot , we can observe that with time , null values are decreasing.
- recent dates have lesser null values
- that means newer pages will have no data of prior to that page hosting date.

In [ ]:

```

```

In [ ]:

```
# replacing all the null values with 0.
```

In [ ]:

```
data.fillna(0,inplace =True)
```

In [ ]:

```
data.isnull().sum()[indexes]
```

Out[17]:

```
2015-07-01    0
2015-07-21    0
2015-08-10    0
2015-08-30    0
2015-09-19    0
2015-10-09    0
2015-10-29    0
2015-11-18    0
2015-12-08    0
2015-12-28    0
2016-01-17    0
2016-02-06    0
2016-02-26    0
2016-03-17    0
2016-04-06    0
2016-04-26    0
2016-05-16    0
2016-06-05    0
2016-06-25    0
2016-07-15    0
2016-08-04    0
2016-08-24    0
2016-09-13    0
2016-10-03    0
2016-10-23    0
2016-11-12    0
2016-12-02    0
2016-12-22    0
dtype: int64
```

In [ ]:

In [ ]:

Exploratory Analysis :

In [ ]:

```
# Extracting Language , access type and access origin
```

```
The page name contains data in this format:

SPECIFICNAME_LANGUAGE.wikipedia.org_ACCESS TYPE_ ACCESS ORIGIN
```

Extracting Language

In [ ]:

```
data.Page[0]
```

Out[19]:

```
'2NE1_zh.wikipedia.org_all-access_spider'
```

In [ ]:

```
import re
re.findall(r'_(.{2}).wikipedia.org_', "2NE1_zh.wikipedia.org_all-access_spider")
```

Out[20]:

```
['zh']
```

In [ ]:

```
data.Page.str.findall(pat="_(.{2}).wikipedia.org_").sample(10)
```

Out[21]:

```
76731     [en]
109746    [en]
121386    [ja]
63880     [zh]
132788    [ja]
80305       []
37467     [en]
131419    [fr]
102592    [ru]
126060    [ru]
Name: Page, dtype: object
```

In [ ]:

```
# extracting language
def Extract_Language(name):
  if len(re.findall(r'_(.{2}).wikipedia.org_', name)) == 1 :
    return re.findall(r'_(.{2}).wikipedia.org_', name)[0]
  else:
    return 'Unknown'
```

In [ ]:

```
data["Language"] = data["Page"].map(Extract_Language)
```

In [ ]:

```
data["Language"].unique()
```

Out[24]:

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

[https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)](https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)

In [ ]:

```
dict_ ={'de':'German',
        'en':'English',
        'es': 'Spanish',
        'fr': 'French',
        'ja': 'Japenese' ,
        'ru': 'Russian',
        'zh': 'Chinese',
        'Unknown': 'Unknown_Language'}

data["Language"] = data["Language"].map(dict_)
```

In [ ]:

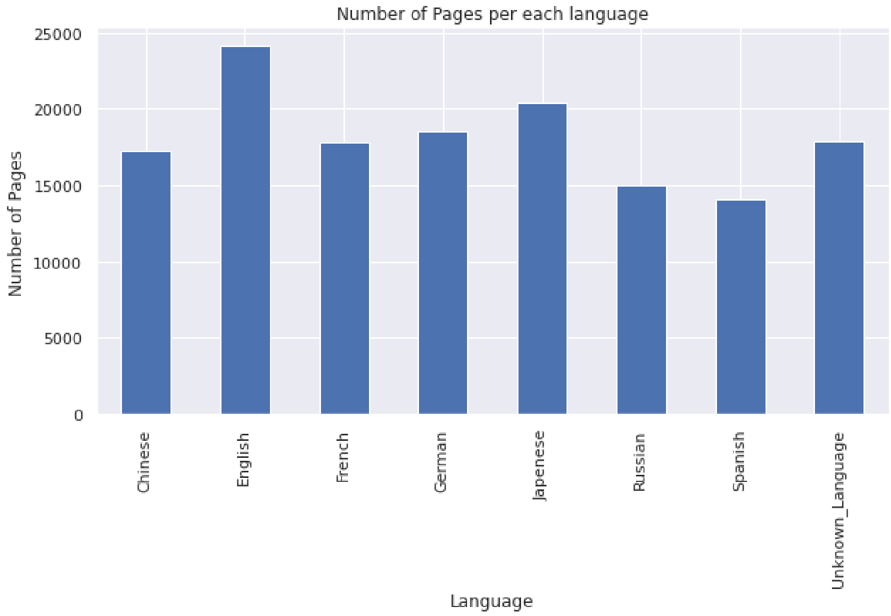
```
data.head()
```

Out[30]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	2015-07-11	2015-07-12	2015-07-13	2015-07-14	2015-07-15
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	19.0	10.0	14.0	15.0	8.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	41.0	65.0	57.0	38.0	20.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	1.0	1.0	1.0	6.0	8.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	16.0	11.0	23.0	145.0	14.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [ ]:

```
plt.figure(figsize=(10, 5))

data.groupby("Language")["Page"].count().plot(kind="bar")
plt.xlabel("Language")
plt.ylabel("Number of Pages")
plt.title("Number of Pages per each language")
plt.show()
```



In [ ]:

```
from locale import normalize
data["Language"].value_counts(normalize=True) * 100
```

Out[31]:

```
English             16.618986
Japenese            14.084225
German              12.785479
Unknown_Language    12.308445
French              12.271909
Chinese             11.876909
Russian             10.355501
Spanish              9.698545
Name: Language, dtype: float64
```

In [ ]:

```
# 12.30 % of pages have unknown language.

# 16.61% of all pages are in English which is highest.
```

Exrtacting ACCESS TYPE :

SPECIFICNAME_LANGUAGE.wikipedia.org_ACCESS TYPE_ ACCESS ORIGIN

In [ ]:

```
# df.Page.sample(20)
```

In [ ]:

```
data["Access_Type"] = data.Page.str.findall(r'all-access|mobile-web|desktop').apply(lambda x:x[0])
```

In [ ]:

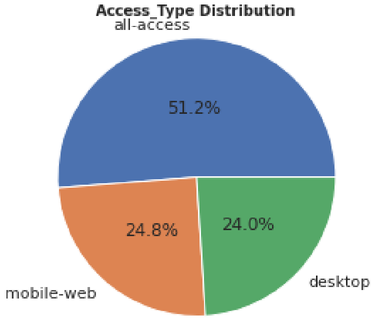
```
data["Access_Type"].value_counts(dropna=False, normalize=True)
```

Out[35]:

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

In [ ]:

```
x = (data["Access_Type"].value_counts(dropna= False, normalize=True) * 100).values
y = (data["Access_Type"].value_counts(dropna= False, normalize=True) * 100).index

plt.pie(x,labels= y,radius=1.5,  autopct='%1.1f%%', pctdistance=0.5 )
plt.title(f'Access_Type Distribution', fontsize = 10, fontweight = 'bold')
plt.axis('equal')
plt.show()
```



In [ ]:

```

```

Exrtacting ACCESS ORIGIN :

```
SPECIFICNAME_LANGUAGE.wikipedia.org_ACCESS TYPE_ ACCESS ORIGIN
```

In [ ]:

```
data.Page.sample(20)
```

Out[37]:

```
62057                                           鯉魚王_zh.wikipedia.org_desktop_all-agents
67479                                 Ernest_Hemingway_de.wikipedia.org_desktop_all-agents
111671                               Kira_Walkenhorst_en.wikipedia.org_all-access_all-agents
130008                                    James_J._Bulger_fr.wikipedia.org_all-access_spider
73297                                Death_of_Harambe_en.wikipedia.org_mobile-web_all-agents
57012                                 ピーター・ノーマン_ja.wikipedia.org_mobile-web_all-agents
100530                                        Каперсы_ru.wikipedia.org_all-access_all-agents
61069                                           反式脂肪_zh.wikipedia.org_desktop_all-agents
121339                      闇金ウシジマくん_(テレビドラマ)_ja.wikipedia.org_all-access_all-agents
78645                     File:PliosaurusDB12.jpg_commons.wikimedia.org_mobile-web_all-agents
96251             Selección_de_básquetbol_de_Argentina_es.wikipedia.org_mobile-web_all-agents
111093                                    HIP_85605_en.wikipedia.org_all-access_all-agents
28725                                  2012年中華民國總統選舉_zh.wikipedia.org_all-access_all-agents
107604                                          柯以敏_zh.wikipedia.org_mobile-web_all-agents
132204                              魔法少女リリカルなのはViVid_ja.wikipedia.org_all-access_spider
48113                                     Rudolf_Wessely_de.wikipedia.org_all-access_spider
78739      File:Small_bodies_of_the_Solar_System.jpg_commons.wikimedia.org_mobile-web_all-agents
28676                                           胡國興_zh.wikipedia.org_all-access_all-agents
40664                              Sherlock_(TV_series)_en.wikipedia.org_all-access_all-agents
101305                            ????:Andrey_Belloly_1.jpg_ru.wikipedia.org_desktop_all-agents
Name: Page, dtype: object
```

In [ ]:

```
data.Page.str.findall(r'spider|agents').apply(lambda x:x[0]).isna().sum()
```

Out[38]:

```
0
```

In [ ]:

```
data["Access_Origin"] =  data.Page.str.findall(r'spider|agents').apply(lambda x:x[0])
```

In [ ]:

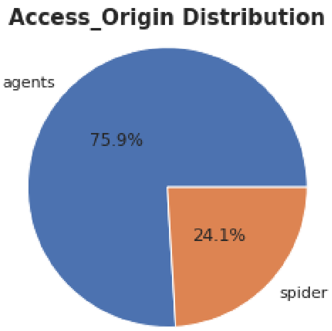
```
data["Access_Origin"].value_counts(dropna= False, normalize=True) * 100
```

Out[40]:

```
agents    75.932526
spider    24.067474
Name: Access_Origin, dtype: float64
```

In [ ]:

```
x = (data["Access_Origin"].value_counts(dropna= False, normalize=True) * 100).values
y = (data["Access_Origin"].value_counts(dropna= False, normalize=True) * 100).index

plt.pie(x,labels= y,radius=1.5,  autopct='%1.1f%%', pctdistance=0.5 )
plt.title(f'Access_Origin Distribution', fontsize = 15, fontweight = 'bold')
plt.axis('equal')
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

```
data
```

Out[42]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	20 07-
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	2
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	1
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
145058	Underworld_(serie_de_películas)_es.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
145060	Enamorándome_de_Ramón_es.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
145061	Hasta_el_último_hombre_es.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
145062	Francisco_el_matemático_(serie_de_televisión_de_2017)_es.wikipedia.org_all-access_spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

145063 rows × 554 columns

In [ ]:

```
data.groupby("Language").mean()
```

Out[43]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2(
Language										
Chinese	240.582042	240.941958	239.344071	241.653491	257.779674	259.114864	258.832260	265.589529	263.964420	27
English	3513.862203	3502.511407	3325.357889	3462.054256	3575.520035	3849.736021	3643.523063	3437.871080	3517.459391	349
French	475.150994	478.202000	459.837659	491.508932	482.557746	502.741209	485.945399	476.998820	472.061903	44
German	714.968405	705.229741	676.877231	621.145145	722.076185	794.832480	770.814256	782.077641	752.939990	70
Japenese	580.647056	666.672801	602.289805	756.509177	725.720914	632.399148	615.184181	611.462337	596.067642	61
Russian	629.999601	640.902876	594.026295	558.728132	595.029157	640.986287	626.293436	623.360205	638.550726	73
Spanish	1085.972919	1037.814557	954.412680	896.050750	974.508210	1110.637145	1082.568342	1050.669557	1030.841282	93
Unknown_Language	83.479922	87.471857	82.680538	70.572557	78.214562	89.720190	94.939457	99.096724	86.445477	8

In [ ]:

```
pd.set_option('display.max_rows', 500)
```

In [ ]:

```
aggregated_data = data.groupby("Language").mean().T.drop("Unknown_Language",axis = 1).reset_index()
```

In [ ]:

```
aggregated_data["index"] = pd.to_datetime(aggregated_data["index"])
aggregated_data = aggregated_data.set_index("index")
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
aggregated_data
```

Out[47]:

Language	Chinese	English	French	German	Japenese	Russian	Spanish
index							
2015-07-01	240.582042	3513.862203	475.150994	714.968405	580.647056	629.999601	1085.972919
2015-07-02	240.941958	3502.511407	478.202000	705.229741	666.672801	640.902876	1037.814557
2015-07-03	239.344071	3325.357889	459.837659	676.877231	602.289805	594.026295	954.412680
2015-07-04	241.653491	3462.054256	491.508932	621.145145	756.509177	558.728132	896.050750
2015-07-05	257.779674	3575.520035	482.557746	722.076185	725.720914	595.029157	974.508210
...	...	...	...	...	...	...	...
2016-12-27	376.019618	6040.680728	858.413100	1085.095379	789.158680	1001.209426	1133.367901
2016-12-28	378.048639	5860.227559	774.155769	1032.640804	790.500465	931.987685	1178.290923
2016-12-29	350.719427	6245.127510	752.712954	994.657141	865.483236	897.282452	1112.171085
2016-12-30	354.704452	5201.783018	700.543422	949.265649	952.018354	803.271868	821.671405
2016-12-31	365.579256	5127.916418	646.258342	893.013425	1197.239440	880.244508	787.399531

550 rows × 7 columns

In [ ]:

```
# import matplotlib.pyplot as plt
# plt.rcParams['figure.figsize'] = (20, 6)
```

In [ ]:

```
aggregated_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 550 entries, 2015-07-01 to 2016-12-31
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Chinese   550 non-null    float64
 1   English   550 non-null    float64
 2   French    550 non-null    float64
 3   German    550 non-null    float64
 4   Japenese  550 non-null    float64
 5   Russian   550 non-null    float64
 6   Spanish   550 non-null    float64
dtypes: float64(7)
memory usage: 34.4 KB
```

In [ ]:

```
aggregated_data.index
```

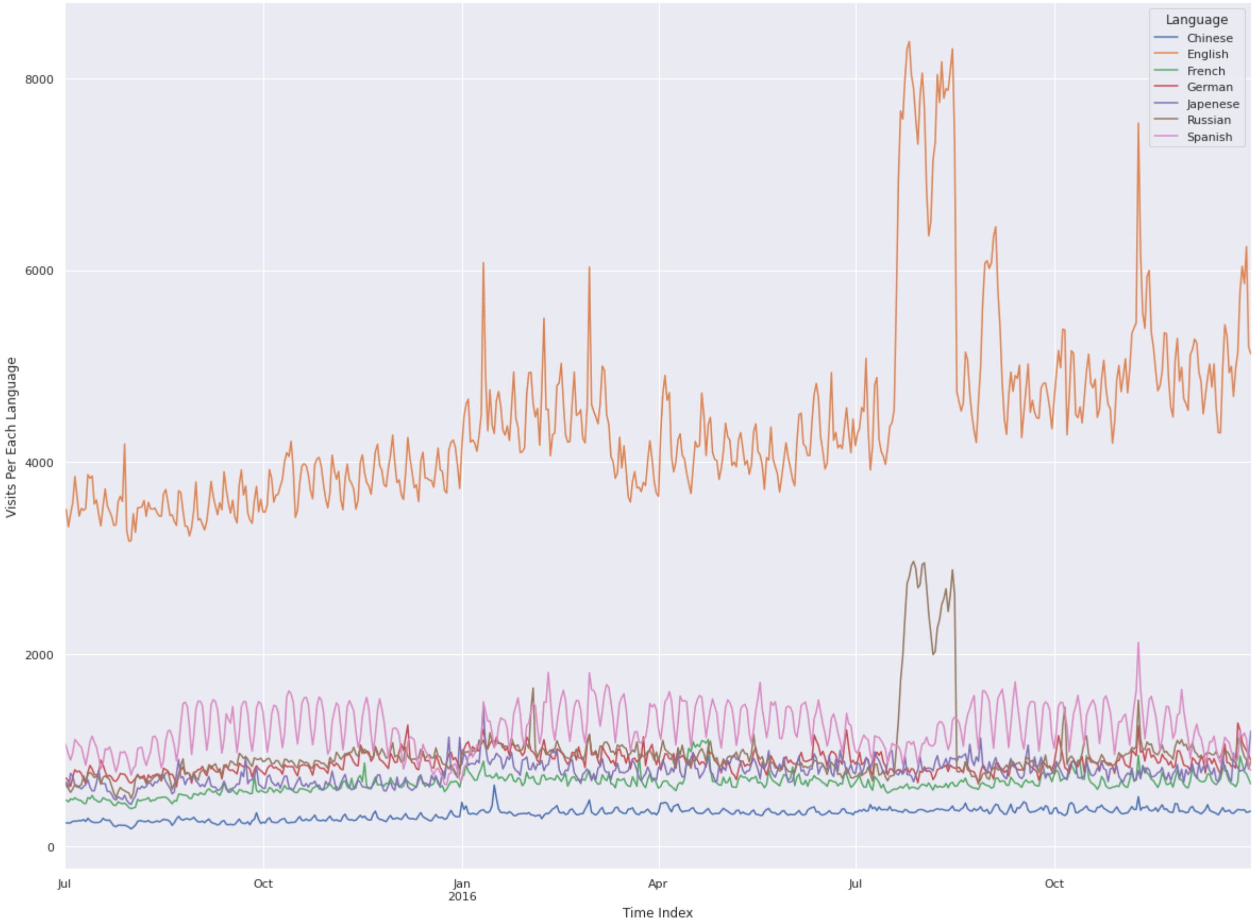
Out[50]:

```
DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06', '2015-07-07', '2
015-07-08', '2015-07-09', '2015-07-10',
               ...
               '2016-12-22', '2016-12-23', '2016-12-24', '2016-12-25', '2016-12-26', '2016-12-27', '2016-12-28', '2
016-12-29', '2016-12-30', '2016-12-31'], dtype='datetime64[ns]', name='index', length=550, freq=None)
```

Visualising Time Series for each languages:

In [ ]:

```
plt.rcParams['figure.figsize'] = (20, 15)

aggregated_data.plot()

plt.xlabel("Time Index")
plt.ylabel("Visits Per Each Language")
plt.show()
```



In [ ]:

```

```

Hypothesis Testing : if Time Series is Stationary or Trending :

- Null Hypothesis: The series is Non-Stationary
- Alternative Hypothesis: The series is Stationary
- significant value : 0.05 (alpha)
- if p-value > 0.05 : we failed to reject Null hypothesis:
  - That means the series is Non-Stationart
- if p-value <= 0.05: we reject Null Hypothesis
  - that means the time series in Stationary

In [ ]:

```
import statsmodels.api as sm
```

In [ ]:

```
def Dickey_Fuller_test(ts,significances_level = 0.05):
    p_value = sm.tsa.stattools.adfuller(ts)[1]
    if p_value <= significances_level:
        print("Time Series is Stationary")
    else:
        print("Time Series is NOT Stationary")
    print("P_value is: ", p_value)
```

In [ ]:

```
for Language in aggregated_data.columns:
  print(Language)
  print(Dickey_Fuller_test(aggregated_data[Language],significances_level = 0.05))
  print()
  print()
```

```
Chinese
Time Series is NOT Stationary
P_value is:  0.447445792293113
None


English
Time Series is NOT Stationary
P_value is:  0.18953359279992366
None


French
Time Series is NOT Stationary
P_value is:  0.05149502195245795
None


German
Time Series is NOT Stationary
P_value is:  0.14097382319729518
None


Japenese
Time Series is NOT Stationary
P_value is:  0.10257133898557613
None


Russian
Time Series is Stationary
P_value is:  0.0018649376536617886
None


Spanish
Time Series is Stationary
P_value is:  0.033588590844779084
None
```

- Based on DickeyFuller test of Stationarity , we can observe Spanish and Russian languages Pages visits Time series are stationary.
- Chinese, English , German , Japanese and French are not stationary.

In [ ]:

```

```

In [ ]:

```
# Further analysing Time Series for English Language Pages Visits :
```

In [ ]:

```
TS_English = aggregated_data.English
```

In [ ]:

```
def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')

    dftest = sm.tsa.stattools.adfuller(timeseries, autolag='AIC')
    df_output = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key, value in dftest[4].items():
       df_output['Critical Value (%s)' %key] = value
    print (df_output)
```

In [ ]:

```
adf_test(TS_English)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.247284
p-value                          0.189534
#Lags Used                      14.000000
Number of Observations Used    535.000000
Critical Value (1%)             -3.442632
Critical Value (5%)             -2.866957
Critical Value (10%)            -2.569655
dtype: float64
```

In [ ]:

```
Dickey_Fuller_test(TS_English)
```

```
Time Series is NOT Stationary
P_value is:  0.18953359279992366
```

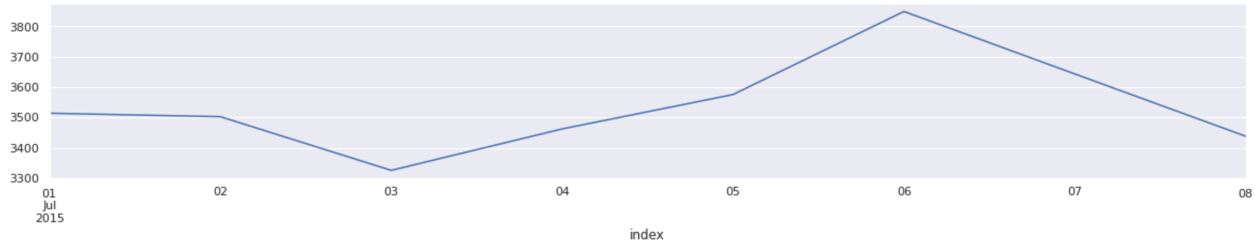
In [ ]:

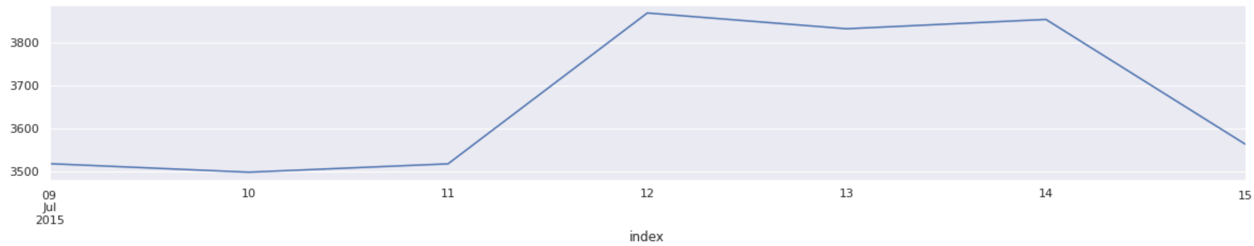
In [ ]:

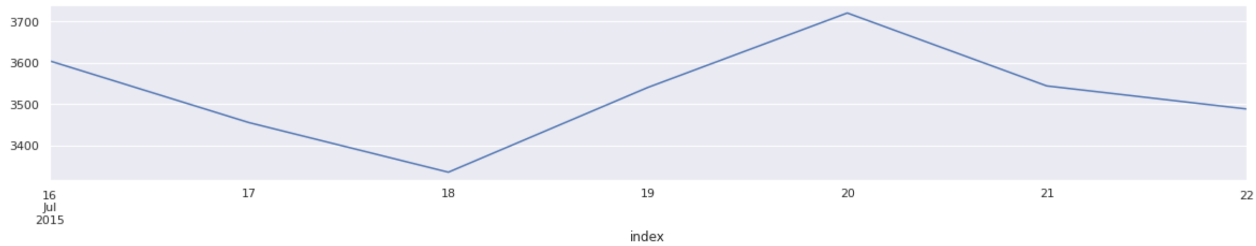
Visualising English-Language Page Visits Time Series manually to identify seasonality and period :

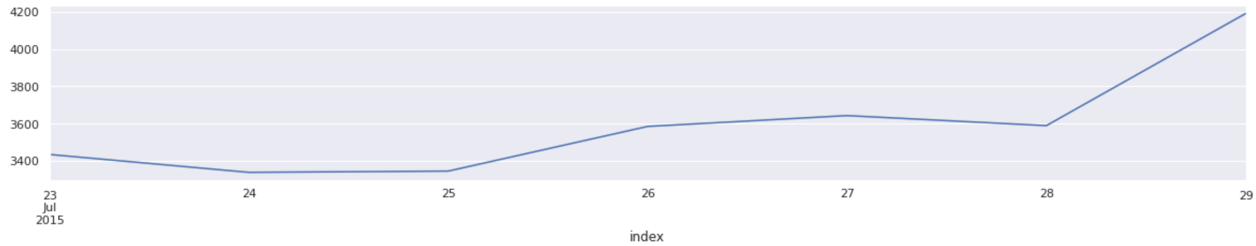
In [ ]:

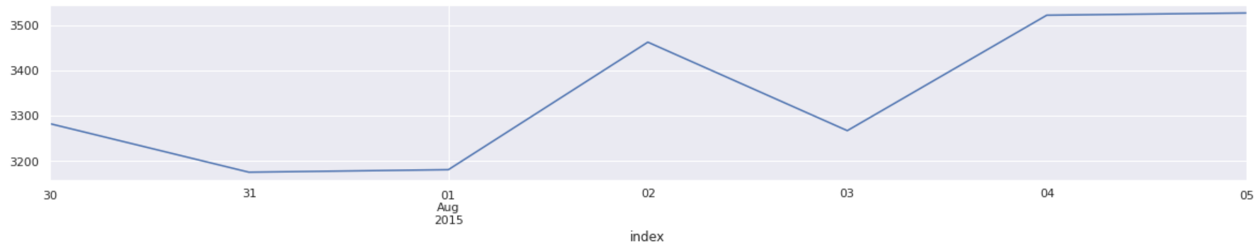
```
plt.rcParams['figure.figsize'] = (20, 3)

TS_English[:8].plot()
plt.show()
TS_English[8:15].plot()
plt.show()
TS_English[15:22].plot()
plt.show()
TS_English[22:29].plot()
plt.show()
TS_English[29:36].plot()
plt.show()

TS_English[36:44].plot()
plt.show()
```

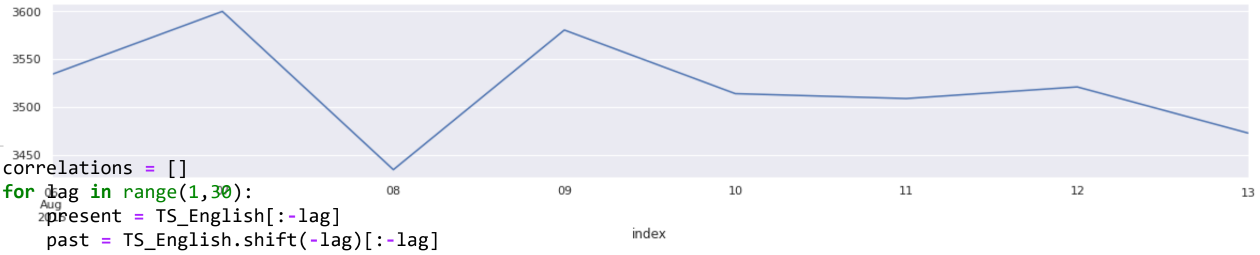













```
correlations = []
for lag in range(1,30):
    present = TS_English[:-lag]
    past = TS_English.shift(-lag)[:-lag]
    corrs = np.corrcoef(present,past)[0][-1]
    print(lag,corrs)
    correlations.append(corrs)
```

```
1 0.9363434527458435
2 0.8682966716039896
3 0.8185418037184544
4 0.7846718829500342
5 0.7612561076942573
6 0.7542260641783559
7 0.7386829287516693
8 0.6912638018189877
9 0.6370978014300401
10 0.6015277501876303
11 0.5825450402423571
12 0.5812931934793534
13 0.6007266462817789
14 0.6142525351445116
15 0.5971084554755528
16 0.5693834937428246
17 0.5488401467532626
18 0.5377431132136109
19 0.5430816743411203
20 0.5552694244923043
21 0.5540623423718063
22 0.5092655604869363
23 0.45373695576813583
24 0.4112336297620323
25 0.38162860616251737
26 0.3651996316699481
27 0.3723603627302601
28 0.37818226683160033
29 0.35939242667328175
```

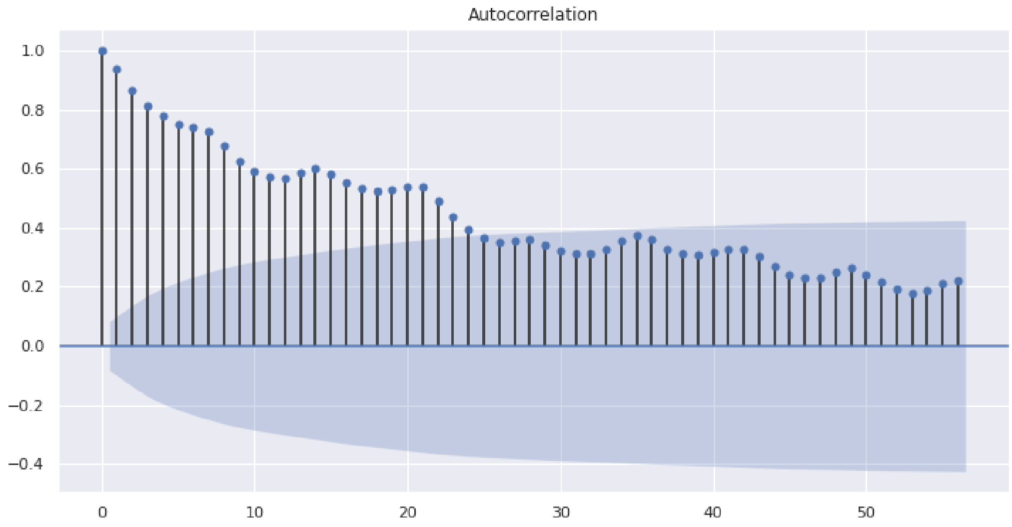
In [ ]:

Time Series Decomposition

```
Y(t) = seasonality + trend + residuals
          S(t)       + T(t)   + R(t)
```

In [ ]:

```
# using auto correlation function plot , to varify the period

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

plt.rcParams['figure.figsize'] = (12, 6)
plot_acf(TS_English,lags=56);
```



In [ ]:

```
plt.rcParams['figure.figsize'] = (15, 10)

Decomposition_model = sm.tsa.seasonal_decompose(TS_English, model='additive',period=7)
Decomposition_model.plot();
```

In [ ]:

```
Dickey_Fuller_test(pd.Series(Decomposition_model.resid).fillna(0))
```

```
Time Series is Stationary
P_value is:  3.727526947812948e-21
```

In [ ]:

```
# Residuals from time series decomposition are Stationary
```

In [ ]:

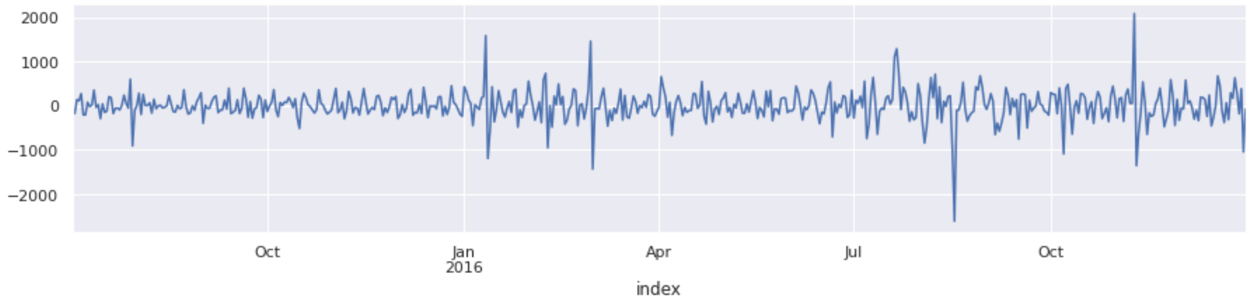
```

```

In [ ]:

```
# Taking the first differentiation of the time series and plotting

plt.rcParams['figure.figsize'] = (15, 3)

TS_English.diff(1).dropna().plot()
```

Out[66]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a525e4af0>
```



In [ ]:

```
Dickey_Fuller_test(TS_English.diff(1).dropna())
```

```
Time Series is Stationary
P_value is:  5.292474635436075e-13
```

In [ ]:

```
# After 1 differentiation  , time series becomes stationary.
# Thus for ARIMA models , we can set d = 1
```

In [ ]:

```

```

In [ ]:

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

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

In [ ]:

```

```

In [ ]:

```

```

Forecasting :

Trying out ExponentialSmoothing Method :

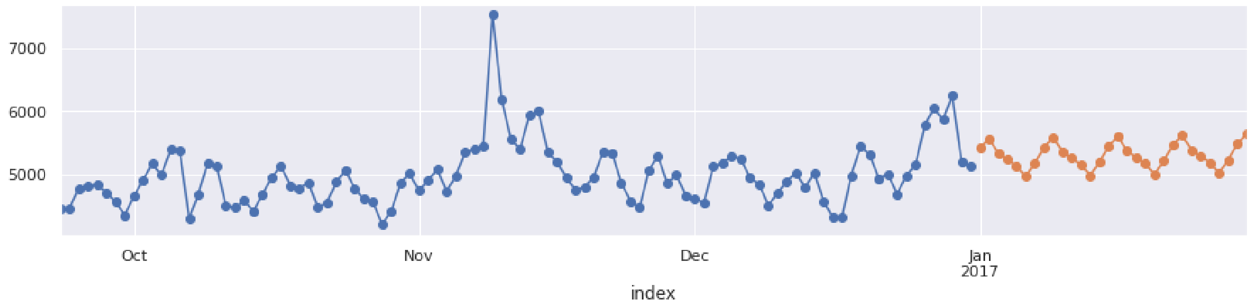
In [ ]:

```
model = sm.tsa.ExponentialSmoothing(TS_English, seasonal='add',trend="add")
model = model.fit()
                                                     # default values
                                                     # of smoothing_level, seasonal_smoothing and
                                                     # and trend smoothing

TS_English.tail(100).plot(style='-o', label='actual')
model.forecast(30).plot(style='-o', label='predicted')
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524: ValueWarning: No frequency informatio
n was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```

Out[71]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f0a29ad9bb0&gt;
```



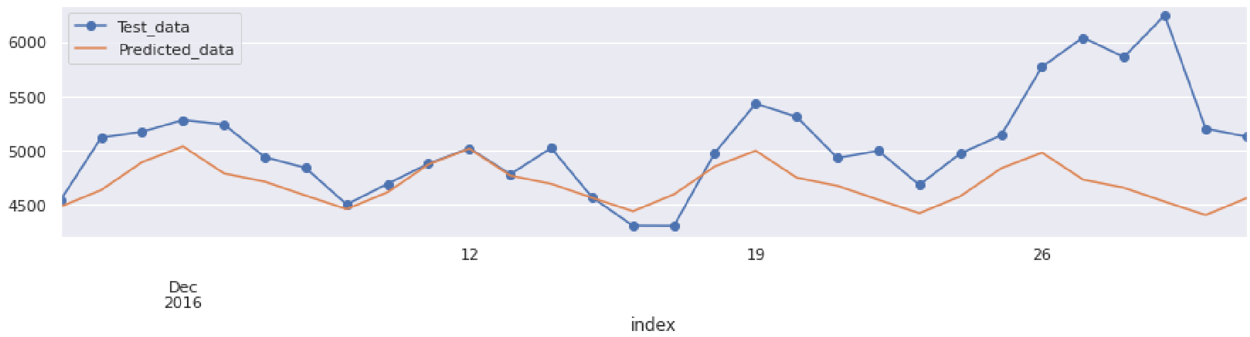
In [ ]:

In [ ]:

In [ ]:

```
X_train = TS_English.loc[TS_English.index < TS_English.index[-30] ].copy()
X_test = TS_English.loc[TS_English.index >= TS_English.index[-30] ].copy()

import warnings # supress warnings
warnings.filterwarnings('ignore')


model = sm.tsa.ExponentialSmoothing(X_train,
                                    trend="add",
                                    damped_trend="add",
                                    seasonal="add")
model = model.fit(smoothing_level=None,     # alpha
            smoothing_trend=None,           # beta
            smoothing_seasonal=None)        # gama)

# X_test.plot()
Pred = model.forecast(steps=30)
performance(X_test,Pred)

X_test.plot(style="-o",label ="Test_data")
Pred.plot(label="Predicted_data")
plt.legend()
plt.show()
```

```
MAE : 401.982
RMSE : 568.477
MAPE: 0.074
```



In [ ]:

```

```

ARIMA :

- **Autoregressive Integrated Moving Average (ARIMA) model, and extensions**

```
This model is the basic interface for ARIMA-type models, including those
with exogenous regressors and those with seasonal components. The most
general form of the model is SARIMAX(p, d, q)x(P, D, Q, s). It also allows
all specialized cases, including

autoregressive models: AR(p)

moving average models: MA(q)

mixed autoregressive moving average models: ARMA(p, q)

integration models: ARIMA(p, d, q)

seasonal models: SARIMA(P, D, Q, s)

regression with errors that follow one of the above ARIMA-type models
```

In [ ]:

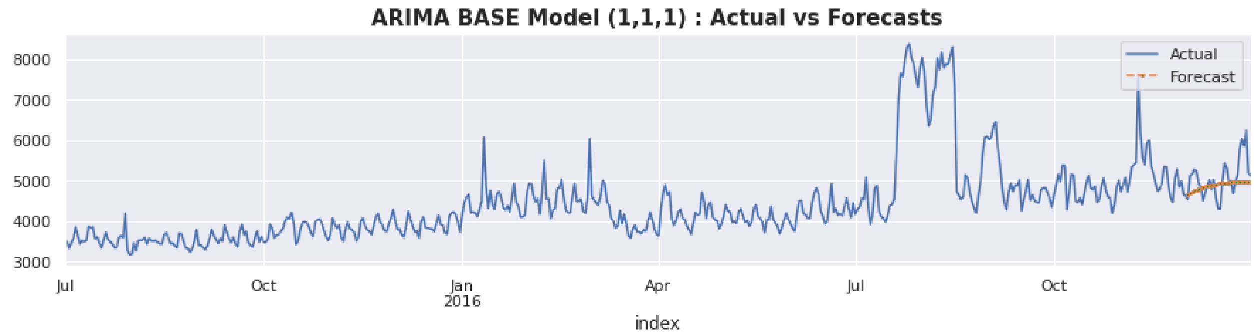
```
from statsmodels.tsa.arima.model import ARIMA
```

In [ ]:

```
TS = TS_English.copy(deep=True)
```

In [ ]:

```
n_forecast = 30


model = ARIMA(TS[:-n_forecast],
              order = (1,1,1))
model = model.fit()

predicted = model.forecast(steps= n_forecast, alpha = 0.05)


TS.plot(label = 'Actual')
predicted.plot(label = 'Forecast', linestyle='dashed', marker='o',markerfacecolor='green', markersize=2)
plt.legend(loc="upper right")
plt.title('ARIMA BASE Model (1,1,1) : Actual vs Forecasts', fontsize = 15, fontweight = 'bold')
plt.show()


#Calculating MAPE & RMSE
actuals = TS.values[-n_forecast:]
errors = TS.values[-n_forecast:] - predicted.values

mape = np.mean(np.abs(errors)/ np.abs(actuals))
rmse = np.sqrt(np.mean(errors**2))

print()
print(f'MAPE of Model : {np.round(mape,5)}')

print(f'RMSE of Model : {np.round(rmse,3)}')
```




```
MAPE of Model : 0.06585
RMSE of Model : 472.186
```

In [ ]:

```

```

SARIMAX model :

In [ ]:

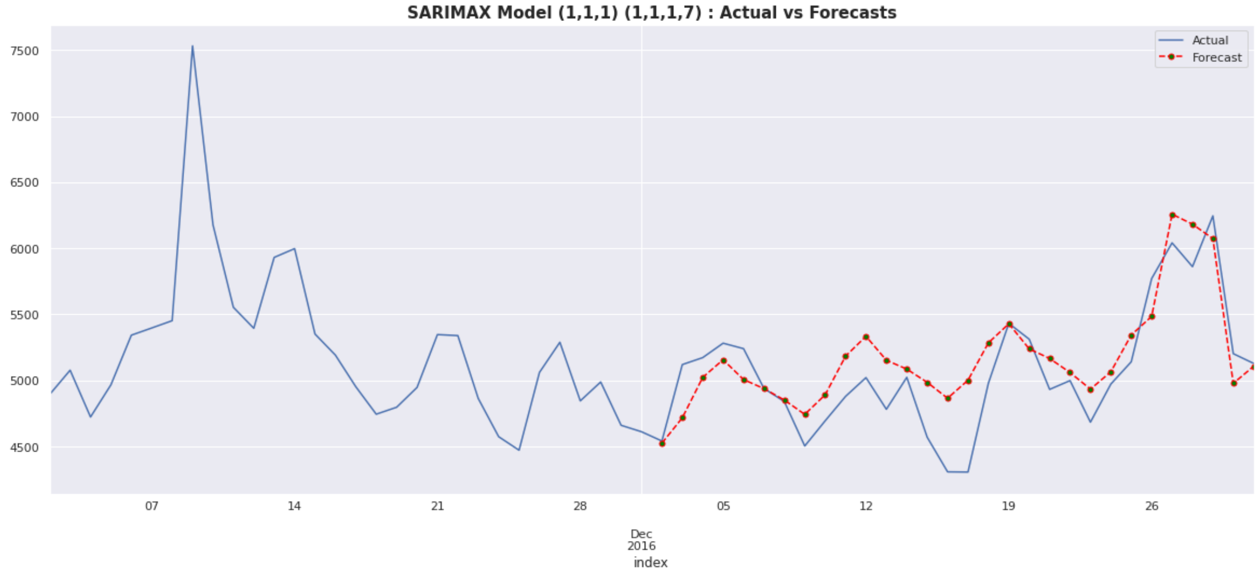
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [ ]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

def sarimax_model(time_series, n, p=0, d=0, q=0, P=0, D=0, Q=0, s=0, exog = []):

    #Creating SARIMAX Model with order(p,d,q) & seasonal_order=(P, D, Q, s)
    model = SARIMAX(time_series[:-n], \
                    order =(p,d,q),
                    seasonal_order=(P, D, Q, s),
                    exog = exog[:-n],
                    initialization='approximate_diffuse')
    model_fit = model.fit()

    #Creating forecast for last n-values
    model_forecast = model_fit.forecast(n, dynamic = True, exog = pd.DataFrame(exog[-n:]))

    #plotting Actual & Forecasted values

    plt.figure(figsize = (20,8))
    time_series[-60:].plot(label = 'Actual')
    model_forecast[-60:].plot(label = 'Forecast', color = 'red',
                              linestyle='dashed', marker='o',markerfacecolor='green', markersize=5)
    plt.legend(loc="upper right")
    plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Forecasts', fontsize = 15, fontweight = 'bold')
    plt.show()

    #Calculating MAPE & RMSE
    actuals = time_series.values[-n:]
    errors = time_series.values[-n:] - model_forecast.values

    mape = np.mean(np.abs(errors)/ np.abs(actuals))
    rmse = np.sqrt(np.mean(errors**2))

    print()
    print(f'MAPE of Model : {np.round(mape,5)}')
    print(f'RMSE of Model : {np.round(rmse,3)}')
```

In [ ]:

```
exog = Exog_Campaign_eng['Exog'].to_numpy()
time_series = aggregated_data.English
test_size= 0.1
p,d,q, P,D,Q,s = 1,1,1,1,1,1,7
n = 30
sarimax_model(time_series, n, p=p, d=d, q=q, P=P, D=D, Q=Q, s=s, exog = exog)
```




```
MAPE of Model : 0.04454
RMSE of Model : 272.775
```

Hyperparamer tuning for SARIMAX model

In [ ]:

```
def SARIMAX_grid_search(time_series, n, param, d_param, s_param, exog = []):
    counter = 0
    #creating df for storing results summary
    param_df = pd.DataFrame(columns = ['serial','pdq', 'PDQs', 'mape', 'rmse'])

    #Creating loop for every paramater to fit SARIMAX model
    for p in param:
        for d in d_param:
            for q in param:
                for P in param:
                    for D in d_param:
                        for Q in param:
                            for s in s_param:
                                #Creating Model
                                model = SARIMAX(time_series[:-n],
                                                order=(p,d,q),
                                                seasonal_order=(P, D, Q, s),
                                                exog = exog[:-n],
                                                initialization='approximate_diffuse')
                                model_fit = model.fit()

                                #Creating forecast from Model
                                model_forecast = model_fit.forecast(n, dynamic = True, exog = pd.DataFrame(exog[-n:]))

                                #Calculating errors for results
                                actuals = time_series.values[-n:]
                                errors = time_series.values[-n:] - model_forecast.values

                                #Calculating MAPE & RMSE
                                mape = np.mean(np.abs(errors)/ np.abs(actuals))
                                rmse = np.sqrt(np.mean(errors**2))
                                mape = np.round(mape,5)
                                rmse = np.round(rmse,3)

                                #Storing the results in param_df
                                counter += 1
                                list_row = [counter, (p,d,q), (P,D,Q,s), mape, rmse]
                                param_df.loc[len(param_df)] = list_row

                #print statement to check progress of Loop
                print(f'Possible Combination: {counter} out of { (len(param)**4)*len(s_param)*(len(d_param)**2)} calculated')

    return param_df
```

In [ ]:

```
#Finding best parameters for English time series

exog = Exog_Campaign_eng['Exog'].to_numpy()
time_series = aggregated_data.English
n = 30
param = [0,1,2]
d_param = [0,1]
s_param = [7]

english_params  = SARIMAX_grid_search(time_series, n, param, d_param,s_param, exog)
```

In [ ]:

```
english_params.sort_values(['mape', 'rmse']).head()
```

Out[86]:

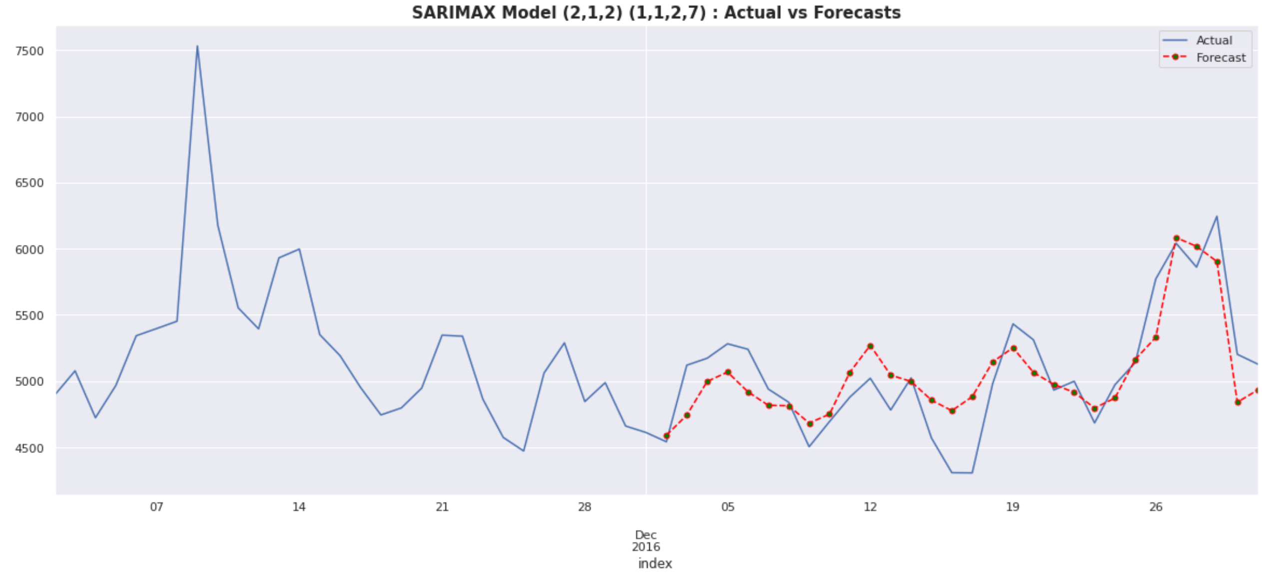
	serial	pdq	PDQs	mape	rmse
317	318	(2, 1, 2)	(1, 1, 2, 7)	0.04052	247.335
323	324	(2, 1, 2)	(2, 1, 2, 7)	0.04188	255.183
40	41	(0, 0, 2)	(0, 1, 1, 7)	0.04199	276.311
41	42	(0, 0, 2)	(0, 1, 2, 7)	0.04206	271.577
46	47	(0, 0, 2)	(1, 1, 1, 7)	0.04212	270.076

In [ ]:

```
# best possible parameters : p ,d ,q,P,D,Q,s = 2,1,2,1,1,2,7
```

In [ ]:

```
exog = Exog_Campaign_eng['Exog'].to_numpy()
time_series = aggregated_data.English
test_size= 0.1
p,d,q, P,D,Q,s = 2,1,2,1,1,2,7
n = 30
sarimax_model(time_series, n, p=p, d=d, q=q, P=P, D=D, Q=Q, s=s, exog = exog)
```




```
MAPE of Model : 0.04052
RMSE of Model : 247.335
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Hyperparameter tuning for all other languages :

In [ ]:

```
def pipeline_sarimax_grid_search_without_exog(languages, data, n, param, d_param, s_param):

    best_param_df  = pd.DataFrame(columns = ['language','p','d', 'q', 'P','D','Q','s','mape'])
    for lang in languages:
        print('')
        print('')
        print(f'--------------------------------------------------------------')
        print(f'          Finding best parameters for {lang}                   ')
        print(f'--------------------------------------------------------------')
        counter = 0
        time_series = data[lang]
        best_mape = 100

        #Creating loop for every paramater to fit SARIMAX model
        for p in param:
            for d in d_param:
                for q in param:
                    for P in param:
                        for D in d_param:
                            for Q in param:
                                for s in s_param:
                                    #Creating Model
                                    model = SARIMAX(time_series[:-n],
                                                    order=(p,d,q),
                                                    seasonal_order=(P, D, Q, s),
                                                    initialization='approximate_diffuse')
                                    model_fit = model.fit()

                                    #Creating forecast from Model
                                    model_forecast = model_fit.forecast(n, dynamic = True)

                                    #Calculating errors for results
                                    actuals = time_series.values[-n:]
                                    errors = time_series.values[-n:] - model_forecast.values

                                    #Calculating MAPE & RMSE
                                    mape = np.mean(np.abs(errors)/ np.abs(actuals))

                                    counter += 1

                                    if (mape < best_mape):
                                        best_mape = mape
                                        best_p = p
                                        best_d = d
                                        best_q = q
                                        best_P = P
                                        best_D = D
                                        best_Q = Q
                                        best_s = s
                                    else: pass

                    #print statement to check progress of Loop
                    print(f'Possible Combination: {counter} out of {(len(param)**4)*len(s_param)*(len(d_param)**2)} calculated')

        best_mape = np.round(best_mape, 5)
        print(f'--------------------------------------------------------------')
        print(f'Minimum MAPE for {lang} = {best_mape}')
        print(f'Corresponding Best Parameters are {best_p , best_d, best_q, best_P, best_D, best_Q, best_s}')
        print(f'--------------------------------------------------------------')

        best_param_row = [lang, best_p, best_d, best_q, best_P, best_D, best_Q, best_s, best_mape]
        best_param_df.loc[len(best_param_df)] = best_param_row

    return best_param_df
```

In [ ]:

```
languages = aggregated_data.columns
n = 30
param = [0,1,2]
d_param = [0,1]
s_param = [7]


best_param_df = pipeline_sarimax_grid_search_without_exog(languages, aggregated_data, n, param, d_param, s_param)
```

```
-------------------------------------------------------------
          Finding best parameters for Chinese
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for Chinese = 0.03074
Corresponding Best Parameters are (0, 1, 0, 1, 0, 2, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for English
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for English = 0.05252
Corresponding Best Parameters are (2, 0, 1, 0, 1, 2, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for French
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for French = 0.06359
Corresponding Best Parameters are (0, 0, 2, 2, 1, 2, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for German
-------------------------------------------------------------
```

```
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for German = 0.06578
Corresponding Best Parameters are (0, 1, 1, 1, 0, 1, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for Japenese
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for Japenese = 0.07122
Corresponding Best Parameters are (0, 1, 2, 2, 1, 0, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for Russian
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for Russian = 0.04763
Corresponding Best Parameters are (0, 0, 2, 1, 0, 2, 7)
-------------------------------------------------------------


-------------------------------------------------------------
          Finding best parameters for Spanish
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
```

```
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
--------------------------------------------------------------
Minimum MAPE for Spanish = 0.08561
Corresponding Best Parameters are (0, 1, 0, 2, 1, 0, 7)
--------------------------------------------------------------
```

In [ ]:

```
```

In [ ]:

```
best_param_df.sort_values(['mape'], inplace = True)
best_param_df
```

Out[92]:

	language	p	d	q	P	D	Q	s	mape
0	Chinese	0	1	0	1	0	2	7	0.03074
5	Russian	0	0	2	1	0	2	7	0.04763
1	English	2	0	1	0	1	2	7	0.05252
2	French	0	0	2	2	1	2	7	0.06359
3	German	0	1	1	1	0	1	7	0.06578
4	Japenese	0	1	2	2	1	0	7	0.07122
6	Spanish	0	1	0	2	1	0	7	0.08561

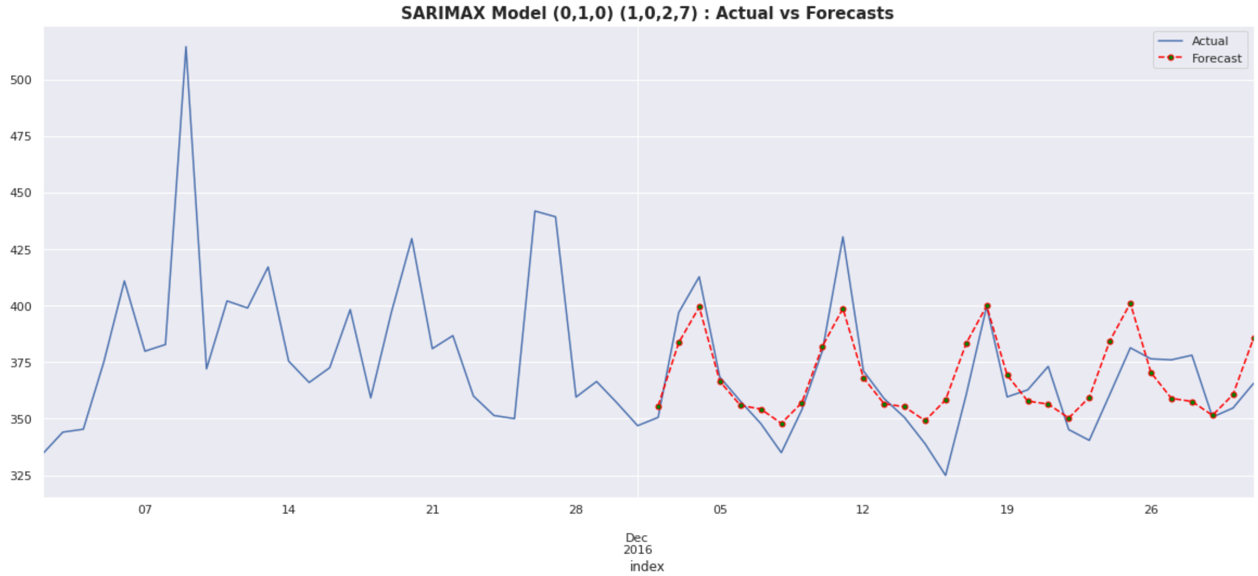
In [ ]:

```
def plot_best_SARIMAX_model(languages, data, n, best_param_df):

    for lang in languages:
        #fetching respective best parameters for that language
        p = best_param_df.loc[best_param_df['language'] == lang, ['p']].values[0][0]
        d = best_param_df.loc[best_param_df['language'] == lang, ['d']].values[0][0]
        q = best_param_df.loc[best_param_df['language'] == lang, ['q']].values[0][0]
        P = best_param_df.loc[best_param_df['language'] == lang, ['P']].values[0][0]
        D = best_param_df.loc[best_param_df['language'] == lang, ['D']].values[0][0]
        Q = best_param_df.loc[best_param_df['language'] == lang, ['Q']].values[0][0]
        s = best_param_df.loc[best_param_df['language'] == lang, ['s']].values[0][0]

        #Creating language time-series
        time_series = data[lang]

        #Creating SARIMAX Model with order(p,d,q) & seasonal_order=(P, D, Q, s)
        model = SARIMAX(time_series[:-n],
                        order =(p,d,q),
                        seasonal_order=(P, D, Q, s),
                        initialization='approximate_diffuse')
        model_fit = model.fit()

        #Creating forecast for last n-values
        model_forecast = model_fit.forecast(n, dynamic = True)

        #Calculating MAPE & RMSE
        actuals = time_series.values[-n:]
        errors = time_series.values[-n:] - model_forecast.values

        mape = np.mean(np.abs(errors)/ np.abs(actuals))
        rmse = np.sqrt(np.mean(errors**2))

        print('')
        print('')
        print(f'------------------------------------------------------------------------------------------')
        print(f'        SARIMAX model for {lang} Time Series                                              ')
        print(f'        Parameters of Model : ({p},{d},{q}) ({P},{D},{Q},{s})                             ')
        print(f'        MAPE of Model       : {np.round(mape,5)}                                          ')
        print(f'        RMSE of Model       : {np.round(rmse,3)}                                          ')
        print(f'------------------------------------------------------------------------------------------')

        #plotting Actual & Forecasted values
        time_series.index = time_series.index.astype('datetime64[ns]')
        model_forecast.index = model_forecast.index.astype('datetime64[ns]')
        plt.figure(figsize = (20,8))
        time_series[-60:].plot(label = 'Actual')
        model_forecast[-60:].plot(label = 'Forecast', color = 'red',
                                  linestyle='dashed', marker='o',markerfacecolor='green', markersize=5)
        plt.legend(loc="upper right")
        plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Forecasts', fontsize = 15, fontweight = 'bold')
        plt.show()

    return 0
```

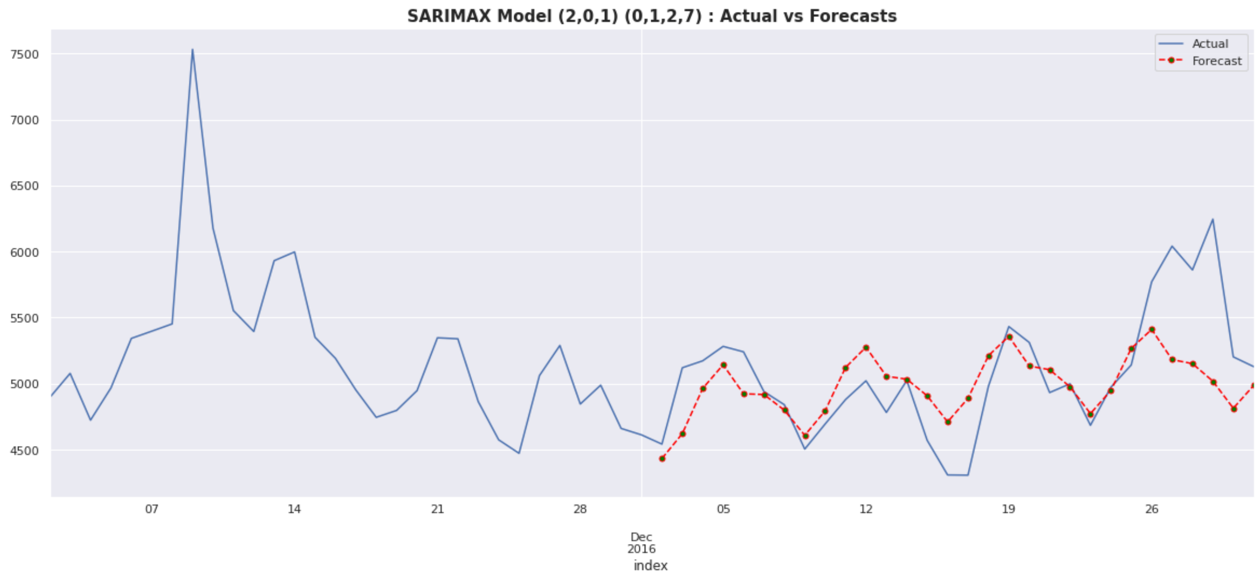
In [ ]:

```
#Plotting SARIMAX model for each Language Time Series
languages = aggregated_data.columns
n = 30
plot_best_SARIMAX_model(languages, aggregated_data, n, best_param_df)
```

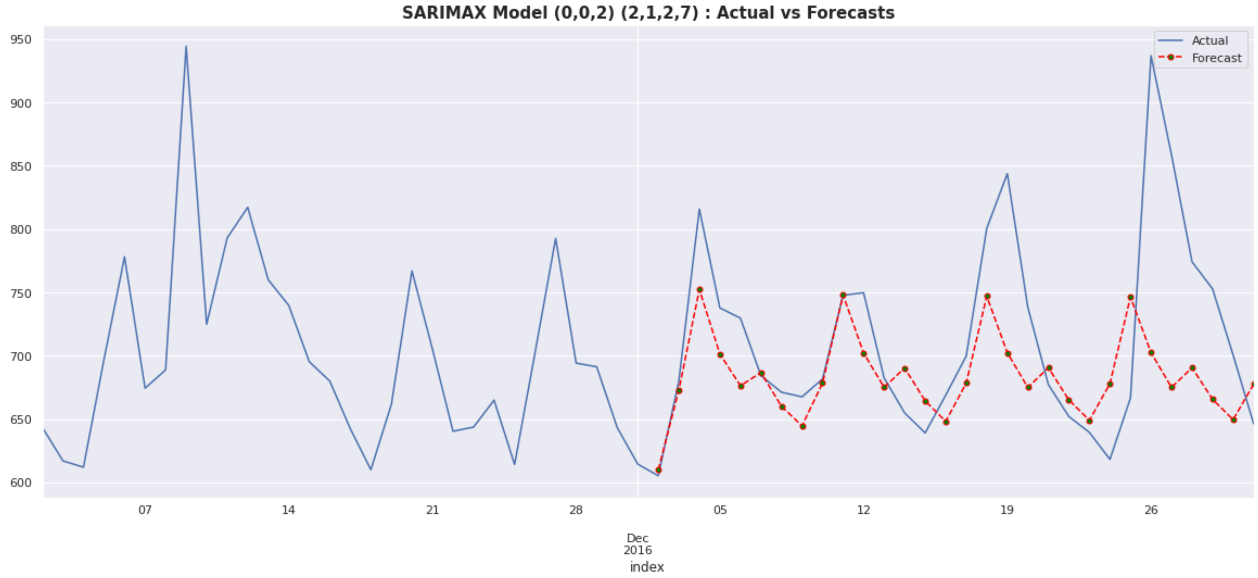
```
--------------------------------------------------------------------------------
        SARIMAX model for Chinese Time Series
        Parameters of Model : (0,1,0) (1,0,2,7)
        MAPE of Model       : 0.03074
        RMSE of Model       : 14.487
--------------------------------------------------------------------------------
```



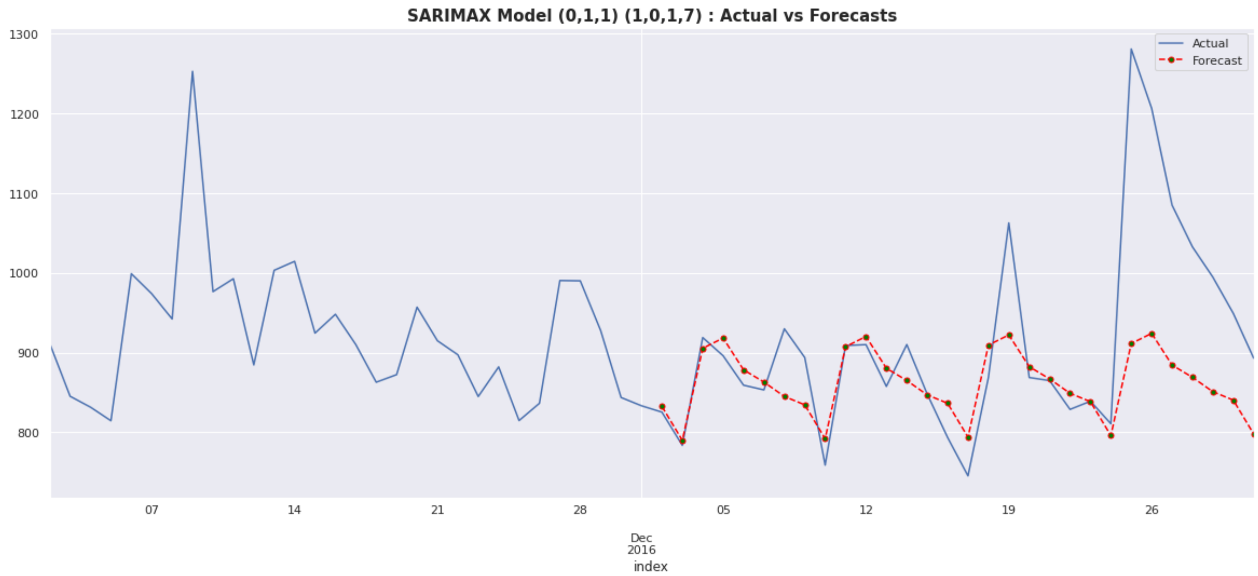

```
--------------------------------------------------------------------------------
        SARIMAX model for English Time Series
        Parameters of Model : (2,0,1) (0,1,2,7)
        MAPE of Model       : 0.05252
        RMSE of Model       : 385.55
--------------------------------------------------------------------------------
```



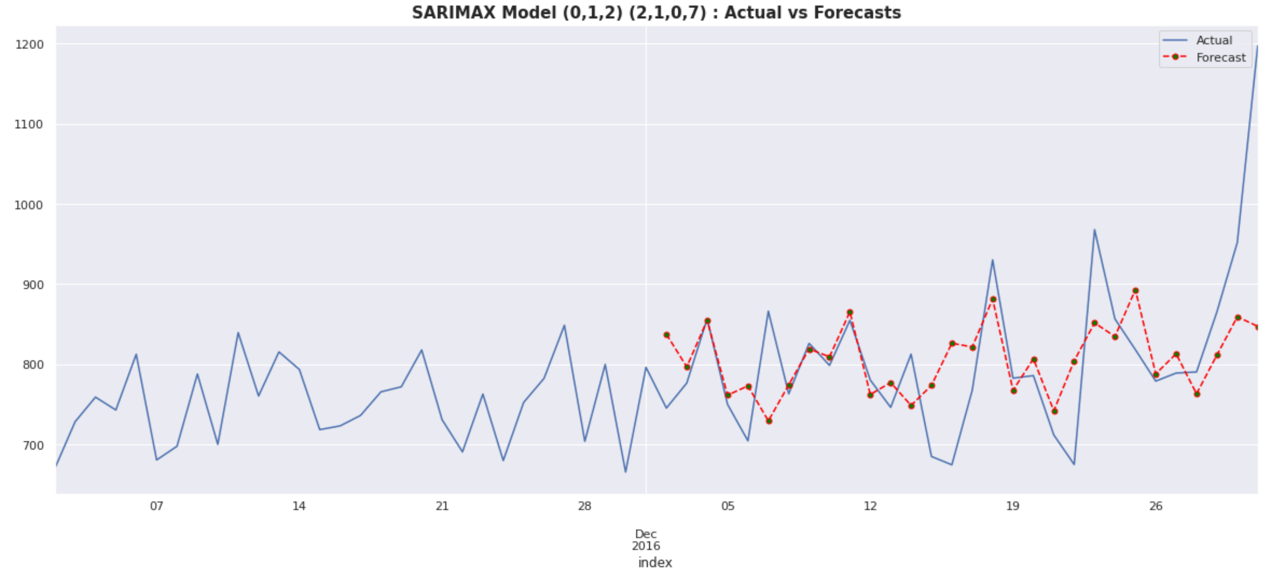

```
--------------------------------------------------------------------------------
        SARIMAX model for French Time Series
        Parameters of Model : (0,0,2) (2,1,2,7)
        MAPE of Model       : 0.06359
        RMSE of Model       : 72.57
--------------------------------------------------------------------------------
```



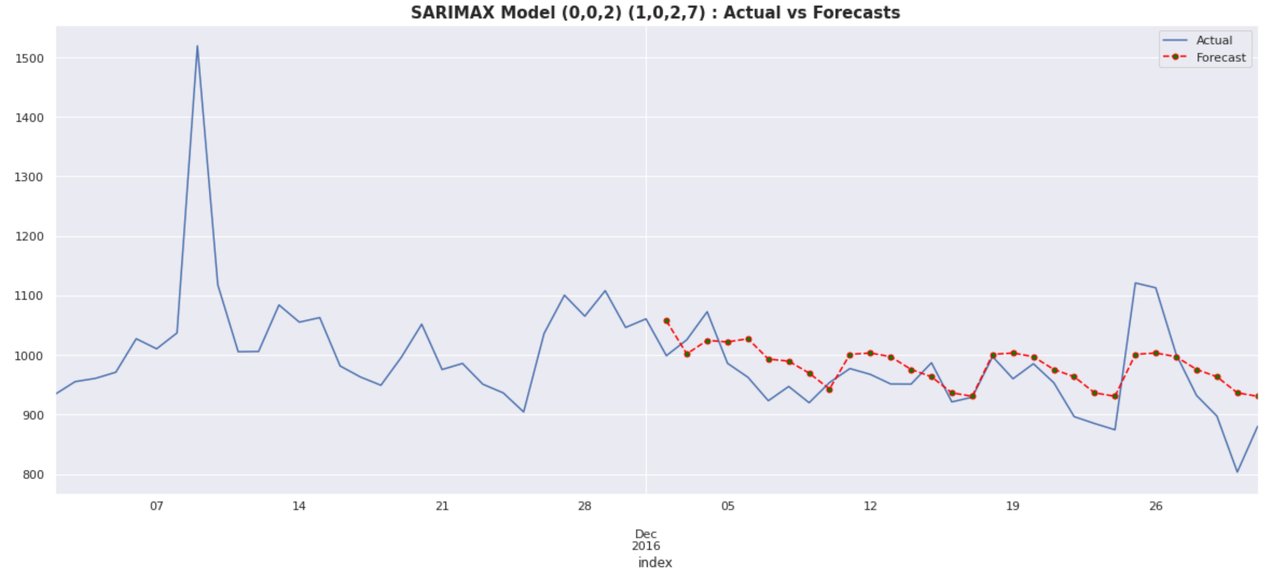

```
---------------------------------------------------------------------------------
       SARIMAX model for German Time Series
       Parameters of Model : (0,1,1) (1,0,1,7)
       MAPE of Model       : 0.06578
       RMSE of Model       : 110.629
---------------------------------------------------------------------------------
```




```
---------------------------------------------------------------------------------
       SARIMAX model for Japenese Time Series
       Parameters of Model : (0,1,2) (2,1,0,7)
       MAPE of Model       : 0.07122
       RMSE of Model       : 90.833
---------------------------------------------------------------------------------
```




```
-----------------------------------------------------------------------------------------
        SARIMAX model for Russian Time Series
        Parameters of Model : (0,0,2) (1,0,2,7)
        MAPE of Model       : 0.04763
        RMSE of Model       : 55.45
-----------------------------------------------------------------------------------------
```




```
-----------------------------------------------------------------------------------------
        SARIMAX model for Spanish Time Series
        Parameters of Model : (0,1,0) (2,1,0,7)
        MAPE of Model       : 0.08561
        RMSE of Model       : 109.03
-----------------------------------------------------------------------------------------
```

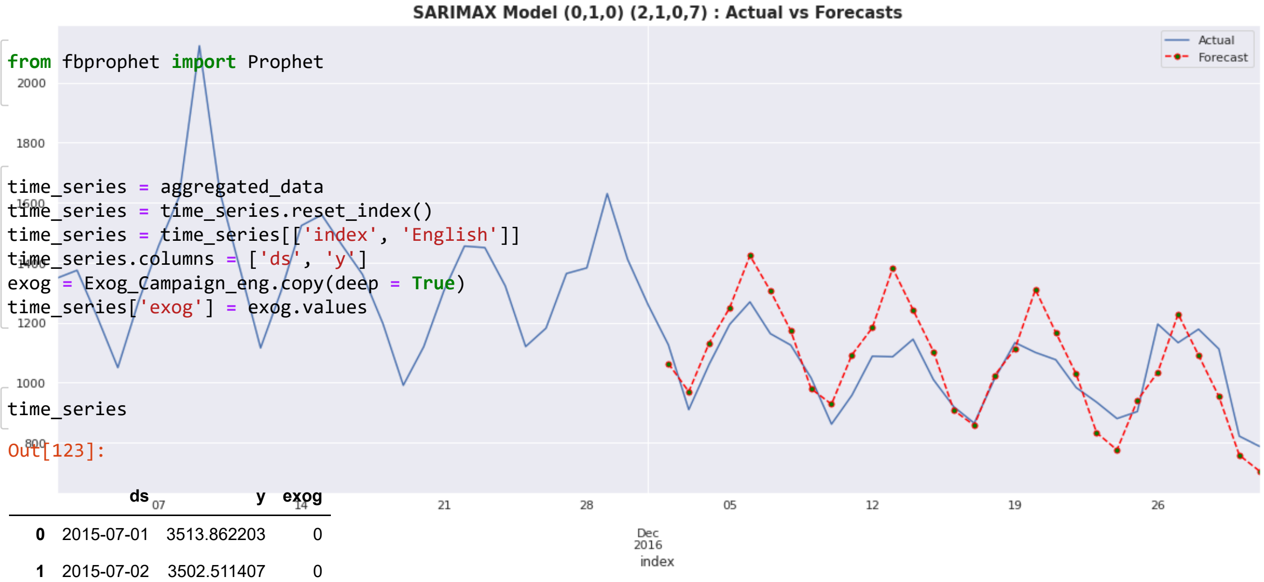
In [ ]:

```

```

In [ ]:

```
# !pip install pystan~=2.14
# !pip install fbprophet
```

Forecasting using Facebook Prophet :




```
from fbprophet import Prophet
```

```
time_series = aggregated_data
time_series = time_series.reset_index()
time_series = time_series[['index', 'English']]
time_series.columns = ['ds', 'y']
exog = Exog_Campaign_eng.copy(deep = True)
time_series['exog'] = exog.values
```

```
time_series
```

Out[123]:

	ds	y	exog
0	2015-07-01	3513.862203	0
1	2015-07-02	3502.511407	0
2	2015-07-03	3325.357889	0
3	2015-07-04	3462.054256	0
4	2015-07-05	3575.520035	0
...	...	...	...
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

550 rows × 3 columns

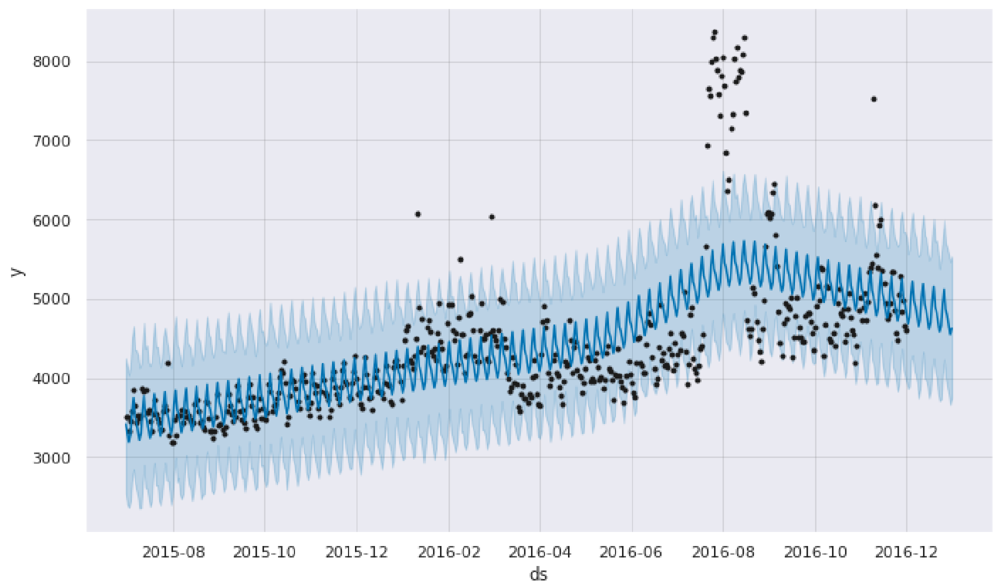
Out[94]:

0

In [ ]:

```
prophet1 = Prophet(weekly_seasonality=True)
prophet1.fit(time_series[['ds', 'y']][:-30])
future = prophet1.make_future_dataframe(periods=30, freq= 'D')
forecast = prophet1.predict(future)
fig1 = prophet1.plot(forecast)
```

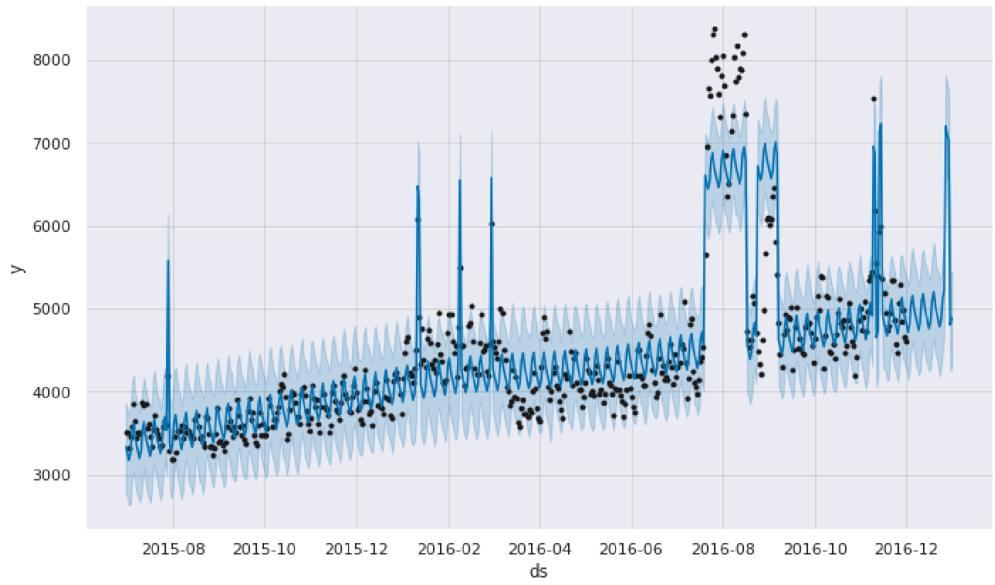
```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```



In [ ]:

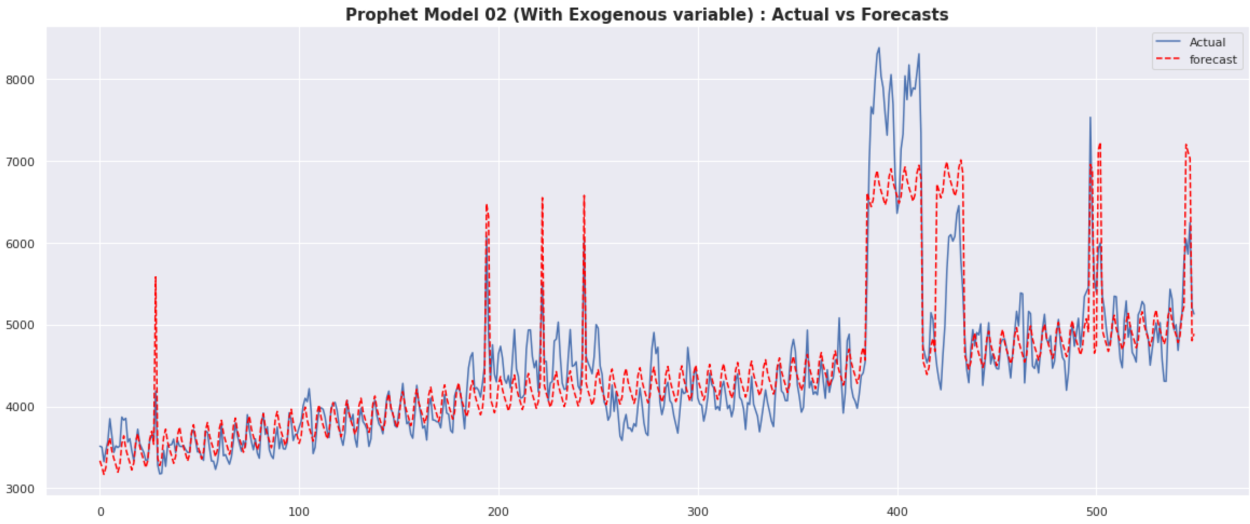
```
prophet2 = Prophet(weekly_seasonality=True)
prophet2.add_regressor('exog')
prophet2.fit(time_series[:-30])
#future2 = prophet2.make_future_dataframe(periods=30, freq= 'D')
forecast2 = prophet2.predict(time_series)
fig2 = prophet2.plot(forecast2)
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```



In [ ]:

```
actual = time_series['y'].values
forecast = forecast2['yhat'].values

plt.figure(figsize = (20,8))
plt.plot(actual, label = 'Actual')
plt.plot(forecast, label = 'forecast', color = 'red', linestyle='dashed')
plt.legend(loc="upper right")
plt.title(f'Prophet Model 02 (With Exogenous variable) : Actual vs Forecasts', fontsize = 15, fontweight = 'bold')
plt.show()
```



In [ ]:

```
errors = abs(actual - forecast)
mape = np.mean(errors/abs(actual))
mape
```

Out[127]:

```
0.0594578640110275
```

In [ ]:

```

```

Inferences and Recommendations :

- inferences made from the data visualizations:
  - Total 7 languages found in data.
  - English has the highest number of pages.

- 3 access types:
  - all-access 51.2295 %
  - mobile-web 24.7748 %
  - desktop 23.9958 %

- 2 access origins:
  - agents 75.932526 %
  - spider 24.067474 %

- English language has the highest pages.
- Maximum ads should be run on English Page.

In [ ]:

```

```

In [ ]:

```

```

- What does the decomposition of series do?
  - 0The decomposition of a time series refers to the process of separating a time series into its components, such as trend, seasonality, and residuals.
  - These components are intended to represent different underlying patterns in the data. The idea behind decomposition is to break down a complex time series into simpler components that can be more easily understood and analyzed.
  - Trend component represents the underlying pattern in the data over time, reflecting long-term changes.
  - Seasonality component represents regular patterns that repeat over a fixed interval, such as daily, weekly, or yearly.
  - Residual component represents the remaining random fluctuations in the data after removing the trend and seasonality components.
  - Decomposition is often used in time series analysis to identify and isolate different patterns in the data and to forecast future values. It is also used to remove seasonality and trend components from the data before applying statistical or machine learning models to the residuals, as this can help to improve the performance of these models.

- What level of differencing gave you a stationary series?
  - Stationarity is an important property of a time series because many time series analysis techniques assume that the time series is stationary.
  - A time series is stationary if its mean, variance, and autocorrelation structure are constant over time.
  - Differencing is a common technique used to make a time series stationary.
  - It involves subtracting the value of the time series at a previous time step from the current time step.
  - This can help to remove trend and seasonality components from the data, making it more stationary.
  - The order of differencing refers to the number of times the differencing operation is performed.
  - in this case study, differencing once yield a stationary time series.

- Difference between arima, sarima & sarimax.

- ARIMA (AutoRegressive Integrated Moving Average) is a statistical model for time series data that accounts for both autoregression (the use of past values to predict future values) and moving average (the use of the residuals of past predictions to predict future values).
- It is a flexible method for modeling non-stationary time series data and can be used for both univariate and multivariate time series.
- ARIMA models are denoted by the notations ARIMA(p, d, q), where p is the order of the autoregression component, d is the order of differencing used to make the time series stationary, and q is the order of the moving average component.

- SARIMA (Seasonal AutoRegressive Integrated Moving Average) is a variation of ARIMA that accounts for both seasonality and non-stationarity in time series data.
- Seasonality refers to repeating patterns in the data over fixed time intervals, such as daily, weekly, or yearly. SARIMA models are denoted by the notations SARIMA(p, d, q)(P, D, Q, S), where p, d, and q are the same as in ARIMA models, P is the order of the seasonal autoregression component, D is the order of seasonal differencing, Q is the order of the seasonal moving average component, and S is the number of seasons in the data.

- SARIMAX (Seasonal AutoRegressive Integrated Moving Average with exogenous regressors) is an extension of SARIMA that allows for the inclusion of exogenous variables, or variables that are not part of the time series data, in the modeling process.

- SARIMAX models are useful when the time series data is influenced by other variables that are not part of the time series data, and can provide more accurate forecasts.
- SARIMAX models are denoted by the notations SARIMAX(p, d, q)(P, D, Q, S)x, where p, d, q, P, D, Q, and S are the same as in SARIMA models and x represents the number of exogenous variables included in the model.

- The equation for a SARIMA (Seasonal AutoRegressive Integrated Moving Average) model can be expressed as follows:

```
ARIMA(p, d, q)(P, D, Q, S):

   y(t) = c + ϕ1 * y(t-1) + ϕ2 * y(t-2) + ... + ϕp * y(t-p)
            + θ1 * e(t-1) + θ2 * e(t-2) + ... + θq * e(t-q)
            + δ * y(t-S) + Φ1 * y(t-S-1) + Φ2 * y(t-S-2) + ... + ΦP * y(t-S-P)
            + Θ1 * e(t-S-1) + Θ2 * e(t-S-2) + ... + ΘQ * e(t-S-Q) + e(t)

          where:

          y(t) is the value of the time series at time step t.
          c is a constant.
          ϕ1, ϕ2, ..., ϕp are the autoregression coefficients.
          θ1, θ2, ..., θq are the moving average coefficients.
          δ is a coefficient for the seasonal autoregression term.
          Φ1, Φ2, ..., ΦP are the seasonal autoregression coefficients.
          Θ1, Θ2, ..., ΘQ are the seasonal moving average coefficients.
          e(t), e(t-1), ..., e(t-q), e(t-S), e(t-S-1), ..., e(t-S-Q) are the residuals.
 - In a SARIMA model, the order of differencing (d) is used to make the time series stationary,
  the autoregression and moving average components (p and q) are used to model the autocorrelation structure of the r
esiduals,
  and the seasonal components (P, D, Q, and S) are used to model the seasonal patterns in the data.
   The coefficients in the model are estimated using maximum likelihood estimation or other optimization techniques,
    and the residuals are used to assess the goodness-of-fit of the model.
```

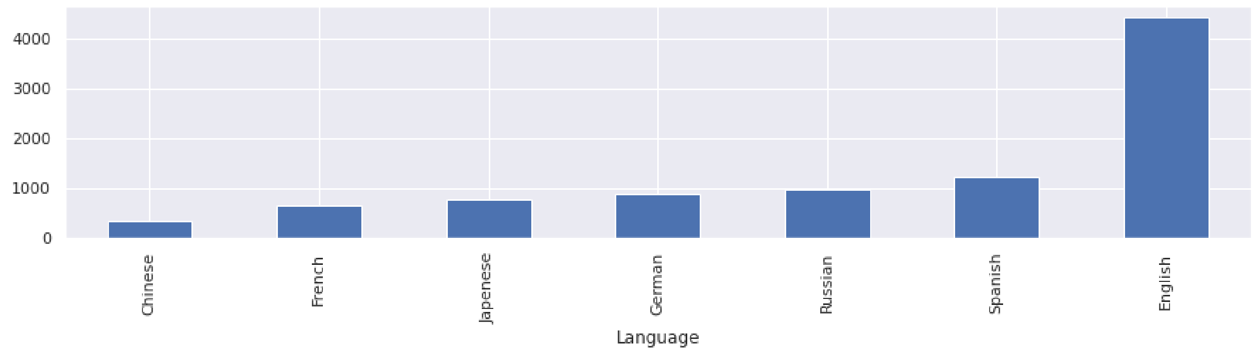
- Compare the number of views in different languages

In [ ]:

```
aggregated_data.mean().sort_values().plot(kind = 'bar')
```

Out[134]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09f1742760>
```



- What other methods other than grid search would be suitable to get the model for all languages?
  - When estimating the values of p, q, and d from the ACF and PACF plots of a time series, the following steps can be taken:
    - Determine if the time series is stationary by conducting an augmented Dickey-Fuller test.
    - If the time series is stationary, attempt to fit an ARMA model. If it is non-stationary, determine the value of d.
    - If stationarity is achieved, plot the autocorrelation and partial autocorrelation graphs of the data.
    - Plot the partial autocorrelation graph (PACF) to determine the value of p, as the cut-off point in the PACF is equal to p.
    - Plot the autocorrelation graph (ACF) to determine the value of q, as the cut-off point in the ACF is equal to q

---

In [ ]:

In [ ]:

```

```