AdEase Time Series

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.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("train_1.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	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	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	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	1
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	1
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

5 rows × 551 columns

In [4]:
```
df.shape
```

Out[4]:
```
(145063, 551)
```

In [5]:
```
Exog_Campaign_eng = pd.read_csv("Exog_Campaign_eng")
```

In [6]:
```
Exog_Campaign_eng.shape
```

Out[6]:
```
(550, 1)
```

In [7]:
```
Exog_Campaign_eng.head()
```

Out[7]:

	Exog
0	0
1	0
2	0
3	0
4	0

In [8]:
```
df.Page.sample(20)
```

Out[8]:
```
103406    Уоррен,_Эд_и_Лоррейн_ru.wikipedia.org_desktop_...
108638           龍珠超_zh.wikipedia.org_mobile-web_all-agents
124757             Дзюдо_ru.wikipedia.org_all-access_spider
128187    Guerre_civile_syrienne_fr.wikipedia.org_all-ac...
49080     Helene_Fischer_de.wikipedia.org_all-access_spider
38527     Logan_(film)_en.wikipedia.org_all-access_all-a...
86575              世耕弘成_ja.wikipedia.org_desktop_all-agents
122363          古賀茂明_ja.wikipedia.org_all-access_all-agents
7063      Wallis-et-Futuna_fr.wikipedia.org_desktop_all-...
133399           ソロモンの偽証_ja.wikipedia.org_all-access_spider
82374     Category:2011_Ansel_Adams_donation_from_U.S._N...
71078       Mesopotamia_es.wikipedia.org_desktop_all-agents
14146     File:Hen_Wlad_Fy_Nhadau.ogg_commons.wikimedia....
142945      Torre_Eiffel_es.wikipedia.org_all-access_spider
47714       Meret_Becker_de.wikipedia.org_all-access_spider
59142     魔法少女まどか☆マギカ_ja.wikipedia.org_mobile-web_all-ag...
20226     Special:EditWatchlist_www.mediawiki.org_all-ac...
110659    Drycothaea_ochreoscutellaris_en.wikipedia.org_...
14281     File:Naumachie_1550.PNG_commons.wikimedia.org_...
48114     Rudolph_William_Schroeder_de.wikipedia.org_all...
Name: Page, dtype: object
```

In [9]:
```
df['Page'].str.extract(r'[^_]+_[^_]+_[^_]+_([^_]+)').head(20)
```

Out[9]:

	0
0	spider
1	spider
2	spider
3	spider
4	Love
5	spider
6	spider
7	spider

	0
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

In [10]:
```
data = df.copy()
```

In [11]:
```
data.duplicated().sum()
# No duplicate data
```

Out[11]:
```
0
```

In [12]:
```
data.dtypes.sample(10)
```

Out[12]:
```
2016-12-19    float64
2016-01-26    float64
2016-12-14    float64
2016-02-18    float64
2016-11-09    float64
2016-07-30    float64
2015-08-08    float64
2016-10-31    float64
2016-08-18    float64
2016-05-13    float64
dtype: object
```

In [13]:
```
indexes = data.head(2).columns[1:][range(0,549,20)].values
indexes
```

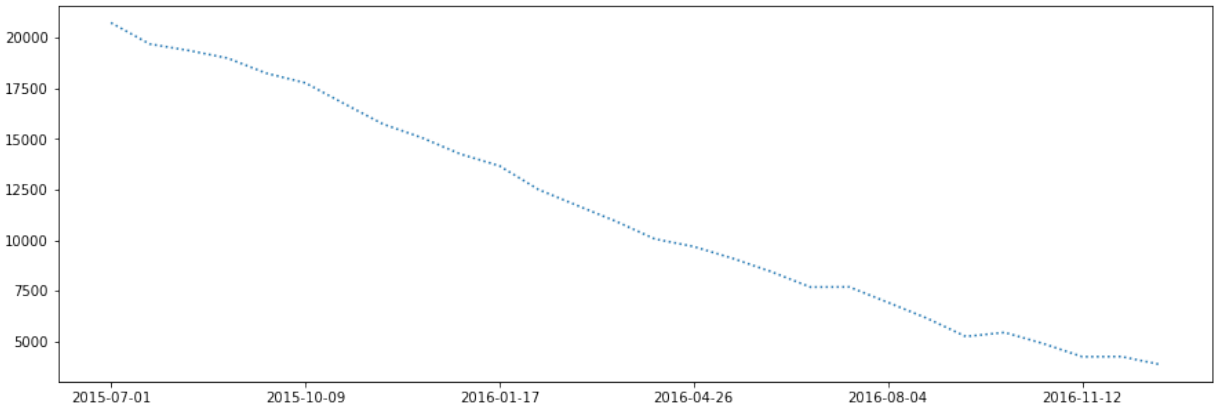
Out[13]:
```
array(['2015-07-01', '2015-07-21', '2015-08-10', '2015-08-30',
       '2015-09-19', '2015-10-09', '2015-10-29', '2015-11-18',
       '2015-12-08', '2015-12-28', '2016-01-17', '2016-02-06',
       '2016-02-26', '2016-03-17', '2016-04-06', '2016-04-26',
       '2016-05-16', '2016-06-05', '2016-06-25', '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 [ ]:
```

```

Checking missing values using plot:

In [14]:
```
plt.figure(figsize=(15, 5))

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

Out[14]: <AxesSubplot:>



- 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 [15]:
```
# Replacing all the null values with 0.

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

In [16]:
```
data.isnull().sum()[indexes]
```

Out[16]:
```
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 [ ]:

EDA:

Extracting Language

In [17]:
```
data.Page[0]
```

Out[17]:
```
'2NE1_zh.wikipedia.org_all-access_spider'
```

In [18]:
```
import re   # re in Python, which provides support for working with regular expressi

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

Out[18]:
```
['zh']
```

In [19]:
```
data.Page.str.findall(pat="_(.{2}).wikipedia.org_").sample(10)
```

Out[19]:
```
122033    [ja]
107534    [zh]
72781     [en]
23987     [fr]
488       [zh]
48253     [de]
5365      [fr]
58989     [ja]
124876    [ru]
121690    [ja]
Name: Page, dtype: object
```

In [20]:
```
# 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 [21]:
```
data["Language"] = data["Page"].map(Extract_Language)
```

In [22]:
```
data["Language"].unique()
```

Out[22]:
```
array(['zh', 'fr', 'en', 'Unknown', 'ru', 'de', 'ja', 'es'], dtype=object)
```

In [23]:
```
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 [24]:

```
data.head()
```

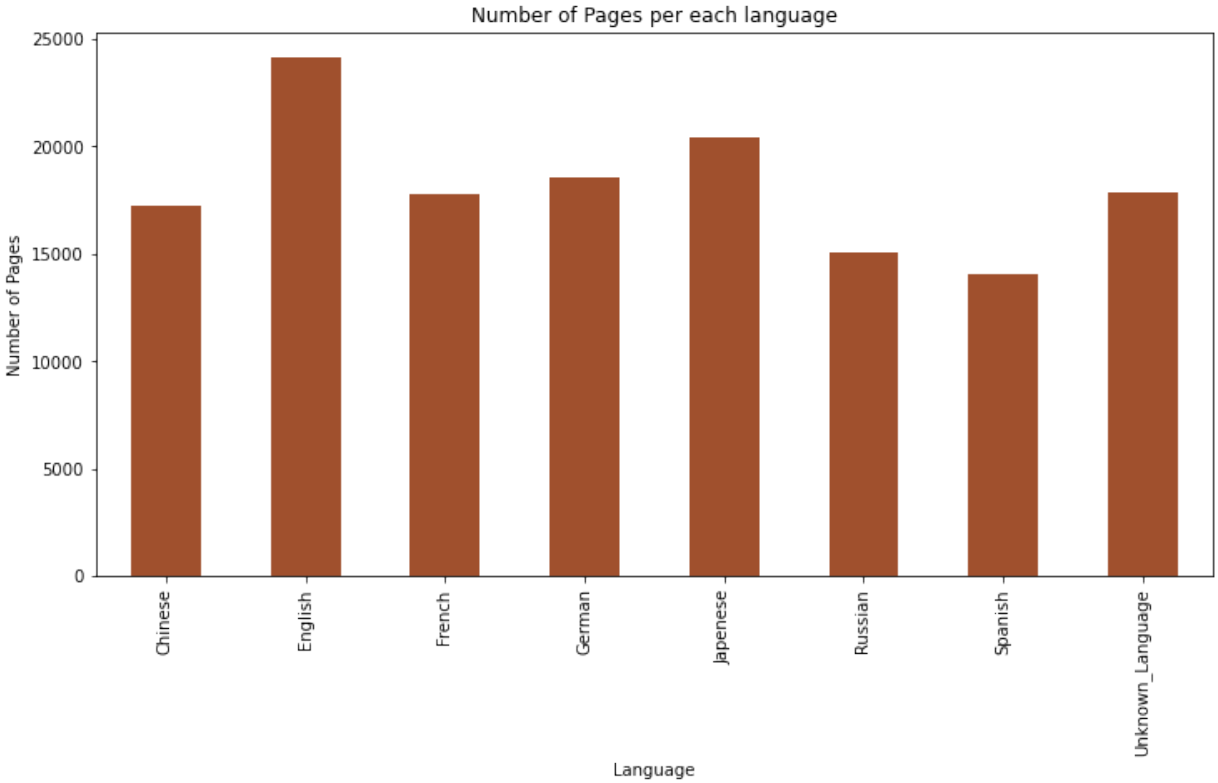
Out[24]:

	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	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	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	1
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	1
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	0.0	

5 rows × 552 columns

In [25]:

```
plt.figure(figsize=(12, 6))

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



In [26]:
```
from locale import normalize
data["Language"].value_counts(normalize=True) * 100
```

Out[26]:
```
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
```

- 12.30 % of pages have unknown language.
- 16.61% of all pages are in English which is highest.

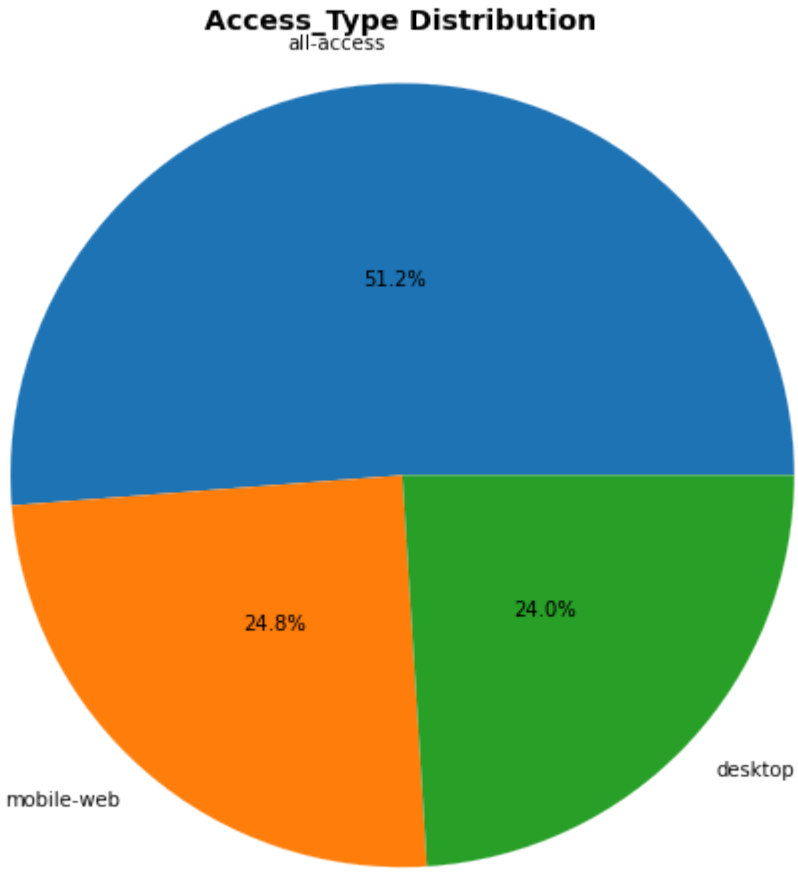
Exrtacting ACCESS TYPE :

In [27]:
```
data["Access_Type"] = data.Page.str.findall(r'all-access|mobile-web|desktop').apply(
```

In [28]:
```
data["Access_Type"].value_counts(dropna=False, normalize=True)
```

Out[28]:
```
all-access    0.512295
mobile-web    0.247748
desktop       0.239958
Name: Access_Type, dtype: float64
```

In [29]:
```
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.figure(figsize=(10, 8))  # Adjust the width and height as needed
plt.pie(x, labels=y, radius=1.5, autopct='%1.1f%%', pctdistance=0.5)
plt.title(f'Access_Type Distribution', fontsize=14, fontweight='bold')  # Adjust fon
plt.axis('equal')
plt.show()
```



In [ ]:

Exrtacting ACCESS ORIGIN :

In [30]:
```
data.Page.sample(20)
```

Out[30]:
```
116444          DVB-T_de.wikipedia.org_mobile-web_all-agents
123565    熊本地震_(1889年)_ja.wikipedia.org_all-access_all-a...
59759           みどりの日_ja.wikipedia.org_mobile-web_all-agents
59482          佐々木つとむ_ja.wikipedia.org_mobile-web_all-agents
99316     Цензура_в_Иране_ru.wikipedia.org_all-access_al...
20062     Manual:Mobiles,_tablets_and_responsive_design_...
70665     Sistema_nervioso_es.wikipedia.org_desktop_all-...
47440     Katharina_II._(Russland)_de.wikipedia.org_all-...
15309     Category:Coitus_of_a_hemisected_man_and_woman_...
55554     Liste_des_partis_et_mouvements_politiques_fran...
784                   范曉萱_zh.wikipedia.org_all-access_spider
144550    House_of_Cards_(serie_de_televisión_de_2013)_e...
134746              田中邦衛_ja.wikipedia.org_all-access_spider
138888    Terence_Hill_de.wikipedia.org_all-access_all-a...
35329     Karan_Singh_Grover_en.wikipedia.org_all-access...
96988     Organización_de_las_Naciones_Unidas_es.wikiped...
10821     Water_polo_at_the_2016_Summer_Olympics_en.wiki...
8440       3_Doors_Down_en.wikipedia.org_desktop_all-agents
142567    Mireia_Belmonte_es.wikipedia.org_all-access_sp...
72755     1896_Summer_Olympics_en.wikipedia.org_mobile-w...
Name: Page, dtype: object
```

In [31]:
```
data.Page.str.findall(r'spider|agents').apply(lambda x:x[0]).isna().sum()
```

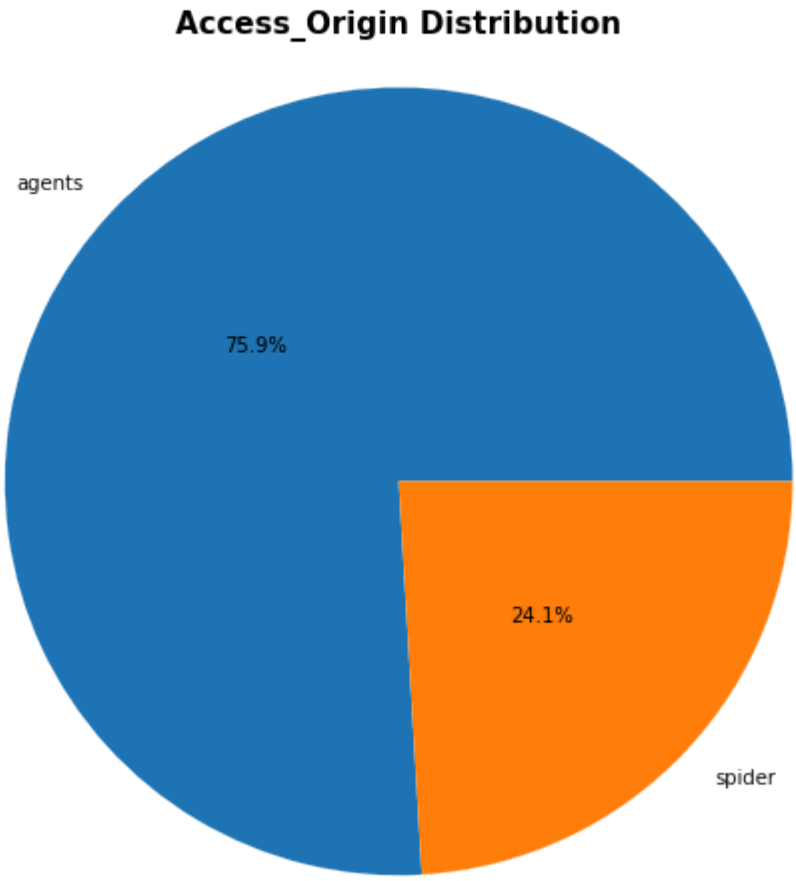
Out[31]:
```
0
```

In [32]:
```
data["Access_Origin"] =  data.Page.str.findall(r'spider|agents').apply(lambda x:x[0]
```

In [33]:
```
data["Access_Origin"].value_counts(dropna= False, normalize=True) * 100
```

Out[33]:
```
agents    75.932526
spider    24.067474
Name: Access_Origin, dtype: float64
```

In [34]:
```
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.figure(figsize=(10, 8))
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 [35]:
```
data
```

Out[35]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	201 07-
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	201 07-
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	2
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	1
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
145058	Underworld_(serie_de_películas)_es.wikipedia.o...	0.0	0.0	0.0	0.0	0.0	0.0	
145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org...	0.0	0.0	0.0	0.0	0.0	0.0	
145060	Enamorándome_de_Ramón_es.wikipedia.org_all-acc...	0.0	0.0	0.0	0.0	0.0	0.0	
145061	Hasta_el_último_hombre_es.wikipedia.org_all-ac...	0.0	0.0	0.0	0.0	0.0	0.0	
145062	Francisco_el_matemático_(serie_de_televisión_d...	0.0	0.0	0.0	0.0	0.0	0.0	

145063 rows × 554 columns

In [36]:
```
data.groupby("Language").mean()
```

Out[36]:

Language	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2
Chinese	240.582042	240.941958	239.344071	241.653491	257.779674	259.114864	
English	3513.862203	3502.511407	3325.357889	3462.054256	3575.520035	3849.736021	3
French	475.150994	478.202000	459.837659	491.508932	482.557746	502.741209	
German	714.968405	705.229741	676.877231	621.145145	722.076185	794.832480	
Japenese	580.647056	666.672801	602.289805	756.509177	725.720914	632.399148	
Russian	629.999601	640.902876	594.026295	558.728132	595.029157	640.986287	
Spanish	1085.972919	1037.814557	954.412680	896.050750	974.508210	1110.637145	1
Unknown_Language	83.479922	87.471857	82.680538	70.572557	78.214562	89.720190	

8 rows × 550 columns

In [37]:
```
pd.set_option('display.max_rows', 500)
```

In [38]:
```
aggregated_data = data.groupby("Language").mean().T.drop("Unknown_Language",axis = 1
aggregated_data["index"] = pd.to_datetime(aggregated_data["index"])
aggregated_data = aggregated_data.set_index("index")
aggregated_data
```

Out[38]:

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 [39]:
```
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 [40]:
```
aggregated_data.index
```

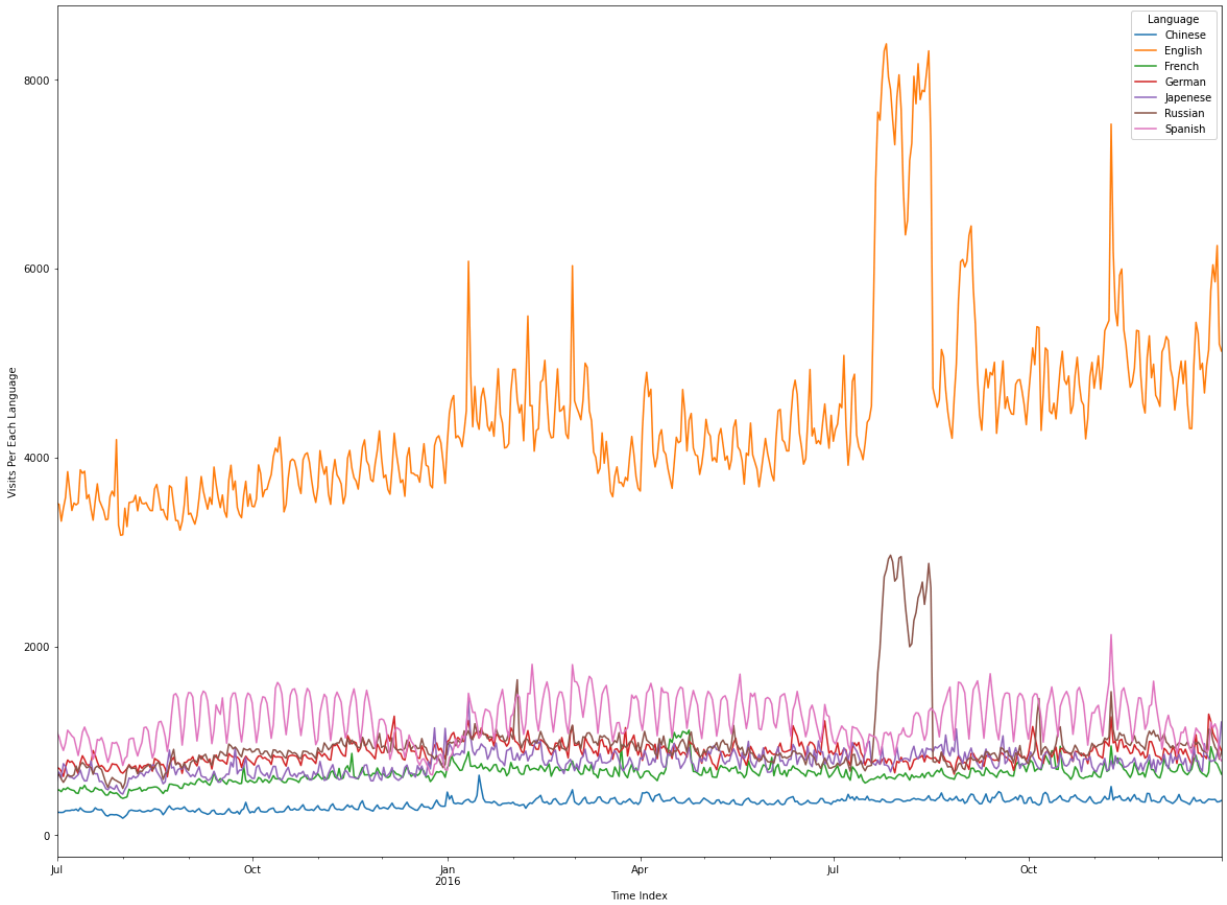
Out[40]:
```
DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',
               '2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08',
               '2015-07-09', '2015-07-10',
               ...
               '2016-12-22', '2016-12-23', '2016-12-24', '2016-12-25',
               '2016-12-26', '2016-12-27', '2016-12-28', '2016-12-29',
               '2016-12-30', '2016-12-31'],
              dtype='datetime64[ns]', name='index', length=550, freq=None)
```

In [ ]:

Visualising Time Series for each languages:

In [41]:

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

aggregated_data.plot()

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



- As per plot, english language is most visited language w.r.t time index. In month of July to September it is in peak.

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-Stationary if p-value <= 0.05: we reject Null Hypothesis
- that means the time series in Stationary

In [42]:
```
import statsmodels.api as sm
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 [43]:
```
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:  0.44744579229311526
None


English
Time Series is NOT Stationary
P_value is:  0.18953359279992338
None


French
Time Series is NOT Stationary
P_value is:  0.05149502195245785
None


German
Time Series is NOT Stationary
P_value is:  0.1409738231972908
None


Japenese
Time Series is NOT Stationary
P_value is:  0.10257133898557586
None


Russian
Time Series is Stationary
P_value is:  0.0018649376536617962
None


Spanish
Time Series is Stationary
P_value is:  0.03358859084479102
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.

Further analysing Time Series for English Language Pages Visits :

In [44]:
```
TS_English = aggregated_data.English
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
    for key, value in dftest[4].items():
       df_output['Critical Value (%s)' %key] = value
    print (df_output)
```

In [45]:
```
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
```

- Augmented Dickey-Fuller (ADF) test, a statistical test used to determine whether a unit root is present in a time series dataset.

In [46]:
```
Dickey_Fuller_test(TS_English)
```

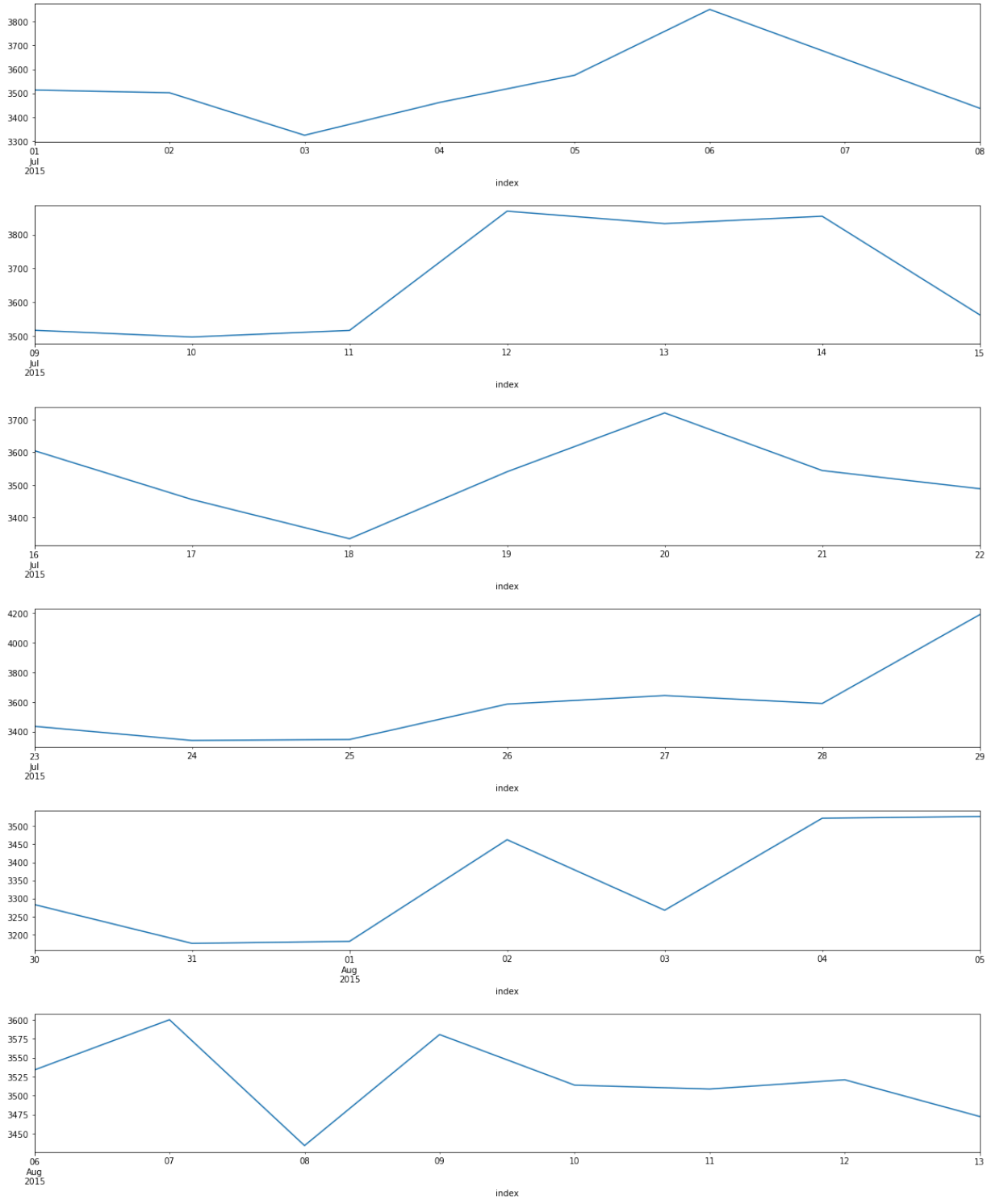
```
Time Series is NOT Stationary
P_value is:  0.18953359279992338
```

In [ ]:
```

```

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

In [47]:
```
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()
```



In [48]:

```
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 0.9363434527458434
2 0.8682966716039895
3 0.8185418037184545
4 0.7846718829500339
5 0.7612561076942571
6 0.7542260641783561
```

```
7 0.7386829287516693
8 0.6912638018189882
9 0.6370978014300401
10 0.6015277501876305
11 0.5825450402423569
12 0.5812931934793535
13 0.6007266462817787
14 0.6142525351445116
15 0.5971084554755529
16 0.5693834937428244
17 0.5488401467532631
18 0.5377431132136109
19 0.5430816743411206
20 0.5552694244923044
21 0.5540623423718065
22 0.5092655604869362
23 0.4537369557681357
24 0.41123362976203237
25 0.38162860616251715
26 0.365199631669948
27 0.37236036273026
28 0.37818226683160033
29 0.3593924266732815
```

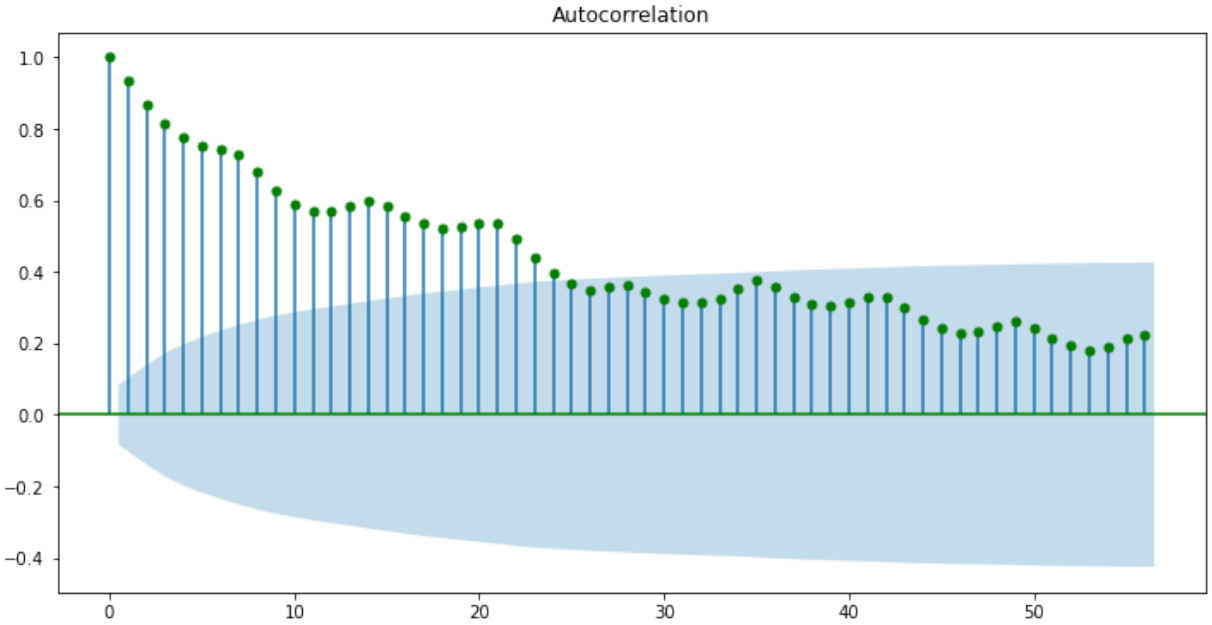
In [ ]:

Time Series Decomposition

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

In [49]:

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

from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt

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



In [50]:

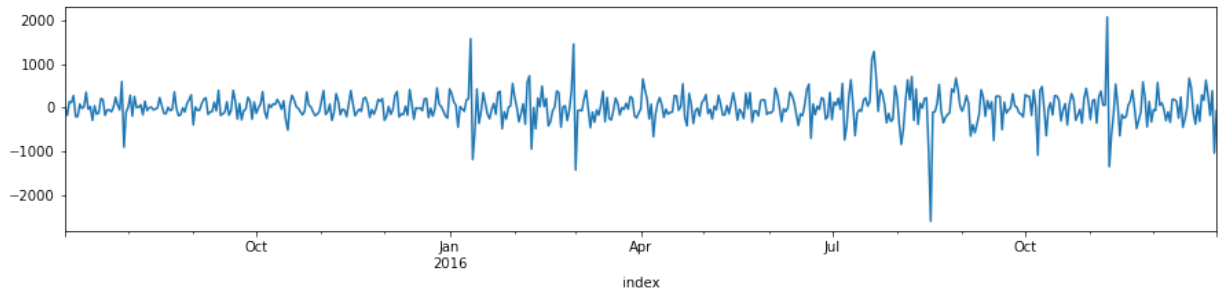
```
plt.rcParams['figure.figsize'] = (15, 10)

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

In [51]:
```
Dickey_Fuller_test(pd.Series(Decomposition_model.resid).fillna(0))
```

```
Time Series is Stationary
P_value is:  3.727526947812843e-21
```

In [52]:
```
# Taking the first differentiation of the time series and plotting

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

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

Out[52]: <AxesSubplot:xlabel='index'>



In [53]:
```
Dickey_Fuller_test(TS_English.diff(1).dropna())
```

```
Time Series is Stationary
P_value is:  5.292474635436519e-13
```

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

In [54]:
```
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))
```

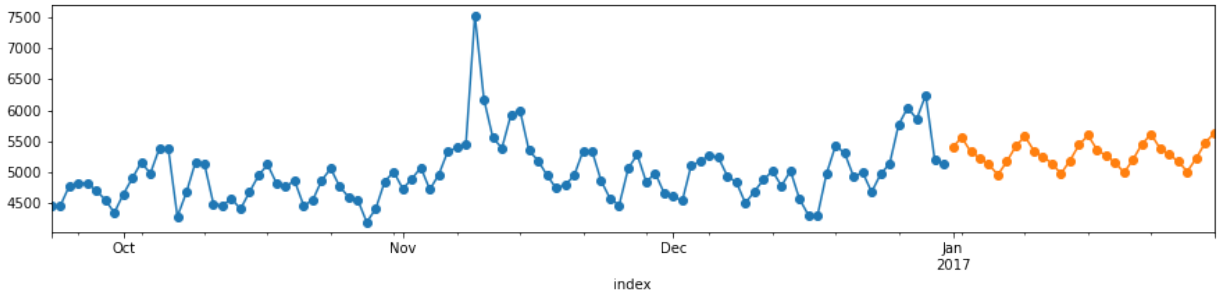
In [ ]:

Forecasting :

In [55]:

```
TS_English.index.freq = 'D'


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

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

Out[55]: <AxesSubplot:xlabel='index'>

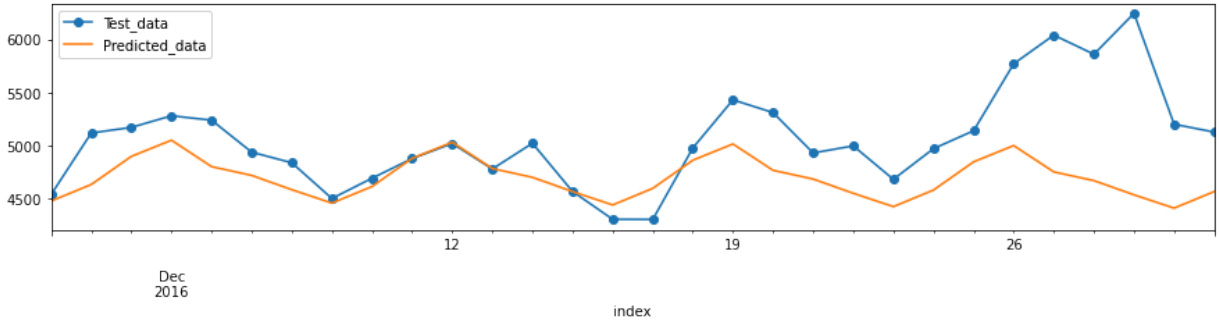


In [56]:

```
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 : 395.144
RMSE : 561.078
MAPE: 0.073
```



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

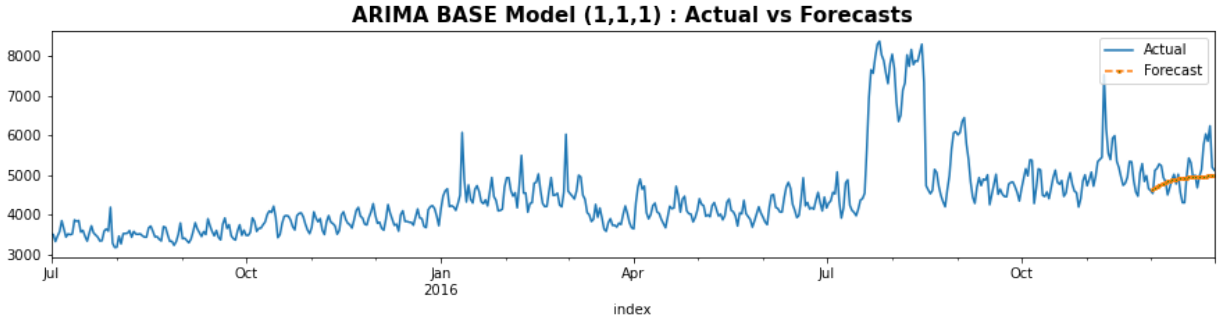
In [57]:

```
from statsmodels.tsa.arima.model import ARIMA
TS = TS_English.copy(deep=True)
```

In [58]:

```
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='g
plt.legend(loc="upper right")
plt.title('ARIMA BASE Model (1,1,1) : Actual vs Forecasts', fontsize = 15, fontweigh
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 :

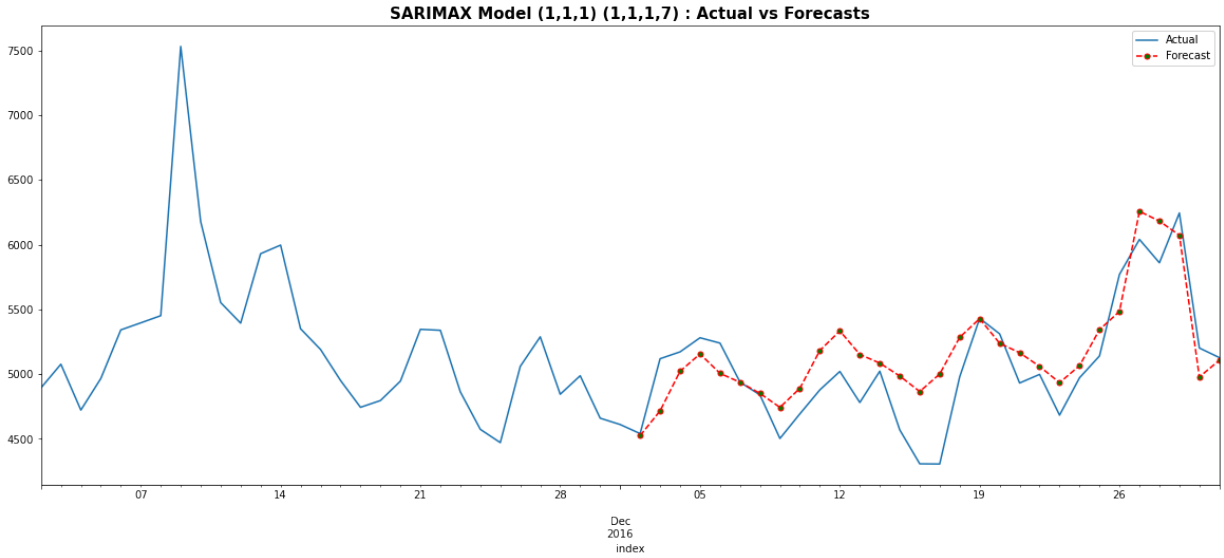
In [59]:

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

    #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'
    plt.legend(loc="upper right")
    plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Forecasts'
    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 [60]:

```
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 : 0.04457
RMSE of Model : 273.007
```

Hyperparamer tuning for SARIMAX model

In [61]:

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

    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:
                                try:
                                    model = SARIMAX(time_series[:-n], order=(p, d, q
                                                    exog=exog[:-n], initialization='
                                    model_fit = model.fit()

                                    model_forecast = model_fit.forecast(n, dynamic=T
                                                                        exog=pd.Dat

                                    actuals = time_series.values[-n:]
                                    errors = time_series.values[-n:] - model_forecas

                                    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)

                                    counter += 1
                                    list_row = [counter, (p, d, q), (P, D, Q, s), ma
                                    param_df.loc[len(param_df)] = list_row

                                    print(f'Possible Combination: {counter} out of {
                                except:
                                    continue

    return param_df
```

In [62]:

```
#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 = [1]

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

In [63]:

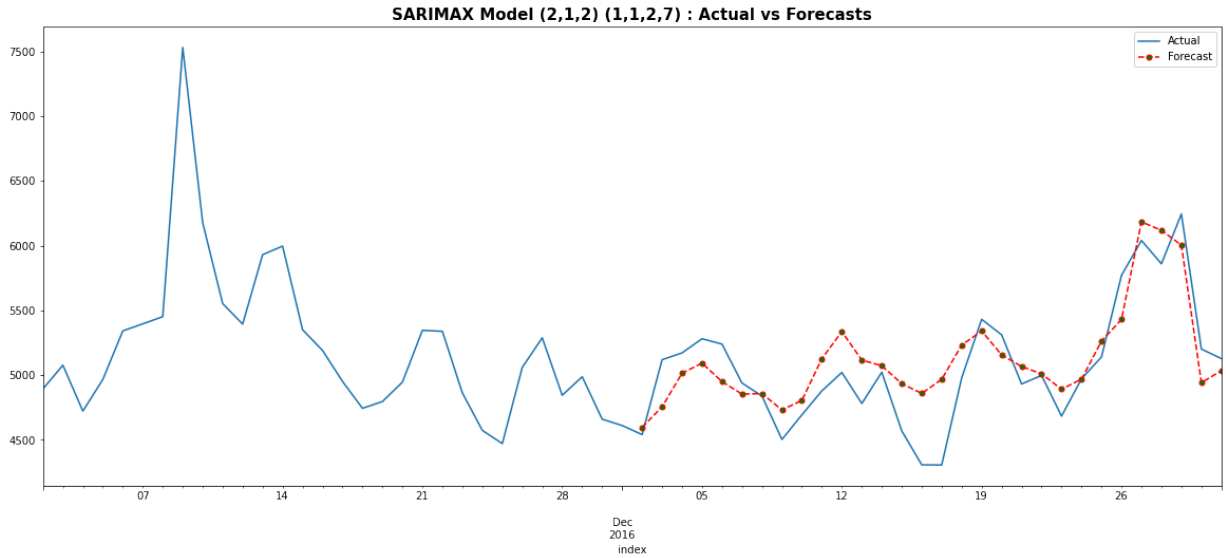
```
english_params.sort_values(['mape', 'rmse']).head()
```

Out[63]:

	serial	pdq	PDQs	mape	rmse

In [64]:

```
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 : 0.04284
RMSE of Model : 259.226
```

Hyperparameter tuning for all other languages :

In [65]:

```
def pipeline_sarimax_grid_search_without_exog(languages, data, n, param, d_param, s_

    best_param_df  = pd.DataFrame(columns = ['language','p','d', 'q', 'P','D','Q','s
    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_diff
                                    model_fit = model.fit()

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

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

                                    #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)*

        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,
        print(f'-----------------------------------------------------------------')

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

```
    return best_param_df
```

In [66]:

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


best_param_df = pipeline_sarimax_grid_search_without_exog(languages, aggregated_data
```

```
------------------------------------------------------------
          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 = 0.04482
Corresponding Best Parameters are (0, 1, 0, 1, 0, 1, 2)
------------------------------------------------------------


------------------------------------------------------------
          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 = 0.06621
Corresponding Best Parameters are (1, 1, 1, 1, 0, 0, 2)
------------------------------------------------------------
```

```
------------------------------------------------------------
          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 = 0.07522
Corresponding Best Parameters are (1, 1, 1, 0, 1, 1, 2)
------------------------------------------------------------


------------------------------------------------------------
          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 = 0.08235
Corresponding Best Parameters are (0, 1, 0, 1, 0, 1, 2)
------------------------------------------------------------


------------------------------------------------------------
          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 = 0.07913
Corresponding Best Parameters are (1, 1, 0, 0, 1, 1, 2)
------------------------------------------------------------


------------------------------------------------------------
          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 = 0.05075
Corresponding Best Parameters are (0, 0, 0, 1, 0, 1, 2)
------------------------------------------------------------


------------------------------------------------------------
          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 = 0.10847
Corresponding Best Parameters are (0, 0, 1, 1, 0, 0, 2)
------------------------------------------------------------
```

In [67]:

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

Out[67]:

	language	p	d	q	P	D	Q	s	mape
0	Chinese	0	1	0	1	0	1	2	0.04482
5	Russian	0	0	0	1	0	1	2	0.05075
1	English	1	1	1	1	0	0	2	0.06621
2	French	1	1	1	0	1	1	2	0.07522
4	Japenese	1	1	0	0	1	1	2	0.07913
3	German	0	1	0	1	0	1	2	0.08235
6	Spanish	0	0	1	1	0	0	2	0.10847

In [68]:

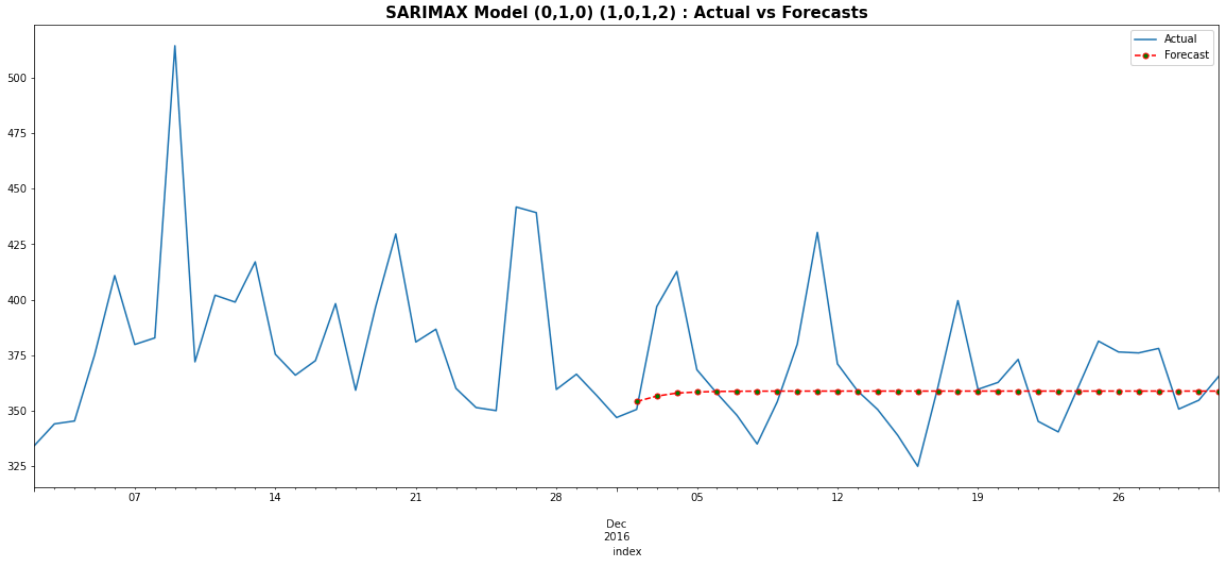
```
def plot_best_SARIMAX_model(languages, data, n, best_param_df):

    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='gr
```

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

    return 0
```

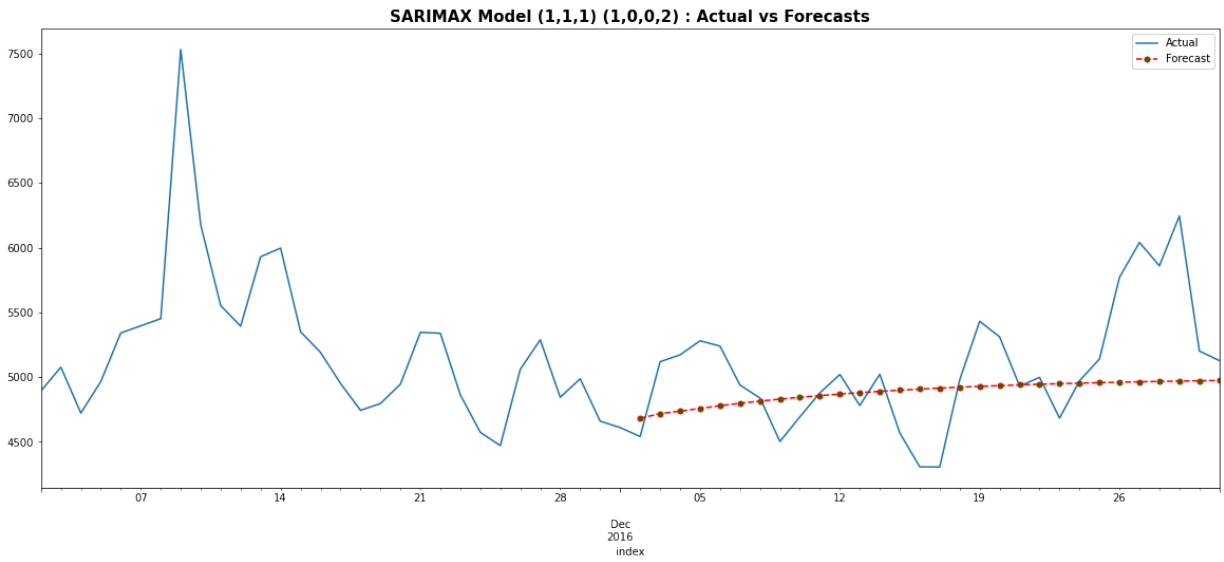
In [69]:

```
#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)
```

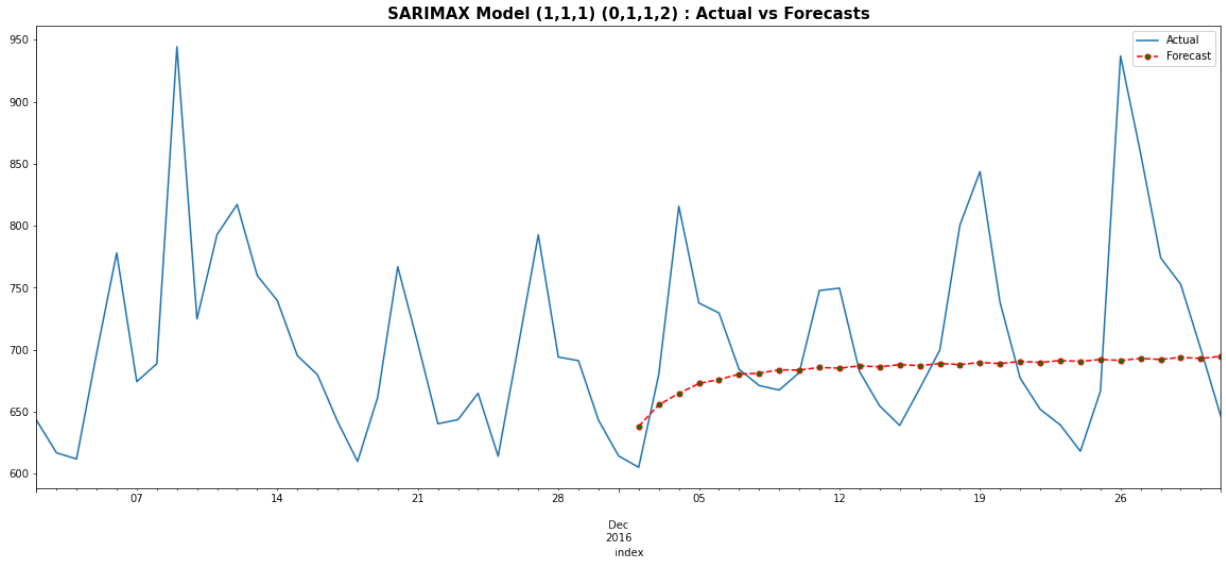
```
--------------------------------------------------------------------------------
----
        SARIMAX model for Chinese Time Series
        Parameters of Model : (0,1,0) (1,0,1,2)
        MAPE of Model       : 0.04482
        RMSE of Model       : 23.669
--------------------------------------------------------------------------------
----
```



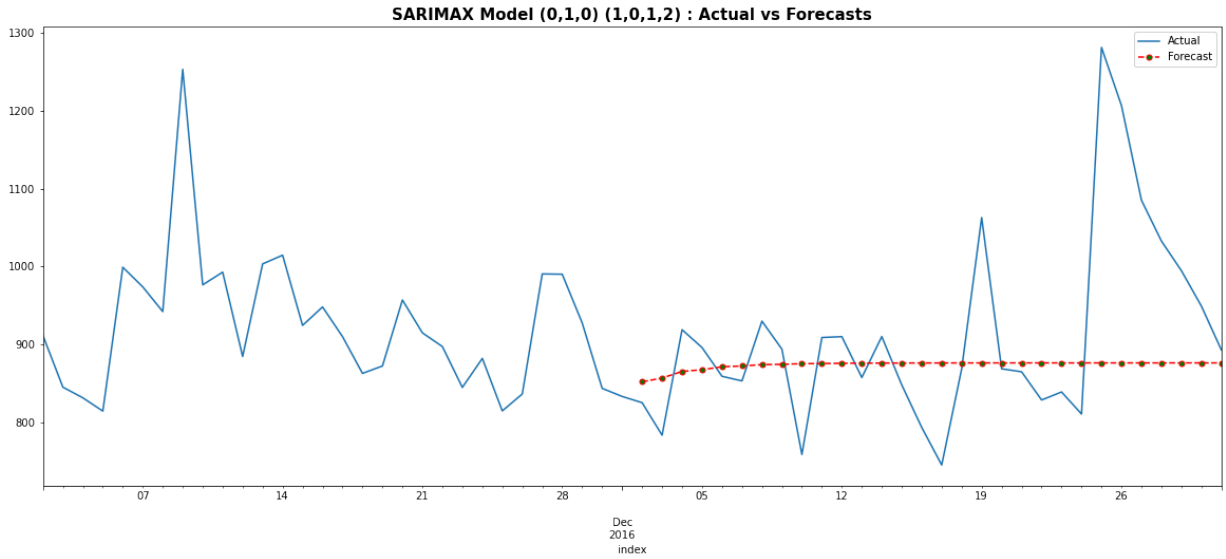

```
--------------------------------------------------------------------------------
----
        SARIMAX model for English Time Series
        Parameters of Model : (1,1,1) (1,0,0,2)
        MAPE of Model       : 0.06621
        RMSE of Model       : 473.207
--------------------------------------------------------------------------------
----
```



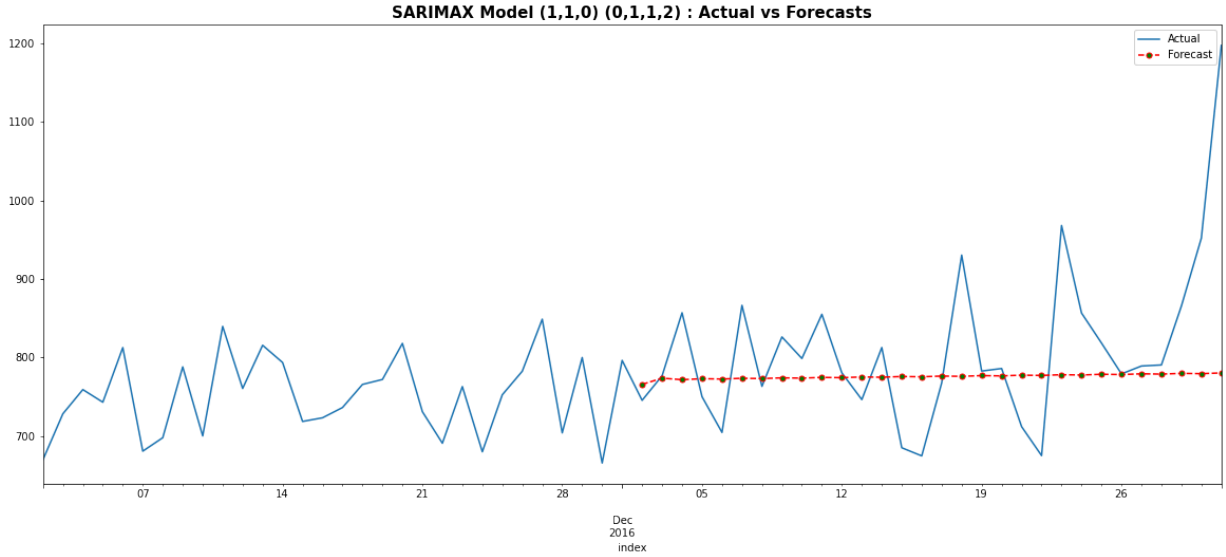
```
--------------------------------------------------------------------------------
----
        SARIMAX model for French Time Series
        Parameters of Model : (1,1,1) (0,1,1,2)
        MAPE of Model       : 0.07522
        RMSE of Model       : 80.192
--------------------------------------------------------------------------------
----
```



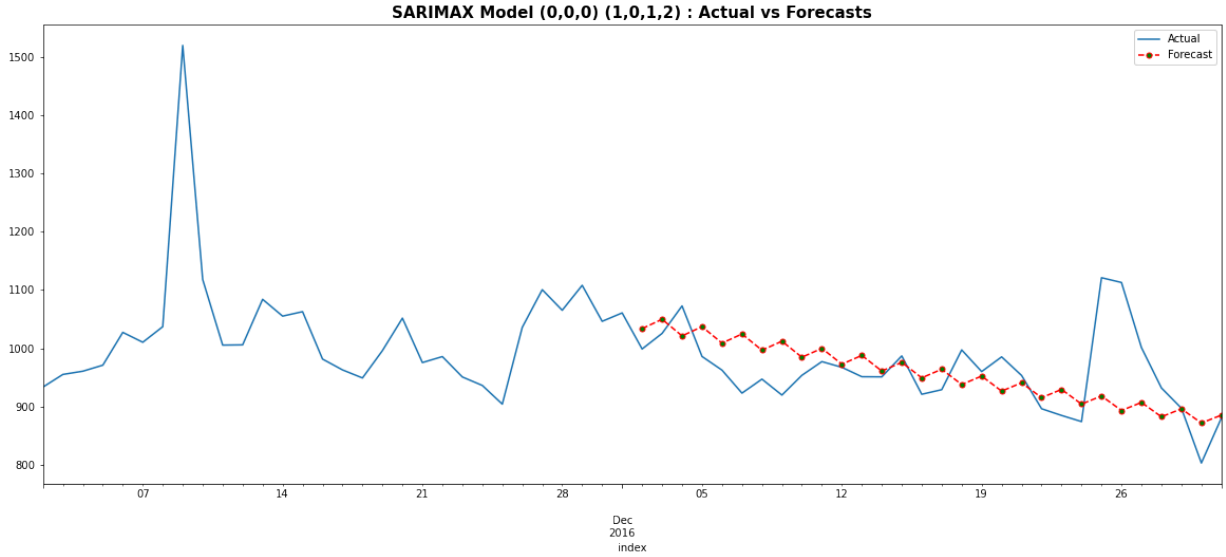

```
--------------------------------------------------------------------------------
----
        SARIMAX model for German Time Series
        Parameters of Model : (0,1,0) (1,0,1,2)
        MAPE of Model       : 0.08235
        RMSE of Model       : 123.736
--------------------------------------------------------------------------------
----
```



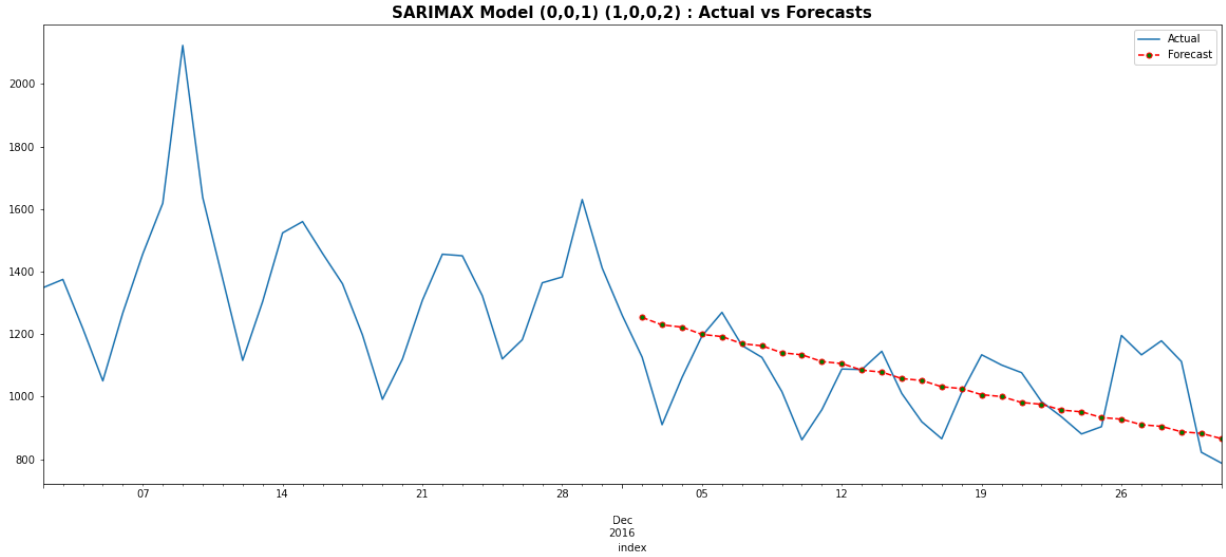

```
--------------------------------------------------------------------------------
----
        SARIMAX model for Japenese Time Series
        Parameters of Model : (1,1,0) (0,1,1,2)
        MAPE of Model       : 0.07913
        RMSE of Model       : 107.35
--------------------------------------------------------------------------------
----
```



SARIMAX model for Russian Time Series
Parameters of Model : (0,0,0) (1,0,1,2)
MAPE of Model : 0.05075
RMSE of Model : 71.022



SARIMAX model for Spanish Time Series
Parameters of Model : (0,0,1) (1,0,0,2)
MAPE of Model : 0.10847
RMSE of Model : 143.24



Out[69]: 0

Forecasting using Facebook Prophet :

In [81]:
```
!pip install numpy==1.22.4
```

```
Collecting numpy==1.22.4
  Downloading numpy-1.22.4-cp38-cp38-win_amd64.whl (14.8 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.24.4
    Uninstalling numpy-1.24.4:
      Successfully uninstalled numpy-1.24.4
Successfully installed numpy-1.22.4
```

In [82]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in c:\users\asus\anaconda3\lib\site-packages
(1.1.5)
Requirement already satisfied: holidays>=0.25 in c:\users\asus\anaconda3\lib\site-pac
kages (from prophet) (0.48)
Requirement already satisfied: numpy>=1.15.4 in c:\users\asus\anaconda3\lib\site-pack
ages (from prophet) (1.22.4)
Requirement already satisfied: tqdm>=4.36.1 in c:\users\asus\anaconda3\lib\site-packa
ges (from prophet) (4.59.0)
Requirement already satisfied: pandas>=1.0.4 in c:\users\asus\anaconda3\lib\site-pack
ages (from prophet) (1.2.4)
Requirement already satisfied: cmdstanpy>=1.0.4 in c:\users\asus\anaconda3\lib\site-p
ackages (from prophet) (1.2.2)
Requirement already satisfied: importlib-resources in c:\users\asus\anaconda3\lib\sit
e-packages (from prophet) (6.4.0)
Requirement already satisfied: matplotlib>=2.0.0 in c:\users\asus\anaconda3\lib\site-
packages (from prophet) (3.3.4)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in c:\users\asus\anaconda3\lib\si
te-packages (from cmdstanpy>=1.0.4->prophet) (0.5.0)
Requirement already satisfied: python-dateutil in c:\users\asus\anaconda3\lib\site-pa
ckages (from holidays>=0.25->prophet) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\a
sus\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet) (3.1.2)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packa
ges (from matplotlib>=2.0.0->prophet) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-
packages (from matplotlib>=2.0.0->prophet) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\asus\anaconda3\lib\site-pack
```

```
ages (from matplotlib>=2.0.0->prophet) (8.2.0)
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packages (from
cycler>=0.10->matplotlib>=2.0.0->prophet) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\asus\anaconda3\lib\site-packa
ges (from pandas>=1.0.4->prophet) (2021.1)
Requirement already satisfied: zipp>=3.1.0 in c:\users\asus\anaconda3\lib\site-packag
es (from importlib-resources->prophet) (3.4.1)
```

In [83]:
```
from prophet import Prophet
```

In [84]:
```
time_series = aggregated_data.reset_index()
time_series = time_series[['index', 'English']]
time_series.columns = ['ds', 'y']
time_series['ds'] = pd.to_datetime(time_series['ds'])  # Ensure 'ds' column is in da

exog = Exog_Campaign_eng.copy(deep=True)
time_series['exog'] = exog.values
```

In [86]:
```
time_series
```

Out[86]:

	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

Insights:

- Conclusions drawn from the data visuals:
- There were 7 languages in all in the data.
- The language with the most pages is English.
- Three categories of access:

1. 51.2295 % of All-access
2. Mobile web 24.7748 percent

3. 23.9958 % on a desktop

- Two points of access:

a. The language with the most pages is English (74.932526%); Agents (74.932526%); Spider (24.067474%).

b. The English page ought to run the most advertising possible.

- What distinguishes arima from sarima and sarimax:

1. Arima:

- A statistical model for time series data called ARIMA (AutoRegressive Integrated Moving Average) takes into consideration both moving average and autoregression, which both involve using the residuals of previous predictions to predict future values.
- It is a versatile technique that works with both univariate and multivariate time series to model non-stationary time series data.
- The notations ARIMA(p, d, q) are used to represent ARIMA models, where p represents the autoregression component's order, d represents the differencing order that stabilizes the time series, and q represents the moving average component's order.

1. Sarima:

- An adaptation of ARIMA that takes into consideration time series data's non-stationarity and seasonality is called SARIMA (Seasonal AutoRegressive Integrated Moving Average).
- Recurring patterns in data across predetermined time intervals—daily, weekly, or annual—are referred to as seasonality. SARIMA models are identified by the notations SARIMA(p, d, q)(P, D, Q, S), where P is the order of the seasonal moving average component, Q is the order of the seasonal autoregression component, S is the number of seasons in the data, and p, d, and q are the same as in ARIMA models.

1. Sarimax:

- An modification of SARIMA that permits the use of exogenous variables—that is, factors not included in the time series data—in the modeling process is called SARIMAX (Seasonal AutoRegressive Integrated Moving Average with exogenous regressors).
- SARIMAX models can produce more accurate forecasts and are helpful when the time series data is impacted by variables outside of the time series data.
- The notations SARIMAX(p, d, q)(P, D, Q, S)x are used to describe SARIMAX models. In these notations, p, d, q, P, D, Q, and S are the same as in SARIMA models, and x is the number of exogenous variables that are included in the model.

Recommendations :

- To ascertain whether the time series is stationary, do an enhanced Dickey-Fuller test.
- Fit an ARMA model if the time series is stationary. If it is non-stationary, find out what d is.
- Plot the data's autocorrelation and partial autocorrelation graphs once stationarity has been attained.
- Since the cut-off point in the partial autocorrelation graph (PACF) equals p, plot the PACF to find the value of p.
- Plot the autocorrelation graph (ACF) to find q, since q is the value of the ACF's cut-off point.

In [ ]: