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.

Importing Libraries

In [5]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:
```
df = pd.read_csv("/Users/bose/Downloads/train_1.csv")
```

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

Out[7]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-0
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 551 columns

In [8]:
```
df.shape
```

Out[8]:
```
(145063, 551)
```

In [221…
```
df2 = pd.read_csv("/Users/bose/Downloads/Exog_Campaign_eng")
```

In [10]:
```
df2.head()
```

Out[10]:

	Exog
0	0
1	0
2	0
3	0
4	0

In [11]:
```
df2.shape
```

Out[11]:
```
(550, 1)
```

In [12]:
```
df.Page.sample(15)
```

Out[12]:
```
143655    Juan_Manuel_Fangio_es.wikipedia.org_all-access...
19396     Дом_странных_детей_ru.wikipedia.org_mobile-web...
38822        Parched_en.wikipedia.org_all-access_all-agents
104653    Список_эпизодов_телесериала_«Доктор_Хаус»_ru.w...
47909     Oroville-Staudamm_de.wikipedia.org_all-access_...
3631                 娜璉_zh.wikipedia.org_all-access_spider
74777     T._J._Miller_en.wikipedia.org_mobile-web_all-a...
121647         田中美佐子_ja.wikipedia.org_all-access_all-agents
102967    Белка_и_Стрелка_ru.wikipedia.org_desktop_all-a...
57816           鏡開き_ja.wikipedia.org_mobile-web_all-agents
136518            11月11日_ja.wikipedia.org_all-access_spider
2638          雲之彼端，約定的地方_zh.wikipedia.org_all-access_spider
89963           テリーザ・メイ_ja.wikipedia.org_desktop_all-agents
22921     User_talk:Syum90_www.mediawiki.org_mobile-web_...
26013            Iran_fr.wikipedia.org_all-access_all-agents
Name: Page, dtype: object
```

In [13]:
```
df['Page'].str.extract(r'[^_]+_[^_]+_[^_]+_([^_]+)').head(20)
```

Out[13]:

	0
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

In [14]:
```
data = df.copy()
```

In [15]:
```
data.duplicated().sum()
```

Out[15]:
```
0
```

In [16]:
```
data.dtypes.sample(10)
```

Out[16]:
```
2016-05-08    float64
2015-11-24    float64
2016-05-14    float64
2016-01-31    float64
2015-12-28    float64
2015-07-14    float64
2016-10-03    float64
2016-11-10    float64
2015-08-09    float64
2016-12-18    float64
dtype: object
```

In [17]:
```
indexes = data.head(2).columns[1:][range(0,549,20)].values
```

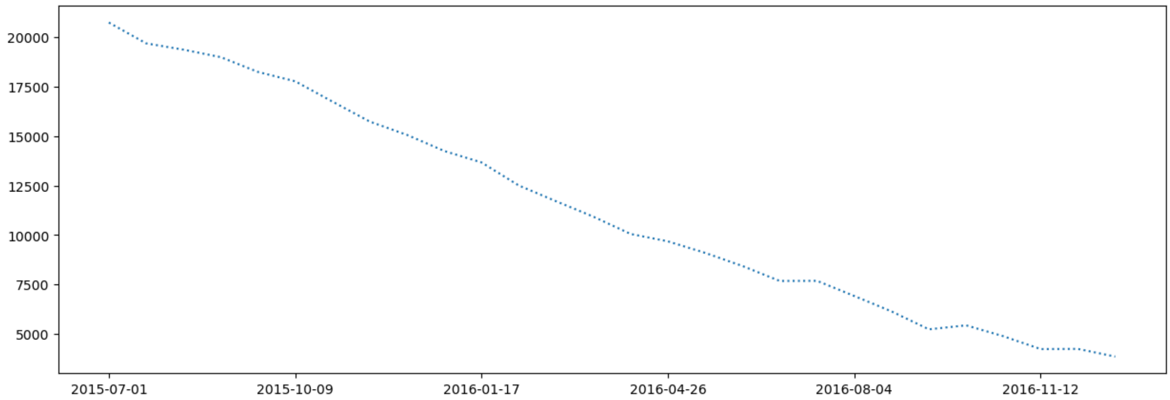
```
indexes
```

Out[17]:
```
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)
```

Checking missing values using plot

In [19]:
```
plt.figure(figsize=(15, 5))
data.isna().sum()[indexes].plot(linestyle='dotted')
```

Out[19]: <Axes: >



- There is a trend of decreasing null values
- Recent dates have lesser null values

In [21]:
```
# Replacing all the null values with 0.
data.fillna(0,inplace =True)
```

In [22]:
```
data.isnull().sum()[indexes]
```

Out[22]:
```
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
```

Extracting Language

In [24]:
```
data.Page[0]
```

Out[24]:
```
'2NE1_zh.wikipedia.org_all-access_spider'
```

In [25]:
```
import re
re.findall(r'_(.{2}).wikipedia.org_', "2NE1_zh.wikipedia.org_all-access_s
```

Out[25]:
```
['zh']
```

In [26]:
```
data.Page.str.findall(pat="_(.{2}).wikipedia.org_").sample(10)
```

Out[26]:
```
9444      [en]
8799      [en]
64376     [zh]
143463    [es]
115461    [de]
23675     [fr]
89065     [ja]
787       [zh]
89149     [ja]
125501    [ru]
Name: Page, dtype: object
```

In [27]:
```
#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 [28]:
```
data["Language"] = data["Page"].map(Extract_Language)
```

In [29]:
```
data["Language"].unique()
```

Out[29]:
```
array(['zh', 'fr', 'en', 'Unknown', 'ru', 'de', 'ja', 'es'], dtype=objec
t)
```

In [30]:
```
dict_ ={'de':'German',
        'en':'English',
        'es': 'Spanish',
        'fr': 'French',
        'ja': 'Japenese' ,
        'ru': 'Russian',
        'zh': 'Chinese',
        'Unknown': 'Unknown_Language'}
data["Language"] = data["Language"].map(dict_)
```

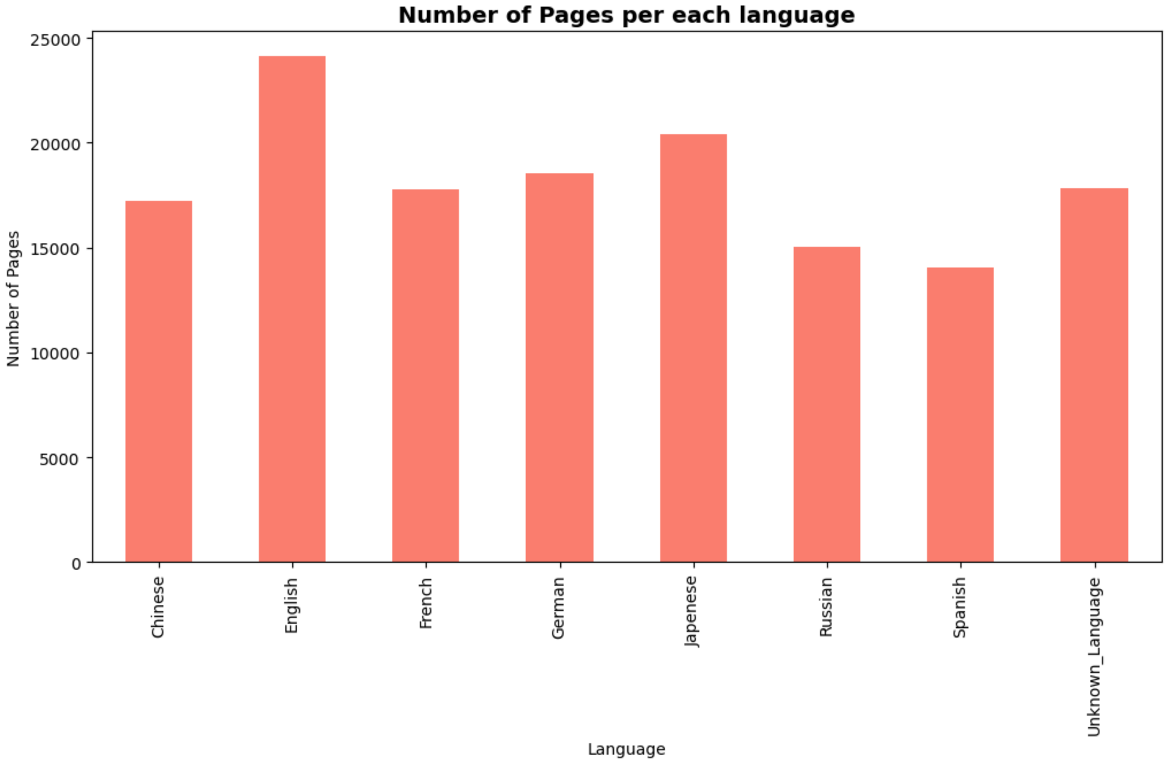
In [31]:
```
data.head()
```

Out[31]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-0
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.

5 rows × 552 columns

In [32]:
```
plt.figure(figsize=(12, 6))
data.groupby("Language")["Page"].count().plot(kind="bar", color='salmon')
plt.xlabel("Language")
plt.ylabel("Number of Pages")
plt.title("Number of Pages per each language",fontsize=14, fontweight='bo
plt.show()
```



In [33]:
```
from locale import normalize
data["Language"].value_counts(normalize=True) * 100
```

Out[33]:
```
Language
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: proportion, dtype: float64
```

- English is the most common language accounting for 16.6 %
- Spanish is the least common language accounting for 9.69 %

Extracting Access Type

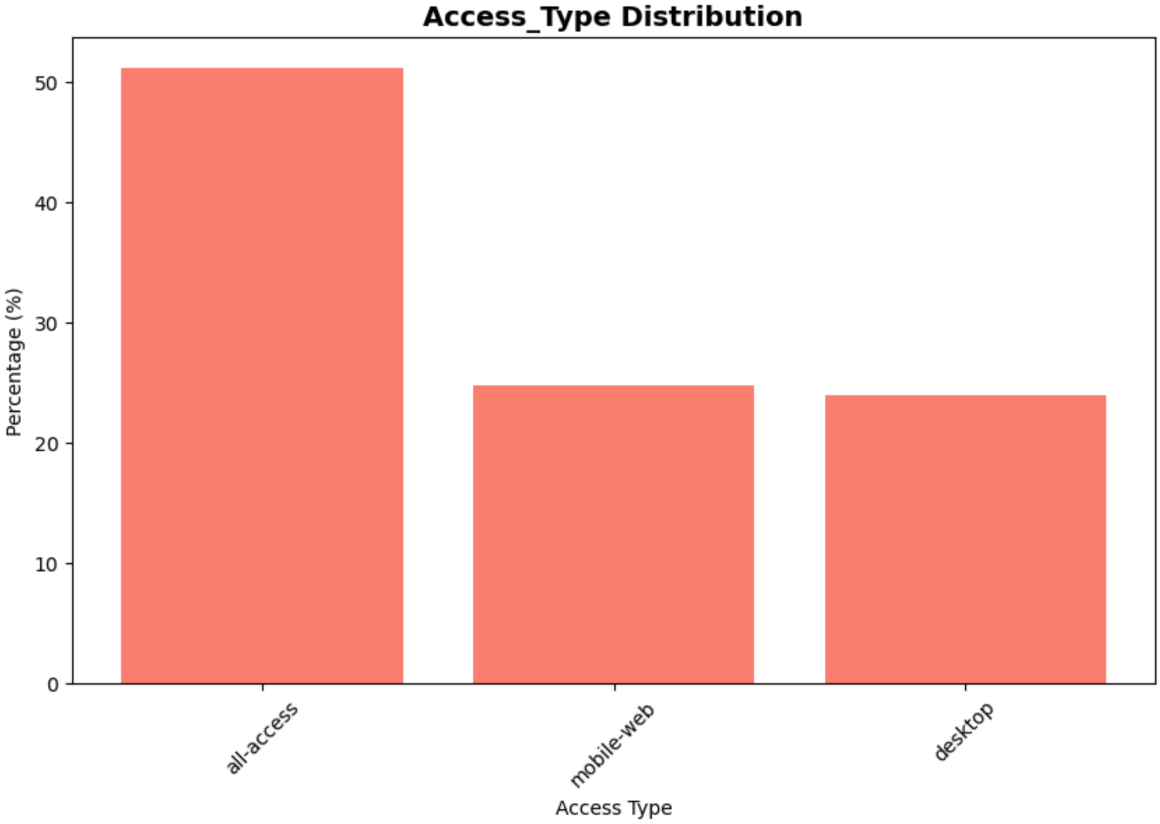
In [36]:
```
data["Access_Type"] = data.Page.str.findall(r'all-access|mobile-web|deskt
```

In [37]:
```
data["Access_Type"].value_counts(dropna=False, normalize=True)
```

Out[37]:
```
Access_Type
all-access    0.512295
mobile-web    0.247748
desktop       0.239958
Name: proportion, dtype: float64
```

In [38]:
```
x = (data["Access_Type"].value_counts(dropna=False, normalize=True) * 100
y = (data["Access_Type"].value_counts(dropna=False, normalize=True) * 100

plt.figure(figsize=(10, 6))
plt.bar(y, x, color='salmon')
```

```
plt.xlabel('Access Type')
plt.ylabel('Percentage (%)')
plt.title('Access_Type Distribution', fontsize=14, fontweight='bold')
plt.xticks(rotation=45)
plt.show()
```



Extracting Access Origin

In [40]:
```
data.Page.sample(15)
```

Out[40]:
```
59529          太田莉菜_ja.wikipedia.org_mobile-web_all-agents
66885     Titanic_(1997)_de.wikipedia.org_desktop_all-ag...
112056    Martine_McCutcheon_en.wikipedia.org_all-access...
92254     Copa_Conmebol_Sudamericana_2017_es.wikipedia.o...
64321               大撲_zh.wikipedia.org_desktop_all-agents
67513     Florence_Nightingale_de.wikipedia.org_desktop_...
129447    Tournoi_des_Six_Nations_2017_fr.wikipedia.org_...
97993        Вьетнам_ru.wikipedia.org_all-access_all-agents
142397    Lemmy_Kilmister_es.wikipedia.org_all-access_sp...
54397     Capucine_Anav_fr.wikipedia.org_mobile-web_all-...
98473     Государственный_комитет_по_чрезвычайному_полож...
21340     Wikimedia_Developer_Summit/2017/Program_www.me...
87209     真田丸_(NHK大河ドラマ)_ja.wikipedia.org_desktop_all-ag...
83276     Help:Logging_in/az_www.mediawiki.org_all-acces...
93234     Copa_Libertadores_de_América_es.wikipedia.org_...
Name: Page, dtype: object
```

In [41]:
```
data.Page.str.findall(r'spider|agents').apply(lambda x:x[0]).isna().sum()
```

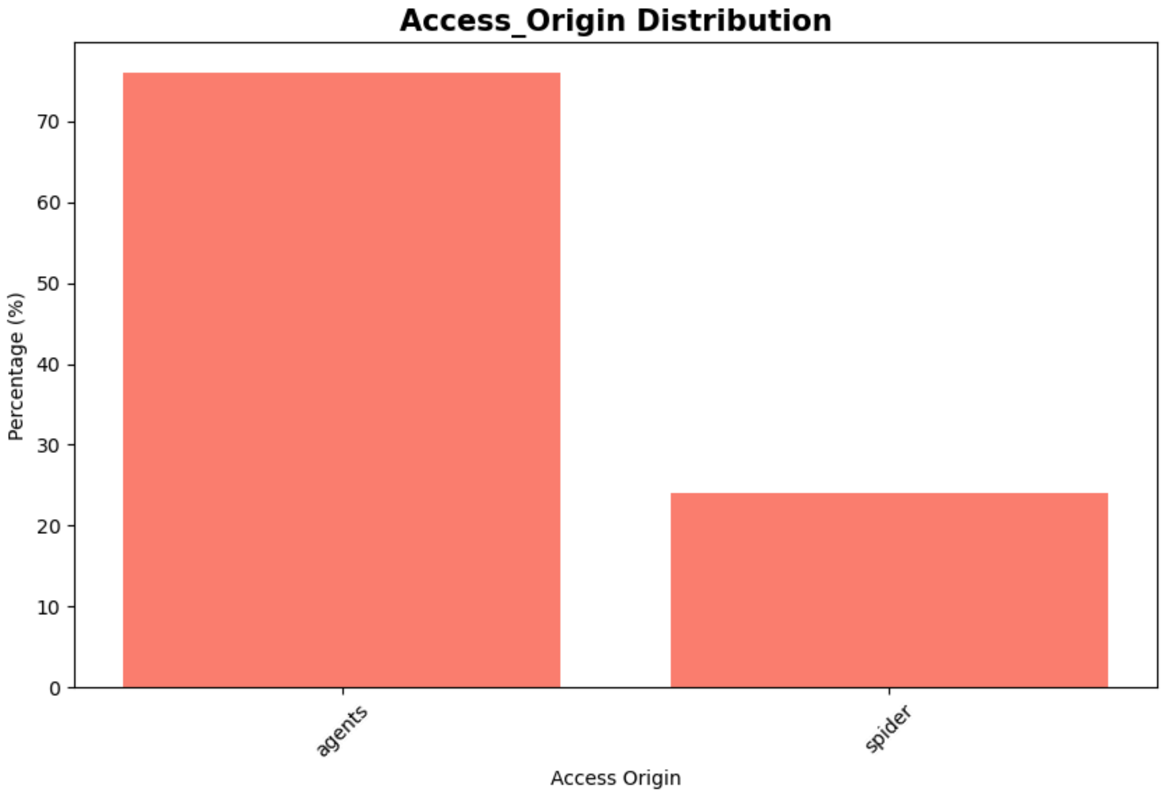
Out[41]: 0

In [42]:
```
data["Access_Origin"] = data.Page.str.findall(r'spider|agents').apply(lam
```

In [43]:
```
data["Access_Origin"].value_counts(dropna= False, normalize=True) * 100
```

Out[43]:
```
Access_Origin
agents    75.932526
spider    24.067474
Name: proportion, dtype: float64
```

In [44]:
```
x = (data["Access_Origin"].value_counts(dropna=False, normalize=True) * 1
y = (data["Access_Origin"].value_counts(dropna=False, normalize=True) * 1

plt.figure(figsize=(10, 6))
plt.bar(y, x, color='salmon')
plt.xlabel('Access Origin')
plt.ylabel('Percentage (%)')
plt.title('Access_Origin Distribution', fontsize=15, fontweight='bold')
plt.xticks(rotation=45)
plt.show()
```



In [45]:
```
data
```

Out[45]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-0
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.
...	...	...	...	...	.
145058	Underworld_(serie_de_películas)_es.wikipedia.o...	0.0	0.0	0.0	0.
145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org...	0.0	0.0	0.0	0.
145060	Enamorándome_de_Ramón_es.wikipedia.org_all-acc...	0.0	0.0	0.0	0.
145061	Hasta_el_último_hombre_es.wikipedia.org_all-ac...	0.0	0.0	0.0	0.
145062	Francisco_el_matemático_(serie_de_televisión_d...	0.0	0.0	0.0	0.

145063 rows × 554 columns

In [49]:
```
data.columns
```

Out[49]:
```
Index(['Page', '2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',
       '2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08', '2015-07-
09',
       ...
       '2016-12-25', '2016-12-26', '2016-12-27', '2016-12-28', '2016-12-
29',
       '2016-12-30', '2016-12-31', 'Language', 'Access_Type', 'Access_Or
igin'],
      dtype='object', length=554)
```

In [50]:
```
data.dtypes
```

Out[50]:
```
Page              object
2015-07-01       float64
2015-07-02       float64
2015-07-03       float64
2015-07-04       float64
                  ...
2016-12-30       float64
2016-12-31       float64
Language          object
Access_Type       object
Access_Origin     object
Length: 554, dtype: object
```

In [51]:
```
data.groupby('Language').mean(numeric_only =True)
```

Out [51]:

Language	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-(
Chinese	240.582042	240.941958	239.344071	241.653491	257.7
English	3513.862203	3502.511407	3325.357889	3462.054256	3575.52
French	475.150994	478.202000	459.837659	491.508932	482.5
German	714.968405	705.229741	676.877231	621.145145	722.0
Japenese	580.647056	666.672801	602.289805	756.509177	725.7
Russian	629.999601	640.902876	594.026295	558.728132	595.0
Spanish	1085.972919	1037.814557	954.412680	896.050750	974.5
Unknown_Language	83.479922	87.471857	82.680538	70.572557	78.2

8 rows × 550 columns

In [52]:
```
pd.set_option('display.max_rows', 500)
```

In [53]:
```
aggregated_data = data.groupby("Language").mean(numeric_only =True).T.dro
aggregated_data["index"] = pd.to_datetime(aggregated_data["index"])
aggregated_data = aggregated_data.set_index("index")
aggregated_data
```

Out[53]:

Language	Chinese	English	French	German	Japenese	F
index						
2015-07-01	240.582042	3513.862203	475.150994	714.968405	580.647056	629.
2015-07-02	240.941958	3502.511407	478.202000	705.229741	666.672801	640.
2015-07-03	239.344071	3325.357889	459.837659	676.877231	602.289805	594.
2015-07-04	241.653491	3462.054256	491.508932	621.145145	756.509177	558.
2015-07-05	257.779674	3575.520035	482.557746	722.076185	725.720914	595.
...	...	...	...	...	...	
2016-12-27	376.019618	6040.680728	858.413100	1085.095379	789.158680	1001.
2016-12-28	378.048639	5860.227559	774.155769	1032.640804	790.500465	931.
2016-12-29	350.719427	6245.127510	752.712954	994.657141	865.483236	897.
2016-12-30	354.704452	5201.783018	700.543422	949.265649	952.018354	803.
2016-12-31	365.579256	5127.916418	646.258342	893.013425	1197.239440	880.

550 rows × 7 columns

In [54]:
```
aggregated_data.info()
```

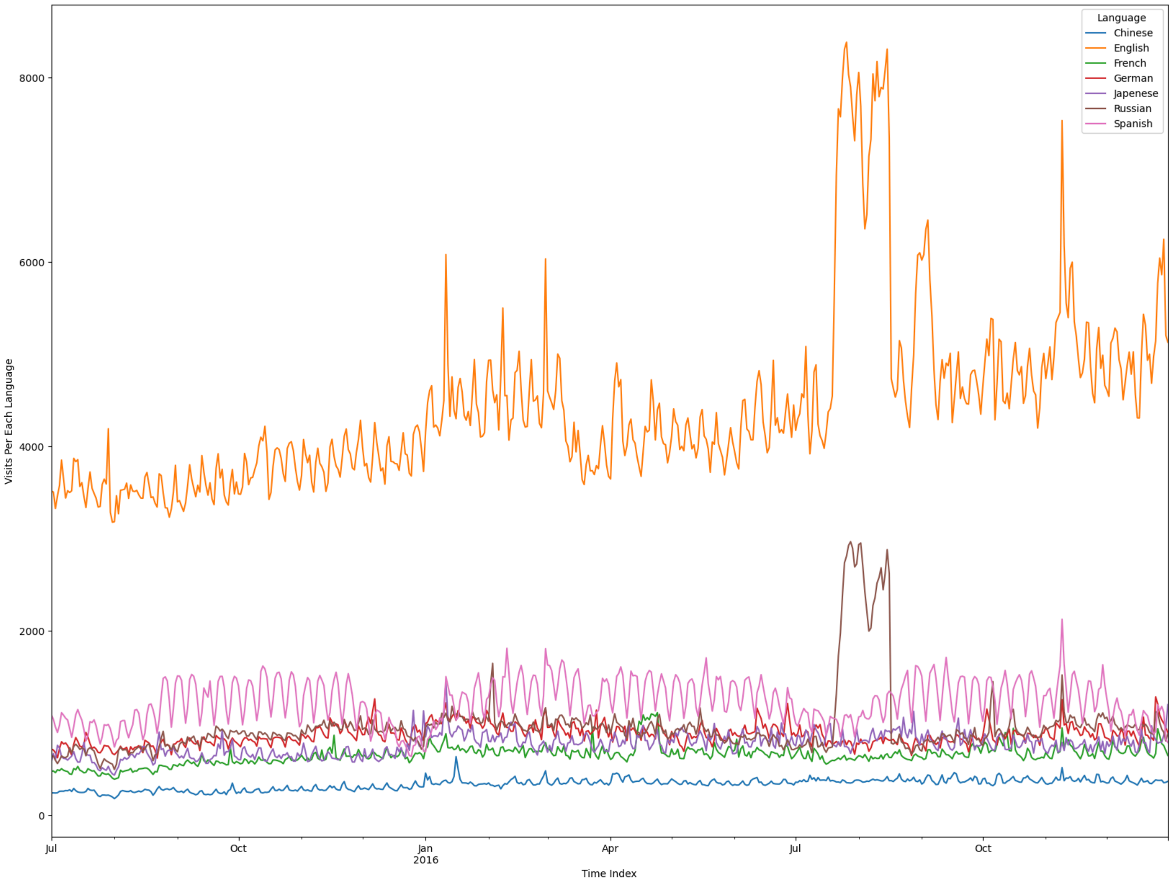
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 550 entries, 2015-07-01 to 2016-12-31
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Chinese   550 non-null    float64
 1   English   550 non-null    float64
 2   French    550 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 [55]:
```
aggregated_data.index
```

```
Out[55]: 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=Non
e)
```

Visualising Time Series for each language

```
In [57]: plt.rcParams['figure.figsize'] = (20, 15)
aggregated_data.plot()
plt.xlabel("Time Index")
plt.ylabel("Visits Per Each Language")
plt.show()
```



- English is the most used language
- There is visible peak in the use of English and Russian during the months of August and September

Hypothesis Testing

- H0: The series is Non-Stationary
- Ha: The series is Stationary
- significant value : 0.05

- if p-value > 0.05 : we fail 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 [61]:
```
warnings.filterwarnings('ignore', "Intel MKL WARNING")
```

In [62]:
```
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 [114…
```
for Language in aggregated_data.columns:
 print(Language)
 print(Dickey_Fuller_test(aggregated_data[Language],significances_level =
 print()
 print()
```

```
Chinese
Time Series is NOT Stationary
P_value is:  0.4474457922931137
None


English
Time Series is NOT Stationary
P_value is:  0.18953359279992366
None


French
Time Series is NOT Stationary
P_value is:  0.05149502195245779
None


German
Time Series is NOT Stationary
P_value is:  0.1409738231972964
None


Japenese
Time Series is NOT Stationary
P_value is:  0.10257133898557663
None


Russian
Time Series is Stationary
P_value is:  0.0018649376536617962
None


Spanish
Time Series is Stationary
P_value is:  0.03358859084479109
None
```

- Time Series is Stationary for Russian and Spanish languages
- Whereas Time Series for the rest of the languages ie. Chinese, English , German , Japanese and French are not stationary.

Analysing Time Series for English Language Pages Visits

In [124…
```
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','#L
 for key, value in dftest[4].items():
     df_output['Critical Value (%s)' %key] = value
 print (df_output)
```

In [126…
```
adf_test(TS_English)
```

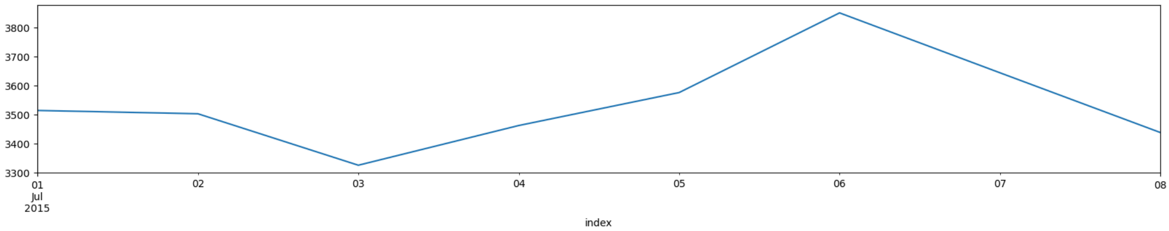
```
Results of Dickey-Fuller Test:
Test Statistic              -2.247284
p-value                      0.189534
#Lags Used                  14.000000
# Observations Used        535.000000
Critical Value (1%)         -3.442632
Critical Value (5%)         -2.866957
Critical Value (10%)        -2.569655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

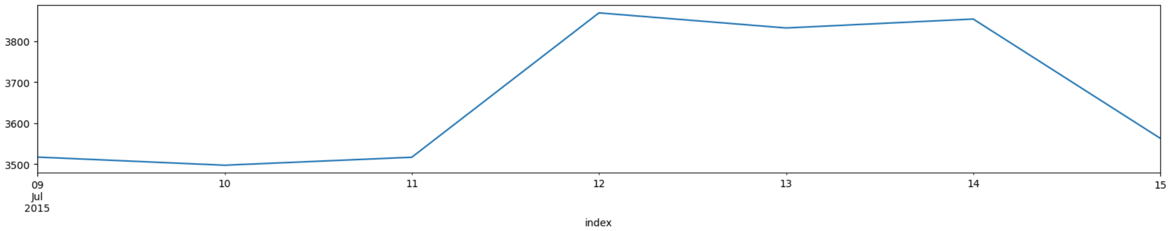
In [128…
```
Dickey_Fuller_test(TS_English)
```

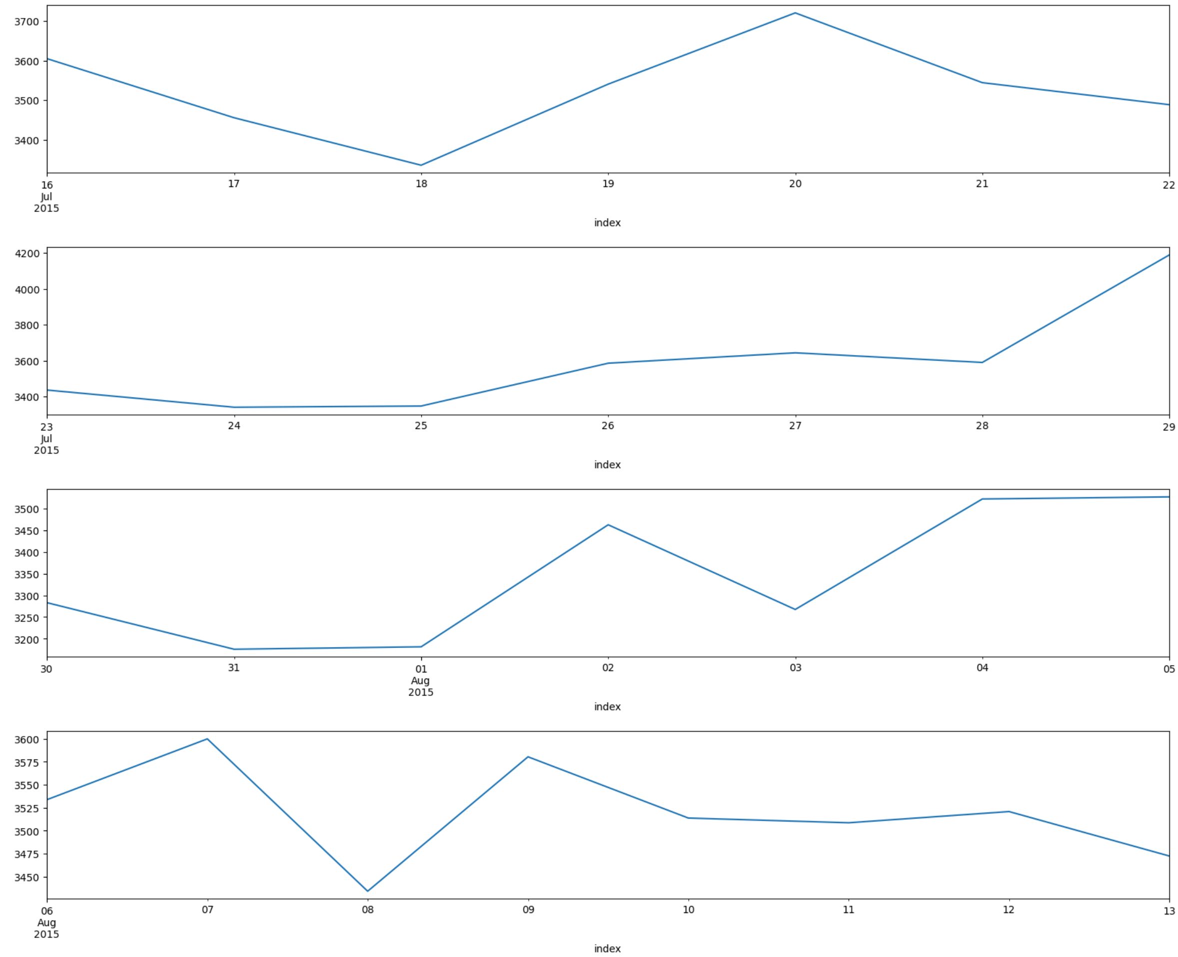
```
Time Series is NOT Stationary
P_value is:  0.18953359279992366
```

Plotting English-Language Page Visits Time Series to identify seasonality and period

In [131…
```
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 [133…

```
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)
```

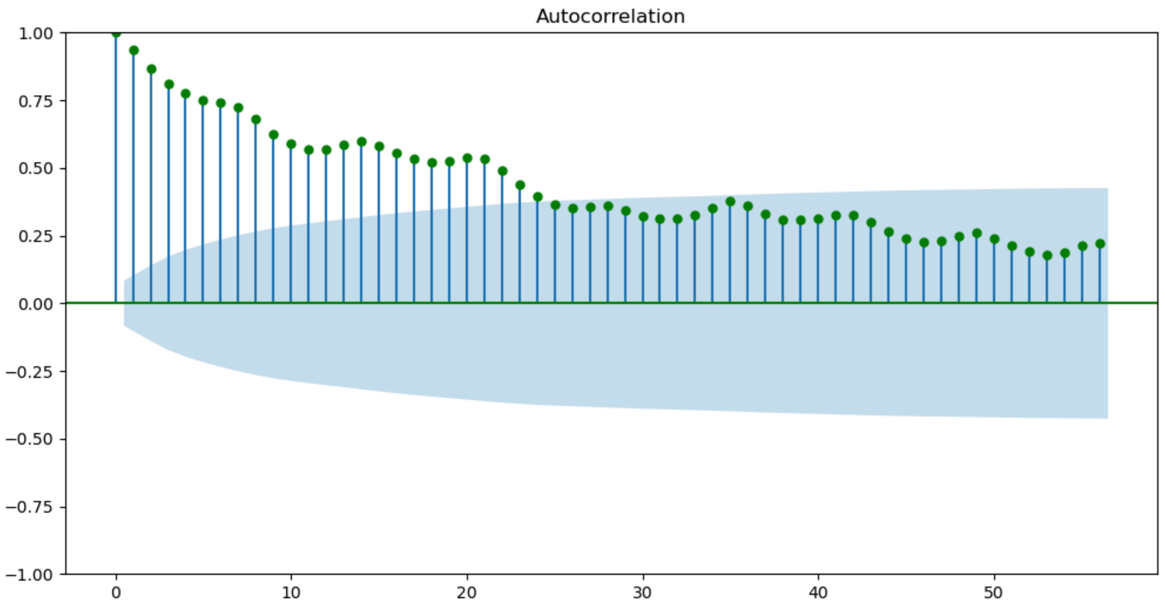
```
1 0.9363434527458431
2 0.8682966716039899
3 0.8185418037184545
4 0.7846718829500337
5 0.7612561076942571
6 0.754226064178356
7 0.7386829287516692
8 0.691263801818988
9 0.6370978014300402
10 0.6015277501876306
11 0.5825450402423569
12 0.5812931934793533
13 0.6007266462817784
14 0.6142525351445116
15 0.5971084554755529
16 0.5693834937428242
17 0.5488401467532629
18 0.5377431132136109
19 0.5430816743411203
20 0.5552694244923044
21 0.5540623423718066
22 0.5092655604869364
23 0.4537369557681359
24 0.4112336297620322
25 0.3816286061625172
26 0.3651996316699481
27 0.37236036273025985
28 0.3781822668316004
29 0.3593924266732817
```

Time Series Decomposition

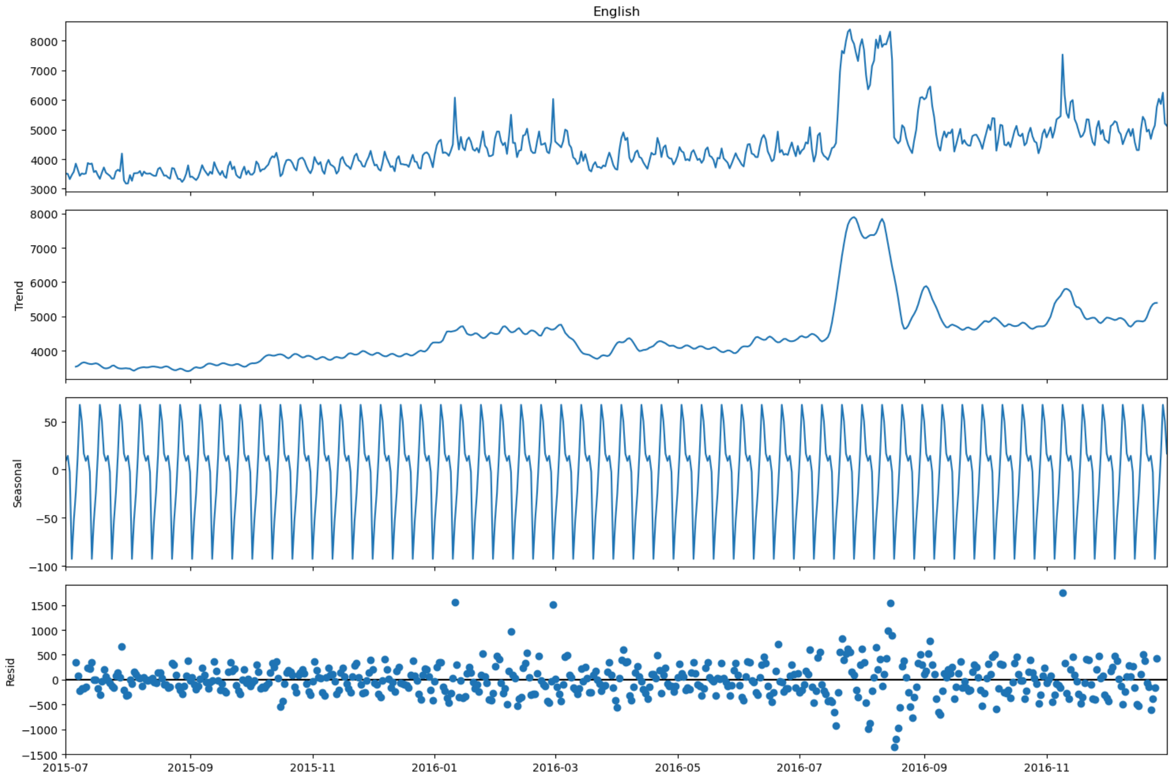
Y(t) = seasonality S(t) + trend T(t) + residuals R(t)

In [137…

```python
# 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 [153…
```
plt.rcParams['figure.figsize'] = (15, 10)
Decomposition_model = sm.tsa.seasonal_decompose(TS_English, model='additi
Decomposition_model.plot();
```

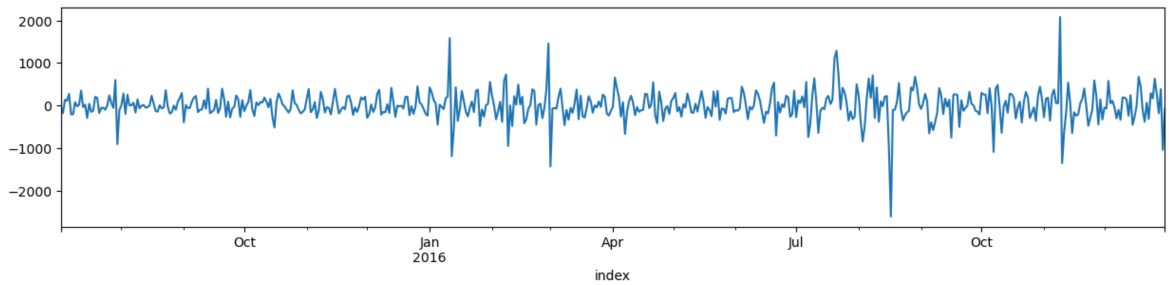


In [155…
```
Dickey_Fuller_test(pd.Series(Decomposition_model.resid).fillna(0))
```

```
Time Series is Stationary
P_value is:  9.809019764165091e-21
```

In [157…
```
# Taking the first differentiation of the time series and plotting
plt.rcParams['figure.figsize'] = (15, 3)
TS_English.diff(1).dropna().plot()
```

Out[157…
```
<Axes: xlabel='index'>
```



In [159…
```
Dickey_Fuller_test(TS_English.diff(1).dropna())
```

```
Time Series is Stationary
P_value is:  5.292474635436327e-13
```

- Time series becomes stationary after 1 differentiation
- We can set d = 1 for ARIMA models

In [162…
```
from sklearn.metrics import (
 mean_squared_error as mse,
 mean_absolute_error as mae,
```

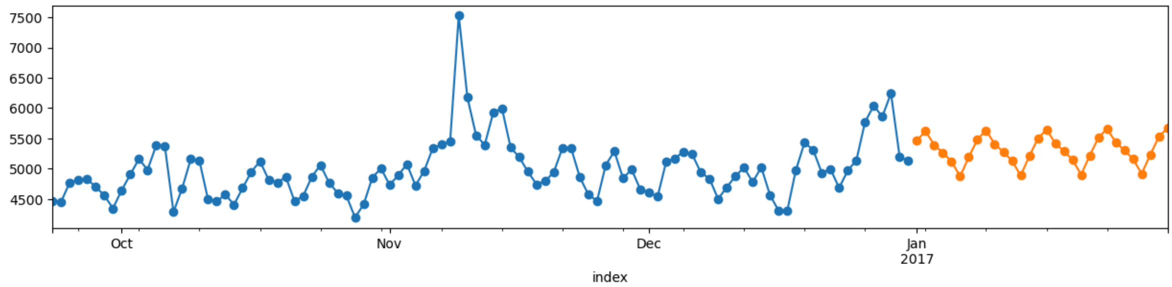
```
 mean_absolute_percentage_error as mape
)
# Creating a function to print values of all these metrics.
def performance(actual, predicted):
 print('MAE :', round(mae(actual, predicted), 3))
 print('RMSE :', round(mse(actual, predicted)**0.5, 3))
 print('MAPE:', round(mape(actual, predicted), 3))
```

Forecasting

In [165…

```
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')
```

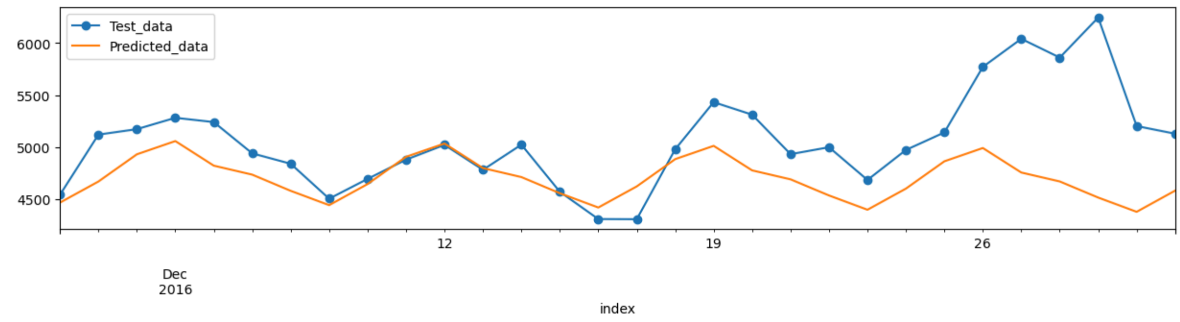
Out[165… <Axes: xlabel='index'>



In [167…

```
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 : 394.982
RMSE : 563.357
MAPE: 0.073
```



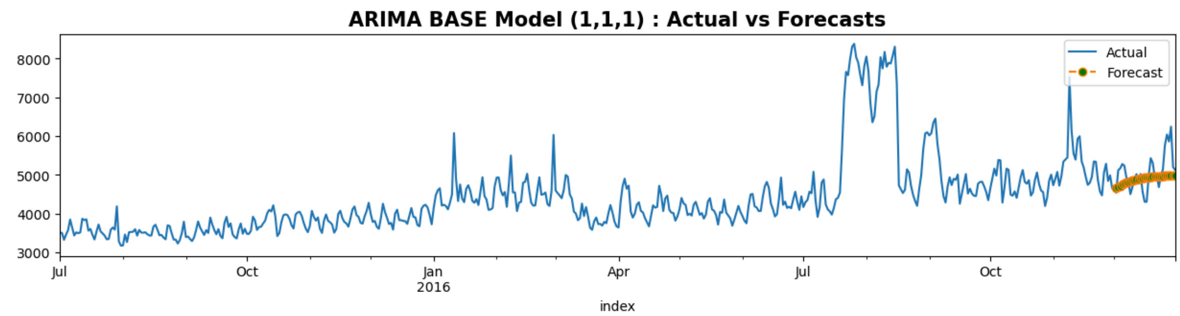
ARIMA

In [174…

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

In [178…

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

SARIMAX model

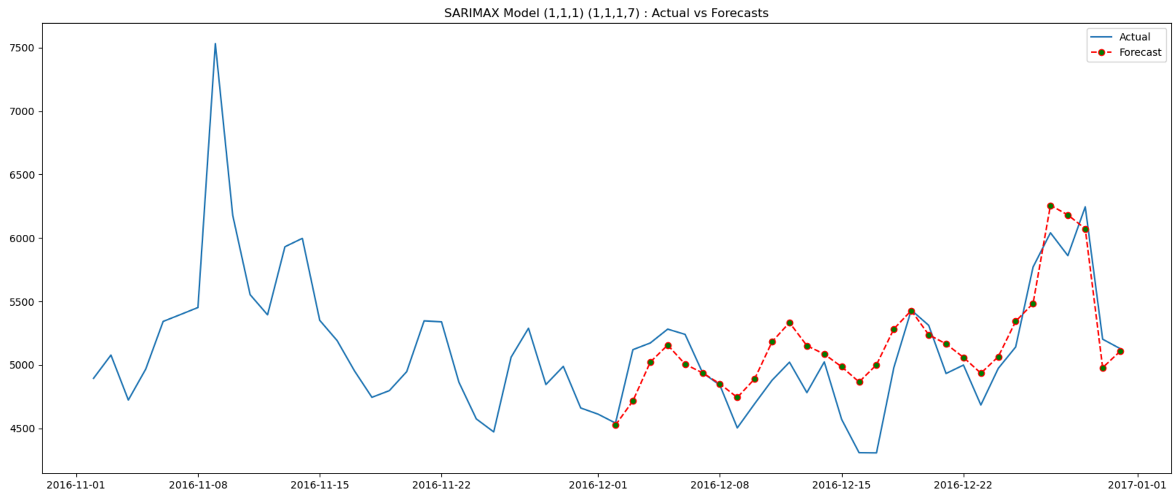
In [280…

```
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

 if exog is None:
     exog = pd.DataFrame(index=time_series.index)
```

```
    else:
        exog = pd.DataFrame(exog, index=time_series.index)

    #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
    if not exog.empty and exog.iloc[-n:].any().any():
        exog_forecast = exog[-n:]
    else:
        exog_forecast = None

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

    #plotting Actual & Forecasted values

    plt.figure(figsize = (20,8))
    time_series[-60:].plot(label = 'Actual')
    model_forecast[-60:].plot(label = 'Forecast', color = 'red', linestyle='
    plt.legend(loc="upper right")
    plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actual vs Fo
    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 [288…

```
exog = df2['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=exo
```



```
MAPE of Model: 4.45011%
RMSE of Model: 272.552
```

Hyperparamer tuning for SARIMAX model

In [298…

```
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=
                                 exog=exog[:-n], initialization='approxim
                                 model_fit = model.fit()


                                 model_forecast = model_fit.forecast(step
                                 #exog=pd.Dat
                                 actuals = time_series.values[-n:]
                                 errors = actuals - model_forecast.values
                                 mape = np.mean(np.abs(errors) / np.abs(a
                                 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
                                 param_df.loc[len(param_df)] = list_row
                                 print(f'Possible Combination: {counter}
                             except Exception as e:
                                 print(f"Error with combination (p={p}, d
                                 continue
 return param_df
```

In [316…

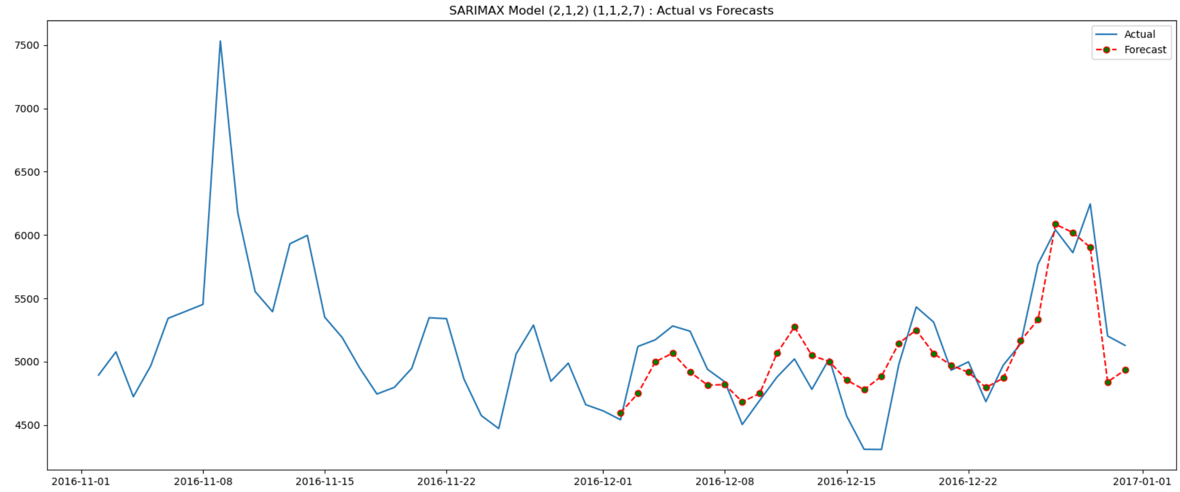
```
english_params.sort_values(['mape', 'rmse']).head()
```

Out[316…

serial	pdq	PDQs	mape	rmse

In [318…

```
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 = e
```




```
MAPE of Model: 4.05472%
RMSE of Model: 247.722
```

Hyperparameter tuning for all other languages

In [398…

```
def pipeline_sarimax_grid_search_without_exog(languages, data, n, param,
    best_param_df = pd.DataFrame(columns=['language', 'p', 'd', 'q', 'P',

    for lang in languages:

        counter = 0
        time_series = data[lang]
        best_mape = float('inf')  # Start with a large value for best_map

        # Creating loop for every parameter 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:
                                    try:
                                        # Creating Model
                                        model = SARIMAX(time_series[:-n],
                                                        order=(p, d, q),
                                                        seasonal_order=(P
                                                        initialization='a
                                        model_fit = model.fit(disp=False)

                                        # Creating forecast from Model
                                        model_forecast = model_fit.foreca

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

                                        # Calculating MAPE
                                        mape = np.mean(np.abs(errors) / n

                                        # Check if the current MAPE is th
                                        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

                                counter += 1

                            except Exception as e:
                                print(f"Error with combination (p
                                continue


        best_mape = np.round(best_mape, 5)

        print(f'Minimum MAPE for {lang} = {best_mape}')
        print(f'Corresponding Best Parameters are (p={best_p}, d={best_d}
        print('-----------------------------------------------------------

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

    return best_param_df
```

In [400…

```
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, aggr
```

```
Minimum MAPE for Chinese = 4.48201
Corresponding Best Parameters are (p=0, d=1, q=0, P=1, D=0, Q=1, s=2)
--------------------------------------------------------------
Minimum MAPE for English = 6.62077
Corresponding Best Parameters are (p=1, d=1, q=1, P=1, D=0, Q=0, s=2)
--------------------------------------------------------------
Minimum MAPE for French = 7.52182
Corresponding Best Parameters are (p=1, d=1, q=1, P=0, D=1, Q=1, s=2)
--------------------------------------------------------------
Minimum MAPE for German = 8.23541
Corresponding Best Parameters are (p=0, d=1, q=0, P=1, D=0, Q=1, s=2)
--------------------------------------------------------------
Minimum MAPE for Japenese = 7.91333
Corresponding Best Parameters are (p=1, d=1, q=0, P=0, D=1, Q=1, s=2)
--------------------------------------------------------------
Minimum MAPE for Russian = 5.07515
Corresponding Best Parameters are (p=0, d=0, q=0, P=1, D=0, Q=1, s=2)
--------------------------------------------------------------
Minimum MAPE for Spanish = 10.84676
Corresponding Best Parameters are (p=0, d=0, q=1, P=1, D=0, Q=0, s=2)
--------------------------------------------------------------
```

In [342…

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

Out[342…

	language	p	d	q	P	D	Q	s	mape
0	Chinese	0	1	0	1	0	1	2	4.48201
5	Russian	0	0	0	1	0	1	2	5.07515
1	English	1	1	1	1	0	0	2	6.62077
2	French	1	1	1	0	1	1	2	7.52182
4	Japenese	1	1	0	0	1	1	2	7.91333
3	German	0	1	0	1	0	1	2	8.23541
6	Spanish	0	0	1	1	0	0	2	10.84676

In [402…

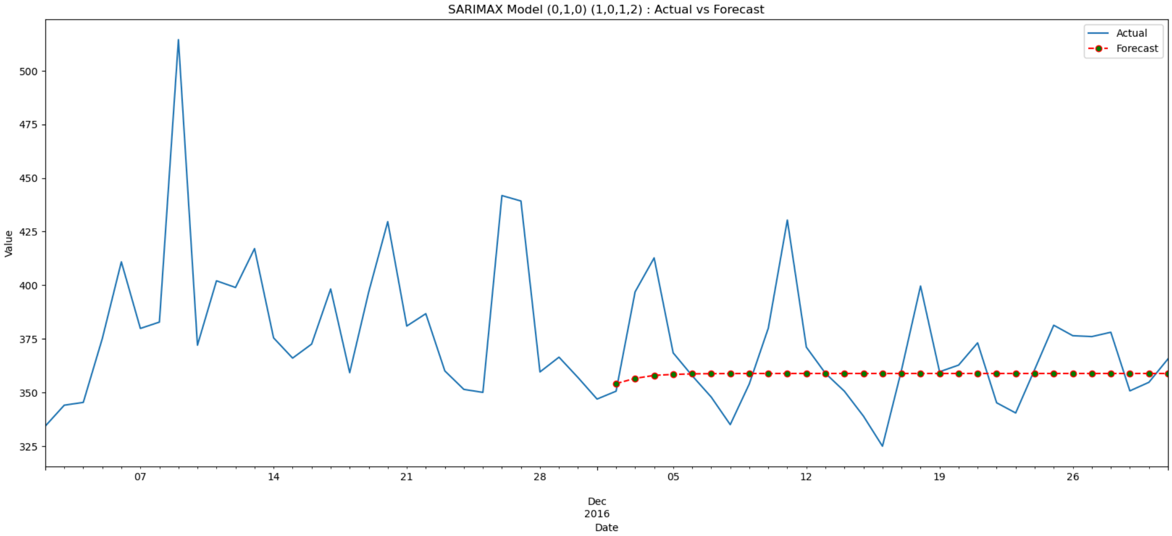
```
def plot_best_SARIMAX_model(languages, data, n, best_param_df):
    for lang in languages:
        # Fetching respective best parameters for that language
        params = best_param_df[best_param_df['language'] == lang]
        if params.empty:
            print(f"No best parameters found for {lang}. Skipping.")
            continue

        p = params['p'].values[0]
        d = params['d'].values[0]
        q = params['q'].values[0]
        P = params['P'].values[0]
        D = params['D'].values[0]
        Q = params['Q'].values[0]
        s = params['s'].values[0]

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

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

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

        # Calculating MAPE & RMSE
        actuals = time_series.values[-n:]
        errors = actuals - model_forecast.values
        mape = np.mean(np.abs(errors) / np.abs(actuals)) * 100  # Convert
        rmse = np.sqrt(np.mean(errors ** 2))

        print('')
        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)}')

        # Plotting Actual & Forecasted values
        plt.figure(figsize=(20, 8))
        time_series[-60:].plot(label='Actual')
        model_forecast.plot(label='Forecast', color='red', linestyle='das
```

```
        plt.legend(loc="upper right")
        plt.title(f'SARIMAX Model ({p},{d},{q}) ({P},{D},{Q},{s}) : Actua
        plt.xlabel('Date')
        plt.ylabel('Value')
        plt.show()

    return 0
```

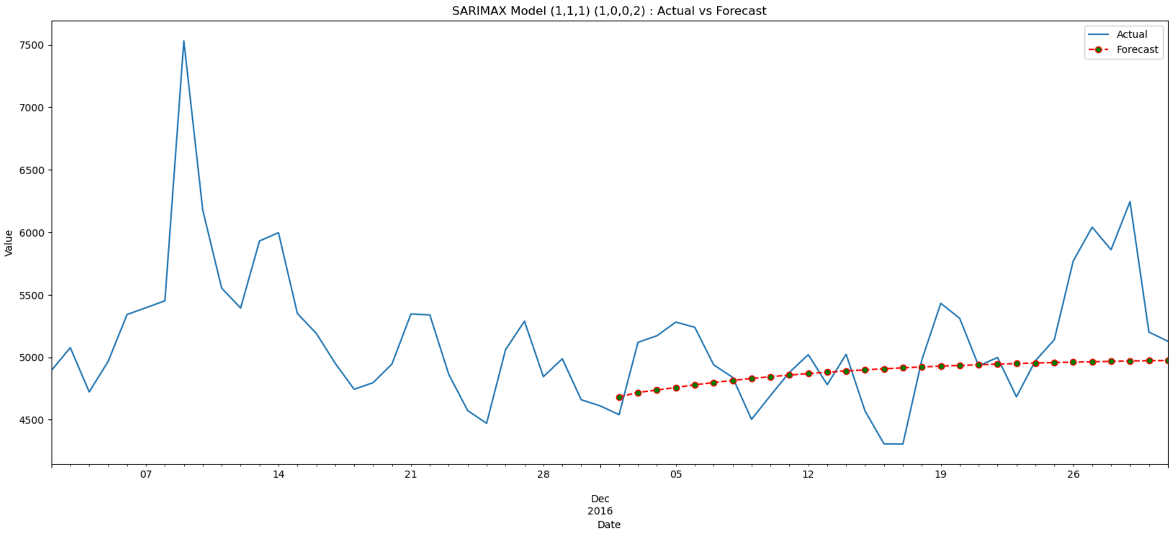
In [404…

```
#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)
```

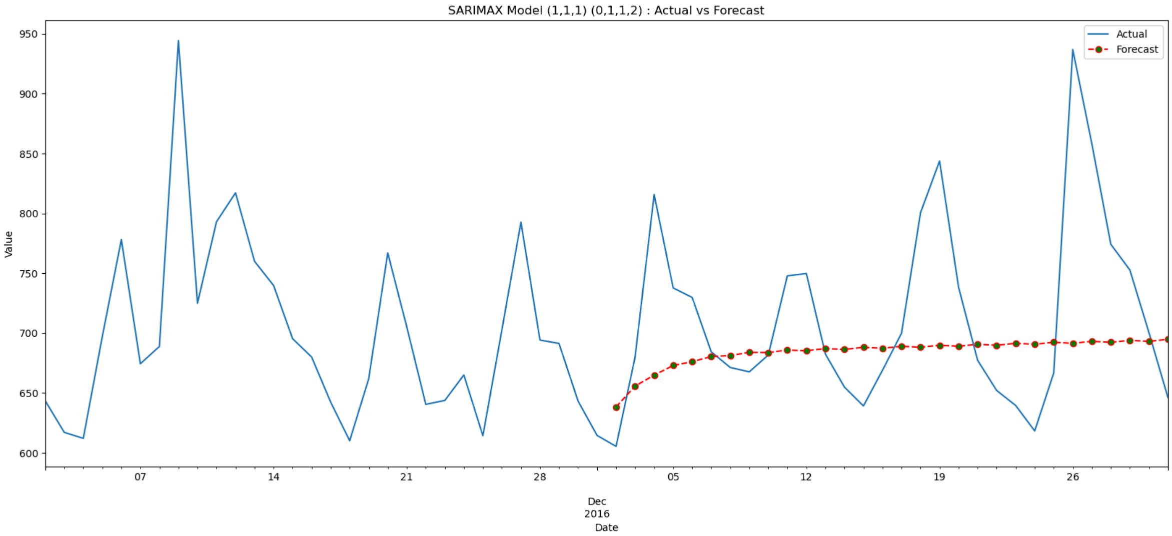
```
SARIMAX model for Chinese Time Series
Parameters of Model : (0,1,0) (1,0,1,2)
MAPE of Model : 4.48201
RMSE of Model : 23.669
```



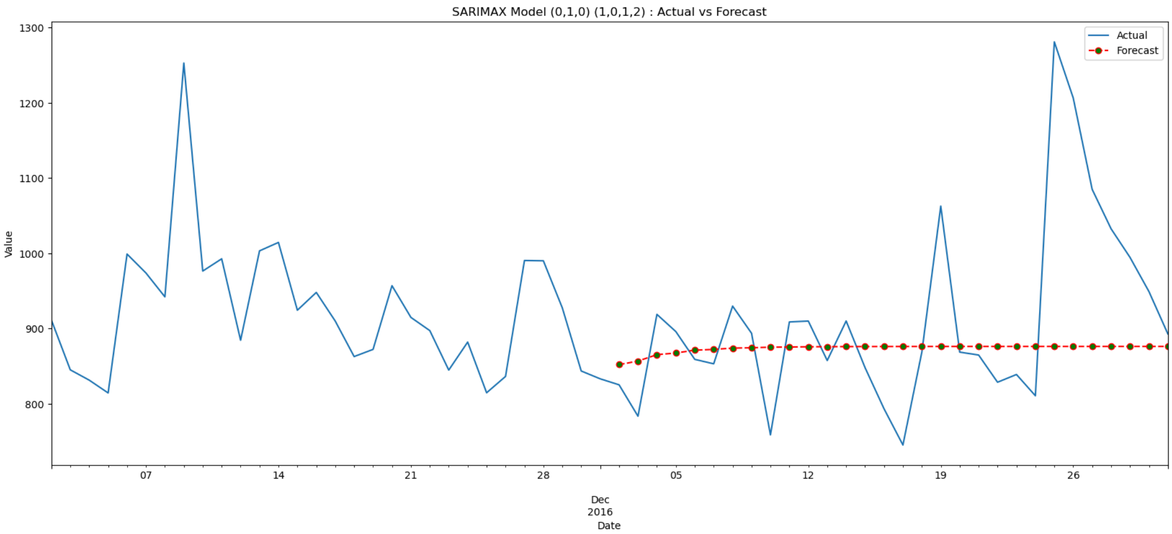

```
SARIMAX model for English Time Series
Parameters of Model : (1,1,1) (1,0,0,2)
MAPE of Model : 6.62077
RMSE of Model : 473.203
```



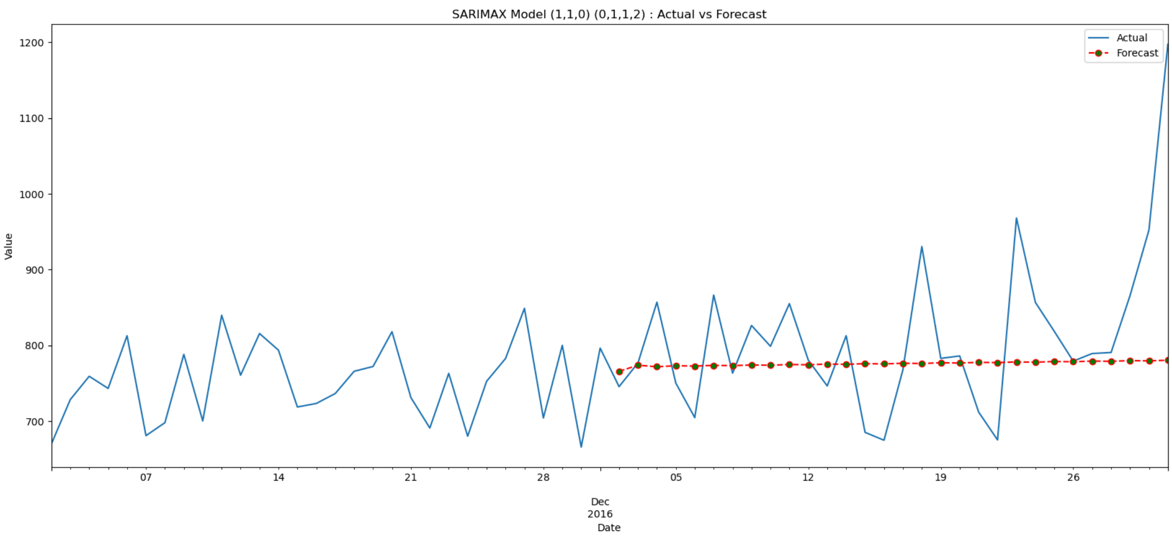

```
SARIMAX model for French Time Series
Parameters of Model : (1,1,1) (0,1,1,2)
MAPE of Model : 7.52182
RMSE of Model : 80.191
```



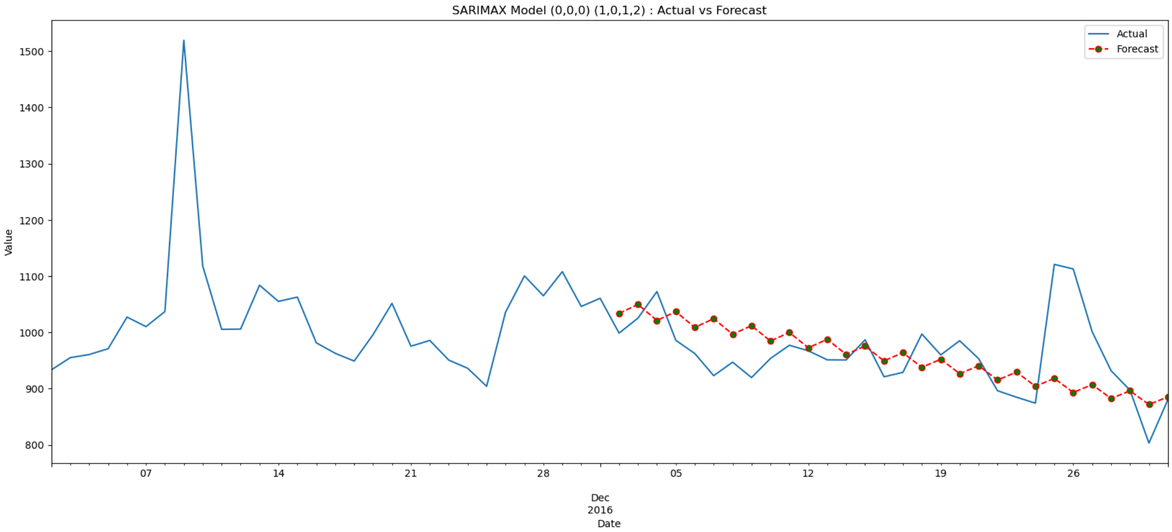

```
SARIMAX model for German Time Series
Parameters of Model : (0,1,0) (1,0,1,2)
MAPE of Model : 8.23541
RMSE of Model : 123.736
```



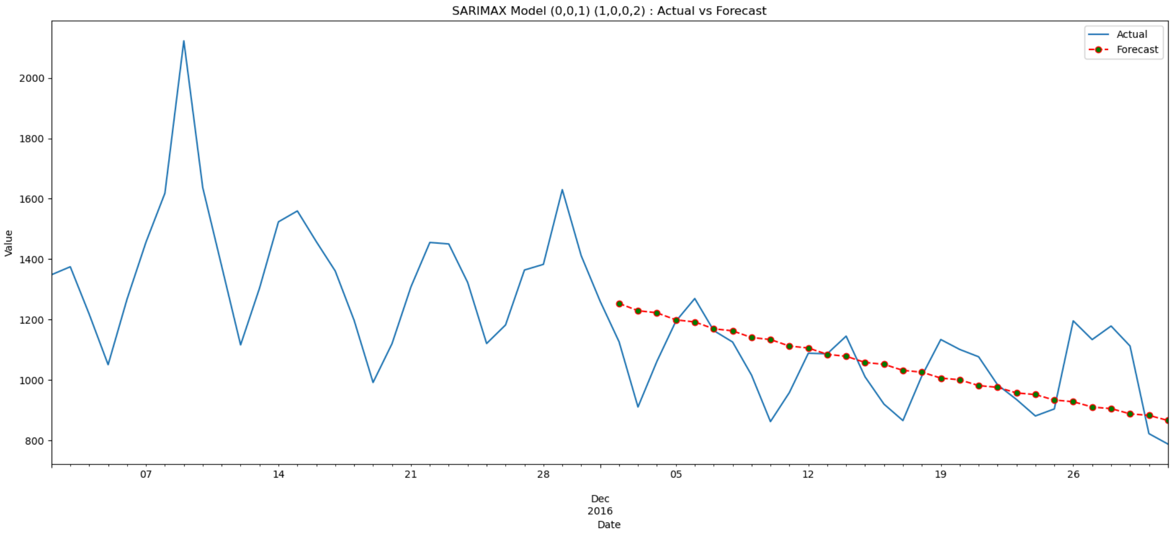

```
SARIMAX model for Japenese Time Series
Parameters of Model : (1,1,0) (0,1,1,2)
MAPE of Model : 7.91333
RMSE of Model : 107.35
```




```
SARIMAX model for Russian Time Series
Parameters of Model : (0,0,0) (1,0,1,2)
MAPE of Model : 5.07515
RMSE of Model : 71.022
```




```
SARIMAX model for Spanish Time Series
Parameters of Model : (0,0,1) (1,0,0,2)
MAPE of Model : 10.84676
RMSE of Model : 143.24
```



Out[404… 0

Forecasting using Facebook Prophet

In [361…

```
from prophet import Prophet
```

In [363…

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

In [365…

```
time_series
```

Out[365…

	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

- There are 7 languages in the given data
- The language with the most pages is English.
- There are 3 types of Access-Types:
    1. 51.22 % of All-access
    2. 24.77 % have mobile-web access
    3. 23.99 % have desktop access
- 16.61% of the total pages are in English which is the highest
- 12.30% of pages contains Unknown languages
- A Single level of Differentiation gave us a Stationary series

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

- Do a Dickey-Fuller test to make sure the time series is stationary
- If the time series is stationary, fit an ARMA model. If it is non-stationary, find out what d is.
- After the series becomes Stationary plot the data's autocorrelation and partial autocorrelation graphs
- Since the cut-off point in the partial autocorrelation graph (PACF) equals p, plot the PACF to

find the value of p.

- Since q is the value of the ACF's cut-off point, plot the autocorrelation graph (ACF) to find q,

In [ ]:

```
```