1. Defining Problem Statement & Data Import

1.1 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.

Get the dataset from below link:-

https://drive.google.com/drive/folders/1mdgQscjqnCtdg7LGItomyK0abN6lcHBb

- Data Dictionary:

There are two csv files given

- **train_1.csv:** In the csv file, each row corresponds to a particular article and each column corresponds to a particular date. The values are the number of visits on that date.

The page name contains data in this format: SPECIFIC NAME *LANGUAGE.wikipedia.org* ACCESS TYPE _ ACCESS ORIGIN having information about the page name, the main domain, the device type used to access the page, and also the request origin(spider or browser agent)

- **Exog_Campaign_eng:** This file contains data for the dates which had a campaign or significant event that could affect the views for that day. The data is just for pages in English.

There's 1 for dates with campaigns and 0 for remaining dates. It is to be treated as an exogenous variable for models when training and forecasting data for pages in English

1.2 Importing Libraries:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style = 'darkgrid')
pd.set_option('display.max_columns', None)
pd.options.display.max_colwidth = 100

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

```
C:\Users\saima\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

1.3 Importing Data & removing non-relevant columns / duplicates

In [2]:
```
raw_data = pd.read_csv(r"C:\Users\saima\Downloads\train_1.csv")
exo_var = pd.read_csv(r"C:\Users\saima\Downloads\Exog_Campaign_eng.csv")
```

In [3]:
```
#creating copy of dataframe for backup
data = raw_data.copy(deep = True)
data.drop_duplicates(keep='last', inplace = True)
```

In [4]:
```
print('-'*80)
print(f'Shape of Data : {data.shape}')
print('-'*80)
print(f'Shape of exogenous variable : {exo_var.shape}')
print('-'*80)
```

```
--------------------------------------------------------------------------------
Shape of Data : (145063, 551)
--------------------------------------------------------------------------------
Shape of exogenous variable : (550, 1)
--------------------------------------------------------------------------------
```

In [5]:
```
data.sample(100).head()
```

Out[5]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2 0
143156	Darth_Vader_es.wikipedia.org_all-access_spider	8.0	4.0	5.0	6.0	12.0	
68058	Scarlett_Johansson_de.wikipedia.org_desktop_all-agents	630.0	553.0	529.0	543.0	725.0	6
32656	Calista_Flockhart_en.wikipedia.org_all-access_spider	54.0	19.0	27.0	28.0	37.0	
40931	48,_XXXX_en.wikipedia.org_all-access_all-agents	1940.0	1348.0	665.0	829.0	1004.0	10
15101	File:Mein_mikropenis.jpg_commons.wikimedia.org_all-access_spider	0.0	1.0	0.0	1.0	1.0	

=> **Data for 550 Dates (1.5 Years / 18 Months) is provided for all pages**

In [7]:
```
print('-'*80)
print('Data types of Starting Columns')
print('-'*80)
print(data.dtypes[:10])
print('-'*80)
print('Data types of Ending Columns')
print('-'*80)
print(data.dtypes[-10:])
print('-'*80)
```

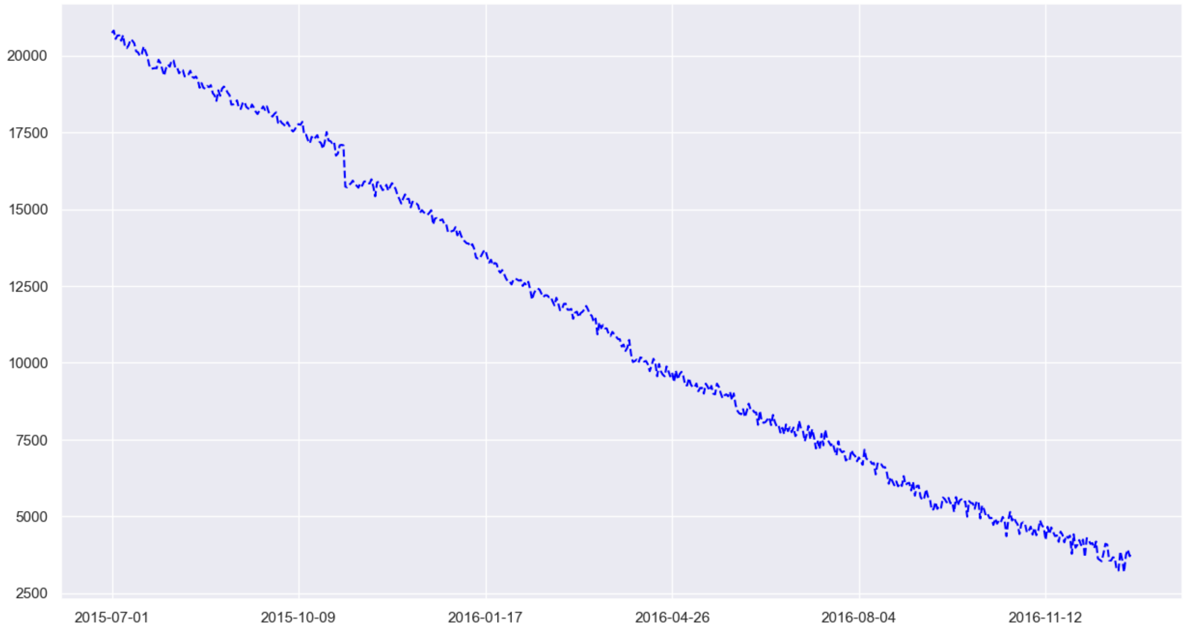
```
--------------------------------------------------------------------------------
Data types of Starting Columns
--------------------------------------------------------------------------------
Page           object
2015-07-01    float64
2015-07-02    float64
2015-07-03    float64
2015-07-04    float64
2015-07-05    float64
2015-07-06    float64
2015-07-07    float64
2015-07-08    float64
2015-07-09    float64
dtype: object
--------------------------------------------------------------------------------
Data types of Ending Columns
--------------------------------------------------------------------------------
2016-12-22    float64
2016-12-23    float64
2016-12-24    float64
2016-12-25    float64
2016-12-26    float64
2016-12-27    float64
2016-12-28    float64
2016-12-29    float64
2016-12-30    float64
2016-12-31    float64
dtype: object
--------------------------------------------------------------------------------
```

1.4 Understanding significance of Null Values

In [8]:
```
#Checking count of Null Values after every 25th Column in Data
data.isnull().sum()[range(1,550,25)]
```

Out[8]:
```
2015-07-01    20740
2015-07-26    19865
2015-08-20    18923
2015-09-14    18407
2015-10-09    17771
2015-11-03    15734
2015-11-28    15847
2015-12-23    14647
2016-01-17    13667
2016-02-11    12057
2016-03-07    11485
2016-04-01    10385
2016-04-26     9679
2016-05-21     9216
2016-06-15     8071
2016-07-10     7836
2016-08-04     6917
2016-08-29     6022
2016-09-23     5457
2016-10-18     4858
2016-11-12     4234
2016-12-07     4130
dtype: int64
```

In [9]:
```
#Visualizing Null-values count for all columns
plt.figure(figsize=(15, 8))
data.iloc[:, 1:-3 ].isnull().sum().plot(color='blue', linestyle='dashed')
plt.show()
```



=> Above Plot indicates that NaN / Null values are decreasing with Time. Later Dates have less Null Values as compared to Older Dates.

=> This is possible as the Pages which were hosted / created towards later dates, will have null values for previous dates (dates before the page was created / hosted).

=> We will drop the rows where more than 300 null values are present and replace remaining Null Values with 0.

In [10]:
```
data.dropna(thresh = 300, inplace = True)
print(f'Shape of Data : {data.shape}')
```

```
Shape of Data : (133617, 551)
```

In [11]:
```
data.fillna(0, inplace = True)
```

In [12]:
```
#Checking count of Null Values after every 25th Column in Data
data.isnull().sum()[range(1,550,25)]
```

Out[12]:
```
2015-07-01    0
2015-07-26    0
2015-08-20    0
2015-09-14    0
2015-10-09    0
2015-11-03    0
2015-11-28    0
2015-12-23    0
2016-01-17    0
2016-02-11    0
2016-03-07    0
2016-04-01    0
2016-04-26    0
2016-05-21    0
2016-06-15    0
2016-07-10    0
2016-08-04    0
2016-08-29    0
2016-09-23    0
2016-10-18    0
2016-11-12    0
2016-12-07    0
dtype: int64

==================================================================
```

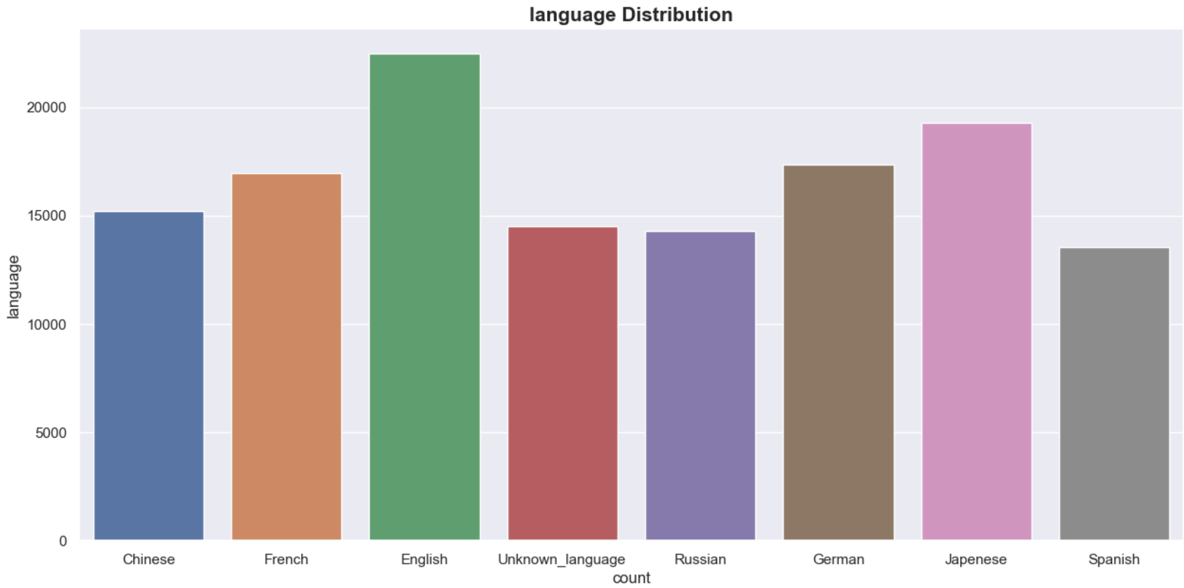
2. Exploratory Data Analysis & Feature Engineering

2.1 Extracting `Language` , `Access_Type` & `Access_Origin` from `Page`

In [13]:
```
import re

#Function to Extract Language from Page using Regex
def get_language(name):
    if len(re.findall(r'_(.{2}).wikipedia.org_', name)) == 1 :
        return re.findall(r'_(.{2}).wikipedia.org_', name)[0]
    else: return 'Unknown_language'

data['language'] = data['Page'].apply(get_language)


language_dict ={'de':'German',
                'en':'English',
                'es': 'Spanish',
                'fr': 'French',
                'ja': 'Japenese' ,
                'ru': 'Russian',
                'zh': 'Chinese',
                'Unknown_language': 'Unknown_language'}

data['language'] = data['language'].map(language_dict)
```

In [14]:
```
#Visualizing distribution of various languages
y = 'language'
```

```
plt.figure(figsize=(15, 7))
sns.countplot(x=y , data=data)
plt.title(f' {y} Distribution')
plt.xlabel('count')
plt.ylabel(f'{y}')
plt.title(f'{y} Distribution', fontsize = 15, fontweight = 'bold')
plt.show()
```



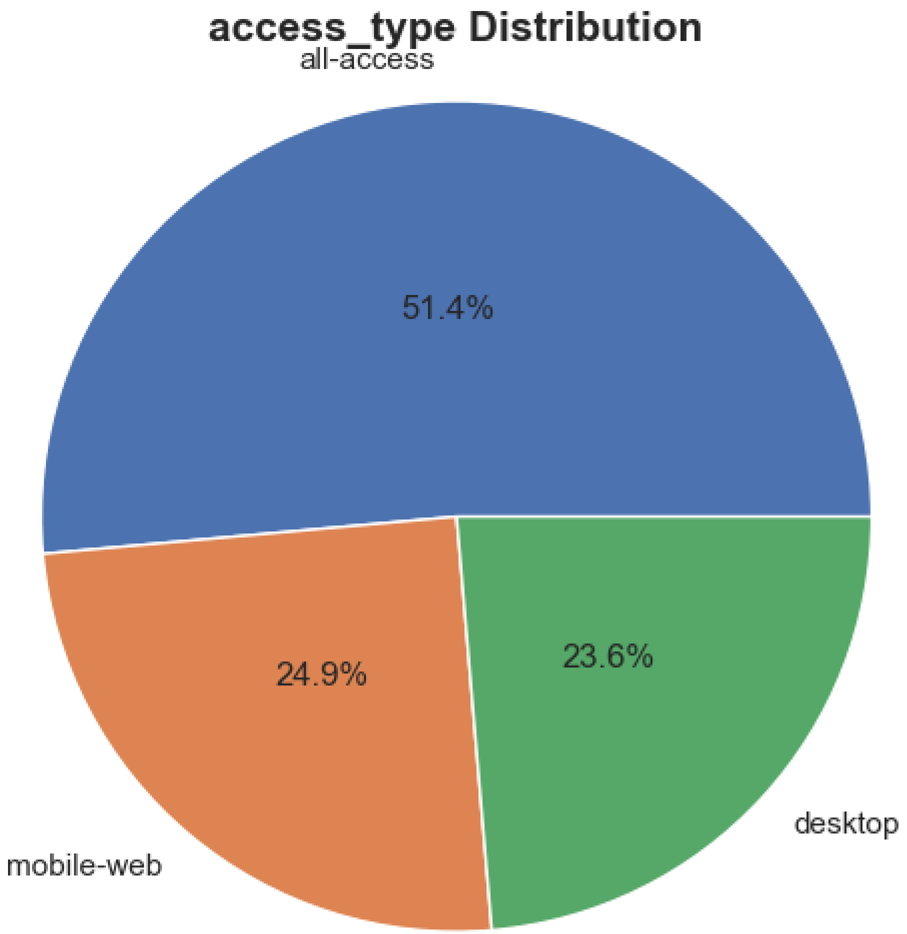
In [15]:
```
data.loc[data['language'] == 'Unknown_language', 'Page'].sample(100).head(10)
```

Out[15]:
```
82713                    Category:Nudes-A-Poppin'_2013_commons.wikimedia.org_
desktop_all-agents
77710    Category:Human_penises,_erect_length_125-150_mm_commons.wikimedia.org_mob
ile-web_all-agents
82853                Glavna_stranica_-_Главна_страница_commons.wikimedia.org_
desktop_all-agents
83403                                Manual:External_editors_www.mediawiki.org
_all-access_spider
84336                                    Manual:Upgrading/es_www.mediawiki.org
_all-access_spider
22212                            Special:MyLanguage/News_www.mediawiki.org_mob
ile-web_all-agents
43354                                How_to_contribute/mai_www.mediawiki.org_
desktop_all-agents
14956             File:Bruno_Mars,_Las_Vegas_2010.jpg_commons.wikimedia.org
_all-access_spider
83073                   Extension:Cargo/SMW_migration_guide_www.mediawiki.org
_all-access_spider
20550                                  Download/es_www.mediawiki.org_all
-access_all-agents
Name: Page, dtype: object
```

=> **Around 10.8% of rows (~14k) don't have Language information**

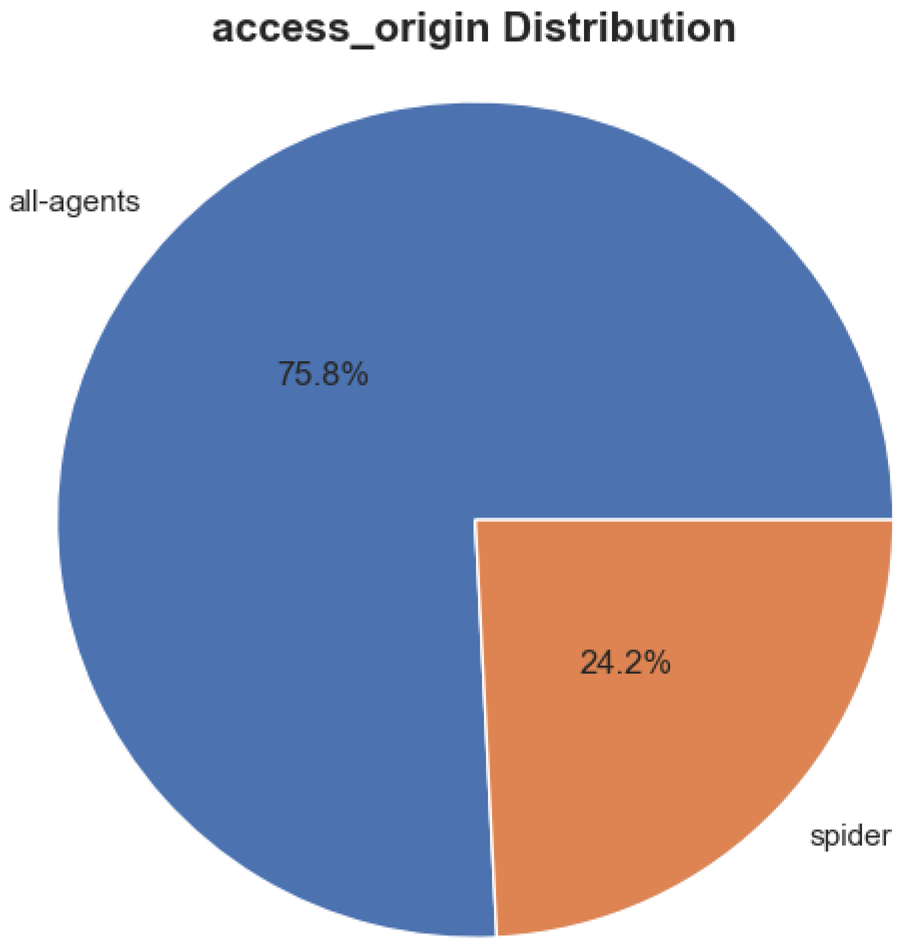
In [16]:
```
#Function to Extract Access Type from Page using Regex
def get_access_type(name):
    if len(re.findall(r'all-access|mobile-web|desktop', name)) == 1 :
        return re.findall(r'all-access|mobile-web|desktop', name)[0]
    else: return 'No Access_type'

data['access_type'] = data['Page'].apply(get_access_type)
```

In [17]:
```
#Visualizing Access types Distribution
var = 'access_type'
x = data[var].value_counts().values
y = data[var].value_counts().index

plt.figure(figsize=(7, 6))
plt.pie(x, labels = y, center=(0, 0), radius=1.5,  autopct='%1.1f%%', pctdistance=0
plt.title(f'{var} Distribution', fontsize = 15, fontweight = 'bold')
plt.axis('equal')
plt.show()
```



In [18]:
```
#Function to Extract Access Origin from Page using Regex
def get_access_origin(name):
    if len(re.findall(r'[ai].org_(.*)_(.*)$', name)) == 1 :
        return re.findall(r'[ai].org_(.*)_(.*)$', name)[0][1]
    else: return 'No Access_origin'

data['access_origin'] = data['Page'].apply(get_access_origin)
```

In [19]:
```
#Visualizing Access Origin Distribution
var = 'access_origin'
x = data[var].value_counts().values
y = data[var].value_counts().index

plt.figure(figsize=(7, 6))
plt.pie(x, labels = y, center=(0, 0), radius=1.5,  autopct='%1.1f%%', pctdistance=0
plt.title(f'{var} Distribution', fontsize = 15, fontweight = 'bold')
plt.axis('equal')
plt.show()
```



==

3. Data Pre-processing

3.1 Creating dataframe: mean page visit per language

In [20]:
```
data_language = pd.DataFrame()
data_language = data.groupby('language').mean().transpose()
data_language.drop(['Unknown_language'], inplace = True, axis = 1)
data_language.reset_index(inplace = True)
data_language.set_index('index', inplace = True)
data_language
```

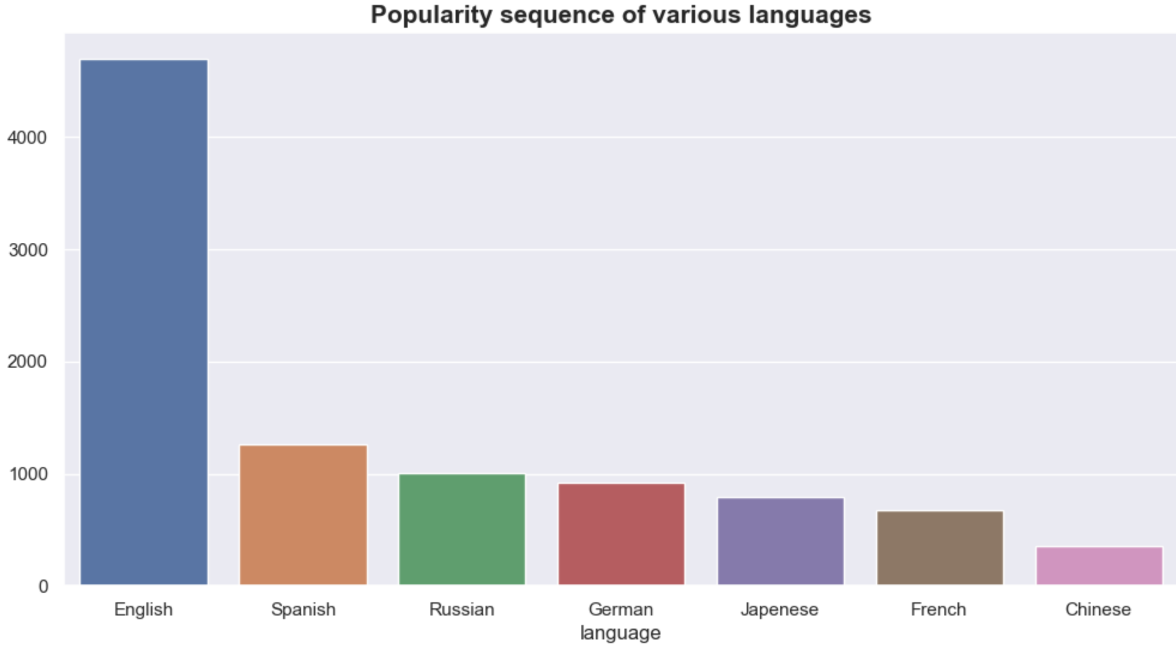
Out[20]:

language	Chinese	English	French	German	Japenese	Russian	Spanish
index							
2015-07-01	272.498521	3767.328604	499.092872	763.765926	614.637160	663.199229	1127.485204
2015-07-02	272.906778	3755.158765	502.297852	753.362861	705.813216	674.677015	1077.485425
2015-07-03	271.097167	3565.225696	483.007553	723.074415	637.451671	625.329783	990.895949
2015-07-04	273.712379	3711.782932	516.275785	663.537323	800.897435	588.171829	930.303151
2015-07-05	291.977713	3833.433025	506.871666	771.358657	768.352319	626.385354	1011.759575
...	...	...	...	...	...	...	...
2016-12-27	363.066991	6314.335275	840.590217	1119.596936	808.541436	998.374071	1070.923400
2016-12-28	369.049701	6108.874144	783.585379	1062.284069	807.430163	945.054730	1108.996753
2016-12-29	340.526330	6518.058525	763.209169	1033.939062	883.752786	909.352207	1058.660320
2016-12-30	342.745316	5401.792360	710.502773	981.786430	979.278777	815.475123	807.551177
2016-12-31	352.184275	5280.643467	654.060656	937.842875	1228.720808	902.600210	776.934322

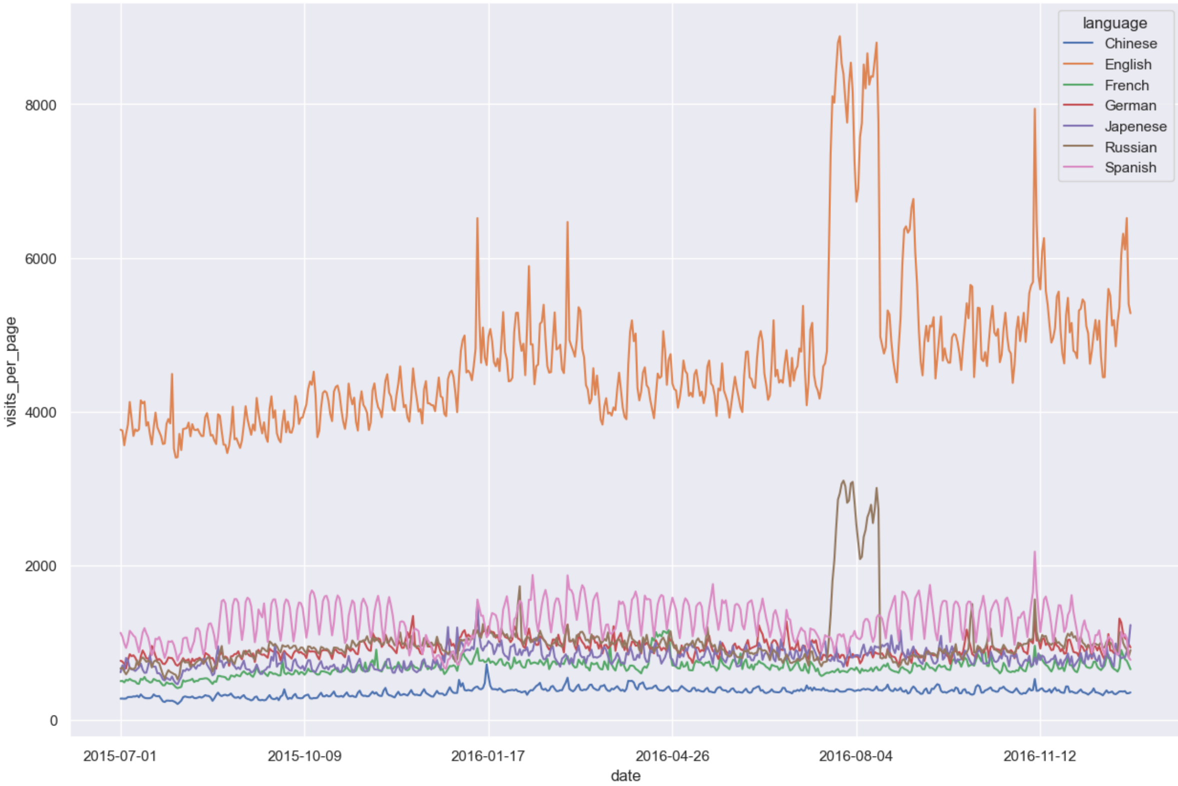
550 rows × 7 columns

In [21]:

```
x = data_language.mean().sort_values(ascending = False).index
y = data_language.mean().sort_values(ascending = False).values

plt.figure(figsize=(12, 6))
sns.barplot(x,y)
plt.title(f'Popularity sequence of various languages', fontsize = 15, fontweight =
plt.show()

## Popularity sequence of various languages : English > Spanish > Russian > German
```



In [22]:
```
data_language.plot(label = data_language.columns,  figsize=(15, 10))
plt.xlabel("date")
plt.ylabel("visits_per_page")
plt.show()
```



---

4. Checking Stationarity using ADF (Augmented Dickey Fuller) Test

ADF Test

- **Null Hypothesis:** The series has a unit root (value of a=1). The series is non-stationary.

- **Alternate Hypothesis:** The series has no unit root. The series is stationary.
- If we fail to reject the null hypothesis, we can say that the series is non-stationary.
- If p_value < 0.05 (alpha) or test statistic is less than the critical value, then we can reject the null hypothesis (aka the series is stationary)

In [23]:
```
#define function for ADF test
from statsmodels.tsa.stattools import adfuller
def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    df_output = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Use
    for key, value in dftest[4].items():
       df_output['Critical Value (%s)' %key] = value
    print (df_output)
```

In [24]:
```
#apply adf test on the series
adf_test(data_language['English'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                 -2.373563
p-value                         0.149337
#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
```

- **The test statistic > critical value / p_value > 5%.**
- **This implies that the series is not stationary.**

==

5. Decomposing Time Series

In this case we have used Additive Model for deconstructing the time series. The term additive means individual components (trend, seasonality, and residual) are added together as shown in equation below:

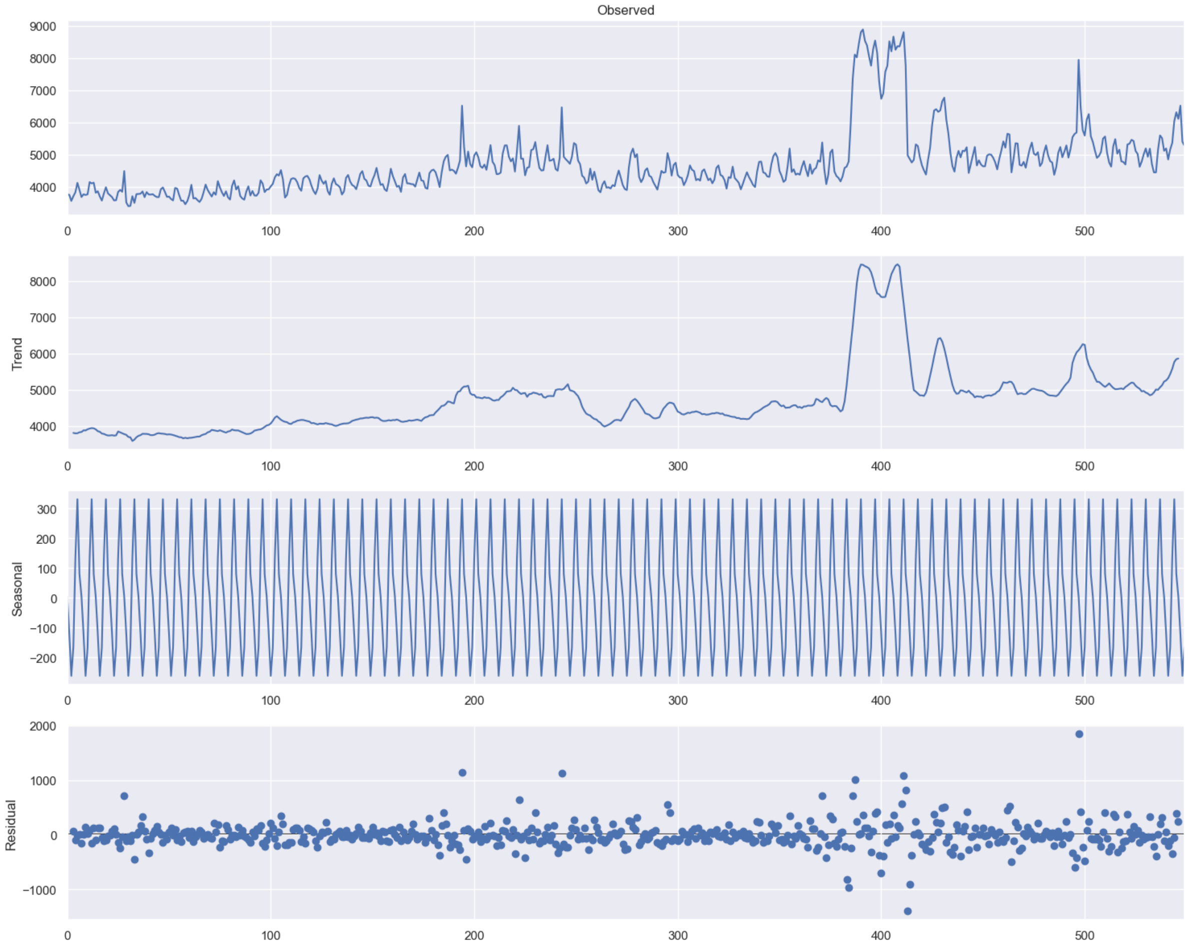
$y_t = T_t + S_t + R_t$

where

- y_t = actual value in time series
- T_t = trend in time series
- S_t = seasonality in time series
- R_t = residuals of time series

In [25]:
```
ts_english = data_language.English.values
```

In [26]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_english, model='additive', period=7)

fig = decomposition.plot()
fig.set_size_inches((15, 12))
fig.tight_layout()
plt.show()
```



In [27]:
```
residual = pd.DataFrame(decomposition.resid).fillna(0)[0].values
adf_test(residual)
```

```
Results of Dickey-Fuller Test:
Test Statistic                 -1.152195e+01
p-value                         4.020092e-21
#Lags Used                      1.700000e+01
Number of Observations Used     5.320000e+02
Critical Value (1%)            -3.442702e+00
Critical Value (5%)            -2.866988e+00
Critical Value (10%)           -2.569672e+00
dtype: float64
```

- **The test statistic < critical value / p_value < 5%.**

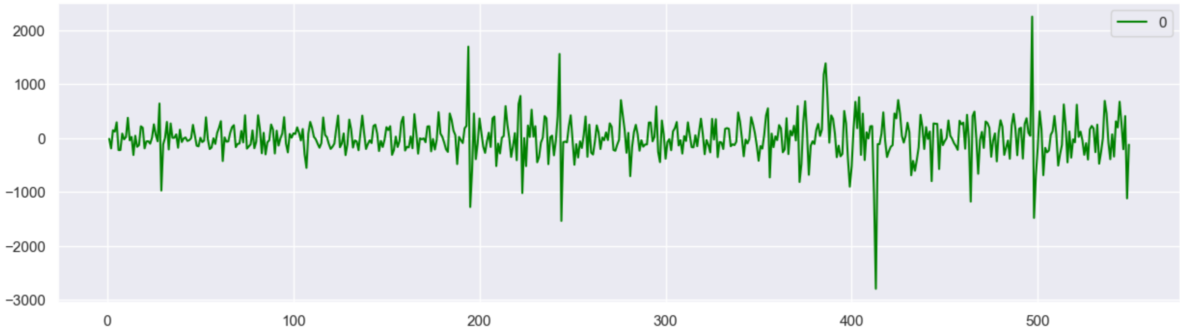
From ADF (Augmented Dickey Fuller) Test it can be shown that **Residuals** from time-series decomposition is **Stationary**

==

6. Estimating (p,q,d) & Interpreting ACF and PACF plots

In [28]:
```
ts_diff = pd.DataFrame(ts_english).diff(1)
ts_diff.dropna(inplace = True)
```

In [29]:
```
ts_diff.plot(color = 'green', figsize=(15, 4))
plt.show()
```

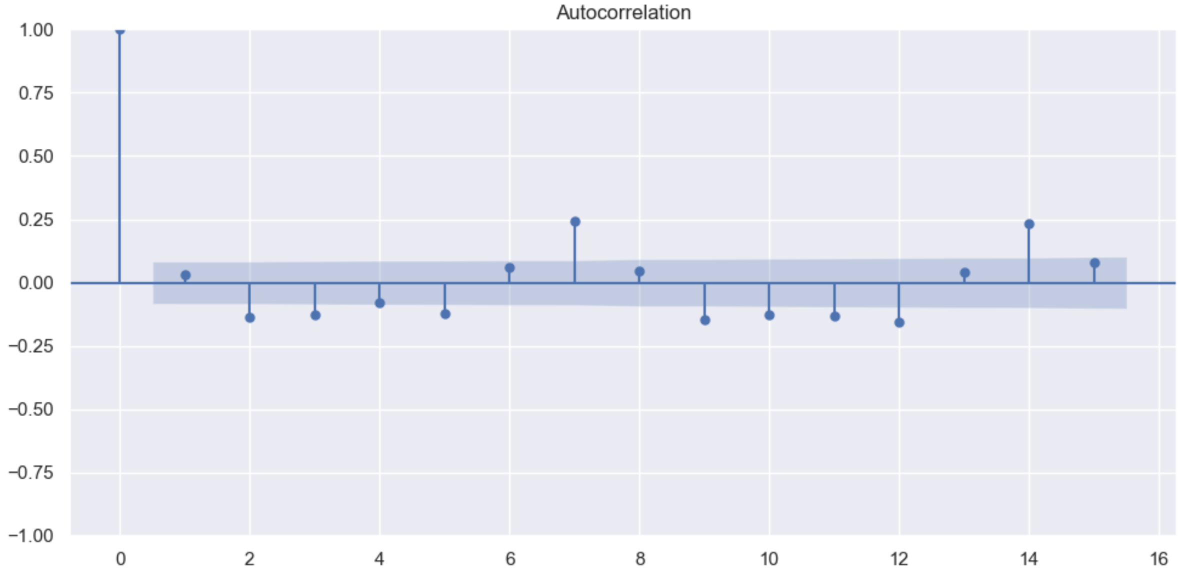


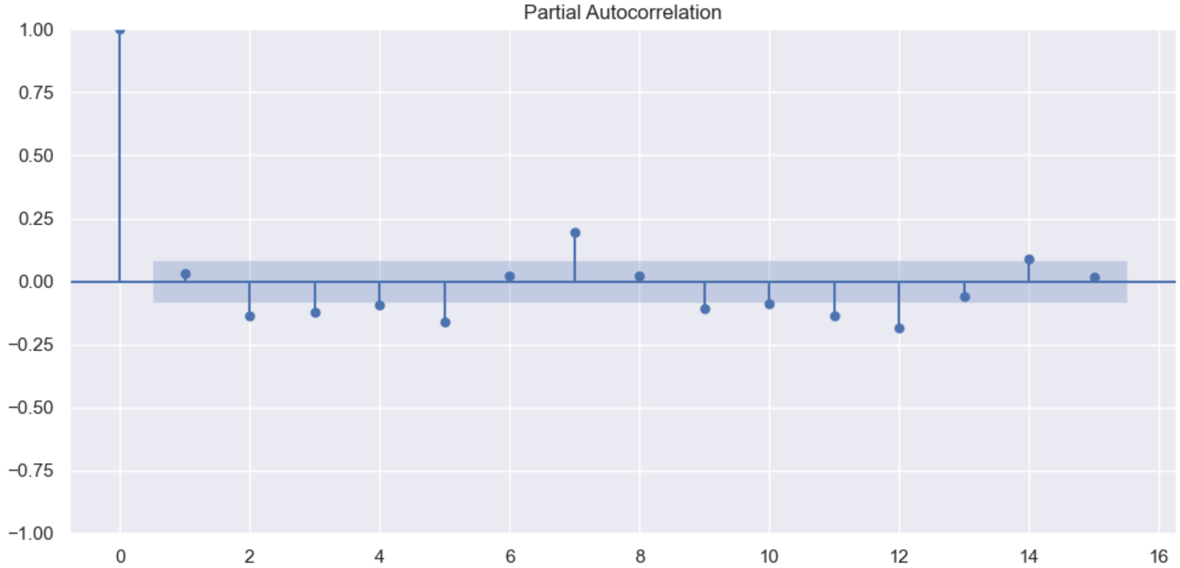
In [30]:
```
#ADF Test for differenced time-series
adf_test(ts_diff)
#p_value < 5%  ==> time series is stationary
```

```
Results of Dickey-Fuller Test:
Test Statistic                -8.273590e+00
p-value                        4.721272e-13
#Lags Used                     1.300000e+01
Number of Observations Used    5.350000e+02
Critical Value (1%)           -3.442632e+00
Critical Value (5%)           -2.866957e+00
Critical Value (10%)          -2.569655e+00
dtype: float64
```

==> **After one differencing time-series becomes stationary. This indicates for ARIMA model, we can set d = 1.**

In [31]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf = plot_acf(ts_diff, lags= 15)
acf.set_size_inches((10, 5))
acf.tight_layout()
pacf = plot_pacf(ts_diff, lags= 15)
pacf.set_size_inches((10, 5))
pacf.tight_layout()
```





==> **ACF & PACF indicates we should choose p = 0 & q = 0. But we will start with p=1 & q=1 for base ARIMA Model**

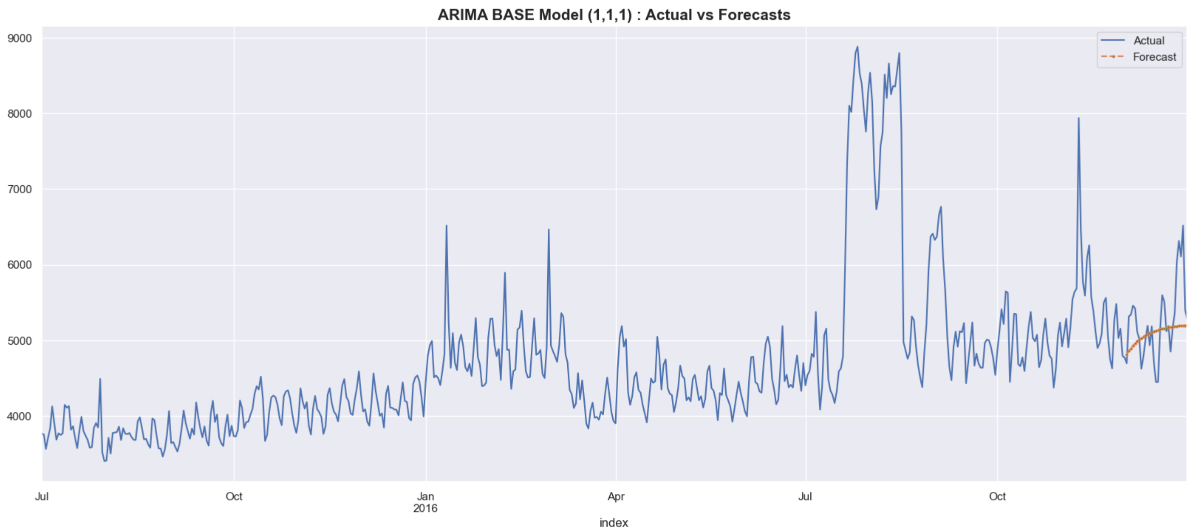
---

7. Forecasting Model Creation

7.1 ARIMA Base Model

In [32]:
```
from statsmodels.tsa.arima.model import ARIMA

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


n = 30
time_series = data_language.English.copy(deep = True)
#Creating Base ARIMA Model with order(1,1,1)
model = ARIMA(time_series[:-n], order =(1,1,1))
model_fit = model.fit()

#Creating forecast for last n-values
```

```
forecast = model_fit.forecast(steps = n, alpha = 0.05)

#plotting Actual & Forecasted values
time_series.index = time_series.index.astype('datetime64[ns]')
forecast.index = forecast.index.astype('datetime64[ns]')
plt.figure(figsize = (20,8))
time_series.plot(label = 'Actual')
forecast.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, fontweig
plt.show()


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

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

print('-'*80)
print(f'MAPE of Model : {np.round(mape,5)}')
print('-'*80)
print(f'RMSE of Model : {np.round(rmse,3)}')
print('-'*80)
```




```
--------------------------------------------------------------------------------
MAPE of Model : 0.06691
--------------------------------------------------------------------------------
RMSE of Model : 496.72
--------------------------------------------------------------------------------
```

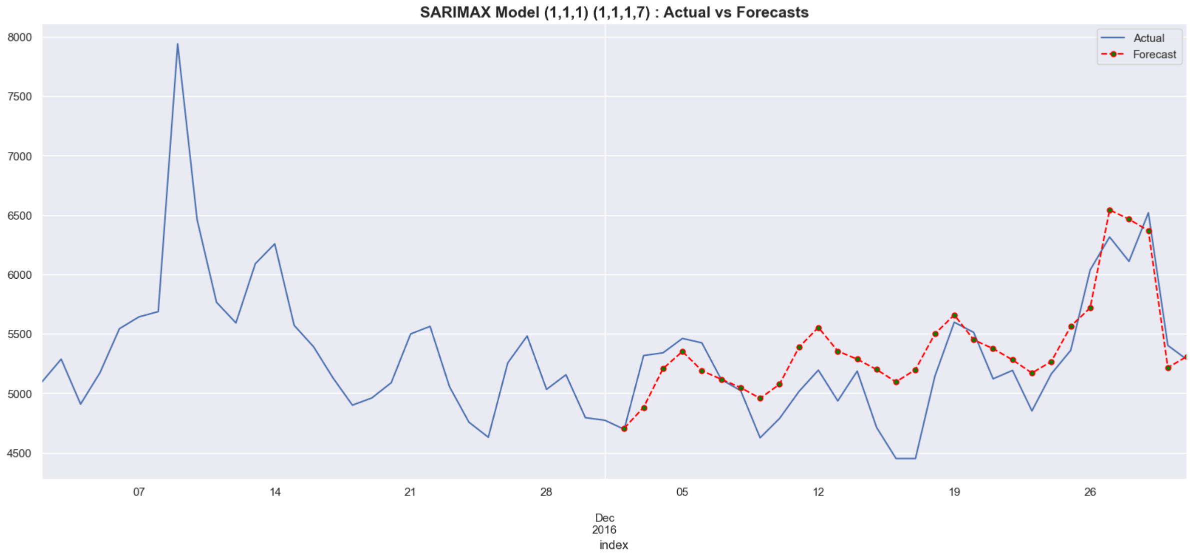
==> **ARIMA Base model has ~6% MAPE and RMSE ~ 500.**

7.2 Creation for function for SARIMAX model

In [33]:
```
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
    time_series.index = time_series.index.astype('datetime64[ns]')
    model_forecast.index = model_forecast.index.astype('datetime64[ns]')
    plt.figure(figsize = (20,8))
    time_series[-60:].plot(label = 'Actual')
    model_forecast[-60:].plot(label = 'Forecast', color = 'red',
                              linestyle='dashed', marker='o',markerfacecolor='green
    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('-'*80)
    print(f'MAPE of Model : {np.round(mape,5)}')
    print('-'*80)
    print(f'RMSE of Model : {np.round(rmse,3)}')
    print('-'*80)
```

In [34]:
```
#Checking a SARIMAX model with seasonality (p,d,q,P,D,Q,s = 1,1,1,1,1,1,7)
exog = exo_var['Exog'].to_numpy()
time_series = data_language.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 : 0.04888
--------------------------------------------------------------------------------
RMSE of Model : 307.388
--------------------------------------------------------------------------------
```

==> **SIMPLE SARIMAX model has ~4.9% MAPE and RMSE ~ 300.**

==> Impact of Seasonality & exogenous variable was captured properly in this model.

7.3 Searching for best parameters for SARIMAX model

7.3.1 Finding Best parameters for 'English' Pages

In [35]:

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

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

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

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

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

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

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

    return param_df
```

In [36]:

```
#long time to execute
#Finding best parameters for English time series

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

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

```
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
```

In [37]:
```
english_params.sort_values(['mape', 'rmse']).head()
```

Out[37]:

	serial	pdq	PDQs	mape	rmse
322	323	(2, 1, 2)	(2, 1, 1, 7)	0.04141	271.076
196	197	(1, 1, 1)	(2, 1, 1, 7)	0.04177	270.774
241	242	(2, 0, 1)	(1, 0, 1, 7)	0.04226	270.011
41	42	(0, 0, 2)	(0, 1, 2, 7)	0.04325	287.492
46	47	(0, 0, 2)	(1, 1, 1, 7)	0.04330	285.502

==> **Best Possible parameters English Time Series are pdq = (2, 1, 2) & PDQs = (2, 1, 1, 7).**

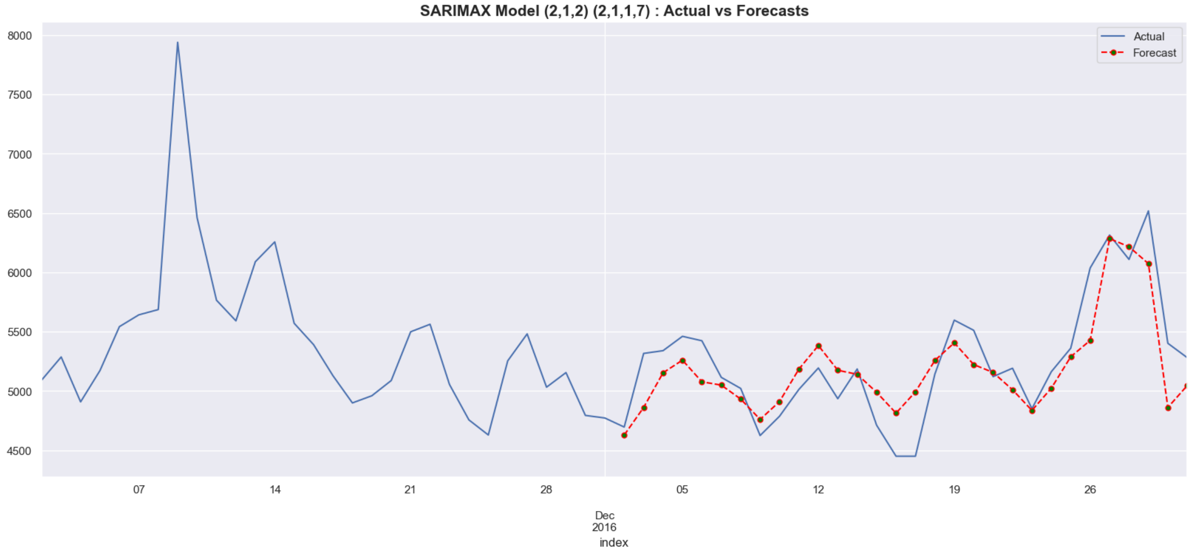
==> **Minimum MAPE = 4.141% and corresponding RMSE = 271.076.**

In [38]:
```
#Function to fetch best parameters for each language

def pipeline_sarimax_grid_search_without_exog(languages, data_language, n, param, d

    best_param_df  = pd.DataFrame(columns = ['language','p','d', 'q', 'P','D','Q','
    for lang in languages:
        print('')
        print('')
        print(f'--------------------------------------------------------------')
        print(f'          Finding best parameters for {lang}                   ')
        print(f'--------------------------------------------------------------')
        counter = 0
        time_series = data_language[lang]
        #creating df for storing results summary
        #param_df = pd.DataFrame(columns = ['serial','pdq', 'PDQs', 'mape', 'rmse']
        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_dif
                        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_foreca

                        #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, bes
        best_param_df.loc[len(best_param_df)] = best_param_row

    return best_param_df
```

In [39]:

```
#Plotting the SARIMAX model corresponding to best parameters
exog = exo_var['Exog'].to_numpy()
time_series = data_language.English
p,d,q, P,D,Q,s = 2,1,2, 2,1,1,7
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 : 0.04141
----------------------------------------------------------------------------------
RMSE of Model : 271.076
----------------------------------------------------------------------------------
```

7.4 Creating Pipeline to search Best parameters for all Pages

In [40]:

```
#long time to execute
#calculating best parameters for all languages
languages = ['Chinese', 'French', 'German', 'Japenese', 'Russian', 'Spanish']
n = 30
param = [0,1,2]
d_param = [0,1]
s_param = [7]


best_param_df = pipeline_sarimax_grid_search_without_exog(languages, data_language,
```

```
--------------------------------------------------------------
          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 = 0.03352
Corresponding Best Parameters are (0, 1, 1, 0, 0, 2, 7)
--------------------------------------------------------------



--------------------------------------------------------------
          Finding best parameters for French
--------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
--------------------------------------------------------------
Minimum MAPE for French = 0.05989
Corresponding Best Parameters are (0, 0, 2, 2, 1, 2, 7)
--------------------------------------------------------------



--------------------------------------------------------------
          Finding best parameters for German
--------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
```

```
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
--------------------------------------------------------------
Minimum MAPE for German = 0.06553
Corresponding Best Parameters are (2, 1, 0, 0, 1, 1, 7)
--------------------------------------------------------------


--------------------------------------------------------------
          Finding best parameters for Japenese
--------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
--------------------------------------------------------------
Minimum MAPE for Japenese = 0.07284
Corresponding Best Parameters are (0, 0, 2, 2, 0, 2, 7)
--------------------------------------------------------------


--------------------------------------------------------------
          Finding best parameters for Russian
--------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
```

```
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for Russian = 0.05342
Corresponding Best Parameters are (0, 0, 0, 1, 0, 1, 7)
-------------------------------------------------------------



-------------------------------------------------------------
          Finding best parameters for Spanish
-------------------------------------------------------------
Possible Combination: 18 out of 324 calculated
Possible Combination: 36 out of 324 calculated
Possible Combination: 54 out of 324 calculated
Possible Combination: 72 out of 324 calculated
Possible Combination: 90 out of 324 calculated
Possible Combination: 108 out of 324 calculated
Possible Combination: 126 out of 324 calculated
Possible Combination: 144 out of 324 calculated
Possible Combination: 162 out of 324 calculated
Possible Combination: 180 out of 324 calculated
Possible Combination: 198 out of 324 calculated
Possible Combination: 216 out of 324 calculated
Possible Combination: 234 out of 324 calculated
Possible Combination: 252 out of 324 calculated
Possible Combination: 270 out of 324 calculated
Possible Combination: 288 out of 324 calculated
Possible Combination: 306 out of 324 calculated
Possible Combination: 324 out of 324 calculated
-------------------------------------------------------------
Minimum MAPE for Spanish = 0.08209
Corresponding Best Parameters are (0, 1, 0, 2, 1, 0, 7)
-------------------------------------------------------------
```

In [41]:

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

Out[41]:

	language	p	d	q	P	D	Q	s	mape
0	Chinese	0	1	1	0	0	2	7	0.03352
4	Russian	0	0	0	1	0	1	7	0.05342
1	French	0	0	2	2	1	2	7	0.05989
2	German	2	1	0	0	1	1	7	0.06553
3	Japenese	0	0	2	2	0	2	7	0.07284
5	Spanish	0	1	0	2	1	0	7	0.08209

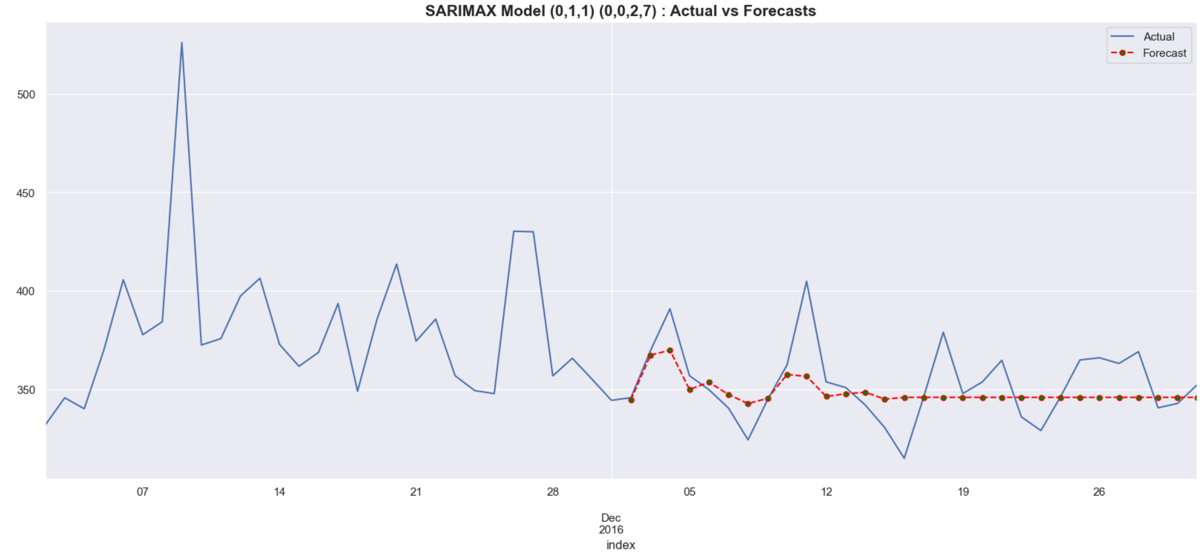
In [42]:

```
#Function to plot SARIMAX model for each Language

def plot_best_SARIMAX_model(languages, data_language, 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
```

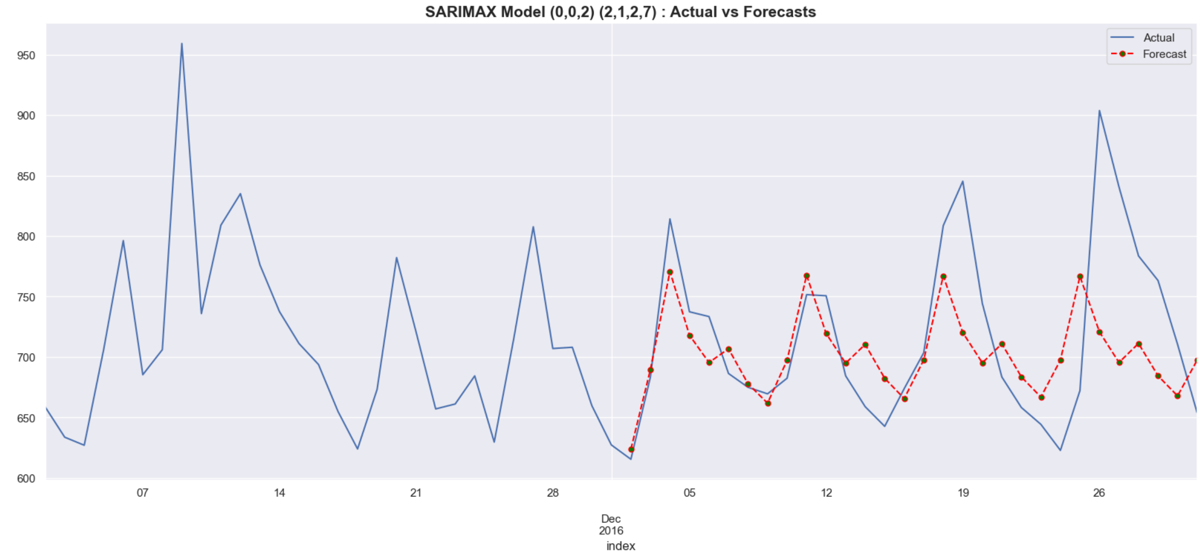
```
        #Creating language time-series
        time_series = data_language[lang]

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

    return 0
```

In [43]:
```
#Plotting SARIMAX model for each Language Time Series
languages = ['Chinese', 'French', 'German', 'Japenese', 'Russian', 'Spanish']
n = 30
plot_best_SARIMAX_model(languages, data_language, n, best_param_df)
```

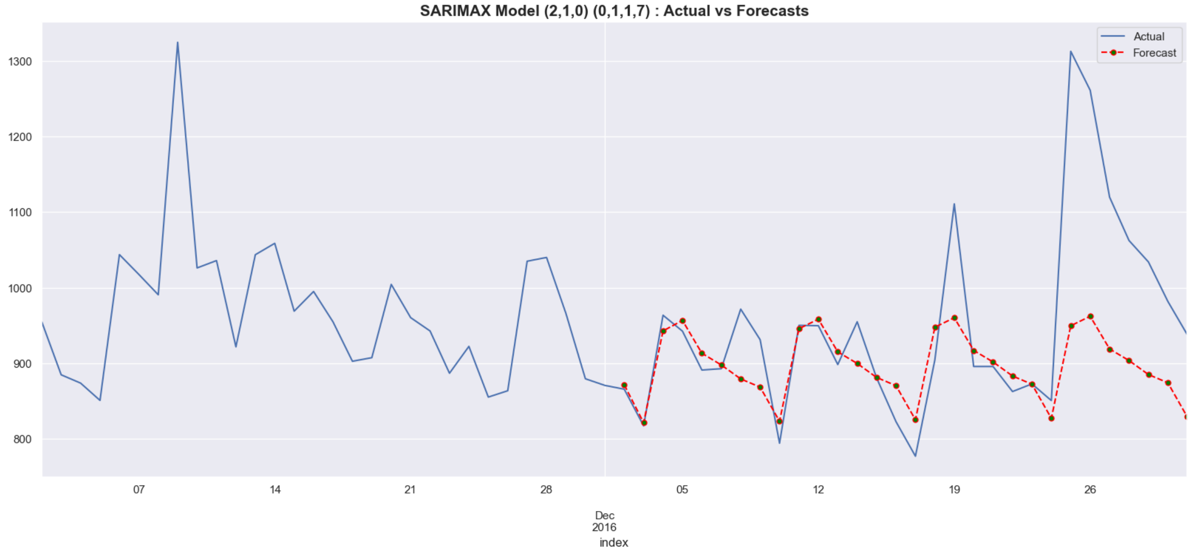
```
--------------------------------------------------------------------------------
-------
        SARIMAX model for Chinese Time Series
        Parameters of Model : (0,1,1) (0,0,2,7)
        MAPE of Model       : 0.03352
        RMSE of Model       : 16.433
--------------------------------------------------------------------------------
-------
```



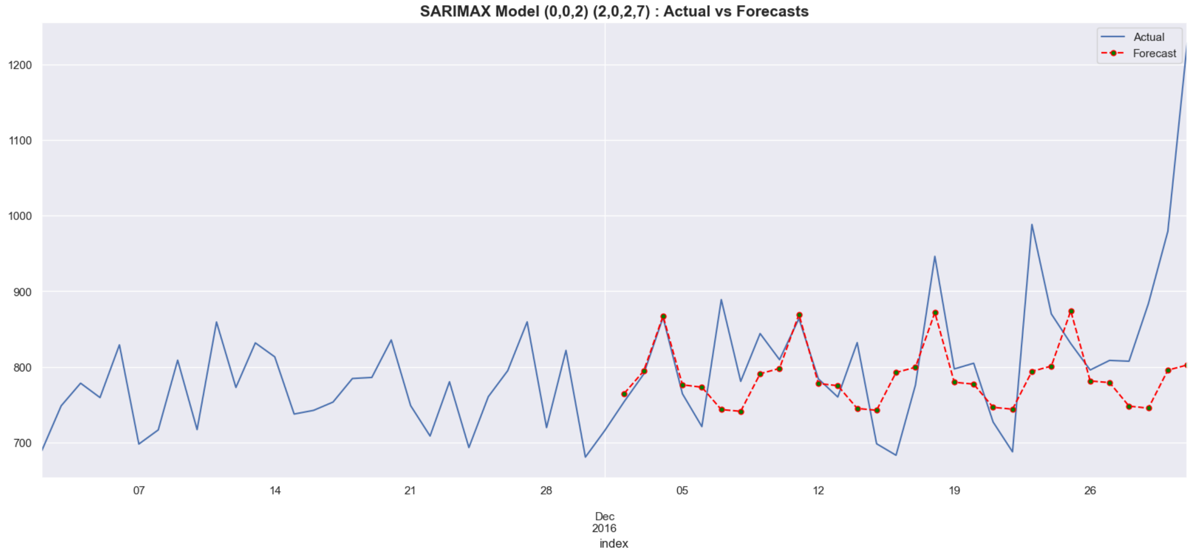

```
--------------------------------------------------------------------------------
-------
        SARIMAX model for French Time Series
        Parameters of Model : (0,0,2) (2,1,2,7)
        MAPE of Model       : 0.05989
        RMSE of Model       : 62.201
--------------------------------------------------------------------------------
-------
```



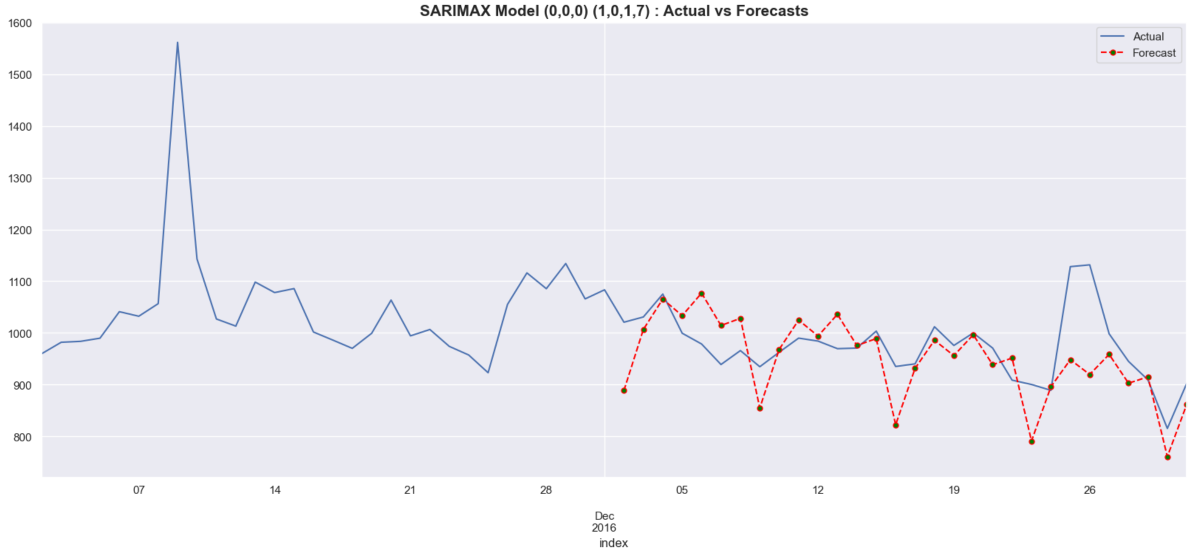

```
--------------------------------------------------------------------------------
-------
        SARIMAX model for German Time Series
        Parameters of Model : (2,1,0) (0,1,1,7)
        MAPE of Model       : 0.06553
        RMSE of Model       : 112.628
--------------------------------------------------------------------------------
-------
```



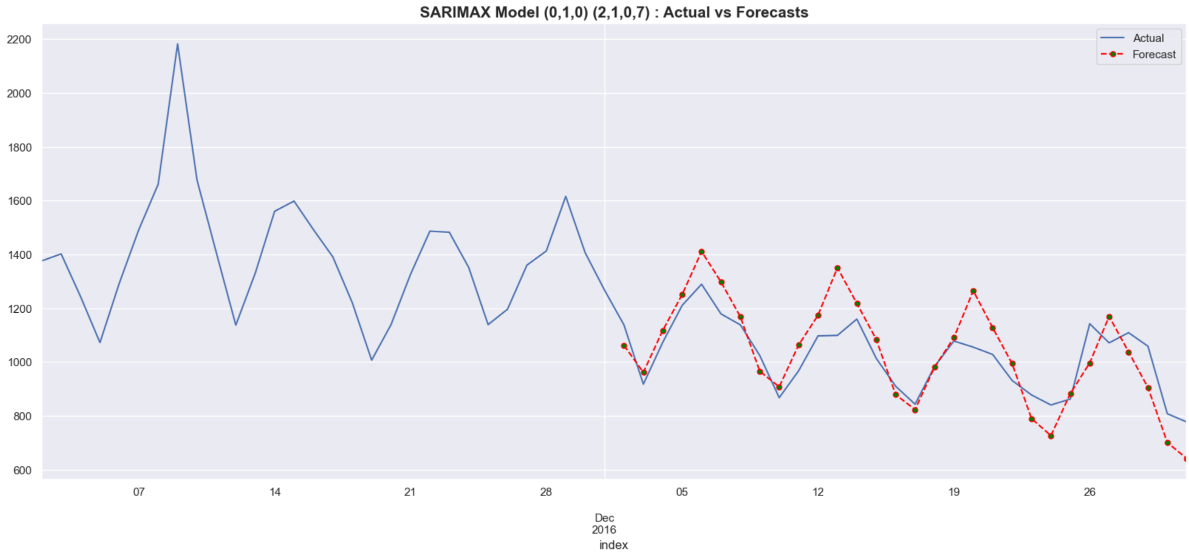

```
--------------------------------------------------------------------------------
-------
        SARIMAX model for Japenese Time Series
        Parameters of Model : (0,0,2) (2,0,2,7)
        MAPE of Model       : 0.07284
        RMSE of Model       : 107.208
--------------------------------------------------------------------------------
-------
```




```
--------------------------------------------------------------------------------
-------
        SARIMAX model for Russian Time Series
        Parameters of Model : (0,0,0) (1,0,1,7)
        MAPE of Model       : 0.05342
        RMSE of Model       : 74.078
--------------------------------------------------------------------------------
-------
```




```
--------------------------------------------------------------------------------
-------
        SARIMAX model for Spanish Time Series
        Parameters of Model : (0,1,0) (2,1,0,7)
        MAPE of Model       : 0.08209
        RMSE of Model       : 100.474
--------------------------------------------------------------------------------
-------
```



Out[43]:

```
0
```

8. Forecasting using Facebook Prophet

In [44]:

```python
from prophet import Prophet
```

In [45]:

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

In [46]:

```python
time_series
```

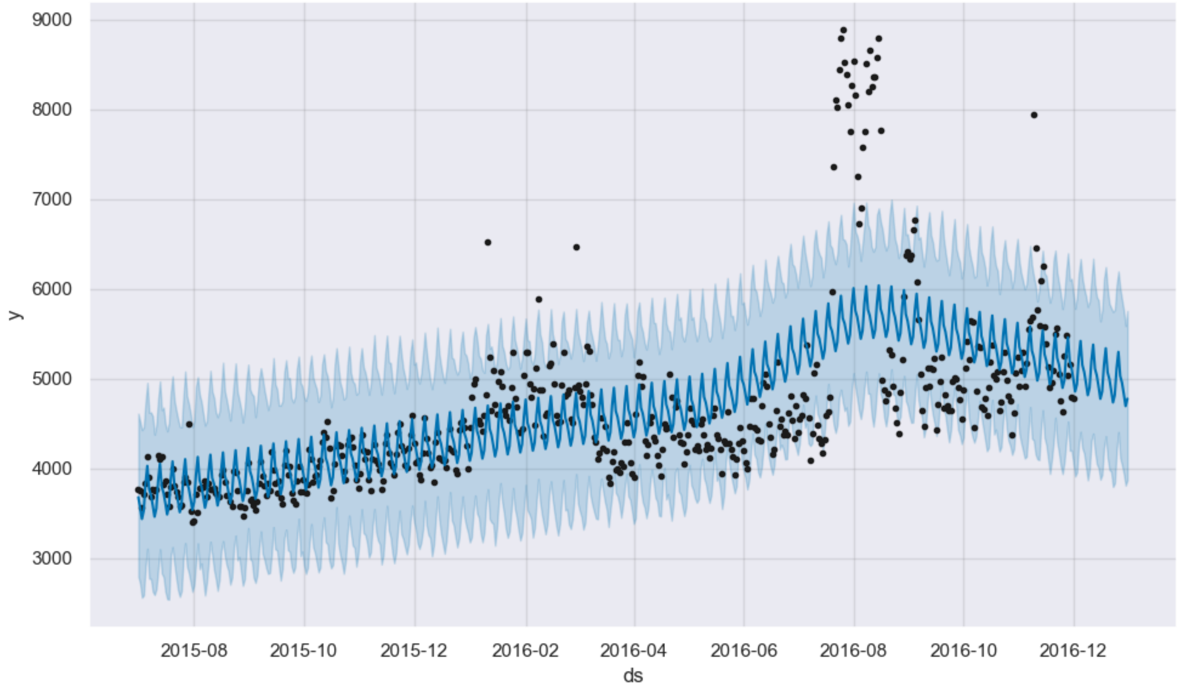
Out[46]:

	ds	y	exog
0	2015-07-01	3767.328604	0
1	2015-07-02	3755.158765	0
2	2015-07-03	3565.225696	0
3	2015-07-04	3711.782932	0
4	2015-07-05	3833.433025	0
...	...	...	...
545	2016-12-27	6314.335275	1
546	2016-12-28	6108.874144	1
547	2016-12-29	6518.058525	1
548	2016-12-30	5401.792360	0
549	2016-12-31	5280.643467	0

550 rows × 3 columns

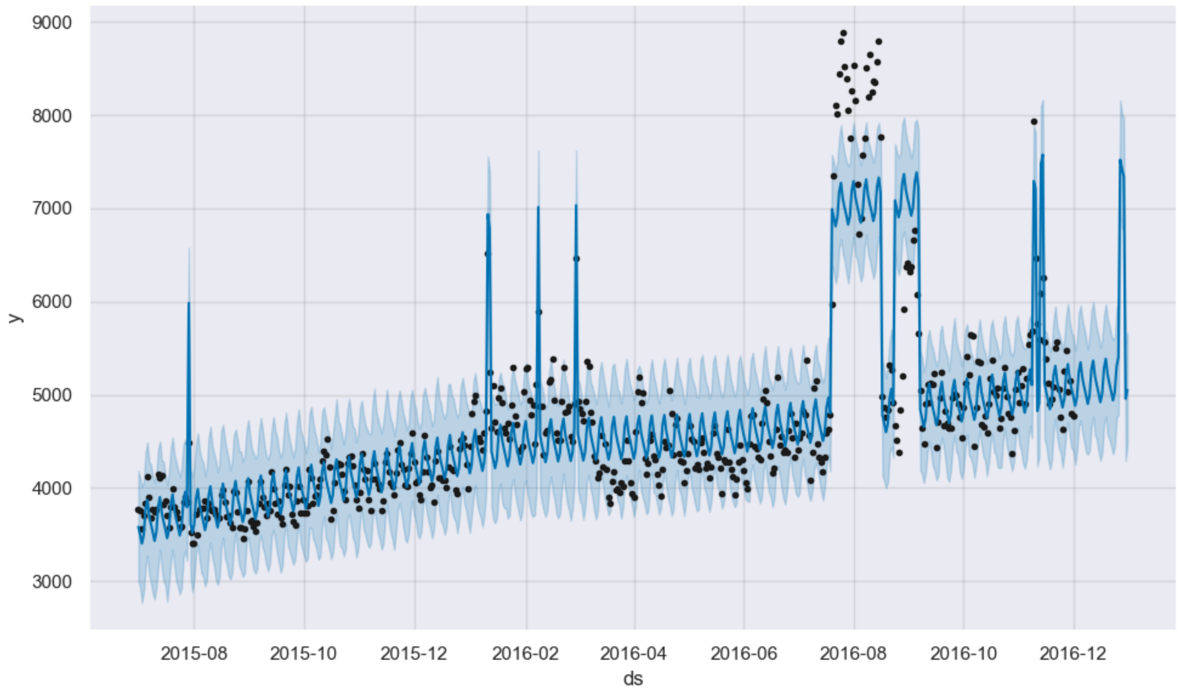
In [47]:
```
prophet1 = Prophet(weekly_seasonality=True)
prophet1.fit(time_series[['ds', 'y']][:-30])
future = prophet1.make_future_dataframe(periods=30, freq= 'D')
forecast = prophet1.predict(future)
fig1 = prophet1.plot(forecast)
```

```
12:33:56 - cmdstanpy - INFO - Chain [1] start processing
12:33:58 - cmdstanpy - INFO - Chain [1] done processing
```

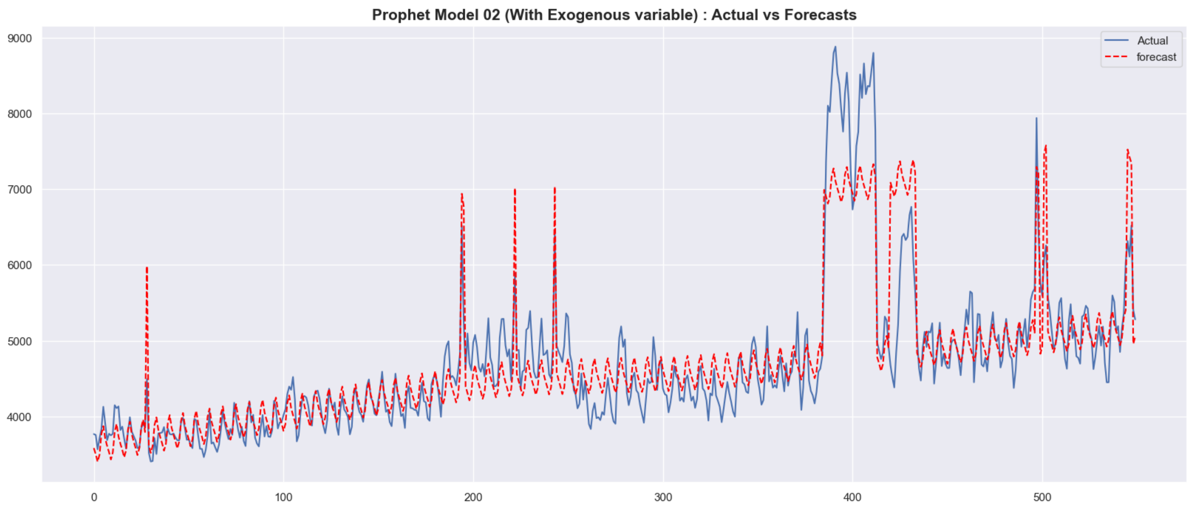


In [48]:
```
prophet2 = Prophet(weekly_seasonality=True)
prophet2.add_regressor('exog')
prophet2.fit(time_series[:-30])
#future2 = prophet2.make_future_dataframe(periods=30, freq= 'D')
forecast2 = prophet2.predict(time_series)
fig2 = prophet2.plot(forecast2)
```

```
12:34:06 - cmdstanpy - INFO - Chain [1] start processing
12:34:06 - cmdstanpy - INFO - Chain [1] done processing
```



In [49]:
```
actual = time_series['y'].values
forecast = forecast2['yhat'].values

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



In [50]:
```
errors = abs(actual - forecast)
mape = np.mean(errors/abs(actual))
mape
```

Out[50]: 0.059846174776769345

FB Prophet Model was created successfully. Forecast seems decent. This model is able to capture peaks because of exogenous variable.

Overall MAPE from Prophet model = ~6%

```
==================================================================
```

9. Business decisions / Recommendations

9.1 MAPE vrs Visits per Language

In [51]:
```
new_row = ['English', 1,1,1,2,1,1,7, 0.04189]
best_param_df.loc[len(best_param_df)] = new_row

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

Out[51]:

	language	p	d	q	P	D	Q	s	mape
0	Chinese	0	1	1	0	0	2	7	0.03352
6	English	1	1	1	2	1	1	7	0.04189
4	Russian	0	0	0	1	0	1	7	0.05342
1	French	0	0	2	2	1	2	7	0.05989
2	German	2	1	0	0	1	1	7	0.06553
3	Japenese	0	0	2	2	0	2	7	0.07284
5	Spanish	0	1	0	2	1	0	7	0.08209

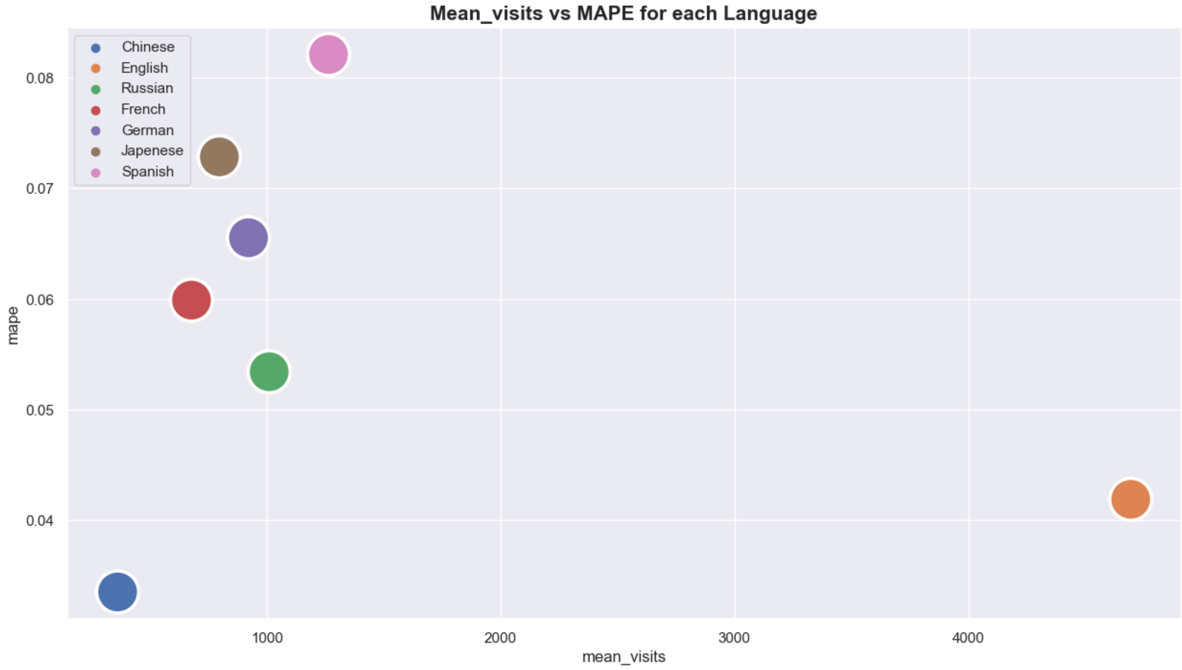
In [52]:
```
mean_visits = pd.DataFrame(data_language.mean()).reset_index()
mean_visits.columns = ['language', 'mean_visits']
df_visit_mape = best_param_df.merge(mean_visits, on = 'language')
```

In [53]:
```
df_visit_mape
```

Out[53]:

	language	p	d	q	P	D	Q	s	mape	mean_visits
0	Chinese	0	1	1	0	0	2	7	0.03352	360.019883
1	English	1	1	1	2	1	1	7	0.04189	4696.102005
2	Russian	0	0	0	1	0	1	7	0.05342	1008.694303
3	French	0	0	2	2	1	2	7	0.05989	676.223824
4	German	2	1	0	0	1	1	7	0.06553	920.132431
5	Japenese	0	0	2	2	0	2	7	0.07284	795.415559
6	Spanish	0	1	0	2	1	0	7	0.08209	1262.718183

In [54]:
```
plt.figure(figsize = (15,8))
sns.scatterplot(x="mean_visits", y="mape", hue="language", data=df_visit_mape, s=1(
plt.legend(loc="upper left")
plt.title(f'Mean_visits vs MAPE for each Language', fontsize = 15, fontweight = 'bc
plt.show()
```



Recommendations based on MAPE & mean_visits:

- **English** language is a clear winner. Maximum advertisement should be done on English pages. Their MAPE is low & mean visits are high.
- **Chinese** language has lowest number of visits. Advertisements on these pages should be avoided unless business has specific marketing strategy for Chinese populations.
- **Russian** language pages have decent number of visits and low MAPE. If used properly, these pages can result in maximum conversion.
- **Spanish** language has second highest number of visits but their MAPE is highest. There is a possibility advertisements on these pages won't reach the final people.
- **French, German & Japenese** have medium level of visits & medium MAPE levels. Depending on target customers advertisements should be run on these pages.

10. Questionnaire

Defining the problem statements and where can this and modifications of this be used?

Write 3 inferences you made from the data visualizations

What does the decomposition of series do?

What level of differencing gave you a stationary series?

Difference between arima, sarima & sarimax.

Compare the number of views in different languages

What other methods other than grid search would be suitable to get the model for all languages?

1. Defining the problem statements and where can this and modifications of this be used?

- We 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. We are provided with the data of 145k wikipedia pages and daily view count for each of them. Our clients belong to different regions and need data on how their ads will perform on pages in different languages.

- By creating a proper forecasting model to predict the fluctuations of visits on pages, we can help the business team to optimise the marketing spend. If we can predict the days with higher visits properly, the business will run the ads for those specific days and still be able to reach wider audience with most optimized spend.

```
In [ ]:
```

2. Write 3 inferences you made from the data visualizations.

- There are **7 Languages** found based on data provided. **English has highest number of pages** followed by Japense, German & French.
- There are **3 Access types** : **All-access(51.4%)**, mobile-web (24.9%) and desktop(23.6%).
- There are **2 Access-origins**: **all-agents (75.8%)** and spider (24.2%).

- **English** language is a clear winner. Maximum advertisement should be done on English pages. Their MAPE is low & mean visits are high.

- **Chinese** language has lowest number of visits. Advertisements on these pages should be avoided unless business has specific marketing strategy for Chinese populations.
- **Russian** language pages have decent number of visits and low MAPE. If used properly, these pages can result in maximum conversion.
- **Spanish** language has second highest number of visits but their MAPE is highest. There is a possibility advertisements on these pages won't reach the final people.
- **French, German & Japenese** have medium level of visits & medium MAPE levels. Depending on target customers advertisements should be run on these pages.

3. What does the decomposition of series do?

- The decomposition of time series is a statistical task that deconstructs a time series into several components, each representing one of the underlying categories of patterns.
- There are two principal types of decomposition : Additive & Multiplicative.
- In present business case we have used Additive Model for deconstructing the time series. The term additive means individual components (trend, seasonality, and residual) are added together as shown in equation below:

$y_t = T_t + S_t + R_t$

where

- y_t = actual value in time series
- T_t = trend in time series
- S_t = seasonality in time series
- R_t = residuals of time series

4. What level of differencing gave you a stationary series?

- A non-stationary time series can be converted to a stationary time series through a technique called differencing. Differencing series is the change between consecutive data points in the series.

y'_t = y_t - y_{t-1}

This is called first order differencing.

- In some cases, just differencing once will still yield a nonstationary time series. In that case a second order differencing is required.
- Seasonal differencing is the change between the same period in two different seasons. Assume a season has period, m

y'_t = y_t - y_{t-m}

- Once the time series becomes stationary, no differencing is required.

5. Difference between arima, sarima & sarimax.

- The **ARIMA model** is an ARMA model yet with a preprocessing step included in the model that we represent using I(d). I(d) is the difference order, which is the number of transformations needed to make the data stationary. So, an ARIMA model is simply an ARMA model on the differenced time series.

Equation of ARIMA model can be represented as below:

$$d_t = c + \sum_{n=1}^{p} \alpha_n d_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \epsilon_t$$

- In **SARIMA models** there is an additional set of autoregressive and moving average components.The additional lags are offset by the frequency of seasonality (ex. 12 — monthly, 24 — hourly). SARIMA models allow for differencing data by seasonal frequency, yet also by non-seasonal differencing.

Equation of SARIMA model can be represented as below:

$$y_t = c + \sum_{n=1}^{p} \alpha_n y_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \sum_{n=1}^{P} \phi_n y_{t-sn} + \sum_{n=1}^{Q} \eta_n \epsilon_{t-sn} + \epsilon_t$$

- **SARIMAX model** takes into account exogenous variables, or in other words, use external data in our forecast. Some real-world examples of exogenous variables include gold price, oil price, outdoor temperature, exchange rate.

Equation of SARIMAX model can be represented as below:

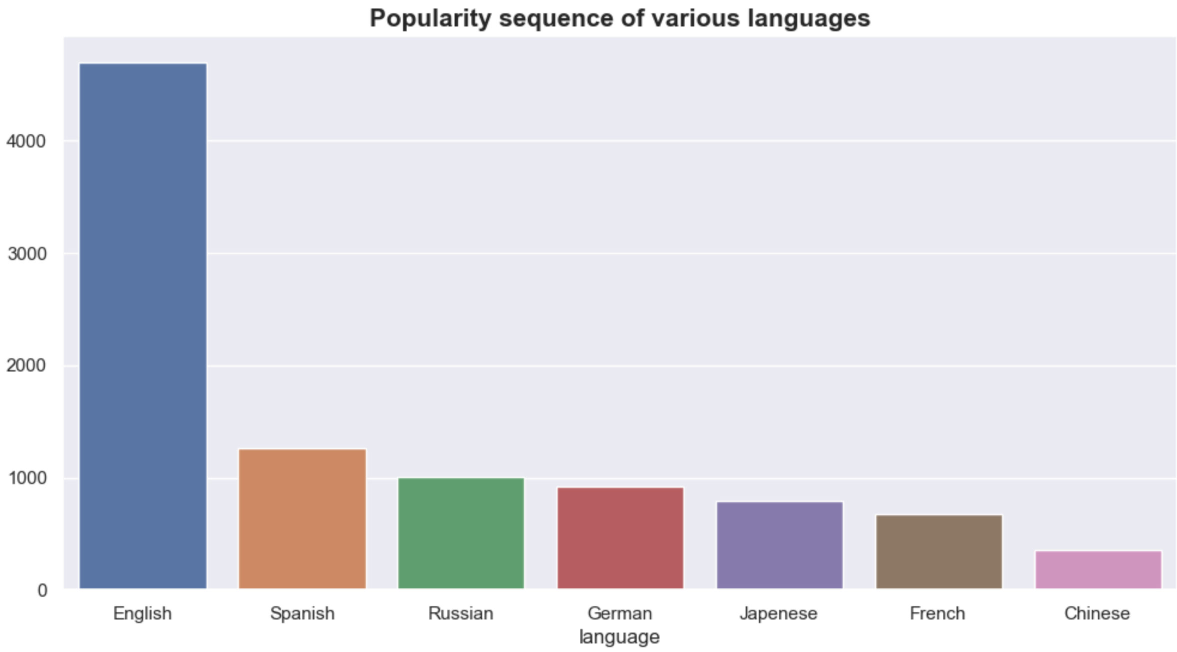
$$d_t = c + \sum_{n=1}^{p} \alpha_n d_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \sum_{n=1}^{r} \beta_n x_{n_t} + \sum_{n=1}^{P} \phi_n d_{t-sn} + \sum_{n=1}^{Q} \eta_n \epsilon_{t-sn} + \epsilon_t$$

6. Compare the number of views in different languages

- Mean number of views (Popularity sequence) of various languages have the following :

English > Spanish > Russian > German > Japenese > French > Chinese

In [55]:
```
x = data_language.mean().sort_values(ascending = False).index
y = data_language.mean().sort_values(ascending = False).values

plt.figure(figsize=(12, 6))
sns.barplot(x,y)
plt.title(f'Popularity sequence of various languages', fontsize = 15, fontweight =
plt.show()
```



7. What other methods other than grid search would be suitable to get the model for all languages?

- **Deep understanding of Domain / Business or relevant experience** in the same field can be good starting point for estimating the parameters of the model intiuitavely.
- Second level estimation can come from **ACF & PACF plots** of the time series. We can take following steps for estimation of p, q, d:
    - Test for stationarity using the augmented dickey fuller test.
    - If the time series is stationary try to fit the ARMA model, and if the time series is non-stationary then seek the **value of d.**

- If the data is getting stationary then draw the autocorrelation and partial autocorrelation graph of the data.
- Draw a partial autocorrelation graph(ACF) of the data. This will help us in finding the value of p because the **cut-off point to the PACF is p**.
- Draw an autocorrelation graph(ACF) of the data. This will help us in finding the value of q because the **cut-off point to the ACF is q**.

In [ ]:

In [ ]:

In [ ]: