Ad Ease website Analytics

Dataset: Web Traffic Time Series Forecasting

Forecasting the future values of multiple time series. More specifically the problem of forecasting future web traffic for approximately 145,000 wikipedia articles.

The training dataset consists of approximately 145k time series. Each of these time series represent a number of daily views of a different Wikipedia article, starting from July, 1st, 2015 up until December 31st, 2016. For each time series, you are provided the name of the article as well as the type of traffic that this time series represent (all, mobile, desktop, spider). You may use this metadata and any other publicly available data to make predictions. Unfortunately, the data source for this dataset does not distinguish between traffic values of zero and missing values. A missing value may mean the traffic was zero or that the data is not available for that day.

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 correspond to a particular date. The values are the number of visits in that date.

**Exog_Campaign_eng : *

this file contaings 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 is a 1 for dates with campaign 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

Intent of the notebook

1. We will start by loading the data and handling the values.
2. Then some Exploratory data analysis to get an understanding of the data and get some useful insight, based on various parameters, and visualizing them.
3. Preparing the data for feeding to the model(checking stationarity, transformations).
4. Preparing the model followed by some predictions.
5. Comparing the same with the given data and calculating the accuracy of the model.

Importing the libraries

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

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

```
import pandas as pd
import numpy as np
import pylab as p
import matplotlib.pyplot as plot
from collections import Counter
import re
import os
import seaborn as sns
```

```
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter("ignore")
```

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

```
#Data_link- https://drive.google.com/file/d/1sRlqw7-6S1u5p1YLoxJbyDZtmRegUooU/vie

train = pd.read_csv('/content/drive/MyDrive/train_1.csv')
```

Reading the dataset and printing head and tail to get basic idea

```
train.head()
```

```
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 609.8+ MB
None
```

We can see that ther are some null values in the data, we will plot them to see how it looks

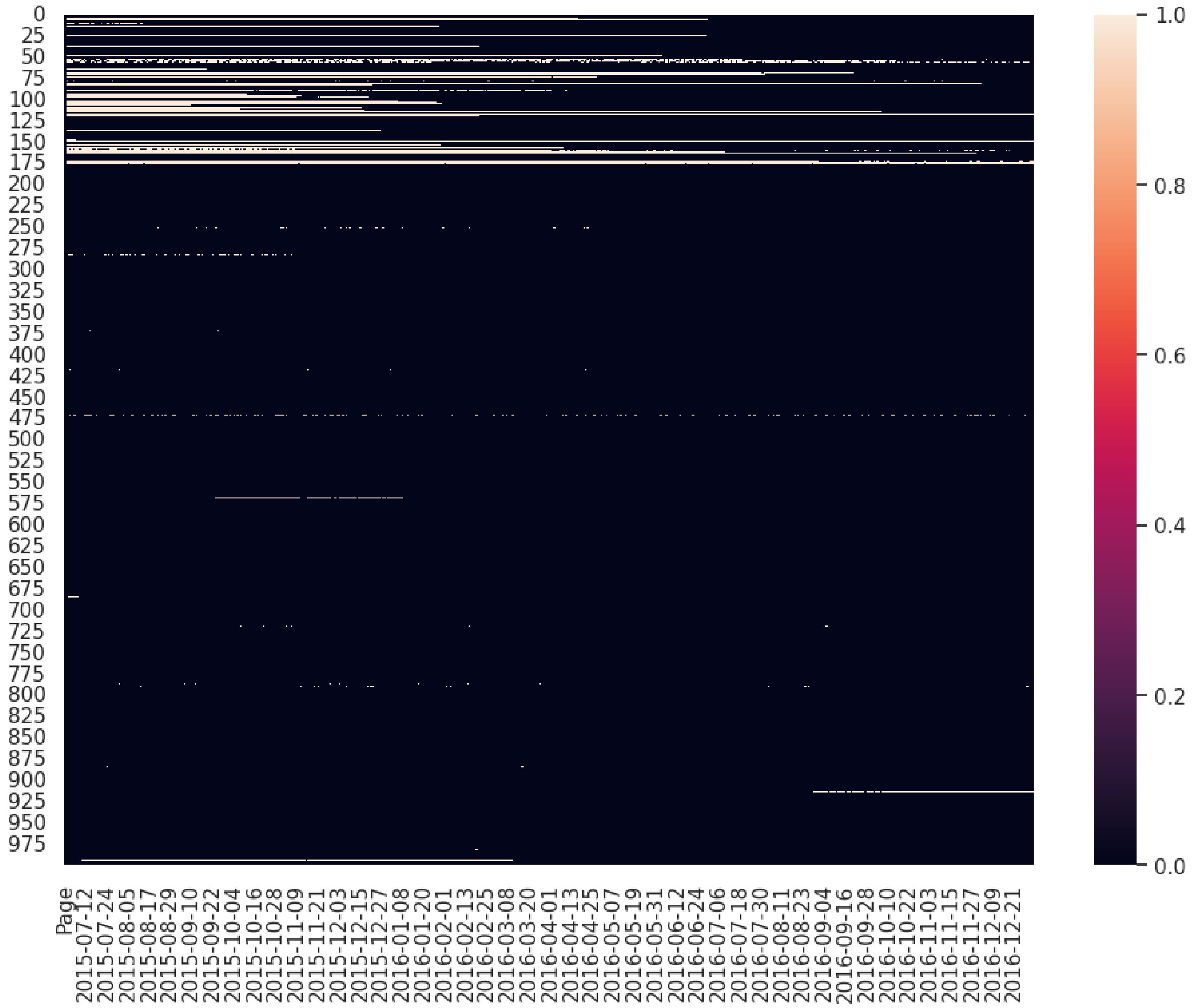
```
train.head(1000).isna().sum()
```

```
Page           0
2015-07-01    65
2015-07-02    65
```

```
2015-07-03    67
2015-07-04    64
              ..
2016-12-27     9
2016-12-28    10
2016-12-29     9
2016-12-30     8
2016-12-31     9
Length: 551, dtype: int64
```

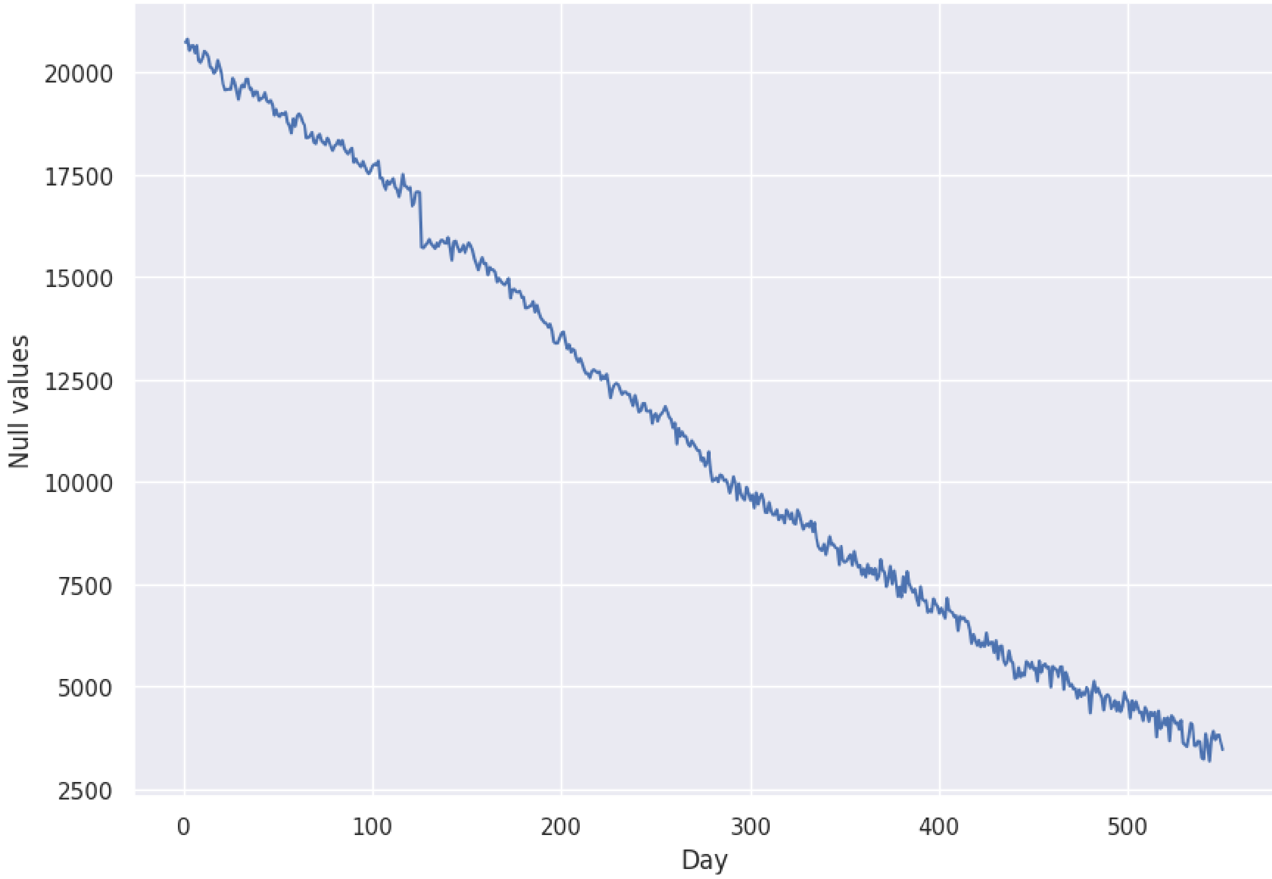
```
import seaborn as sbn
sbn.heatmap(train.head(1000).isna())
```

```
<Axes: >
```



```
days = [r for r in range(1, len(train.columns))]
plot.figure(figsize=(10,7))
plot.xlabel('Day')
plot.ylabel('Null values')
plot.plot(days, train.isnull().sum()[1:])
```

```
[<matplotlib.lines.Line2D at 0x7a77bdfbdae0>]
```



We see that the number of nan values decrease with time.

Probable reason: Some website have all nan values in the begining, that can be due to the fact that those were created after that time so there is no traffic reading for that time

```
print(train.shape)
train=train.dropna(how='all')
#'all' : If all values are NA, drop that row or column.
print(train.shape)

train=train.dropna(thresh=300)
print(train.shape)
```

```
(145063, 551)
(145063, 551)
(133617, 551)
```

1. We try droping the rows that have all values as nan, none in our case.
2. We then also drop rows that have nan more than 300 days, because the time series for that would not make much sense
3. We fill all the remaining values with zero assuming there was no traffic on the date that the values are nan for.

```
train=train.fillna(0)
train.tail()
```

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04
145012	Legión_(Marvel_Comics)_es.wikipedia.org_all-ac...	0.0	0.0	0.0	0.0
145013	Referéndum_sobre_la_permanencia_del_Reino_Unid...	0.0	0.0	0.0	0.0
145014	Salida_del_Reino_Unido_de_la_Unión_Europea_es....	0.0	0.0	0.0	0.0
145015	Amar,_después_de_amar_es.wikipedia.org_all-acc...	0.0	0.0	0.0	0.0
145016	Anexo:89.°_Premios_Óscar_es.wikipedia.org_all-...	0.0	0.0	0.0	0.0

5 rows × 551 columns

EDA

The page values are in this format

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

having information about page name, the main domain, device type used to access the page, and also the request origin(spider or browser agent)

```
#train['langgg']=train['Page'].apply(lambda x: x[x.find('wikipedia')-3:x.find('wi
```

```
#Usage of Regex
def split_page(page):
    w = re.split('_|\.', page)
    print(w)
    return ' '.join(w[:-5]), w[-5], w[-2], w[-1]

split_page('2NE1_zh.wikipedia.org_all-access_spider')
```

```
['2NE1', 'zh', 'wikipedia', 'org', 'all-access', 'spider']
('2NE1', 'zh', 'all-access', 'spider')
```

```
def split_page(page):
  w = re.split('_|\.', page)
  return ' '.join(w[:-5]), w[-5], w[-2], w[-1]

li = list(train.Page.apply(lambda x: split_page(str(x))))
df = pd.DataFrame(li)
df.columns = ['Title', 'Language', 'Access_type','Access_origin']
df = pd.concat([train, df], axis = 1)
```

We split the page name and get that information joining it with a temporary database. below we get some rows to see the structure of the data

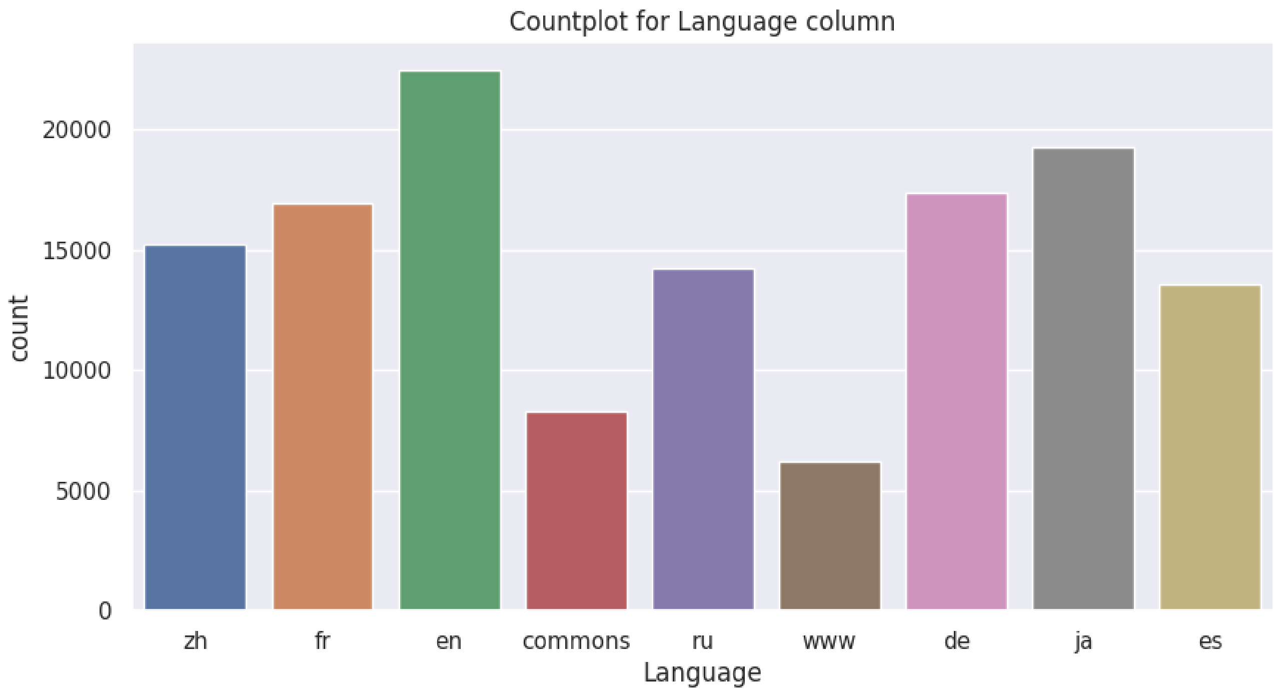
```
df.head()
```

2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Titl
9.0	9.0	22.0	26.0	...	14.0	20.0	22.0	19.0	18.0	20.0	2NE
13.0	22.0	11.0	10.0	...	9.0	30.0	52.0	45.0	26.0	20.0	2PN
4.0	0.0	3.0	4.0	...	4.0	4.0	6.0	3.0	4.0	17.0	3(
26.0	14.0	9.0	11.0	...	16.0	11.0	17.0	19.0	10.0	11.0	4minut
8.0	5.0	17.0	24.0	...	32.0	19.0	23.0	17.0	17.0	50.0	A'N'I

Generate Using ... countplot for language column using valuecounts Close

1 of 4 Use code with caution

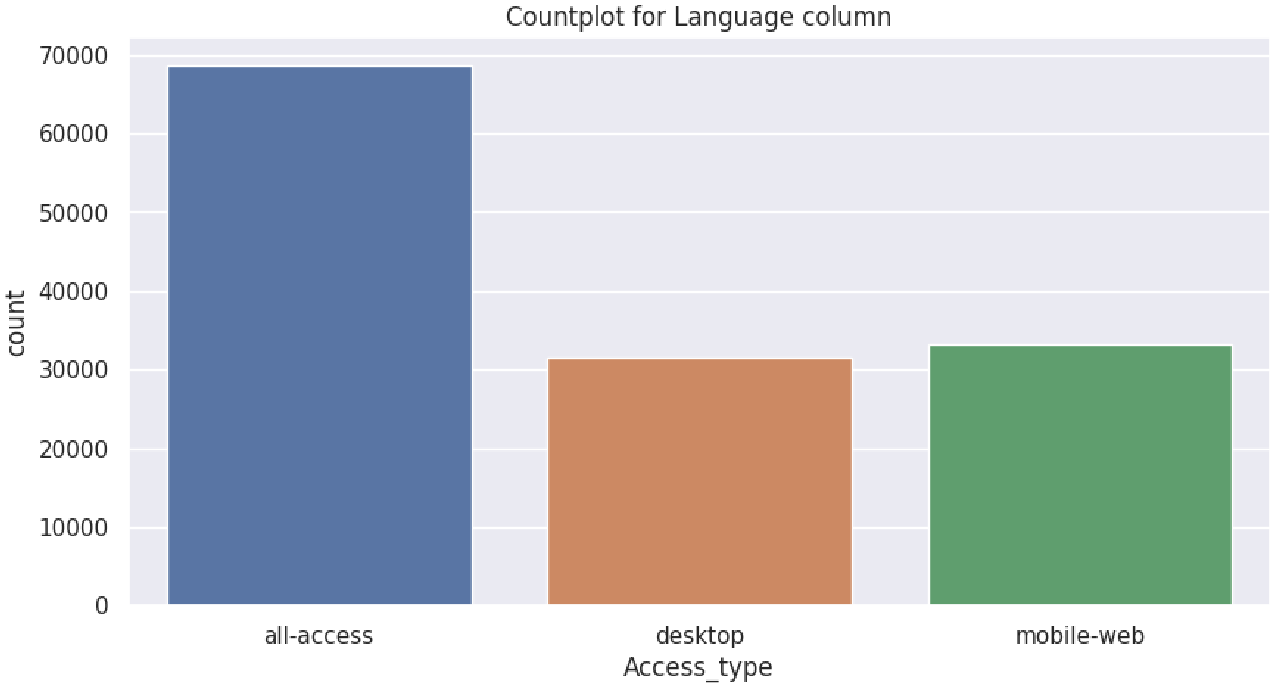
```
# prompt: countplot for language column using valuecounts

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
sns.countplot(x='Language', data=df)
plt.title('Countplot for Language column')
plt.show()
```



This above is the comparision number of articles in each language

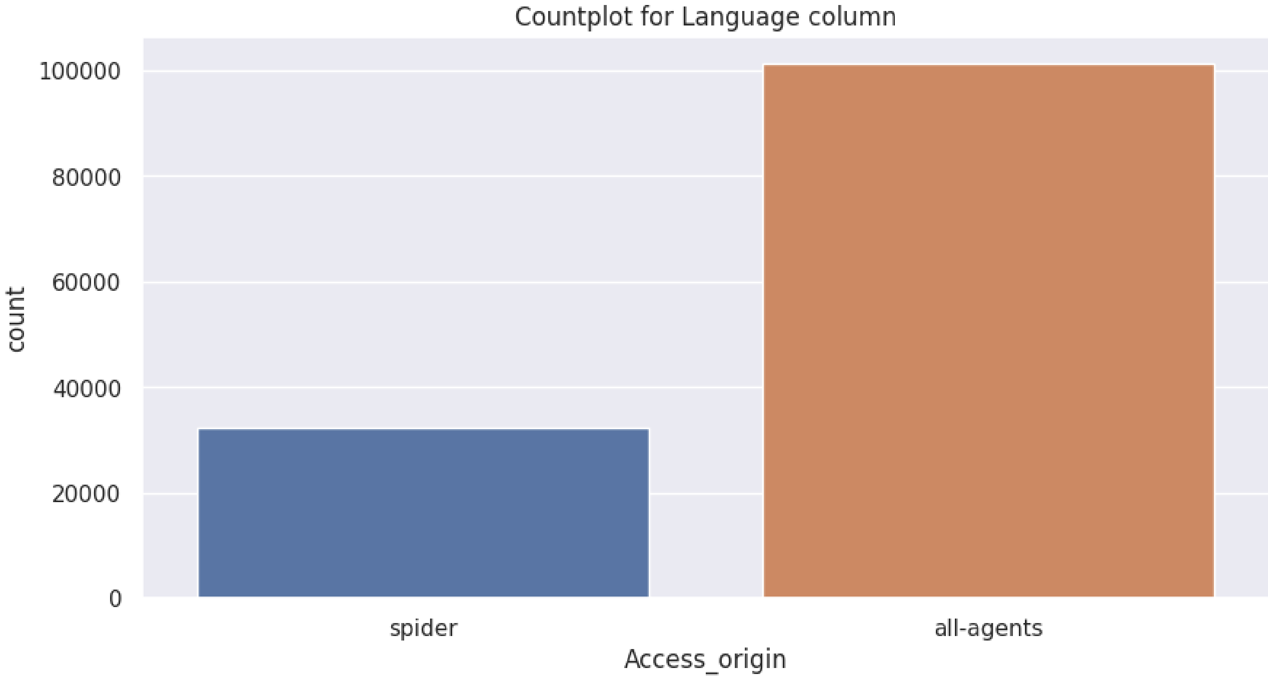
{'ja':'Japanese', 'de':'German', 'en' : 'English', 'no_lang':'Media_File', 'fr':'French', 'zh':'Chinese', 'ru':'Russian', 'es':'Spanish'}

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
sns.countplot(x='Access_type', data=df)
plt.title('Countplot for Language column')
plt.show()
```



This comparision shows that usage from desktop and mobile is almost the same

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
sns.countplot(x='Access_origin', data=df)
plt.title('Countplot for Language column')
plt.show()
```



This shows that organic view is far more than that of spiders or bots

*Now we want to compare the views for different languages *

```
#here we see that the languages are not treated properly as there are commons anc
df.groupby('Language').count()
```

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
Language									
commons	7672	7672	7672	7672	7672	7672	7672	7672	7672
de	15946	15946	15946	15946	15946	15946	15946	15946	15946
en	20758	20758	20758	20758	20758	20758	20758	20758	20758
es	12268	12268	12268	12268	12268	12268	12268	12268	12268
fr	15418	15418	15418	15418	15418	15418	15418	15418	15418
ja	17132	17132	17132	17132	17132	17132	17132	17132	17132
ru	12955	12955	12955	12955	12955	12955	12955	12955	12955
www	5743	5743	5743	5743	5743	5743	5743	5743	5743
zh	14845	14845	14845	14845	14845	14845	14845	14845	14845

9 rows × 554 columns

```
df[df['Language']=='commons']
```

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	20 07
12271	Burning_Man_en.wikipedia.org_desktop_all-agents	1693.0	1490.0	1186.0	1099.0	10
12272	Cali_Cartel_en.wikipedia.org_desktop_all-agents	348.0	363.0	214.0	252.0	2
12273	Call_of_Duty:_Modern_Warfare_2_en.wikipedia.or...	806.0	768.0	700.0	725.0	7
12274	Calvin_Harris_en.wikipedia.org_desktop_all-agents	7114.0	5599.0	7685.0	15844.0	93
12275	Carl_Sagan_en.wikipedia.org_desktop_all-agents	1808.0	1759.0	1838.0	1631.0	17
...	...	...	...	...	...	
75129	NaN	NaN	NaN	NaN	NaN	
75150	NaN	NaN	NaN	NaN	NaN	
75178	NaN	NaN	NaN	NaN	NaN	
75237	NaN	NaN	NaN	NaN	NaN	
75253	NaN	NaN	NaN	NaN	NaN	

8266 rows × 555 columns

```
# Checking another way of fetching the language out of the string
def lang(Page):
    val = re.search('[a-z][a-z].wikipedia.org',Page)
    if val:
        #print(val)
        #print(val[0][0:2] )

        return val[0][0:2]

    return 'no_lang'

df['Language']=df['Page'].apply(lambda x: lang(str(x)))
```

```
df.groupby('Language').count() #now the count has increased. You can go back and
```

	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
Language										
de	17362	17362	17362	17362	17362	17362	17362	17362	17362	17362
en	22486	22486	22486	22486	22486	22486	22486	22486	22486	22486
es	13551	13551	13551	13551	13551	13551	13551	13551	13551	13551
fr	16948	16948	16948	16948	16948	16948	16948	16948	16948	16948
ja	19295	19295	19295	19295	19295	19295	19295	19295	19295	19295
no_lang	14494	14494	14494	14494	14494	14494	14494	14494	14494	14494
ru	14270	14270	14270	14270	14270	14270	14270	14270	14270	14270
zh	15211	15211	15211	15211	15211	15211	15211	15211	15211	15211

8 rows × 554 columns

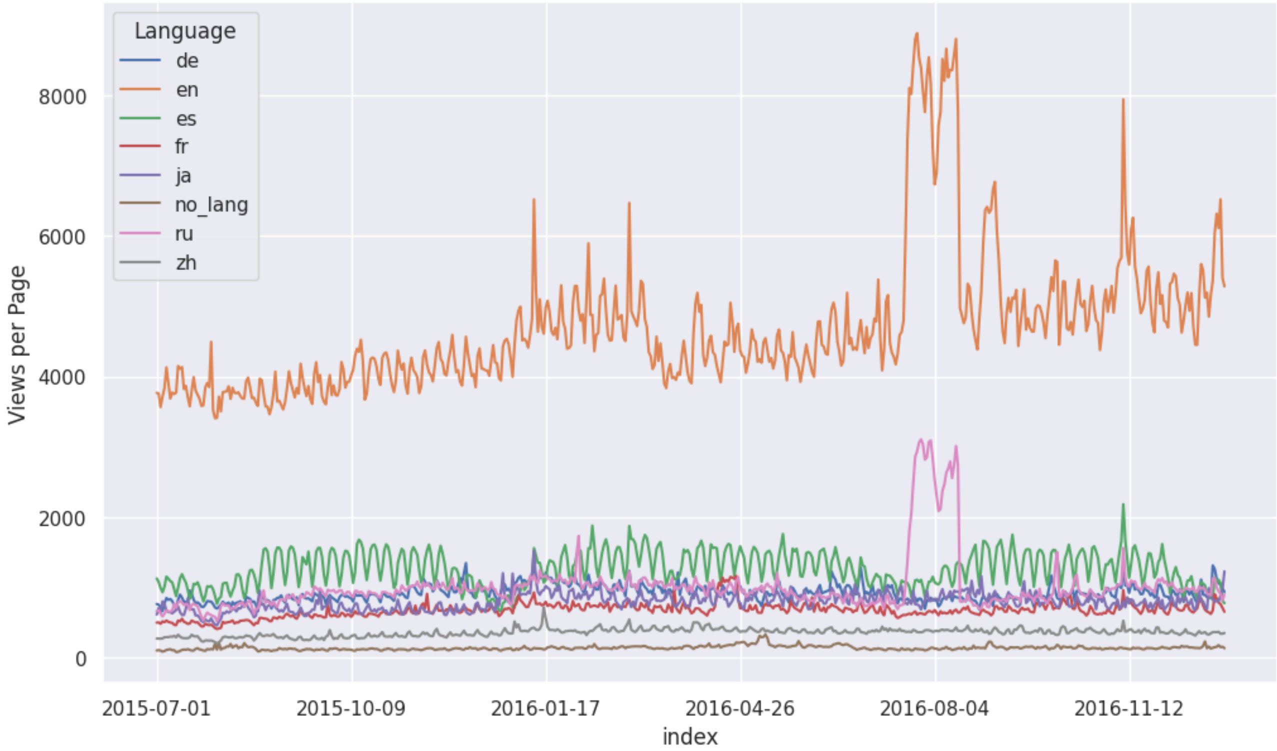
```
df_language=df.groupby('Language').mean().transpose()
df_language
```

Language	de	en	es	fr	ja	no_lang
2015-07-01	763.765926	3767.328604	1127.485204	499.092872	614.637160	102.733545
2015-07-02	753.362861	3755.158765	1077.485425	502.297852	705.813216	107.663447
2015-07-03	723.074415	3565.225696	990.895949	483.007553	637.451671	101.769629
2015-07-04	663.537323	3711.782932	930.303151	516.275785	800.897435	86.853871
2015-07-05	771.358657	3833.433025	1011.759575	506.871666	768.352319	96.254105
...	...	...	...	...	...	...
2016-12-27	1119.596936	6314.335275	1070.923400	840.590217	808.541436	155.270181
2016-12-28	1062.284069	6108.874144	1108.996753	783.585379	807.430163	178.561267

```
df_language.reset_index(inplace=True)
df_language.set_index('index', inplace=True)
```

```
df_language.plot(figsize=(12,7))
plot.ylabel('Views per Page')
```

```
Text(0, 0.5, 'Views per Page')
```

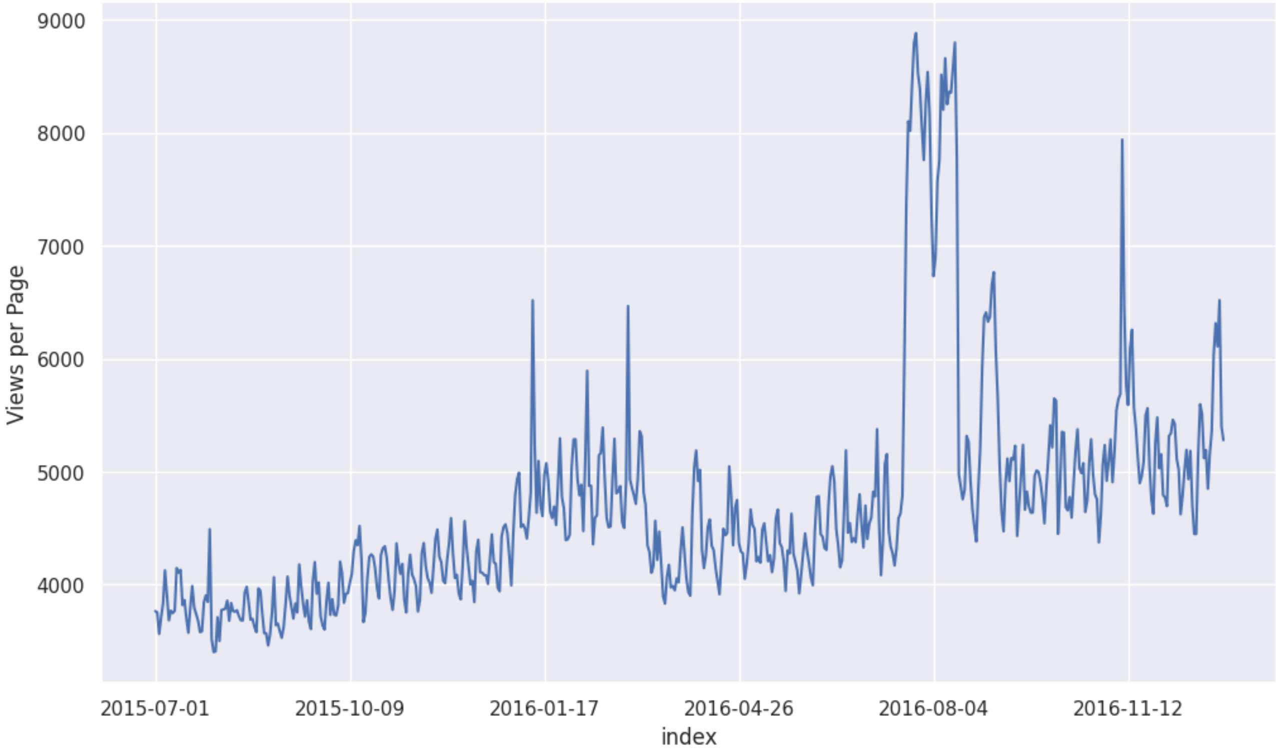


Ploting the data shows that articles in english get the most number of views as compared to different languages, there are some spikes at different times in different laguages

Ploting just for english because we are going to use this for our furthur investigation and predictions

```
df_language['en'].plot(figsize=(12,7))
plot.ylabel('Views per Page')
```

```
Text(0, 0.5, 'Views per Page')
```




```
total_view=df_language.copy()
```

```
##########################################################################################
```

Checking the stationarity

Dickey-Fuller test

Here the null hypothesis is that the TS is non-stationary: The test results comprise of a Test Statistic and some Critical Values for difference confidence levels.

```
from statsmodels.tsa.stattools import adfuller
def df_test(x):
    result=adfuller(x)
    print('ADF Stastistic: %f'%result[0])
    print('p-value: %f'%result[1])

df_test(total_view['en'])
```

```
ADF Stastistic: -2.373563
p-value: 0.149337
```

We see that the p value is not low enough(<0.05). Therefore, we can say our series in not stationary as we fail to reject the null hypothesis

Making the time series stationary

```
ts=total_view['en']
```

1. Remove trend and seasonality with decomposition

```
# Naive decomposition of our Time Series as explained above
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts.values, model='multiplicative',extrapolate_

""" Additive or multiplicative?
  It’s important to understand what the difference between a multiplicative time

  There are three components to a time series:
  – trend how things are overall changing
  – seasonality how things change within a given period e.g. a year, month, week,
  – error/residual/irregular activity not explained by the trend or the seasonal

  How these three components interact determines the difference between a multipl

  In a multiplicative time series, the components multiply together to make the t

  In an additive time series, the components add together to make the time series


"""

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plot.figure(figsize=(10,7))
plot.subplot(411)
plot.title('Observed = Trend + Seasonality + Residuals')
plot.plot(ts.values,label='Observed')
plot.legend(loc='best')
plot.subplot(412)
plot.plot(trend, label='Trend')
plot.legend(loc='best')
plot.subplot(413)
plot.plot(seasonal,label='Seasonality')
plot.legend(loc='best')
plot.subplot(414)
plot.plot(residual, label='Residuals')
plot.legend(loc='best')
plot.tight_layout()
plot.show()
```

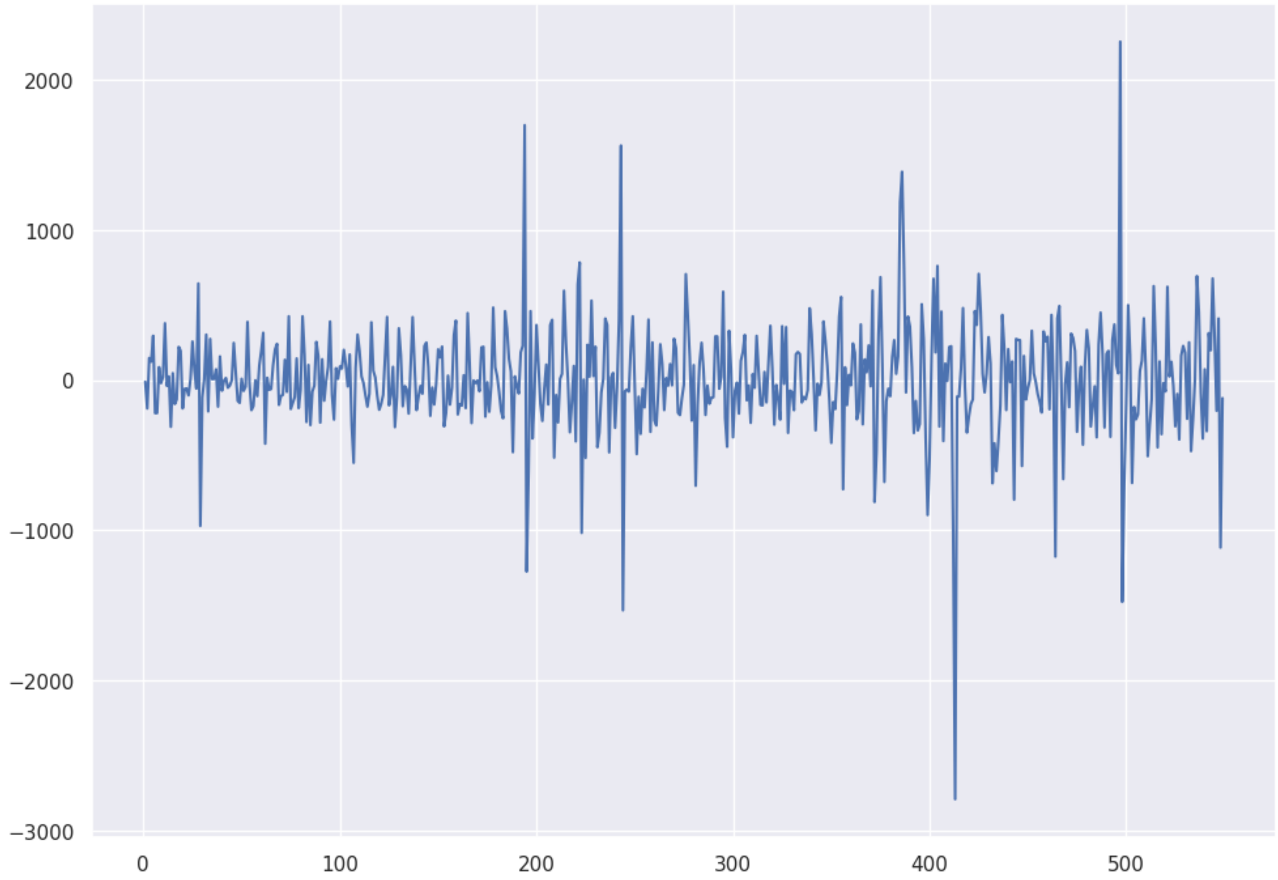

```
ts_decompose=pd.DataFrame(residual).fillna(0)[0].values
df_test(ts_decompose)
```

```
ADF Stastistic: -10.254420
p-value: 0.000000
```

We can see that our series is now stationary, we can also try diffrencing to see what results we can get.

2. Remove trend and seasonality with differencing

```
ts_diff = ts - ts.shift(1)
plot.plot(ts_diff.values)
plot.show()
```




```
ts_diff.dropna(inplace=True)
df_test(ts_diff)
```

```
ADF Stastistic: -8.273590
p-value: 0.000000
```

Also the p value is 0. So we can say that our graph is now stationery. Now we can apply the ARIMA model

How do we choose p,d,q

a thumb rule that for choosing the p,q values are when the lag goes below the significant level

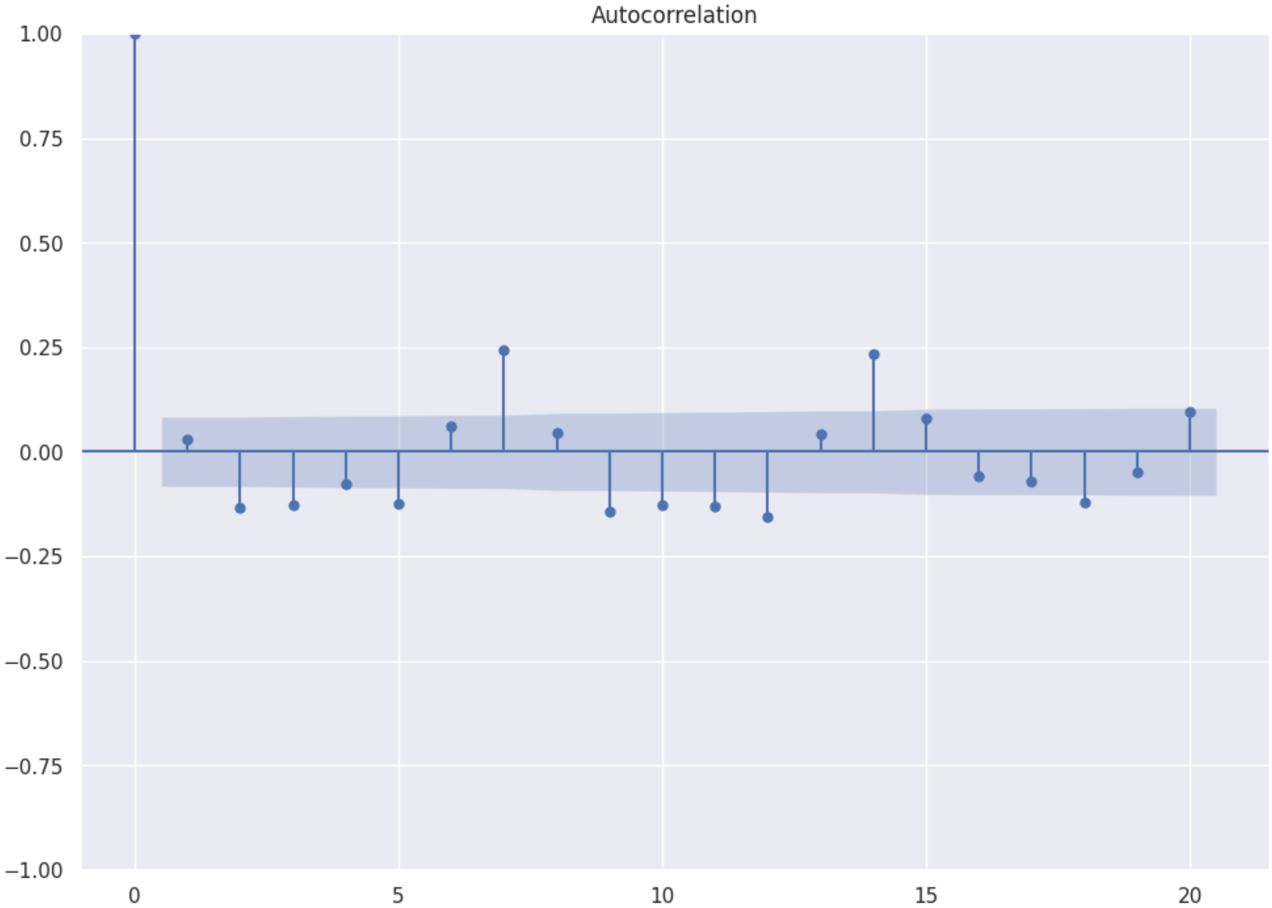
- we use PACF for p, here we see that till lag 5 there are significat lines, if we want our model to be simpler we can start with a smaller number like 3/4
- we use ACF for q. here we can see that lag 4 is below significant level so we will use till lag 3

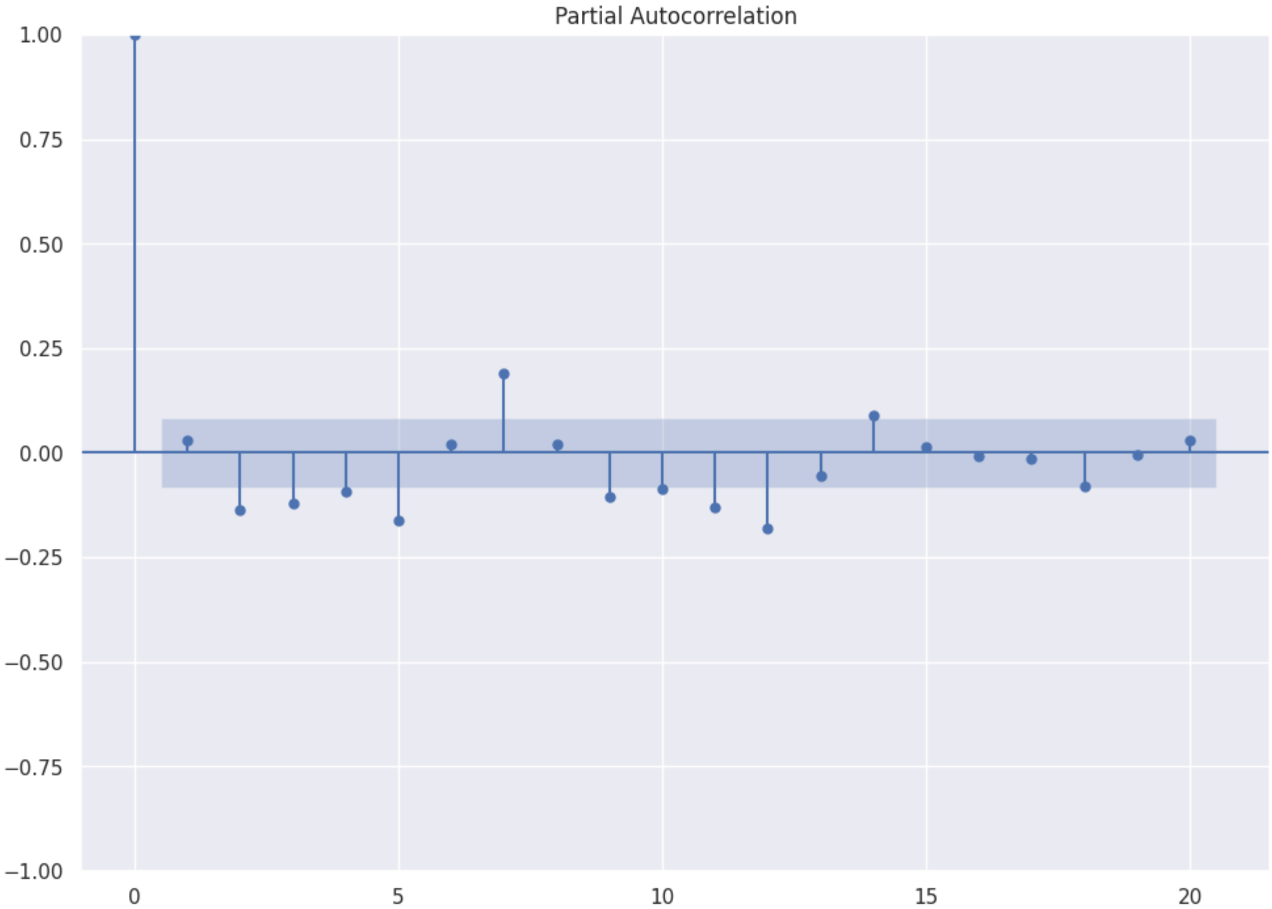
as for d we can see that at 1 diffencing the series becomes stationary so we choose d as 1

Plot the autocorreltaion and partial auto correlation functions

Plotting the graphs and getting the p,q,d values for arima

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
acf=plot_acf(ts_diff,lags=20)
pacf=plot_pacf(ts_diff,lags=20)
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20

https://people.duke.edu/~rnau/411arim3.htm

ARIMA MODEL

from the decomposition we can see that there is a weekly seasonality and still some spikes in the residual, that may be because of some external factors, which we can take into account by using them as our exogenous variable

```
# Exog_data is a exogenous feature
# Here is the link : - https://drive.google.com/file/d/1rD4OpoCtV45qbFUzo3vk7a837
```

```
ex_df = pd.read_csv('Exog_Campaign_eng')
ex_df.head()
```

	Exog
0	0
1	0
2	0
3	0
4	0

We get the exogenous data from this csv file for english pages

```
exog=ex_df['Exog'].to_numpy()
```

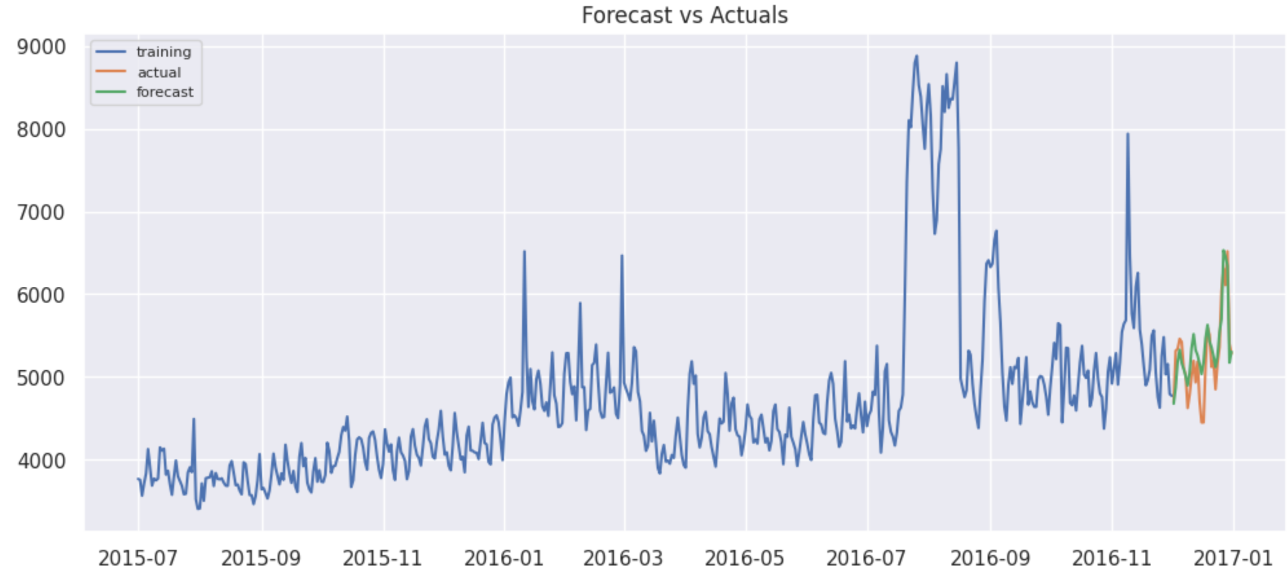
we will train a sarimax model for that and see if we get anyimprovements from using the two information.

the seasonal order and the values of PDQ are based upon various trials and comparision of the models

- we see a seasonality of 7 from the plots ie: weekly seasonality (from the plots we can see that afte some insignificant plots we have some significant values repeating at intervals of 7 ie: 7,14 ...)
- the non seasonal order we can keep the same

```
import statsmodels.api as sm
train=ts[:520]
test=ts[520:]
model=sm.tsa.statespace.SARIMAX(train,order=(4, 1, 3),seasonal_order=(1,1,1,7),e›
results=model.fit()

fc=results.forecast(30,dynamic=True,exog=pd.DataFrame(exog[520:]))

# Make as pandas series
fc_series = pd.Series(fc)
# Plot
train.index=train.index.astype('datetime64[ns]')
test.index=test.index.astype('datetime64[ns]')
plot.figure(figsize=(12,5), dpi=100)
plot.plot(train, label='training')
plot.plot(test, label='actual')
plot.plot(fc_series, label='forecast')

plot.title('Forecast vs Actuals')
plot.legend(loc='upper left', fontsize=8)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
  warnings.warn("Maximum Likelihood optimization failed to "
<matplotlib.legend.Legend at 0x7a773ed94d30>
```




```
mape = np.mean(np.abs(fc - test.values)/np.abs(test.values))
rmse = np.mean((fc - test.values)**2)**.5
print("mape:",mape)
print("rsme:",rmse)
```

```
mape: 0.0462866734689745
rsme: 291.1051339895362
```

The mean absolute percentage error and the root mean squared error is low

regression for a time series

```
ts_df=ts.to_frame()
ts_df.head()
```

	en
index	
2015-07-01	3767.328604
2015-07-02	3755.158765
2015-07-03	3565.225696
2015-07-04	3711.782932
2015-07-05	3833.433025

```
ts_df.reset_index(level=0, inplace=True)
ts_df['date']=pd.to_datetime(ts_df['index'])
ts_df.drop(['index'],axis=1,inplace=True)
ts_df.head()
```

	en	date
0	3767.328604	2015-07-01
1	3755.158765	2015-07-02
2	3565.225696	2015-07-03
3	3711.782932	2015-07-04
4	3833.433025	2015-07-05

```
ts_df['day_of_week']=ts_df['date'].dt.day_name()
ts_df.head()
```

	en	date	day_of_week
0	3767.328604	2015-07-01	Wednesday
1	3755.158765	2015-07-02	Thursday
2	3565.225696	2015-07-03	Friday
3	3711.782932	2015-07-04	Saturday
4	3833.433025	2015-07-05	Sunday

```
ts_df=pd.get_dummies(ts_df, columns = ['day_of_week'])
```

```
ts_df.head()
```

	en	date	day_of_week_Friday	day_of_week_Monday	day_of_week_Satur
0	3767.328604	2015-07-01	0	0	
1	3755.158765	2015-07-02	0	0	
2	3565.225696	2015-07-03	1	0	
3	3711.782932	2015-07-04	0	0	
4	3833.433025	2015-07-05	0	0	

```
ts_df['exog']=ex_df['Exog']
ts_df['rolling_mean']=ts_df['en'].rolling(7).mean()
```

```
ts_df=ts_df.dropna()
ts_df.head()
```

	en	date	day_of_week_Friday	day_of_week_Monday	day_of_week_Satu
6	3906.341724	2015-07-07	0	0	
7	3685.854621	2015-07-08	0	0	
8	3771.183714	2015-07-09	0	0	
9	3749.860313	2015-07-10	1	0	
10	3770.749355	2015-07-11	0	0	

```
X=ts_df[['day_of_week_Friday',  'day_of_week_Monday',   'day_of_week_Saturday', '
y=ts_df[['en']]

train_x = X[:-20]
test_x = X[-20:]

train_y = y[:-20]
test_y = y[-20:]
```

```
from sklearn.linear_model import LinearRegression

# Train and pred
model = LinearRegression()
model.fit(train_x, train_y)
y_pred = (model.predict(test_x))


mape = np.mean(np.abs(y_pred - test_y.values)/np.abs(test_y.values))
print("mape:",mape)
```

We can see here that aur mape is better than our arima model but worse than our sarimax model

using Facebook Prophet

```
!pip install prophet
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dis
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dis
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/di
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.
Requirement already satisfied: stanio~=0.3.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/d
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
```

```
ts_df['ds']=ts_df['date']
ts_df['y']=ts_df['en']
```

```
df2=ts_df[['date','en','exog']].copy()
df2.columns = ['ds', 'y', 'exog']
df2.head()
```

	ds	y	exog
6	2015-07-07	3906.341724	0
7	2015-07-08	3685.854621	0