```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
# from fbprophet import Prophet
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import mean_absolute_percentage_error
```

```
# Problem Statement:

# As a member of the Data Science team at AdEase, your task is to analyze the per-page view report spanning 550 days for 145,000 Wikipedia pages.
# The goal is to forecast the number of views for these pages to predict and optimize ad placement for clients


# Objective:

# Data Analysis: Analyze the historical data of Wikipedia page views to identify trends, patterns, and seasonality.

# Forecasting: Develop predictive models to forecast the future number of views for each Wikipedia page. This forecasting should encompass variations
#  across different languages and regions.

# Optimization: Utilize the forecasted data to optimize ad placement strategies for clients based on language, region, and expected viewership.

# Performance Evaluation: Evaluate the accuracy and reliability of the forecasting models using appropriate metrics such as Mean Absolute Error (MAE),
# Root Mean Squared Error (RMSE), etc.

# Recommendations: Provide actionable insights and recommendations to clients regarding ad placement strategies, considering the forecasted viewership
# on different Wikipedia pages.
```

```
path = "/content/drive/MyDrive/train_1.csv"
```

```
train_data = pd.read_csv(path)
```

```
train_data = pd.DataFrame(train_data)
```

Double-click (or enter) to edit

```
path1 = "/Exog_Campaign_eng (2)"
```

```
exog_campaign_eng = pd.read_csv(path1)
```

```
train_data.head()
```

	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	...	2016-12-22	2016-12-23	201 12-
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	1
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	2
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	2
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	2

5 rows × 551 columns

```
print("Rows - ",train_data.shape[0])
print("Columns - ",train_data.shape[1])
```

```
Rows -  145063
Columns -  551
```

```
train_data.info()
#Among 551 columns 550 columns are float data type and 1 column is object data type
```

```
<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
```

```
train_data.isna().sum()
# There are many null values in the data set
```

```
Page              0
2015-07-01    20740
2015-07-02    20816
2015-07-03    20544
```

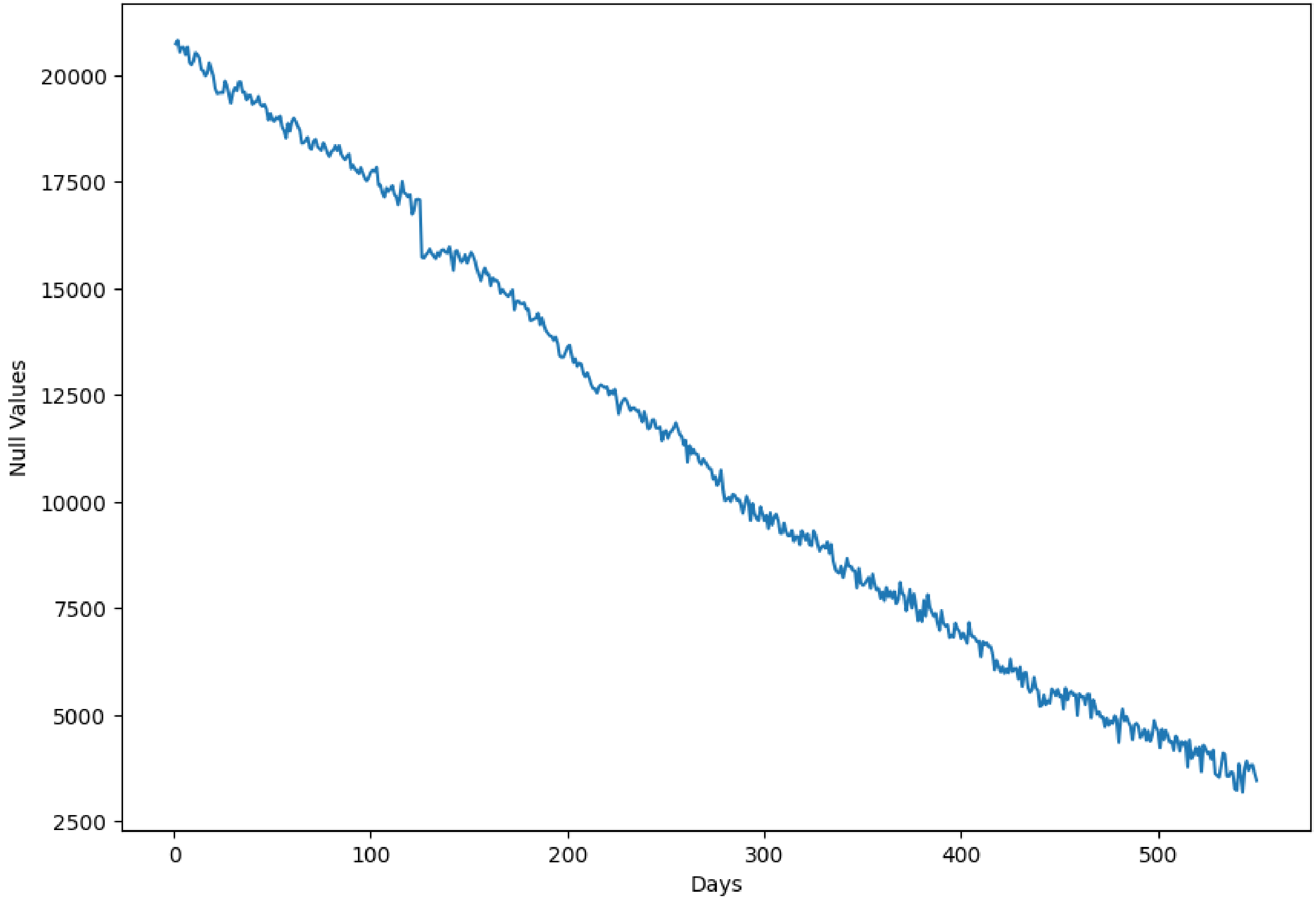
```
2015-07-04    20654
              ...
2016-12-27     3701
2016-12-28     3822
2016-12-29     3826
2016-12-30     3635
2016-12-31     3465
Length: 551, dtype: int64
```

```
# Lets plot a graph to check the null values over time
```

```
days = [n for n in range(1,len(train_data.columns))]
```

```
plt.figure(figsize=(10,7))
plt.xlabel("Days")
plt.ylabel("Null Values")
plt.plot(days, train_data.isnull().sum()[1:])
```

```
[<matplotlib.lines.Line2D at 0x7c4cae941de0>]
```




```
# As we can see number of null values are decreasing over time
# According to me why null values are decreasing because Ads were added later in the timeline
```

```
train_data.shape
```

```
(145063, 551)
```

```
train_data = train_data.dropna(how="all")
# Here we are dropping the rows where all the values are null
```

```
train_data.shape
```

```
(145063, 551)
```

```
train_data = train_data.dropna(thresh = 300)
```

```
train_data.shape
# Here we are dropping the rows that have null values more than 300 days
```

```
(133617, 551)
```

```
train_data = train_data.fillna(0)
```

```
train_data.head()
```

	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	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-3
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	15.0	26.0	14.0	20.0	22.0	19.0	18
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	28.0	15.0	9.0	30.0	52.0	45.0	26
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	1.0	7.0	4.0	4.0	6.0	3.0	4
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	26.0	27.0	16.0	11.0	17.0	19.0	10
5	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0	17.0	24.0	...	16.0	27.0	8.0	17.0	32.0	19.0	23.0	17.0	17

5 rows × 551 columns

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

```
li = list(train_data.Page.apply(lambda x: split_page(str(x))))
```

```
df = pd.DataFrame(li)
df.columns = ["Title", "Language", "Access_type", "Access_origin"]
```

```
df.head()
```

	Title	Language	Access_type	Access_origin
0	2NE1	zh	all-access	spider
1	2PM	zh	all-access	spider
2	3C	zh	all-access	spider
3	4minute	zh	all-access	spider
4	5566	zh	all-access	spider

```
df['Language'].unique()
# Here we can see the different languanges
```

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

```
df = pd.concat([train_data,df], axis = 1)
```

```
df.head()
```

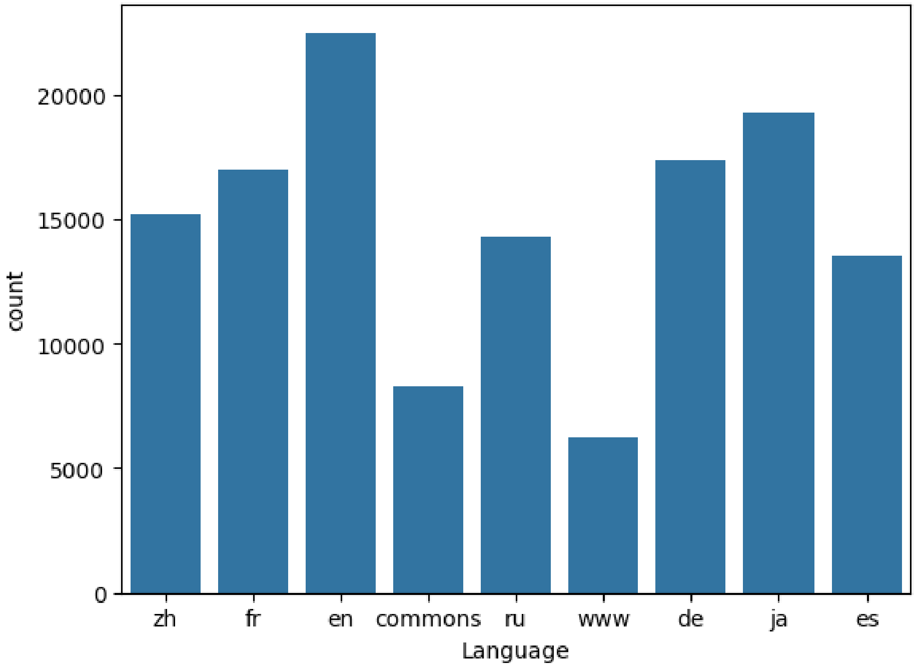
	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Title	Language
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	14.0	20.0	22.0	19.0	18.0	20.0	2NE1	zh
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	9.0	30.0	52.0	45.0	26.0	20.0	2PM	zh
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	4.0	4.0	6.0	3.0	4.0	17.0	3C	zh
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	16.0	11.0	17.0	19.0	10.0	11.0	4minute	zh
5	5566_zh.wikipedia.org_all-access_spider	12.0	7.0	4.0	5.0	20.0	8.0	5.0	17.0	24.0	...	32.0	19.0	23.0	17.0	17.0	50.0	A'N'D	zh

5 rows × 555 columns

```
# Here we can see those four columns are added
```

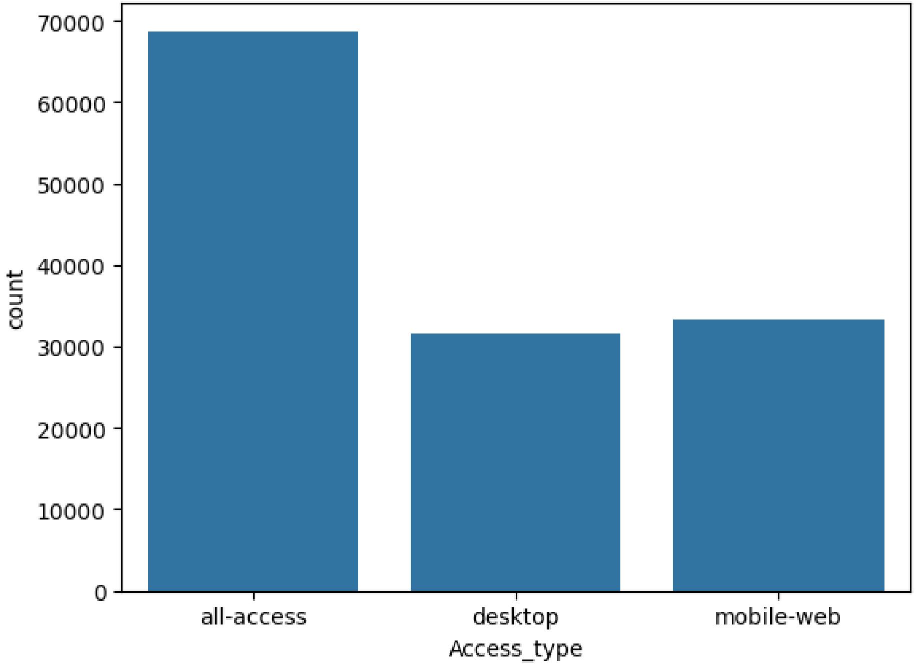
```
# Lets check the distribution of data points in Language
import seaborn as sns
sns.countplot(data=df, x='Language')
```

```
<Axes: xlabel='Language', ylabel='count'>
```




```
# Lets check the distribution of data points in Access_type
sns.countplot(data = df, x = "Access_type")
```

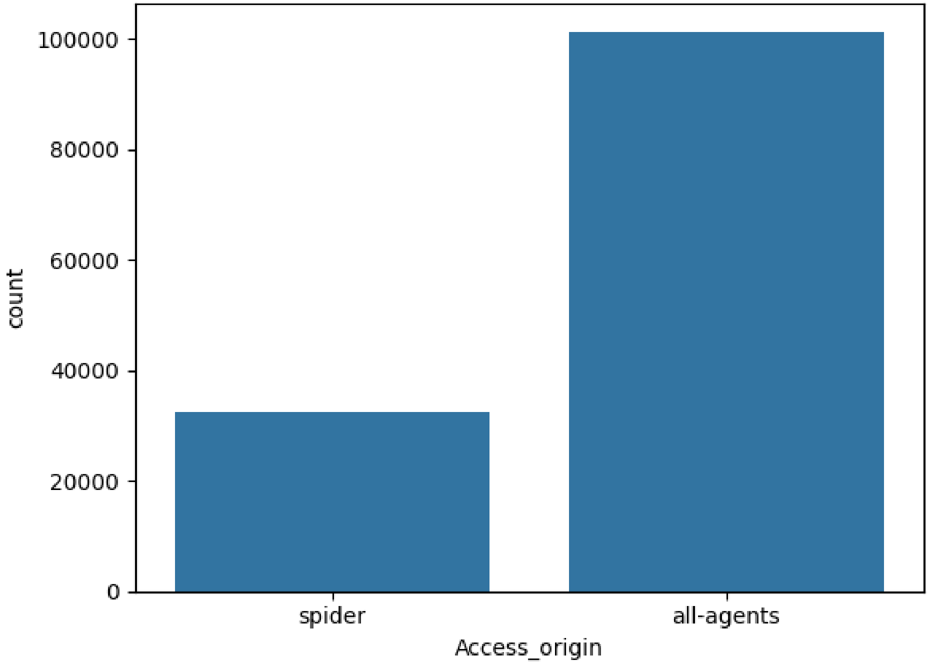
```
<Axes: xlabel='Access_type', ylabel='count'>
```




```
# Here we can see that usage from desktop and mobile-web is almost the same
```

```
# Lets check the distribution of data points in Access_origin
sns.countplot(data = df, x = "Access_origin")
```

```
<Axes: xlabel='Access_origin', ylabel='count'>
```




```
# Organic views significantly outnumber views generated by spiders or bots
```

```
# Now we will see the views for different languages
```

```
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	...	2016-12-25	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Title	Access_type	A
Language																					
commons	7672	7672	7672	7672	7672	7672	7672	7672	7672	7672	...	7672	7672	7672	7672	7672	7672	7672	8266	8266	
de	15946	15946	15946	15946	15946	15946	15946	15946	15946	15946	...	15946	15946	15946	15946	15946	15946	15946	17362	17362	
en	20758	20758	20758	20758	20758	20758	20758	20758	20758	20758	...	20758	20758	20758	20758	20758	20758	20758	22486	22486	
es	12268	12268	12268	12268	12268	12268	12268	12268	12268	12268	...	12268	12268	12268	12268	12268	12268	12268	13551	13551	
fr	15418	15418	15418	15418	15418	15418	15418	15418	15418	15418	...	15418	15418	15418	15418	15418	15418	15418	16948	16948	
ja	17132	17132	17132	17132	17132	17132	17132	17132	17132	17132	...	17132	17132	17132	17132	17132	17132	17132	19295	19295	
ru	12955	12955	12955	12955	12955	12955	12955	12955	12955	12955	...	12955	12955	12955	12955	12955	12955	12955	14270	14270	
www	5743	5743	5743	5743	5743	5743	5743	5743	5743	5743	...	5743	5743	5743	5743	5743	5743	5743	6228	6228	
zh	14845	14845	14845	14845	14845	14845	14845	14845	14845	14845	...	14845	14845	14845	14845	14845	14845	14845	15211	15211	

9 rows × 554 columns

```
df_language = df.groupby("Language").mean(numeric_only = True).transpose()
df_language
```

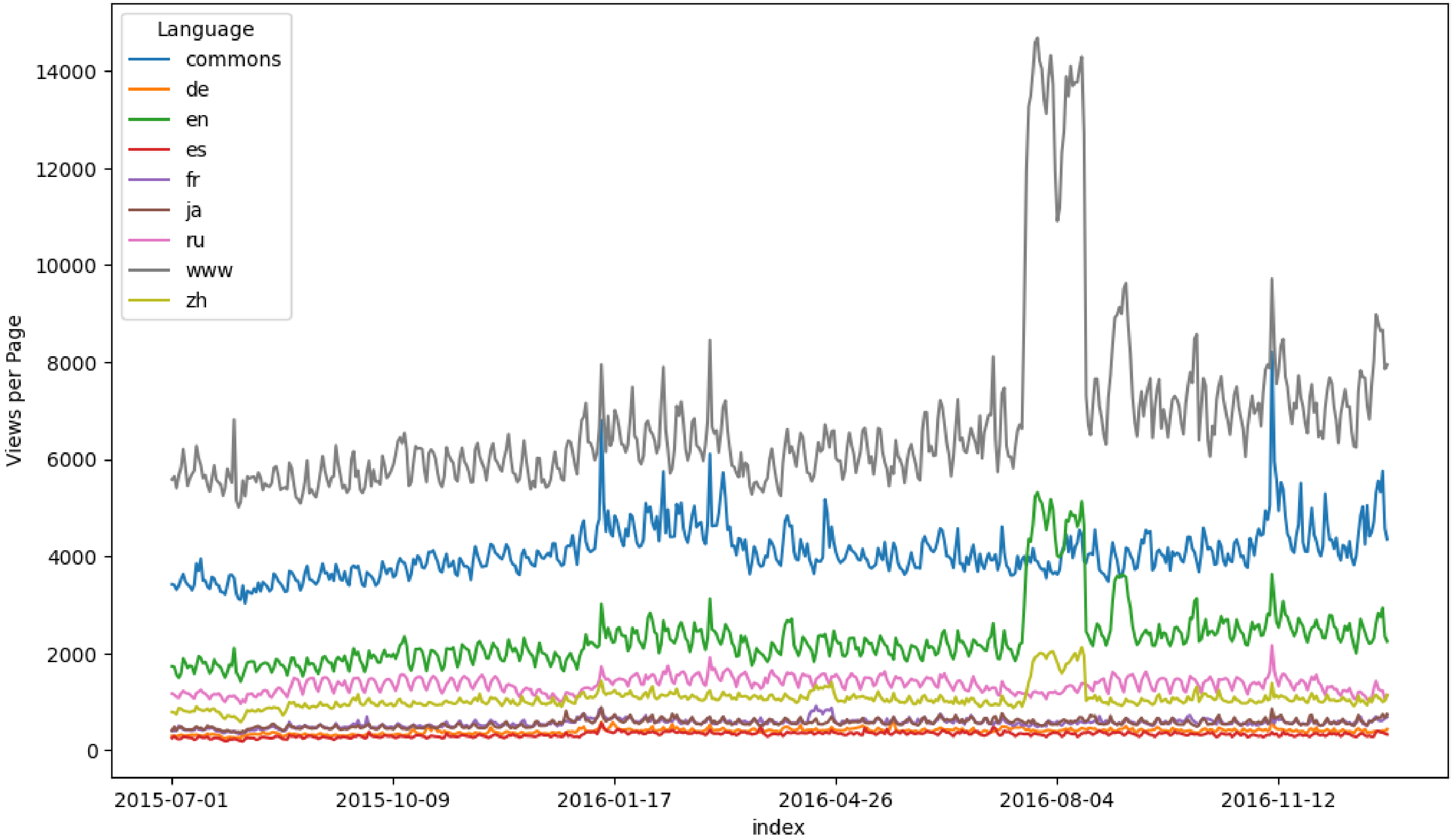
Language	commons	de	en	es	fr	ja	ru	www	zh
2015-07-01	3418.187826	295.493666	1727.580740	251.141995	406.026917	403.823138	1166.175839	5578.340763	784.954665
2015-07-02	3401.103363	290.430641	1713.580595	291.923459	396.203463	482.443264	1140.778155	5641.656277	772.002156
2015-07-03	3307.163060	286.196287	1555.654639	242.822383	400.320016	413.968772	1096.175531	5400.232283	741.925160
2015-07-04	3390.042492	303.238743	1493.560266	229.509048	501.218187	461.208499	1080.473331	5664.424865	864.563220
2015-07-05	3522.260036	317.944688	1581.818817	237.705249	460.476067	490.145634	1170.957545	5814.667247	853.669653
...	...	...	...	...	...	...	...	...	...
2016-12-27	5547.263686	386.211401	2825.008960	395.420199	619.959398	673.732431	1266.212273	8786.032561	1078.681037
2016-12-28	5320.972106	384.710774	2740.355477	370.171177	602.927098	668.514826	1228.020224	8632.590110	1056.301650
2016-12-29	5753.922706	404.172959	2934.513248	363.423133	614.458295	739.548214	1226.597916	8661.561553	994.841361
2016-12-30	4566.859359	389.619528	2350.004721	344.576215	654.068232	670.735641	1057.697260	7855.572697	1017.394005
2016-12-31	4348.738530	431.513295	2243.430581	325.006684	683.744844	738.753502	1132.103667	7942.211910	1134.651735

550 rows × 9 columns

Next steps: Generate code with `df_language` View recommended plots

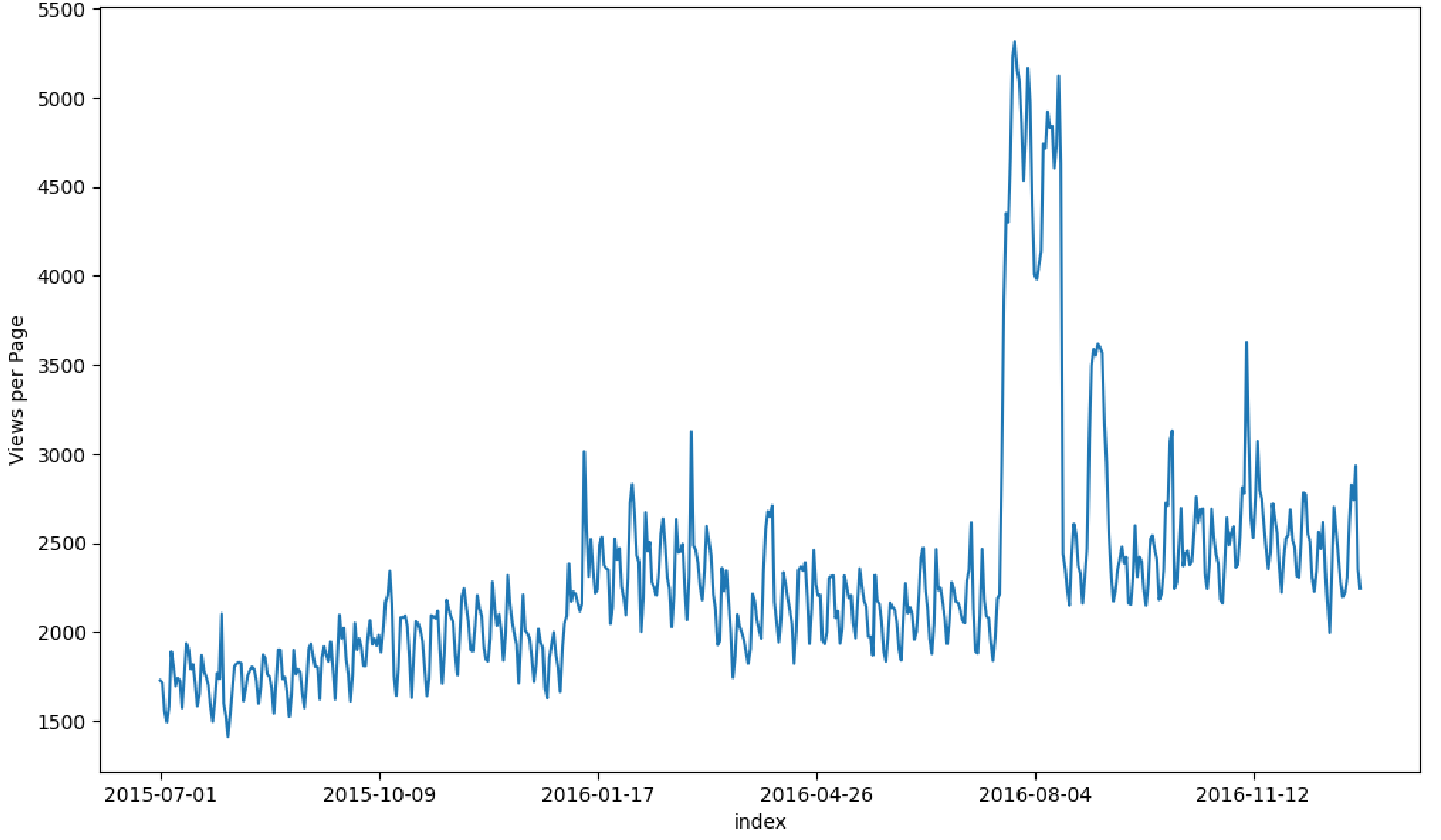
```
df_language.reset_index(inplace = True)
df_language.set_index('index', inplace = True)


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

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




```
# English articles generally receive more views than articles in other languages. However, there are occasional spikes in viewership
# for articles in different languages at different times.


# Now we will plot for English Language because we are going to use this for further predictions


df_language["en"].plot(figsize=(12,7))
plt.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 TS is non stationary
```

```
def df_test(x):
  result = adfuller(x)
  print("ADF Statisic: %f" %result[0])
  print("p-value: %f" %result[1])
```

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

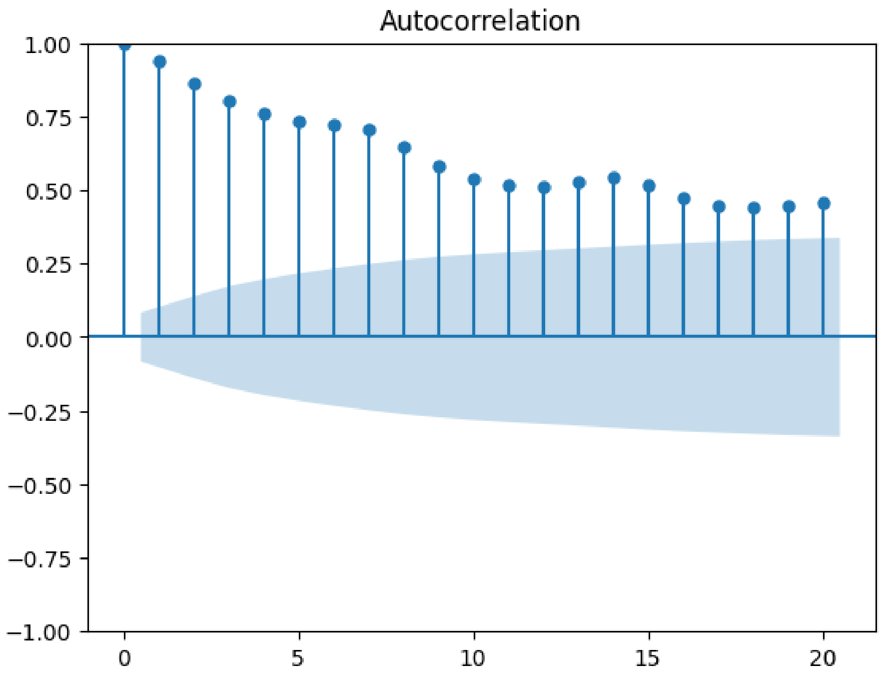
```
ADF Statisic: -2.930375
p-value: 0.041928
```

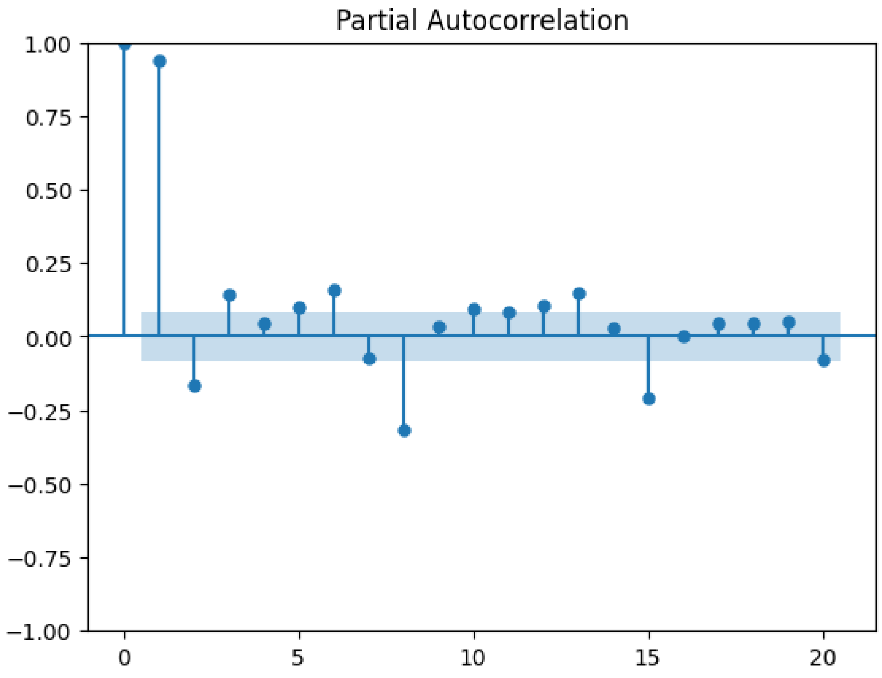
```
# The ADF statistic being negative indicates evidence against the presence of a unit root.
# The p-value being less than 0.05 suggests rejecting the null hypothesis of the presence of a unit root, indicating that the data is stationary.
```

```
# Here we will not be using Decomposition of series, Differencing the series because the data is stationary
```

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

```
acf=plot_acf(ts,lags=20)
pacf=plot_pacf(ts,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

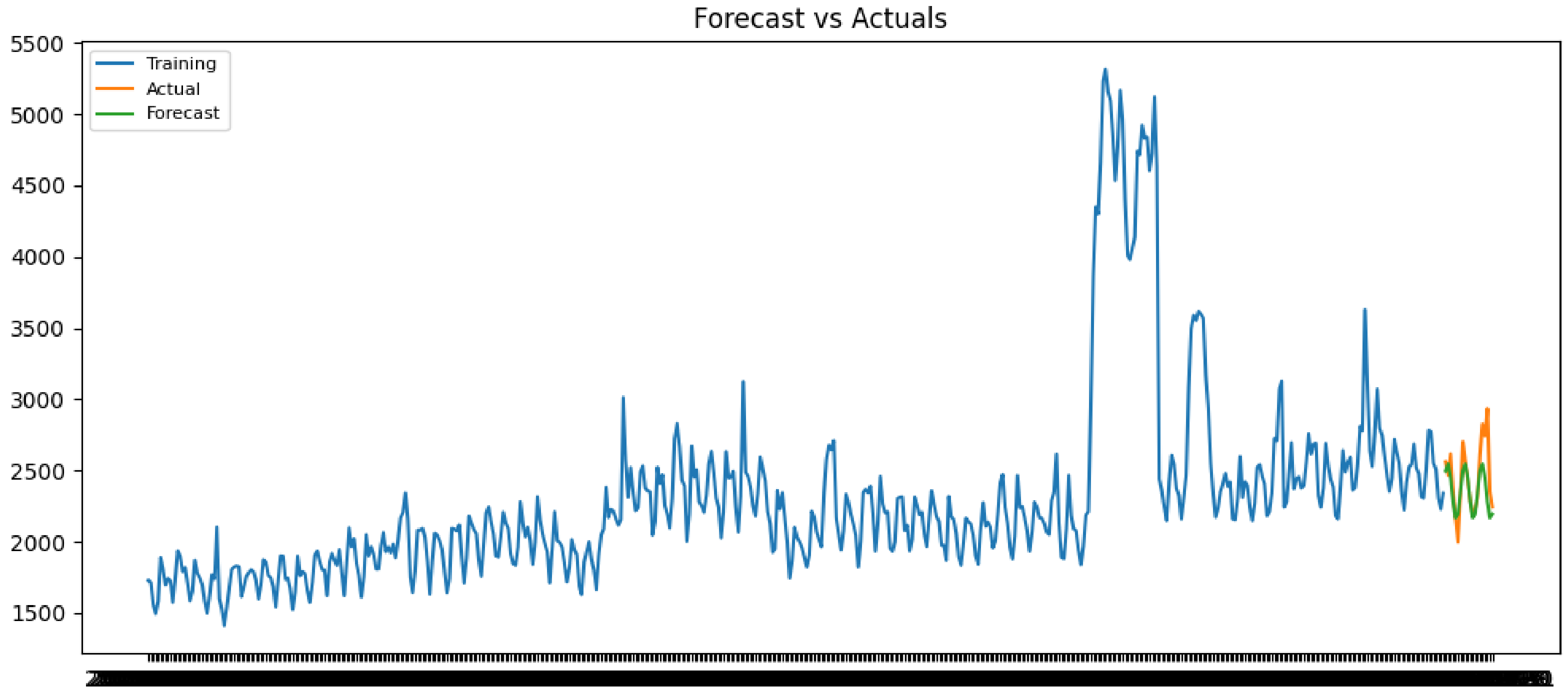
```
model = ARIMA(ts, order=(4,1,3))
model_fit = model.fit()
model_fit
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters fou
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using ze
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
  warnings.warn("Maximum Likelihood optimization failed to "
<statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x7c4c8103e740>
```

```
train = ts[:-20]
test = ts[-20:]
```

```
model = ARIMA(train, order=(4, 1, 3))
fitted = model.fit()

# Forecast
fc = fitted.forecast(steps=20)

# Make as pandas series
fc_series = pd.Series(fc, index=test.index)

# Plot
plt.figure(figsize=(12, 5), dpi=100)
plt.plot(train, label='Training')
plt.plot(test, label='Actual')
plt.plot(fc_series, label='Forecast')

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

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters fou
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using ze
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
  warnings.warn("Maximum Likelihood optimization failed to "
```




```
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: 0.04831729382572765
rsme: 195.58429193838197
```

```
# MAPE and RMSE are measures of forecasting accuracy, with lower values indicating better performance.
```

```
ex_df=exog_campaign_eng
ex_df.head()
```

	Exog
0	0
1	0
2	0
3	0
4	0

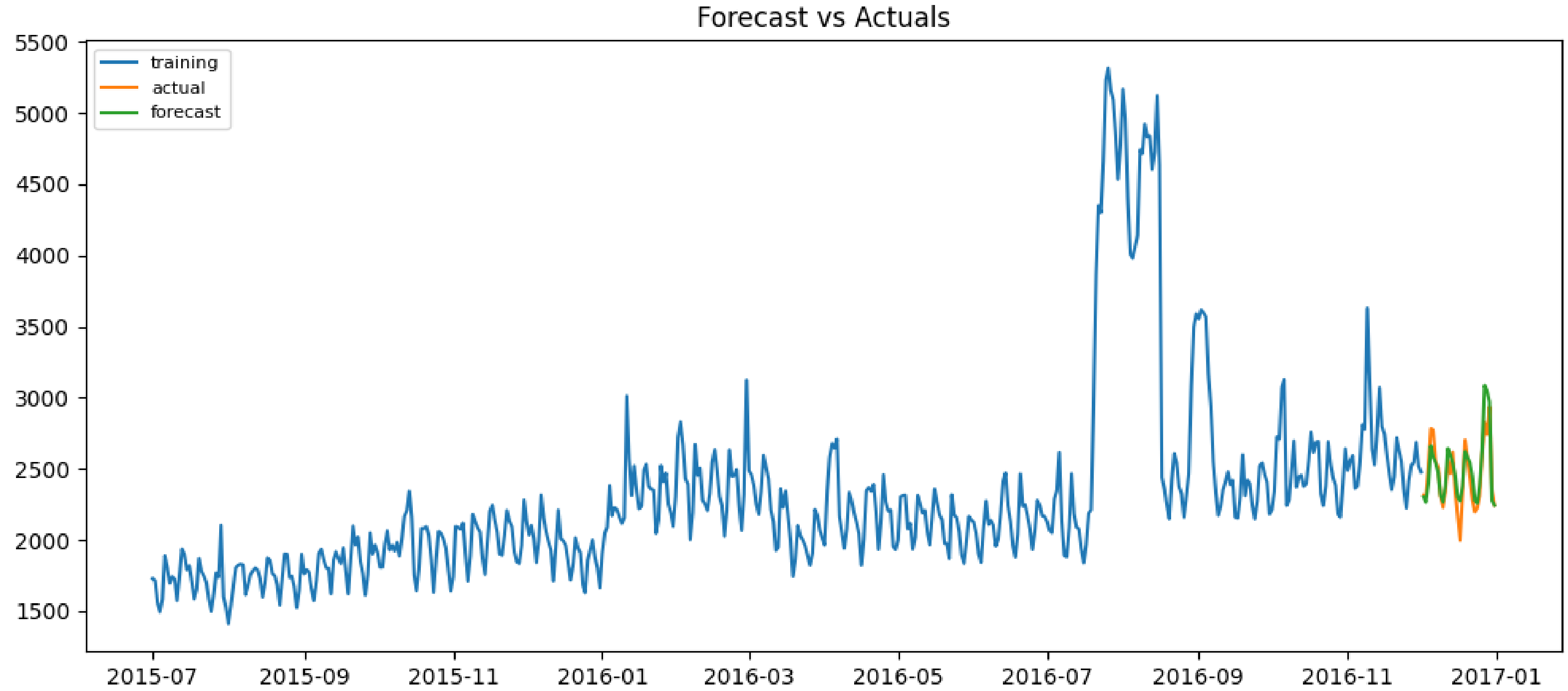
```
# We get the exogenous data from this csv file for english pages
```

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

```
# we will train a sarimax model for that and see if we get any improvements from using the two information.
```

```
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),exog=exog[:520])
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]')
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')

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

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequ
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters fou
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using ze
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
  warnings.warn("Maximum Likelihood optimization failed to "
<matplotlib.legend.Legend at 0x7c4c4786e2c0>
```




```
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: 0.036867162754078815
rsme: 120.70762191870872
```

```
# 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()
```

index	en
2015-07-01	1727.580740
2015-07-02	1713.580595
2015-07-03	1555.654639
2015-07-04	1493.560266
2015-07-05	1581.818817

```
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	1727.580740	2015-07-01
1	1713.580595	2015-07-02
2	1555.654639	2015-07-03
3	1493.560266	2015-07-04
4	1581.818817	2015-07-05

Next steps: Generate code with `ts_df` View recommended plots

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

	en	date	day_of_week
0	1727.580740	2015-07-01	Wednesday
1	1713.580595	2015-07-02	Thursday
2	1555.654639	2015-07-03	Friday
3	1493.560266	2015-07-04	Saturday
4	1581.818817	2015-07-05	Sunday

Next steps: Generate code with ts_df View recommended plots

```
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_Saturday	day_of_week_Sunday	day_of_week_Thursday	day_of_week_Tuesday	day_o
0	1727.580740	2015-07-01	0	0	0	0	0	0	
1	1713.580595	2015-07-02	0	0	0	0	1	0	
2	1555.654639	2015-07-03	1	0	0	0	0	0	
3	1493.560266	2015-07-04	0	0	1	0	0	0	
4	1581.818817	2015-07-05	0	0	0	1	0	0	

Next steps: Generate code with ts_df View recommended plots

```
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_Saturday	day_of_week_Sunday	day_of_week_Thursday	day_of_week_Tuesday	day_
6	1808.734657	2015-07-07	0	0	0	0	0	1	
7	1695.638356	2015-07-08	0	0	0	0	0	0	
8	1740.442962	2015-07-09	0	0	0	0	1	0	
9	1724.306870	2015-07-10	1	0	0	0	0	0	
10	1573.523268	2015-07-11	0	0	1	0	0	0	

Next steps: Generate code with `ts_df` View recommended plots

```
X=ts_df[['day_of_week_Friday',  'day_of_week_Monday',   'day_of_week_Saturday', 'day_of_week_Sunday',   'day_of_week_Thursday', 'day_of_week_Tuesday',
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)
```

```
mape: 0.03897198323630918
```

```
# using Facebook Prophet
```