About AdEase 🌟

AdEase is an **advertising and marketing company** that empowers businesses to achieve **maximum clicks at minimum cost**. It serves as a robust **ad infrastructure** designed to help brands **promote themselves easily, effectively, and economically**.

At the heart of AdEase is an advanced **AI-driven advertising system**, powered by three core modules:

- **Design**: Craft customized, high-performing ad creatives.
- **Dispense**: Efficiently distribute ads across relevant platforms.
- **Decipher**: Analyze ad performance to optimize campaigns and maximize ROI.

This **end-to-end 3-step process** makes AdEase the **go-to digital advertising solution** for businesses aiming for optimal growth and visibility.

Project Objective & Problem Statement 📊

The goal of this project is to **analyze and forecast daily page views** for various **Wikipedia pages** over **550 days** to predict future trends and **optimize ad placement strategies** for AdEase's clients. With data for **145,000 Wikipedia pages**, your task is to:

1. **Understand the per-page view trends** for different Wikipedia pages.
2. **Forecast future page views** to predict traffic spikes.
3. **Provide region-specific insights** on ad performance for pages in different languages, enabling clients to **target the right audience more effectively**.

Key Deliverables 🚀

➔ **Exploratory Data Analysis**: Discover trends and patterns in page views.
➔ **Time Series Forecasting**: Build a model to predict future page views.
➔ **Region & Language-Based Insights**: Tailor recommendations for clients in different regions and languages.
➔ **Optimization Strategy**: Suggest ad placement improvements based on forecasted traffic.

Business Impact 💡

This project will help AdEase's clients:

- **Maximize ad visibility** by predicting high-traffic periods.
- **Optimize marketing budgets** through accurate forecasting.
- **Increase engagement and conversion** by targeting the right pages at the right time.

📃 Features of the Dataset

The dataset comprises two CSV files containing information about web traffic and external campaign events that influence Wikipedia page views. Here's a breakdown of the dataset:

1. train_1.csv

This file contains web traffic data for various Wikipedia pages. Each row represents a specific page, and each column corresponds to a date. The values indicate the number of visits on that particular date.

- **Page Name**: Contains information about the page in the format: `SPECIFIC_NAME_LANGUAGE.wikipedia.org_ACCESS_TYPE_ACCESS_ORIGIN`. This provides details about the page name, language, access type, and request origin.
  - **SPECIFIC_NAME**: The unique title of the Wikipedia page.
  - **LANGUAGE**: Language version of the page (e.g., `en` for English, `fr` for French).
  - **ACCESS_TYPE**: The type of access—`desktop`, `mobile-web`, or `mobile-app`.
  - **ACCESS_ORIGIN**: Specifies whether the request is from `all-agents` (human visitors) or `spiders` (bots/crawlers).
- **Date Columns**: Each column from `2015-07-01` to `2018-09-10` represents the number of visits on the respective date.

2. Exog_Campaign_eng.csv

This file contains information about significant events or marketing campaigns that may have influenced page views for English-language Wikipedia pages.

- **Date**: Indicates the specific date of an event or campaign.
- **Campaign Indicator**: A binary variable where `1` denotes the presence of a campaign on that date, and `0` indicates no campaign. This serves as an exogenous variable for forecasting models.

🔮 The Road Ahead:

☑ 📊 **Exploratory Data Analysis**: Analyzed web traffic trends, seasonality, and campaign impact on page views.

☑ ⌛ **Time Series** Built predictive models using **ARIMA**, **SARIMAX**, and **Prophet** to forecast future traffic.

☑ 🤝 **Insights & Recommendations**: Identified key patterns to optimize marketing strategies and improve campaign planning.

Basic Importation & Downloading Dataset

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import plotly.express as px

import seaborn as sns

import statsmodels.api as sm

from statsmodels.formula.api import ols

from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings

import re

from locale import normalize

warnings.filterwarnings('ignore')

%matplotlib inline

!gdown
'https://drive.google.com/uc?export=download&id=1qQkymAitU6l2pSe702rDUhQpo
P8MUZXl' -O train_1.csv

!gdown
'https://drive.google.com/uc?export=download&id=19qvuu7E8yD63o4WkOdy_lLFSr
ZlZPpuE' -O Exog_Campaign_eng.csv
```

```
Downloading...
From (original): https://drive.google.com/uc?export=download&id=1qQkymAitU6l2pSe702rDUhQpoP8MUZXl
From (redirected): https://drive.google.com/uc?export=download&id=1qQkymAitU6l2pSe702rDUhQpoP8MUZXl&confirm=t&uuid=6874f107-ad6f-41c5-8a48-5c065839afa9
To: /content/train_1.csv
100% 278M/278M [00:03<00:00, 81.6MB/s]
Downloading...
From: https://drive.google.com/uc?export=download&id=19qvuu7E8yD63o4WkOdy_lLFSrZlZPpuE
To: /content/Exog_Campaign_eng.csv
100% 1.10k/1.10k [00:00<00:00, 4.78MB/s]
```

```python
exog = pd.read_csv('Exog_Campaign_eng.csv')

print("train_1 Head:")

print(exog.head())

df = pd.read_csv('train_1.csv')

print("\Exogineous variable Head:")

print(df.head())
```

```
train_1 Head:
   Exog
0     0
1     0
2     0
3     0
4     0
\Exogineous variable Head:
                                              Page  2015-07-01  2015-07-02  \
0            2NE1_zh.wikipedia.org_all-access_spider        18.0        11.0
1             2PM_zh.wikipedia.org_all-access_spider        11.0        14.0
2              3C_zh.wikipedia.org_all-access_spider         1.0         0.0
3         4minute_zh.wikipedia.org_all-access_spider        35.0        13.0
4  52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...         NaN         NaN

   2015-07-03  2015-07-04  2015-07-05  2015-07-06  2015-07-07  2015-07-08  \
0         5.0        13.0        14.0         9.0         9.0        22.0
1        15.0        18.0        11.0        13.0        22.0        11.0
2         1.0         1.0         0.0         4.0         0.0         3.0
3        10.0        94.0         4.0        26.0        14.0         9.0
4         NaN         NaN         NaN         NaN         NaN         NaN

   2015-07-09  ...  2016-12-22  2016-12-23  2016-12-24  2016-12-25  \
0        26.0  ...        32.0        63.0        15.0        26.0
1        10.0  ...        17.0        42.0        28.0        15.0
2         4.0  ...         3.0         1.0         1.0         7.0
3        11.0  ...        32.0        10.0        26.0        27.0
4         NaN  ...        48.0         9.0        25.0        13.0

   2016-12-26  2016-12-27  2016-12-28  2016-12-29  2016-12-30  2016-12-31
0        14.0        20.0        22.0        19.0        18.0        20.0
1         9.0        30.0        52.0        45.0        26.0        20.0
2         4.0         4.0         6.0         3.0         4.0        17.0
3        16.0        11.0        17.0        19.0        10.0        11.0
4         3.0        11.0        27.0        13.0        36.0        10.0

[5 rows x 551 columns]
```

📊 Exploratory Data Analysis:

```
exog.sample(10)
```

	Exog
158	0
404	1
488	0
303	0
384	0
464	0
397	1
190	0
343	0
86	0

```
exog.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Exog    550 non-null    int64
dtypes: int64(1)
memory usage: 4.4 KB
```

```
exog.isnull().sum()

exog.shape
```

	0
Exog	0

dtype: int64

```
(550, 1)
```

```
df.sample(5)

df_new.isna().sum()
```

	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	2016-12-30	2016-12-31
68466	Post_Tower_de.wikipedia.org_desktop_all-agents	49.0	79.0	47.0	30.0	33.0	40.0	58.0	58.0	49.0	...	34.0	26.0	28.0	19.0	23.0	34.0	47.0	43.0	39.0	32.0
113021	Saydy,_Verkhoyansky_District,_Sakha_Republic_e...	NaN	1.0	NaN	1.0	3.0	3.0	NaN	2.0	NaN	...	1.0	2.0	7.0	2.0	4.0	4.0	NaN	1.0	1.0	3.0
96008	Prodigiosa:_Las_aventuras_de_Ladybug_es.wikipe...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1649.0	1946.0	1622.0	1553.0	2048.0	2011.0	2185.0	2033.0	2039.0	1802.0
21093	Special:ExtensionDistributor/Math_www.mediawik...	7.0	18.0	20.0	7.0	4.0	7.0	21.0	23.0	14.0	...	22.0	17.0	5.0	13.0	15.0	6.0	10.0	14.0	12.0	12.0
27550	Liste_des_présidents_des_États-Unis_fr.wikiped...	1475.0	2157.0	1294.0	1500.0	1606.0	1396.0	2129.0	1501.0	1363.0	...	2759.0	2824.0	2927.0	2962.0	3170.0	3072.0	3640.0	3375.0	3481.0	3111.0

5 rows × 551 columns

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

Insights 🔍

The dataset contains **145,063 entries and 551 columns**, with **time-series data** from **2016-12-31** onward. Null values in the dataset were replaced with **0** to maintain consistency.

An additional feature, **Exog**, was imported for external factors analysis, with **550 entries** and values ranging between **0 and 1**, representing external influences.

Splitting and Extracting Components from the 'Page' Column.

```
# Function to split the 'Page' column into multiple parts
def split_page(page):
    w = re.split('_|\.', page)  # Split by underscores and periods
    return ' '.join(w[:-5]), w[-5], w[-2], w[-1]

li = list(df['Page'].apply(lambda x: split_page(str(x))))
df_details = pd.DataFrame(li, columns=['Title', 'Language', 'Access_type',
'Access_origin'])
df_new = pd.concat([df.reset_index(), df_details], axis=1)
df_new.sample(5)
```

	index	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Title	Language	Access_type	Access_origin
116353	116353	Ben_Hur_(1959)_de.wikipedia.org_mobile-web_all...	81.0	71.0	84.0	124.0	128.0	103.0	83.0	121.0	...	1181.0	487.0	421.0	406.0	398.0	399.0	Ben Hur (1959)	de	mobile-web	all-agents
321	321	香港熱帶氣旋警告信號_zh.wikipedia.org_all-access_spider	0.0	2.0	1.0	6.0	2.0	3.0	8.0	11.0	...	7.0	10.0	10.0	9.0	12.0	7.0	香港熱帶氣旋警告信號	zh	all-access	spider
29794	29794	全秀珍_zh.wikipedia.org_all-access_all-agents	56.0	42.0	49.0	50.0	60.0	62.0	67.0	50.0	...	90.0	73.0	90.0	66.0	53.0	66.0	全秀珍	zh	all-access	all-agents
18425	18425	Бетховен,_Людвиг_ван_ru.wikipedia.org_mobile-w...	431.0	397.0	379.0	475.0	382.0	297.0	314.0	346.0	...	834.0	665.0	606.0	480.0	394.0	343.0	Бетховен, Людвиг ван	ru	mobile-web	all-agents
48382	48382	Thrombose_de.wikipedia.org_all-access_spider	23.0	7.0	6.0	2.0	6.0	4.0	11.0	8.0	...	11.0	8.0	12.0	10.0	8.0	15.0	Thrombose	de	all-access	spider

5 rows × 556 columns

```
df_new.shape
```

```
(145063, 556)
```

Mapping Language Codes to Full Language Names:

```
df_new['Language'].unique()

lang_map = {

    'zh': 'Chinese',

    'fr': 'French',

    'en': 'English',

    'commons': 'Commons',
```

```
    'ru': 'Russian',

    'www': 'Global',

    'de': 'German',

    'ja': 'Japanese',

    'es': 'Spanish'

}

df_new['Language'] = df_new['Language'].map(lang_map).fillna('Unknown')

df_new.head()
```

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

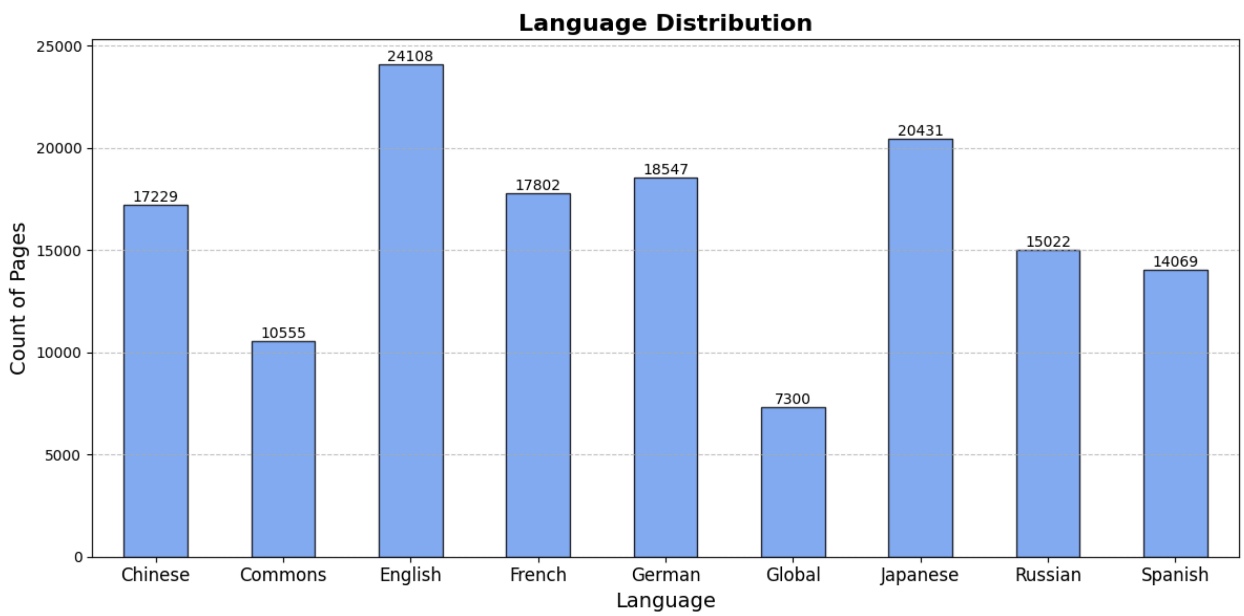
	index	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Title	Language	Access_type	Access_origin
0	0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	...	14.0	20.0	22.0	19.0	18.0	20.0	2NE1	Chinese	all-access	spider
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	...	9.0	30.0	52.0	45.0	26.0	20.0	2PM	Chinese	all-access	spider
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	4.0	6.0	3.0	4.0	17.0	3C	Chinese	all-access	spider
3	3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	...	16.0	11.0	17.0	19.0	10.0	11.0	4minute	Chinese	all-access	spider
4	4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	3.0	11.0	27.0	13.0	36.0	10.0	52 Hz I Love You	Chinese	all-access	spider

5 rows × 556 columns

Univariant & Bi Variant Analysis:

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

ax = df_new.groupby('Language')['Page'].count().plot(

    kind='bar', color='cornflowerblue', edgecolor='black', alpha=0.8

)

plt.title('Language Distribution', fontsize=16, fontweight='bold')

plt.xlabel('Language', fontsize=14)

plt.ylabel('Count of Pages', fontsize=14)

plt.xticks(rotation=0, fontsize=12)
```

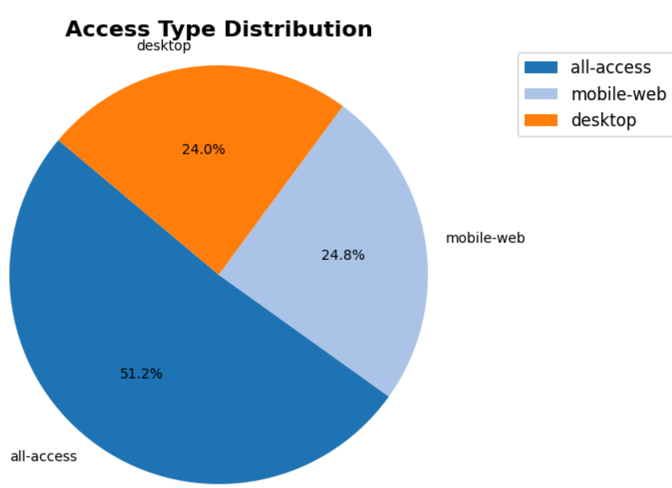
```
plt.grid(axis='y', linestyle='--', alpha=0.7)

for p in ax.patches:

    ax.annotate(f'{int(p.get_height())}',

                (p.get_x() + p.get_width() / 2, p.get_height()),

                ha='center', va='bottom', fontsize=10)

plt.tight_layout()

plt.show()
```



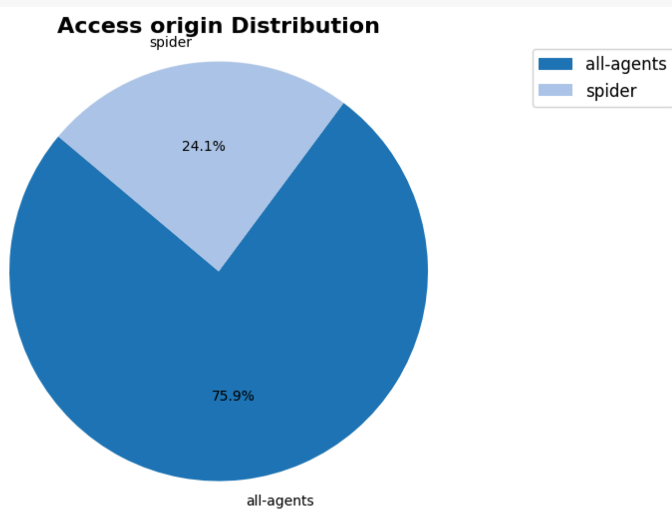
🔍Insights on Language Distribution

- The highest count per page is for **English** with **24,180** pages, followed by **Japanese** (**20,431**) and **German** (**18,547**).
- The **last three languages** with the lowest counts are **Global (7,300)**, **Commons (10,555)**, and **Spanish (14,069)**.

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

access_counts = df_new['Access_type'].value_counts()

plt.pie(

    access_counts,

    labels=access_counts.index,

    autopct='%1.1f%%',

    startangle=140,

    colors=plt.cm.tab20.colors

)

plt.title('Access Type Distribution', fontsize=16, fontweight='bold')

plt.legend(loc='best', fontsize=12)

plt.axis('equal')

plt.show()
```



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

access_counts = df_new['Access_origin'].value_counts()

plt.pie(

    access_counts,

    labels=access_counts.index,

    autopct='%1.1f%%',

    startangle=140,

    colors=plt.cm.tab20.colors

)

plt.title('Access origin Distribution', fontsize=16, fontweight='bold')

plt.legend(loc='best', fontsize=12)

plt.axis('equal')

plt.show()
```



Access Type Distribution Insights 📊

- **All-access** type dominates with **51.2%**, followed by **Mobile web (24.8%)** and **Desktop (24.0%)**.

Access Origin Distribution Insights 🌐

- **All-agents** account for **75.9%** of the data, while **Spider** represents the remaining **24.1%**.

```
df_new
```

	index	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	Title	Language	Access_type	Access_origin
0	0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	...	14.0	20.0	22.0	19.0	18.0	20.0	2NE1	Chinese	all-access	spider
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	...	9.0	30.0	52.0	45.0	26.0	20.0	2PM	Chinese	all-access	spider
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	4.0	6.0	3.0	4.0	17.0	3C	Chinese	all-access	spider
3	3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	...	16.0	11.0	17.0	19.0	10.0	11.0	4minute	Chinese	all-access	spider
4	4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	3.0	11.0	27.0	13.0	36.0	10.0	52 Hz I Love You	Chinese	all-access	spider
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
145058	145058	Underworld_(serie_de_películas)_es.wikipedia.o...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	13.0	12.0	13.0	3.0	5.0	10.0	Underworld (serie de películas)	Spanish	all-access	spider
145059	145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Resident Evil: Capítulo Final	Spanish	all-access	spider
145060	145060	Enamorándome_de_Ramón_es.wikipedia.org_all-acc...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Enamorándome de Ramón	Spanish	all-access	spider
145061	145061	Hasta_el_último_hombre_es.wikipedia.org_all-ac...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Hasta el último hombre	Spanish	all-access	spider
145062	145062	Francisco_el_matemático_(serie_de_televisión_d...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Francisco el matemático (serie de televisión d...	Spanish	all-access	spider

145063 rows × 556 columns

Aggregating the dataset for further time series Analysis:

```
# Select the date columns only
date_columns = df_new.columns[1:-4]  # Assuming the last 4 columns are
metadata (Title, Language, Access_type, Access_origin)

# Reshape the data: group by 'Language' and sum views for each date
language_views_by_date = (
    df_new.groupby('Language')[date_columns]
    .sum()
    .T  # Transpose the DataFrame to make dates rows and languages columns
    .reset_index()
)

# Rename the columns
language_views_by_date.rename(columns={'index': 'Date'}, inplace=True)
```

```python
print(language_views_by_date.head())
df_clean = language_views_by_date.drop(0).copy()

# Convert the 'Date' column to datetime format
df_clean['Date'] = pd.to_datetime(df_clean['Date'])

# Convert all other columns to numeric (in case of any non-numeric values)
for col in df_clean.columns[2:]:
    df_clean[col] = pd.to_numeric(df_clean[col], errors='coerce')

df_clean.set_index('Date', inplace=True)
# Check the cleaned data
print(df_clean.head())
```

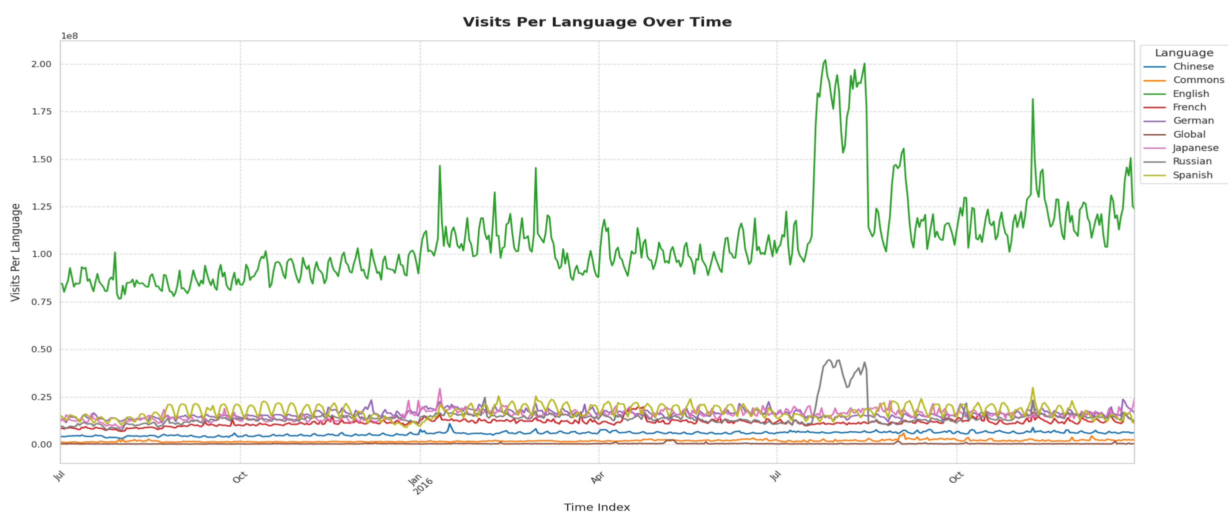
```
Language      Chinese    Commons     English     French      German    Global  \
Date
2015-07-01  4144988.0  1140821.0  84712190.0  8458638.0  13260519.0  349713.0
2015-07-02  4151189.0  1178130.0  84438545.0  8512952.0  13079896.0  383680.0
2015-07-03  4123659.0  1150547.0  80167728.0  8186030.0  12554042.0  325714.0
2015-07-04  4163448.0   951317.0  83463204.0  8749842.0  11520379.0  308756.0
2015-07-05  4441286.0  1058036.0  86198637.0  8590493.0  13392347.0  338485.0

Language      Japanese    Russian     Spanish
Date
2015-07-01  11863200.0  9463854.0  15278553.0
2015-07-02  13620792.0  9627643.0  14601013.0
2015-07-03  12305383.0  8923463.0  13427632.0
2015-07-04  15456239.0  8393214.0  12606538.0
2015-07-05  14827204.0  8938528.0  13710356.0
```

```python
df_clean
```

Language Date	Chinese	Commons	English	French	German	Global	Japanese	Russian	Spanish
2015-07-01	4144988.0	1140821.0	84712190.0	8458638.0	13260519.0	349713.0	11863200.0	9463854.0	15278553.0
2015-07-02	4151189.0	1178130.0	84438545.0	8512952.0	13079896.0	383680.0	13620792.0	9627643.0	14601013.0
2015-07-03	4123659.0	1150547.0	80167728.0	8186030.0	12554042.0	325714.0	12305383.0	8923463.0	13427632.0
2015-07-04	4163448.0	951317.0	83463204.0	8749842.0	11520379.0	308756.0	15456239.0	8393214.0	12606538.0
2015-07-05	4441286.0	1058036.0	86198637.0	8590493.0	13392347.0	338485.0	14827204.0	8938528.0	13710356.0
...	...	...	...	...	...	...	...	...	...
2016-12-27	6478442.0	2305363.0	145628731.0	15281470.0	20125264.0	320017.0	16123301.0	15040168.0	15945353.0
2016-12-28	6513400.0	2599015.0	141278366.0	13781521.0	19152389.0	729836.0	16150715.0	14000319.0	16577375.0
2016-12-29	6042545.0	2309293.0	150557534.0	13399796.0	18447906.0	320695.0	17682688.0	13478977.0	15647135.0
2016-12-30	6111203.0	2506163.0	125404585.0	12471074.0	17606030.0	431709.0	19450687.0	12066750.0	11560095.0
2016-12-31	6298565.0	2177323.0	123623809.0	11504691.0	16562720.0	392930.0	24460799.0	13223033.0	11077924.0

550 rows × 9 columns

```
sns.set(style="whitegrid")
plt.figure(figsize=(20, 10))
color_palette = sns.color_palette("tab10")  # Use Seaborn's tab10 color palette for distinct lines
df_clean.plot(ax=plt.gca(), linewidth=2, color=color_palette)
plt.xlabel("Time Index", fontsize=14, labelpad=10)
plt.ylabel("Visits Per Language", fontsize=14, labelpad=10)
plt.title("Visits Per Language Over Time", fontsize=18, fontweight="bold", pad=20)
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.7)
plt.legend(title="Language", fontsize=12, title_fontsize=14, loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



Page Views Over Time Distribution 📈

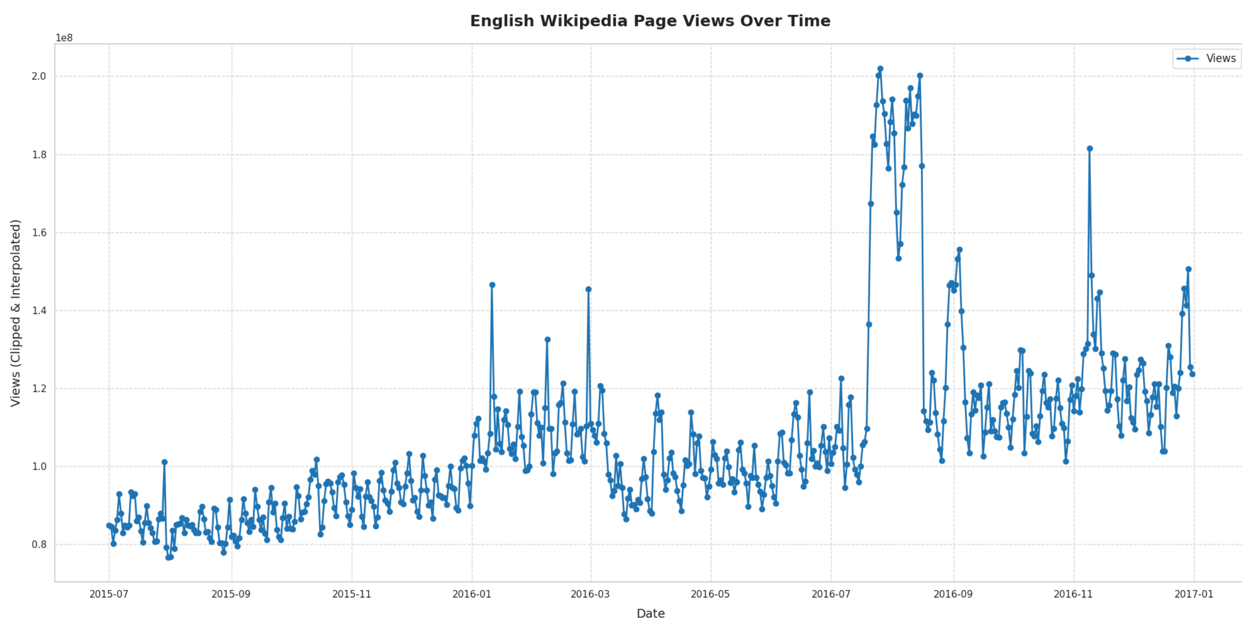
- The **visit per page over time plot** shows that the **English language** pages have the **highest number of views**, making it the primary focus for **further analysis**.

⌛ Time Series:

```
df_eng = df_clean['English']
df_eng.head(10)

df_eng = df_clean['English'].to_frame().reset_index()

# Rename columns
df_eng.columns = ["date", "views"]

df_eng.set_index('date', inplace=True)

df_eng.head()
```

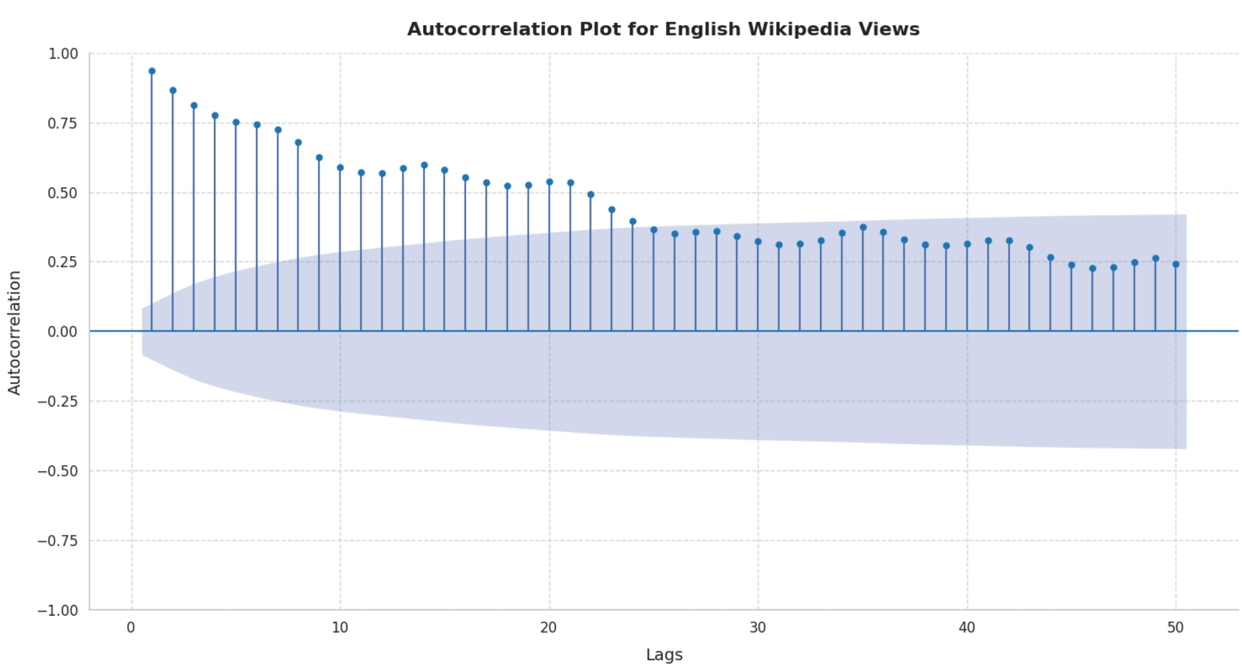
	views
date	
2015-07-01	84712190.0
2015-07-02	84438545.0
2015-07-03	80167728.0
2015-07-04	83463204.0
2015-07-05	86198637.0

```
sns.set(style="whitegrid")
plt.figure(figsize=(20, 10))
plt.plot(df_eng.index, df_eng.views, marker='o', linestyle='-',
color='#1f77b4', markersize=6, linewidth=2, label='Views')
# Customize labels, title, and legend
plt.xlabel("Date", fontsize=14, labelpad=10)
plt.ylabel("Views (Clipped & Interpolated)", fontsize=14, labelpad=10)
plt.title("English Wikipedia Page Views Over Time", fontsize=18,
fontweight="bold", pad=20)
plt.xticks(rotation=0)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=12)
# Display the plot
plt.tight_layout()
plt.show()
```



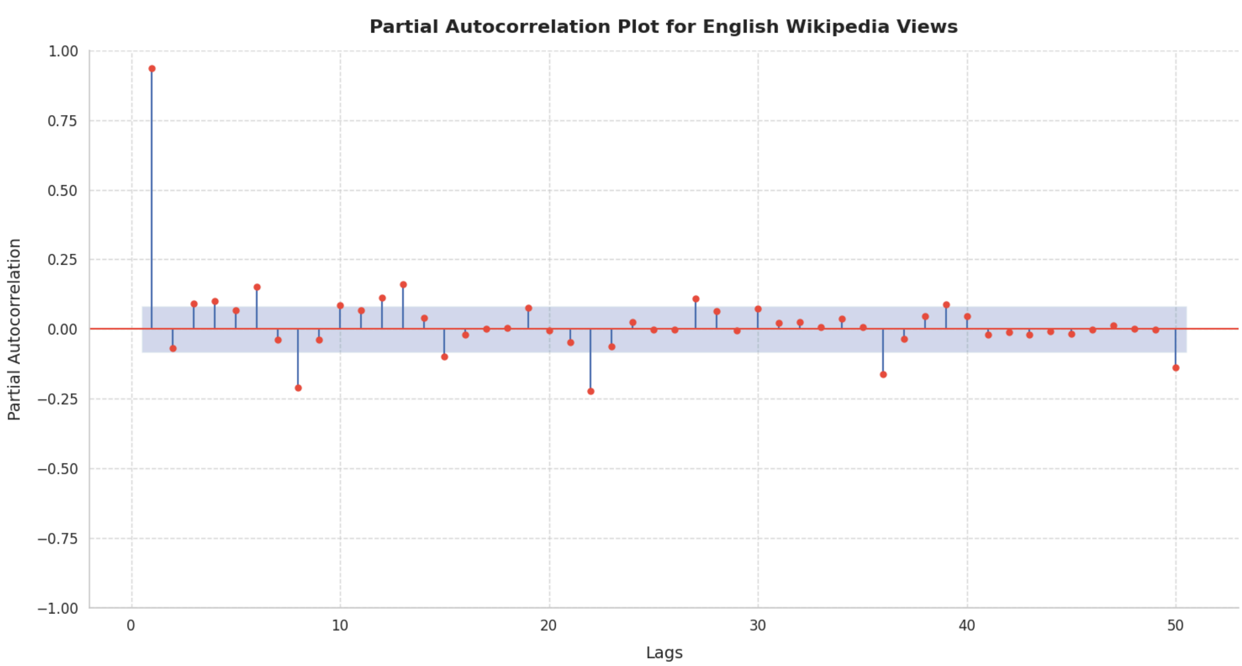
🔄 Autocorrelation:

```
from statsmodels.graphics.tsaplots import plot_acf
sns.set(style="whitegrid")
fig, ax = plt.subplots(figsize=(15, 8))
plot_acf(df_eng.views, ax=ax, lags=50, alpha=0.05, zero=False,
color='#1f77b4')
ax.set_title("Autocorrelation Plot for English Wikipedia Views",
fontsize=16, fontweight='bold', pad=15)
ax.set_xlabel("Lags", fontsize=14, labelpad=10)
ax.set_ylabel("Autocorrelation", fontsize=14, labelpad=10)
ax.grid(True, linestyle='--', alpha=0.7)
ax.tick_params(axis='both', labelsize=12)
sns.despine()
plt.tight_layout()
plt.show()
```



📉 Partial AutoCorrelation:

```
from statsmodels.graphics.tsaplots import plot_pacf
sns.set(style="whitegrid")
fig, ax = plt.subplots(figsize=(15, 8))
plot_pacf(df_eng.views, ax=ax, lags=50, alpha=0.05, zero=False,
color='#e74c3c')

# Customize the plot
ax.set_title("Partial Autocorrelation Plot for English Wikipedia Views",
fontsize=16, fontweight='bold', pad=15)
ax.set_xlabel("Lags", fontsize=14, labelpad=10)
ax.set_ylabel("Partial Autocorrelation", fontsize=14, labelpad=10)
ax.grid(True, linestyle='--', alpha=0.7)
ax.tick_params(axis='both', labelsize=12)
sns.despine()
plt.tight_layout()
plt.show()
```



✅Augmented Dickey-Fuller (ADF) test:

```python
import statsmodels.api as sm

def adf_test(series):
    p_value = sm.tsa.adfuller(series)[1]

    if p_value <= 0.05:
        print(f"✅ The series is stationary (p-value = {p_value:.4f})")
    else:
        print(f"❌ The series is NOT stationary (p-value = {p_value:.4f})")

# Perform the ADF test on your data
adf_test(df_eng['views'])
```

```
❌ The series is NOT stationary (p-value = 0.1895)
```

The initial check shows that the time series is not stationary, so we need to transform it into a stationary series to improve the accuracy of our models

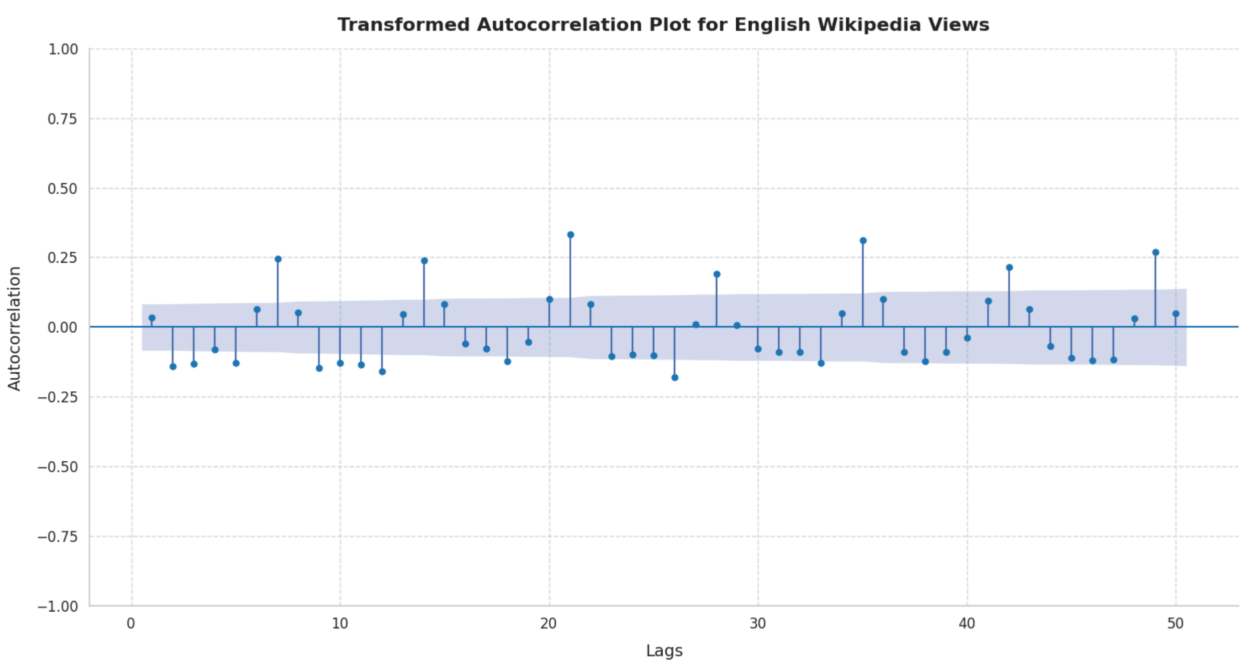
```python
# Create a differenced version of the time series to achieve stationarity
df_eng_st = df_eng.copy()
df_eng_st['views'] = df_eng_st['views'].diff(periods=1)  # Apply first-order differencing
df_eng_st.dropna(inplace=True)  # Remove NaN values resulting from differencing
# Perform the ADF test on the differenced series
print("Performing ADF test on the differenced series...")
adf_test(df_eng_st['views'])
```

```
Performing ADF test on the differenced series...
✅ The series is stationary (p-value = 0.0000)
```

Differencing has successfully transformed the time series into a stationary series, making it suitable for forecasting.

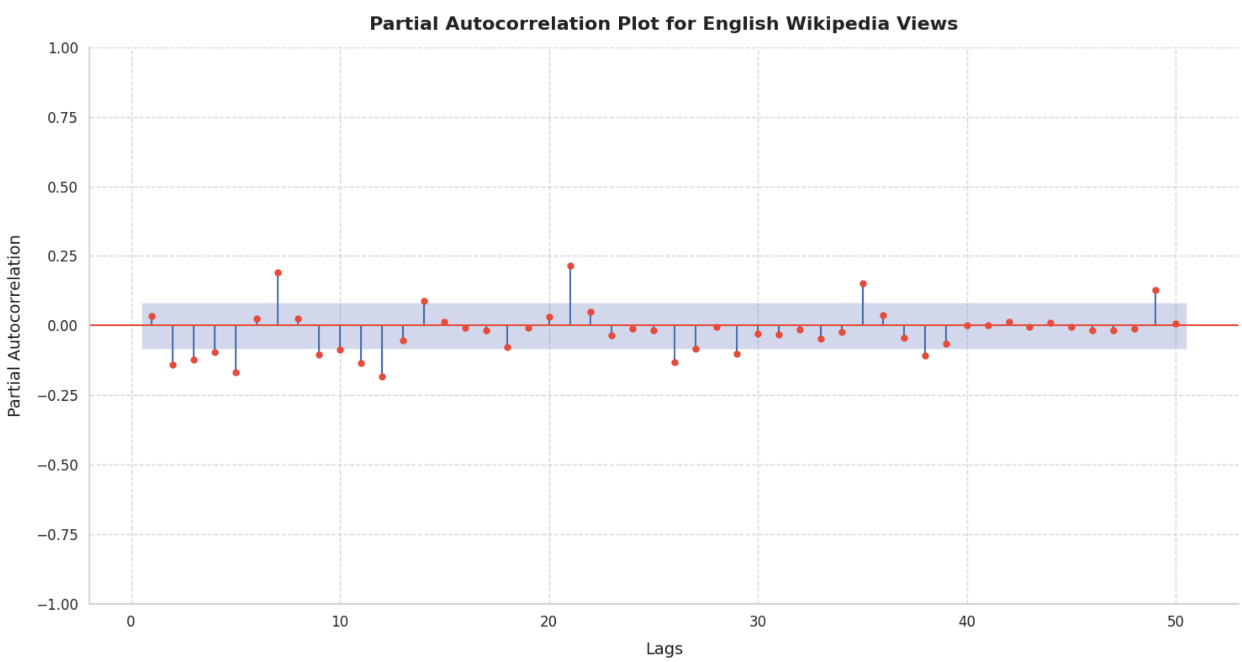
Transformed AutoCorrelation:

```
sns.set(style="whitegrid")
fig, ax = plt.subplots(figsize=(15, 8))
plot_acf(df_eng_st.views, ax=ax, lags=50, alpha=0.05, zero=False,
color='#1f77b4')
ax.set_title("Transformed Autocorrelation Plot for English Wikipedia
Views", fontsize=16, fontweight='bold', pad=15)
ax.set_xlabel("Lags", fontsize=14, labelpad=10)
ax.set_ylabel("Autocorrelation", fontsize=14, labelpad=10)
ax.grid(True, linestyle='--', alpha=0.7)
ax.tick_params(axis='both', labelsize=12)
sns.despine()
plt.tight_layout()
plt.show()
```

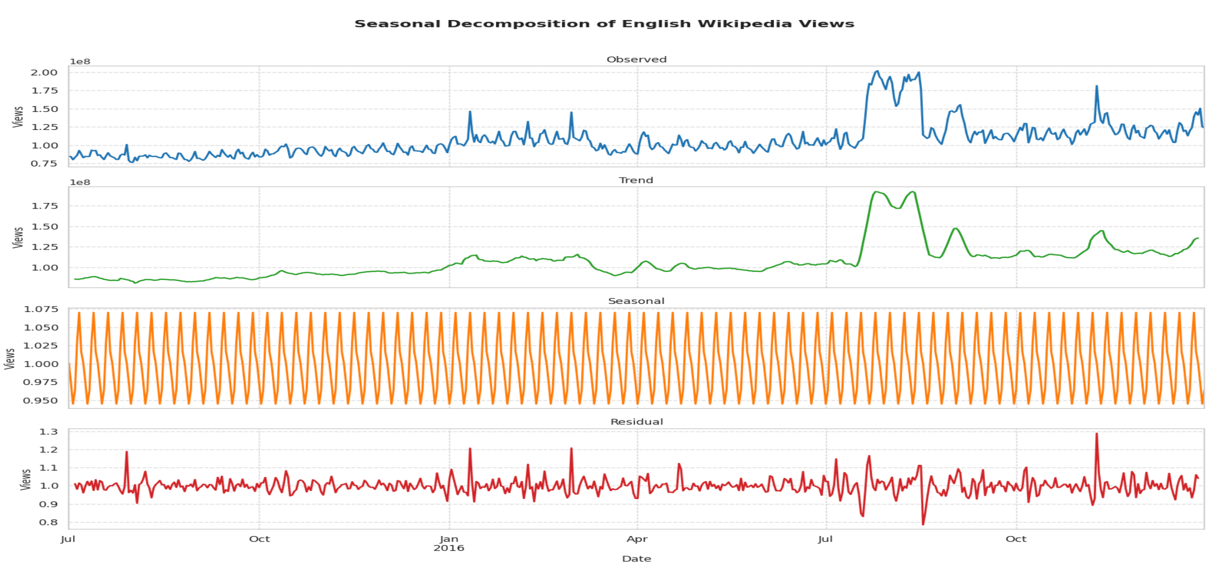


Transformed Partial Autocorrelation:

```
sns.set(style="whitegrid")
fig, ax = plt.subplots(figsize=(15, 8))
plot_pacf(df_eng_st.views, ax=ax, lags=50, alpha=0.05, zero=False,
color='#e74c3c')

# Customize the plot
ax.set_title("Partial Autocorrelation Plot for English Wikipedia Views",
fontsize=16, fontweight='bold', pad=15)
ax.set_xlabel("Lags", fontsize=14, labelpad=10)
ax.set_ylabel("Partial Autocorrelation", fontsize=14, labelpad=10)
ax.grid(True, linestyle='--', alpha=0.7)
ax.tick_params(axis='both', labelsize=12)
sns.despine()
plt.tight_layout()
plt.show()
```



```python
from statsmodels.tsa.seasonal import seasonal_decompose
sns.set(style="whitegrid")
result = seasonal_decompose(df_eng['views'], model='multiplicative')
# Plot the results
fig, axes = plt.subplots(4, 1, figsize=(16, 12), sharex=True)
result.observed.plot(ax=axes[0], color="#1f77b4", title="Observed",
linewidth=2)
result.trend.plot(ax=axes[1], color="#2ca02c", title="Trend", linewidth=2)
result.seasonal.plot(ax=axes[2], color="#ff7f0e", title="Seasonal",
linewidth=2)
result.resid.plot(ax=axes[3], color="#d62728", title="Residual",
linewidth=2)
# Customize the plots
for ax in axes:
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.tick_params(axis='both', labelsize=12)
    ax.set_ylabel("Views", fontsize=12)
    ax.set_xlabel("Date", fontsize=12)
# Set a common title
fig.suptitle("Seasonal Decomposition of English Wikipedia Views",
fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```

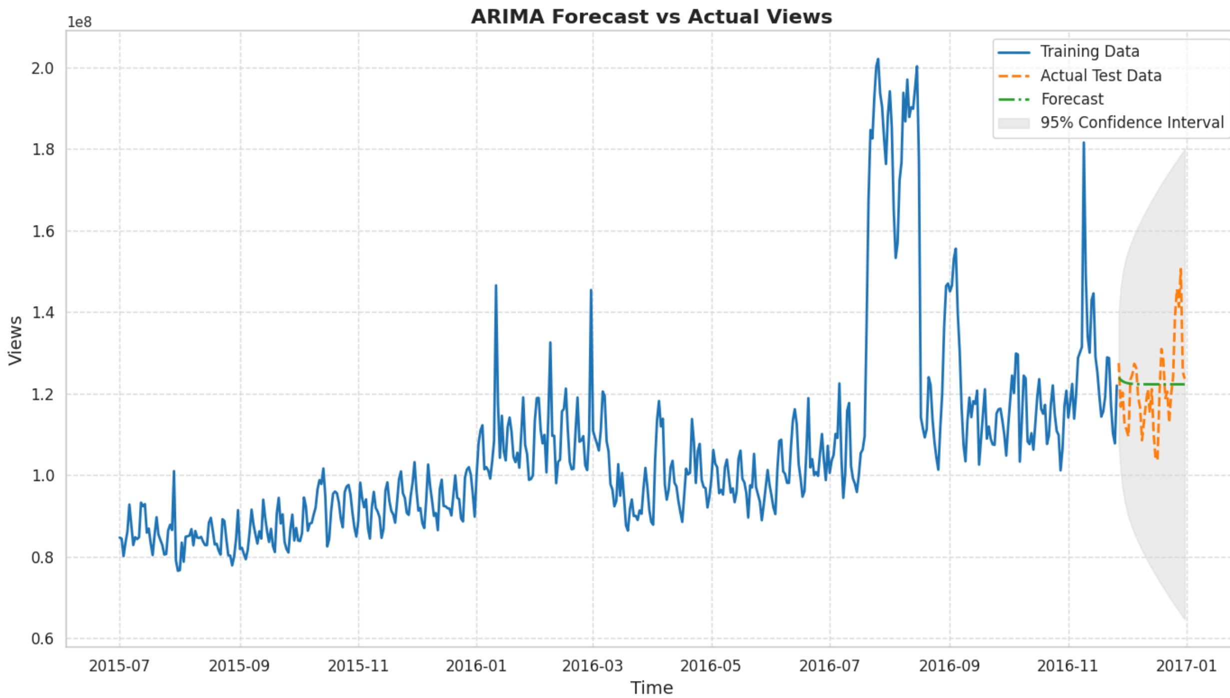


⚙️ARIMA Model:

```
!pip install pmdarima
import pandas as pd
import pmdarima as pm
from sklearn.metrics import (
    mean_squared_error as mse,
    mean_absolute_error as mae,
    mean_absolute_percentage_error as mape
)
import matplotlib.pyplot as plt

# Custom performance evaluation function
def performance(actual, predicted):
    print("Model Performance Metrics:")
    print(f"  ✅ MAE  : {round(mae(actual, predicted), 3)}")
    print(f"  ✅ RMSE : {round(mse(actual, predicted) ** 0.5, 3)}")
    print(f"  ✅ MAPE : {round(mape(actual, predicted), 3)}")
    print("-" * 40)

# Split the data (90% train, 10% test)
train_size = 515
train, test = df_eng['views'][:train_size], df_eng['views'][train_size:]

# Fit auto_arima on the training set
model = pm.auto_arima(train,
                      seasonal=False,
                      stepwise=True,
                      suppress_warnings=True,
                      trace=True)

# Summary of the best ARIMA model
print(model.summary())
# Forecast the next 35 values (same length as the test set)
n_forecast = len(test)
forecast, conf_int = model.predict(n_periods=n_forecast,
return_conf_int=True)
# Plot actual vs forecasted values (Enhanced)
plt.figure(figsize=(14, 8))
```

```python
plt.plot(train.index, train, label='Training Data', color='#1f77b4',
linewidth=2)
plt.plot(test.index, test, label='Actual Test Data', color='#ff7f0e',
linestyle='--', linewidth=2)
plt.plot(test.index, forecast, label='Forecast', color='#2ca02c',
linestyle='-.', linewidth=2)
plt.fill_between(test.index, conf_int[:, 0], conf_int[:, 1],
                 color='lightgray', alpha=0.4, label='95% Confidence
Interval')
# Customizing the plot
plt.xlabel('Time', fontsize=14)
plt.ylabel('Views', fontsize=14)
plt.title('ARIMA Forecast vs Actual Views', fontsize=16,
fontweight='bold')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
performance(test, forecast)
```

```
Best model:  ARIMA(1,1,2)(0,0,0)[0]
Total fit time: 10.490 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  515
Model:               SARIMAX(1, 1, 2)   Log Likelihood               -8909.154
Date:                Mon, 10 Feb 2025   AIC                          17826.309
Time:                        03:59:36   BIC                          17843.277
Sample:                    07-01-2015   HQIC                         17832.959
                         - 11-26-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6808      0.097      7.032      0.000       0.491       0.871
ma.L1         -0.6777      0.103     -6.609      0.000      -0.879      -0.477
ma.L2         -0.1479      0.039     -3.761      0.000      -0.225      -0.071
sigma2      6.594e+13   7.99e-16   8.25e+28      0.000    6.59e+13    6.59e+13
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):              2721.67
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):               5.43   Skew:                            -0.26
Prob(H) (two-sided):                  0.00   Kurtosis:                        14.26
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number  1e+46. Standard errors may be unstable.
```




```
Model Performance Metrics:
  ✅ MAE  : 8067911.564
  ✅ RMSE : 10643074.853
  ✅ MAPE : 0.066
```

🔍 Insights:

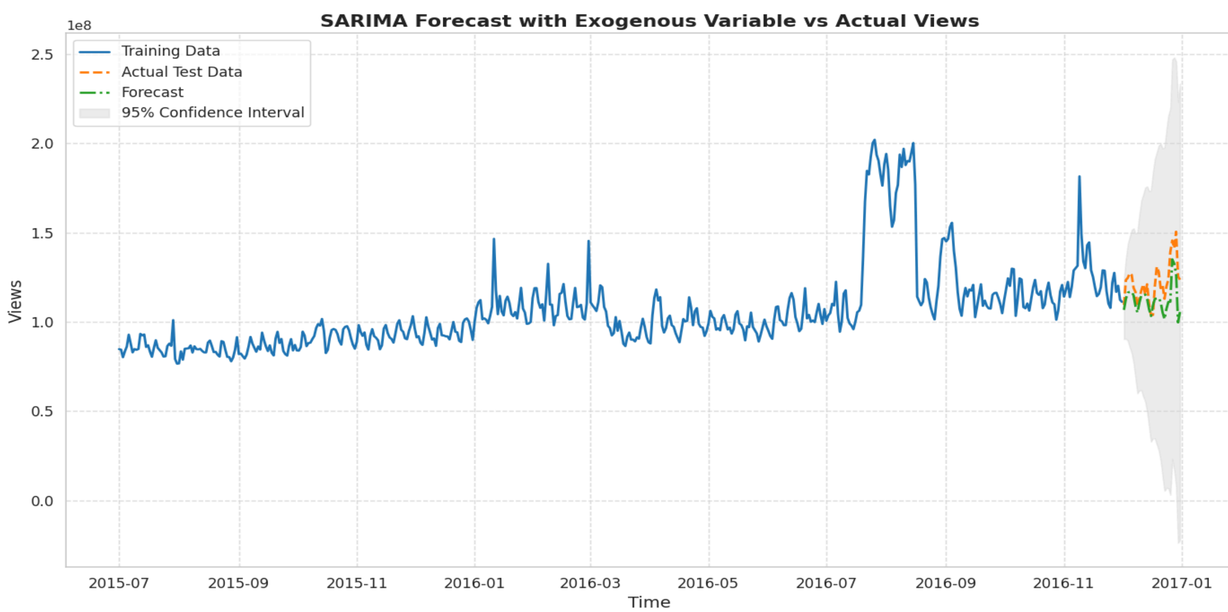
- The **best ARIMA model** identified is **ARIMA(1,1,2)(0,0,0)[0]** with a total fit time of **10.49 seconds**.
- **Model Coefficients**:
  - **AR.L1**: Significant with a coefficient of **0.6808**, indicating strong autoregressive influence.
  - **MA.L1** and **MA.L2**: Both moving average terms are significant, suggesting past forecast errors help in predicting future values.
- **Model Performance Metrics**:
  - **Mean Absolute Error (MAE)**: **8,067,911.56**
  - **Root Mean Square Error (RMSE)**: **10,643,074.85**
  - **Mean Absolute Percentage Error (MAPE)**: **6.6%**, indicating the model's predictions have a good level of accuracy.
- **Diagnostic Tests**:
  - **Ljung-Box Test (Q)**: No significant autocorrelation in residuals (**Prob(Q) = 0.83**).
  - **Heteroskedasticity Test (H)**: Indicates presence of heteroskedasticity (**Prob(H) = 0.00**).
  - **Jarque-Bera Test**: Non-normal distribution of residuals (**Prob(JB) = 0.00**).

⚙️SARIMA Model:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error as mae, mean_squared_error
as mse
import itertools
# Split the data (520 rows for training, remaining for testing)
train_size = 520
train, test = df_eng[:train_size], df_eng[train_size:]
# Convert the exogenous variable to a NumPy array
ex = exog['Exog'].to_numpy()

# Split the exogenous variable
ex_train, ex_test = ex[:train_size], ex[train_size:]

# Custom performance evaluation function
def mape(actual, predicted):
    """Calculate Mean Absolute Percentage Error."""
    return np.mean(np.abs((actual - predicted) / actual)) * 100

# SARIMA parameter selection using grid search
p = d = q = range(0, 3)
seasonal_p = seasonal_d = seasonal_q = range(0, 2)
s = 7  # Assuming weekly seasonality; adjust this based on your data

param_combinations = list(itertools.product(p, d, q))
seasonal_param_combinations = [(x[0], x[1], x[2], s) for x in
itertools.product(seasonal_p, seasonal_d, seasonal_q)]

best_aic = np.inf
best_params = None
best_seasonal_params = None

# Grid search to find the best parameters
for param in param_combinations:
    for seasonal_param in seasonal_param_combinations:
        try:
            model = SARIMAX(train['views'],
```

```
                           order=param,
                           seasonal_order=seasonal_param,
                           exog=ex_train,
                           enforce_stationarity=False,
                           enforce_invertibility=False)
            result = model.fit(disp=False)
            if result.aic < best_aic:
                best_aic = result.aic
                best_params = param
                best_seasonal_params = seasonal_param
        except Exception:
            continue

print(f"Best SARIMA params: {best_params}, Seasonal params:
{best_seasonal_params}, AIC: {best_aic}")

# Build the SARIMA model using the best parameters
model = SARIMAX(train['views'],
               order=best_params,
               seasonal_order=best_seasonal_params,
               exog=ex_train,
               enforce_stationarity=False,
               enforce_invertibility=False)
result = model.fit(disp=False)
# Forecasting
forecast_result = result.get_forecast(steps=len(test), exog=ex_test)
forecast = forecast_result.predicted_mean
conf_int = forecast_result.conf_int(alpha=0.05)
# Plot actual vs forecasted values with confidence intervals
plt.figure(figsize=(14, 8))
plt.plot(train.index, train['views'], label='Training Data',
color='#1f77b4', linewidth=2)
plt.plot(test.index, test['views'], label='Actual Test Data',
color='#ff7f0e', linestyle='--', linewidth=2)
plt.plot(test.index, forecast, label='Forecast', color='#2ca02c',
linestyle='-.', linewidth=2)
plt.fill_between(test.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1],
color='lightgray', alpha=0.4, label='95% Confidence Interval')
```

```
# Customize the plot
plt.xlabel('Time', fontsize=14)
plt.ylabel('Views', fontsize=14)
plt.title('SARIMA Forecast with Exogenous Variable vs Actual Views',
fontsize=16, fontweight='bold')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(fontsize=12)
plt.tight_layout()

plt.show()

# Evaluate the forecast using performance metrics
performance(test['views'].values, forecast.values)
```




```
Best SARIMA params: (1, 1, 2), Seasonal params: (0, 1, 1, 7), AIC: 17268.19166403674

Model Performance Metrics:
  ✅ MAE  : 9983413.102
  ✅ RMSE : 12211002.335
  ✅ MAPE : 7.902%
```

🔍 Insights:

- The **best SARIMA model** identified has parameters **(1, 1, 2)** with **seasonal parameters (0, 1, 1, 7)**.
- The model's **AIC (Akaike Information Criterion)** value is **17268.19**, indicating a relatively good fit for the data.

Model Performance Metrics:

- **Mean Absolute Error (MAE)**: **9,983,413.10**
- **Root Mean Square Error (RMSE)**: **12,211,002.34**
- **Mean Absolute Percentage Error (MAPE)**: **7.90%** — suggests moderate prediction accuracy with some room for improvement.

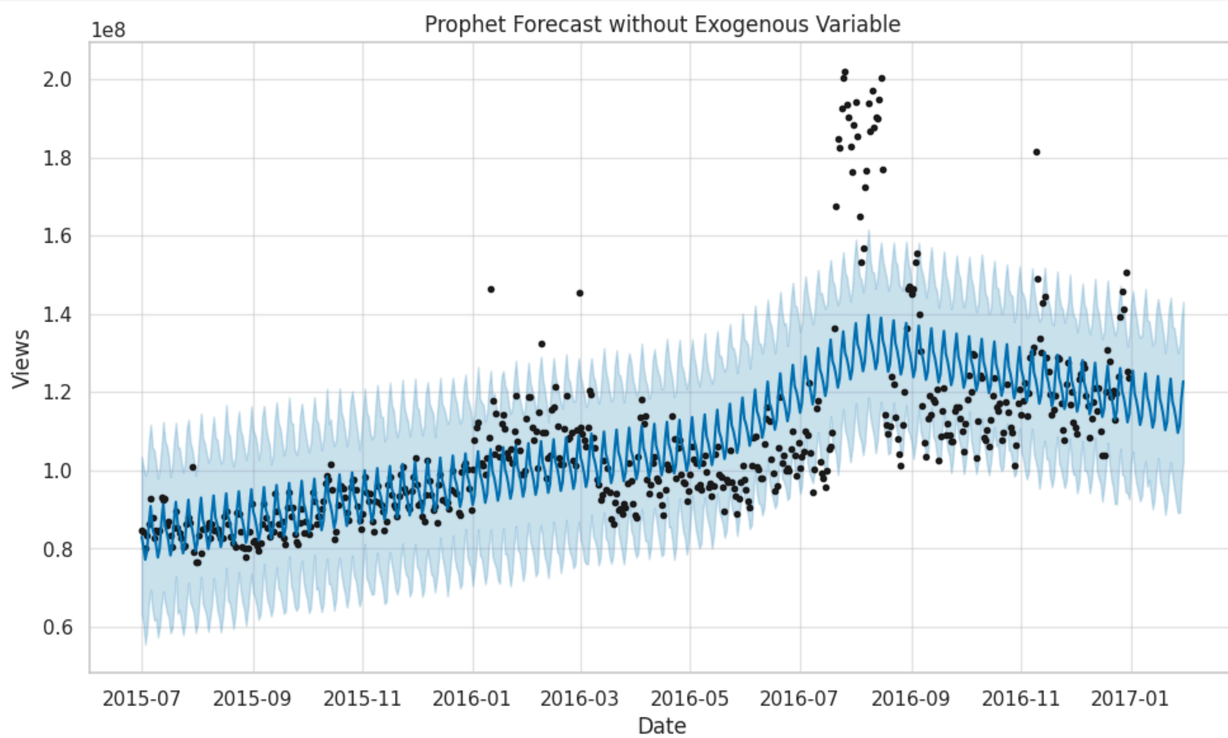
Interpretation:

- This model accounts for **weekly seasonality** and shows reasonable forecasting accuracy for the given time series data.
- Higher **RMSE** compared to the ARIMA model indicates slightly more prediction error; however, the SARIMA model better captures the seasonal patterns, making it more suitable for long-term forecasts.

⚙️FB Prophet:

```
# Prepare the data for Prophet
df_prophet = df_eng[['views']].reset_index()
df_prophet.columns = ['ds', 'y']  # Prophet requires 'ds' for dates and
'y' for values

# Initialize and fit the model
model = Prophet()
model.fit(df_prophet)

# Create a dataframe for future predictions
future = model.make_future_dataframe(periods=len(test), freq='D')

# Forecast
forecast = model.predict(future)
```

```python
# Extract the predicted values for the test period
predicted = forecast['yhat'].iloc[-len(test):].values
actual = test['views'].values

# Plot the forecast
fig = model.plot(forecast)
plt.title('Prophet Forecast without Exogenous Variable')
plt.xlabel('Date')
plt.ylabel('Views')
plt.show()
# Display performance metrics
performance(actual, predicted)
```

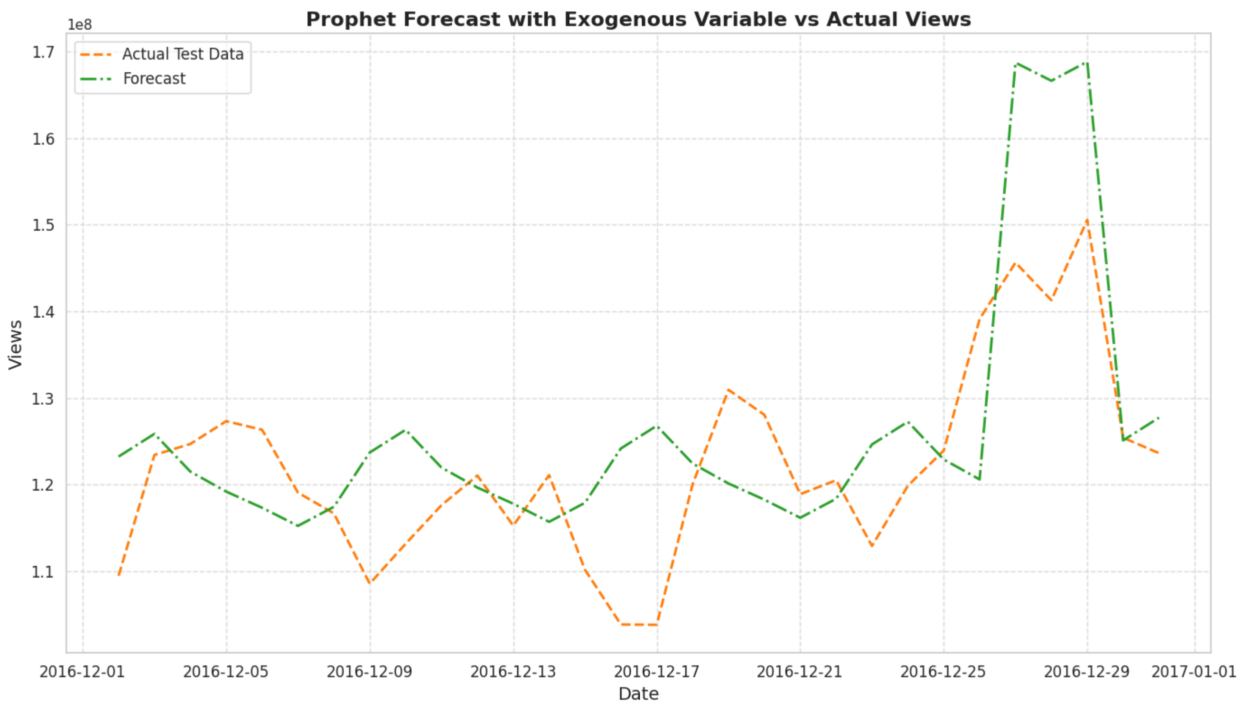



```
odel Performance Metrics:
✅ MAE  : 10870936.424
✅ RMSE : 14526242.803
✅ MAPE : 8.611%
```

FB Prophet using Exgo Variable:

```
from prophet import Prophet

# Prepare the data for Prophet with an exogenous variable
df_prophet = df_eng[['views']].reset_index()
df_prophet['extra'] = ex[:len(df_eng)]  # Add the exogenous variable from
exog to match df_eng length
df_prophet.columns = ['ds', 'y', 'extra']

# Initialize the model and add the exogenous variable
model = Prophet()
model.add_regressor('extra')

# Fit the model
model.fit(df_prophet)

# Create a dataframe for future predictions
future = model.make_future_dataframe(periods=len(test), freq='D')

# Add the exogenous variable for future periods
future['extra'] = np.concatenate([ex[:len(df_eng)], ex[-len(test):]])

# Forecast
forecast = model.predict(future)

# Extract the forecasted values for the test period
predicted = forecast['yhat'].iloc[-len(test):].values
actual = test['views'].values

# Display performance metrics
performance(actual, predicted)

# Plot the forecast
plt.figure(figsize=(14, 8))
plt.plot(test.index, actual, label='Actual Test Data', color='#ff7f0e',
linestyle='--', linewidth=2)
plt.plot(test.index, predicted, label='Forecast', color='#2ca02c',
linestyle='-.', linewidth=2)
```

```
plt.title('Prophet Forecast with Exogenous Variable vs Actual Views',
fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Views', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```




```
Model Performance Metrics:
  ✅ MAE  : 9058842.775
  ✅ RMSE : 11719708.492
  ✅ MAPE : 7.415%
```

🔍 Insights:from FB Prophet Model 📈

Without EXO Variable

- **Mean Absolute Error (MAE)**: **10,870,936.42**
- **Root Mean Square Error (RMSE)**: **14,526,242.80**
- **Mean Absolute Percentage Error (MAPE)**: **8.61%**

👉 The model shows a reasonable forecasting accuracy; however, the errors are quite high, indicating potential improvement if external factors are considered.

With EXO Variable

- **Mean Absolute Error (MAE)**: **9,058,842.78**
- **Root Mean Square Error (RMSE)**: **11,719,708.49**
- **Mean Absolute Percentage Error (MAPE)**: **7.41%**

👉 **Significant improvement** in model performance is observed after introducing the **EXO variable**, reducing both **MAE** and **RMSE**, and increasing prediction accuracy.

- **MAPE improvement** from **8.61%** to **7.41%** suggests that external factors have a meaningful impact on the time series and should be included for better forecasts.

Overall Insights 🔍

1️⃣Data Preprocessing and Exploration

- The dataset consists of **daily page views for 145,063 Wikipedia pages** over a period of **550 days**, spanning multiple languages, access types, and access origins.
- The Page column was split into **Title**, **Language**, **Access Type**, and **Access Origin**, which enabled deeper insights.
- **Missing values** were replaced with zeros, ensuring that the time series data was consistent for further analysis.

2️⃣Language Distribution 🌍

- **English** pages have the highest count (**24,180 pages**), followed by **Japanese** (**20,431 pages**) and **German** (**18,547 pages**).
- The least represented languages are **Global** (**7,300 pages**), **Commons** (**10,555 pages**), and **Spanish** (**14,069 pages**).

3️⃣Access Type and Access Origin Distribution 📱💻

- **Access Type** Distribution:
    - **All-access**: 51.2%
    - **Mobile Web**: 24.8%
    - **Desktop**: 24.0%
- **Access Origin** Distribution:
    - **All-agents**: 75.9%
    - **Spider**: 24.1%

4️⃣ Time Series Analysis and Forecasting 📊

- The initial time series was **non-stationary**, so **differencing** was applied to achieve stationarity, which is crucial for accurate forecasting.
- **English-language pages** were selected for focused analysis, as they had the highest number of views.

1️⃣ ARIMA Model (1,1,2):

- **MAE**: 8,067,911
- **RMSE**: 10,643,074
- **MAPE**: 6.6%

2️⃣ SARIMA Model (1,1,2)(0,1,1,7):

- **MAE**: 9,983,413
- **RMSE**: 12,211,002
- **MAPE**: 7.90%

SARIMA performed better for seasonal patterns compared to ARIMA but had slightly higher error metrics.

3️⃣ FB Prophet Results

- **Without EXO Variable**:
    - **MAE**: 10,870,936
    - **RMSE**: 14,526,242
    - **MAPE**: 8.61%
- **With EXO Variable**:
    - **MAE**: 9,058,843
    - **RMSE**: 11,719,708
    - **MAPE**: 7.41%

Adding the EXO variable significantly improved prediction accuracy, demonstrating the value of incorporating external factors.

Key Takeaways ✨

- **Language-specific trends** help identify where to focus resources for maximizing engagement and optimizing content strategies.
- **Access origin and type** data provide insights for enhancing **user experience** by improving performance across mobile and desktop platforms.
- **FB Prophet with an EXO variable** is the most accurate forecasting model, suggesting that external factors are vital for boosting prediction performance.
- **SARIMA** is still valuable for identifying and understanding **seasonality patterns**, even though its predictive performance was slightly lower.

Recommendations ✅

1️⃣ Focus on Language-Specific Content

- Prioritize **English, Japanese, and German** pages, as they represent the majority of views.
- Expand content for **Spanish and Commons**, as these have potential for growth despite being less represented.

2️⃣ Optimize for Mobile Web

- With **24.8% of traffic** coming from mobile web, enhancing the mobile experience will likely boost overall engagement.

3️⃣ Incorporate External Factors in Forecasting

- Use **FB Prophet with EXO variables** for future predictions to improve accuracy and capture external trends.
- Consider using factors like **seasonal trends, holidays, or major events** as additional variables.

4️⃣ Improve Resource Allocation

- Use **seasonality insights** from SARIMA to plan content releases during peak traffic periods for maximum impact.

5️⃣ Monitor Access Origin Trends

- Keep a close watch on **spider traffic (24.1%)**, ensuring it doesn't disrupt analytics or skew user engagement metrics.