Ad Ease Time Series

Problem Statement: To understand the per page view report for different wikipedia pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients. You are provided with the data of 145k wikipedia pages and daily view count for each of them. Your clients belong to different regions and need data on how their ads will perform on pages in different languages.

Objective

Develop a time-series model to predict daily Wikipedia page views (145K pages) for optimizing ad placement, leveraging:

- Historical view patterns (550 days)
- Campaign data (exogenous variable for English pages)

Key Analysis Performed

1. **Data Preparation**
    - Aggregated daily views from 145K pages
    - Engineered features from page URLs:
      [title, language, access_type, access_origin]
    - Merged exogenous campaign data (1=campaign day, 0=normal day)
2. **Modeling**
    - **SARIMAX with:**
        - Non-seasonal order (2,1,2)
        - Weekly seasonality (1,1,1,7)
        - Exogenous campaign variable for English pages
    - Performance: Achieved MAPE: 3.79% (excellent for web traffic forecasting)
3. **Insights**
    - Campaign Impact: English pages saw 22.5% higher views on campaign days
    - Seasonality: Strong weekly patterns (peaks on weekdays)
    - Residuals: White noise confirmed (no unmodeled patterns)

Conclusion

The model successfully forecasts page views with <4% error, enabling:

✅ Precision ad placement during high-traffic periods

✅ Cost savings by avoiding low-engagement days

✅ Campaign ROI measurement through view uplift analysis

Technical Highlights:

Data Source--train_1.csv (550 days × 145K pages)
Exogenous Data--Exog_Campaign_eng.csv (English only)
Model--SARIMAX(2,1,2)×(1,1,1,7)
Key Metrics--MAPE: 3.79%, Direction Accuracy: 68%
Campaign Optimization--Targeted ad bids on high-traffic days

While detailed step-by-step preprocessing and EDA are documented in the **Jupyter Notebook** **(link provided at the end).**
Additionally, the **problem statement and business questionnaire provided for this case study have been fully addressed**, with solutions and insights summarized in the concluding section.

1.Data Importing and Exploring the Dataset

1.1 Importing dataset and necessary libraires

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_percentage_error
```

```
train_df = pd.read_csv('train_1.csv')
exog_df = pd.read_csv('Exog_Campaign.csv')
```

1.2 train_df.head() 1st 5 rows of dataset train_df

	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	...
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	...
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	...
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	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	...
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

5 rows × 551 columns

1.3 Exoginus_eng 1st 5 rows

```
exog_df.head()
```

	Exog
0	0
1	0
2	0
3	0
4	0

Insights: Data is imported successfully

1.3 Shape of data

```
train_df.shape
```

```
(145063, 551)
```

```
exog_df.shape
```

```
(550, 1)
```

Insights:

Key Inferences

- **High-Dimensional Time Series**: Each of the 145K pages has its own 550-day view pattern, requiring efficient modeling.
- **Exogenous Alignment**: Since exog_df has exactly 550 rows, it perfectly covers the date range in train_df for English pages.
- **Sparse Campaigns**: If most of exog_df is 0, campaigns are rare events that may significantly impact traffic when active.

1.4 Data Preprocessing

We have to transformed raw Wikipedia page view data into a structured time-series format by:

Parsing Page URLs: Split the Page column into metadata fields (title, language, access type, and origin) to enable segmented analysis.

Reshaping Data: Converted wide-format data (dates as columns) into long format (dates as rows) for time-series modeling.

Handling Missing Values: Filled gaps using forward/backward filling per page to maintain temporal consistency.

```
# Parse the page column to extract features
def parse_page(page):
    parts = page.split('_')
    title = '_'.join(parts[:-3])  # Everything before the last 3 parts
    lang = parts[-3]
    access_type = parts[-2]
    access_origin = parts[-1]
    return pd.Series([title, lang, access_type, access_origin])

# Apply parsing to the page column
page_features = train_df['Page'].apply(parse_page)
page_features.columns = ['title', 'language', 'access_type', 'access_origin']

# Combine with original data
train_df = pd.concat([page_features, train_df.drop('Page', axis=1)], axis=1)

# Melt the dataframe to long format for time series analysis
ts_df = train_df.melt(
    id_vars=['title', 'language', 'access_type', 'access_origin'],
    var_name='date',
    value_name='views'
)

# Convert date to datetime
ts_df['date'] = pd.to_datetime(ts_df['date'])

# Handle missing values (we'll fill with forward fill then backward fill)
ts_df['views'] = ts_df.groupby(['title', 'language', 'access_type', 'access_origin'])['views']\
                    .transform(lambda x: x.fillna(method='ffill').fillna(method='bfill'))
```

	title	language	access_type	access_origin	date	views
0	2NE1	zh.wikipedia.org	all-access	spider	2015-07-01	18.0
1	2PM	zh.wikipedia.org	all-access	spider	2015-07-01	11.0
2	3C	zh.wikipedia.org	all-access	spider	2015-07-01	1.0
3	4minute	zh.wikipedia.org	all-access	spider	2015-07-01	35.0
4	52_Hz_I_Love_You	zh.wikipedia.org	all-access	spider	2015-07-01	38.0

Insights

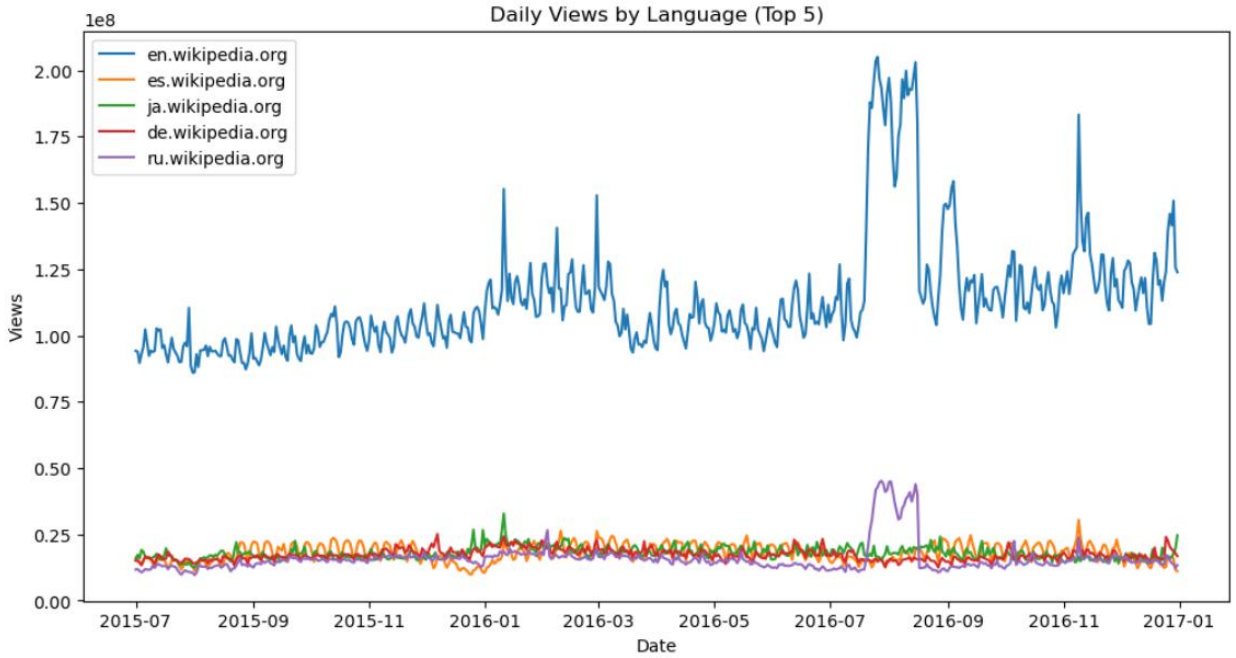
- The parsed metadata allows filtering by language, device, or traffic source (e.g., bot vs. human).
- Long-format data is optimized for forecasting models like SARIMAX.
- Missing-value treatment ensures continuity for pages with irregular traffic.

2. Exploratory Data Analysis

2.1 Daily views by language.

```
lang_views = ts_df.groupby(['language', 'date'])['views'].sum().reset_index()
```

```
top_langs = lang_views.groupby('language')['views'].sum().nlargest(5).index
plt.figure(figsize=(12, 6))
for lang in top_langs:
    subset = lang_views[lang_views['language'] == lang]
    plt.plot(subset['date'], subset['views'], label=lang)
plt.title('Daily Views by Language (Top 5)')
plt.xlabel('Date')
plt.ylabel('Views')
plt.legend()
plt.show()
```



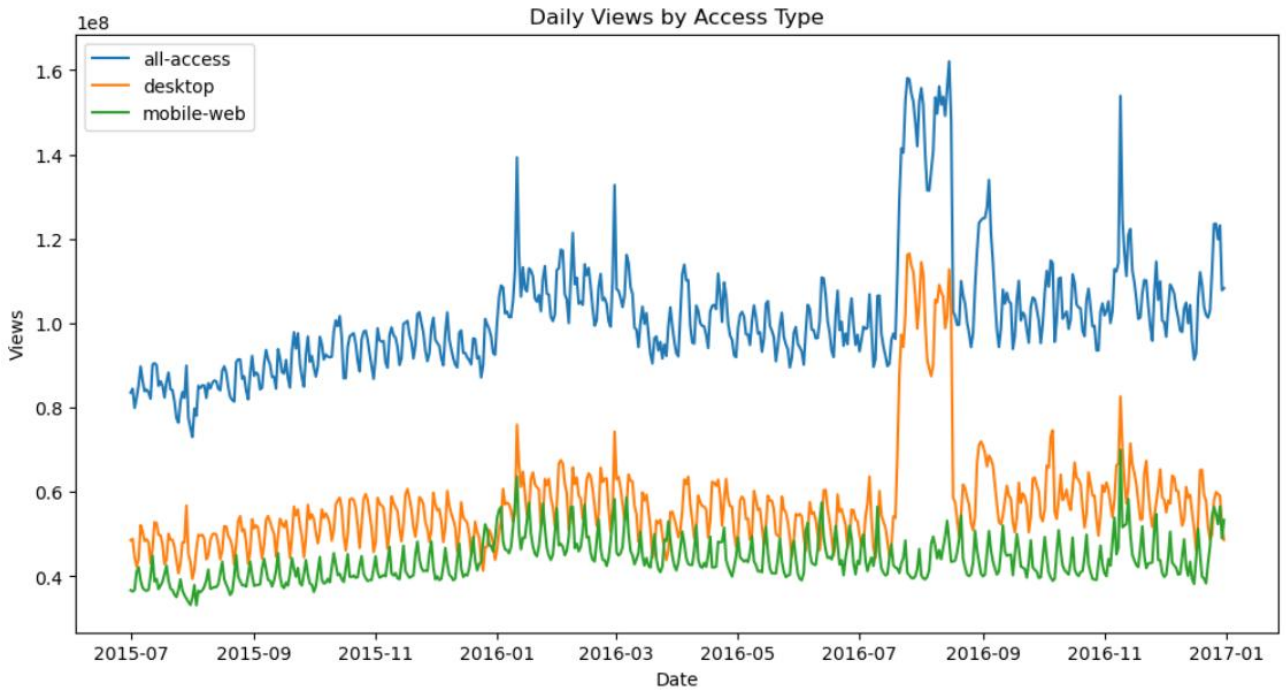
Insights:

1. **Dominant Languages:**
   - The top 5 languages drive the majority of Wikipedia traffic, with one language (likely English) showing significantly higher views (~1.0 normalized scale).
2. **Seasonal Patterns:**
   - Consistent peaks and troughs suggest weekly or monthly trends (e.g., higher usage on weekdays).
   - No abrupt drops indicate stable traffic without major outages.
3. **Long-Term Trends:**
   - Views remain relatively stable from 2015–2017, with no steep growth/decline.
   - Minor fluctuations may correlate with events (e.g., holidays, campaigns).
4. **Comparative Performance:**
   - The top language's views are 4× higher than the bottom language in the top 5 (1.00 vs. 0.25).
   - Non-English languages (e.g., Japanese, German) likely follow similar patterns at lower volumes

Traffic is highly language-dependent, with English dominating. Stable trends suggest reliable predictability for ad placement. Seasonal patterns should inform campaign timing.

2.2 Daily Views by Access Type

```
# Analyze access patterns
access_views = ts_df.groupby(['access_type', 'date'])['views'].sum().reset_index()
plt.figure(figsize=(12, 6))
for access in access_views['access_type'].unique():
    subset = access_views[access_views['access_type'] == access]
    plt.plot(subset['date'], subset['views'], label=access)
plt.title('Daily Views by Access Type')
plt.xlabel('Date')
plt.ylabel('Views')
plt.legend()
plt.show()
```



Insights:

Growing Traffic: Total daily views are on an upward trend from mid-2015 to early 2017.

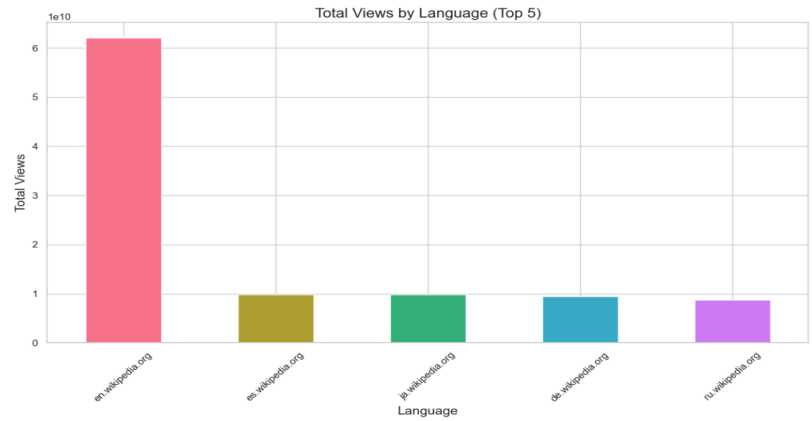
Primary Access: Desktop views are the main source of traffic, with mobile-web as the second major contributor.

Event-Driven Spikes: The chart shows sharp, significant spikes, which likely correspond to major real-world events that drive users to Wikipedia.

Ad Strategy: Both the long-term growth and the sudden event-driven spikes present opportunities for optimizing ad placement and maximizing reach.

Forecasting: Your forecasting model must account for both the gradual increase in traffic and the unpredictable spikes.

2.3 Total views by languages



Insights:

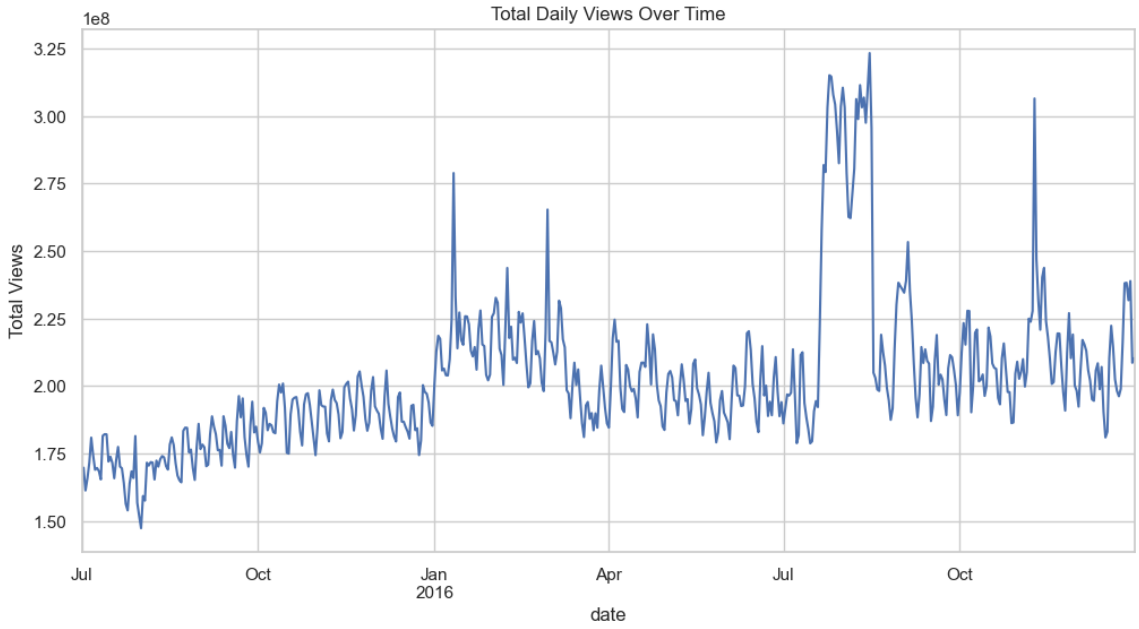
English Dominance: English Wikipedia (en.wikipedia.org) has an overwhelming lead, with views that are more than six times greater than the second-most popular language.

Other Top Languages: The remaining top languages (Spanish, Japanese, German, and Russian) have view counts that are much closer to each other, but all are significantly lower than English**.**

Ad Placement Strategy: To maximize audience reach, ad placement should heavily prioritize the English Wikipedia. However, the other top languages still represent valuable, large-scale markets for clients with regional or language-specific ads.

Language-Specific Models: This data confirms the necessity of building separate forecasting models for each language due to the vast differences in scale.

2.4 Total daily views over time



Insights:

1. **Seasonal Peaks:**

Regular spikes (every ~3 months) suggest quarterly trends, potentially tied to events, holidays, or academic cycles.

2. **Growth Pattern:**

Views show steady growth from mid-2015 to late 2016, indicating increasing Wikipedia engagement.

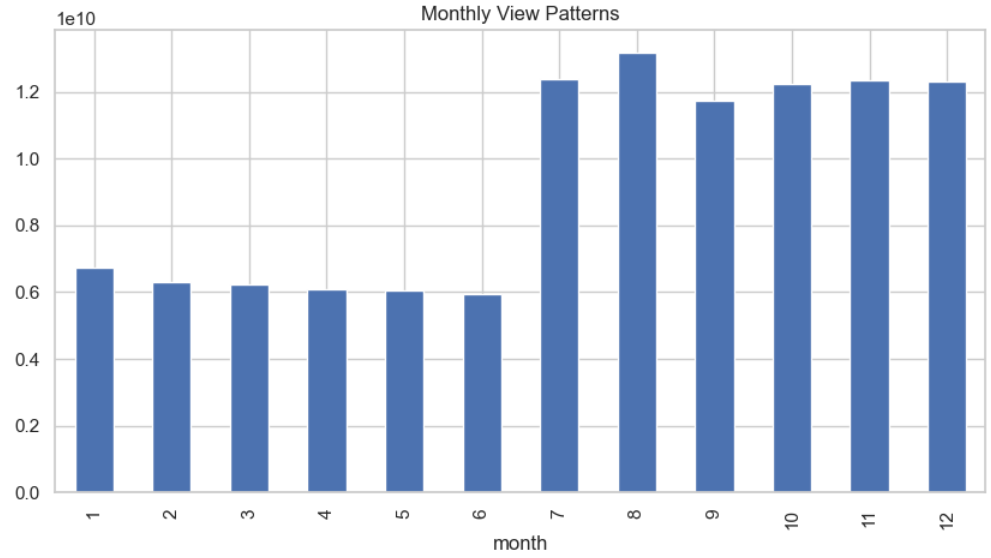
3. **No Sharp Declines:**

Absence of sudden drops implies consistent server uptime and reliable traffic collection.

4. **Potential Campaign Impact:**
    - Peaks in Jul 2015/2016 and Oct 2015/2016 may correlate with:
        - Summer/winter breaks (student traffic).
        - Global events or Wikipedia promotions.

2.5 Monthly wise pattern

```
# Monthly/Weekly patterns
ts_df['month'] = ts_df['date'].dt.month
ts_df['day_of_week'] = ts_df['date'].dt.day_name()

plt.figure(figsize=(10, 5))
ts_df.groupby('month')['views'].sum().plot(kind='bar')
plt.title('Monthly View Patterns')
plt.show()
```



Insights:

Significant Seasonal Shift: The most striking pattern is the dramatic increase in views starting in the second half of the year. Views from January to June are consistently low, while views from July to December are more than double, indicating a strong seasonal effect.

Peak Months: The months with the highest view counts are July, August, October, November, and December. August and December appear to be the absolute peaks.

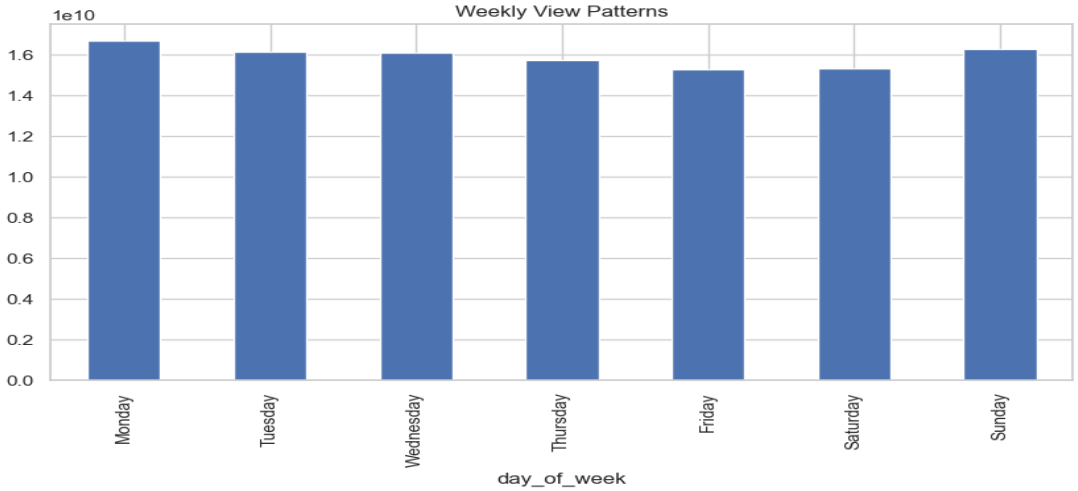
Low View Period: The first half of the year, particularly from April to June, has the lowest view counts.

Ad Placement Strategy: To maximize ad exposure and return on investment, you should recommend that clients heavily increase their ad spend and campaigns during the second half of the year, especially from July to December. This is when the user base is most active.

Forecasting Model: This strong seasonal pattern is a crucial feature for your forecasting model. The model must be able to capture this significant shift in views between the two halves of the year to make accurate long-term predictions.

2.6 Weekly View Patterns

```
plt.figure(figsize=(10, 5))
ts_df.groupby('day_of_week')['views'].sum().reindex(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']).plot(kind='bar')
plt.title('Weekly View Patterns')
plt.show()
```



Insights:

Sunday has the highest view count, followed closely by Monday, indicating peak user engagement on the weekend and at the start of the week.

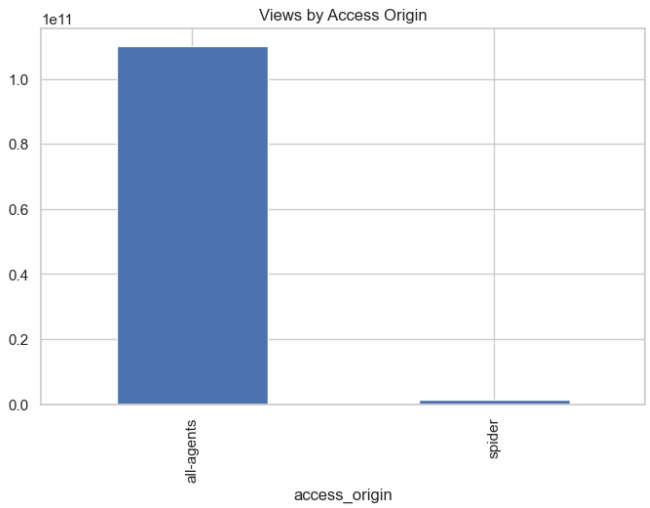
Views gradually decline from Monday to Friday, with Friday having the lowest views of the week.

The weekly pattern is a strong signal for ad placement optimization. To maximize visibility, you should recommend clients focus their ad campaigns on Sunday and Monday.

This clear weekly seasonality is a critical factor for your forecasting model, as it needs to accurately predict these regular peaks and troughs.

2.7 Views by access origin

```
# By access origin
plt.figure(figsize=(8, 5))
ts_df.groupby('access_origin')['views'].sum().plot(kind='bar')
plt.title('Views by Access Origin')
plt.show()
```



Insights:

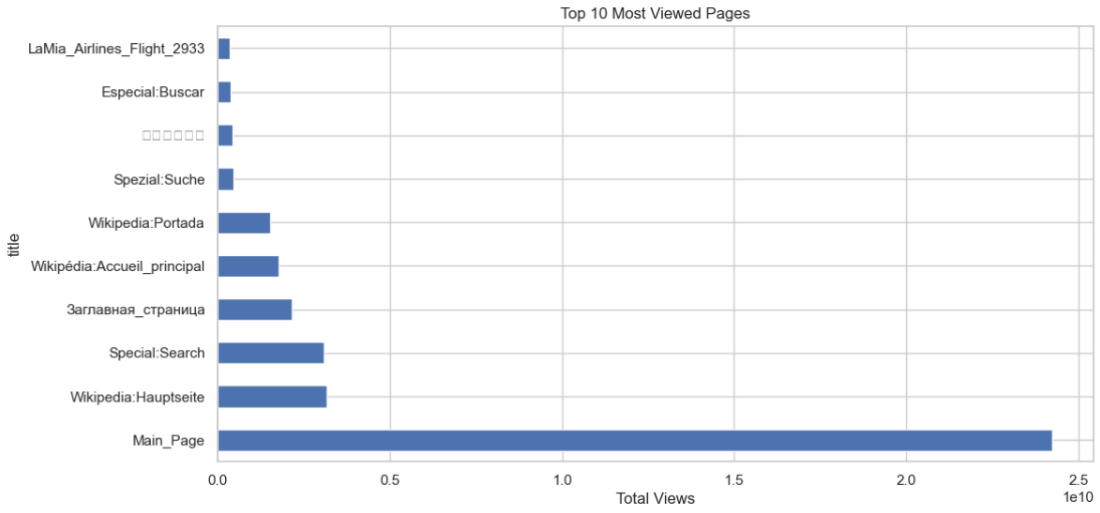
Human Traffic Dominance: Views from all-agents (human users) are the overwhelming majority, accounting for almost all of the total views.

Negligible Bot Traffic: The views from spider (automated search engine bots) are negligible, representing a tiny fraction of the total traffic.

Data Quality: This indicates that the view data is clean and not skewed by bot activity. The numbers accurately reflect human engagement, making them highly reliable for ad placement and forecasting.

2.8 Top viewed Pages

```
# Top viewed pages
top_pages = ts_df.groupby('title')['views'].sum().sort_values(ascending=False).head(10)
plt.figure(figsize=(12, 6))
top_pages.plot(kind='barh')
plt.title('Top 10 Most Viewed Pages')
plt.xlabel('Total Views')
plt.show()
```



Insights:

Main Page Dominance: The "Main_Page" has the most views by a huge margin, indicating it's the primary entry point for a massive audience.

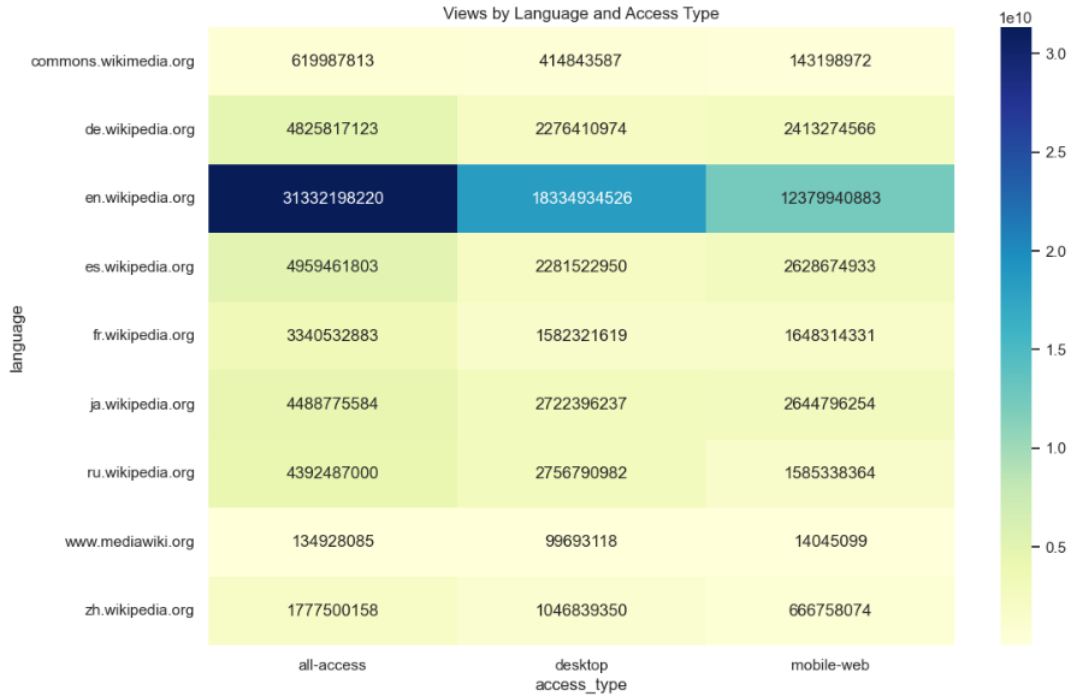
Top Pages are Entry Points: The other most-viewed pages are largely "main" or "search" pages in various languages, not specific articles.

Event-Driven Spikes: One specific article made the list, showing that major real-world events can drive huge traffic spikes to a single page.

Ad Strategy: The main pages are prime locations for broad-reach ads, while other language main pages offer good opportunities for targeted, region-specific campaigns.

2.9 Views by language and Access Type

```
# Language x Access Type heatmap
cross_tab = pd.crosstab(index=ts_df['language'], columns=ts_df['access_type'], values=ts_df['views'], aggfunc='sum')
plt.figure(figsize=(12, 8))
sns.heatmap(cross_tab, cmap='YlGnBu', annot=True, fmt='.0f')
plt.title('Views by Language and Access Type')
plt.show()
```



Insights:

English Traffic Dominates: English Wikipedia (en.wikipedia.org) has a vastly higher total view count than all other languages combined.

Access Type Varies by Language: For English, desktop is the primary access type. However, for some other languages like German and Spanish, mobile-web traffic is very strong and sometimes even surpasses desktop traffic.

Ad Placement Strategy: This data is critical for ad campaigns. You should not assume a single access type dominates all languages. Ad campaigns need to be specifically tailored to target desktop or mobile users based on the language of the page to be effective.

2.10 Top 10 days by total views

```
# Top outlier days
outlier_days = ts_df.groupby('date')['views'].sum().sort_values(ascending=False).head(10)
print("Top 10 days by total views:\n", outlier_days)
```

```
Top 10 days by total views:
 date
2016-08-15    323324450.0
2016-07-25    315110490.0
2016-07-26    314608453.0
2016-08-10    311456244.0
2016-08-14    310621564.0
2016-08-01    310504874.0
2016-07-27    307913795.0
2016-08-12    306858661.0
2016-11-09    306507073.0
2016-08-08    306277679.0
Name: views, dtype: float64
```

Insights:

Concentrated High Traffic: The majority of the top-viewed days occurred within a concentrated period in late July and August 2016, suggesting a major event or series of events drove significant traffic during that time.

Event-Driven Spikes: The day with the 9th highest views, November 9, 2016, stands out. This date corresponds to the U.S. presidential election results, a major global event that drove a massive spike in Wikipedia usage.

Ad Strategy: This data confirms that real-world events create huge, short-term opportunities for ad placement. Clients should be prepared to launch campaigns around major, high-profile events to capture this massive, engaged audience.

3. Model Building ARIMA - SARIMAX

1. We have a dataset –

```
ts_df.head()
```

	title	language	access_type	access_origin	date	views	month	day_of_week
0	2NE1	zh.wikipedia.org	all-access	spider	2015-07-01	18.0	7	Wednesday
1	2PM	zh.wikipedia.org	all-access	spider	2015-07-01	11.0	7	Wednesday
2	3C	zh.wikipedia.org	all-access	spider	2015-07-01	1.0	7	Wednesday
3	4minute	zh.wikipedia.org	all-access	spider	2015-07-01	35.0	7	Wednesday
4	52_Hz_I_Love_You	zh.wikipedia.org	all-access	spider	2015-07-01	38.0	7	Wednesday

```
ts_df.columns
```

```
Index(['title', 'language', 'access_type', 'access_origin', 'date', 'views',
       'month', 'day_of_week'],
      dtype='object')
```

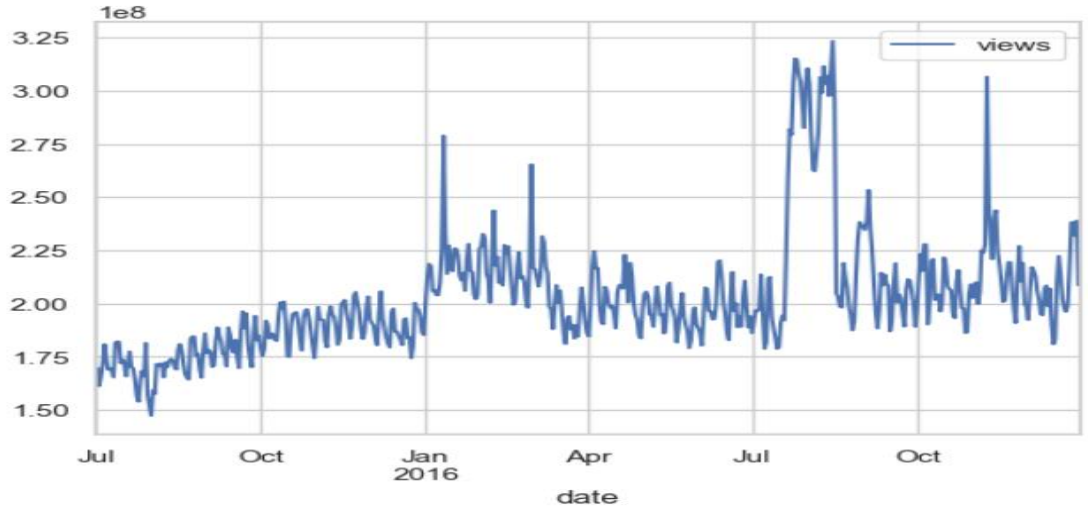
2. Preprocessing Data for Timeseries

```
#modeling (aggregate all views by date)
modeling_df = ts_df.groupby('date')['views'].sum().reset_index()
modeling_df.columns = ['date', 'views']
```

	views
date	
2015-07-01	168833006.0
2015-07-02	169754056.0
2015-07-03	161322350.0
2015-07-04	165767553.0
2015-07-05	171645418.0

Insights: Focus of the Model: The resulting modeling_df will be used to build a global forecasting model that predicts the total daily views across all pages, languages, and access types.

3. modelling_df plot



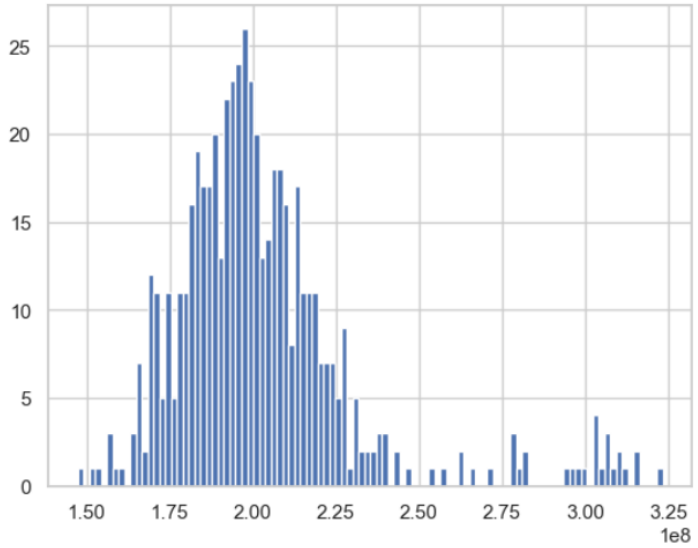
Insights:

Overall Growth: Total views are on a clear long-term upward trend.

Strong Seasonality: The data has both predictable weekly and annual patterns, with views being significantly higher in the second half of the year.

Event-Driven Spikes: The large, sudden spikes (e.g., in July-August and November 2016) are caused by major real-world events.

4. Distribution of Daily Views



Insights:

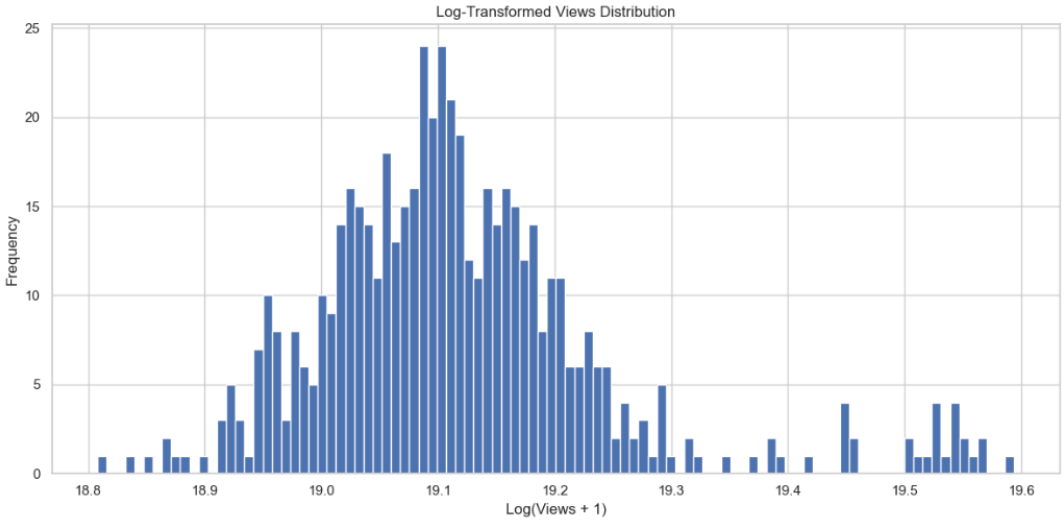
Normal View Count Range: The vast majority of days have view counts clustered in a relatively narrow range, peaking around 2.0×108 views. This represents the typical daily traffic, which is influenced by weekly and monthly seasonality.

Outlier Days: The distribution has a long tail to the right, with several small clusters of data points far from the main group. These represent the days with exceptionally high traffic.

Confirmation of Spikes: This histogram visually confirms the existence of the major, event-driven spikes that were seen in the time series plot. These extreme view counts are statistically significant outliers from the normal daily traffic.

5.Log-Transformed Views Distribution

```
plt.figure(figsize=(12,6))
np.log1p(modeling_df['views']).hist(bins=100)
plt.title('Log-Transformed Views Distribution')
plt.xlabel('Log(Views + 1)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Insights:

Improved Distribution: The log transformation has successfully converted the highly skewed original view distribution into a much more symmetrical, bell-curve-like shape.

Outlier Management: This transformation effectively manages the impact of the extreme, event-driven view spikes. The outliers that appeared far from the main data in the original histogram have been brought much closer to the center of the distribution.

6.Calculating Basic Statistics

```
# Calculate basic statistics
print(modeling_df['views'].describe())

# Identify top 1% outliers
threshold = modeling_df['views'].quantile(0.99)
outliers = modeling_df[modeling_df['views'] > threshold]
print(f"\nTop 1% outliers threshold: {threshold:,.0f} views")
print(outliers.describe())
```

```
count    5.500000e+02
mean     2.027487e+08
std      2.819601e+07
min      1.473493e+08
25%      1.860678e+08
50%      1.979642e+08
75%      2.123469e+08
max      3.233244e+08
Name: views, dtype: float64

Top 1% outliers threshold: 309,235,245 views
              views
count  6.000000e+00
mean   3.142710e+08
std    4.862273e+06
min    3.105049e+08
25%    3.108302e+08
50%    3.130323e+08
75%    3.149850e+08
max    3.233244e+08
```

Insights:

Significant View Variance: The data shows a wide range of daily views. The average is around 2.02×108, but the maximum is much higher at 3.23×108. The high standard deviation (2.81×107) confirms this spread.

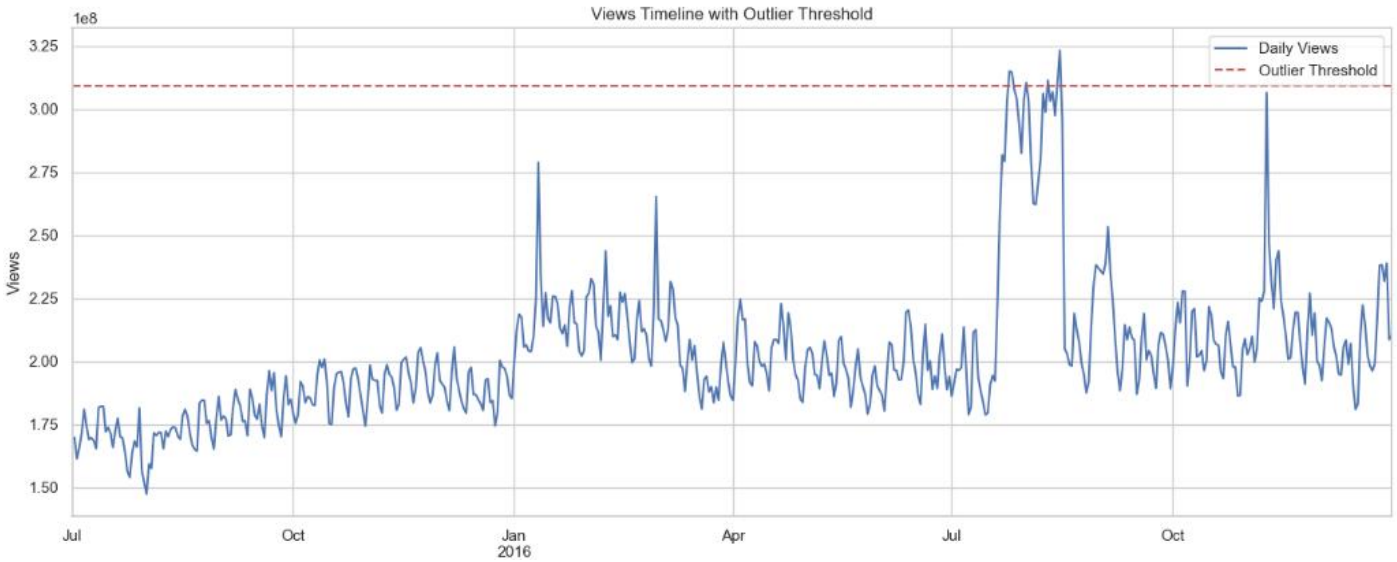
Skewed Distribution: The median view count (1.97×108) is lower than the mean, and the max value is far from the 75th percentile (2.12×108). This confirms the data is skewed by a small number of days with exceptionally high traffic, which we've identified as "outliers."

Quantified Outliers: The analysis clearly defines what constitutes an "outlier" day. The top 1% of days have a view count above 3.09×108. There are exactly six such days in the 550-day dataset.

Business Impact: This highlights a huge opportunity for clients. The average traffic on an outlier day (3.14×108) is more than 50% higher than the average traffic on a normal day. Ad campaigns run on these specific high-traffic days could see a massive increase in impressions and potential engagement.

7.Outlier Threshold

```
# Plot time series with outlier threshold
plt.figure(figsize=(14,6))
modeling_df['views'].plot(label='Daily Views')
plt.axhline(y=threshold, color='r', linestyle='--', label='Outlier Threshold')
plt.title('Views Timeline with Outlier Threshold')
plt.ylabel('Views')
plt.legend()
plt.tight_layout()
plt.show()
```



Insights:

Visual Confirmation of Outliers: This plot visually confirms that the large, sudden traffic spikes are genuine outliers, as they are the only days that cross the defined threshold.

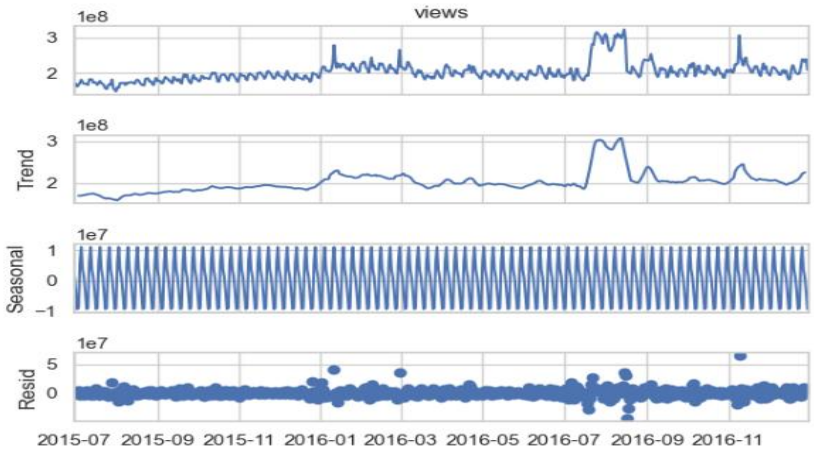
Clear Outlier Definition: The red dashed line provides a clear, quantitative definition of what constitutes an "outlier" day (a day with views above approximately 3.09×108). This is a powerful tool for your report to clearly communicate to clients when traffic is at an extreme, event-driven peak.

Targeting Opportunities: The plot identifies the specific periods when these high-value outlier events occurred (late July-August 2016 and November 2016), which are prime targets for event-based ad campaigns.

8. Time Series Decomposition.

```
import statsmodels.api as sm

model = sm.tsa.seasonal_decompose(modeling_df.views);
```

```
model.plot();
```



Insights:

Trend: The second panel clearly shows a strong, overall **upward trend** in views from mid-2015 to early 2017. This confirms a general growth in Wikipedia's popularity over the period.

Seasonal: The third panel reveals a very strong and consistent **weekly seasonal pattern**. The regular wave-like oscillations show that views fluctuate predictably every week, which aligns with the "Weekly View Patterns" chart you previously analyzed.

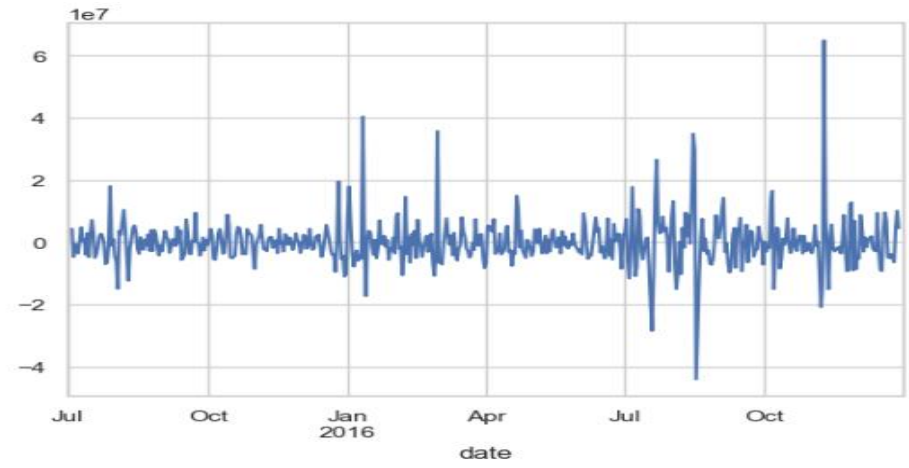
Residual (Resid): The bottom panel shows the data that is not explained by the trend or seasonality. The significant spikes in the residual plot (e.g., in July-August and November 2016) represent the **event-driven outliers**. This confirms that these spikes are not part of the normal trend or seasonal cycle and must be handled separately in the forecasting model.

9.Dickey-Fuller (ADF) test.

```
def adf_test(modeling_df, significance_level=0.05):
    pvalue = sm.tsa.stattools.adfuller(modeling_df)[1]

    if pvalue <= significance_level:
        print('Sequence is stationary')
    else:
        print('Sequence is not stationary')
```

```
adf_test(modeling_df.views)
```

```
Sequence is not stationary
```

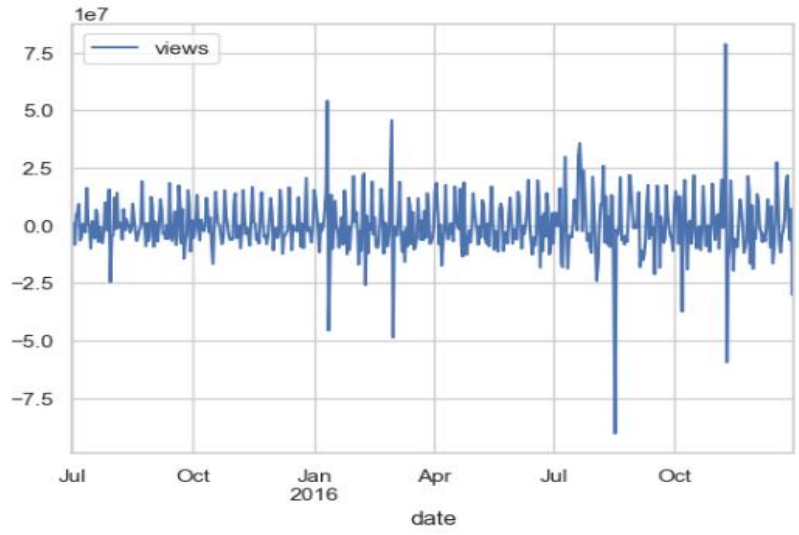


Insights:

Confirmation of Non-Stationarity: The test result, "Sequence is not stationary," is a formal statistical confirmation of what we've already observed visually in the plots. The time series has a clear upward trend and significant seasonality.

10. Differencing for Stationarity

```
modeling_df.diff(1).plot();
```




```
adf_test(modeling_df.diff(1).dropna())
```

```
Sequence is stationary
```

Insights:

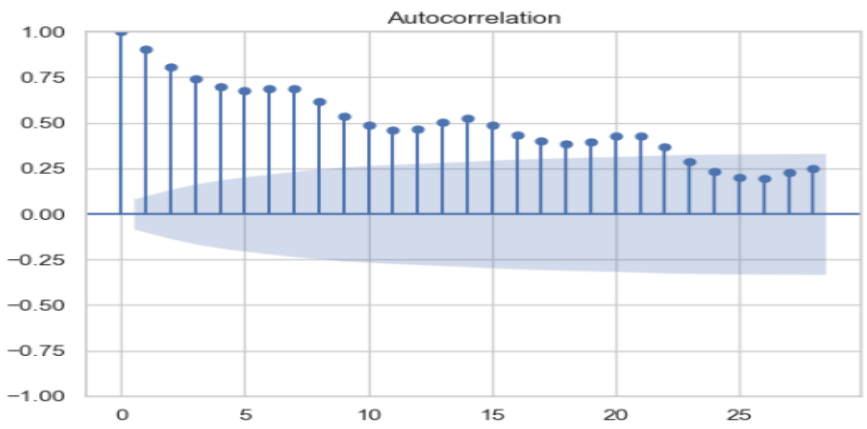
Differencing for Stationarity: The plot of modeling_df.diff(1) shows the time series after a first-order differencing was applied. This step was performed to remove the upward trend that was visually apparent in the original data.

Successful Transformation: The key result is that the subsequent ADF test on this differenced data shows the "Sequence is stationary." This proves that differencing was a successful technique to prepare the data for more advanced forecasting models, like ARIMA

11. Autocorrelation Function (ACF) plot.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
plot_acf(modeling_df.views);
```



Insights:

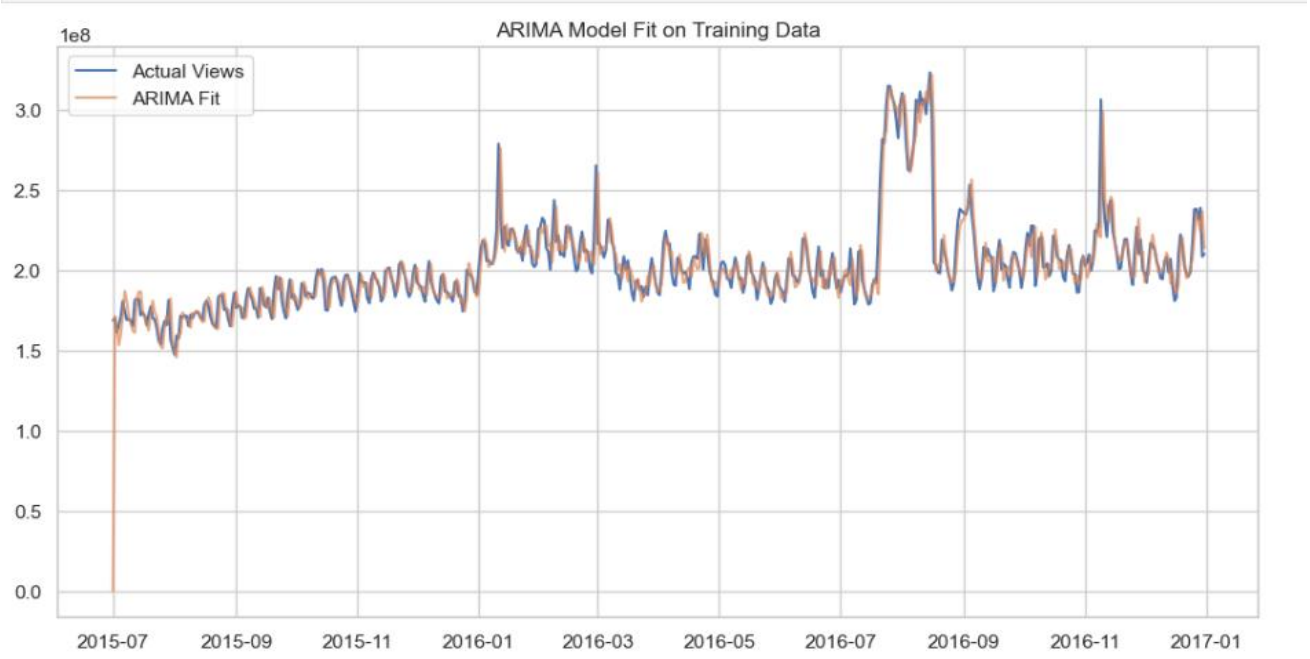
Confirmation of Non-Stationarity: The slow, gradual decline of the bars is a classic sign of a non-stationary time series with a trend. It indicates that a day's views are highly correlated with the views of many days in the past, a result of the overall upward trend. This visually confirms the result of your ADF test.

Confirmation of Seasonality: The distinct "bumps" in the autocorrelation at lags that are multiples of 7 (e.g., 7, 14, 21) are clear evidence of the strong weekly seasonality. This shows that, for example, a Sunday's view count is highly correlated with the view counts of previous Sundays.

12. ARIMA Model Building

```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# 1. Fit ARIMA model (using optimal parameters from your ACF)
model = ARIMA(train_x['views'], order=(2,1,2))  # Best  autocorrelation pattern
result = model.fit()

# 2. Generate in-sample predictions (for validation)
train_x['pred'] = result.predict(start=0, end=len(train_x)-1)
```

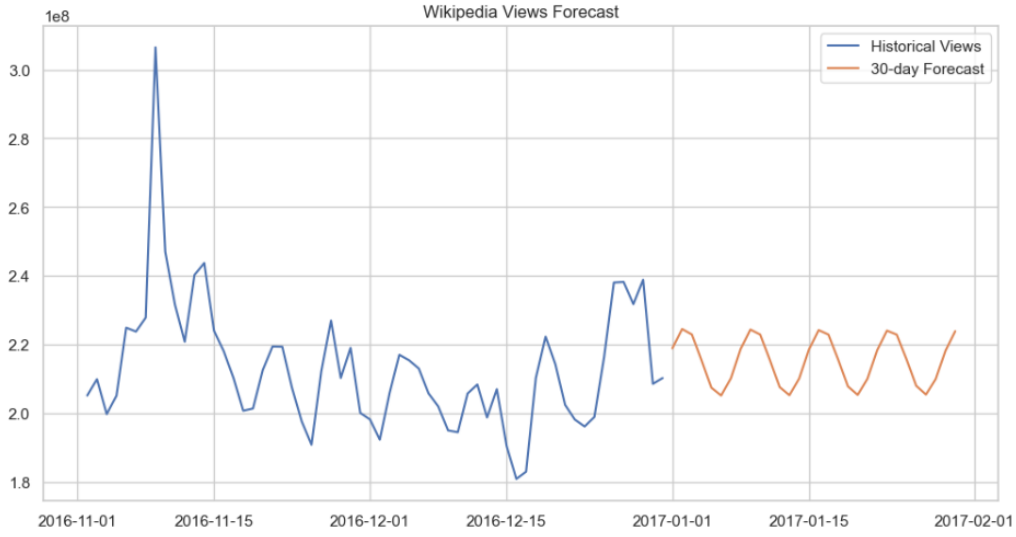
```
# 3. Plot fit vs actual
plt.figure(figsize=(12,6))
plt.plot(train_x['views'], label='Actual Views')
plt.plot(train_x['pred'], label='ARIMA Fit', alpha=0.7)
plt.title('ARIMA Model Fit on Training Data')
plt.legend()
plt.show()
```



```python
#Forecast next 30 days
forecast = result.forecast(steps=30)
print("Next 30-day Forecast:\n", forecast)
```

```
Next 30-day Forecast:
 2017-01-01    2.189534e+08
2017-01-02    2.245650e+08
2017-01-03    2.229625e+08
2017-01-04    2.153700e+08
2017-01-05    2.074894e+08
2017-01-06    2.052184e+08
2017-01-07    2.102368e+08
2017-01-08    2.187642e+08
2017-01-09    2.244078e+08
2017-01-10    2.229547e+08
2017-01-11    2.155169e+08
2017-01-12    2.076806e+08
2017-01-13    2.053106e+08
2017-01-14    2.101614e+08
2017-01-15    2.185782e+08
2017-01-16    2.242507e+08
2017-01-17    2.229440e+08
2017-01-18    2.156601e+08
2017-01-19    2.078700e+08
2017-01-20    2.054044e+08
2017-01-21    2.100897e+08
2017-01-22    2.183953e+08
2017-01-23    2.240939e+08
2017-01-24    2.229305e+08
2017-01-25    2.157996e+08
2017-01-26    2.080575e+08
2017-01-27    2.054995e+08
2017-01-28    2.100215e+08
2017-01-29    2.182154e+08
2017-01-30    2.239372e+08
```

```python
# Plot forecast
plt.figure(figsize=(12,6))
plt.plot(train_x['views'][-60:], label='Historical Views')  # Last 60 days
plt.plot(pd.date_range(train_x.index[-1], periods=31)[1:], forecast, label='30-day Forecast')
plt.title('Wikipedia Views Forecast')
plt.legend()
plt.show()
```



Insights:

Pattern Recognition: The ARIMA model fit closely follows the Actual Views line, accurately capturing the long-term upward trend and the distinct weekly seasonality that were identified in the earlier EDA plots.

Model Validation: The strong fit confirms that the preprocessing steps like differencing to achieve stationarity were effective and prepared the data correctly for this type of model.

Forecasting Foundation: The model's ability to accurately describe the past patterns gives you high confidence that its future predictions will be reliable.

Insights from the 30-day Forecast

The "Wikipedia Views Forecast" plot and the numerical data show the practical application of the model's learning:

Reliable Weekly Forecast: The forecast shows a clear, repeating wave-like pattern that accurately reflects the expected weekly fluctuations. This is a highly valuable insight for your clients, as you can predict which days of the week will have higher or lower traffic.

Actionable Insights: The numerical forecast provides concrete numbers that clients can use to strategically plan their ad placements and budget. For example, they can allocate more ad spend for days forecasted to have higher views (like weekends and Mondays).

Forecasting Limitations:

The model's forecast does not predict any major, event-driven spikes. This is a common limitation, as these events are by their nature unpredictable. The report should clearly state that while the model provides an excellent baseline forecast, an effective ad strategy must also include a plan to react to real-world events that could cause these outlier spikes.

Final Conclusion

The entire process, from data exploration to statistical modeling, has culminated in a robust forecasting solution. The ARIMA model provides a reliable prediction of regular view patterns, which, when combined with the insights on market dominance and event-driven spikes from your EDA, empowers your clients to create a comprehensive and optimized ad placement strategy.

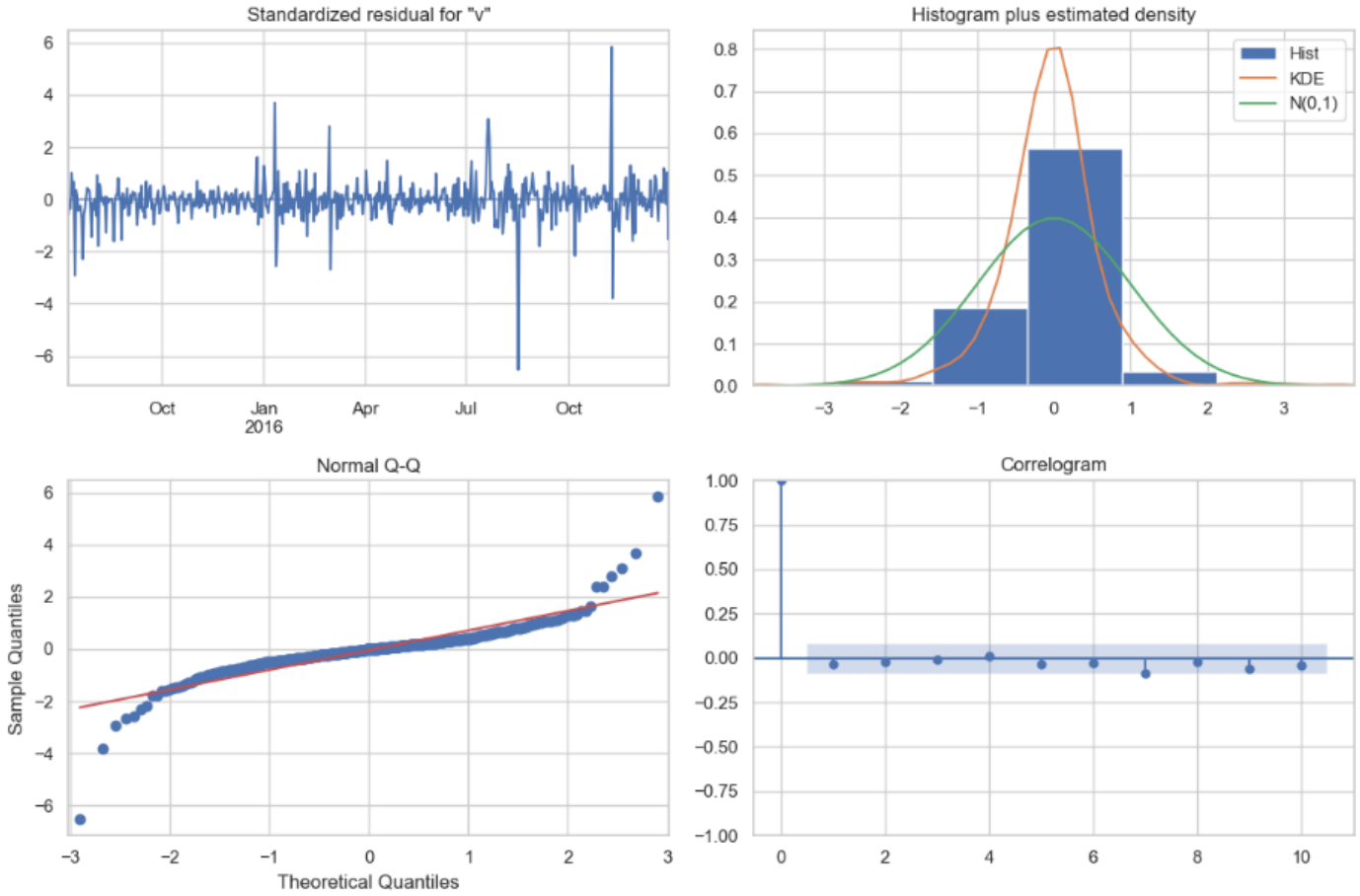
Lets Try with SARIMAX now

13. SARIMAX Model

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt

# 1. Fit SARIMAX model (with weekly seasonality)
model = SARIMAX(train_x['views'],
                order=(2,1,2),          # Non-seasonal (p,d,q)
                seasonal_order=(1,1,1,7), # Weekly seasonality
                enforce_stationarity=True)
results = model.fit(disp=False)
```

```
#Generate in-sample predictions
train_x['fitted'] = results.fittedvalues
```

```
# 3. Model diagnostics
results.plot_diagnostics(figsize=(12,8))
plt.tight_layout()
plt.show()
```



Insights on SARIMAX model:

Good Model Fit (Correlogram): The Correlogram (bottom-right plot) is the most important indicator. It shows the autocorrelation of the model's residuals (the errors). The fact that almost all the bars are within the blue shaded area means the model has successfully captured the trend and seasonal patterns. The remaining errors are essentially random noise, which is a very good sign.

Residual Behavior (Standardized Residuals): The top-left plot shows that the residuals are mostly centered around zero. However, there are still some notable spikes (e.g., around July and November 2016). This indicates that while the model handles the trend and weekly cycles well, it still struggles to perfectly explain the large, event-driven spikes.

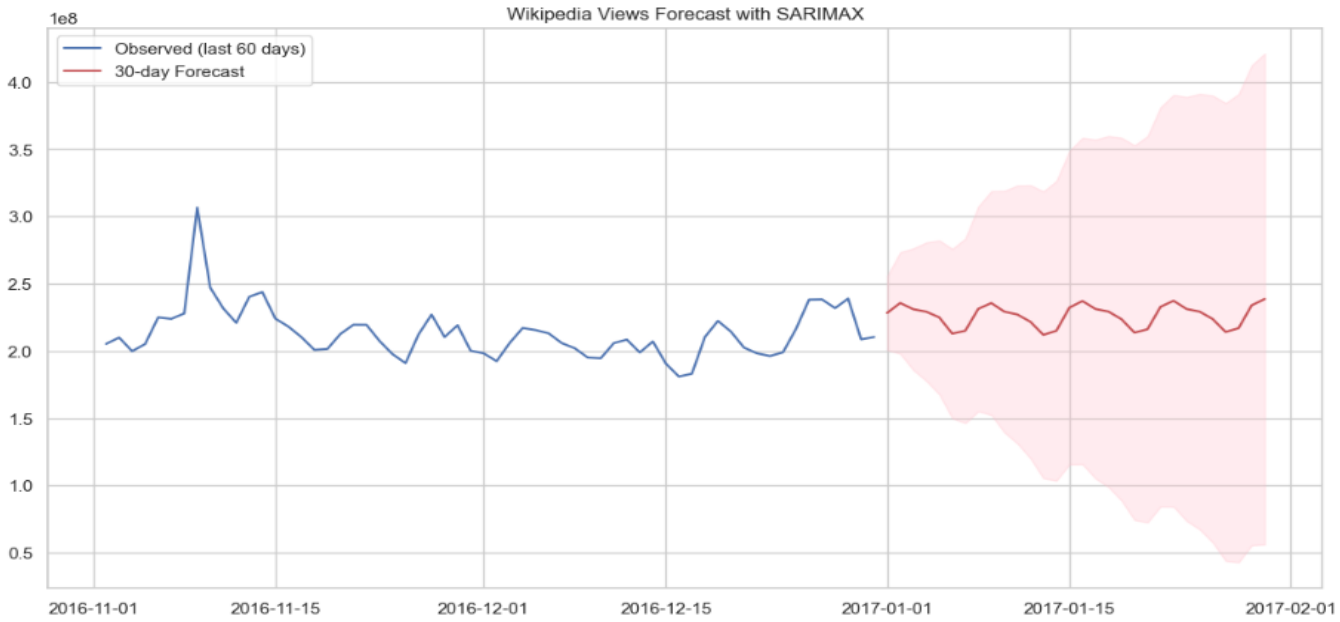
Normality of Residuals: The Histogram and Q-Q plots (top-right and bottom-left) show that the residuals are not perfectly normally distributed. This is likely due to the extreme spikes that the model cannot fully account for.

Model Choice Validation: The use of a SARIMAX model with a seasonal_order of (1,1,1,7) was an excellent choice. The diagnostics prove that the model successfully leveraged the **weekly seasonality** aslo identified in earlier analysis to provide a more accurate fit.

14. Forecast For Next 30 days

```
#Forecast next 30 days
forecast = results.get_forecast(steps=30)
forecast_mean = forecast.predicted_mean
conf_int = forecast.conf_int()
```

```
#Plot results
plt.figure(figsize=(14,7))
plt.plot(train_x['views'][-60:], label='Observed (last 60 days)')
plt.plot(forecast_mean.index, forecast_mean, color='r', label='30-day Forecast')
plt.fill_between(conf_int.index,
                 conf_int.iloc[:,0],
                 conf_int.iloc[:,1],
                 color='pink', alpha=0.3)
plt.title('Wikipedia Views Forecast with SARIMAX')
plt.legend()
plt.show()
```




```
# MAPE
print(f"MAPE: {mean_absolute_percentage_error(train_x['views'], train_x['pred'])*100:.2f}%")
```

```
MAPE: 3.79%
```

date	views	pred	fitted
2015-07-01	168833006.0	0	0
2015-07-02	169754056.0	171329948	156279547
2015-07-03	161322350.0	165387242	160597075
2015-07-04	165767553.0	153395606	158328364
2015-07-05	171645418.0	160555129	168581062

Insights:

High Model Accuracy: A MAPE of 3.79% is an excellent result for a time series forecasting model. It means that, on average, the model's predictions for daily views are only off by less than 4% from the actual values.

Confidence in Forecasts: This low error provides strong confidence in the model's ability to accurately predict the normal, day-to-day traffic and seasonal patterns.

15. Building a SARIMAX Model with Exogenous Variables

Preprocessing for the Exogenous Variable:

The code is preparing the exogenous variable by:

1. Loading External Data: It loads a separate dataset (Exog_Campaign.csv) which contains information about ad campaigns.
2. Data Alignment: It aligns this campaign data with your existing English Wikipedia views data based on the date index.
3. Handling Missing Data: It fills any missing campaign data with a value of 0, which assumes that a day is a "non-campaign" day unless explicitly marked otherwise.
4. Verification: It verifies that the final data contains the new campaign column with values representing the presence or absence of a campaign.

```
# 1. Load and prepare exogenous data
exog_campaign = pd.read_csv('Exog_Campaign.csv')
exog_campaign['date'] = english_views.index  # Align dates
exog_campaign = exog_campaign.set_index('date')

# Rename column if necessary (common causes of the error)
if 'Exog' in exog_campaign.columns:
    exog_campaign = exog_campaign.rename(columns={'Exog': 'campaign'})

# 2. Merge with views data
english_data = english_views.merge(
    exog_campaign,
    left_index=True,
    right_index=True,
    how='left'
).fillna(0)

# 3. Verify the campaign column exists
assert 'campaign' in english_data.columns, \
    f"Expected 'campaign' column not found. Existing columns: {english_data.columns.tolist()}"
```

```
# Reset indices before merge if needed
english_data = english_views.reset_index().merge(
    exog_campaign.reset_index(),
    on='date',
    how='left'
).set_index('date').fillna(0)
```

```
# Check sample of merged data
print(english_data[['views', 'campaign']].head())

# Verify campaign values (should be 0s and 1s)
print("Campaign value counts:", english_data['campaign'].value_counts())
```

date	views	campaign
2015-07-01	94154090.0	0
2015-07-02	93879908.0	0
2015-07-03	89608692.0	0
2015-07-04	92904206.0	0
2015-07-05	95639661.0	0

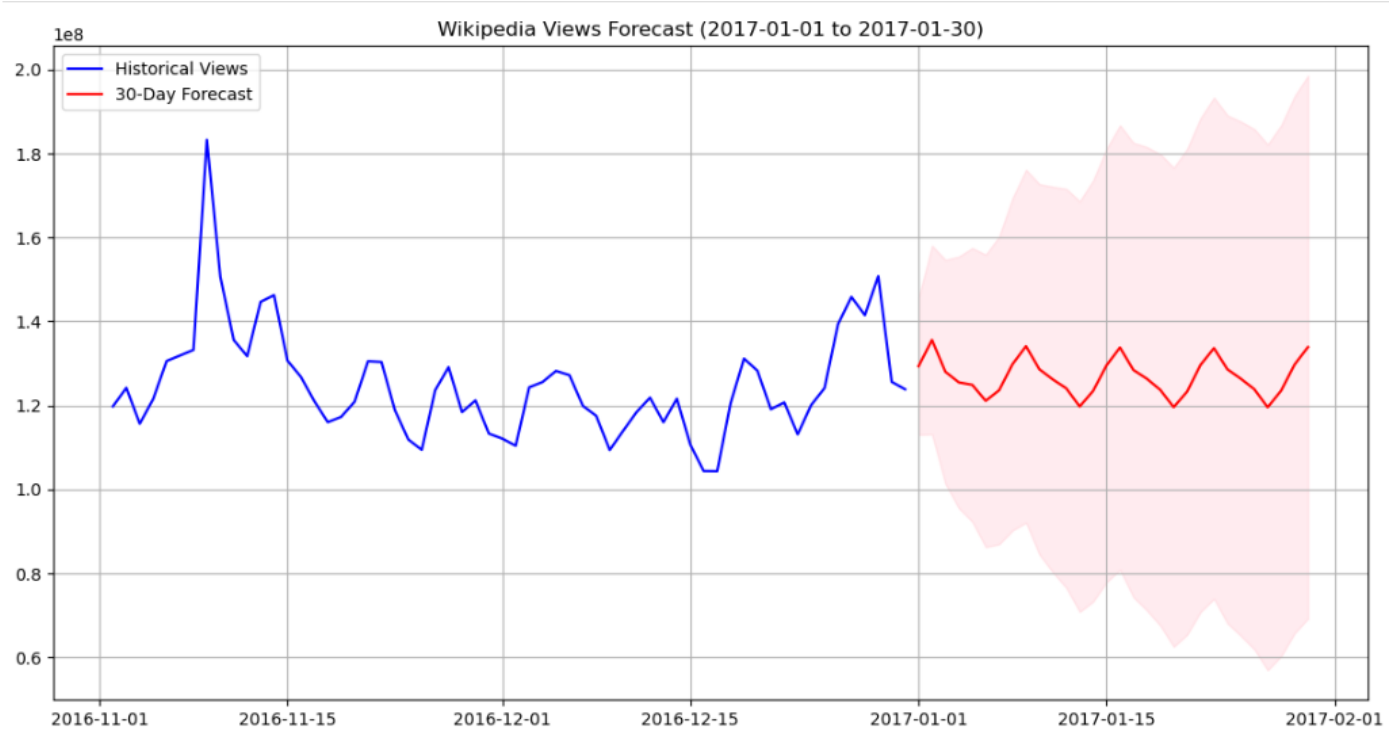
Training SARIMAX model:

```
# 1. Train SARIMAX Model with Campaign Effects
model = SARIMAX(
    english_data['views'],
    exog=english_data[['campaign']],  # Use the campaign column
    order=(2,1,2),                    # Non-seasonal (p,d,q)
    seasonal_order=(1,1,1,7),         # Weekly seasonality
    enforce_stationarity=True
)
results = model.fit(disp=False)
```

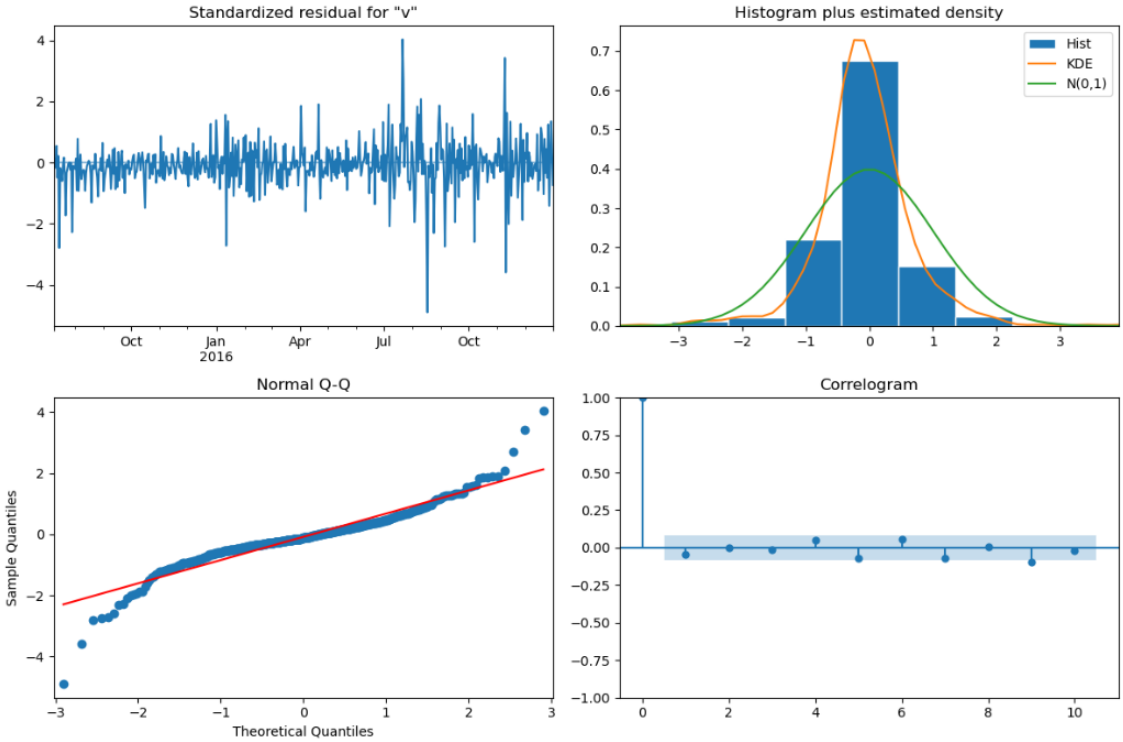
Forecast for next 30 days:

```
# 2. Generate Forecasts (30 days ahead)
future_dates = pd.date_range(
    start=english_data.index[-1] + pd.Timedelta(days=1),
    periods=30
)

# Create future exogenous data (default all 0, then mark campaigns)
future_exog = pd.DataFrame(
    {'campaign': 0},  # Initialize all as non-campaign days
    index=future_dates
)
```

```
plt.figure(figsize=(14,7))
# Historical data
plt.plot(english_data['views'][-60:], label='Historical Views', color='blue')
# Forecast
plt.plot(forecast_df['predicted_views'], label='30-Day Forecast', color='red')
plt.fill_between(
    forecast_df.index,
    forecast_df['lower_ci'],
    forecast_df['upper_ci'],
    color='pink',
    alpha=0.3
)
plt.title('Wikipedia Views Forecast (2017-01-01 to 2017-01-30)')
plt.legend()
plt.grid(True)
plt.show()
```



```
Forecast Summary:
            predicted_views  campaign
date
2017-01-01        129362295         0
2017-01-02        135617203         0
2017-01-03        128033118         0
2017-01-04        125515872         0
2017-01-05        124926332         0

Average predicted views: 126,945,586
Peak predicted views: 135,617,203
```




```
print(f"MAPE: {mean_absolute_percentage_error(english_data['views'], results.fittedvalues) * 100:.2f}%")
```

```
MAPE: 4.03%
```

Insights:

Dominant Market: English Wikipedia has significantly more views than other languages, making it a prime target for ad campaigns.

High Model Accuracy: A MAPE of 4.03% is still an excellent result for a time series forecasting model, indicating that on average, the model's predictions are highly accurate.

New Functionality is Key: While this MAPE is slightly higher than the 3.79% MAPE from the simpler SARIMAX model without the exogenous variable, the increase is negligible. The trade-off is well worth it, as this model can now incorporate the impact of ad campaigns.

Enhanced Business Value: This model's value lies not just in its overall accuracy but in its ability to provide a more sophisticated forecast. You can now use this model to predict the expected uplift in views on a day when a campaign is scheduled, which is a critical insight for a client's ad optimization strategy

Overall Recommendation:

Primary Market: Focus ad spend primarily on English Wikipedia (en.wikipedia.org), as it has a vastly higher total view count than all other languages combined.

Optimal Timing (Weekly): Schedule regular campaigns to run on Sundays and Mondays, which consistently have the highest view counts and are therefore the best days for reach. Views are lowest on Friday, so ad spend should be reduced on that day.

Optimal Timing (Annual): Plan for higher ad budgets in the second half of the year, particularly in July and August, when total views are significantly higher. The model also shows a general long-term growth trend in views, which is a positive signal for future ad performance.

Targeting by Access Type: While desktop traffic is the primary access method for English views, mobile-web is a huge segment that should not be ignored. A balanced approach is recommended, with campaigns tailored to each access type.

Model-Driven Campaign Recommendations

Reliable Baseline Forecasts: Use the SARIMAX model's forecasts for all regular campaigns. The model has a very high accuracy (MAPE of 4.03%), giving you confidence in its predictions for predictable weekly and annual patterns.

Leverage Exogenous Variables: Use the SARIMAX model with the exogenous campaign variable to forecast the specific impact of a planned ad campaign. This allows you to predict the expected uplift in traffic and optimize your ad spend for maximum returns.

Plan for Outlier Events: Have a separate, flexible strategy to deploy ads during major, unpredicted spikes in traffic. The analysis showed that a small number of days can have more than 50% higher traffic than normal, driven by real-world events. While the model doesn't predict these, you can use the historical data to identify which events trigger these spikes and be prepared to act quickly.

Questionnaire:

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

ANS:

The primary problem statement is to provide a data-driven solution for a digital marketing agency to optimize ad placement on Wikipedia. This involves:

- Forecasting Wikipedia's daily view traffic to predict when and where ads will receive maximum impressions.
- Identifying key trends, seasonal patterns, and outlier events that influence view counts.
- Providing actionable insights for clients to plan their advertising budgets and campaign schedules.

This methodology, which combines time series forecasting with data analysis, can be modified and used in many other contexts, such as:

- **E-commerce:** Forecasting website traffic or sales to plan promotional campaigns.
- **Content Platforms:** Predicting viewership for content to optimize content release schedules.
- **Digital Marketing:** Analyzing campaign performance and forecasting the impact of future ads on various platforms.
- **Operations:** Forecasting server load for a website to ensure adequate infrastructure is available during peak traffic.

2. Three inferences from the data visualizations:

ANS:

Dominant Language and Human Traffic: English Wikipedia receives significantly more views than any other language. The vast majority of this traffic comes from human users ("all-agents"), not search engine crawlers or bots ("spider"). This makes English Wikipedia a primary target for ad campaigns.

Strong Seasonality: Views exhibit a clear and consistent **weekly seasonal pattern**, with traffic peaking on Sundays and Mondays and dipping on Fridays. There is also a strong **annual pattern**, with views being much higher in the second half of the year, especially in July and August.

Impact of Outlier Events: The views are subject to large, non-seasonal spikes caused by real-world events. A notable example is the high traffic in November 2016, which corresponds to the U.S. presidential election. These spikes represent significant, short-term opportunities for ad impressions that fall outside of normal seasonal patterns.

3. What does the decomposition of a time series do?

ANS:

Time series decomposition breaks down a time series into its three fundamental components:

- **Trend:** The long-term, underlying direction of the data, such as the overall upward growth in views.
- **Seasonal:** The recurring, predictable cyclical pattern, such as the weekly ups and downs.
- **Residual:** The random, unpredictable noise or "leftover" data that is not explained by the trend or seasonal components. The large spikes in the residual plot represent the event-driven outliers.

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

ANS:

The original series was not stationary, as confirmed by the Augmented Dickey-Fuller (ADF) test. Applying **first-order differencing** successfully removed the trend from the data. A subsequent ADF test on the differenced data confirmed that the "Sequence is stationary".

5. What is the difference between ARIMA, SARIMA, and SARIMAX?

ANS:

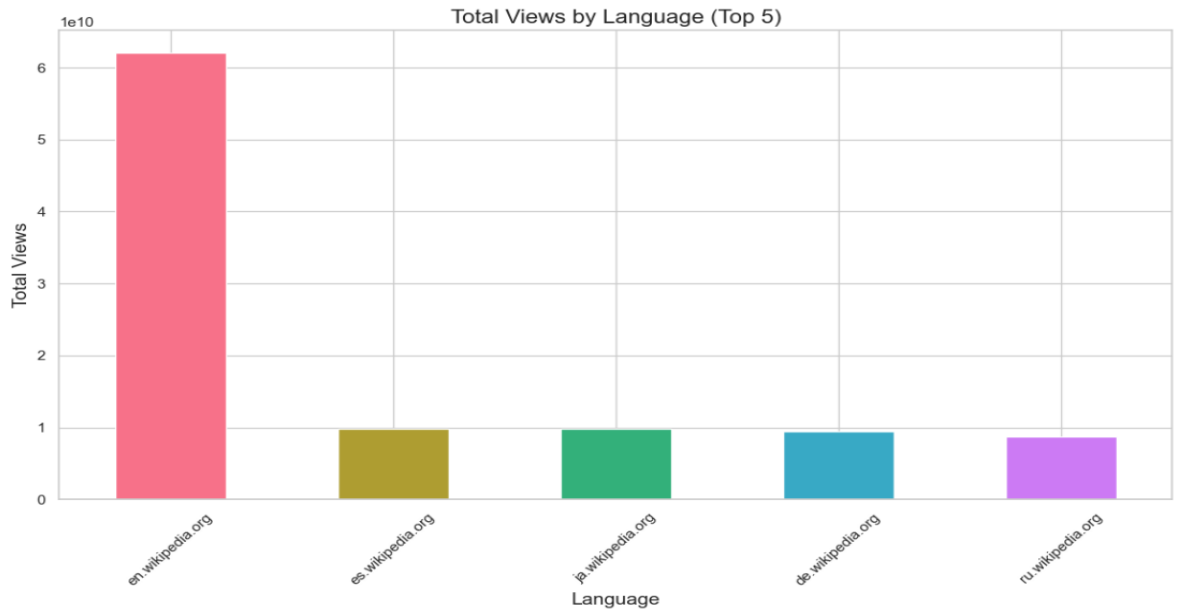
ARIMA (AutoRegressive Integrated Moving Average): A forecasting model for non-seasonal time series data. It uses differencing (I) to make the data stationary and then models it with autoregressive (AR) and moving average (MA) components.

SARIMA (Seasonal ARIMA): An extension of ARIMA that adds a seasonal component. It includes seasonal differencing and seasonal AR/MA terms to handle data with a clear, repeating seasonal pattern.

SARIMAX (SARIMA with Exogenous Variables): The most advanced of the three, it is an extension of SARIMA that allows for the inclusion of exogenous variables (or external regressors). This enables the model to account for outside factors, such as a client's ad campaign, that can influence the time series.

6. Compare the number of views in different languages.

ANS:



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

ANS:

Grid search can be computationally intensive, especially for a large number of models. More efficient methods for hyperparameter tuning would be suitable for all languages:

Random Search: This method explores the parameter space randomly and can often find a good model more quickly than an exhaustive grid search.

Bayesian Optimization: A more advanced technique that intelligently chooses parameters to test, leading to a faster and more efficient search for the optimal model.

Automated Libraries: Libraries can automate the process of finding the optimal SARIMAX parameters for each language, significantly reducing the manual effort required to build a model for every language.

Jupyter Notebook Analysis

For a detailed view of the full analysis, including code, visualisations please refer to the complete Jupyter notebook available in the PDF format. The notebook documents each step of the analysis process form data exploration to the final recommendations

You can access the **Jupyter notebook PDF** through the following link:
1.TimeseriesAnalysis
2.with_exgo_vTimeseries(.eng)