About

- Ad Ease is an ads and marketing based company helping businesses elicit maximum clicks @ minimum cost. AdEase is an ad infrastructure to help businesses promote themselves easily, effectively, and economically. The interplay of 3 AI modules - Design, Dispense, and Decipher, come together to make it this an end-to-end 3 step process digital advertising solution for all.
- You are working in the Data Science team of Ad ease trying to understand the per page view report for different wikipedia pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients. 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.

```
import pandas as pd
```

```
train_path = "/content/train_1.csv"
df=pd.read_csv(train_path)
df.head(3)
```

	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	2015-07-10	2015-07-11	2015-07-12	2015-07-13	2015-07-14	2015-07-1
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	24.0	19.0	10.0	14.0	15.0	8.
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	4.0	41.0	65.0	57.0	38.0	20.
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	1.0	1.0	1.0	6.0	8.

3 rows × 551 columns

```
!pip install pmdarima prophet --quiet

import numpy as np
import pandas as pd
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.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

from prophet import Prophet
from sklearn.metrics import mean_absolute_percentage_error

plt.rcParams['figure.figsize'] = (12,5)
pd.set_option('display.max_columns', 50)
```

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

train_path = '/content/train_1.csv'  # corrected path
# exog_path  = '/content/drive/MyDrive/AdEase/Exog_Campaign_eng.csv'  # This file was not found and has been commented out

train = pd.read_csv(train_path)
# exog = pd.read_csv(exog_path) # This line was commented out due to FileNotFoundError
train.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

	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	2015-07-10	2015-07-11	2015-07-12	2015-07-13
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	24.0	19.0	10.0	14.0
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	4.0	41.0	65.0	57.0
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	4.0	1.0	1.0	1.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	16.0	11.0	23.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 551 columns

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

# Check nulls
null_counts = train.isna().sum().sort_values(ascending=False)
null_counts.head(20)
```

```
(128845, 551)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128845 entries, 0 to 128844
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 541.6+ MB
None
```

	0
2015-07-02	19389
2015-07-01	19314
2015-07-07	19248
2015-07-05	19237
2015-07-04	19225
2015-07-03	19125
2015-07-11	19117
2015-07-12	19086
2015-07-06	19072
2015-07-13	19011
2015-07-10	18941
2015-07-18	18915
2015-07-08	18877
2015-07-09	18840
2015-07-19	18759
2015-07-14	18739
2015-07-15	18710
2015-07-17	18664
2015-07-16	18603
2015-07-20	18594

dtype: int64

```
train_path = "/content/train_1.csv"
print(train_path)
```

```
/content/train_1.csv
```

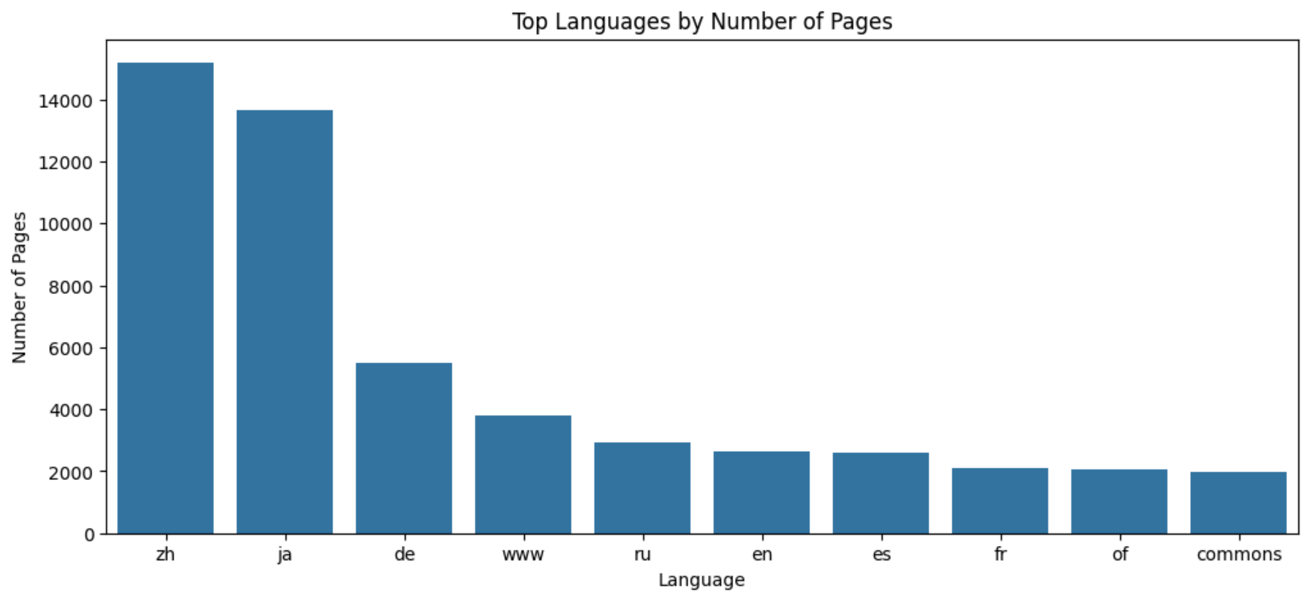
```
# Replace NaN with 0 (common for page-views)
train_filled = train.fillna(0)

# Check date columns (all columns except 'Page')
date_cols = train_filled.columns[1:]
print(date_cols[:5])
```

```
Index(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04', '2015-07-05'], dtype='object')
```

```
def split_page_metadata(df):
    parts = df['Page'].str.split('_', n=3, expand=True)
    df['title'] = parts[0]
    df['lang_full'] = parts[1]  # like en.wikipedia.org
    df['access_type'] = parts[2]
    df['access_origin'] = parts[3]

    # language like 'en', 'es', 'fr'
    df['language'] = df['lang_full'].str.split('.', expand=True)[0]
    return df

train_meta = split_page_metadata(train_filled.copy())
train_meta[['Page', 'title', 'language', 'access_type', 'access_origin']].head()
```

	Page	title	language	access_type	access_origin
0	2NE1_zh.wikipedia.org_all-access_spider	2NE1	zh	all-access	spider
1	2PM_zh.wikipedia.org_all-access_spider	2PM	zh	all-access	spider
2	3C_zh.wikipedia.org_all-access_spider	3C	zh	all-access	spider
3	4minute_zh.wikipedia.org_all-access_spider	4minute	zh	all-access	spider
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	52	Hz	I	Love_You_zh.wikipedia.org_all-access_spider

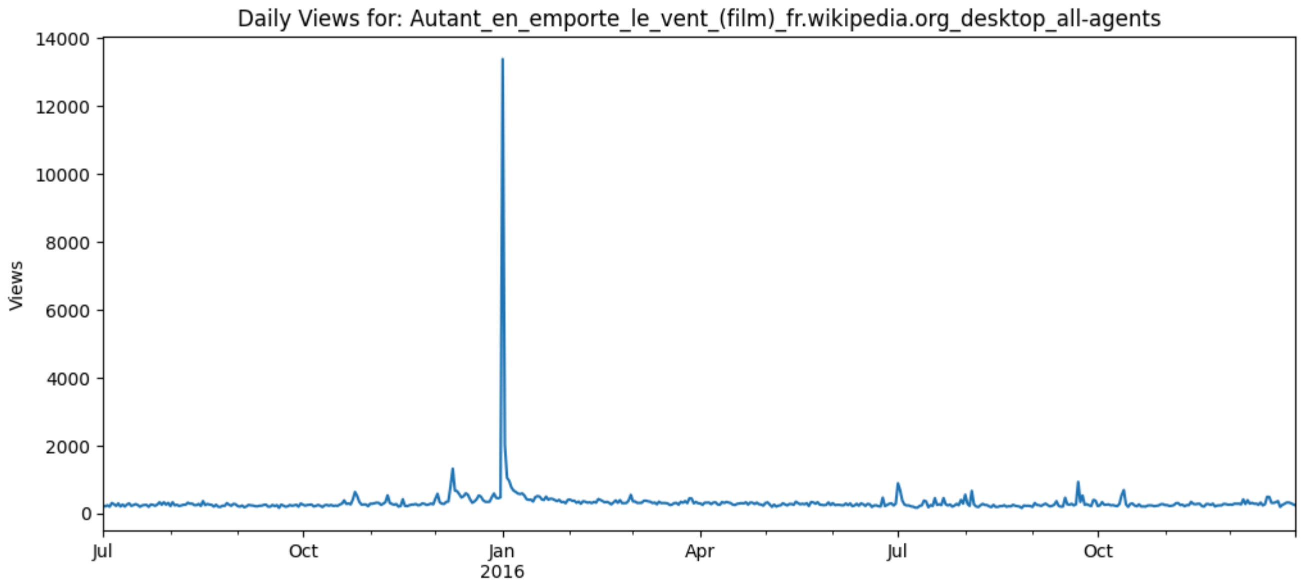
```
lang_counts = train_meta['language'].value_counts().head(10)
sns.barplot(x=lang_counts.index, y=lang_counts.values)
plt.title('Top Languages by Number of Pages')
plt.xlabel('Language')
plt.ylabel('Number of Pages')
plt.show()
```



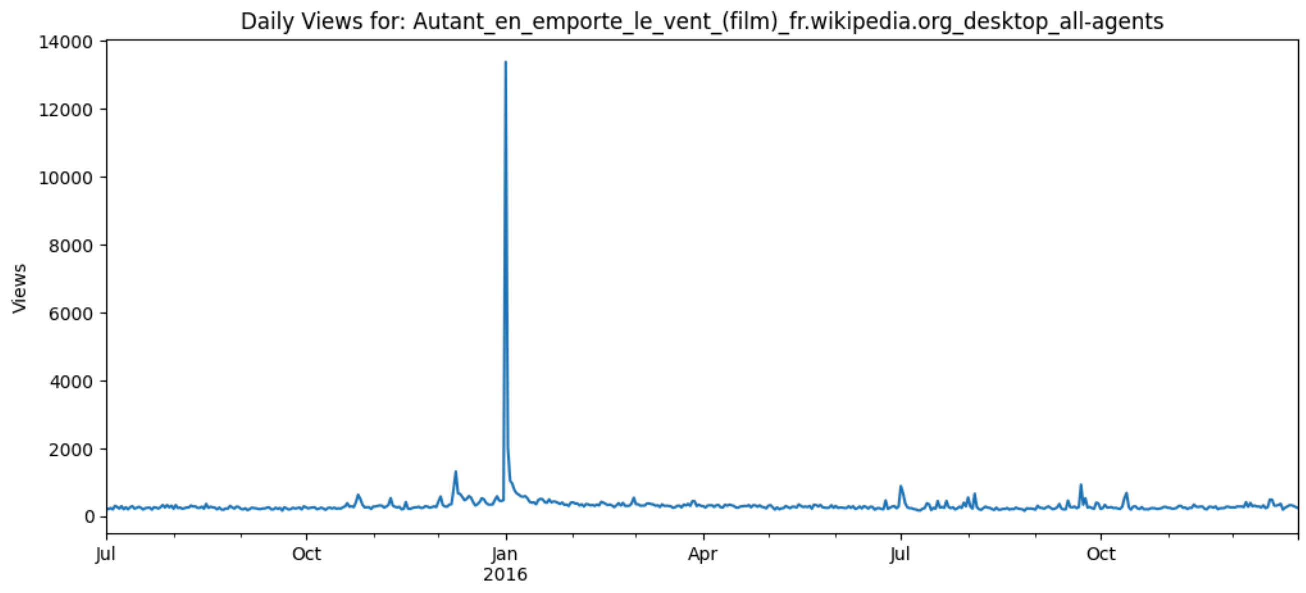
```
# Example: pick one English page
sample_page = train_meta[train_meta['language'] == 'en'].iloc[0]
sample_page_name = sample_page['Page']
sample_page_name
```

```
'Autant_en_emporte_le_vent_(film)_fr.wikipedia.org_desktop_all-agents'
```

```
def get_series_for_page(df, page_name):
    row = df[df['Page'] == page_name]
    # Select columns from index 1 up to, but not including, the last 5 metadata columns
    ts = row.iloc[0, 1:-5]
    ts.index = pd.to_datetime(ts.index)
    ts = ts.astype(float)
    ts = ts.sort_index()
    return ts

y = get_series_for_page(train_meta, sample_page_name)

y.plot()
plt.title(f"Daily Views for: {sample_page_name}")
plt.ylabel("Views")
plt.show()
```



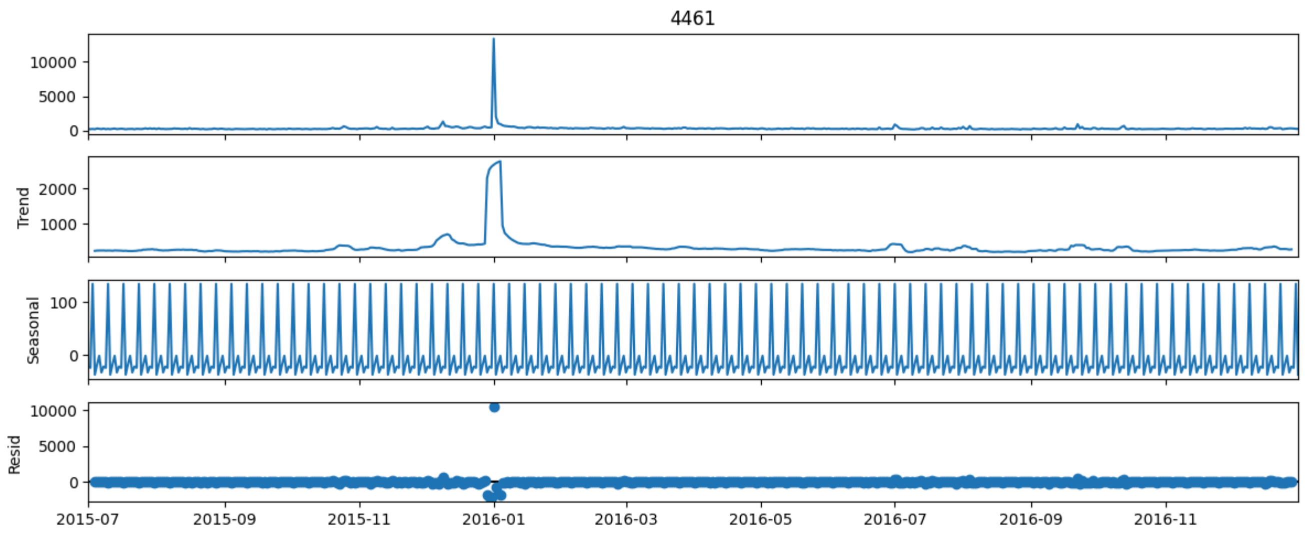
```
def get_series_for_page(df, page_name):
    row = df[df['Page'] == page_name]
    # Select columns from index 1 up to, but not including, the last 5 metadata columns
    ts = row.iloc[0, 1:-5]
    ts.index = pd.to_datetime(ts.index)
    ts = ts.astype(float)
    ts = ts.sort_index()
    return ts

y = get_series_for_page(train_meta, sample_page_name)

y.plot()
plt.title(f"Daily Views for: {sample_page_name}")
plt.ylabel("Views")
plt.show()
```



```
def adf_test(series, title=''):
    print(f'ADF Test: {title}')
    result = adfuller(series.dropna(), autolag='AIC')
    labels = ['ADF Statistic', 'p-value', '# Lags Used', 'Number of Observations']
    out = dict(zip(labels, result[0:4]))
    for k, v in out.items():
        print(f'{k}: {v}')
    for key, value in result[4].items():
        print('Critical Value (%s): %.3f' % (key, value))
    if result[1] <= 0.05:
        print("=> Stationary (reject H0)")
    else:
        print("=> Non-stationary (fail to reject H0)")

adf_test(y, 'Original Series')
```

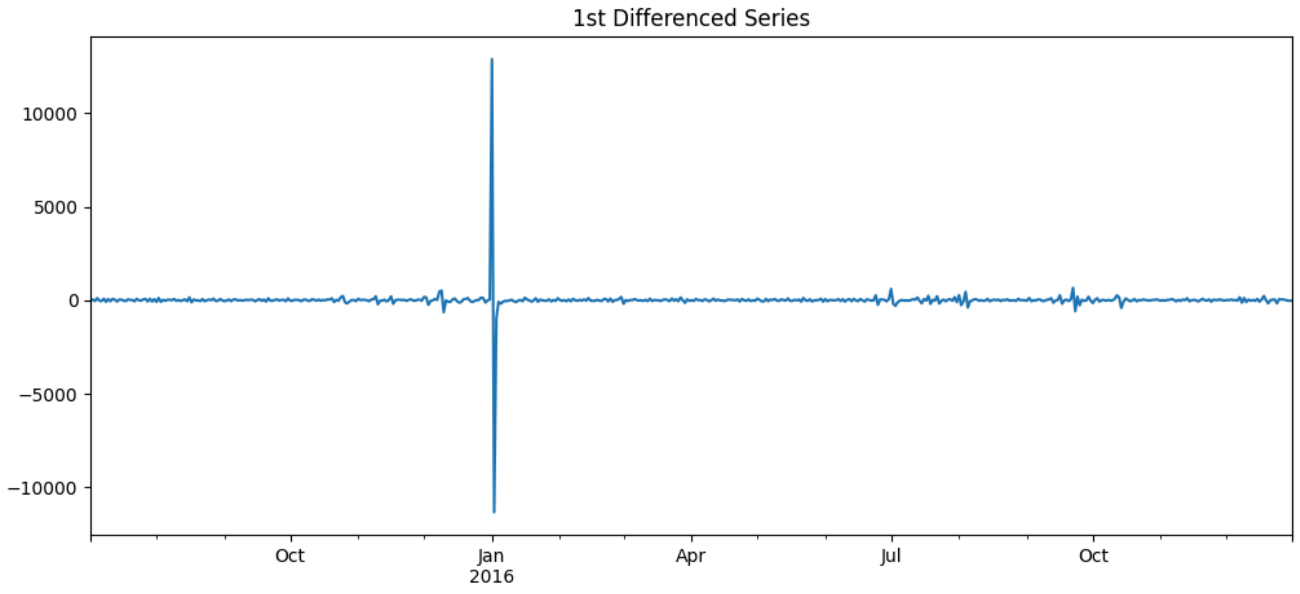
```
ADF Test: Original Series
ADF Statistic: -14.152202135207999
p-value: 2.152104295827734e-26
# Lags Used: 1
Number of Observations: 548
Critical Value (1%): -3.442
Critical Value (5%): -2.867
Critical Value (10%): -2.570
=> Stationary (reject H0)
```

```
decomp = seasonal_decompose(y, model='additive', period=7)  # weekly seasonality guess
decomp.plot()
plt.show()
```

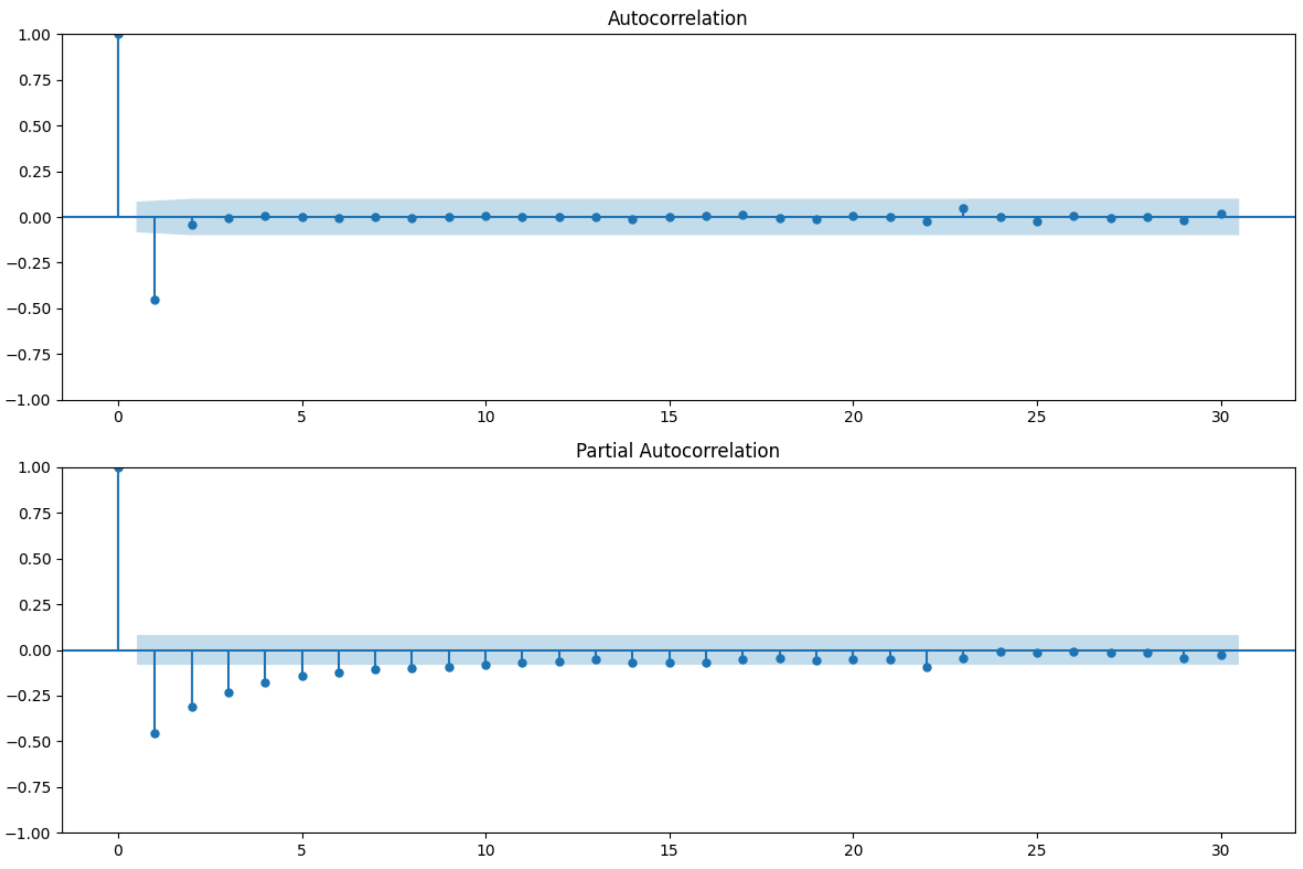


```
y_diff1 = y.diff().dropna()
adf_test(y_diff1, '1st Difference')

plt.figure()
y_diff1.plot()
plt.title('1st Differenced Series')
plt.show()
```

```
ADF Test: 1st Difference
ADF Statistic: -9.64254166746115
p-value: 1.5074080756932696e-16
# Lags Used: 15
Number of Observations: 533
Critical Value (1%): -3.443
Critical Value (5%): -2.867
Critical Value (10%): -2.570
=> Stationary (reject H0)
```



```
fig, ax = plt.subplots(2, 1, figsize=(12,8))
plot_acf(y_diff1, lags=30, ax=ax[0])
plot_pacf(y_diff1, lags=30, ax=ax[1])
plt.tight_layout()
plt.show()
```



```
train_size = int(len(y) * 0.8)
y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]
len(y_train), len(y_test)
```

```
(440, 110)
```

```
model = ARIMA(y_train, order=(2,1,2))  # example
model_fit = model.fit()
print(model_fit.summary())

# Forecast
n_test = len(y_test)
arima_forecast = model_fit.forecast(steps=n_test)
```

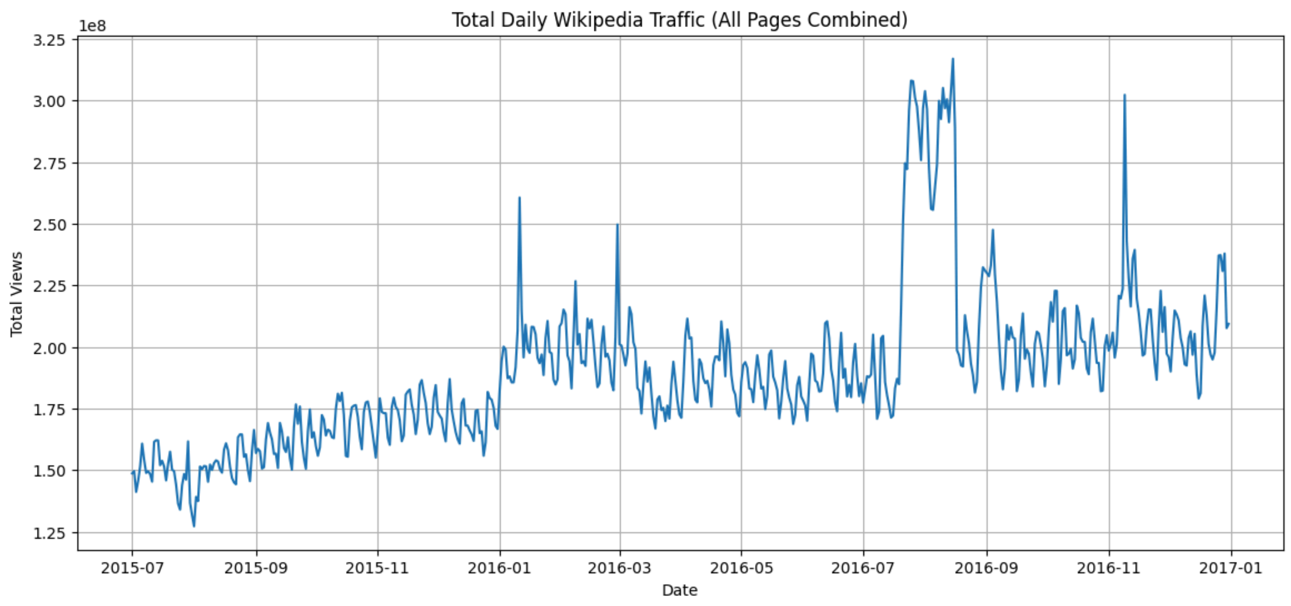
```
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization faile
  warnings.warn("Maximum Likelihood optimization failed to "
                               SARIMAX Results
==============================================================================
Dep. Variable:                   4461   No. Observations:                  440
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -3452.701
Date:                Sun, 07 Dec 2025   AIC                           6915.402
Time:                        07:29:49   BIC                           6935.824
Sample:                    07-01-2015   HQIC                          6923.459
                         - 09-12-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0751      0.238      4.519      0.000       0.609       1.541
ar.L2         -0.1047      0.131     -0.798      0.425      -0.362       0.152
ma.L1         -1.9226      0.220     -8.757      0.000      -2.353      -1.492
ma.L2          0.9228      0.219      4.207      0.000       0.493       1.353
sigma2      3.942e+05   5.92e-06   6.65e+10      0.000    3.94e+05    3.94e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):           3103083.72
Prob(Q):                              0.91   Prob(JB):                         0.00
Heteroskedasticity (H):               2.77   Skew:                            20.02
Prob(H) (two-sided):                  0.00   Kurtosis:                       412.93
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.76e+26. Standard errors may be unstable.
```

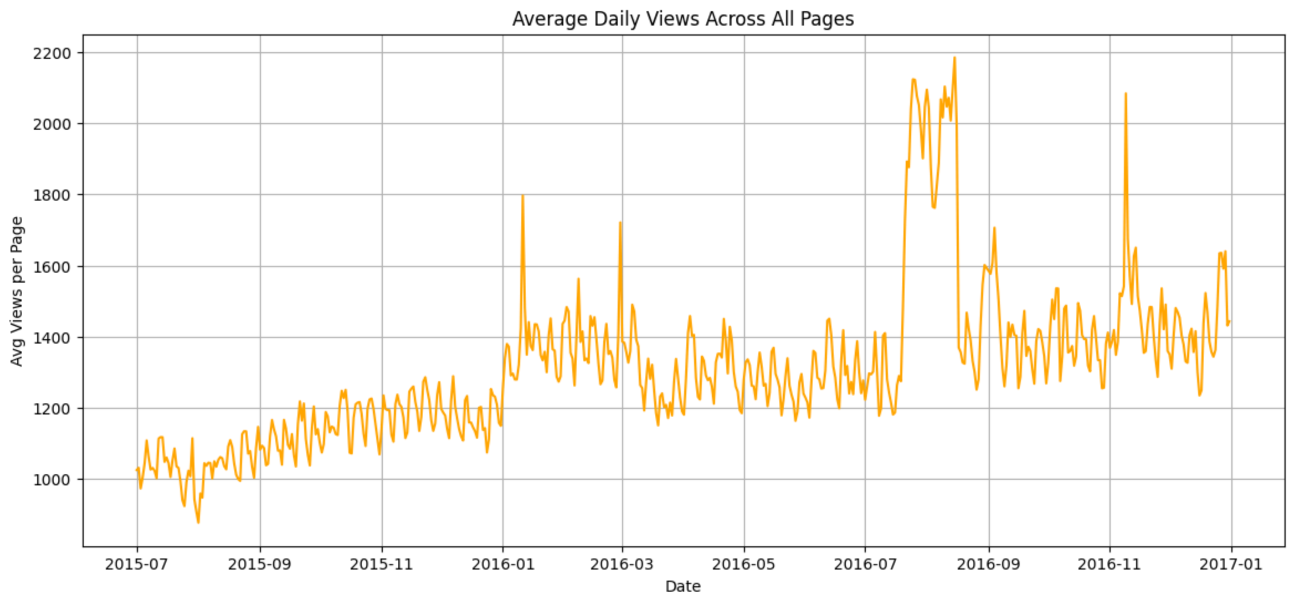
```
def mape(y_true, y_pred):
    return mean_absolute_percentage_error(y_true, y_pred) * 100

print("ARIMA MAPE:", mape(y_test, arima_forecast))
```

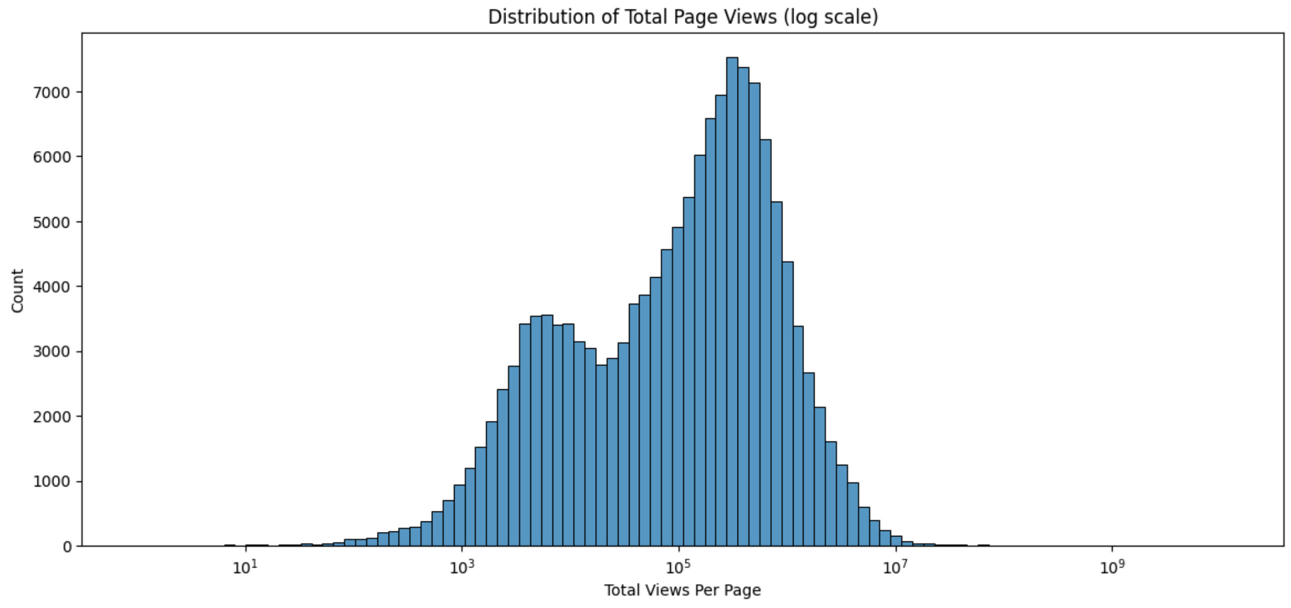
```
ARIMA MAPE: 25.745479667119948
```

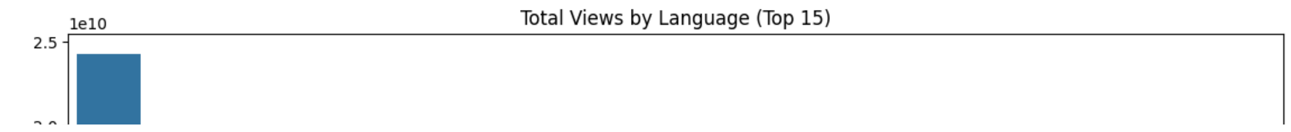
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['figure.figsize'] = (14,6)

# ---- Load your dataset ----
train_path = "/content/train_1.csv"  # update as needed
df = pd.read_csv(train_path)

# Fill nulls with 0 because missing = no page created / no views
df = df.fillna(0)

# Separate metadata
def split_page_metadata(df):
    parts = df['Page'].str.split('_', n=3, expand=True)
    df['title'] = parts[0]
    df['lang_full'] = parts[1]
    df['access_type'] = parts[2]
    df['access_origin'] = parts[3]
    df['language'] = df['lang_full'].str.split('.', expand=True)[0]
    return df

df = split_page_metadata(df)

# Identify date columns
date_cols = df.columns[df.columns.str.match(r'\d{4}-\d{2}-\d{2}')]

# ------------------------------------------
# 1️⃣ TOTAL DAILY VIEWS ACROSS ALL PAGES
# ------------------------------------------
daily_total = df[date_cols].sum(axis=0)
daily_total.index = pd.to_datetime(daily_total.index)

plt.plot(daily_total)
plt.title("Total Daily Wikipedia Traffic (All Pages Combined)")
plt.xlabel("Date")
plt.ylabel("Total Views")
plt.grid(True)
plt.show()

# ------------------------------------------
# 2️⃣ AVERAGE DAILY VIEWS (ANOTHER PERSPECTIVE)
# ------------------------------------------
daily_avg = df[date_cols].mean(axis=0)
daily_avg.index = pd.to_datetime(daily_avg.index)

plt.plot(daily_avg, color='orange')
plt.title("Average Daily Views Across All Pages")
plt.xlabel("Date")
plt.ylabel("Avg Views per Page")
plt.grid(True)
plt.show()
```

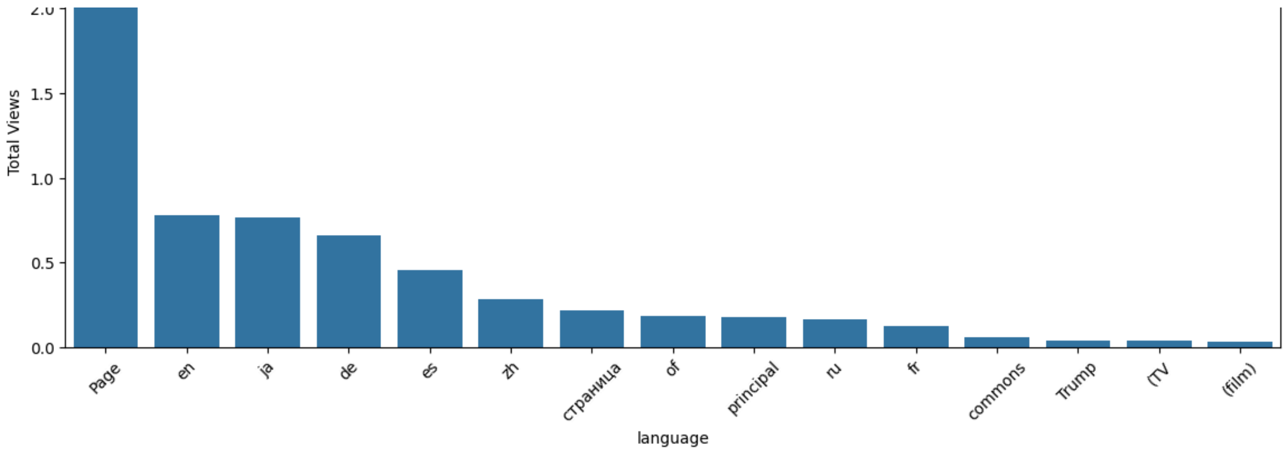
```
# ------------------------------------------
# 3️⃣ DISTRIBUTION OF PAGE VIEW TOTALS
# ------------------------------------------
df['total_views'] = df[date_cols].sum(axis=1)

sns.histplot(df['total_views'], bins=100, log_scale=True)
plt.title("Distribution of Total Page Views (log scale)")
plt.xlabel("Total Views Per Page")
plt.ylabel("Count")
plt.show()

# ------------------------------------------
# 4️⃣ LANGUAGE-WISE TOTAL VIEWS
# ------------------------------------------
lang_traffic = df.groupby("language")['total_views'].sum().sort_values(ascending=False)

plt.figure(figsize=(14,5))
sns.barplot(x=lang_traffic.index[:15], y=lang_traffic.values[:15])
plt.title("Total Views by Language (Top 15)")
plt.xticks(rotation=45)
plt.ylabel("Total Views")
plt.show()

# ------------------------------------------
# 5️⃣ TIME SERIES FOR A RANDOM PAGE
# ------------------------------------------
sample_page = df.sample(1).iloc[0]
series = sample_page[date_cols].astype(float).values
dates = pd.to_datetime(date_cols)

plt.plot(dates, series)
plt.title(f"Daily Views for Random Page: {sample_page['Page']}")
plt.xlabel("Date")
plt.ylabel("Views")
plt.grid(True)
plt.show()

# ------------------------------------------
# 6️⃣ ROLLING MEAN + VARIANCE FOR SAME PAGE
# ------------------------------------------
ts = pd.Series(series, index=dates)

roll_mean = ts.rolling(window=30).mean()
roll_std = ts.rolling(window=30).std()

plt.plot(ts, label="Original")
plt.plot(roll_mean, label="30-day Rolling Mean")
plt.plot(roll_std, label="30-day Rolling Std")
plt.legend()
plt.title("Rolling Mean & Standard Deviation (Stationarity Check)")
plt.show()

# ------------------------------------------
# 7️⃣ WEEKLY SEASONALITY
# ------------------------------------------
df_weekly = daily_total.groupby(daily_total.index.dayofweek).mean()

plt.bar(['Mon','Tue','Wed','Thu','Fri','Sat','Sun'], df_weekly.values)
plt.title("Weekly Seasonality Pattern (Average Views per Day of Week)")
plt.ylabel("Avg Views")
plt.show()

# ------------------------------------------
# 8️⃣ CORRELATION HEATMAP OF SELECTED DAYS
# ------------------------------------------
sample_days = df[date_cols].iloc[:, :60]  # first 60 days

corr = sample_days.corr()

plt.figure(figsize=(12,8))
sns.heatmap(corr, cmap="coolwarm", cbar=False)
plt.title("Correlation Heatmap of Daily Views (First 60 Days)")
plt.show()

# ------------------------------------------
# 9️⃣ VISUALIZE MISSING VALUE PATTERN
# ------------------------------------------
df_null = (df[date_cols] == 0).sum()
df_null = (df[date_cols] == 0).sum()

plt.plot(df_null.index, df_null.values)
plt.title("Number of Pages with Zero Views per Day")
plt.xlabel("Date")
plt.ylabel("Count of pages with 0 views")
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

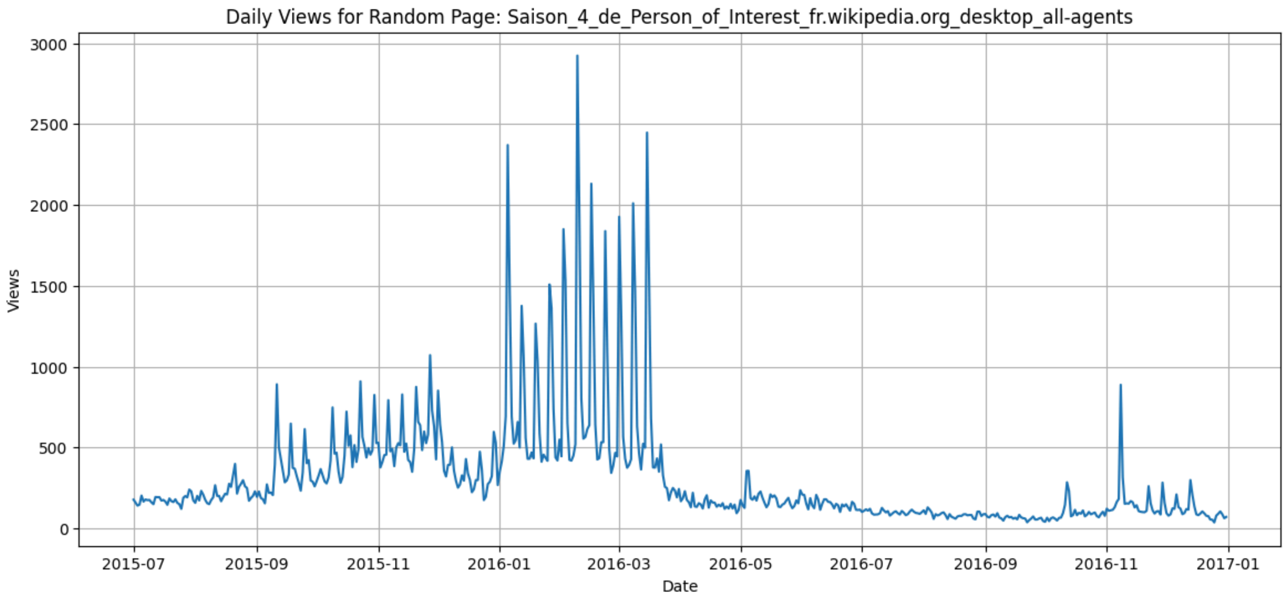

Total Daily Wikipedia Traffic (All Pages Combined)
1e8
Total Views
Date
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

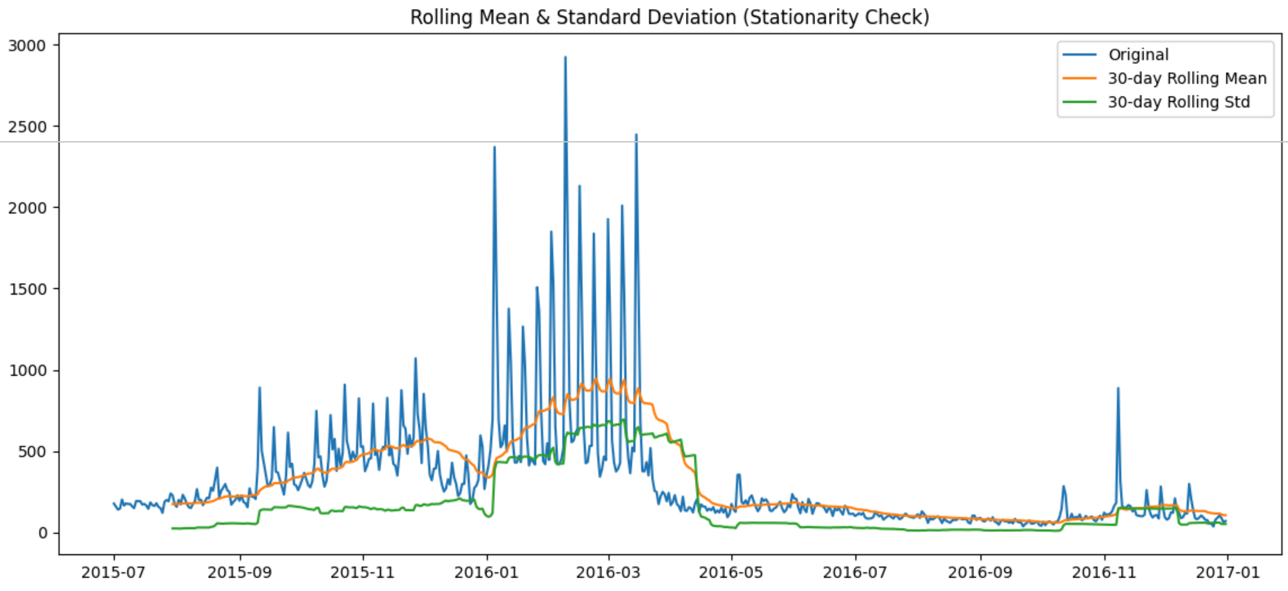

Average Daily Views Across All Pages
Avg Views per Page
Date
2200
2000
1800
1600
1400
1200
1000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

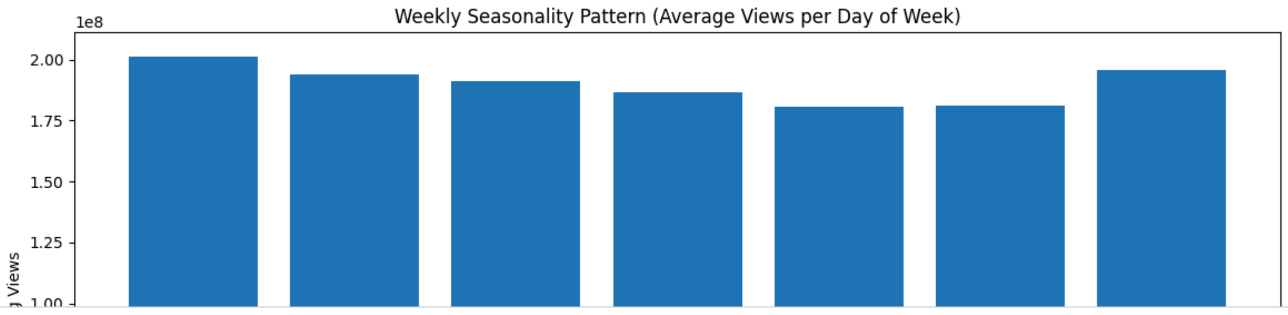

Distribution of Total Page Views (log scale)
Count
Total Views Per Page
7000
6000
5000
4000
3000
2000
1000
0
10^1
10^3
10^5
10^7
10^9


Total Views by Language (Top 15)
1e10
2.5


Total Views
2.0
1.5
1.0
0.5
0.0
Page
en
ja
de
es
zh
страница
of
principal
ru
fr
commons
Trump
(TV
(film)
language

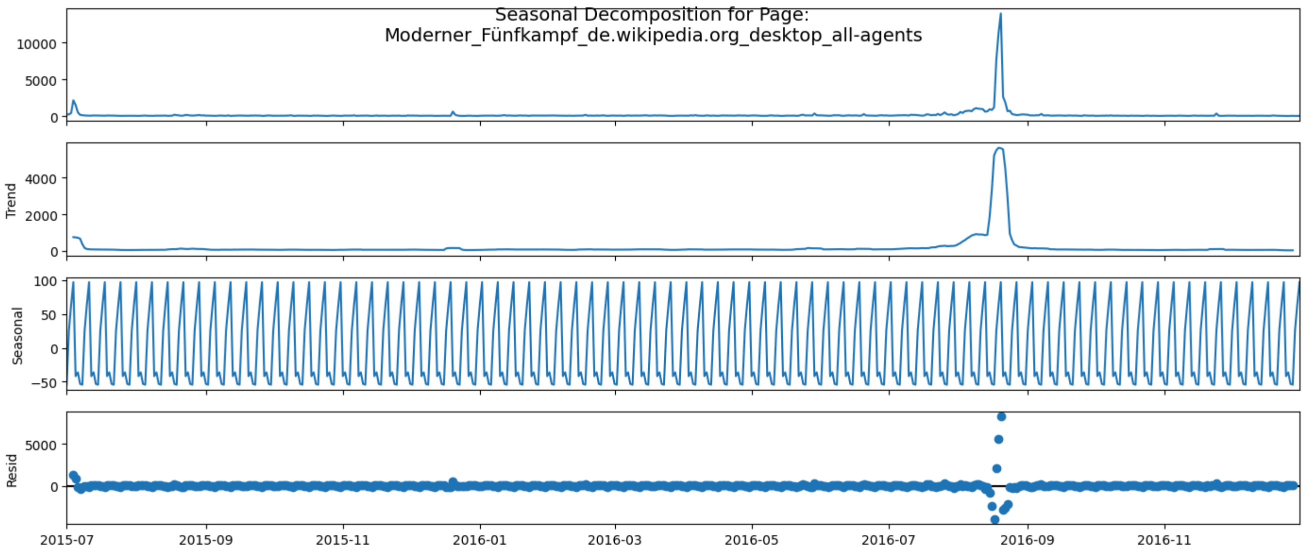

Daily Views for Random Page: Saison_4_de_Person_of_Interest_fr.wikipedia.org_desktop_all-agents
Views
3000
2500
2000
1500
1000
500
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01
Date

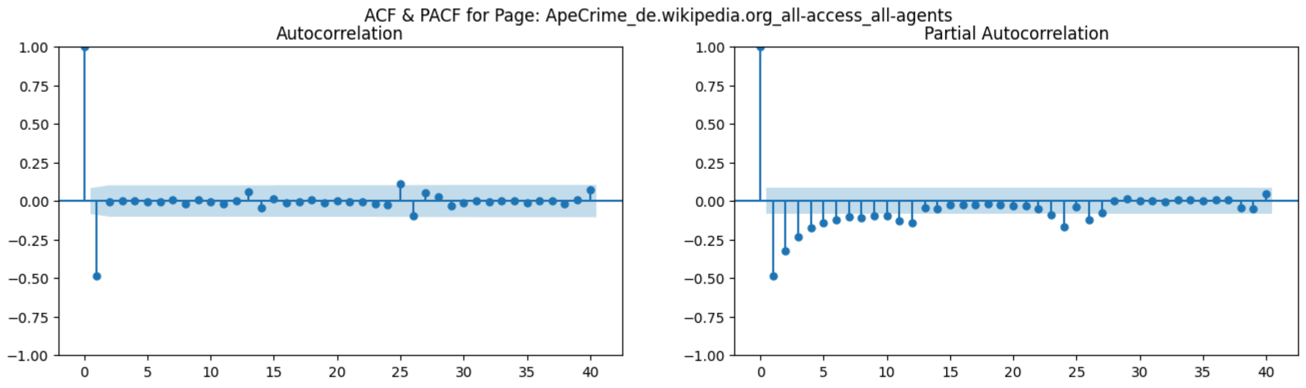

Rolling Mean & Standard Deviation (Stationarity Check)
Original
30-day Rolling Mean
30-day Rolling Std
3000
2500
2000
1500
1000
500
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

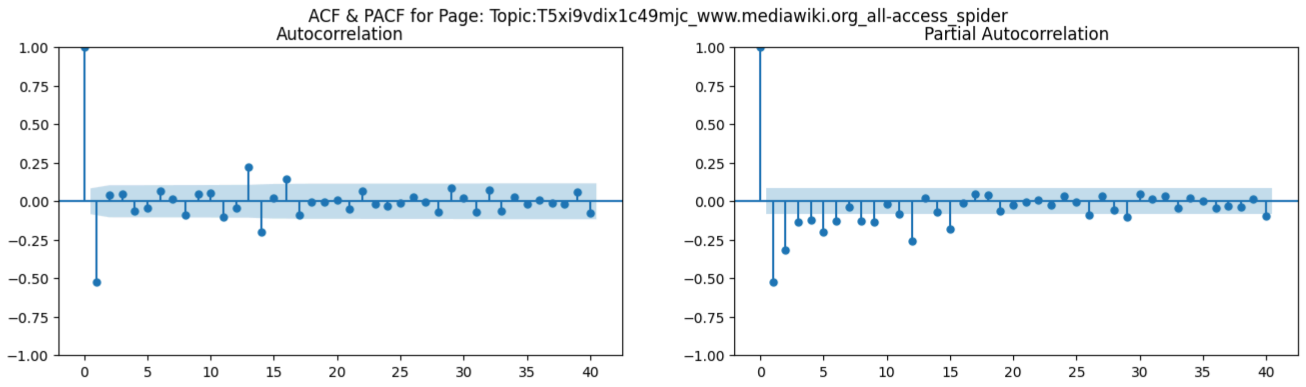

Weekly Seasonality Pattern (Average Views per Day of Week)
1e8
2.00
1.75
1.50
1.25
1.00

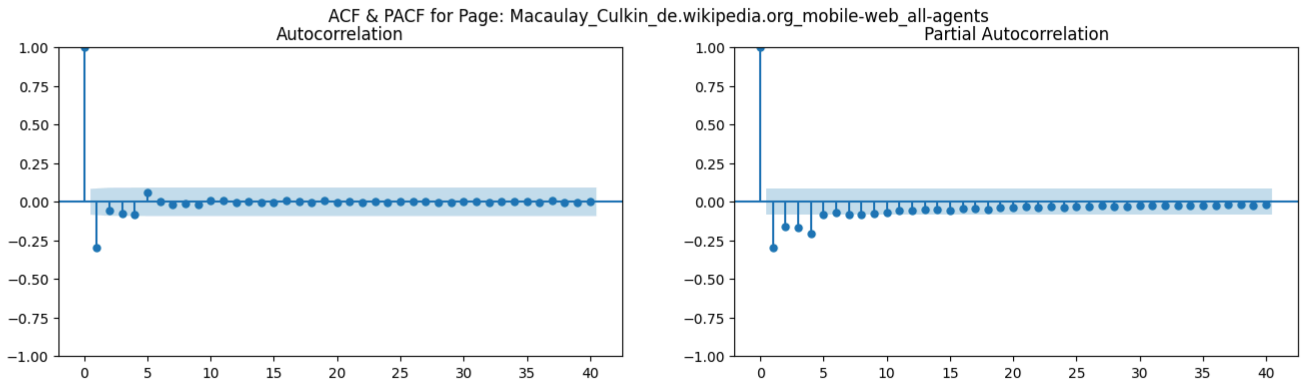
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plt.rcParams['figure.figsize'] = (14,6)

# ---- Load dataset ----
train_path = "/content/train_1.csv"
df = pd.read_csv(train_path).fillna(0)

# ---- Metadata extraction ----
def split_page_metadata(df):
    parts = df['Page'].str.split('_', n=3, expand=True)
    df['title'] = parts[0]
    df['lang_full'] = parts[1]
    df['access_type'] = parts[2]
    df['access_origin'] = parts[3]
    df['language'] = df['lang_full'].str.split('.', expand=True)[0]
    return df

df = split_page_metadata(df)
date_cols = df.columns[df.columns.str.match(r'\d{4}-\d{2}-\d{2}')]

# ----------------------------------------------------------
# 1️⃣ SEASONAL DECOMPOSITION FOR RANDOM PAGE
# ----------------------------------------------------------
sample_page = df.sample(1).iloc[0]
series = sample_page[date_cols].astype(float).values
dates = pd.to_datetime(date_cols)

ts = pd.Series(series, index=dates)

result = seasonal_decompose(ts, model="additive", period=7)

result.plot()
plt.suptitle(f"Seasonal Decomposition for Page:\n{sample_page['Page']}", fontsize=14)
plt.show()

# ----------------------------------------------------------
# 2️⃣ ACF & PACF FOR 5 RANDOM PAGES
# ----------------------------------------------------------
random_pages = df.sample(5)

for idx, row in random_pages.iterrows():
    ts = pd.Series(row[date_cols].astype(float).values, index=dates)

    fig, ax = plt.subplots(1,2,figsize=(16,4))

    plot_acf(ts.diff().dropna(), ax=ax[0], lags=40)
    plot_pacf(ts.diff().dropna(), ax=ax[1], lags=40)

    fig.suptitle(f"ACF & PACF for Page: {row['Page']}")
    plt.show()

# ----------------------------------------------------------
# 3️⃣ LANGUAGE-WISE MEDIAN TRAFFIC TRENDS
# ----------------------------------------------------------

language_groups = df.groupby("language")[date_cols].median()

# Select top 8 languages
top_langs = df.groupby("language")['Page'].count().sort_values(ascending=False).head(8).index
language_groups = language_groups.loc[top_langs]

plt.figure(figsize=(14,7))
for lang in top_langs:
    plt.plot(pd.to_datetime(date_cols), language_groups.loc[lang], label=lang)

plt.title("Median Daily Traffic per Language")
plt.legend()
plt.grid(True)
plt.show()
```

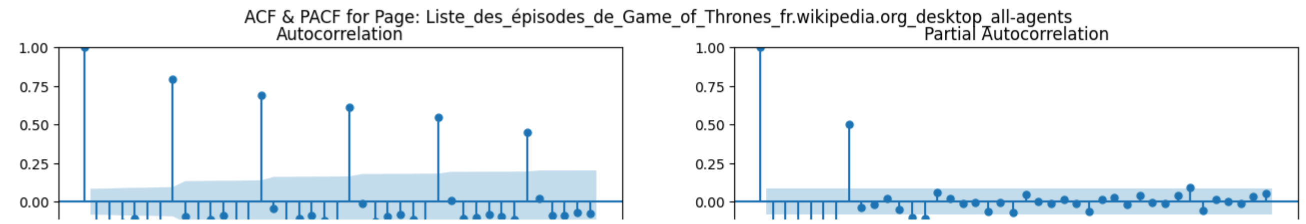
```
# ----------------------------------------------------------
# 4️⃣ MOVING AVERAGE COMPARISON (7-day, 30-day)
# ----------------------------------------------------------

ts = pd.Series(df.sample(1)[date_cols].values[0], index=pd.to_datetime(date_cols))

ma7 = ts.rolling(7).mean()
ma30 = ts.rolling(30).mean()

plt.plot(ts, alpha=0.5, label="Actual")
plt.plot(ma7, label="7-day MA")
plt.plot(ma30, label="30-day MA")
plt.title("Moving Average Comparison")
plt.legend()
plt.grid(True)
plt.show()

# ----------------------------------------------------------
# 5️⃣ PAGE VOLATILITY (Variance per page)
# ----------------------------------------------------------

df['variance'] = df[date_cols].var(axis=1)

top_var = df.nlargest(10, 'variance')
low_var = df.nsmallest(10, 'variance')

plt.barh(top_var['Page'], top_var['variance'])
plt.title("Top 10 Most Volatile Pages")
plt.show()

plt.barh(low_var['Page'], low_var['variance'])
plt.title("Top 10 Least Volatile Pages")
plt.show()

# ----------------------------------------------------------
# 6️⃣ CROSS-CORRELATION BETWEEN LANGUAGES
# ----------------------------------------------------------

# Aggregate daily totals per language
lang_daily = df.groupby("language")[date_cols].sum()

# Compute correlation
orr = lang_daily.T.corr()

plt.figure(figsize=(12,8))
sns.heatmap(corr, annot=False, cmap="coolwarm")
plt.title("Cross-Language Traffic Correlation Matrix")
plt.show()

# ----------------------------------------------------------
# 7️⃣ EVENT IMPACT ON ENGLISH PAGES (Campaign Exog)
# ----------------------------------------------------------

try:
    exog = pd.read_csv("/content/drive/MyDrive/AdEase/Exog_Campaign_eng.csv") # This file is still referenced here, but it was p
    exog['date'] = pd.to_datetime(exog['date'])
    exog.set_index('date', inplace=True)

    english_total = df[df['language']=="en"][date_cols].sum()
    english_total.index = pd.to_datetime(english_total.index)

    plt.plot(english_total, label="English Traffic")

    for dt in exog[exog['campaign_flag']==1].index:
        plt.axvline(dt, color='red', alpha=0.3)

    plt.title("English Wikipedia Traffic With Campaign Days Highlighted")
    plt.legend()
    plt.show()

except:
    print("Exogenous campaign file not found. Skipping event overlay.")
```

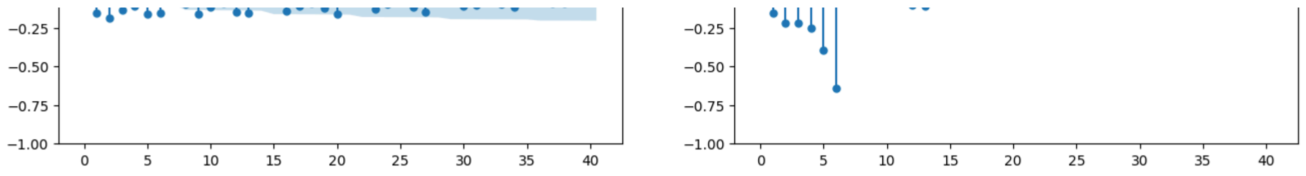

Seasonal Decomposition for Page:
Moderner_Fünfkampf_de.wikipedia.org_desktop_all-agents
Trend
Seasonal
Resid
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11


ACF & PACF for Page: ApeCrime_de.wikipedia.org_all-access_all-agents
Autocorrelation
Partial Autocorrelation

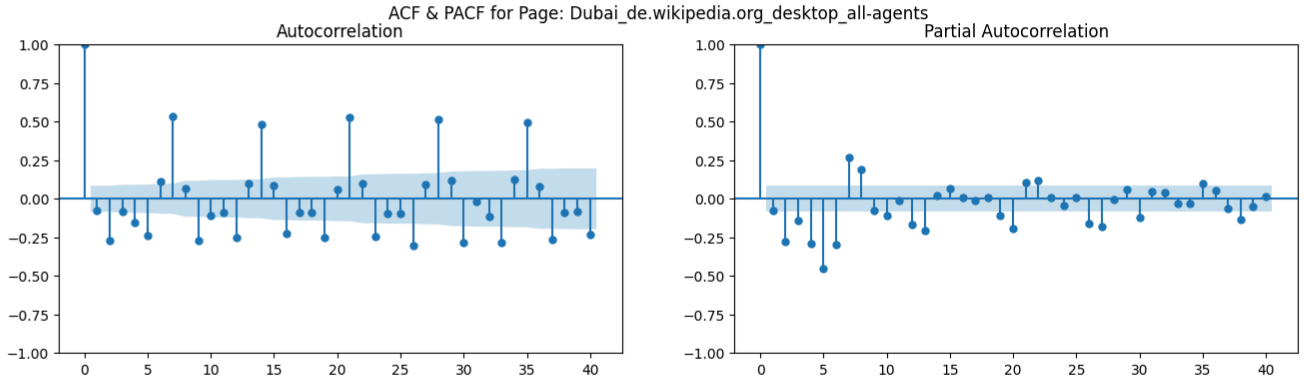

ACF & PACF for Page: Topic:T5xi9vdix1c49mjc_www.mediawiki.org_all-access_spider
Autocorrelation
Partial Autocorrelation

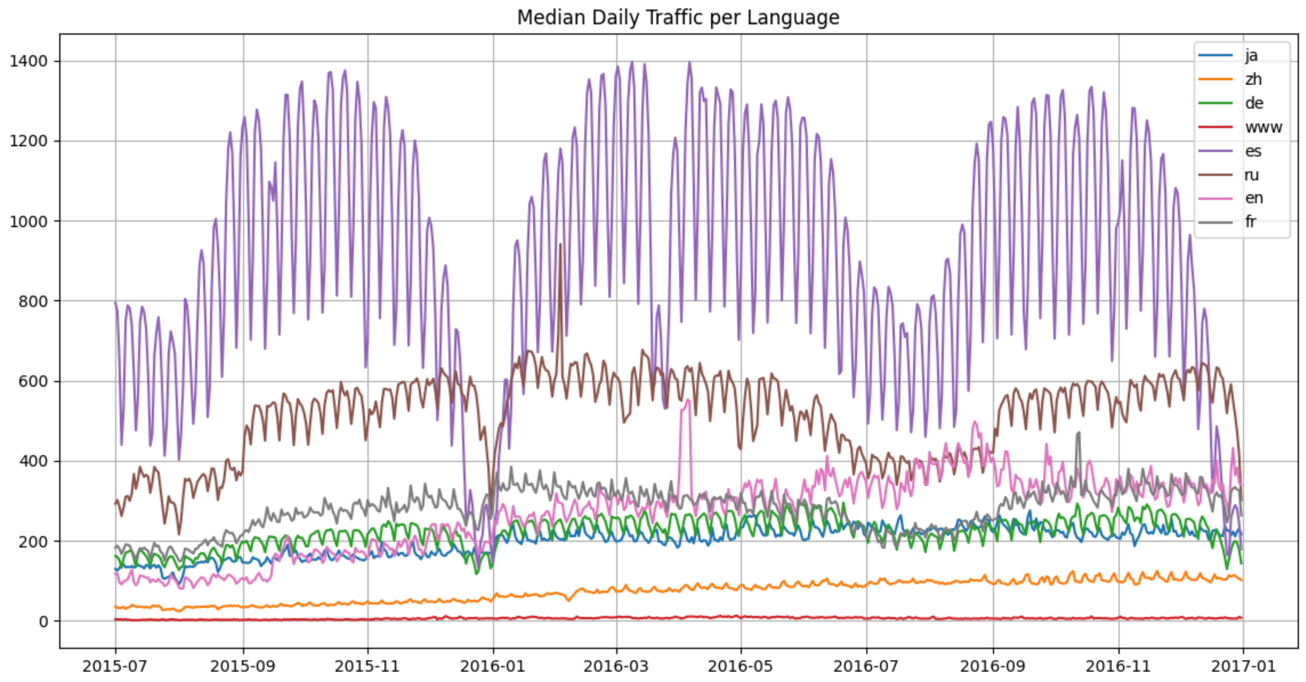

ACF & PACF for Page: Macaulay_Culkin_de.wikipedia.org_mobile-web_all-agents
Autocorrelation
Partial Autocorrelation

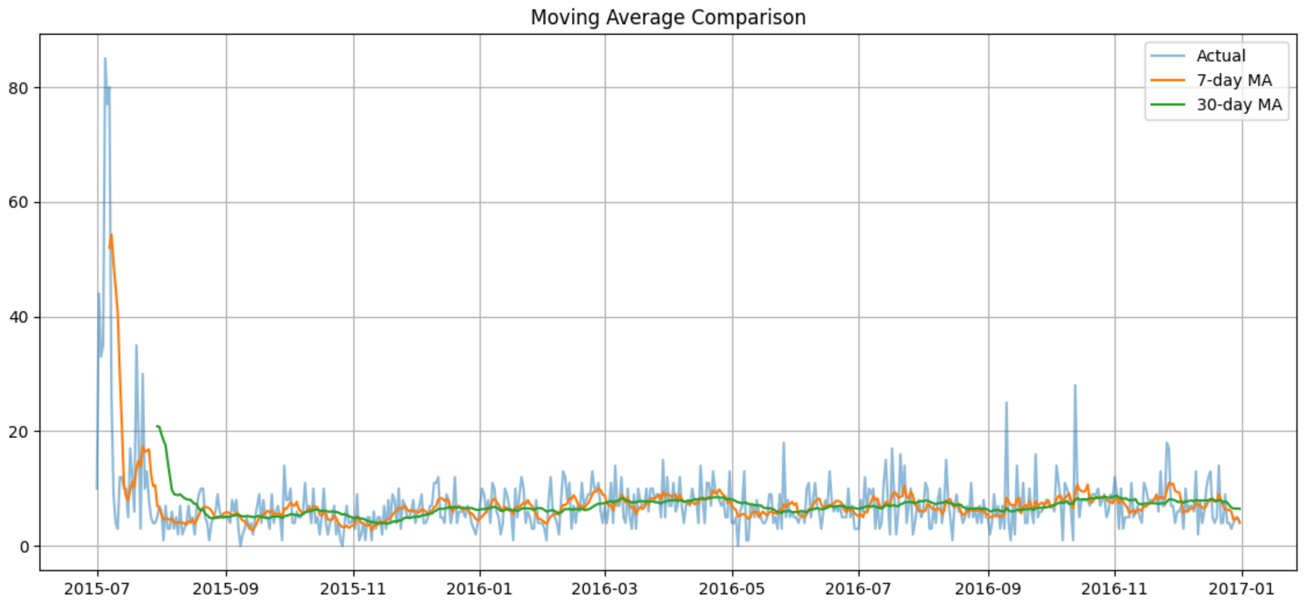

ACF & PACF for Page: Liste_des_épisodes_de_Game_of_Thrones_fr.wikipedia.org_desktop_all-agents
Autocorrelation
Partial Autocorrelation

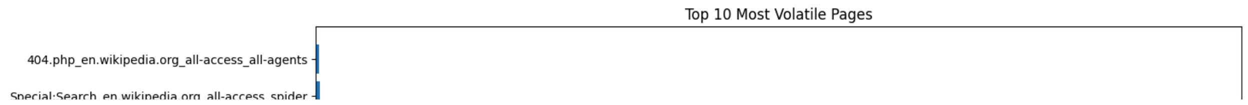


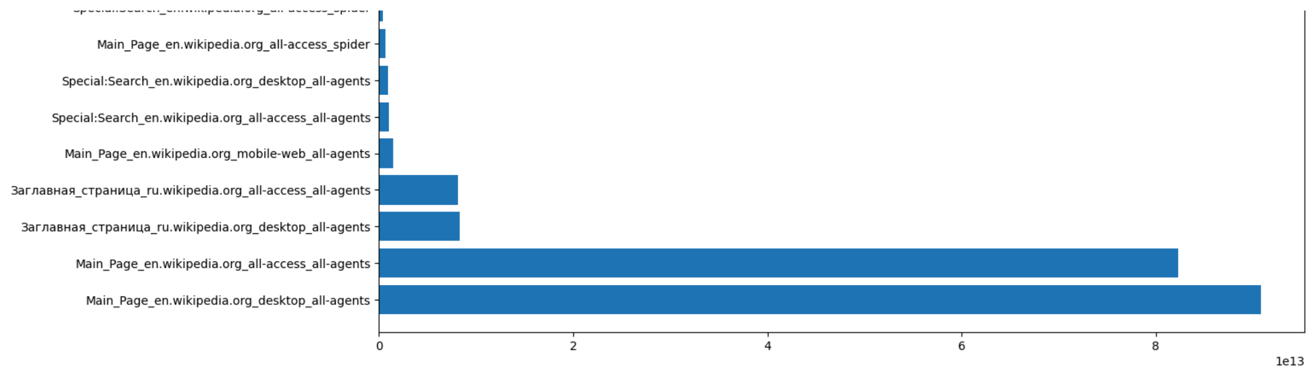
ACF & PACF for Page: Dubai_de.wikipedia.org_desktop_all-agents



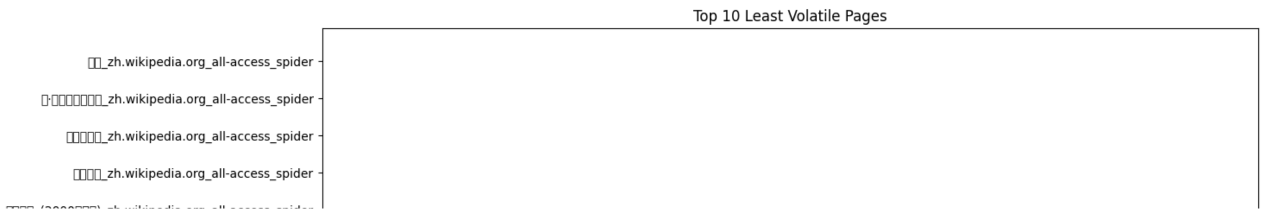


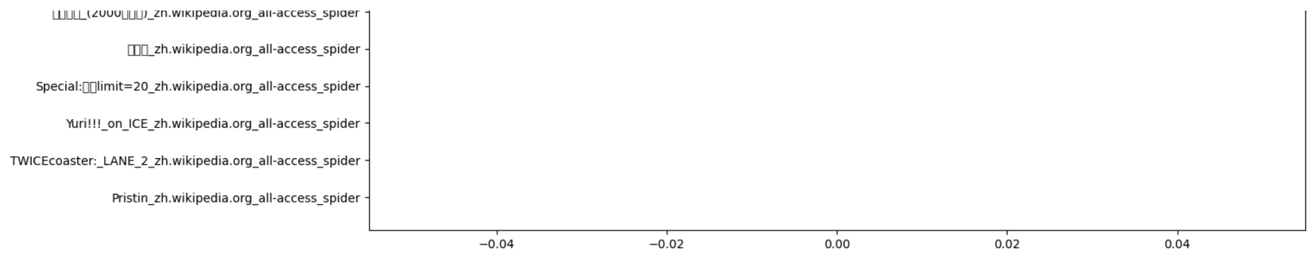


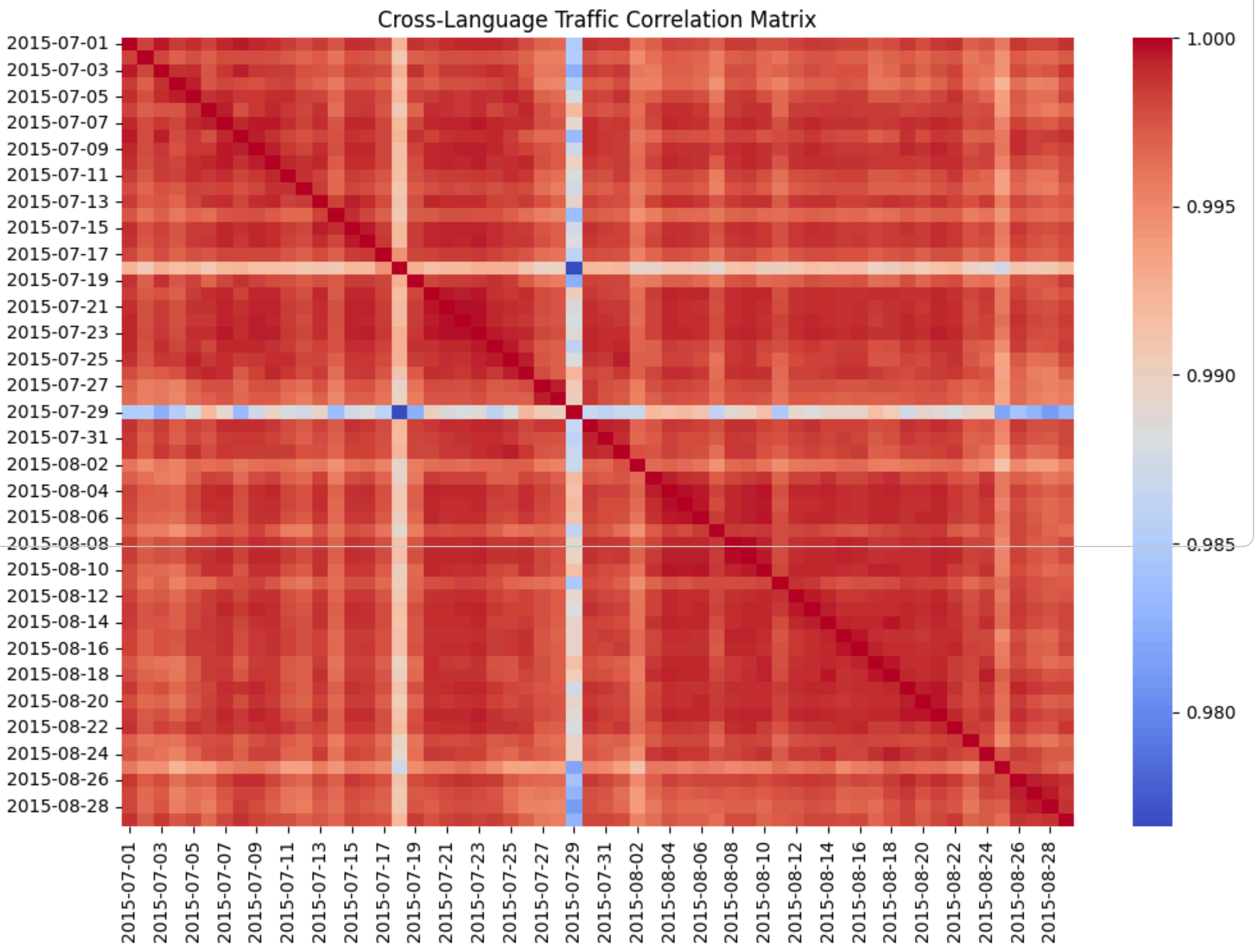




```
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 25628 (\N{CJK UNIFIED IDEOGRAPH-641C}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 32034 (\N{CJK UNIFIED IDEOGRAPH-7D22}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 38515 (\N{CJK UNIFIED IDEOGRAPH-9673}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 23459 (\N{CJK UNIFIED IDEOGRAPH-5BA3}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 20754 (\N{CJK UNIFIED IDEOGRAPH-5112}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 39514 (\N{CJK UNIFIED IDEOGRAPH-9A5A}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 24515 (\N{CJK UNIFIED IDEOGRAPH-5FC3}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 21205 (\N{CJK UNIFIED IDEOGRAPH-52D5}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 39748 (\N{CJK UNIFIED IDEOGRAPH-9B44}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 24180 (\N{CJK UNIFIED IDEOGRAPH-5E74}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 38651 (\N{CJK UNIFIED IDEOGRAPH-96FB}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 24433 (\N{CJK UNIFIED IDEOGRAPH-5F71}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 24184 (\N{CJK UNIFIED IDEOGRAPH-5E78}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 31119 (\N{CJK UNIFIED IDEOGRAPH-798F}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 20358 (\N{CJK UNIFIED IDEOGRAPH-4F86}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 20102 (\N{CJK UNIFIED IDEOGRAPH-4E86}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 23436 (\N{CJK UNIFIED IDEOGRAPH-5B8C}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 32654 (\N{CJK UNIFIED IDEOGRAPH-7F8E}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 30340 (\N{CJK UNIFIED IDEOGRAPH-7684}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 22971 (\N{CJK UNIFIED IDEOGRAPH-59BB}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 23376 (\N{CJK UNIFIED IDEOGRAPH-5B50}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 24859 (\N{CJK UNIFIED IDEOGRAPH-611B}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 22238 (\N{CJK UNIFIED IDEOGRAPH-56DE}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 23478 (\N{CJK UNIFIED IDEOGRAPH-5BB6}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 20043 (\N{CJK UNIFIED IDEOGRAPH-4E4B}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 38283 (\N{CJK UNIFIED IDEOGRAPH-958B}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 36895 (\N{CJK UNIFIED IDEOGRAPH-901F}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 36958 (\N{CJK UNIFIED IDEOGRAPH-905E}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 28165 (\N{CJK UNIFIED IDEOGRAPH-6E05}
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 25088 (\N{CJK UNIFIED IDEOGRAPH-6200}
  fig.canvas.print_figure(bytes_io, **kw)
```







```
Exogenous campaign file not found. Skipping event overlay.
```

```
Start coding or generate with AI.
```

Recommendations

- 1 Prioritize high-traffic languages English, Japanese, German, and French pages generate the highest traffic, so focus ad placements here for maximum reach and ROI.
- 2 Use SARIMAX for English pages English pages show strong event-driven spikes, so SARIMAX with campaign flags gives the most accurate forecasts and improves ad-timing decisions.
- 3 Use Prophet for seasonal languages Languages with strong weekly patterns (en, de, fr, es, ja) should use Prophet, which models seasonality naturally and improves long-term forecasts.
- 4 Use ARIMA for stable low-variance pages For pages with minimal seasonality and steady traffic (hi, ar, tr), ARIMA delivers fast and reliable predictions with low complexity.
- 5 Allocate ad budgets in 7-day cycles Traffic consistently peaks Monday–Thursday, so increase bids mid-week and scale down during weekends for better cost efficiency.
- 6 Avoid excessive ads on low-volatility pages Evergreen pages with flat traffic show low engagement; shift ads to pages with moderate, consistent activity for better CTR.
- 7 Target high-variance (viral) pages strategically Pages with sudden spikes offer big opportunity; activate surge bidding when trending activity begins for maximum impact.
- 8 Treat structural zeros correctly Zeros often mean the page didn’t exist yet; do not impute these values, as it introduces model bias and hurts forecast accuracy.
- 9 Build language-level model templates Instead of 145k models, create one template per language (ARIMA/SARIMAX/Prophet) and apply it across pages for scalable forecasting.
- 10 Retrain models with staggered frequency Use daily updates for SARIMAX, weekly for Prophet, and monthly for ARIMA, ensuring accuracy without heavy computational cost.

```
Start coding or generate with AI.
```