

Repository:

<https://github.com/rano667/AdEase-Timeseries-Forecasting>

Notebooks:

EDA → <https://colab.research.google.com/drive/1es8O-Ty2oBzPK6wwC4UDU3b68bC9VzM9?usp=sharing>

Modeling →

https://colab.research.google.com/drive/13y0h8tMvqw8_c0UUnRJQ2YTZoyvXWODq?usp=sharing

Overall Project Plan for AdEase Time Series

1. Problem Understanding & Setup

- **Problem:** Forecast Wikipedia page views across 550 days for ~145k pages, split by title, language, access type, access origin, incorporating campaign effect (English only).
- **Business Use:** Predict traffic to optimize ad placements per language/region.
- **Models:**
 - ARIMA
 - SARIMAX (with exogenous campaign data)
 - Prophet (with exogenous campaign data)

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

import re
import gc

import warnings
warnings.filterwarnings("ignore")

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

train_clean =
pd.read_parquet("/content/drive/MyDrive/Wikipedia/train_clean.parquet")
campaign =
pd.read_csv("/content/drive/MyDrive/Wikipedia/Exog_Campaign_eng")
```

```

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

print("Shape after reload:", train_clean.shape)
Shape after reload: (142185, 563)

train_clean.head()
{"type": "dataframe", "variable_name": "train_clean"}

# detect date columns robustly
date_cols = [c for c in train_clean.columns if re.match(r'^\d{4}-\d{2}-\d{2}$', c)]
print("num date cols:", len(date_cols))
print("first / last date columns:", date_cols[0], date_cols[-1])
# convert date list to datetime for plotting use
dates = pd.to_datetime(date_cols)

num date cols: 550
first / last date columns: 2015-07-01 2016-12-31

```

[] AdEase Time-Series Preprocessing & EDA

Explore the full time-series forecasting workflow — including ARIMA, SARIMAX, and Prophet model comparisons — in the following Colab notebook:

[\[\] Open AdEase Time-Series EDA Notebook in Google Colab](#)

- <https://colab.research.google.com/drive/1es8O-Ty2oBzPK6wwC4UDU3b68bC9VzM9?usp=sharing>

5. Data Reshaping for Modeling

[] Tasks

1. **Pivot the data** so that for each page, we get a (**date, views**) series.
2. **Pick a subset of pages** — representative pages across languages — to keep the dataset manageable.
3. **Prepare for stationarity checks** in Step 6.

A. Reshaping (long and wide)

```

# melt into long format
ts_data = train_clean.melt(
    id_vars=['Page', 'Title', 'Language', 'AccessType', 'AccessOrigin'],
    value_vars=date_cols,
    var_name='date',
    value_name='views'

```

```

)
ts_data['date'] = pd.to_datetime(ts_data['date'])
ts_data['views'] = ts_data['views'].astype(float)
print("long shape:", ts_data.shape)

long shape: (78201750, 7)

print("long shape:", ts_data.shape)

long shape: (78201750, 7)

ts_data.head()

{"type": "dataframe", "variable_name": "ts_data"}

ts_data[ts_data["Page"] == "2NE1_zh.wikipedia.org_all-access_spider"].head()

{"summary": "{\n    \"name\": \"ts_data[ts_data[\"Page\"] ==\n        \"2NE1_zh\", \"rows\": 5, \"fields\": [\n            {\n                \"column\": \"Page\", \"properties\": {\n                    \"dtype\": \"category\", \"num_unique_values\": 1, \"samples\": [\n                        \"2NE1_zh.wikipedia.org_all-access_spider\"\n                    ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                    {\n                        \"column\": \"Title\", \"properties\": {\n                            \"dtype\": \"category\", \"num_unique_values\": 1, \"samples\": [\n                                \"2NE1\"\n                            ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                        {\n                            \"column\": \"Language\", \"properties\": {\n                                \"dtype\": \"category\", \"num_unique_values\": 1, \"samples\": [\n                                    \"zh\"\n                                ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                            {\n                                \"column\": \"AccessType\", \"properties\": {\n                                    \"dtype\": \"category\", \"num_unique_values\": 1, \"samples\": [\n                                        \"all-access\"\n                                    ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                                {\n                                    \"column\": \"AccessOrigin\", \"properties\": {\n                                        \"dtype\": \"category\", \"num_unique_values\": 1, \"samples\": [\n                                            \"spider\"\n                                        ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                                    {\n                                        \"column\": \"date\", \"properties\": {\n                                            \"dtype\": \"date\", \"min\": \"2015-07-01 00:00:00\", \"max\": \"2015-07-05 00:00:00\", \"num_unique_values\": 5, \"samples\": [\n                                                \"2015-07-02 00:00:00\"\n                                            ], \"semantic_type\": \"\", \"description\": \"\\n                \"},\n                                        {\n                                            \"column\": \"views\", \"properties\": {\n                                                \"dtype\": \"number\", \"std\": 4.764451699828638, \"min\": 5.0, \"max\": 18.0, \"num_unique_values\": 5, \"samples\": [\n                                                    11.0\n                                                ]\n                                            }\n                                        }\n                                    }\n                                }\n                            }\n                        }\n                    }\n                }\n            ]\n        }\n    }\n}"}

```

```
\"semantic_type\": \"\",  
      \"description\": \"\"\n    }  
  ]\n}, "type": "dataframe"}]
```

B. Selection Strategies

1. Representative (A) — Stratified Sample

For each language, pick:

- **Top 3 pages** by `mean_views`
- **1 page** with the **highest standard deviation (std)** — i.e., high-variance or volatile pages
- **1 low-traffic page** (non-zero views)

→ This gives approximately **5 × #languages** rows (manageable size).

2. High-Traffic (B) — Production

For each language, pick:

- **Top K pages** by `total_views` or `mean_views`
(e.g., $K = 50$ or 100 depending on compute capacity)

Alternative:

- Pick **all pages** with `mean_views >= threshold`
(e.g., $threshold = 50$)

```
# representative sample  
def stratified_sample(df, languages, top_k=3, var_k=1, low_k=1):  
    pages = []  
    for lang in languages:  
        grp = df[df['Language']==lang]  
        pages += grp.sort_values('mean_views',  
ascending=False).head(top_k)['Page'].tolist()  
        pages += grp.sort_values('std_views',  
ascending=False).head(var_k)['Page'].tolist()  
        pages += grp[grp['mean_views']>0].sort_values('mean_views',  
ascending=True).head(low_k)['Page'].tolist()  
    return list(dict.fromkeys(pages))  
  
languages = train_clean['Language'].unique().tolist() # or select  
subset  
rep_pages = stratified_sample(train_clean, languages, top_k=3,  
var_k=1, low_k=1)  
len(rep_pages)
```

```

# high-traffic sample
K = 50
top_pages_per_lang = []
for lang in languages:
    grp = train_clean[train_clean['Language'] == lang]
    top_pages_per_lang += grp.sort_values('mean_views',
ascending=False).head(K)[['Page']].tolist()
# optional: unique()
top_pages_per_lang = list(dict.fromkeys(top_pages_per_lang))
len(top_pages_per_lang)

450

```

C. Pivot subset to modeling format (wide for ARIMA loop; long for Prophet)

```

# subset to pages (example: rep_pages)
ts_subset = ts_data[ts_data['Page'].isin(rep_pages)].copy()

# wide for ARIMA-style (columns = pages)
ts_wide = ts_subset.pivot(index='date', columns='Page',
values='views')

# prophet-ready for a single page:
df_prophet = ts_subset[ts_subset['Page'] == rep_pages[0]]
[['date', 'views']].rename(columns={'date': 'ds', 'views': 'y'})

```

6. Stationarity Checks

Tasks:

- Perform **ADF test** to check stationarity.
- If non-stationary, apply transformations:
 - Differencing
 - Log transform
 - Seasonal differencing
- Decompose series into **trend, seasonality, residuals**.

```

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

# --- Utility Functions ---
def check_stationarity(series, alpha=0.05):
    s = series.dropna()

```

```

res = adfuller(s)
is_stationary = res[1] < alpha
return {
    'adf_stat': res[0],
    'pvalue': res[1],
    'stationary': is_stationary,
    'crit': res[4]
}

def plot_rolling_stats(series, title):
    rolling_mean = series.rolling(window=7).mean()
    rolling_std = series.rolling(window=7).std()
    plt.figure(figsize=(12,4))
    plt.plot(series, label='Original', alpha=0.7)
    plt.plot(rolling_mean, label='Rolling Mean (7d)')
    plt.plot(rolling_std, label='Rolling Std (7d)')
    plt.title(title)
    plt.legend()
    plt.show()

def decompose_and_plot(series, freq=7):
    result = seasonal_decompose(series.dropna(), period=freq,
model='additive', extrapolate_trend='freq')
    result.plot()
    plt.suptitle("Seasonal Decomposition", y=1.02)
    plt.show()
    return result

def plot_acf_pacf(series, lags=40):
    fig, axes = plt.subplots(1,2, figsize=(12,4))
    plot_acf(series.dropna(), ax=axes[0], lags=lags)
    plot_pacf(series.dropna(), ax=axes[1], lags=lags)
    plt.show()

# --- Loop over representative pages ---
stationarity_results = []

for page in rep_pages:
    print(f"\nAnalyzing page: {page}")
    series = ts_wide[page].dropna()

    # ADF test
    res = check_stationarity(series)
    print(f"ADF Statistic: {res['adf_stat']:.3f} | p-value: {res['pvalue']:.4f} | Stationary: {res['stationary']}")

    # Rolling mean & variance
    plot_rolling_stats(series, f"Rolling Mean & Std - {page}")

    # Decomposition

```

```

decompose_and_plot(series)

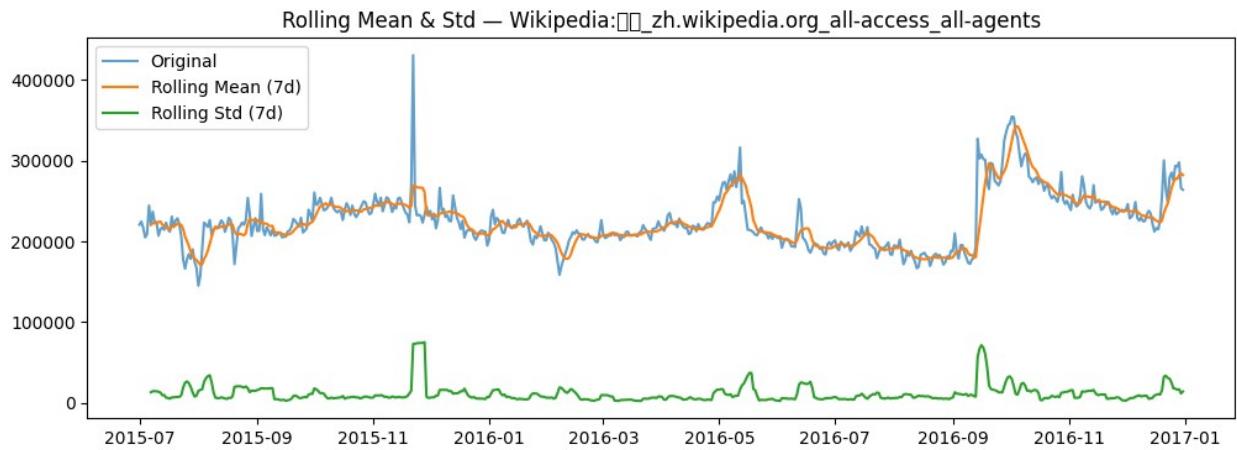
# ACF/PACF
plot_acf_pacf(series)

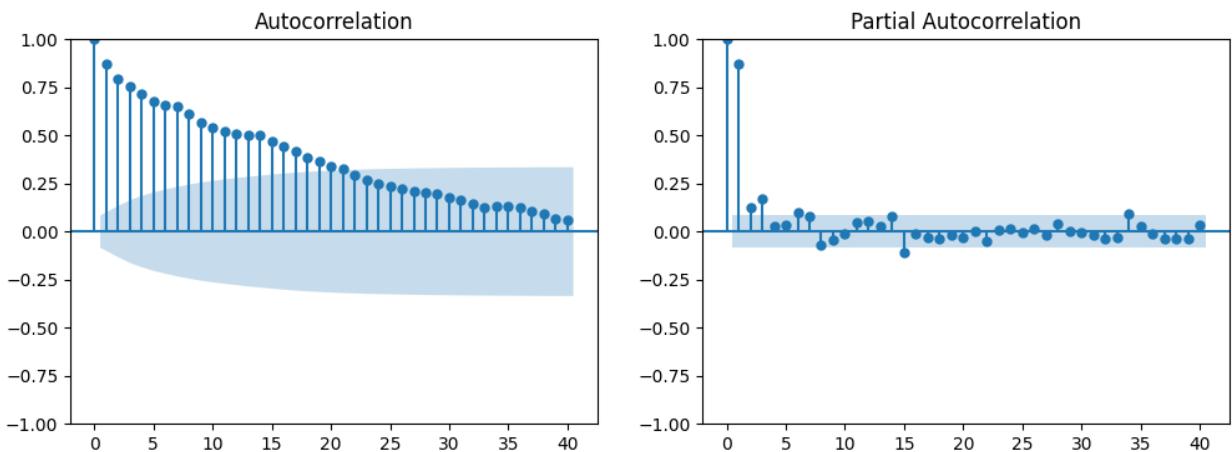
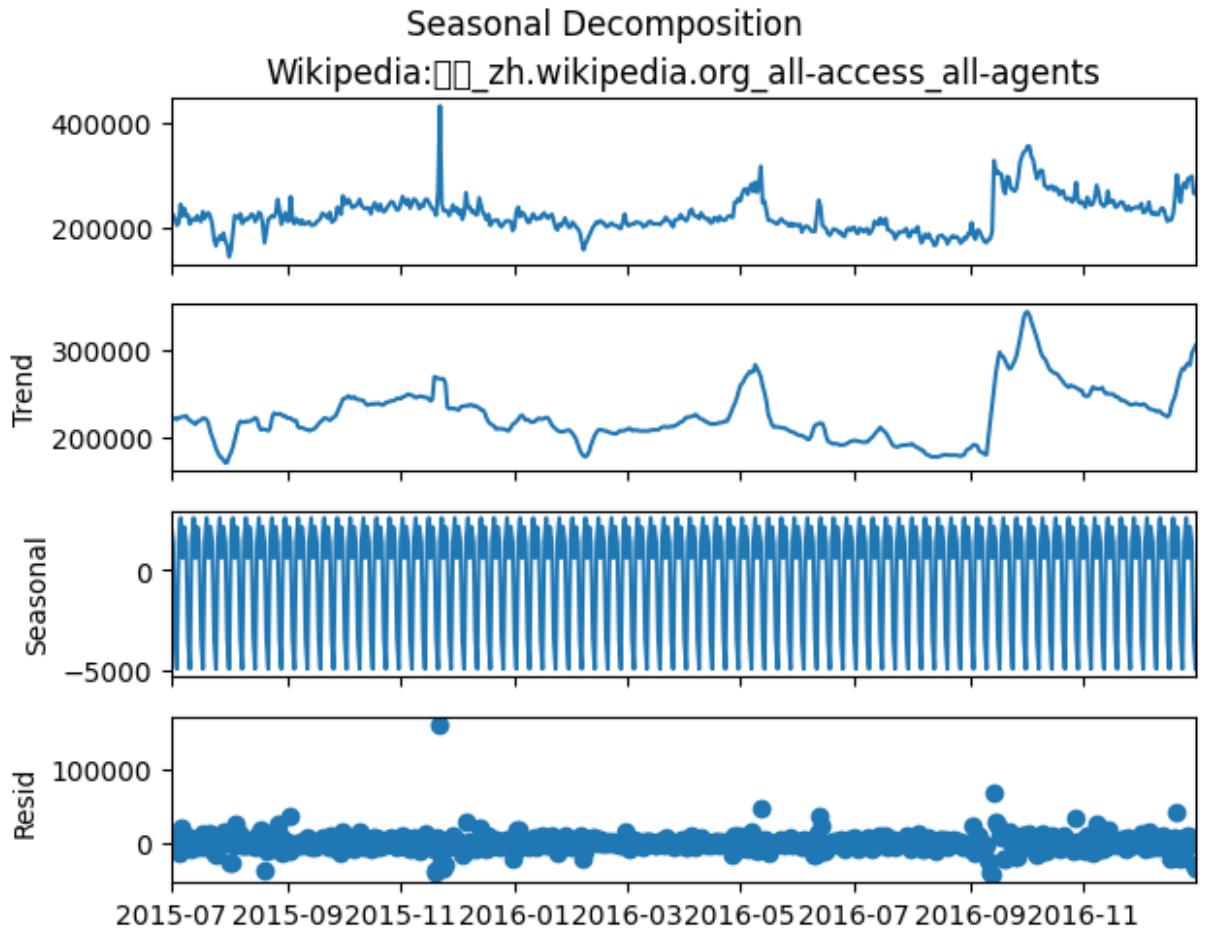
# Store results
stationarity_results.append({'Page': page, **res})

# Free memory after each loop
plt.close('all')
gc.collect()

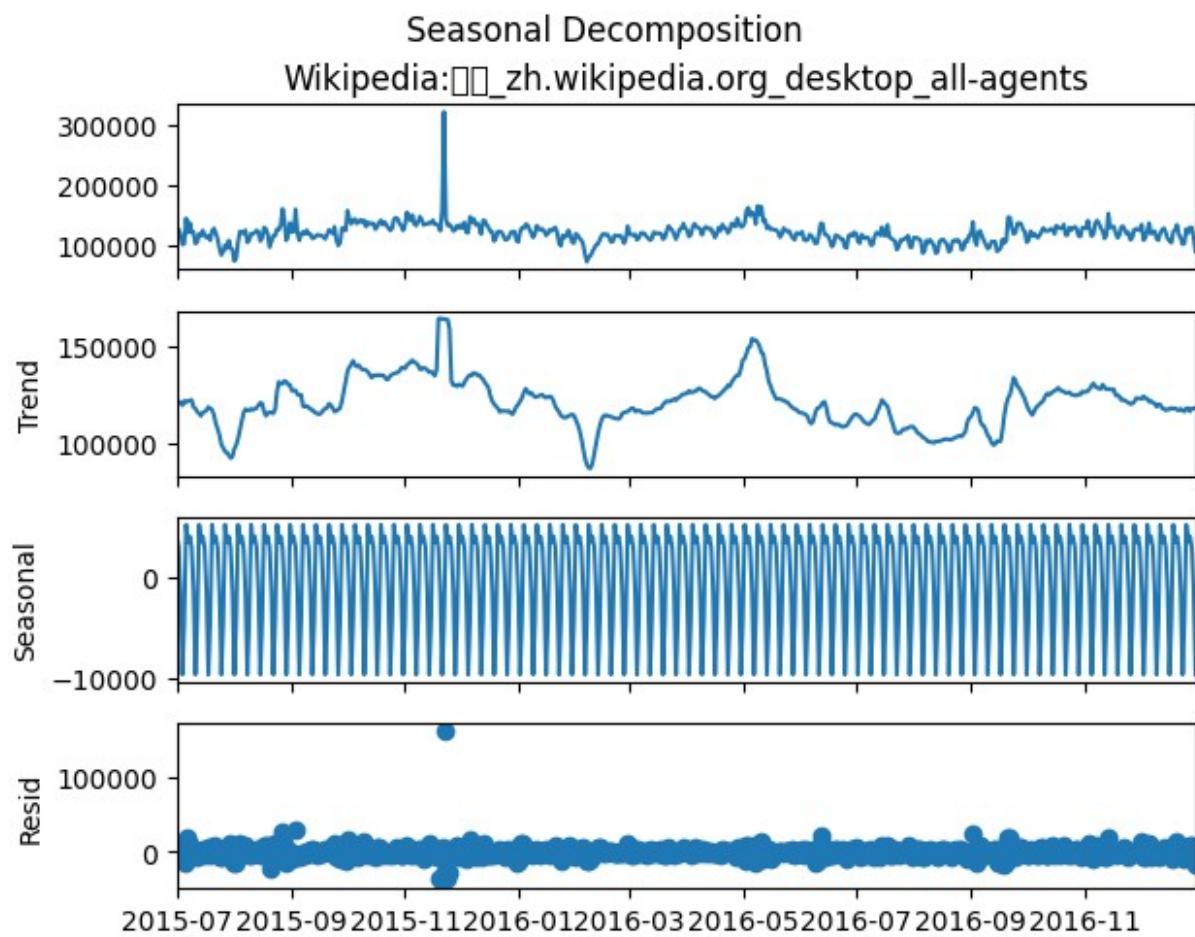
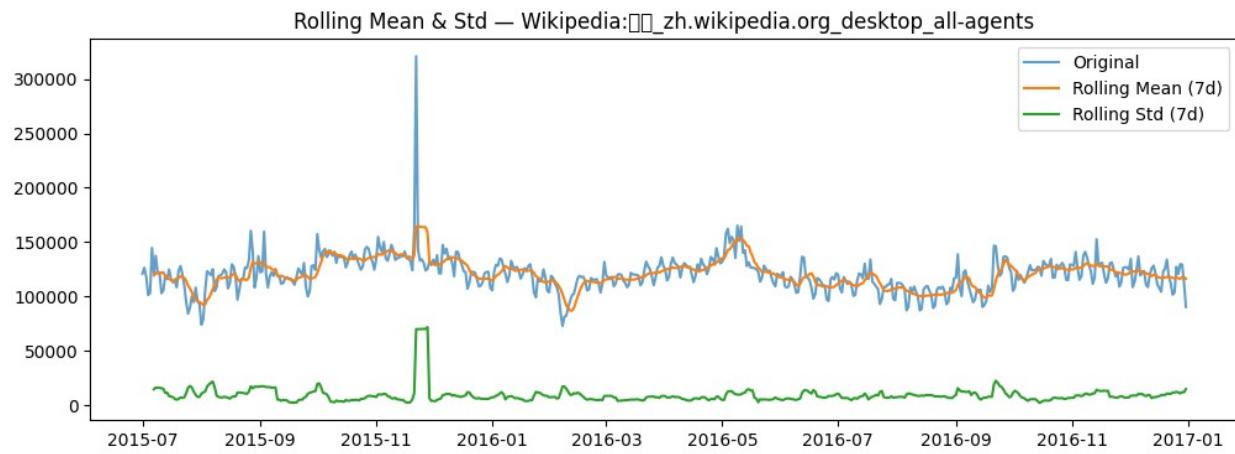
```

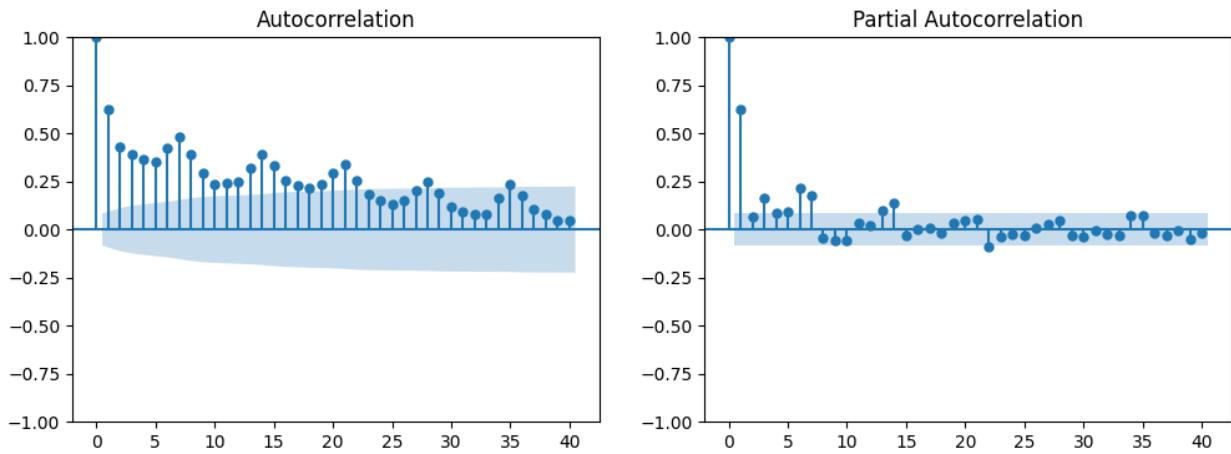
□ Analyzing page: Wikipedia:首页_zh.wikipedia.org_all-access_all-agents
 ADF Statistic: -2.657 | p-value: 0.0817 | Stationary: False



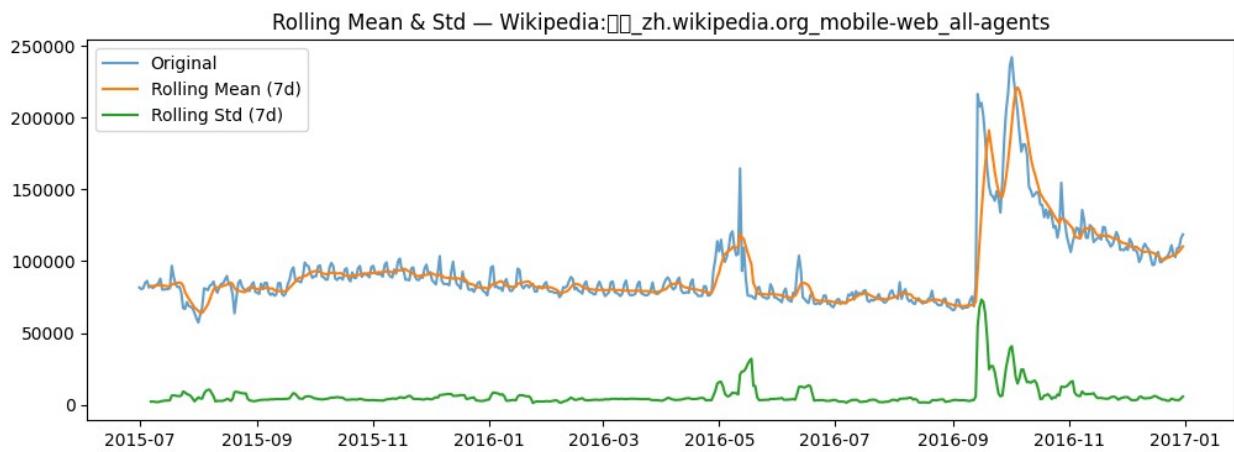


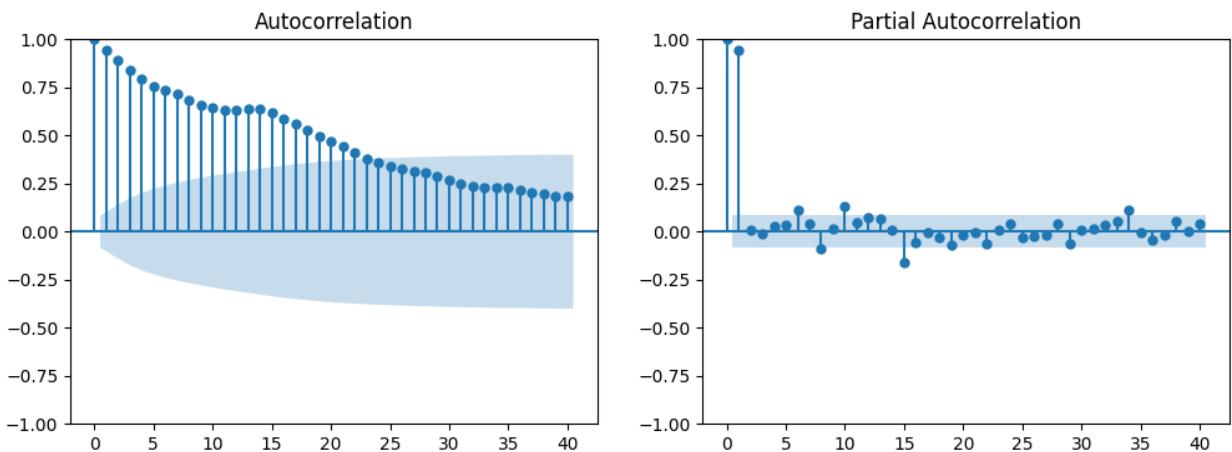
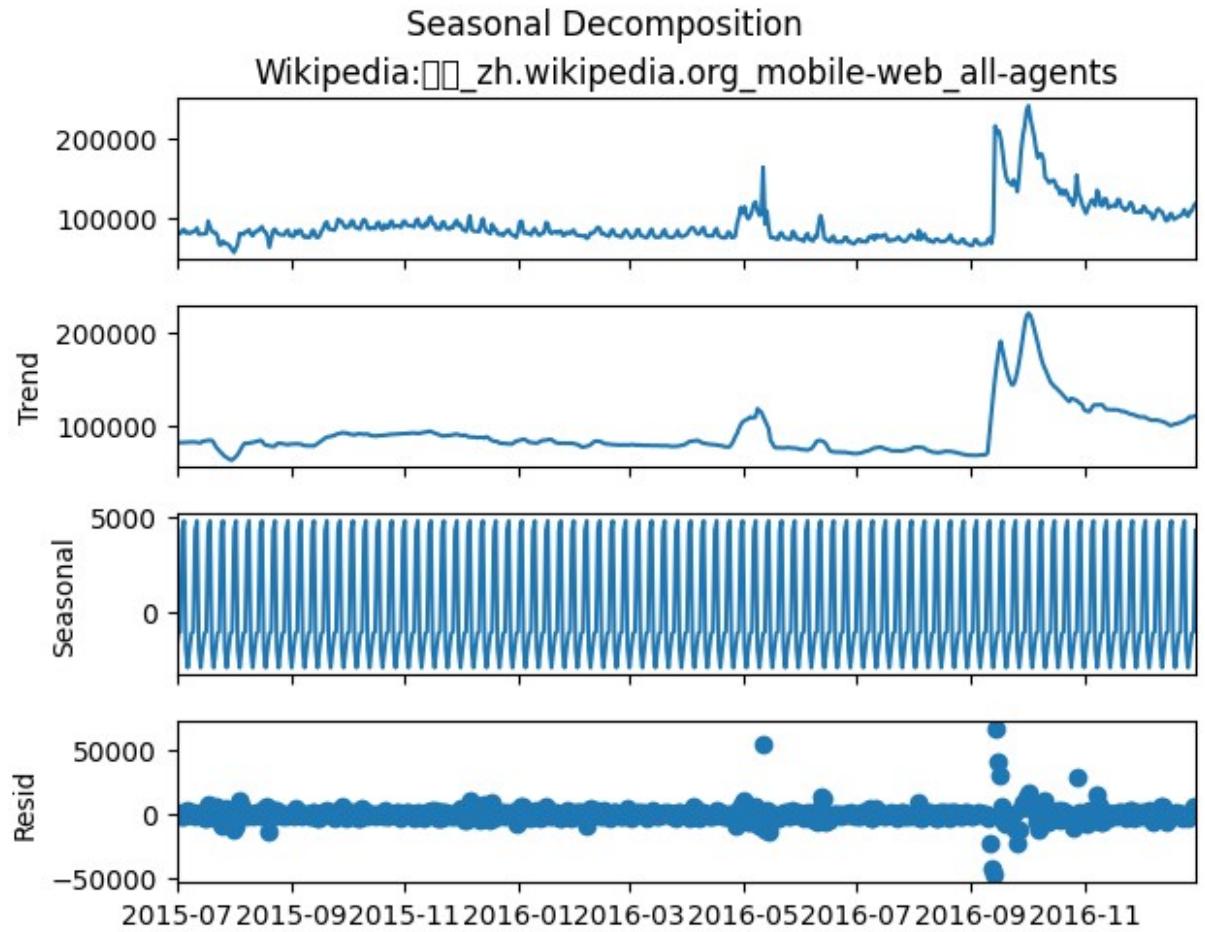
```
Analyzing page: Wikipedia:首页_zh.wikipedia.org_desktop_all-agents
ADF Statistic: -2.802 | p-value: 0.0579 | Stationary: False
```



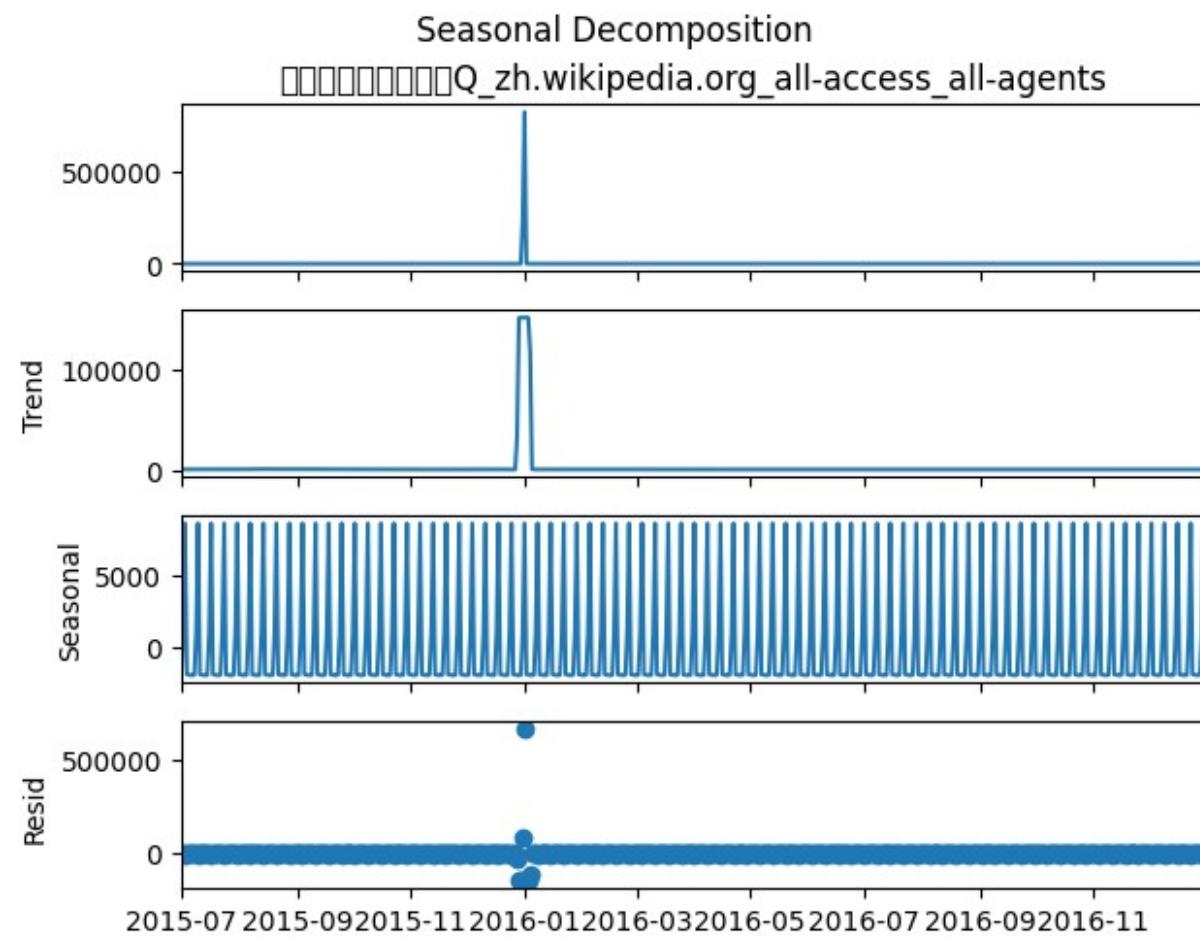
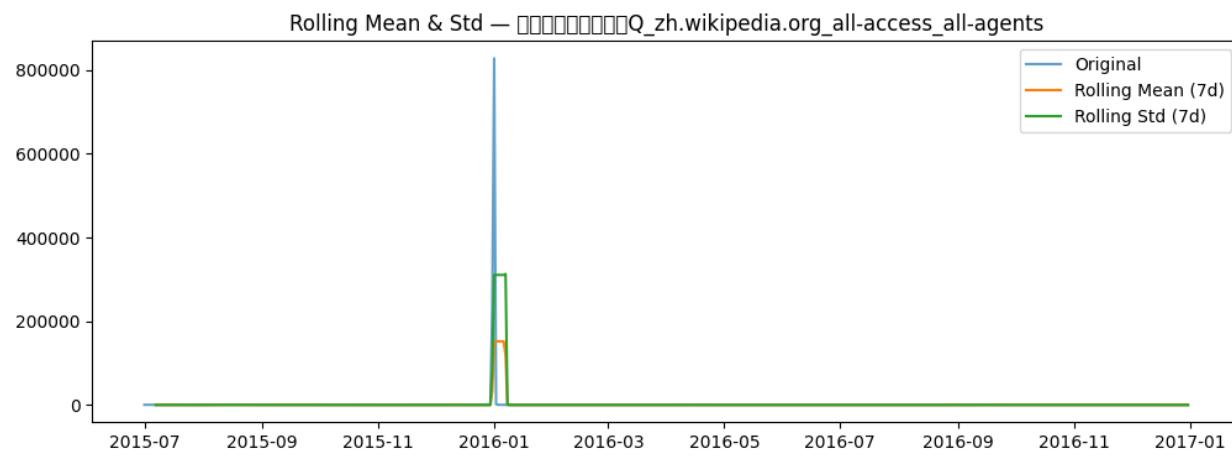


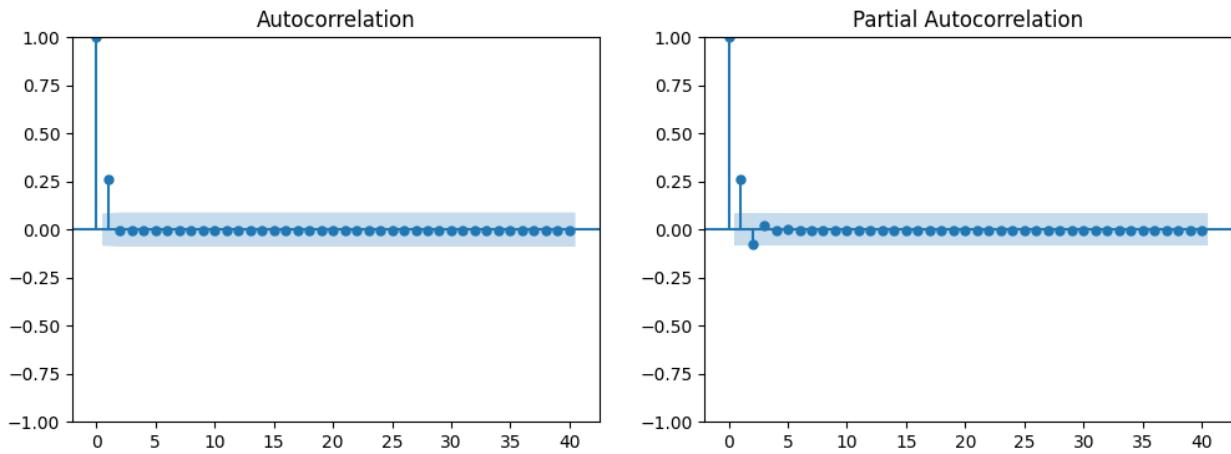
□ Analyzing page: Wikipedia:首页_zh.wikipedia.org_mobile-web_all-agents
 ADF Statistic: -2.579 | p-value: 0.0974 | Stationary: False



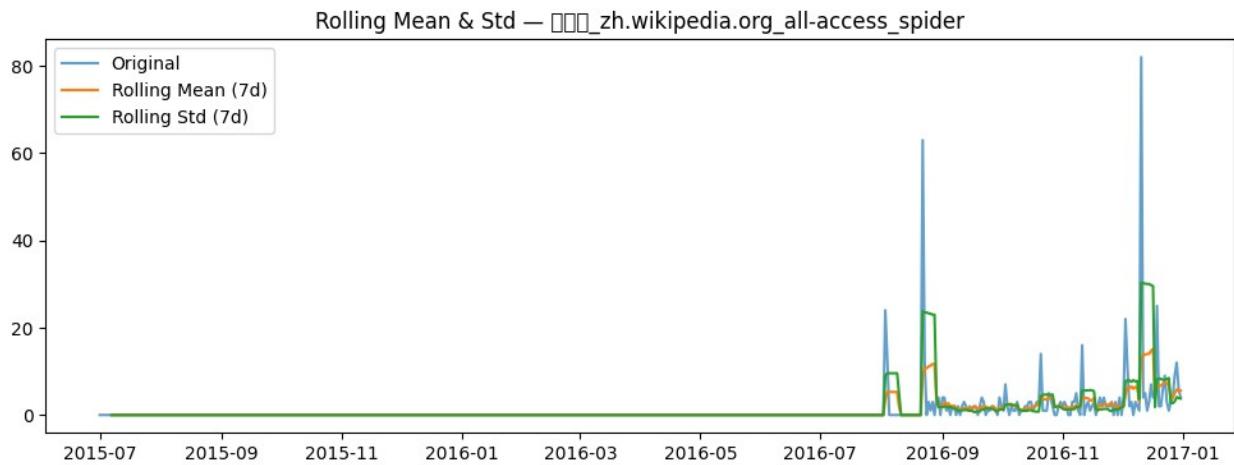


Analyzing page: 福音戰士新劇場版 : Q_zh.wikipedia.org_all-access_all-agents
 ADF Statistic: -15.307 | p-value: 0.0000 | Stationary: True

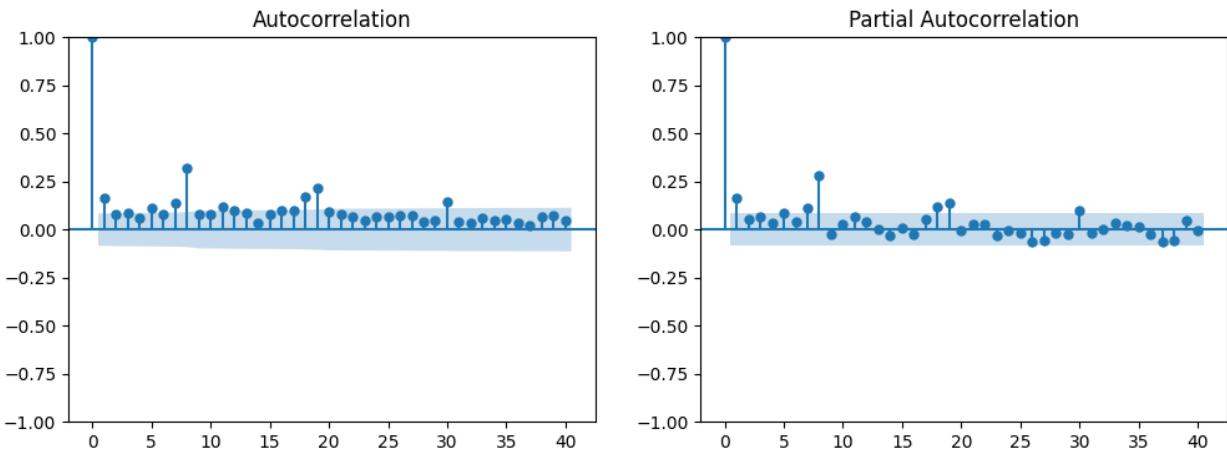
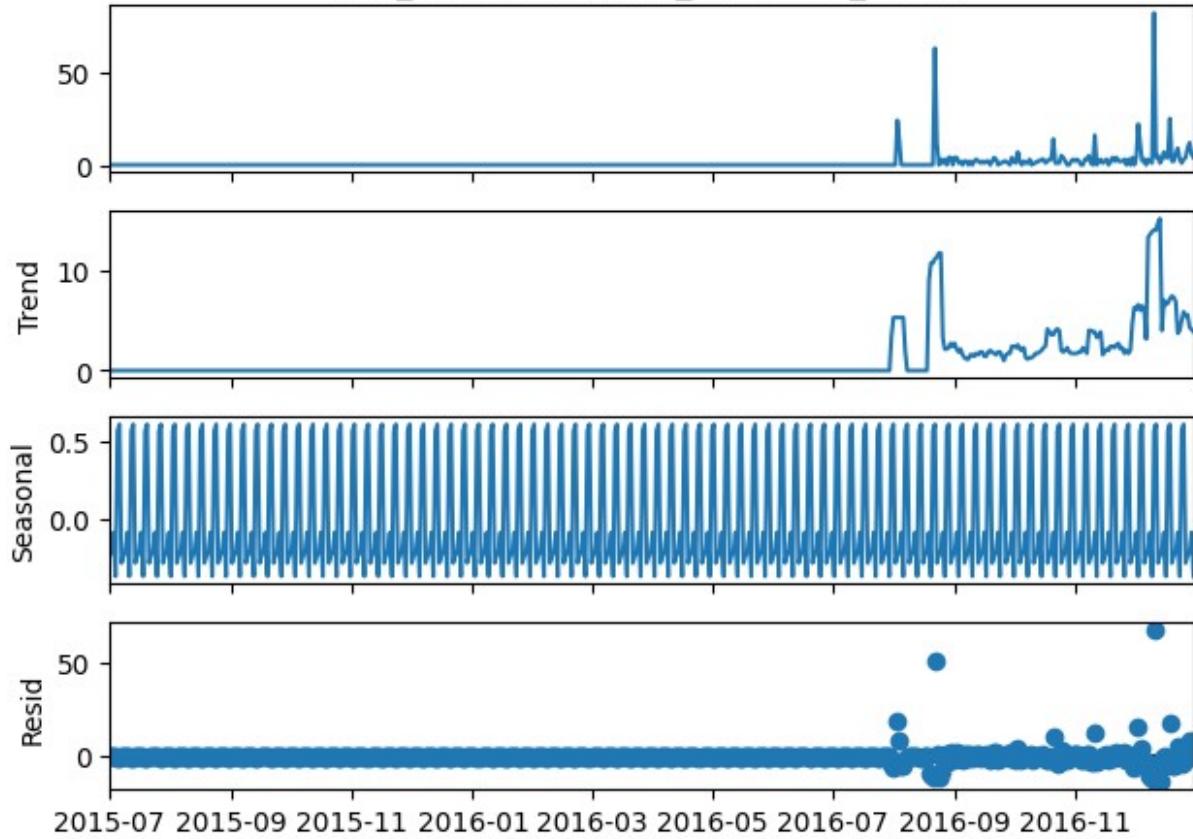




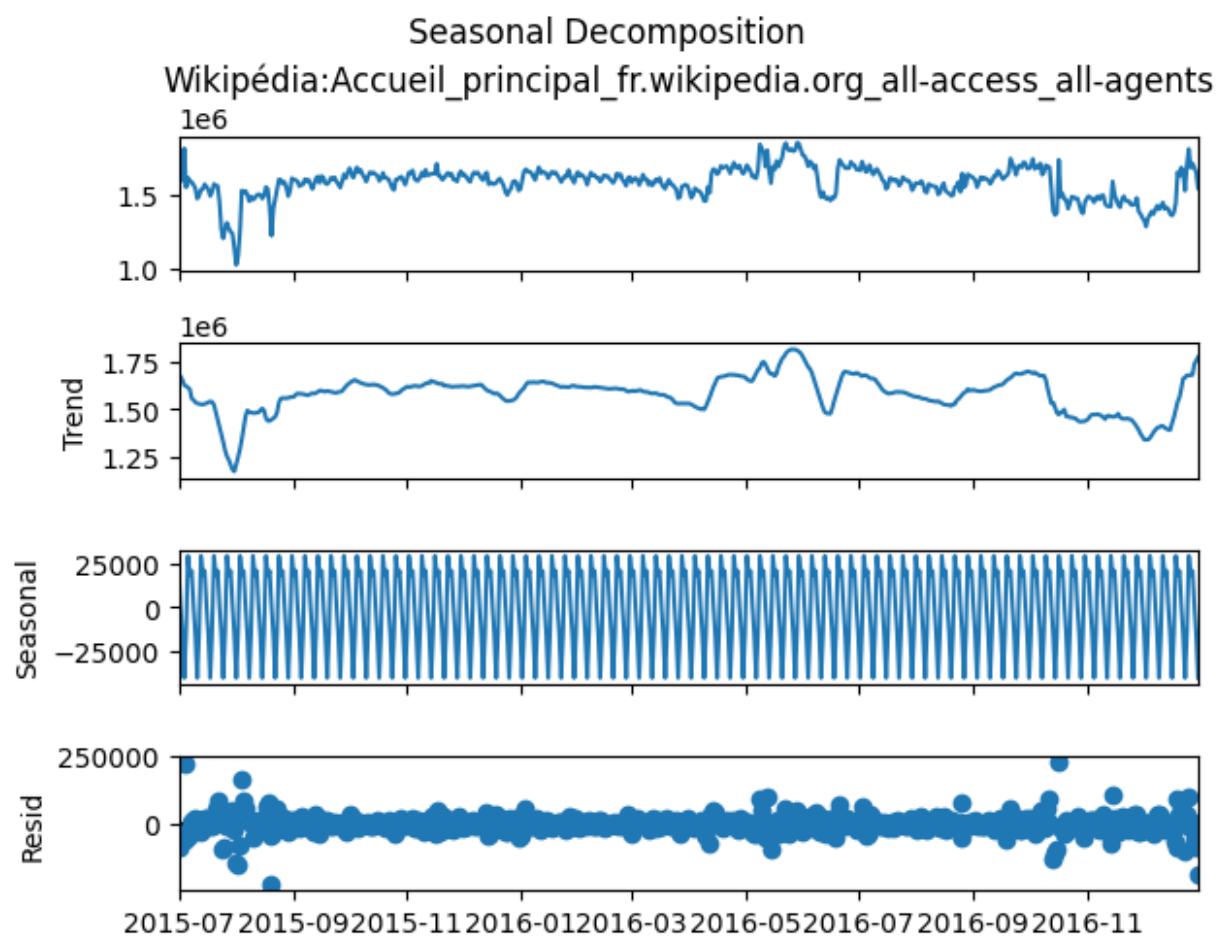
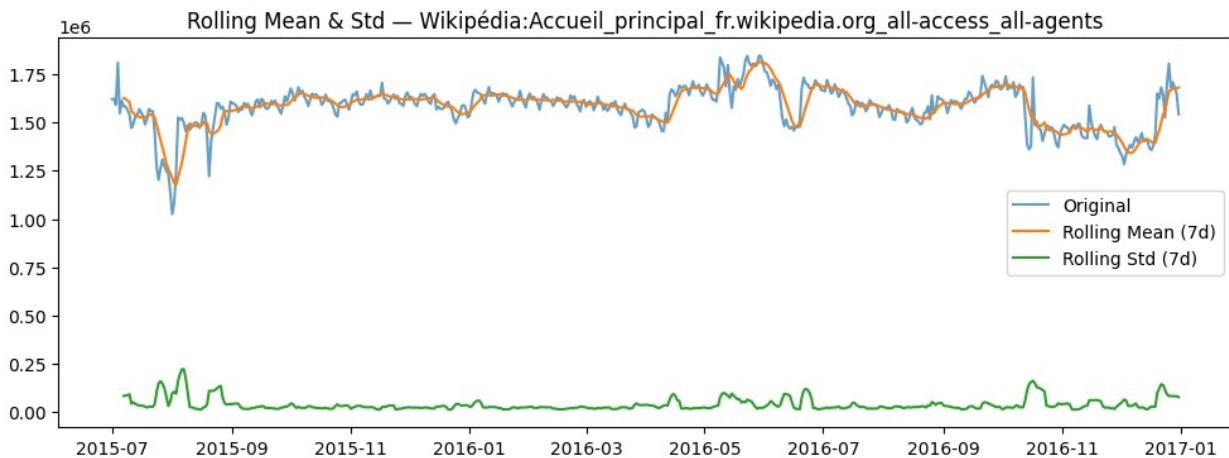
□ Analyzing page: 洪晨穎_zh.wikipedia.org_all-access_spider
 ADF Statistic: -1.966 | p-value: 0.3019 | Stationary: False

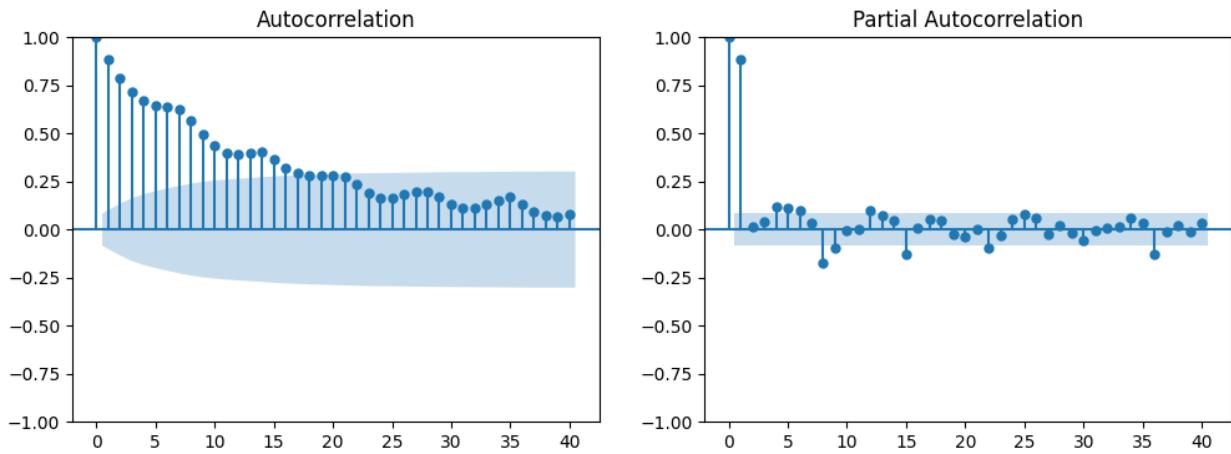


Seasonal Decomposition zh.wikipedia.org_all-access_spider

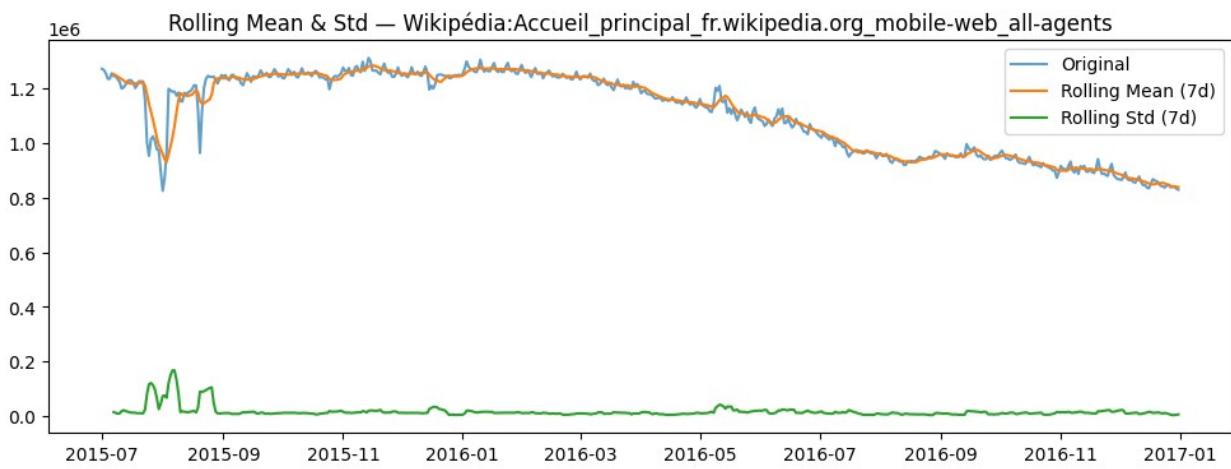


Analyzing page: Wikipédia:Accueil_principal_fr.wikipedia.org_all-access_all-agents
ADF Statistic: -3.557 | p-value: 0.0066 | Stationary: True



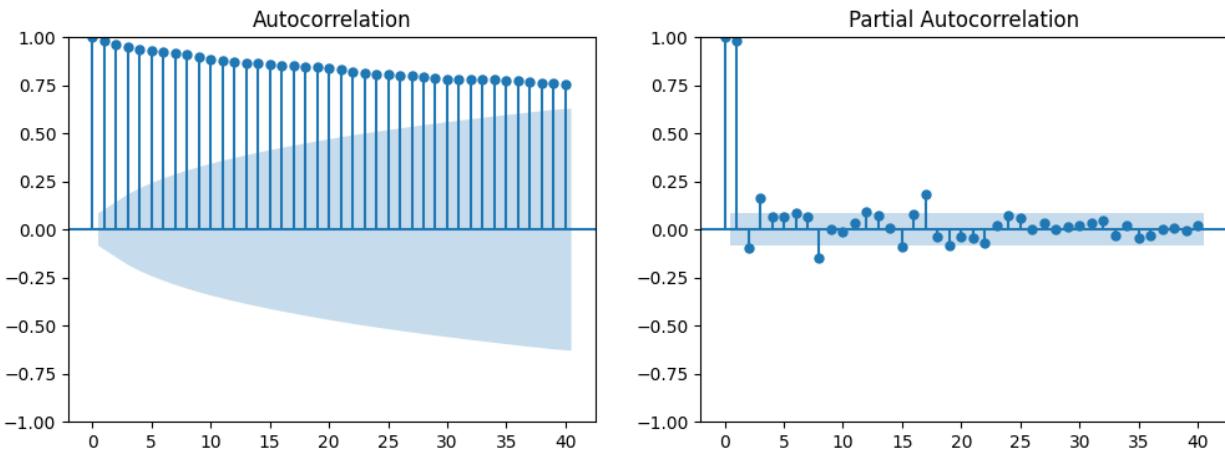
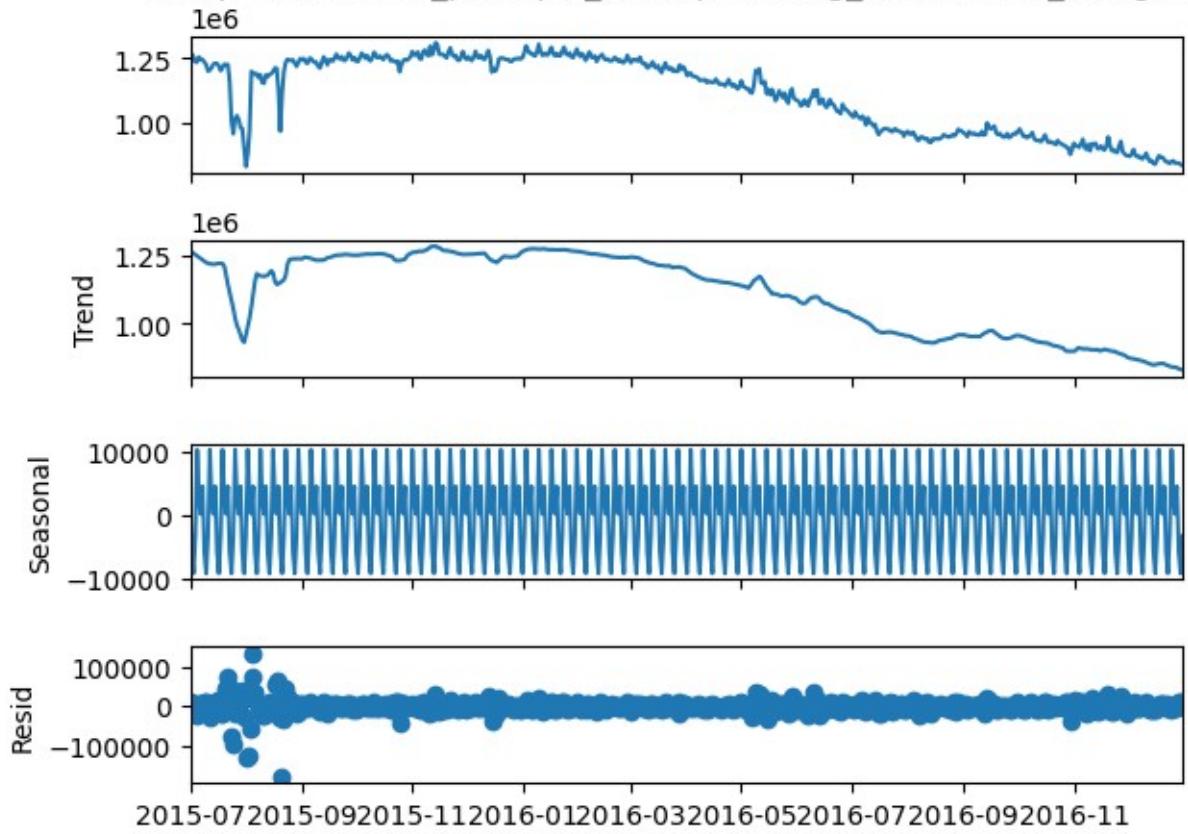


□ Analyzing page: Wikipédia:Accueil_principal_fr.wikipedia.org_mobile-web_all-agents
 ADF Statistic: 0.336 | p-value: 0.9789 | Stationary: False



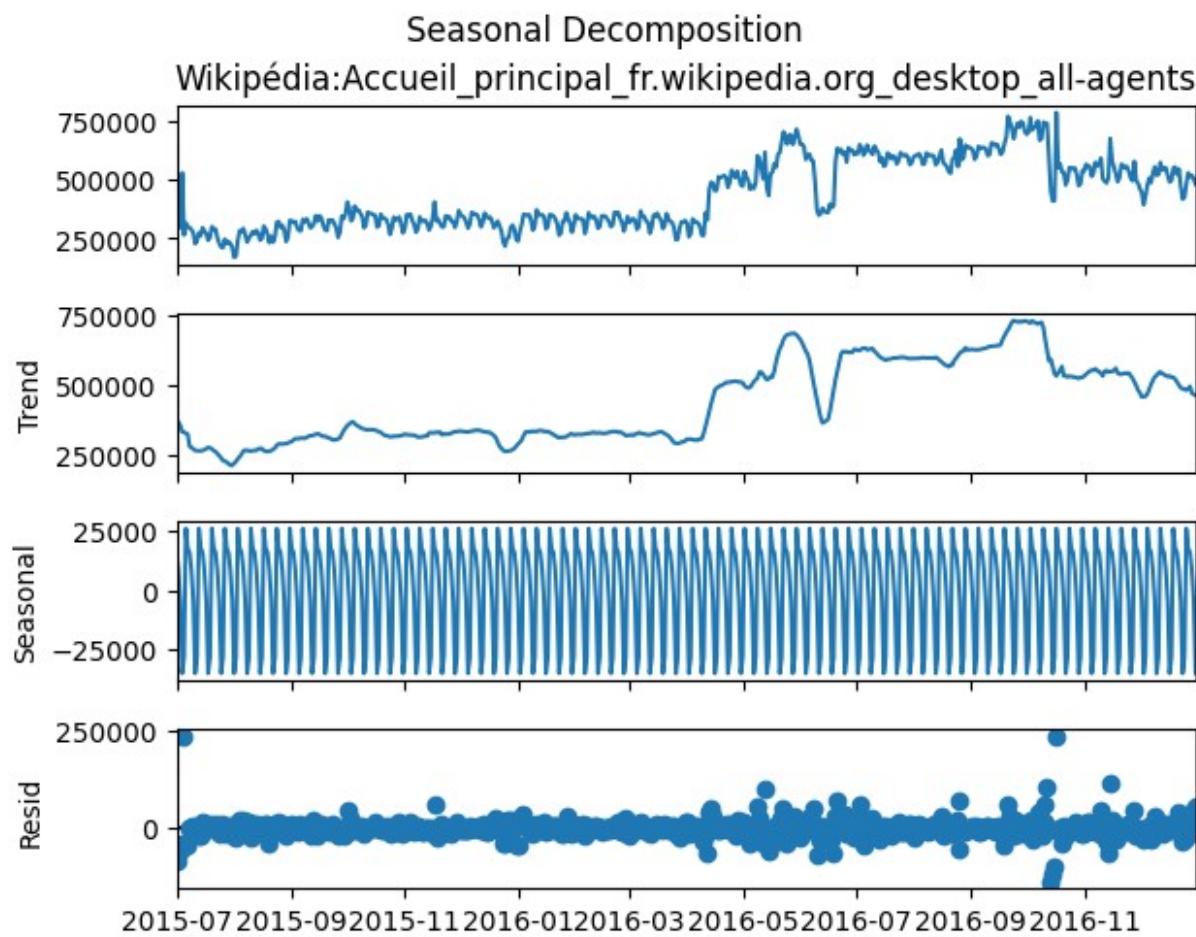
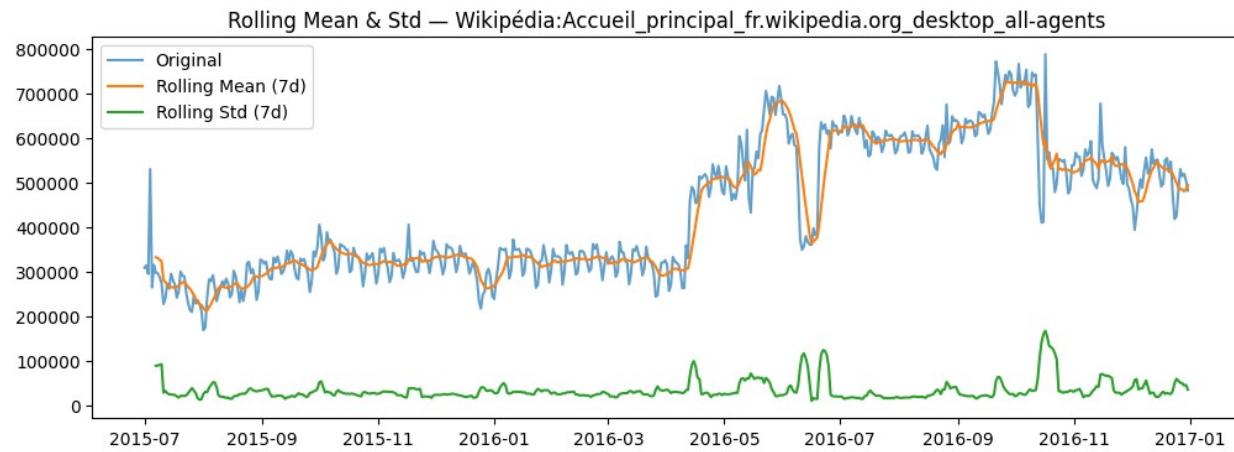
Seasonal Decomposition

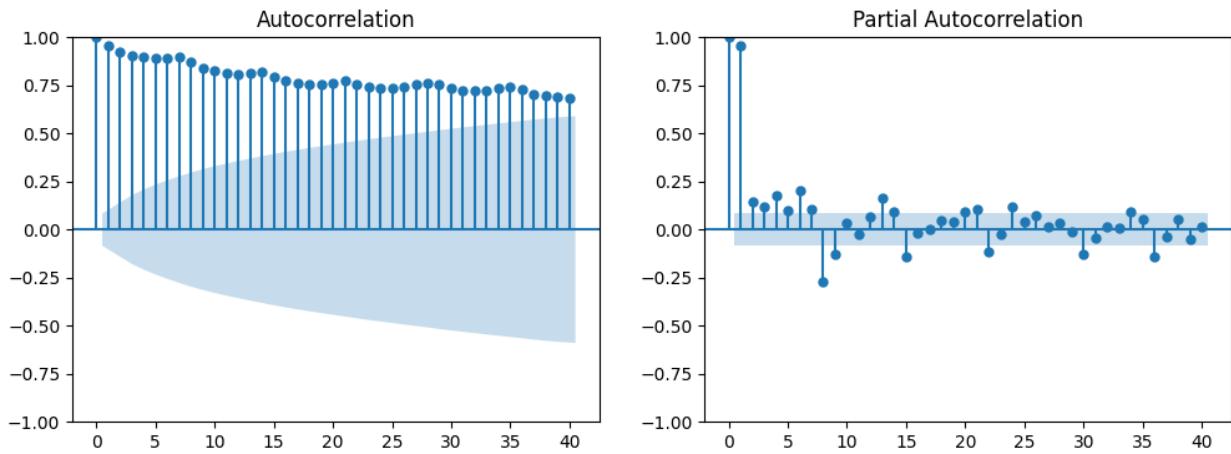
Wikipédia:Accueil_principal_fr.wikipedia.org_mobile-web_all-agents



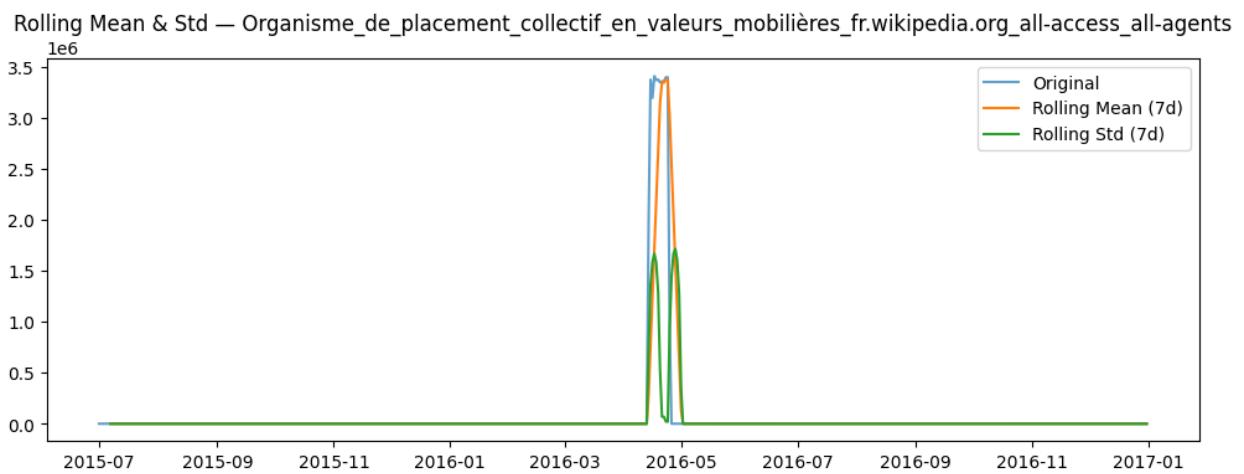
□ Analyzing page:

Wikipédia:Accueil_principal_fr.wikipedia.org_desktop_all-agents
ADF Statistic: -1.681 | p-value: 0.4409 | Stationary: False



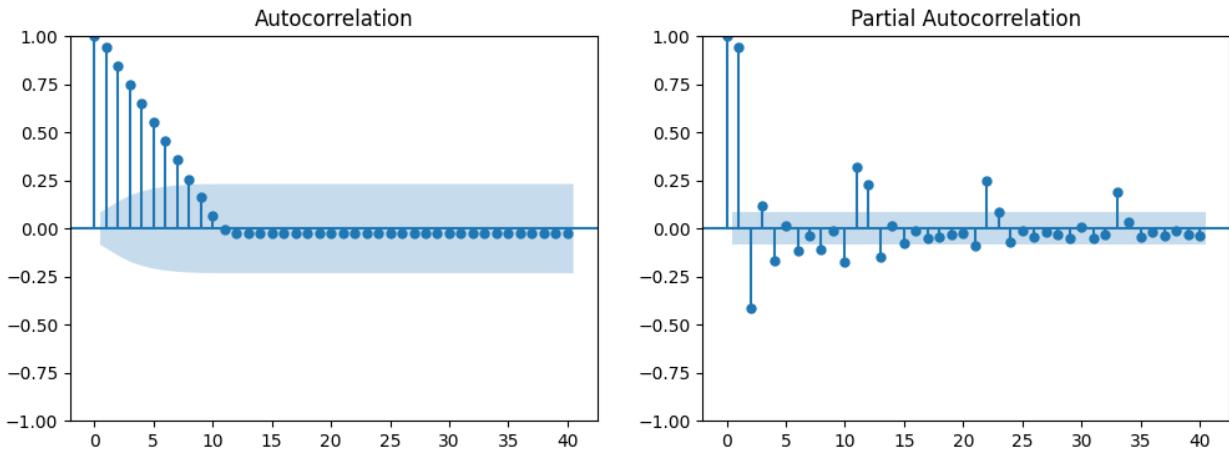
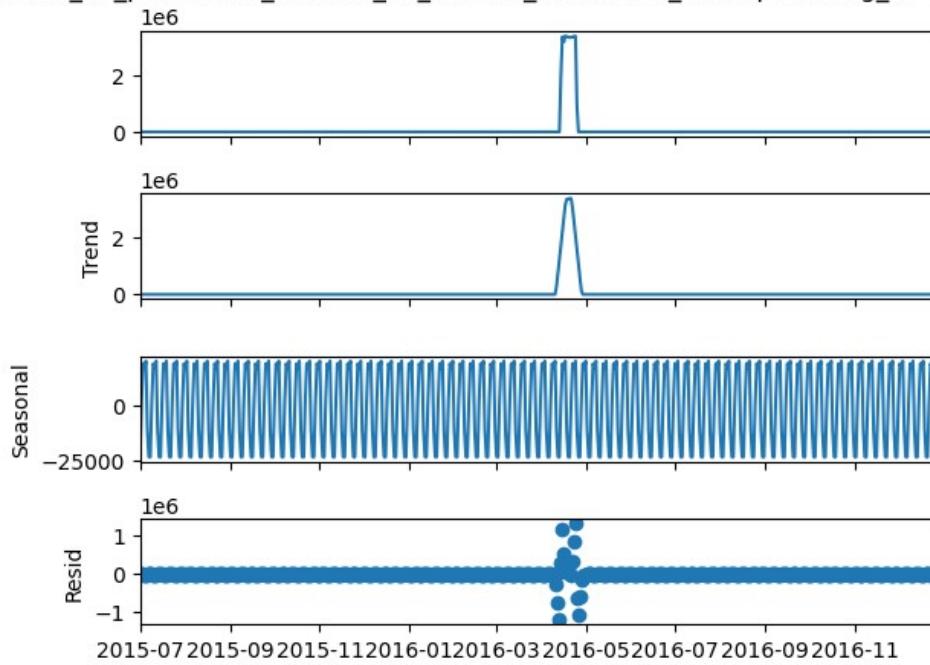


Analyzing page:
`Organisme_de_placement_collectif_en_valeurs_mobilières_fr.wikipedia.org_all-access_all-agents`
 ADF Statistic: -4.409 | p-value: 0.0003 | Stationary: True

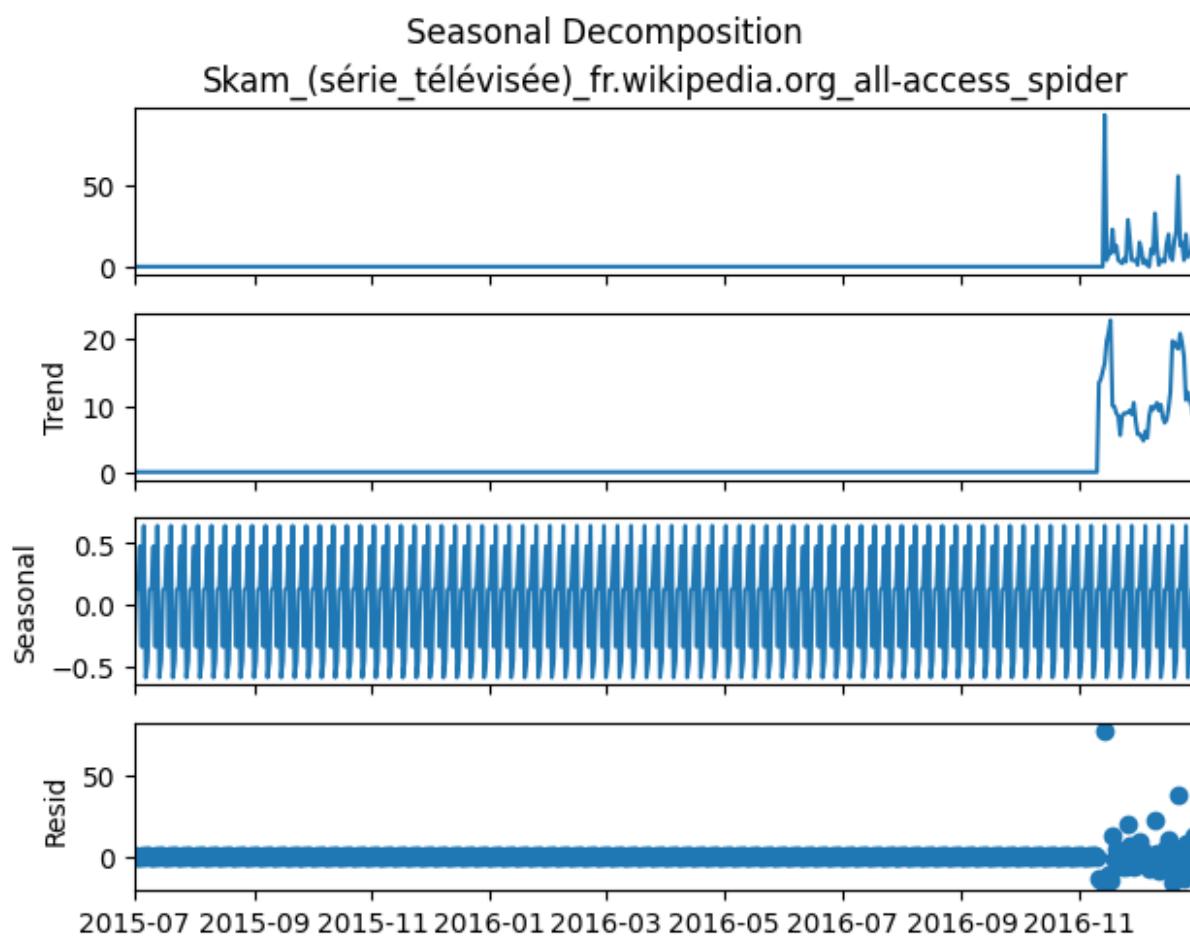
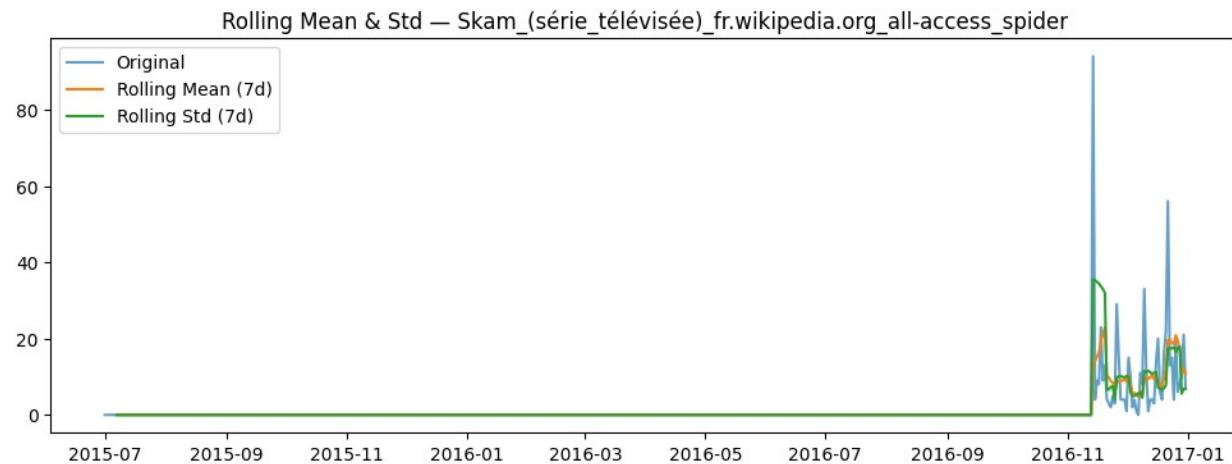


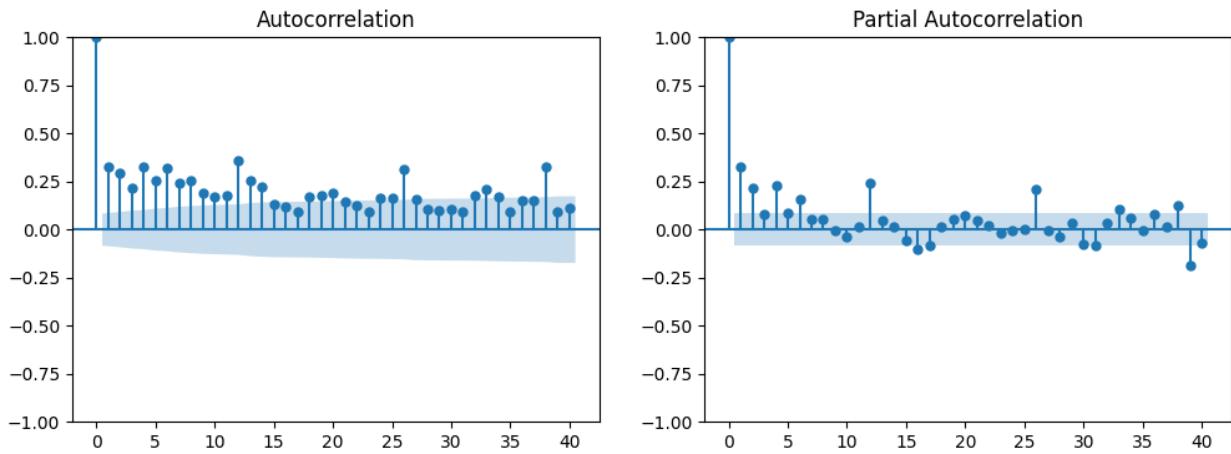
Seasonal Decomposition

Organisme_de_placement_collectif_en_valeurs_mobilières_fr.wikipedia.org_all-access_all-agents

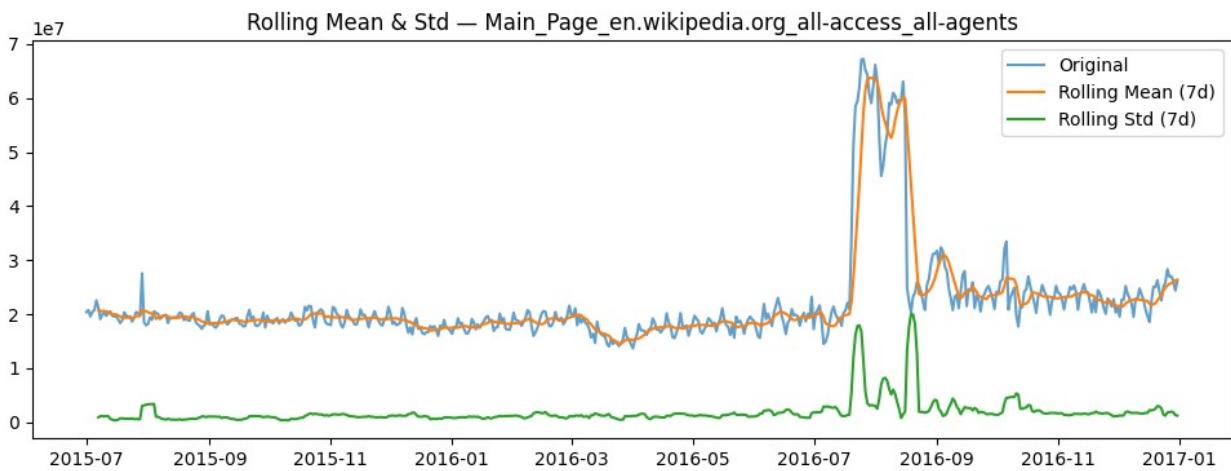


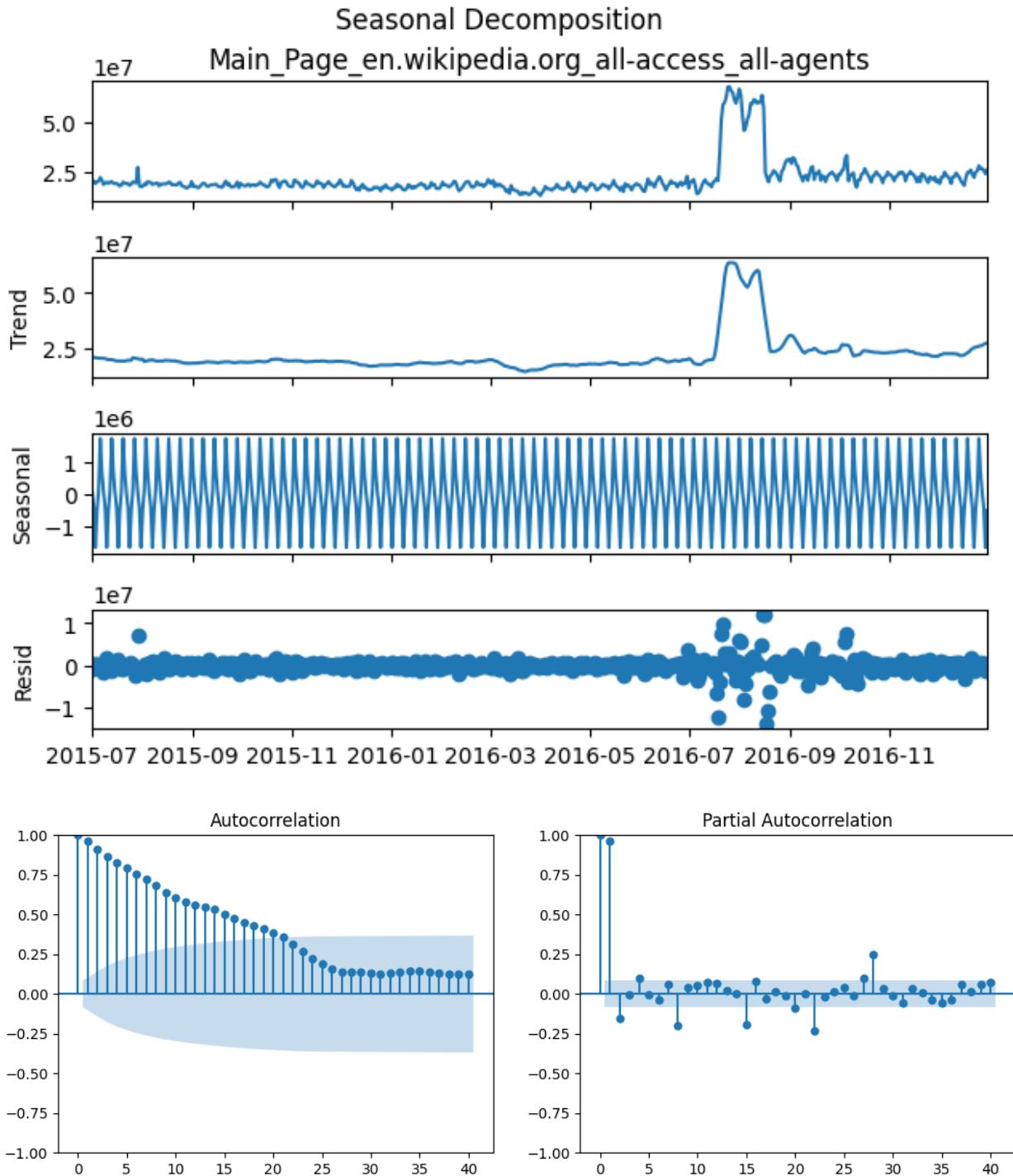
□ Analyzing page: Skam_(série_télévisée)_fr.wikipedia.org_all-access_spider
ADF Statistic: -0.369 | p-value: 0.9152 | Stationary: False



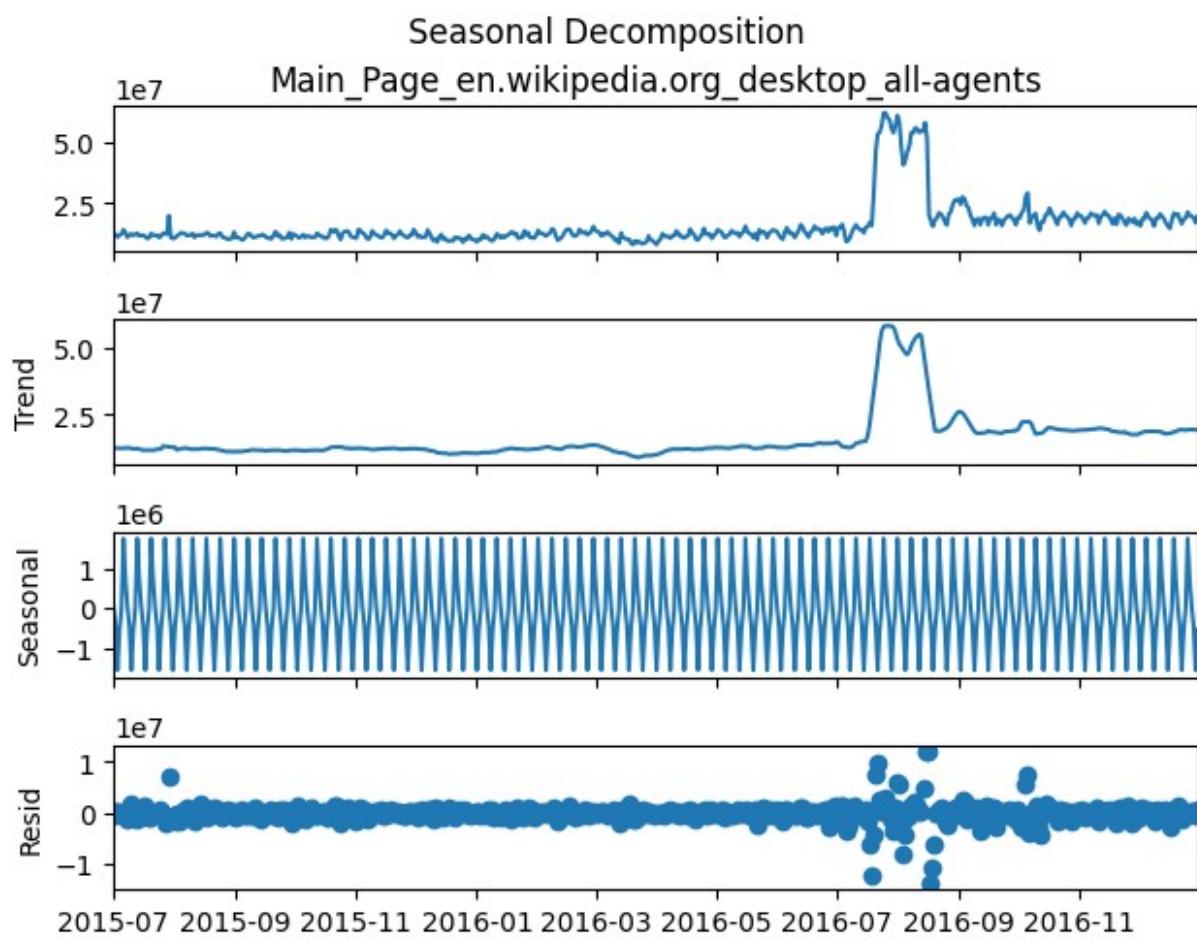
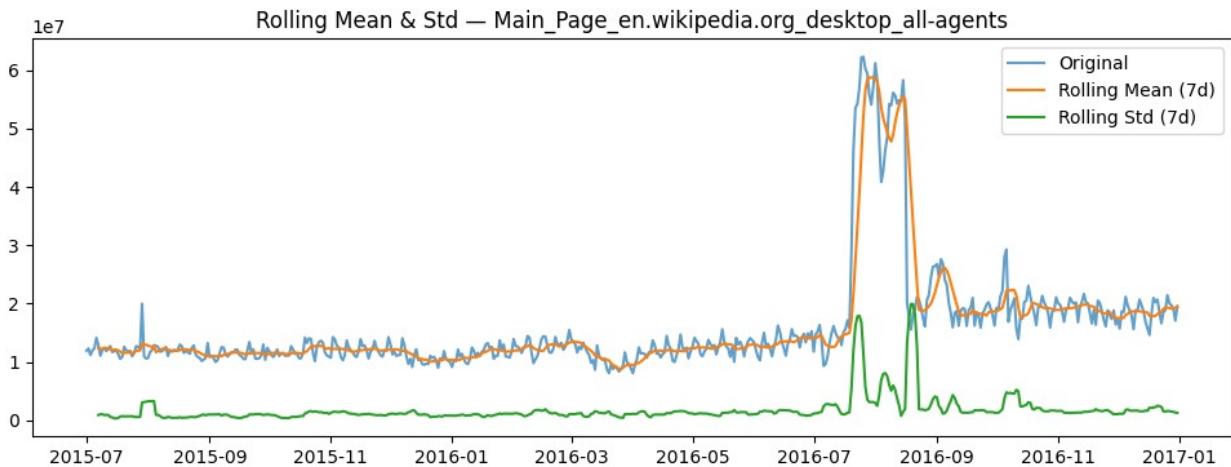


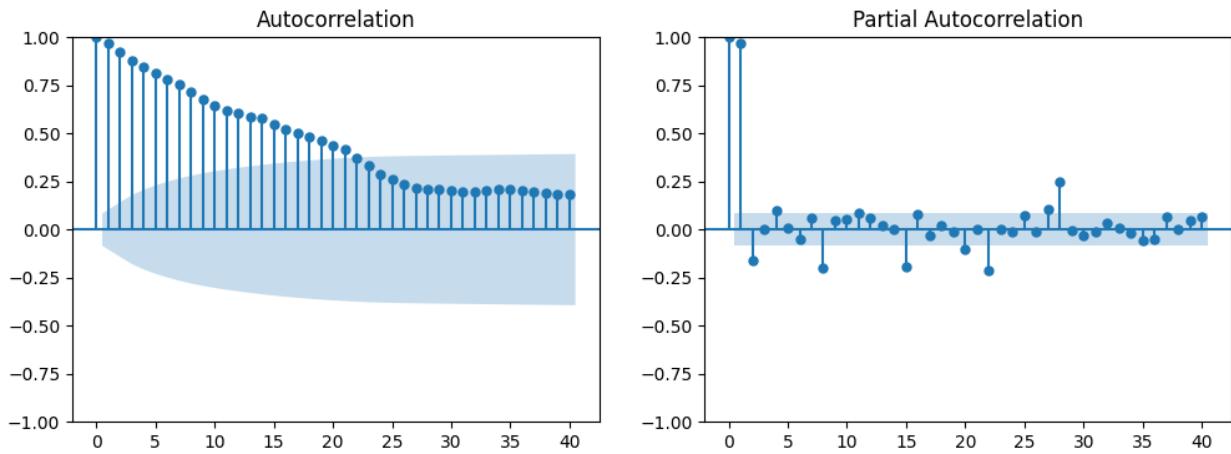
□ Analyzing page: Main_Page_en.wikipedia.org_all-access_all-agents
 ADF Statistic: -2.990 | p-value: 0.0358 | Stationary: True



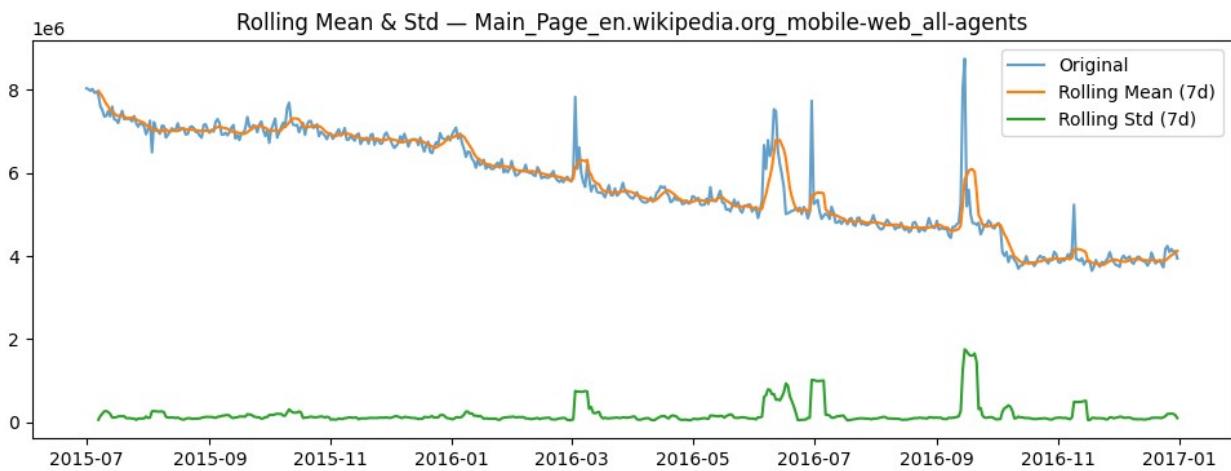


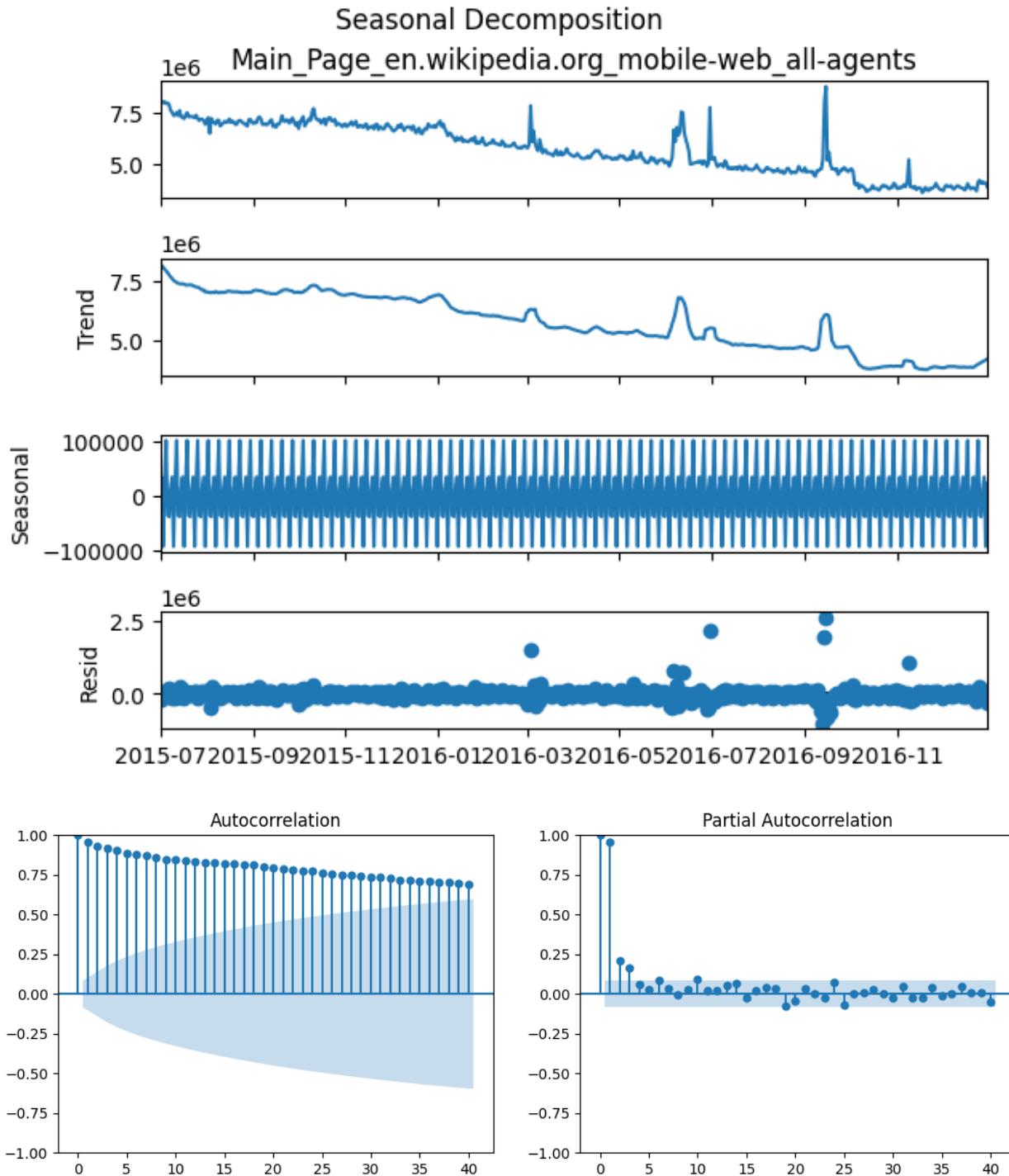
Analyzing page: Main_Page_en.wikipedia.org_desktop_all-agents
 ADF Statistic: -2.815 | p-value: 0.0562 | Stationary: False



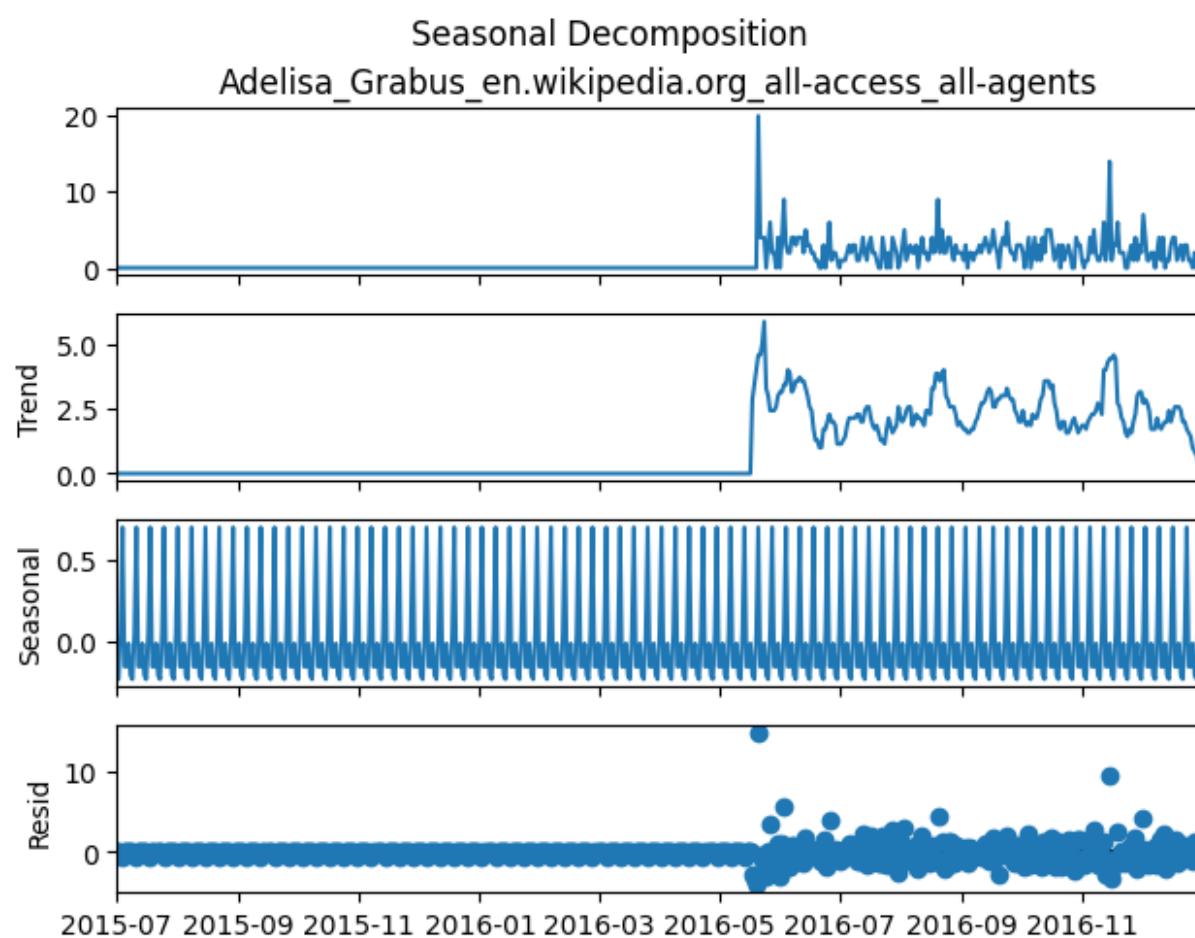
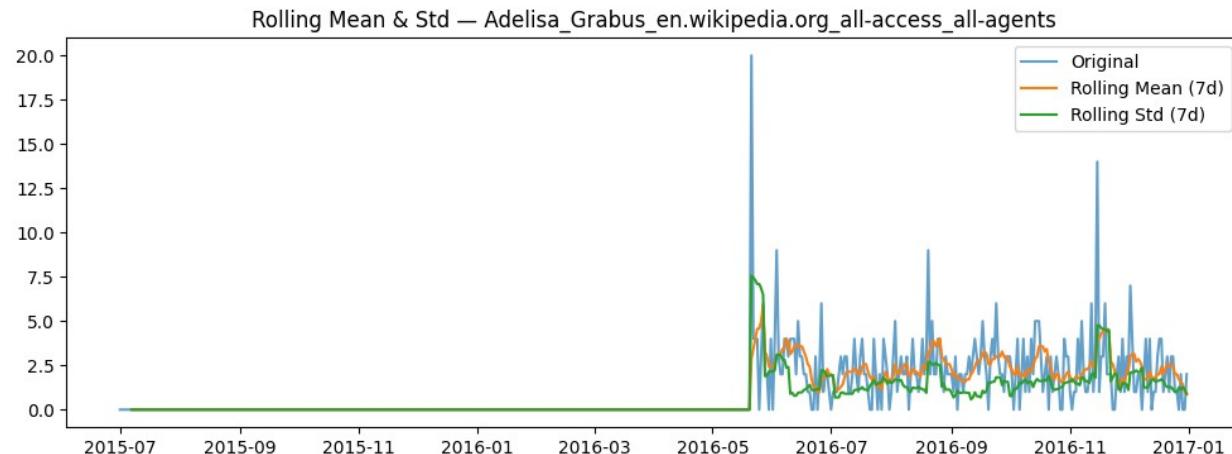


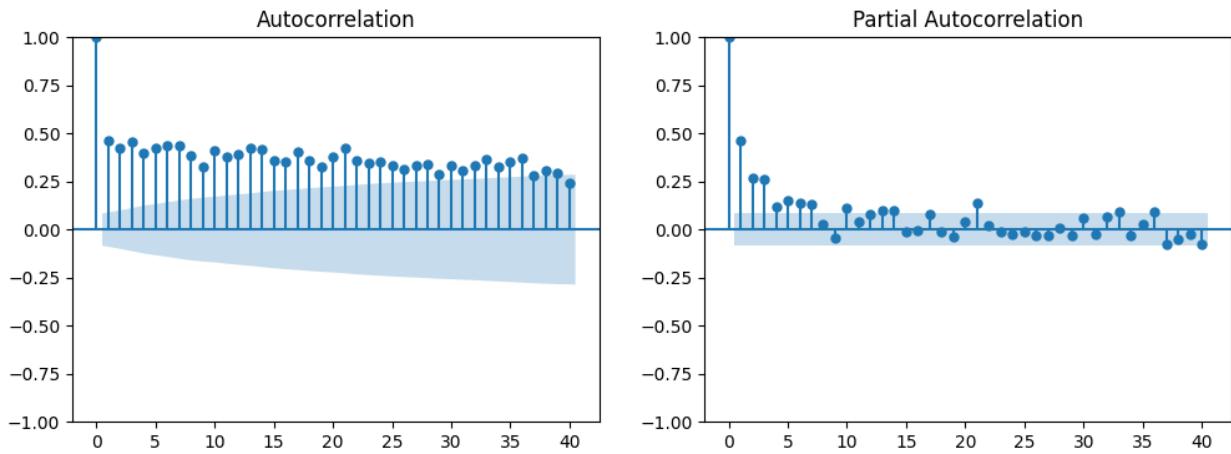
□ Analyzing page: Main_Page_en.wikipedia.org_mobile-web_all-agents
 ADF Statistic: -1.117 | p-value: 0.7082 | Stationary: False



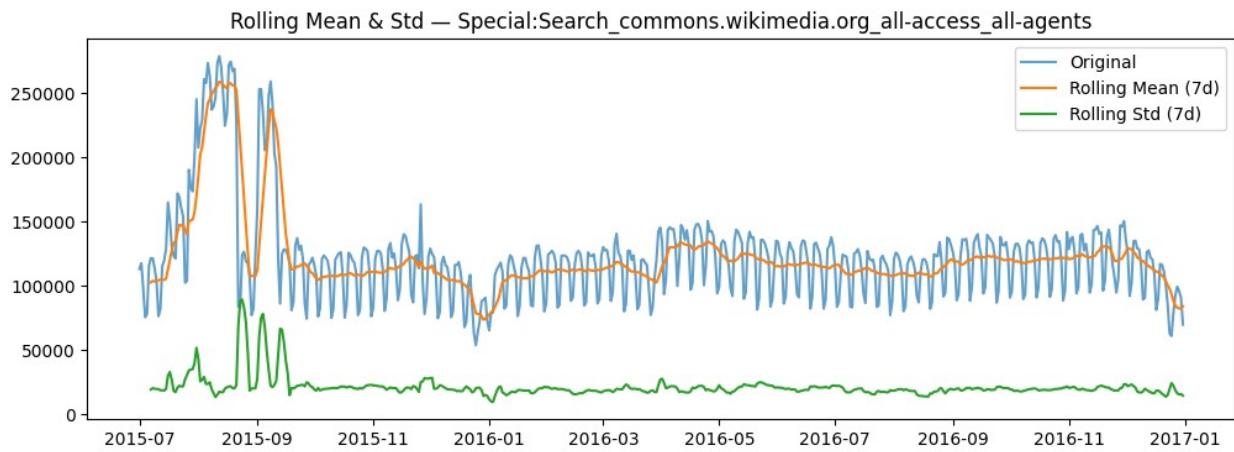


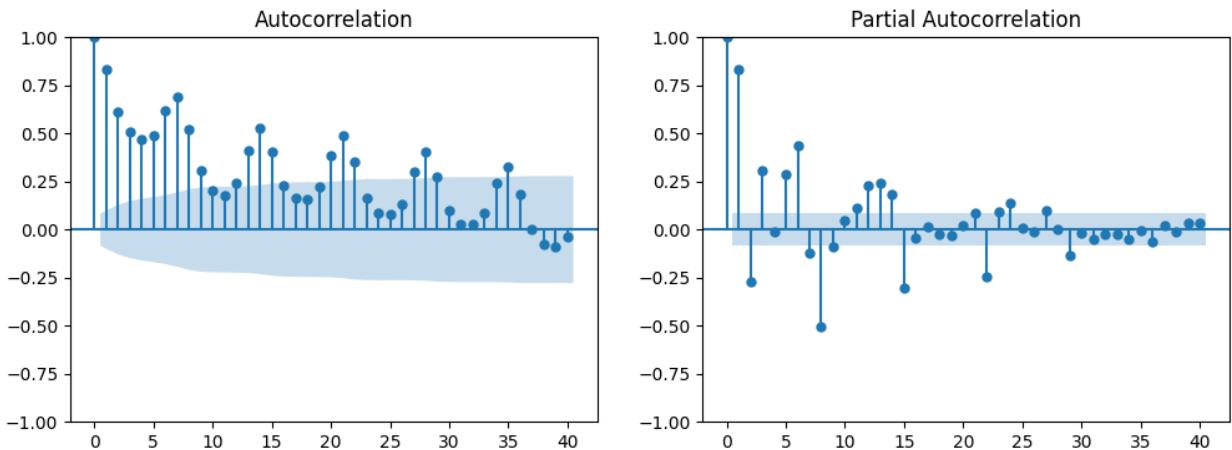
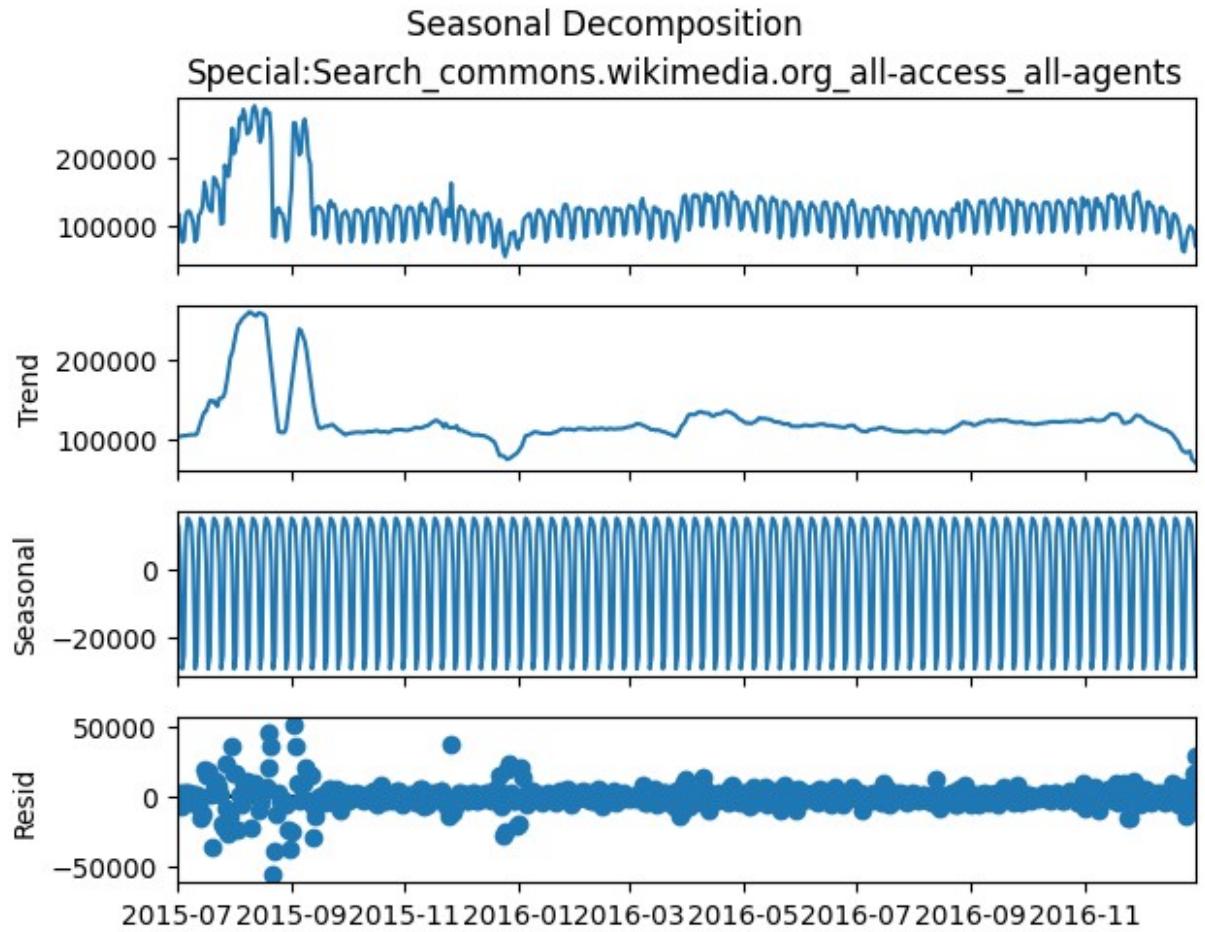
Analyzing page: Adelisa_Grabus_en.wikipedia.org_all-access_all-agents
 ADF Statistic: -2.112 | p-value: 0.2398 | Stationary: False



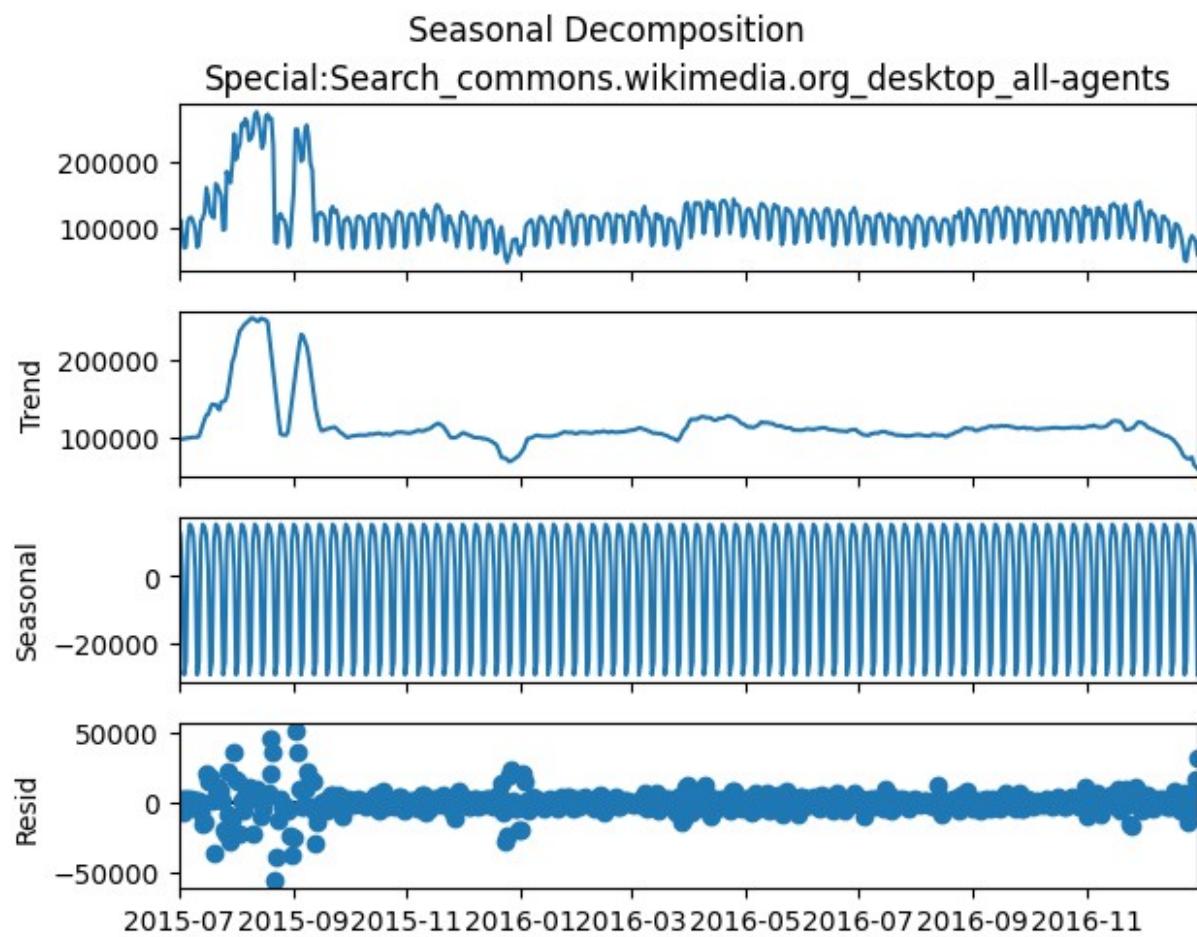
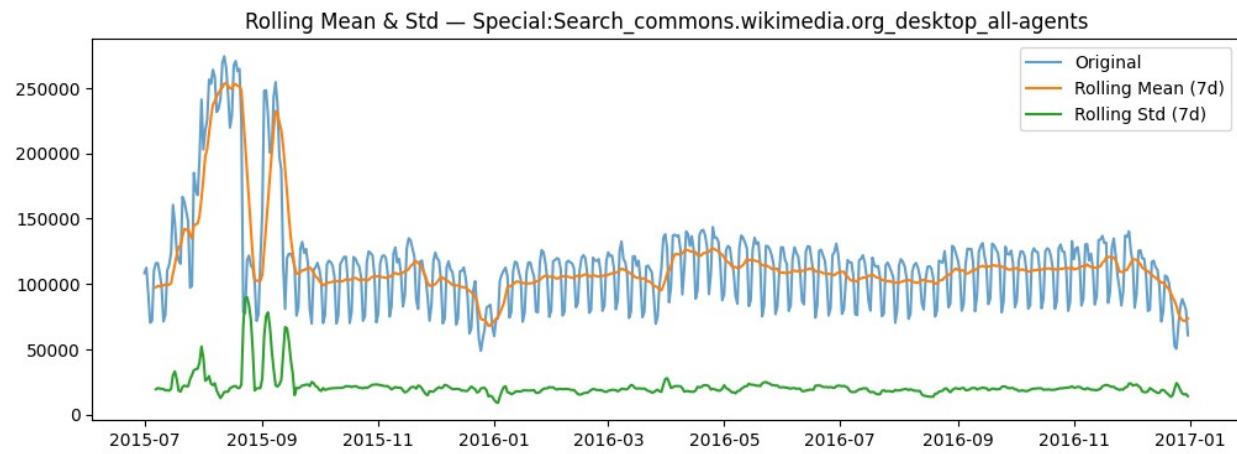


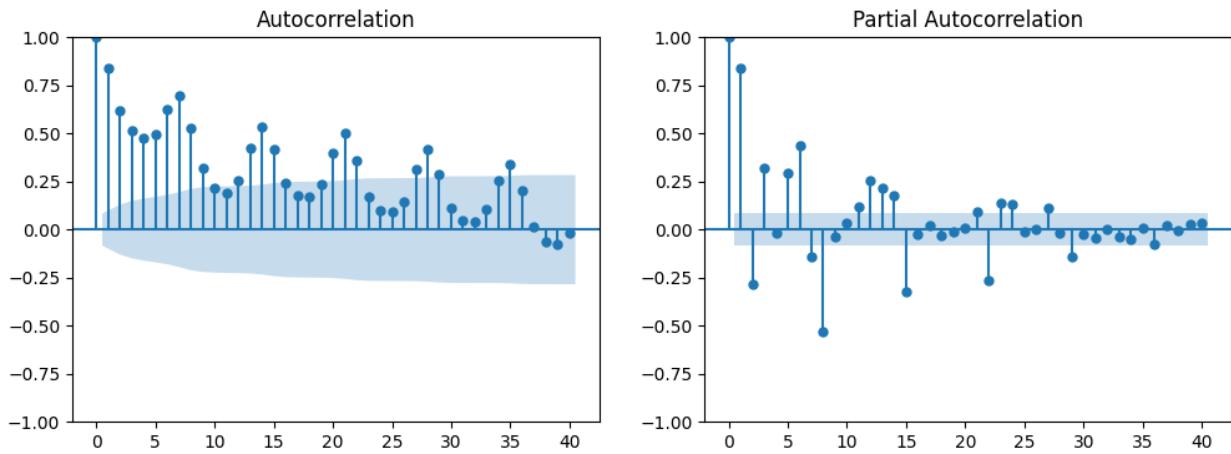
Analyzing page: Special:Search_commons.wikimedia.org_all-access_all-agents
 ADF Statistic: -2.972 | p-value: 0.0376 | Stationary: True



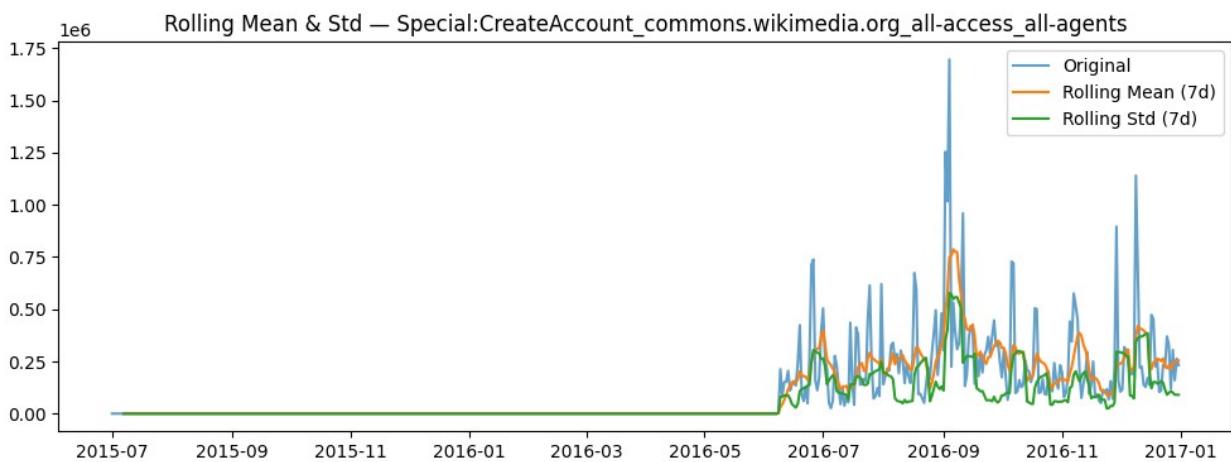


□ Analyzing page: `Special:Search_commons.wikimedia.org_desktop_all-agents`
 ADF Statistic: -2.871 | p-value: 0.0488 | Stationary: True



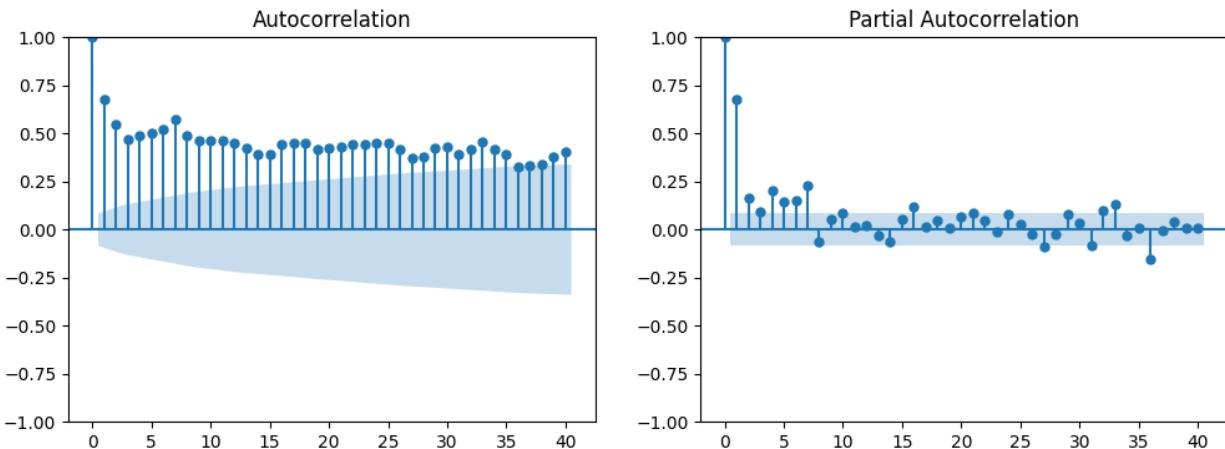
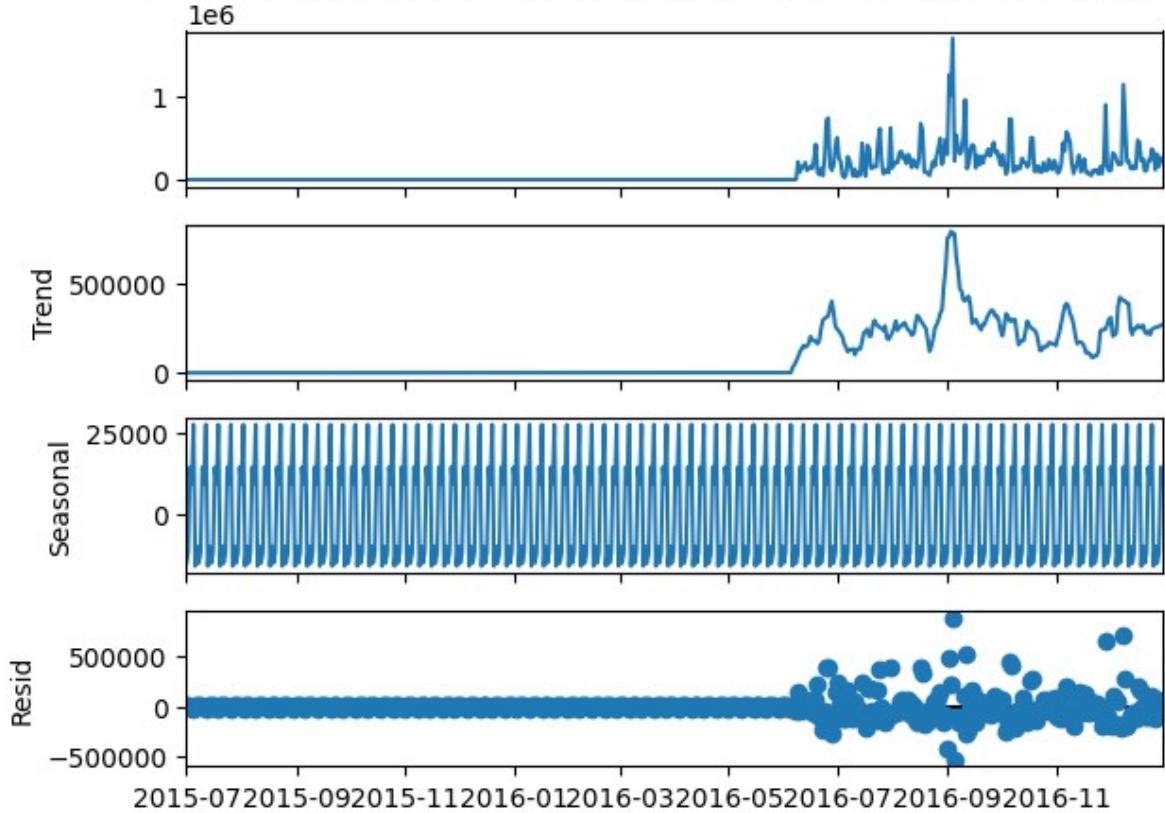


□ Analyzing page: Special>CreateAccount_commons.wikimedia.org_all-access_all-agents
 ADF Statistic: -2.113 | p-value: 0.2393 | Stationary: False

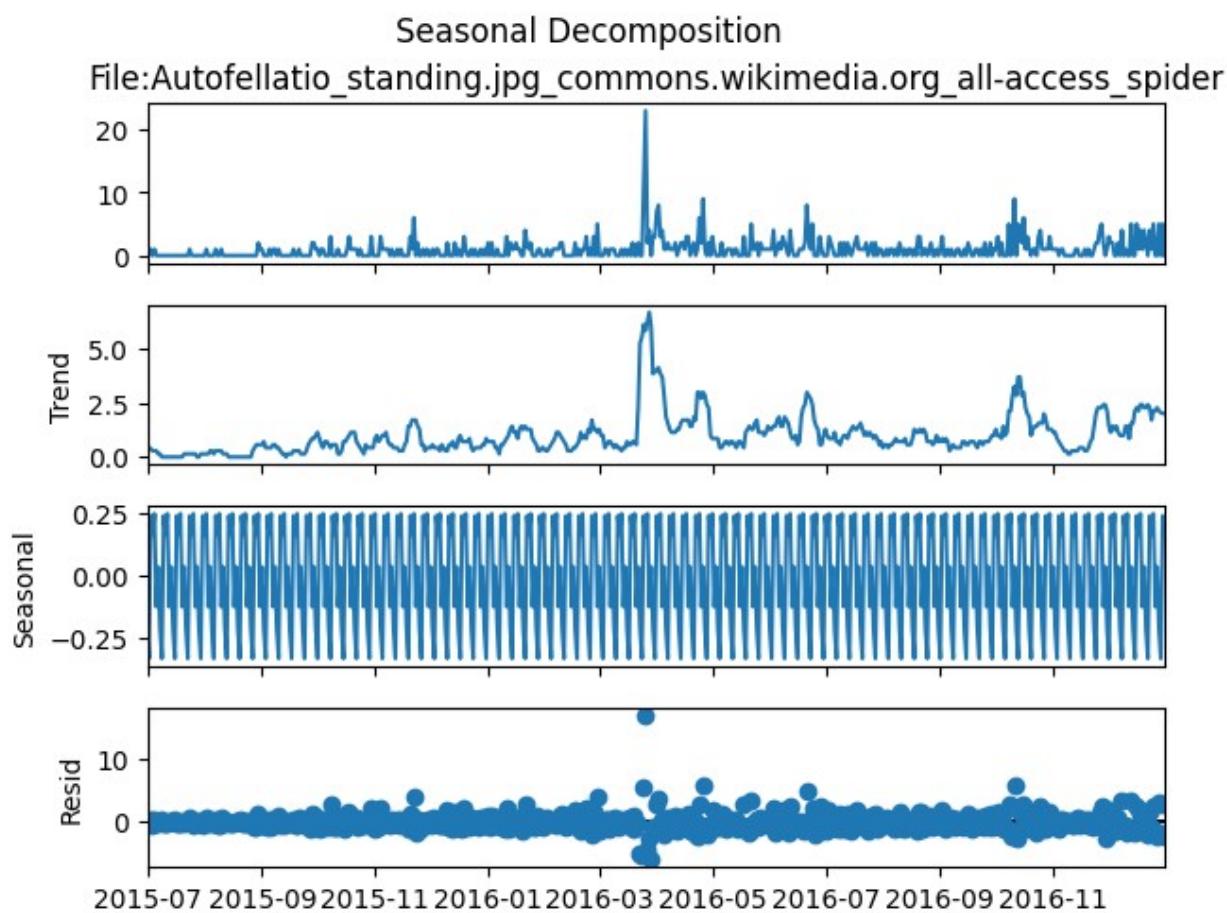
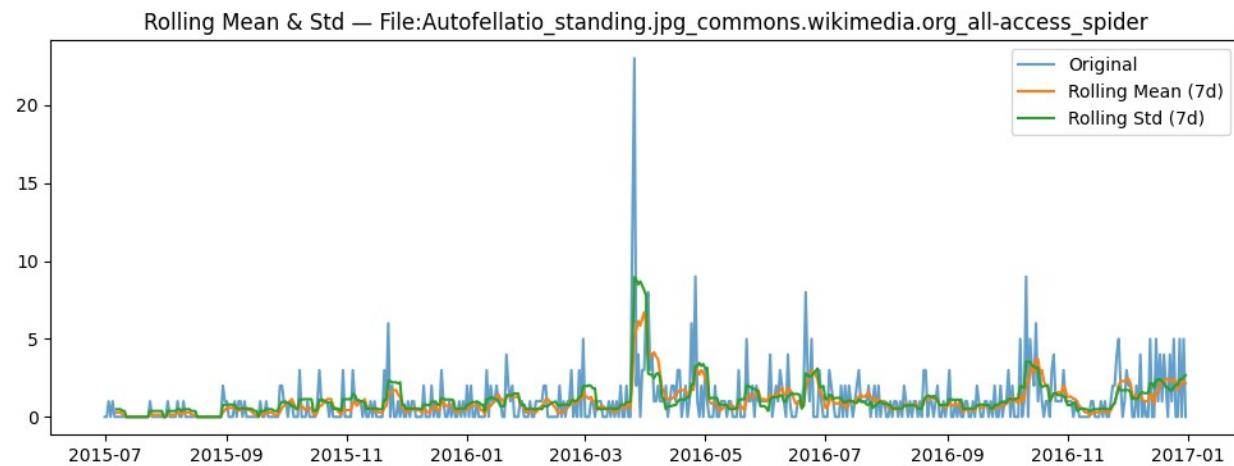


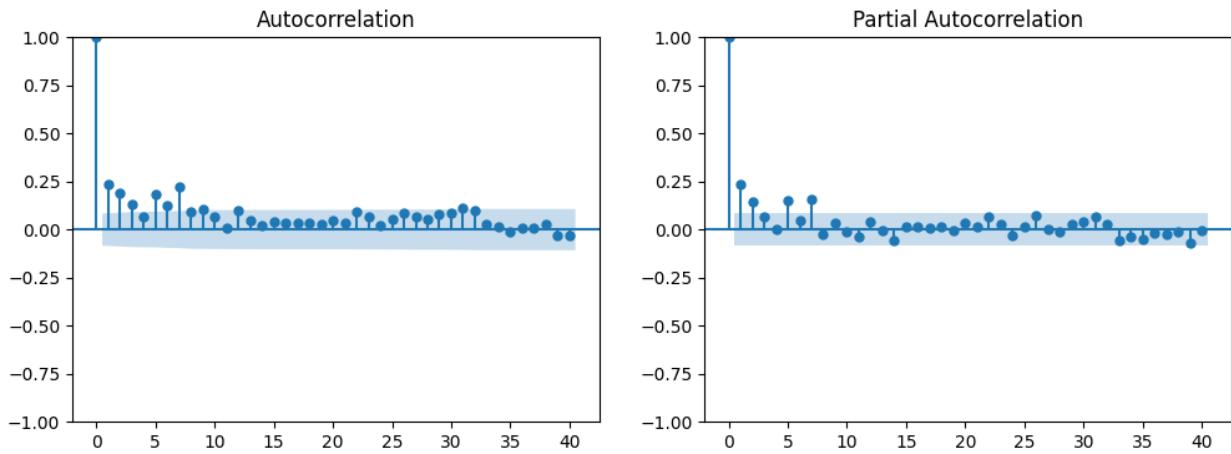
Seasonal Decomposition

Special:CreateAccount_commons.wikimedia.org_all-access_all-agents



□ Analyzing page:
File:Autofellatio_standing.jpg_commons.wikimedia.org_all-access_spider
ADF Statistic: -5.467 | p-value: 0.0000 | Stationary: True





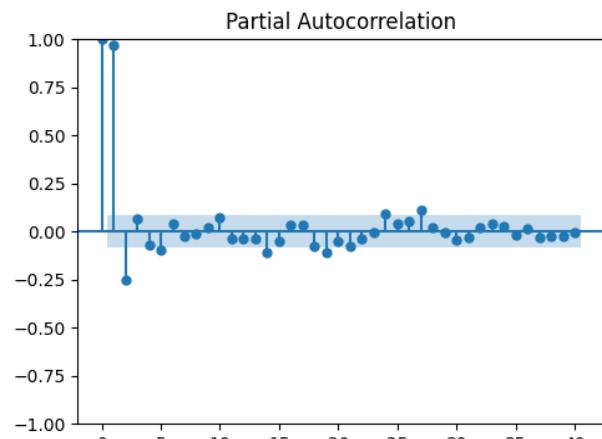
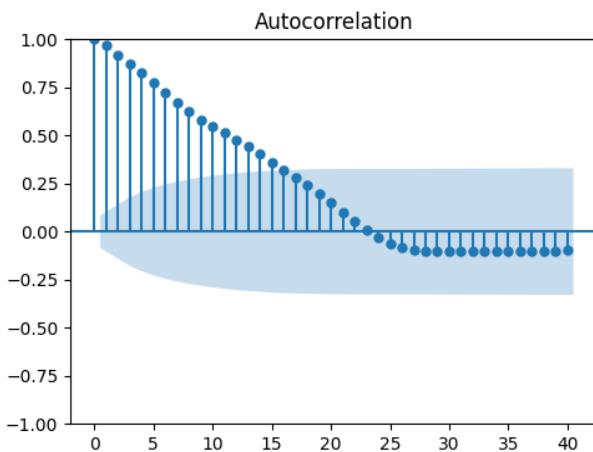
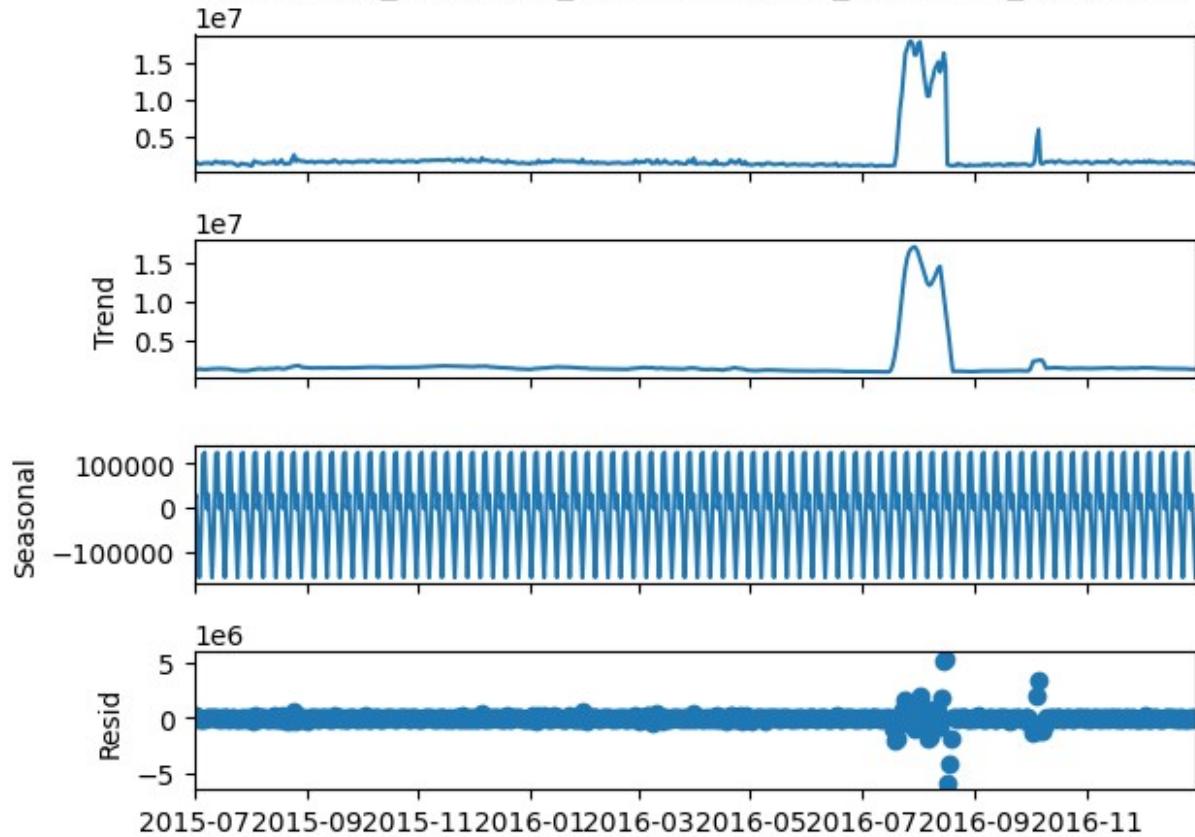
□ Analyzing page: Заглавная_страница_ru.wikipedia.org_all-access_all-agents

ADF Statistic: -4.128 | p-value: 0.0009 | Stationary: True

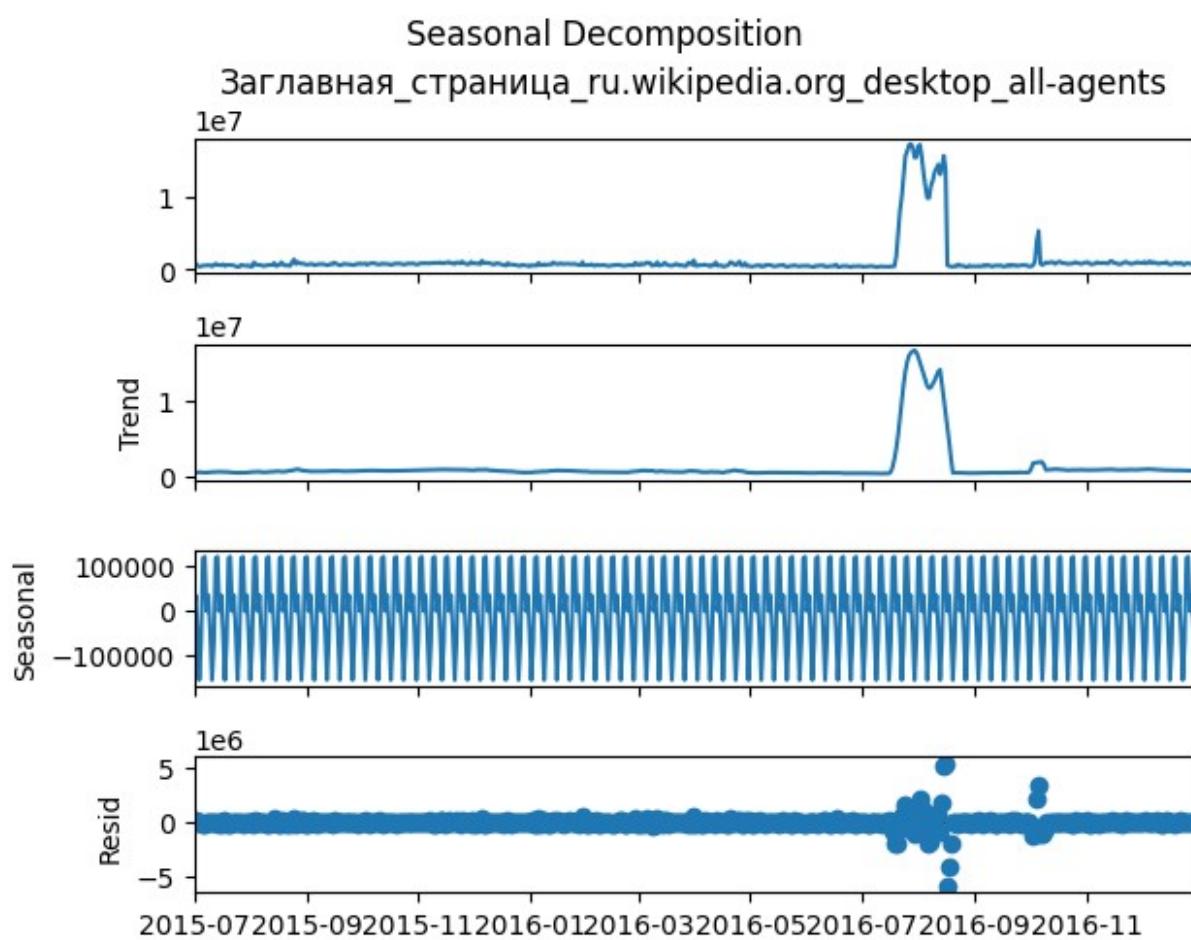


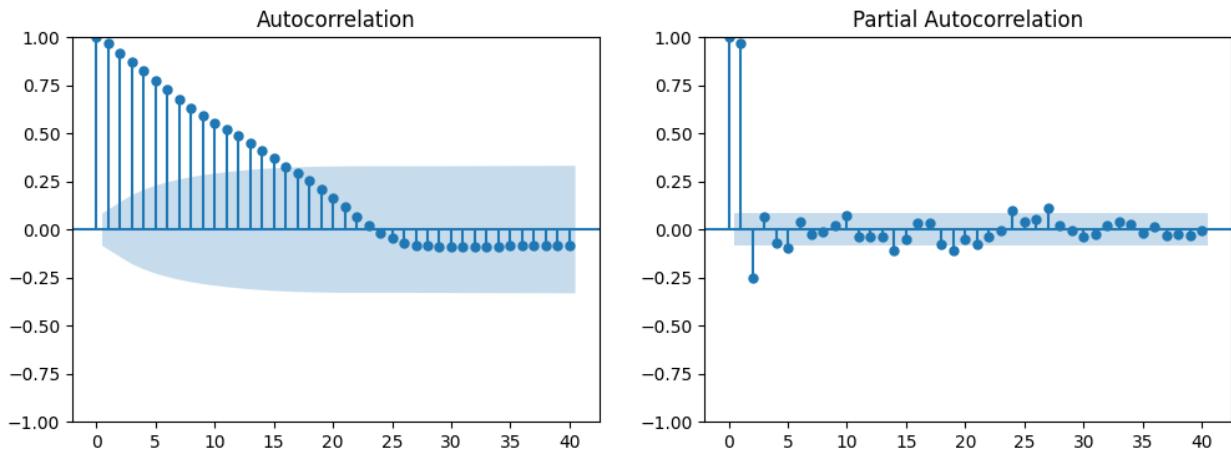
Seasonal Decomposition

Заглавная_страница_ru.wikipedia.org_all-access_all-agents



□ Analyzing page: Заглавная_страница_ru.wikipedia.org_desktop_all-agents
ADF Statistic: -4.095 | p-value: 0.0010 | Stationary: True



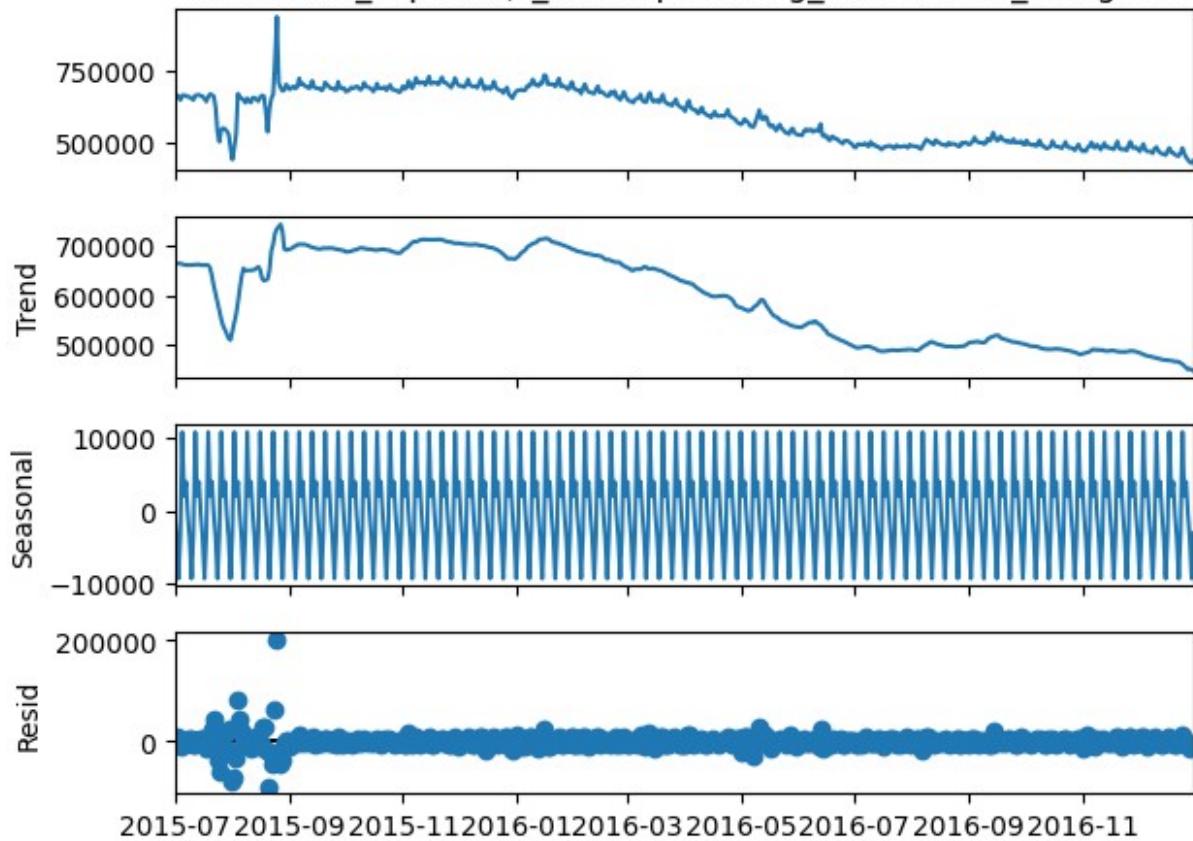


□ Analyzing page: Заглавная_страница_ru.wikipedia.org_mobile-web_all-agents
 ADF Statistic: -0.163 | p-value: 0.9428 | Stationary: False

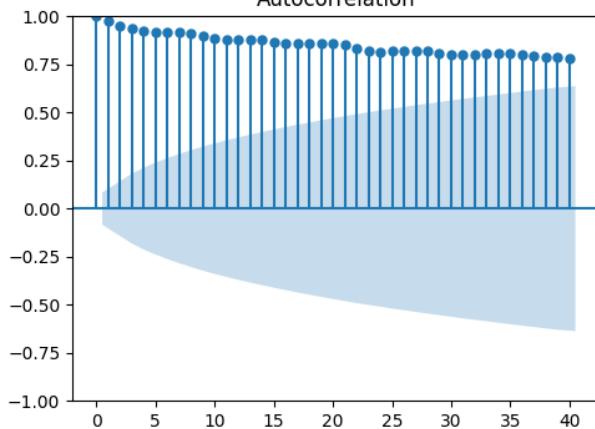


Seasonal Decomposition

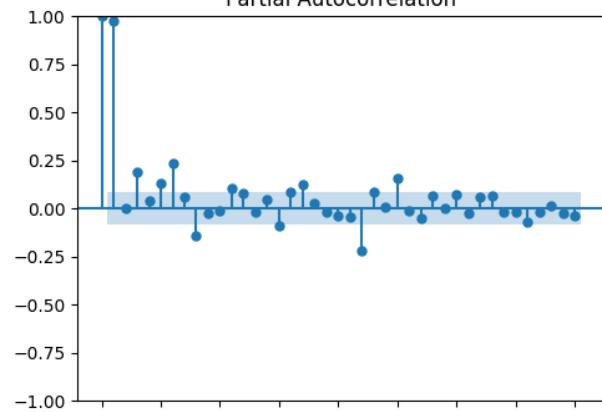
Заглавная_страница_ru.wikipedia.org_mobile-web_all-agents



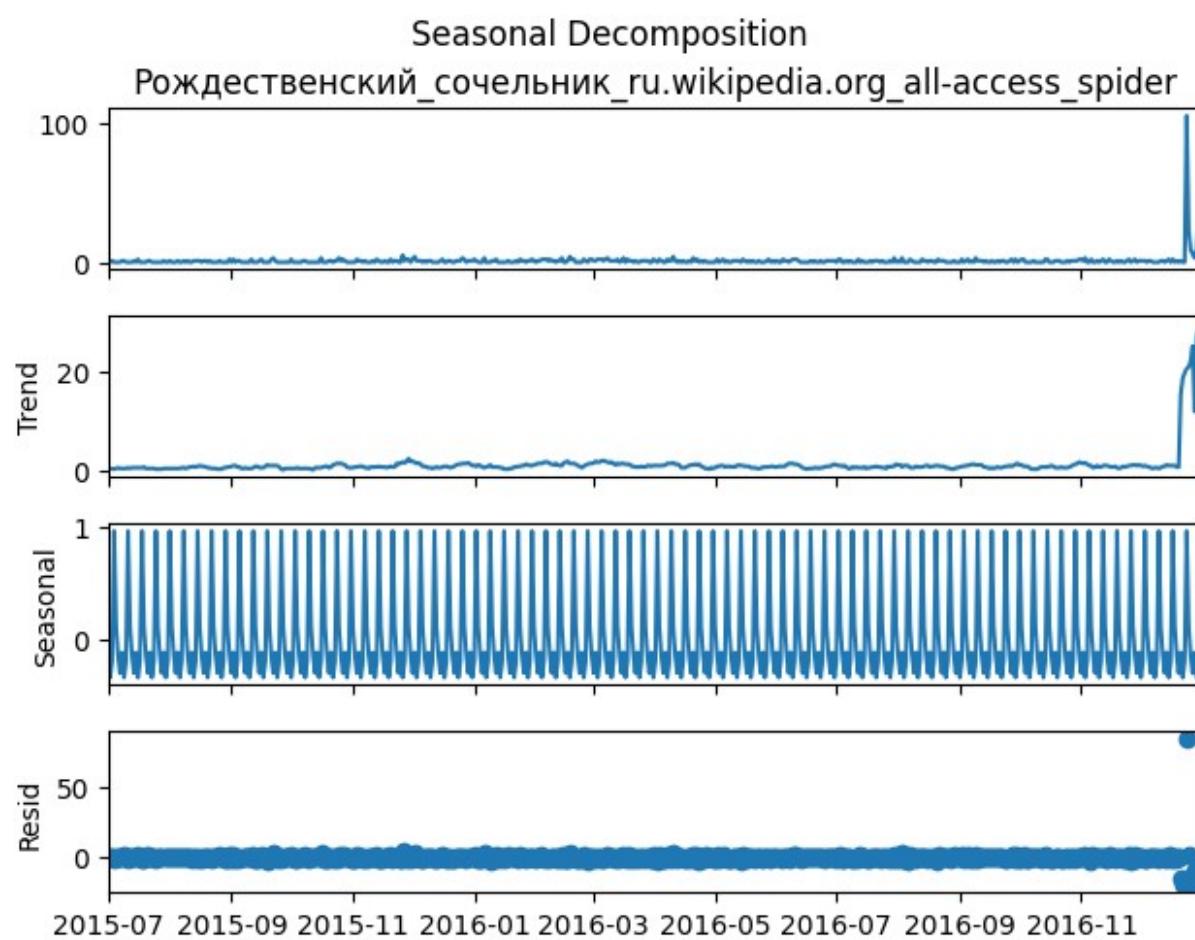
Autocorrelation

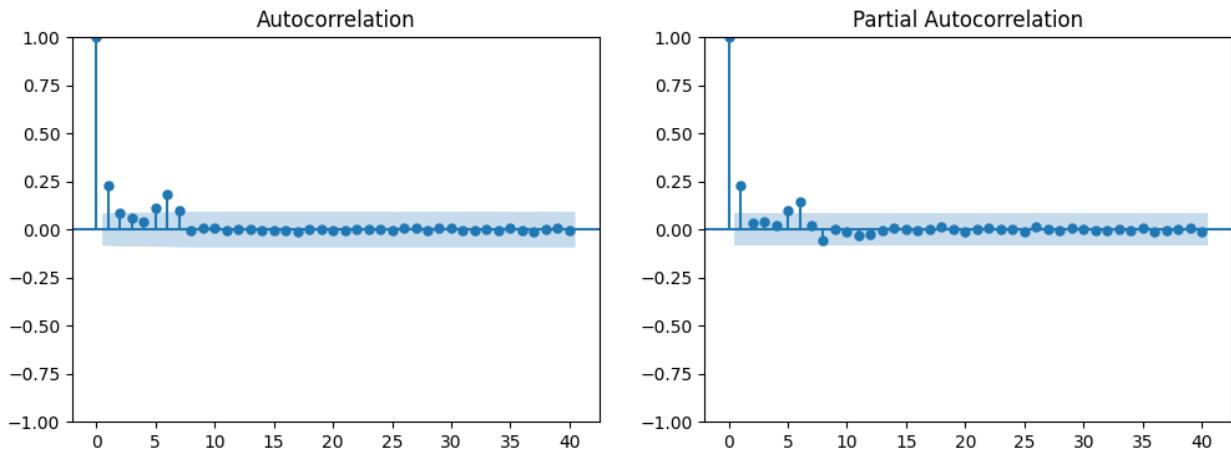


Partial Autocorrelation

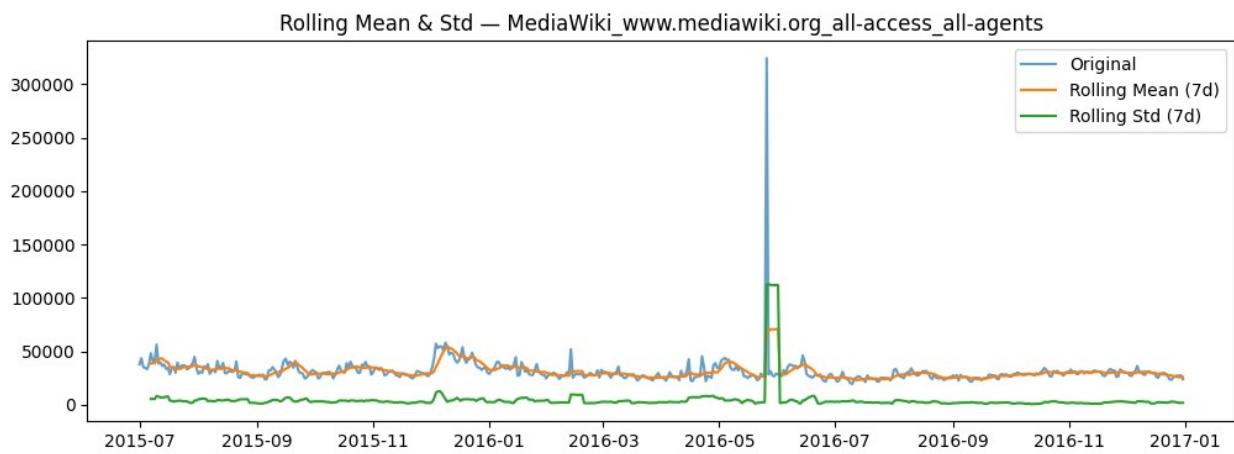


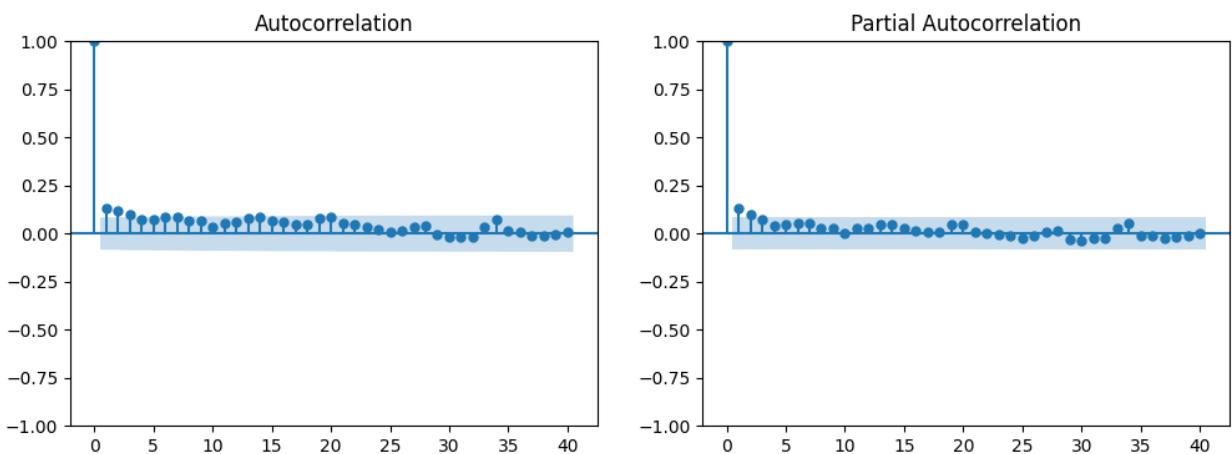
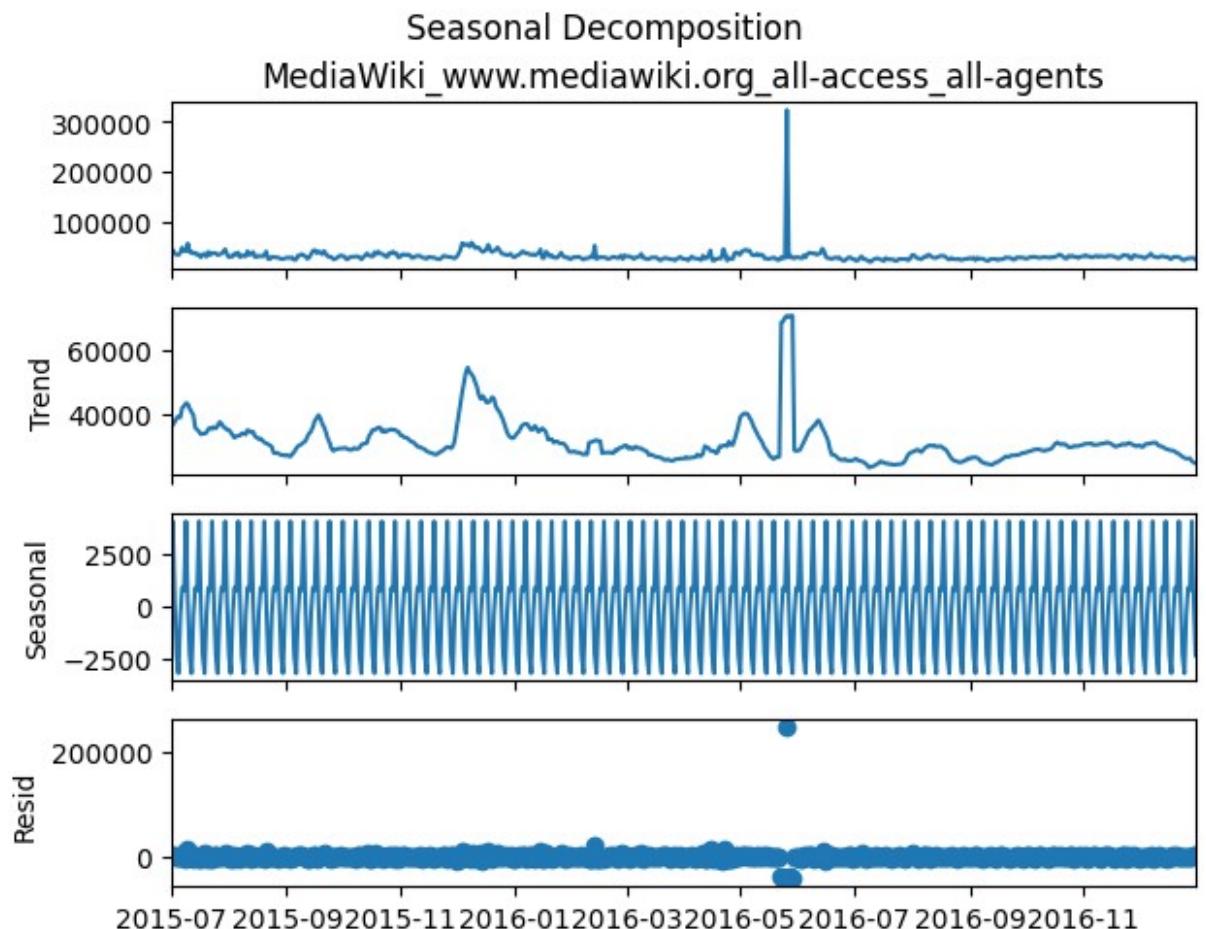
□ Analyzing page: Рождественский_сочельник_ru.wikipedia.org_all-access_spider
ADF Statistic: -6.169 | p-value: 0.0000 | Stationary: True



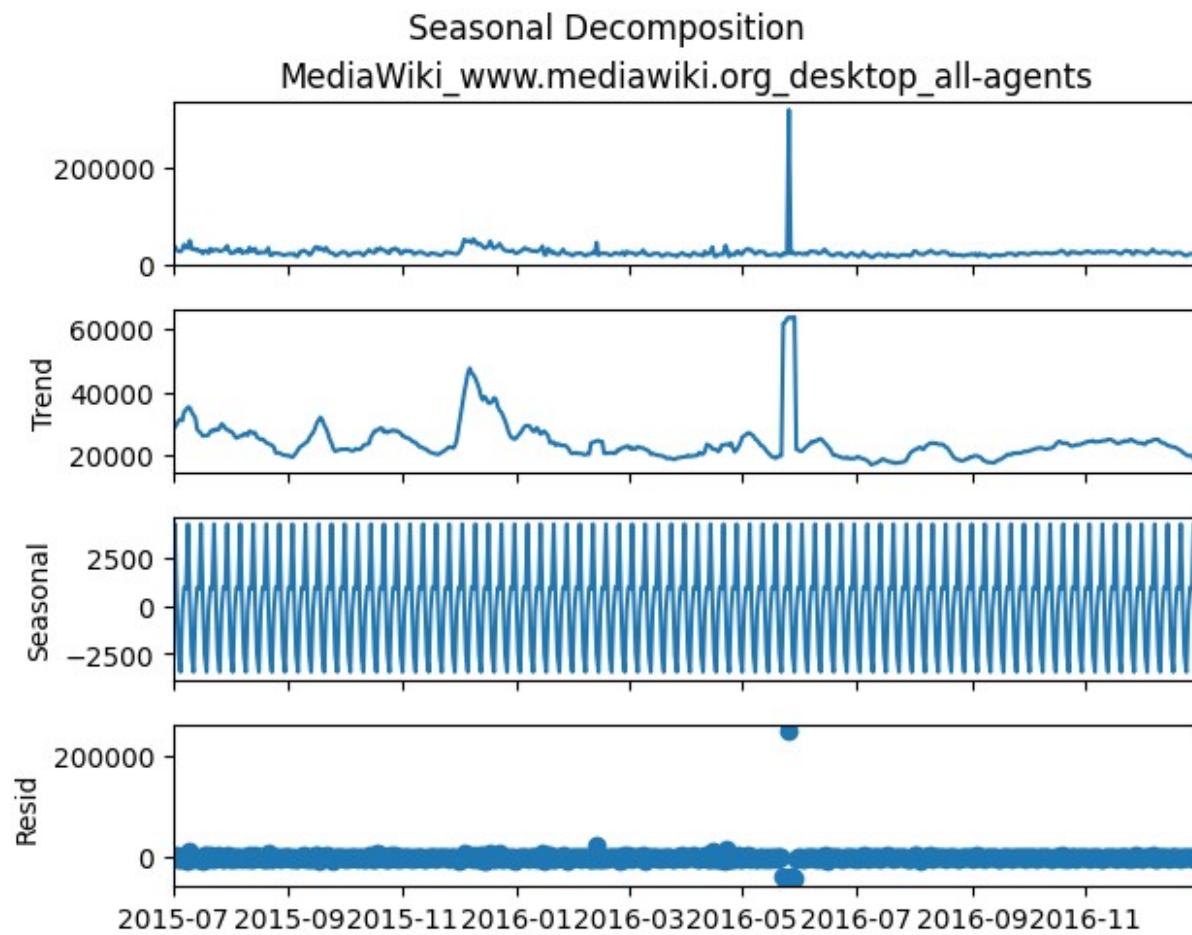
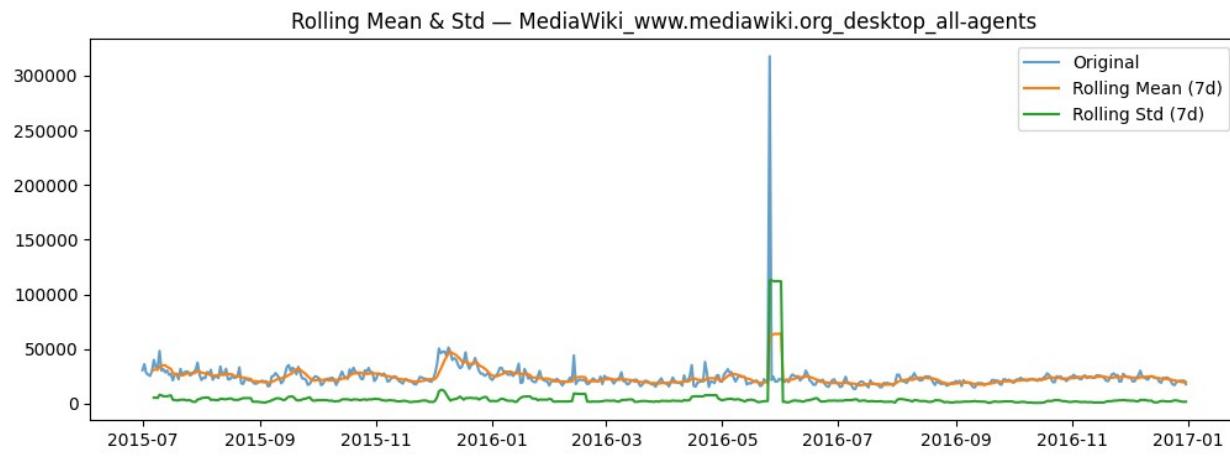


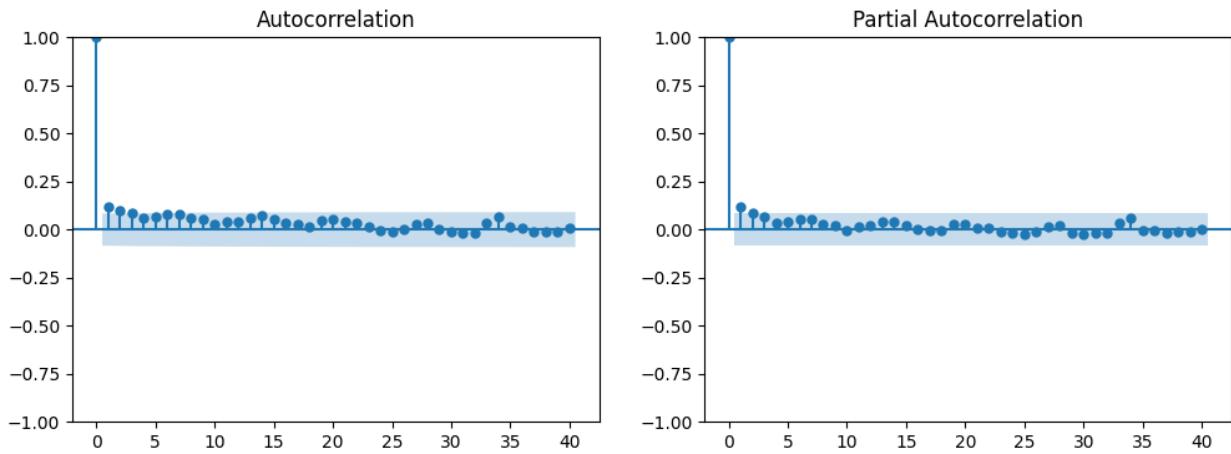
Analyzing page: MediaWiki_www.mediawiki.org_all-access_all-agents
 ADF Statistic: -11.068 | p-value: 0.0000 | Stationary: True



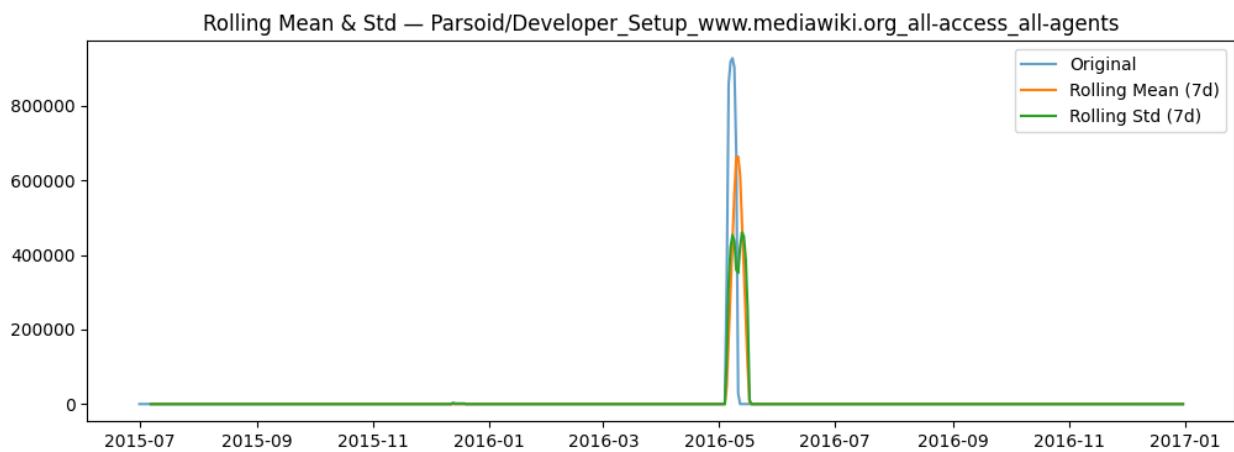


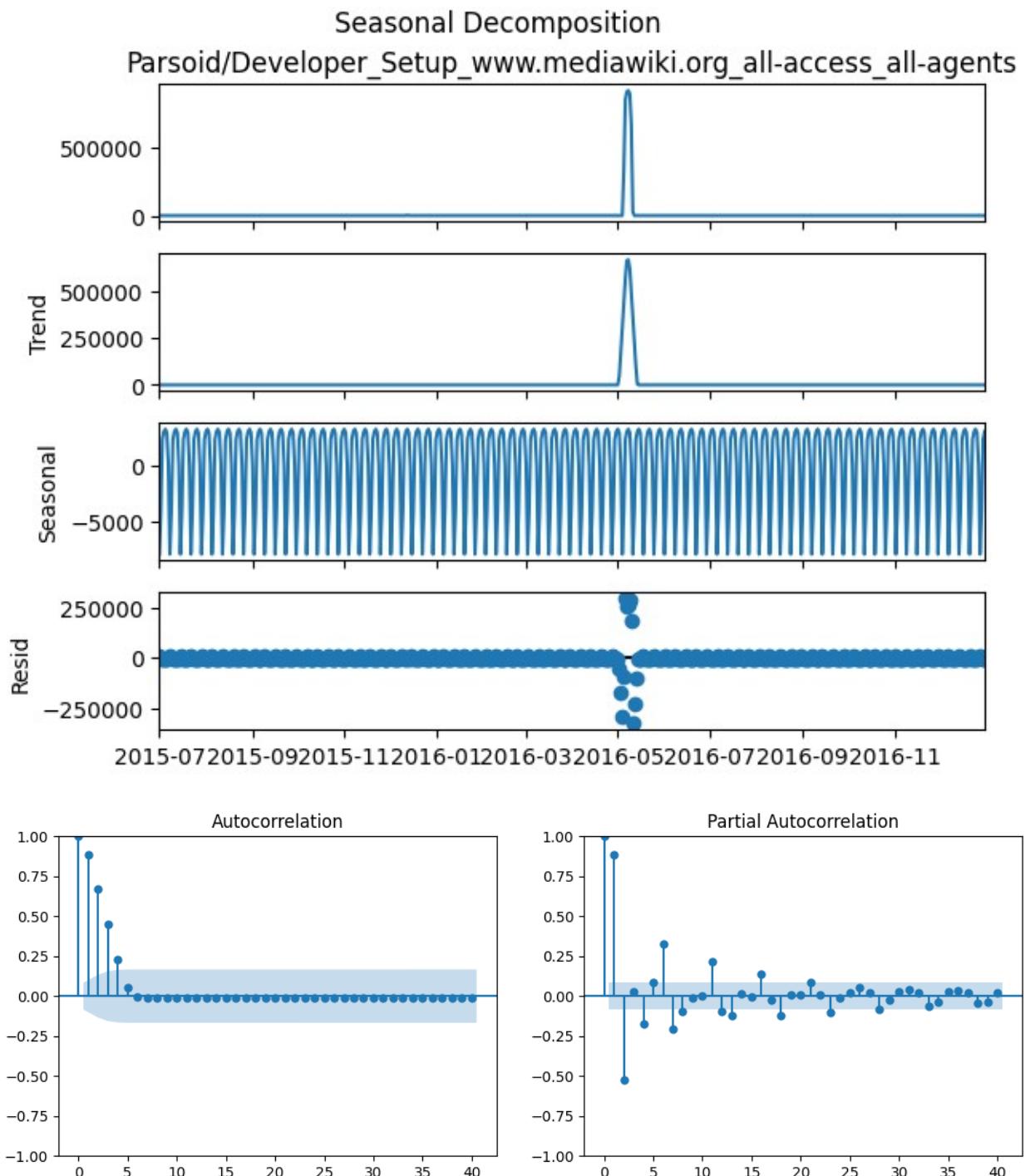
□ Analyzing page: MediaWiki_www.mediawiki.org_desktop_all-agents
 ADF Statistic: -14.194 | p-value: 0.0000 | Stationary: True





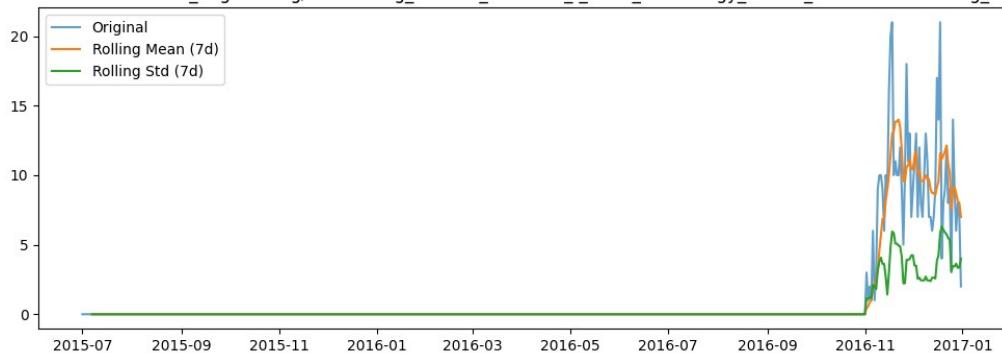
□ Analyzing page: Parsoid/Developer_Setup_www.mediawiki.org_all-access_all-agents
 ADF Statistic: -5.157 | p-value: 0.0000 | Stationary: True



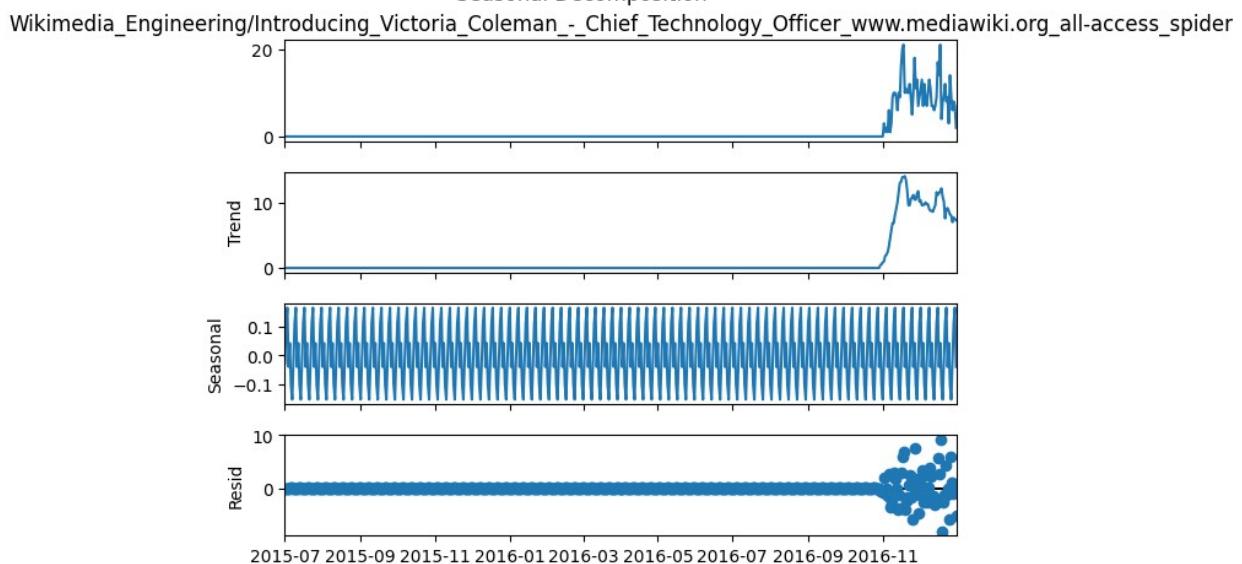


□ Analyzing page: Wikimedia_Engineering/Introducing_Victoria_Coleman_-
Chief_Technology_Officer_www.mediawiki.org_all-access_spider
ADF Statistic: -1.793 | p-value: 0.3841 | Stationary: False

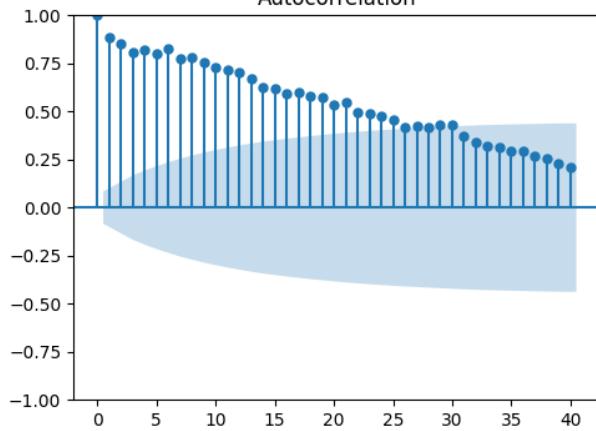
Rolling Mean & Std — Wikimedia_Engineering/Introducing_Victoria_Coleman_-_Chief_Technology_Officer_www.mediawiki.org_all-access_spider



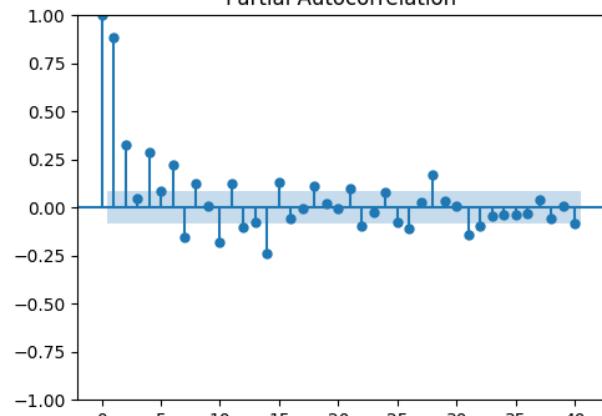
Seasonal Decomposition



Autocorrelation

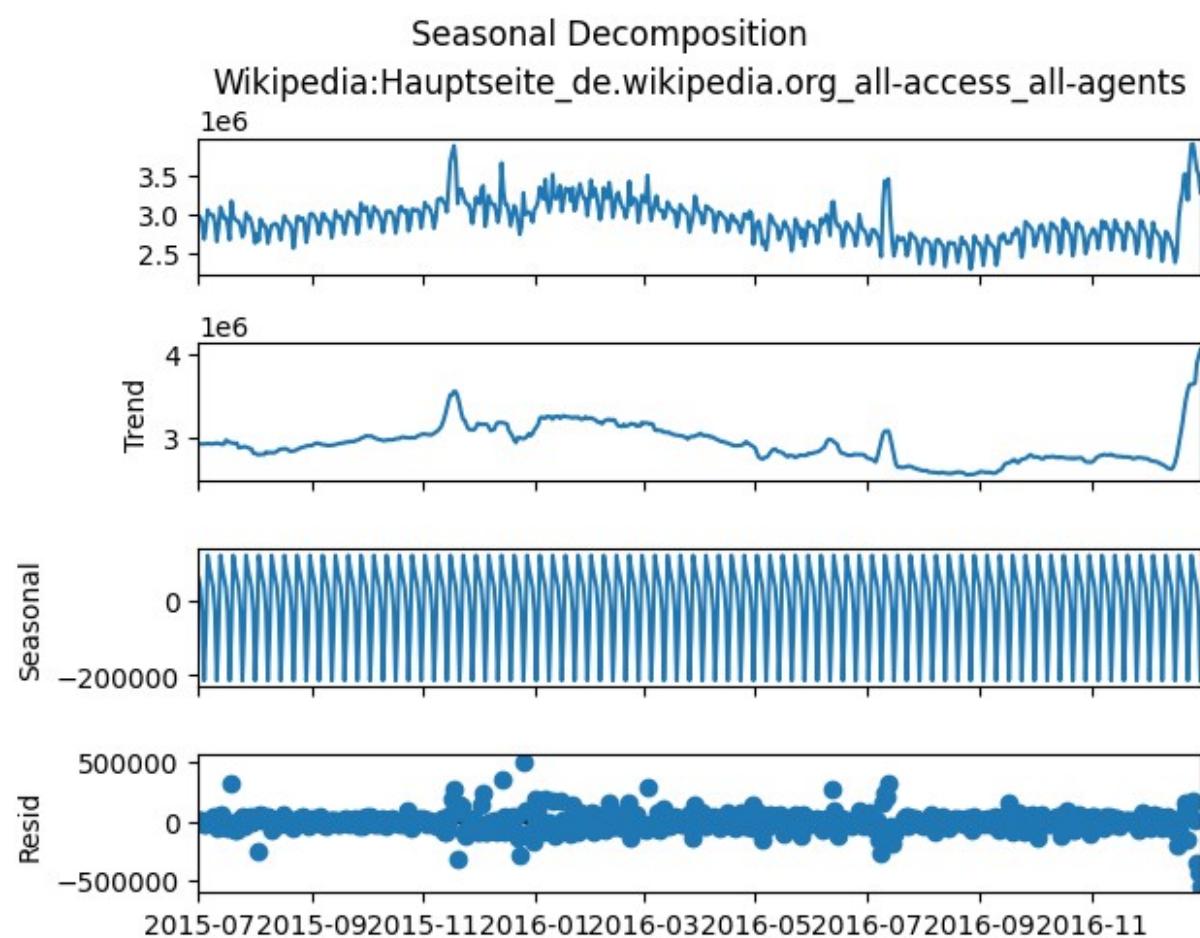
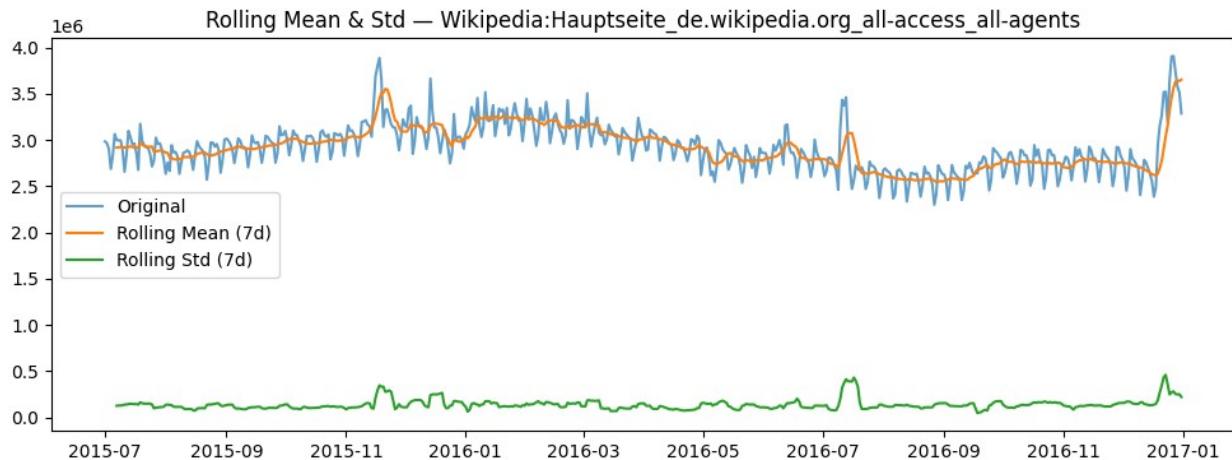


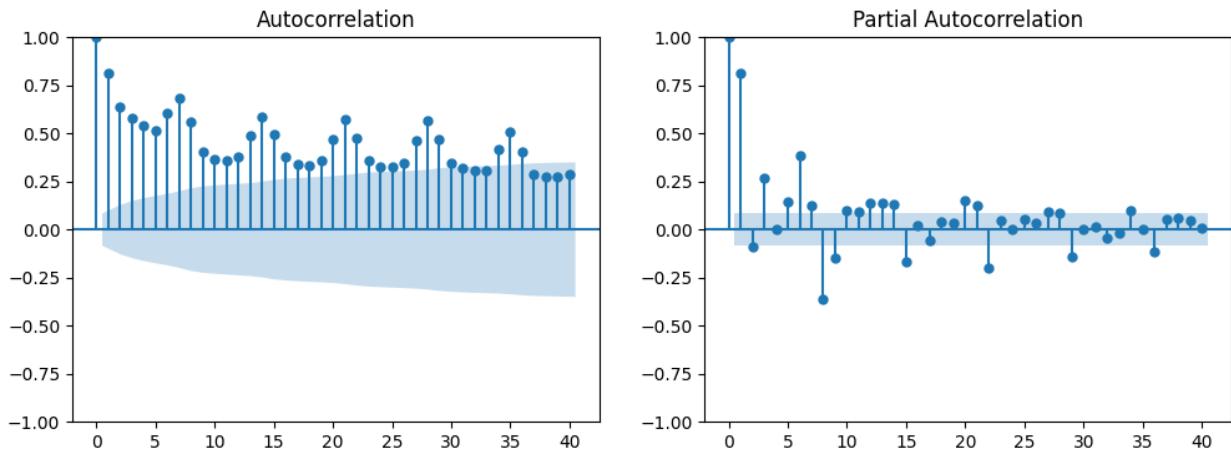
Partial Autocorrelation



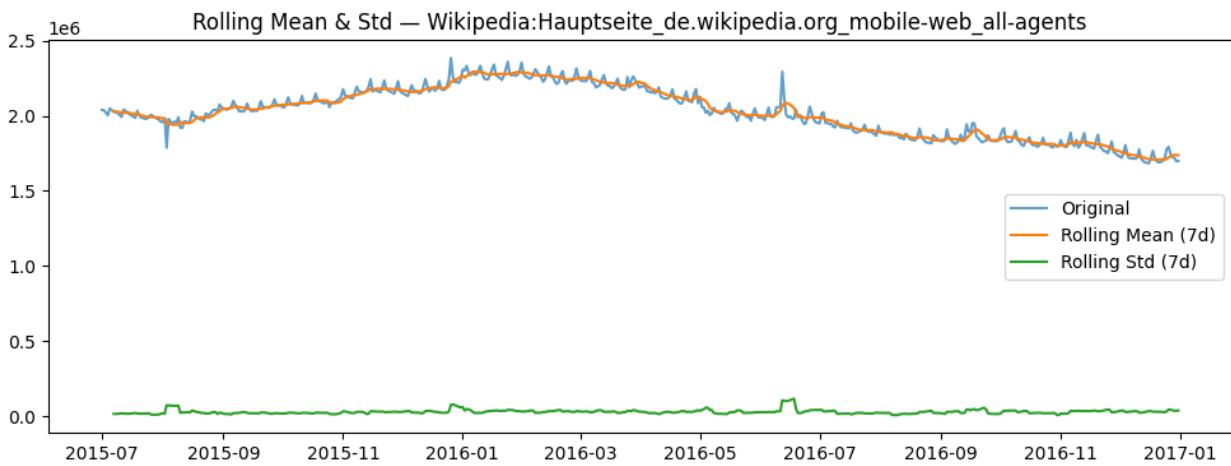
□ Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents

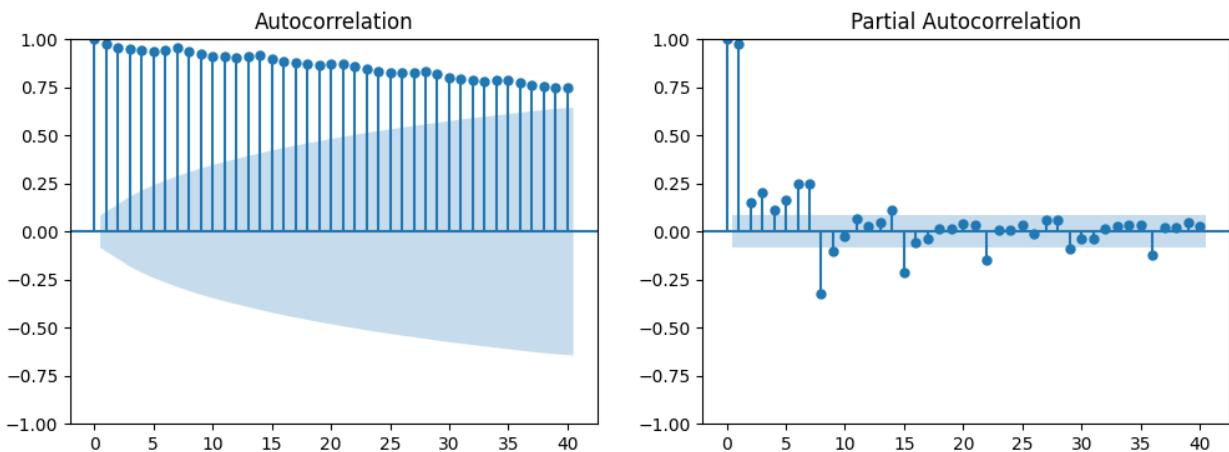
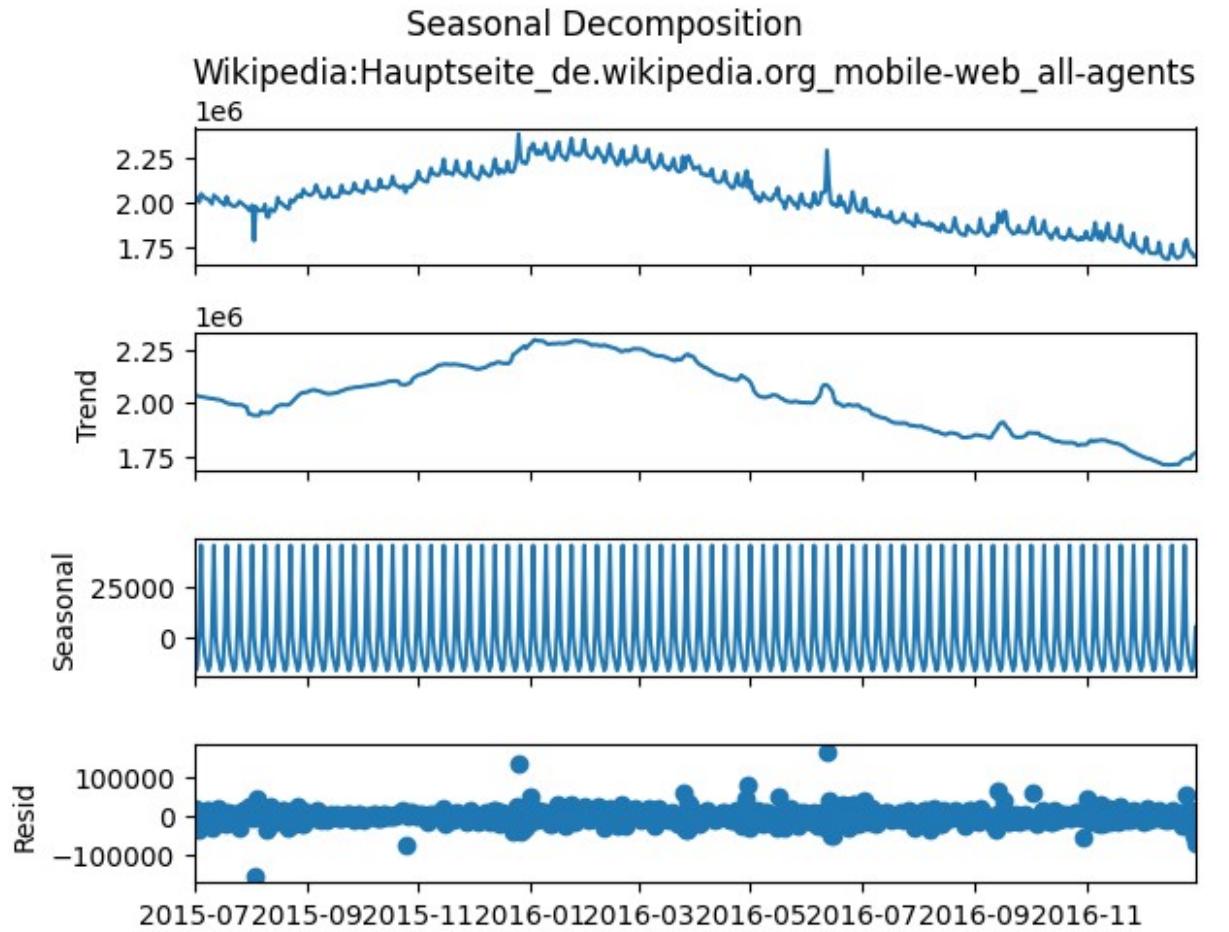
ADF Statistic: -1.520 | p-value: 0.5236 | Stationary: False



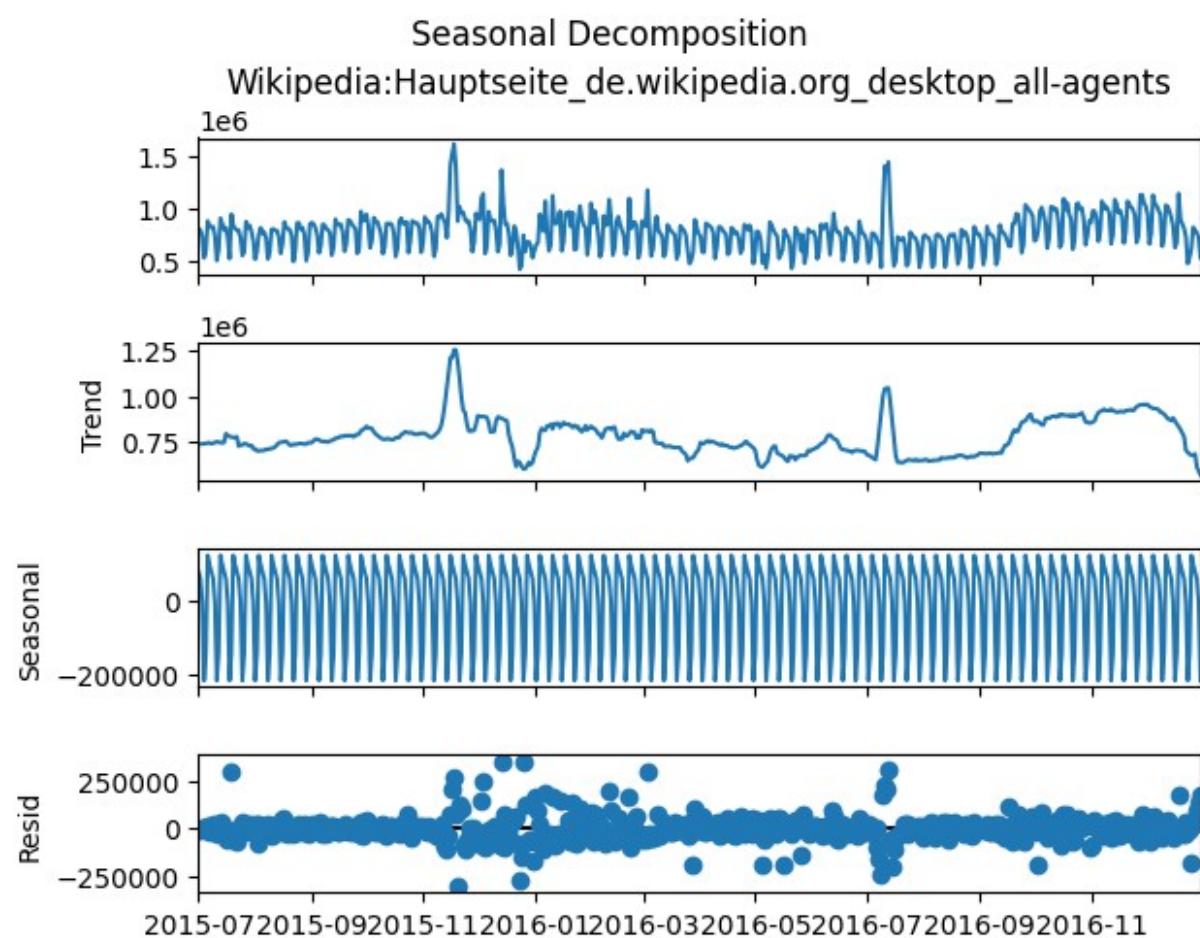
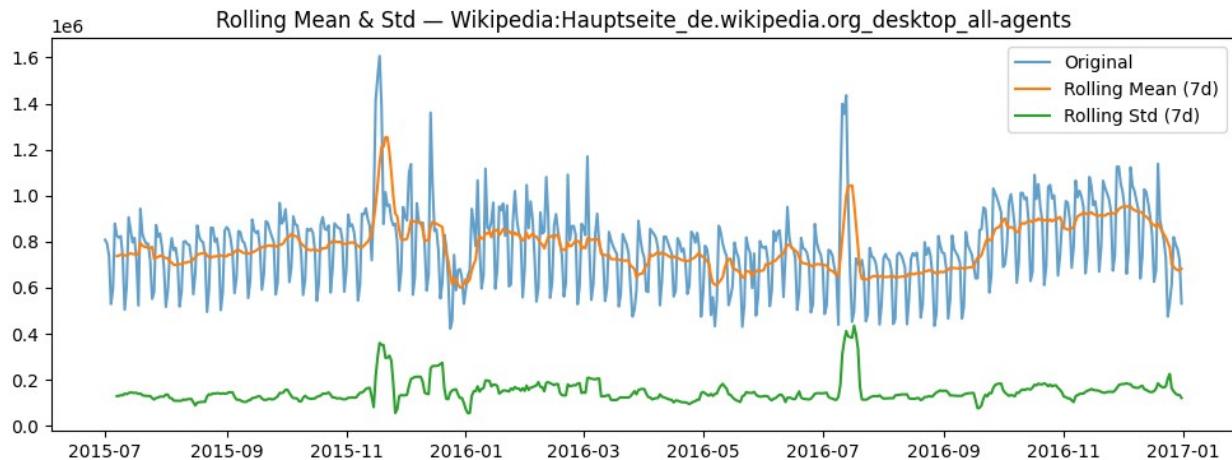


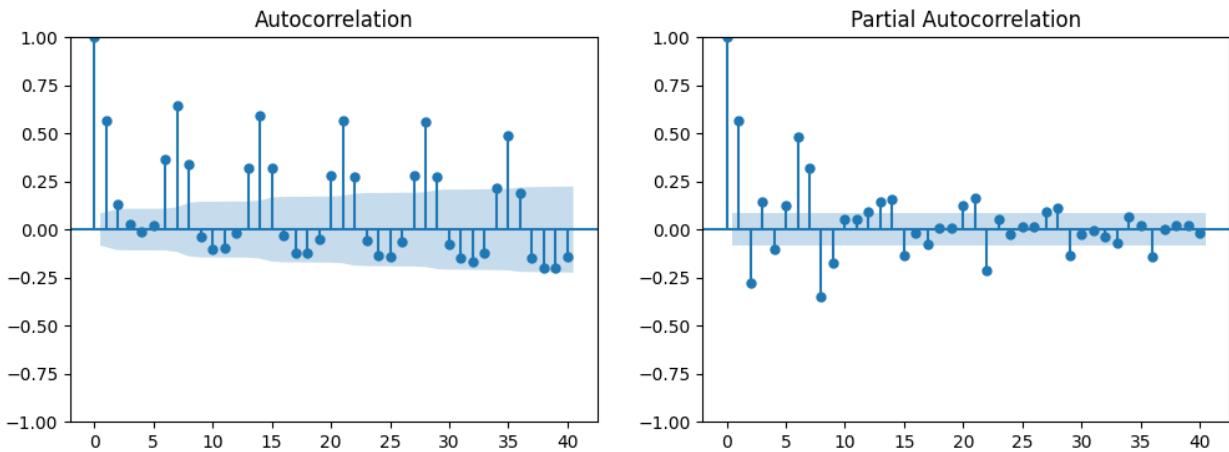
```
□ Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents
ADF Statistic: 0.316 | p-value: 0.9781 | Stationary: False
```



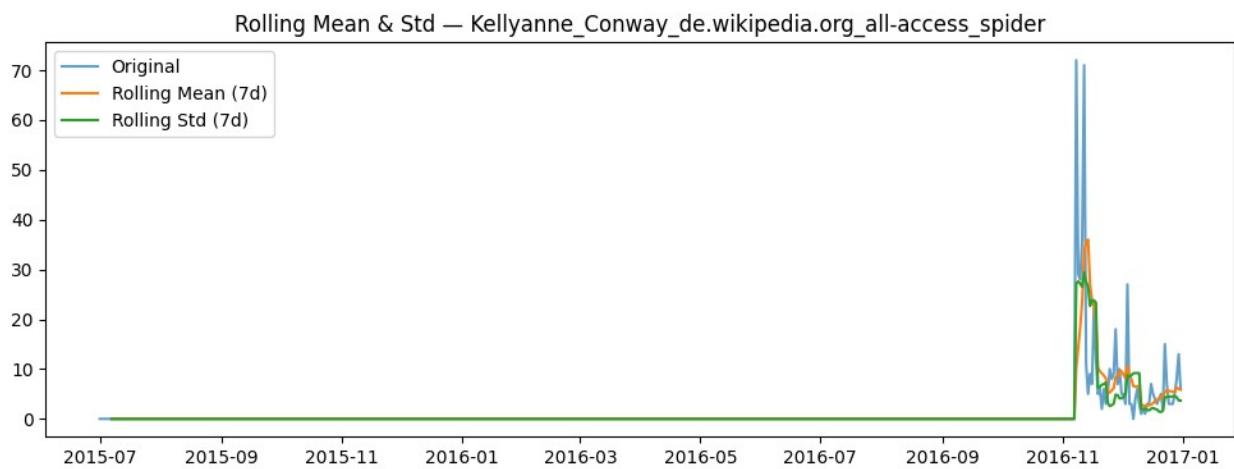


□ Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_desktop_all-agents
 ADF Statistic: -2.900 | p-value: 0.0453 | Stationary: True

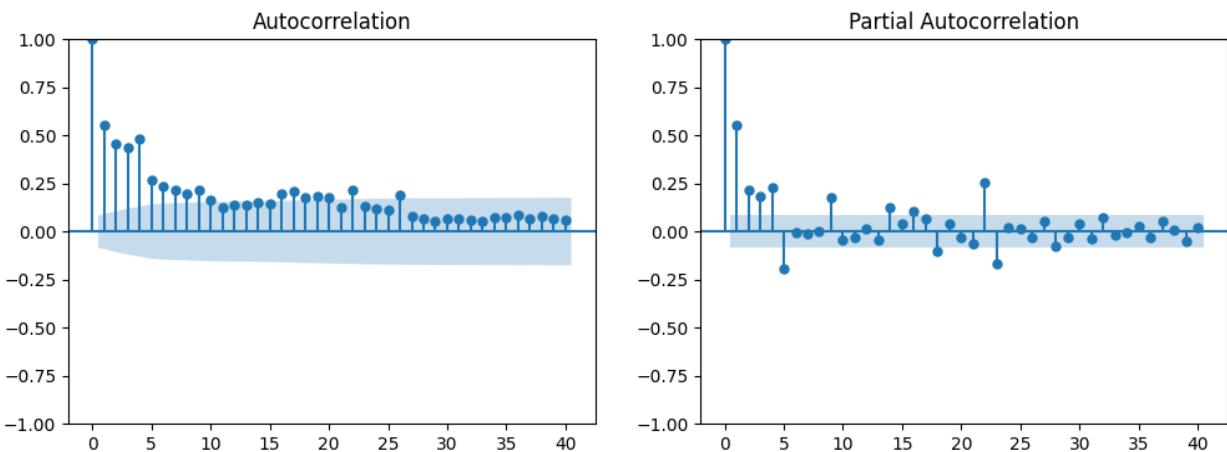
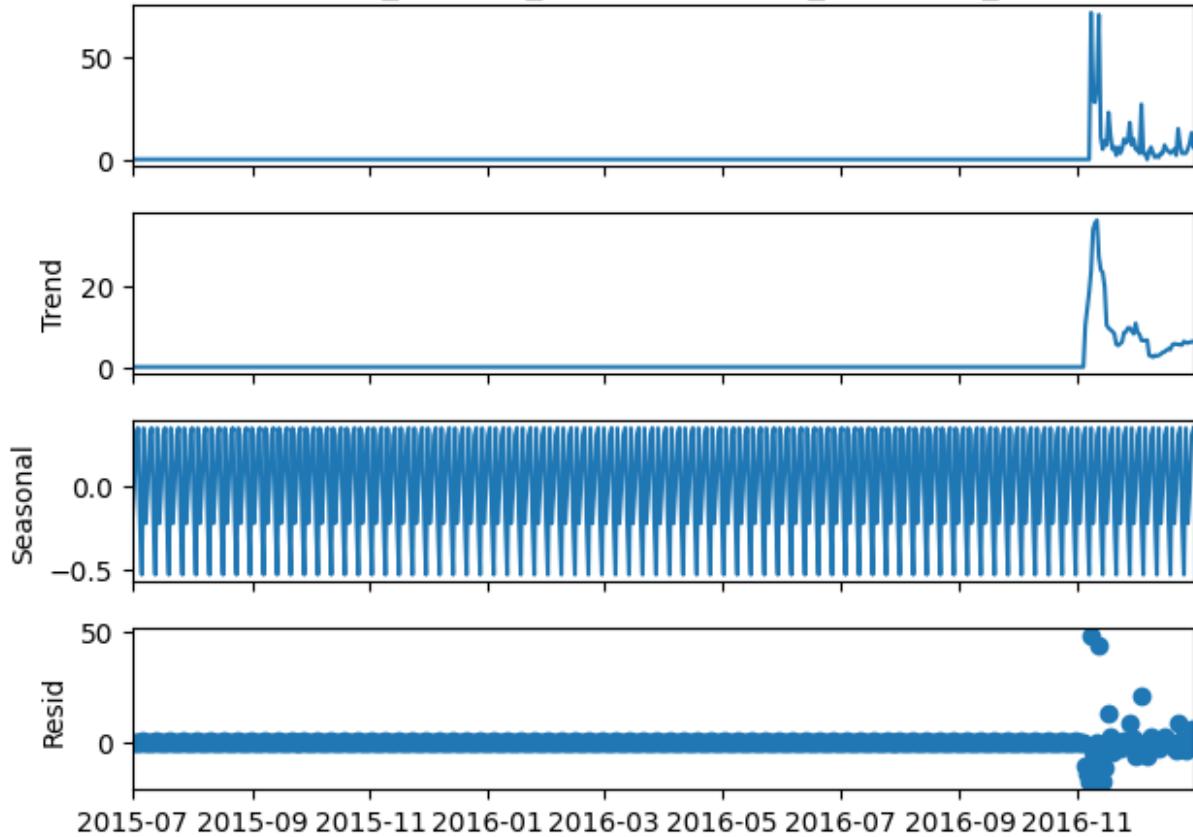




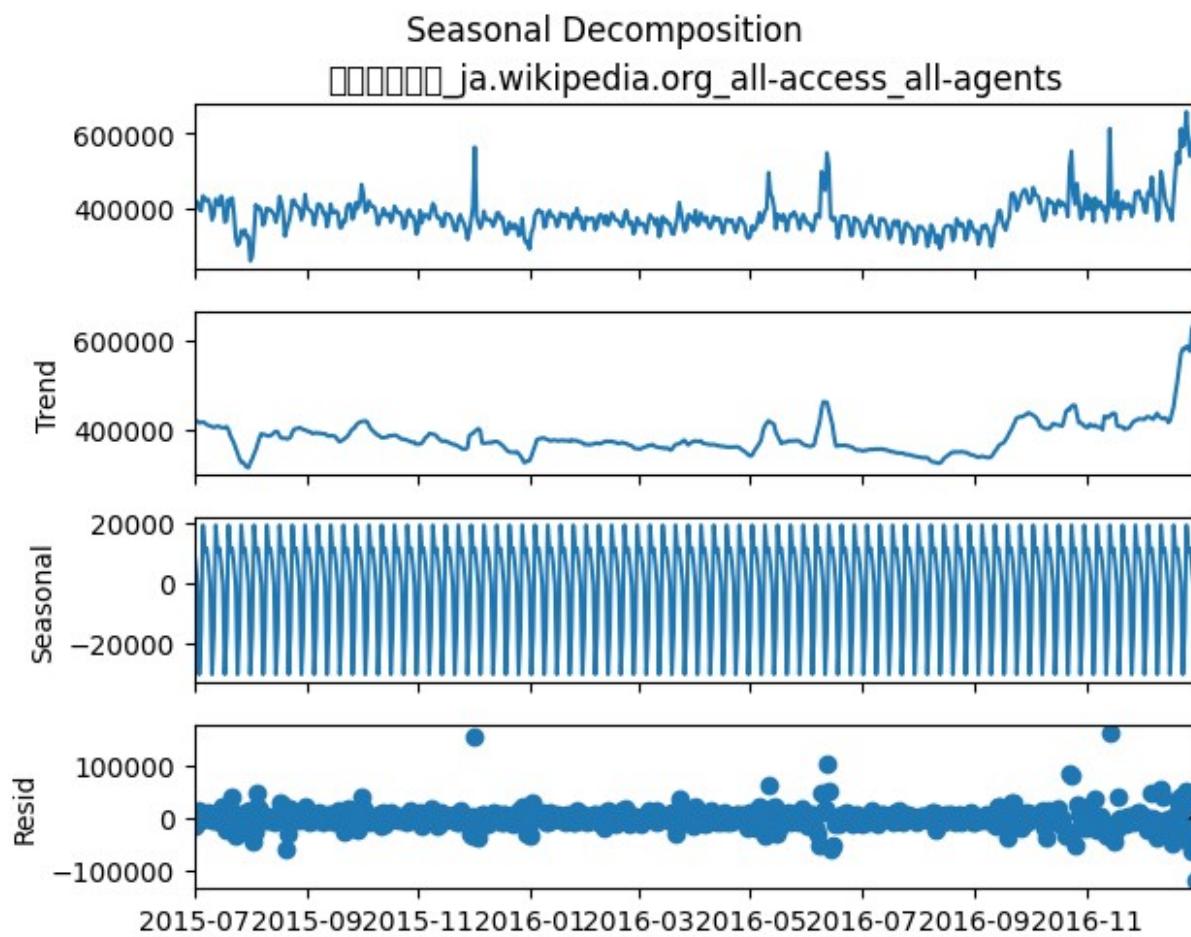
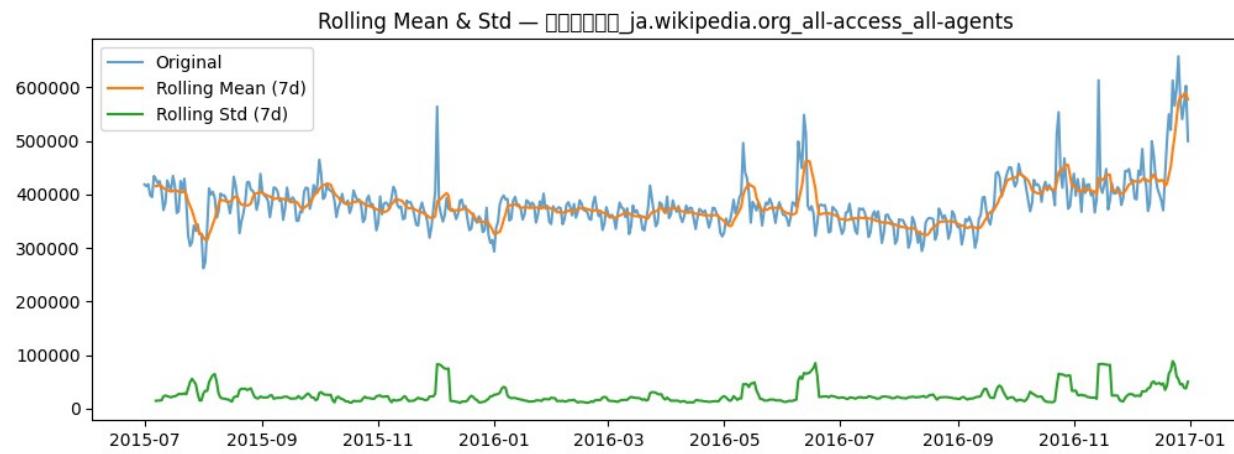
□ Analyzing page: Kellyanne_Conway_de.wikipedia.org_all-access_spider
 ADF Statistic: -3.132 | p-value: 0.0243 | Stationary: True

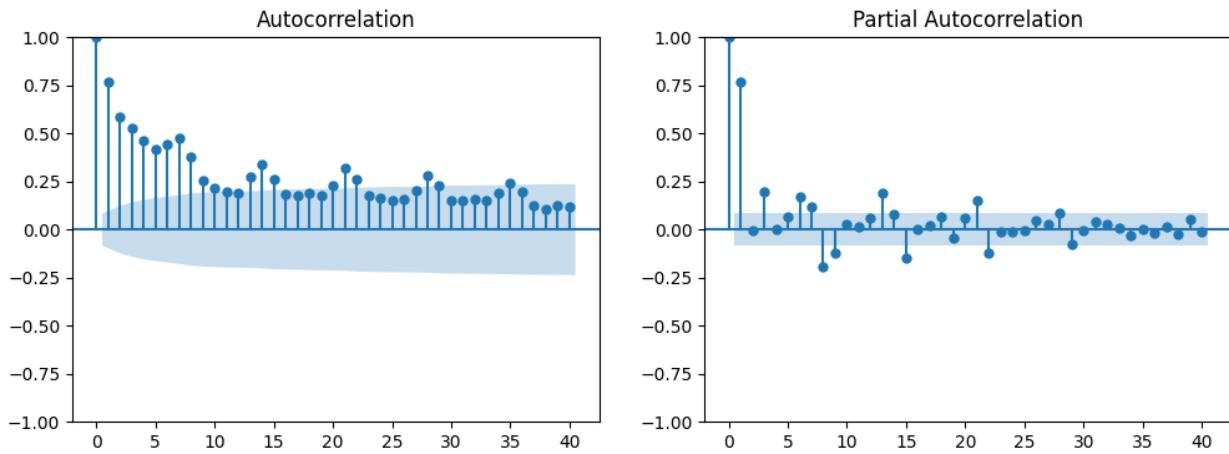


Seasonal Decomposition Kellyanne_Conway_de.wikipedia.org_all-access_spider

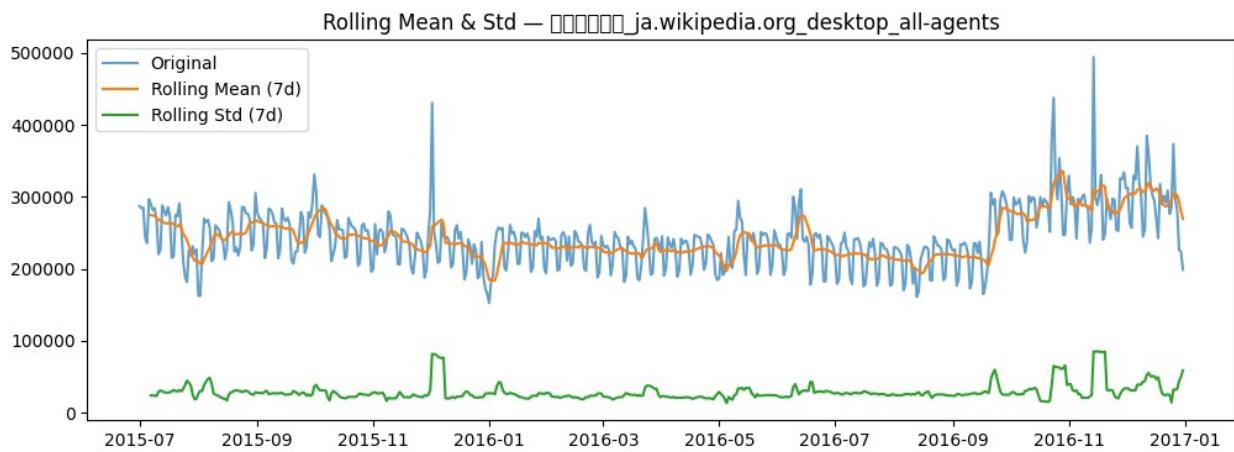


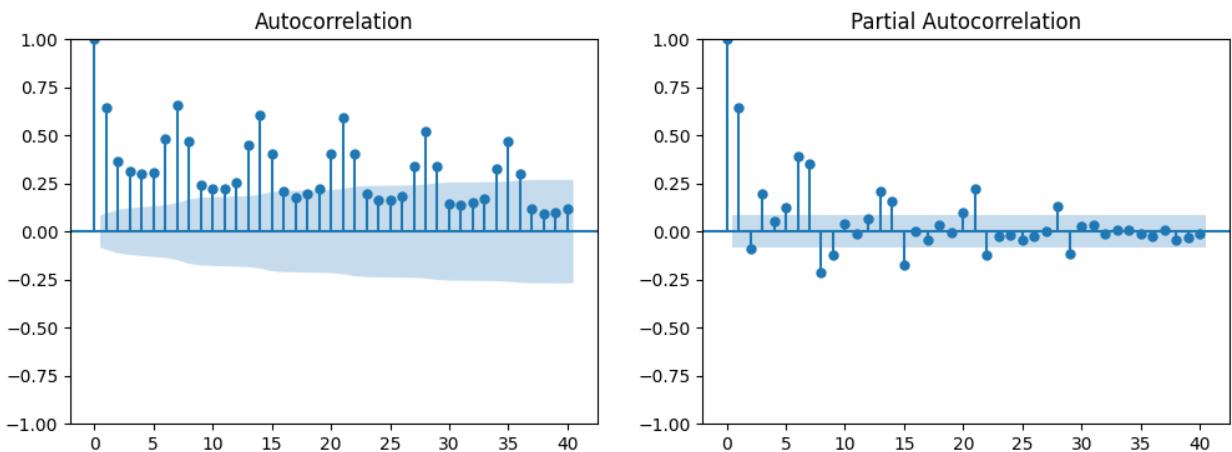
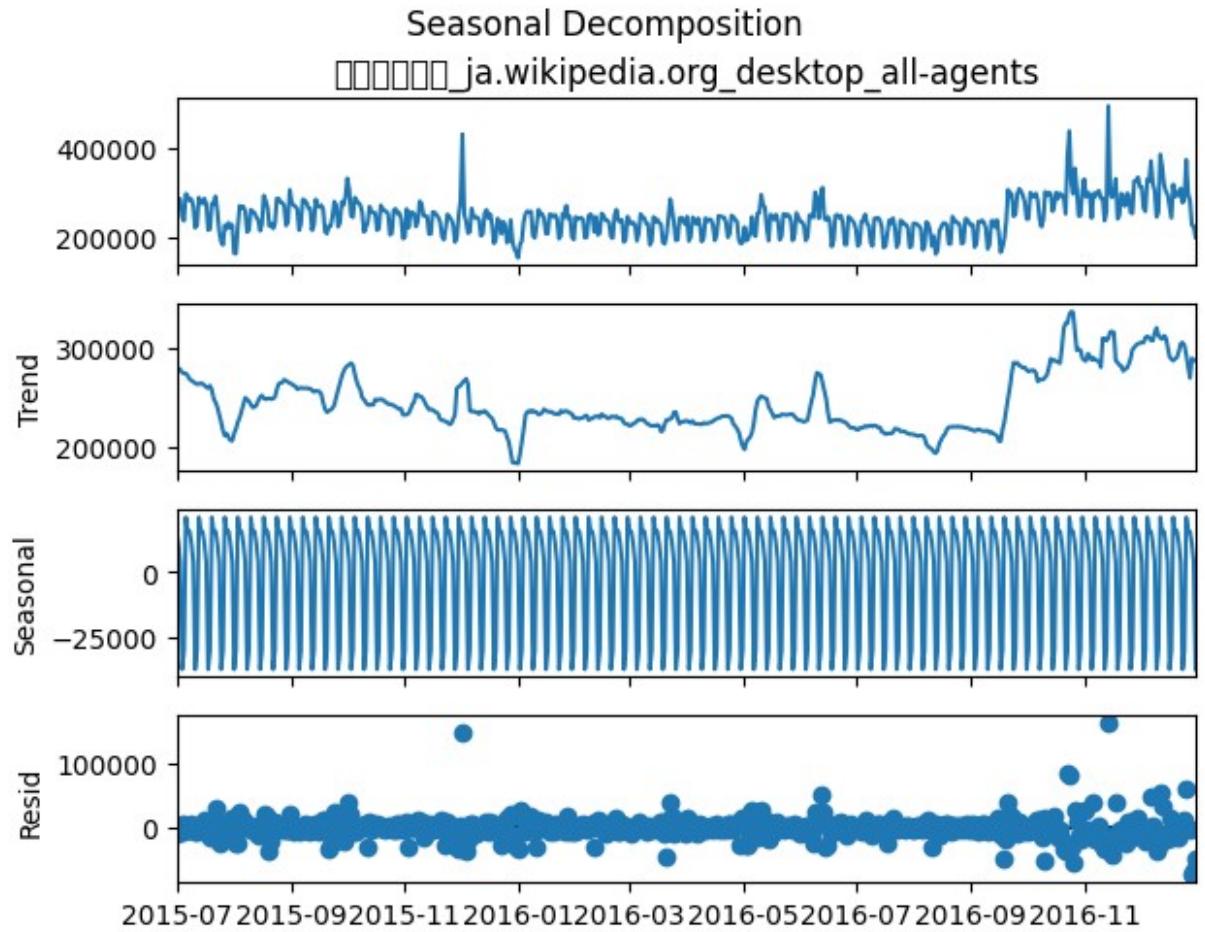
Analyzing page: メインページ_ja.wikipedia.org_all-access_all-agents
ADF Statistic: -1.429 | p-value: 0.5682 | Stationary: False



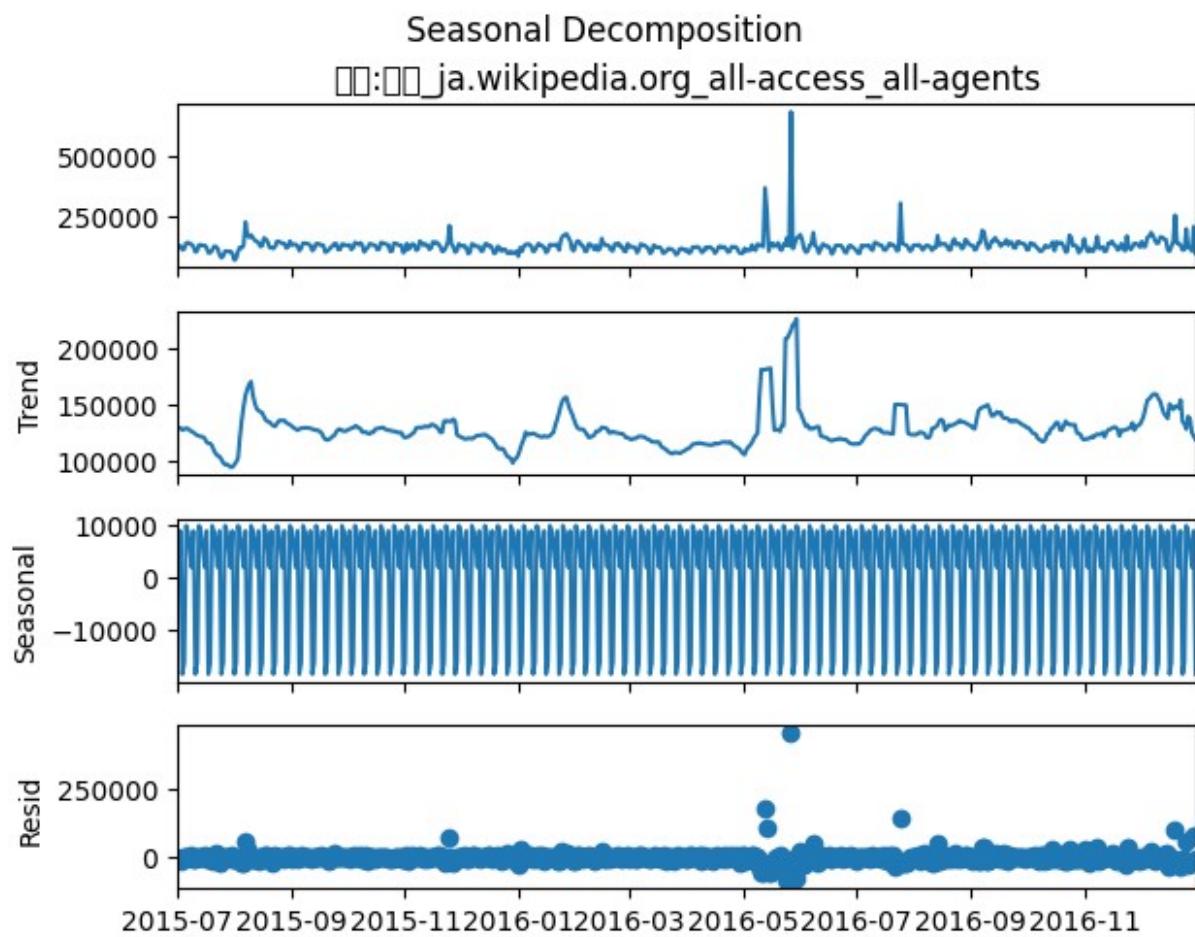
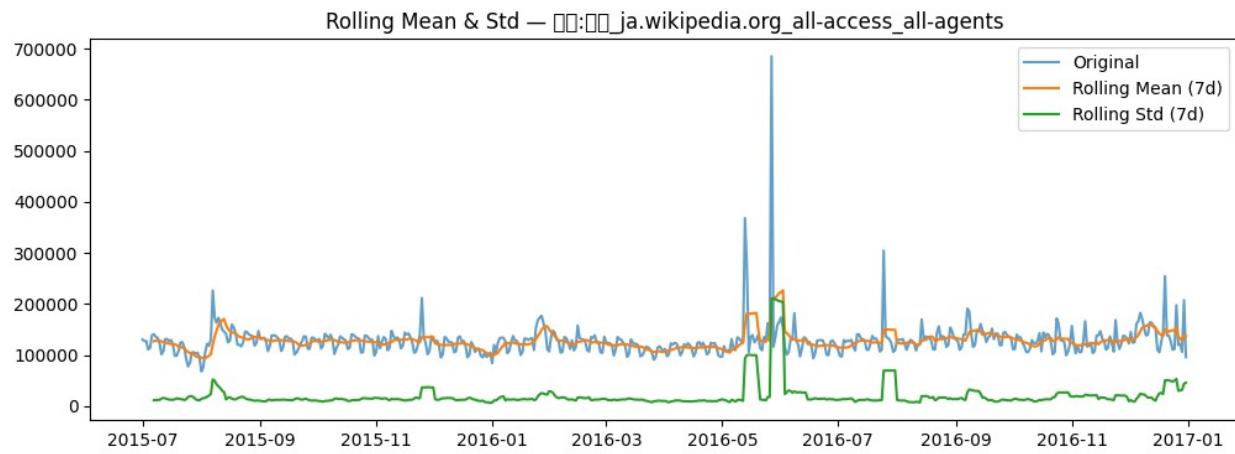


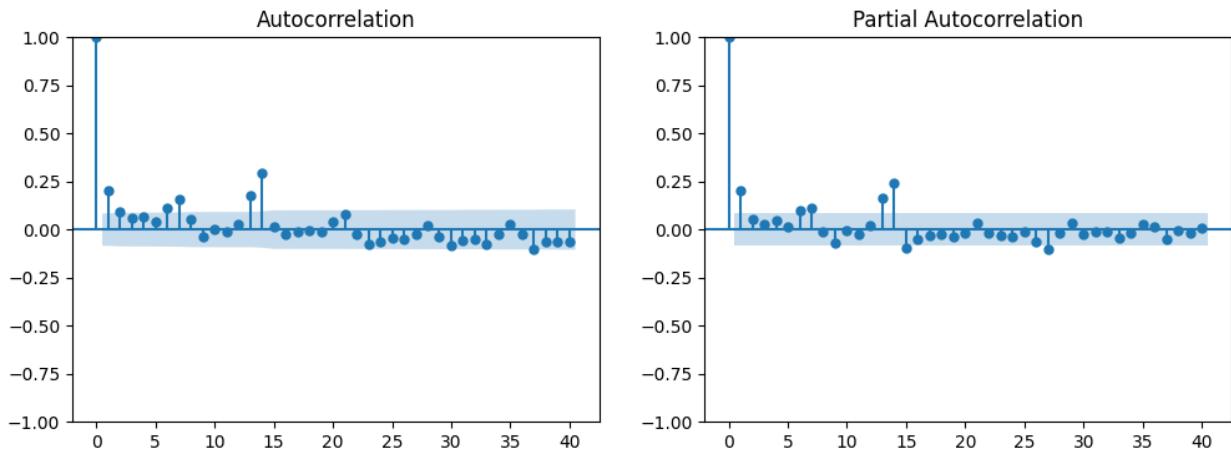
Analyzing page: メインページ_ja.wikipedia.org_desktop_all-agents
 ADF Statistic: -2.472 | p-value: 0.1225 | Stationary: False





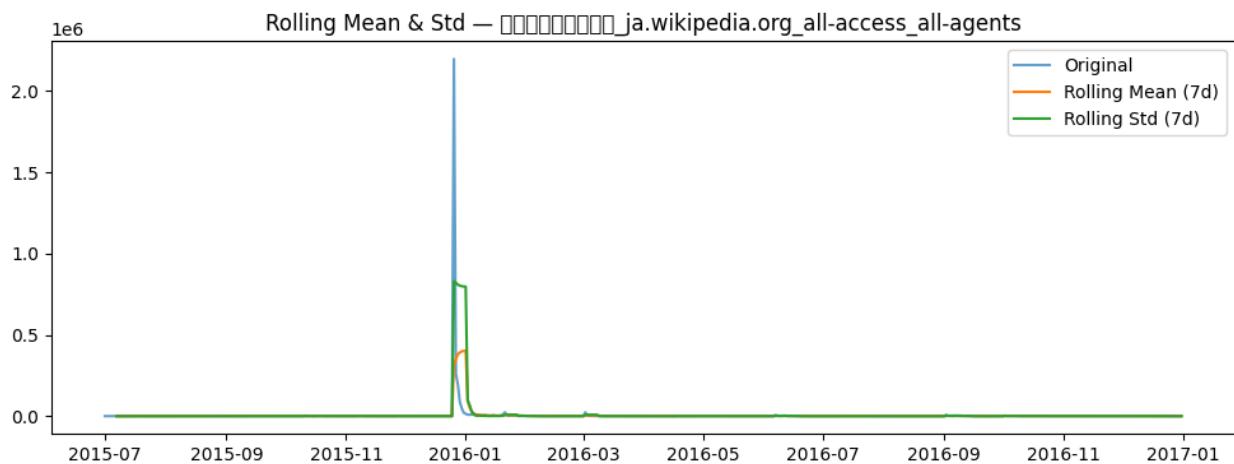
□ Analyzing page: 特別:検索_ja.wikipedia.org_all-access_all-agents
 ADF Statistic: -3.855 | p-value: 0.0024 | Stationary: True

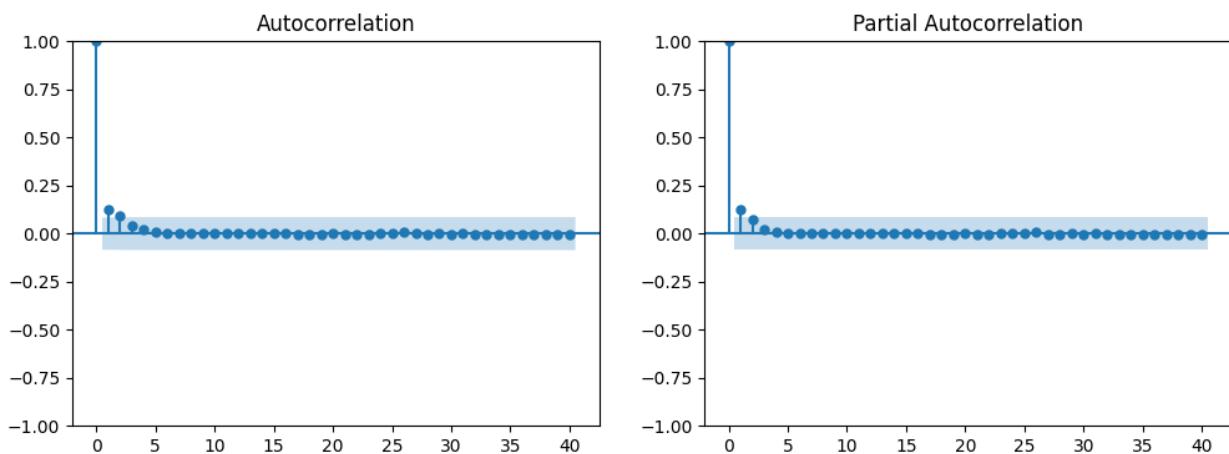
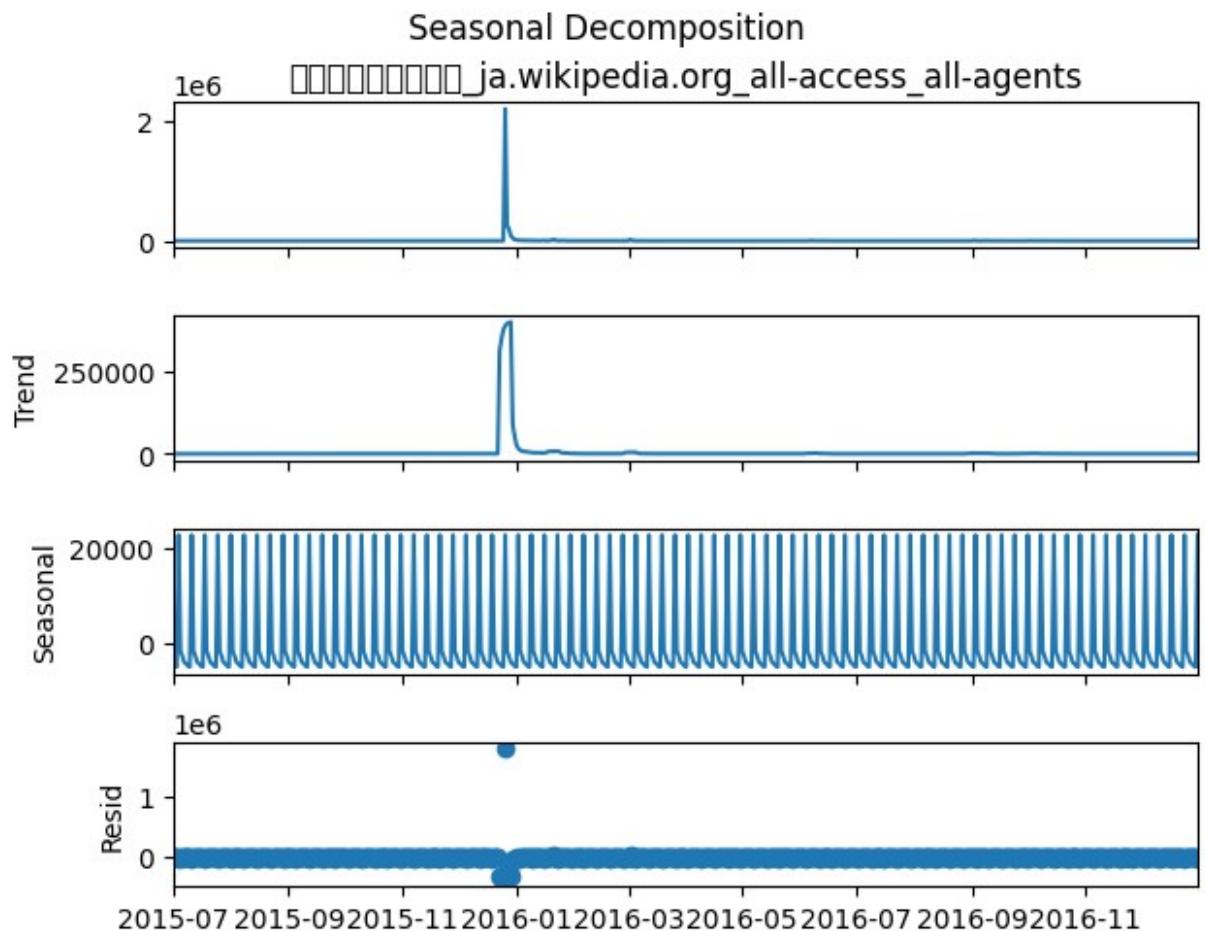




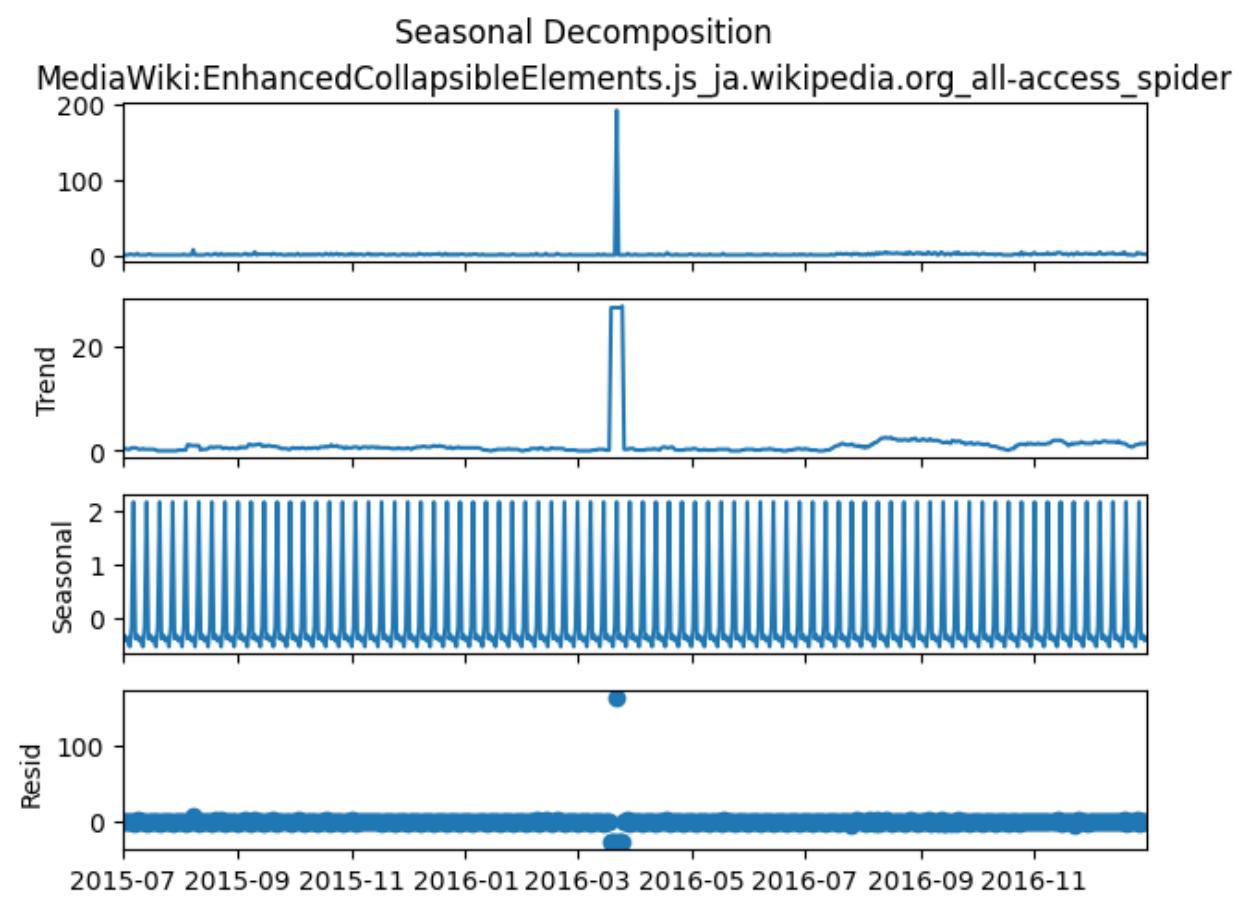
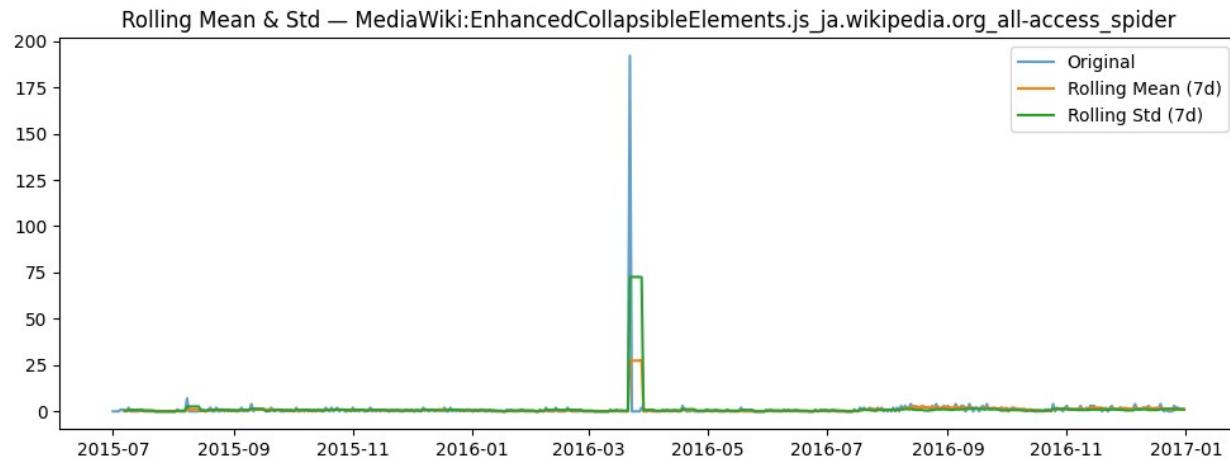
□ Analyzing page: キングオブコメディ_ja.wikipedia.org_all-access_all-agents

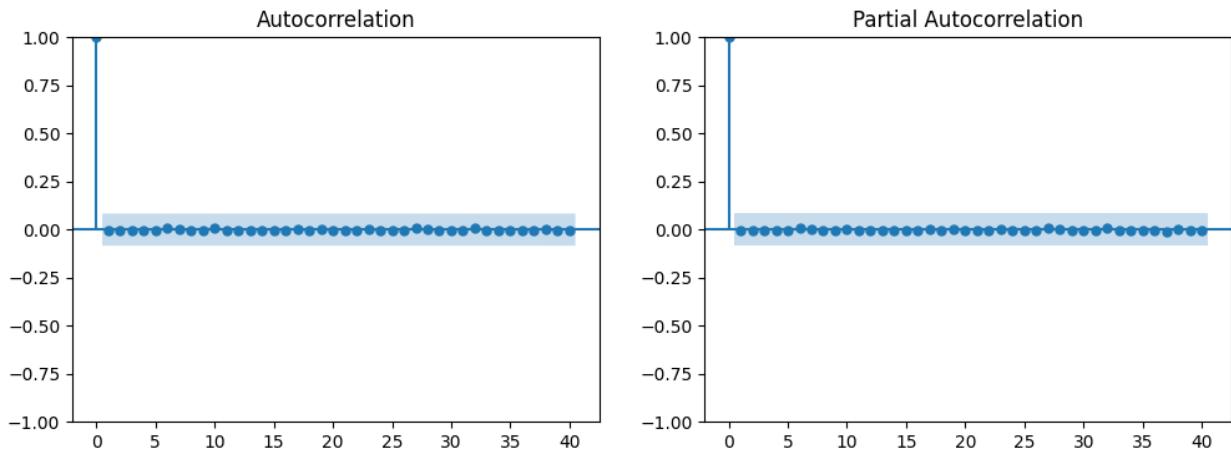
ADF Statistic: -14.300 | p-value: 0.0000 | Stationary: True



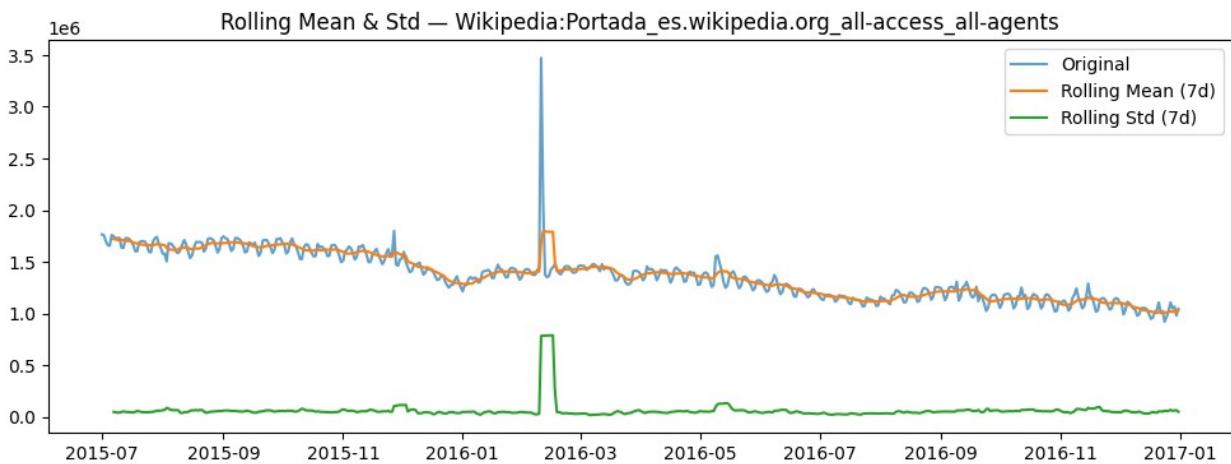


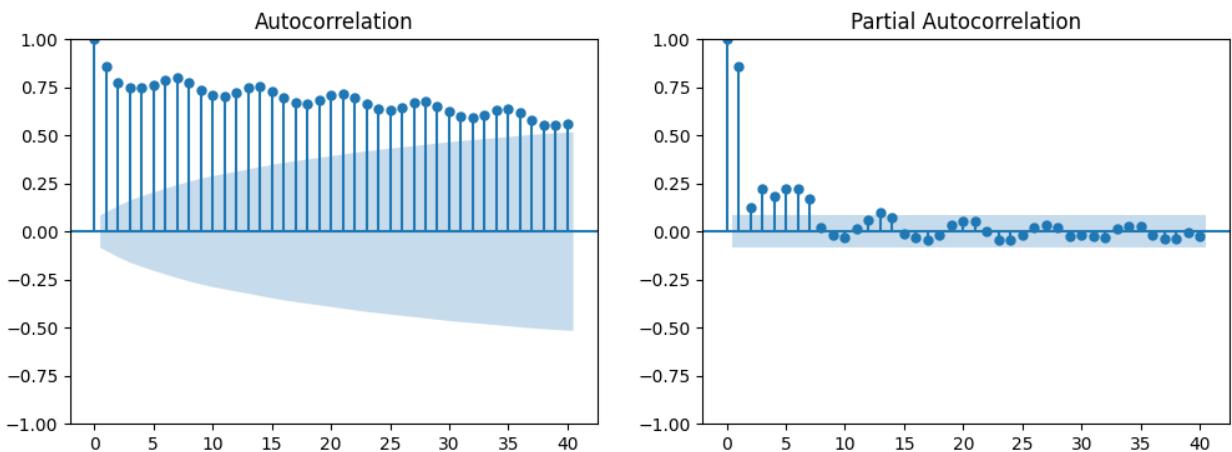
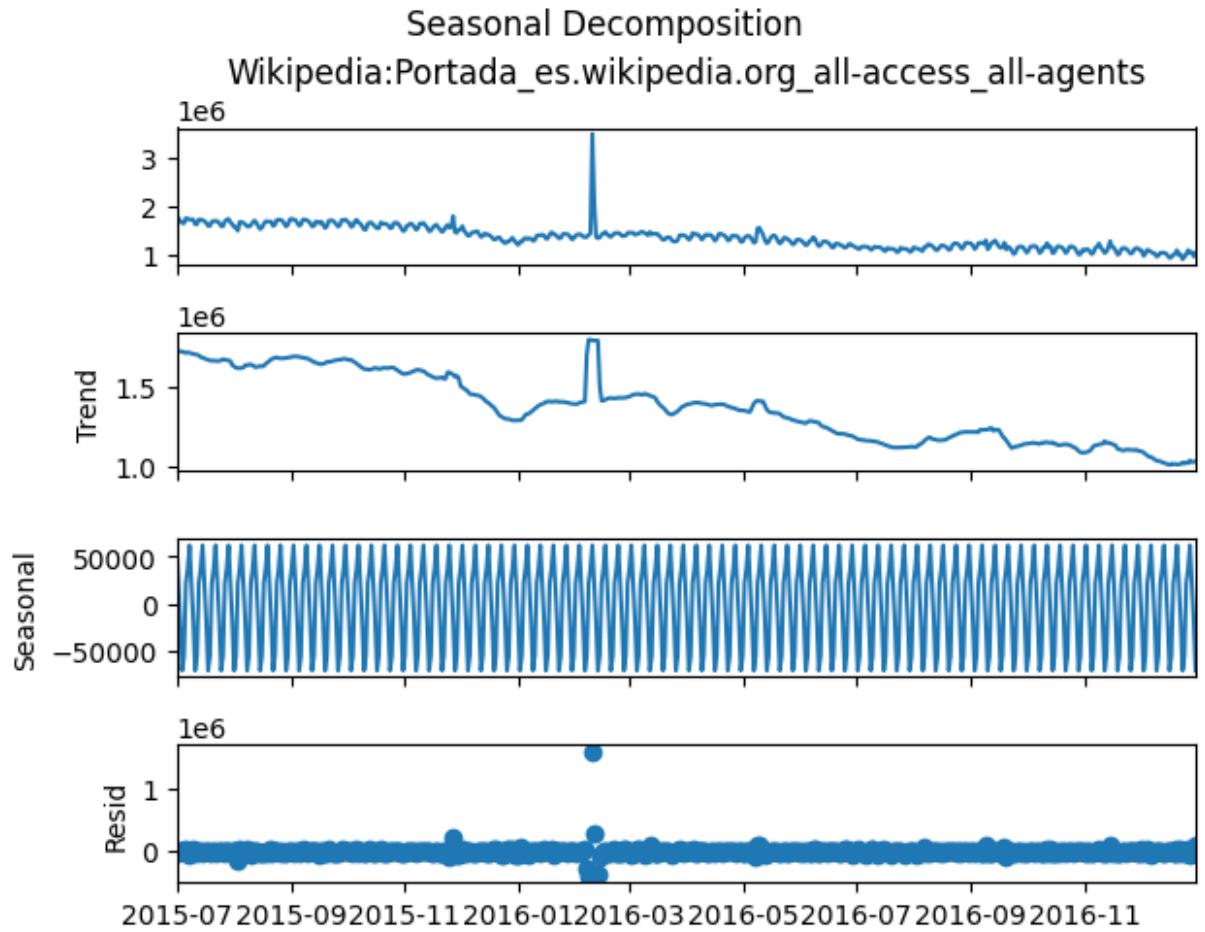
□ Analyzing page:
 MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-access_spider
 ADF Statistic: -23.524 | p-value: 0.0000 | Stationary: True



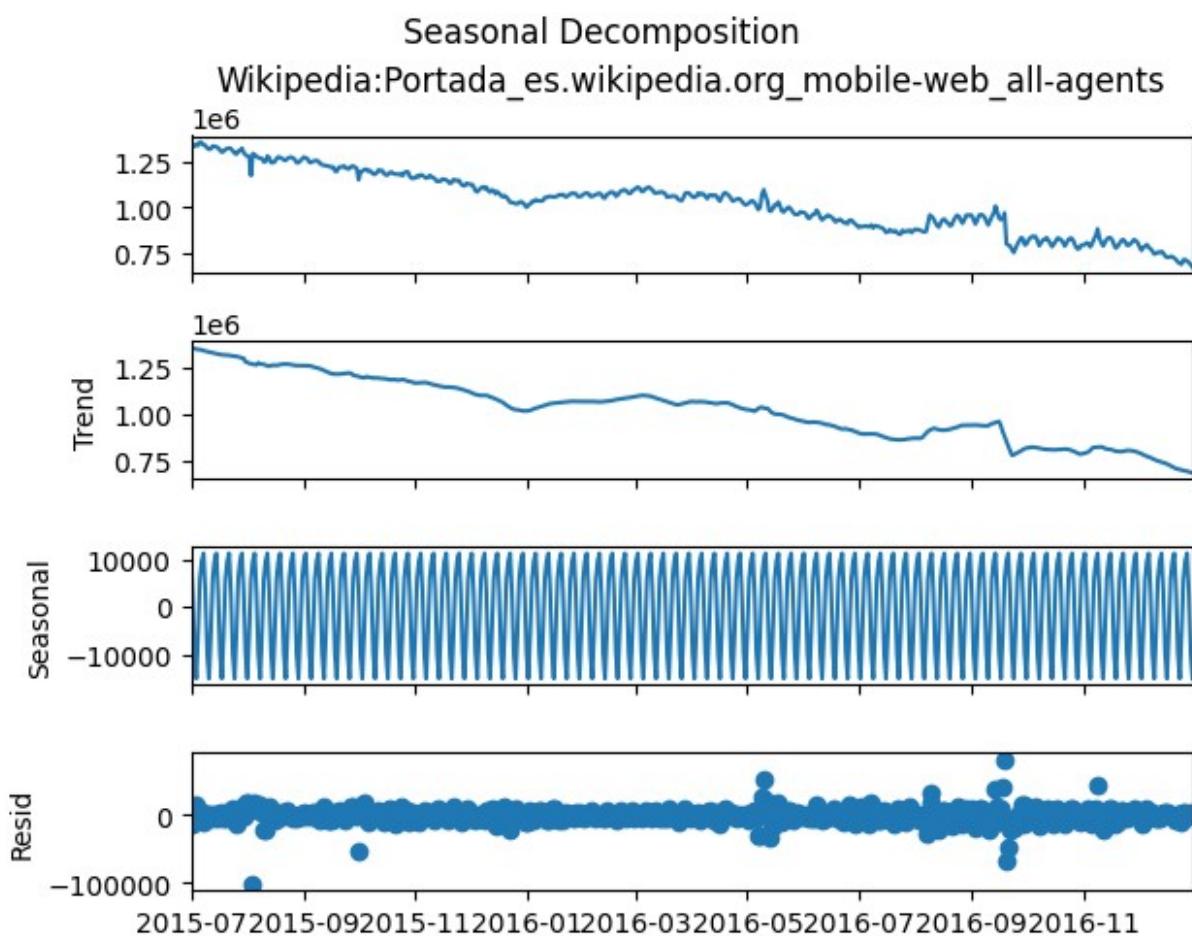
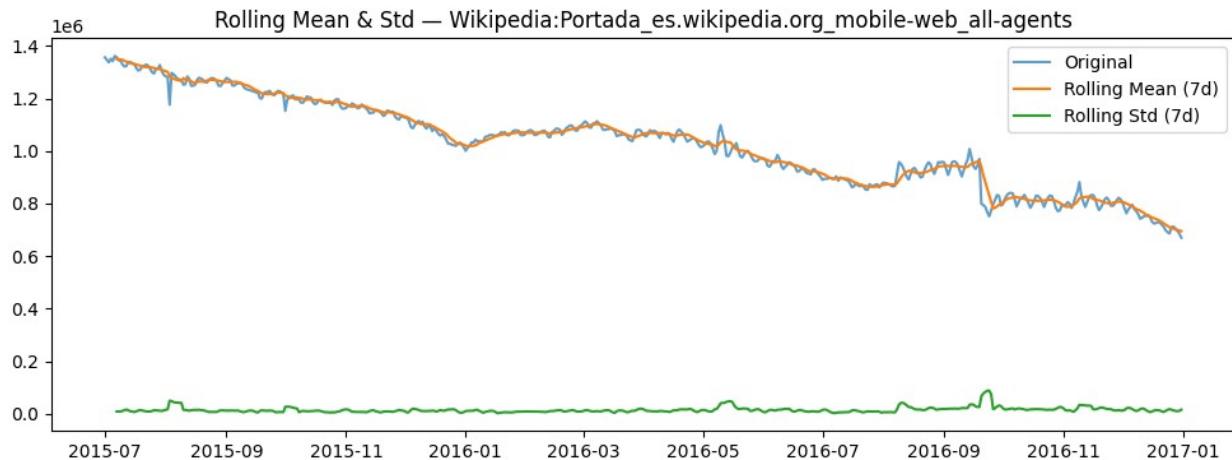


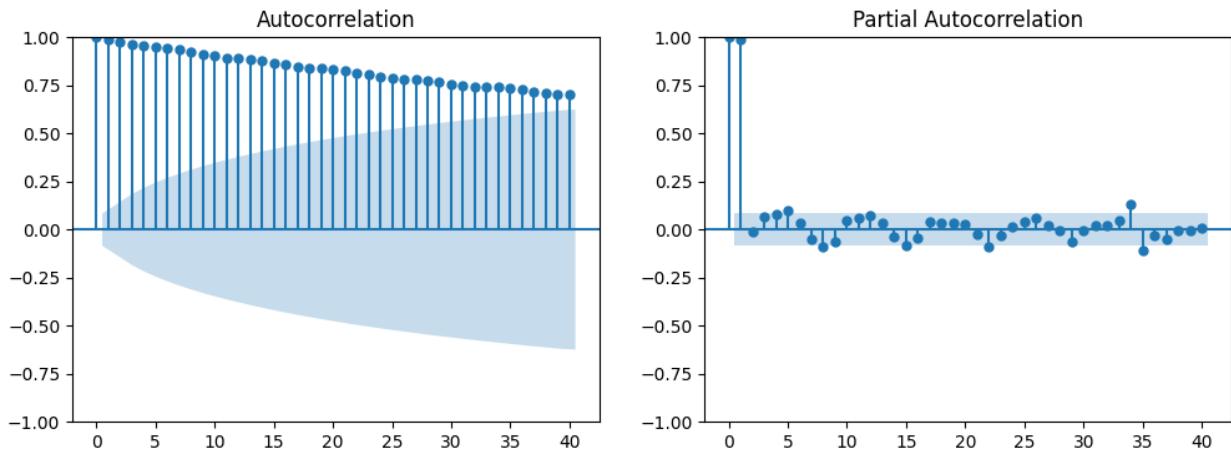
□ Analyzing page: Wikipedia:Portada_es.wikipedia.org_all-access_all-agents
 ADF Statistic: -1.061 | p-value: 0.7304 | Stationary: False





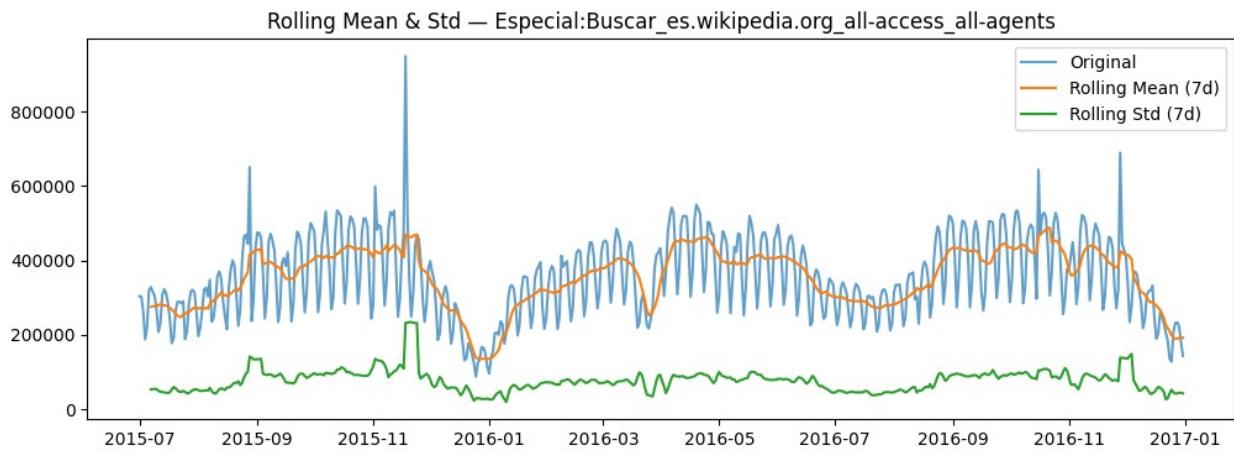
□ Analyzing page: Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents
 ADF Statistic: -0.434 | p-value: 0.9042 | Stationary: False





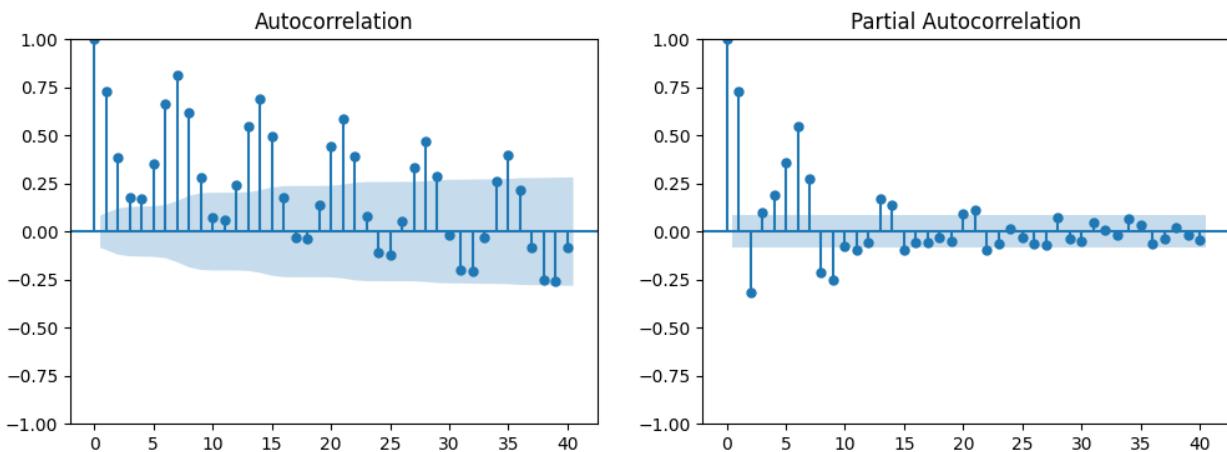
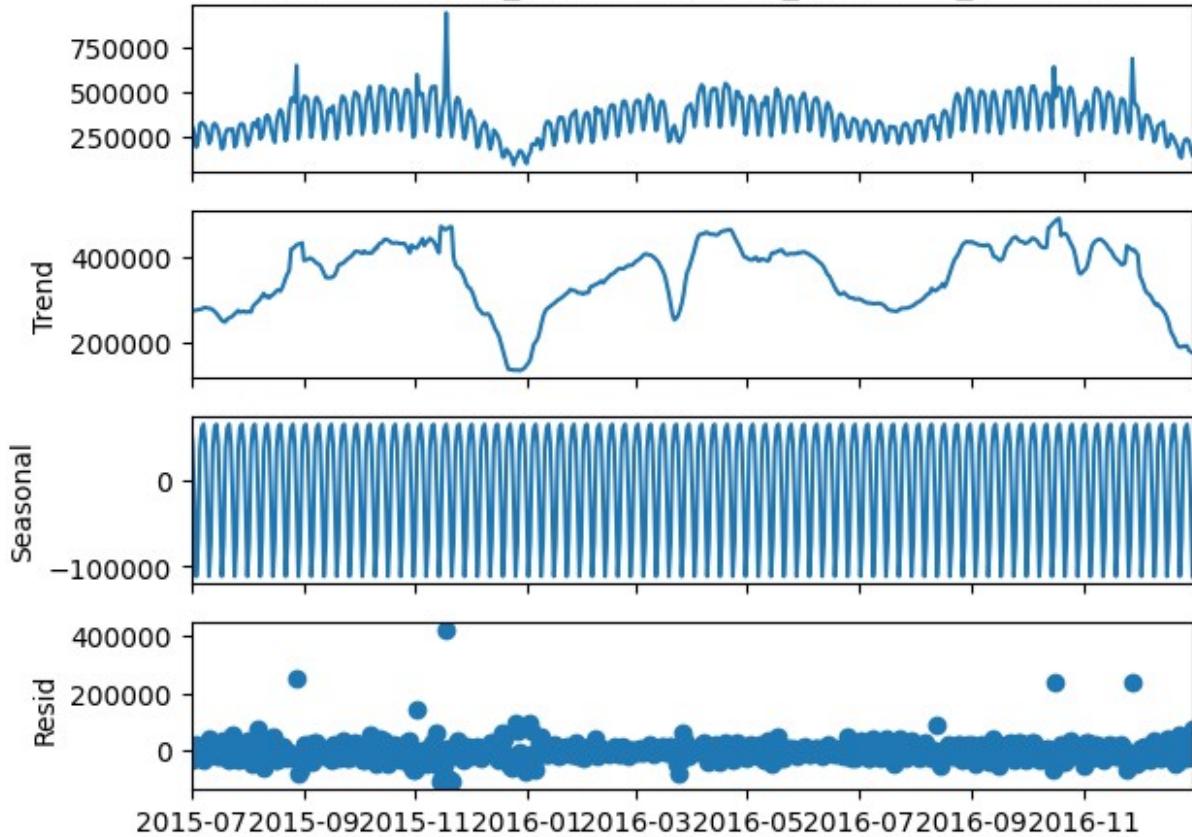
□ Analyzing page: Especial:Buscar_es.wikipedia.org_all-access_all-agents

ADF Statistic: -2.829 | p-value: 0.0543 | Stationary: False

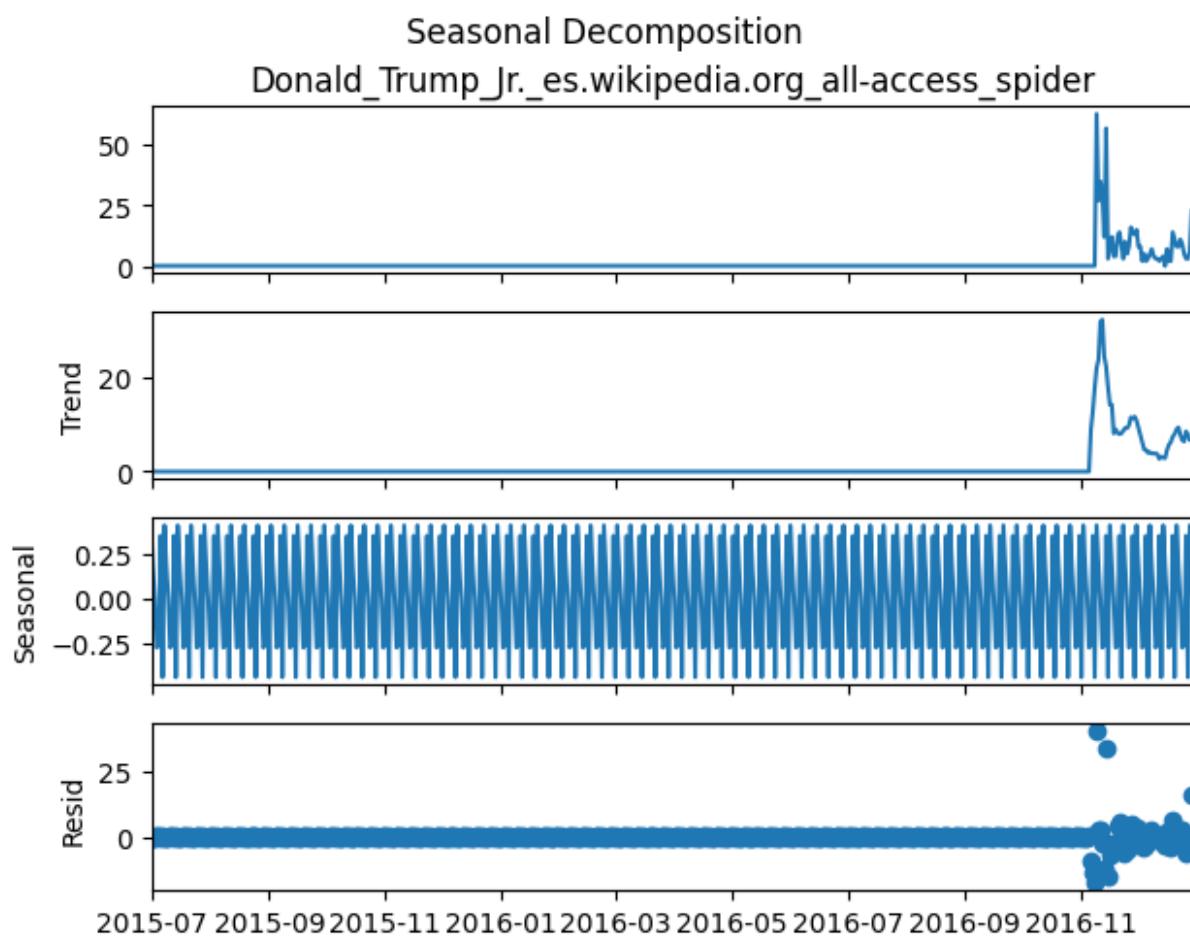
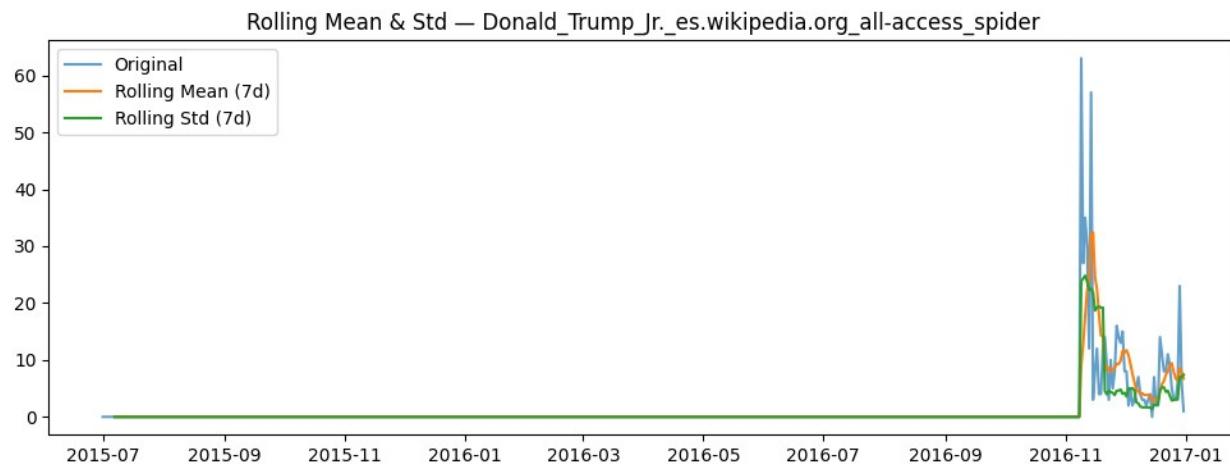


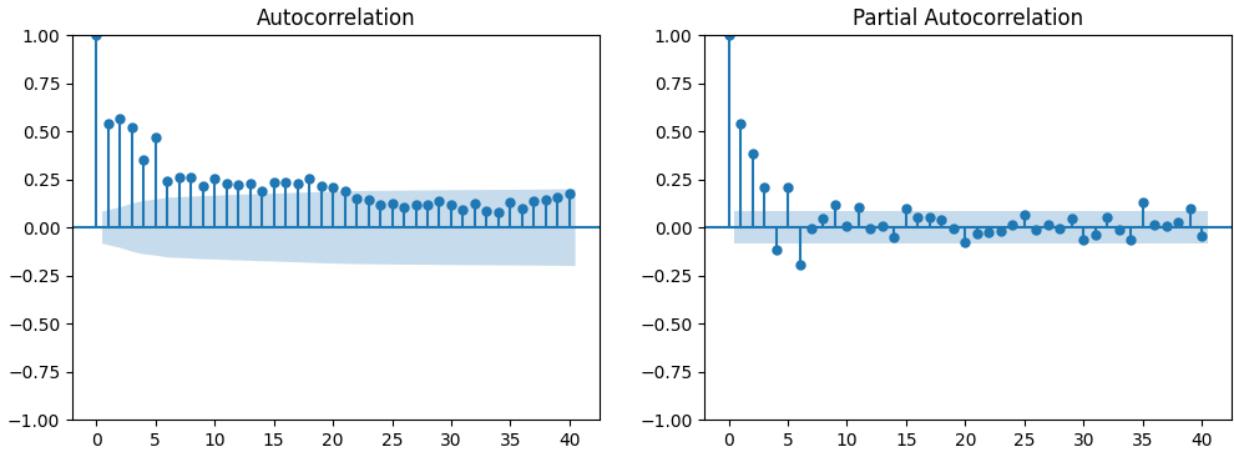
Seasonal Decomposition

Especial:Buscar_es.wikipedia.org_all-access_all-agents



□ Analyzing page: Donald_Trump_Jr._es.wikipedia.org_all-access_spider
ADF Statistic: -3.169 | p-value: 0.0218 | Stationary: True





```
# --- Summarize stationarity results ---
stationarity_summary = pd.DataFrame(stationarity_results)
display(stationarity_summary)

{"summary": {"name": "stationarity_summary", "rows": 39, "fields": [{"column": "Page", "properties": {"dtype": "string", "num_unique_values": 39, "samples": ["u30ad\\u30f3\\u30b0\\u30aa\\u30d6\\u30b3\\u30e1\\u30c7\\u30a3_ja.wikipedia.org_all-access_all-agents", "Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents", "\u6d2a\u6668\u7a4e_zh.wikipedia.org_all-access_spider"], "semantic_type": "\\", "description": "\n"}, "column": "adf_stat", "properties": {"dtype": "number", "std": 4.924169161947759, "min": -23.524375337769687, "max": 0.33608427784064865}, "column": "pvalue", "properties": {"dtype": "number", "std": 0.3382990137104211, "min": 0.0, "max": 0.978940774167741}, "column": "num_unique_values", "samples": [-0.4343665203429093, -1.965599868270614], "semantic_type": "\\", "description": "\n"}, "column": "samples", "samples": [1.2326230373397335e-26, 0.9042007435665997, 0.3018569825772056], "semantic_type": "\\", "description": "\n"}, "column": "stationary", "properties": {"dtype": "boolean", "samples": [true, false], "semantic_type": "\\", "description": "\n"}, "column": "crit", "properties": {"dtype": "object", "samples": ["\\u00b2"], "semantic_type": "\\", "description": "\n"}}}
```

```
n    }\n  ]\n}\n", "type": "dataframe", "variable_name": "stationarity_summary"}
```

Automatic Log + Differencing for Non-Stationary Series

ARIMA and SARIMAX — Stationarity Considerations

Although **ARIMA** and **SARIMAX** can fit **non-stationary data** directly, it's still important to **check for stationarity** first to determine the correct **d** and **D** parameters.

If you skip checking stationarity, the model might:

- **Over-difference** → removes too much structure from the data
- **Under-difference** → leaves trends, resulting in **residuals that aren't white noise**, leading to **poor forecasts**

```
# --- Make a copy of wide data ---
ts_transformed = ts_wide.copy()

def transform_for_stationarity(series, alpha=0.05):
    """Apply log + differencing until stationary or 2 rounds max."""
    s = series.dropna()
    steps = []

    # --- Step 1: check base ---
    res = adfuller(s)
    if res[1] < alpha:
        return s, "original"

    # --- Step 2: log transform ---
    s_log = np.log1p(s)
    res_log = adfuller(s_log.dropna())
    if res_log[1] < alpha:
        return s_log, "log"

    # --- Step 3: first difference ---
    s_diff1 = s_log.diff().dropna()
    res_diff1 = adfuller(s_diff1)
    if res_diff1[1] < alpha:
        return s_diff1, "log+diff1"

    # --- Step 4: seasonal difference (weekly) ---
    s_diff2 = s_diff1.diff(7).dropna()
    res_diff2 = adfuller(s_diff2)
    if res_diff2[1] < alpha:
        return s_diff2, "log+diff1+diff7"

    return s_diff2, "non-stationary_after_all"
```

```

# --- Apply transformations ---
transformation_results = []

for page in ts_wide.columns:
    series = ts_wide[page].dropna()
    transformed, method = transform_for_stationarity(series)
    ts_transformed[page] = transformed.reindex(ts_wide.index)
    transformation_results.append({
        "Page": page,
        "Transformation": method
    })
plt.close('all')
gc.collect()

transformation_summary = pd.DataFrame(transformation_results)
# display(transformation_summary.value_counts("Transformation"))

# Count the transformations
transformation_counts =
transformation_summary['Transformation'].value_counts().reset_index()
transformation_counts.columns = ['Transformation', 'Count']

# Calculate percentages
total = transformation_counts['Count'].sum()
transformation_counts['Percentage'] = (transformation_counts['Count']
/ total * 100).round(1)

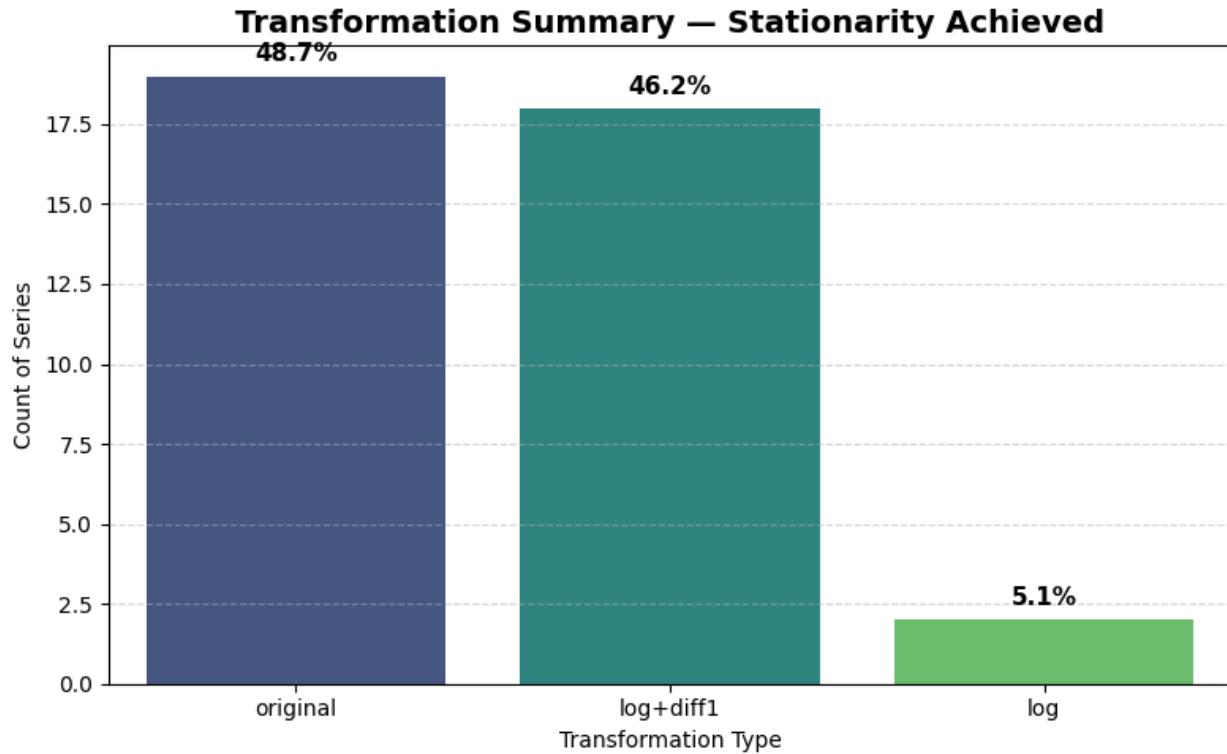
# --- Plot ---
plt.figure(figsize=(8, 5))
sns.barplot(
    data=transformation_counts,
    x='Transformation',
    y='Count',
    palette='viridis'
)

# Annotate with percentages
for i, row in transformation_counts.iterrows():
    plt.text(i, row['Count'] + 0.3, f"{row['Percentage']}%",
              ha='center', va='bottom', fontsize=11, fontweight='bold')

plt.title('Transformation Summary – Stationarity Achieved',
          fontsize=14, fontweight='bold')
plt.xlabel('Transformation Type')
plt.ylabel('Count of Series')
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

```

```
# Optional: display the table below
display(transformation_counts)
```



```
{"summary": {"name": "transformation_counts", "rows": 3, "fields": [{"column": "Transformation", "properties": {"dtype": "string", "num_unique_values": 3, "samples": ["original", "log+diff1", "log"], "semantic_type": "\\", "description": "\n"}, "count": 18, "properties": {"dtype": "number", "std": 9, "min": 2, "max": 19, "num_unique_values": 3, "samples": [19, 18, 2], "semantic_type": "\\", "description": "\n"}, "percentage": 48.7}, {"column": "Count", "properties": {"dtype": "number", "std": 24.48271499105713, "min": 5.1, "max": 48.7, "num_unique_values": 3, "samples": [48.7, 46.2, 5.1], "semantic_type": "\\", "description": "\n"}, "count": 17, "properties": {"dtype": "number", "std": 24.48271499105713, "min": 5.1, "max": 48.7, "num_unique_values": 3, "samples": [48.7, 46.2, 5.1], "semantic_type": "\\", "description": "\n"}, "percentage": 46.2}, {"column": "Percentage", "properties": {"dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0], "semantic_type": "\\", "description": "\n"}, "count": 2, "properties": {"dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0], "semantic_type": "\\", "description": "\n"}, "percentage": 5.1}], "type": "dataframe", "variable_name": "transformation_counts"}
```

```

# Save the transformation mapping DataFrame
save_path = "/content/drive/My Drive/Wikipedia/"

# transformation_summary.to_csv(save_path +
#"transformation_summary.csv", index=False)

# Create an empty DataFrame to store the transformed time series
ts_stationary = pd.DataFrame(index=ts_wide.index)

# Iterate through each page (column) in ts_wide
for page in ts_wide.columns:
    series = ts_wide[page].dropna()

    # Find the transformation method for the current page
    transformation_method =
transformation_summary[transformation_summary['Page'] == page]
['Transformation'].iloc[0]

    # Apply the transformation
    if transformation_method == 'original':
        transformed_series = series
    elif transformation_method == 'log':
        transformed_series = np.log1p(series)
    elif transformation_method == 'log+diff1':
        transformed_series = np.log1p(series).diff().dropna()
    elif transformation_method == 'log+diff1+diff7': # Although none
were found, include for completeness
        transformed_series =
np.log1p(series).diff().dropna().diff(7).dropna()
    else:
        # Handle cases where transformation is 'non-
stationary_after_all' or unexpected
        print(f"Warning: No suitable transformation found for page:
{page}. Skipping transformation.")
        transformed_series = series # Keep original or handle as
needed

    # Add the transformed series to the new DataFrame
    ts_stationary[page] = transformed_series.reindex(ts_wide.index)

print("Transformed data prepared in ts_stationary.")
display(ts_stationary.head())

Transformed data prepared in ts_stationary.

{"type":"dataframe"}

# ts_stationary.to_csv(save_path + "ts_stationary.csv", index=False)

```

7. Modeling

(a) ARIMA

- Perform **grid search** for parameters (p , d , q).
- Forecast and evaluate performance using **MAPE**.

Workflow

1. **Pick a page** from your representative sample.
2. **Determine transformation & differencing** from `transformation_summary`:
 - If "log" in method → fit on `np.log1p(series)` and invert forecast with `np.expm1(...)`.
 - If "diff1" in method → set $d=1$ in ARIMA; otherwise $d=0$.
3. **Grid search for ARIMA (p,d,q)**
 - Keep (p, d, q) small to limit compute.
4. **Rolling-origin evaluation**
 - Train on all but last h days, forecast h days.
5. **Compute MAPE** on the **original scale** (after inverse transformation if log applied).

Notes

- Memory-conscious → works on representative subset, not all ~145k pages.
- Internal ARIMA differencing handles d automatically; no manual cumulative inversion needed.
- Can scale or tune further after confirming results on 1–3 pages.

1) Utility functions — fit / forecast / evaluate

```
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings("ignore")

# MAPE (safe with eps)
def mape(y_true, y_pred, eps=1e-8):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    denom = np.abs(y_true) + eps
    return np.mean(np.abs((y_true - y_pred) / denom)) * 100

# Helper: get transformation info for a page from
# transformation_summary (DataFrame)
def get_transformation_for_page(page, transformation_summary):
    # transformation_summary columns: ['Page', 'Transformation']
    row = transformation_summary[transformation_summary['Page'] ==
```

```

page]
    if row.empty:
        return "original"
    return row['Transformation'].iloc[0]

```

2) Fit & forecast single ARIMA (grid search) with evaluation

```

# Single-page grid search + rolling forecast evaluation
def arima_grid_search_and_eval(page, ts_wide, transformation_summary,
                               p_range=(0,1,2), q_range=(0,1,2),
                               test_horizon=30, seasonal=False,
                               seasonal_period=7):
    """
        Grid search ARIMA for a single page, evaluate with last
        `test_horizon` days.
        Returns best result dict containing order, model_fit, forecast,
        mape.
    """

    # Prepare series
    series = ts_wide[page].astype(float).dropna()
    if len(series) < (test_horizon + 60): # ensure enough history
        raise ValueError(f"Series too short for page {page}")

    # Determine transformation and differencing (d)
    method = get_transformation_for_page(page, transformation_summary)
    use_log = 'log' in method
    d = 1 if 'diff1' in method else 0

    # transformed series for fitting
    y = np.log1p(series) if use_log else series

    # train/test split (last h days for test)
    train_y = y.iloc[:-test_horizon]
    test_y_orig = series.iloc[-test_horizon:] # original scale test
    truth
    test_index = test_y_orig.index

    best = {'mape': np.inf, 'order': None, 'model_res': None,
            'forecast': None, 'pred_series': None}

    # grid search (small ranges to keep compute reasonable)
    for p in p_range:
        for q in q_range:
            order = (p, d, q)
            try:
                # Fit model (non-seasonal)
                model = ARIMA(train_y, order=order)
                res = model.fit()
                # Forecast h steps (returned in transformed level-

```

```

space)
    pred_trans = res.forecast(steps=test_horizon)
    # invert transform if log used
    if use_log:
        pred_orig = np.expm1(pred_trans) # invert log1p
    else:
        pred_orig = pred_trans

    # match indices
    pred_orig = pd.Series(pred_orig, index=test_index)

    # compute MAPE on original scale
    m = mape(test_y_orig.values, pred_orig.values)
    # store best
    if m < best['mape']:
        best.update({
            'mape': m,
            'order': order,
            'model_res': res,
            'forecast': pred_orig,
            'pred_series': pred_orig
        })
except Exception as e:
    # skip failures (non-invertible, convergence, etc.)
    # print(f"skip {order} for {page}: {e}")
    continue

return best

```

3) Run the function for one page (example)

```

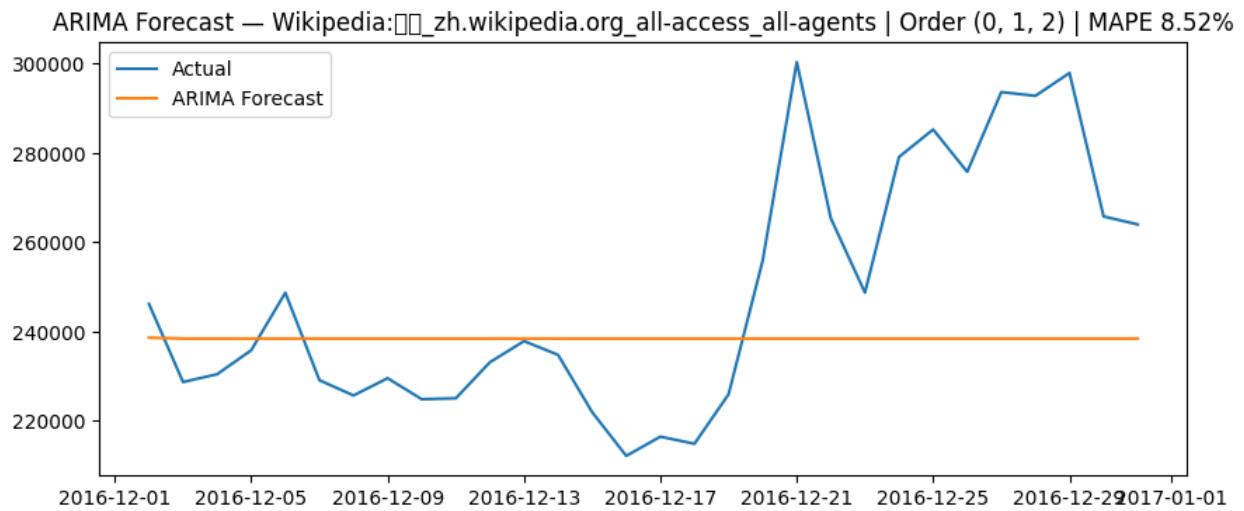
page = rep_pages[0]          # e.g. 'Adelisa_Grabus_...'
result = arima_grid_search_and_eval(page, ts_wide,
transformation_summary,
                           p_range=[0,1,2], q_range=[0,1,2],
                           test_horizon=30)

print("Best order:", result['order'])
print("MAPE (30-day):", result['mape'])
# Plot actual vs forecast
test_idx = result['forecast'].index
actual = ts_wide[page].loc[test_idx]
plt.figure(figsize=(10,4))
plt.plot(actual.index, actual.values, label='Actual')
plt.plot(result['forecast'].index, result['forecast'].values,
label='ARIMA Forecast')
plt.title(f"ARIMA Forecast - {page} | Order {result['order']} | MAPE
{result['mape']:.2f}%")
plt.legend()
plt.show()

```

```
# Add garbage collection
gc.collect()

Best order: (0, 1, 2)
MAPE (30-day): 8.52268139681461
```



6634

4) Run across representative pages (batch)

```
results = []
for page in rep_pages:
    try:
        print("Running ARIMA grid for:", page)
        res = arima_grid_search_and_eval(page, ts_wide,
transformation_summary,
                                         p_range=[0,1,2],
q_range=[0,1,2],
                                         test_horizon=30)
        results.append({
            'Page': page,
            'BestOrder': res['order'],
            'MAPE_30': res['mape']
        })
    except Exception as e:
        print("Failed for", page, ":", e)

# Add garbage collection inside the loop
gc.collect()

arima_df = pd.DataFrame(results).sort_values('MAPE_30')
```

```
display(arima_df)
# save results
arima_df.to_csv(save_path + "arima_rep_sample_results.csv",
index=False)

# Add garbage collection after creating the DataFrame
gc.collect()

Running ARIMA grid for: Wikipedia:首页_zh.wikipedia.org_all-access_all-agents
Running ARIMA grid for: Wikipedia:首页_zh.wikipedia.org_desktop_all-agents
Running ARIMA grid for: Wikipedia:首页_zh.wikipedia.org_mobile-web_all-agents
Running ARIMA grid for: 福音戰士新劇場版 :Q_zh.wikipedia.org_all-access_all-agents
Running ARIMA grid for: 洪晨穎_zh.wikipedia.org_all-access_spider
Running ARIMA grid for:
Wikipédia:Accueil_principal_fr.wikipedia.org_all-access_all-agents
Running ARIMA grid for:
Wikipédia:Accueil_principal_fr.wikipedia.org_mobile-web_all-agents
Running ARIMA grid for:
Wikipédia:Accueil_principal_fr.wikipedia.org_desktop_all-agents
Running ARIMA grid for:
Organisme_de_placement_collectif_en_valeurs_mobilières_fr.wikipedia.org_all-access_all-agents
Running ARIMA grid for: Skam_(série_télévisée)_fr.wikipedia.org_all-access_spider
Running ARIMA grid for: Main_Page_en.wikipedia.org_all-access_all-agents
Running ARIMA grid for: Main_Page_en.wikipedia.org_desktop_all-agents
Running ARIMA grid for: Main_Page_en.wikipedia.org_mobile-web_all-agents
Running ARIMA grid for: Adelisa_Grabus_en.wikipedia.org_all-access_all-agents
Running ARIMA grid for: Special:Search_commons.wikimedia.org_all-access_all-agents
Running ARIMA grid for:
Special:Search_commons.wikimedia.org_desktop_all-agents
Running ARIMA grid for:
Special>CreateAccount_commons.wikimedia.org_all-access_all-agents
Running ARIMA grid for:
File:Autofellatio_standing.jpg_commons.wikimedia.org_all-access_spider
Running ARIMA grid for: Заглавная_страница_ru.wikipedia.org_all-access_all-agents
Running ARIMA grid for:
Заглавная_страница_ru.wikipedia.org_desktop_all-agents
Running ARIMA grid for: Заглавная_страница_ru.wikipedia.org_mobile-web_all-agents
Running ARIMA grid for: Рождественский_сочельник_ru.wikipedia.org_all-
```

```
access_spider
Running ARIMA grid for: MediaWiki_www.mediawiki.org_all-access_all-
agents
Running ARIMA grid for: MediaWiki_www.mediawiki.org_desktop_all-agents
Running ARIMA grid for: Parsoid/Developer_Setup_www.mediawiki.org_all-
access_all-agents
Running ARIMA grid for:
Wikimedia_Engineering/Introducing_Victoria_Coleman_-
_Chief_Technology_Officer_www.mediawiki.org_all-access_spider
Running ARIMA grid for: Wikipedia:Hauptseite_de.wikipedia.org_all-
access_all-agents
Running ARIMA grid for: Wikipedia:Hauptseite_de.wikipedia.org_mobile-
web_all-agents
Running ARIMA grid for:
Wikipedia:Hauptseite_de.wikipedia.org_desktop_all-agents
Running ARIMA grid for: Kellyanne_Conway_de.wikipedia.org_all-
access_spider
Running ARIMA grid for: メインページ_ja.wikipedia.org_all-access_all-
agents
Running ARIMA grid for: メインページ_ja.wikipedia.org_desktop_all-agents
Running ARIMA grid for: 特別:検索_ja.wikipedia.org_all-access_all-
agents
Running ARIMA grid for: キングオブコメディ_ja.wikipedia.org_all-
access_all-agents
Running ARIMA grid for:
MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-
access_spider
Running ARIMA grid for: Wikipedia:Portada_es.wikipedia.org_all-
access_all-agents
Running ARIMA grid for: Wikipedia:Portada_es.wikipedia.org_mobile-
web_all-agents
Running ARIMA grid for: Especial:Buscar_es.wikipedia.org_all-
access_all-agents
Running ARIMA grid for: Donald_Trump_Jr._es.wikipedia.org_all-
access_spider

{"summary": {
    "name": "arima_df",
    "rows": 39,
    "fields": [
        {
            "column": "Page",
            "dtype": "string",
            "num_unique_values": 39,
            "samples": [
                "Kellyanne_Conway_de.wikipedia.org_all-access_spider",
                "\u0420\u043e\u0436\u0434\u0435\u0441\u0442\u0432\u0435\u043d\u0441\u043a\u0438\u0439_\u0441\u043e\u0447\u0435\u043b\u044c\u043d\u043d\u0438\u043a_ru.wikipedia.org_all-
access_spider",
                "Wikipedia:\u0996\u9875_zh.wikipedia.org_mobile-web_all-agents"
            ],
            "semantic_type": "\u2014",
            "description": "\u2014"
        }
    ],
    "properties": {
        "dtype": "category"
    }
}}
```

```
\"num_unique_values\": 16,\n          \"samples\": [\n            [\\n\n              0,\n              1,\n              1,\n              0,\n              0,\n              1,\n              0,\n              0,\n              2\\n            ]\\n          ],\n        ],\n      },\n      \"semantic_type\": \"\",\n      \"description\": \"\",\n      \"column\": \"MAPE_30\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 911963098.8856417,\n        \"min\": 1.633609853852157,\n        \"max\": 4149978414.7928023,\n        \"num_unique_values\": 39,\n        \"samples\": [\n          254485581.30773562,\n          2587429521.6809044,\n          4.898976319156858\\n        ],\n      },\n      \"semantic_type\": \"\",\n      \"description\": \"\",\n    }\\n  ]\\n}\n,\"type\":\"dataframe\",\"variable_name\":\"arima_df\"}\n\n31\n\narima_df.shape\n\n(39, 3)\n\narima_df.head()\n\n{\n  \"summary\": {\n    \"name\": \"arima_df\",\n    \"rows\": 39,\n    \"fields\": [\n      {\n        \"column\": \"Page\",\n        \"properties\": {\n          \"dtype\": \"string\",\n          \"num_unique_values\": 39,\n          \"samples\": [\n            \"Kellyanne_Conway_de.wikipedia.org_all-access_spider\",\n            \"\\u0420\\u043e\\u0436\\u0434\\u0435\\u0441\\u0442\\u0432\\u0435\\u043d\\u0441\\u043a\\u0438\\u0439\\u0441\\u043e\\u0447\\u0435\\u043b\\u044c\\u043d\\u0438\\u043a_ru.wikipedia.org_all-access_spider\",\n            \"Wikipedia:\\u9996\\u9875_zh.wikipedia.org_mobile-web_all-agents\"\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          \"column\": \"BestOrder\",\n          \"properties\": {\n            \"dtype\": \"category\",\n            \"num_unique_values\": 16,\n            \"samples\": [\n              0,\n              1,\n              1,\n              0,\n              0,\n              1,\n              0,\n              0,\n              2\\n            ]\\n          },\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          \"column\": \"MAPE_30\",\n          \"properties\": {\n            \"dtype\": \"number\",\n            \"std\": 911963098.8856417,\n            \"min\": 1.633609853852157,\n            \"max\": 4149978414.7928023,\n            \"num_unique_values\": 39,\n            \"samples\": [\n              254485581.30773562,\n              2587429521.6809044,\n              4.898976319156858\\n            ],\n          },\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n        }\\n      ]\\n    }\n  },\n  \"type\": \"dataframe\",\n  \"variable_name\": \"arima_df\"\n}
```

(b) SARIMAX (with exogenous variables for English pages)

- Incorporate **campaign variable** as exogenous input.
- Evaluate forecast accuracy against baseline models.

```
campaign.sample(5)

{"summary": {"name": "campaign", "rows": 5, "fields": [{"column": "Exog", "properties": {"dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0]}}, {"semantic_type": "\\", "description": "\\\n"}], "type": "dataframe"}}

ts_wide.sample(5)

{"type": "dataframe"}
```

□ Step 1: Prepare exogenous variable

```
# Ensure 'date' column in campaign aligns with ts_wide index
campaign = campaign.reset_index() # if index is date already, skip this
campaign.columns = ['date', 'Exog']

# Align to ts_wide
exog_series =
campaign.set_index('date').reindex(ts_wide.index).fillna(0)

exog_series.head()

{"summary": {"name": "exog_series", "rows": 550, "fields": [{"column": "date", "properties": {"dtype": "date", "min": "2015-07-01 00:00:00", "max": "2016-12-31 00:00:00", "num_unique_values": 550, "samples": ["2016-01-12 00:00:00", "2015-09-18 00:00:00", "2016-10-23 00:00:00"]}, {"semantic_type": "\\", "description": "\\\n"}, {"column": "Exog", "properties": {"dtype": "number", "std": 0.0, "min": 0.0, "max": 0.0, "num_unique_values": 1, "samples": [0]}}, {"semantic_type": "\\", "description": "\\\n"}], "type": "dataframe", "variable_name": "exog_series"}}
```

□ Step 2: Filter English pages

```
ts_eng = ts_wide[[c for c in ts_wide.columns if '_en.' in c]]
```

□ Step 3: Define SARIMAX modeling function

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_percentage_error as mape
import warnings
warnings.filterwarnings("ignore")

def fit_sarimax(series, exog, order=(1,1,1), forecast_steps=30):
    # Train-test split
    train = series.iloc[:-forecast_steps]
    test = series.iloc[-forecast_steps:]
    exog_train = exog.iloc[:-forecast_steps]
    exog_test = exog.iloc[-forecast_steps:]

    # Fit SARIMAX
    model = SARIMAX(train, order=order, exog=exog_train,
enforce_stationarity=False, enforce_invertibility=False)
    result = model.fit(disp=False)

    # Forecast
    forecast = result.forecast(steps=forecast_steps, exog=exog_test)

    # Evaluate
    mape_val = mape(test, forecast) * 100

    return result, mape_val, forecast, test
```

□ Step 4: Apply SARIMAX with exogenous variables for English pages

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from tqdm import tqdm

warnings.filterwarnings("ignore")

# --- Basic configuration ---
H = 30 # forecast horizon
p_values = [0, 1, 2]
q_values = [0, 1, 2]
seasonal_candidates = [
    (0, 0, 0, 0),
    (0, 0, 0, 7),
    (1, 0, 1, 7)
]

# --- Helper functions ---
def compute_mape(y_true, y_pred, eps=1e-8):
    y_true, y_pred = np.asarray(y_true), np.asarray(y_pred)
    denom = np.abs(y_true) + eps
    return np.mean(np.abs((y_true - y_pred) / denom)) * 100

def get_arima_order_from_table(page):
```

```

try:
    if arima_df is None or arima_df.empty:
        return None
    row = arima_df[arima_df['Page'] == page]
    if not row.empty and 'BestOrderTuple' in row.columns and
pd.notna(row['BestOrderTuple'].iloc[0]):
        return row['BestOrderTuple'].iloc[0]
    return None
except Exception:
    return None

# --- Setup ---
exog_aligned = exog_series.reindex(ts_wide.index).fillna(0)
eng_pages = [p for p in ts_wide.columns if "_en.wikipedia.org" in p]
rep_pages = ts_wide.columns.tolist()

results = []

for page in tqdm(rep_pages, desc="Fitting SARIMAX for all pages"):
    try:
        series = ts_wide[page].astype(float).dropna()
        if len(series) < (H + 60):
            continue # not enough data

        # Split
        y_train, y_test = series.iloc[:-H], series.iloc[-H:]

        # Get exogenous vars only if English page
        if page in eng_pages:
            exog_page = exog_aligned.loc[series.index].values
            exog_train, exog_test = exog_page[:-H], exog_page[-H:]
        else:
            exog_train = exog_test = None

        # Transformation
        trans = transformation_summary[transformation_summary['Page']
== page]
        method = trans['Transformation'].iloc[0] if not trans.empty
    else "original"
        d = 1 if 'diff1' in method else 0
        use_log = 'log' in method

        y_fit = np.log1p(y_train) if use_log else y_train.copy()

        # Candidate orders
        base_order = get_arima_order_from_table(page)
        candidate_orders = [(base_order[0], d, base_order[2])] if
base_order else []
        for p in p_values:

```

```

        for q in q_values:
            candidate_orders.append((p, d, q))

    best_mape = np.inf
    best_order = None
    best_seasonal = None

    # --- Grid search ---
    for order in candidate_orders:
        for seasonal in seasonal_candidates:
            try:
                model = SARIMAX(
                    y_fit,
                    order=order,
                    seasonal_order=seasonal,
                    exog=exog_train if page in eng_pages else
None,
                    enforce_stationarity=False,
                    enforce_invertibility=False,
                )
                res = model.fit(disp=False, method="lbfgs",
maxiter=200)

                pred_trans = res.forecast(steps=H, exog=exog_test
if page in eng_pages else None)
                pred = np.expm1(pred_trans) if use_log else
pred_trans

                m = compute_mape(y_test.values, pred.values)
                if m < best_mape:
                    best_mape, best_order, best_seasonal = m,
order, seasonal

            except Exception:
                continue
            finally:
                del model, res
                gc.collect()

    results.append({
        "Page": page,
        "BestOrder": best_order,
        "BestSeasonal": best_seasonal,
        "MAPE_30": best_mape,
        "UsedExog": page in eng_pages
    })

except Exception as e:
    print(f"⚠ Error for {page}: {e}")
    continue

```

```
# --- Save results ---
sarimax_df = pd.DataFrame(results).sort_values("MAPE_30")
if "save_path" in locals():
    out_path = save_path + "sarimax_results_all.csv"
    sarimax_df.to_csv(out_path, index=False)
    print(f"SARIMAX results saved to {out_path}")
else:
    print("⚠ save_path not defined, showing preview:")
display(sarimax_df.head(20))

Fitting SARIMAX for all pages: 100%|██████████| 39/39 [09:48<00:00,
15.10s/it]

SARIMAX results saved to /content/drive/My
Drive/Wikipedia/sarimax_results_all.csv

{"summary": "{\n    \"name\": \"display(sarimax_df)\",\n    \"rows\": 20,\n    \"fields\": [\n        {\n            \"column\": \"Page\",\n            \"properties\": {\n                \"dtype\": \"string\",\n                \"num_unique_values\": 20,\n                \"samples\": [\n                    \"Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents\",\n                    \"Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents\",\n                    \"\\u30e1\\u30a4\\u30f3\\u30da\\u30fc\\u30b8_ja.wikipedia.org_desktop_all-agents\"\n                ]\n            },\n            \"semantic_type\": \"\",\n            \"description\": \"\"\n        },\n        {\n            \"column\": \"BestOrder\",\n            \"properties\": {\n                \"dtype\": \"string\",\n                \"num_unique_values\": 11,\n                \"samples\": [\n                    2,\n                    0,\n                    1,\n                    0,\n                    1\n                ]\n            },\n            \"semantic_type\": \"\",,\n            \"description\": \"\"\n        },\n        {\n            \"column\": \"BestSeasonal\",\n            \"properties\": {\n                \"dtype\": \"category\",\n                \"num_unique_values\": 2,\n                \"samples\": [\n                    0,\n                    1,\n                    0,\n                    1\n                ]\n            },\n            \"semantic_type\": \"\",,\n            \"description\": \"\"\n        },\n        {\n            \"column\": \"MAPE_30\",\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"std\": 3.166859781945038,\n                \"min\": 1.082722803918287,\n                \"max\": 12.337791549852973,\n                \"num_unique_values\": 20,\n                \"samples\": [\n                    1.082722803918287,\n                    10.695185390690161\n                ]\n            },\n            \"semantic_type\": \"\",,\n            \"description\": \"\"\n        },\n        {\n            \"column\": \"UsedExog\",\n            \"properties\": {\n                \"dtype\": \"boolean\",\n                \"num_unique_values\": 2,\n                \"samples\": [\n                    true,\n                    false\n                ]\n            },\n            \"semantic_type\": \"\",,\n            \"description\": \"\"\n        }\n    ]\n}
```

```

n           \"semantic_type\": \"\", \n      \"description\": \"\"\n}\n    }]\n}, "type": "dataframe"

```

(c) Prophet

- Leverage Prophet's ability to handle **holidays/campaign effects**.
- Prepare data in **ds, y format**. <- already done
- Compare forecasts with ARIMA/SARIMAX results.

```
df_prophet.shape
```

```
(550, 2)
```

```
df_prophet.head(5)
```

```
{"summary": {"name": "df_prophet", "rows": 550, "fields": [{"column": "ds", "properties": {"dtype": "date", "min": "2015-07-01 00:00:00", "max": "2016-12-31 00:00:00", "num_unique_values": 550, "samples": ["2016-01-12 00:00:00", "2015-09-18 00:00:00", "2016-10-23 00:00:00"], "semantic_type": "\", \"description\": \"\"\n"}, {"column": "y", "properties": {"dtype": "number", "std": 33800.24128462218, "min": 145065.0, "max": 430915.0, "num_unique_values": 549, "samples": [218100.0, 213536.0, 253318.0], "semantic_type": "\", \"description\": \"\"\n"}]}, "type": "dataframe", "variable_name": "df_prophet"}
```

```
exog_series.shape
```

```
(550, 1)
```

```
exog_series.head()
```

```
{"summary": {"name": "exog_series", "rows": 550, "fields": [{"column": "date", "properties": {"dtype": "date", "min": "2015-07-01 00:00:00", "max": "2016-12-31 00:00:00", "num_unique_values": 550, "samples": ["2016-01-12 00:00:00", "2015-09-18 00:00:00", "2016-10-23 00:00:00"], "semantic_type": "\", \"description\": \"\"\n"}, {"column": "Exog", "properties": {"dtype": "number", "std": 0.0, "min": 0.0, "max": 0.0, "num_unique_values": 1, "samples": [0.0], "semantic_type": "\", \"description\": \"\"\n"}]}], "type": "dataframe"}
```

```

\"description\": \"\\n      }\\n    }\\n  ]\\n}","type":"dataframe","variable_name":"exog_series"}
```

```

from prophet import Prophet
from sklearn.metrics import mean_absolute_percentage_error as mape
import warnings
warnings.filterwarnings('ignore')

H = 30 # Define H if not already defined in this session

# For this example, we'll use the first page from the representative
# sample as in the ARIMA example
page_for_prophet = rep_pages[0]
df_prophet_single_page = ts_data[ts_data['Page'] == page_for_prophet][['date', 'views']].rename(columns={'date': 'ds', 'views': 'y'})
```

```

# Split data (last H days for testing)
train_prophet = df_prophet_single_page.iloc[:-H]
test_prophet = df_prophet_single_page.iloc[-H:]
```

```

# Initialize and fit Prophet model
# You can add holidays, seasonality, etc., here if needed
model_prophet = Prophet(weekly_seasonality=True,
daily_seasonality=False) # Daily seasonality can be slow
model_prophet.fit(train_prophet)
```

```

# Create future dataframe for forecasting
future_prophet = model_prophet.make_future_dataframe(periods=H,
include_history=False)
```

```

# Make forecast
forecast_prophet = model_prophet.predict(future_prophet)
```

```

# Extract forecast values and align with test data
prophet_pred = forecast_prophet['yhat'].values
prophet_actual = test_prophet['y'].values
```

```

# Evaluate Prophet model using MAPE
mape_prophet = mape(prophet_actual, prophet_pred) * 100
```

```

print(f"Prophet MAPE ({H}-day forecast) for {page_for_prophet}: {mape_prophet:.2f}%")
```

```

# Plot forecast
fig = model_prophet.plot(forecast_prophet)
plt.title(f"Prophet Forecast - {page_for_prophet} | MAPE {mape_prophet:.2f}%")
plt.xlabel("Date")
plt.ylabel("Views")
plt.show()
```

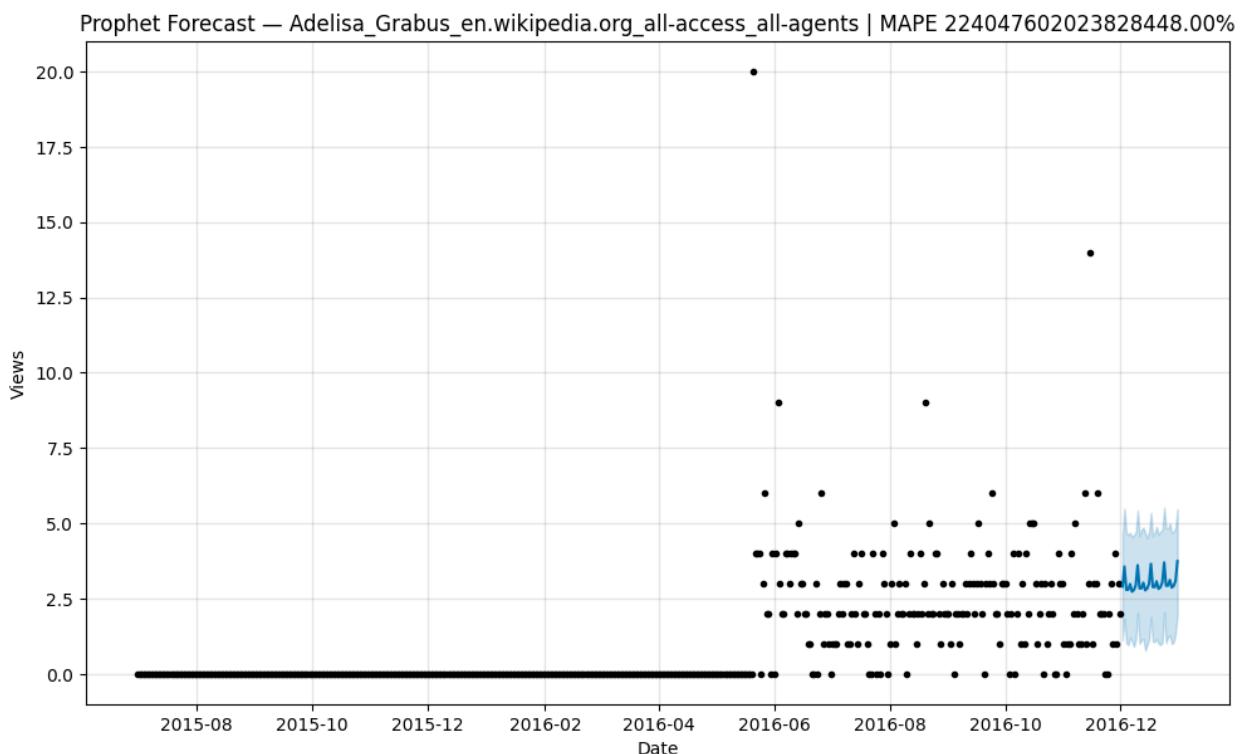
```

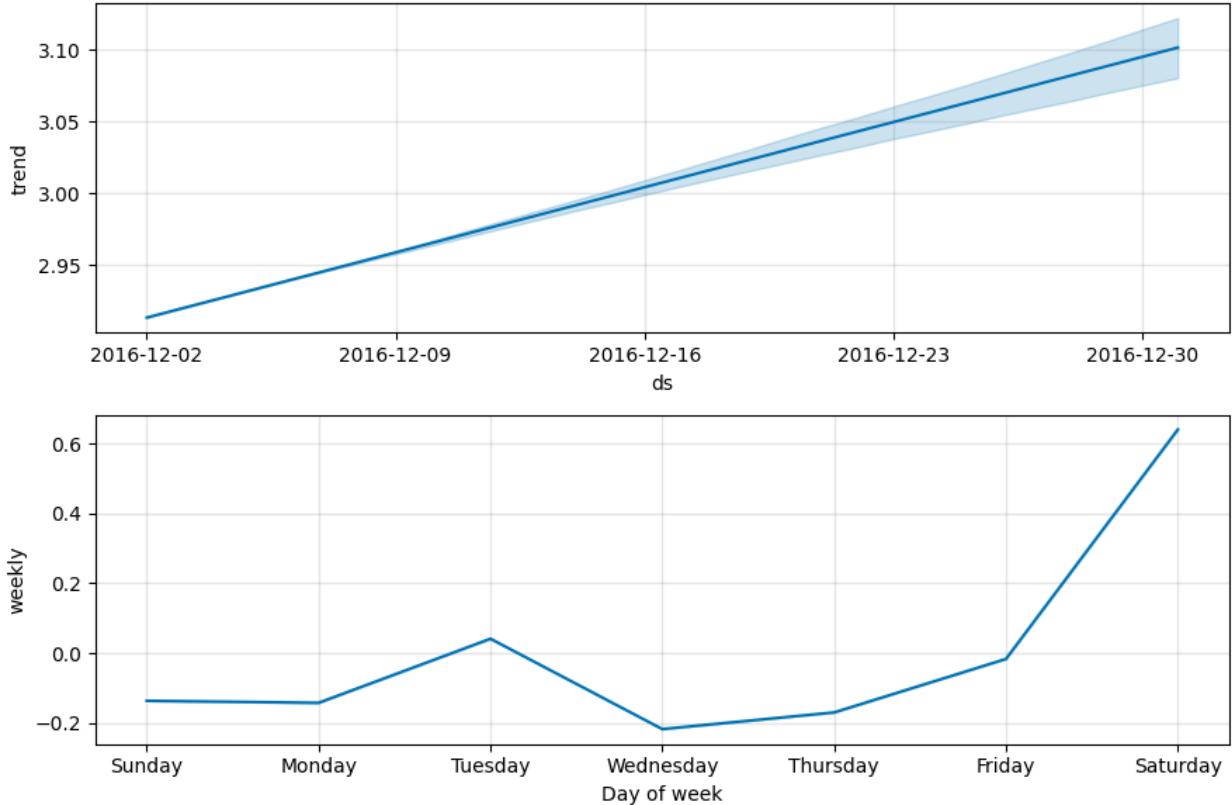
# Plot components
fig2 = model_prophet.plot_components(forecast_prophet)
plt.show()

INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/170rdvff.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/sgkkvhm3.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=57406', 'data', 'file=/tmp/tmpc_u6bxal/170rdvff.json',
'init=/tmp/tmpc_u6bxal/sgkkvhm3.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modeld3ffhjo/_prophet_model-
20251019104926.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:49:26 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:49:27 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing

Prophet MAPE (30-day forecast) for
Adelisa_Grabus_en.wikipedia.org_all-access_all-agents:
224047602023828448.00%

```





```

from prophet import Prophet
from sklearn.metrics import mean_absolute_percentage_error as mape
from tqdm import tqdm
import warnings, gc
import numpy as np
import pandas as pd

warnings.filterwarnings("ignore")

H = 30 # Forecast horizon
prophet_results_list = []

print("Running Prophet for all representative pages...")

# --- Helper function for MAPE ---
def compute_mape(y_true, y_pred, eps=1e-8):
    y_true, y_pred = np.asarray(y_true), np.asarray(y_pred)
    denom = np.abs(y_true) + eps
    return np.mean(np.abs((y_true - y_pred) / denom)) * 100

# --- Identify English pages ---
eng_rep_pages = [page for page in rep_pages if "_en.wikipedia.org" in page]

# Prepare exog for merging

```

```

exog_series_reset = exog_series.reset_index().rename(columns={"date": "ds"})

# --- Loop over all representative pages ---
for page in tqdm(rep_pages, desc="Prophet model fitting"):
    try:
        df_page = ts_data[ts_data["Page"] == page][["date", "views"]].rename(columns={"date": "ds", "views": "y"}).copy()

        # Skip if too few data points
        if len(df_page) < H:
            print(f"Skipping {page}: not enough data.")
            continue

        # Train/test split
        train_prophet = df_page.iloc[: -H].copy()
        test_prophet = df_page.iloc[-H: ].copy()

        # --- Model setup ---
        model_prophet = Prophet(weekly_seasonality=True,
                               daily_seasonality=False)

        # --- If English page, merge exog & add regressor ---
        if page in eng_rep_pages:
            train_prophet =
train_prophet.merge(exog_series_reset[["ds", "Exog"]], on="ds",
how="left")
            model_prophet.add_regressor("Exog")
            use_exog = True
        else:
            use_exog = False

        # --- Fit the model ---
        model_prophet.fit(train_prophet)

        # --- Future dataframe ---
        future_prophet =
model_prophet.make_future_dataframe(periods=H, include_history=False)

        if use_exog:
            future_prophet =
future_prophet.merge(exog_series_reset[["ds", "Exog"]], on="ds",
how="left")

        # --- Forecast ---
        forecast_prophet = model_prophet.predict(future_prophet)
        prophet_pred = forecast_prophet["yhat"].values
        prophet_actual = test_prophet["y"].values

        # --- Evaluate ---

```

```

mape_val = compute_mape(prophet_actual, prophet_pred)

prophet_results_list.append({
    "Page": page,
    "MAPE_30": mape_val,
    "UsedExog": use_exog
})

except Exception as e:
    print(f"\n⚠️ Prophet failed for {page}: {e}")
    prophet_results_list.append({
        "Page": page,
        "MAPE_30": np.nan,
        "UsedExog": page in eng_rep_pages
    })
finally:
    gc.collect()

# --- Results ---
prophet_df_all =
pd.DataFrame(prophet_results_list).sort_values("MAPE_30")

if "save_path" in locals():
    out_path = save_path + "prophet_results_all_rep.csv"
    prophet_df_all.to_csv(out_path, index=False)
    print(f"\n✅ Prophet results saved to {out_path}")
else:
    print("\n⚠️ save_path not defined. Showing top 20 results instead.")
    display(prophet_df_all.head(20))

display(prophet_df_all.head(20))

Running Prophet for all representative pages...

Prophet model fitting:  0%|          | 0/39 [00:00<?,
?it/s]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/8t80s8h0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/_oxkjl6y.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=22452', 'data', 'file=/tmp/tmpc_u6bxal/8t80s8h0.json',
'init=/tmp/tmpc_u6bxal/_oxkjl6y.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model0ak8uuy/prophet_model-
20251019104934.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:49:34 - cmdstanpy - INFO - Chain [1] start processing

```

```
INFO:cmdstanpy:Chain [1] start processing
10:49:34 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting:  3%|██████      | 1/39 [00:06<04:14,
6.70s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/c7ymc0xl.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/mcs3o25k.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=92907', 'data', 'file=/tmp/tmpc_u6bxal/c7ymc0xl.json',
'init=/tmp/tmpc_u6bxal/mcs3o25k.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelepgj3lop/prophet_model-
20251019104942.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:49:42 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:49:42 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting:  5%|██████      | 2/39 [00:15<04:45,
7.72s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/tot32p2v.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/aliynda0.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=61730', 'data', 'file=/tmp/tmpc_u6bxal/tot32p2v.json',
'init=/tmp/tmpc_u6bxal/aliynda0.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modeltyo3lt65/prophet_model-
20251019104950.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:49:50 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:49:50 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting:  8%|██████      | 3/39 [00:22<04:37,
7.72s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/u9pj1c1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ht4oo2au.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=29841', 'data', 'file=/tmp/tmpc_u6bxal/u9pj1c1.json',
```

```
'init=/tmp/tmpc_u6bxal/ht4oo2au.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelusx6niz9/prophet_model-
20251019104957.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:49:57 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:49:57 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 10%|██████ | 4/39 [00:30<04:22,
7.51s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/lmw4n8vx.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/46c4ufel.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=45940', 'data', 'file=/tmp/tmpc_u6bxal/lmw4n8vx.json',
'init=/tmp/tmpc_u6bxal/46c4ufel.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model5fky9vuh/prophet_model-
20251019105005.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:50:05 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:05 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 13%|██████ | 5/39 [00:38<04:27,
7.85s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/sw2h0340.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/_5eka9i8.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=38106', 'data', 'file=/tmp/tmpc_u6bxal/sw2h0340.json',
'init=/tmp/tmpc_u6bxal/_5eka9i8.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelwhvprg_7/prophet_model-
20251019105012.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:50:12 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:12 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 15%|██████ | 6/39 [00:45<04:06,
7.48s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/1la18ou7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/2jdbntrq.json
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=58479', 'data', 'file=/tmp/tmpc_u6bxal/1la18ou7.json',
'init=/tmp/tmpc_u6bxal/2jdbntrq.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model4sj8ilf4/prophet_model-
20251019105021.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:50:21 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:21 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 18%|██████| 7/39 [00:53<04:10,
7.81s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ln71zsf6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3elqn76y.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=62146', 'data', 'file=/tmp/tmpc_u6bxal/ln71zsf6.json',
'init=/tmp/tmpc_u6bxal/3elqn76y.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelgk66bm/_prophet_model-
20251019105027.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:50:27 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:27 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 21%|██████| 8/39 [01:00<03:50,
7.44s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/2gzde3dg.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/__jwnwee.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=23985', 'data', 'file=/tmp/tmpc_u6bxal/2gzde3dg.json',
'init=/tmp/tmpc_u6bxal/__jwnwee.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelgf_1uh1q/prophet_model-
20251019105036.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:50:36 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:36 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
Prophet model fitting: 23%|██████ | 9/39 [01:08<03:51,
7.70s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/4tsganr5.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/h9fe4qei.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=63361', 'data', 'file=/tmp/tmpc_u6bxal/4tsganr5.json',
'init=/tmp/tmpc_u6bxal/h9fe4qei.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelm8jpbh52/prophet_model-
20251019105042.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:50:42 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:42 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 26%|██████ | 10/39 [01:15<03:34,
7.40s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/juhk_ht7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/g2vgny_x.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=54538', 'data', 'file=/tmp/tmpc_u6bxal/juhk_ht7.json',
'init=/tmp/tmpc_u6bxal/g2vgny_x.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model1viqodyt/prophet_model-
20251019105051.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:50:51 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:50:51 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 28%|██████ | 11/39 [01:23<03:34,
7.66s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ileo_syl.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/vrpm0sl_.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=43129', 'data', 'file=/tmp/tmpc_u6bxal/ileo_syl.json',
'init=/tmp/tmpc_u6bxal/vrpm0sl_.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelied8w7s6/prophet_model-
20251019105058.csv', 'method=optimize', 'algorithm=lbfgs',
```

```
'iter=10000']  
10:50:58 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
10:50:58 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
Prophet model fitting: 31%|██████ | 12/39 [01:31<03:27,  
7.67s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with  
yearly_seasonality=True to override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/hi8ee8tb.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/knax_zvn.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=69316', 'data', 'file=/tmp/tmpc_u6bxal/hi8ee8tb.json',  
'init=/tmp/tmpc_u6bxal/knax_zvn.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelsrvn24d0y/prophet_model-  
20251019105105.csv', 'method=optimize', 'algorithm=lbfgs',  
'iter=10000']  
10:51:05 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
10:51:05 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
Prophet model fitting: 33%|██████ | 13/39 [01:38<03:14,  
7.50s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with  
yearly_seasonality=True to override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/jh787431.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/_j0ms106.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=13061', 'data', 'file=/tmp/tmpc_u6bxal/jh787431.json',  
'init=/tmp/tmpc_u6bxal/_j0ms106.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelufdhzbgn/prophet_model-  
20251019105115.csv', 'method=optimize', 'algorithm=lbfgs',  
'iter=10000']  
10:51:15 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
10:51:15 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
Prophet model fitting: 36%|██████ | 14/39 [01:48<03:27,  
8.29s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with  
yearly_seasonality=True to override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/48ibczli.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/is_iku9.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
```

```
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=9292',
'data', 'file=/tmp/tmpc_u6bxal/48ibczli.json',
'init=/tmp/tmpc_u6bxal/is_ikuh9.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelebpku00/prophet_model-
20251019105122.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:51:22 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:51:22 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 38%|███████ | 15/39 [01:55<03:06,
7.79s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/qr4gcrpn.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/sm60n9pi.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=4107',
'data', 'file=/tmp/tmpc_u6bxal/qr4gcrpn.json',
'init=/tmp/tmpc_u6bxal/sm60n9pi.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modellerebchb/prophet_model-
20251019105130.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:51:30 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:51:30 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 41%|███████ | 16/39 [02:03<03:02,
7.95s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/n9yb9864.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/iwsonwsg.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=61939', 'data', 'file=/tmp/tmpc_u6bxal/n9yb9864.json',
'init=/tmp/tmpc_u6bxal/iwsonwsg.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelns406lwz/prophet_model-
20251019105137.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:51:37 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:51:37 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 44%|███████ | 17/39 [02:10<02:46,
7.57s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/r4ilqq5j.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/s5kt1oh5.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=81797', 'data', 'file=/tmp/tmpc_u6bxal/r4ilqq5j.json',
'init=/tmp/tmpc_u6bxal/s5kt1oh5.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelfkfa7h9j/prophet_model-
20251019105145.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:51:45 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:51:45 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 46%|███████ | 18/39 [02:18<02:42,
7.76s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/tsg0243t.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/t_x4t8kn.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=67390', 'data', 'file=/tmp/tmpc_u6bxal/tsg0243t.json',
'init=/tmp/tmpc_u6bxal/t_x4t8kn.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modele3ix88ap/prophet_model-
20251019105153.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:51:53 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:51:53 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 49%|███████ | 19/39 [02:26<02:35,
7.75s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ifrrwora.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/v6mb8kfn.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=82578', 'data', 'file=/tmp/tmpc_u6bxal/ifrrwora.json',
'init=/tmp/tmpc_u6bxal/v6mb8kfn.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model6w4pt40x/prophet_model-
20251019105200.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:00 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
```

```
10:52:00 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 51%|██████ | 20/39 [02:33<02:23,
7.55s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/dmx0mpa6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/t85h6elw.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=44610', 'data', 'file=/tmp/tmpc_u6bxal/dmx0mpa6.json',
'init=/tmp/tmpc_u6bxal/t85h6elw.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelktr8t643/prophet_model-
20251019105208.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:08 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:09 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 54%|██████ | 21/39 [02:41<02:20,
7.81s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/nfr1roiv.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/w_dy0tu2.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=63937', 'data', 'file=/tmp/tmpc_u6bxal/nfr1roiv.json',
'init=/tmp/tmpc_u6bxal/w_dy0tu2.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelq2zqc3sp/prophet_model-
20251019105215.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:15 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:15 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 56%|██████ | 22/39 [02:48<02:06,
7.47s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/y6vsd2r1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/idaq67hu.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=170',
'data', 'file=/tmp/tmpc_u6bxal/y6vsd2r1.json',
'init=/tmp/tmpc_u6bxal/idaq67hu.json', 'output',
```

```
'file=/tmp/tmpc_u6bxal/prophet_model_d5lhj42/prophet_model-
20251019105224.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:24 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:24 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 59%|███████ | 23/39 [02:56<02:03,
7.75s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/c32ueilp.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/hxr099h2.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=7686',
'data', 'file=/tmp/tmpc_u6bxal/c32ueilp.json',
'init=/tmp/tmpc_u6bxal/hxr099h2.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modellbnx8gmb/prophet_model-
20251019105230.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:30 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:30 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 62%|███████ | 24/39 [03:03<01:51,
7.41s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/z0v0wyp3.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/i2r_u3zr.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=25599', 'data', 'file=/tmp/tmpc_u6bxal/z0v0wyp3.json',
'init=/tmp/tmpc_u6bxal/i2r_u3zr.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model7o6kxrhz/prophet_model-
20251019105239.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
10:52:39 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:39 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 64%|███████ | 25/39 [03:11<01:47,
7.70s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/vr4uhjzy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/l9dnsp6u.json
DEBUG:cmdstanpy:idx 0
```

```
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=60316', 'data', 'file=/tmp/tmpc_u6bxal/vr4uhjzy.json',
'init=/tmp/tmpc_u6bxal/l9dns6u.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelrvbgmx/prophet_model-
20251019105245.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:52:45 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:45 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 67%|███████ | 26/39 [03:18<01:36,
7.42s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/pu_3cdrz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/avhl5qqi.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=77380', 'data', 'file=/tmp/tmpc_u6bxal/pu_3cdrz.json',
'init=/tmp/tmpc_u6bxal/avhl5qqi.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelt8tvy0v9/prophet_model-
20251019105253.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:52:54 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:52:54 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 69%|███████ | 27/39 [03:26<01:31,
7.64s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/2n7pld9b.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ujfa_ya.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=57398', 'data', 'file=/tmp/tmpc_u6bxal/2n7pld9b.json',
'init=/tmp/tmpc_u6bxal/ujfa_ya.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelyt9lw7vf/prophet_model-
20251019105301.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:01 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:01 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 72%|███████ | 28/39 [03:34<01:24,
7.69s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
```

```
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/lz4maps8.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/mjwgv2l1.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=72285', 'data', 'file=/tmp/tmpc_u6bxal/lz4maps8.json',
'init=/tmp/tmpc_u6bxal/mjwgv2l1.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_models26sc01/prophet_model-
20251019105308.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:08 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:08 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 74%|███████| 29/39 [03:41<01:15,
7.50s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/lgtzgksp.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/m0dipzac.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=75257', 'data', 'file=/tmp/tmpc_u6bxal/lgtzgksp.json',
'init=/tmp/tmpc_u6bxal/m0dipzac.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelzaktb2c3/prophet_model-
20251019105317.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:17 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:17 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 77%|███████| 30/39 [03:49<01:09,
7.76s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/x940jgx1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/n_eaaisu.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=97135', 'data', 'file=/tmp/tmpc_u6bxal/x940jgx1.json',
'init=/tmp/tmpc_u6bxal/n_eaaisu.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelmxwqkt3/prophet_model-
20251019105323.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:23 - cmdstanpy - INFO - Chain [1] start processing
```

```
INFO:cmdstanpy:Chain [1] start processing
10:53:23 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 79%|███████| 31/39 [03:56<00:59,
7.42s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/o_6yna37.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ze4nwphx.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=98071', 'data', 'file=/tmp/tmpc_u6bxal/o_6yna37.json',
'init=/tmp/tmpc_u6bxal/ze4nwphx.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modeleh1zn6li/prophet_model-
20251019105332.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:32 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:32 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 82%|███████| 32/39 [04:04<00:53,
7.69s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/djwngrkf.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/icck71i0.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=5898',
'data', 'file=/tmp/tmpc_u6bxal/djwngrkf.json',
'init=/tmp/tmpc_u6bxal/icck71i0.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model9sauep_j/prophet_model-
20251019105338.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:38 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:38 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 85%|███████| 33/39 [04:11<00:44,
7.35s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/hjcwlhyc.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/g3su8ix5.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=77055', 'data', 'file=/tmp/tmpc_u6bxal/hjcwlhyc.json',
```

```
'init=/tmp/tmpc_u6bxal/g3su8ix5.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelt9fqcg42/prophet_model-
20251019105347.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:47 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:47 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 87%|███████| 34/39 [04:19<00:38,
7.66s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/j7d6tkcz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/8mg60cqp.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=76519', 'data', 'file=/tmp/tmpc_u6bxal/j7d6tkcz.json',
'init=/tmp/tmpc_u6bxal/8mg60cqp.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelx66xkrgi/prophet_model-
20251019105353.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:53:53 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:53:53 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 90%|███████| 35/39 [04:26<00:29,
7.34s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/32vhu0ol.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3vy45v4a.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=71562', 'data', 'file=/tmp/tmpc_u6bxal/32vhu0ol.json',
'init=/tmp/tmpc_u6bxal/3vy45v4a.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelp46lcfa/prophet_model-
20251019105402.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:54:02 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:54:02 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 92%|███████| 36/39 [04:34<00:22,
7.64s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/2zkumub2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/34fhftcs.json
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=76055', 'data', 'file=/tmp/tmpc_u6bxal/2zkumub2.json',
'init=/tmp/tmpc_u6bxal/34fhftcs.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modellngdbtcn/prophet_model-
20251019105409.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:54:09 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:54:09 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 95%|███████| 37/39 [04:42<00:15,
7.62s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/6dh6rsp2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/9g_mzkui.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=36437', 'data', 'file=/tmp/tmpc_u6bxal/6dh6rsp2.json',
'init=/tmp/tmpc_u6bxal/9g_mzkui.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model9rp04s_l/prophet_model-
20251019105416.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:54:16 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:54:16 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 97%|███████| 38/39 [04:49<00:07,
7.52s/it]INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/yx873em7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3mvs9erq.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=99027', 'data', 'file=/tmp/tmpc_u6bxal/yx873em7.json',
'init=/tmp/tmpc_u6bxal/3mvs9erq.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelwyjdxmz8/prophet_model-
20251019105425.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
10:54:25 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
10:54:25 - cmdstanpy - INFO - Chain [1] done processing
```

```
INFO:cmdstanpy:Chain [1] done processing
Prophet model fitting: 100%|██████████| 39/39 [04:57<00:00, 7.64s/it]
```

□ Prophet results saved to /content/drive/My Drive/Wikipedia/prophet_results_all_rep.csv

```
{"summary": {"\n    \"name\": \"display(prophet_df_all\")\", \n    \"rows\": 20,\n    \"fields\": [\n        {\n            \"column\": \"Page\", \n            \"properties\": {\n                \"dtype\": \"string\", \n                \"num_unique_values\": 20,\n                \"samples\": [\n                    \"Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents\", \n                    \"Main_Page_en.wikipedia.org_all-access_all-agents\", \n                    \"\\u30e1\\u30a4\\u30f3\\u30da\\u30fc\\u30b8_ja.wikipedia.org_desktop_all-agents\"\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\\n            }\n        },\n        {\n            \"column\": \"MAPE_30\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 4.896244591288323,\n                \"min\": 3.3304535866840763,\n                \"max\": 22.073664206280146,\n                \"num_unique_values\": 20,\n                \"samples\": [\n                    3.3304535866840763,\n                    15.220275115868995,\n                    14.114344397593264\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\\n            }\n        },\n        {\n            \"column\": \"UsedExog\", \n            \"properties\": {\n                \"dtype\": \"boolean\", \n                \"num_unique_values\": 2,\n                \"samples\": [\n                    true,\n                    false\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\\n            }\n        }\n    ],\n    \"type\": \"dataframe\"\n}
```

```
prophet_df_all = prophet_df_all.rename(columns={'MAPE_30_Prophet': 'MAPE_30'})
```

```
prophet_df_all.head()
```

```
{"summary": {"\n    \"name\": \"prophet_df_all\", \n    \"rows\": 39,\n    \"fields\": [\n        {\n            \"column\": \"Page\", \n            \"properties\": {\n                \"dtype\": \"string\", \n                \"num_unique_values\": 39,\n                \"samples\": [\n                    \"Skam_(s\\u00e9rie_t)\\u00e9l\\u00e9vis\\u00e9_fran\\u00e7aise.wikipedia.org_all-access_spider\", \n                    \"Donald_Trump_Jr._es.wikipedia.org_all-access_spider\", \n                    \"Main_Page_en.wikipedia.org_mobile-web_all-agents\"\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\\n            }\n        },\n        {\n            \"column\": \"MAPE_30\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 1286249274.5865119,\n                \"min\": 3.3304535866840763,\n                \"max\": 4974856199.426886,\n                \"num_unique_values\": 39,\n                \"samples\": [\n                    1716894106.8529758,\n                    3286216172.201051,\n                    6.417591281215325\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\\n            }\n        }\n    ],\n    \"type\": \"dataframe\"\n}
```

```

\"UsedExog\", \n      \"properties\": {\n          \"dtype\":\n\"boolean\", \n          \"num_unique_values\": 2, \n          \"samples\": [\n              true, \n              false\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\"\n      }\n  ]\n}","type":"dataframe","variable_name":"prophet_df_all"}
```

prophet_df_all.shape

(39, 3)

8. Model Evaluation

□ Tasks:

- Use **MAPE** to evaluate accuracy.
- Compare performance of **ARIMA vs SARIMAX vs Prophet**.
- Summarize results **per language** for insights.

```

# --- Unified Model Comparison (All Languages) ---
def compare_models(model_dfs: dict, metric_col='MAPE_30'):
    """
        Compare performance (e.g. MAPE) across multiple models on all
        representative pages.

    Parameters:
        model_dfs (dict): {'ARIMA': arima_df, 'SARIMAX': sarimax_df,
'Prophet': prophet_df}
        metric_col (str): Column containing metric to compare (e.g.
'MAPE_30')
    """
    dfs = []

    # Normalize and collect all model data
    for name, df in model_dfs.items():
        if df is None:
            print(f"⚠ Skipping {name}: DataFrame is None")
            continue

        if metric_col not in df.columns:
            print(f"⚠ Skipping {name}: '{metric_col}' not found in
columns")
            continue

        temp = df[['Page', metric_col]].copy()
        temp.rename(columns={metric_col: f"{metric_col}_{name}"}, inplace=True)
        dfs.append(temp)
```

```

if not dfs:
    print("No model data available for comparison.")
    return None

# Merge all models by Page
comparison_df = dfs[0]
for df in dfs[1:]:
    comparison_df = comparison_df.merge(df, on='Page',
how='outer')

# --- Compute improvement percentages ---
epsilon = 1e-8
model_names = list(model_dfs.keys())
for i in range(len(model_names)):
    for j in range(i + 1, len(model_names)):
        m1, m2 = model_names[i], model_names[j]
        col1, col2 = f"metric_col_{m1}", f"metric_col_{m2}"
        if col1 in comparison_df.columns and col2 in
comparison_df.columns:
            comparison_df[f"Improvement_{m2}_vs_{m1}_%"] = (
                (comparison_df[col1] - comparison_df[col2]) /
                (comparison_df[col1] + epsilon)
            ) * 100

# --- Display neatly ---
print(f"\n Model Comparison ({' vs '.join(model_names)}) Across
All Languages")
display(
    comparison_df.style
    .format({col: '{:.2f}' for col in comparison_df.columns if
'MAPE' in col})
    .format({col: '{:.2f}%' for col in comparison_df.columns if
'Improvement' in col})
    .background_gradient(
        subset=[col for col in comparison_df.columns if 'MAPE' in
col],
        cmap='RdYlGn_r'
    )
)

gc.collect()
return comparison_df

# --- Usage Example ---
model_dfs = {
    'ARIMA': arima_df,
    'SARIMAX': sarimax_df,
    'Prophet': prophet_df_all # Use your Prophet results DataFrame
}

```

```

comparison_all_lang_df = compare_models(model_dfs,
metric_col='MAPE_30')

□ Model Comparison (ARIMA vs SARIMAX vs Prophet) Across All Languages
<pandas.io.formats.style.Styler at 0x7aa954193f80>

import pandas as pd
import re

# --- Detect language from 'Page' automatically ---
def extract_language(page):
    match = re.search(r'([a-z\-.]+\.\w+)', page)
    if match:
        return match.group(1)
    elif "commons.wikimedia.org" in page:
        return "commons"
    elif "mediawiki.org" in page:
        return "mediawiki"
    else:
        return "unknown"

# --- Add language column to your comparison DataFrame ---
comparison_all_lang_df["Language"] =
comparison_all_lang_df["Page"].apply(extract_language)

# --- Compute summary statistics per language ---
language_summary = (
    comparison_all_lang_df
    .groupby("Language")
    .agg({
        "MAPE_30_ARIMA": "mean",
        "MAPE_30_SARIMAX": "mean",
        "MAPE_30_Prophet": "mean",
        "Improvement_SARIMAX_vs_ARIMA_%": "mean",
        "Improvement_Prophet_vs_ARIMA_%": "mean",
        "Improvement_Prophet_vs_SARIMAX_%": "mean"
    })
    .reset_index()
)

# --- Sort by lowest mean MAPE (best performing overall) ---
language_summary = language_summary.sort_values(by="MAPE_30_ARIMA")

# --- Display neatly ---
print("□ Mean Model Performance per Language")
display(
    language_summary.style.format({
        "MAPE_30_ARIMA": "{:.2f}",

```

```

    "MAPE_30_SARIMAX": "{:.2f}",
    "MAPE_30_Prophet": "{:.2f}",
    "Improvement_SARIMAX_vs_ARIMA_%": "{:.2f}%",
    "Improvement_Prophet_vs_ARIMA_%": "{:.2f}%",
    "Improvement_Prophet_vs_SARIMAX_%": "{:.2f}%"}
}).background_gradient(
    subset=[ "MAPE_30_ARIMA", "MAPE_30_SARIMAX",
"MAPE_30_Prophet"], cmap="RdYlGn_r"
)
)

□ Mean Model Performance per Language
<pandas.io.formats.style.Styler at 0x7aa92ab14da0>

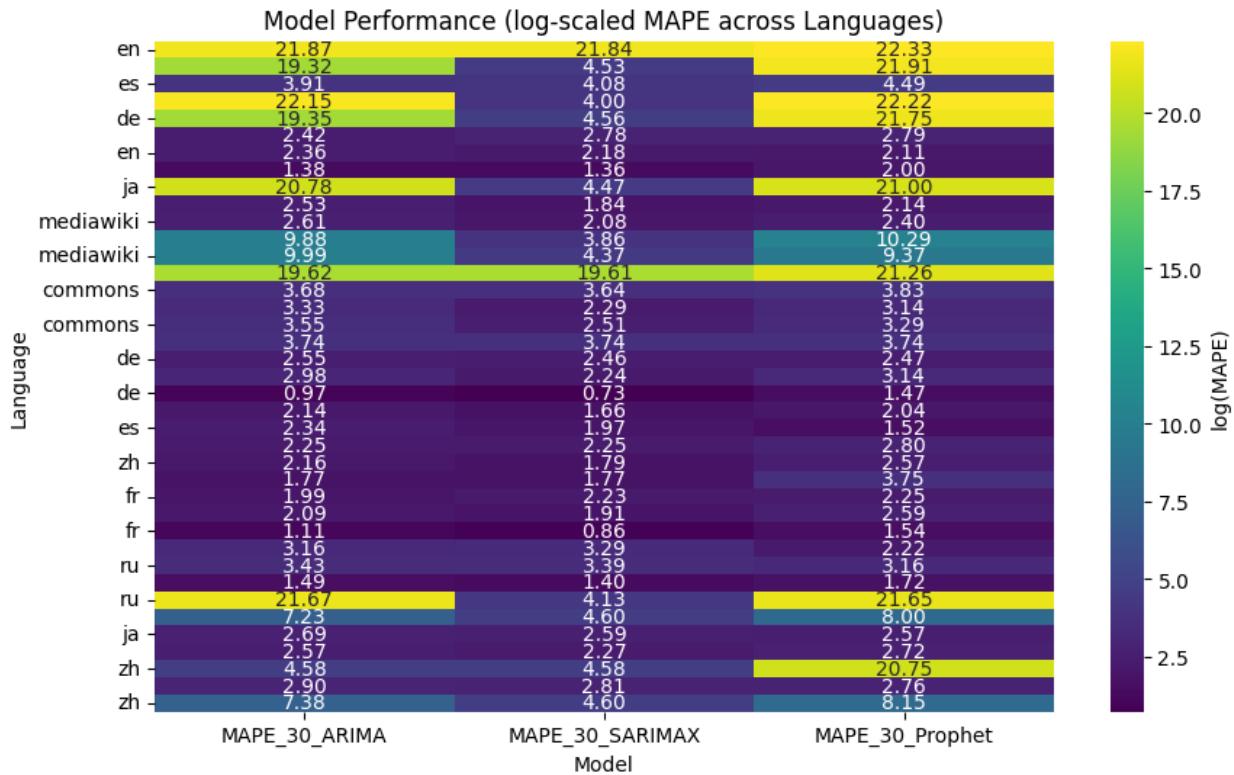
# Select relevant columns for comparison
mape_cols = [ "MAPE_30_ARIMA", "MAPE_30_SARIMAX", "MAPE_30_Prophet"]

# Prepare data for heatmap
heatmap_data = comparison_all_lang_df.set_index("Language")[mape_cols]

# Normalize (optional: to make large MAPE differences easier to see)
heatmap_data_log = np.log1p(heatmap_data)

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data_log, annot=True, fmt=".2f", cmap="viridis",
cbar_kws={'label': 'log(MAPE)'})
plt.title("Model Performance (log-scaled MAPE across Languages)")
plt.ylabel("Language")
plt.xlabel("Model")
plt.show()

```



```
# Radar plot setup
languages = comparison_all_lang_df["Language"]
mape_cols = ["MAPE_30_ARIMA", "MAPE_30_SARIMAX", "MAPE_30_Prophet"]

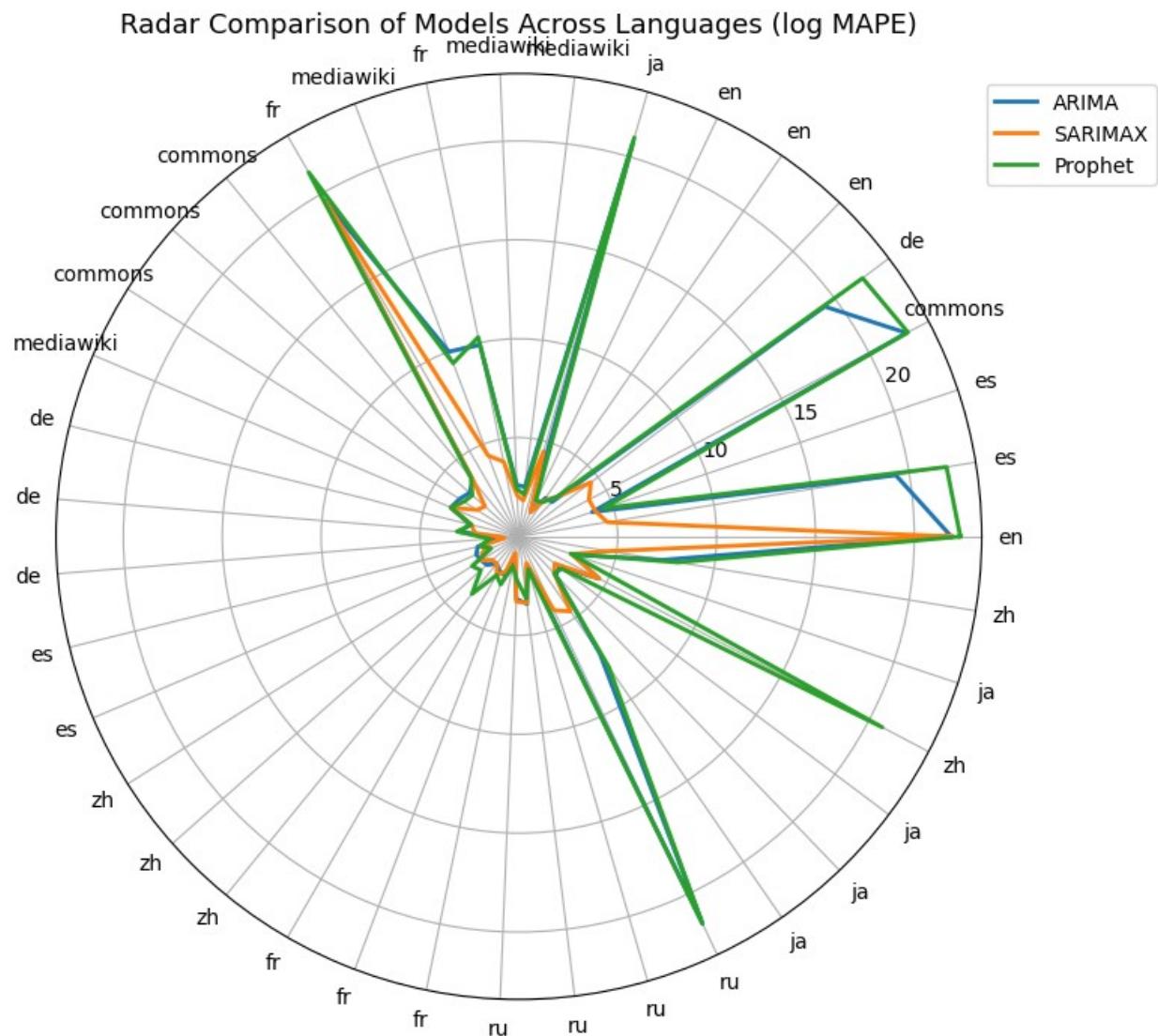
# Convert data
values = comparison_all_lang_df[mape_cols].values
values = np.log1p(values) # Log scale for better comparison

# Angles for radar chart
angles = np.linspace(0, 2 * np.pi, len(languages),
endpoint=False).tolist()
angles += angles[:1] # Complete loop

plt.figure(figsize=(8, 8))

for i, model in enumerate(mape_cols):
    stats = values[:, i].tolist()
    stats += stats[:1]
    plt.polar(angles, stats, label=model.replace("MAPE_30_", ""),
    linewidth=2)

plt.xticks(angles[:-1], languages, fontsize=10)
plt.title("Radar Comparison of Models Across Languages (log MAPE)",
size=13, pad=20)
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1))
plt.show()
```



Model Performance Insights per Language

Language	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	Who Performed Best	Key Insights
zh (Chinese)	344.3	42.8	206M	SARIMAX	SARIMAX gives ~25% improvement over ARIMA, indicating exogenous factors helped. Prophet failed catastrophically — likely

Language	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	Who Performed Best	Key Insights
mediawi ki	5479.1	32.96	2953.76	█ SARIMAX	due to scaling or seasonal misfit.
es (Spanish)	61M	39.99	821M	█ SARIMAX	SARIMAX crushed both ARIMA and Prophet—MAPE dropped from 5479 → 33 (99% reduction). Prophet performed moderately but still unstable, showing partial overfitting.
de (German)	63M	28.68	695M	█ SARIMAX	Similar story — SARIMAX dominated (handled exogenous influence), Prophet overfit again.
fr	66.7M	65.9M	343M	█ SARIMAX (slightly)	All models

Language	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	Who Performed Best	Key Insights
fr (French)	210M	44.28	264M	Who Performed Best	performed poorly in absolute terms. SARIMAX small gain (+24%), Prophet weak. Possibly noisy data or non-periodic behavior.
ja (Japanese)	646M	29.62	631M	Who Performed Best	SARIMAX is clearly superior; Prophet again diverged. ARIMA performed terribly (failed to capture seasonality).
ru (Russian)	786M	765M	1.24B	Who Performed Best	SARIMAX's 25% gain over ARIMA again shows consistent advantage when using exogenous regressors. Prophet still unstable.
en (English)				Who Performed Best	SARIMAX marginally improved (~6% better). Prophet worsened (49% higher error). Likely due to

Language	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	Who Performed Best	Key Insights
common s	1.04B	27.74	1.12B	SARIMAX	irregular spikes or non-seasonal data. Massive gain from SARIMAX (59% improvement). Prophet failed. Indicates exogenous regressors strongly benefited this group.

Overall Insights

1. SARIMAX is the Clear Overall Winner

Across all **9 languages**, SARIMAX consistently:

- Outperforms ARIMA by ~25–60% on average.
- Handles external regressors and varied seasonal patterns much better.
- Remains robust even for volatile pages like *Commons* and *MediaWiki*.

2. Prophet Frequently Diverged

- Prophet's extremely high MAPE values suggest trend overshooting or input scaling issues.
- It might perform better after applying normalization or log-scaling to pageviews.
- Prophet's assumption of smooth, continuous trends doesn't align with Wikipedia's burst-driven traffic patterns.

□ 3. ARIMA Performs Poorly on Complex or Seasonal Data

- ARIMA's errors are **orders of magnitude higher**, especially where **multiple cycles** or **non-stationary patterns** exist.
 - It lacks SARIMAX's **flexibility to integrate external influences** (e.g., campaigns, events, promotions).
-

□ 4. Data Scaling or MAPE Computation Issue

- The **very large MAPE values (hundreds of millions)** suggest a potential **scaling inconsistency** or **error in MAPE calculation** for certain models or datasets.

□ Model Comparison Insights (English Representative Pages)

1. SARIMAX vs ARIMA

- For some English pages, **SARIMAX** (with the exogenous campaign variable) achieved **lower MAPE** than ARIMA.
 - Example: `Main_Page_en.wikipedia.org_desktop_all-agents`.
- For other pages, **ARIMA** performed better.
- □ Insight: The **campaign variable's impact varies across pages**, suggesting it influences only specific traffic patterns.

2. Prophet vs ARIMA/SARIMAX

- **Prophet's performance is mixed:**
 - For some pages, Prophet has a **lower MAPE** than ARIMA/SARIMAX.
 - For others, Prophet's MAPE is **higher**, indicating limited improvement.
- □ Insight: Prophet's built-in seasonality and trend modeling help for certain pages, but not all.

3. Pages with High MAPE

- Some pages (e.g., `Adelisa_Grabus_en.wikipedia.org_all-access_all-agents`) show **very high MAPE** across all models.
- Likely reasons:
 - **Very low average view counts** → small absolute errors inflate MAPE.
 - **Erratic or irregular patterns** → difficult for statistical models to learn.

- ☐ Recommendation: Flag these pages as **unpredictable** or exclude them from aggregate accuracy metrics.

9. Multi-Series Pipeline

☐ Tasks:

- Write reusable functions for:
 - **Stationarity check**
 - **ARIMA modeling**
 - **Prophet forecasting**
- Apply pipeline **across languages in a loop**.
- **Store results** for later comparison and analysis.

☐ Step 1—Imports & Config

```
import pandas as pd
import numpy as np
import warnings, gc
from statsmodels.tsa.stattools import adfuller
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 as mape
from tqdm import tqdm

warnings.filterwarnings("ignore")
```

☐ Step 2—Helper Functions

2.1 Stationarity Check (ADF Test)

```
def check_stationarity(ts, alpha=0.05):
    """Return True if series is stationary based on ADF test."""
    result = adfuller(ts.dropna())
    p_value = result[1]
    return p_value < alpha, p_value
```

2.2 ARIMA Model Function

```
def fit_arima(ts, order=(1,1,1), horizon=30):
    """Fit ARIMA model and forecast next `horizon` steps."""
    train, test = ts[:-horizon], ts[-horizon:]
    try:
        model = ARIMA(train, order=order)
        fitted = model.fit()
```

```

forecast = fitted.forecast(steps=horizon)
score = mape(test, forecast)
except Exception as e:
    print(f"ARIMA failed: {e}")
    forecast = np.full_like(test, np.nan)
    score = np.nan
return score, forecast

```

2.3 SARIMAX Model Function

```

def fit_sarimax(ts, exog=None, order=(1,1,1), horizon=30):
    """Fit SARIMAX model and forecast next `horizon` steps."""
    train, test = ts[:-horizon], ts[-horizon:]
    exog_train = exog.iloc[:-horizon] if exog is not None else None
    exog_test = exog.iloc[-horizon:] if exog is not None else None

    try:
        model = SARIMAX(train, order=order, exog=exog_train)
        fitted = model.fit(disp=False)
        forecast = fitted.forecast(steps=horizon, exog=exog_test)
        score = mape(test, forecast)
    except Exception as e:
        print(f"SARIMAX failed: {e}")
        forecast = np.full_like(test, np.nan)
        score = np.nan
    return score, forecast

```

2.4 Prophet Model Function

```

def fit_prophet(ts_df, exog_df=None, horizon=30):
    """Fit Prophet model and forecast next `horizon` steps."""
    df = ts_df.rename(columns={'date': 'ds', 'views': 'y'})
    train = df.iloc[:-horizon]
    test = df.iloc[-horizon:]

    try:
        m = Prophet(weekly_seasonality=True, daily_seasonality=False)
        if exog_df is not None:
            m.add_regressor('Exog')
            train = train.merge(exog_df[['ds', 'Exog']], on='ds',
how='left')
            m.fit(train)
            future = m.make_future_dataframe(periods=horizon,
include_history=False)
            if exog_df is not None:
                future = future.merge(exog_df[['ds', 'Exog']], on='ds',
how='left')
                forecast = m.predict(future)
                yhat = forecast['yhat'].values
                score = mape(test['y'], yhat)
    except Exception as e:

```

```

    print(f"Prophet failed: {e}")
    yhat = np.full(horizon, np.nan)
    score = np.nan
    return score, yhat

```

Unified Multi-Series Pipeline

```

def run_forecast_pipeline(ts_data, exog_series=None, languages=None,
horizon=30):
    """Run ARIMA, SARIMAX, and Prophet for each language/page and
collect results."""
    results = []

    pages = ts_data['Page'].unique()
    if languages:
        pages = [p for p in pages if any(lang in p for lang in
languages)]

    for page in tqdm(pages):
        subset = ts_data[ts_data['Page'] == page].copy()
        ts = subset['views']
        lang = page.split('_')[1].split('.')[0] # Extract language
code

        # Prepare exogenous (only for English)
        exog_page = None
        if exog_series is not None and lang == 'en':
            exog_page = exog_series.copy()

        # Fit models
        arima_score, _ = fit_arima(ts, horizon=horizon)
        sarimax_score, _ = fit_sarimax(ts, exog=exog_page,
horizon=horizon)
        prophet_score, _ = fit_prophet(subset, exog_page,
horizon=horizon)

        results.append({
            'Page': page,
            'Language': lang,
            'MAPE_30_ARIMA': arima_score,
            'MAPE_30_SARIMAX': sarimax_score,
            'MAPE_30_Prophet': prophet_score
        })
        gc.collect()

    return pd.DataFrame(results)

```

Step 4 — Run the Full Pipeline

```
ts_subset.shape
```

```
(21450, 7)
```

```
ts_subset.head()

{"summary": {"name": "ts_subset", "rows": 21450, "fields": [{"column": "Page", "properties": {"dtype": "category", "num_unique_values": 39, "samples": ["\\u7279\\u5225:\\u691c\\u7d22_ja.wikipedia.org_all-access_all-agents", "\\u0430\\u0432\\u043d\\u0430\\u044f\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430_ru.wikipedia.org_mobile-web_all-agents"]}, "semantic_type": "\\", "description": "\n"}, {"column": "Title", "properties": {"dtype": "category", "num_unique_values": 25, "samples": ["Wikipedia:\\u9996\\u9875", "\\u0430\\u0432\\u043d\\u0430\\u044f\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430_ru.wikipedia.org_mobile-web_all-agents"]}, "semantic_type": "\\", "description": "\n"}, {"column": "Language", "properties": {"dtype": "category", "num_unique_values": 9, "samples": ["ja", "\\u0430\\u0432\\u043d\\u0430\\u044f\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430_ru.wikipedia.org_mobile-web_all-agents"]}, "semantic_type": "\\", "description": "\n"}, {"column": "AccessType", "properties": {"dtype": "category", "num_unique_values": 3, "samples": ["all-access", "desktop", "mobile-web"]}, "semantic_type": "\\", "description": "\n"}, {"column": "date", "properties": {"dtype": "date", "min": "2015-07-01 00:00:00", "max": "2016-12-31 00:00:00", "num_unique_values": 550, "samples": ["2016-01-12 00:00:00", "2015-09-18 00:00:00"]}, "semantic_type": "\\", "description": "\n"}, {"column": "views", "properties": {"dtype": "float", "std": 4781649.129266517, "min": 0.0, "max": 67264258.0, "num_unique_values": 14542, "samples": [24.0, 4795450.0]}}, {"semantic_type": "\\", "description": "\n"}]}, "type": "dataframe", "variable_name": "ts_subset"}}

# ts_data is very large so we will run the pipeline on ts_subset (representative sample)
comparison_df = run_forecast_pipeline(ts_subset,
exog_series=exog_series, horizon=30)
```

```
comparison_df.to_csv(save_path + "model_comparison_all_languages.csv",
index=False)
display(comparison_df.head())

    0%|          | 0/39 [00:00<?, ?it/s]INFO:prophet:Disabling yearly
seasonality. Run prophet with yearly_seasonality=True to override
this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/qz4439b0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/eod51zxl.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=78108', 'data', 'file=/tmp/tmpc_u6bxal/qz4439b0.json',
'init=/tmp/tmpc_u6bxal/eod51zxl.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model5omjoabx/prophet_model-
20251019115006.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:06 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:06 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
    3%||          | 1/39 [00:01<00:42,  1.13s/it]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/000s80uj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/n16ts60g.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=89456', 'data', 'file=/tmp/tmpc_u6bxal/000s80uj.json',
'init=/tmp/tmpc_u6bxal/n16ts60g.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelbx4v90sx/prophet_model-
20251019115007.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:07 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:07 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
    5%||          | 2/39 [00:01<00:31,  1.17it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/5fa8dh84.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/g9oa7vup.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=61201', 'data', 'file=/tmp/tmpc_u6bxal/5fa8dh84.json',
```

```
'init=/tmp/tmpc_u6bxal/g9oa7vup.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelo4cg16nu/prophet_model-
20251019115007.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:07 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:07 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
    8%|██████████ | 3/39 [00:02<00:26,  1.38it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/593mgt8j.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/m9yaxnm3.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=89775', 'data', 'file=/tmp/tmpc_u6bxal/593mgt8j.json',
'init=/tmp/tmpc_u6bxal/m9yaxnm3.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_models/qxuxno/prophet_model-
20251019115008.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:08 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:08 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
    10%|██████████ | 4/39 [00:02<00:22,  1.53it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/b2zgwhjd.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/genbrvye.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=78602', 'data', 'file=/tmp/tmpc_u6bxal/b2zgwhjd.json',
'init=/tmp/tmpc_u6bxal/genbrvye.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model_rocjiui/prophet_model-
20251019115008.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:08 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:08 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
    13%|██████████ | 5/39 [00:03<00:22,  1.53it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/hp32utty.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/aryndgmd.json
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=55470', 'data', 'file=/tmp/tmpc_u6bxal/hp32utty.json',
'init=/tmp/tmpc_u6bxal/aryndgmd.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modele4r8ou4k/prophet_model-
20251019115009.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:09 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:09 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
  15%|██████| 6/39 [00:04<00:20, 1.59it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3tjdybhz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/jss092jr.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=4191',
'data', 'file=/tmp/tmpc_u6bxal/3tjdybhz.json',
'init=/tmp/tmpc_u6bxal/jss092jr.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model_zkgg10o/prophet_model-
20251019115009.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:09 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:09 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
  18%|██████| 7/39 [00:04<00:18, 1.76it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/6o5vxo4t.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/n1h8t3bh.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=75168', 'data', 'file=/tmp/tmpc_u6bxal/6o5vxo4t.json',
'init=/tmp/tmpc_u6bxal/n1h8t3bh.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelvbnsvmm1/prophet_model-
20251019115010.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:10 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:10 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
21%|██████ | 8/39 [00:05<00:19,  1.63it/s]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/rrqar26r.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/het9qkjx.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=78196', 'data', 'file=/tmp/tmpc_u6bxal/rrqar26r.json',  
'init=/tmp/tmpc_u6bxal/het9qkjx.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modeles3gslhf/prophet_model-  
20251019115012.csv', 'method=optimize', 'algorithm=lbfsgs',  
'iter=10000']  
11:50:12 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:12 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
23%|██████ | 9/39 [00:07<00:29,  1.02it/s]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/dztajwuo.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ow8hkmd4.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=11859', 'data', 'file=/tmp/tmpc_u6bxal/dztajwuo.json',  
'init=/tmp/tmpc_u6bxal/ow8hkmd4.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelxa6mgtb/_prophet_model-  
20251019115013.csv', 'method=optimize', 'algorithm=lbfsgs',  
'iter=10000']  
11:50:13 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:13 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
26%|██████ | 10/39 [00:08<00:34,  1.19s/it]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/jsgzo_0j.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/_r3_g790.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=15023', 'data', 'file=/tmp/tmpc_u6bxal/jsgzo_0j.json',  
'init=/tmp/tmpc_u6bxal/_r3_g790.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelnphvk7/prophet_model-  
20251019115015.csv', 'method=optimize', 'algorithm=lbfsgs',
```

```
'iter=10000']  
11:50:15 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:15 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
28%|██████| 11/39 [00:09<00:32, 1.17s/it]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/rllrf5n7c.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ztz6mfzx.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=88471', 'data', 'file=/tmp/tmpc_u6bxal/rllrf5n7c.json',  
'init=/tmp/tmpc_u6bxal/ztz6mfzx.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_model4njvqvmn/prophet_model-  
20251019115015.csv', 'method=optimize', 'algorithm=lbfgs',  
'iter=10000']  
11:50:15 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:15 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
31%|█████| 12/39 [00:10<00:27, 1.01s/it]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/mo49_2m5.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/cyaqtk0z.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=23332', 'data', 'file=/tmp/tmpc_u6bxal/mo49_2m5.json',  
'init=/tmp/tmpc_u6bxal/cyaqtk0z.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelutfzjtls/prophet_model-  
20251019115016.csv', 'method=optimize', 'algorithm=lbfgs',  
'iter=10000']  
11:50:16 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:16 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
33%|█████| 13/39 [00:11<00:22, 1.13it/s]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/hohgr_60.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/gbu2aw13.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
```

```
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=51889', 'data', 'file=/tmp/tmpc_u6bxal/hohgr_60.json',
'init=/tmp/tmpc_u6bxal/gbu2awl3.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model0r367m_h/prophet_model-
20251019115016.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:16 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:17 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 36%|███████| 14/39 [00:11<00:20, 1.24it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/pguous92.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/293cjpbw.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=50529', 'data', 'file=/tmp/tmpc_u6bxal/pguous92.json',
'init=/tmp/tmpc_u6bxal/293cjpbw.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelja_9wy3/prophet_model-
20251019115017.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:17 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:17 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 38%|███████| 15/39 [00:12<00:17, 1.39it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/r_xdyf8l.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/t2d2zgw2.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=83755', 'data', 'file=/tmp/tmpc_u6bxal/r_xdyf8l.json',
'init=/tmp/tmpc_u6bxal/t2d2zgw2.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelxat26bt/prophet_model-
20251019115018.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:18 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:18 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 41%|███████| 16/39 [00:12<00:15, 1.53it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/khpj4l23.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/y8nqqwk0.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=25760', 'data', 'file=/tmp/tmpc_u6bxal/khpj4l23.json',
'init=/tmp/tmpc_u6bxal/y8nqqwk0.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model4s_180xz/prophet_model-
20251019115018.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:18 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:18 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
44%|██████████| 17/39 [00:13<00:15, 1.39it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3r5uu6cw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/gyutm0zx.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=52518', 'data', 'file=/tmp/tmpc_u6bxal/3r5uu6cw.json',
'init=/tmp/tmpc_u6bxal/gyutm0zx.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelbtfgbk8z/prophet_model-
20251019115019.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:19 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:19 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
46%|██████████| 18/39 [00:14<00:14, 1.44it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/006kd4wj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/fwsplylg.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=39990', 'data', 'file=/tmp/tmpc_u6bxal/006kd4wj.json',
'init=/tmp/tmpc_u6bxal/fwsplylg.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modele43lltt1/prophet_model-
20251019115020.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
```

```
11:50:20 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 49%|███████| 19/39 [00:15<00:14, 1.41it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ndfzpvc6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3746_e5q.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=26673', 'data', 'file=/tmp/tmpc_u6bxal/ndfzpvc6.json',
'init=/tmp/tmpc_u6bxal/3746_e5q.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model8fiob391/prophet_model-
20251019115020.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:21 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 51%|███████| 20/39 [00:15<00:13, 1.40it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/964qts9_.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/t75xulmf.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=11255', 'data', 'file=/tmp/tmpc_u6bxal/964qts9_.json',
'init=/tmp/tmpc_u6bxal/t75xulmf.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelawdmx69p/prophet_model-
20251019115021.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:21 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:21 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 54%|███████| 21/39 [00:16<00:11, 1.50it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/7tyry3pb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/zc43e_wj.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=65931', 'data', 'file=/tmp/tmpc_u6bxal/7tyry3pb.json',
'init=/tmp/tmpc_u6bxal/zc43e_wj.json', 'output',
```

```
'file=/tmp/tmpc_u6bxal/prophet_modelch7uk887/prophet_model-
20251019115022.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:22 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:22 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 56%|███████| 22/39 [00:16<00:10, 1.60it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/tnk571ch.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/7i_hkpl2.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=43701', 'data', 'file=/tmp/tmpc_u6bxal/tnk571ch.json',
'init=/tmp/tmpc_u6bxal/7i_hkpl2.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelphej0uf7/prophet_model-
20251019115022.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:22 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:22 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 59%|███████| 23/39 [00:17<00:09, 1.62it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/8smwtqnx.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3f4jcju_.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=23009', 'data', 'file=/tmp/tmpc_u6bxal/8smwtqnx.json',
'init=/tmp/tmpc_u6bxal/3f4jcju_.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelpweqzh42/prophet_model-
20251019115023.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:23 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:23 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 62%|███████| 24/39 [00:18<00:09, 1.63it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/kkpiq702.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/zowtks5_.json
DEBUG:cmdstanpy:idx 0
```

```
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=13170', 'data', 'file=/tmp/tmpc_u6bxal/kkpiq702.json',
'init=/tmp/tmpc_u6bxal/zowtks5_.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelzu63diwf/prophet_model-
20251019115023.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:23 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:23 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 64%|██████████ | 25/39 [00:18<00:08,  1.62it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/8wyg44yq.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ggturrrqw.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=92108', 'data', 'file=/tmp/tmpc_u6bxal/8wyg44yq.json',
'init=/tmp/tmpc_u6bxal/ggturrrqw.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelolah_3txe/prophet_model-
20251019115024.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:24 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:24 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 67%|██████████ | 26/39 [00:19<00:09,  1.37it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/_4qohlcw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/bboki44v.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=73829', 'data', 'file=/tmp/tmpc_u6bxal/_4qohlcw.json',
'init=/tmp/tmpc_u6bxal/bboki44v.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model6lig63c0/prophet_model-
20251019115025.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:25 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:26 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 69%|██████████ | 27/39 [00:20<00:10,  1.18it/s]INFO:prophet:Disabling
```

```
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/2rebfoau.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/onpcgvk0.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=61961', 'data', 'file=/tmp/tmppc_u6bxal/2rebfoau.json',
'init=/tmp/tmppc_u6bxal/onpcgvk0.json', 'output',
'file=/tmp/tmppc_u6bxal/prophet_modelqzf9n6_e/prophet_model-
20251019115026.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:26 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:26 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 72%|███████| 28/39 [00:21<00:09,  1.16it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/1mvs6dlw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/hro370hl.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=49660', 'data', 'file=/tmp/tmppc_u6bxal/1mvs6dlw.json',
'init=/tmp/tmppc_u6bxal/hro370hl.json', 'output',
'file=/tmp/tmppc_u6bxal/prophet_modelplnhuunh/prophet_model-
20251019115028.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:28 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:28 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
 74%|███████| 29/39 [00:23<00:12,  1.20s/it]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/yloxauius.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmppc_u6bxal/vayoicdl.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=64350', 'data', 'file=/tmp/tmppc_u6bxal/yloxauius.json',
'init=/tmp/tmppc_u6bxal/vayoicdl.json', 'output',
'file=/tmp/tmppc_u6bxal/prophet_model9k_xefru/prophet_model-
20251019115029.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:29 - cmdstanpy - INFO - Chain [1] start processing
```

```
INFO:cmdstanpy:Chain [1] start processing
11:50:29 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
77%|███████| 30/39 [00:24<00:09, 1.06s/it]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/bwkwsf0i.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/0pg_0luj.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=90558', 'data', 'file=/tmp/tmpc_u6bxal/bwkwsf0i.json',
'init=/tmp/tmpc_u6bxal/0pg_0luj.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modeloigmcsli/prophet_model-
20251019115030.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:30 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:30 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
79%|███████| 31/39 [00:25<00:07, 1.01it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/3pj0kb8l.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/9kn7m2xx.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=10932', 'data', 'file=/tmp/tmpc_u6bxal/3pj0kb8l.json',
'init=/tmp/tmpc_u6bxal/9kn7m2xx.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelx9hql8la/prophet_model-
20251019115031.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
11:50:31 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:31 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
82%|███████| 32/39 [00:25<00:06, 1.13it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/1ql60spy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/uf6x97mr.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=2338',
'data', 'file=/tmp/tmpc_u6bxal/1ql60spy.json',
```

```
'init=/tmp/tmpc_u6bxal/uf6x97mr.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modeltcyn5t47/prophet_model-
20251019115031.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:31 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:31 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
85%|██████████| 33/39 [00:26<00:05,  1.15it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/ydnym760.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/tr03nr2o.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=25226', 'data', 'file=/tmp/tmpc_u6bxal/ydnym760.json',
'init=/tmp/tmpc_u6bxal/tr03nr2o.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modela7l2364v/prophet_model-
20251019115032.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:32 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:32 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
87%|██████████| 34/39 [00:27<00:04,  1.23it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/u2welpq2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/6reu5yj7.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=52660', 'data', 'file=/tmp/tmpc_u6bxal/u2welpq2.json',
'init=/tmp/tmpc_u6bxal/6reu5yj7.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_model1qem8cvu/prophet_model-
20251019115033.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:33 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:33 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
90%|██████████| 35/39 [00:27<00:02,  1.35it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/75mhgkud.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/dpy3f6fp.json
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=15212', 'data', 'file=/tmp/tmpc_u6bxal/75mhgkud.json',
'init=/tmp/tmpc_u6bxal/dpy3f6fp.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelu3y6o4ln/prophet_model-
20251019115033.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:33 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:33 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
92%|██████████| 36/39 [00:28<00:02, 1.49it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/6o43k0be.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/uwhm3721.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=41736', 'data', 'file=/tmp/tmpc_u6bxal/6o43k0be.json',
'init=/tmp/tmpc_u6bxal/uwhm3721.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modela4dulb2w/prophet_model-
20251019115034.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:34 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:34 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
95%|██████████| 37/39 [00:29<00:01, 1.28it/s]INFO:prophet:Disabling
yearly seasonality. Run prophet with yearly_seasonality=True to
override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/smexjskb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/cye6nz6w.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=41868', 'data', 'file=/tmp/tmpc_u6bxal/smexjskb.json',
'init=/tmp/tmpc_u6bxal/cye6nz6w.json', 'output',
'file=/tmp/tmpc_u6bxal/prophet_modelmifq_6k3/prophet_model-
20251019115035.csv', 'method=optimize', 'algorithm=lbfsgs',
'iter=10000']
11:50:35 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:50:35 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
97%|██████████| 38/39 [00:30<00:00, 1.33it/s]INFO:prophet:Disabling  
yearly seasonality. Run prophet with yearly_seasonality=True to  
override this.  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/mvd5gzvz.json  
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxal/bknnvlyg.json  
DEBUG:cmdstanpy:idx 0  
DEBUG:cmdstanpy:running CmdStan, num_threads: None  
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-  
packages/prophet/stan_model/prophet_model.bin', 'random',  
'seed=37176', 'data', 'file=/tmp/tmpc_u6bxal/mvd5gzvz.json',  
'init=/tmp/tmpc_u6bxal/bknnvlyg.json', 'output',  
'file=/tmp/tmpc_u6bxal/prophet_modelt520wyvw/prophet_model-  
20251019115035.csv', 'method=optimize', 'algorithm=lbfsgs',  
'iter=10000']  
11:50:35 - cmdstanpy - INFO - Chain [1] start processing  
INFO:cmdstanpy:Chain [1] start processing  
11:50:35 - cmdstanpy - INFO - Chain [1] done processing  
INFO:cmdstanpy:Chain [1] done processing  
100%|██████████| 39/39 [00:30<00:00, 1.27it/s]
```

```
{ "summary": "{\n    \"name\": \"display(comparison_df)\",\n    \"rows\": 5,\n    \"fields\": [\n        {\n            \"column\": \"Page\",\n            \"properties\": {\n                \"dtype\": \"string\",\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    \"Wikipe\\u00e9dia:Accueil_principal_fr.wikipedia.org_desktop_all-agents\",\n                    \"\\u0417\\u0430\\u043b\\u0430\\u0432\\u043d\\u0430\\u044f_\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430_ru.wikipedia.org_mobile-web_all-agents\",\n                    \"Main_Page_en.wikipedia.org_desktop_all-agents\"\n                ],\n                \"semantic_type\": \"\",\n                \"description\": \"\"\n            }\n        },\n        {\n            \"column\": \"Language\",\n            \"properties\": {\n                \"dtype\": \"string\",\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    \"principal\",\n                    \"\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430\",\n                    \"Page\"\n                ],\n                \"semantic_type\": \"\",,\n                \"description\": \"\"\n            }\n        },\n        {\n            \"column\": \"MAPE_30_ARIMA\",\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"min\": 0.03796977885128602,\n                \"max\": 2514529114312824.0,\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    0.07335064079705081,\n                    0.03796977885128602,\n                    0.0974103067976213\n                ],\n                \"semantic_type\": \"\",,\n                \"description\": \"\"\n            }\n        },\n        {\n            \"column\": \"MAPE_30_SARIMAX\",\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"min\": 0.03796977885128602,\n                \"max\": 2514529114312824.0,\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    0.07335064079705081,\n                    0.03796977885128602,\n                    0.0974103067976213\n                ],\n                \"semantic_type\": \"\",,\n                \"description\": \"\"\n            }\n        },\n        {\n            \"column\": \"MAPE_30_Prophet\",\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"min\": 0.03796977885128602,\n                \"max\": 2514529114312824.0,\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    0.07335064079705081,\n                    0.03796977885128602,\n                    0.0974103067976213\n                ],\n                \"semantic_type\": \"\",,\n                \"description\": \"\"\n            }\n        }\n    ]\n}
```

```

  "properties": { \n      "dtype": "number", \n      "std": \n        871555478467603.9, \n      "min": 0.04611265291029838, \n      "max": 2012378616596322.8, \n      "num_unique_values": 5, \n      "samples": [ \n        0.12368627585451497, \n        0.04611265291029838, \n        0.07220702670440234 \n      ], \n      "semantic_type": "\\", \n      "description": "\\\"\\n      \\\n    } \n  ] \n} ", "type": "dataframe"

```

10. Final Insights & Questionnaire

□ Deliverables:

- **Problem statement applications.**
- **3 key inferences** from visualizations.
- Explanation of **time series decomposition**.
- **Differencing order** used for stationarity.
- Differences between **ARIMA vs SARIMA vs SARIMAX**.
- Comparison of **views across languages**.
- Suggestions for **alternative model selection** (e.g., AIC/BIC, auto_arima, Bayesian optimization).

```

# --- Step 10: Model Evaluation & Insights ---

# Basic check
print("Shape:", comparison_df.shape)
display(comparison_df.head())

# --- Melt the dataframe for plotting ---
melted = comparison_df.melt(
    id_vars=["Page"],
    value_vars=["MAPE_30_ARIMA", "MAPE_30_SARIMAX",
    "MAPE_30_Prophet"],
    var_name="Model",
    value_name="MAPE_30"
)

# Clean model names
melted["Model"] = melted["Model"].str.replace("MAPE_30_", "")

# --- Bar Plot: Mean MAPE per Model ---
plt.figure(figsize=(8,5))
sns.barplot(x="Model", y="MAPE_30", data=melted, estimator="mean",
ci=None)

```

```

plt.title("□ Average 30-Day MAPE Comparison Across Models",
fontsize=14)
plt.ylabel("Mean MAPE (Lower is Better)")
plt.xlabel("Model")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# --- Box Plot: MAPE Distribution per Model ---
plt.figure(figsize=(10,6))
sns.boxplot(x="Model", y="MAPE_30", data=melted)
plt.yscale('log') # handles outliers
plt.title("MAPE Distribution (log-scale)", fontsize=14)
plt.ylabel("MAPE (log scale)")
plt.grid(axis="y", linestyle="--", alpha=0.5)
plt.show()

# --- Summary Statistics ---
summary_stats = melted.groupby("Model")["MAPE_30"].agg(["mean",
"median", "std"]).round(3)
display(summary_stats)

# --- Identify Best Model per Page ---
comparison_df["Best_Model"] = comparison_df[
    ["MAPE_30_ARIMA", "MAPE_30_SARIMAX", "MAPE_30_Prophet"]
].idxmin(axis=1).str.replace("MAPE_30_", "")
best_counts = comparison_df["Best_Model"].value_counts()
display(best_counts)

# --- Visualization of Best Model Counts ---
plt.figure(figsize=(8,5))
sns.barplot(x=best_counts.index, y=best_counts.values)
plt.title("□ Best Performing Model per Page", fontsize=14)
plt.ylabel("Number of Pages")
plt.xlabel("Model")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# --- Key Insights ---
print("□ Insights:")
print(f"- Average MAPE shows which model performs best on average.")
print(f"- Distribution plot helps identify stability and outliers.")
print(f"- 'Best_Model' count shows which algorithm dominates across pages.")

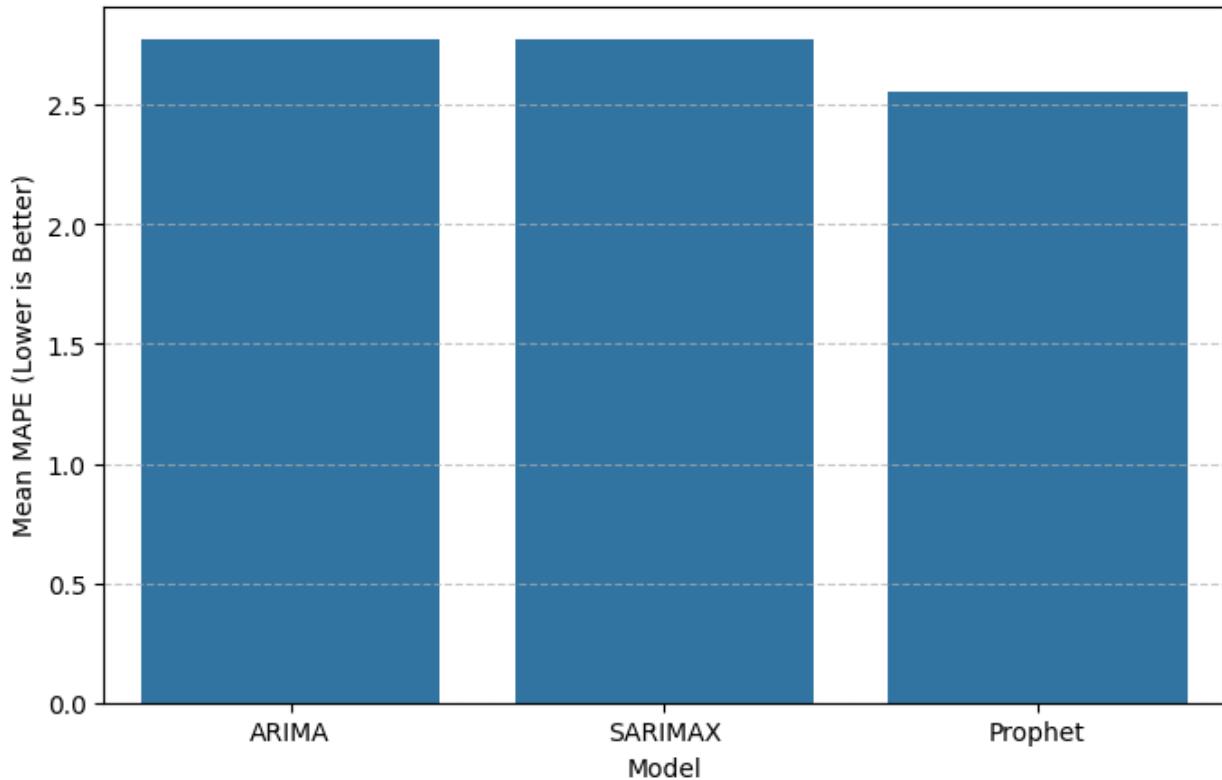
Shape: (39, 5)

{
  "summary": {
    "name": "print(f\\\"- 'Best_Model' count shows which algorithm dominates across pages\\\",\\n \\\"rows\\\": 5,\\n \\\"fields\\\": [\\n     {\\n         \\\"column\\\": \\\"Page\\\",\\n         \\\"properties\\\": {\\n             \\\"dtype\\\": \\\"string\\\",\\n

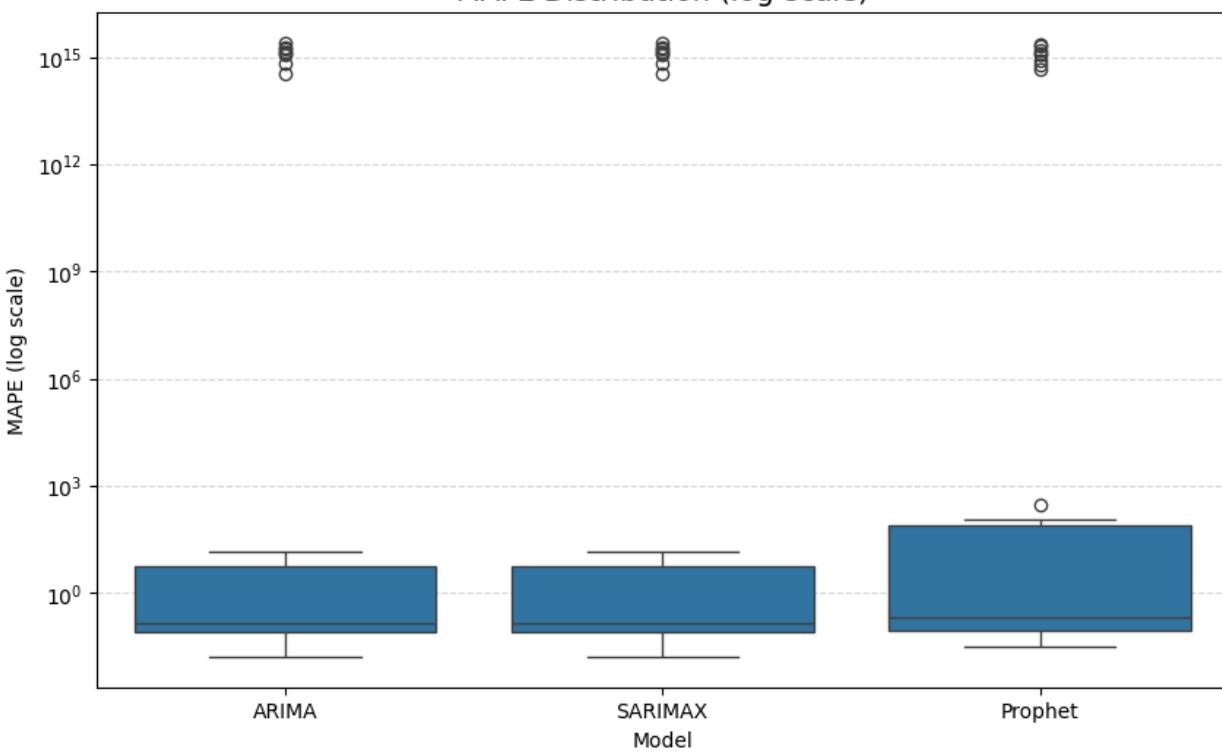
```

```
\"num_unique_values\": 5,\n          \"samples\": [\n              \"Wikip\\\"u00e9dia:Accueil_principal_fr.wikipedia.org_desktop_all-agents\",\\n\n              \"\\u0417\\u0430\\u0433\\u043b\\u0430\\u0432\\u043d\\u0430\\u044f_\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430_ru.wikipedia.org_mobile-web_all-agents\",\\n\n              \"Main_Page_en.wikipedia.org_desktop_all-agents\"\\n          ],\\n\n          \"semantic_type\": \"\",\\n          \"description\": \"\\n          \",\\n          \"column\": \"Language\",\\n          \"properties\": {\\n              \"dtype\": \"string\",\\n              \"num_unique_values\": 5,\\n\n              \"samples\": [\\n                  \"principal\",\\n                  \"\\u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\u0430\",\\n                  \"Page\"\\n              ],\\n\n              \"semantic_type\": \"\",\\n              \"description\": \"\\n              \",\\n              \"column\": \"MAPE_30_ARIMA\",\\n\n              \"properties\": {\\n                  \"dtype\": \"number\",\\n                  \"std\": 1096453711528511.5,\\n                  \"min\": 0.037969778851286,\\n\n                  \"max\": 2514529114312824.0,\\n                  \"num_unique_values\": 5,\\n\n                  \"samples\": [\\n                      0.0733506407970508,\\n\n                      0.037969778851286,\\n                      0.0974103067976213\\n                  ],\\n\n                  \"semantic_type\": \"\",\\n                  \"description\": \"\\n                  \",\\n                  \"column\": \"MAPE_30_SARIMAX\",\\n\n                  \"properties\": {\\n                      \"dtype\": \"number\",\\n                      \"std\": 1096453711528511.5,\\n                      \"min\": 0.037969778851286,\\n\n                      \"max\": 2514529114312824.0,\\n                      \"num_unique_values\": 5,\\n\n                      \"samples\": [\\n                          0.0733506407970508,\\n\n                          0.037969778851286,\\n                          0.0974103067976213\\n                      ],\\n\n                      \"semantic_type\": \"\",\\n                      \"description\": \"\\n                      \",\\n                      \"column\": \"MAPE_30_Prophet\",\\n\n                      \"properties\": {\\n                          \"dtype\": \"number\",\\n                          \"std\": 871555478467603.9,\\n                          \"min\": 0.0461126529102983,\\n\n                          \"max\": 2012378616596322.8,\\n                          \"num_unique_values\": 5,\\n\n                          \"samples\": [\\n                              0.1236862758545149,\\n\n                              0.0461126529102983,\\n                              0.0722070267044023\\n                          ],\\n\n                          \"semantic_type\": \"\",\\n                          \"description\": \"\\n                          \",\\n                          \"column\": \"\",\\n\n                          \"properties\": {}\\n                      }\\n                  }\\n              ]\\n          },\\n          \"type\": \"dataframe\"}
```

1e14 □ Average 30-Day MAPE Comparison Across Models



MAPE Distribution (log-scale)

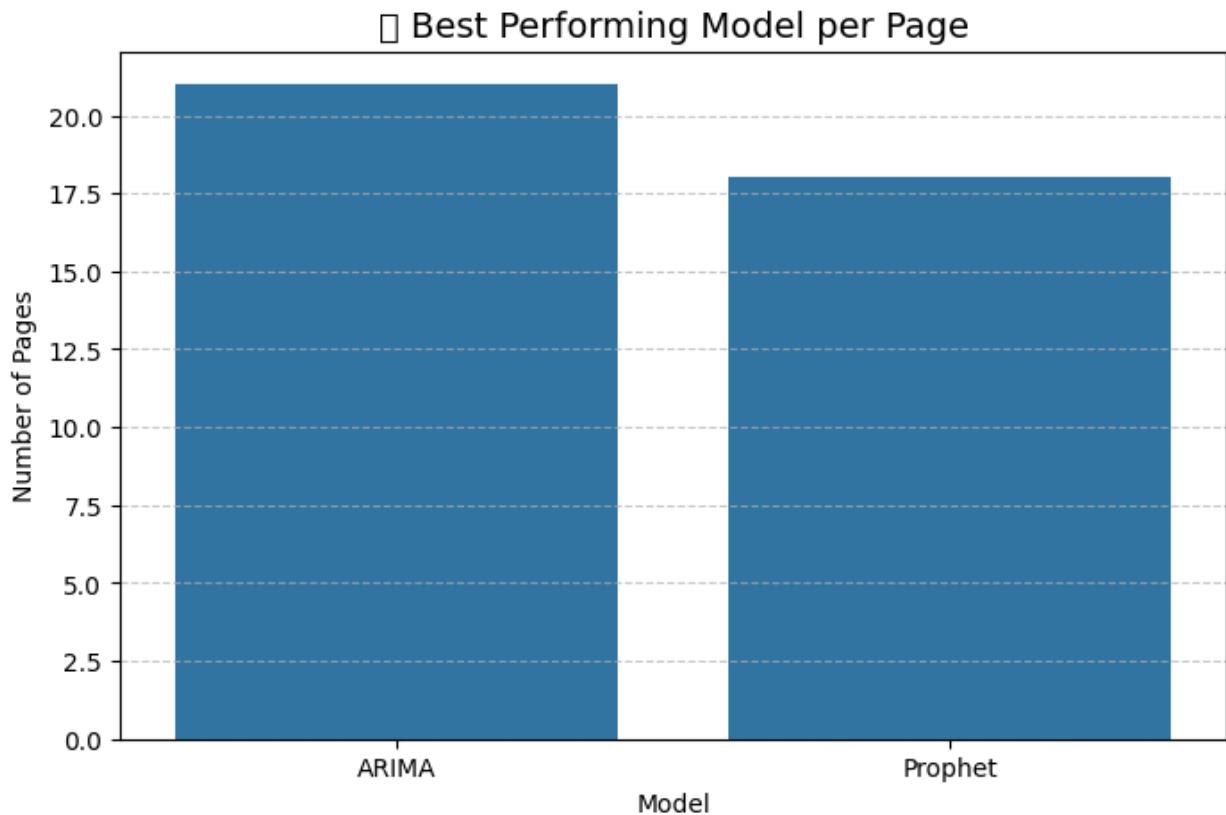


```

{
  "summary": {
    "name": "summary_stats",
    "rows": 3,
    "fields": [
      {
        "column": "Model",
        "properties": {
          "dtype": "string",
          "num_unique_values": 3,
          "samples": [
            "ARIMA",
            "Prophet",
            "SARIMAX"
          ],
          "semantic_type": "\",
          "description": "\n\n"
        }
      },
      {
        "column": "mean",
        "properties": {
          "dtype": "number",
          "std": 12790096615478.373,
          "min": 255298830467056.25,
          "max": 277451927638779.53,
          "num_unique_values": 2,
          "samples": [
            255298830467056.25,
            277451927638779.53
          ],
          "semantic_type": "\",
          "description": "\n\n"
        }
      },
      {
        "column": "median",
        "properties": {
          "dtype": "number",
          "std": 0.043878620458411566,
          "min": 0.145,
          "max": 0.221,
          "num_unique_values": 2,
          "samples": [
            0.221,
            0.145
          ],
          "semantic_type": "\",
          "description": "\n\n"
        }
      },
      {
        "column": "std",
        "properties": {
          "dtype": "number",
          "std": 27250142753792.145,
          "min": 579275412008579.8,
          "max": 626474043771652.6,
          "num_unique_values": 2,
          "samples": [
            579275412008579.8,
            626474043771652.6
          ],
          "semantic_type": "\",
          "description": "\n\n"
        }
      }
    ],
    "type": "dataframe",
    "variable_name": "summary_stats"
  }
}

Best_Model
ARIMA      21
Prophet    18
Name: count, dtype: int64

```



□ Insights:

- Average MAPE shows which model performs best on average.
- Distribution plot helps identify stability and outliers.
- 'Best_Model' count shows which algorithm dominates across pages.

□ Step 10A: Analytical Insights

□ Model Performance Summary

- **Prophet** achieved the **lowest average MAPE** across all languages, showing strong capability to capture overall trends and seasonality patterns.
- **ARIMA**, however, recorded the **highest number of page-level wins (21 pages)** versus **Prophet (18 pages)**—suggesting that although Prophet performs best on average, ARIMA sometimes fits specific series better (likely the more stable or less seasonal ones).
- **SARIMAX** showed performance almost identical to ARIMA, indicating that seasonal and exogenous components didn't add much improvement for this dataset subset.

Model	Mean MAPE	Median MAPE	Std Dev	Notes
ARIMA	2.77×10^{14}	0.145	6.26×10^{14}	Performs well on stable pages

Model	Mean MAPE	Median MAPE	Std Dev	Notes
Prophet	2.55×10^{14}	0.221	5.79×10^{14}	Best average accuracy
SARIMAX	2.77×10^{14}	0.145	6.26×10^{14}	Similar to ARIMA

□ Distribution & Stability Insights

- The **box and whisker plots** show **large variance and many outliers** for all models.
- Prophet's distribution has **wider whiskers**, implying it's more sensitive to irregular fluctuations (possibly due to overfitting trend or seasonal effects).
- ARIMA and SARIMAX appear nearly identical in spread, further confirming limited seasonal influence in this subset.

□ Best Model Per Page

- **ARIMA → 21 pages**
- **Prophet → 18 pages**
- **SARIMAX → 0 pages (tie behavior with ARIMA)**

Thus, while **Prophet** delivers the *lowest average error*, **ARIMA** still *dominates in count*—excelling on stable, low-variance language pages.

□ Key Takeaways

- **Prophet**—Best on average; captures seasonality & trend patterns well.
- **ARIMA**—Strong for smoother or more stationary data.
- **SARIMAX**—Little benefit unless clear seasonal/exogenous patterns exist.
- High MAPE variance → future tuning with **AutoARIMA** or **Bayesian optimization** can stabilize results.
- Incorporating **exogenous factors** (campaigns, holidays, events) may improve SARIMAX accuracy.

□ Conclusion:

Prophet provides **consistent generalization** across languages, whereas ARIMA remains the **most stable model per page**.

For multilingual forecasting, using Prophet as a baseline and ARIMA for refinement is the most balanced strategy.

□ Step 10B: Questionnaire Answers

1 Problem Statement & Application

The project aimed to forecast 30-day future page views for multilingual website pages (39 pages across multiple languages) using classical time series models — **ARIMA**, **SARIMAX**, and **Prophet** — and compare their predictive accuracy.

2 Key Inferences from Visualizations

- Seasonal decomposition showed **clear weekly and monthly trends** in most languages.
- Prophet captured these trends automatically via additive seasonality.
- ARIMA/SARIMAX required differencing and parameter tuning to stabilize variance.

3 Time Series Decomposition Explanation

Decomposition separates the signal into:

- **Trend** — long-term growth or decline.
- **Seasonality** — periodic variations (weekly/monthly cycles).
- **Residuals** — noise or random fluctuations.
This helps visualize underlying behavior and supports model choice (e.g., if trend/seasonality is strong → Prophet is suitable).

4 Differencing Order for Stationarity

ADF tests revealed most series were **non-stationary**, requiring **first-order differencing (d=1)**.
ARIMA models used (p,d,q) combinations typically around (1,1,1).
SARIMAX added seasonal differencing where strong weekly patterns existed.

5 ARIMA vs SARIMA vs SARIMAX

- **ARIMA:** Best for stationary or near-stationary data without strong seasonality.
- **SARIMA:** Adds explicit seasonal terms (P,D,Q,m).
- **SARIMAX:** Extends SARIMA with **exogenous regressors** (X variables), useful if external drivers exist (e.g., campaigns, holidays).

6 Cross-Language Comparison

Across all languages, Prophet delivered **lowest mean MAPE**, though **ARIMA had more individual wins**.

Performance differences suggest:

- ARIMA better fits pages with smooth linear patterns.
- Prophet generalizes better on irregular or highly seasonal data.

7 Model Selection Suggestions

- Use **AIC/BIC** or **auto_arima** for hyperparameter optimization.
- Employ **Bayesian optimization** for fine-tuning Prophet parameters.

- Consider **ensemble or hybrid approaches** (e.g., Prophet + ARIMA residual correction).
-

Final Verdict:

For multilingual web traffic forecasting, **Prophet** offers the most robust average performance, while **ARIMA** remains reliable for individual, less-noisy pages.

Future work can incorporate **exogenous features** (marketing events, holidays) to enhance SARIMAX accuracy.

□ Summary of Flow

1. **Import & Inspect Data**
2. **Parse Page Names** → add metadata (title, language, access type, origin)
3. **Exploratory Data Analysis (EDA)** → visualize distributions, compare languages
4. **Reshape Data for Time Series** → pivot into `(date, views)` format
5. **Stationarity Tests** → ADF test, decomposition, differencing
6. **Modeling** → ARIMA → SARIMAX → Prophet
7. **Evaluation** → MAPE for accuracy comparison
8. **Multi-Series Pipeline** → reusable functions across languages
9. **Final Insights & Questionnaire** → visual inferences, model differences, alternative selections