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 warnings
warnings.filterwarnings("ignore")
```

2. Data Import, Initial Exploration & handle null values

Tasks:

- Load `train_1.csv` and `Exog_Campaign_eng`.
- Check size, columns, and missing values.
- Verify date columns (should start around 2015–2017).

- Handle **null values** (likely 0 views → fill with 0).

```
train = pd.read_csv("/content/drive/MyDrive/Wikipedia/train_1.csv")

train.head()

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

print("shape:", train.shape)
print("total missing values:", train.isnull().sum().sum())

shape: (145063, 551)
total missing values: 6192931
```

Missingness diagnostics & baseline cleaning choice

```
train['Page'].isnull().sum()

np.int64(0)
```

- no missing value in Page column only missing value in date columns. We will handle them next.

```
# detect date columns robustly
date_cols = [c for c in train.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

# recompute missing counts if not present
train['missing_count'] = train[date_cols].isnull().sum(axis=1)
train['missing_pct'] = train['missing_count'] / len(date_cols) * 100

print("Total missing values (cells):",
train[date_cols].isnull().sum().sum())
print("Pages with all dates missing:", (train['missing_count'] ==
len(date_cols)).sum())
print("Pages with >50% missing:", (train['missing_pct'] > 50).sum())
print("Pages with >20% missing:", (train['missing_pct'] > 20).sum())

Total missing values (cells): 6192931
Pages with all dates missing: 652
Pages with >50% missing: 10484
Pages with >20% missing: 18536

# # Recommended baseline for EDA & modeling: drop rows that are ALL-
NaN, then fill remaining NaNs with 0
# train_clean = train.loc[train['missing_count'] !=
```

```
len(date_cols)].copy()
# train_clean[date_cols] = train_clean[date_cols].fillna(0)

# print("train_clean shape:", train_clean.shape)
# print("total missing after baseline fill:",
train_clean[date_cols].isnull().sum().sum())
```

```
train.sample(10)
```

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

```
# Identify pages with NaNs at the beginning
pages_with_leading_nans = train[train[date_cols[0]].isnull()]

# Identify pages with NaNs somewhere in between (not just leading or
trailing)
# This is a bit more complex. We can check if there are NaNs and if
the first and last values are not NaN.
# This is a simplification, as NaNs could be in the middle even if the
first/last are NaN,
# but it helps differentiate from purely leading/trailing NaNs for
visualization purposes.
pages_with_internal_nans = train[
    (train[date_cols].isnull().sum(axis=1) > 0) & # has some missing
values
    (train[date_cols[0]].notnull()) & # but not missing at the very
start
    (train[date_cols[-1]].notnull()) # and not missing at the very end
]

# Identify pages with NaNs at the end
pages_with_trailing_nans = train[train[date_cols[-1]].isnull()]


# Select a few sample pages from each category for visualization
sample_leading_nan_pages = pages_with_leading_nans.sample(min(5,
len(pages_with_leading_nans)), random_state=42)
sample_internal_nan_pages = pages_with_internal_nans.sample(min(5,
len(pages_with_internal_nans)), random_state=42)
sample_trailing_nan_pages = pages_with_trailing_nans.sample(min(5,
len(pages_with_trailing_nans)), random_state=42)


print("Sample pages with leading NaNs:")
display(sample_leading_nan_pages[['Page', 'missing_pct']].head())

print("\nSample pages with internal NaNs:")
display(sample_internal_nan_pages[['Page', 'missing_pct']].head())

print("\nSample pages with trailing NaNs:")
display(sample_trailing_nan_pages[['Page', 'missing_pct']].head())
```

```
# Function to plot a single page's time series
def plot_page_views(page_row, date_cols):
    page_name = page_row['Page']
    views = page_row[date_cols].astype(float)
    plt.figure(figsize=(12, 4))
    plt.plot(pd.to_datetime(date_cols), views)
    plt.title(f"Page Views for: {page_name}")
    plt.xlabel("Date")
    plt.ylabel("Views")
    plt.grid(True)
    plt.show()

# Visualize sample pages with leading NaNs
print("\nVisualizing sample pages with leading NaNs:")
for index, row in sample_leading_nan_pages.iterrows():
    plot_page_views(row, date_cols)

# Visualize sample pages with internal NaNs
print("\nVisualizing sample pages with internal NaNs:")
for index, row in sample_internal_nan_pages.iterrows():
     plot_page_views(row, date_cols)

# Visualize sample pages with trailing NaNs
print("\nVisualizing sample pages with trailing NaNs:")
for index, row in sample_trailing_nan_pages.iterrows():
     plot_page_views(row, date_cols)
```

```
Sample pages with leading NaNs:
```


```
{"summary":"{\n  \"name\": \"     plot_page_views(row, date_cols)\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"W_(\\
u96fb\\u8996\\u5287)_zh.wikipedia.org_all-access_spider\",\n
\"File:Feuerzeichen,_1979,_ein_Film_von_Herbert_Br\\
u00f6dl.jpg_commons.wikimedia.org_all-access_spider\",\n          \"\\
u7279\\u5225:\\u30d5\\u30a3\\u30fc\\u30c9\\u9805\\u76ee/featured/
20161010000000/ja_ja.wikipedia.org_desktop_all-agents\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"missing_pct\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25.95864479647236,\n        \"min\": 36.18181818181818,\n
\"max\": 92.9090909090909,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          40.54545454545455,\n
76.18181818181819,\n          92.9090909090909\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

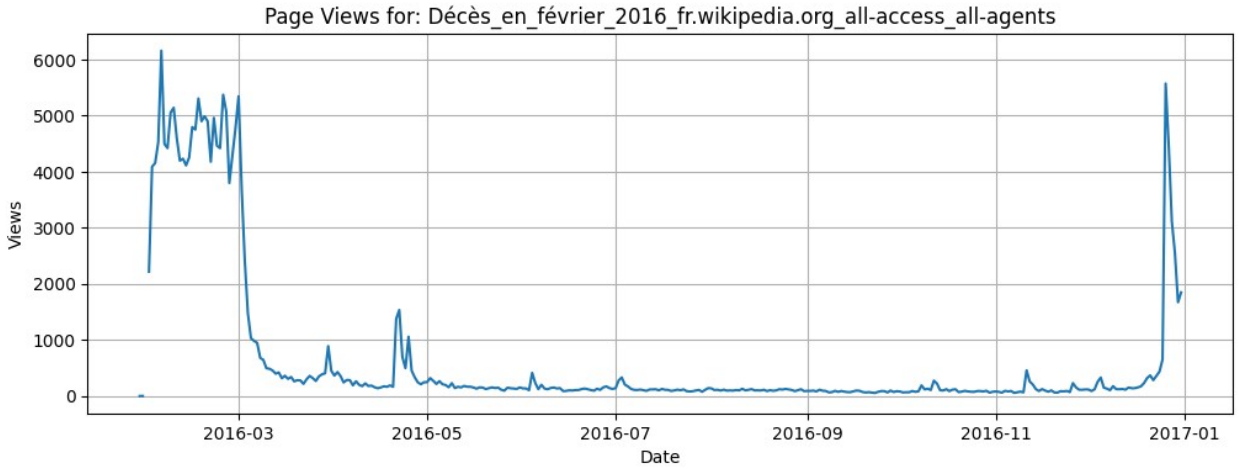
Sample pages with internal NaNs:

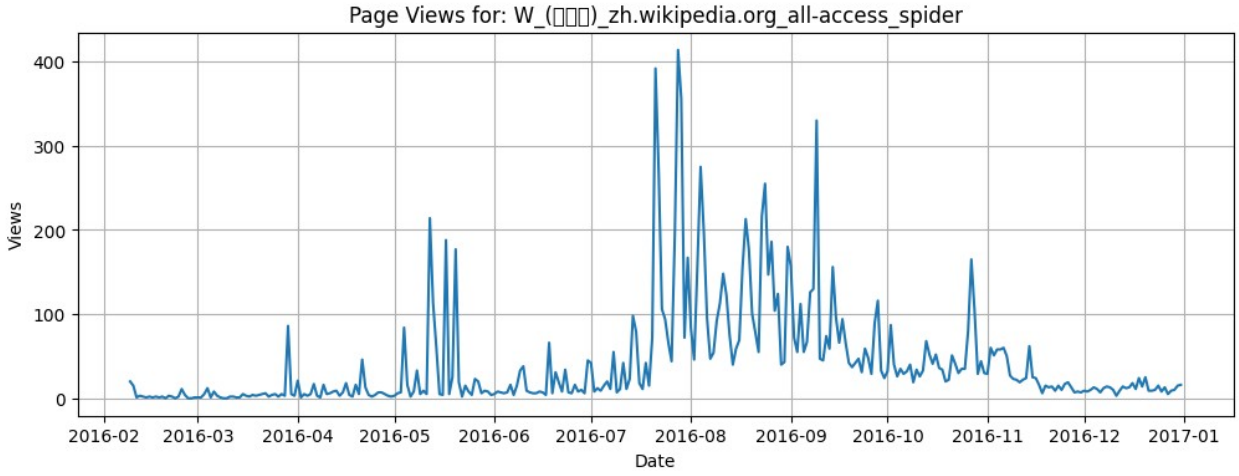

```
{"summary":"{\n  \"name\": \"     plot_page_views(row, date_cols)\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"\\
u540d\\u53e4\\u5c4b\\u897f\\u672c\\u7dda\\u6599\\u91d1\\
u6240_ja.wikipedia.org_desktop_all-agents\",\n
\"Category:Extensions/pt-br_www.mediawiki.org_desktop_all-agents\",\n
\"File:Wembleyold.jpg_commons.wikimedia.org_mobile-web_all-agents\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"missing_pct\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.37056574932882,\n        \"min\": 0.36363636363636365,\n
\"max\": 42.72727272727273,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.36363636363636365,\n
6.7272727272727275,\n          0.9090909090909091\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

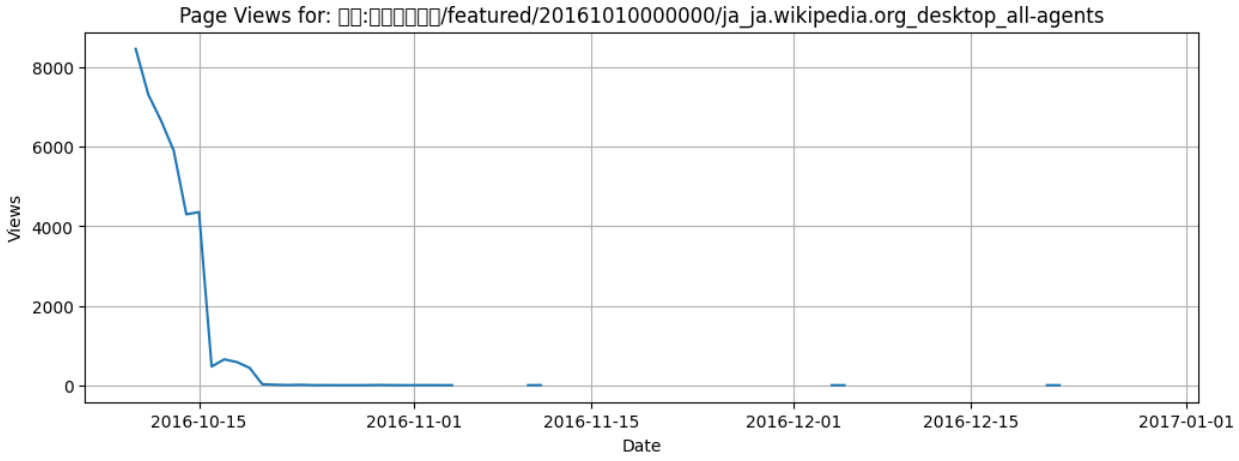

Sample pages with trailing NaNs:

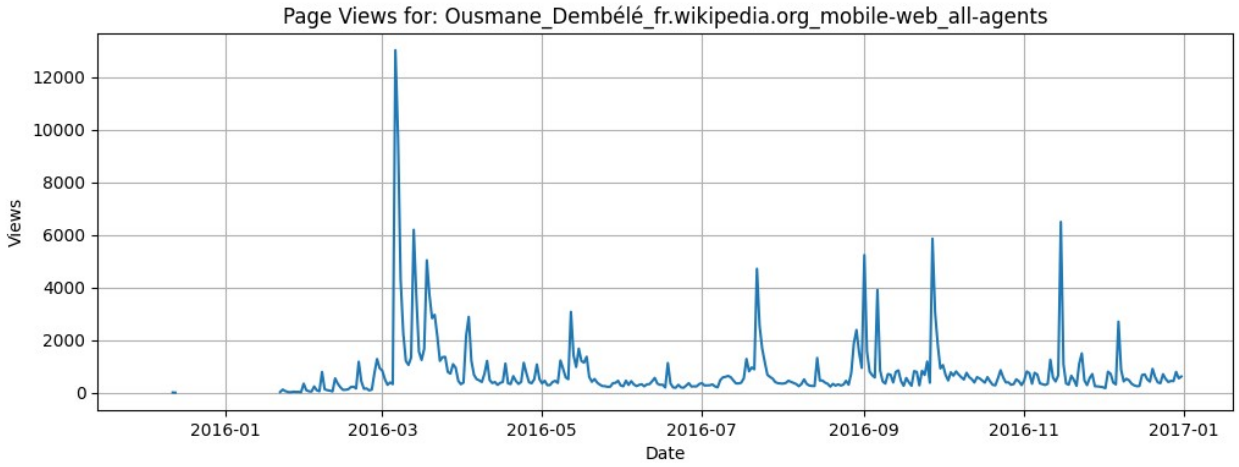

```
{"summary":"{\n  \"name\": \"     plot_page_views(row, date_cols)\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"File:Zeitschrift_des_Vereins_fuer_Volkskunde_25_b_A_006.jpg_commons.
wikimedia.org_all-access_spider\",\n
\"User:CallieGraham09_commons.wikimedia.org_all-access_all-agents\",\n
\"Asceua_en.wikipedia.org_all-access_all-agents\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"missing_pct\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
23.14060507648374,\n        \"min\": 44.36363636363637,\n
\"max\": 100.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          91.45454545454545,\n
88.18181818181819,\n          99.63636363636364\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

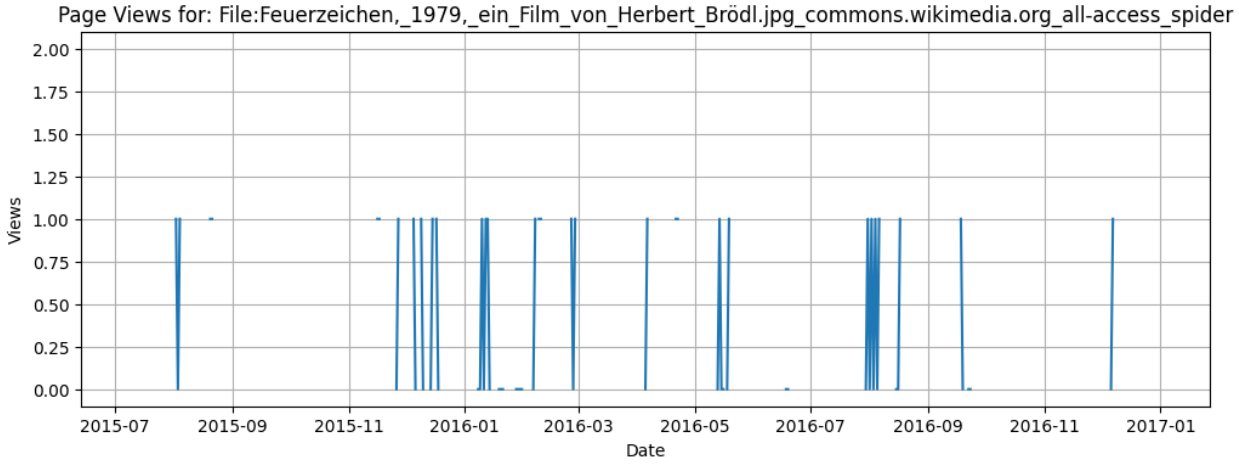

Visualizing sample pages with leading NaNs:


Page Views for: Décès_en_février_2016_fr.wikipedia.org_all-access_all-agents
Views
Date
0
1000
2000
3000
4000
5000
6000
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

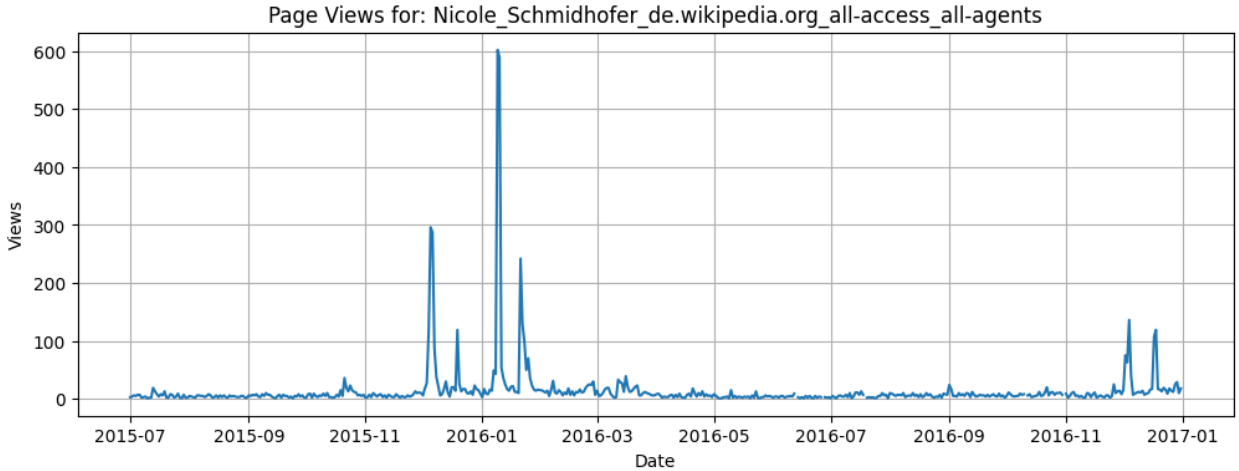

Page Views for: W_(□□□)_zh.wikipedia.org_all-access_spider
Views
Date
0
100
200
300
400
2016-02
2016-03
2016-04
2016-05
2016-06
2016-07
2016-08
2016-09
2016-10
2016-11
2016-12
2017-01

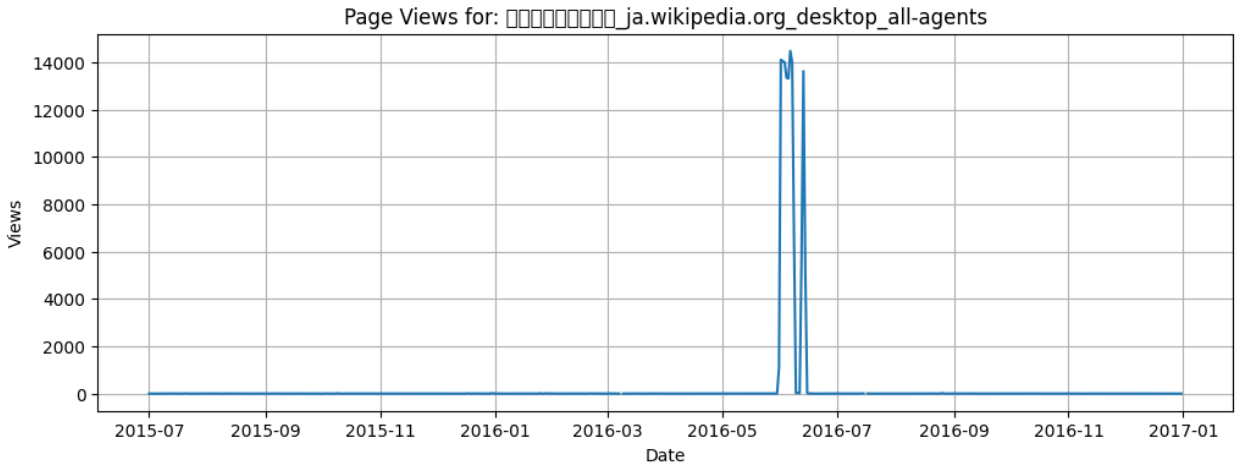

Page Views for: □□:□□□□□□/featured/20161010000000/ja_ja.wikipedia.org_desktop_all-agents
Views
Date
0
2000
4000
6000
8000
2016-10-15
2016-11-01
2016-11-15
2016-12-01
2016-12-15
2017-01-01

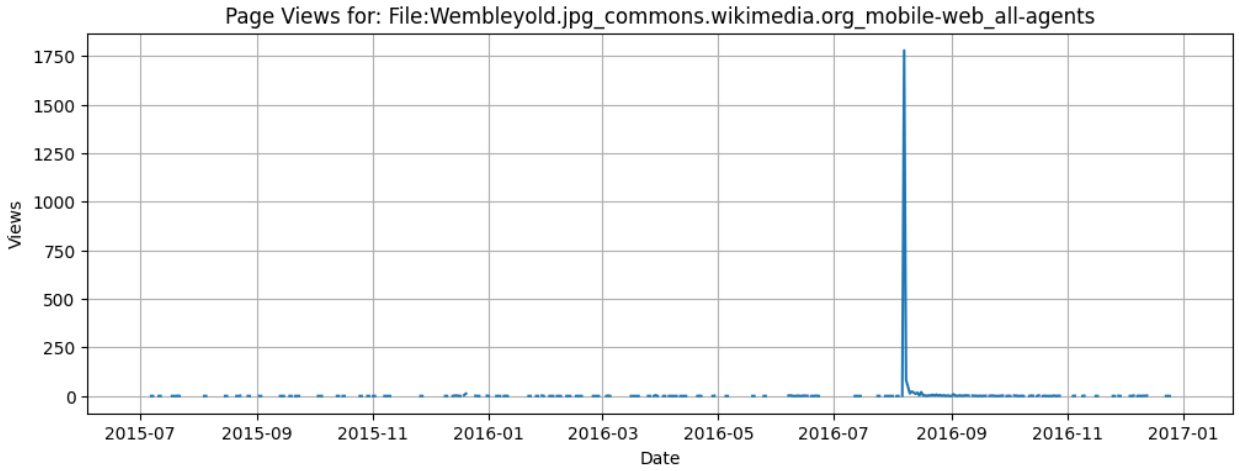


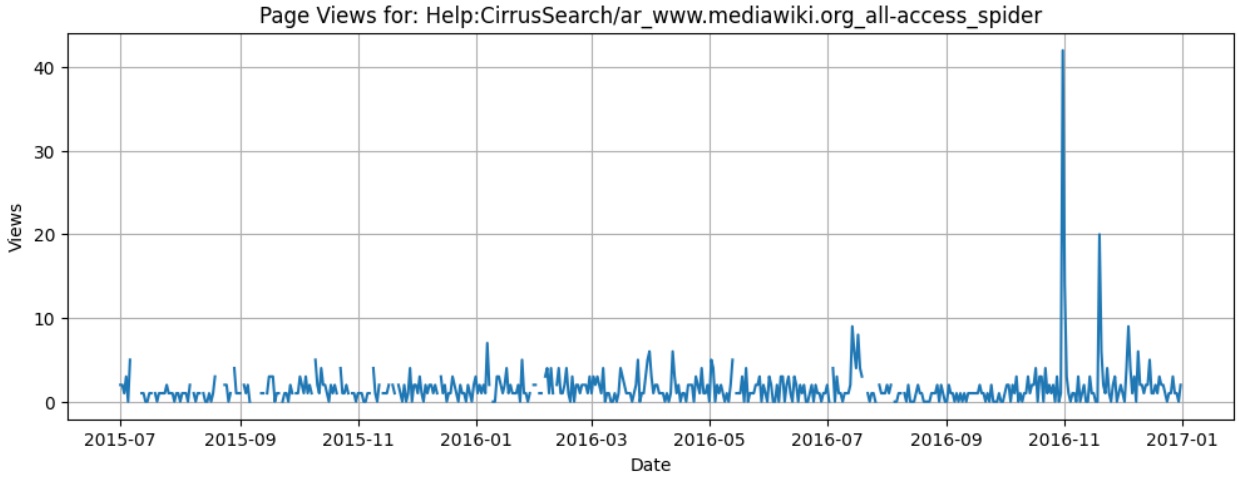


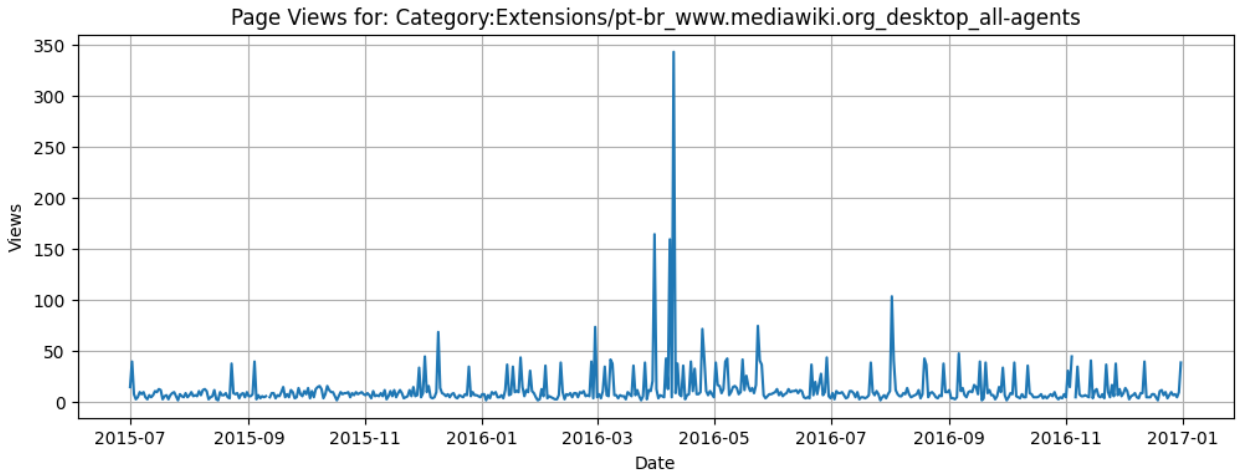
Visualizing sample pages with internal NaNs:



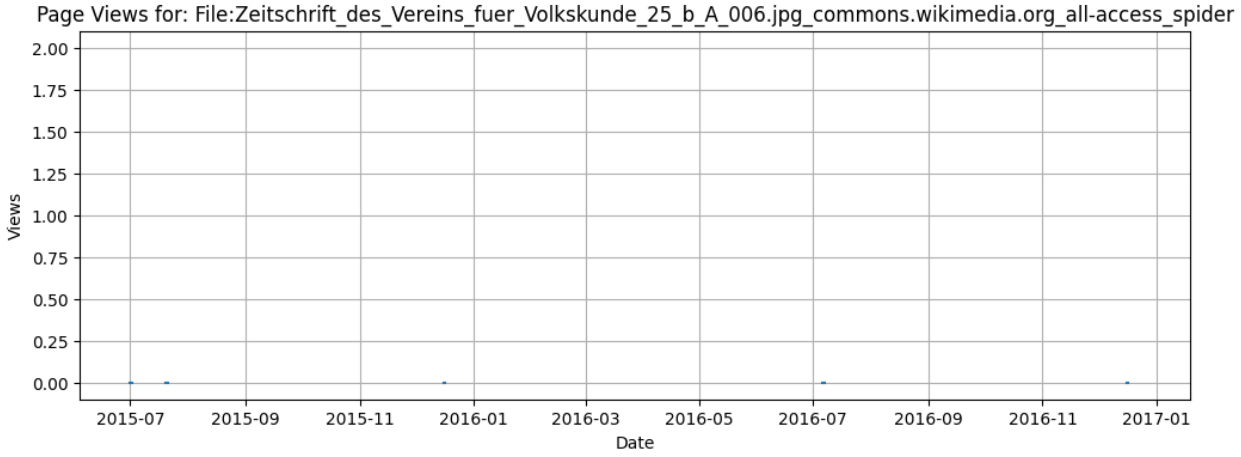

Page Views for: □□□□□□□□□□_ja.wikipedia.org_desktop_all-agents
Views
Date
0
2000
4000
6000
8000
10000
12000
14000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

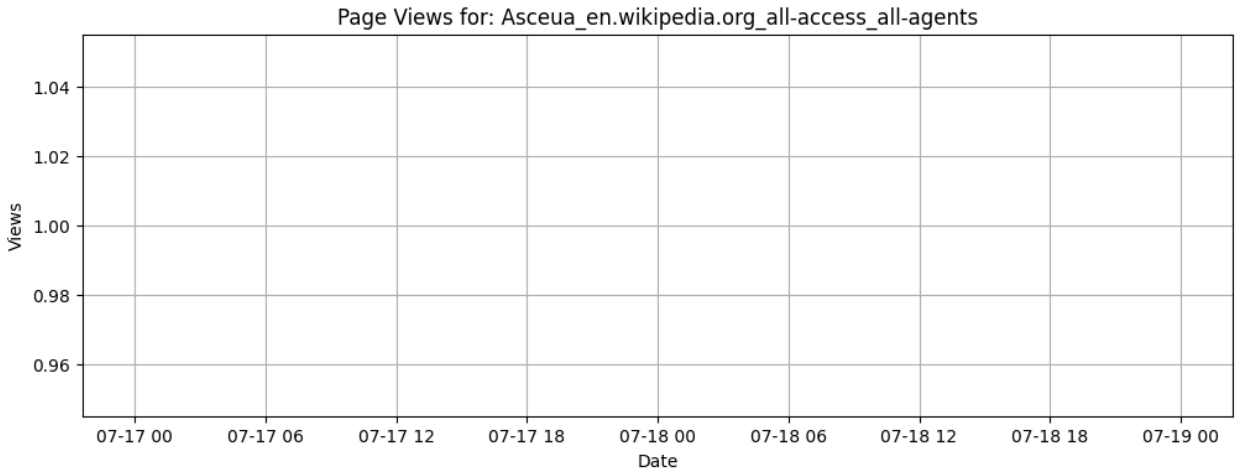

Page Views for: File:Wembleyold.jpg_commons.wikimedia.org_mobile-web_all-agents
Views
Date
0
250
500
750
1000
1250
1500
1750
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

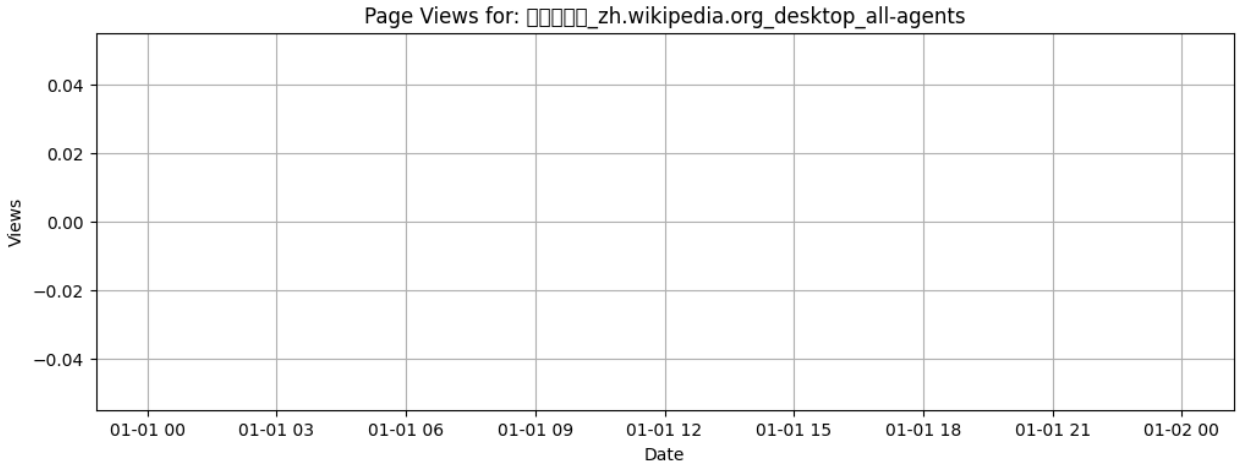

Page Views for: Help:CirrusSearch/ar_www.mediawiki.org_all-access_spider
Views
Date
0
10
20
30
40
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

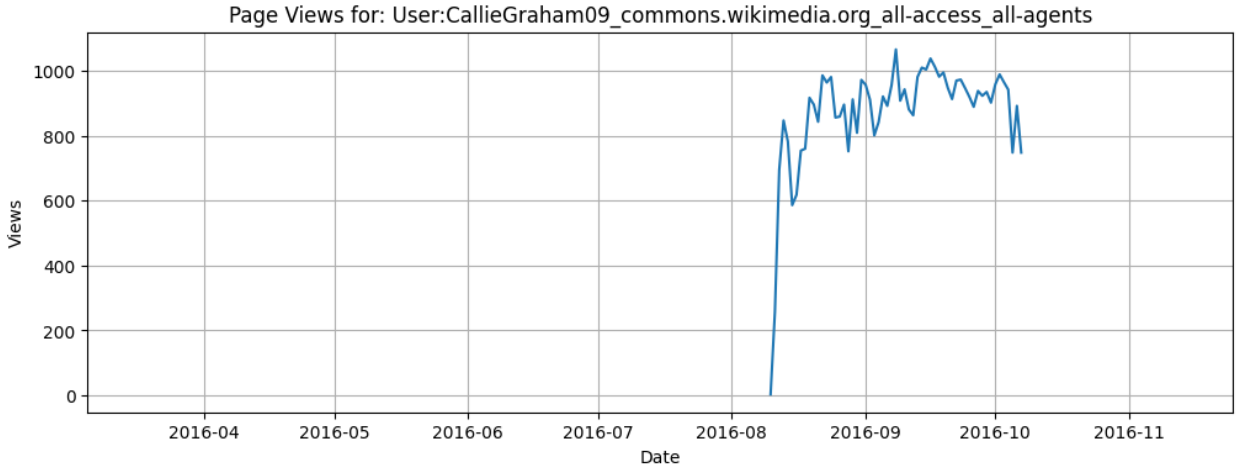



```
Visualizing sample pages with trailing NaNs:
```










```
# all_nan count
train[train['missing_pct'] == 100].shape[:1]
```

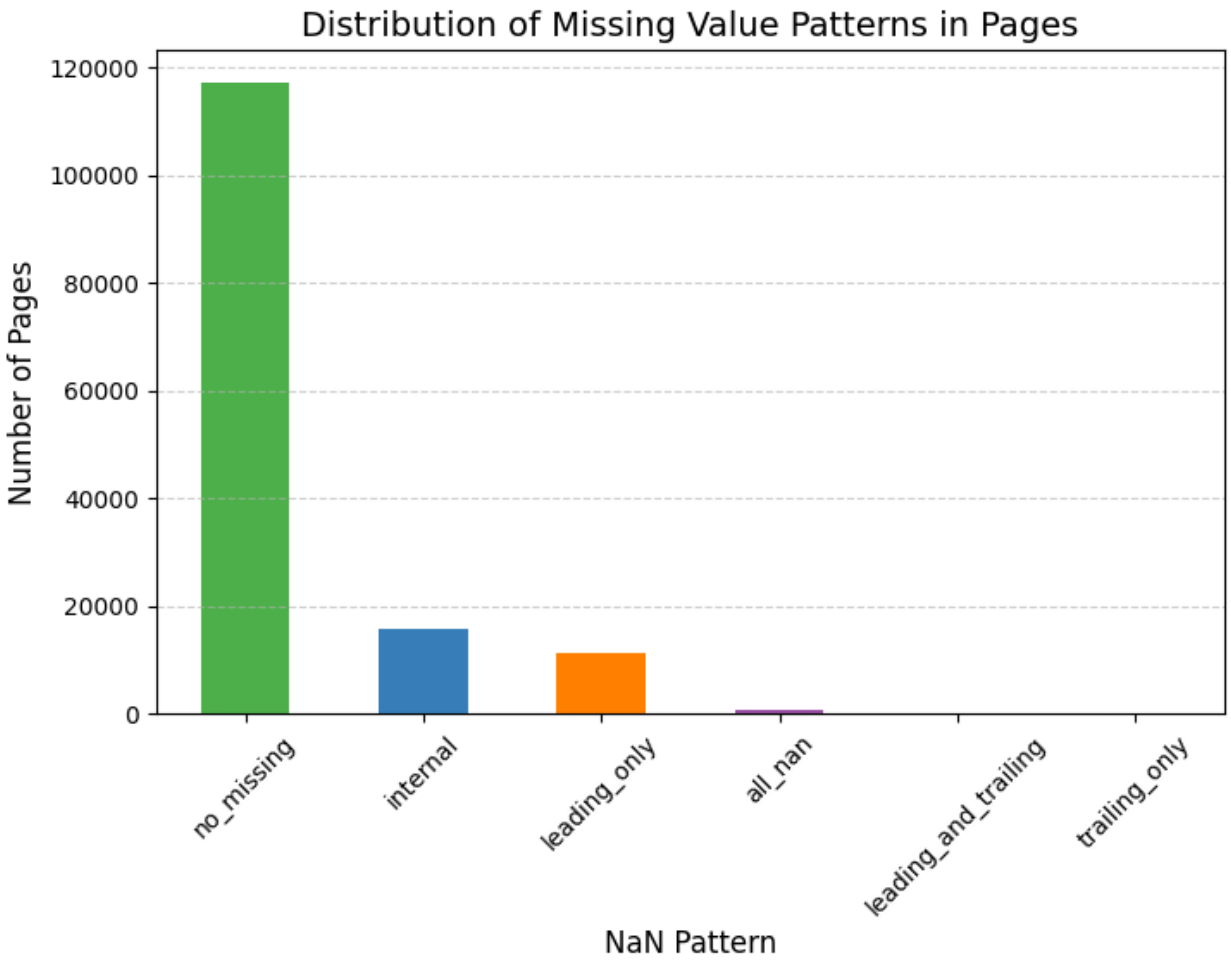
```
(652,)
```

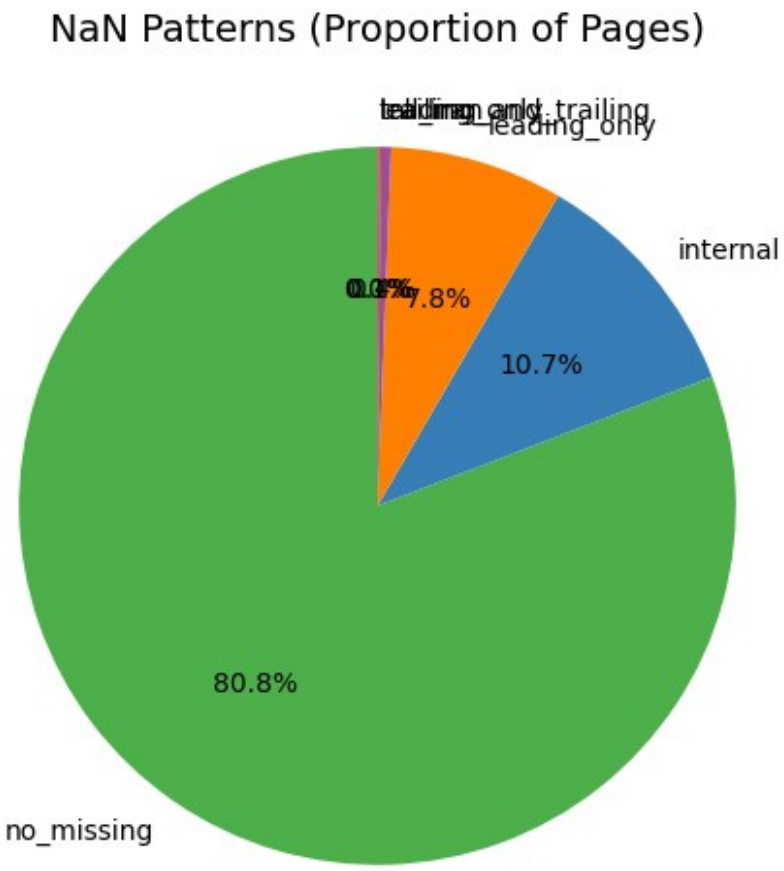
```python
# Helper to classify NaN pattern for each row
def classify_nan_pattern(row, date_cols):
    values = row[date_cols].values
    isnan = pd.isna(values)

    if isnan.all():
        return "all_nan"

    # Find first and last non-NaN indices
    first_valid = (~isnan).argmax()
    last_valid = len(isnan) - 1 - (~isnan[::-1]).argmax()

    leading = first_valid > 0
    trailing = last_valid < len(isnan) - 1
    internal = isnan[first_valid:last_valid+1].any()

    if leading and not trailing and not internal:
        return "leading_only"
    elif trailing and not leading and not internal:
        return "trailing_only"
    elif leading and trailing and not internal:
        return "leading_and_trailing"
    elif internal:
        return "internal"
    else:
        return "no_missing"

# Apply classification
train['nan_pattern'] = train.apply(lambda row:
classify_nan_pattern(row, date_cols), axis=1)

# Count results
nan_pattern_counts = train['nan_pattern'].value_counts()
print(nan_pattern_counts)
```

```
nan_pattern
no_missing              117277
internal                 15591
leading_only             11321
all_nan                    652
leading_and_trailing       186
trailing_only               36
Name: count, dtype: int64
```

```python
# Count patterns
nan_pattern_counts = train['nan_pattern'].value_counts()

# plot stacked bar chart to visualize the distribution
plt.figure(figsize=(8,5))
nan_pattern_counts.plot(
```

```
    kind="bar",
    color=["#4daf4a", "#377eb8", "#ff7f00", "#984ea3", "#e41a1c",
"#999999"]
)
plt.title("Distribution of Missing Value Patterns in Pages",
fontsize=14)
plt.xlabel("NaN Pattern", fontsize=12)
plt.ylabel("Number of Pages", fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.6)

plt.show()

# percentage view (to compare proportions)
plt.figure(figsize=(6,6))
(nan_pattern_counts / nan_pattern_counts.sum() * 100).plot(
    kind="pie",
    autopct="%.1f%%",
    startangle=90,
    colors=["#4daf4a", "#377eb8", "#ff7f00", "#984ea3", "#e41a1c",
"#999999"]
)

plt.title("NaN Patterns (Proportion of Pages)", fontsize=14)
plt.ylabel("")  # remove default y-label
plt.show()
```


Distribution of Missing Value Patterns in Pages
Number of Pages
120000
100000
80000
60000
40000
20000
0
no_missing
internal
leading_only
all_nan
leading_and_trailing
trailing_only
NaN Pattern



Handling Missing Values in Page View Series

1. Leading NaNs (before the series starts)
   - **Likely cause:** Page didn't exist yet (new page created after 2015).
   - **Meaning:** Structural missing values, not true gaps.
   - **Best strategies:**
     - Fill with **0s** → page didn't exist, so views = 0 is valid.
     - Alternatively, leave as **NaN** if you want to explicitly model "page not yet created."
   - **Recommended:** Fill with **0s** (common in competitions/research papers).

---

2. Internal NaNs (gaps in the middle of the series)
   - **Likely cause:** Logging/collection issues, temporary data drop, or bot/spam detection filters.

- **Meaning:** True missing values (traffic existed but wasn't captured).
- **Best strategies:**
  - **Forward/Backward fill** → propagate nearby values (good for short gaps).
  - **Linear interpolation** → smooth across missing days.
  - **Rolling mean imputation** → fill using average of nearby window (e.g., 7 days).
  - ⚠ Avoid **global 0-filling** → introduces fake inactivity.
- ✅ **Recommended:** Interpolation or forward/backfill depending on gap size.

---

3. Trailing NaNs (after the series ends)

- **Likely cause:** Page deleted or stopped being tracked.
- **Meaning:** Structural missing values (page ceased to exist).
- **Best strategies:**
  - Fill with **0s** → no views after deletion.
  - Optionally **drop page** if very high % missing (e.g., >90%).
- ✅ **Recommended:** Fill with **0s**.

⚖ Hybrid Imputation Policy

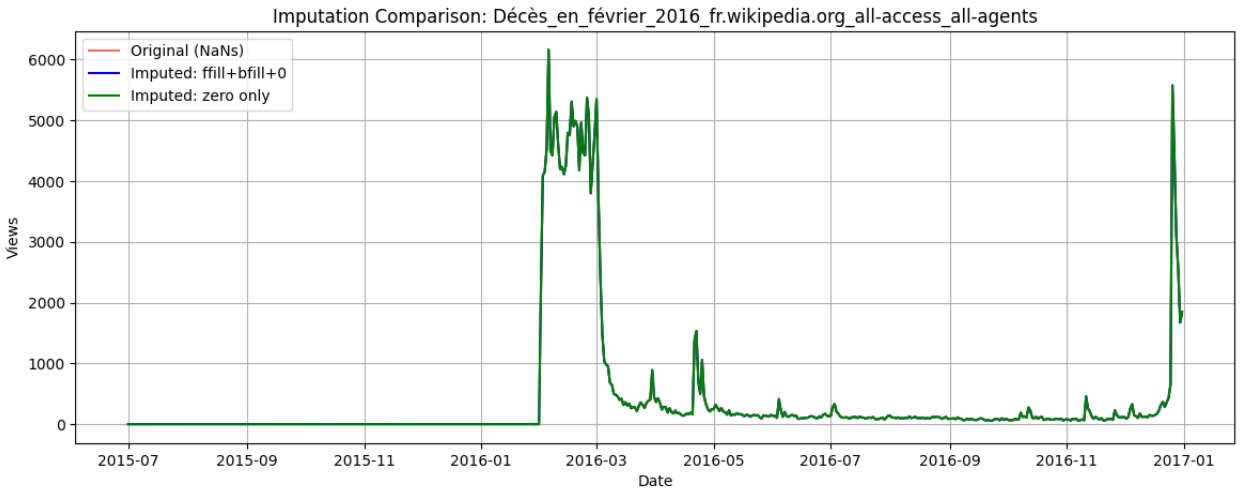
- **Leading NaNs** → fill with **0**.
- **Trailing NaNs** → fill with **0**.
- **Internal NaNs** → use **interpolation** (linear or forward/backward fill).
- **All-NaN pages** (e.g., page 652) → **drop** (no useful data).

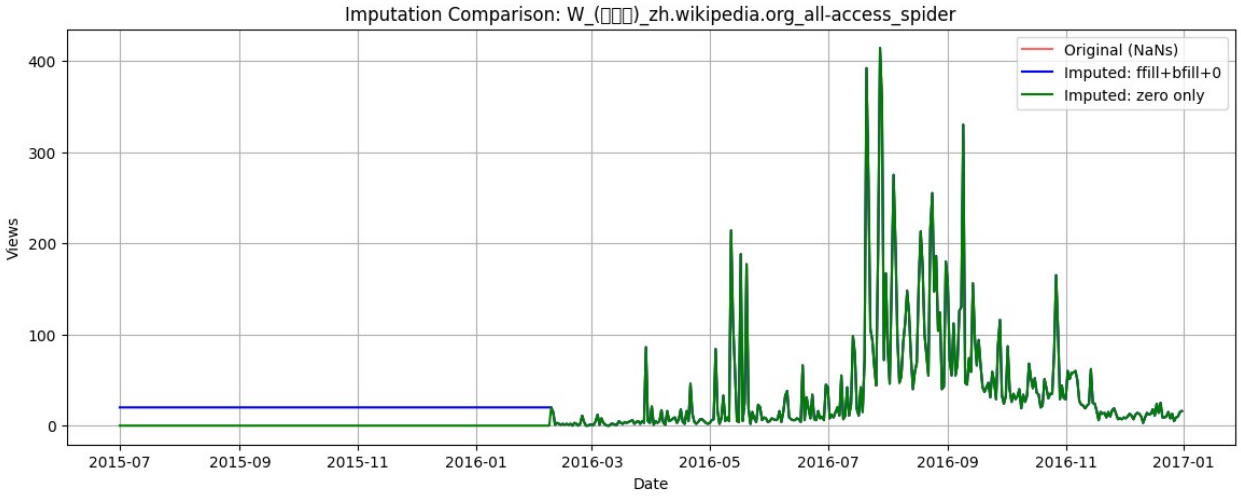
```
sample_trailing_nan_pages['Page'].to_list()
```

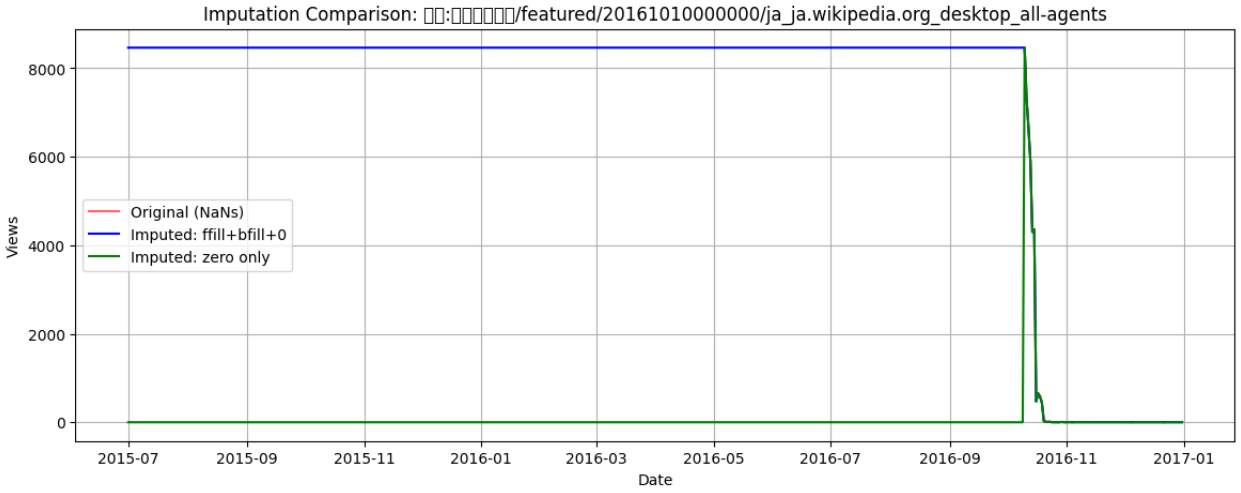
```
['Категория:Гидра_(созвездие)_ru.wikipedia.org_all-access_all-agents',
 'File:Zeitschrift_des_Vereins_fuer_Volkskunde_25_b_A_006.jpg_commons.wikimedia.org_all-access_spider',
 'Asceua_en.wikipedia.org_all-access_all-agents',
 '是松豐三郎_zh.wikipedia.org_desktop_all-agents',
 'User:CallieGraham09_commons.wikimedia.org_all-access_all-agents']
```

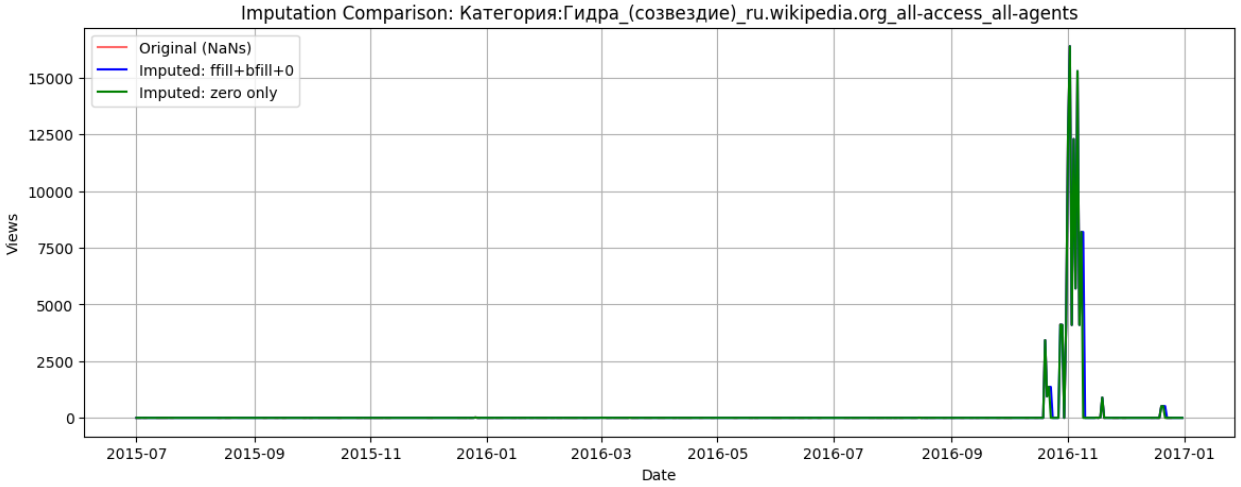
```
# --- Create both imputation strategies ---
def impute_ffill_bfill(df, date_cols):
    filled = df.copy()
    filled = filled[filled[date_cols].notna().any(axis=1)]
    # Forward + backward fill, then 0
    filled[date_cols] = (
        filled[date_cols]
        .fillna(method='ffill', axis=1)
        .fillna(method='bfill', axis=1)
        .fillna(0)
    )
    filled[date_cols] = filled[date_cols].interpolate(axis=1,
limit_direction='both')
    return filled

def impute_zero_only(df, date_cols):
    filled = df.copy()
    filled = filled[date_cols].notna().any(axis=1)
    # Directly set missing to 0 (leading/trailing)
    filled = df[df[date_cols].notna().any(axis=1)].copy()
    filled[date_cols] = filled[date_cols].fillna(0)
    filled[date_cols] = filled[date_cols].interpolate(axis=1,
limit_direction='both')
    return filled

train_imputed_ffill = impute_ffill_bfill(train, date_cols)
train_imputed_zero = impute_zero_only(train, date_cols)


# --- Plot comparison ---
def plot_imputation_comparison(page_row, date_cols, df_ffill,
df_zero):
    page_name = page_row['Page']
    idx = page_row.name

    original = page_row[date_cols].astype(float)
    page_name = row['Page']

    ffill_vals = df_ffill.loc[df_ffill['Page'] == page_name,
date_cols].iloc[0].astype(float)
    zero_vals  = df_zero.loc[df_zero['Page'] == page_name,
date_cols].iloc[0].astype(float)

    dates = pd.to_datetime(date_cols)

    plt.figure(figsize=(14, 5))
    plt.plot(dates, original, label="Original (NaNs)", color="red",
alpha=0.6)
    plt.plot(dates, ffill_vals, label="Imputed: ffill+bfill+0",
color="blue")
```

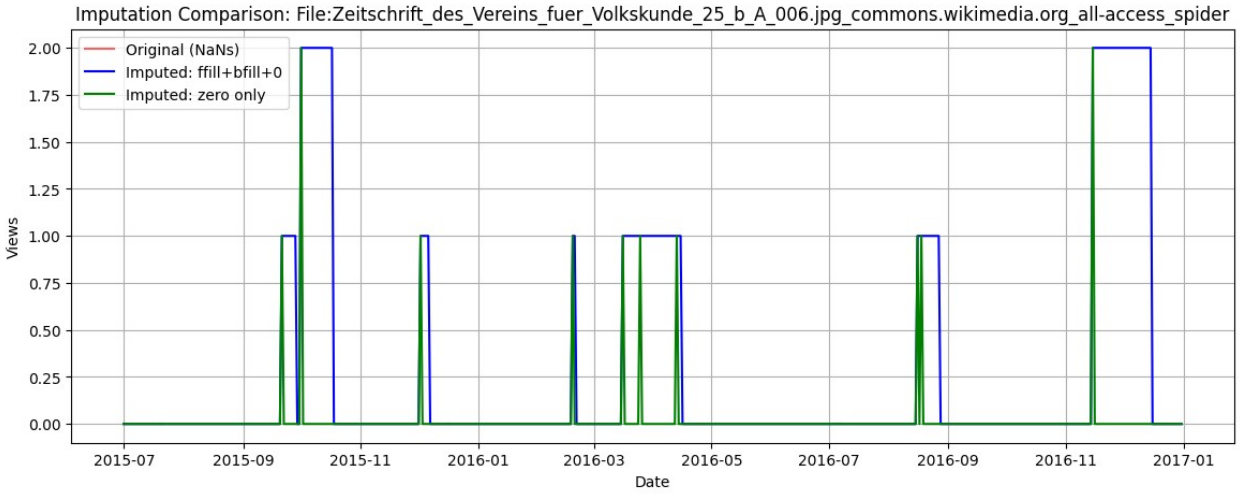
```
    plt.plot(dates, zero_vals, label="Imputed: zero only",
color="green")

    plt.title(f"Imputation Comparison: {page_name}")
    plt.xlabel("Date")
    plt.ylabel("Views")
    plt.legend()
    plt.grid(True)
    plt.show()


# Example: compare for 3 leading-NaN sample pages
for idx, row in sample_leading_nan_pages.head(3).iterrows():
    plot_imputation_comparison(row, date_cols, train_imputed_ffill,
train_imputed_zero)

# Example: compare for 3 trailing-NaN sample pages
for idx, row in sample_trailing_nan_pages.head(3).iterrows():
    plot_imputation_comparison(row, date_cols, train_imputed_ffill,
train_imputed_zero)
```

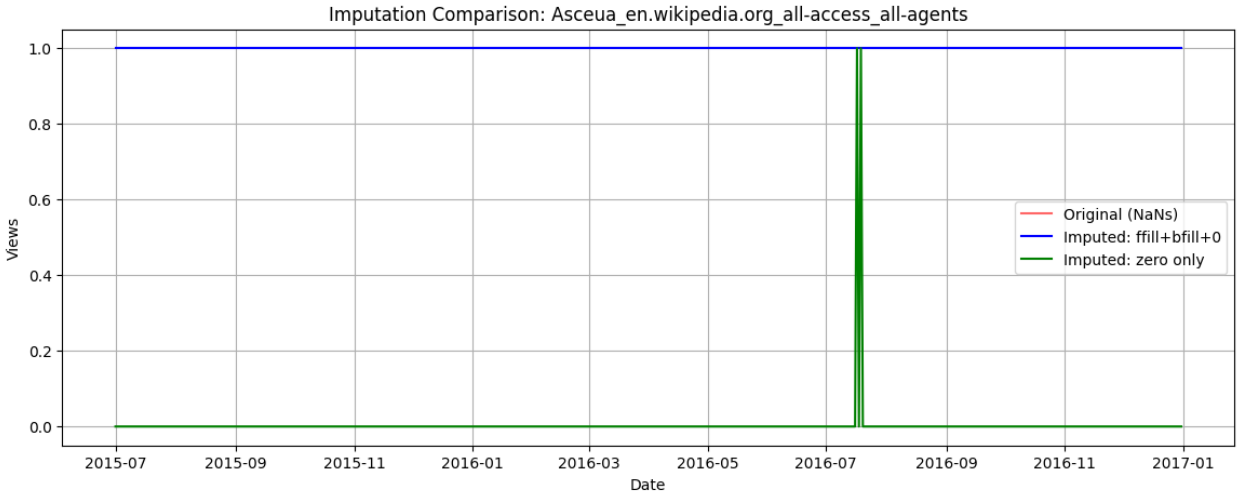



Imputation Comparison: W_(□□□)_zh.wikipedia.org_all-access_spider
Original (NaNs)
Imputed: ffill+bfill+0
Imputed: zero only
Views
Date
0
100
200
300
400
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Imputation Comparison: □□:□□□□□□/featured/20161010000000/ja_ja.wikipedia.org_desktop_all-agents
Original (NaNs)
Imputed: ffill+bfill+0
Imputed: zero only
Views
Date
0
2000
4000
6000
8000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Imputation Comparison: Категория:Гидра_(созвездие)_ru.wikipedia.org_all-access_all-agents
Original (NaNs)
Imputed: ffill+bfill+0
Imputed: zero only
Views
Date
0
2500
5000
7500
10000
12500
15000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01






```python
def impute_page_views(df, date_cols):
    filled = df.copy()

    # 1. Drop rows where all dates are NaN
    filled = filled[filled[date_cols].notna().any(axis=1)]

    # 2. Fill leading/trailing NaNs with 0
    filled[date_cols] = (
        filled[date_cols]
        .fillna(0)                          # if still NaN, set to 0
    )

    # 3. Interpolate internal NaNs
    filled[date_cols] = filled[date_cols].interpolate(
        axis=1,
        limit_direction='both'
    )

    return filled
```

```
# Apply hybrid imputation
train_imputed = impute_page_views(train, date_cols)

print("Shape after imputation:", train_imputed.shape)
print("Total missing after imputation:",
train_imputed[date_cols].isnull().sum().sum())
```

```
Shape after imputation: (144411, 554)
Total missing after imputation: 0
```

Remove never-viewed pages (flatlined zeros)

- **Low-traffic/noisy pages:** A subset of pages (flatlined at zero) provide negligible signal and can be excluded to reduce noise in downstream models.

 Practical Rule of Thumb for Filtering Pages

- **eps = 1**
  - Removes pages averaging **<1 view/day**
  - Over ~550 days → <550 total views
  - Good baseline to keep only meaningful pages
- **eps = 5 or 10**
  - Keeps only pages with at least **moderate recurring traffic**
  - More aggressive filtering
- **eps = 0.1**
  - Drops pages below **0.1 daily average** (~55 total views over dataset)
  - Ultra-safe: only eliminates pages with almost no activity

```
def remove_zero_traffic_pages(df, date_cols, eps=1):
    """
    Remove pages with no real traffic:
    - mean_views <= eps
    - OR std_views <= eps (flatlined or near-flat)
    """
    if 'mean_views' not in df.columns or 'std_views' not in
df.columns:
        df = df.copy()
        df['mean_views'] = df[date_cols].mean(axis=1)
        df['std_views'] = df[date_cols].std(axis=1)

    mask = ~(
        (df['mean_views'] <= eps) |
        (df['std_views'] <= eps)
    )
```

```
    cleaned = df[mask].copy()
    print(f"Removed {len(df) - len(cleaned)} near-zero traffic pages
out of {len(df)}")
    return cleaned

train_clean = remove_zero_traffic_pages(train_imputed, date_cols)
```

```
Removed 2226 near-zero traffic pages out of 144411
```

3. Page Name Parsing

Tasks:

- Split `Page` into:
  - **Title** (article name)
  - **Language** (en, de, fr, etc.)
  - **Access Type** (desktop, mobile, all-access)
  - **Access Origin** (spider, all-agents)
- Quick diagnostics:
  - How many unique languages
  - Top languages by #pages
  - Missingness distribution by row (per page) and by column (per date)

```
print("\nexample Page values:\n",
train_clean['Page'].sample(10).to_list())
```

```
example Page values:
 ['王岐山_zh.wikipedia.org_all-access_all-agents',
'Альбус_Дамблдор_ru.wikipedia.org_all-access_spider', '西游记之孙悟空三打
白骨精_zh.wikipedia.org_all-access_all-agents', 'カエンタケ
_ja.wikipedia.org_desktop_all-agents', '訃報_2016年3月
_ja.wikipedia.org_desktop_all-agents', 'A級戦犯_ja.wikipedia.org_all-
access_spider', 'Category:Courageous_Cunts_commons.wikimedia.org_all-
access_all-agents', '维基百科_zh.wikipedia.org_desktop_all-agents',
'Saison_5_de_The_Walking_Dead_fr.wikipedia.org_desktop_all-agents',
'Terminator:_Die_Erlösung_de.wikipedia.org_mobile-web_all-agents']
```

Correct Parsing Logic

We need to handle **two cases** robustly:

1. `<title>_<lang>.wikipedia.org_<access>_<origin>`
2. `<title>_commons.wikimedia.org_<access>_<origin>`

```python
# Parse Page into components:
# Better approach with regex

def split_page_columns(s):
    # Case 1: language-specific Wikipedia
    m = re.match(r'^(.*)_(?P<lang>[a-z\-]+)\.wikipedia\.org_(?
P<access>[^_]+)_(?P<origin>[^_]+)$', s)
    if m:
        return pd.Series([m.group(1), m.group("lang"),
m.group("access"), m.group("origin")])

    # Case 2: Wikimedia Commons (no language, treat lang='commons')
    m = re.match(r'^(.*)_commons\.wikimedia\.org_(?P<access>[^_]+)_(?
P<origin>[^_]+)$', s)
    if m:
        return pd.Series([m.group(1), "commons", m.group("access"),
m.group("origin")])

    # Fallback: split by "_" and guess
    parts = s.split('_')
    title = "_".join(parts[:-3]) if len(parts) > 3 else parts[0]
    lang = parts[-3].split('.')[0] if len(parts) > 2 else np.nan
    access = parts[-2] if len(parts) > 1 else np.nan
    origin = parts[-1] if len(parts) > 0 else np.nan
    return pd.Series([title, lang, access, origin])

# Apply
train_clean[['Title','Language','AccessType','AccessOrigin']] =
train_clean['Page'].apply(split_page_columns)

# quick checks
print("\nUnique languages:", train_clean['Language'].nunique())
print("Top 10 languages by page count:\n",
train_clean['Language'].value_counts().head(10))
print("\nAccess types:\n",
train_clean['AccessType'].value_counts().head())
print("\nAccess origins:\n",
train_clean['AccessOrigin'].value_counts().head())
```

```
Unique languages: 9
Top 10 languages by page count:
 Language
en         23728
ja         20321
de         18339
fr         17687
zh         17028
ru         14936
es         13990
```

```
commons     9575
www         6581
Name: count, dtype: int64

Access types:
 AccessType
all-access    72433
mobile-web    35269
desktop       34483
Name: count, dtype: int64

Access origins:
 AccessOrigin
all-agents    108645
spider         33540
Name: count, dtype: int64
```

🔍 Handling `commons` and `www` in "languages"

- **commons** → Refers to **Wikimedia Commons** (shared media repository).
  - Not tied to a specific language.

- **www** → Refers to **www.wikipedia.org** (multilingual portal/homepage).
  - Also not tied to a specific language edition.

Key Points

- These are **valid projects**, not parsing errors.

- They **don't behave like real language editions** (e.g., en, ja, de).

What You Can Do

1. ✅ **Keep them** if you want to capture *all traffic* across Wikipedia + sister projects.
   - Includes Commons + homepage, which sometimes get huge traffic.

2. ❌ **Filter them out** if your focus is *strictly language editions*.
   - Drop rows where `Language ∈ {commons, www}`.

3. 🏷️ **Tag them separately** for flexibility:
   - Example: `ProjectType = "language-wiki"` vs `ProjectType = "other-wiki"`.

   - Prevents mixing them into language-based stats while still preserving traffic info.

4. Exploratory Data Analysis (EDA)

Tasks:

- Analyze **distribution of page views**.
- Compare **views by language**.
- Visualize **daily views for sample pages**.

Deliverables for This Step

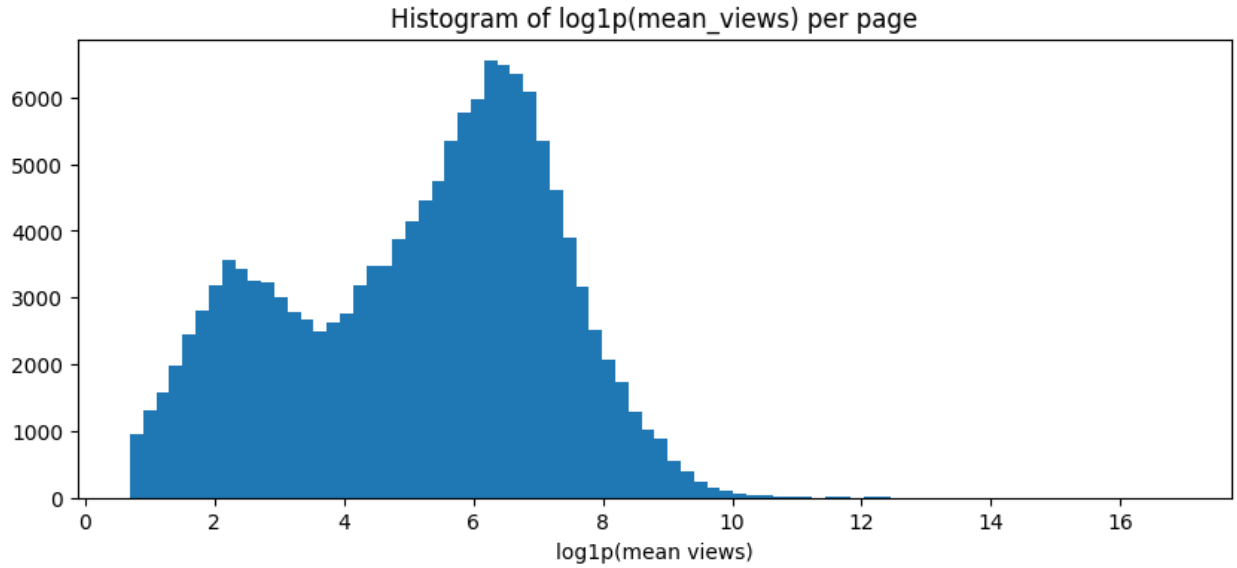
- **Global summary stats**
  - Counts, date range, missingness overview
- **Per-language statistics**
  - Total, mean, median views
  - Number of pages per language
- **Device & Agent breakdowns**
  - Desktop / Mobile / All-access
  - Spider / All-agents
- **Distribution visualizations**
  - Log-scale histograms
  - Boxplots for page view spread
- **Top pages by language**
  - Top 10 pages overall
  - Examples for deeper inspection
- **Time-series aggregates per language**
  - Daily totals
  - Day-of-week patterns
- **Actionable insights**
  - 3–5 concise insights to highlight in the final report

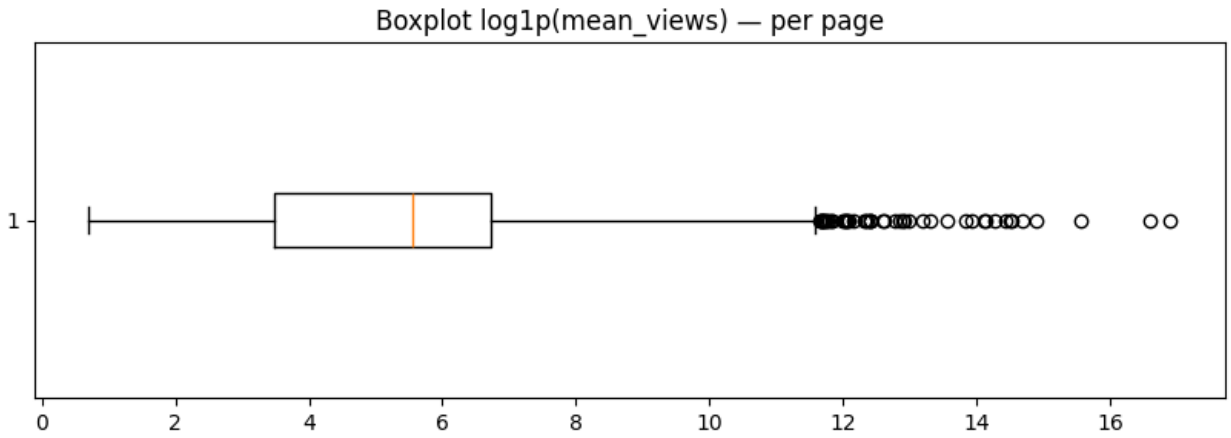
page-level summary stats (mean/median/std) and global distribution → Univariate

```
# page-level stats and distribution
train_clean['mean_views']   = train_clean[date_cols].mean(axis=1)
train_clean['median_views'] = train_clean[date_cols].median(axis=1)
train_clean['std_views']    = train_clean[date_cols].std(axis=1)
train_clean['max_views']    = train_clean[date_cols].max(axis=1)

print("Overall pages:", len(train_clean))
print("mean_views describe:\n", train_clean['mean_views'].describe())
```

```
# Histogram (log1p) of mean views across pages
plt.figure(figsize=(10,4))
plt.hist(np.log1p(train_clean['mean_views']), bins=80)
plt.title("Histogram of log1p(mean_views) per page")
plt.xlabel("log1p(mean views)")
plt.show()

# Boxplot (log scale)
plt.figure(figsize=(10,3))
plt.boxplot(np.log1p(train_clean['mean_views']), vert=False)
plt.title("Boxplot log1p(mean_views) — per page")
plt.show()
```

```
Overall pages: 142185
mean_views describe:
 count    1.421850e+05
mean     1.336270e+03
std      7.530055e+04
min      1.001818e+00
25%      3.139273e+01
50%      2.542636e+02
75%      8.393073e+02
max      2.193851e+07
Name: mean_views, dtype: float64
```





 Page-Level Statistics

- **Total pages:** 144,411

 Page Views (across time, per page)

- **Mean:** ~1.3k views
- **Median:** ~243 views
- **Standard Deviation:** ~74k (huge spread, very skewed)
- **Minimum:** 0
- **Maximum:** ~21.9M (English *Main_Page* dominates)

 Distribution Insight

- Most pages have **very low views**.
- A small number of pages have **extremely high values** → heavy-tailed distribution.

top pages by language (top 10 per language)

```
# top pages per language by mean_views
top_n = 10
top_by_lang = {}
for lang, grp in train_clean.groupby('Language'):
    top = grp.sort_values('mean_views', ascending=False).head(top_n)
[['Page','mean_views','median_views','std_views']]
    top_by_lang[lang] = top
    print(f"\nTop {top_n} for language: {lang}")
    display(top.reset_index(drop=True))
```

```
Top 10 for language: commons
```


```
{"summary":"{\n  \"name\": \"    display(top\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
```

\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Special:UploadWizard_commons.wikimedia.org_desktop_all-agents\",\n
\"Special:Search_commons.wikimedia.org_desktop_all-agents\",\n
\"Main_Page_commons.wikimedia.org_desktop_all-agents\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"mean_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
31559.988429701523,\n   \"min\": 39845.821818181816,\n
\"max\": 122092.06909090909,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   40015.534545454546,\n
115181.37454545454,\n   71201.67818181818\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"median_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
44306.58510938928,\n   \"min\": 0.0,\n   \"max\": 121104.5,\
n   \"num_unique_values\": 9,\n   \"samples\": [\n
34848.5,\n   114557.5,\n   17663.5\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"std_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
57712.36724594245,\n   \"min\": 26117.818260808715,\n
\"max\": 185744.9916048866,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   26117.818260808715,\n
38157.66838630386,\n   41287.41101106279\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

Top 10 for language: de

{"summary":"{\n \"name\": \"   display(top\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"Page\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Special:MyPage/toolserverhelferleinconfig.js_de.wikipedia.org_deskto
p_all-agents\",\n
\"Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents\",\n
\"Spezial:Anmelden_de.wikipedia.org_all-access_all-agents\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mean_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 991983.6827262248,\n   \"min\":
57695.37818181818,\n   \"max\": 2916244.0872727274,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
59245.37272727273,\n   2023071.0618181818,\n
73257.82909090909\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"median_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 991960.761956144,\n   \"min\":

```
29495.5,\n        \"max\": 2900707.0,\n        \"num_unique_values\":
10,\n        \"samples\": [\n          87593.0,\n          2027171.0,\
n          33561.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_views\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 80143.15174188945,\n        \"min\":
16331.940082491492,\n        \"max\": 256910.33175604572,\n
\"num_unique_values\": 10,\n        \"samples\": [\n
46392.637360446584,\n          166525.60620171492,\n
206388.13501341807\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top 10 for language: en


```
{"summary":"{\n  \"name\": \"    display(top\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Main_Page_en.wikipedia.org_all-access_spider\",\n
\"Main_Page_en.wikipedia.org_desktop_all-agents\",\n
\"Special:Search_en.wikipedia.org_mobile-web_all-agents\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mean_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7710132.07429472,\n        \"min\": 226018.55636363637,\n
\"max\": 21938511.094545454,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          234582.5581818182,\n
15953631.741818182,\n          531205.1618181819\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6647262.124116644,\n        \"min\": 105600.0,\n        \"max\":
19457533.0,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          121381.0,\n          12733277.5,\n          421291.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"std_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3668689.194522506,\n        \"min\": 126779.14336756086,\n
\"max\": 9529193.607360583,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          832404.2754017095,\n
9529193.607360583,\n          292909.9244350451\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Top 10 for language: es


```
{"summary":"{\n  \"name\": \"    display(top\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
```

\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Lali_Esp\\u00f3sito_es.wikipedia.org_all-access_all-agents\",\n
\"Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents\",\n
\"Especial:Buscar_es.wikipedia.org_mobile-web_all-agents\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mean_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 463184.6335238652,\n   \"min\":
46812.983636363635,\n   \"max\": 1366349.6436363636,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
48368.523636363636,\n   1027413.4036363637,\n
61927.21090909091\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"median_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 472076.62058762595,\n   \"min\":
2331.0,\n   \"max\": 1357293.0,\n   \"num_unique_values\":
10,\n   \"samples\": [\n   2331.0,\n   1048781.5,\n
59732.5\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"std_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 59549.690920814675,\n   \"min\":
32317.615274376032,\n   \"max\": 233505.247116279,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
85492.20331576202,\n   166223.58173266854,\n
39916.426989287735\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Top 10 for language: fr

{"summary":"{\n \"name\": \"   display(top\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"Page\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Organisme_de_placement_collectif_en_valeurs_mobili\\
u00e8res_fr.wikipedia.org_all-access_all-agents\",\n
\"Wikip\\u00e9dia:Accueil_principal_fr.wikipedia.org_mobile-web_all-
agents\",\n   \"Sp?cial:Search_fr.wikipedia.org_all-access_all-
agents\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mean_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 521273.85093527025,\n   \"min\":
66589.35454545454,\n   \"max\": 1579055.7581818183,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
66632.59818181818,\n   1111459.6745454546,\n
138535.5781818182\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"median_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 543935.8111251015,\n   \"min\":

```
115.0,\n        \"max\": 1596963.0,\n        \"num_unique_values\":
9,\n        \"samples\": [\n          152.5,\n          1161585.5,\n
164142.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_views\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 158212.74697159854,\n        \"min\":
35781.20982460971,\n        \"max\": 458745.49078014784,\n
\"num_unique_values\": 10,\n        \"samples\": [\n
458745.49078014784,\n          146910.7015735104,\n
86550.696745552\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top 10 for language: ja


```
{"summary":"{\n  \"name\": \"    display(top\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"\\
u7279\\u5225:\\u5916\\u90e8\\u30ea\\u30f3\\u30af\\u691c\\
u7d22_ja.wikipedia.org_all-access_all-agents\",\n          \"\\u30e1\\
u30a4\\u30f3\\u30da\\u30fc\\u30b8_ja.wikipedia.org_desktop_all-
agents\",\n          \"\\u7279\\u5225:\\u6700\\u8fd1\\u306e\\u66f4\\
u65b0_ja.wikipedia.org_all-access_all-agents\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mean_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
120253.81016233828,\n        \"min\": 18778.03090909091,\n
\"max\": 383188.7181818182,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          18783.983636363635,\n
243904.3909090909,\n          32114.165454545455\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
120746.58725290012,\n        \"min\": 11398.0,\n        \"max\":
376056.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
11398.0,\n          241206.0,\n          24564.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
10938.8859319756,\n        \"min\": 14531.007331050005,\n
\"max\": 47683.4074038522,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          19034.972093178232,\n
41216.8268421326,\n          21238.06387791103\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Top 10 for language: ru

{"summary":"{\n \"name\": \"   display(top\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"Page\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Special:Search_ru.wikipedia.org_desktop_all-agents\",\n
\"\\u0417\\u0430\\u0433\\u043b\\u0430\\u0432\\u043d\\u0430\\u044f_\\
u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\
u0430_ru.wikipedia.org_desktop_all-agents\",\n   \"\\u0421\\
u043b\\u0443\\u0436\\u0435\\u0431\\u043d\\u0430\\u044f:\\u0421\\
u0441\\u044b\\u043b\\u043a\\u0438_\\u0441\\u044e\\u0434\\
u0430_ru.wikipedia.org_all-access_all-agents\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"mean_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
678347.0444042308,\n   \"min\": 22113.794545454544,\n
\"max\": 1974580.8218181818,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   39924.494545454545,\n
1350690.9381818182,\n   45640.00727272727\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"median_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
445399.2505049574,\n   \"min\": 15828.5,\n   \"max\":
1352373.0,\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   22068.5,\n   702277.5,\n   17462.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"std_views\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1179633.2752707992,\n   \"min\": 29939.71501695548,\n
\"max\": 2879428.0588861187,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   89013.72320920828,\n
2879428.0588861187,\n   55203.97480716372\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

Top 10 for language: www

{"summary":"{\n \"name\": \"   display(top\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"Page\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Download_www.mediawiki.org_all-access_all-agents\",\n
\"MediaWiki_www.mediawiki.org_desktop_all-agents\",\n
\"Special:MyLanguage/Help:Logging_in_www.mediawiki.org_all-access_all-
agents\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mean_views\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 8890.110011066368,\n   \"min\":
5697.74,\n   \"max\": 31103.154545454545,\n
\"num_unique_values\": 10,\n   \"samples\": [\n

```
5789.94,\n          24010.00181818182,\n          6969.269090909091\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"median_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9327.131621868893,\n        \"min\": 51.0,\n        \"max\": 29155.5,\
n        \"num_unique_values\": 10,\n        \"samples\": [\n
5728.5,\n          22207.0,\n          8378.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32825.14050077509,\n        \"min\": 670.6221875493798,\n
\"max\": 83038.0087000102,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          670.6221875493798,\n
13825.900827247333,\n          5260.058864764876\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Top 10 for language: zh


```
{"summary":"{\n  \"name\": \"    display(top\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Running_Man_zh.wikipedia.org_desktop_all-agents\",\n
\"Wikipedia:\\u9996\\u9875_zh.wikipedia.org_desktop_all-agents\",\n
\"Running_Man_zh.wikipedia.org_all-access_all-agents\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mean_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
68307.5905648291,\n        \"min\": 10711.98,\n        \"max\":
224898.74909090908,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          14488.078181818182,\n
120792.07454545454,\n          20883.354545454546\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
67385.76755261868,\n        \"min\": 7732.5,\n        \"max\":
218543.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
13765.5,\n          120227.0,\n          20069.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std_views\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9209.745965960328,\n        \"min\": 5682.706155743809,\n
\"max\": 33800.24128462218,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          5682.706155743809,\n
16697.201657607093,\n          8221.56764042321\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

 Top Pages per Language

- **English (en):**
  - Dominated by **Main_Page** → >20M average views.
- **Japanese (ja):**
  - Top page: **メインページ** → ~383k average views.
- **German (de):**
  - Top page: **Wikipedia:Hauptseite** → ~2.9M average views.

 Key Insight

- Across languages, the **homepage / main entry point** is consistently the **most viewed page**.

device & agent breakdown (aggregated view) → Bivariate

```
# device & agent breakdowns
# compute total views per page (sum across dates)
train_clean['total_views'] = train_clean[date_cols].sum(axis=1)

# total views by language
lang_agg = train_clean.groupby('Language')
['total_views'].agg(['sum','mean','median','count']).sort_values('sum'
, ascending=False)
print("Per-language total / mean / median / count:\n", lang_agg)

# device (AccessType) breakdown per language (sum)
device_lang = train_clean.groupby(['Language','AccessType'])
['total_views'].sum().unstack(fill_value=0)
print("\nDevice breakdown (sum of total_views) per Language:\n")
display(device_lang)

# agent (AccessOrigin) breakdown per language
agent_lang = train_clean.groupby(['Language','AccessOrigin'])
['total_views'].sum().unstack(fill_value=0)
print("\nAgent breakdown (sum of total_views) per Language:\n")
display(agent_lang)
```

```
Per-language total / mean / median / count:
                   sum          mean    median  count
Language
en        5.874867e+10  2.475922e+06  543944.0  23728
es        9.490190e+09  6.783553e+05  350652.0  13990
de        8.861985e+09  4.832317e+05  146402.0  18339
ja        8.571286e+09  4.217945e+05  221499.0  20321
ru        7.998352e+09  5.355083e+05  243498.0  14936
fr        6.377805e+09  3.605928e+05  138933.0  17687
zh        3.171961e+09  1.862791e+05   86884.0  17028
commons   1.049261e+09  1.095833e+05   15704.0   9575
www       2.291383e+08  3.481815e+04    6912.0   6581
```

Device breakdown (sum of total_views) per Language:


```
{"summary":"{\n  \"name\": \"device_lang\",\n  \"rows\": 9,\n
\"fields\": [\n    {\n      \"column\": \"Language\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 9,\n        \"samples\": [\n          \"www\",\
n          \"de\",\n          \"ja\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"all-access\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9082704087.750776,\n        \"min\": 122217819.0,\n        \"max\":
29635920609.0,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          122217819.0,\n          4497298259.0,\n
4377222576.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"desktop\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 5363617884.363953,\n        \"min\": 94185719.0,\n
\"max\": 17309057812.0,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          94185719.0,\n          2004060568.0,\n
1597378984.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mobile-web\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3600559593.0808177,\n        \"min\":
12734738.0,\n        \"max\": 11803692052.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n
12734738.0,\n          2360626518.0,\n          2596684227.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"device_lang"}
```


Agent breakdown (sum of total_views) per Language:


```
{"summary":"{\n  \"name\": \"agent_lang\",\n  \"rows\": 9,\n
\"fields\": [\n    {\n      \"column\": \"Language\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 9,\n        \"samples\": [\n          \"www\",\
n          \"de\",\n          \"ja\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"all-agents\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
17847651460.23753,\n        \"min\": 213444433.0,\n        \"max\":
58174300205.0,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          213444433.0,\n          8780998053.0,\n
8337673719.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"spider\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 173497133.60523707,\n        \"min\": 15693843.0,\n
```

```
\"max\": 574370268.0,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          15693843.0,\n          80987292.0,\n
233612068.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"agent_lang"}
```


 Language Totals

- **English (en):** ~5.87e10 views (≈59B, by far the largest)
- **Spanish (es):** ~9.49e9
- **German (de):** ~8.86e9
- **Japanese (ja):** ~8.57e9
- **Russian (ru):** ~7.99e9
- Then **French (fr), Chinese (zh), etc.** follow.

 Device Breakdown (AccessType – English example)

- **All-access total:** ~29.6B
- **Desktop:** ~17.3B
- **Mobile-web:** ~11.8B
- **Insight:** For English, **desktop > mobile-web** in this dataset.

 Agent Breakdown (AccessOrigin – English example)

- **All-agents total:** ~58.1B
- **Spider (bots/crawlers):** ~574M
- **Insight:** Spiders account for **<1% of total traffic**.

time-series aggregate per language (daily totals) & plot

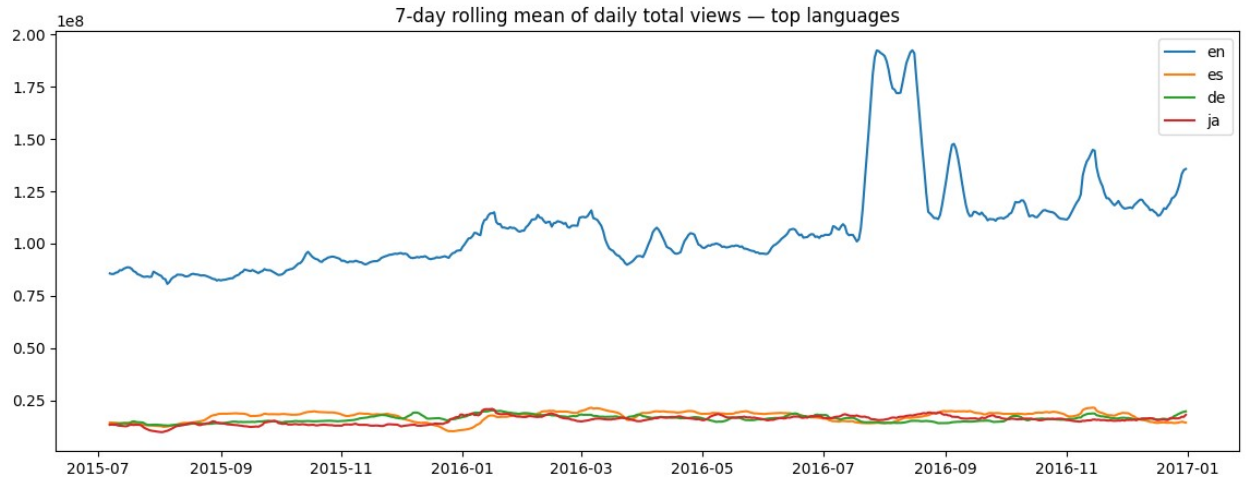
```
# daily aggregate per language (memory efficient)
# compute daily totals by language: result will be DataFrame indexed
by dates with languages as columns
lang_daily = train_clean.groupby('Language')[date_cols].sum().T
lang_daily.index = pd.to_datetime(lang_daily.index)
print("lang_daily shape (dates x languages):", lang_daily.shape)
display(lang_daily.head())
```

```
# Plot daily totals for top 4 languages by total views
top_langs = lang_agg.sort_values('sum',
ascending=False).head(4).index.tolist()
plt.figure(figsize=(14,5))
for lang in top_langs:
    plt.plot(lang_daily.index, lang_daily[lang].rolling(7).mean(),
label=lang)
plt.legend()
plt.title("7-day rolling mean of daily total views — top languages")
plt.show()
```

```
lang_daily shape (dates x languages): (550, 9)
```


```
{"summary":"{\n  \"name\": \"plt\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"commons\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 92347.21552651167,\n
\"min\": 951111.0,\n        \"max\": 1177955.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
1177955.0,\n          1057864.0,\n          1150360.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"de\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 763508.7427891707,\n
\"min\": 11520376.0,\n        \"max\": 13392345.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
13079896.0,\n          13392345.0,\n          12554041.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"en\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2252784.403057958,\n
\"min\": 80167646.0,\n        \"max\": 86198561.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
84438494.0,\n          86198561.0,\n          80167646.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"es\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1039336.0134876978,\n
\"min\": 12606538.0,\n        \"max\": 15278549.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
14601012.0,\n          13710355.0,\n          13427631.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"fr\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 206799.04205701727,\n
\"min\": 8186029.0,\n        \"max\": 8749839.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
8512948.0,\n          8590490.0,\n          8186029.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ja\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1552702.2156698946,\n
\"min\": 11863198.0,\n        \"max\": 15456239.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
13620791.0,\n          14827203.0,\n          12305383.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"ru\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 490635.01027902606,\n
\"min\": 8393202.0,\n        \"max\": 9627637.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
9627637.0,\n          8938519.0,\n          8923451.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"www\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 28190.269388567394,\n
\"min\": 308658.0,\n        \"max\": 383609.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
383609.0,\n          338390.0,\n          325634.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"zh\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 132920.7659837243,\n
\"min\": 4123651.0,\n        \"max\": 4441283.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
4151183.0,\n          4441283.0,\n          4123651.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

 Plot Interpretation

- **English (en):**
  - Completely dominates with **~80M–100M daily views**, spiking above **150M mid-2016**.
  - Noticeable **traffic spikes around July–Sept 2016** (likely global events driving traffic).
- **Spanish (es), German (de), Japanese (ja):**
  - Much smaller scale (**~10M–20M daily views**).
  - Stay relatively steady without such large anomalies.

 Key Takeaway

- **English Wikipedia** drives the majority of traffic and shows sensitivity to **global events/news**.

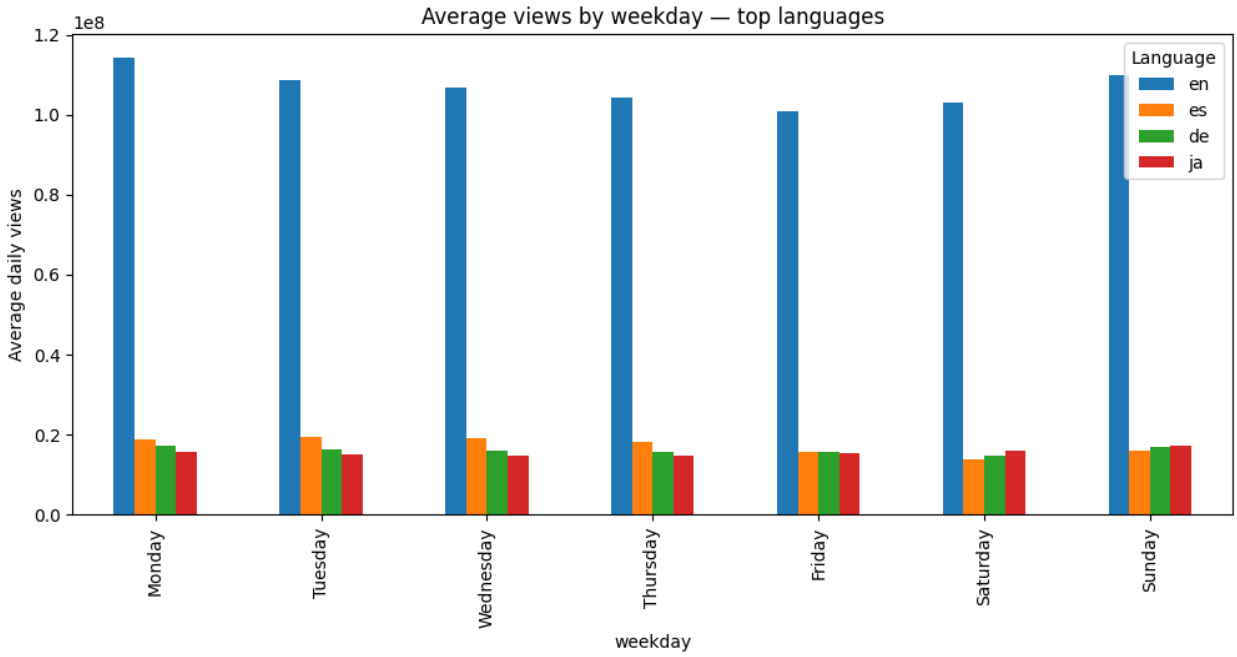
- **Other languages** are steady and less volatile, reflecting more consistent regional usage patterns.

day-of-week pattern (weekday seasonality)

```
# day-of-week effect
lang_daily['weekday'] = lang_daily.index.day_name()
weekday_avg = lang_daily.groupby('weekday')[top_langs].mean()

# reorder weekdays
order =
['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday
']
weekday_avg = weekday_avg.reindex(order)

print("Average daily views by weekday for top languages:\n")
display(weekday_avg)

# plot
weekday_avg.plot(kind='bar', figsize=(12,5))
plt.title("Average views by weekday — top languages")
plt.ylabel("Average daily views")
plt.show()
```

```
Average daily views by weekday for top languages:


{"summary":"{\n  \"name\": \"weekday_avg\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"weekday\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n
\"Monday\",\n          \"Tuesday\",\n          \"Saturday\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"en\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4574562.761666223,\n        \"min\": 100886471.06329113,\n
\"max\": 114372426.53846154,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          114372426.53846154,\n
108732705.97435898,\n          103085213.98734178\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"es\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2116382.76789639,\n
\"min\": 13796576.278481012,\n        \"max\": 19298361.653846152,\n
\"num_unique_values\": 7,\n        \"samples\": [\n
18630062.807692308,\n          19298361.653846152,\n
13796576.278481012\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"de\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 848394.289922219,\n        \"min\": 14751508.683544304,\n
\"max\": 17218916.487179488,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          17218916.487179488,\n
16421426.76923077,\n          14751508.683544304\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ja\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 849769.2517433934,\n
\"min\": 14792592.658227848,\n        \"max\": 17169123.53846154,\n
\"num_unique_values\": 7,\n        \"samples\": [\n
15764574.948717948,\n          15001692.166666666,\n
16076535.620253164\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"weekday_avg"}
```

Key Patterns in Page Views by Language

- English (en)
  - **Pattern:** Peak on **Monday**, gradual decline toward **Friday**, slight rebound on **weekends**.
  - **Interpretation:** Strong weekday usage (likely work/school related).
- Spanish (es)
  - **Pattern:** Peaks on **Tuesday/Wednesday**, lowest on **weekends**.
  - **Interpretation:** Stronger weekday preference.
- German (de)

  - **Pattern:** Similar weekday pattern as English, but **Sunday rebound** is stronger.
  - **Interpretation:** Mix of weekday usage with higher weekend engagement.
- Japanese (ja)
  - **Pattern:** Lowest on **Wednesday–Thursday**, highest on **Sunday**.
  - **Interpretation:** More **leisure/weekend usage** compared to Western patterns.

 Cultural Insight

- **English/Spanish** → **Weekday-heavy**, likely work/study driven.
- **German/Japanese** → **Weekend engagement** is stronger, with cultural differences in browsing habits.

sample pages for deep dive (pick top + random + low traffic)

 Page Selection Rules per Language

For each language:

1. **Top 2 pages** with highest **mean views** (`top_k = 2`)
2. **Top 1 page** with highest **variance** (`var_k = 1`)
3. **Bottom 1 page** with lowest **mean views** (`rand_k = 1`)

 That's **4 pages per language** (though `dict.fromkeys` can remove duplicates if overlaps occur).

- With **4 languages** (`en, es, de, ja`):
  - Maximum = **4 × 4 = 16 pages**.

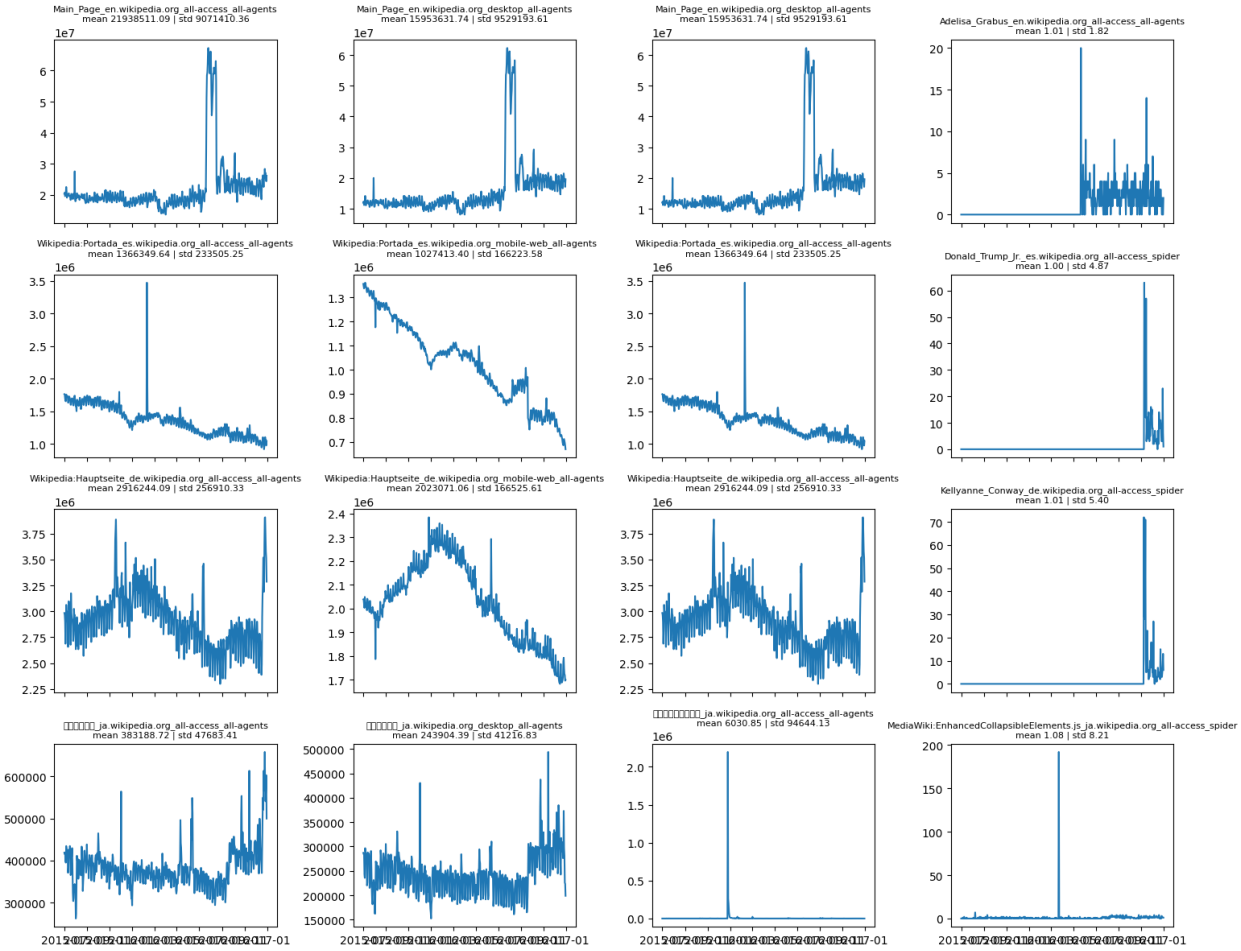
```
# pick representative pages to inspect deeply
# selection strategy: top by mean, high-variance, and random low-
traffic
def pick_pages_for_lang(df, lang, top_k=2, var_k=1, rand_k=1,
seed=42):
    grp = df[df['Language']==lang]
    top = grp.sort_values('mean_views', ascending=False).head(top_k)
['Page'].tolist()
    high_var = grp.sort_values('std_views',
ascending=False).head(var_k)['Page'].tolist()
    low = grp.sort_values('mean_views', ascending=True).head(rand_k)
['Page'].tolist()
    return list(top + high_var + low)  # order is not unique
    # return list(dict.fromkeys(top + high_var + low))  # keep order
unique
```

```
sample_pages = []
for lang in top_langs:
    sample_pages += pick_pages_for_lang(train_clean, lang, top_k=2,
var_k=1, rand_k=1)

print("Sample pages to inspect across languages (for time series
plots):")
print(sample_pages)

# # plot them
# for page in sample_pages:
#     row = train_clean[train_clean['Page']==page].iloc[0]
#     s = row[date_cols].astype(float)
#     s.index = pd.to_datetime(date_cols)
#     plt.figure(figsize=(12,3))
#     plt.plot(s.index, s.values)
#     plt.title(f"{page} — mean {row['mean_views']:.2f} | std
{row['std_views']:.2f}")
#     plt.show()
```

```
Sample pages to inspect across languages (for time series plots):
['Main_Page_en.wikipedia.org_all-access_all-agents',
'Main_Page_en.wikipedia.org_desktop_all-agents',
'Main_Page_en.wikipedia.org_desktop_all-agents',
'Adelisa_Grabus_en.wikipedia.org_all-access_all-agents',
'Wikipedia:Portada_es.wikipedia.org_all-access_all-agents',
'Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents',
'Wikipedia:Portada_es.wikipedia.org_all-access_all-agents',
'Donald_Trump_Jr._es.wikipedia.org_all-access_spider',
'Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents',
'Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents',
'Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents',
'Kellyanne_Conway_de.wikipedia.org_all-access_spider', 'メインページ
_ja.wikipedia.org_all-access_all-agents', 'メインページ
_ja.wikipedia.org_desktop_all-agents', 'キングオブコメディ
_ja.wikipedia.org_all-access_all-agents',
'MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-
access_spider']
```

```
import math

n = len(sample_pages)
cols = 4
rows = math.ceil(n / cols)

fig, axes = plt.subplots(rows, cols, figsize=(15, rows*3),
sharex=True)
axes = axes.flatten()

for i, page in enumerate(sample_pages):
```

```
    row = train_clean[train_clean['Page']==page].iloc[0]
    s = row[date_cols].astype(float)
    s.index = pd.to_datetime(date_cols)

    axes[i].plot(s.index, s.values)
    axes[i].set_title(f"{page}\nmean {row['mean_views']:.2f} | std
{row['std_views']:.2f}", fontsize=8)

for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])  # remove unused subplots

plt.tight_layout()
plt.show()
```




```
# # Save critical CSVs for reproducibility
# lang_agg.to_csv('lang_agg_summary.csv')
# for lang, df in top_by_lang.items():
#     df.to_csv(f"top_pages_{lang}.csv", index=False)
# lang_daily.to_csv('lang_daily.csv')
```

Actionable Insights (3–5 points)

1. **Language traffic dominance**
   - English Wikipedia drives the majority of global traffic (~100M+ daily views).
   - Spanish (~18–19M), German (~15–17M), and Japanese (~15–17M) form the second tier.
   - Campaign targeting should prioritize **English first**, then Spanish/German/Japanese.
2. **Device usage split**
   - Spanish & Japanese: **mobile-web dominates**.
   - English: **balanced** desktop vs. mobile.
   - German: leans **desktop-first**.
   - Non-English campaigns should be **mobile-first**, while English/German can support cross-device.
3. **Weekday patterns**
   - English peaks **Mon–Tue**, dips **Fri–Sat**.
   - Japanese rises into **weekends (Sat–Sun)**.
   - German & Spanish relatively **stable**, but slight midweek dips.
   - Campaign launch timing should align with **local audience rhythms**.
4. **Main page dominance**
   - Language homepages (Main_Page_en, Wikipedia:Portada_es, etc.) absorb **disproportionate traffic**.
   - Treat these as **traffic hubs** and monitor separately from article traffic.

Visualization Takeaways (3 key inferences)

1. **Language dominance**
   - EN: ~100–114M avg daily views
   - ES: ~18–19M
   - DE: ~15–17M
   - JA: ~15–17M
   - ➡ English has ~**6× more traffic** than the next language.
2. **Device split**

- Spanish & Japanese: **mobile-heavy**.
- English: **balanced split**.
- German: **desktop-skewed**.
- ➡ Clear implication for **ad/channel strategy**.

3. **Weekday effect**
   - English → early-week heavy (Mon–Tue), lighter Fri–Sat.
   - Japanese → weekend uplift.
   - German/Spanish → steady, mild midweek dips.
   - ➡ Campaign timing should **adapt per-language**.

Save train_clean to file

```
save_path = "/content/drive/My Drive/Wikipedia/"


train_clean.to_csv(save_path + "train_clean.csv", index=False)
train_clean.to_parquet(save_path + "train_clean.parquet", index=False)
```

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

AdEase Time-Series Modeling

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

Open AdEase Time-Series Modeling Notebook in Google Colab

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

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

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\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)
```

```
39
```

```
# 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:

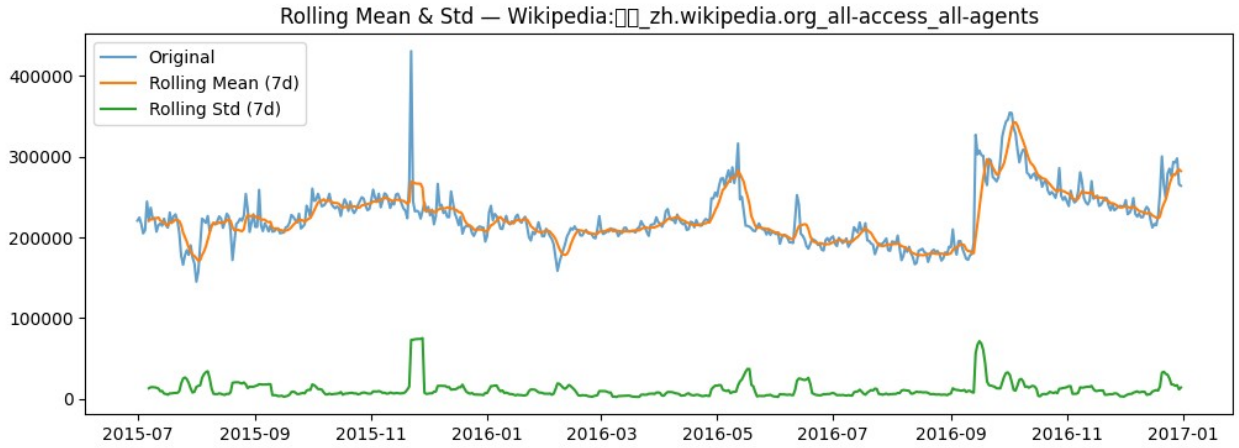
- Perform **ADF test** to check stationarity.
- If non-stationary, apply transformations:
  - Differencing
  - Log transform
  - Seasonal differencing
- Decompose series into **trend, seasonality, residuals**.

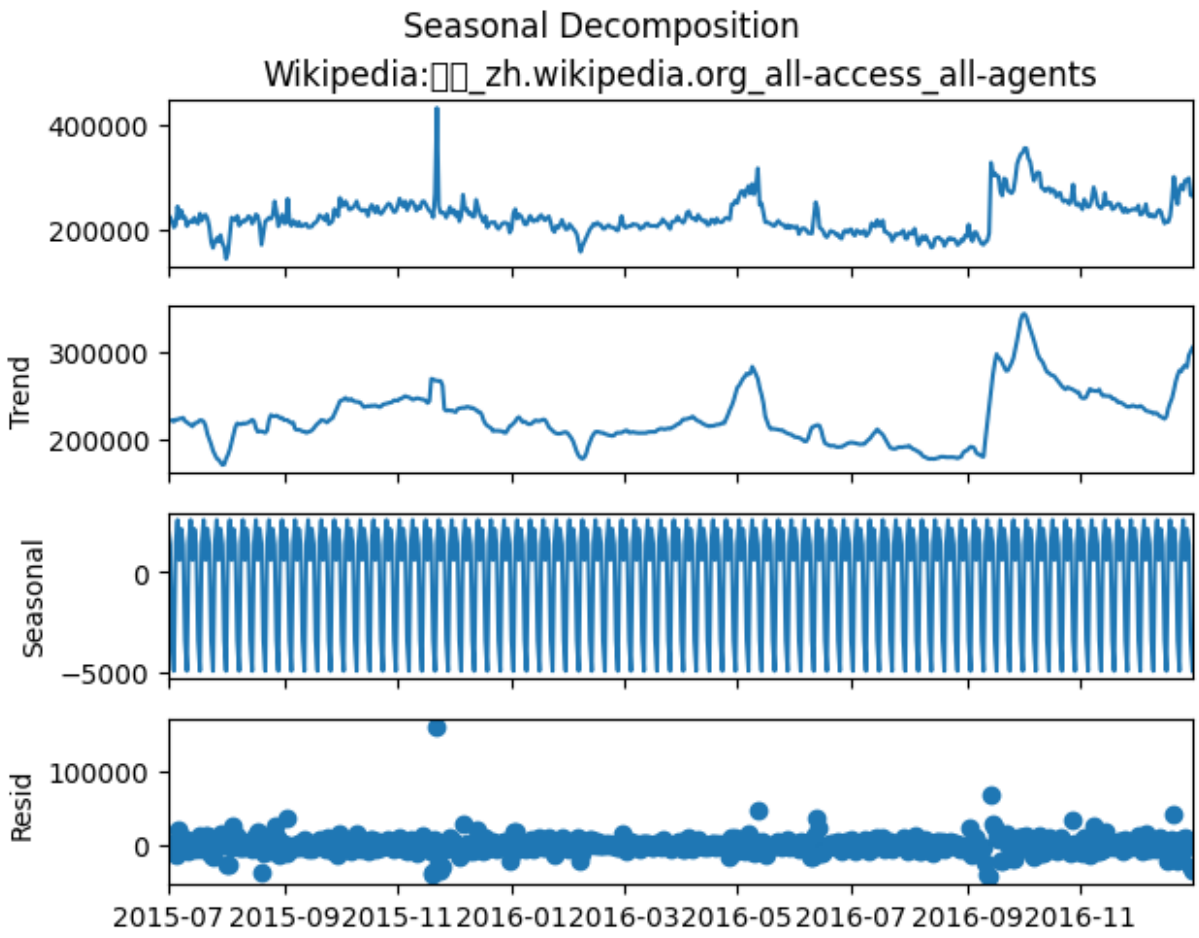
```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

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

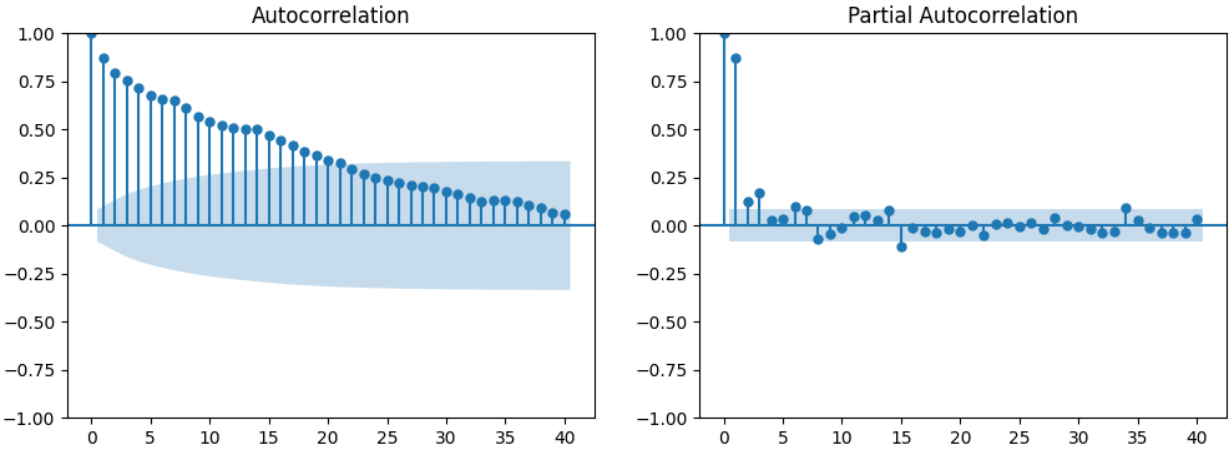
```
    res = adfuller(s)
    is_stationary = res[1] < alpha
    return {
        'adf_stat': res[0],
        'pvalue': res[1],
        'stationary': is_stationary,
        'crit': res[4]
    }

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"\n Analyzing 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()
```

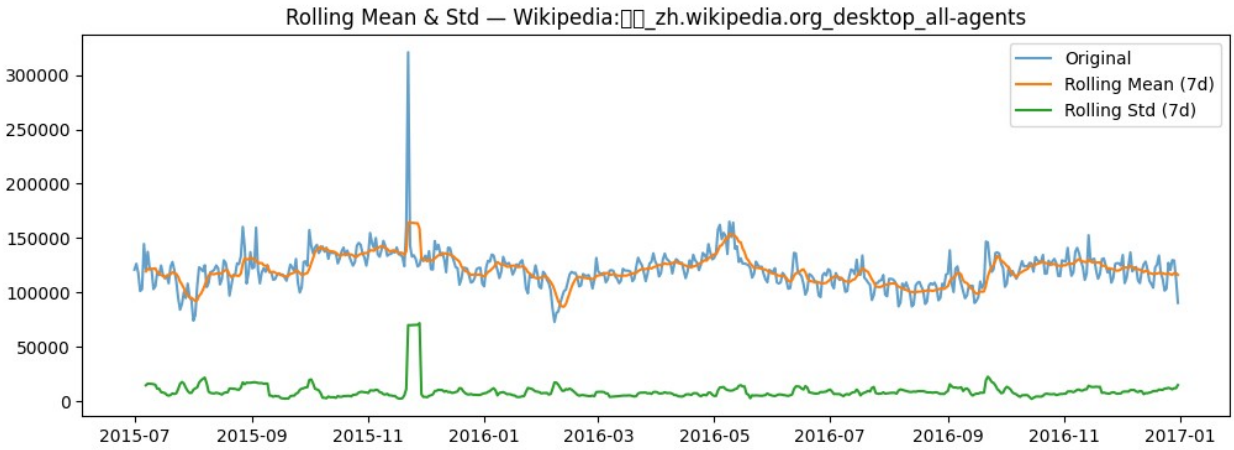
```
 Analyzing page: Wikipedia:首页_zh.wikipedia.org_all-access_all-agents
ADF Statistic: -2.657 | p-value: 0.0817 | Stationary: False
```

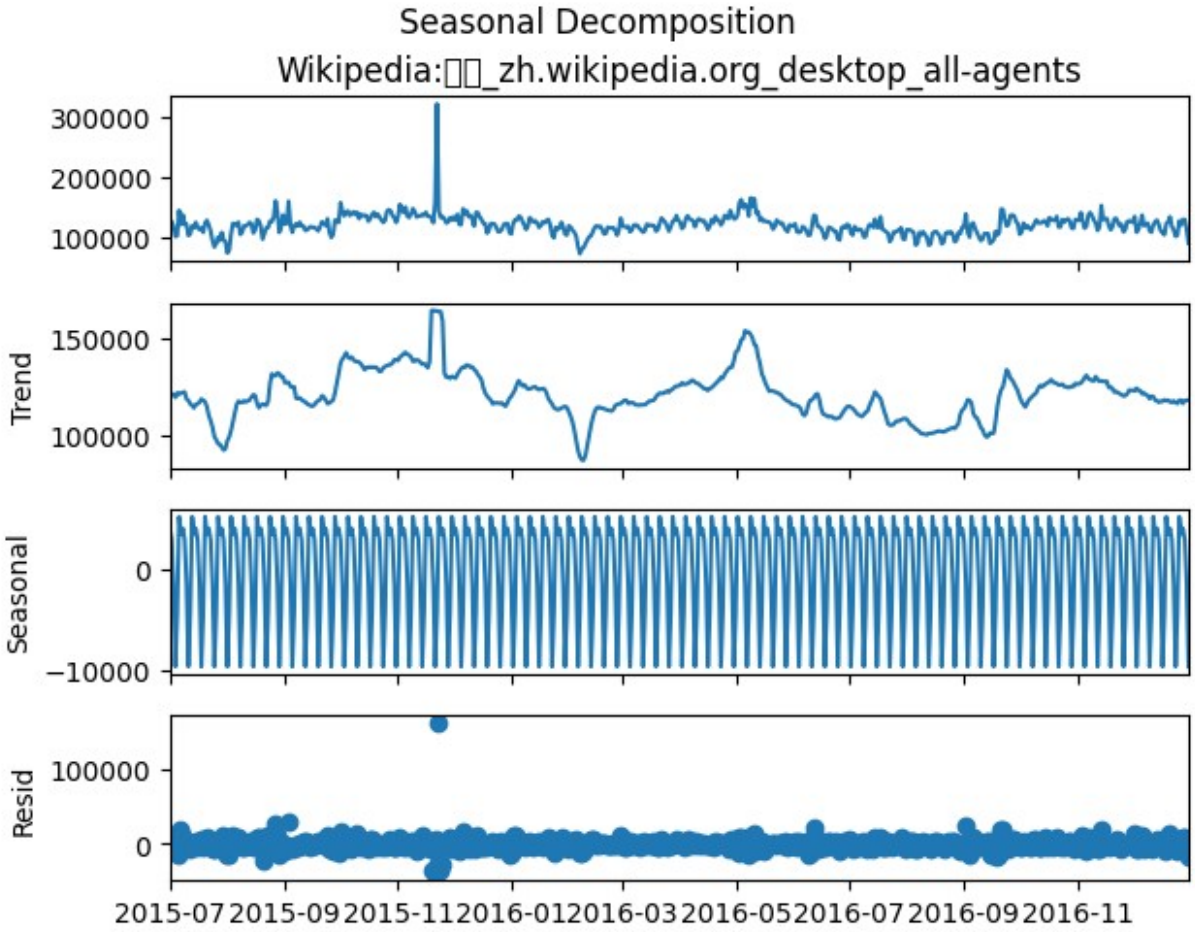


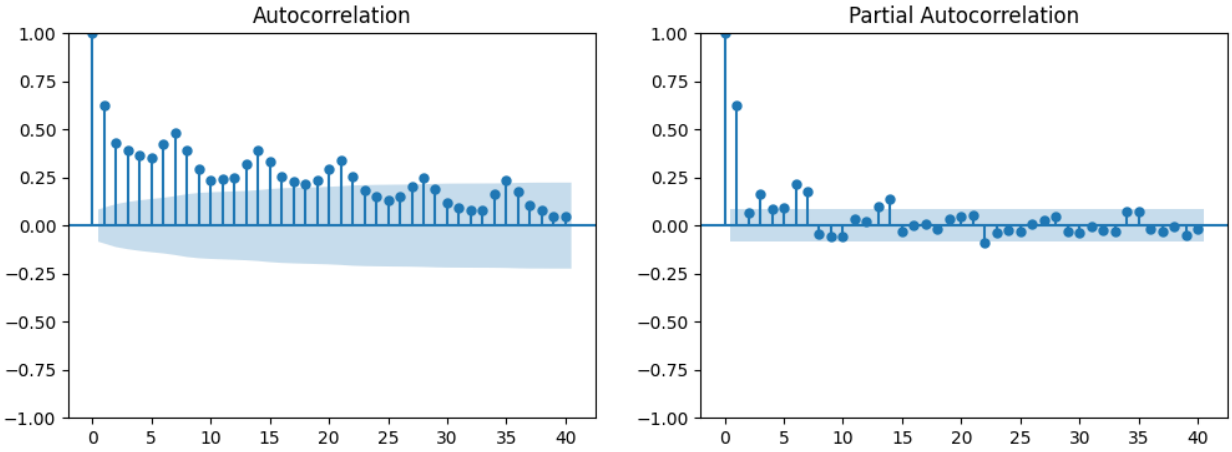




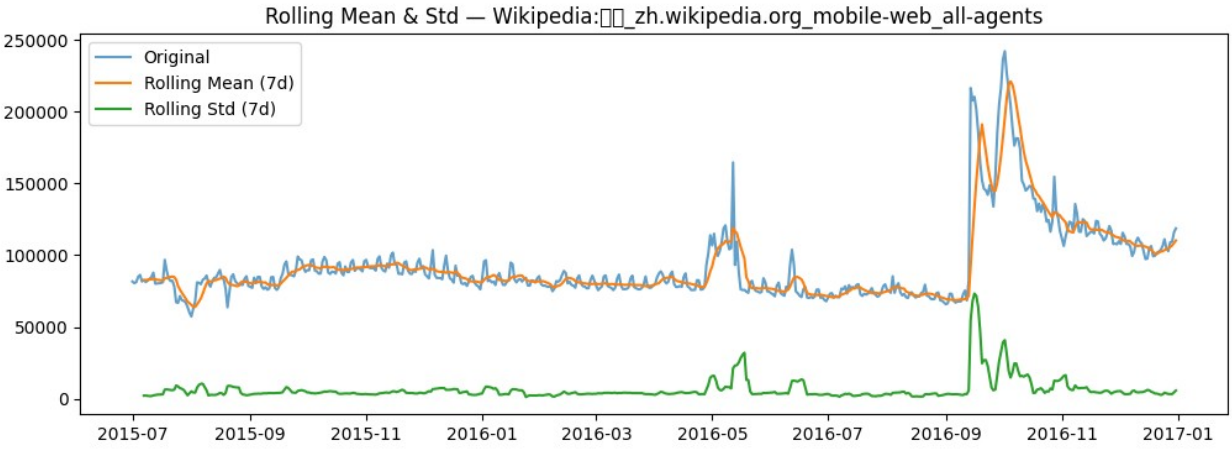

```
□ Analyzing page: Wikipedia:首页_zh.wikipedia.org_desktop_all-agents
ADF Statistic: -2.802 | p-value: 0.0579 | Stationary: False
```

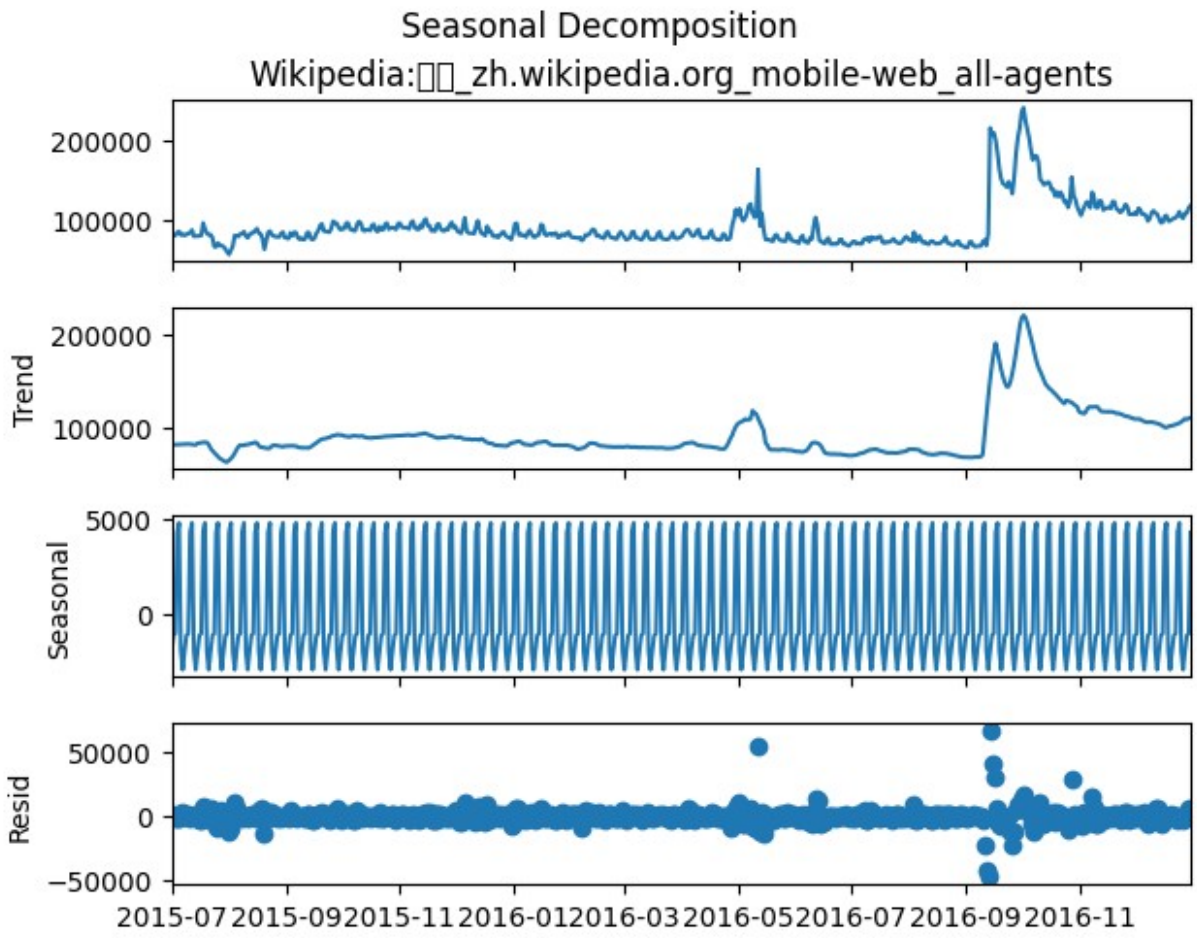

Rolling Mean & Std — Wikipedia:□□_zh.wikipedia.org_desktop_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
300000
250000
200000
150000
100000
50000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

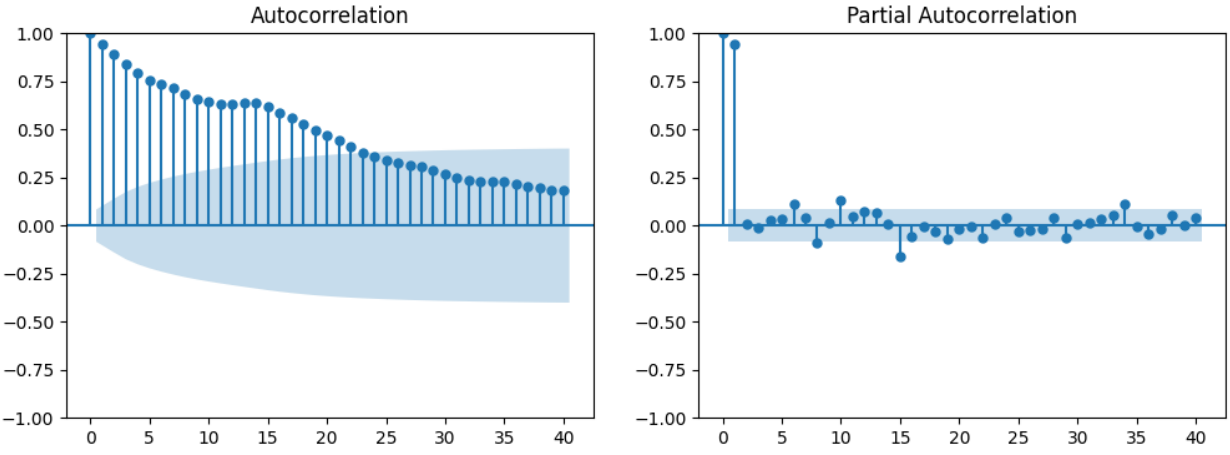

Seasonal Decomposition
Wikipedia:□□_zh.wikipedia.org_desktop_all-agents
300000
200000
100000
Trend
150000
100000
Seasonal
0
−10000
Resid
100000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



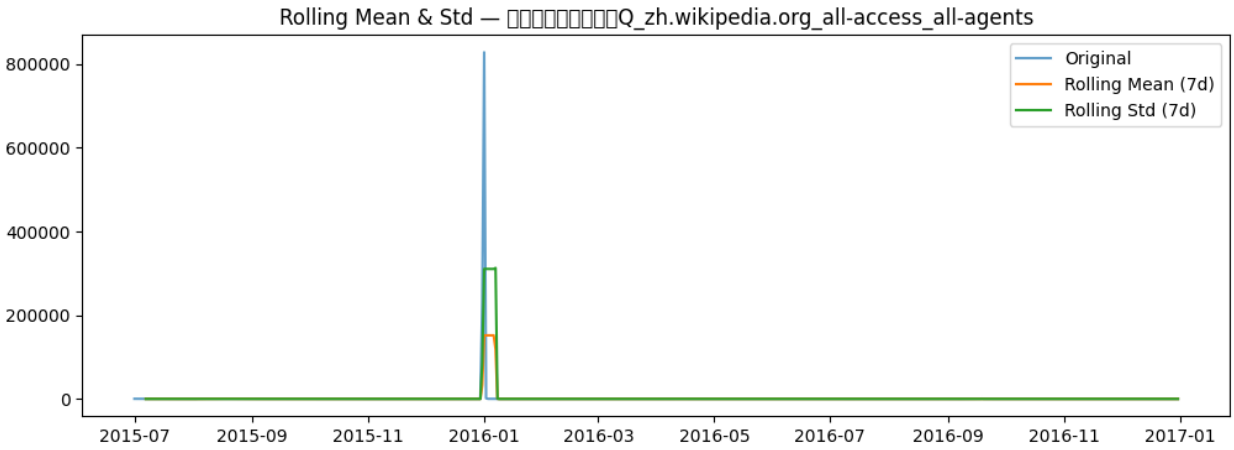

```
 Analyzing page: Wikipedia:首页_zh.wikipedia.org_mobile-web_all-agents
ADF Statistic: -2.579 | p-value: 0.0974 | Stationary: False
```

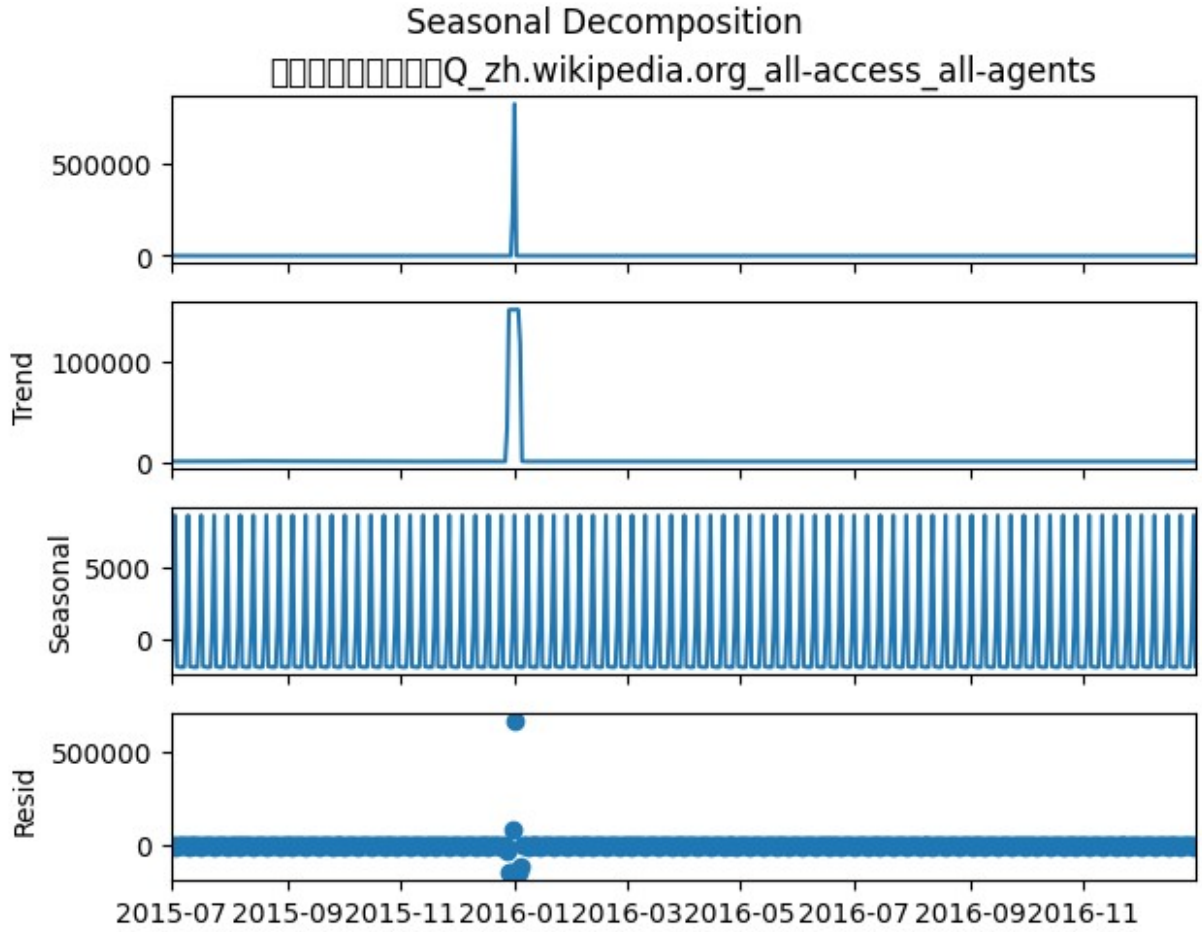


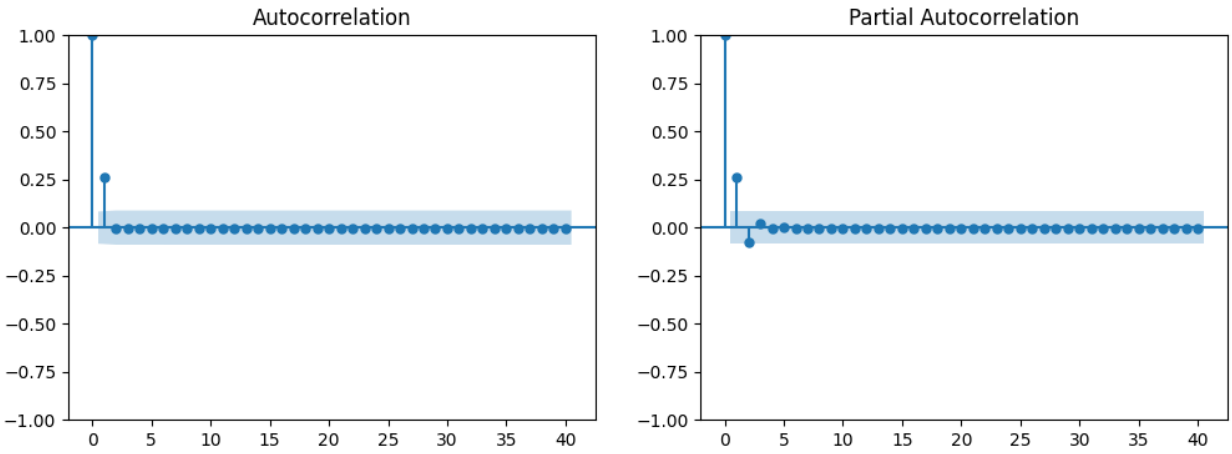




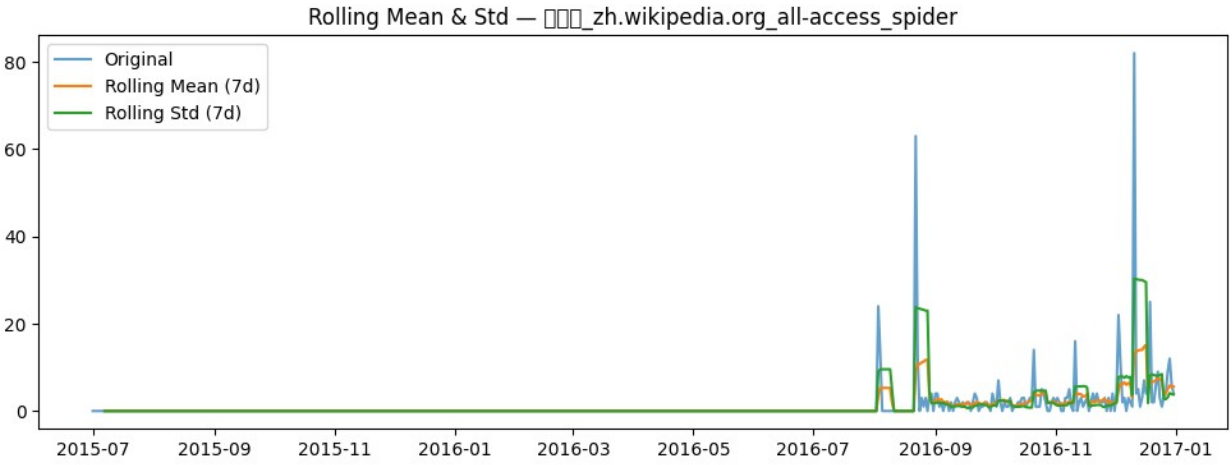

```
□ Analyzing page: 福音戰士新劇場版：Q_zh.wikipedia.org_all-access_all-
agents
ADF Statistic: -15.307 | p-value: 0.0000 | Stationary: True
```

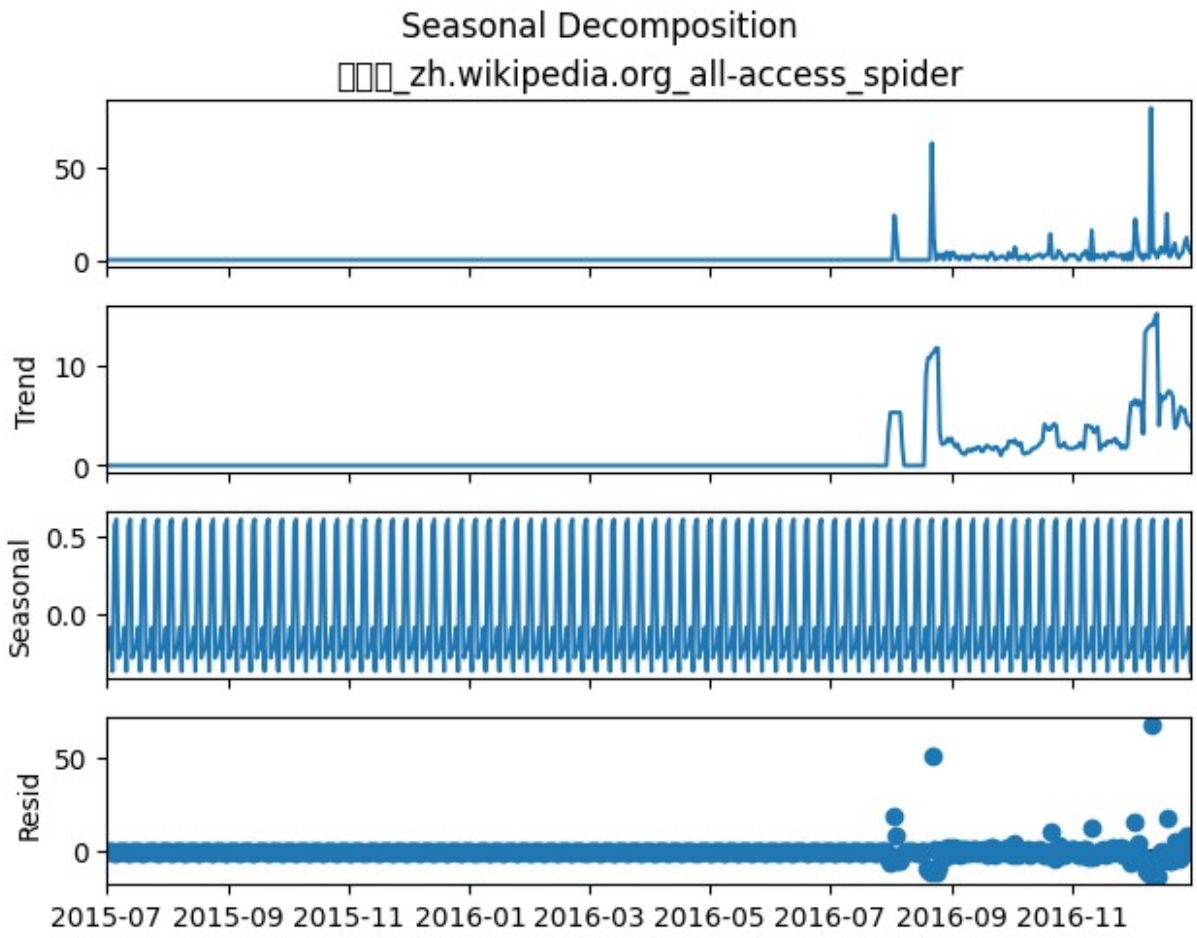

Rolling Mean & Std — □□□□□□□□□□Q_zh.wikipedia.org_all-access_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
800000
600000
400000
200000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

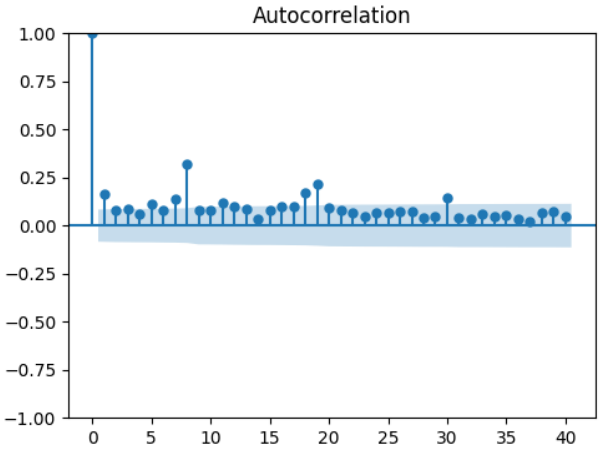

Seasonal Decomposition
□□□□□□□□□□Q_zh.wikipedia.org_all-access_all-agents
500000
0
Trend
100000
0
Seasonal
5000
0
Resid
500000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11

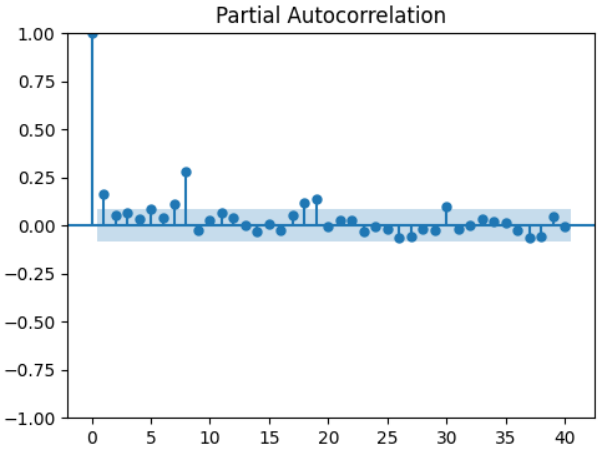



```
 Analyzing page: 洪晨穎_zh.wikipedia.org_all-access_spider
ADF Statistic: -1.966 | p-value: 0.3019 | Stationary: False
```

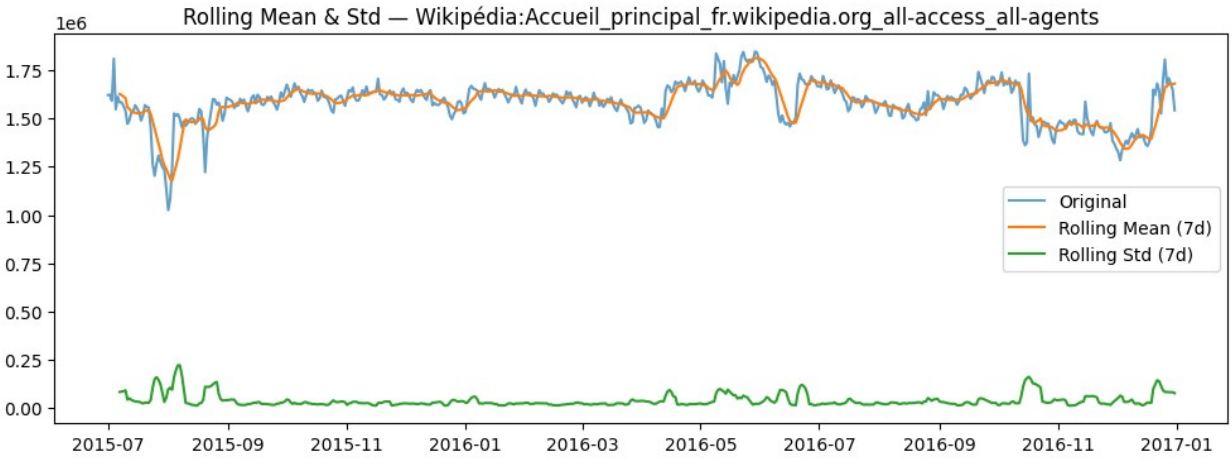


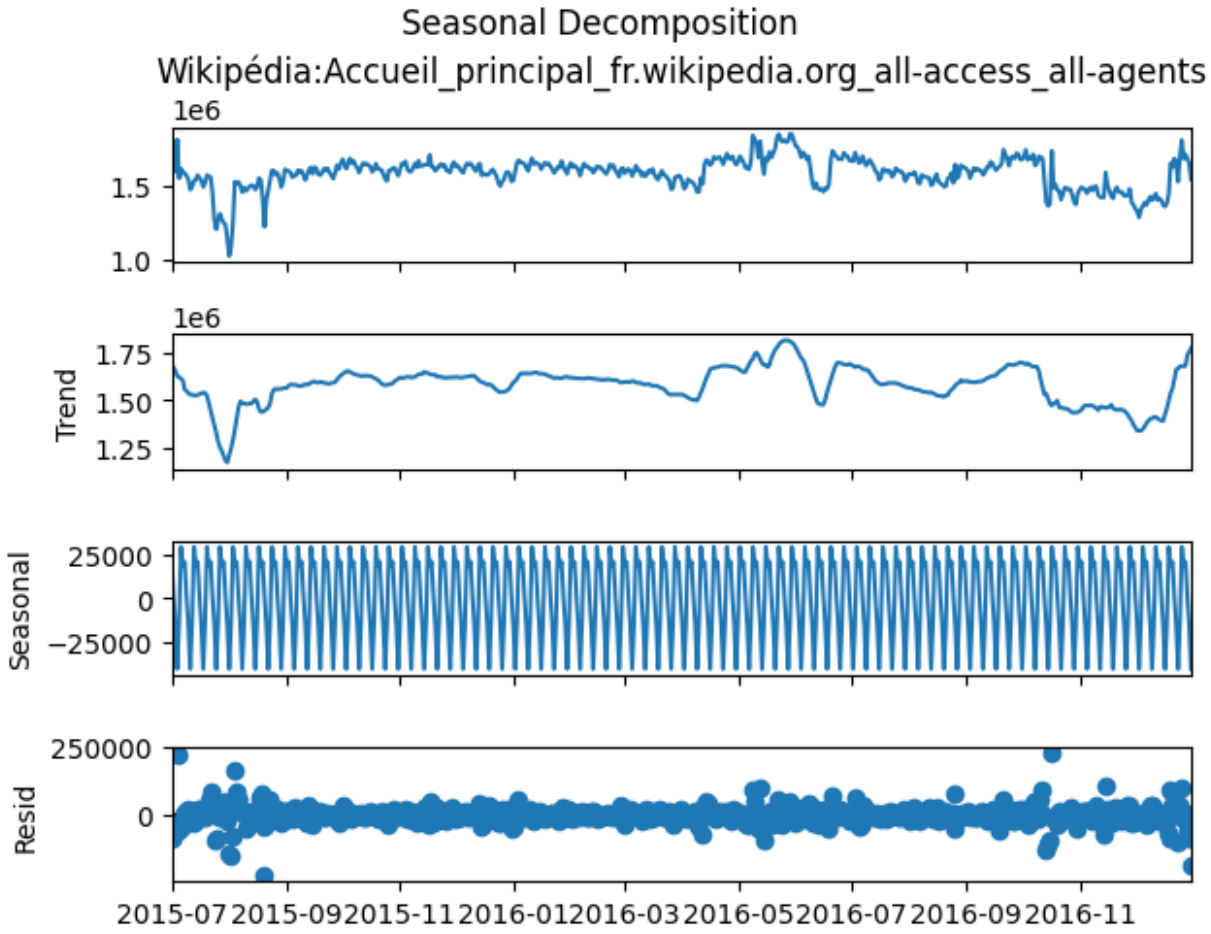


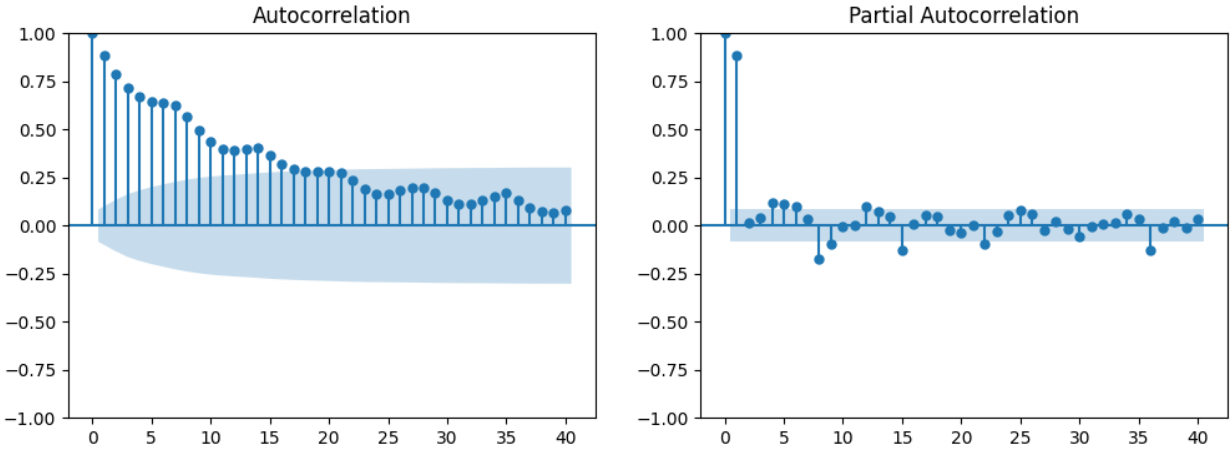




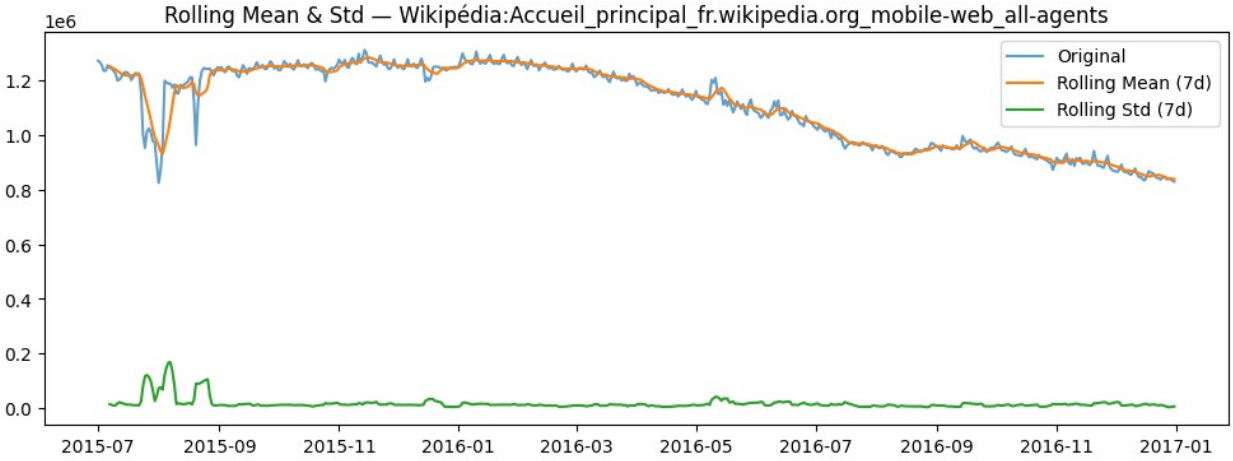

```
□ Analyzing page: Wikipédia:Accueil_principal_fr.wikipedia.org_all-
access_all-agents
ADF Statistic: -3.557 | p-value: 0.0066 | Stationary: True
```

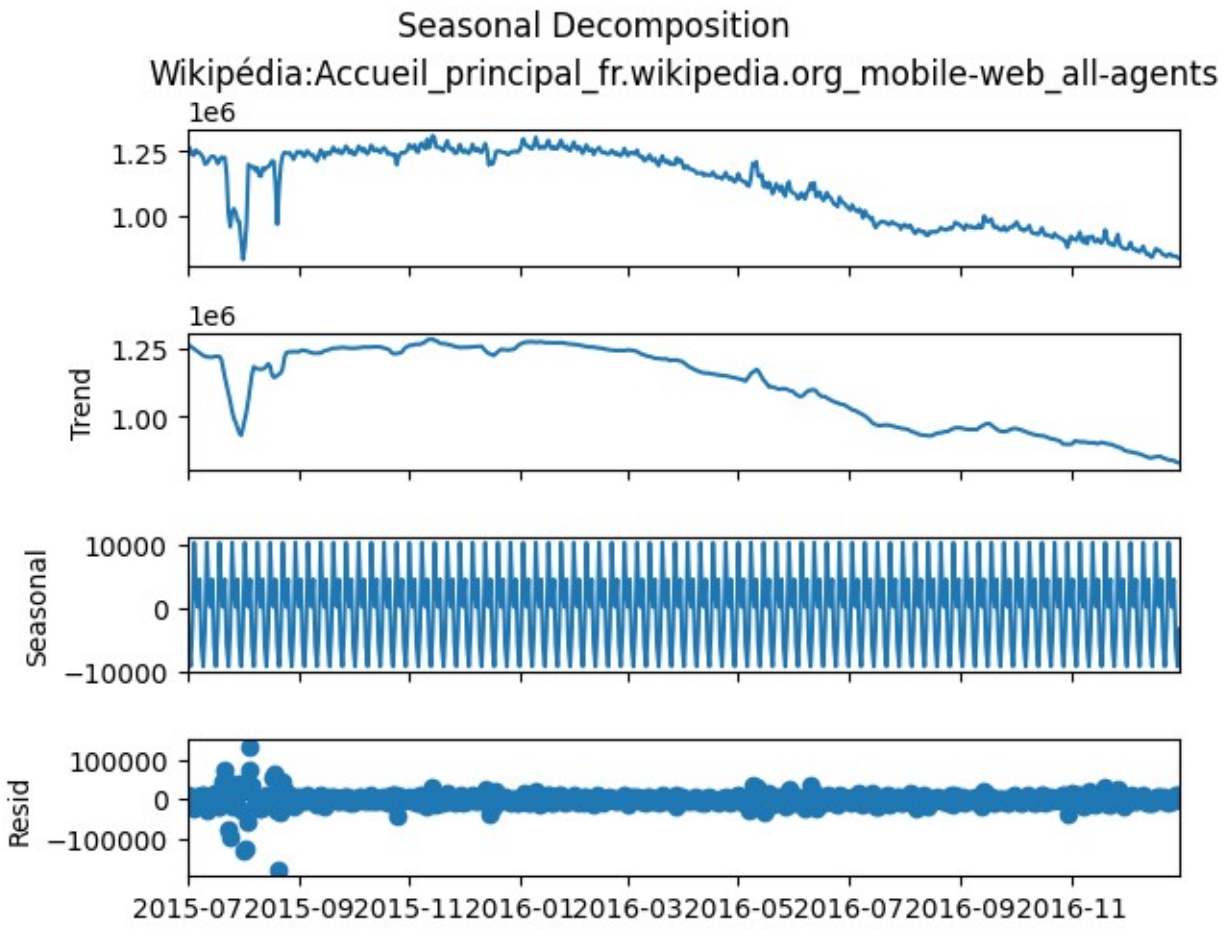

Rolling Mean & Std — Wikipédia:Accueil_principal_fr.wikipedia.org_all-access_all-agents
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Original
Rolling Mean (7d)
Rolling Std (7d)
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

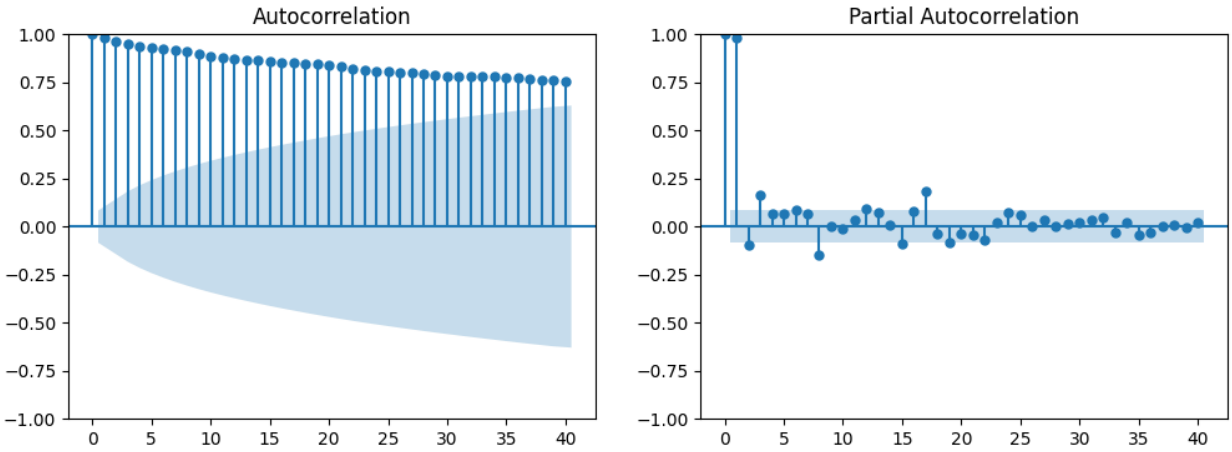

Seasonal Decomposition
Wikipédia:Accueil_principal_fr.wikipedia.org_all-access_all-agents
1e6
1.5
1.0
1e6
Trend
1.75
1.50
1.25
Seasonal
25000
0
−25000
Resid
250000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



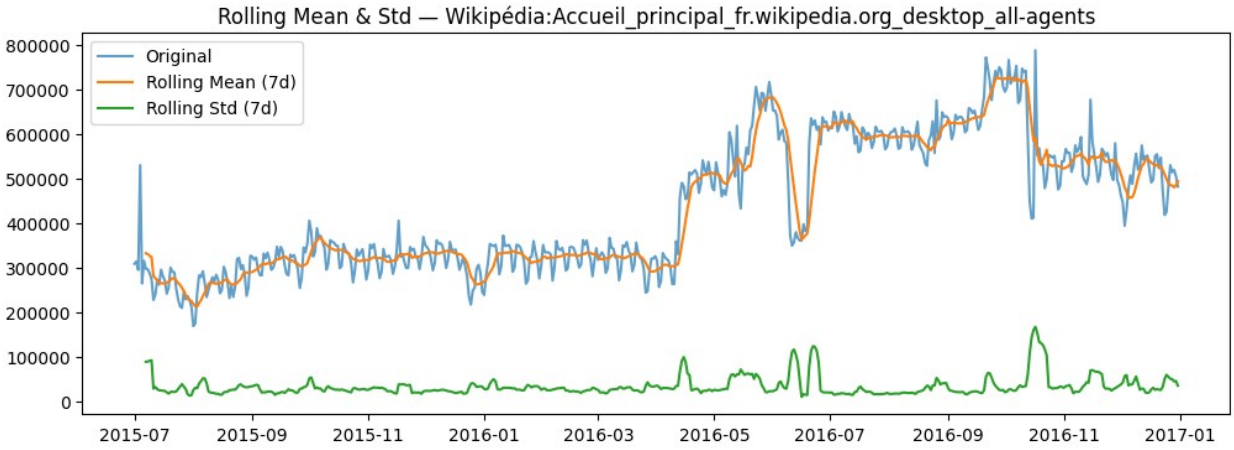

```
🔍 Analyzing page: Wikipédia:Accueil_principal_fr.wikipedia.org_mobile-web_all-agents
ADF Statistic: 0.336 | p-value: 0.9789 | Stationary: False
```

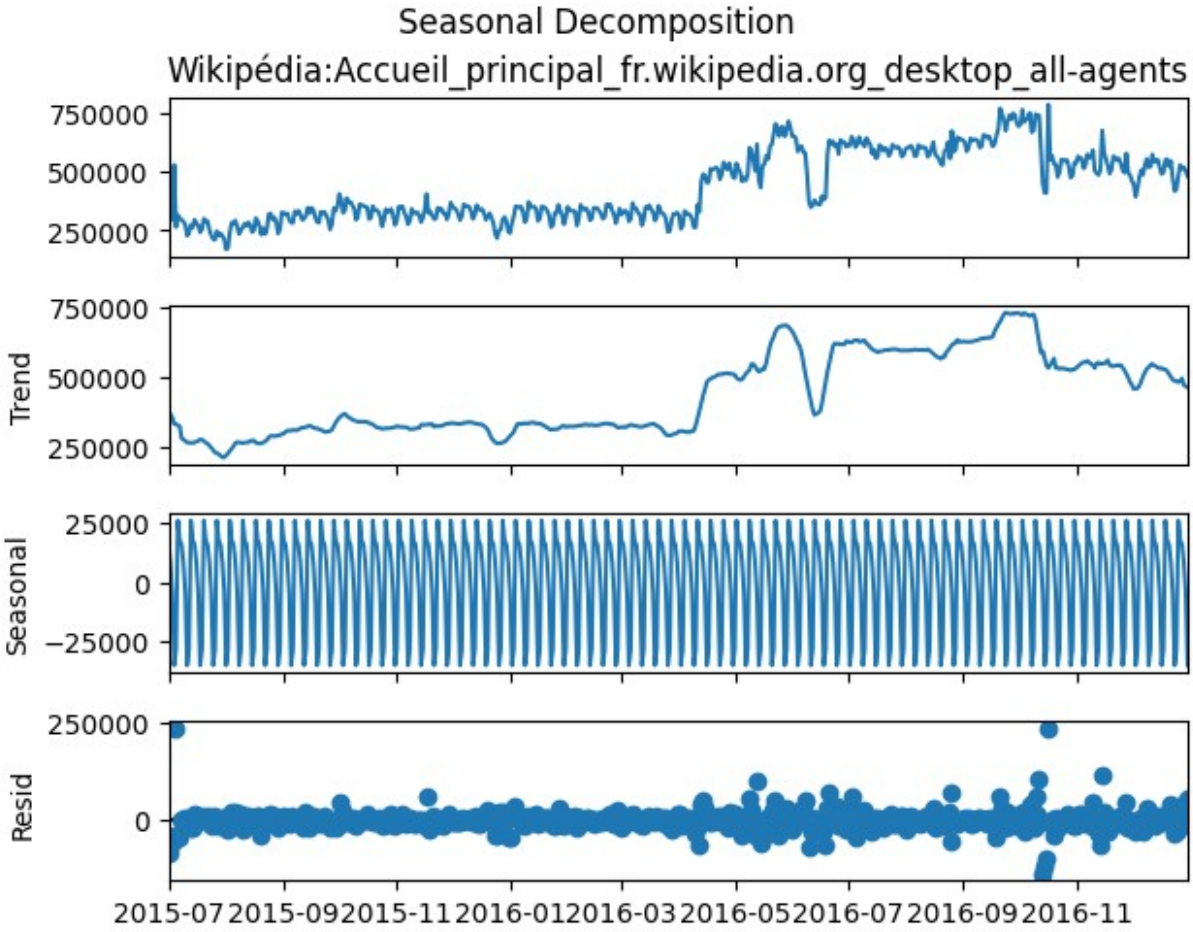


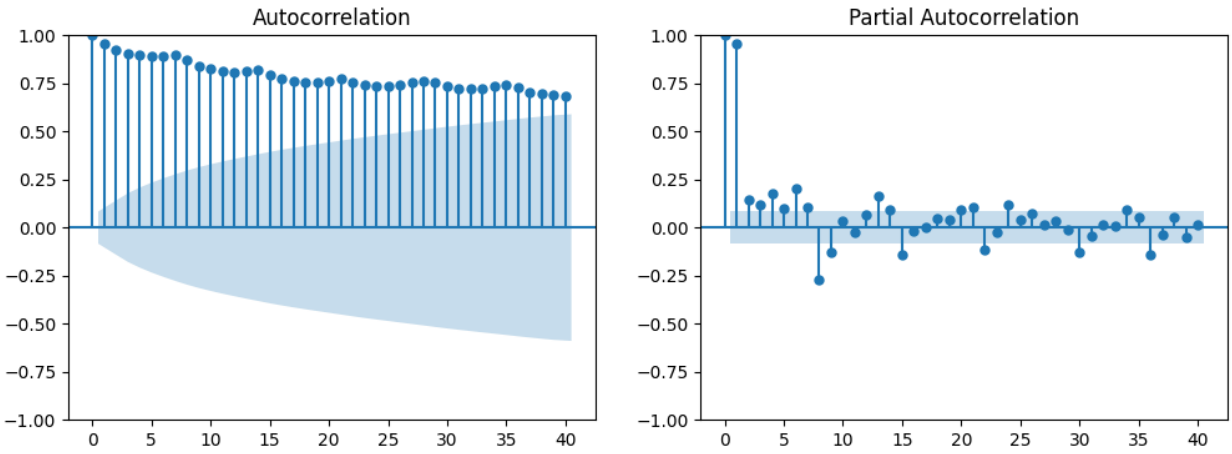




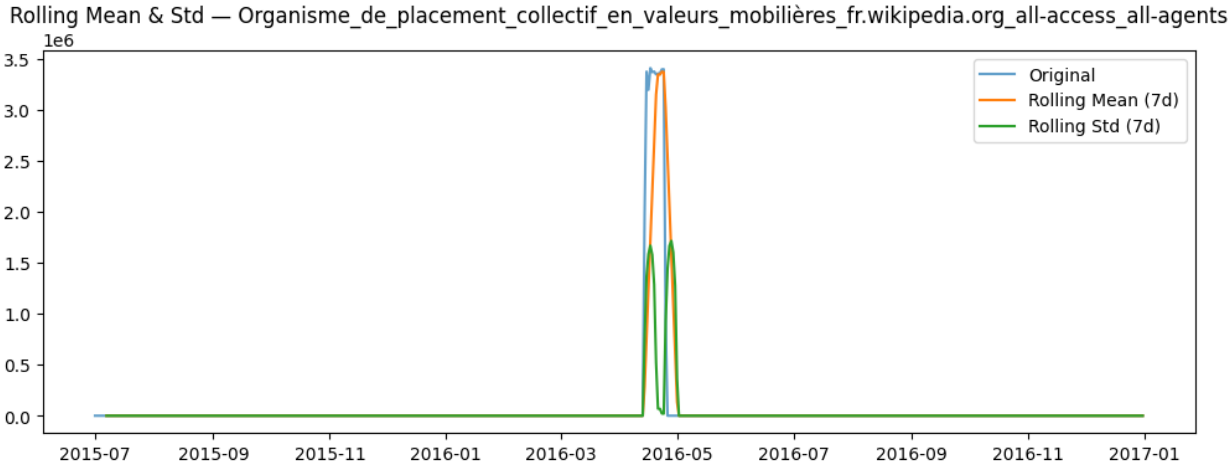

```
🔍 Analyzing page:
Wikipédia:Accueil_principal_fr.wikipedia.org_desktop_all-agents
ADF Statistic: -1.681 | p-value: 0.4409 | Stationary: False
```

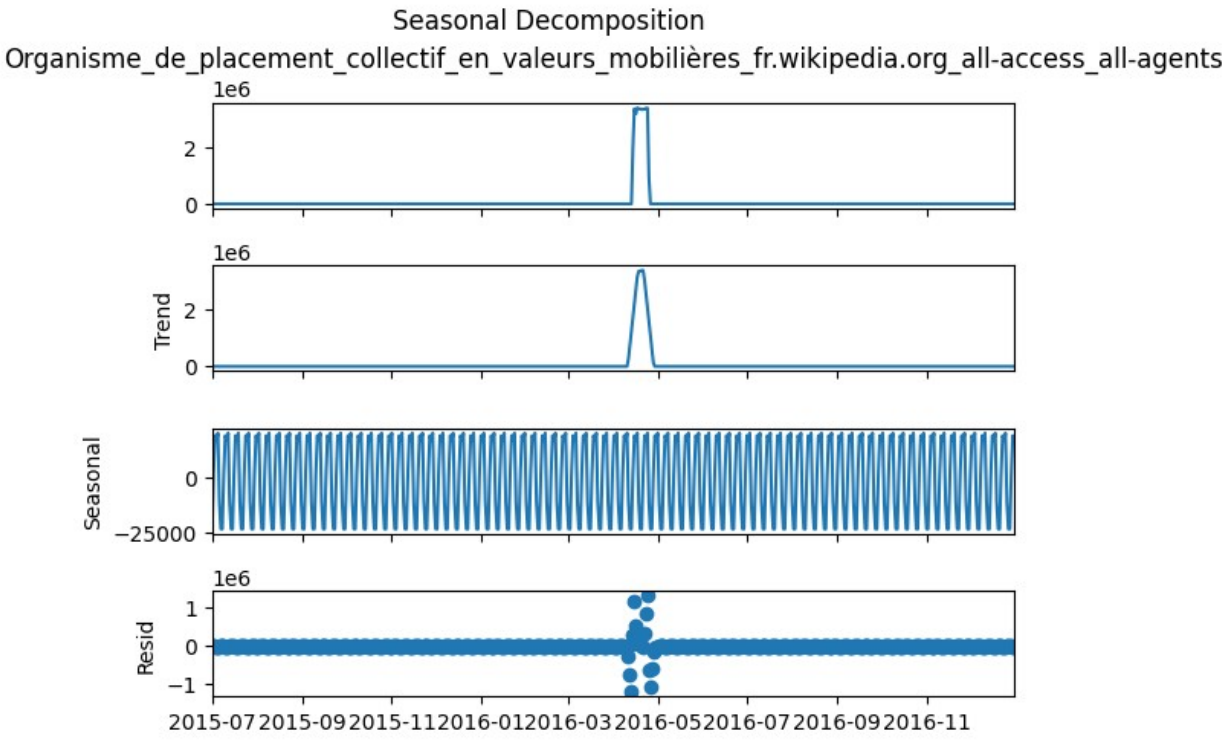

Rolling Mean & Std — Wikipédia:Accueil_principal_fr.wikipedia.org_desktop_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
800000
700000
600000
500000
400000
300000
200000
100000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

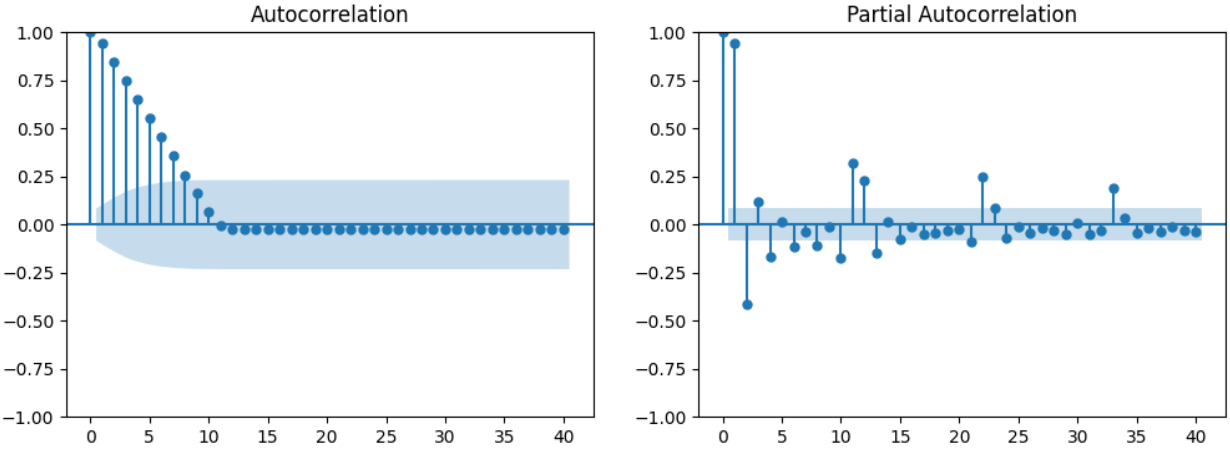

Seasonal Decomposition
Wikipédia:Accueil_principal_fr.wikipedia.org_desktop_all-agents
Trend
Seasonal
Resid
750000
500000
250000
25000
0
−25000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



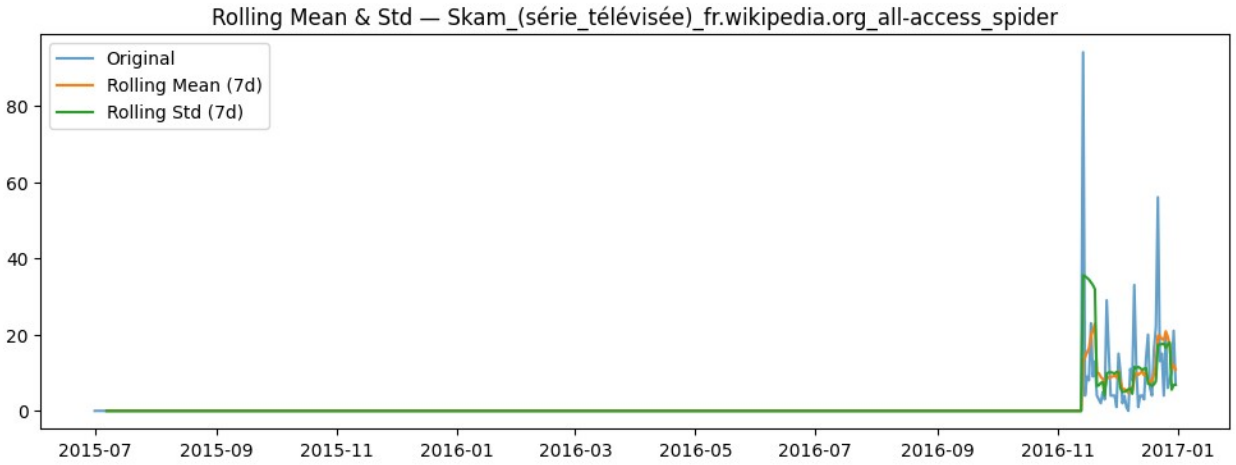

```
🔍 Analyzing page:
Organisme_de_placement_collectif_en_valeurs_mobilières_fr.wikipedia.org_all-access_all-agents
ADF Statistic: -4.409 | p-value: 0.0003 | Stationary: True
```

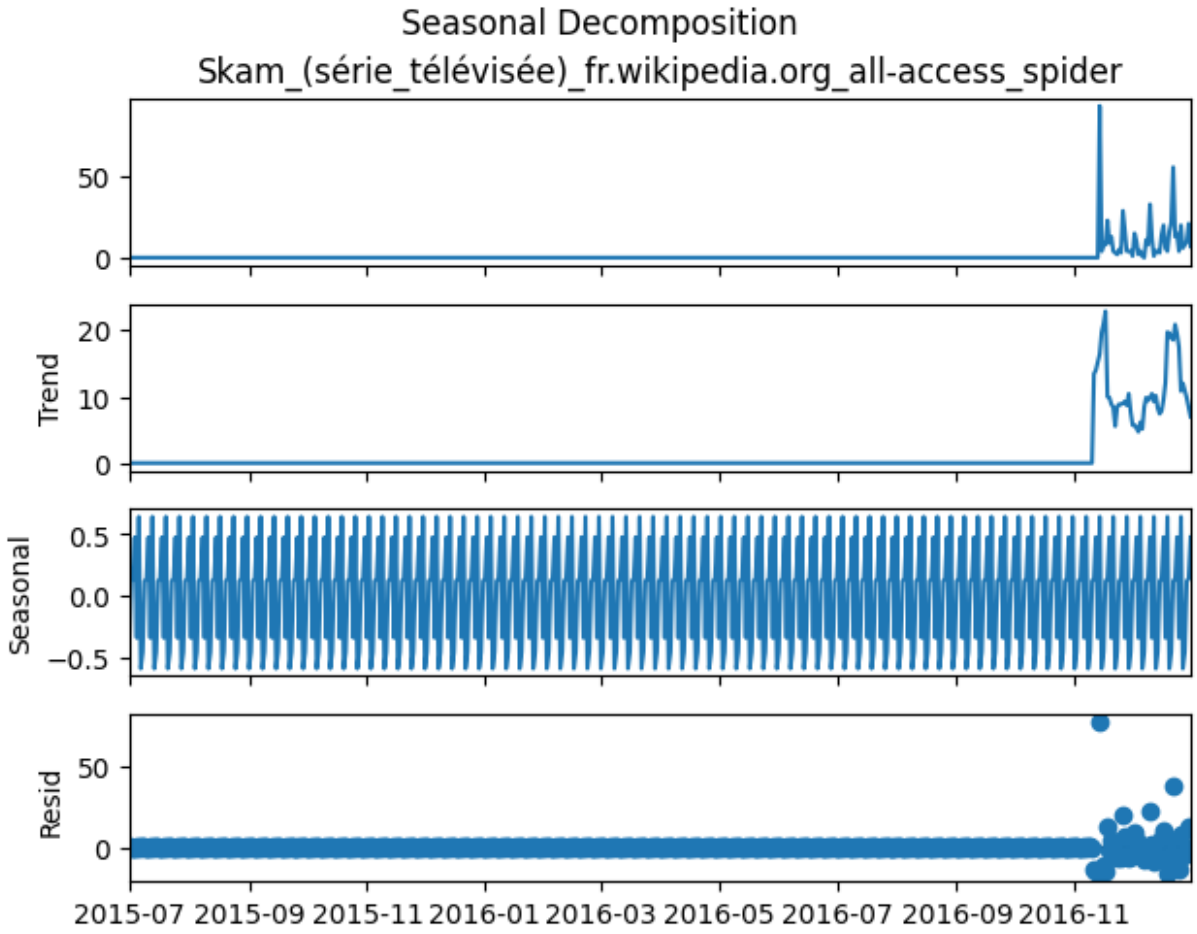


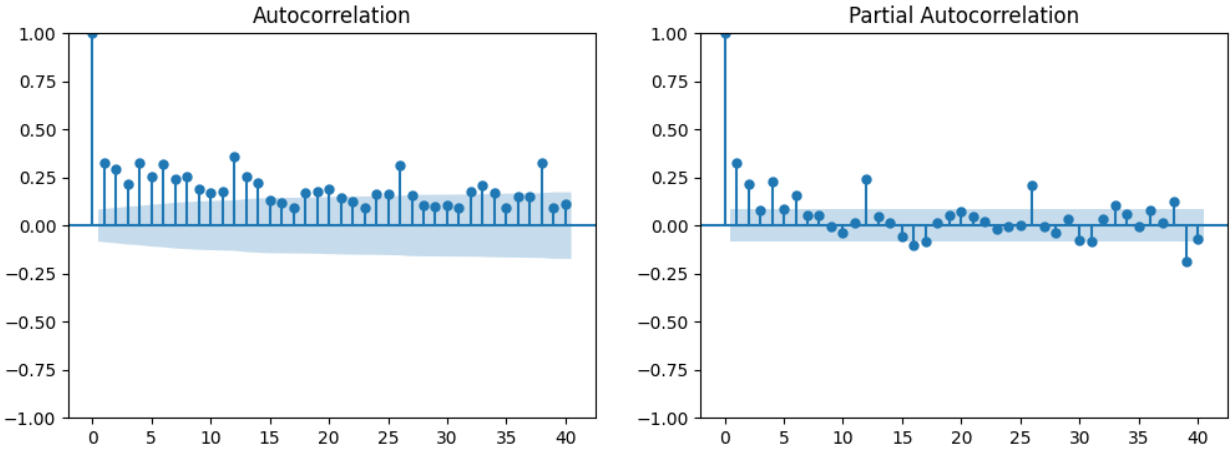




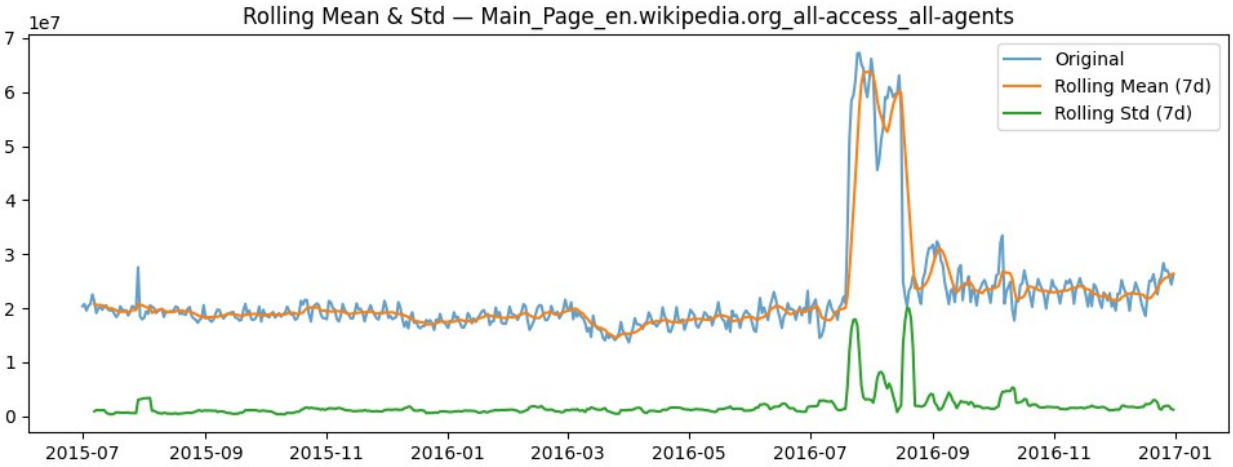

```
🔍 Analyzing page: Skam_(série_télévisée)_fr.wikipedia.org_all-
access_spider
ADF Statistic: -0.369 | p-value: 0.9152 | Stationary: False
```

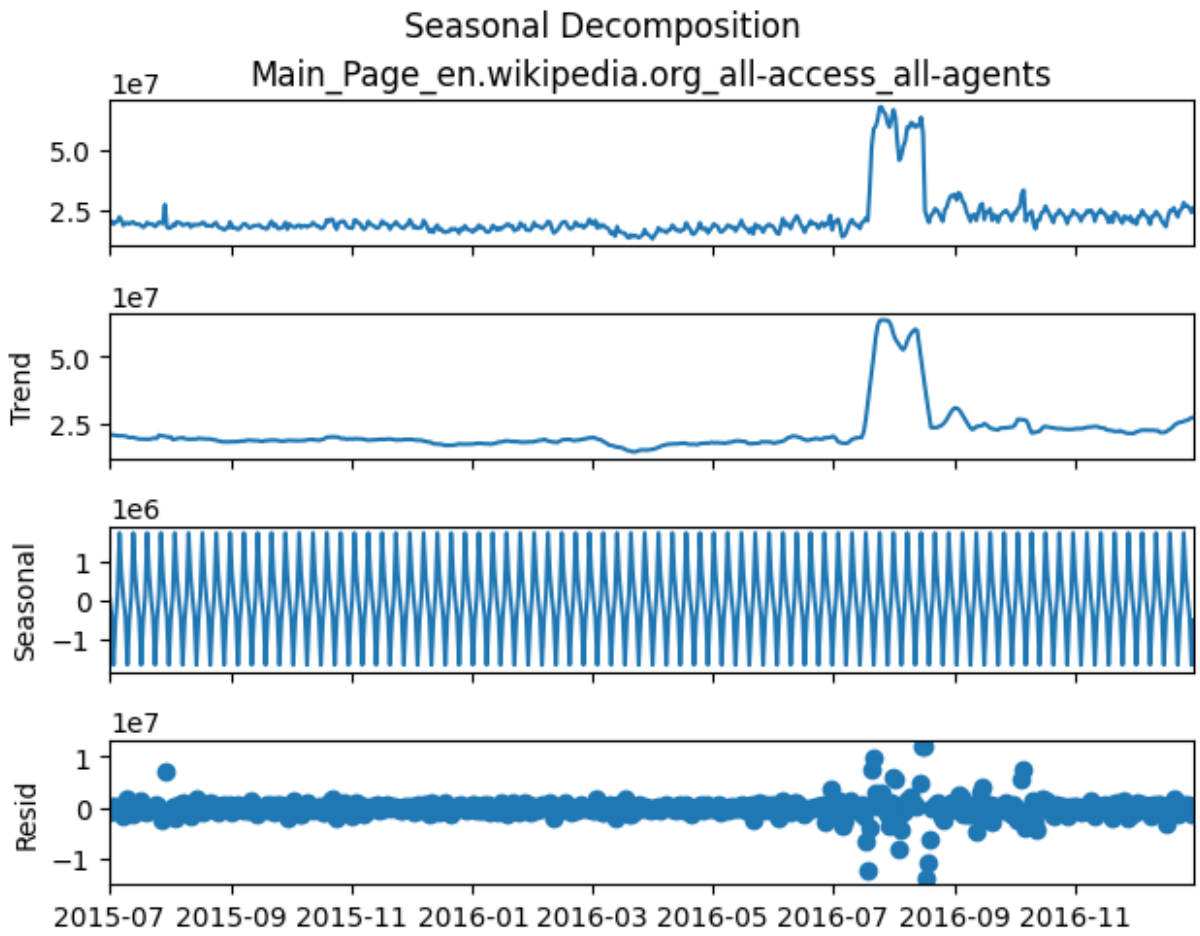

Rolling Mean & Std — Skam_(série_télévisée)_fr.wikipedia.org_all-access_spider
Original
Rolling Mean (7d)
Rolling Std (7d)
80
60
40
20
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

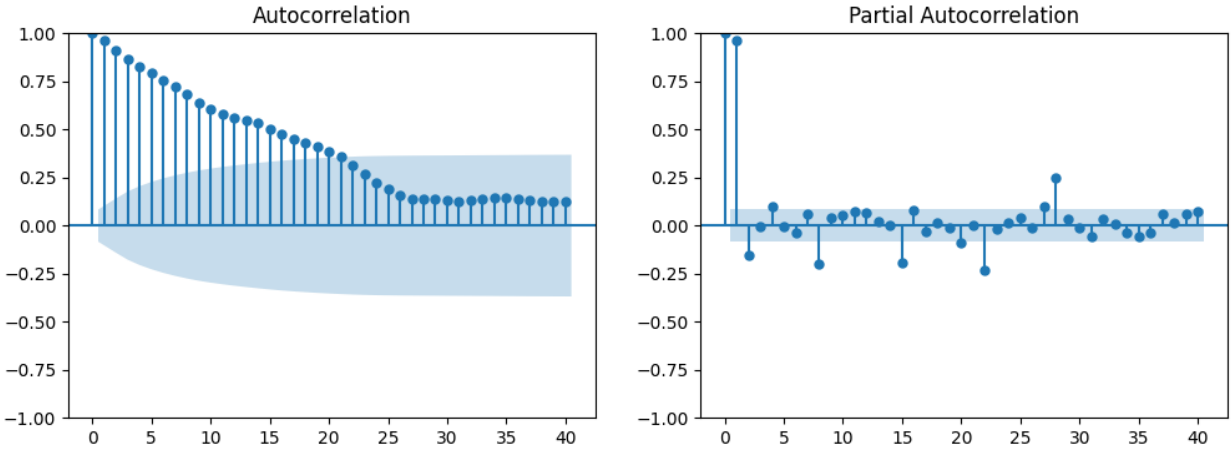

Seasonal Decomposition
Skam_(série_télévisée)_fr.wikipedia.org_all-access_spider
50
0
Trend
20
10
0
Seasonal
0.5
0.0
−0.5
Resid
50
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



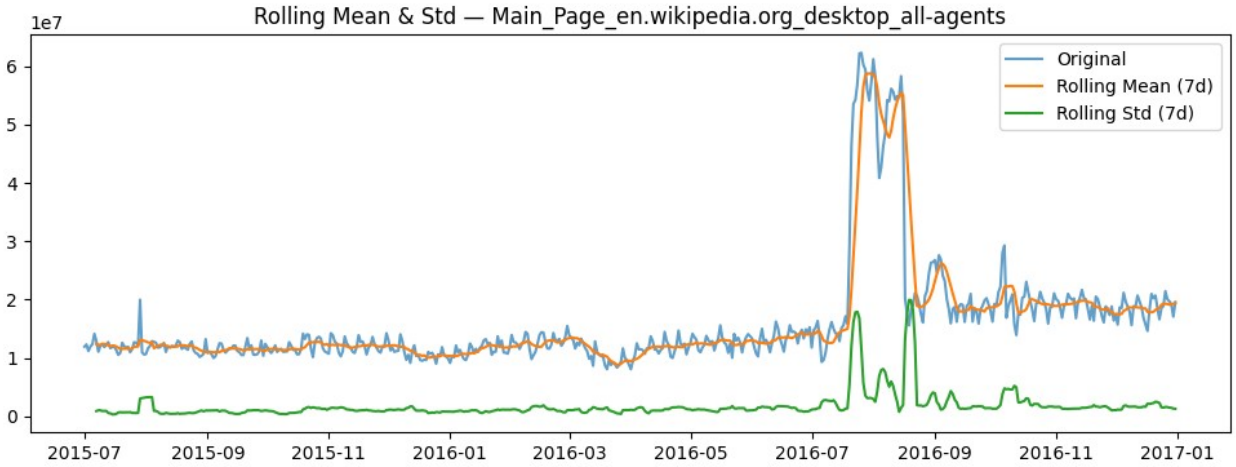

```
🔍 Analyzing page: Main_Page_en.wikipedia.org_all-access_all-agents
ADF Statistic: -2.990 | p-value: 0.0358 | Stationary: True
```

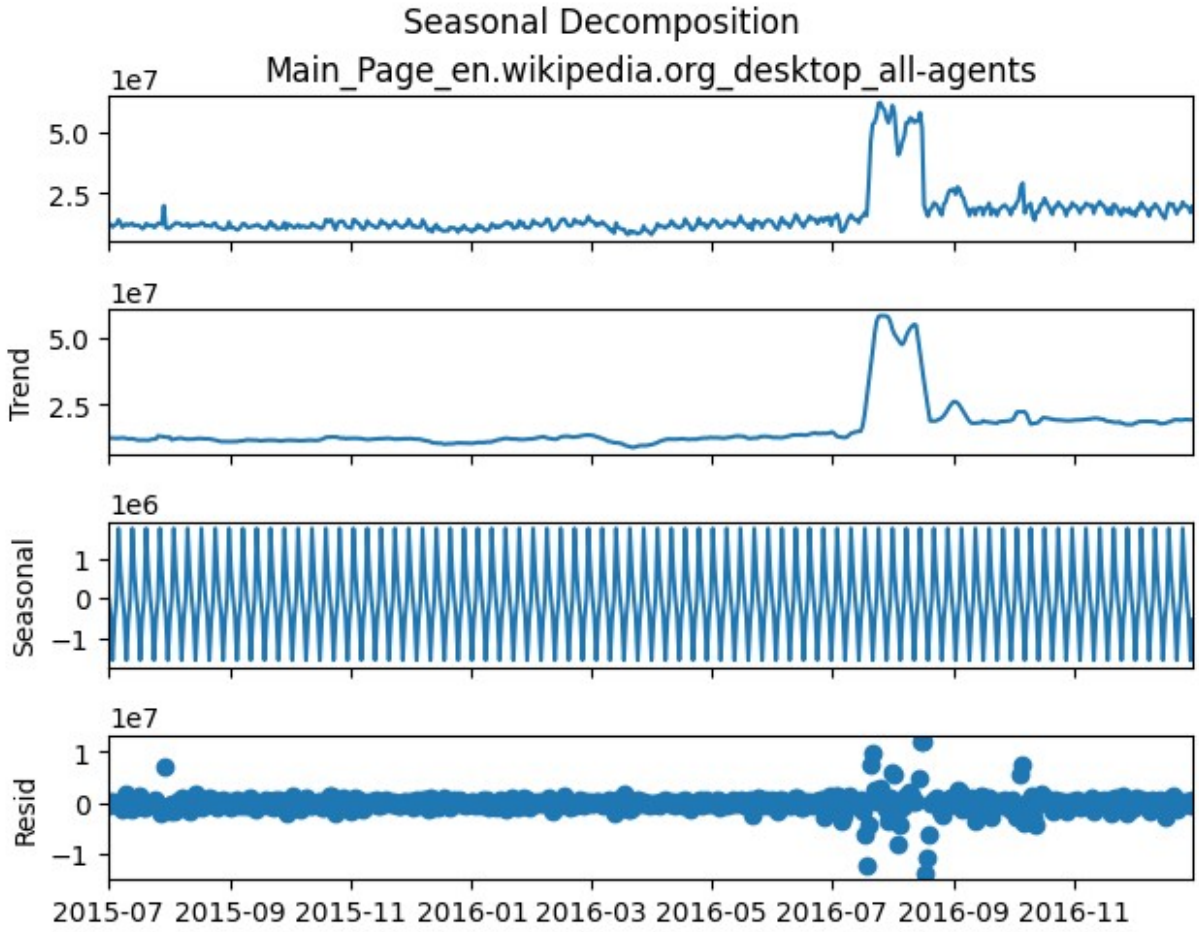


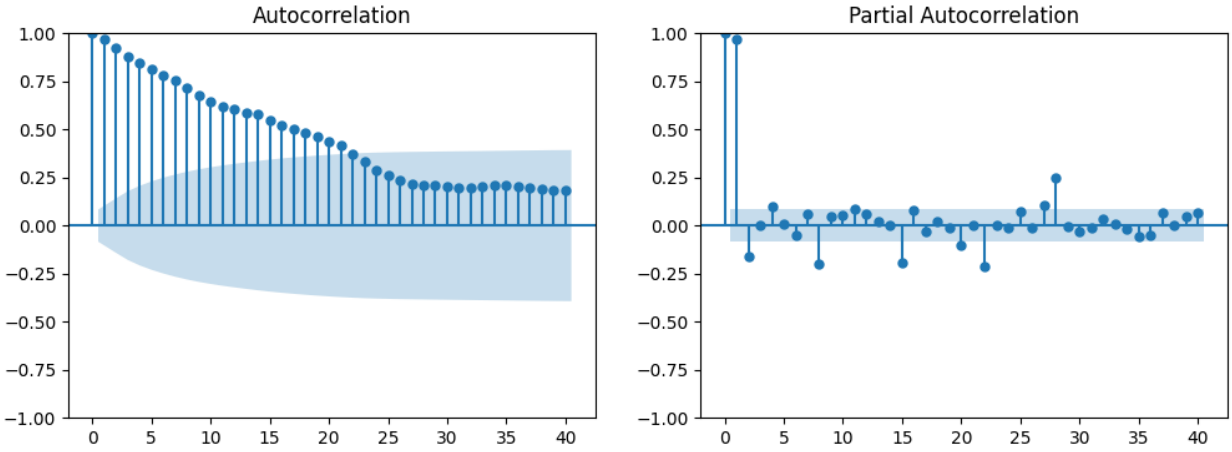




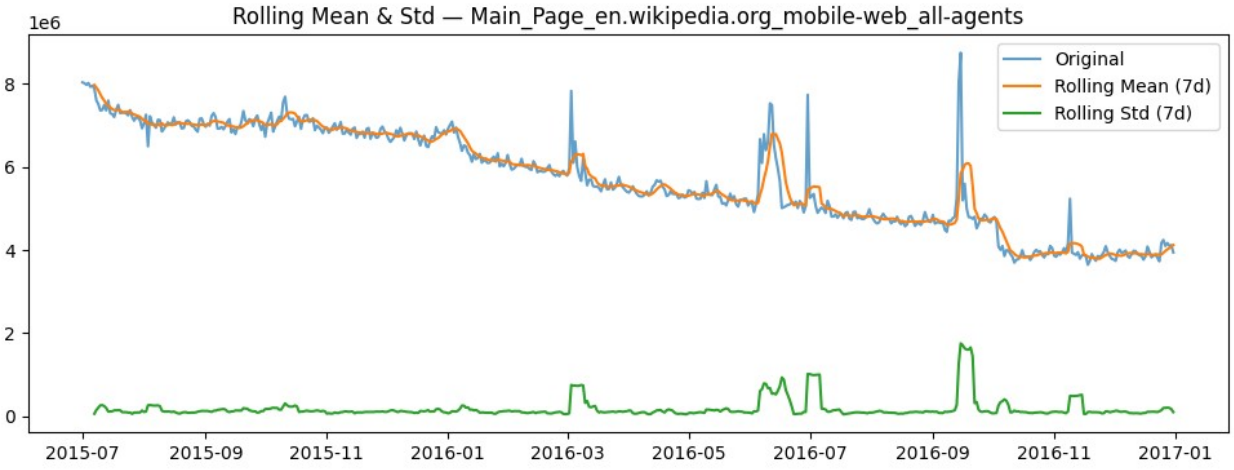

```
 Analyzing page: Main_Page_en.wikipedia.org_desktop_all-agents
ADF Statistic: -2.815 | p-value: 0.0562 | Stationary: False
```

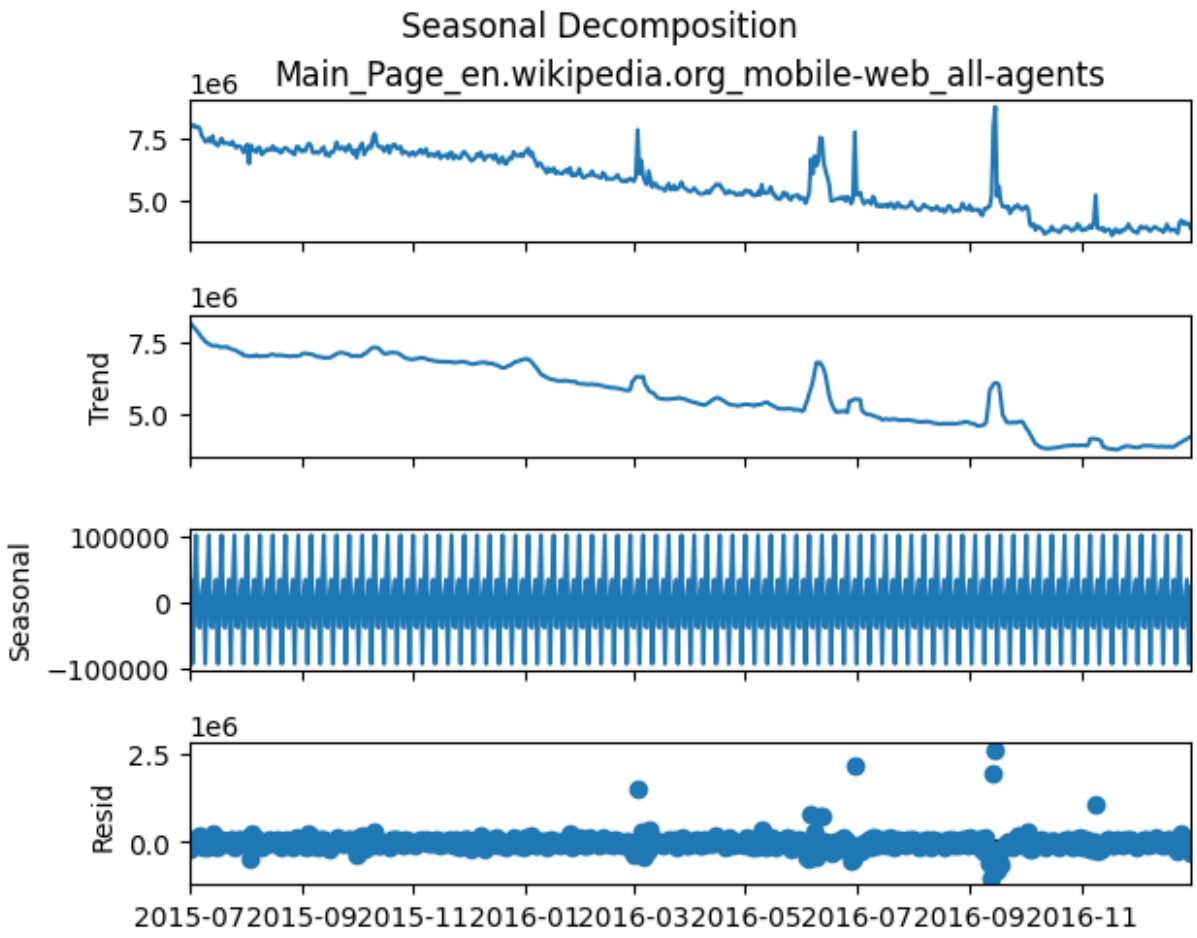

Rolling Mean & Std — Main_Page_en.wikipedia.org_desktop_all-agents
1e7
Original
Rolling Mean (7d)
Rolling Std (7d)
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

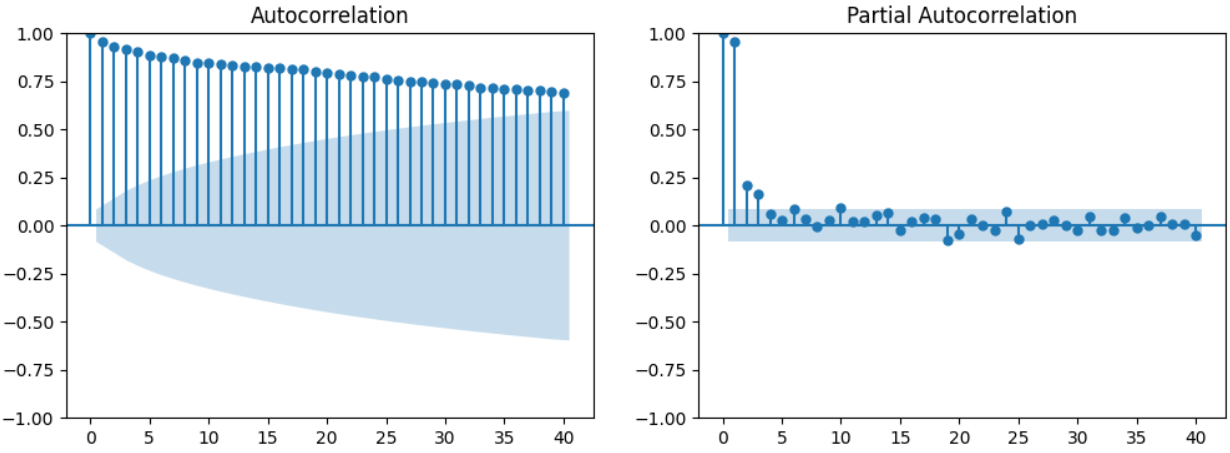

Seasonal Decomposition
Main_Page_en.wikipedia.org_desktop_all-agents
Trend
Seasonal
Resid
1e7
1e6
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



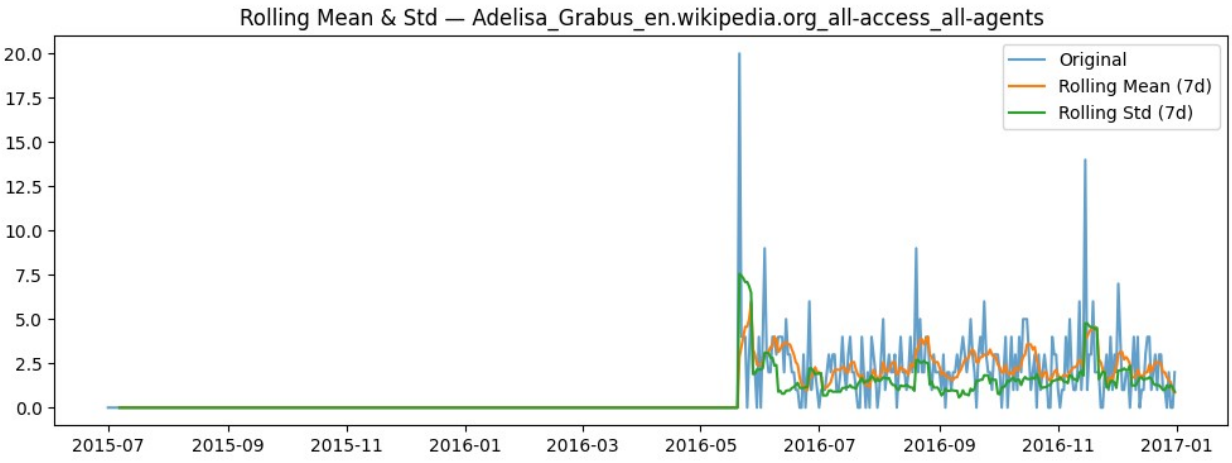

```
🔍 Analyzing page: Main_Page_en.wikipedia.org_mobile-web_all-agents
ADF Statistic: -1.117 | p-value: 0.7082 | Stationary: False
```

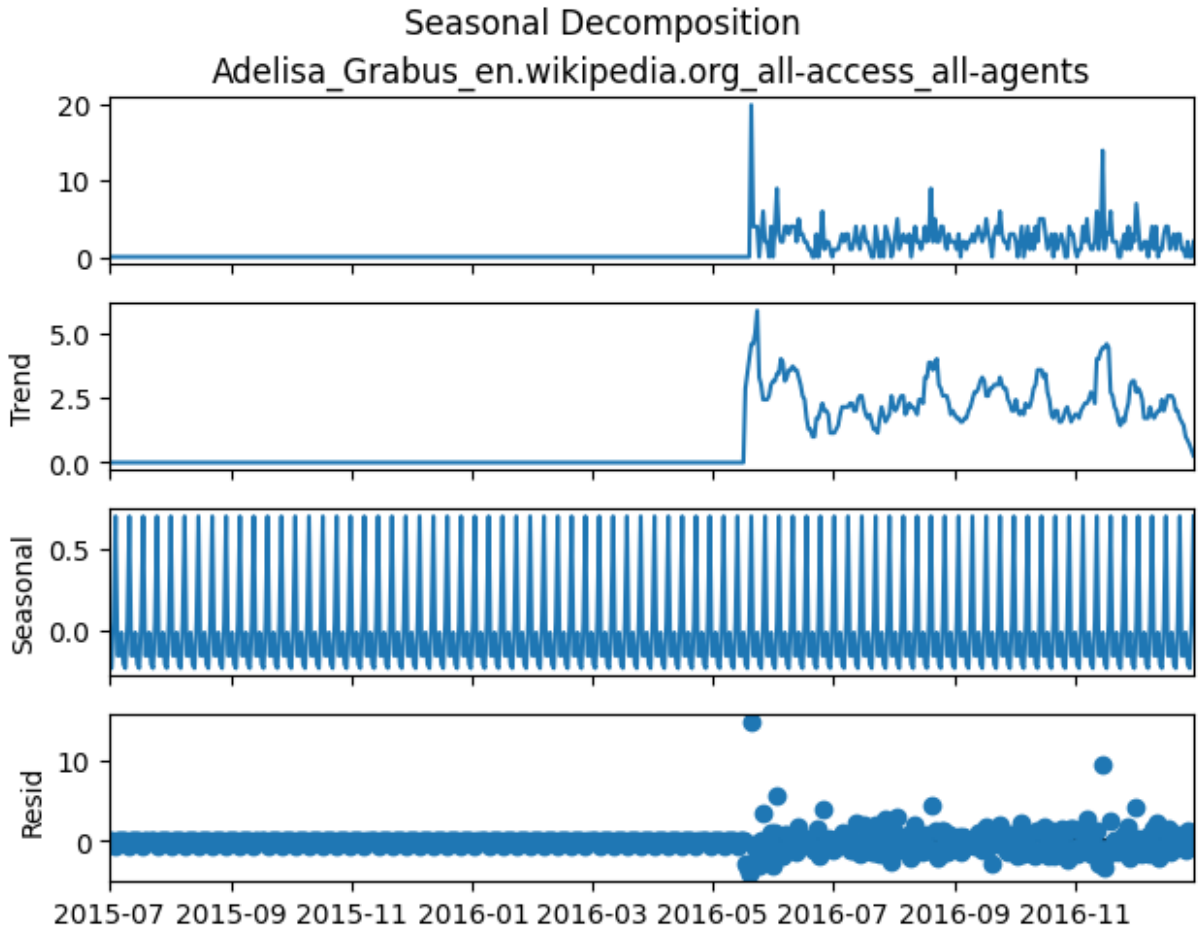


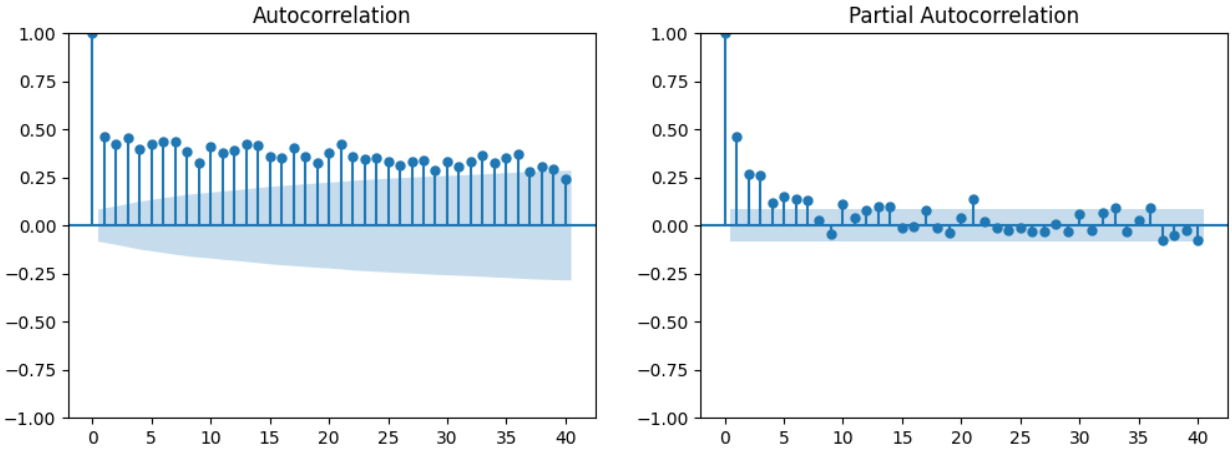




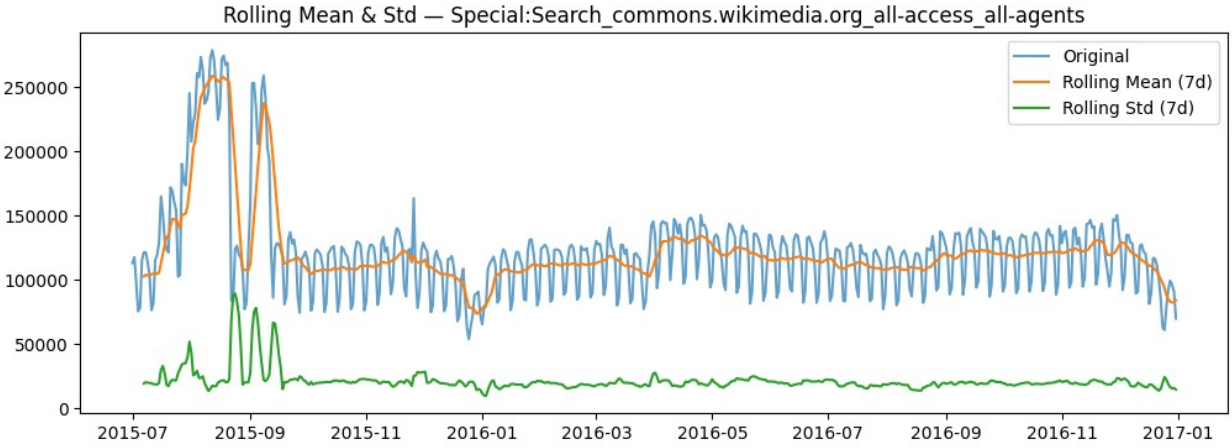

```
🔍 Analyzing page: Adelisa_Grabus_en.wikipedia.org_all-access_all-
agents
ADF Statistic: -2.112 | p-value: 0.2398 | Stationary: False
```

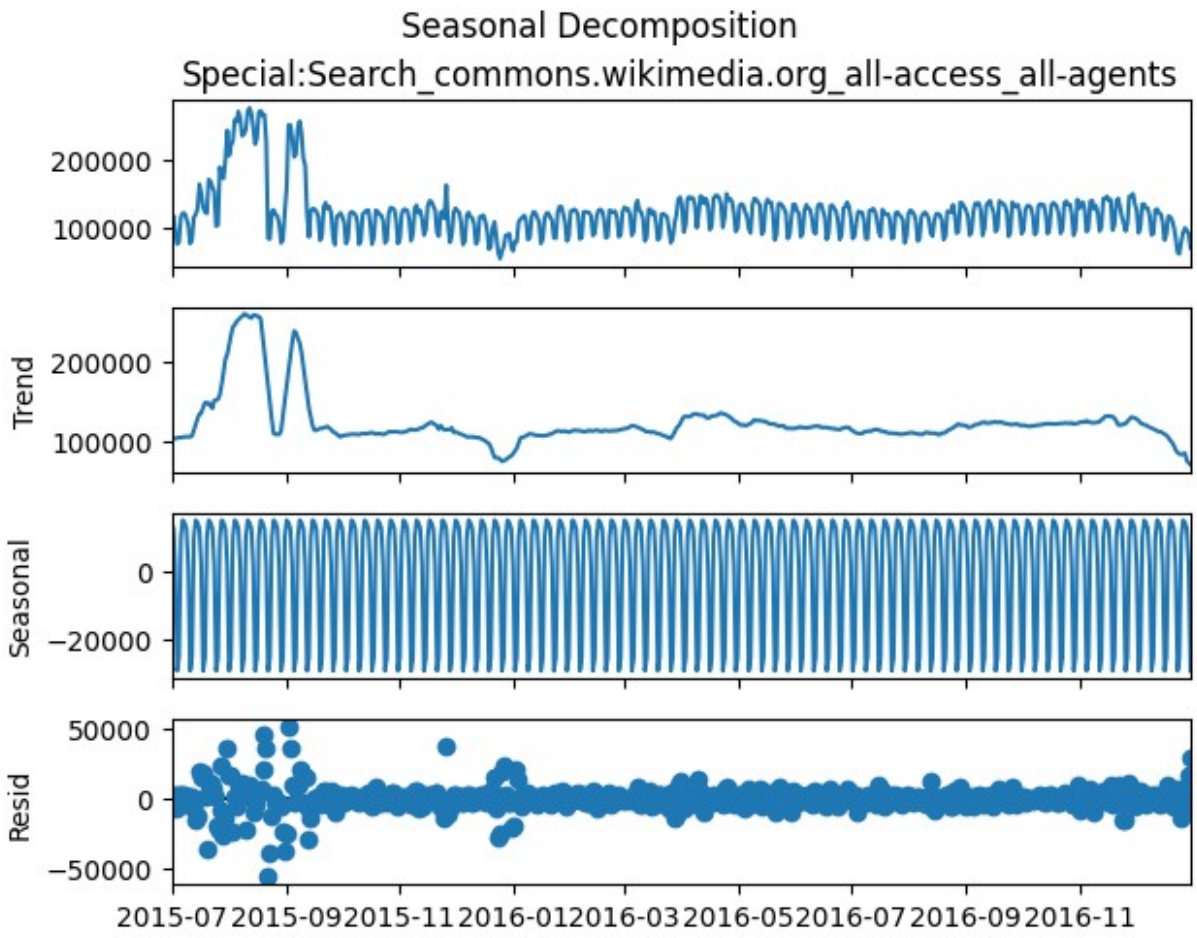

Rolling Mean & Std — Adelisa_Grabus_en.wikipedia.org_all-access_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

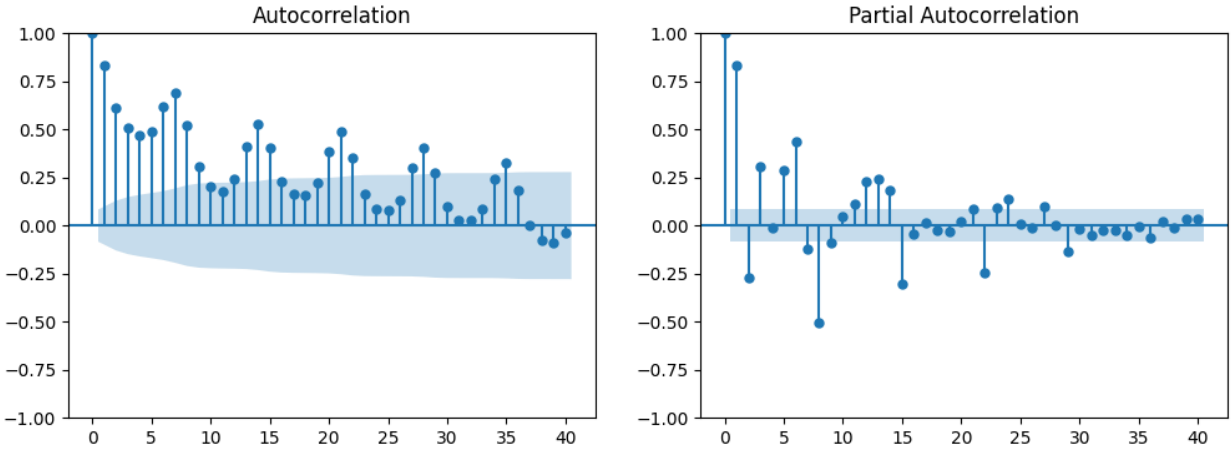

Seasonal Decomposition
Adelisa_Grabus_en.wikipedia.org_all-access_all-agents
20
10
0
Trend
5.0
2.5
0.0
Seasonal
0.5
0.0
Resid
10
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



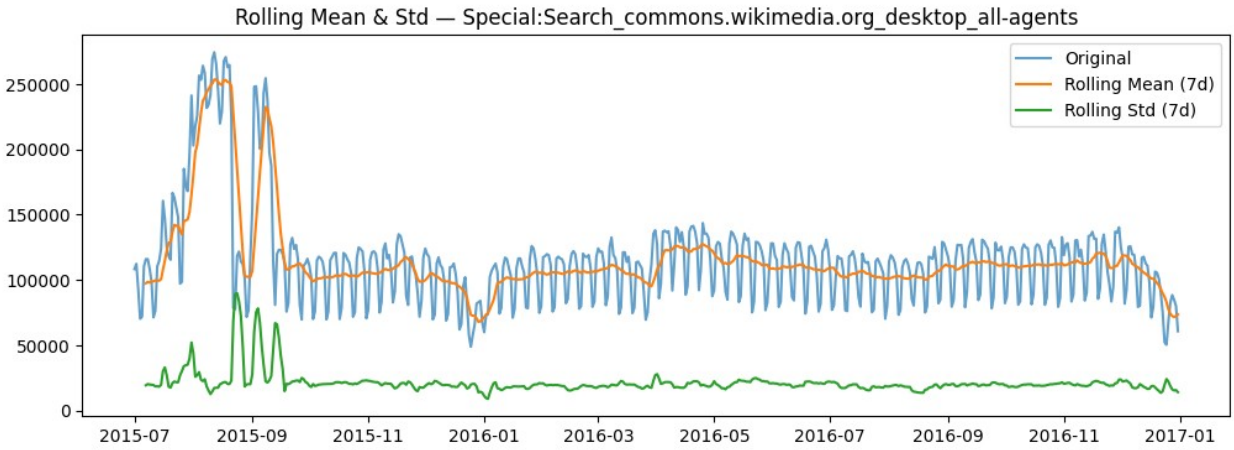

```
 Analyzing page: Special:Search_commons.wikimedia.org_all-access_all-agents
ADF Statistic: -2.972 | p-value: 0.0376 | Stationary: True
```

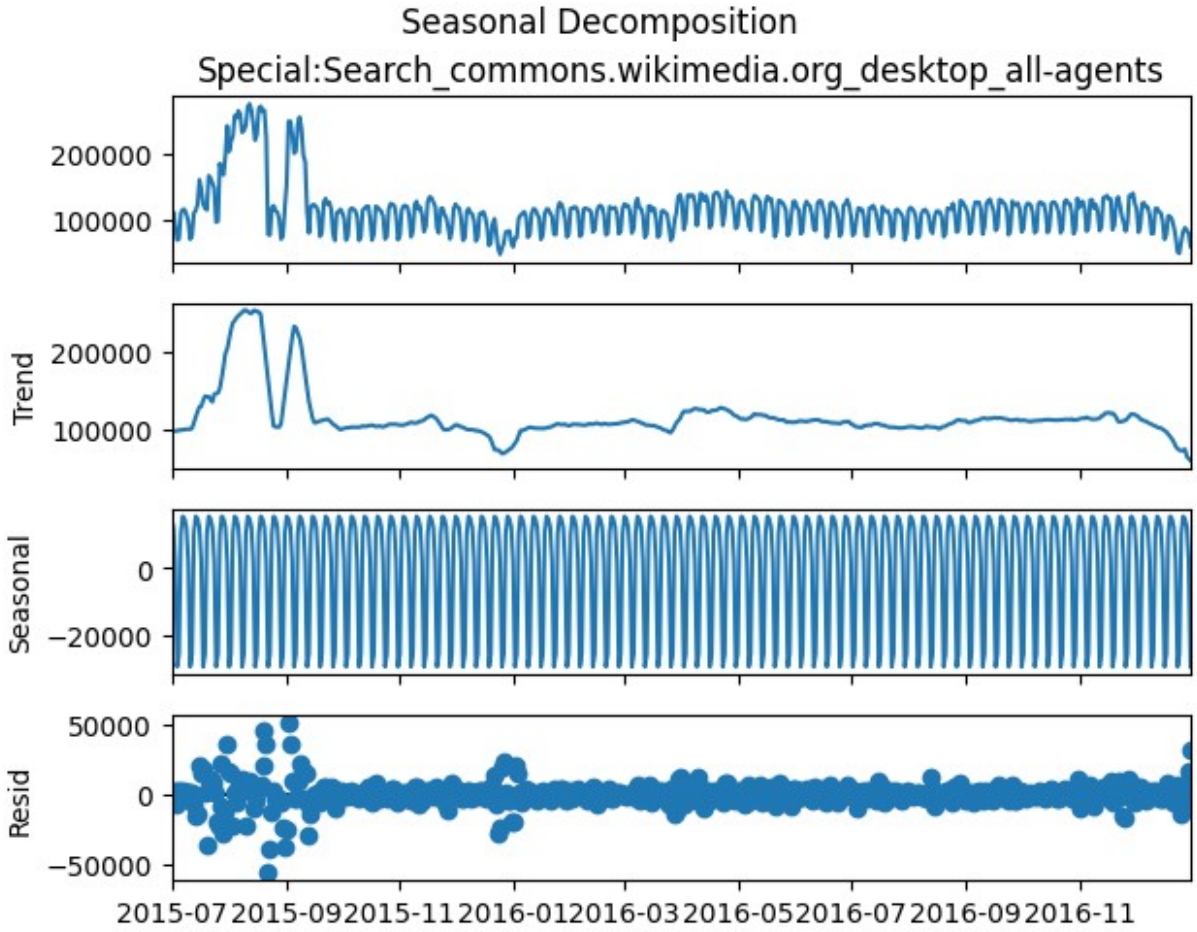


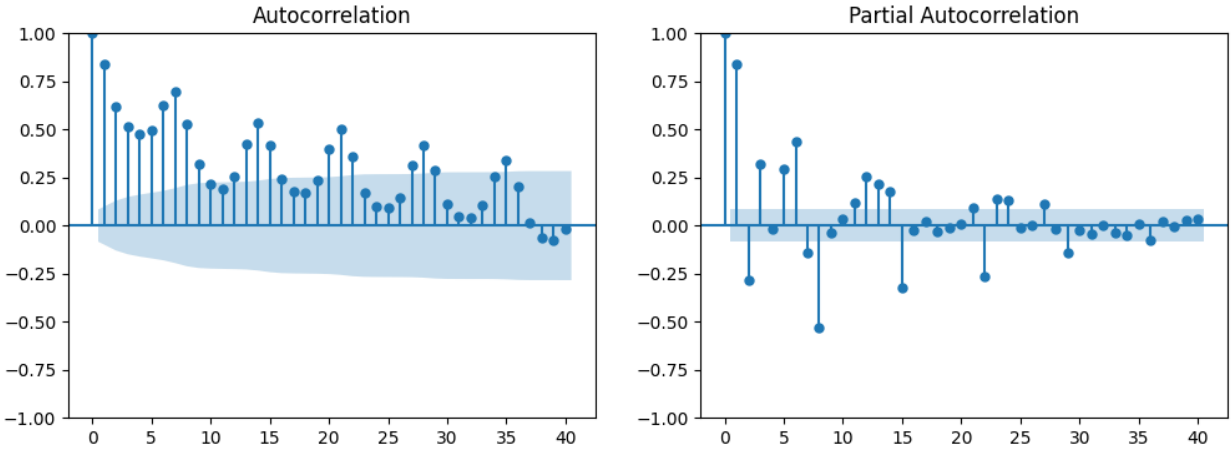




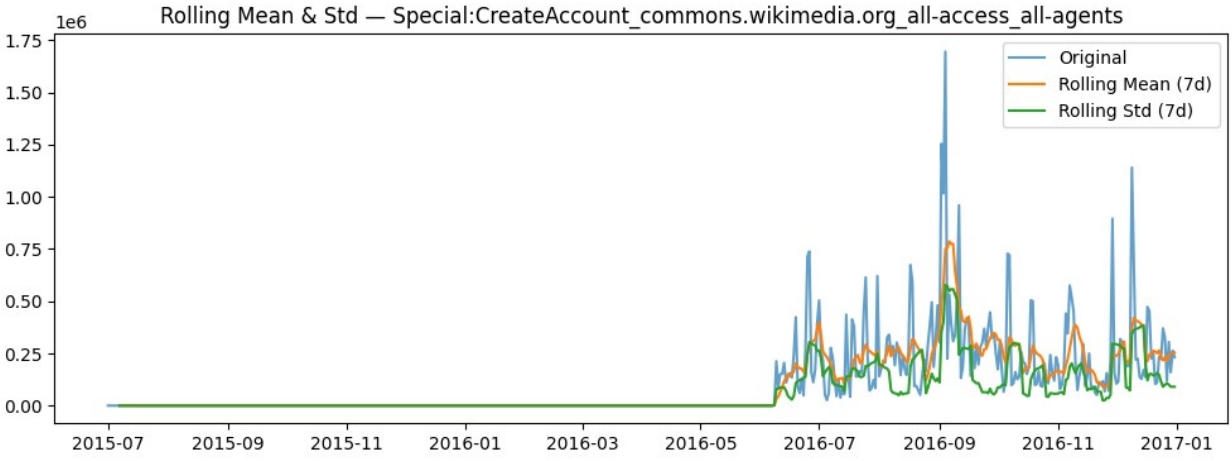

```
🔍 Analyzing page: Special:Search_commons.wikimedia.org_desktop_all-
agents
ADF Statistic: -2.871 | p-value: 0.0488 | Stationary: True
```

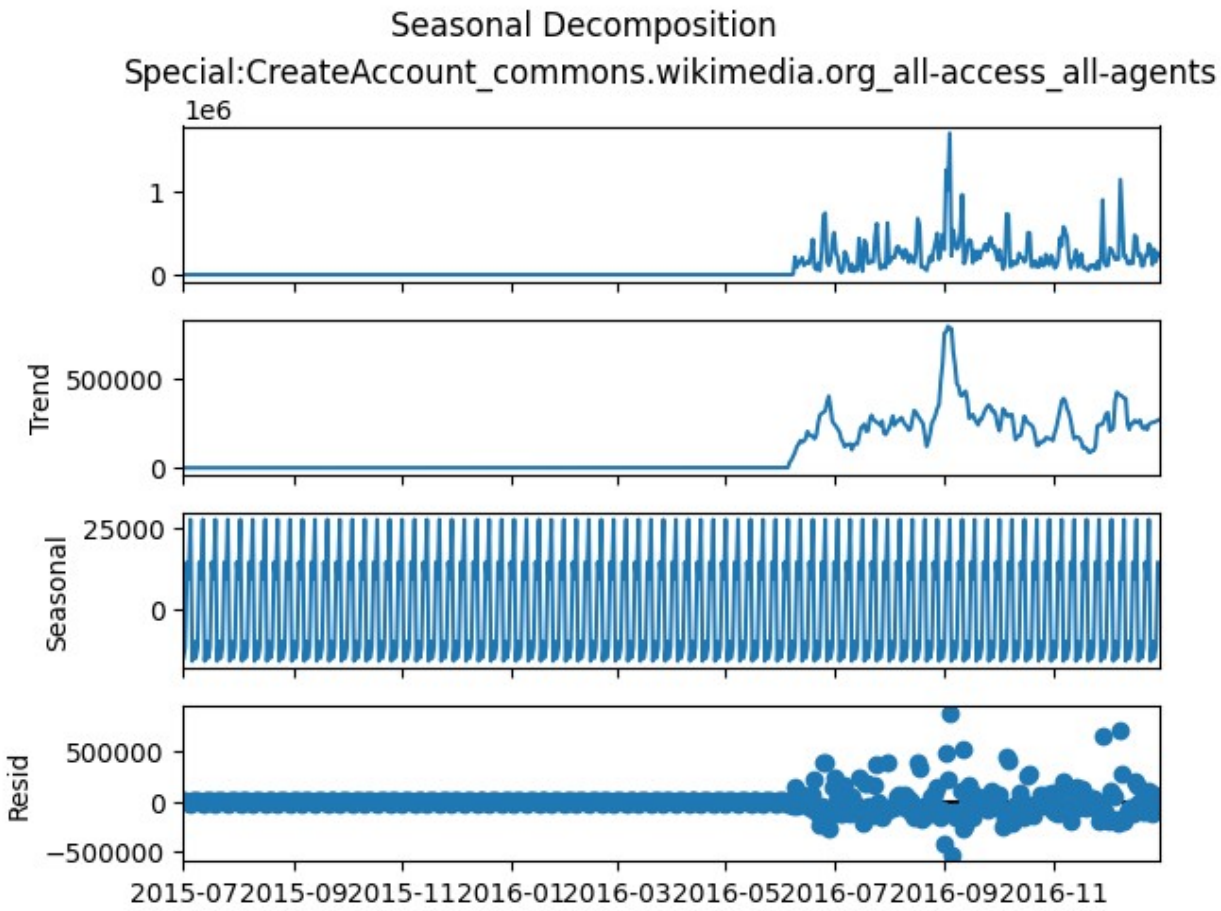

Rolling Mean & Std — Special:Search_commons.wikimedia.org_desktop_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
250000
200000
150000
100000
50000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

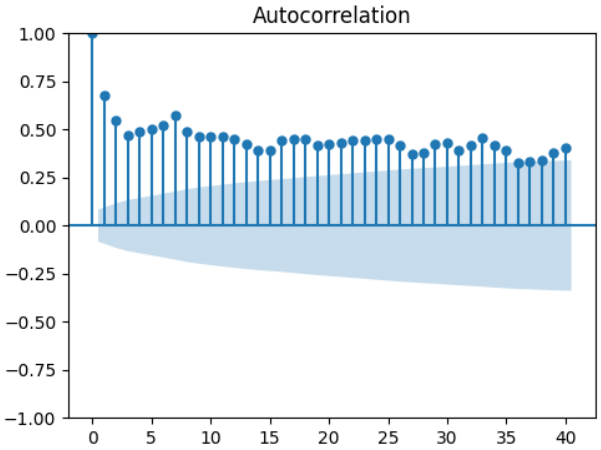

Seasonal Decomposition
Special:Search_commons.wikimedia.org_desktop_all-agents
200000
100000
Trend
200000
100000
Seasonal
0
−20000
Resid
50000
0
−50000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11

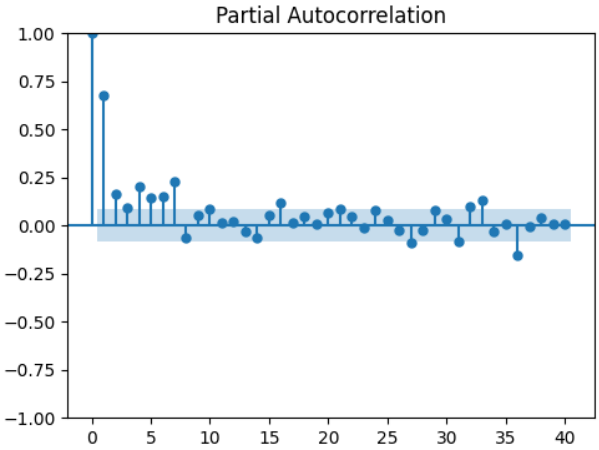



```
🔍 Analyzing page: Special:CreateAccount_commons.wikimedia.org_all-access_all-agents
ADF Statistic: -2.113 | p-value: 0.2393 | Stationary: False
```

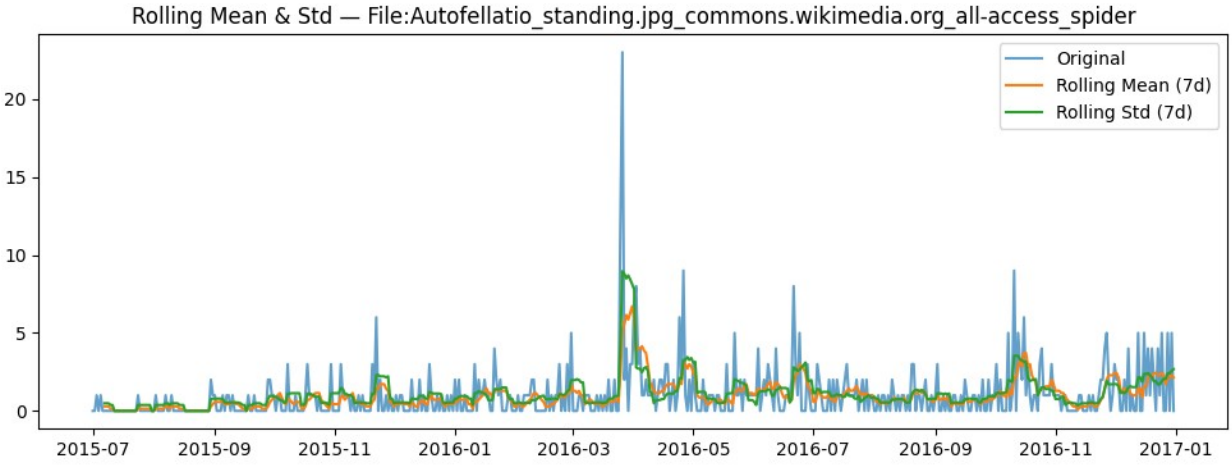


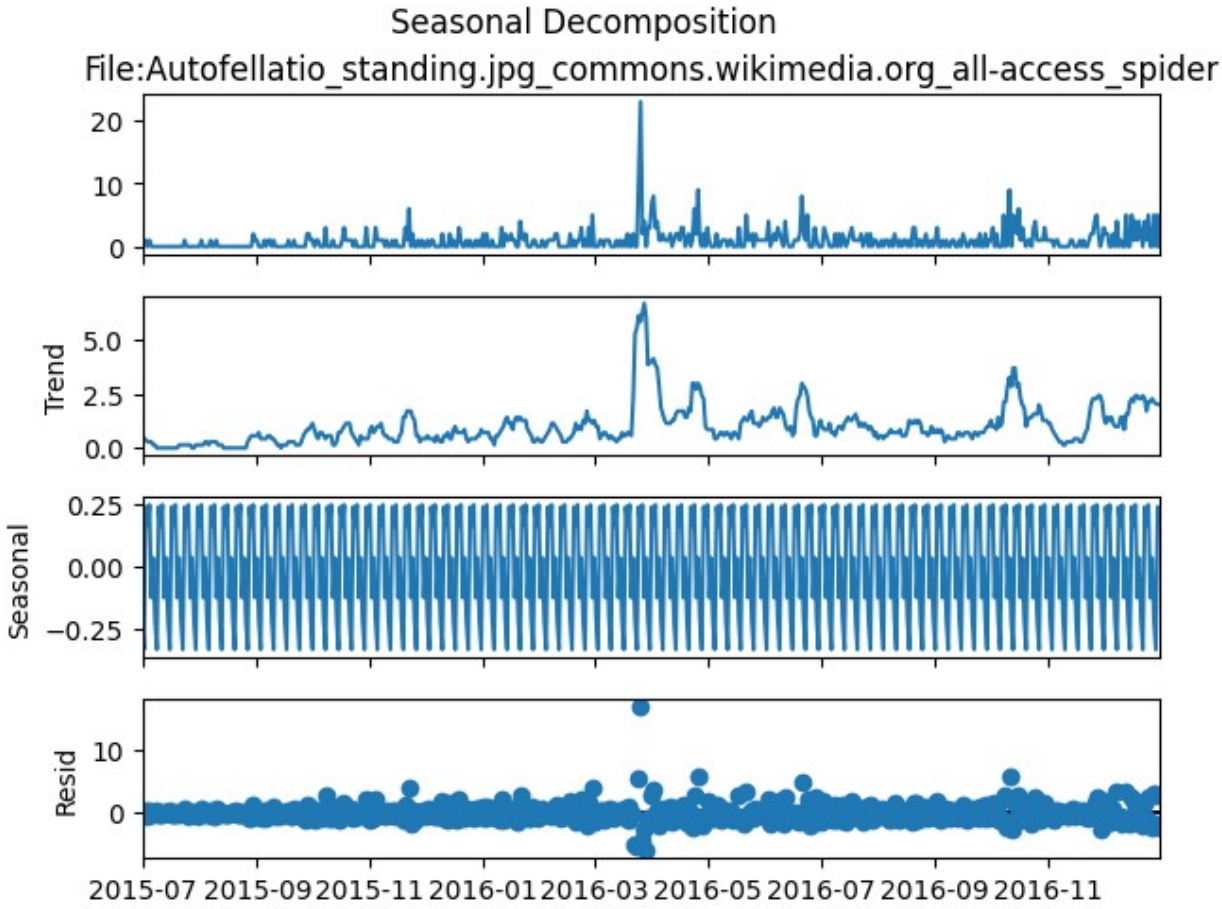


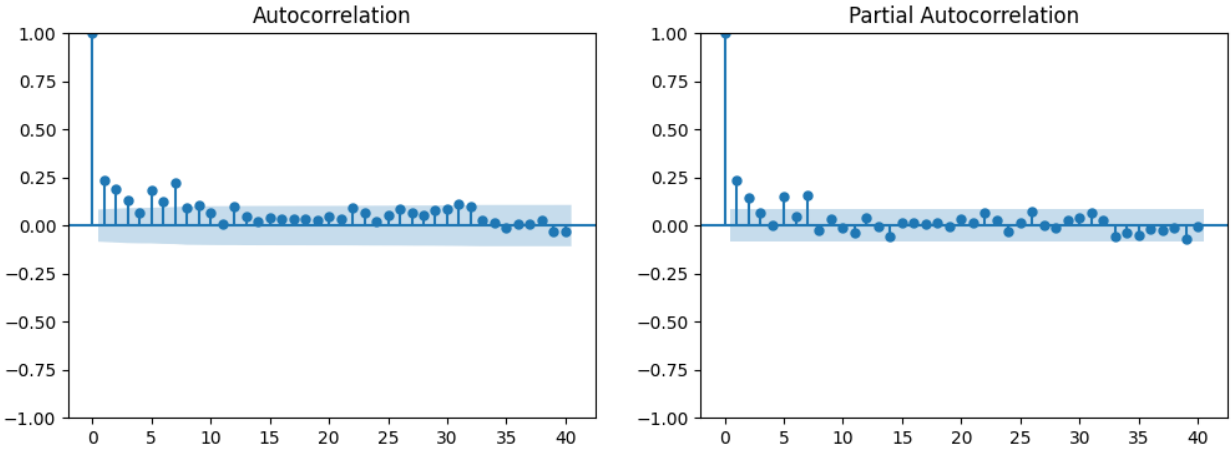




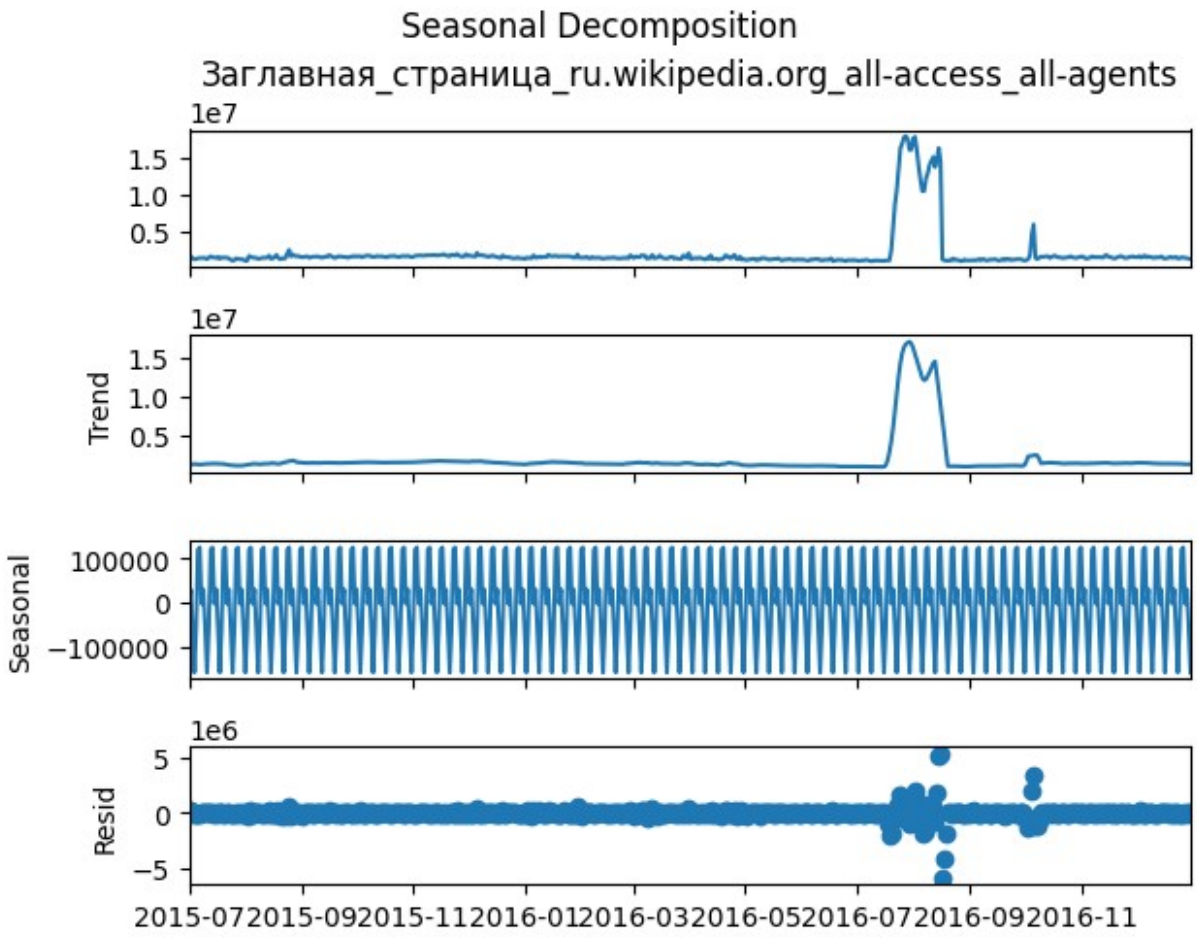

```
 Analyzing page:
File:Autofellatio_standing.jpg_commons.wikimedia.org_all-access_spider
ADF Statistic: -5.467 | p-value: 0.0000 | Stationary: True
```

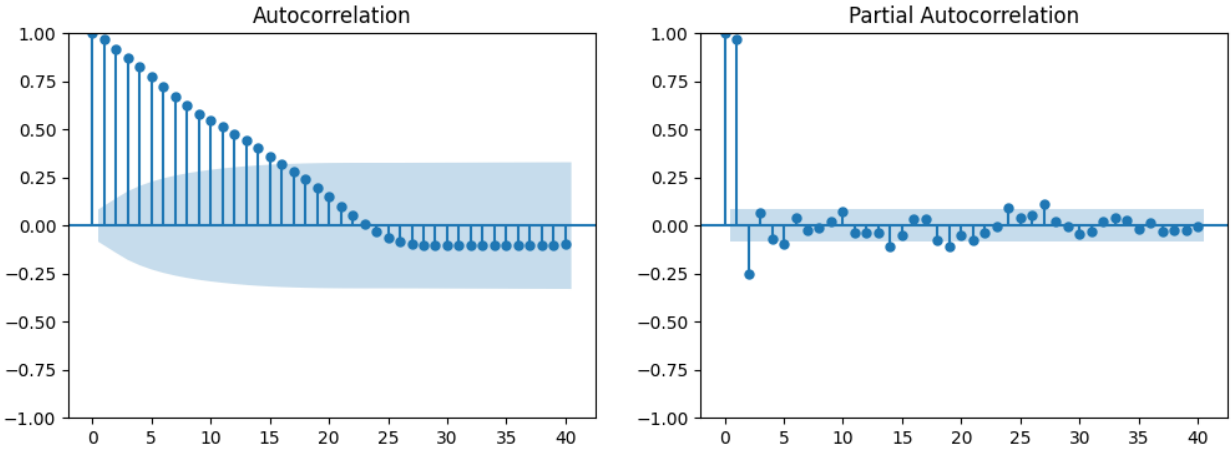

Rolling Mean & Std — File:Autofellatio_standing.jpg_commons.wikimedia.org_all-access_spider
Original
Rolling Mean (7d)
Rolling Std (7d)
20
15
10
5
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Seasonal Decomposition
File:Autofellatio_standing.jpg_commons.wikimedia.org_all-access_spider
20
10
0
Trend
5.0
2.5
0.0
Seasonal
0.25
0.00
−0.25
Resid
10
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



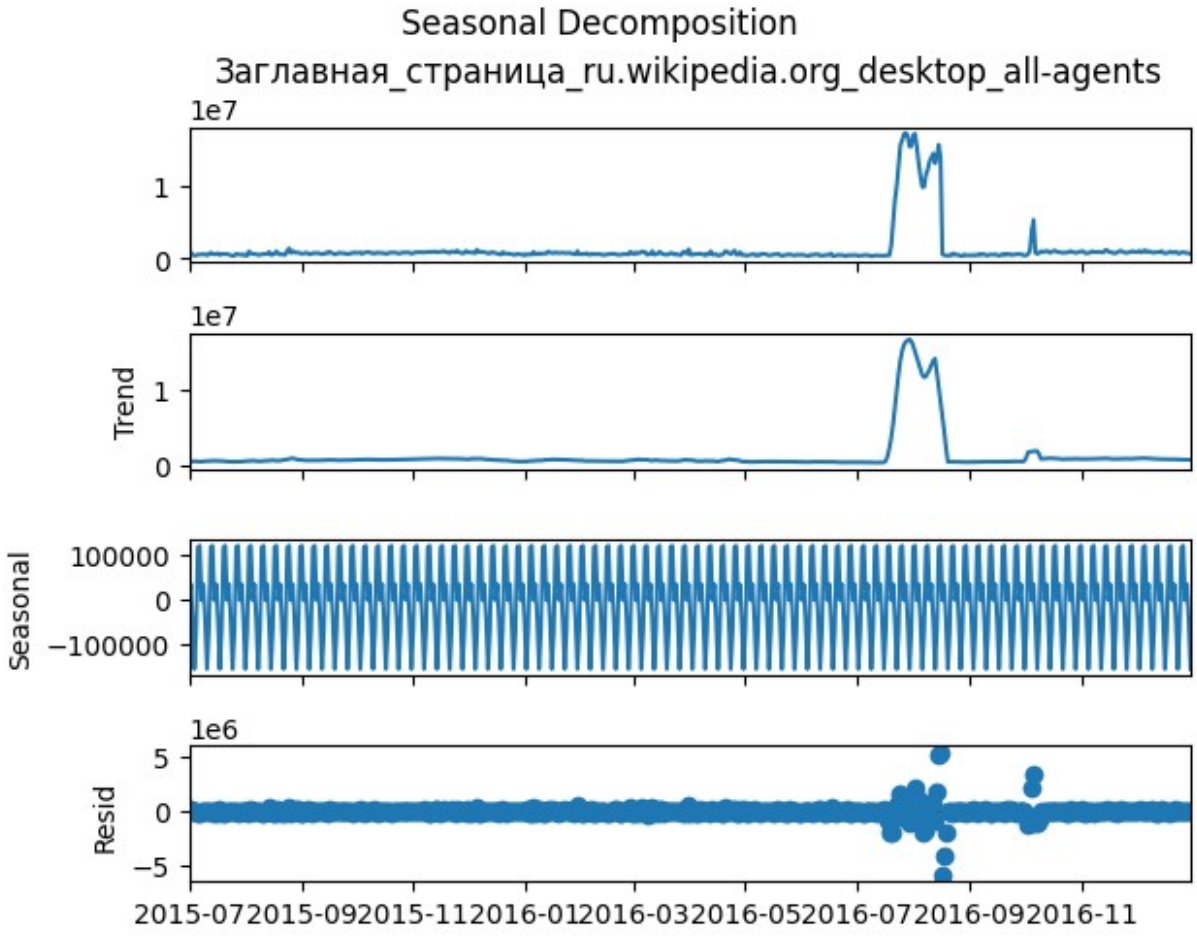

```
🔍 Analyzing page: Заглавная_страница_ru.wikipedia.org_all-access_all-agents
ADF Statistic: -4.128 | p-value: 0.0009 | Stationary: True
```

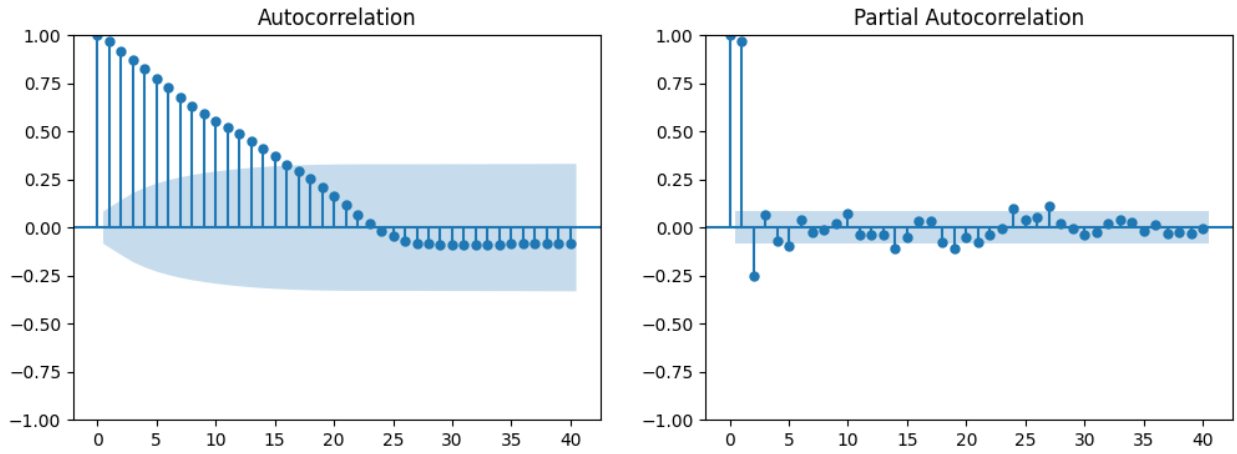




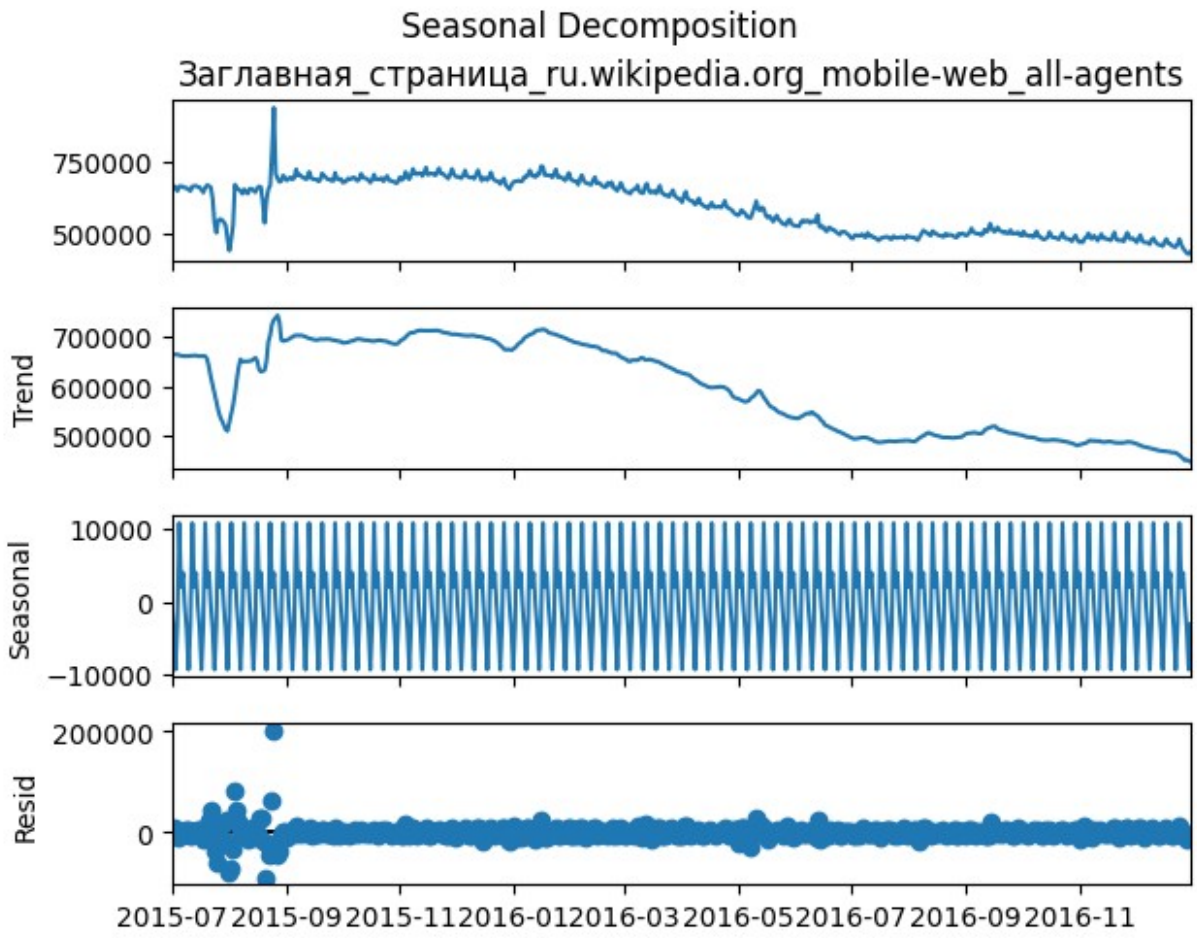

```
 Analyzing page: Заглавная_страница_ru.wikipedia.org_desktop_all-
agents
ADF Statistic: -4.095 | p-value: 0.0010 | Stationary: True
```

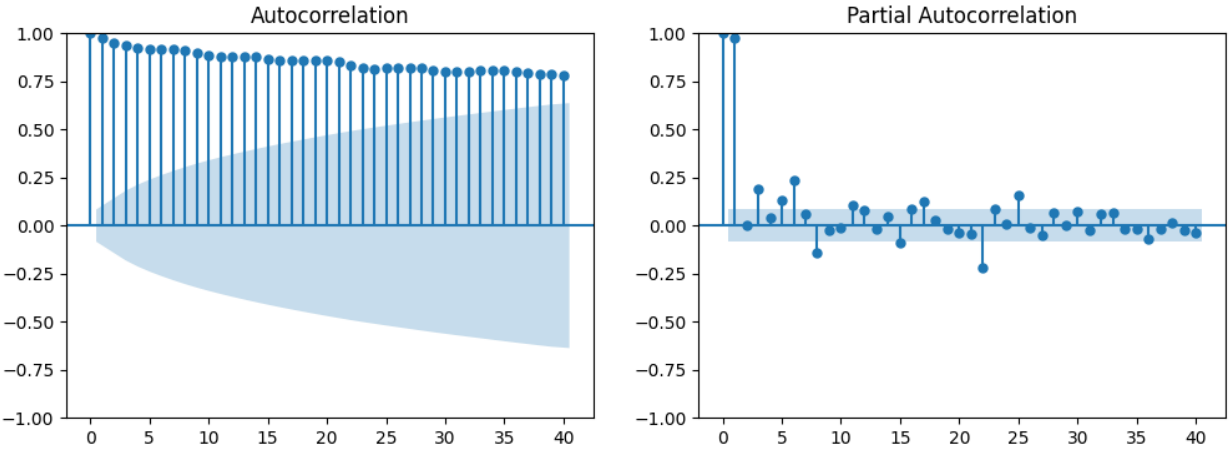

Rolling Mean & Std — Заглавная_страница_ru.wikipedia.org_desktop_all-agents
1e7
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Original
Rolling Mean (7d)
Rolling Std (7d)
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Seasonal Decomposition
Заглавная_страница_ru.wikipedia.org_desktop_all-agents
1e7
1
0
1e7
Trend
1
0
Seasonal
100000
0
−100000
1e6
Resid
5
0
−5
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



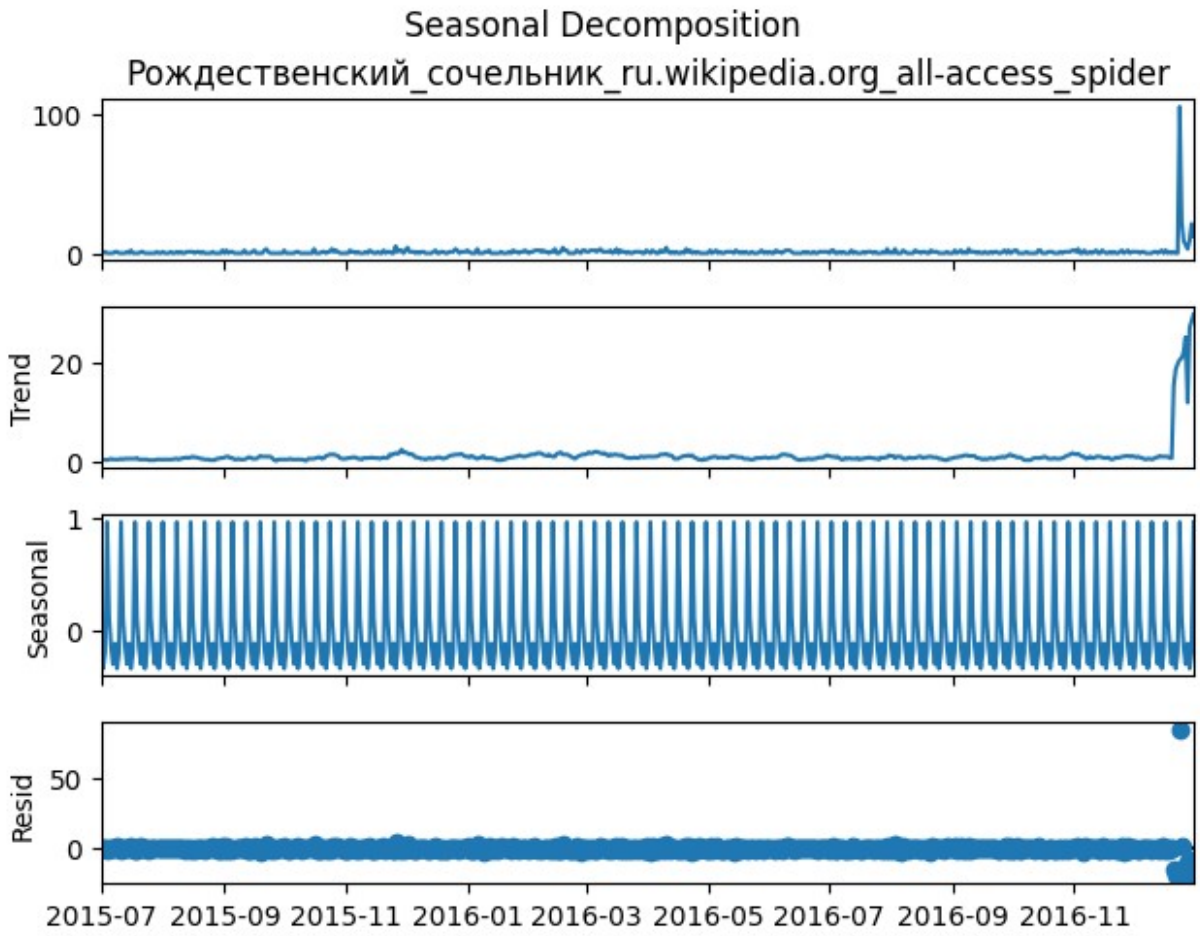

```
🔍 Analyzing page: Заглавная_страница_ru.wikipedia.org_mobile-web_all-agents
ADF Statistic: -0.163 | p-value: 0.9428 | Stationary: False
```

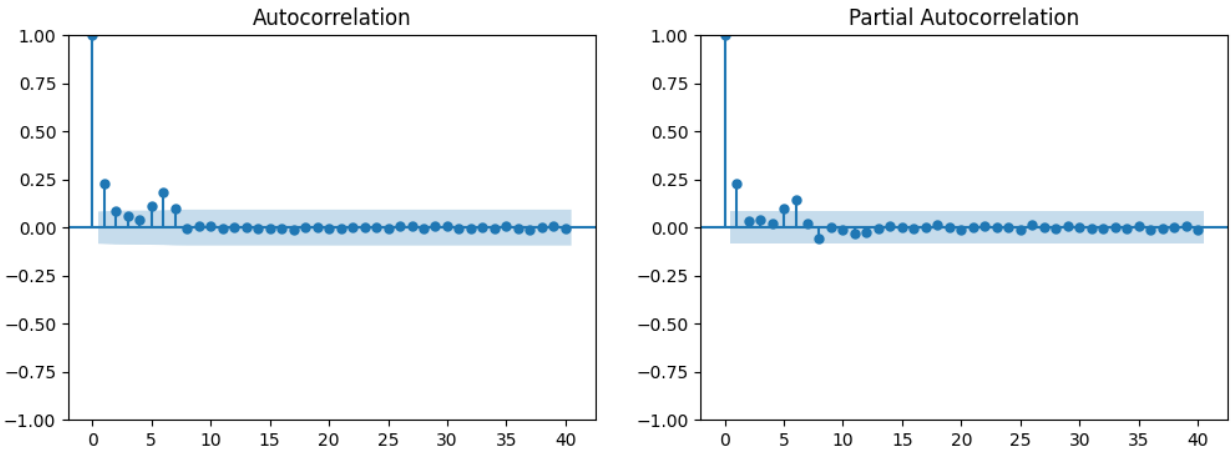




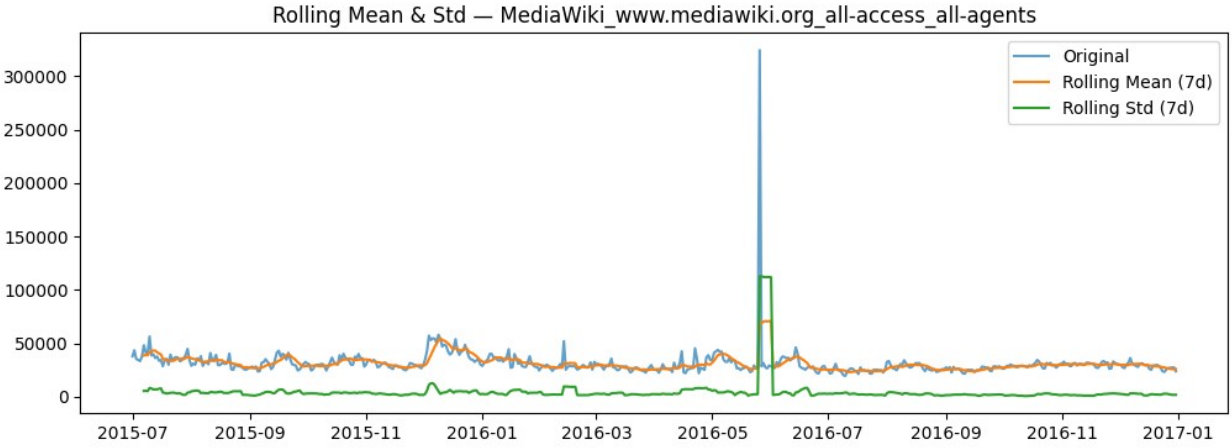

```
🔍 Analyzing page: Рождественский_сочельник_ru.wikipedia.org_all-access_spider
ADF Statistic: -6.169 | p-value: 0.0000 | Stationary: True
```

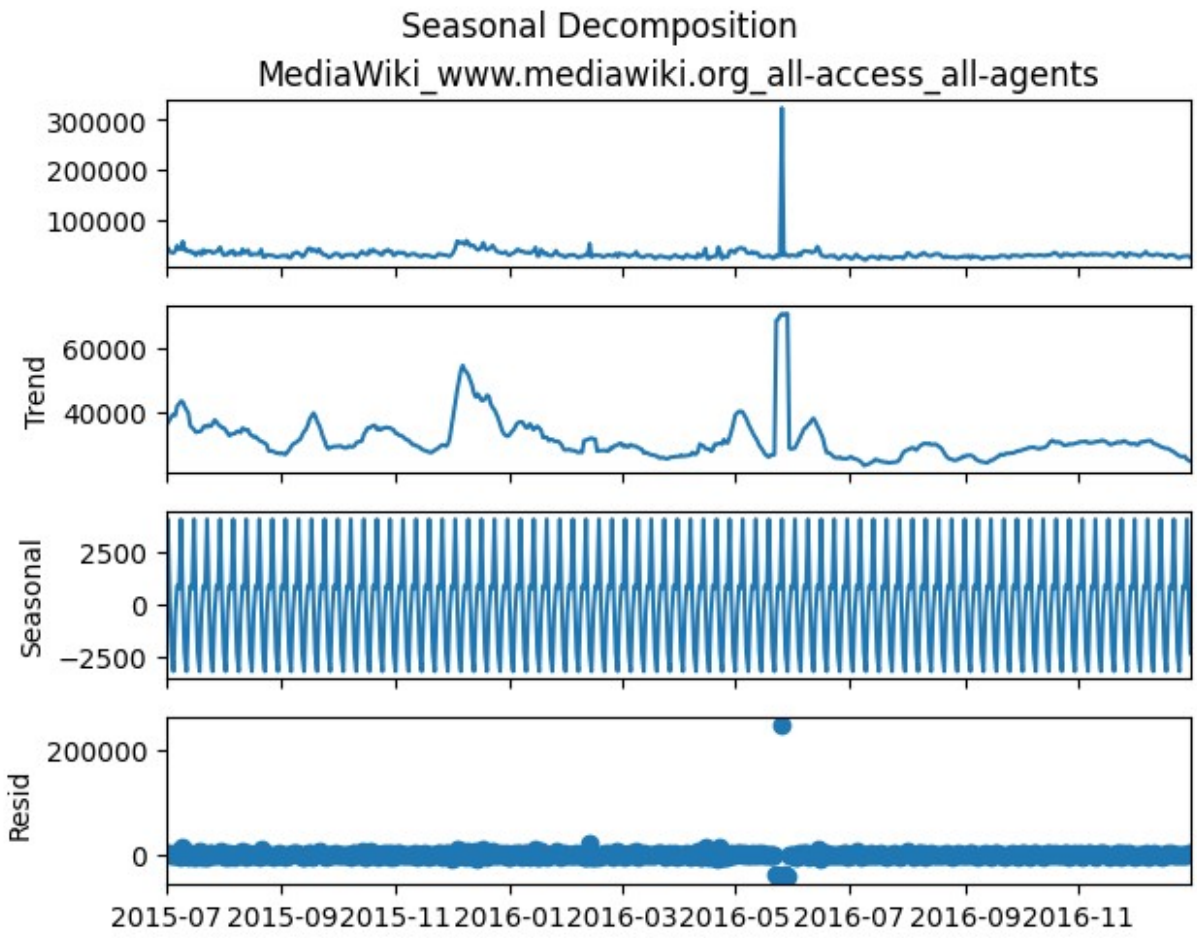

Rolling Mean & Std — Рождественский_сочельник_ru.wikipedia.org_all-access_spider
Original
Rolling Mean (7d)
Rolling Std (7d)
100
80
60
40
20
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

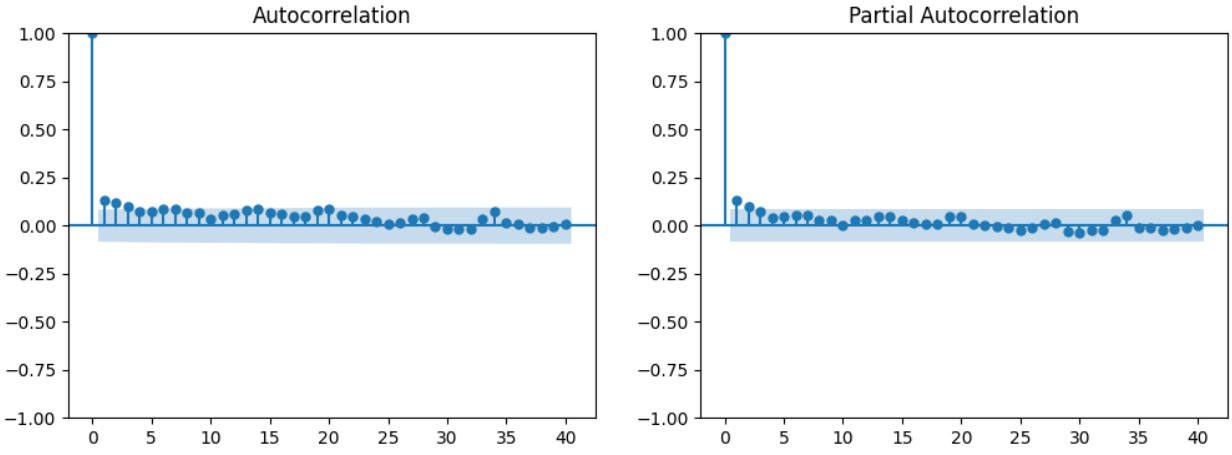

Seasonal Decomposition
Рождественский_сочельник_ru.wikipedia.org_all-access_spider
100
0
Trend
20
0
Seasonal
1
0
Resid
50
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



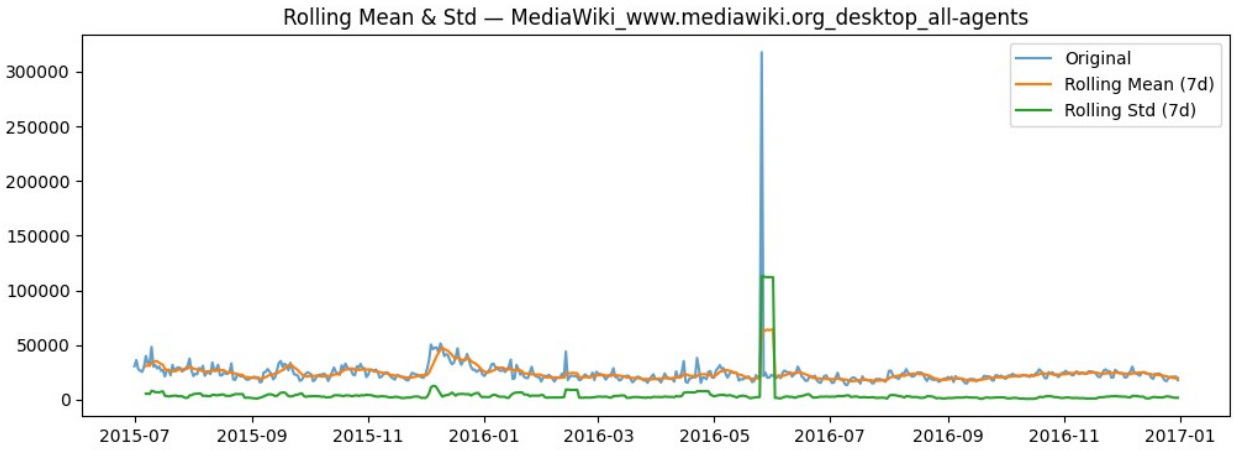

```
🔍 Analyzing page: MediaWiki_www.mediawiki.org_all-access_all-agents
ADF Statistic: -11.068 | p-value: 0.0000 | Stationary: True
```

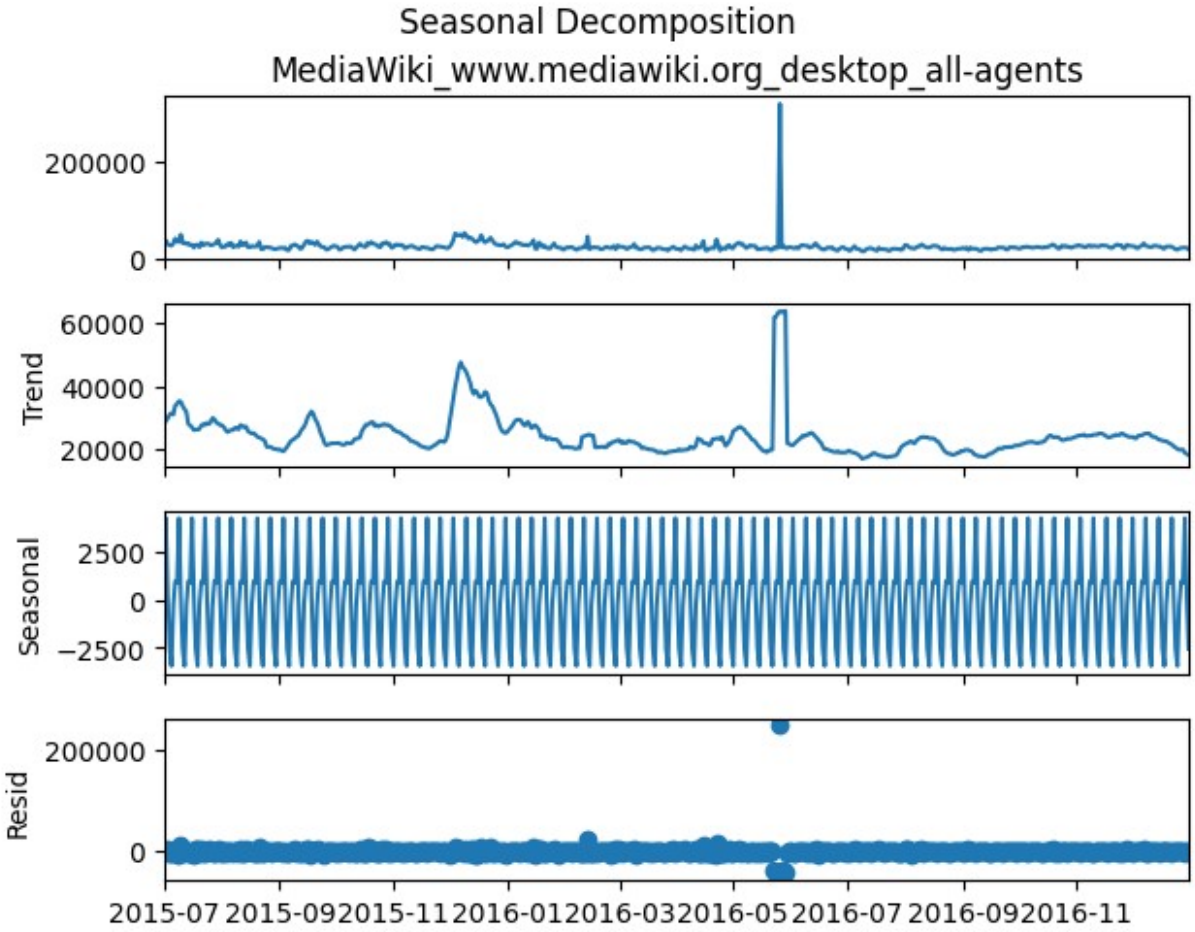


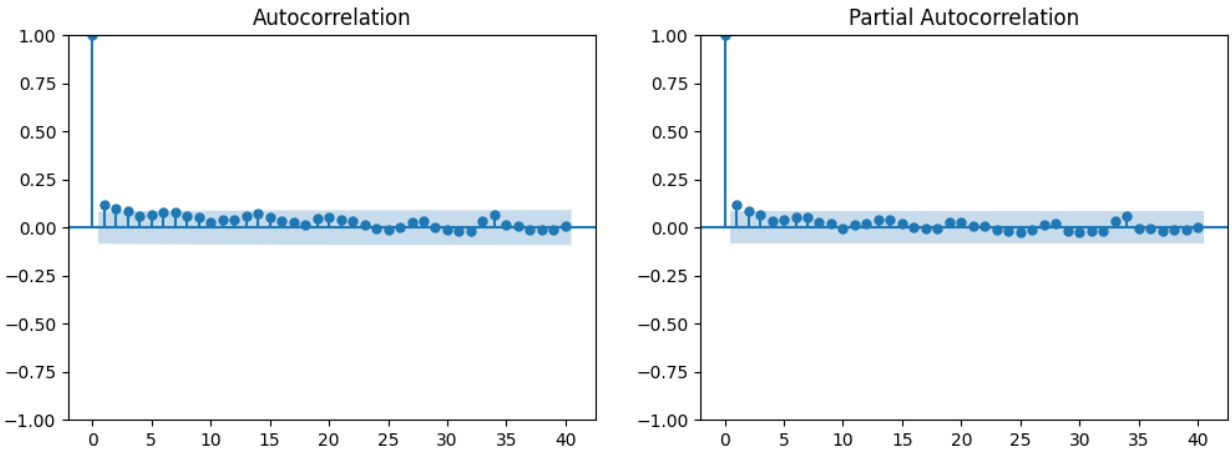




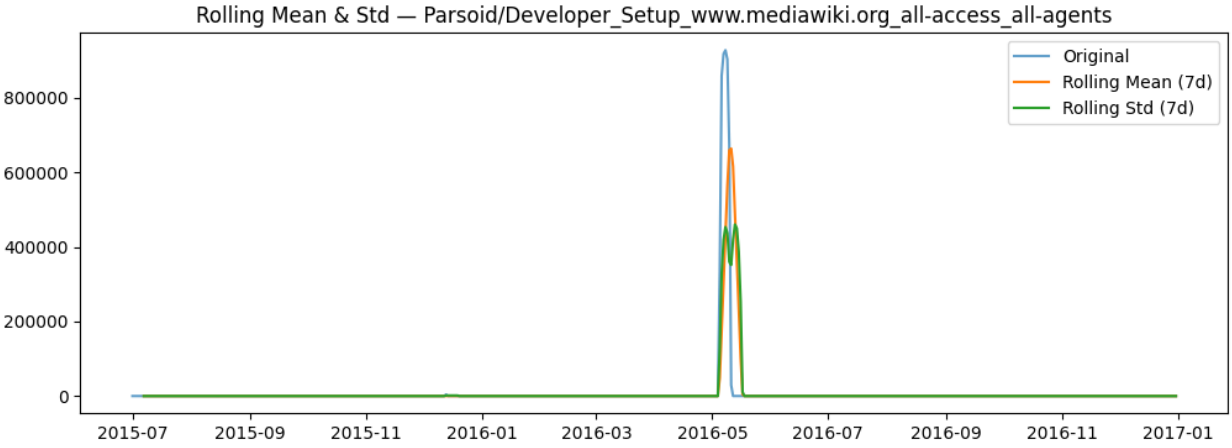

```
 Analyzing page: MediaWiki_www.mediawiki.org_desktop_all-agents
ADF Statistic: -14.194 | p-value: 0.0000 | Stationary: True
```

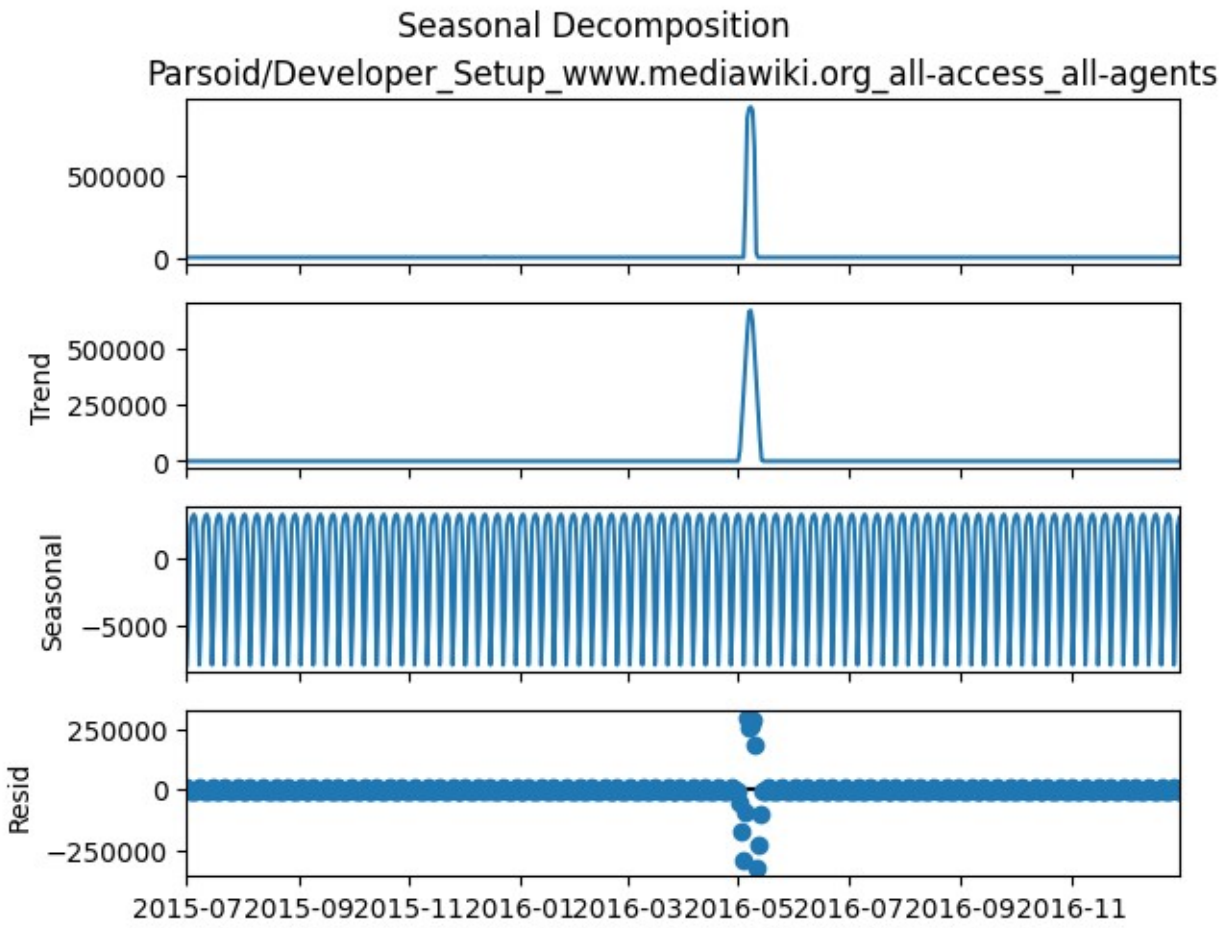

Rolling Mean & Std — MediaWiki_www.mediawiki.org_desktop_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
300000
250000
200000
150000
100000
50000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

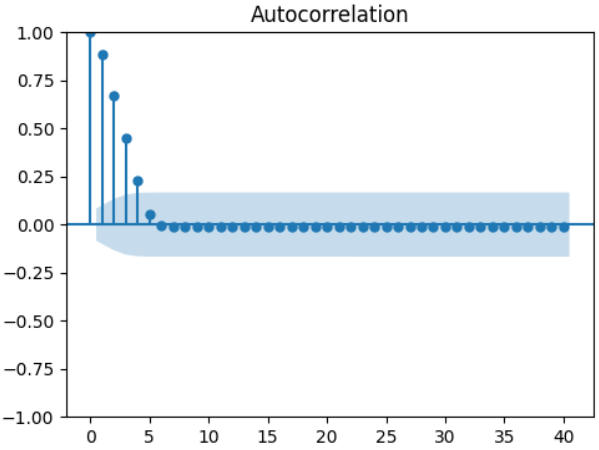

Seasonal Decomposition
MediaWiki_www.mediawiki.org_desktop_all-agents
200000
0
Trend
60000
40000
20000
Seasonal
2500
0
−2500
Resid
200000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11

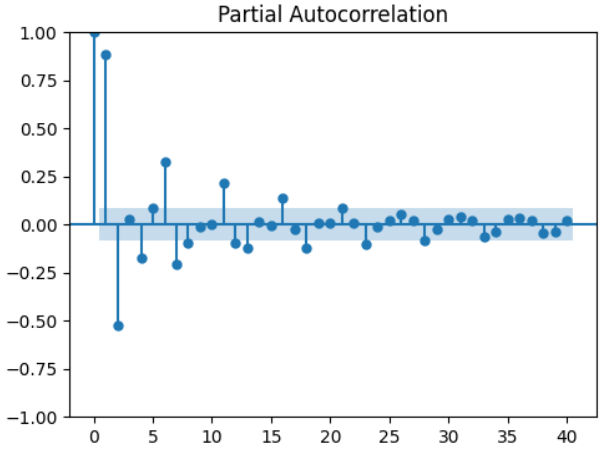



```
🔍 Analyzing page: Parsoid/Developer_Setup_www.mediawiki.org_all-access_all-agents
ADF Statistic: -5.157 | p-value: 0.0000 | Stationary: True
```

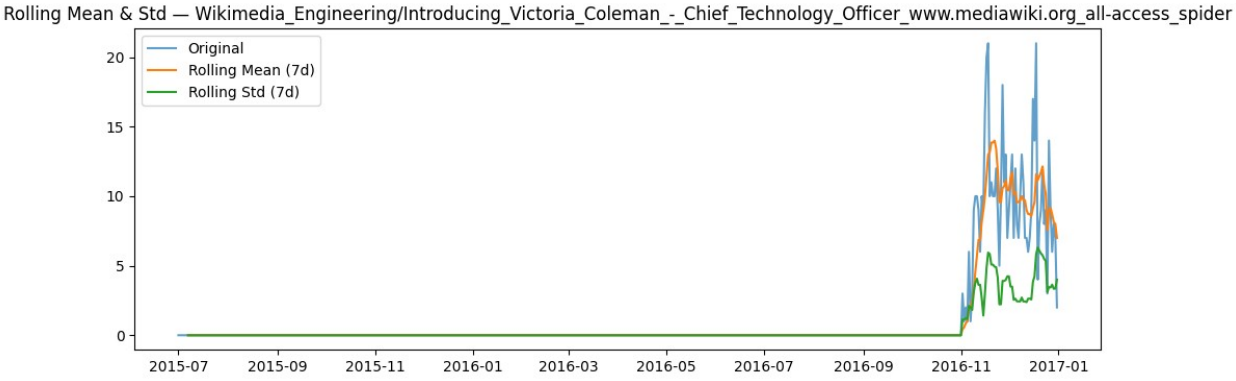


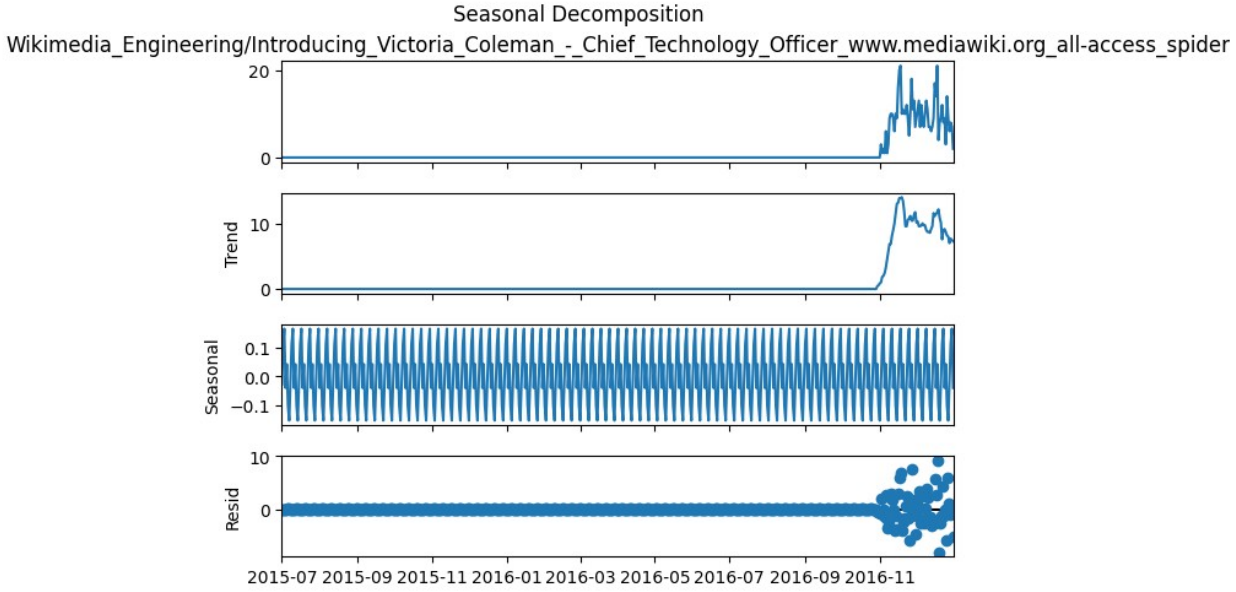


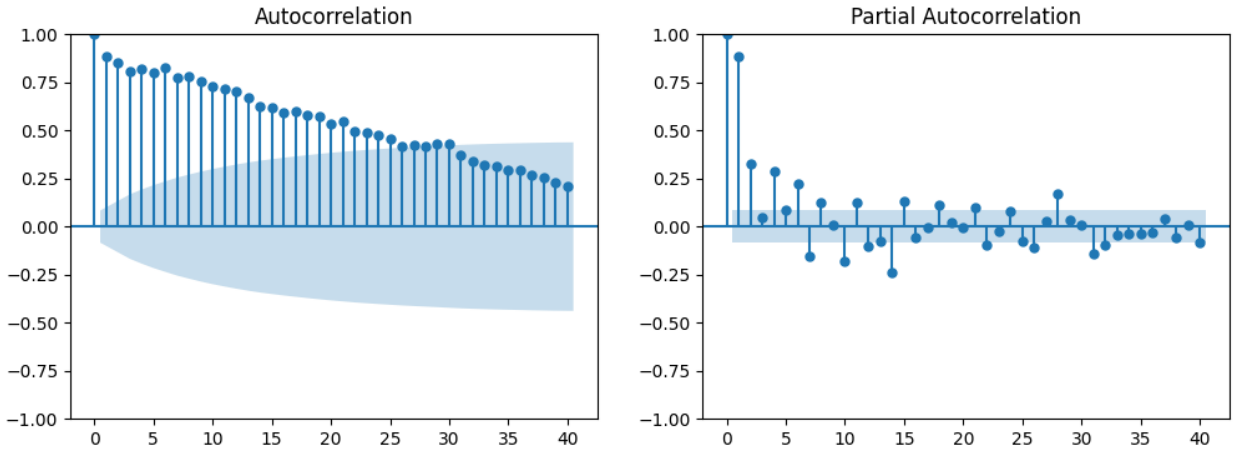




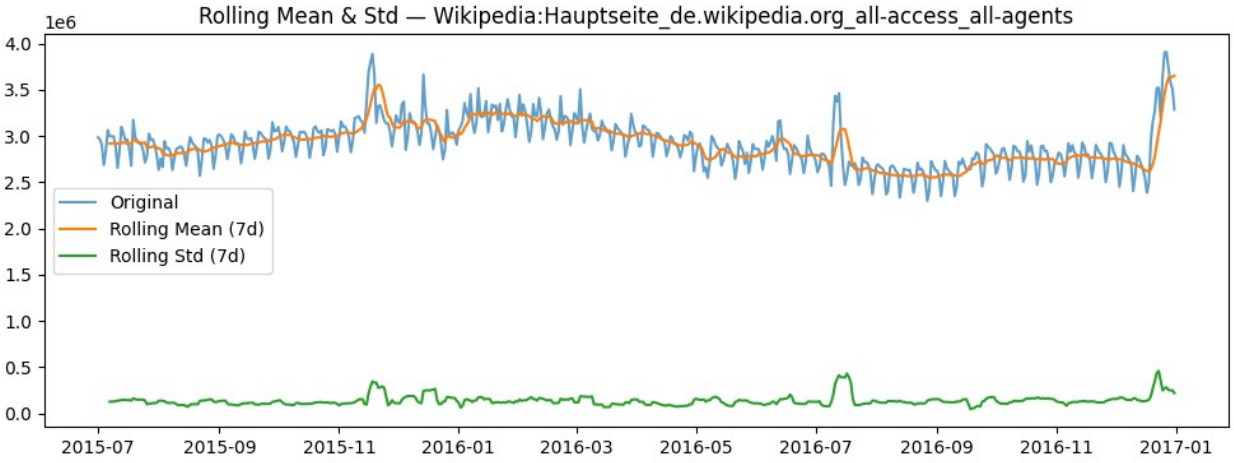

```
 Analyzing page: Wikimedia_Engineering/Introducing_Victoria_Coleman_-_Chief_Technology_Officer_www.mediawiki.org_all-access_spider
ADF Statistic: -1.793 | p-value: 0.3841 | Stationary: False
```

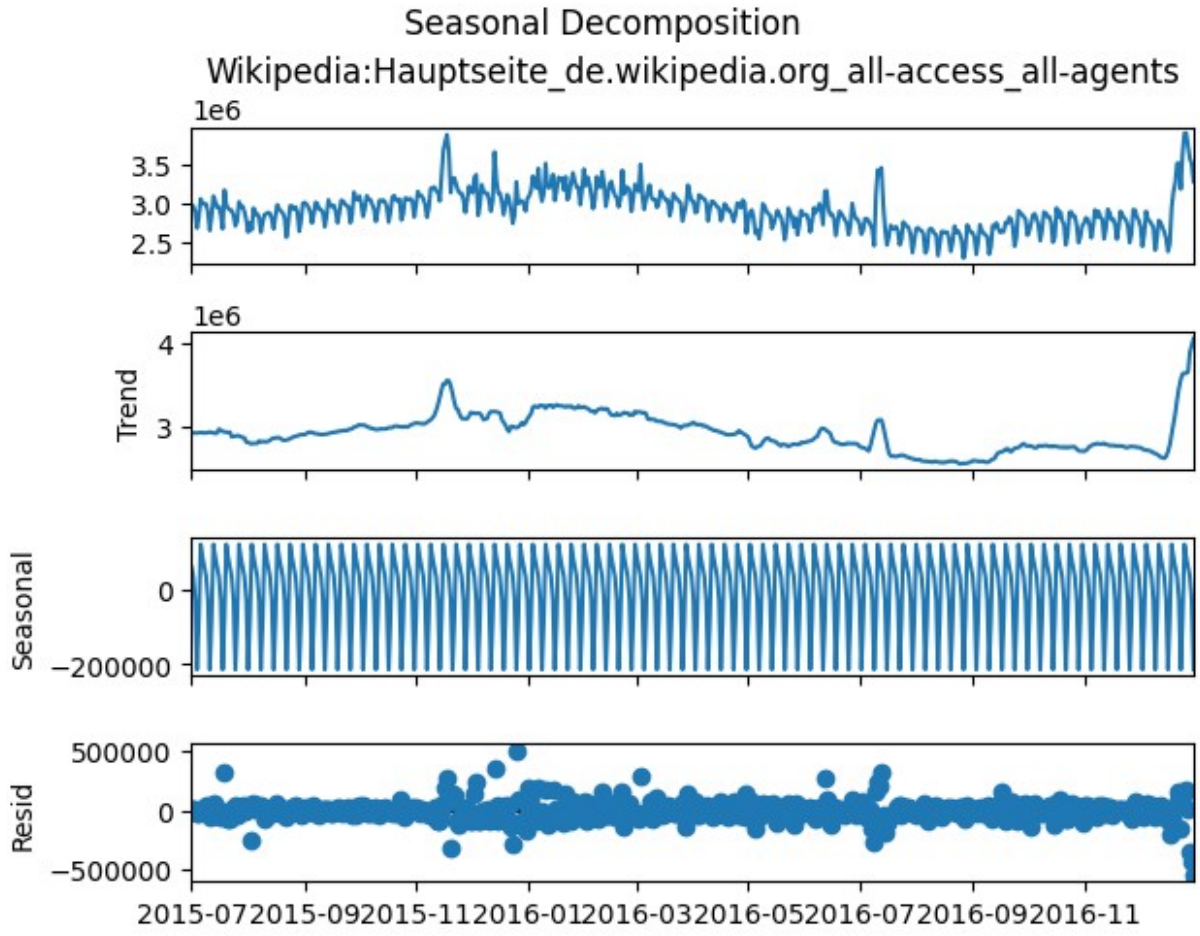


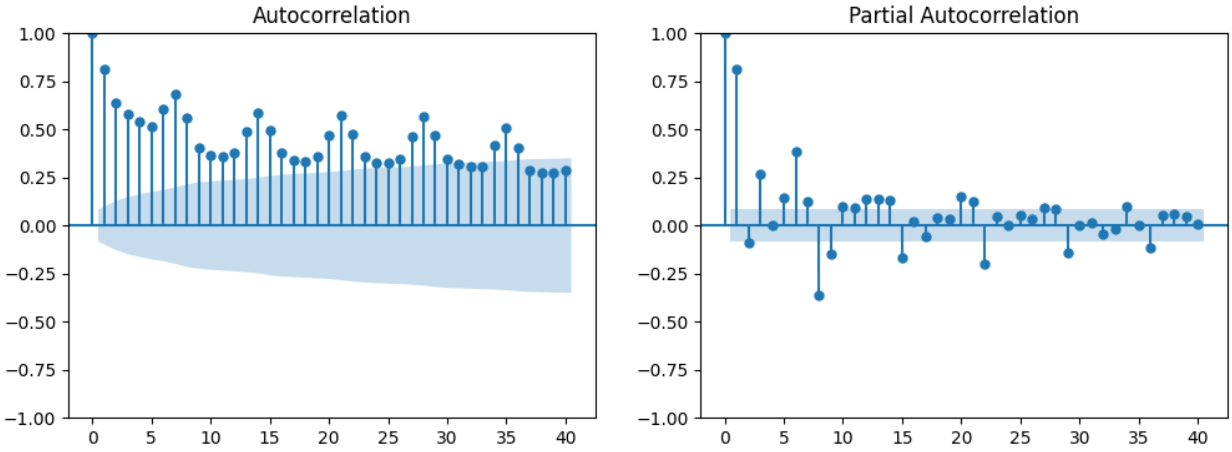




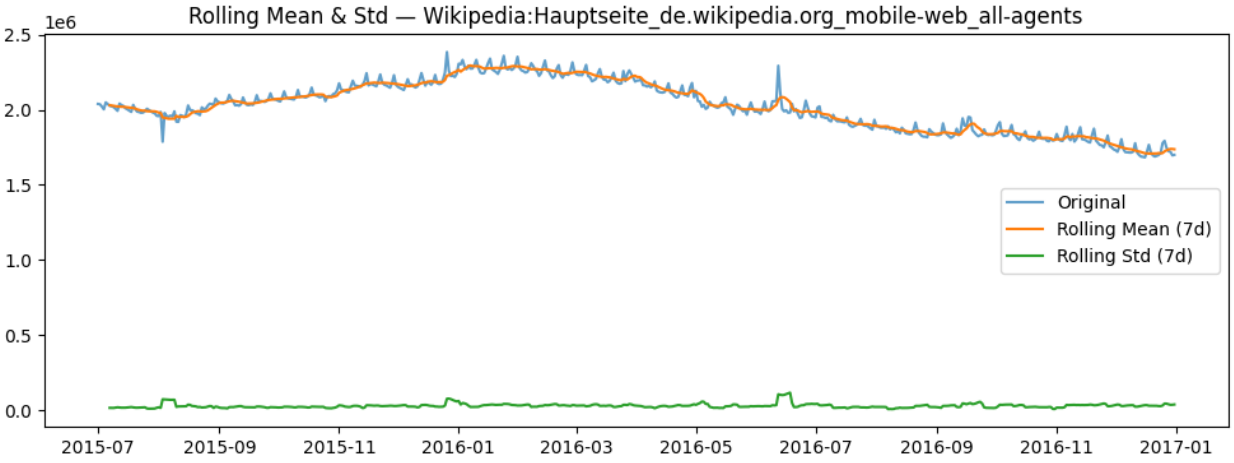

```
🔍 Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents
ADF Statistic: -1.520 | p-value: 0.5236 | Stationary: False
```

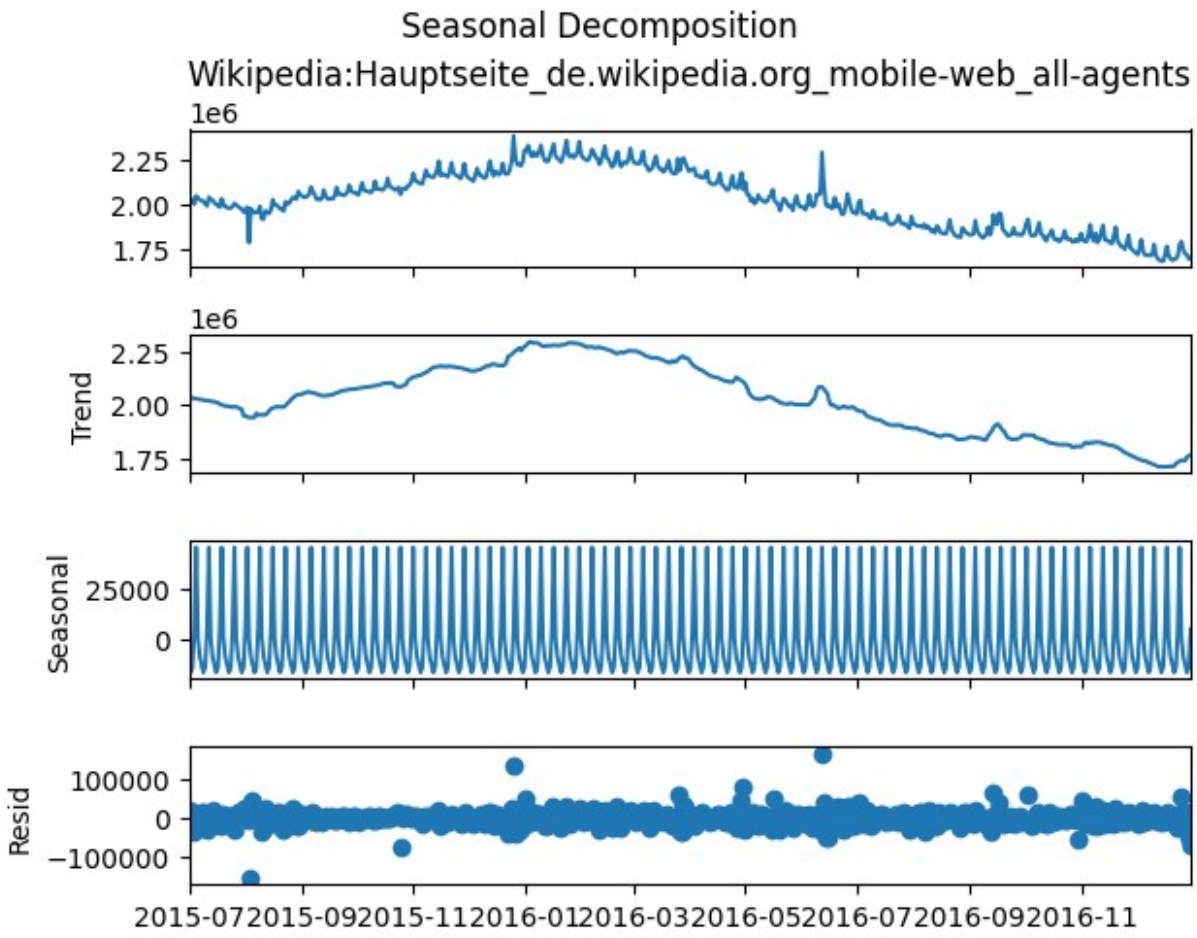

Rolling Mean & Std — Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Original
Rolling Mean (7d)
Rolling Std (7d)
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

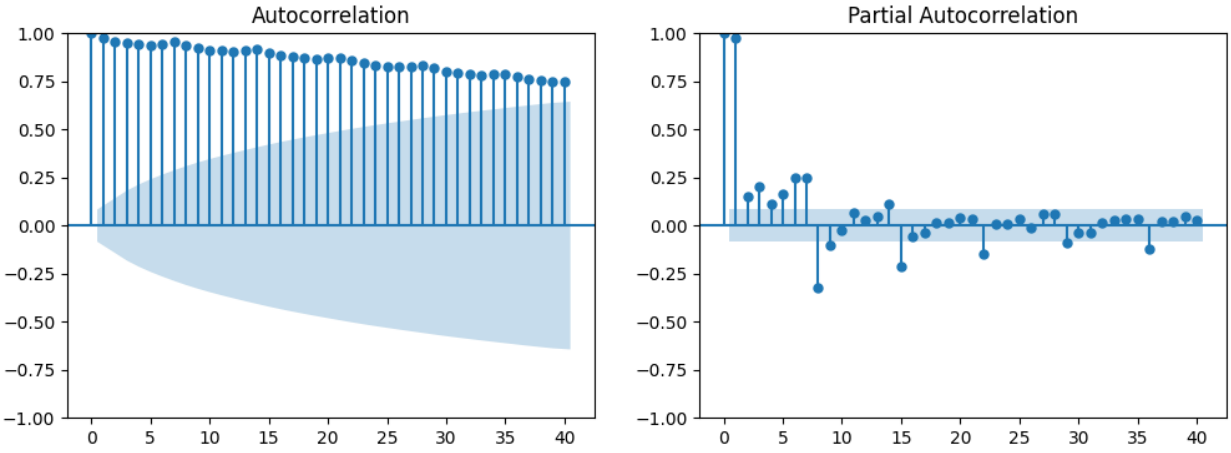

Seasonal Decomposition
Wikipedia:Hauptseite_de.wikipedia.org_all-access_all-agents
1e6
3.5
3.0
2.5
1e6
Trend
4
3
Seasonal
0
−200000
Resid
500000
0
−500000
2015-072015-092015-112016-012016-032016-052016-072016-092016-11



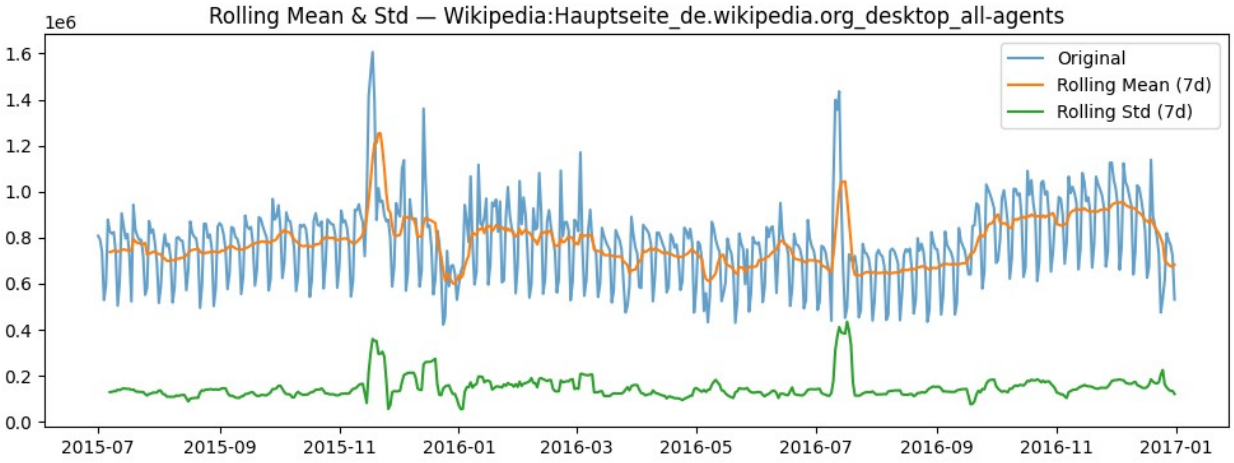

```
🔍 Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_mobile-web_all-agents
ADF Statistic: 0.316 | p-value: 0.9781 | Stationary: False
```

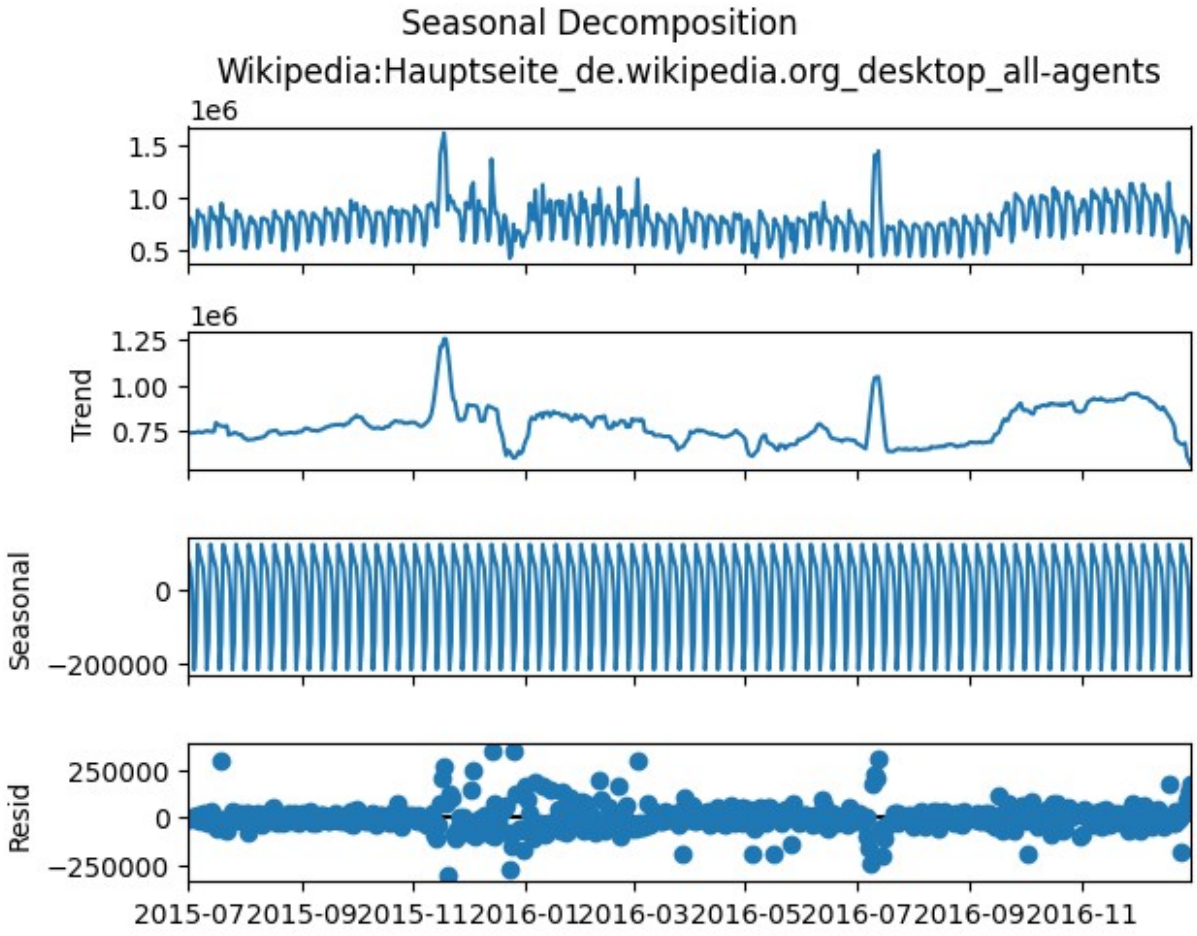


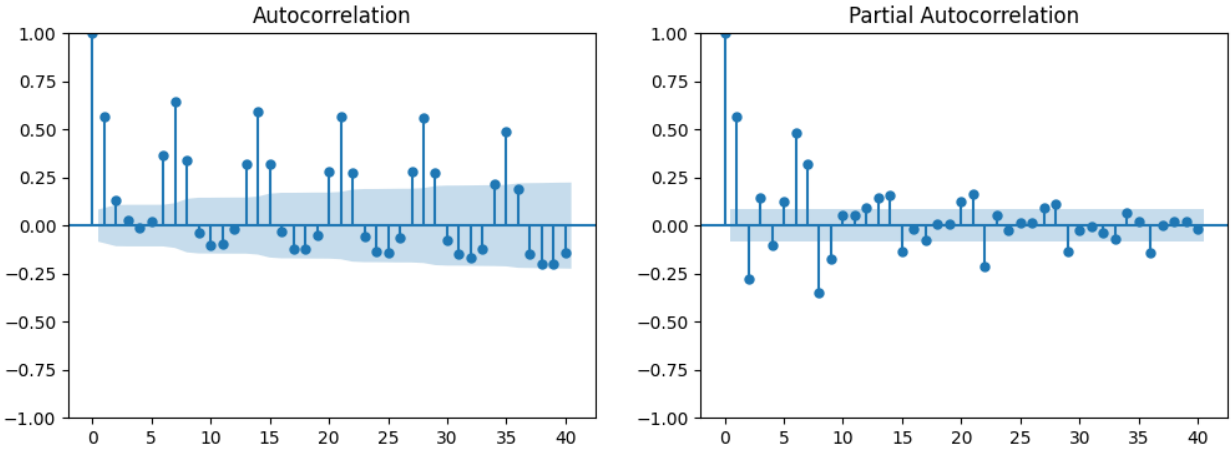




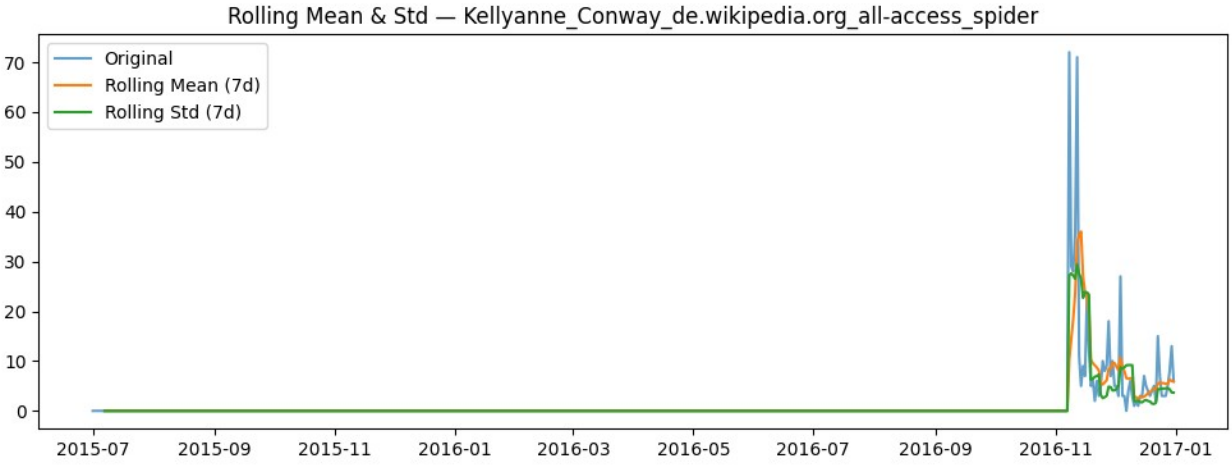

```
🔍 Analyzing page: Wikipedia:Hauptseite_de.wikipedia.org_desktop_all-
agents
ADF Statistic: -2.900 | p-value: 0.0453 | Stationary: True
```

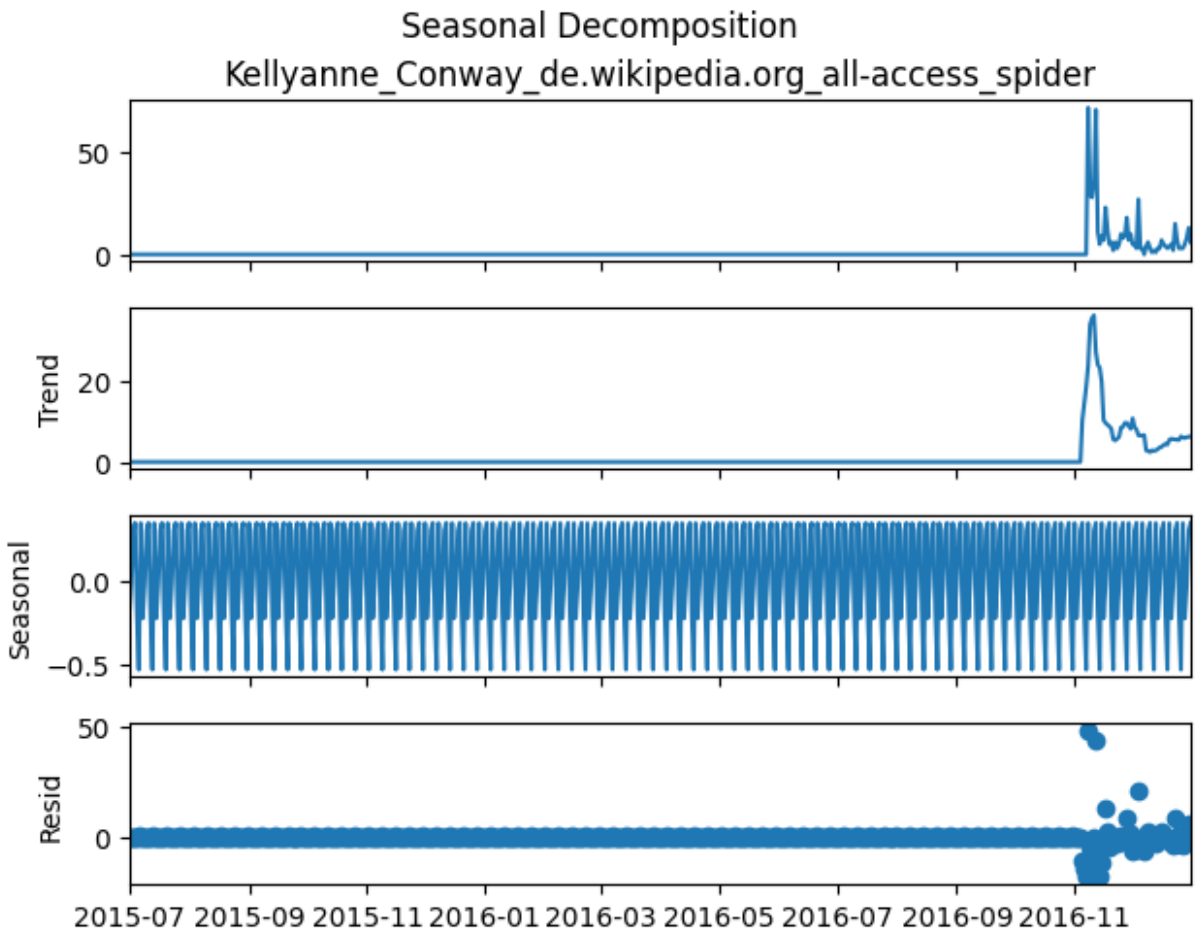

Rolling Mean & Std — Wikipedia:Hauptseite_de.wikipedia.org_desktop_all-agents
1e6
Original
Rolling Mean (7d)
Rolling Std (7d)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

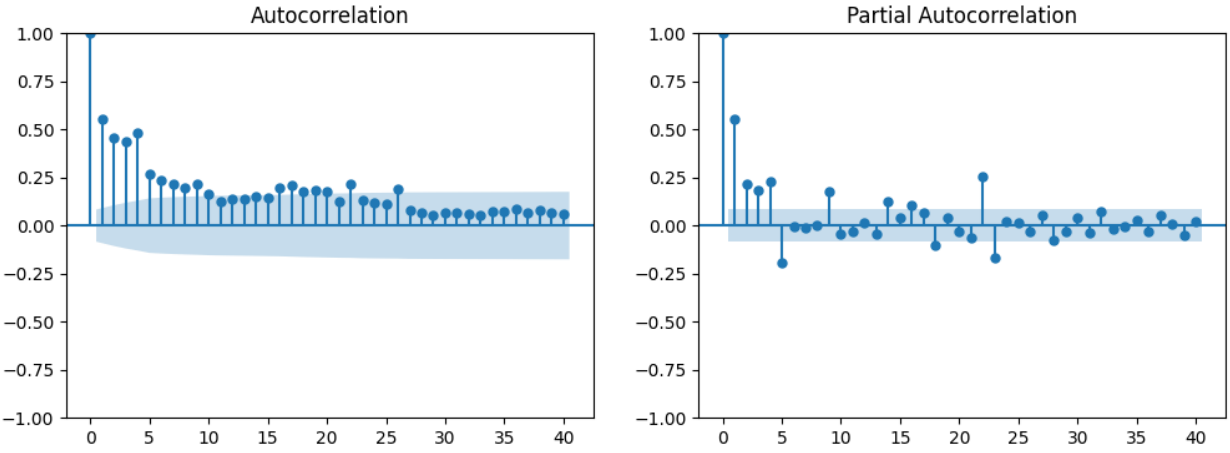

Seasonal Decomposition
Wikipedia:Hauptseite_de.wikipedia.org_desktop_all-agents
1e6
1.5
1.0
0.5
1e6
Trend
1.25
1.00
0.75
Seasonal
0
−200000
Resid
250000
0
−250000
2015-072015-092015-112016-012016-032016-052016-072016-092016-11



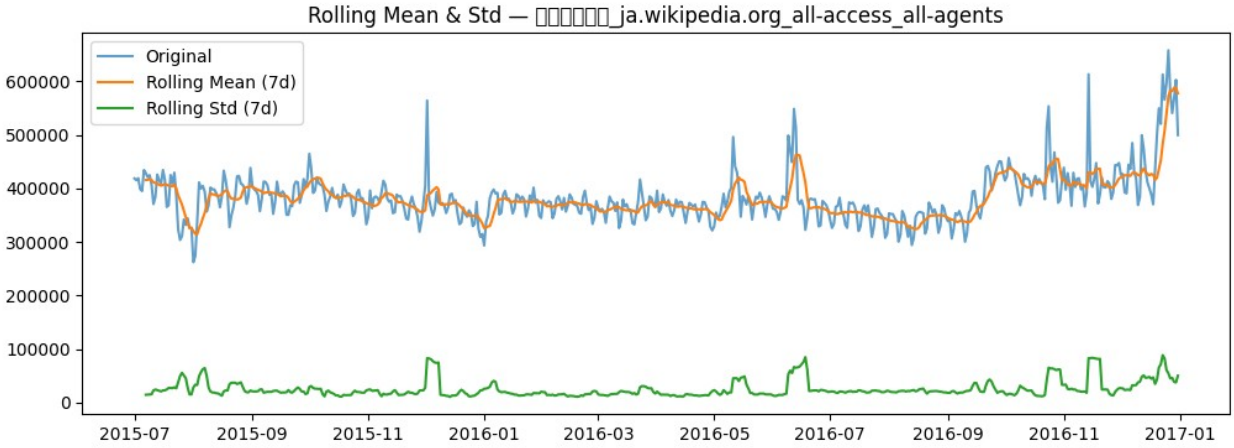

```
🔍 Analyzing page: Kellyanne_Conway_de.wikipedia.org_all-access_spider
ADF Statistic: -3.132 | p-value: 0.0243 | Stationary: True
```

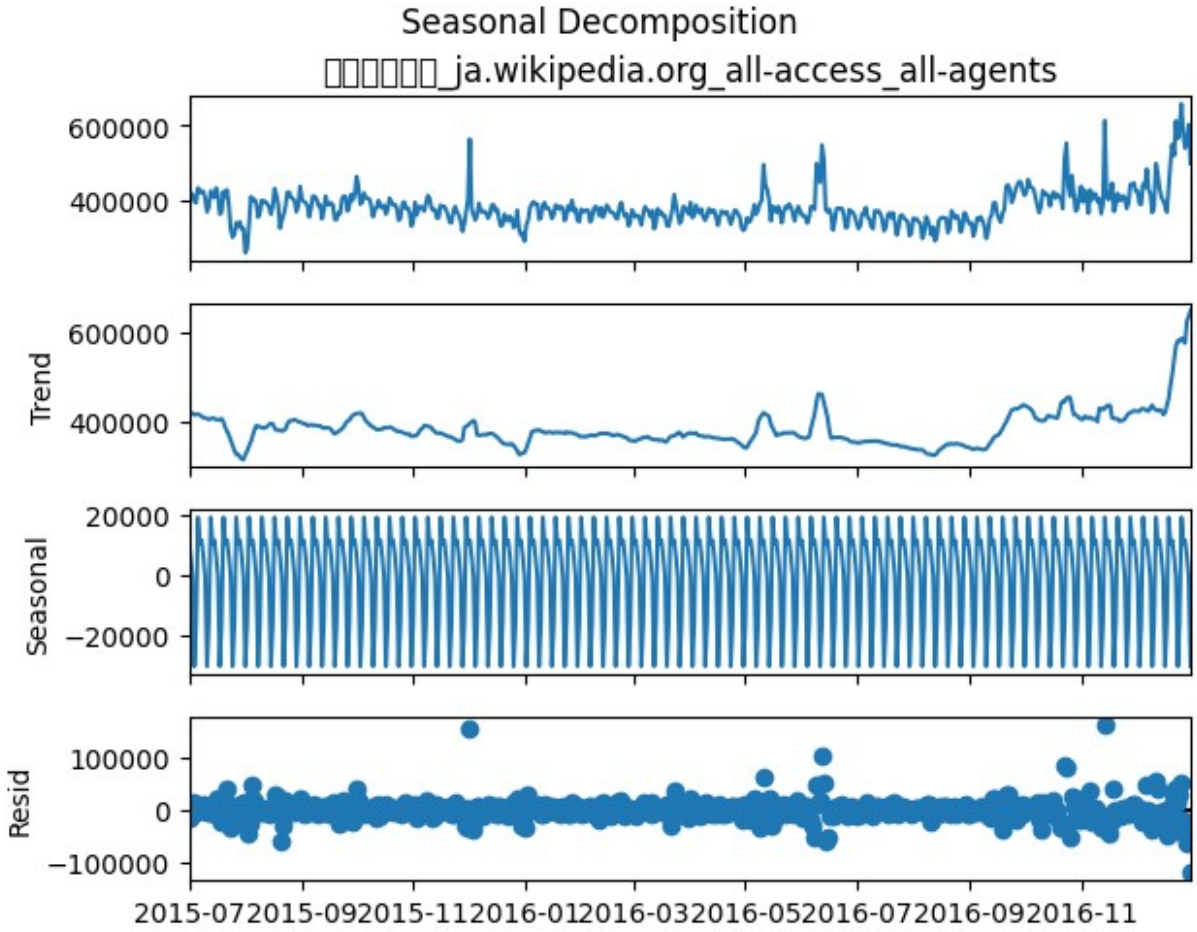


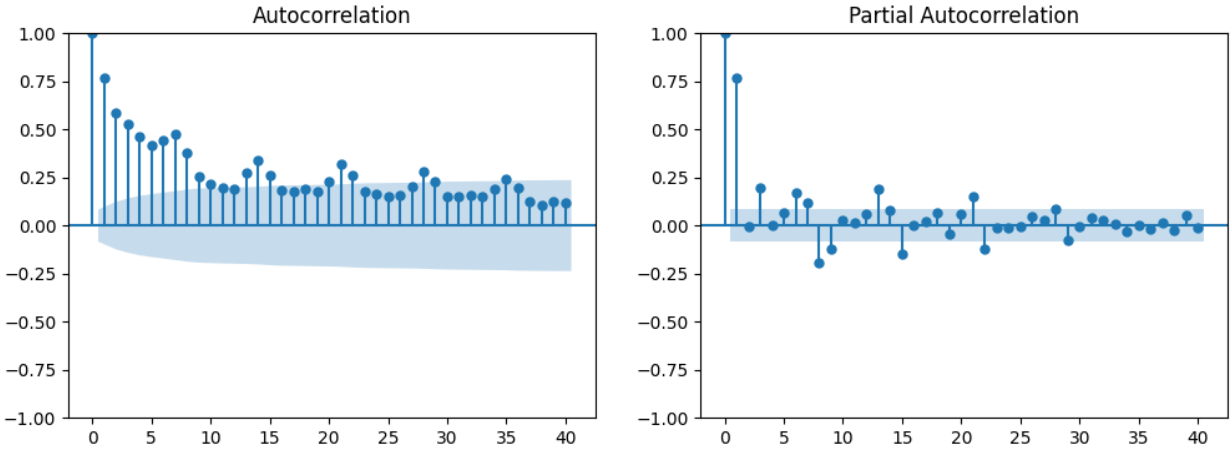




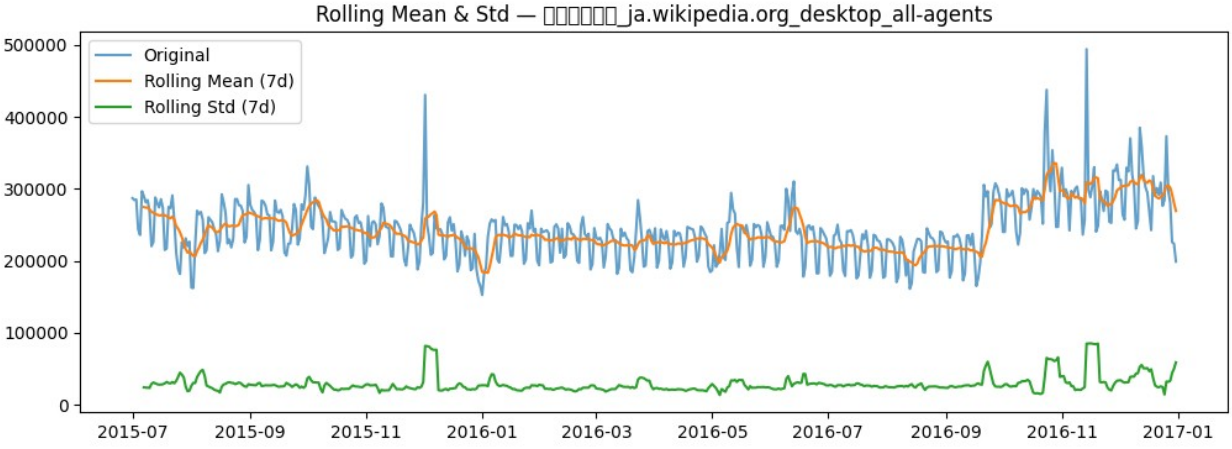

```
🔍 Analyzing page: メインページ_ja.wikipedia.org_all-access_all-agents
ADF Statistic: -1.429 | p-value: 0.5682 | Stationary: False
```

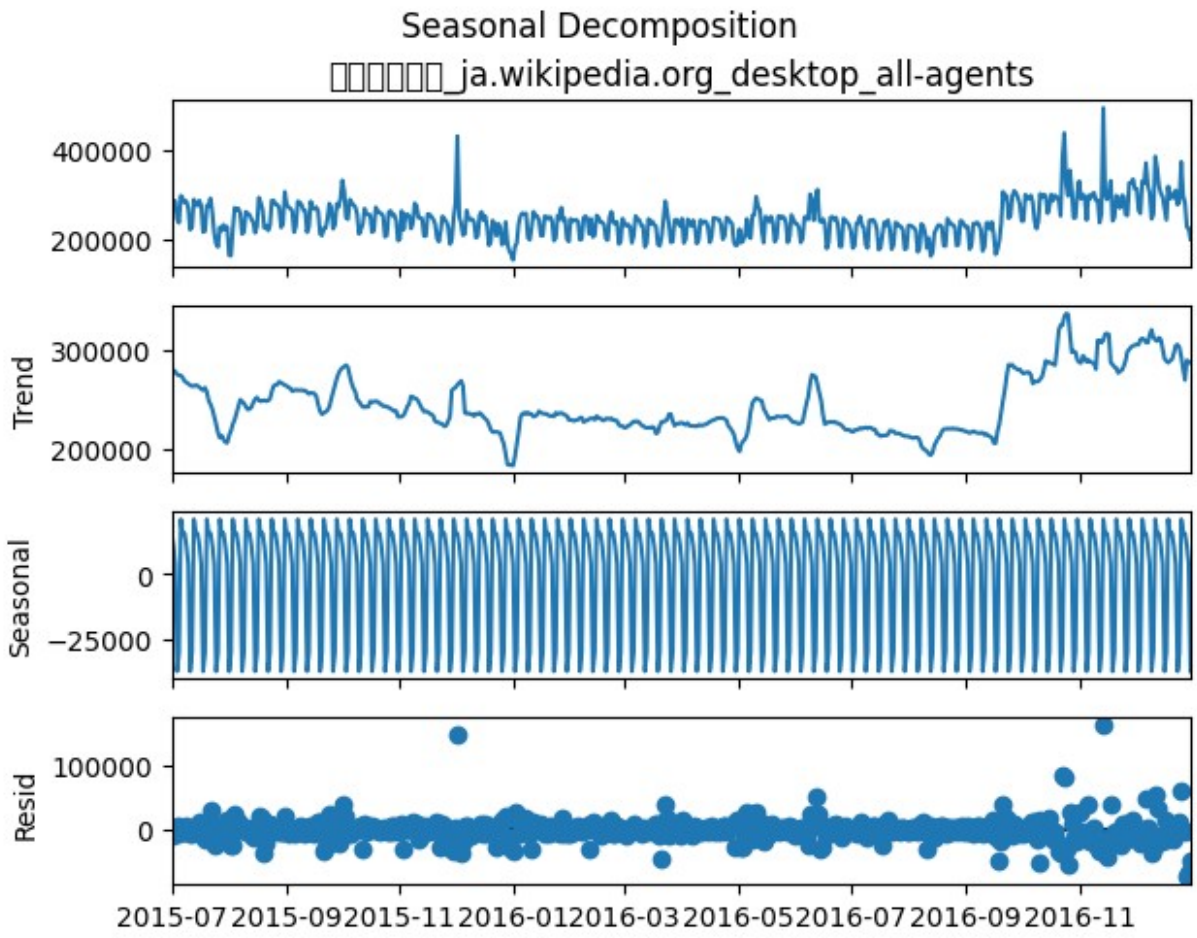

Rolling Mean & Std — □□□□□□_ja.wikipedia.org_all-access_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
600000
500000
400000
300000
200000
100000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

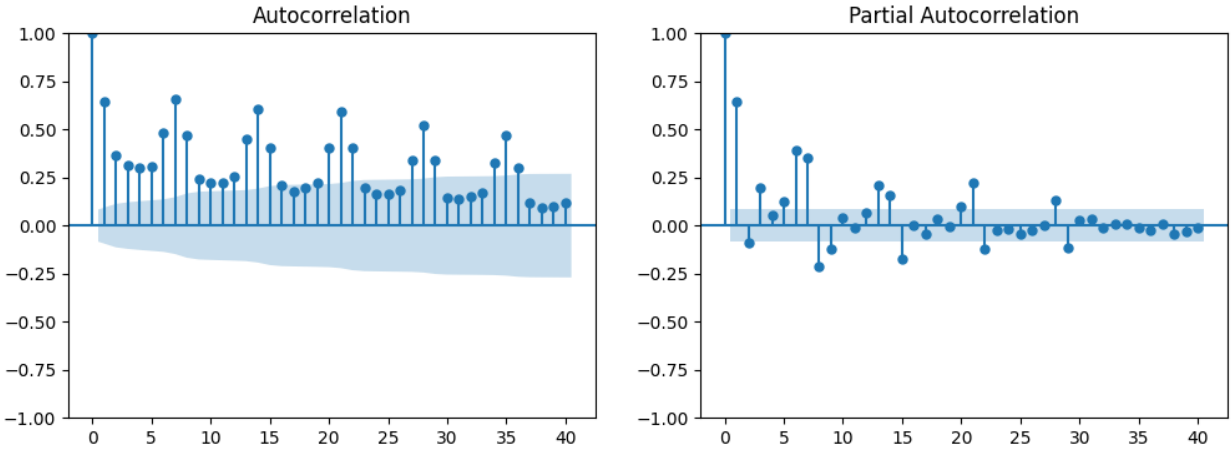

Seasonal Decomposition
□□□□□□_ja.wikipedia.org_all-access_all-agents
600000
400000
Trend
600000
400000
Seasonal
20000
0
−20000
Resid
100000
0
−100000
2015-072015-092015-112016-012016-032016-052016-072016-092016-11



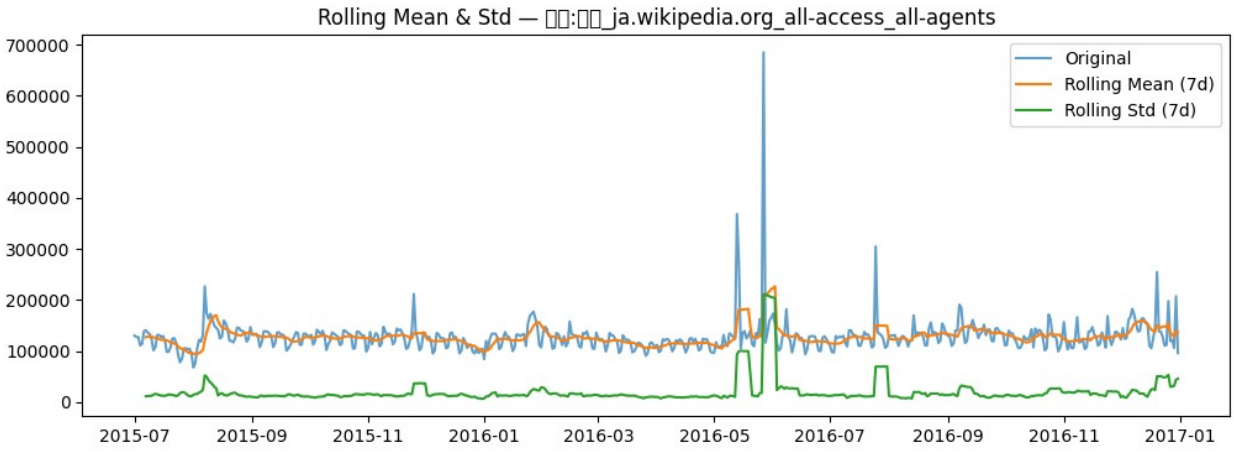

```
 Analyzing page: メインページ_ja.wikipedia.org_desktop_all-agents
ADF Statistic: -2.472 | p-value: 0.1225 | Stationary: False
```

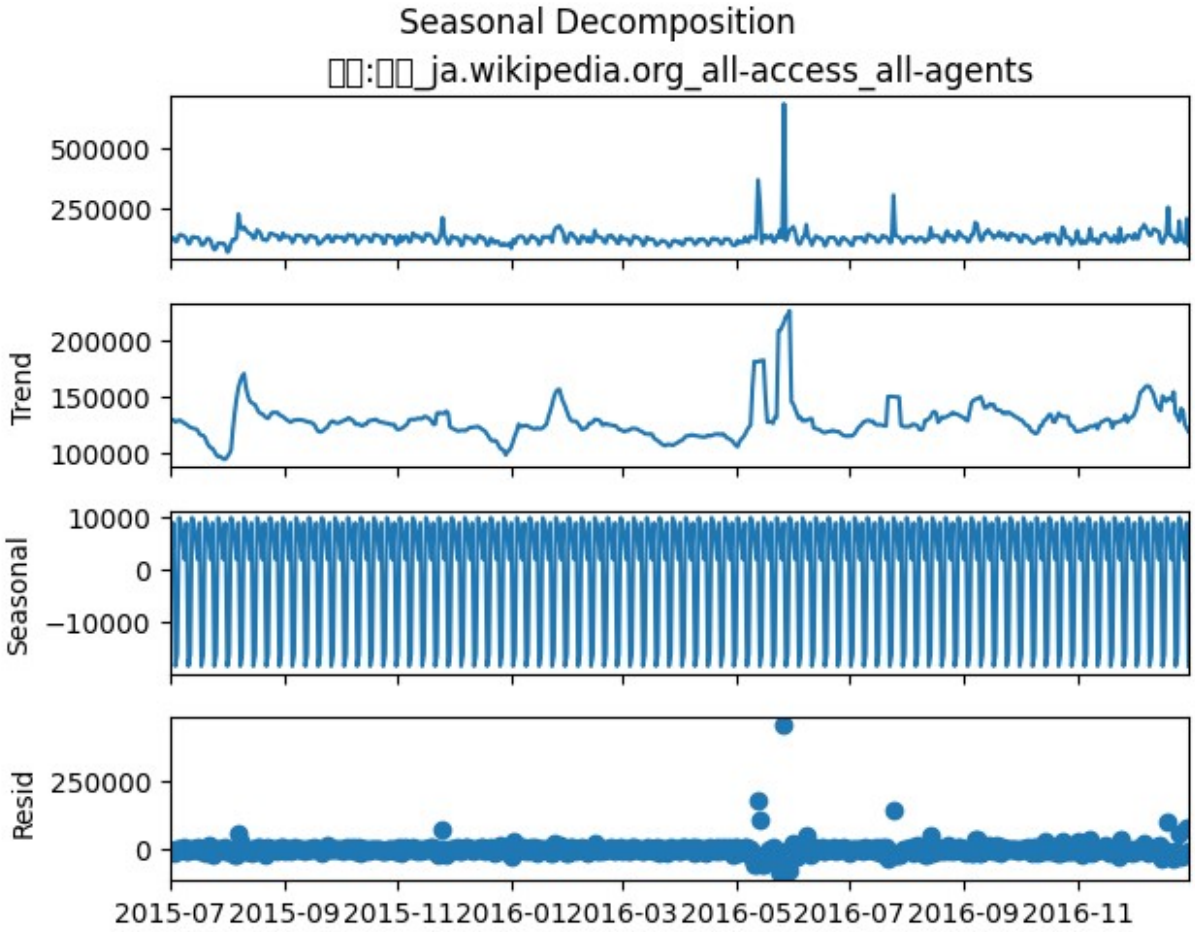


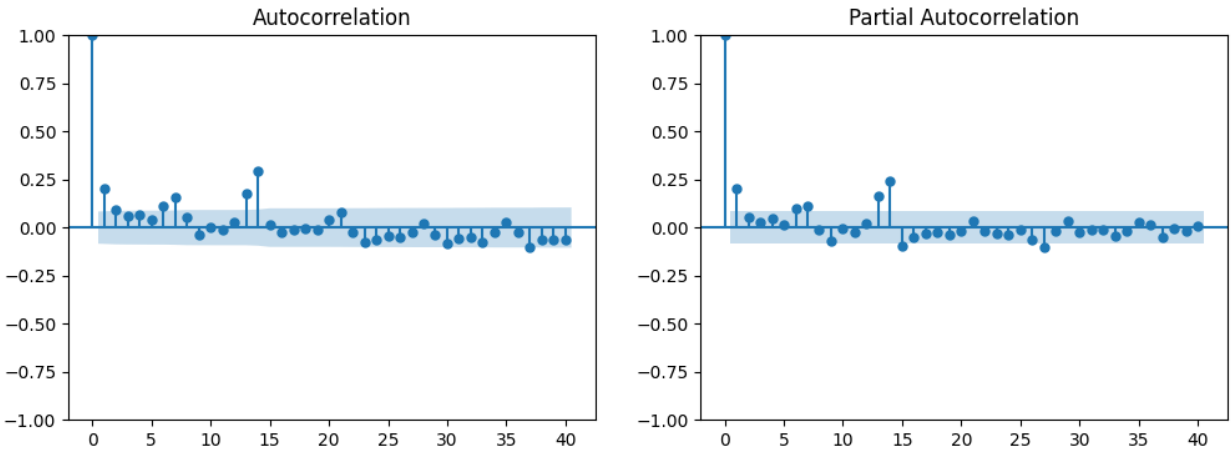




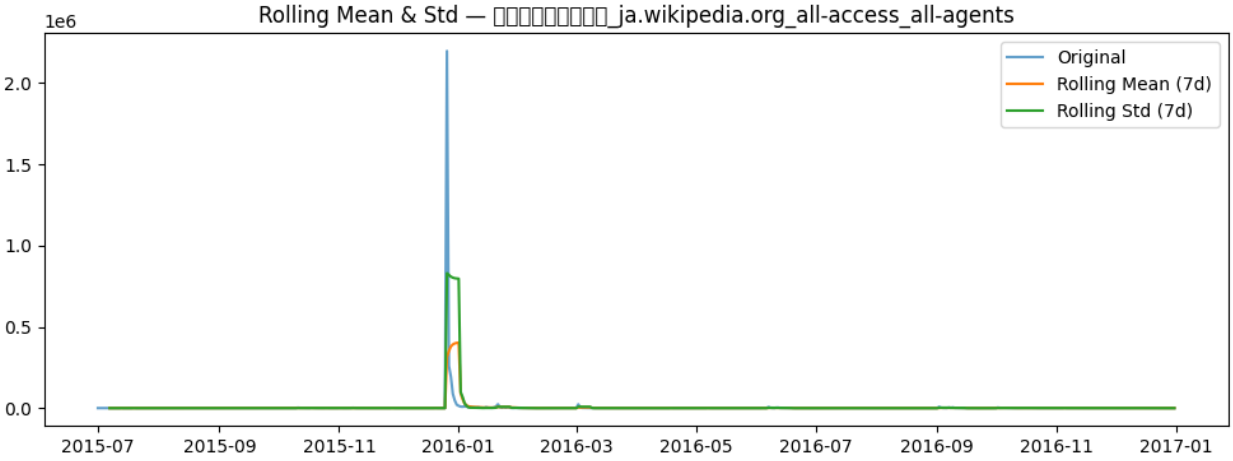

```
□ Analyzing page: 特別:検索_ja.wikipedia.org_all-access_all-agents
ADF Statistic: -3.855 | p-value: 0.0024 | Stationary: True
```

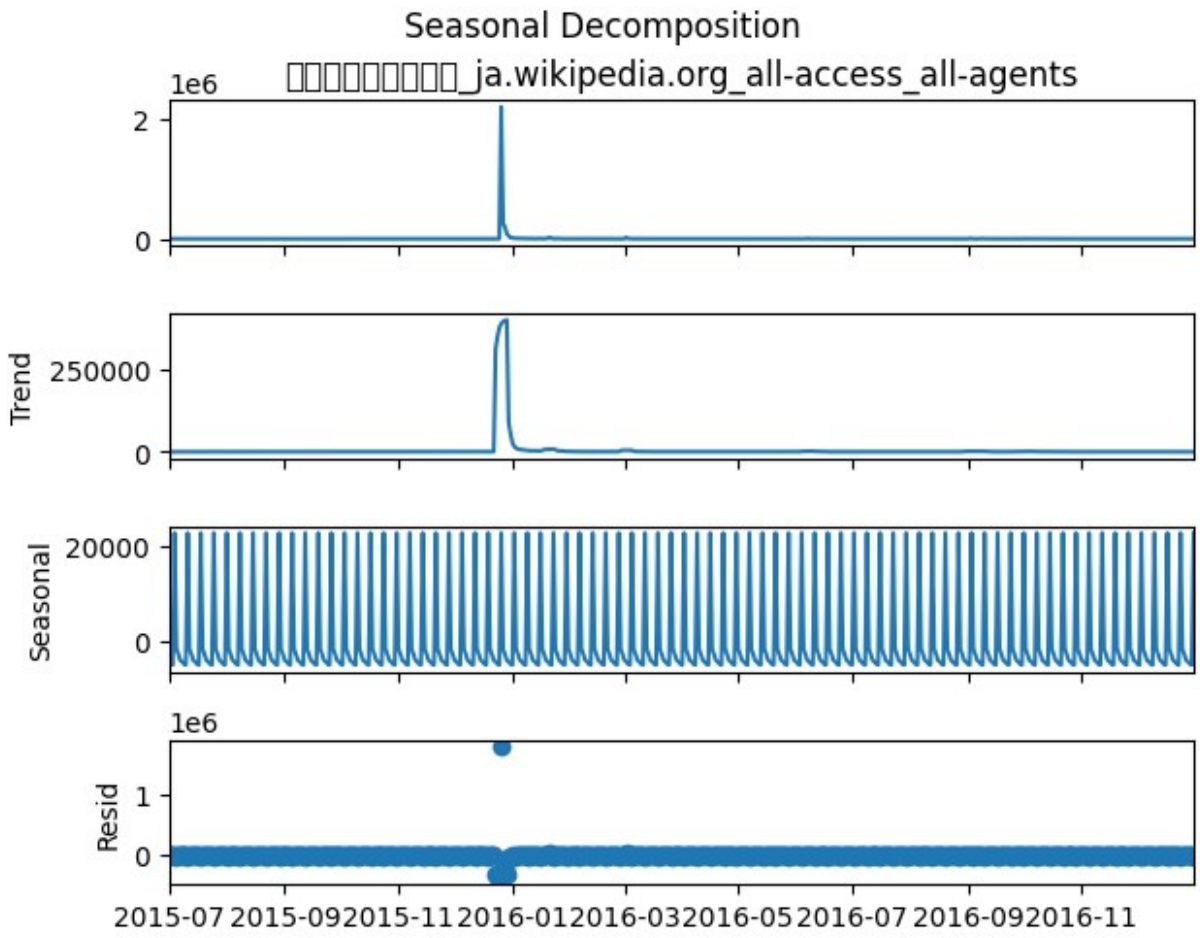

Rolling Mean & Std — □□:□□_ja.wikipedia.org_all-access_all-agents
Original
Rolling Mean (7d)
Rolling Std (7d)
700000
600000
500000
400000
300000
200000
100000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

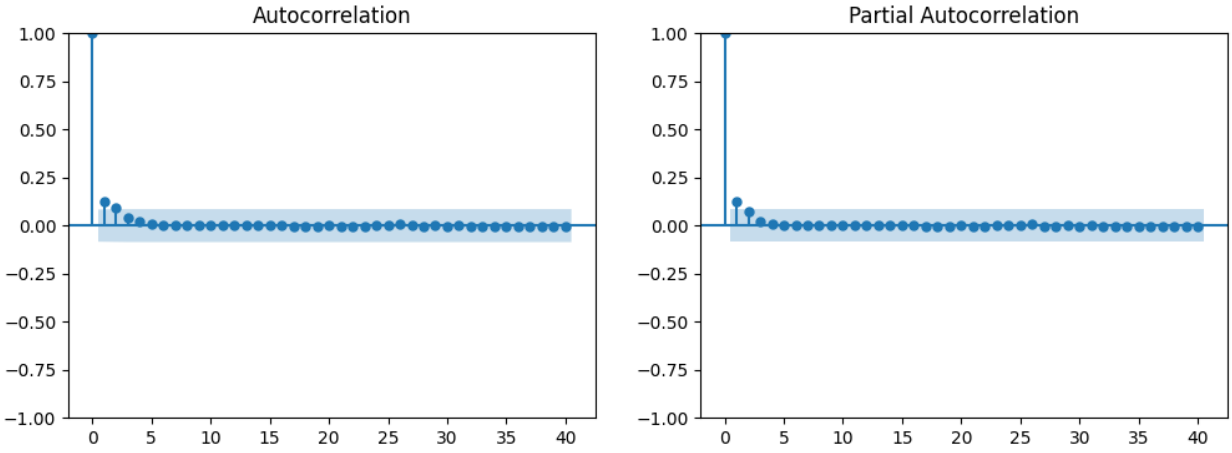

Seasonal Decomposition
□□:□□_ja.wikipedia.org_all-access_all-agents
500000
250000
Trend
200000
150000
100000
Seasonal
10000
0
−10000
Resid
250000
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11



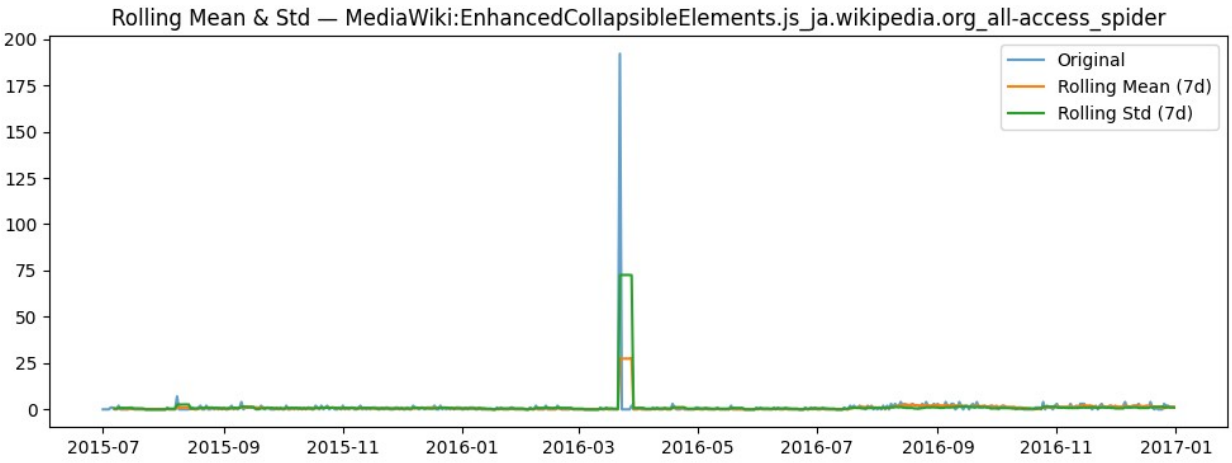

```
 Analyzing page: キングオブコメディ_ja.wikipedia.org_all-access_all-agents
ADF Statistic: -14.300 | p-value: 0.0000 | Stationary: True
```

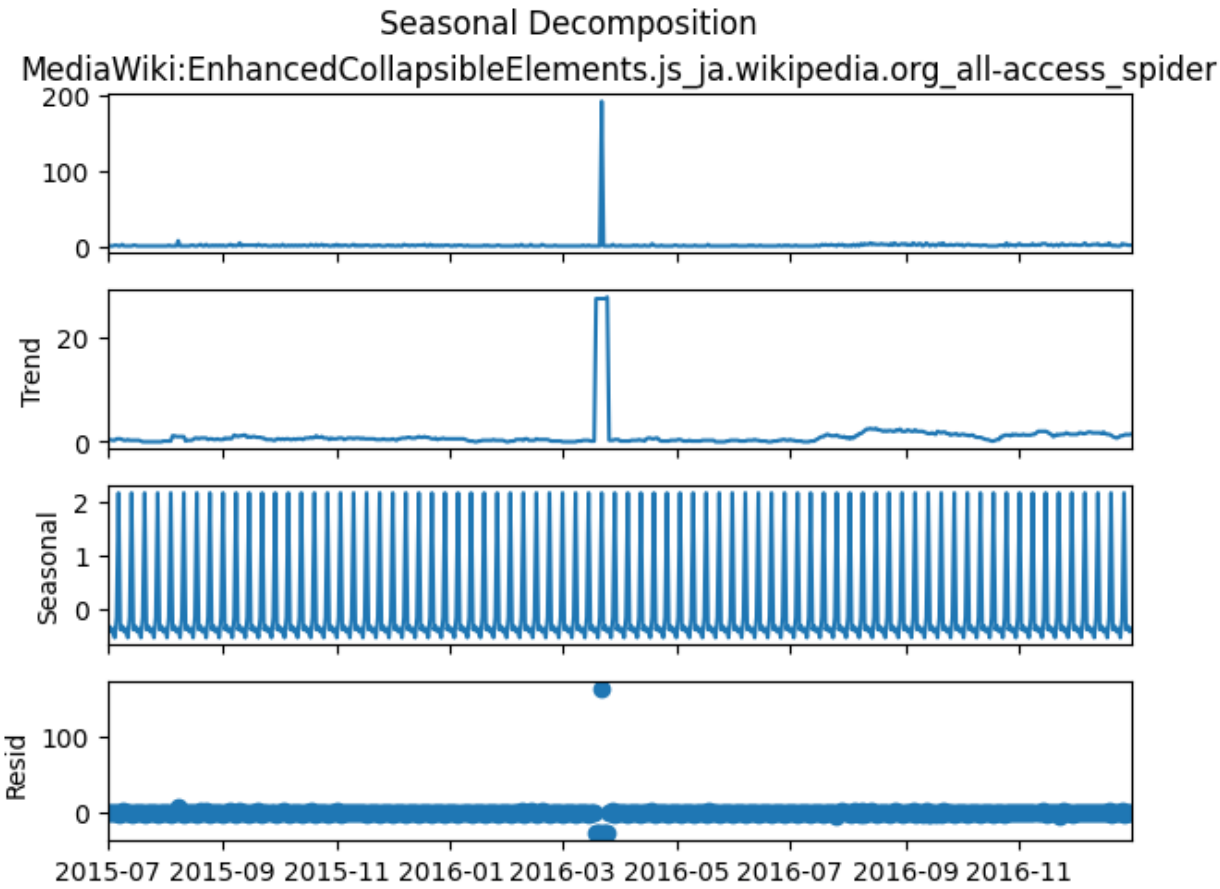


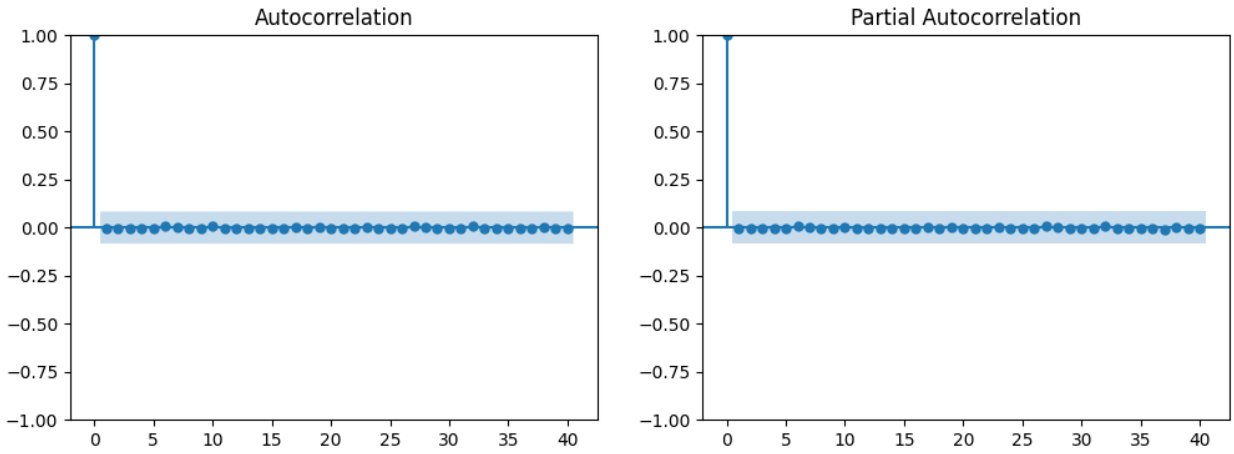




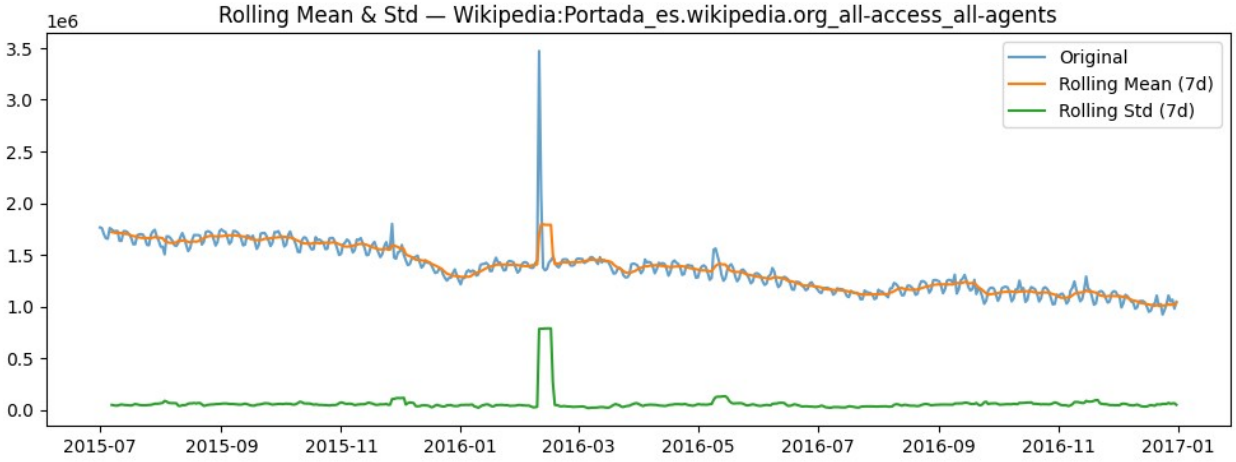

```
□ Analyzing page:
MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-
access_spider
ADF Statistic: -23.524 | p-value: 0.0000 | Stationary: True
```

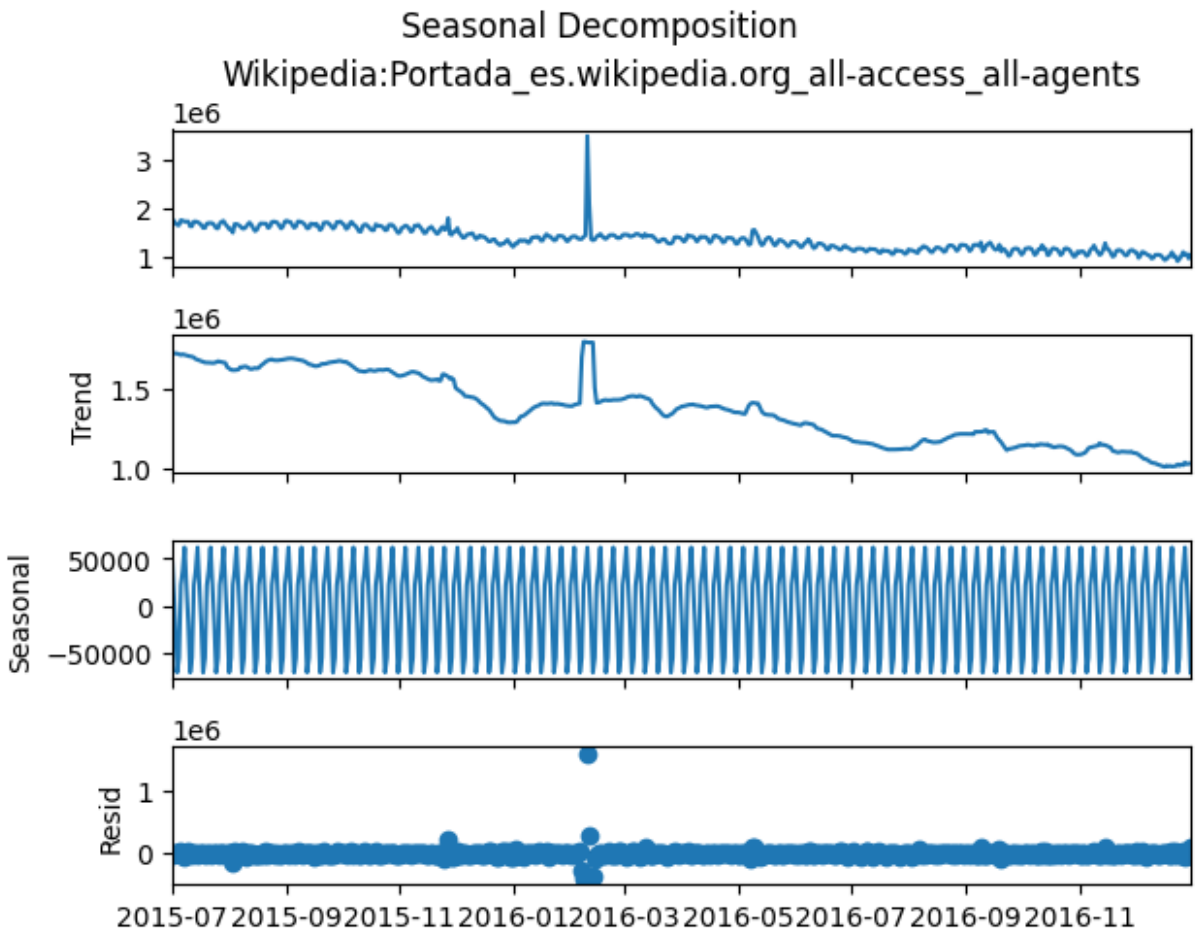

Rolling Mean & Std — MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-access_spider
Original
Rolling Mean (7d)
Rolling Std (7d)
200
175
150
125
100
75
50
25
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

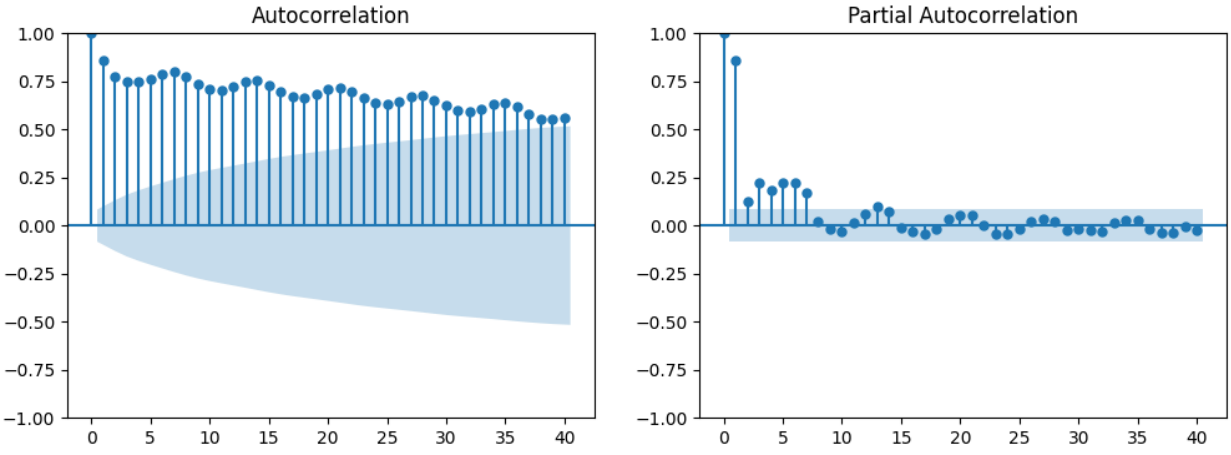

Seasonal Decomposition
MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org_all-access_spider
200
100
0
Trend
20
0
Seasonal
2
1
0
Resid
100
0
2015-07 2015-09 2015-11 2016-01 2016-03 2016-05 2016-07 2016-09 2016-11



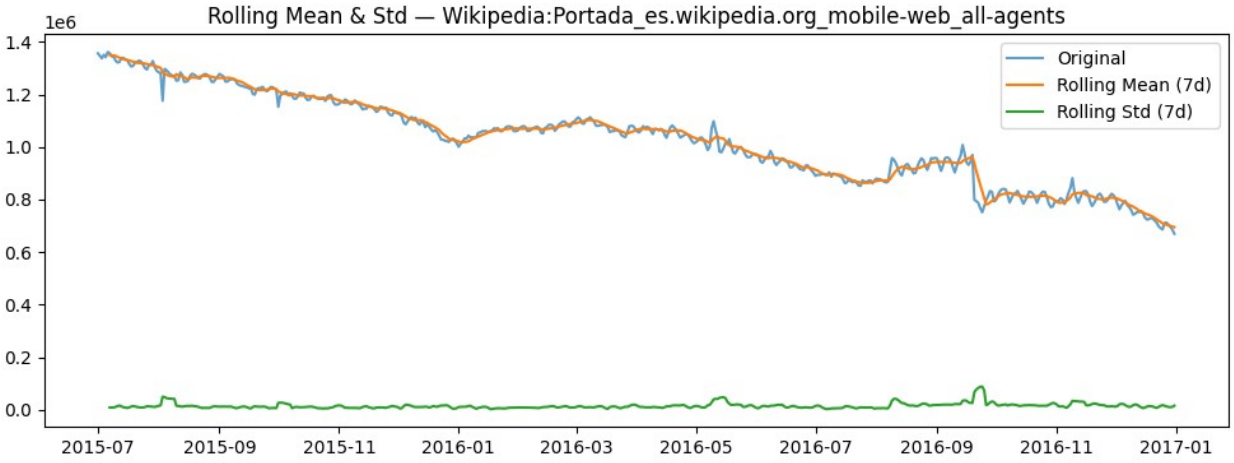

```
🔍 Analyzing page: Wikipedia:Portada_es.wikipedia.org_all-access_all-agents
ADF Statistic: -1.061 | p-value: 0.7304 | Stationary: False
```

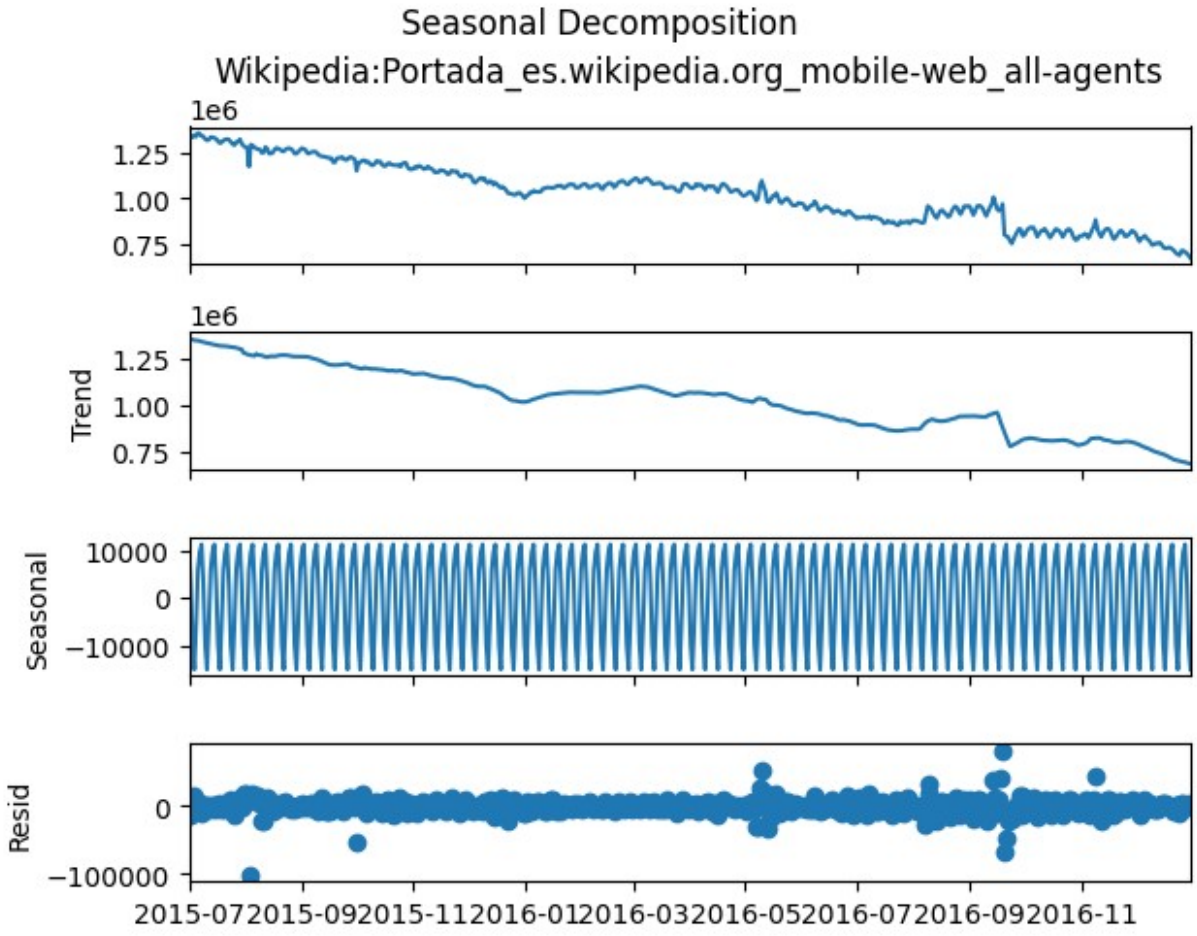


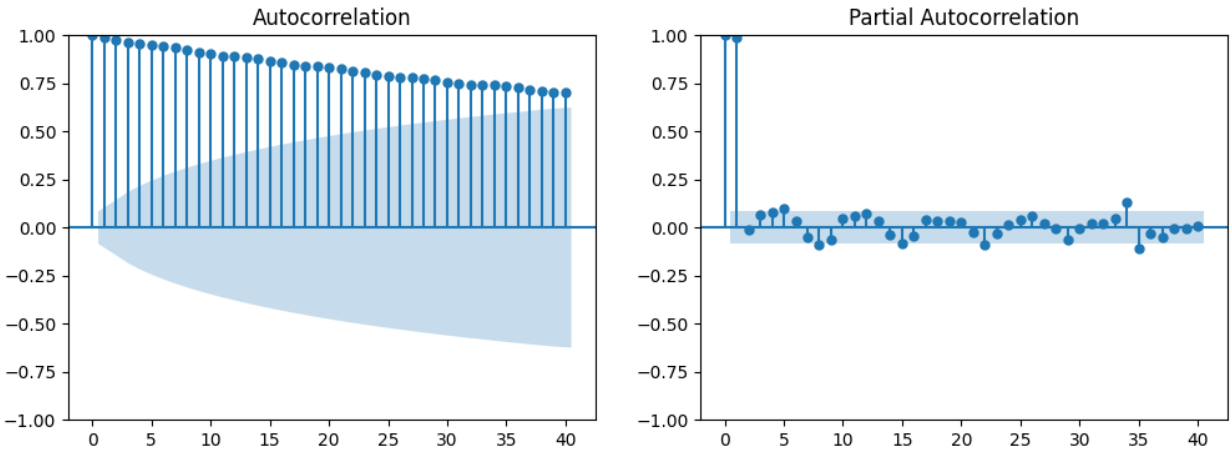




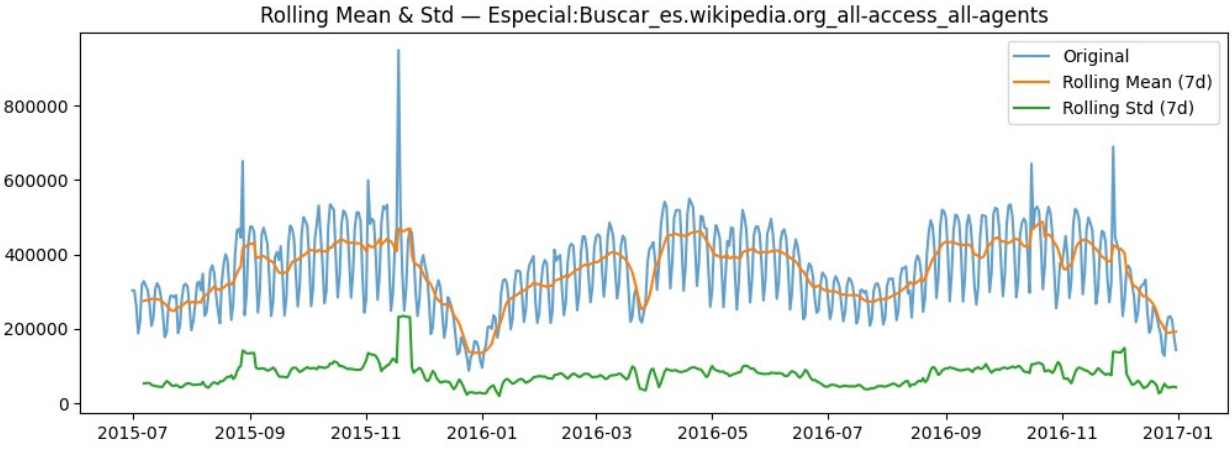

```
🔍 Analyzing page: Wikipedia:Portada_es.wikipedia.org_mobile-web_all-
agents
ADF Statistic: -0.434 | p-value: 0.9042 | Stationary: False
```

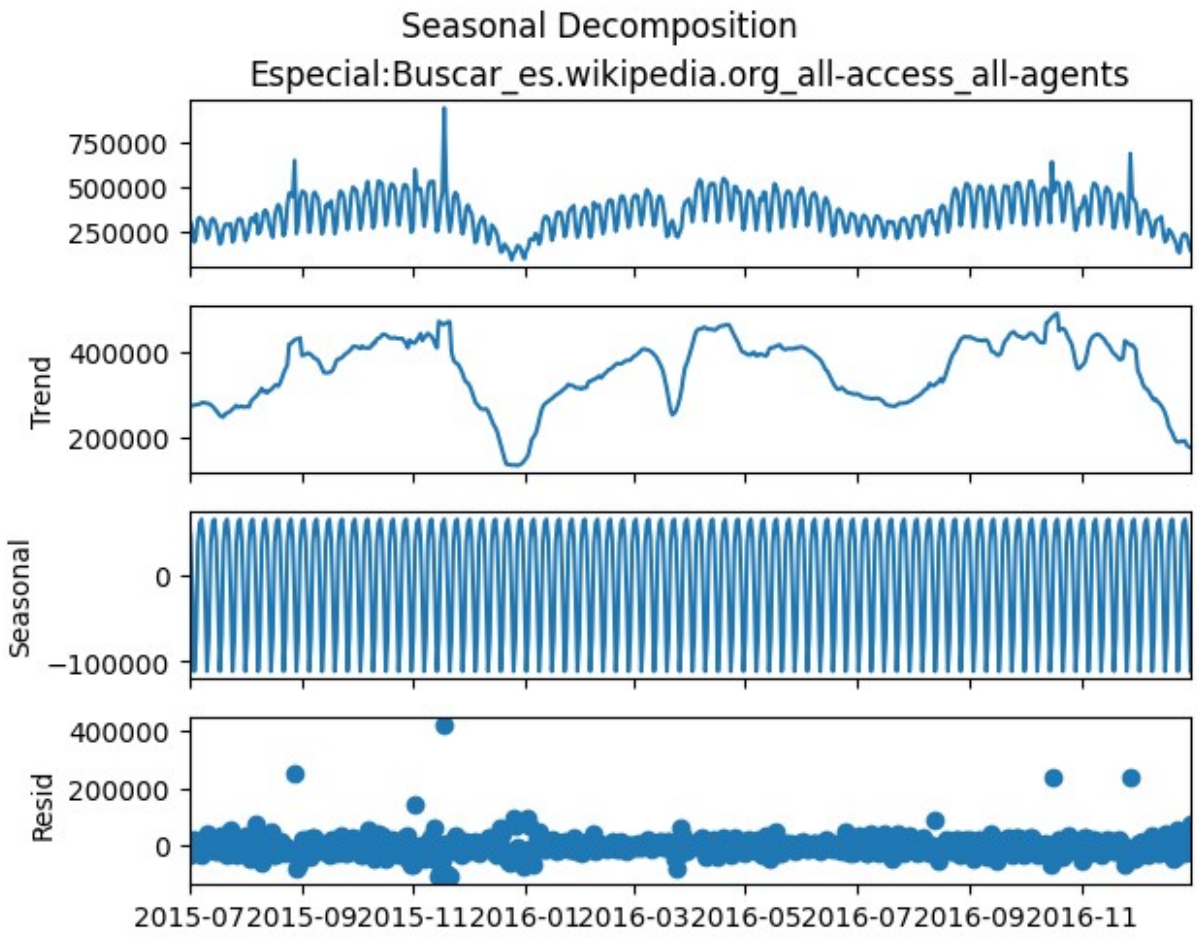

Rolling Mean & Std — Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Original
Rolling Mean (7d)
Rolling Std (7d)
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

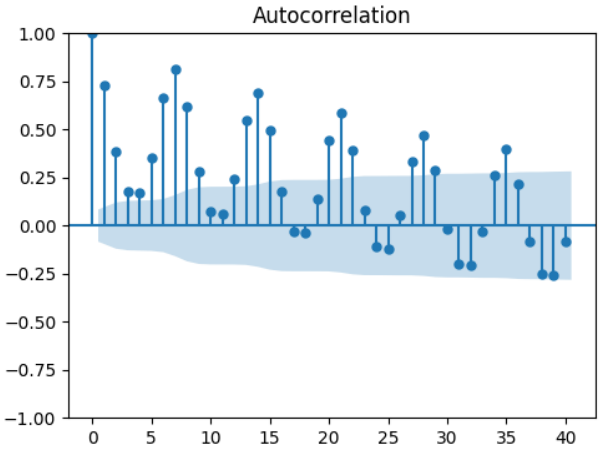

Seasonal Decomposition
Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents
1e6
1.25
1.00
0.75
Trend
Seasonal
10000
0
−10000
Resid
−100000
2015-072015-092015-112016-012016-032016-052016-072016-092016-11

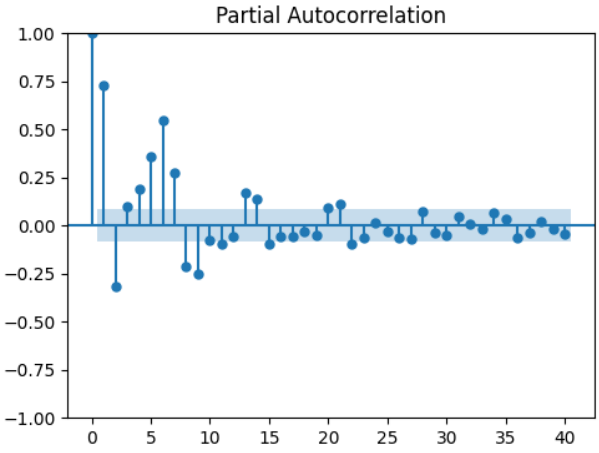



```
 Analyzing page: Especial:Buscar_es.wikipedia.org_all-access_all-agents
ADF Statistic: -2.829 | p-value: 0.0543 | Stationary: False
```

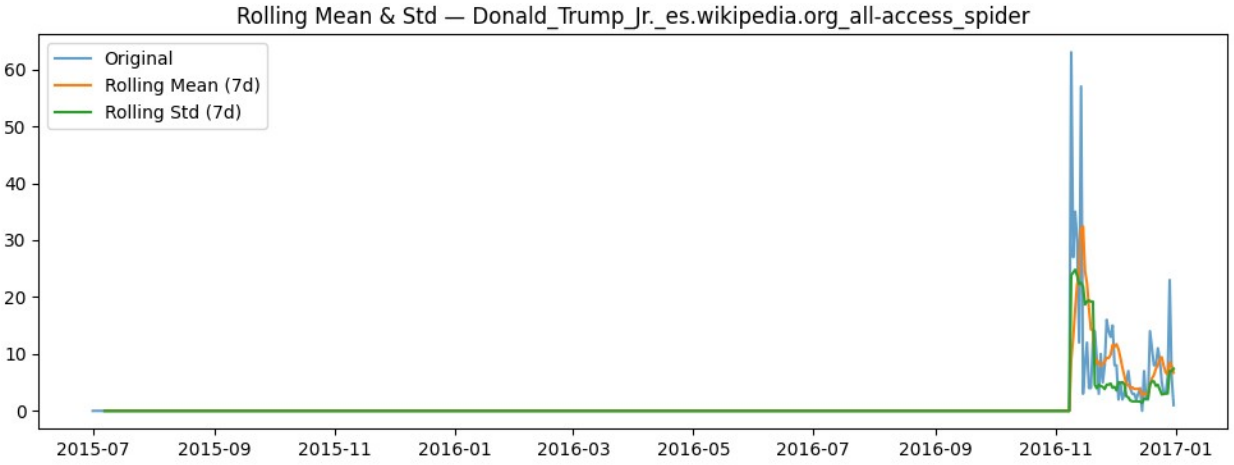


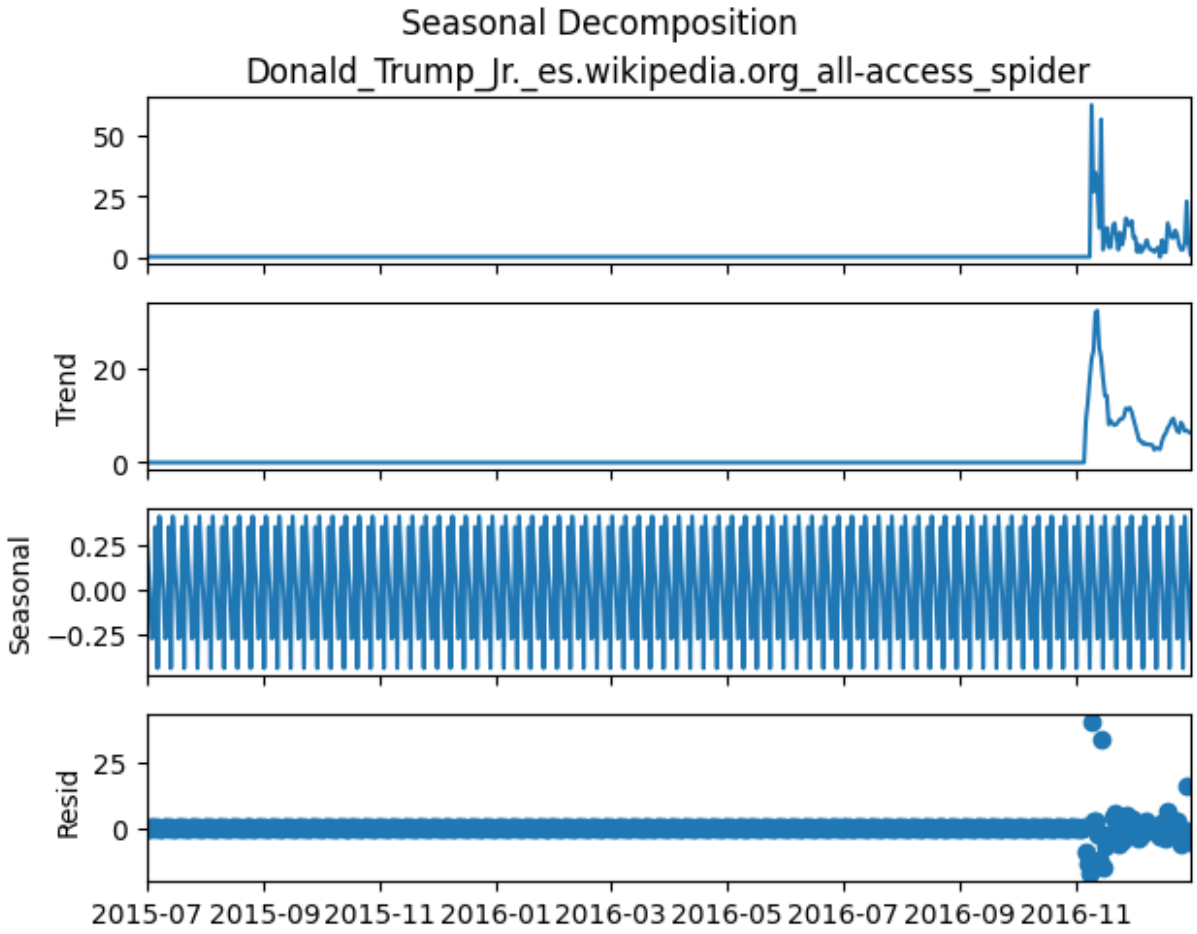


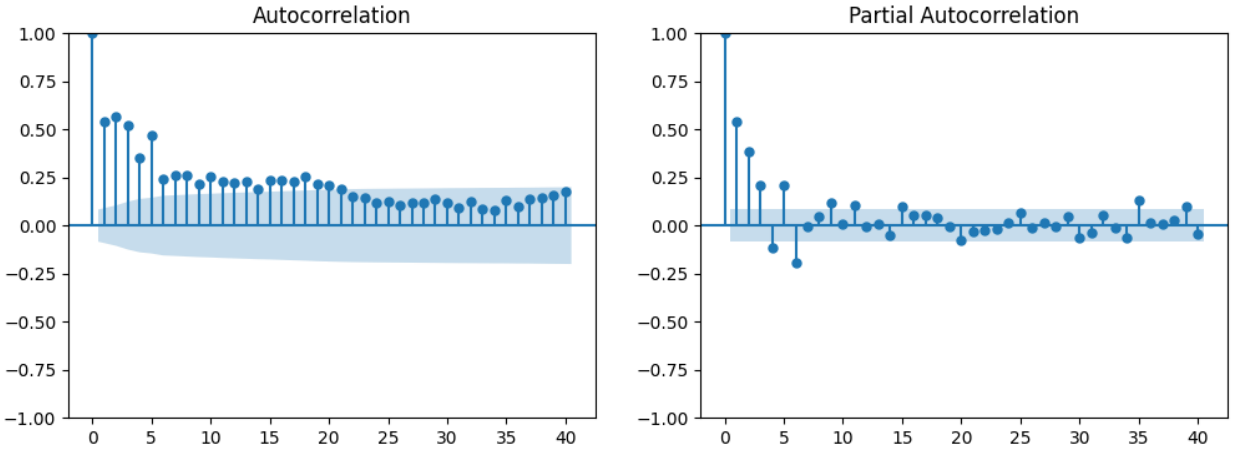





```
 Analyzing page: Donald_Trump_Jr._es.wikipedia.org_all-access_spider
ADF Statistic: -3.169 | p-value: 0.0218 | Stationary: True
```


Rolling Mean & Std — Donald_Trump_Jr._es.wikipedia.org_all-access_spider
Original
Rolling Mean (7d)
Rolling Std (7d)
60
50
40
30
20
10
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


Seasonal Decomposition
Donald_Trump_Jr._es.wikipedia.org_all-access_spider
50
25
0
Trend
20
0
Seasonal
0.25
0.00
−0.25
Resid
25
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11




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


```
{"summary":"{\n  \"name\": \"stationarity_summary\",\n  \"rows\": 39,\
n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 39,\n        \"samples\": [\n          \"\\
u30ad\\u30f3\\u30b0\\u30aa\\u30d6\\u30b3\\u30e1\\u30c7\\
u30a3_ja.wikipedia.org_all-access_all-agents\",\n
\"Wikipedia:Portada_es.wikipedia.org_mobile-web_all-agents\",\n
\"\\u6d2a\\u6668\\u7a4e_zh.wikipedia.org_all-access_spider\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"adf_stat\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.924169161947759,\n        \"min\": -
23.524375337769687,\n        \"max\": 0.33608427784064865,\n
\"num_unique_values\": 39,\n        \"samples\": [\n          -
14.300235623036485,\n          -0.4343665203429093,\n          -
1.965599868270614\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pvalue\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.3382990137104211,\n        \"min\": 0.0,\n        \"max\":
0.978940774167741,\n        \"num_unique_values\": 39,\n
\"samples\": [\n          1.2326230373397335e-26,\n
0.9042007435665997,\n          0.3018569825772056\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"stationary\",\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      \"column\":
\"crit\",\n      \"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\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.

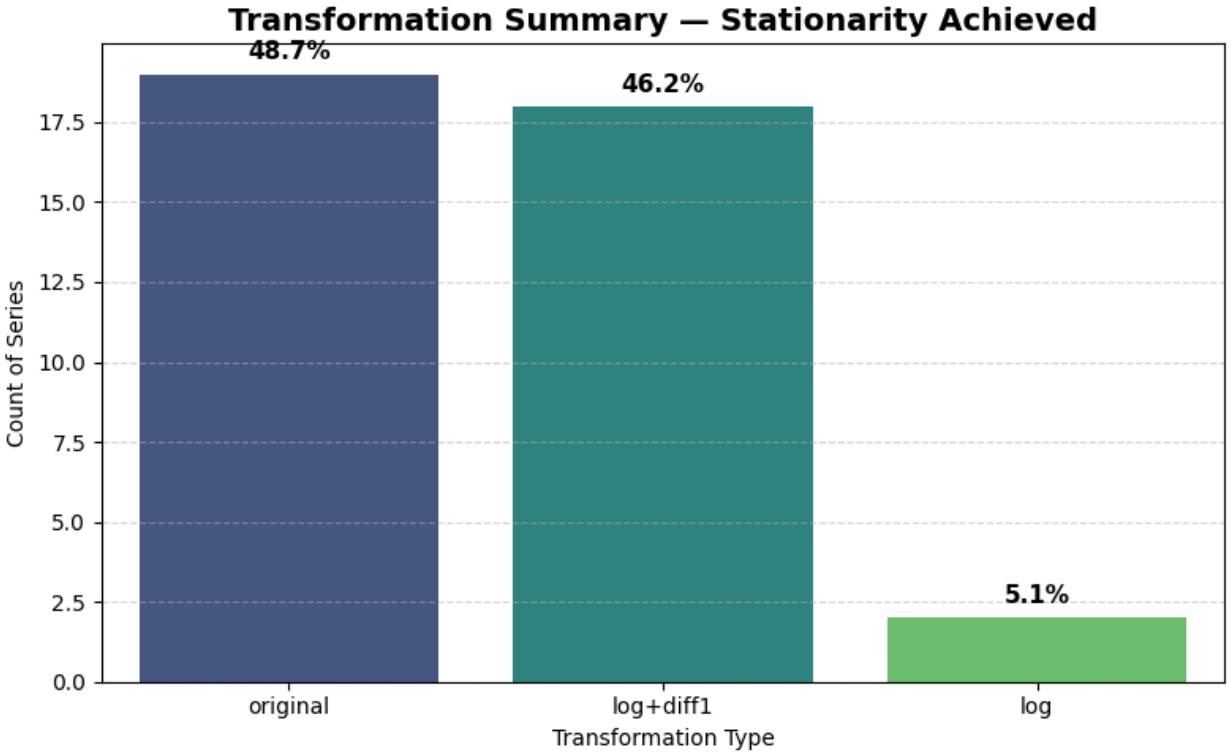
If you skip checking stationarity, the model might:

- **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":"{\n  \"name\": \"transformation_counts\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n      \"column\": \"Transformation\",\n\"properties\": {\n        \"dtype\": \"string\",\n\"num_unique_values\": 3,\n        \"samples\": [\n\"original\",\n          \"log+diff1\",\n          \"log\"\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n\"std\": 9,\n        \"min\": 2,\n        \"max\": 19,\n\"num_unique_values\": 3,\n        \"samples\": [\n          19,\n18,\n          2\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Percentage\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 24.48271499105713,\n        \"min\": 5.1,\n        \"max\": 48.7,\n        \"num_unique_values\": 3,\n\"samples\": [\n          48.7,\n          46.2,\n          5.1\n],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    }\n  ]\n}","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-
```

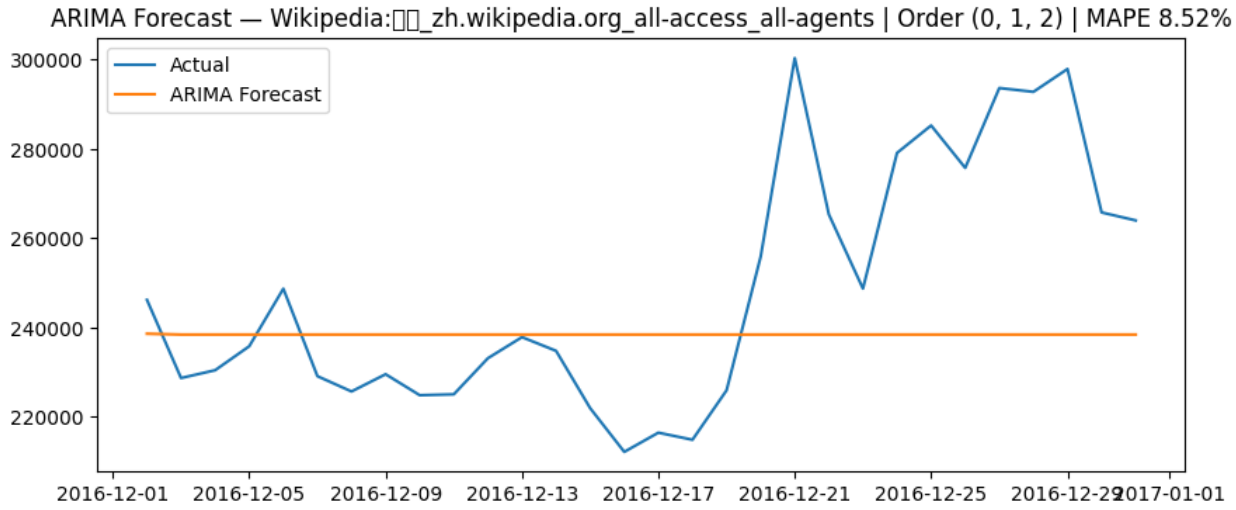
```
space)
                pred_trans = res.forecast(steps=test_horizon)
                # invert transform if log used
                if use_log:
                    pred_orig = np.expm1(pred_trans)  # invert log1p
                else:
                    pred_orig = pred_trans

                # 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.or
g_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":"{\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      }\
n    },\n    {\n      \"column\": \"BestOrder\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
```

\"num_unique_values\": 16,\n   \"samples\": [\n   [\n
0,\n   1,\n   0\n   ],\n   [\n
1,\n   1,\n   0\n   ],\n   [\n
2,\n   0,\n   2\n   ]\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\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
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"arima_df"}

31

arima_df.shape

(39, 3)

arima_df.head()

{"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   }\
n   },\n   {\n   \"column\": \"BestOrder\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 16,\n   \"samples\": [\n   [\n
0,\n   1,\n   0\n   ],\n   [\n
1,\n   1,\n   0\n   ],\n   [\n
2,\n   0,\n   2\n   ]\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\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
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"arima_df"}

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

- Incorporate **campaign variable** as exogenous input.

- Evaluate forecast accuracy against baseline models.

```
campaign.sample(5)
```

```
{"summary":"{\n  \"name\": \"campaign\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Exog\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\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":"{\n  \"name\": \"exog_series\",\n  \"rows\": 550,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2015-07-01 00:00:00\",\n        \"max\": \"2016-12-31 00:00:00\",\n
\"num_unique_values\": 550,\n        \"samples\": [\n          \"2016-
01-12 00:00:00\",\n          \"2015-09-18 00:00:00\",\n
\"2016-10-23 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Exog\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
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
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BestOrder\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n        \"samples\": [\n          [\n
2,\n            0,\n            0\n          ],\n          [\n
2,\n            1,\n            0\n          ],\n          [\n
2,\n            0,\n            1\n          ]\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BestSeasonal\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          [\n
0,\n            0,\n            0,\n            0\n          ],\n
[\n            1,\n            0,\n            1,\n            7\n
]\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\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
\"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"}
```


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


```
exog_series.shape
```

```
(550, 1)
```

```
exog_series.head()
```

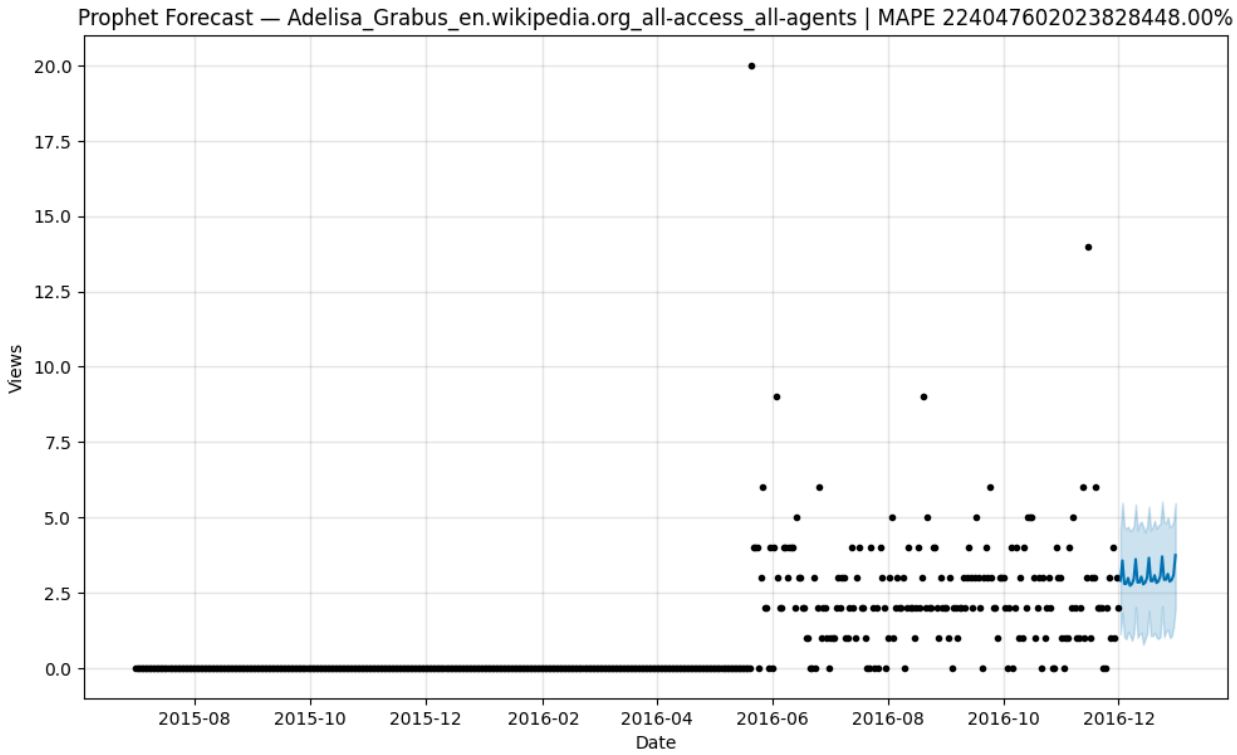

```
{"summary":"{\n  \"name\": \"exog_series\",\n  \"rows\": 550,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2015-07-01 00:00:00\",\n        \"max\": \"2016-12-31 00:00:00\",\n
\"num_unique_values\": 550,\n        \"samples\":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

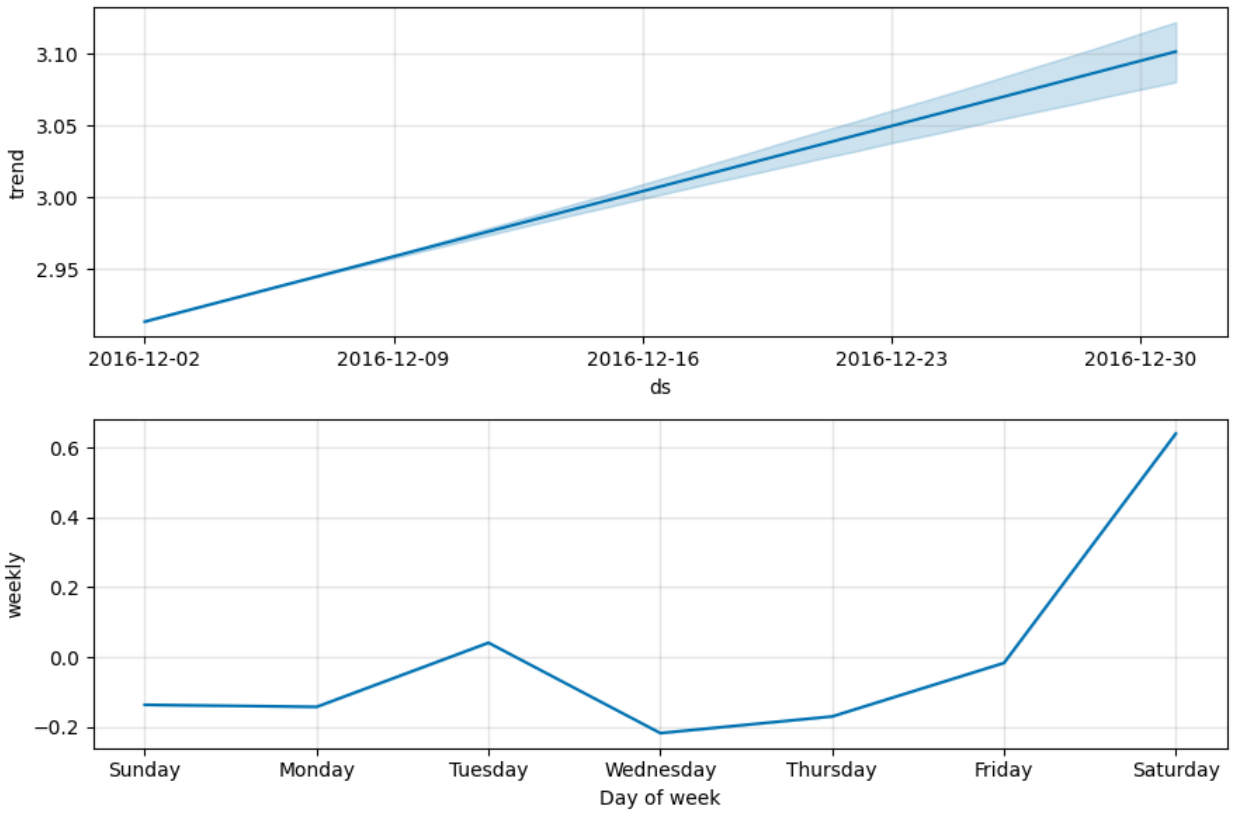
```
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"exog_series"}

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_u6bxa1/170rdvff.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/170rdvff.json',
'init=/tmp/tmpc_u6bxa1/sgkkvhm3.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeld3ffhjo_/prophet_model-
20251019104926.csv', 'method=optimize', 'algorithm=lbfgs',
'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"⚠ 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_u6bxa1/8t80s8h0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/_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_u6bxa1/8t80s8h0.json',
'init=/tmp/tmpc_u6bxa1/_oxkjl6y.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelu0ak8uuy/prophet_model-
20251019104934.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/c7ymc0xl.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/c7ymc0xl.json',
'init=/tmp/tmpc_u6bxa1/mcs3o25k.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/tot32p2v.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/tot32p2v.json',
'init=/tmp/tmpc_u6bxa1/aliynda0.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/u9pjj1c1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/u9pjj1c1.json',
```

```
'init=/tmp/tmpc_u6bxa1/ht4oo2au.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelusx6niz9/prophet_model-
20251019104957.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/lmw4n8vx.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/46c4ufe1.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_u6bxa1/lmw4n8vx.json',
'init=/tmp/tmpc_u6bxa1/46c4ufe1.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model5fky9vuh/prophet_model-
20251019105005.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/sw2h0340.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/_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_u6bxa1/sw2h0340.json',
'init=/tmp/tmpc_u6bxa1/_5eka9i8.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelwhvprg_7/prophet_model-
20251019105012.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/1la18ou7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/2jdbntrq.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_u6bxa1/1la18ou7.json',
'init=/tmp/tmpc_u6bxa1/2jdbntrq.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model4sj8ilf4/prophet_model-
20251019105021.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/ln71zsf6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/3e1qn76y.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_u6bxa1/ln71zsf6.json',
'init=/tmp/tmpc_u6bxa1/3e1qn76y.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelgk66bmk_/prophet_model-
20251019105027.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/2gzde3dg.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/__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_u6bxa1/2gzde3dg.json',
'init=/tmp/tmpc_u6bxa1/__jwnwee.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelgf_1uh1q/prophet_model-
20251019105036.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/4tsganr5.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/4tsganr5.json',
'init=/tmp/tmpc_u6bxa1/h9fe4qei.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/juhk_ht7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/juhk_ht7.json',
'init=/tmp/tmpc_u6bxa1/g2vgny_x.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/i1eo_syl.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/i1eo_syl.json',
'init=/tmp/tmpc_u6bxa1/vrpm0sl_.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/hi8ee8tb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/hi8ee8tb.json',
'init=/tmp/tmpc_u6bxa1/knax_zvn.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelsvn24d0y/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_u6bxa1/jh787431.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/_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_u6bxa1/jh787431.json',
'init=/tmp/tmpc_u6bxa1/_j0ms106.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/48ibczli.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/is_ikuh9.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_u6bxa1/48ibczli.json',
'init=/tmp/tmpc_u6bxa1/is_ikuh9.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelebpuku00/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_u6bxa1/qr4gcrpn.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/qr4gcrpn.json',
'init=/tmp/tmpc_u6bxa1/sm60n9pi.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model1erebchb/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_u6bxa1/n9yb9864.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/n9yb9864.json',
'init=/tmp/tmpc_u6bxa1/iwsonwsg.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/r4i1qq5j.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/r4i1qq5j.json',
'init=/tmp/tmpc_u6bxa1/s5kt1oh5.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/tsg0243t.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/tsg0243t.json',
'init=/tmp/tmpc_u6bxa1/t_x4t8kn.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/ifrrwora.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/ifrrwora.json',
'init=/tmp/tmpc_u6bxa1/v6mb8kfn.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/dmx0mpa6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/dmx0mpa6.json',
'init=/tmp/tmpc_u6bxa1/t85h6elw.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/nfr1roiv.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/nfr1roiv.json',
'init=/tmp/tmpc_u6bxa1/w_dy0tu2.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/y6vsd2r1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/y6vsd2r1.json',
'init=/tmp/tmpc_u6bxa1/idaq67hu.json', 'output',
```

```
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/c32ueilp.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/c32ueilp.json',
'init=/tmp/tmpc_u6bxa1/hxr099h2.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/z0v0wyp3.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/z0v0wyp3.json',
'init=/tmp/tmpc_u6bxa1/i2r_u3zr.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/vr4uhjzy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/vr4uhjzy.json',
'init=/tmp/tmpc_u6bxa1/l9dnsp6u.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelmrvbgmxd/prophet_model-
20251019105245.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/pu_3cdrz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/pu_3cdrz.json',
'init=/tmp/tmpc_u6bxa1/avhl5qqi.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelt8tvy0v9/prophet_model-
20251019105253.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/2n7p1d9b.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/2n7p1d9b.json',
'init=/tmp/tmpc_u6bxa1/ujfa__ya.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelyt9lw7vf/prophet_model-
20251019105301.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/lz4maps8.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/mjwgw2l1.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_u6bxa1/lz4maps8.json',
'init=/tmp/tmpc_u6bxa1/mjwgw2l1.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelsw26sc01/prophet_model-
20251019105308.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/1gtzgksp.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/1gtzgksp.json',
'init=/tmp/tmpc_u6bxa1/m0dipzac.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelzaktb2c3/prophet_model-
20251019105317.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/x940jgx1.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/x940jgx1.json',
'init=/tmp/tmpc_u6bxa1/n_eaaisu.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelmxwqktd3/prophet_model-
20251019105323.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/o_6yna37.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/o_6yna37.json',
'init=/tmp/tmpc_u6bxa1/ze4nwphx.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeleh1zn6li/prophet_model-
20251019105332.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/djwngrkf.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/djwngrkf.json',
'init=/tmp/tmpc_u6bxa1/icck71i0.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model9sauep_j/prophet_model-
20251019105338.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/hjcw1hyc.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/hjcw1hyc.json',
```

```
'init=/tmp/tmpc_u6bxa1/g3su8ix5.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelt9fqcg42/prophet_model-
20251019105347.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/j7d6tkcz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/8mg60cqp.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_u6bxa1/j7d6tkcz.json',
'init=/tmp/tmpc_u6bxa1/8mg60cqp.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelx66xkrgi/prophet_model-
20251019105353.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/32vhu0ol.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/3vy45v4a.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_u6bxa1/32vhu0ol.json',
'init=/tmp/tmpc_u6bxa1/3vy45v4a.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelp46lcfua/prophet_model-
20251019105402.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/2zkumub2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/34fhftcs.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_u6bxa1/2zkumub2.json',
'init=/tmp/tmpc_u6bxa1/34fhftcs.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modellngdbtcn/prophet_model-
20251019105409.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/6dh6rsp2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/9g_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_u6bxa1/6dh6rsp2.json',
'init=/tmp/tmpc_u6bxa1/9g_mzkui.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model9rp04s_l/prophet_model-
20251019105416.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/yx873em7.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/3mvs9erq.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_u6bxa1/yx873em7.json',
'init=/tmp/tmpc_u6bxa1/3mvs9erq.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelwyjdxmz8/prophet_model-
20251019105425.csv', 'method=optimize', 'algorithm=lbfgs',
'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"}
```


```
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\\u00e9e)_fr.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      \"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","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>
```

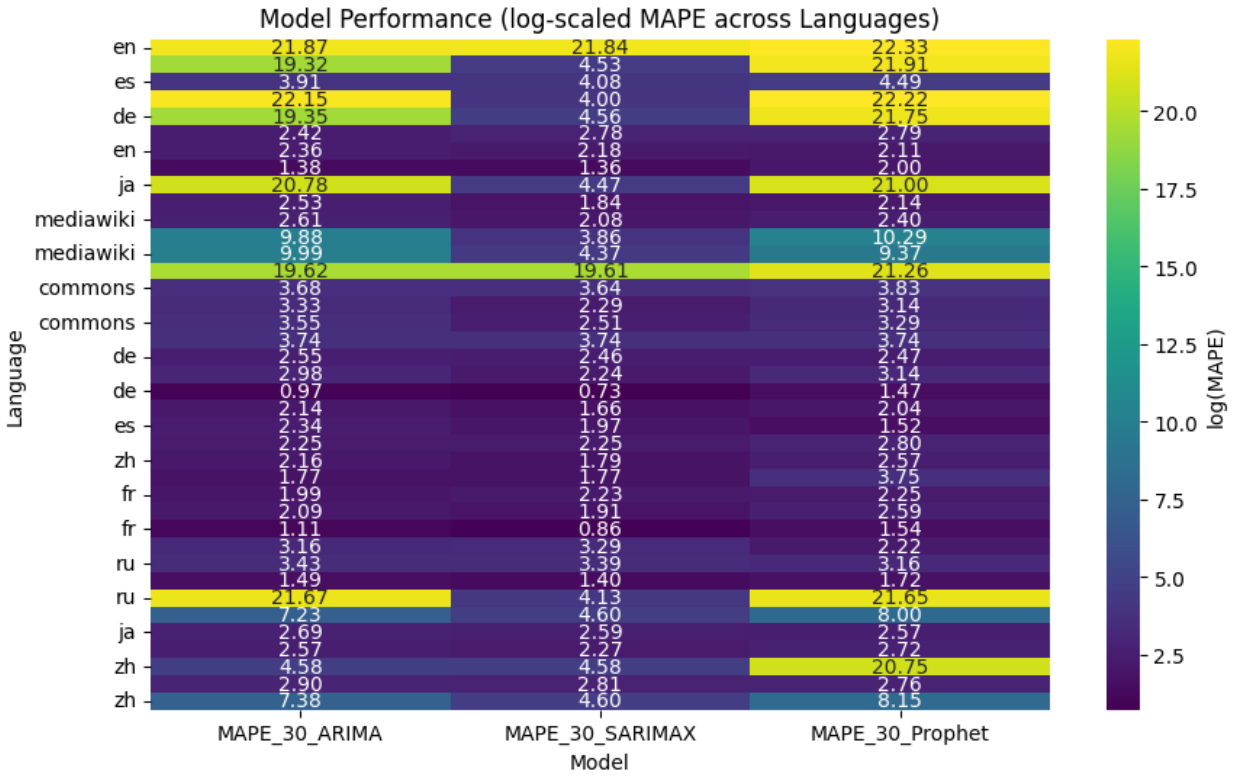
```
import pandas as pd
import re

# --- Detect language from 'Page' automatically ---
def extract_language(page):
    match = re.search(r'([a-z\-]+)\.wikipedia\.org', 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"
    )
)
```

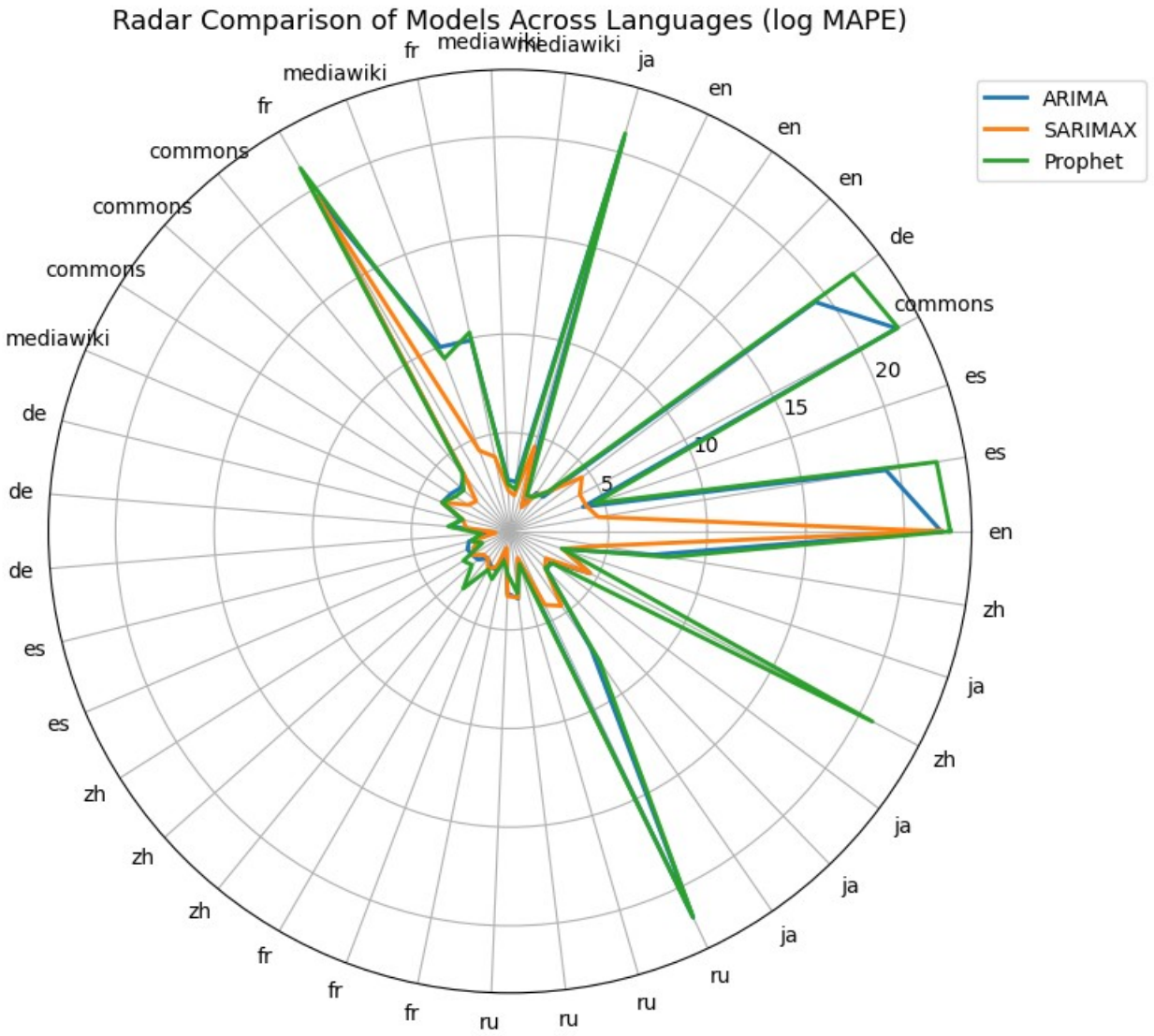
 Mean Model Performance per Language

```
<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

Languag e	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	🏆 Who Performed Best	Key Insights
					due to scaling or seasonal misfit.
mediawi ki	5479.1	32.96	2953.76	🏆 **SARIMAX**	SARIMAX crushed both ARIMA and Prophet — MAPE dropped from 5479 → 33 (99% reduction). Prophet performed moderately but still unstable, showing partial overfitting.
es (Spanish)	61M	39.99	821M	🏆 **SARIMAX**	SARIMAX drastically reduced error vs ARIMA (~99.9999% improvement). Prophet exploded, suggesting poor trend handling or non-stationary input.
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
(French)					performed poorly in absolute terms. SARIMAX small gain (+24%), Prophet weak. Possibly noisy data or non-periodic behavior.
ja (Japanese)	210M	44.28	264M	**SARIMAX**	SARIMAX is clearly superior; Prophet again diverged. ARIMA performed terribly (failed to capture seasonality).
ru (Russian)	646M	29.62	631M	**SARIMAX**	SARIMAX's 25% gain over ARIMA again shows consistent advantage when using exogenous regressors. Prophet still unstable.
en (English)	786M	765M	1.24B	**SARIMAX (barely)**	SARIMAX marginally improved (~6% better). Prophet worsened (49% higher error). Likely due to

Languag e	ARIMA MAPE	SARIMAX MAPE	Prophet MAPE	Who Performed Best	Key Insights
					irregular spikes or non-seasonal data.
common s	1.04B	27.74	1.12B	**SARIMAX**	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":"{\n  \"name\": \"ts_subset\",\n  \"rows\": 21450,\n
\"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 39,\n        \"samples\": [\n          \"\\
u7279\\u5225:\\u691c\\u7d22_ja.wikipedia.org_all-access_all-agents\",\
n          \"MediaWiki:EnhancedCollapsibleElements.js_ja.wikipedia.org
_all-access_spider\",\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        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Title\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
25,\n        \"samples\": [\n          \"Wikipedia:\\u9996\\u9875\",\n
\"Especial:Buscar\",\n          \"\\u6d2a\\u6668\\u7a4e\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Language\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 9,\n        \"samples\": [\n          \"ja\",\n
\"fr\",\n          \"www\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"AccessType\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"all-access\",\n          \"desktop\",\n
\"mobile-web\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"AccessOrigin\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"all-agents\",\n          \"spider\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2015-07-01 00:00:00\",\n
\"max\": \"2016-12-31 00:00:00\",\n        \"num_unique_values\":
550,\n        \"samples\": [\n          \"2016-01-12 00:00:00\",\n
\"2015-09-18 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"views\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4781649.129266517,\n        \"min\":
0.0,\n        \"max\": 67264258.0,\n        \"num_unique_values\":
14542,\n        \"samples\": [\n          24.0,\n          4795450.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\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_u6bxa1/qz4439b0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/qz4439b0.json',
'init=/tmp/tmpc_u6bxa1/eod51zxl.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model5omjoabx/prophet_model-
20251019115006.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/000s80uj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/000s80uj.json',
'init=/tmp/tmpc_u6bxa1/n16ts60g.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelbx4v90sx/prophet_model-
20251019115007.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/5fa8dh84.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/5fa8dh84.json',
```

```
'init=/tmp/tmpc_u6bxa1/g9oa7vup.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelo4cg16nu/prophet_model-
20251019115007.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/593mgt8j.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/593mgt8j.json',
'init=/tmp/tmpc_u6bxa1/m9yaxnm3.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelslqxuxno/prophet_model-
20251019115008.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/b2zgwhjd.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/b2zgwhjd.json',
'init=/tmp/tmpc_u6bxa1/genbrvye.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model_rocjiui/prophet_model-
20251019115008.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/hp32utty.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/hp32utty.json',
'init=/tmp/tmpc_u6bxa1/aryndgmd.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modele4r8ou4k/prophet_model-
20251019115009.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/3tjdybhz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/jsso92jr.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_u6bxa1/3tjdybhz.json',
'init=/tmp/tmpc_u6bxa1/jsso92jr.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model_zkggl0o/prophet_model-
20251019115009.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/6o5vxo4t.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/6o5vxo4t.json',
'init=/tmp/tmpc_u6bxa1/n1h8t3bh.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelvbnsvmm1/prophet_model-
20251019115010.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/rrqar26r.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/rrqar26r.json',
'init=/tmp/tmpc_u6bxa1/het9qkjx.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeles3gslhf/prophet_model-
20251019115012.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/dztajwuo.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/dztajwuo.json',
'init=/tmp/tmpc_u6bxa1/ow8hkmd4.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelxa6mgtb_/prophet_model-
20251019115013.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/jsgzo_0j.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/_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_u6bxa1/jsgzo_0j.json',
'init=/tmp/tmpc_u6bxa1/_r3_g790.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelnphvkvk7/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
 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_u6bxa1/rlrf5n7c.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/rlrf5n7c.json',
'init=/tmp/tmpc_u6bxa1/ztz6mfzx.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/mo49_2m5.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/mo49_2m5.json',
'init=/tmp/tmpc_u6bxa1/cyaqtk0z.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/hohgr_60.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/hohgr_60.json',
'init=/tmp/tmpc_u6bxa1/gbu2aw13.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model0r367m_h/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
11:50:17 - 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_u6bxa1/pguous92.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/293cjpbw.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_u6bxa1/pguous92.json',
'init=/tmp/tmpc_u6bxa1/293cjpbw.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelja__9wy3/prophet_model-
20251019115017.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/r_xdyf8l.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/t2d2zwg2.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_u6bxa1/r_xdyf8l.json',
'init=/tmp/tmpc_u6bxa1/t2d2zwg2.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelxat26btl/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
 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_u6bxa1/khpj4l23.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/khpj4l23.json',
'init=/tmp/tmpc_u6bxa1/y8nqqwk0.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model4s_18oxz/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_u6bxa1/3r5uu6cw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/3r5uu6cw.json',
'init=/tmp/tmpc_u6bxa1/gyutm0zx.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/006kd4wj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/006kd4wj.json',
'init=/tmp/tmpc_u6bxa1/fwsplylg.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/ndfzpvc6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/3746_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_u6bxa1/ndfzpvc6.json',
'init=/tmp/tmpc_u6bxa1/3746_e5q.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/964qts9_.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/964qts9_.json',
'init=/tmp/tmpc_u6bxa1/t75xulmf.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/7tyry3pb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/7tyry3pb.json',
'init=/tmp/tmpc_u6bxa1/zc43e_wj.json', 'output',
```

```
'file=/tmp/tmpc_u6bxa1/prophet_modelch7uk887/prophet_model-
20251019115022.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/tnk571ch.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/7i_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_u6bxa1/tnk571ch.json',
'init=/tmp/tmpc_u6bxa1/7i_hkpl2.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelphej0uf7/prophet_model-
20251019115022.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/8smwtqnx.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/3f4jcju_.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_u6bxa1/8smwtqnx.json',
'init=/tmp/tmpc_u6bxa1/3f4jcju_.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelpweqzh42/prophet_model-
20251019115023.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/kkpiq702.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/kkpiq702.json',
'init=/tmp/tmpc_u6bxa1/zowtks5_.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelzu63diwf/prophet_model-
20251019115023.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/8wyg44yq.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/ggturrqw.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_u6bxa1/8wyg44yq.json',
'init=/tmp/tmpc_u6bxa1/ggturrqw.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeloah_3txe/prophet_model-
20251019115024.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/_4qohlcw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/_4qohlcw.json',
'init=/tmp/tmpc_u6bxa1/bboki44v.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model6lig63c0/prophet_model-
20251019115025.csv', 'method=optimize', 'algorithm=lbfgs',
'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/tmpc_u6bxa1/2rebfoau.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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/tmpc_u6bxa1/2rebfoau.json',
'init=/tmp/tmpc_u6bxa1/onpcgvk0.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelqzf9n6_e/prophet_model-
20251019115026.csv', 'method=optimize', 'algorithm=lbfgs',
'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/tmpc_u6bxa1/1mvs6dlw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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/tmpc_u6bxa1/1mvs6dlw.json',
'init=/tmp/tmpc_u6bxa1/hro370hl.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelplnhuunh/prophet_model-
20251019115028.csv', 'method=optimize', 'algorithm=lbfgs',
'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/tmpc_u6bxa1/yloxaius.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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/tmpc_u6bxa1/yloxaius.json',
'init=/tmp/tmpc_u6bxa1/vayoicdl.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model9k_xefru/prophet_model-
20251019115029.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/bwkwsf0i.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/0pg_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_u6bxa1/bwkwsf0i.json',
'init=/tmp/tmpc_u6bxa1/0pg_0luj.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeloigmcs1i/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_u6bxa1/3pj0kb8l.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/9kn7m2xx.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_u6bxa1/3pj0kb8l.json',
'init=/tmp/tmpc_u6bxa1/9kn7m2xx.json', 'output',
'file=/tmp/tmpc_u6bxa1/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_u6bxa1/1ql60spy.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/1ql60spy.json',
```

```
'init=/tmp/tmpc_u6bxa1/uf6x97mr.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modeltcyn5t47/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
 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_u6bxa1/ydnym760.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/ydnym760.json',
'init=/tmp/tmpc_u6bxa1/tr03nr2o.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modela7l2364v/prophet_model-
20251019115032.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/u2welpq2.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/6reu5yj7.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_u6bxa1/u2welpq2.json',
'init=/tmp/tmpc_u6bxa1/6reu5yj7.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_model1qem8cvu/prophet_model-
20251019115033.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/75mhgkud.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/75mhgkud.json',
'init=/tmp/tmpc_u6bxa1/dpy3f6fp.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelu3y6o4ln/prophet_model-
20251019115033.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/6o43k0be.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/6o43k0be.json',
'init=/tmp/tmpc_u6bxa1/uwhm3721.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modela4dulb2w/prophet_model-
20251019115034.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/smexjskb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/smexjskb.json',
'init=/tmp/tmpc_u6bxa1/cye6nz6w.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelmifq_6k3/prophet_model-
20251019115035.csv', 'method=optimize', 'algorithm=lbfgs',
'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_u6bxa1/mvd5gzvz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpc_u6bxa1/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_u6bxa1/mvd5gzvz.json',
'init=/tmp/tmpc_u6bxa1/bknnvlyg.json', 'output',
'file=/tmp/tmpc_u6bxa1/prophet_modelt520wyvw/prophet_model-
20251019115035.csv', 'method=optimize', 'algorithm=lbfgs',
'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          \"Wikip\\
u00e9dia:Accueil_principal_fr.wikipedia.org_desktop_all-agents\",\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
\"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        \"std\":
1096453711528511.5,\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        \"std\":
1096453711528511.5,\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        \"std\":
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).

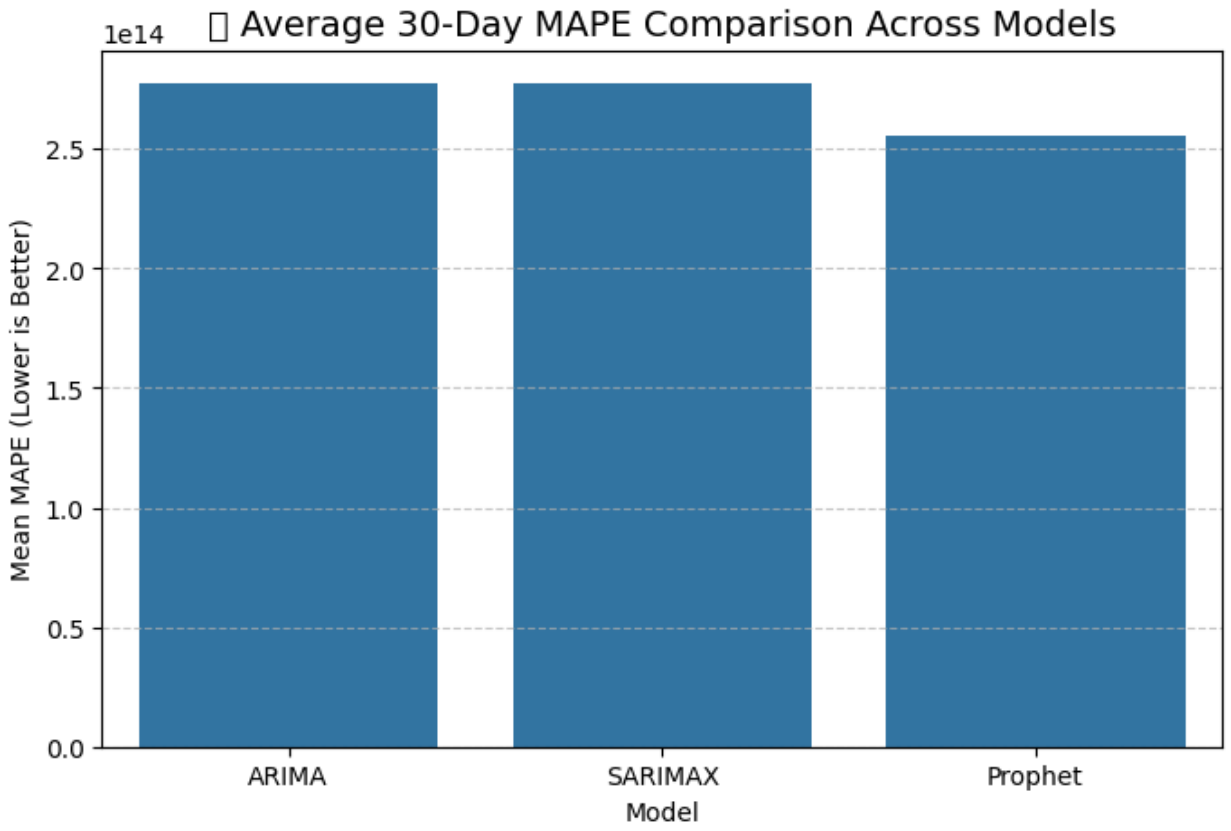
```
# --- Step 10: Model Evaluation & Insights ---

# 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)
```

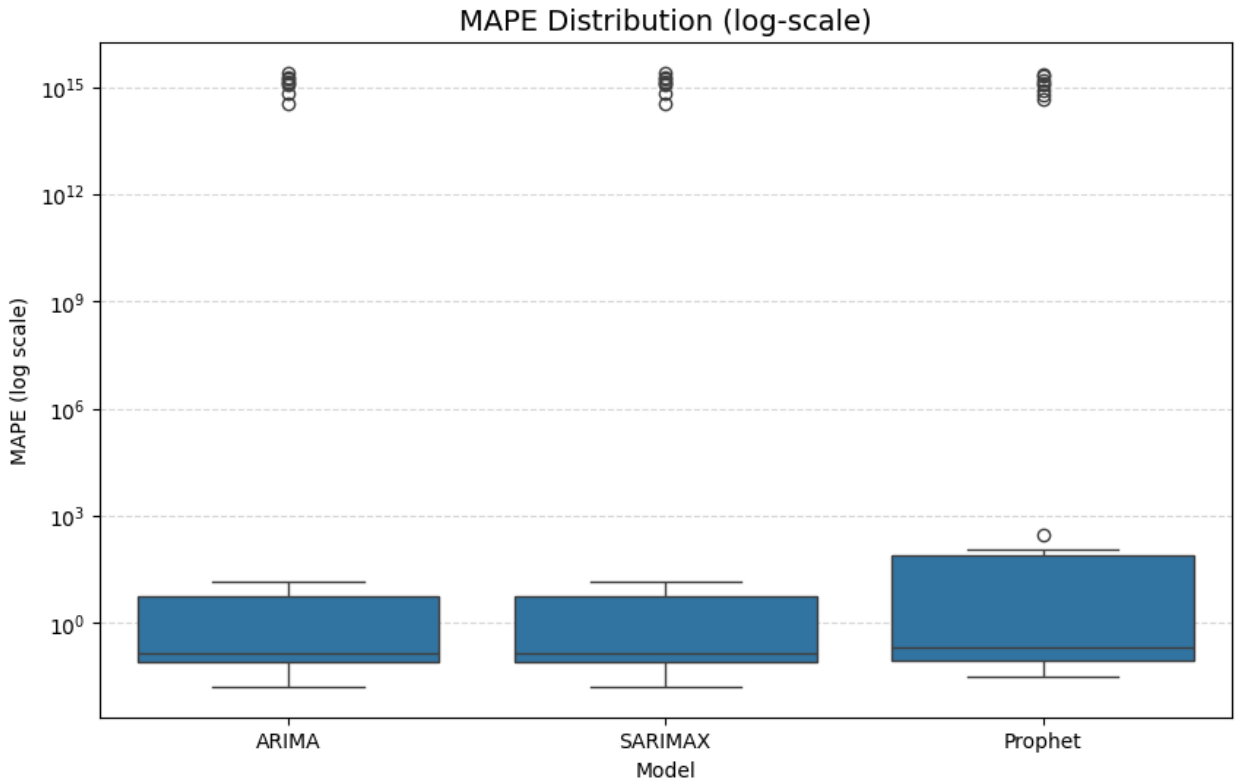
```
plt.title("📊 Average 30-Day MAPE Comparison Across Models",
fontsize=14)
plt.ylabel("Mean MAPE (Lower is Better)")
plt.xlabel("Model")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# --- 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":"{\n  \"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
```

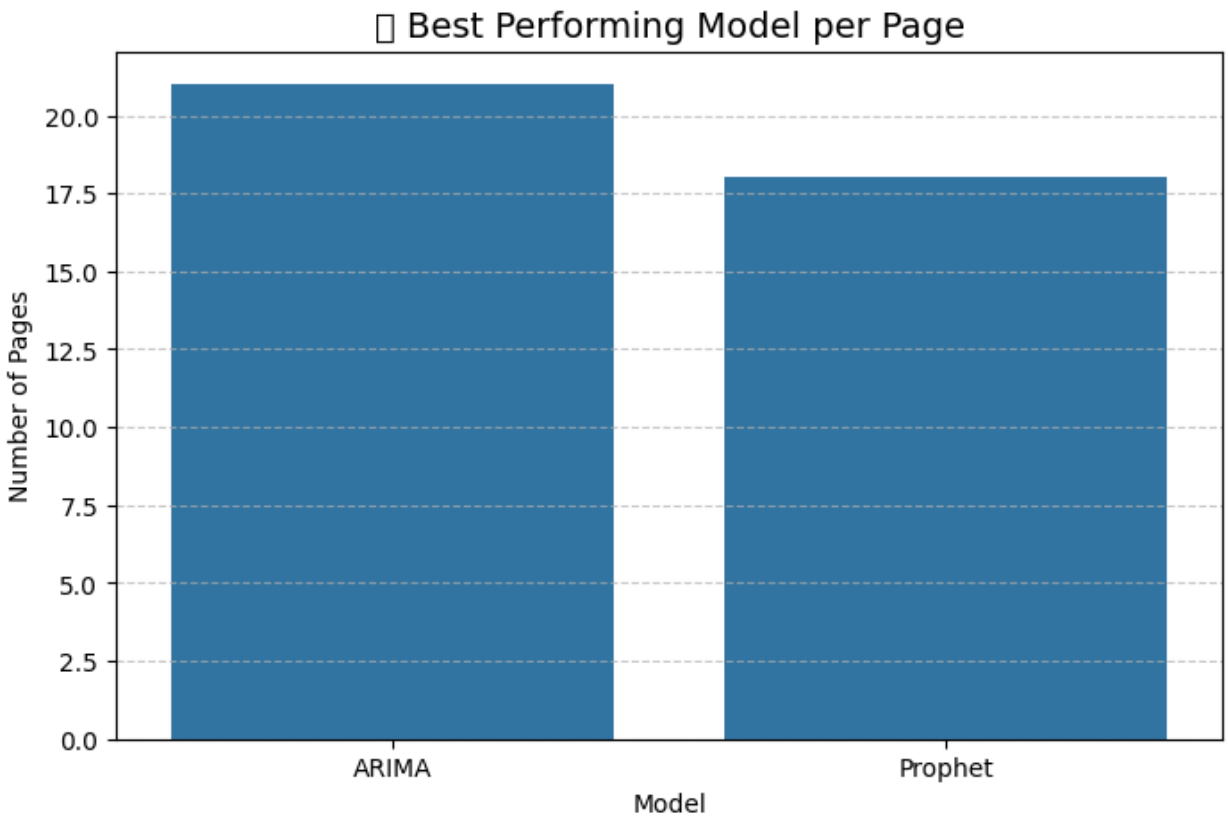
```
\"num_unique_values\": 5,\n        \"samples\": [\n          \"Wikip\\
u00e9dia:Accueil_principal_fr.wikipedia.org_desktop_all-agents\",\n
\"\\u0417\\u0430\\u0433\\u043b\\u0430\\u0432\\u043d\\u0430\\u044f_\\
u0441\\u0442\\u0440\\u0430\\u043d\\u0438\\u0446\\
u0430_ru.wikipedia.org_mobile-web_all-agents\",\n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


Average 30-Day MAPE Comparison Across Models
1e14
Mean MAPE (Lower is Better)
2.5
2.0
1.5
1.0
0.5
0.0
ARIMA
SARIMAX
Prophet
Model


MAPE Distribution (log-scale)
MAPE (log scale)
10^{15}
10^{12}
10^{9}
10^{6}
10^{3}
10^{0}
ARIMA
SARIMAX
Prophet
Model

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

1️⃣ 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.

2️⃣ 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.

3️⃣ 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).

4️⃣ 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.

5️⃣ 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).

6️⃣ 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.

7️⃣ 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