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"}
```

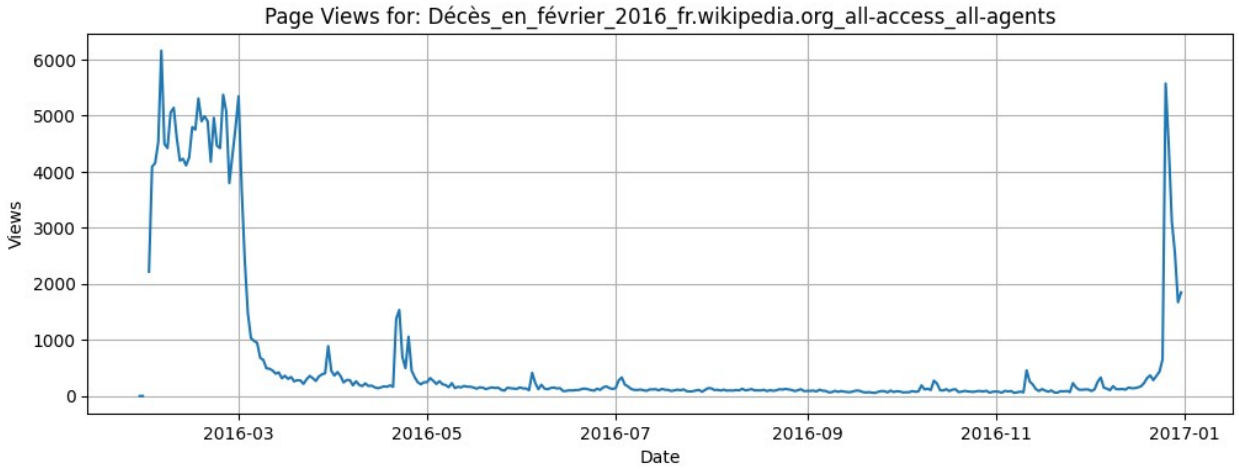
Sample pages with internal NaNs:

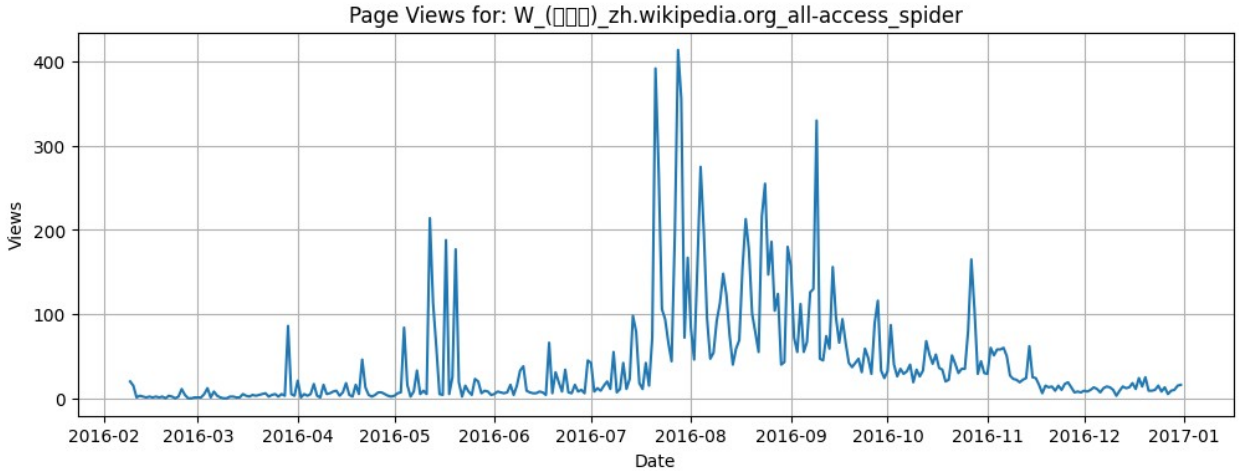

```
{"summary":"{\n  \"name\": \"     plot_page_views(row, date_cols)\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"\\
u540d\\u53e4\\u5c4b\\u897f\\u672c\\u7dda\\u6599\\u91d1\\
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"}
```

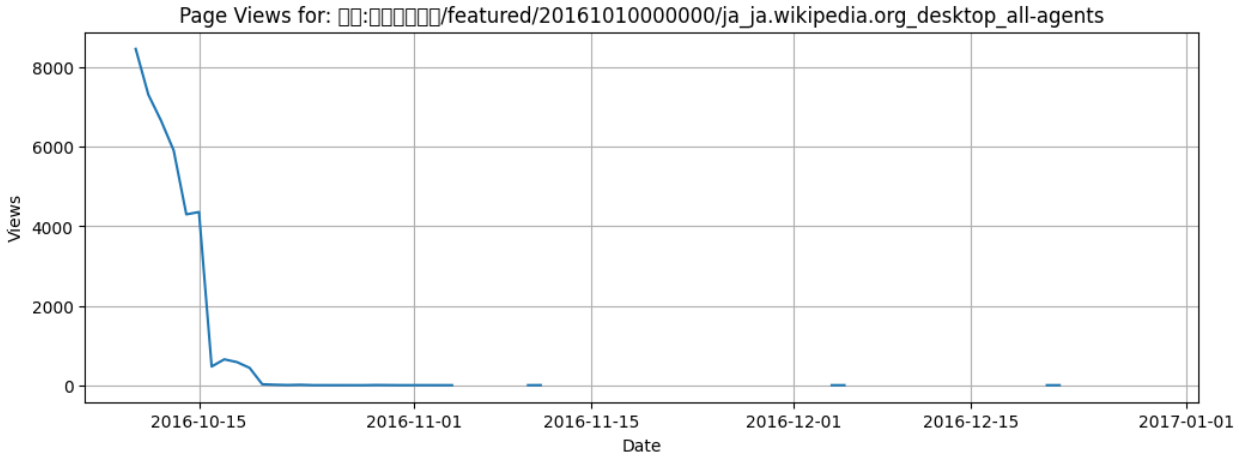

Sample pages with trailing NaNs:

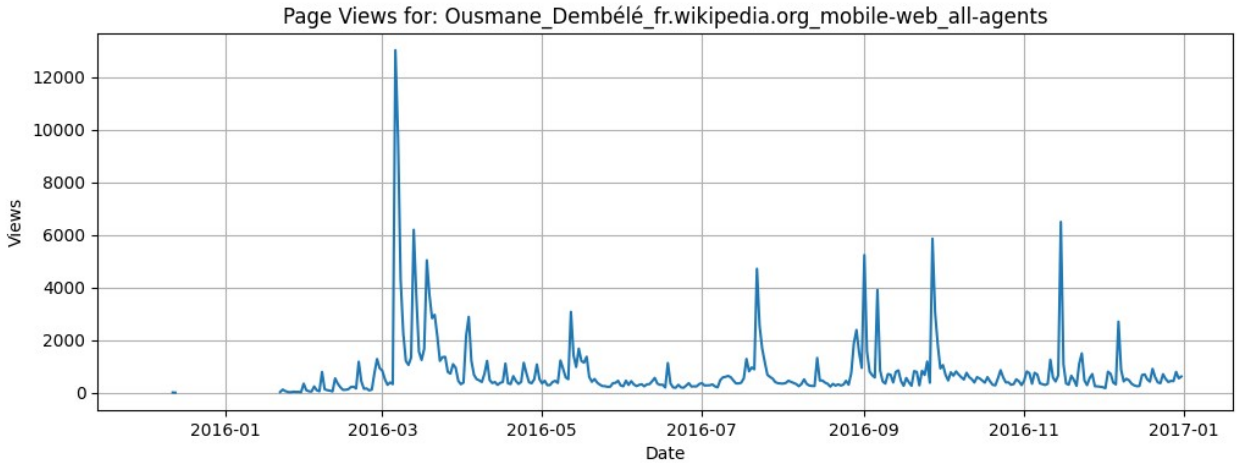

```
{"summary":"{\n  \"name\": \"     plot_page_views(row, date_cols)\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Page\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"File:Zeitschrift_des_Vereins_fuer_Volkskunde_25_b_A_006.jpg_commons.
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"}
```

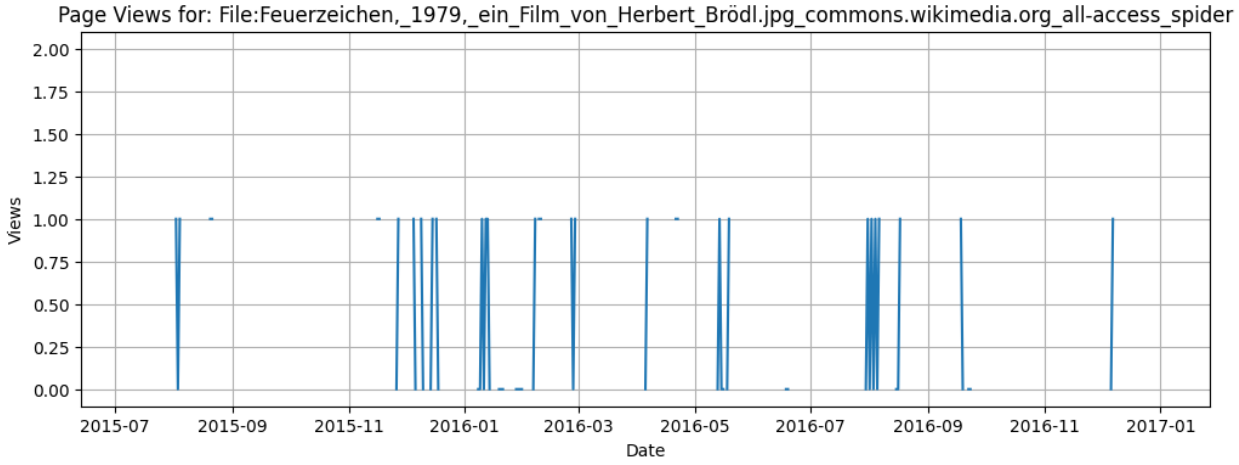

Visualizing sample pages with leading NaNs:


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

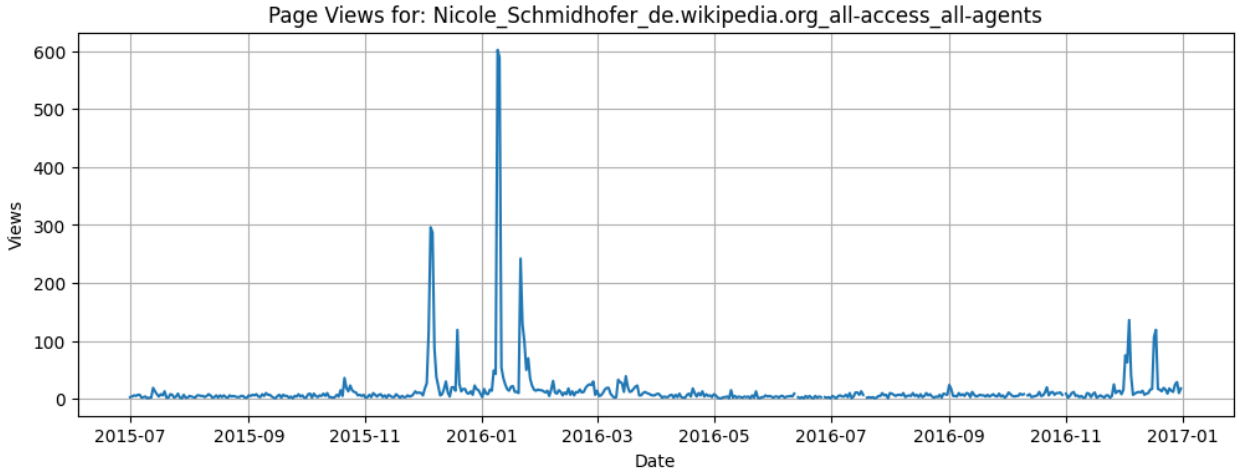

Page Views for: W_(□□□)_zh.wikipedia.org_all-access_spider
Views
Date
0
100
200
300
400
2016-02
2016-03
2016-04
2016-05
2016-06
2016-07
2016-08
2016-09
2016-10
2016-11
2016-12
2017-01

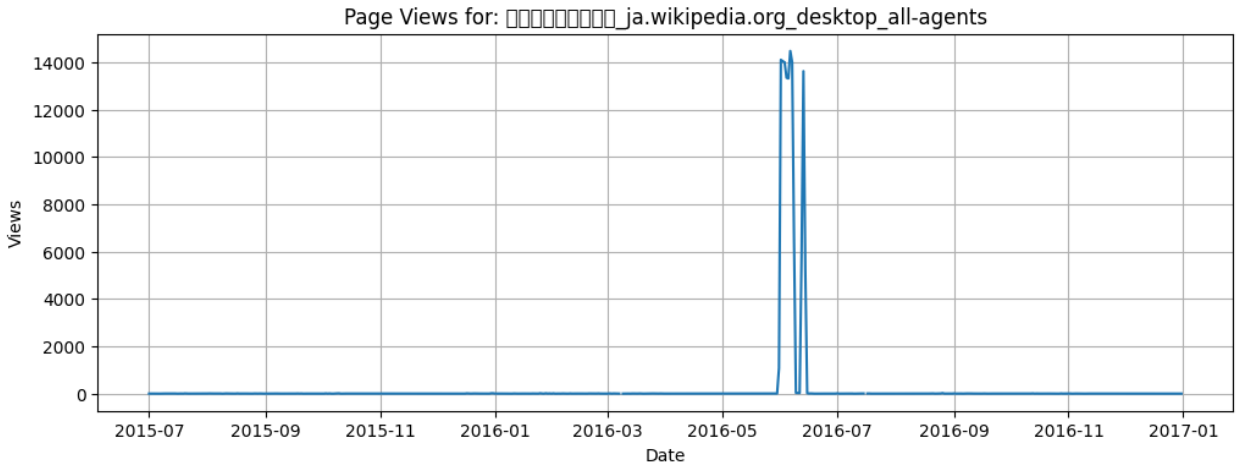

Page Views for: □□:□□□□□□/featured/20161010000000/ja_ja.wikipedia.org_desktop_all-agents
Views
Date
0
2000
4000
6000
8000
2016-10-15
2016-11-01
2016-11-15
2016-12-01
2016-12-15
2017-01-01

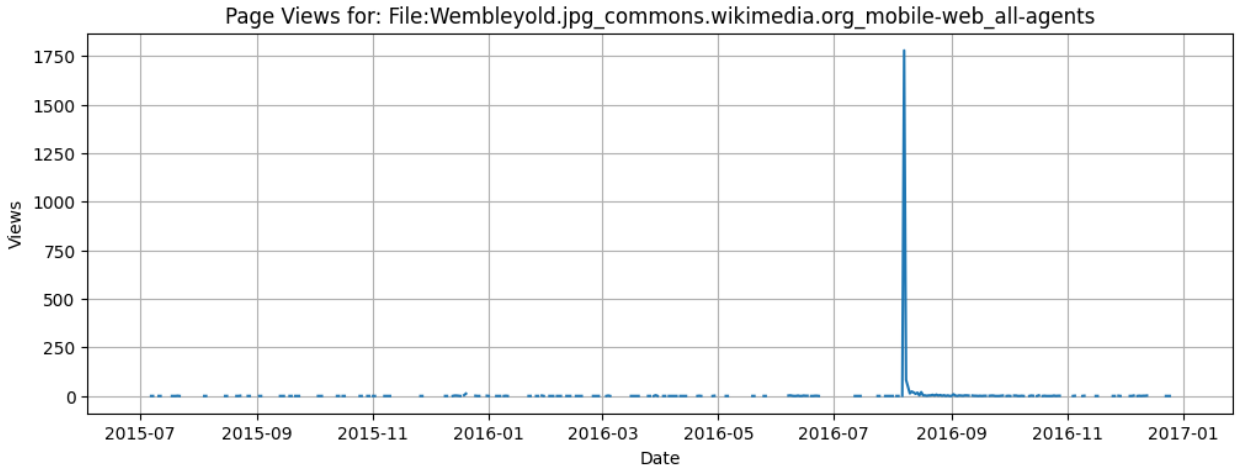


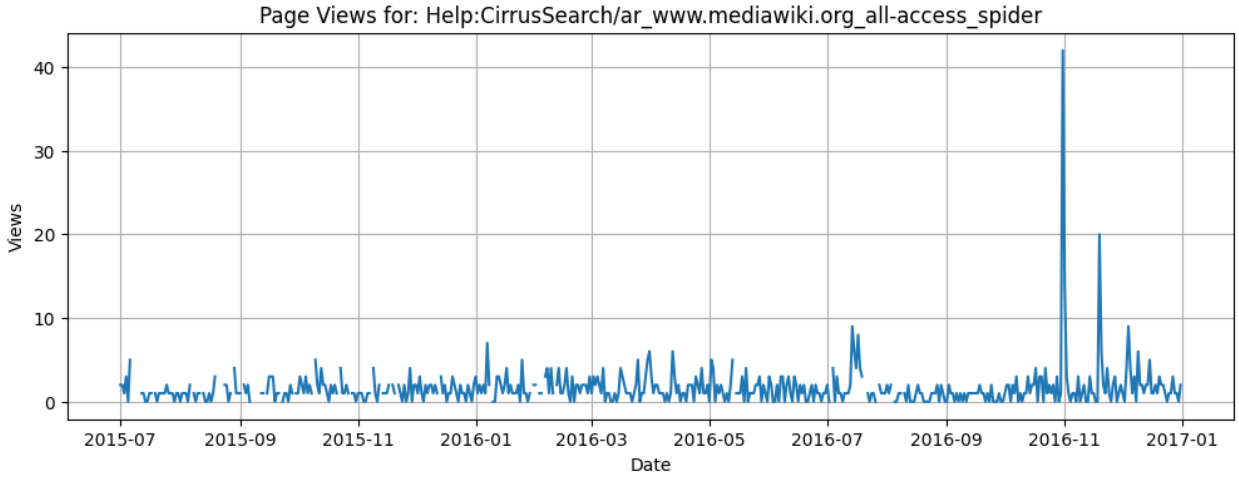


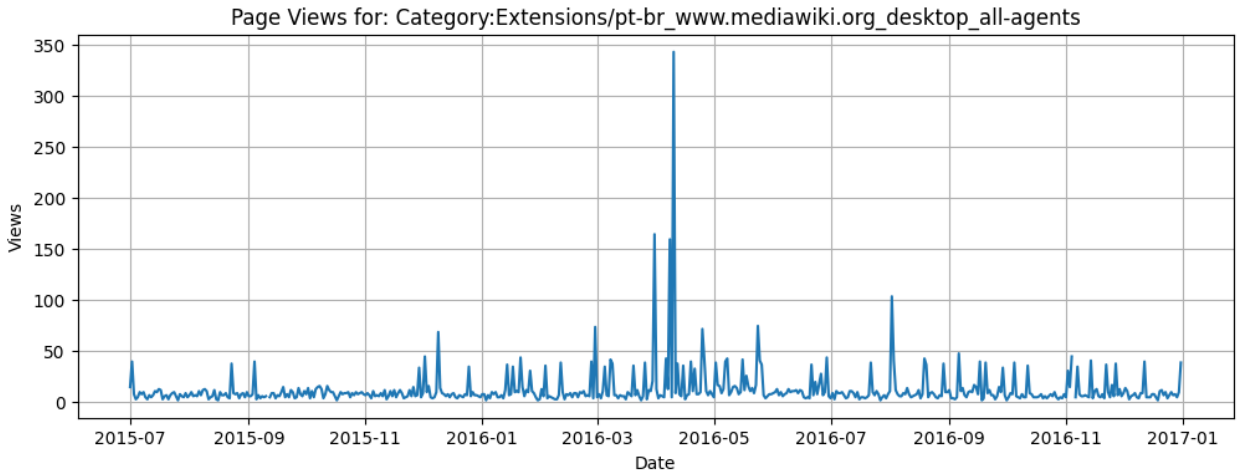
Visualizing sample pages with internal NaNs:



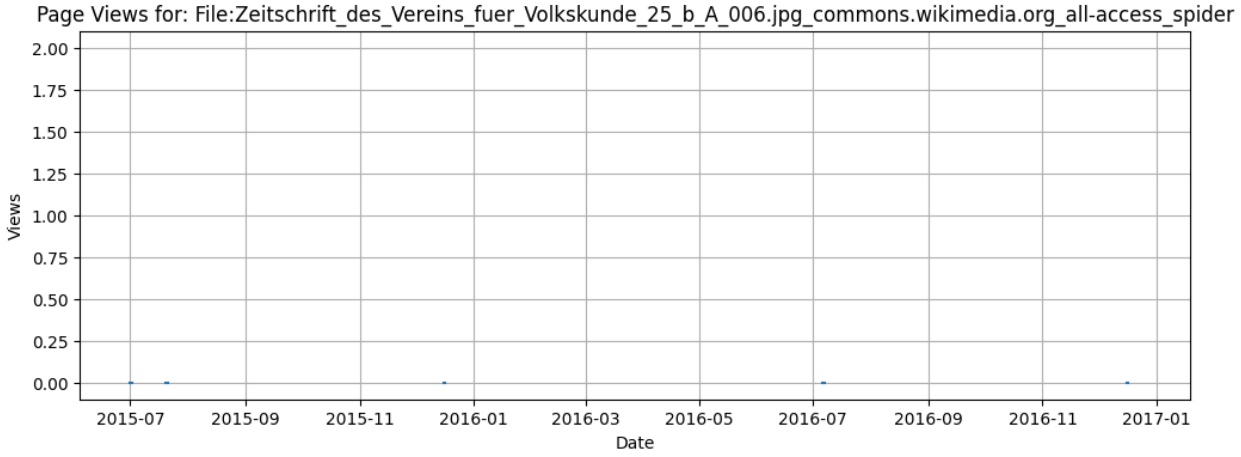

Page Views for: □□□□□□□□□□_ja.wikipedia.org_desktop_all-agents
Views
Date
0
2000
4000
6000
8000
10000
12000
14000
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

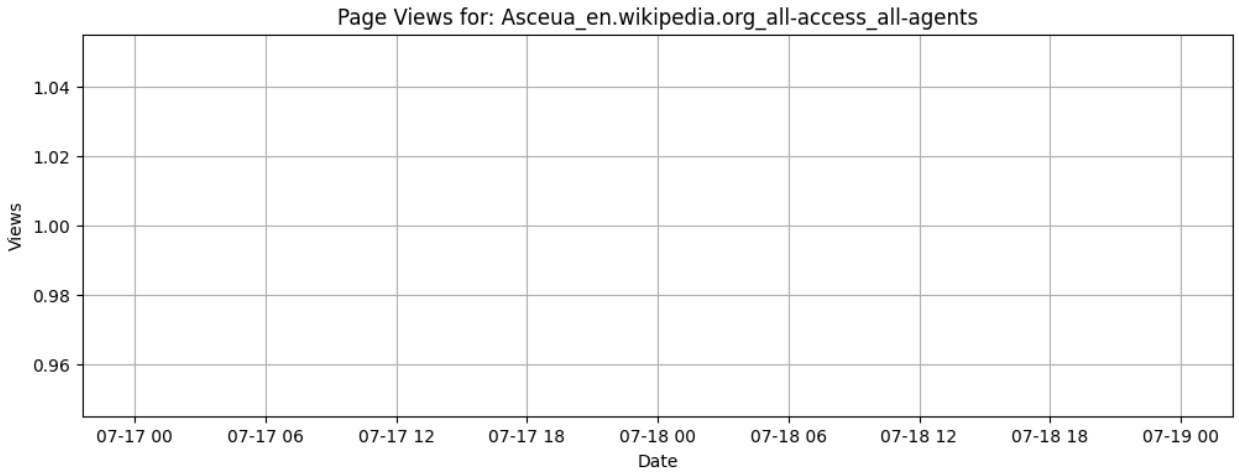

Page Views for: File:Wembleyold.jpg_commons.wikimedia.org_mobile-web_all-agents
Views
Date
0
250
500
750
1000
1250
1500
1750
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

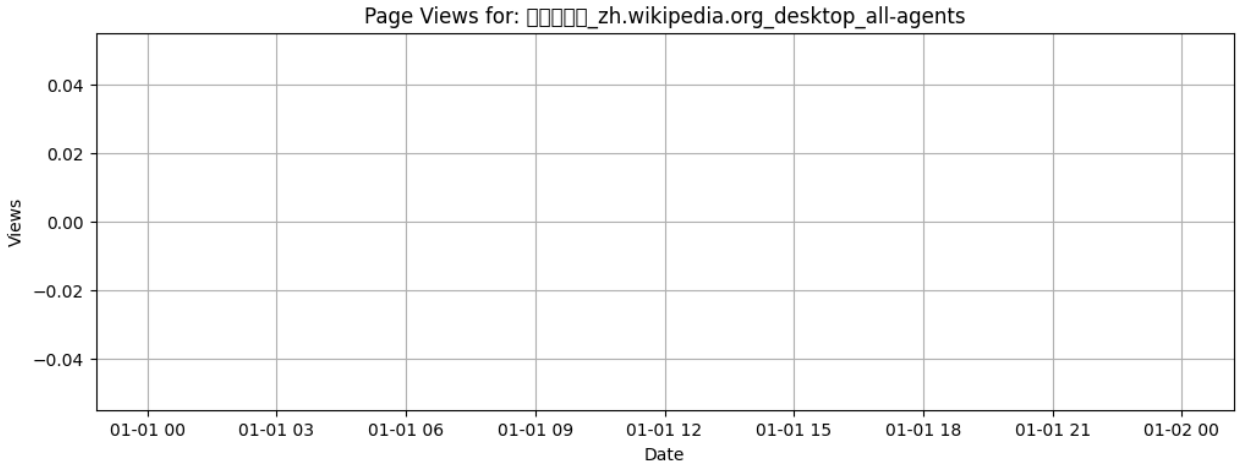

Page Views for: Help:CirrusSearch/ar_www.mediawiki.org_all-access_spider
Views
Date
0
10
20
30
40
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01

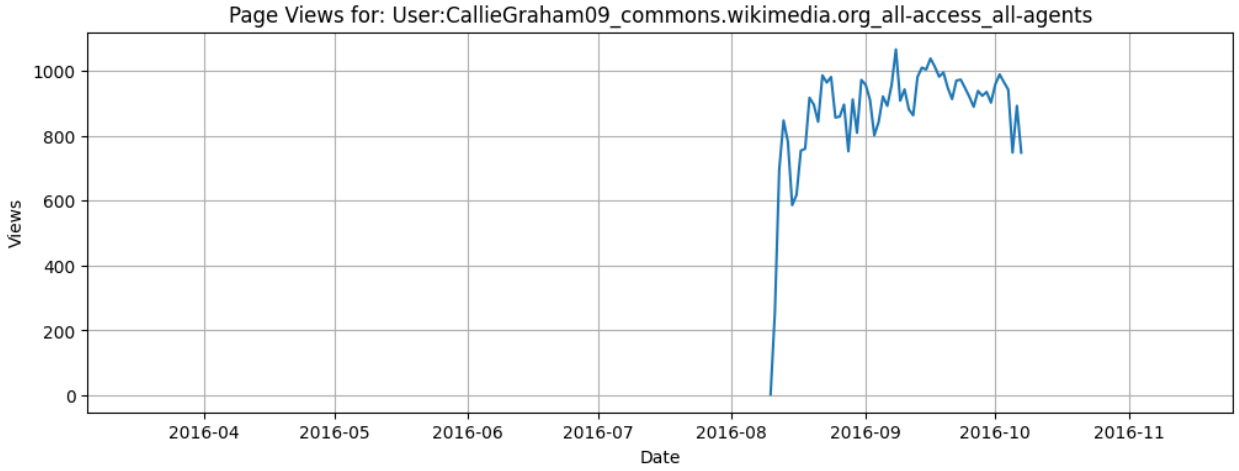



```
Visualizing sample pages with trailing NaNs:
```










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

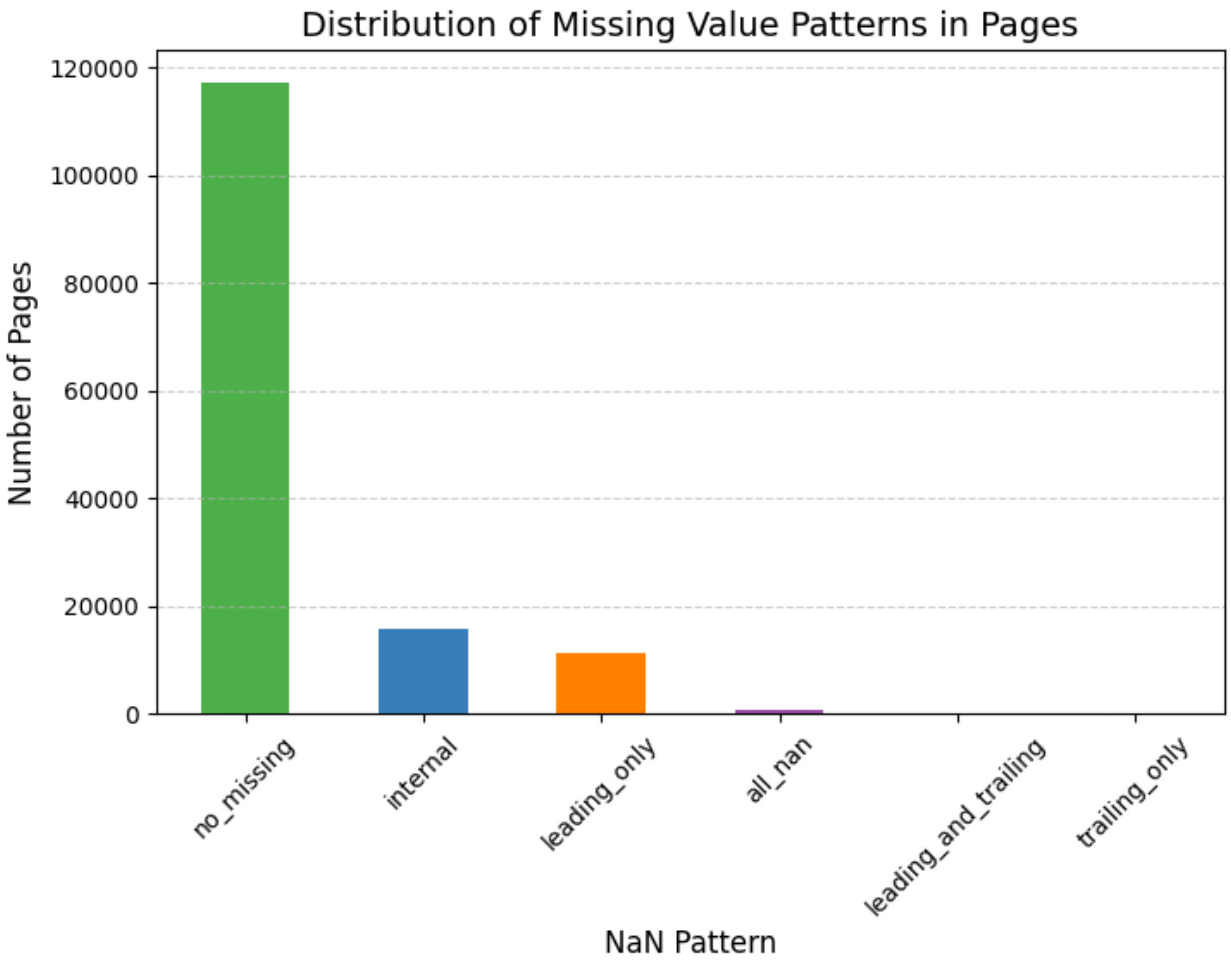
```
(652,)
```

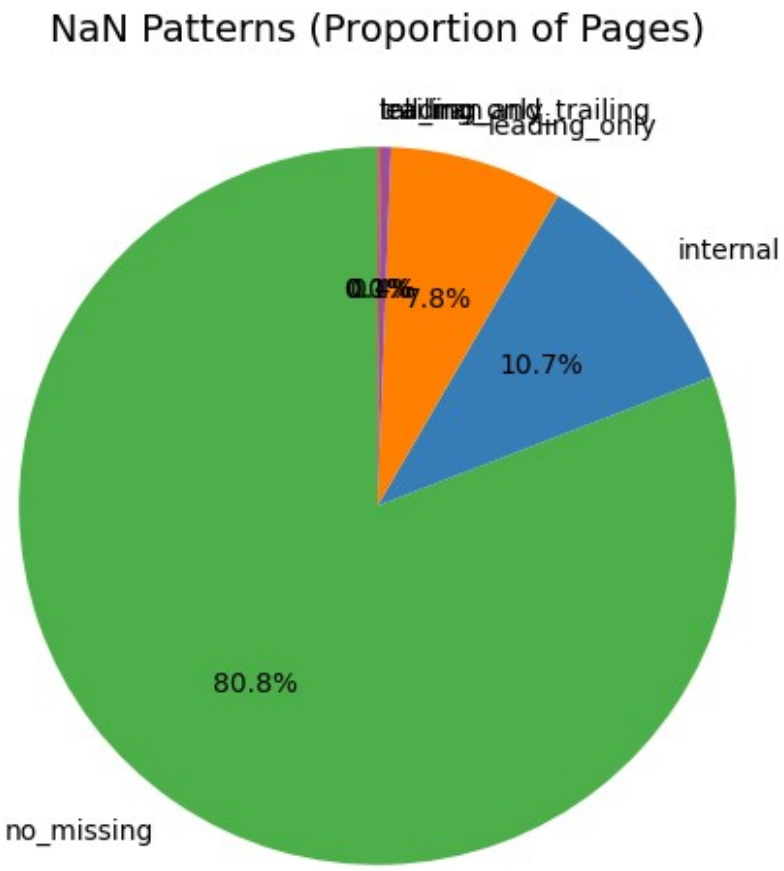
```python
# Helper to classify NaN pattern for each row
def classify_nan_pattern(row, date_cols):
    values = row[date_cols].values
    isnan = pd.isna(values)

    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

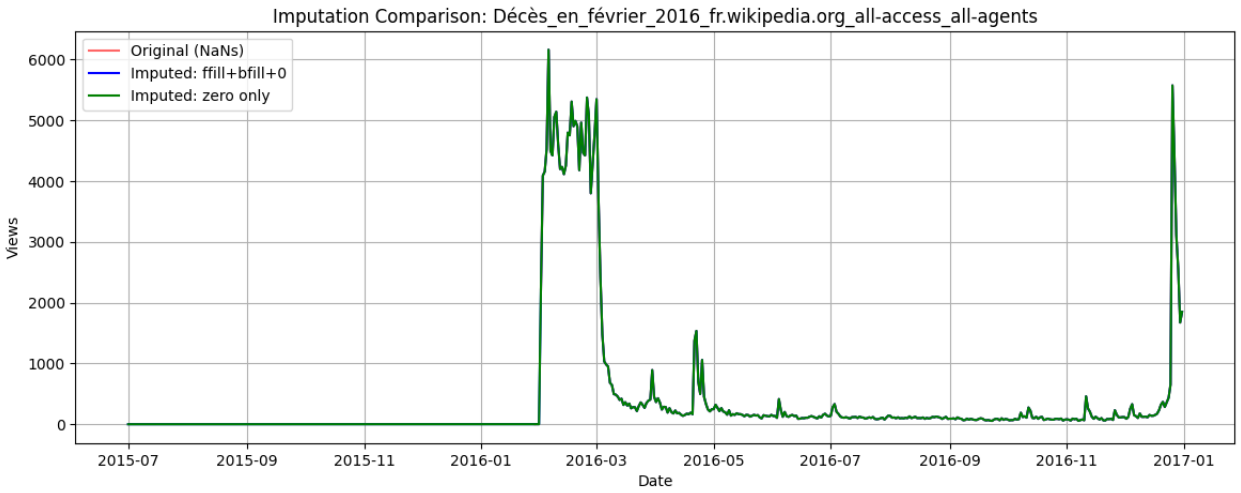
- **Leading NaNs** → fill with **0**.
- **Trailing NaNs** → fill with **0**.
- **Internal NaNs** → use **interpolation** (linear or forward/backward fill).
- **All-NaN pages** (e.g., page 652) → **drop** (no useful data).

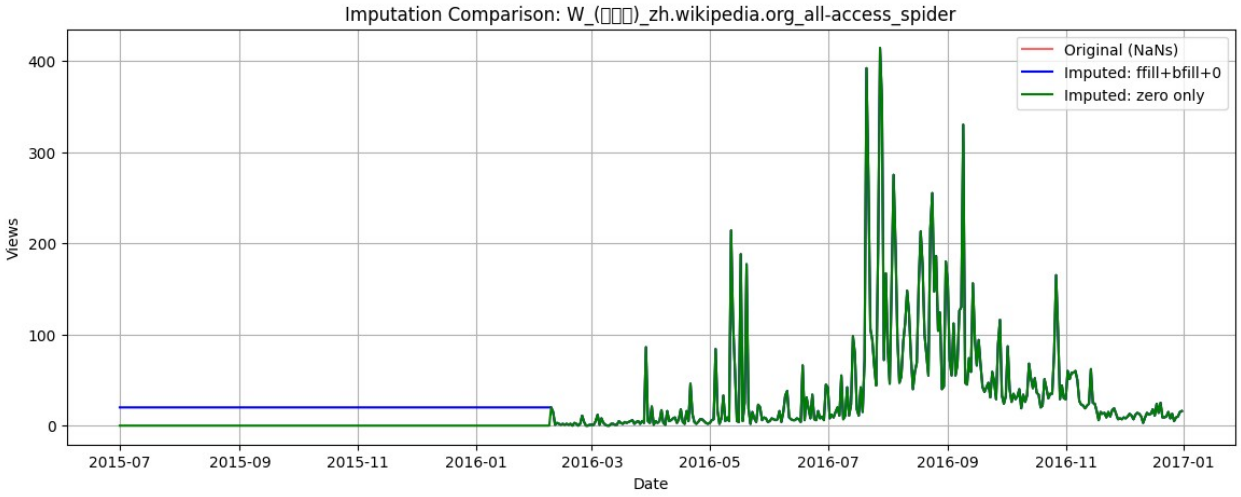
```
sample_trailing_nan_pages['Page'].to_list()
```

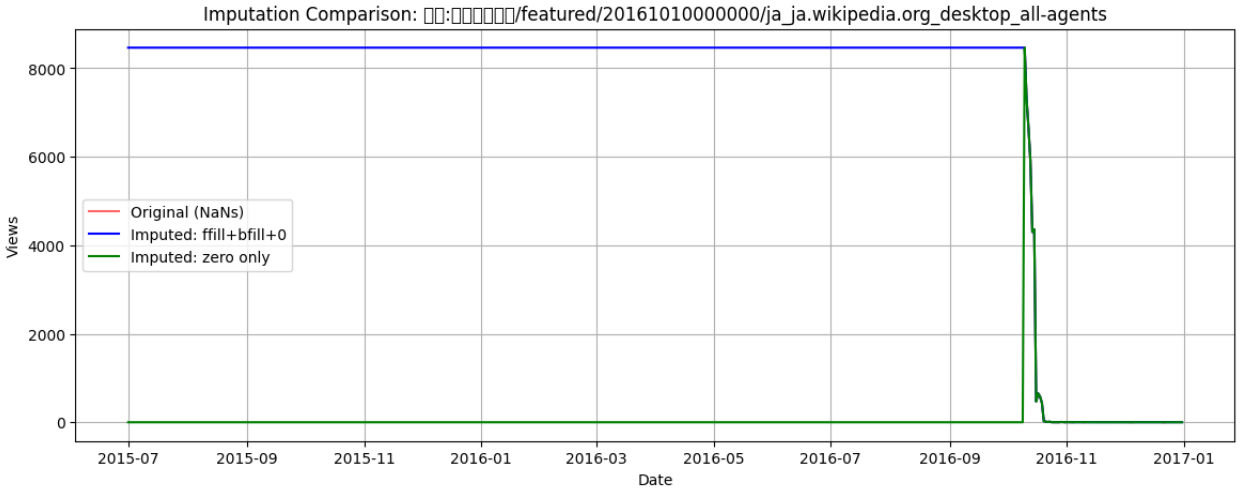
```
['Категория:Гидра_(созвездие)_ru.wikipedia.org_all-access_all-agents',
 'File:Zeitschrift_des_Vereins_fuer_Volkskunde_25_b_A_006.jpg_commons.wikimedia.org_all-access_spider',
 'Asceua_en.wikipedia.org_all-access_all-agents',
 '是松豐三郎_zh.wikipedia.org_desktop_all-agents',
 'User:CallieGraham09_commons.wikimedia.org_all-access_all-agents']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

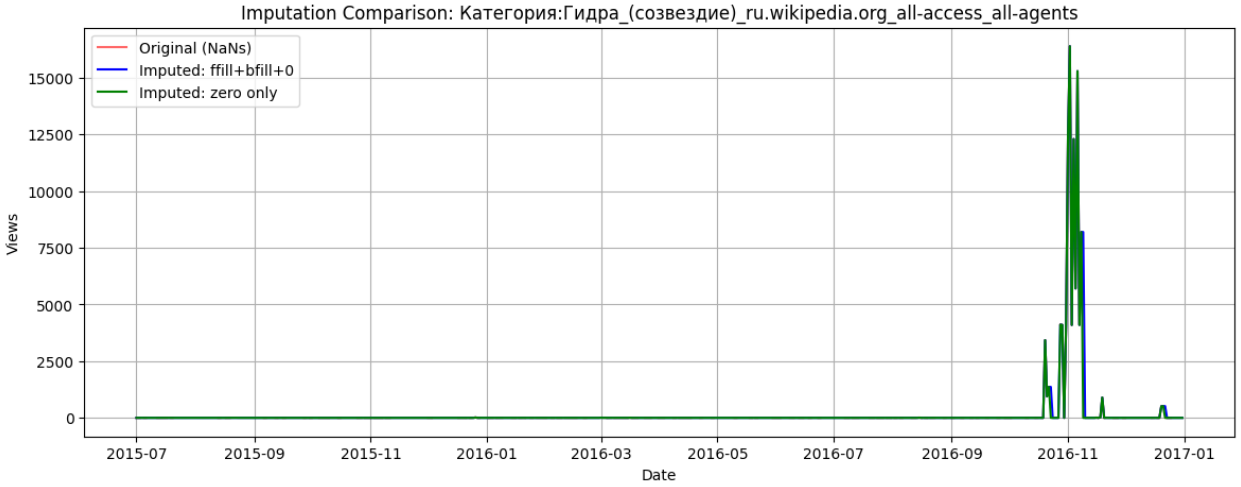
```
# --- Create both imputation strategies ---
def impute_ffill_bfill(df, date_cols):
    filled = df.copy()
    filled = filled[filled[date_cols].notna().any(axis=1)]
    # Forward + backward fill, then 0
    filled[date_cols] = (
        filled[date_cols]
        .fillna(method='ffill', axis=1)
 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

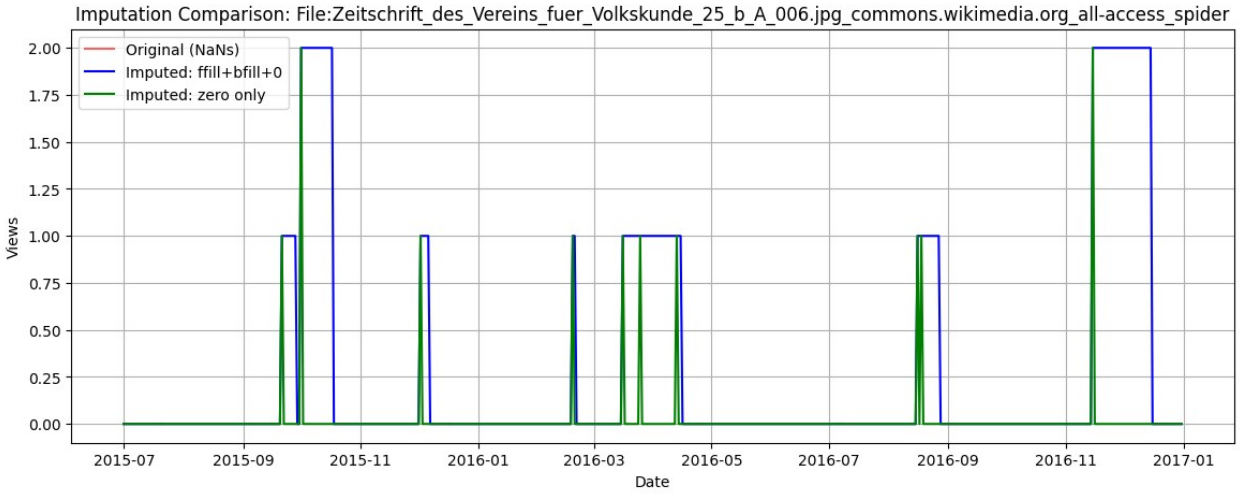
```
    plt.plot(dates, zero_vals, label="Imputed: zero only",
color="green")

    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)
```

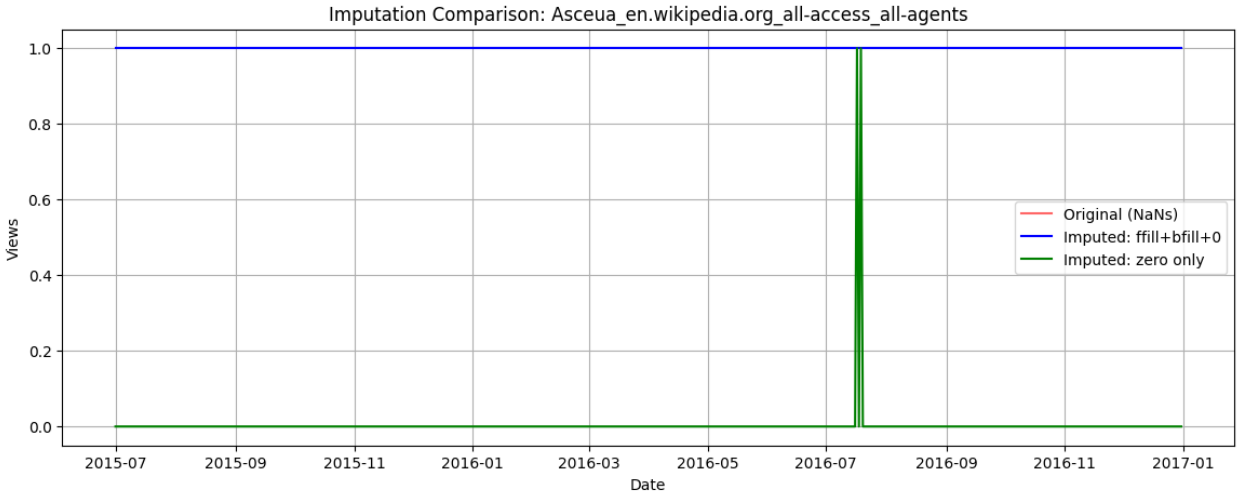



Imputation Comparison: W_(□□□)_zh.wikipedia.org_all-access_spider
Original (NaNs)
Imputed: ffill+bfill+0
Imputed: zero only
Views
Date
0
100
200
300
400
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01


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

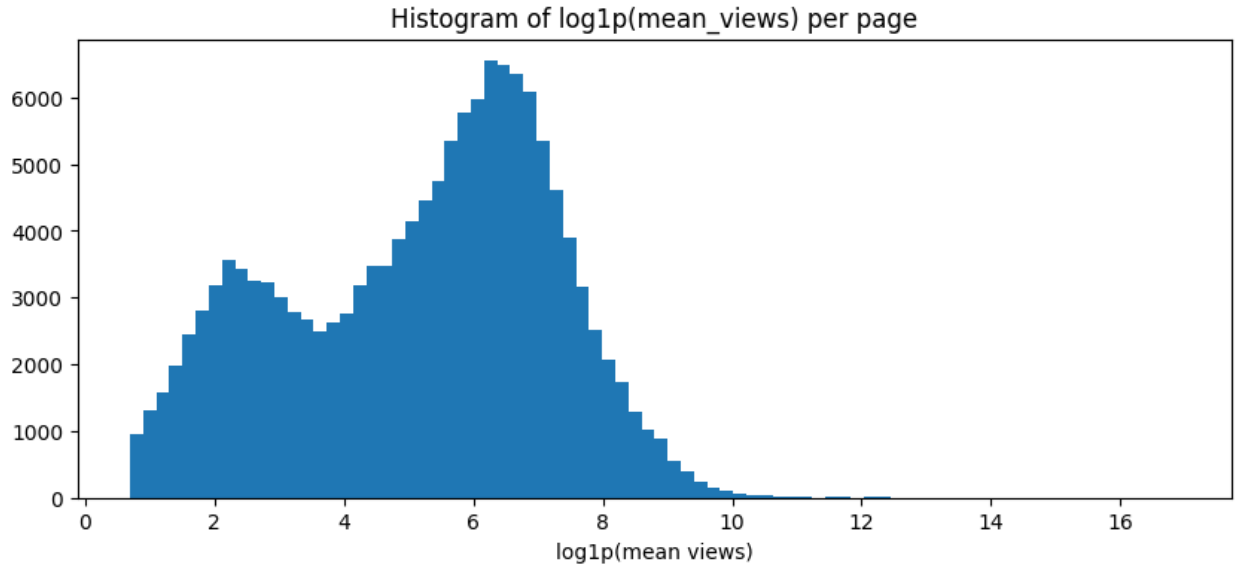
- **Global summary stats**
  - Counts, date range, missingness overview
- **Per-language statistics**
  - Total, mean, median views
  - Number of pages per language
- **Device & Agent breakdowns**
  - Desktop / Mobile / All-access
  - Spider / All-agents
- **Distribution visualizations**
  - Log-scale 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

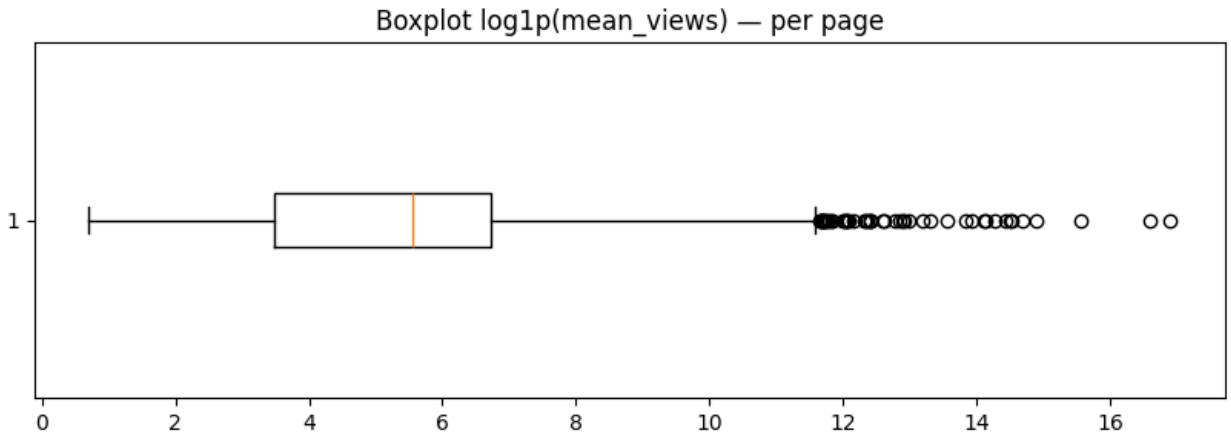
page-level summary stats (mean/median/std) and global distribution → Univariate

```
# 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

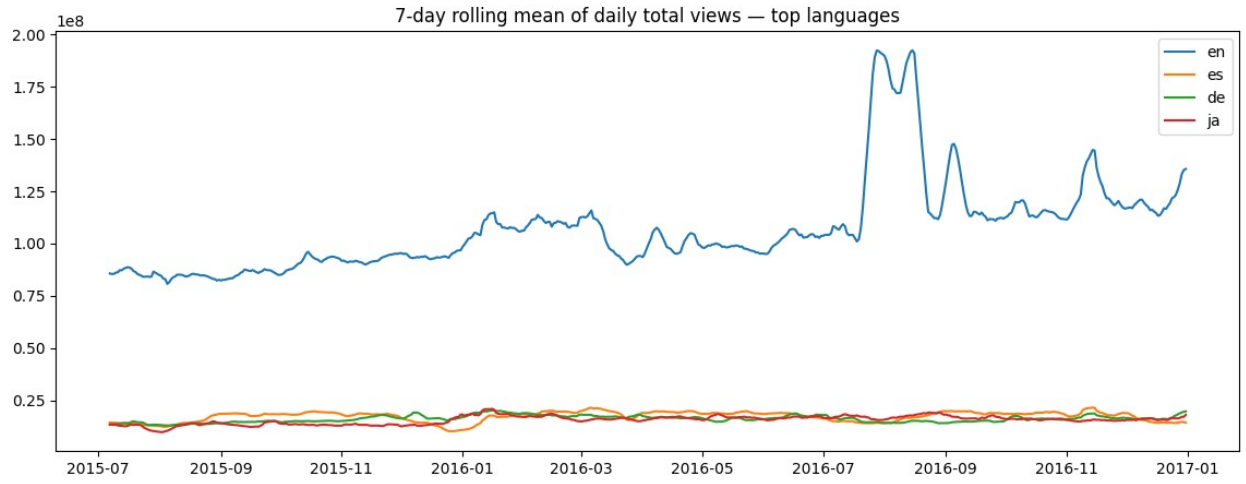
```
# daily aggregate per language (memory efficient)
# compute daily totals by language: result will be DataFrame indexed
by dates with languages as columns
lang_daily = train_clean.groupby('Language')[date_cols].sum().T
lang_daily.index = pd.to_datetime(lang_daily.index)
print("lang_daily shape (dates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

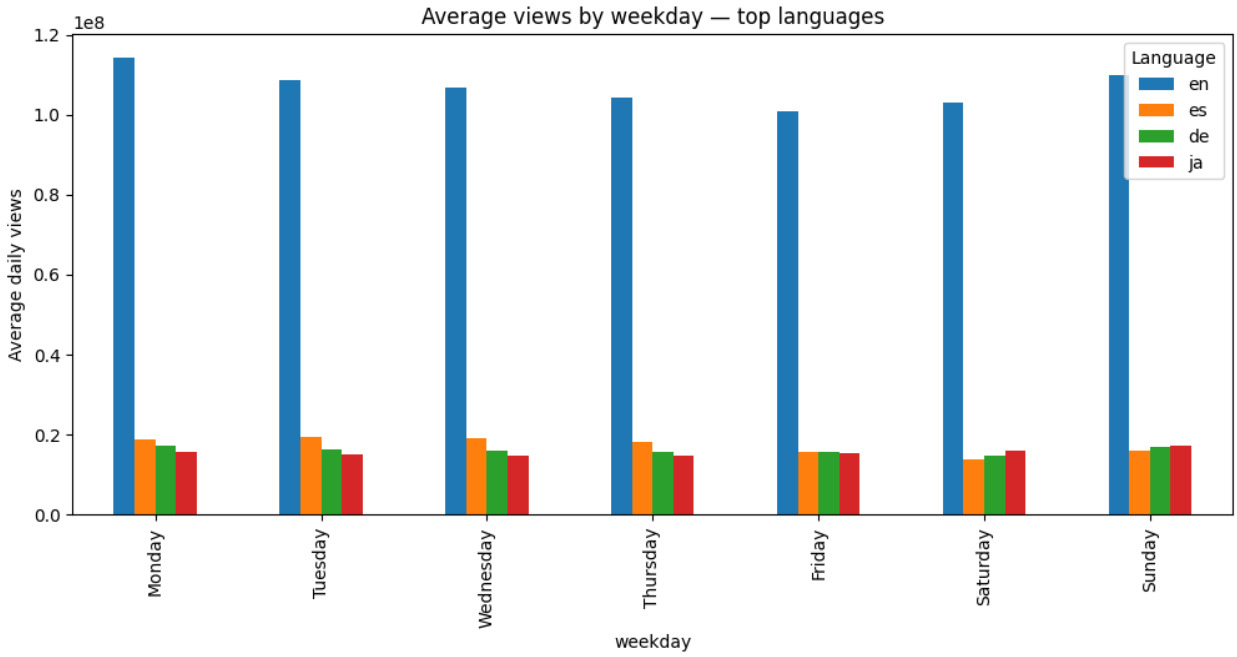
- **Other languages** are steady and less volatile, reflecting more consistent regional usage patterns.

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**.

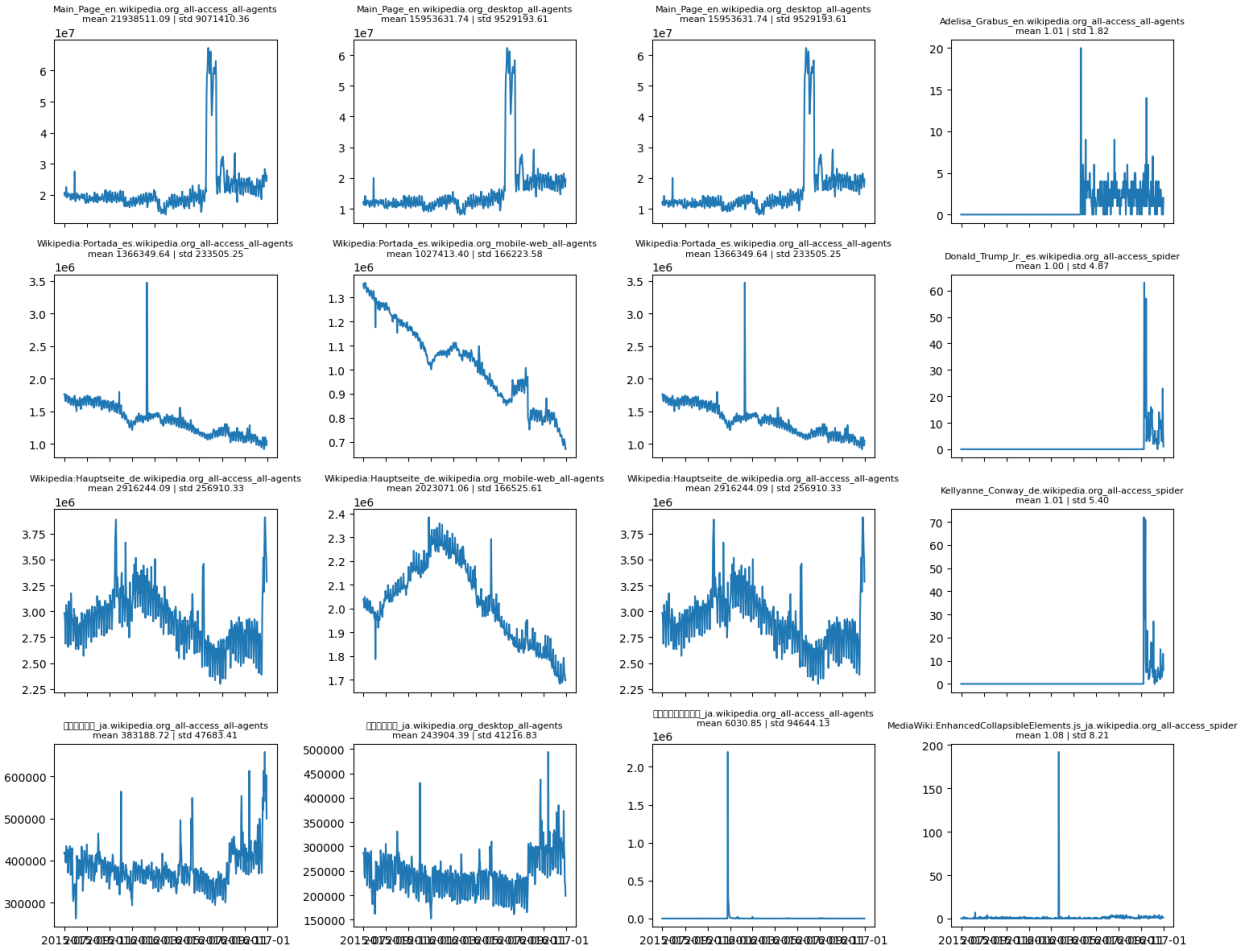
```
# pick representative pages to inspect deeply
# selection strategy: top by mean, high-variance, and random low-
traffic
def pick_pages_for_lang(df, lang, top_k=2, var_k=1, rand_k=1,
seed=42):
    grp = df[df['Language']==lang]
    top = grp.sort_values('mean_views', ascending=False).head(top_k)
['Page'].tolist()
    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