youtube_analysis

August 28, 2025

[1]:
```
import os
import sys
import time
import math
import json
import re
from datetime import datetime, timezone, timedelta
import requests
import pandas as pd

# Add your API KEY below
os.environ["YT_API_KEY"] = ""
API_KEY = os.getenv("YT_API_KEY")
if not API_KEY:
    sys.exit("Please set environment variable YT_API_KEY with your YouTube Data␣
 ↪API v3 key.")

YOUTUBE_API = "https://www.googleapis.com/youtube/v3"

# ------------------------------
# TODO: Paste your channels here
# You can use full URLs (e.g., "https://www.youtube.com/@AlexTheAnalyst")
# OR just handles like "@AlexTheAnalyst"
# OR channel URLs like "https://www.youtube.com/channel/UC123..."
# ------------------------------
CHANNEL_URLS = [
    "https://www.youtube.com/@AlexTheAnalyst",
    "https://www.youtube.com/@codebasics",
    "https://www.youtube.com/@statquest",
    "https://www.youtube.com/@3blue1brown",
    "https://www.youtube.com/@TwoMinutePapers",
    "https://www.youtube.com/@sentdex",
    "https://www.youtube.com/@coreyms",
    "https://www.youtube.com/@TinaHuang1",
]

# ------------- Helpers --------------
```

```
HANDLE_RE = re.compile(r"(?:https?://(?:www\.)?youtube\.com/)?@([A-Za-z0-9_.
 ↪-]+)")
CHANNEL_ID_RE = re.compile(r"(?:https?://(?:www\.)?youtube\.com/)?channel/
 ↪([A-Za-z0-9_-]{20,})")

def extract_handle_or_channel_id(url_or_handle: str):
    s = url_or_handle.strip()
    # channel ID?
    m = CHANNEL_ID_RE.match(s)
    if m:
        return {"type": "channel_id", "value": m.group(1)}
    # handle?
    if s.startswith("@"):
        return {"type": "handle", "value": s[1:]}
    m = HANDLE_RE.match(s)
    if m:
        return {"type": "handle", "value": m.group(1)}
    # last resort: treat as search query
    return {"type": "search", "value": s}

def yt_get(path, **params):
    params["key"] = API_KEY
    r = requests.get(f"{YOUTUBE_API}/{path}", params=params, timeout=30)
    r.raise_for_status()
    return r.json()

def resolve_channel_id(item):
    """Resolve input to a canonical channelId using search (robust for handles).
 ↪"""
    itype = item["type"]
    val = item["value"]

    if itype == "channel_id":
        return val

    if itype in ("handle", "search"):
        # Use search to find the channel
        # q = handle or free text; type=channel limits results to channels
        data = yt_get("search",
                      part="snippet",
                      q=val,
                      type="channel",
                      maxResults=1)
        items = data.get("items", [])
        if not items:
            raise ValueError(f"Could not resolve channel from: {val}")
```

```
        return items[0]["snippet"]["channelId"]

    raise ValueError(f"Unsupported identifier: {item}")

def iso8601_duration_to_seconds(iso_dur: str) -> int:
    """
    Parse ISO 8601 durations like 'PT12M34S', 'PT1H2M', 'PT45S', 'P0D', etc.
    """
    # Pattern covers H, M, S; days/months/years are (practically) not used by␣
 ↪YouTube durations
    pattern = re.compile(r'^P(?:\d+Y)?(?:\d+M)?(?:\d+D)?(?:T(?:(\d+)H)?(?:
 ↪(\d+)M)?(?:(\d+)S)?)?$')
    m = pattern.match(iso_dur)
    if not m:
        return 0
    h = int(m.group(1) or 0)
    m_ = int(m.group(2) or 0)
    s = int(m.group(3) or 0)
    return h * 3600 + m_ * 60 + s

def safe_int(v):
    try:
        return int(v)
    except:
        return None

def is_short_video(duration_seconds: int, title: str) -> bool:
    title_lower = (title or "").lower()
    return (duration_seconds is not None and duration_seconds <= 60) or␣
 ↪("#short" in title_lower or "#shorts" in title_lower)

FAMILIES = [
    ("python", ["python", "pandas", "numpy", "polars", "jupyter"]),
    ("r", [" r ", "tidyverse", "dplyr", "ggplot", "shiny"]),
    ("sql", [
        "sql", "postgres", "mysql", "sqlite", "bigquery", "redshift",␣
 ↪"snowflake",
        "select ", " join", "cte", "window function"
    ]),
    ("ml", ["machine learning", " ml ", "scikit", "sklearn", "xgboost",␣
 ↪"lightgbm"]),
    ("deep learning", ["deep learning", "neural network", "pytorch",␣
 ↪"tensorflow", "llm", "transformer"]),
    ("genai/llm", ["chatgpt", "gpt", "llama", "langchain", "rag", "prompt"]),
    ("nlp", ["nlp", "token", "bert", "gpt-"]),
    ("computer vision", ["computer vision", "opencv", "yolo"]),
```

```
    ("time series", ["time series", "forecast", "arima", "prophet"]),
    ("statistics", ["statistics", "regression", "hypothesis", "p-value",␣
 ↪"bayes", "probability"]),
    ("data viz", ["visualization", "data viz", "plotly", "tableau", "power bi",␣
 ↪"dashboard",
                  "matplotlib", "seaborn", "ggplot"]),
    ("spark/databricks", ["spark", "databricks"]),
    ("airflow", ["airflow"]),
    ("excel", ["excel", "vlookup", "xlookup", "pivot", "power query"]),
    ("cloud/mlops", ["aws", "gcp", "azure", "s3", "mlflow", "docker",␣
 ↪"kubernetes", "mlops"]),
    ("career/interview", ["interview", "resume", "cv", "portfolio", "career",␣
 ↪"job",
                          "roadmap", "my path", "how i became"]),
    ("projects/case", ["project", "case study", "end-to-end", "capstone"]),
    ("tutorial/guide", ["tutorial", "guide", "crash course", "step-by-step",␣
 ↪"hands-on"]),
    ("math", ["calculus", "linear algebra", "eigen", "gradient", "matrix",␣
 ↪"algebra"]),
    ("livestream/qa", ["live", "livestream", "q&a", "ama"]),
]

def _norm_text(s: str) -> str:
    if not s: return ""
    # add spaces around punctuation to help token boundaries
    s = s.lower()
    s = re.sub(r"[_/\\\-]+", " ", s)
    return f" {s} "

def extract_topics_from_text(title: str = "", description: str = "", tags: str␣
 ↪= ""):
    """Return up to 3 topic labels based on lightweight keyword hits across␣
 ↪title/desc/tags."""
    t = _norm_text(" ".join([title or "", description or "", " ".join(tags) if␣
 ↪isinstance(tags, list) else (tags or "")]))

    hits = []
    for label, kws in FAMILIES:
        if any(kw in t for kw in kws):
            hits.append(label)

    # de-dup while preserving order
    uniq = []
    for h in hits:
        if h not in uniq:
            uniq.append(h)
```

```
    # Fallback if nothing matched
    if not uniq:
        uniq = ["general/data"]

    # Return up to 3; you can keep first as primary, second as secondary
    return uniq[:3]

def to_utc_dt(iso_str):
    return datetime.fromisoformat(iso_str.replace("Z", "+00:00")).
 ↪astimezone(timezone.utc)

def channel_details(channel_id):
    data = yt_get("channels",
                  part="snippet,statistics,brandingSettings,contentDetails",
                  id=channel_id)

    items = data.get("items", [])
    if not items:
        return None
    it = items[0]

    snippet = it.get("snippet", {})
    stats = it.get("statistics", {})
    branding = it.get("brandingSettings", {})
    # keywords can be a long string in brandingSettings.channel.keywords
    keywords = (branding.get("channel", {}) or {}).get("keywords")

    return {
        "channel_id": channel_id,
        "channel_title": snippet.get("title"),
        "custom_url": snippet.get("customUrl"),
        "published_at": snippet.get("publishedAt"),
        "country": snippet.get("country"),
        "default_language": snippet.get("defaultLanguage"),
        "subscribers": safe_int(stats.get("subscriberCount")),
        "lifetime_views": safe_int(stats.get("viewCount")),
        "video_count": safe_int(stats.get("videoCount")),
        "keywords": keywords
    }

def list_channel_videos_last12m(channel_id, max_items=300):
    """Use search to fetch recent videos within last 12 months (or up to␣
 ↪max_items)."""
    published_after = (datetime.now(timezone.utc) - timedelta(days=365)).
 ↪isoformat().replace("+00:00", "Z")
```

```
    videos = []
    page_token = None

    while True:
        data = yt_get("search",
                      part="snippet",
                      channelId=channel_id,
                      type="video",
                      order="date",
                      publishedAfter=published_after,
                      maxResults=50,
                      pageToken=page_token if page_token else None)
        items = data.get("items", [])
        for it in items:
            videos.append({
                "video_id": it["id"]["videoId"],
                "title": it["snippet"]["title"],
                "published_at": it["snippet"]["publishedAt"],
                "description": it["snippet"].get("description"),
                "channel_id": channel_id
            })
            if len(videos) >= max_items:
                break
        if len(videos) >= max_items:
            break
        page_token = data.get("nextPageToken")
        if not page_token:
            break

    return videos

def chunked(iterable, n):
    for i in range(0, len(iterable), n):
        yield iterable[i:i+n]

def enrich_videos_stats_and_details(video_rows):
    """Call videos.list in batches to add statistics, contentDetails,␣
 ↪topicCategories, tags, etc."""
    if not video_rows:
        return video_rows

    id_list = [v["video_id"] for v in video_rows]

    id_to_details = {}
    for batch in chunked(id_list, 50):
        data = yt_get("videos",
                      part="snippet,contentDetails,statistics,topicDetails",
```

```
                    id=",".join(batch))
        for it in data.get("items", []):
            vid = it["id"]
            snip = it.get("snippet", {})
            stats = it.get("statistics", {})
            cdet = it.get("contentDetails", {})
            tdet = it.get("topicDetails", {}) or {}

            tags = snip.get("tags") or []
            topic_categories = tdet.get("topicCategories") or []
            id_to_details[vid] = {
                "duration_sec": iso8601_duration_to_seconds(cdet.
 ↪get("duration", "PT0S")),
                "view_count": safe_int(stats.get("viewCount")),
                "like_count": safe_int(stats.get("likeCount")),
                "comment_count": safe_int(stats.get("commentCount")),
                "made_for_kids": snip.get("madeForKids"),
                "live_flag": snip.get("liveBroadcastContent"),
                "default_audio_language": snip.get("defaultAudioLanguage"),
                "topic_categories": "|".join(topic_categories) if␣
 ↪topic_categories else None,
                "tags": "|".join(tags) if tags else None,
                "full_title": snip.get("title"),  # overrides if different␣
 ↪casing
                "thumbnails": json.dumps(snip.get("thumbnails", {})),
            }
        time.sleep(0.1)  # be gentle with quota

    # merge
    out = []
    for row in video_rows:
        det = id_to_details.get(row["video_id"], {})
        merged = {**row, **det}
        out.append(merged)
    return out

def compute_features_df(videos_df, channels_df):
    if videos_df.empty:
        return pd.DataFrame()

    now = datetime.now(timezone.utc)

    def _age_days(published_at):
        try:
            dt = to_utc_dt(published_at)
            days = (now - dt).total_seconds() / 86400.0
            return max(1.0, days)
```

```
        except:
            return 1.0

    videos_df["age_days"] = videos_df["published_at"].apply(_age_days)
    videos_df["views_per_day"] = videos_df.apply(
        lambda r: (r.get("view_count") or 0) / r["age_days"], axis=1
    )

    # engagement proxies
    def per_1k(n, denom):
        if not denom or denom == 0 or n is None:
            return 0.0
        return (n / denom) * 1000.0

    videos_df["likes_per_1k"] = videos_df.apply(
        lambda r: per_1k(r.get("like_count"), r.get("view_count")), axis=1
    )
    videos_df["comments_per_1k"] = videos_df.apply(
        lambda r: per_1k(r.get("comment_count"), r.get("view_count")), axis=1
    )
    videos_df["engagement_rate"] = videos_df.apply(
        lambda r: ((r.get("like_count") or 0) + (r.get("comment_count") or 0)) /
↪ (r.get("view_count") or 1),
        axis=1
    )

    # value density
    def views_per_minute(r):
        dur = r.get("duration_sec") or 0
        if dur <= 0:
            return None
        return (r.get("view_count") or 0) / (dur / 60.0)
    videos_df["views_per_min"] = videos_df.apply(views_per_minute, axis=1)

    # title stats
    videos_df["title_len"] = videos_df["title"].fillna("").apply(len)
    videos_df["emoji_cnt"] = videos_df["title"].fillna("").apply(
        lambda s: sum(1 for ch in s if ord(ch) > 10000)
    )
    videos_df["question_mark_flag"] = videos_df["title"].fillna("").
↪apply(lambda s: "?" in s)

    # Shorts flag
    videos_df["is_short"] = videos_df.apply(
        lambda r: is_short_video(int(r.get("duration_sec") or 0), r.
↪get("title") or ""),
        axis=1
```

```
    )

    # time features
    def _weekday(iso_ts):
        try:
            return to_utc_dt(iso_ts).weekday()  # 0=Mon
        except:
            return None

    def _hour(iso_ts):
        try:
            return to_utc_dt(iso_ts).hour
        except:
            return None

    videos_df["weekday"] = videos_df["published_at"].apply(_weekday)
    videos_df["hour"] = videos_df["published_at"].apply(_hour)

    # topic tagging
    def topics_from_row(r):
        labs = extract_topics_from_text(
            title=r.get("title", ""),
            description=r.get("description", ""),
            tags=r.get("tags", []),   # string or list - the extractor handles␣
↪both
        )
        return pd.Series(
            {
                "topic_primary":  labs[0] if len(labs) > 0 else "general/data",
                "topic_secondary": labs[1] if len(labs) > 1 else None,
                # optional:
                # "topic_third":     labs[2] if len(labs) > 2 else None,
            },
            dtype="object",
        )

    topics_df = videos_df.apply(topics_from_row, axis=1)
    videos_df = pd.concat(
    [videos_df.drop(columns=["topic_primary", "topic_secondary"],␣
↪errors="ignore"), topics_df],
    axis=1,
    )


    # views_per_sub (join subs)
    submap = dict(zip(channels_df["channel_id"], channels_df["subscribers"]))
    videos_df["subscribers"] = videos_df["channel_id"].map(submap)
```

```
    def vps(r):
        subs = r.get("subscribers") or 0
        if subs <= 0:
            return None
        return (r.get("view_count") or 0) / subs
    videos_df["views_per_sub"] = videos_df.apply(vps, axis=1)

    # Build features.csv
    features_cols = [
        "video_id", "channel_id", "title", "published_at",
        "age_days", "views_per_day", "engagement_rate",
        "likes_per_1k", "comments_per_1k", "views_per_min",
        "title_len", "emoji_cnt", "question_mark_flag",
        "duration_sec", "is_short", "weekday", "hour",
        "topic_primary", "topic_secondary", "views_per_sub"
    ]
    features_df = videos_df[features_cols].copy()
    return features_df, videos_df

# ------------ Main --------------

def main():
    if not CHANNEL_URLS:
        print("Please add channel links/handles to CHANNEL_URLS and re-run.")
        return

    print("Resolving channels...")
    channel_ids = []
    for raw in CHANNEL_URLS:
        ident = extract_handle_or_channel_id(raw)
        cid = resolve_channel_id(ident)
        channel_ids.append((raw, cid))
        time.sleep(0.1)

    # Channels info
    channels = []
    for raw, cid in channel_ids:
        info = channel_details(cid)
        if info:
            channels.append(info)
        time.sleep(0.1)

    channels_df = pd.DataFrame(channels)
    if channels_df.empty:
        print("No channels resolved; exiting.")
        return
```

```
    # Videos (last 12 months up to 300)
    videos = []
    for raw, cid in channel_ids:
        vids_meta = list_channel_videos_last12m(cid, max_items=300)
        videos.extend(vids_meta)
        time.sleep(0.2)

    # Enrich videos with stats/details
    videos = enrich_videos_stats_and_details(videos)
    videos_df = pd.DataFrame(videos)

    # Compute features
    features_df, full_videos_df = compute_features_df(videos_df, channels_df)

    # Save CSVs
    channels_df.to_csv("channels.csv", index=False)
    full_videos_df.to_csv("videos.csv", index=False)
    features_df.to_csv("features.csv", index=False)

    print(f"Saved: channels.csv ({len(channels_df)} rows), "
          f"videos.csv ({len(full_videos_df)} rows), "
          f"features.csv ({len(features_df)} rows)")

if __name__ == "__main__":
    main()
```

```
Resolving channels…
Saved: channels.csv (8 rows), videos.csv (476 rows), features.csv (476 rows)
```

[2]:
```
def clean_published_at(df, col="published_at"):
    """
    Cleans and standardizes a datetime column (default: 'published_at')
    - Handles trailing 'Z' → '+00:00'
    - Handles fractional seconds vs no fractions
    - Ensures timezone-aware (UTC)
    - Returns the DataFrame with parsed column
    """

    # Work on a copy of the column as string
    pub = df[col].astype(str).str.strip()

    # Normalize common ISO patterns
    pub = pub.str.replace("Z", "+00:00", regex=False).str.replace("z", "+00:
↪00", regex=False)

    # First parse attempt
    df[col] = pd.to_datetime(pub, errors="coerce", utc=True)
```

```
    # Fallback for rows still NaT (often no fractional seconds)
    mask_nat = df[col].isna()
    if mask_nat.any():
        fallback = pd.to_datetime(pub[mask_nat], format="%Y-%m-%dT%H:%M:%S%z",␣
↪errors="coerce")
        df.loc[mask_nat, col] = fallback

    # Optional logging
    n_nat = df[col].isna().sum()
    if n_nat > 0:
        print(f"[WARN] {n_nat} rows in '{col}' could not be parsed.")

    return df
```

[3]:
```
# Load your CSVs
channels = pd.read_csv("channels.csv")
videos = pd.read_csv("videos.csv")
features = pd.read_csv("features.csv")

# --- Quick overviews ---
print("Channels dataset:")
print(channels.info())
print(channels.head(), "\n")
```

```
Channels dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   channel_id        8 non-null      object
 1   channel_title     8 non-null      object
 2   custom_url        8 non-null      object
 3   published_at      8 non-null      object
 4   country           8 non-null      object
 5   default_language  0 non-null      float64
 6   subscribers       8 non-null      int64
 7   lifetime_views    8 non-null      int64
 8   video_count       8 non-null      int64
 9   keywords          8 non-null      object
dtypes: float64(1), int64(3), object(6)
memory usage: 772.0+ bytes
None
                channel_id                channel_title        custom_url  \
0  UC7cs8q-gJRlGwj4A8OmCmXg             Alex The Analyst   @alextheanalyst
1  UCh9nVJoWXmFb7sLApWGcLPQ                    codebasics       @codebasics
2  UCtYLUTtgS3k1Fg4y5tAhLbw  StatQuest with Josh Starmer        @statquest
```

```
3  UCYO_jab_esuFRV4b17AJtAw                  3Blue1Brown      @3blue1brown
4  UCbfYPyITQ-7l4upoX8nvctg            Two Minute Papers  @twominutepapers

                 published_at country  default_language  subscribers  \
0  2020-01-08T05:04:24.970712Z      US               NaN      1140000
1         2015-11-07T17:29:46Z      US               NaN      1370000
2         2011-05-24T01:52:48Z      US               NaN      1480000
3         2015-03-03T23:11:55Z      US               NaN      7600000
4         2006-08-18T00:05:41Z      HU               NaN      1680000

   lifetime_views  video_count  \
0        53686282          389
1       140793273         1104
2        82588243          291
3       674504510          218
4       154349863          996

                                          keywords
0  "Data Analyst" "Data Analyst Salary" "How to b…
1  "programming tutorial" python git github "juli…
2  Statistics "Machine Learning" "Data Science" S…
3                                       Mathematics
4  "two minute papers" ai "machine learning" soft…
```

[4]:
```
channels = channels.drop(columns=["default_language"])
```

[5]:
```
channels = clean_published_at(channels, col="published_at")
```

[6]:
```
channels.dtypes
# channel_id/object, channel_title/object, custom_url/object,
# published_at/datetime64[ns, UTC], country/object,
# subscribers/int64, lifetime_views/int64, video_count/int64, keywords/object

channels.isna().sum()
# should be 0 for published_at
```

[6]:
```
channel_id        0
channel_title     0
custom_url        0
published_at      0
country           0
subscribers       0
lifetime_views    0
video_count       0
keywords          0
dtype: int64
```

[7]:
```
print("Channels dataset:")
print(channels.info())
```

```
Channels dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   channel_id      8 non-null      object
 1   channel_title   8 non-null      object
 2   custom_url      8 non-null      object
 3   published_at    8 non-null      datetime64[ns, UTC]
 4   country         8 non-null      object
 5   subscribers     8 non-null      int64
 6   lifetime_views  8 non-null      int64
 7   video_count     8 non-null      int64
 8   keywords        8 non-null      object
dtypes: datetime64[ns, UTC](1), int64(3), object(5)
memory usage: 708.0+ bytes
None
```

[8]:
```
print("Videos dataset:")
print(videos.info())
print(videos.head(), "\n")
```

```
Videos dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   video_id                476 non-null    object
 1   title                   476 non-null    object
 2   published_at            476 non-null    object
 3   description             337 non-null    object
 4   channel_id              476 non-null    object
 5   duration_sec            476 non-null    int64
 6   view_count              476 non-null    int64
 7   like_count              476 non-null    int64
 8   comment_count           476 non-null    int64
 9   made_for_kids           0 non-null      float64
 10  live_flag               476 non-null    object
 11  default_audio_language  476 non-null    object
 12  topic_categories        408 non-null    object
 13  tags                    259 non-null    object
 14  full_title              476 non-null    object
 15  thumbnails              476 non-null    object
```

```
 16  age_days                476 non-null    float64
 17  views_per_day           476 non-null    float64
 18  likes_per_1k            476 non-null    float64
 19  comments_per_1k         476 non-null    float64
 20  engagement_rate         476 non-null    float64
 21  views_per_min           476 non-null    float64
 22  title_len               476 non-null    int64
 23  emoji_cnt               476 non-null    int64
 24  question_mark_flag      476 non-null    bool
 25  is_short                476 non-null    bool
 26  weekday                 476 non-null    int64
 27  hour                    476 non-null    int64
 28  topic_primary           476 non-null    object
 29  topic_secondary         126 non-null    object
 30  subscribers             476 non-null    int64
 31  views_per_sub           476 non-null    float64
dtypes: bool(2), float64(8), int64(9), object(13)
memory usage: 112.6+ KB
None
      video_id                                                title  \
0  Rcpidz-jnZQ                                          SQL is King
1  QzvA7r-WndM                                What is Git and GitHub?
2  yhlqKsYpzgE  Alex The Analyst Q/A Livestream | Come Ask Me …
3  kk5zEOQzTmQ  Data Visualization and Presentation in R | R f…
4  TP2OJuZhbIQ              Things I Learned as a Data Analyst p1

           published_at                                        description  \
0  2025-08-27T11:05:28Z                                                NaN
1  2025-08-26T12:00:54Z  Take my Full Git and GitHub Course: https://ww…
2  2025-08-21T14:09:29Z  Come ask me anything in my Weekly Q/A! In this…
3  2025-08-19T12:01:22Z  Take my Full R Programming for Data Analysts C…
4  2025-08-15T11:46:04Z                                                NaN

                 channel_id  duration_sec  view_count  like_count  \
0  UC7cs8q-gJRlGwj4A8OmCmXg            45        4208         196
1  UC7cs8q-gJRlGwj4A8OmCmXg           512        4925         228
2  UC7cs8q-gJRlGwj4A8OmCmXg          3673        2917         126
3  UC7cs8q-gJRlGwj4A8OmCmXg          1264        2693          77
4  UC7cs8q-gJRlGwj4A8OmCmXg            38        6881         240

   comment_count  made_for_kids  … title_len emoji_cnt question_mark_flag  \
0              8            NaN  …        11         0              False
1              7            NaN  …        23         0               True
2             11            NaN  …        54         0              False
3             12            NaN  …        70         0              False
4             13            NaN  …        37         0              False

  is_short weekday hour     topic_primary  topic_secondary  subscribers  \
```

```
0     True       2   11               sql              NaN      1140000
1    False       1   12  career/interview              NaN      1140000
2    False       3   14  career/interview   livestream/qa      1140000
3    False       1   12                 r        data viz      1140000
4     True       4   11      general/data              NaN      1140000

   views_per_sub
0       0.003691
1       0.004320
2       0.002559
3       0.002362
4       0.006036

[5 rows x 32 columns]
```

[9]:
```
cols_keep = [
    "video_id","channel_id","title","published_at",
    "duration_sec","is_short","live_flag",
    "view_count","like_count","comment_count",
    "age_days","views_per_day","likes_per_1k","comments_per_1k",
    "engagement_rate","views_per_min","views_per_sub",
    "weekday","hour",
    "topic_primary","topic_secondary","tags","topic_categories",
    "subscribers","default_audio_language"
]

videos = videos[cols_keep].copy()

# If you want to drop text-heavy columns now:
videos = videos.drop(columns=["tags","topic_categories"])
```

[10]:
```
videos = clean_published_at(videos, col="published_at")
```

[11]:
```
videos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   video_id              476 non-null    object
 1   channel_id            476 non-null    object
 2   title                 476 non-null    object
 3   published_at          476 non-null    datetime64[ns, UTC]
 4   duration_sec          476 non-null    int64
 5   is_short              476 non-null    bool
 6   live_flag             476 non-null    object
```

```
 7   view_count              476 non-null    int64
 8   like_count              476 non-null    int64
 9   comment_count           476 non-null    int64
 10  age_days                476 non-null    float64
 11  views_per_day           476 non-null    float64
 12  likes_per_1k            476 non-null    float64
 13  comments_per_1k         476 non-null    float64
 14  engagement_rate         476 non-null    float64
 15  views_per_min           476 non-null    float64
 16  views_per_sub           476 non-null    float64
 17  weekday                 476 non-null    int64
 18  hour                    476 non-null    int64
 19  topic_primary           476 non-null    object
 20  topic_secondary         126 non-null    object
 21  subscribers             476 non-null    int64
 22  default_audio_language  476 non-null    object
dtypes: bool(1), datetime64[ns, UTC](1), float64(7), int64(7), object(7)
memory usage: 82.4+ KB
```

[12]:
```
# --- Ensure datetimes are proper and preserved as UTC ---
videos["published_at"] = pd.to_datetime(videos["published_at"], utc=True,␣
 ↪errors="coerce")

# --- Cast IDs & labels to strings (good for Tableau) ---
str_cols = [
    "video_id","channel_id","live_flag","default_audio_language",
    "topic_primary","topic_secondary","title"
]
for c in str_cols:
    if c in videos.columns:
        videos[c] = videos[c].astype("string")

# --- Booleans as TRUE/FALSE (Tableau friendly) ---
if "is_short" in videos.columns:
    videos["is_short"] = videos["is_short"].astype("boolean")

# --- Numeric columns: keep numeric ---
int_cols =␣
 ↪["view_count","like_count","comment_count","duration_sec","subscribers"]
for c in int_cols:
    if c in videos.columns:
        videos[c] = pd.to_numeric(videos[c], errors="coerce")

float_cols = ["age_days","views_per_day","likes_per_1k","comments_per_1k",
              "engagement_rate","views_per_min","views_per_sub"]
for c in float_cols:
    if c in videos.columns:
```

```
        videos[c] = pd.to_numeric(videos[c], errors="coerce")

# --- Optional: also keep weekday/hour as numbers (Tableau can bin/order) ---
for c in ["weekday","hour"]:
    if c in videos.columns:
        videos[c] = pd.to_numeric(videos[c], errors="coerce")
```

[13]:
```
videos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 23 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   video_id                476 non-null    string
 1   channel_id              476 non-null    string
 2   title                   476 non-null    string
 3   published_at            476 non-null    datetime64[ns, UTC]
 4   duration_sec            476 non-null    int64
 5   is_short                476 non-null    boolean
 6   live_flag               476 non-null    string
 7   view_count              476 non-null    int64
 8   like_count              476 non-null    int64
 9   comment_count           476 non-null    int64
 10  age_days                476 non-null    float64
 11  views_per_day           476 non-null    float64
 12  likes_per_1k            476 non-null    float64
 13  comments_per_1k         476 non-null    float64
 14  engagement_rate         476 non-null    float64
 15  views_per_min           476 non-null    float64
 16  views_per_sub           476 non-null    float64
 17  weekday                 476 non-null    int64
 18  hour                    476 non-null    int64
 19  topic_primary           476 non-null    string
 20  topic_secondary         126 non-null    string
 21  subscribers             476 non-null    int64
 22  default_audio_language  476 non-null    string
dtypes: boolean(1), datetime64[ns, UTC](1), float64(7), int64(7), string(7)
memory usage: 82.9 KB
```

[ ]:

[ ]:

[14]:
```
print("Features dataset:")
print(features.info())
print(features.head(), "\n")
```

```
Features dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   video_id            476 non-null    object
 1   channel_id          476 non-null    object
 2   title               476 non-null    object
 3   published_at        476 non-null    object
 4   age_days            476 non-null    float64
 5   views_per_day       476 non-null    float64
 6   engagement_rate     476 non-null    float64
 7   likes_per_1k        476 non-null    float64
 8   comments_per_1k     476 non-null    float64
 9   views_per_min       476 non-null    float64
 10  title_len           476 non-null    int64
 11  emoji_cnt           476 non-null    int64
 12  question_mark_flag  476 non-null    bool
 13  duration_sec        476 non-null    int64
 14  is_short            476 non-null    bool
 15  weekday             476 non-null    int64
 16  hour                476 non-null    int64
 17  topic_primary       476 non-null    object
 18  topic_secondary     126 non-null    object
 19  views_per_sub       476 non-null    float64
dtypes: bool(2), float64(7), int64(5), object(6)
memory usage: 68.0+ KB
None
      video_id                channel_id  \
0  Rcpidz-jnZQ  UC7cs8q-gJRlGwj4A8OmCmXg
1  QzvA7r-WndM  UC7cs8q-gJRlGwj4A8OmCmXg
2  yhlqKsYpzgE  UC7cs8q-gJRlGwj4A8OmCmXg
3  kk5zEOQzTmQ  UC7cs8q-gJRlGwj4A8OmCmXg
4  TP2OJuZhbIQ  UC7cs8q-gJRlGwj4A8OmCmXg

                                                 title          published_at  \
0                                          SQL is King  2025-08-27T11:05:28Z
1                                What is Git and GitHub?  2025-08-26T12:00:54Z
2  Alex The Analyst Q/A Livestream | Come Ask Me …  2025-08-21T14:09:29Z
3  Data Visualization and Presentation in R | R f…  2025-08-19T12:01:22Z
4              Things I Learned as a Data Analyst p1  2025-08-15T11:46:04Z

    age_days  views_per_day  engagement_rate  likes_per_1k  comments_per_1k  \
0   1.163805    3615.727543         0.048479     46.577947         1.901141
1   2.125309    2317.309949         0.047716     46.294416         1.421320
2   7.036015     414.581256         0.046966     43.195063         3.770998
3   9.124985     295.123771         0.033049     28.592648         4.455997
```

```
4  13.135610     523.843199         0.036768     34.878651         1.889260

   views_per_min  title_len  emoji_cnt  question_mark_flag  duration_sec  \
0    5610.666667         11          0               False            45
1     577.148438         23          0                True           512
2      47.650422         54          0               False          3673
3     127.832278         70          0               False          1264
4   10864.736842         37          0               False            38

   is_short  weekday  hour     topic_primary topic_secondary  views_per_sub
0      True        2    11               sql             NaN       0.003691
1     False        1    12  career/interview             NaN       0.004320
2     False        3    14  career/interview   livestream/qa       0.002559
3     False        1    12                 r        data viz       0.002362
4      True        4    11      general/data             NaN       0.006036
```

[15]:
```
# 1) Parse datetime in UTC
features["published_at"] = pd.to_datetime(features["published_at"], utc=True,␣
 ↪errors="coerce")

# 2) Integer columns
int_cols = ["duration_sec", "title_len", "emoji_cnt", "weekday", "hour"]
for c in int_cols:
    features[c] = pd.to_numeric(features[c], errors="coerce").astype("Int64")

# 3) Float columns (ratios/derived)
float_cols = [
    "age_days","views_per_day","engagement_rate","likes_per_1k",
    "comments_per_1k","views_per_min","views_per_sub"
]
for c in float_cols:
    features[c] = pd.to_numeric(features[c], errors="coerce")

# 4) Booleans already okay; but enforce just in case
for c in ["is_short", "question_mark_flag"]:
    features[c] = features[c].astype("boolean")

# 5) Text columns as string (helps avoid mixed types)
text_cols = ["video_id","channel_id","title","topic_primary","topic_secondary"]
for c in text_cols:
    features[c] = features[c].astype("string")

# 6) Optional: fill missing topic_secondary for Tableau filters
features["topic_secondary"] = features["topic_secondary"].fillna("none")

# Quick sanity checks
```

```
assert features["duration_sec"].gt(0).all(skipna=True)
assert features["view_count"].ge(0).all(skipna=True) if "view_count" in␣
 ↪features.columns else True

features.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   video_id            476 non-null    string
 1   channel_id          476 non-null    string
 2   title               476 non-null    string
 3   published_at        476 non-null    datetime64[ns, UTC]
 4   age_days            476 non-null    float64
 5   views_per_day       476 non-null    float64
 6   engagement_rate     476 non-null    float64
 7   likes_per_1k        476 non-null    float64
 8   comments_per_1k     476 non-null    float64
 9   views_per_min       476 non-null    float64
 10  title_len           476 non-null    Int64
 11  emoji_cnt           476 non-null    Int64
 12  question_mark_flag  476 non-null    boolean
 13  duration_sec        476 non-null    Int64
 14  is_short            476 non-null    boolean
 15  weekday             476 non-null    Int64
 16  hour                476 non-null    Int64
 17  topic_primary       476 non-null    string
 18  topic_secondary     476 non-null    string
 19  views_per_sub       476 non-null    float64
dtypes: Int64(5), boolean(2), datetime64[ns, UTC](1), float64(7), string(5)
memory usage: 222.6 KB
```

[16]:
```
# --- Quick shapes ---
print("Shape of channels:", channels.shape)
print("Shape of videos:", videos.shape)
print("Shape of features:", features.shape)
```

```
Shape of channels: (8, 9)
Shape of videos: (476, 23)
Shape of features: (476, 20)
```

[17]:
```
# Save to CSV (good for Tableau)
channels.to_csv("channels_clean.csv", index=False, encoding="utf-8")
videos.to_csv("videos_clean.csv", index=False, encoding="utf-8")
features.to_csv("features_clean.csv", index=False, encoding="utf-8")
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: