**YouTube Trending Video Analytics**

Objective: Uncover patterns in trending videos across regions.

Tools: Python (pandas, Matplotlib, Seaborn, NLTK VADER), SQL (Postgres/SQLite), Tableau.

**1) Repository & Data Layout**

youtube-trending-analytics/

data/

raw/

USvideos.csv # + other country files (e.g., INvideos.csv, GBvideos.csv, ...)

US\_category\_id.json # + matching category maps per region

interim/

processed/

notebooks/

01\_clean\_standardize.ipynb

02\_sentiment.ipynb

03\_time\_series\_trending\_duration.ipynb

sql/

schema.sql

load.sql

analysis\_category\_rankings.sql

timeseries.sql

tableau/

datasource.tdsx # produced after publishing

dashboard.twbx # packaged workbook

src/

config.py

utils\_io.py

clean.py

sentiment.py

features.py

viz\_examples.py

README.md

**2) Data Dictionary (Kaggle Trending format)**

video\\_id: unique YouTube video id

title : video title

channel\\_title : channel name

category\\_id : numeric id; needs mapping via the region’s JSON

tags : pipe‑separated tags string ("|"), "\[none]" if missing

views, likes, dislikes, comment\\_count : integers

thumbnail\\_link : URL

comments\\_disabled, ratings\\_disabled, video\\_error\\_or\\_removed : booleans

description : free text

publish\\_time : ISO timestamp

trending\\_date : date the video appeared on trending (YY.DD.MM in some dumps)

**3) Standardization & Cleaning (Python)**

> Consolidates all regions, normalizes schemas, cleans text, harmonizes timestamps, and attaches human‑readable categories.

python

# src/clean.py

import pandas as pd

import json

from pathlib import Path

DTYPES = {

"views": "Int64",

"likes": "Int64",

"dislikes": "Int64",

"comment\_count": "Int64",

}

TRENDING\_DATE\_FMT = "%y.%d.%m" # e.g., '17.14.05' -> 2017-05-14

CATEGORY\_COLS = [

"video\_id","title","channel\_title","category\_id","category","tags",

"views","likes","dislikes","comment\_count","thumbnail\_link",

"comments\_disabled","ratings\_disabled","video\_error\_or\_removed",

"description","publish\_time","trending\_date","region"

]

def \_read\_category\_map(json\_path: Path) -> dict:

obj = json.loads(Path(json\_path).read\_text(encoding="utf-8"))

items = obj.get("items", [])

return {int(it["id"]): it["snippet"]["title"] for it in items}

def \_normalize\_dates(df: pd.DataFrame) -> pd.DataFrame:

# publish\_time -> UTC naïve timestamp

df["publish\_time"] = pd.to\_datetime(df["publish\_time"], errors="coerce", utc=True).dt.tz\_localize(None)

# trending\_date may be in YY.DD.MM

df["trending\_date"] = pd.to\_datetime(df["trending\_date"], format=TRENDING\_DATE\_FMT, errors="coerce")

return df

def \_clean\_text\_fields(df: pd.DataFrame) -> pd.DataFrame:

for col in ["title", "channel\_title", "description"]:

df[col] = df[col].fillna("").str.normalize("NFKC").str.strip()

df["tags"] = (

df["tags"].fillna("").replace({"[none]": ""}).str.replace("\"", "", regex=False)

)

return df

def standardize\_region(csv\_path: Path, cat\_json: Path, region\_code: str) -> pd.DataFrame:

df = pd.read\_csv(csv\_path, dtype=DTYPES, encoding="utf-8")

cat\_map = \_read\_category\_map(cat\_json)

df = \_normalize\_dates(df)

df = \_clean\_text\_fields(df)

# enforce booleans

for b in ["comments\_disabled","ratings\_disabled","video\_error\_or\_removed"]:

if b in df.columns:

df[b] = df[b].astype(bool)

else:

df[b] = False

# attach human readable category

df["category"] = df["category\_id"].map(cat\_map).fillna("Unknown")

df["region"] = region\_code.upper()

return df[CATEGORY\_COLS]

def build\_processed(raw\_dir: Path, processed\_dir: Path) -> pd.DataFrame:

processed\_dir.mkdir(parents=True, exist\_ok=True)

frames = []

for csv\_file in raw\_dir.glob(" videos.csv"):

region = csv\_file.name[:2]

cat\_file = raw\_dir / f"{region}\_category\_id.json"

frames.append(standardize\_region(csv\_file, cat\_file, region))

out = pd.concat(frames, ignore\_index=True)

out.to\_parquet(processed\_dir/"trending\_clean.parquet", index=False)

return out

if \_\_name\_\_ == "\_\_main\_\_":

build\_processed(Path("data/raw"), Path("data/processed"))

Quality checks to run after cleaning

No nulls in `region`, `category` after fill

Monotonicity: `publish\_time` ≤ each `trending\_date` per `video\_id`

Unique keys: `video\_id` + `trending\_date` + `region`

**4) Sentiment Analysis (Titles & Tags)**

> Uses VADER for title (and tag) sentiment. Works best for English; we include a light language gate to avoid noisy scores.

python

# src/sentiment.py

import pandas as pd

from nltk.sentiment import SentimentIntensityAnalyzer

from langdetect import detect, LangDetectException

sia = SentimentIntensityAnalyzer()

KEEP\_LANGS = {"en"} # extend as needed

def \_safe\_lang(text: str) -> str:

try:

return detect(text) if text and text.strip() else ""

except LangDetectException:

return ""

def score\_text(text: str) -> dict:

lang = \_safe\_lang(text)

if lang in KEEP\_LANGS:

s = sia.polarity\_scores(text)

return {"neg": s["neg"], "neu": s["neu"], "pos": s["pos"], "compound": s["compound"], "lang": lang}

return {"neg": 0.0, "neu": 1.0, "pos": 0.0, "compound": 0.0, "lang": lang or "unknown"}

def add\_sentiment(df: pd.DataFrame) -> pd.DataFrame:

title\_scores = df["title"].fillna("").apply(score\_text)

tag\_text = df["tags"].fillna("").str.replace("|", " ")

tag\_scores = tag\_text.apply(score\_text)

for prefix, series in [("title", title\_scores), ("tags", tag\_scores)]:

df[[f"{prefix}\_neg", f"{prefix}\_neu", f"{prefix}\_pos", f"{prefix}\_compound", f"{prefix}\_lang"]] = pd.DataFrame(series.tolist(), index=df.index)

return df

Output Columns

`title\_neg/neu/pos/compound`, `title\_lang`

`tags\_neg/neu/pos/compound`, `tags\_lang`

**5) Feature Engineering (Trending Duration & Time Series)**

python

# src/features.py

import pandas as pd

# Per video\_id within region

def trending\_duration(df: pd.DataFrame) -> pd.DataFrame:

# Number of distinct trending days for a video per region

dur = (df.groupby(["region","video\_id"], as\_index=False)

.agg(trending\_days=("trending\_date", "nunique"),

first\_trending=("trending\_date", "min"),

last\_trending=("trending\_date", "max"),

category=("category", "first"),

title=("title", "first"),

channel\_title=("channel\_title", "first")))

dur["trending\_span\_days"] = (dur["last\_trending"] - dur["first\_trending"]).dt.days + 1

return dur

# Daily aggregates by region/category

def daily\_region\_category(df: pd.DataFrame) -> pd.DataFrame:

return (df.groupby(["trending\_date","region","category"], as\_index=False)

.agg(avg\_views=("views","mean"),

median\_views=("views","median"),

videos=("video\_id","nunique")))

**6) SQL Schema & Loads**

> Works with Postgres (recommended) or SQLite (simpler). Use Parquet→DB load via pandas.

sql

sql/schema.sql

CREATE TABLE IF NOT EXISTS trending (

video\_id TEXT,

title TEXT,

channel\_title TEXT,

category\_id INT,

category TEXT,

tags TEXT,

views BIGINT,

likes BIGINT,

dislikes BIGINT,

comment\_count BIGINT,

comments\_disabled BOOLEAN,

ratings\_disabled BOOLEAN,

video\_error\_or\_removed BOOLEAN,

description TEXT,

publish\_time TIMESTAMP,

trending\_date DATE,

region TEXT,

PRIMARY KEY (video\_id, trending\_date, region)

);

CREATE INDEX IF NOT EXISTS idx\_trending\_region\_date ON trending(region, trending\_date);

CREATE INDEX IF NOT EXISTS idx\_trending\_category ON trending(category);

sql

sql/analysis\_category\_rankings.sql

Rank categories by average views (per region and overall)

Per region

WITH cat\_stats AS (

SELECT region, category,

AVG(views)::BIGINT AS avg\_views,

PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY views) AS median\_views,

COUNT(DISTINCT video\_id) AS unique\_videos

FROM trending

GROUP BY region, category

)

SELECT region, category, avg\_views, median\_views, unique\_videos,

RANK() OVER (PARTITION BY region ORDER BY avg\_views DESC) AS rank\_in\_region

FROM cat\_stats

ORDER BY region, rank\_in\_region;

Overall

WITH cat\_stats AS (

SELECT category,

AVG(views)::BIGINT AS avg\_views,

COUNT(DISTINCT video\_id) AS unique\_videos

FROM trending

GROUP BY category

)

SELECT category, avg\_views, unique\_videos,

RANK() OVER (ORDER BY avg\_views DESC) AS rank\_overall

FROM cat\_stats

ORDER BY rank\_overall;

sql

sql/timeseries.sql

Daily time series for trending duration and volume

How long do videos trend? (per video\_id/region)

WITH spans AS (

SELECT region, video\_id, category,

MIN(trending\_date) AS first\_trending,

MAX(trending\_date) AS last\_trending,

COUNT(DISTINCT trending\_date) AS trending\_days

FROM trending

GROUP BY region, video\_id, category

)

SELECT region, category,

PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY trending\_days) AS median\_trending\_days,

AVG(trending\_days) AS avg\_trending\_days

FROM spans

GROUP BY region, category

ORDER BY region, avg\_trending\_days DESC;

Loading from Python

python

# notebooks/01\_clean\_standardize.ipynb (snippet)

import pandas as pd

from sqlalchemy import create\_engine

from src.clean import build\_processed

from src.sentiment import add\_sentiment

from src.features import trending\_duration, daily\_region\_category

clean = build\_processed(Path("data/raw"), Path("data/processed"))

clean = add\_sentiment(clean)

clean.to\_parquet("data/processed/trending\_with\_sentiment.parquet", index=False)

# Load to DB

engine = create\_engine("sqlite:///data/processed/trending.db") # swap for Postgres URI

clean.to\_sql("trending", engine, if\_exists="replace", index=False)

spans = trending\_duration(clean)

daily = daily\_region\_category(clean)

spans.to\_parquet("data/processed/spans.parquet", index=False)

daily.to\_parquet("data/processed/daily.parquet", index=False)

**7) Visualizations (Matplotlib/Seaborn)**

> Use these for EDA and to vet what you’ll push into Tableau.

python

# src/viz\_examples.py

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# 1) Most popular genres by region (avg views)

def bar\_avg\_views\_by\_category(df: pd.DataFrame, region: str):

sub = (df[df.region == region]

.groupby("category", as\_index=False)

.agg(avg\_views=("views","mean"))

.sort\_values("avg\_views", ascending=False)[:12])

plt.figure(figsize=(10,5))

sns.barplot(data=sub, x="avg\_views", y="category")

plt.title(f"Top Categories by Avg Views — {region}")

plt.xlabel("Average Views")

plt.ylabel("Category")

plt.tight\_layout()

# 2) Sentiment distribution (titles)

def sentiment\_violin(df: pd.DataFrame, region: str):

sub = df[df.region == region]

plt.figure(figsize=(8,5))

sns.violinplot(data=sub, x="category", y="title\_compound", inner="quartile")

plt.title(f"Title Sentiment (Compound) by Category — {region}")

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

# 3) Trending duration over time

def trending\_duration\_over\_time(daily: pd.DataFrame, region: str):

sub = daily[daily.region == region]

plt.figure(figsize=(10,4))

sns.lineplot(data=sub, x="trending\_date", y="median\_views", hue="category")

plt.title(f"Daily Median Views by Category — {region}")

plt.xlabel("Date"); plt.ylabel("Median Views")

plt.tight\_layout()

**8) Tableau — Datasource & Dashboard**

Data Sources:

`data/processed/trending\_with\_sentiment.parquet`

`data/processed/spans.parquet`

`data/processed/daily.parquet`

Recommended Data Model in Tableau:

Primary: `trending\_with\_sentiment` (record grain = region + video\\_id + trending\\_date)

Left‑join `spans` on (region, video\\_id) to add `trending\_days`, `first\_trending`, `last\_trending`

Calculated Fields:

Engagement Rate : `(ZN(likes) + ZN(comment\_count)) / NULLIF(views, 0)`

Sentiment Bucket :

IF [title\_compound] >= 0.2 THEN 'Positive'

ELSEIF [title\_compound] <= -0.2 THEN 'Negative'

ELSE 'Neutral' END

Sheets (suggested):

1. Top Genres by Avg Views (Region Selector) — bar chart

2. Sentiment vs Views — scatter; color by Sentiment Bucket

3. Trending Duration by Category — box plot using `trending\_days`

4. Daily Trending Volume — area chart of `videos` per day by category

5. Region‑wise Comparison — highlight table of `avg\_views` by category × region

Dashboard Layout:

Title: “YouTube Trending — Genres & Sentiment Across Regions”

Top‑left: Region filter (single value, dropdown)

Top‑right KPIs: Total Unique Videos, Median Trending Days, Avg Views (selected region)

Middle: Bar chart (Top Genres) + Sentiment scatter (linked by actions)

Bottom: Trending Duration box plot + Daily Volume area chart

Right pane: Highlight table for quick cross‑region scan

Interactivity:

Use Dashboard Actions : Selecting a category highlights corresponding points/lines across all charts.

Region filter applies to all sheets.

**9) Final Report — Data Story Outline**

1. Hook : How do tastes differ across regions? (Quick KPIs)

2. Genres : Which categories dominate and where (bar + highlight table)

3. Emotion in Titles : Do positive/negative titles correlate with higher views? (scatter + summary)

4. Staying Power : Which categories sustain trending longer? (box plot; `trending\_days`)

5. Seasonality/Events : Spikes in certain dates aligned with cultural events (time series)

6. Takeaways : Actionable insights for creators/brands per region

**10) Reproducible Environment**

`python>=3.10`

`pip install pandas pyarrow nltk langdetect seaborn matplotlib sqlalchemy`

Then in Python:

python

import nltk

nltk.download('vader\_lexicon')

DB: `sqlite` (bundled) or Postgres (`psycopg`)

**11) Validation & QC Checklist**

Duplicate key rate (`video\_id`+`trending\_date`+`region`) < 0.5%

Non‑null sentiment fields present; language gating behaves (inspect ` \_lang`)

Spot‑check category mapping for 5 random videos per region

Compare Matplotlib/Seaborn charts vs Tableau outputs for consistency

Save packaged workbook `.twbx` with embedded data for delivery

**12) Deliverables Mapping**

Dashboard : Tableau workbook `tableau/dashboard.twbx` (published or packaged)

Region‑wise visuals : Sheets 1 & 5 + PNG exports for report appendix

Final Report : `README.md` executive summary + narrative with key visuals

**13) Quickstart (TL;DR)**

1. Drop raw CSV/JSON files into `data/raw/`.

2. Run `python -m src.clean` → creates `trending\_clean.parquet`.

3. Run a small notebook or script to add sentiment & features → saves `trending\_with\_sentiment.parquet`, `spans.parquet`, `daily.parquet` and loads SQLite DB.

4. Connect Tableau to `data/processed/` Parquet files → build dashboard using specs above.

5. Export `.twbx` and write the short report using the Story Outline.

**14) Tailoring to IN/US/GB + Postgres**

Data Files to Place in `data/raw/`

`INvideos.csv` and `IN\_category\_id.json`

`USvideos.csv` and `US\_category\_id.json`

`GBvideos.csv` and `GB\_category\_id.json`

> You can add more countries later without changing code — they’ll be discovered automatically.

`src/config.py` (region filter + DB URI)

python

from pathlib import Path

# Only process these regions for now

REGIONS = {"IN", "US", "GB"}

# Paths

RAW\_DIR = Path("data/raw")

PROCESSED\_DIR = Path("data/processed")

# Database — Postgres

# Format: postgresql+psycopg://user:password@host:port/dbname

DB\_URI = "postgresql+psycopg://yt\_user:yt\_pass@localhost:5432/youtube\_trending"

# Tableau export helper

EXPORT\_PNG\_DIR = PROCESSED\_DIR / "exports"

EXPORT\_PNG\_DIR.mkdir(parents=True, exist\_ok=True)

#Region‑scoped Build Script (Python)

python

# scripts/build\_in\_us\_gb.py

import pandas as pd

from pathlib import Path

from sqlalchemy import create\_engine

from src.config import RAW\_DIR, PROCESSED\_DIR, DB\_URI, REGIONS

from src.clean import standardize\_region

from src.sentiment import add\_sentiment

from src.features import trending\_duration, daily\_region\_category

frames = []

for csv\_file in RAW\_DIR.glob(" videos.csv"):

region = csv\_file.name[:2].upper()

if region not in REGIONS:

continue

cat\_file = RAW\_DIR / f"{region}\_category\_id.json"

frames.append(standardize\_region(csv\_file, cat\_file, region))

clean = pd.concat(frames, ignore\_index=True)

clean = add\_sentiment(clean)

PROCESSED\_DIR.mkdir(parents=True, exist\_ok=True)

clean.to\_parquet(PROCESSED\_DIR/"trending\_with\_sentiment.parquet", index=False)

spans = trending\_duration(clean)

daily = daily\_region\_category(clean)

spans.to\_parquet(PROCESSED\_DIR/"spans.parquet", index=False)

daily.to\_parquet(PROCESSED\_DIR/"daily.parquet", index=False)

# Load to Postgres

engine = create\_engine(DB\_URI)

clean.to\_sql("trending", engine, if\_exists="replace", index=False)

print("Loaded table: trending (IN, US, GB)")

# Postgres DDL + Indexes

sql

-- Run once in your Postgres database

CREATE SCHEMA IF NOT EXISTS yt;

SET search\_path TO yt, public;

CREATE TABLE IF NOT EXISTS yt.trending (

video\_id TEXT,

title TEXT,

channel\_title TEXT,

category\_id INT,

category TEXT,

tags TEXT,

views BIGINT,

likes BIGINT,

dislikes BIGINT,

comment\_count BIGINT,

comments\_disabled BOOLEAN,

ratings\_disabled BOOLEAN,

video\_error\_or\_removed BOOLEAN,

description TEXT,

publish\_time TIMESTAMP,

trending\_date DATE,

region TEXT,

PRIMARY KEY (video\_id, trending\_date, region)

);

CREATE INDEX IF NOT EXISTS idx\_trending\_region\_date ON yt.trending(region, trending\_date);

CREATE INDEX IF NOT EXISTS idx\_trending\_category ON yt.trending(category);

CREATE INDEX IF NOT EXISTS idx\_trending\_views ON yt.trending(views DESC);

```

### Category Rankings Focused on IN/US/GB

```sql

-- Rank categories by average views per region (restricted to IN, US, GB)

WITH base AS (

SELECT FROM yt.trending WHERE region IN ('IN','US','GB')

), cat\_stats AS (

SELECT region, category,

AVG(views)::BIGINT AS avg\_views,

PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY views) AS median\_views,

COUNT(DISTINCT video\_id) AS unique\_videos

FROM base

GROUP BY region, category

)

SELECT region, category, avg\_views, median\_views, unique\_videos,

RANK() OVER (PARTITION BY region ORDER BY avg\_views DESC) AS rank\_in\_region

FROM cat\_stats

ORDER BY region, rank\_in\_region;

### Cross‑Region Comparison (Highlight Table Feeder)

sql

-- pivot‑ish view: average views per category × region for IN/US/GB

WITH base AS (

SELECT FROM yt.trending WHERE region IN ('IN','US','GB')

), cat\_region AS (

SELECT category, region, AVG(views) AS avg\_views

FROM base

GROUP BY category, region

)

SELECT category,

MAX(CASE WHEN region='IN' THEN avg\_views END) AS avg\_IN,

MAX(CASE WHEN region='US' THEN avg\_views END) AS avg\_US,

MAX(CASE WHEN region='GB' THEN avg\_views END) AS avg\_GB

FROM cat\_region

GROUP BY category

ORDER BY COALESCE(avg\_IN,0)+COALESCE(avg\_US,0)+COALESCE(avg\_GB,0) DESC;

### Time‑Series: Median Views by Category (Region Filterable)

sql

-- Daily median views so Tableau can line/area this by region & category

SELECT trending\_date, region, category,

PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY views) AS median\_views,

COUNT(DISTINCT video\_id) AS videos

FROM yt.trending

WHERE region IN ('IN','US','GB')

GROUP BY trending\_date, region, category

ORDER BY trending\_date, region, category

### Tableau Setup Tweaks for IN/US/GB

Add a Region parameter with allowed values `IN`, `US`, `GB` (or a multi‑select filter).

Default sort categories by Avg Views (selected region) .

In the Region‑wise Comparison highlight table, use a diverging palette centered on the median of all three regions.

Add a dashboard toggle (parameter) to switch Sentiment Bucket thresholds (±0.2 default, ±0.35 stricter) and recompute bucket via a calculated field.

### Sanity Checks Specific to These Regions

Cricket spikes (IN) around IPL windows; verify peaks align with expected dates.

Music & Late‑night (US) should dominate consistent high views; check outliers for removals.

Entertainment/News (GB) mix; validate category mapping shows BBC/ITV channels correctly.

### Makefile Convenience (optional)

```make

.PHONY: build pgload

build:

python scripts/build\_in\_us\_gb.py

pgload:

psql "postgresql://yt\_user:yt\_pass@localhost:5432/youtube\_trending" \

-f sql/schema.sql