"""

TrendWatch: Social Media Trends Analyzer

Quick-start

-----------

1) Create a virtual environment (optional but recommended)

python -m venv .venv && source .venv/bin/activate # on Windows: .venv\Scripts\activate

2) Install dependencies

pip install -r requirements.txt

3) Run the app

streamlit run trendwatch\_app.py

Suggested requirements.txt

--------------------------

streamlit

pandas

numpy

scikit-learn

nltk

altair

snscrape

python-dateutil

langdetect

emoji

"""

import os

import io

import re

import json

import math

import time

import emoji

import string

import typing as t

from datetime import datetime, timedelta, timezone

from dataclasses import dataclass

import pandas as pd

import numpy as np

from dateutil import parser as dateparser

from langdetect import detect, DetectorFactory

DetectorFactory.seed = 0 # make langdetect deterministic

import streamlit as st

import altair as alt

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.decomposition import NMF, LatentDirichletAllocation

from sklearn.preprocessing import MinMaxScaler

from sklearn.pipeline import make\_pipeline

from sklearn.base import TransformerMixin, BaseEstimator

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

# Ensure VADER lexicon is available

try:

nltk.data.find('sentiment/vader\_lexicon.zip')

except LookupError:

nltk.download('vader\_lexicon')

# ---- Helpers & Preprocessing ----

URL\_RE = re.compile(r"https?://\S+|www\.\S+")

MENTION\_RE = re.compile(r"@[A-Za-z0-9\_]+")

HASHTAG\_RE = re.compile(r"#[A-Za-z0-9\_]+")

WHITESPACE\_RE = re.compile(r"\s+")

EMOJI\_RE = emoji.get\_emoji\_regexp()

STOP\_EXTRA = set("""

rt amp via https http www com org co t co

""".split())

@dataclass

class ColumnMap:

text: str

timestamp: str

author: t.Optional[str] = None

like\_count: t.Optional[str] = None

reshare\_count: t.Optional[str] = None

reply\_count: t.Optional[str] = None

url: t.Optional[str] = None

platform: t.Optional[str] = None

# Basic cleaning

def clean\_text(s: str) -> str:

if not isinstance(s, str):

return ""

s = URL\_RE.sub(" ", s)

s = MENTION\_RE.sub(" ", s)

s = s.replace("&amp;", " and ")

s = EMOJI\_RE.sub(" ", s)

s = s.translate(str.maketrans('', '', string.punctuation))

s = HASHTAG\_RE.sub(lambda m: m.group(0).replace('#', ' '), s) # keep keyword, drop '#'

s = WHITESPACE\_RE.sub(" ", s)

return s.strip().lower()

class TextCleaner(BaseEstimator, TransformerMixin):

def \_\_init\_\_(self):

pass

def fit(self, X, y=None):

return self

def transform(self, X):

return [clean\_text(x) for x in X]

# ---- Data Loaders ----

def load\_csv(file: io.BytesIO) -> pd.DataFrame:

df = pd.read\_csv(file)

return df

# Optional: Fetch from X (Twitter) using snscrape (no API key required)

def fetch\_from\_x(query: str, limit: int = 500, since: t.Optional[str] = None, until: t.Optional[str] = None) -> pd.DataFrame:

try:

from snscrape.modules.twitter import TwitterSearchScraper

except Exception as e:

st.warning("snscrape is not installed. Run: pip install snscrape")

return pd.DataFrame()

q = query

if since:

q += f" since:{since}"

if until:

q += f" until:{until}"

rows = []

for i, tweet in enumerate(TwitterSearchScraper(q).get\_items()):

if i >= limit:

break

rows.append({

"text": tweet.rawContent,

"timestamp": tweet.date,

"author": getattr(tweet.user, 'username', None),

"like\_count": getattr(tweet, 'likeCount', None),

"reshare\_count": getattr(tweet, 'retweetCount', None),

"reply\_count": getattr(tweet, 'replyCount', None),

"url": f"https://x.com/{getattr(tweet.user, 'username', '')}/status/{tweet.id}",

"platform": "x"

})

return pd.DataFrame(rows)

# ---- Language filter ----

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

try:

return detect(text)

except Exception:

return "unknown"

# ---- Sentiment ----

SIA = SentimentIntensityAnalyzer()

def score\_sentiment(texts: t.List[str]) -> pd.Series:

return pd.Series([SIA.polarity\_scores(t).get('compound', 0.0) for t in texts])

# ---- Keyword & Topics ----

def top\_tfidf\_terms(corpus: t.List[str], n\_terms: int = 25, ngram\_range=(1,2), min\_df=5) -> pd.DataFrame:

vect = TfidfVectorizer(ngram\_range=ngram\_range, min\_df=min\_df, max\_features=5000)

X = vect.fit\_transform(corpus)

scores = np.asarray(X.sum(axis=0)).ravel()

terms = np.array(vect.get\_feature\_names\_out())

order = np.argsort(scores)[::-1][:n\_terms]

return pd.DataFrame({"term": terms[order], "score": scores[order]})

def topic\_model\_nmf(corpus: t.List[str], n\_topics: int = 6, ngram\_range=(1,2), min\_df=5, max\_features=10000, topn=10):

vect = TfidfVectorizer(ngram\_range=ngram\_range, min\_df=min\_df, max\_features=max\_features)

X = vect.fit\_transform(corpus)

nmf = NMF(n\_components=n\_topics, random\_state=42)

W = nmf.fit\_transform(X)

H = nmf.components\_

terms = np.array(vect.get\_feature\_names\_out())

topics = []

for k in range(n\_topics):

top\_idx = np.argsort(H[k])[::-1][:topn]

topics.append({

"topic": k,

"keywords": ", ".join(terms[top\_idx])

})

doc\_topics = W.argmax(axis=1)

return pd.DataFrame(topics), pd.Series(doc\_topics)

# ---- Trend Metrics ----

def resample\_trends(df: pd.DataFrame, time\_col: str, value\_col: str = None, rule: str = 'D') -> pd.DataFrame:

g = df.set\_index(time\_col).sort\_index()

if value\_col and value\_col in df.columns:

s = g[value\_col].resample(rule).sum()

else:

s = g[time\_col].resample(rule).count()

return s.reset\_index(name='count')

# ---- Streamlit UI ----

st.set\_page\_config(page\_title="TrendWatch: Social Media Trends Analyzer", layout="wide")

st.title("📈 TrendWatch: Social Media Trends Analyzer")

with st.sidebar:

st.header("Data Source")

source = st.radio("Choose input:", ["Upload CSV", "Fetch from X (Twitter)"])

st.markdown("---")

st.header("Filters")

lang\_keep = st.text\_input("Keep language (e.g., en, hi). Leave blank for all.", value="en")

since = st.date\_input("Since", value=datetime.now().date() - timedelta(days=7))

until = st.date\_input("Until", value=datetime.now().date())

st.caption("Dates are inclusive. Timezone is your local environment.")

st.markdown("---")

st.header("Keyword & Topic Settings")

n\_terms = st.slider("Top keywords", 5, 50, 20)

n\_topics = st.slider("Number of topics", 2, 12, 6)

min\_df = st.slider("Min doc frequency", 1, 20, 3)

st.markdown("---")

st.header("Visual Tweaks")

rule = st.selectbox("Time bucket", ["H", "D", "W"], index=1)

# Load data

raw\_df = pd.DataFrame()

colmap = ColumnMap(text="text", timestamp="timestamp", author="author", like\_count="like\_count", reshare\_count="reshare\_count", reply\_count="reply\_count", url="url", platform="platform")

if source == "Upload CSV":

st.subheader("Upload CSV with at least: text, timestamp")

upl = st.file\_uploader("CSV file", type=["csv"])

if upl:

try:

raw\_df = load\_csv(upl)

except Exception as e:

st.error(f"Failed to read CSV: {e}")

else:

st.subheader("Fetch recent posts from X (Twitter)")

query = st.text\_input("Search query", value="(AI OR artificial intelligence) -filter:replies")

limit = st.slider("Max posts", 100, 3000, 500, step=100)

if st.button("Fetch"):

with st.spinner("Fetching posts..."):

raw\_df = fetch\_from\_x(query, limit=limit, since=str(since), until=str(until))

if raw\_df.empty:

st.warning("No data fetched. Check query, dates, or install snscrape.")

if raw\_df.empty:

st.info("Load data to see analyses. Sample CSV columns: text,timestamp,author,like\_count,reshare\_count,reply\_count,url,platform")

st.stop()

# Normalize columns

if colmap.timestamp in raw\_df.columns:

raw\_df[colmap.timestamp] = pd.to\_datetime(raw\_df[colmap.timestamp], errors='coerce')

else:

st.error("Missing required column 'timestamp'.")

st.stop()

if colmap.text not in raw\_df.columns:

st.error("Missing required column 'text'.")

st.stop()

# Drop missing timestamps / text

raw\_df = raw\_df.dropna(subset=[colmap.text, colmap.timestamp])

# Language filter & cleaning

st.subheader("Preprocessing")

with st.spinner("Cleaning & detecting language..."):

raw\_df["clean\_text"] = raw\_df[colmap.text].map(clean\_text)

raw\_df["lang"] = raw\_df["clean\_text"].map(detect\_lang\_safe)

if lang\_keep:

keep\_mask = raw\_df["lang"].eq(lang\_keep)

st.caption(f"Keeping language: {lang\_keep}. Rows kept: {int(keep\_mask.sum())}/{len(raw\_df)}")

df = raw\_df.loc[keep\_mask].copy()

else:

df = raw\_df.copy()

if df.empty:

st.warning("No data after language filtering.")

st.stop()

# Sentiment

with st.spinner("Scoring sentiment..."):

df["sentiment"] = score\_sentiment(df["clean\_text"].tolist())

# Time range filter

mask\_time = (df[colmap.timestamp].dt.date >= since) & (df[colmap.timestamp].dt.date <= until)

df = df.loc[mask\_time]

if df.empty:

st.warning("No data in the selected date range.")

st.stop()

# KPIs

st.markdown("---")

kpi1, kpi2, kpi3, kpi4 = st.columns(4)

with kpi1:

st.metric("Posts", len(df))

with kpi2:

st.metric("Avg sentiment", f"{df['sentiment'].mean():.2f}")

with kpi3:

st.metric("Peak day", df.set\_index(colmap.timestamp).resample('D').size().idxmax().date().isoformat() if not df.empty else "-")

with kpi4:

st.metric("Unique authors", df[colmap.author].nunique() if colmap.author in df.columns else 0)

# Trends over time

st.subheader("Volume & Sentiment over time")

by\_time = df.set\_index(colmap.timestamp).sort\_index()

count\_ts = by\_time.resample(rule).size().rename("count").reset\_index()

mean\_sent = by\_time["sentiment"].resample(rule).mean().rename("avg\_sentiment").reset\_index()

chart1 = alt.Chart(count\_ts).mark\_line(point=True).encode(

x=alt.X(f"{colmap.timestamp}:T", title="Time"),

y=alt.Y("count:Q", title="Posts")

).properties(height=300)

chart2 = alt.Chart(mean\_sent).mark\_line(point=True).encode(

x=alt.X(f"{colmap.timestamp}:T", title="Time"),

y=alt.Y("avg\_sentiment:Q", title="Avg sentiment")

).properties(height=300)

st.altair\_chart(chart1, use\_container\_width=True)

st.altair\_chart(chart2, use\_container\_width=True)

# Top terms (TF-IDF)

st.subheader("Top Keywords & Hashtags")

ng1, ng2 = st.columns(2)

with ng1:

st.caption("TF-IDF keywords (1-2 grams)")

tfidf\_df = top\_tfidf\_terms(df["clean\_text"].tolist(), n\_terms=n\_terms, ngram\_range=(1,2), min\_df=min\_df)

st.dataframe(tfidf\_df, use\_container\_width=True)

with ng2:

if colmap.text in df.columns:

# Extract original hashtags for reference

all\_tags = re.findall(r"#[A-Za-z0-9\_]+", " ".join(df[colmap.text].astype(str).tolist()))

tags\_df = pd.Series([t.lower() for t in all\_tags]).value\_counts().head(n\_terms).reset\_index()

tags\_df.columns = ["hashtag", "count"]

st.caption("Most frequent hashtags (raw)")

st.dataframe(tags\_df, use\_container\_width=True)

# Topic modeling

st.subheader("Topic Modeling (NMF)")

with st.spinner("Discovering topics..."):

topics\_df, doc\_topics = topic\_model\_nmf(df["clean\_text"].tolist(), n\_topics=n\_topics, ngram\_range=(1,2), min\_df=min\_df)

df["topic"] = doc\_topics.values

c1, c2 = st.columns([1,2])

with c1:

st.dataframe(topics\_df, use\_container\_width=True)

with c2:

topic\_counts = df.groupby("topic").size().reset\_index(name="count")

topic\_chart = alt.Chart(topic\_counts).mark\_bar().encode(

x=alt.X("topic:N", title="Topic"),

y=alt.Y("count:Q", title="Posts")

).properties(height=300)

st.altair\_chart(topic\_chart, use\_container\_width=True)

# Topic trends

st.subheader("Topic trends over time")

topic\_time = df.set\_index(colmap.timestamp).groupby([pd.Grouper(freq=rule), "topic"]).size().reset\_index(name="count")

trend\_chart = alt.Chart(topic\_time).mark\_line(point=True).encode(

x=alt.X(f"{colmap.timestamp}:T", title="Time"),

y=alt.Y("count:Q", title="Posts"),

color="topic:N"

).properties(height=360)

st.altair\_chart(trend\_chart, use\_container\_width=True)

# Sentiment distribution

st.subheader("Sentiment distribution")

hist = alt.Chart(df).mark\_bar().encode(

x=alt.X("sentiment:Q", bin=alt.Bin(maxbins=30), title="VADER compound"),

y=alt.Y('count()', title='Posts')

).properties(height=300)

st.altair\_chart(hist, use\_container\_width=True)

# Export

st.markdown("---")

st.subheader("Export results")

@st.cache\_data

def make\_export(df: pd.DataFrame, topics\_df: pd.DataFrame):

out = {

"dataset": df.to\_dict(orient="records"),

"topics": topics\_df.to\_dict(orient="records"),

}

return json.dumps(out, ensure\_ascii=False, indent=2)

json\_blob = make\_export(df, topics\_df)

st.download\_button("Download JSON report", data=json\_blob, file\_name="trendwatch\_report.json", mime="application/json")

st.caption("Tip: Use the CSV uploader for Instagram/YouTube/TikTok by exporting data with columns 'text' and 'timestamp'. Add like/share columns for richer analysis.")