import streamlit as st

import yfinance as yf

import requests

from bs4 import BeautifulSoup

import pandas as pd

import plotly.express as px

from transformers import pipeline

from datetime import datetime, timedelta

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import re

import os

# --- Global Constants & Setup ---

NEWS\_API\_KEY = os.environ.get("NEWS\_API\_KEY") # Get your API key from newsapi.org and store it as an environment variable

FINANCIAL\_MODELING\_PREP\_API\_KEY = os.environ.get("FMP\_API\_KEY") # Get your API key from financialmodelingprep.com and store it

if not NEWS\_API\_KEY or not FINANCIAL\_MODELING\_PREP\_API\_KEY:

st.error("API keys for NewsAPI and FinancialModelingPrep are required. Please set them as environment variables (NEWS\_API\_KEY and FMP\_API\_KEY).")

st.stop()

# --- Helper Functions ---

def get\_stock\_data(ticker):

try:

ticker\_object = yf.Ticker(ticker)

info = ticker\_object.info

#Basic sanity check. Avoid Index Funds, etc.

if info.get('quoteType') != 'EQUITY':

st.error(f"{ticker} does not appear to be a valid stock ticker. Please enter a valid stock ticker for a company (e.g., AAPL, MSFT).")

st.stop()

return info

except Exception as e:

st.error(f"Error fetching data for {ticker}: {e}")

return None

@st.cache\_data(ttl=3600, show\_spinner="Fetching financial statements...") # Cache for 1 hour

def get\_financial\_statements(ticker, statement\_type, period='annual'):

"""Fetches income statement, balance sheet, or cash flow statement from FMP."""

base\_url = "https://financialmodelingprep.com/api/v3"

if statement\_type == "income-statement":

url = f"{base\_url}/income-statement/{ticker}?period={period}&apikey={FINANCIAL\_MODELING\_PREP\_API\_KEY}"

elif statement\_type == "balance-sheet-statement":

url = f"{base\_url}/balance-sheet-statement/{ticker}?period={period}&apikey={FINANCIAL\_MODELING\_PREP\_API\_KEY}"

elif statement\_type == "cash-flow-statement":

url = f"{base\_url}/cash-flow-statement/{ticker}?period={period}&apikey={FINANCIAL\_MODELING\_PREP\_API\_KEY}"

else:

return None

try:

response = requests.get(url)

response.raise\_for\_status() # Raise HTTPError for bad requests (4xx or 5xx)

return response.json()

except requests.exceptions.RequestException as e:

st.error(f"Error fetching {statement\_type}: {e}")

return None

@st.cache\_data(ttl=3600, show\_spinner="Fetching key metrics...") # Cache data

def get\_key\_metrics(ticker, period='annual'):

"""Fetches key metrics from FMP."""

url = f"https://financialmodelingprep.com/api/v3/key-metrics/{ticker}?period={period}&apikey={FINANCIAL\_MODELING\_PREP\_API\_KEY}"

try:

response = requests.get(url)

response.raise\_for\_status()

return response.json()

except requests.exceptions.RequestException as e:

st.error(f"Error fetching key metrics: {e}")

return None

@st.cache\_data(ttl=3600, show\_spinner="Fetching financial ratios...") # Cache data

def get\_financial\_ratios(ticker, period='annual'):

"""Fetches financial ratios from FMP."""

url = f"https://financialmodelingprep.com/api/v3/ratios/{ticker}?period={period}&apikey={FINANCIAL\_MODELING\_PREP\_API\_KEY}"

try:

response = requests.get(url)

response.raise\_for\_status()

return response.json()

except requests.exceptions.RequestException as e:

st.error(f"Error fetching financial ratios: {e}")

return None

def calculate\_financial\_ratios(info, income\_statement, balance\_sheet, cash\_flow\_statement):

try:

if not income\_statement or not balance\_sheet or not cash\_flow\_statement:

return {}

# Use the first element (most recent period) from the lists, if they exist.

income\_data = income\_statement[0] if income\_statement else {}

balance\_data = balance\_sheet[0] if balance\_sheet else {}

cash\_flow\_data = cash\_flow\_statement[0] if cash\_flow\_statement else {}

ratios = {}

# Profitability Ratios

ratios['Gross Profit Margin'] = (income\_data.get('grossProfit', 0) / income\_data.get('revenue', 1)) \* 100 if income\_data.get('revenue') else None # Avoid ZeroDivisionError

ratios['Operating Profit Margin'] = (income\_data.get('operatingIncome', 0) / income\_data.get('revenue', 1)) \* 100 if income\_data.get('revenue') else None

ratios['Net Profit Margin'] = (income\_data.get('netIncome', 0) / income\_data.get('revenue', 1)) \* 100 if income\_data.get('revenue') else None

# Liquidity Ratios

ratios['Current Ratio'] = balance\_data.get('totalCurrentAssets', 0) / balance\_data.get('totalCurrentLiabilities', 1) if balance\_data.get('totalCurrentLiabilities') else None

ratios['Quick Ratio'] = (balance\_data.get('cashAndCashEquivalents', 0) + balance\_data.get('shortTermInvestments', 0) + balance\_data.get('netReceivables',0)) / balance\_data.get('totalCurrentLiabilities', 1) if balance\_data.get('totalCurrentLiabilities') else None

# Solvency Ratios

ratios['Debt-to-Equity Ratio'] = balance\_data.get('totalLiabilities', 0) / balance\_data.get('totalStockholdersEquity', 1) if balance\_data.get('totalStockholdersEquity') else None

ratios['Debt-to-Asset Ratio'] = balance\_data.get('totalLiabilities', 0) / balance\_data.get('totalAssets', 1) if balance\_data.get('totalAssets') else None

# Efficiency Ratios

if balance\_data.get('totalAssets') and balance\_data.get('totalAssets')!=0:

ratios['Asset Turnover Ratio'] = income\_data.get('revenue', 0) / ((balance\_data.get('totalAssets', 0) + balance\_data.get('totalAssets', 0)) / 2) # Using average assets as a simplified approach

else:

ratios['Asset Turnover Ratio'] = None

return ratios

except Exception as e:

st.error(f"Error in calculating ratios: {e}")

return {}

@st.cache\_data(ttl=600, show\_spinner="Fetching news articles...") # Cache for 10 minutes

def get\_news(query, from\_date, to\_date):

all\_articles = []

page = 1

while True:

try:

url = (f"https://newsapi.org/v2/everything?q={query}&from={from\_date}&to={to\_date}&sortBy=publishedAt&apiKey={NEWS\_API\_KEY}&pageSize=100&page={page}") #Added page size for the max

response = requests.get(url)

response.raise\_for\_status()

data = response.json()

if data['status'] == 'ok':

articles = data.get('articles', [])

if not articles:

break # No more articles

all\_articles.extend(articles)

page += 1 # Increment to the next page

else:

st.error(f"News API Error: {data.get('message', 'Unknown error')}")

break

except requests.exceptions.RequestException as e:

st.error(f"Error fetching news: {e}")

break

return all\_articles

def summarize\_earnings\_call(transcript\_url):

# 1. Fetch the transcript (using a simplified method here - ideally use a robust library)

try:

response = requests.get(transcript\_url)

response.raise\_for\_status()

soup = BeautifulSoup(response.text, 'html.parser')

# This is a very basic extraction, assuming the transcript is within <p> tags.

# This needs to be \*much\* more robust for real-world use, handling different website structures.

transcript = ' '.join([p.text for p in soup.find\_all('p')])

# Regex to remove timestamps and speaker labels (VERY basic, needs improvement)

transcript = re.sub(r'\(\d+:\d+\)', '', transcript) # Remove timestamps

transcript = re.sub(r'^[A-Za-z\s]+:', '', transcript, flags=re.MULTILINE) # Remove speaker labels

except requests.exceptions.RequestException as e:

return f"Error fetching transcript: {e}"

# 2. Summarize using Hugging Face Transformers

if transcript:

try:

summarizer = pipeline("summarization") # Or specify a model like "bart-large-cnn"

summary = summarizer(transcript, max\_length=250, min\_length=50, do\_sample=False)[0]['summary\_text']

return summary

except Exception as e:

return f"Error during summarization: {e}. Ensure transformers and a summarization model are installed."

else:

return "Could not retrieve transcript."

def perform\_sentiment\_analysis(articles):

if not articles:

return "No articles to analyze."

try:

nltk.data.find('vader\_lexicon')

except LookupError:

nltk.download('vader\_lexicon', quiet=True)

analyzer = SentimentIntensityAnalyzer()

sentiments = []

for article in articles:

title = article.get('title', '')

description = article.get('description', '')

text = f"{title}. {description}" # Combine for analysis

scores = analyzer.polarity\_scores(text)

sentiments.append(scores)

# Aggregate sentiments (simple average - could be weighted by recency, source credibility, etc.)

avg\_sentiment = {

'positive': sum(s['pos'] for s in sentiments) / len(sentiments),

'negative': sum(s['neg'] for s in sentiments) / len(sentiments),

'neutral': sum(s['neu'] for s in sentiments) / len(sentiments),

'compound': sum(s['compound'] for s in sentiments) / len(sentiments),

}

return avg\_sentiment

# --- Streamlit App ---

st.title("Company Analysis Dashboard")

ticker = st.text\_input("Enter Stock Ticker (e.g., AAPL, MSFT):", "AAPL").upper()

if not ticker:

st.warning("Please enter a stock ticker.")

st.stop()

stock\_info = get\_stock\_data(ticker)

if stock\_info:

company\_name = stock\_info.get('longName', ticker) # Fallback to ticker if longName is missing

st.header(f"Company: {company\_name}")

# --- Overview Section ---

with st.expander("Company Overview", expanded=True):

col1, col2, col3 = st.columns(3)

col1.metric("Sector", stock\_info.get('sector', 'N/A'))

col2.metric("Industry", stock\_info.get('industry', 'N/A'))

col3.metric("Market Cap", f"{stock\_info.get('marketCap', 'N/A'):,}" if stock\_info.get('marketCap') else 'N/A') # handle None

st.write(stock\_info.get('longBusinessSummary', 'No summary available.'))

# --- Financials Section ---

st.header("Financial Analysis")

# Fetch and display financial statements

income\_statement = get\_financial\_statements(ticker, "income-statement")

balance\_sheet = get\_financial\_statements(ticker, "balance-sheet-statement")

cash\_flow\_statement = get\_