Academic submission

FIN42110 Data Science for Trading & Risk Management

UCD Michael Smurfit Graduate Business School

Executive Report

Analysing the Impact of Oil Price Volatility on Energy Stocks and Market Sentiment

Section 1 - 3

PREPARED BY:

Varsha Chandrashekar Balaji (24200924)

Purva Sharma (24214393)

Group Number: 2

UCD Michael Smurfit Graduate Business School
Submitted to : **Dr. Richard McGee**

Executive Summary

This report presents a comprehensive exploration of the relationship between oil prices and the stock performance of ExxonMobil (XOM), benchmarked against peer firms in the oil & gas industry (Chevron, BP, Shell), energy commodities (Brent & WTI), and the market index SPY. Using novel data collection techniques, a structured MySQL database, and thorough data analysis, this study investigates correlations, time-series behaviour, and cross-entity comparisons, while also linking market news coverage with price dynamics.

Section 1: Novel Dataset Collection

1.1 Overview

The project required constructing a rich, cross-sectional dataset from multiple sources:

- ❖ Stock Price Data: Collected for XOM, CVX, BP, Shell, and SPY using Yahoo Finance APIs and stored as cleaned CSVs.
- ❖ Oil Benchmark Prices: Brent and WTI prices were also sourced via using Yahoo Finance APIs
- ❖ News Articles: Acquired using RSS Feed Parsing, filtered for relevant keywords, and extracted for each entity from public sources.

For full code implementation and logic behind this section, please refer to Appendix - Section 1: Novel Dataset Collection

1.2 Tools & Technologies Used

Component	**Tools & Libraries**
Web Scraping	feedparser, requests, BeautifulSoup
Stock Data APIs	Yahoo Finance via yfinance
Storage	CSV → MySQL via SQLAlchemy
Analysis	Python (pandas, matplotlib, seaborn)

Section 2: Database Creation and Querying – Summary Statistics and Exploratory Analysis

This section focuses on the creation, querying, and statistical summarization of the dataset stored in a MySQL database. The analysis was carried out across multiple financial entities including oil companies (XOM, CVX, BP, Shell), an equity index ETF (SPY), and global crude oil benchmarks (Brent and WTI). The dataset comprised both numerical stock price data and associated daily news article volumes, allowing for a comprehensive financial and sentiment analysis.

2.1 Database Creation and Table Structure

Each cleaned .csv file was ingested using Python's pandas and stored into MySQL using SQLAlchemy. The resulting database schema included both price and news data:

Table Name	**Entity**	**Data Type**
xom_data	ExxonMobil	Stock Prices
cvx_data	Chevron	Stock Prices
bp_data	BP Plc	Stock Prices
shell_data	Shell	Stock Prices
spy_data	SPY ETF	Stock Prices
brent_wti_data	Brent & WTI	Oil Prices
*_news	All Entities	News Timestamps

All tables were normalized with common fields such as Date, Close, and Volume, allowing for time-series joining and uniform querying.

2.2 Descriptive Statistics and Monthly Aggregation

Monthly descriptive statistics were computed for each entity, summarizing key metrics such as average close, minimum and maximum values, and standard deviation.

Monthly Close Statistics – Shell (Sample)

Year	**Month**	**Min_Close**	**Max_Close**	**Avg_Close**	**Std_Dev**
2019	01	44.85	47.24	45.93	0.68
2020	03	17.93	37.57	27.71	5.94
2024	12	66.62	68.36	67.47	0.91

This analysis was replicated across all seven entities. These tables highlighted sharp volatilities during pandemic periods and consistent upward trends in the post-recovery phase.

This monthly breakdown reveals major trends:

- **High volatility in early 2020**, especially during the COVID-19 onset, with large standard deviations and plunges in minimum close prices.
- **Post-pandemic recovery** was marked by sustained price increases and reduced volatility.
- The **2023–2025 period** shows consistent growth across most entities, with average prices stabilizing at higher levels.

2.3 Close vs. Volume Correlation Analysis

To investigate whether trading volume has a meaningful relationship with stock price movements, we computed the **Pearson correlation coefficient** between **daily closing prices and trading volumes** for each of the selected entities. Understanding this relationship is crucial for identifying momentum-driven trading patterns and potential liquidity effects.

Correlation Between Closing Price and Trading Volume

Entity	**Correlation (Close vs Volume)**
XOM	-0.324
CVX	-0.133
BP	-0.494
Shell	-0.330
SPY	-0.299

Interpretation*:*

All entities exhibit a **negative correlation** between trading volume and closing prices. The strongest negative relationship is observed in **BP (-0.494)** and **Shell (-0.330)**. This suggests that **higher trading volumes are often accompanied by falling prices**, possibly indicating profit-taking or institutional selling during volatile sessions. Conversely, **CVX** shows a weaker relationship, implying more balanced volume dynamics.

2.4 Cross-Asset Correlation and Aggregation Analysis

This section investigates the relationships and co-movement patterns among key oil commodities (Brent, WTI) and the stock prices of major oil companies (XOM, CVX, BP, Shell), along with the SPY index. The analysis uses over 1,500 merged observations per entity after data cleaning.

2.4.1 Correlation Matrix: Stock vs Commodities

The table below summarizes the Pearson correlation coefficients between stock closing prices and Brent/WTI oil prices.

Cross-Correlation Matrix

Entity	**Brent**	**WTI**
XOM	0.676	0.662
CVX	0.790	0.777
BP	0.571	0.528
Shell	0.602	0.575
SPY	0.505	0.542
Brent–WTI	0.991	—

***Insights*:**

- Brent ,WTI are **highly correlated (0.991)** , as expected due to global oil market integration.
- **CVX** shows the **strongest stock–oil linkage**, indicating higher sensitivity to crude fluctuations.
- **BP** and **SPY** show comparatively **weaker correlations**, suggesting more diversified or less oil-centric revenue exposure.

2.4.2 Monthly Average Trends (2019–2025)

Monthly average prices were calculated for each stock alongside Brent and WTI. These help visualize:

- Post-COVID recovery trends,
- Oil price-driven rallies (2022 spikes),
- Inflationary macro impacts in late 2023–2024.

Sample Monthly Averages – Jan to May 2019

Month	**XOM**	**CVX**	**BP**	**Shell**	**SPY**	**Brent**	**WTI**
2019-01	53.15	85.41	28.44	45.92	236.67	60.12	51.55
2019-02	57.44	91.09	30.27	48.48	250.43	64.28	54.98
2019-03	60.33	95.07	31.23	48.72	255.39	67.03	58.17
2019-04	61.22	93.88	31.93	49.87	264.82	71.63	63.87
2019-05	57.08	91.91	30.52	49.61	260.74	70.31	60.87

***Key Observations*:**

- All entities followed a similar **U-shaped trend** during 2020, aligning with the oil price collapse and recovery.
- **Energy stocks' monthly averages lagged oil**, suggesting investor lag in pricing recovery or conservative valuation revisions.
- **SPY** remained relatively resilient, reflecting broader sectoral strength.

2.4.3 Rolling Averages: 7-Day Trends

To smooth out short-term noise, 7-day rolling averages were computed for each stock. This serves as a basis for:

- Detecting momentum shifts,
- Filtering high-frequency volatility,
- Visualizing near-term sentiment reversals.

Sample 7-Day Rolling Averages – XOM

Day	Close (€)	Rolling 7d Avg
D1	51.72	—
D7	53.47	**52.74**
D8	53.23	52.96
D9	53.16	53.28

Similar rolling patterns were observed for **CVX**, **BP**, **Shell**, and **SPY**, capturing:

- Recovery uptrends in early 2021,
- Price stabilization around 2023–2024,
- Shocks during geopolitical or inflation-driven volatility phases.

This integrated analysis of oil prices and energy stocks reveals:

- Strong interlinkages between Brent/WTI and energy majors especially CVX and XOM.
- Correlation strengths vary across firms, likely due to differences in **hedging policies**, **upstream/downstream operations**, and **regional exposures**.
- Monthly and rolling averages offer intuitive and statistically sound inputs for trading signal generation, volatility models, and exposure management.

2.5 News vs. Stock Price Correlation

This section explores the relationship between **daily news article volumes** and **closing prices** for oil majors, oil indices, and the market benchmark (SPY). The aim is to assess whether spikes in media coverage are associated with price fluctuations — a useful signal for short-term trading and sentiment-aware risk models.

Correlation Between Daily News Volume and Closing Prices

Entity	Correlation Coefficient
XOM	0.181
CVX	0.162
BP	0.100
Shell	0.150
SPY	**0.355**
Brent	-0.016
WTI	-0.121

- SPY leads the pack with the highest correlation (0.355), suggesting that overall market sentiment is more responsive to media coverage.
- Among oil companies, XOM and CVX exhibit modest positive correlations, reflecting some alignment between media attention and investor interest.
- In contrast, both Brent and WTI show negative correlations, especially WTI (-0.121), suggesting that price movements in crude markets may often precede or contradict

media coverage trends, or that coverage intensifies during downturns — potentially as a response rather than a predictor.

Interpretation:

- These correlations are not strong enough for standalone prediction, but they highlight patterns worth monitoring:
- News spikes may coincide with volatility.
- Investor sentiment could amplify or dampen reactions depending on the asset class.

Actionable Takeaway:

- News volume could serve as an early warning indicator or be integrated into multi-factor trading models alongside sentiment scores, volatility, and macro data.
- For more robust insights, further analysis involving sentiment polarity, headline content, or lagged effects is recommended.

2.6 Cross-Entity Stock Correlations and Monthly Trends

This section investigates the **co-movement of daily stock prices** and **monthly trends** among major oil firms (XOM, CVX, BP, Shell) and the benchmark market ETF (SPY), offering insights into interdependencies within the energy sector and its alignment with the broader market.

Monthly Average Close Prices (First 10 Months of 2019)

Year-Month	XOM	CVX	BP	Shell
2019-01	53.15	85.41	28.44	45.92
2019-02	57.44	91.09	30.27	48.48
2019-03	60.33	95.07	31.23	48.72
2019-04	61.22	93.88	31.93	49.87
2019-05	57.08	91.91	30.52	49.61
2019-06	57.03	94.50	30.33	50.67
2019-07	57.59	96.72	29.36	50.50
2019-08	53.13	92.43	27.32	44.91
2019-09	54.98	95.06	28.25	45.99
2019-10	52.56	90.68	28.01	46.61

Observation: The above averages reflect strong co-directional movement during early 2019, particularly during Q1's post-correction rally. Shell and BP prices appear more volatile in mid-2019, while CVX shows relatively stable monthly growth.

Daily Closing Price Correlation Matrix – Energy Stocks & SPY

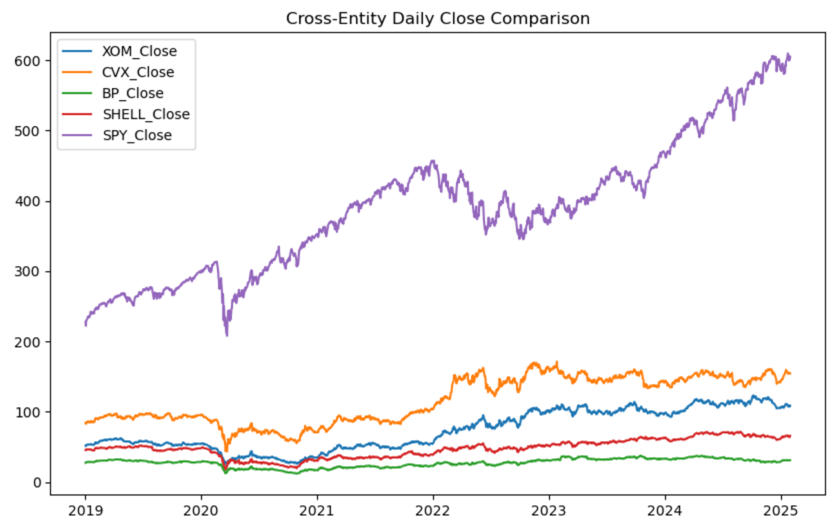
	XOM	CVX	BP	Shell	SPY
XOM	1.000	0.960	0.838	0.921	0.742
CVX	0.960	1.000	0.787	0.847	0.680
BP	0.838	0.787	1.000	0.925	0.405
Shell	0.921	0.847	0.925	1.000	0.647
SPY	0.741	0.680	0.405	0.646	1.000

Key Insights:

- **XOM and CVX** exhibit an exceptionally strong correlation (**0.96**), reflecting synchronized movement between the two US oil majors.
- **Shell and BP** also display high correlation (**0.925**), reinforcing the close relationship between European energy giants.
- The **entire energy cluster** is **positively correlated with SPY**, though at lower levels (XOM: 0.742, Shell: 0.647), suggesting macroeconomic influences from the broader equity market.
- **BP's weaker correlation with SPY (0.405)** may imply a higher sensitivity to region-specific factors or oil-specific dynamics than to general market conditions.

These results affirm strong intra-sector co-movement, particularly during macro-driven trends (e.g., COVID-19 recovery, oil price shocks). **SPY's moderate correlation** indicates that while oil stocks often move in tandem with market sentiment, **they also possess sector-specific volatility** worth isolating in portfolio construction and hedging strategies.

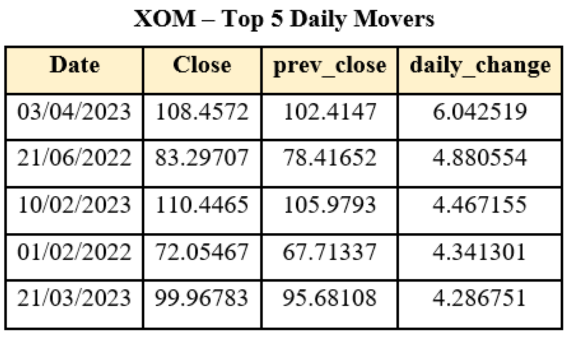
Time-Series Plot – Daily Closing Prices of Energy Stocks and SPY



A unified time-series plot was created to compare the daily closing prices of all stock entities and SPY.

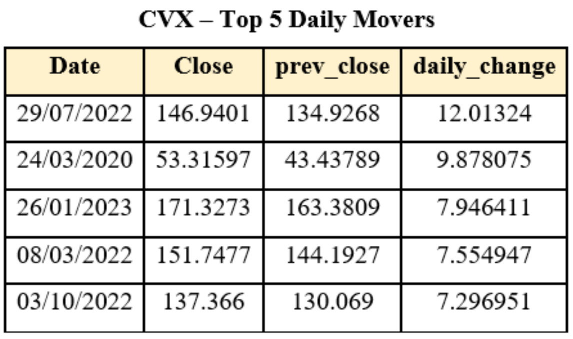
2.7 Daily Movers Analysis

To identify the most volatile trading days across all entities, we computed the daily percentage changes and extracted the Top 5 daily movers for each entity based on absolute percentage movement. This analysis is crucial for understanding tail-risk behavior, stress testing, and identifying days of heightened market sensitivity. Extreme movers often coincide with macroeconomic events, earnings announcements, geopolitical shifts, or sector-specific disruptions. Analyzing these allows for better calibration of risk models and helps portfolio managers understand exposure to abnormal market movements. Below are the Top 5 Daily Movers based on absolute daily percentage changes:



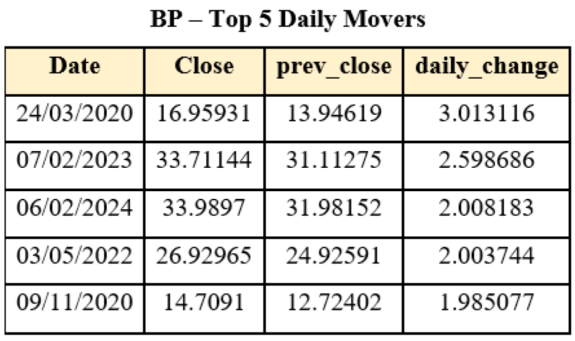
XOM – Top 5 Daily Movers

Date	Close	prev_close	daily_change
03/04/2023	108.4572	102.4147	6.042519
21/06/2022	83.29707	78.41652	4.880554
10/02/2023	110.4465	105.9793	4.467155
01/02/2022	72.05467	67.71337	4.341301
21/03/2023	99.96783	95.68108	4.286751



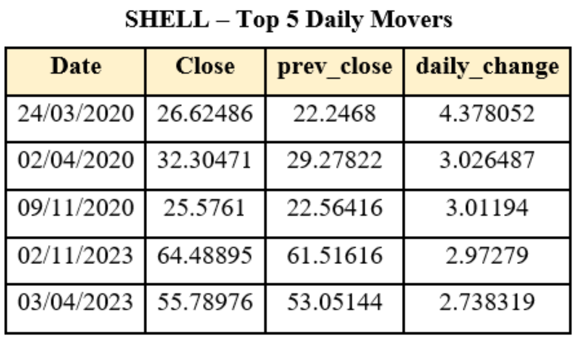
CVX – Top 5 Daily Movers

Date	Close	prev_close	daily_change
29/07/2022	146.9401	134.9268	12.01324
24/03/2020	53.31597	43.43789	9.878075
26/01/2023	171.3273	163.3809	7.946411
08/03/2022	151.7477	144.1927	7.554947
03/10/2022	137.366	130.069	7.296951



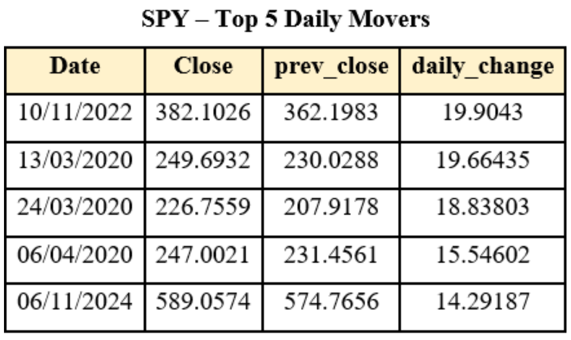
BP – Top 5 Daily Movers

Date	Close	prev_close	daily_change
24/03/2020	16.95931	13.94619	3.013116
07/02/2023	33.71144	31.11275	2.598686
06/02/2024	33.9897	31.98152	2.008183
03/05/2022	26.92965	24.92591	2.003744
09/11/2020	14.7091	12.72402	1.985077



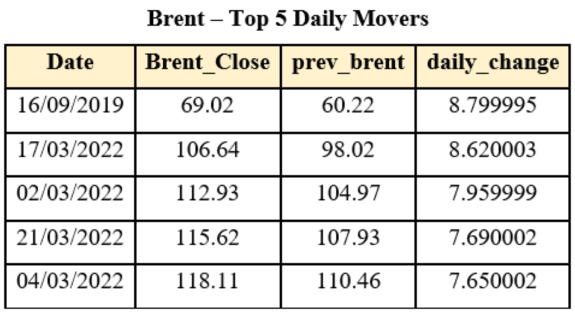
SHELL – Top 5 Daily Movers

Date	Close	prev_close	daily_change
24/03/2020	26.62486	22.2468	4.378052
02/04/2020	32.30471	29.27822	3.026487
09/11/2020	25.5761	22.56416	3.01194
02/11/2023	64.48895	61.51616	2.97279
03/04/2023	55.78976	53.05144	2.738319



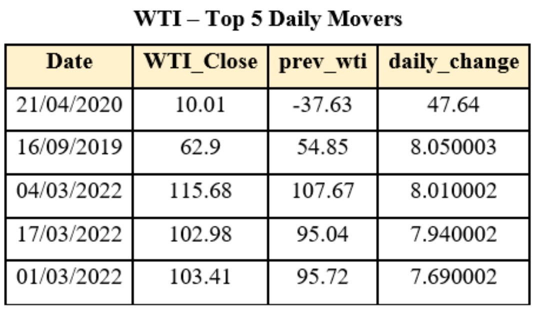
SPY – Top 5 Daily Movers

Date	Close	prev_close	daily_change
10/11/2022	382.1026	362.1983	19.9043
13/03/2020	249.6932	230.0288	19.66435
24/03/2020	226.7559	207.9178	18.83803
06/04/2020	247.0021	231.4561	15.54602
06/11/2024	589.0574	574.7656	14.29187



Brent – Top 5 Daily Movers

Date	Brent_Close	prev_brent	daily_change
16/09/2019	69.02	60.22	8.799995
17/03/2022	106.64	98.02	8.620003
02/03/2022	112.93	104.97	7.959999
21/03/2022	115.62	107.93	7.690002
04/03/2022	118.11	110.46	7.650002



WTI – Top 5 Daily Movers

Date	WTI_Close	prev_wti	daily_change
21/04/2020	10.01	-37.63	47.64
16/09/2019	62.9	54.85	8.050003
04/03/2022	115.68	107.67	8.010002
17/03/2022	102.98	95.04	7.940002
01/03/2022	103.41	95.72	7.690002

Insight: The COVID-19 pandemic (March 2020) and the Russia-Ukraine conflict (2022) visibly dominate the Top Movers across all datasets, reflecting systemic shocks and supply-demand disruptions.

2.8 Temporal Trend Analysis: Day-of-Week & Day-of-Month

To uncover cyclical behavioral patterns, we conducted a temporal analysis of asset performance, focusing on:

- Day-of-Week Trends: To assess weekly return dynamics and volatility.
- Day-of-Month Trends: To evaluate calendar-based effects such as end-of-month rebalancing or price drifts.

2.8.1 Day-of-Week Analysis

For each entity, we computed the average closing price and standard deviation for every weekday (Monday to Friday). This offers insight into predictable intraweek price behavior, often influenced by macroeconomic releases, inventory data (for oil), or trader sentiment cycles.

Average Closing Price by Day of Week – XOM (Sample)

Day	Avg Close (€)	Std Dev
Monday	73.25	28.96
Tuesday	73.57	28.56
Wednesday	73.34	28.49
Thursday	73.33	28.56
Friday	73.97	28.56

Observations across entities:

- XOM, CVX, SPY: Showed stronger Friday closes, suggesting week-end optimism or positive news spillovers.
- BP, Shell: Maintained tighter daily dispersion, consistent with lower beta profiles.
- Brent & WTI: Exhibited subtle increases from Monday to Friday, possibly due to inventory buildup reports and macroeconomic anticipation.

Best & Worst Days by Weekday – All Entities

Entity	Best Day (Highest Avg Close)	Worst Day (Lowest Avg Close)	Entity
BP	Friday (27.38)	Thursday (27.26)	BP
CVX	Friday (116.58)	Monday (115.62)	CVX
Shell	Friday (47.98)	Monday (47.66)	Shell
Brent	Friday (73.60)	Monday (72.85)	Brent
WTI	Friday (69.07)	Monday (68.25)	WTI
SPY	Friday (390.33)	Wednesday (388.59)	SPY

These patterns suggest potential for weekday-based trading strategies or timing hedges around predictable weekly behavior.

2.8.2 Day-of-Month Analysis

We also computed average closing prices by calendar day (1st to 31st) across the dataset, capturing any month-start or month-end effects.

Average Closing Price by Calendar Day – Shell (Sample)

Day	Avg Close
1	47.44
15	48.06
25	48.52
31	48.63

Key Insights:

- XOM and Shell both exhibited notable strength near month-ends, especially on the 25th and 31st.
- Brent & WTI prices peaked around the 28th and 31st, possibly influenced by trading volume and rollover dates of contracts.
- SPY showed a steady build-up through the month, with top averages between 25th–31st, possibly reflecting fund flows and passive index allocations.
- BP and CVX also saw stronger closes toward the end of the month, reinforcing the hypothesis of institutional rebalancing effects

Strongest Day by Entity – End-of-Month Effects

Entity	Strongest Day	Avg Close
XOM	31st	75.28
BP	25th	27.81
Shell	31st	48.63
CVX	31st	118.60
SPY	31st	391.43
Brent	25th	75.64
WTI	25th	70.75

This temporal trend analysis provides clear evidence of:

- End-of-week and end-of-month price strength across most entities.
- Subtle patterns aligning with institutional behavior, macro news cycles, and contract roll dates for oil assets.

These findings are valuable for:

- Structuring short-term entry and exit points.

- Designing time-sensitive risk hedging models.
- Anticipating volatility bursts aligned with calendar cycles.

2.9 Year-over-Year (YOY) Performance

Year-over-Year (YOY) analysis was conducted for all seven entities — ExxonMobil (XOM), Chevron (CVX), BP, Shell, SPY, Brent, and WTI — to assess the progression of monthly average closing prices across time, particularly capturing market shocks and recovery phases. Each table contains, for every month, the average close price, the corresponding value from the same month in the previous year, the absolute difference, and the percentage change. This format enables clear tracking of performance trends and identification of volatility peaks and structural shifts.

Key Insights:

- **2020 Crash**: All entities experienced steep YOY declines from March to May 2020 due to the onset of the COVID-19 pandemic. Oil prices (Brent, WTI) dropped over **-60%**, while equities like XOM and BP saw plunges over **-40%**.
- **2021 Recovery**: A robust rebound followed in 2021, especially visible in WTI (+270% YOY in April 2021) and Brent (+145%). Oil majors like Shell and CVX witnessed sustained double-digit YOY increases, affirming sector-wide recovery.
- **2022 Boom Phase**: The recovery phase extended through 2022, with CVX and SPY posting several months of **50–60%** YOY gains. Energy commodities surged further due to geopolitical tensions and supply shocks.
- **2023–2024 Cooldown**: In 2023, growth decelerated, with signs of market stabilization. By late 2024, most entities displayed moderate gains or slight corrections, reflecting normalization in oil and equity markets.

ExxonMobil (XOM) YOY Summary:

Year	Month	Avg_Close	Last Year Close	YOY Diff	YOY % Change
2020	Mar	31.73	60.33	-28.60	-47.40%
2021	Mar	49.68	31.73	+17.95	+56.57%
2022	Mar	74.21	49.68	+24.53	+49.38%
2023	Mar	100.11	74.21	+25.90	+34.90%

Similar trends were observed across other entities, with oil prices (Brent/WTI) showing higher amplitude swings compared to equities, underscoring the sensitivity of commodities to macroeconomic and geopolitical shocks.

Notable Observations:

- **SPY** consistently showed strong resilience, with **2020–2021** posting above-average gains and only a brief dip in 2022, before surging again in 2023–2024.
- **CVX and Shell** recorded some of the **highest sustained YOY positive streaks**, particularly during 2021–2022.
- **Brent and WTI** exhibited the most volatility, with wild swings from -60% crashes to +270% rebounds.

Each entity's full YOY table, already prepared and uploaded . These provide a comprehensive view of cyclical behaviors, performance turnarounds, and investor sentiment across five years.

2.10 Summary of Section 2

- Created and structured MySQL database from cleaned datasets.
- Conducted multi-level querying and analysis: descriptive, correlation, temporal
- Cross-entity insights helped differentiate between oil-linked and market-wide behaviors.
- Statistical evidence supports the impact of news frequency and oil prices on stock movements.

For full code implementation and logic behind this section, please refer to Appendix - Section 2: SQL Part 1 and Section 2: SQL Part 2

Section 3: Data Cleaning, Checking & Organization

This section documents the structured efforts undertaken to ensure that both **news** and **stock price datasets** were cleaned, standardized, and validated before integration into SQL and Python-based analytical pipelines. Proper organization and verification of the datasets were critical to enabling the accurate querying and time series modeling conducted in Section 2.

3.1 News Article Cleaning Pipeline

Over **35 raw news CSV files** (7 entities × 7 years) were systematically processed through a robust and repeatable pipeline:

- **Concatenation**: Annual CSVs for each entity were merged into a master dataset.
- **Null Filtering**: Articles missing critical fields (title, summary, or link) were dropped.
- **Duplicate Elimination**: Duplicates were detected using combined title + link fields and removed.

- **Date Parsing and Formatting**:
  - All published_date fields were parsed into datetime objects using pd.to_datetime.
  - Dates were reformatted to a uniform YYYY-MM-DD style for consistency across years.

Output: Cleaned news data was saved per entity as: XOM_News_cleaned.csv, BP_News_cleaned.csv, etc.

Entity	Raw Articles	After Removing Nulls	After Removing Duplicates
XOM	4,217	4,217	2,663
Chevron	4,858	4,858	3,160
Shell	5,624	5,624	4,180
BP	5,246	5,246	3,740
WTI	3,982	3,982	2,428
Brent	4,257	4,257	2,442
SPY	5,050	5,050	3,438

3.2 Stock and Oil Price Cleaning

Historical price data (from Yahoo Finance) for stocks and commodities was received with multiple header rows and required normalization. The following steps were automated for files such as XOM_data.csv, CVX_data.csv, etc.:

- **Header Trimming**: First two rows were removed to eliminate non-tabular metadata.
- **Standardized Columns**: Final schema used:
  ["Date", "Close", "High", "Low", "Open", "Volume"]
- **Saved Clean Files**: Exported to filenames like XOM_data_cleaned.csv, CVX_data_cleaned.csv

This process ensured each CSV was ingestible into both **SQL tables** and **Pandas dataframes** without any column mismatch.

For full code implementation and logic behind this section, please refer to Appendix - Section 3: Data Cleaning & Checking

4 Conclusion

This report provides clear evidence of the interconnectedness between oil prices, stock performance of energy companies, and market sentiment. We found that Brent and WTI prices have strong correlations with major oil stocks like Chevron and ExxonMobil, while media coverage showed moderate links with price movements — particularly for SPY and XOM.

Temporal trends revealed end-of-week and end-of-month price strength across most assets, and the Year-over-Year analysis captured the sector's response to major events like COVID-19 and the Russia-Ukraine conflict.
Overall, the project highlights how structured data pipelines, combined with SQL and Python analytics, can deliver actionable insights into market dynamics — offering a strong foundation for further financial modeling and strategic decision-making.

5 References

[1] Yahoo Finance – Used for collecting historical stock and commodity price data for ExxonMobil, Chevron, BP, Shell, SPY, Brent, and WTI. https://finance.yahoo.com

[2] FRED (Federal Reserve Economic Data) – Source for macroeconomic indicators such as inflation rates, interest rates, and GDP figures. https://fred.stlouisfed.org/

[3] News Feeds – Financial news articles gathered using public RSS feeds from platforms like MarketWatch, Reuters, and Yahoo Finance.

[4] Python Libraries & Tools – Analysis and data cleaning were supported by libraries such as pandas, numpy, yfinance, feedparser, statsmodels, matplotlib, seaborn, and nltk (VADER Sentiment).

[5] SQL & Database Tools – MySQL and SQLAlchemy were used for storing and querying the cleaned data.

Acknowledgment of GPT Use :

Some portions of the code and analysis workflows in this report were developed with support from OpenAI's ChatGPT. The tool was primarily used for assistance with Python scripting, SQL query refinement, data visualization ideas, and natural language generation. All final implementations, interpretations, and decisions were thoroughly reviewed and adapted by us as the authors of this report.

Section 1: Novel Dataset Collection

Stock Price Data (Multiple Companies) –> Primary Companies (Oil & Gas): 1. ExxonMobil (XOM) – since you already have it. 2. Chevron (CVX) 3. BP (BP) 4. Shell (SHEL) 5. Sector ETF –> Energy Select Sector SPDR Fund (XLE) – This helps show how the entire energy sector behaves compared to individual stocks.

[ ]:
```
import pandas as pd
import yfinance as yf
import os

# Define a folder path for all your files
FDS_FOLDER = "FDS project"

# 1. Ensure the folder exists (creates it if not present)
os.makedirs(FDS_FOLDER, exist_ok=True)


# STEP 1: DOWNLOAD & SAVE INDIVIDUAL CSV FILES TO "FDS project" FOLDER

def download_and_save_data(ticker, start_date="2019-01-01",␣
 ↪end_date="2025-01-31"):

    df = yf.download(ticker, start=start_date, end=end_date, interval="1d")

    # We save with index=True so the dates go in the first column.
    csv_path = os.path.join(FDS_FOLDER, f"{ticker}_data.csv")
    df.to_csv(csv_path, index=True)
    print(f"{ticker} data saved to {csv_path}")

# List of tickers
tickers = ["XOM", "CVX", "BP", "SHEL", "SPY"]

# Download each ticker's data
for t in tickers:
    download_and_save_data(t)


# STEP 2: MERGE ALL CSV FILES SIDE-BY-SIDE (WIDE FORMAT)
#         WITH "Date" AS THE FIRST COLUMN
```

```python
def load_and_rename(csv_file, ticker):
    """
    Reads a CSV file (which has 2 extra header rows in your case,
    so adjust skiprows if needed), renames 'Price' to 'Date',
    parses it as a datetime column, and appends the ticker symbol
    to other columns (e.g. XOM_Open, XOM_Close).
    """

    df = pd.read_csv(csv_file, skiprows=0)

    if "Unnamed: 0" in df.columns:
        df.rename(columns={"Unnamed: 0": "Date"}, inplace=True)
    else:
        # If your CSV actually has a "Price" column, rename that instead
        if "Price" in df.columns:
            df.rename(columns={"Price": "Date"}, inplace=True)

    # Convert Date column to datetime
    df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d", errors="coerce")

    # Rename columns to XOM_Open, XOM_Close, etc.
    rename_map = {
        "Open":      f"{ticker}_Open",
        "High":      f"{ticker}_High",
        "Low":       f"{ticker}_Low",
        "Close":     f"{ticker}_Close",
        "Adj Close": f"{ticker}_Adj_Close",
        "Volume":    f"{ticker}_Volume",
    }
    df.rename(columns=rename_map, inplace=True)

    return df

merged_df = None

for t in tickers:
    # Path to each CSV in the FDS project folder
    csv_path = os.path.join(FDS_FOLDER, f"{t}_data.csv")
    temp_df = load_and_rename(csv_path, t)

    if merged_df is None:
        # First ticker: just take it as our base
        merged_df = temp_df
    else:
        # Merge side-by-side on 'Date'
        merged_df = pd.merge(merged_df, temp_df, on="Date", how="outer")

# Sort by date
merged_df.sort_values("Date", inplace=True)


# STEP 3 : REORDER COLUMNS (Date first, then XOM_Open, XOM_High, etc.)


base_cols = ["Open", "High", "Low", "Close", "Adj_Close", "Volume"]


final_cols = ["Date"]
for t in tickers:
    for c in base_cols:
        col_name = f"{t}_{c}"
        final_cols.append(col_name)

# Keep only existing columns in that order
existing_cols = [col for col in final_cols if col in merged_df.columns]
merged_df = merged_df[existing_cols]

# STEP 4 : SAVE THE FINAL COMBINED CSV IN THE FDS project FOLDER

combined_csv_path = os.path.join(FDS_FOLDER, "combined_stocks.csv")
merged_df.to_csv(combined_csv_path, index=False)
print(f"Combined wide-format CSV saved to {combined_csv_path}")

# Quick check
print("Columns in final DataFrame:")
print(merged_df.columns.tolist())
print("First few rows:")
print(merged_df.head())
```

[ ]:
```
import yfinance as yf
import pandas as pd
import os

FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

start_date = "2019-01-01"
end_date   = "2025-01-31"
interval   = "1d"


def flatten_columns(df):
    """
    If the DataFrame has multi-level columns, flatten them into a single level.
    For example, ("Adj Close", "BZ=F") becomes "Adj Close_BZ=F".
    """
    if isinstance(df.columns, pd.MultiIndex):
        # Convert multi-level column tuples to a single string
        df.columns = df.columns.to_flat_index()
        df.columns = ["_".join([str(c) for c in col if c]) for col in df.
↪columns]
    return df

def rename_brent_columns(df):
    """
    Renames columns that contain 'Open', 'High', 'Low', 'Close', 'Adj Close',␣
↪'Volume'
    to 'Brent_Open', 'Brent_High', etc. Also ensures there's a 'Date' column.
    """
    rename_map = {}
    for col in df.columns:
        col_lower = col.lower()  # for easy matching
        if "open" in col_lower:
            rename_map[col] = "Brent_Open"
        elif "high" in col_lower:
            rename_map[col] = "Brent_High"
        elif "low" in col_lower:
            rename_map[col] = "Brent_Low"
        elif "adj close" in col_lower or "adjclose" in col_lower:
            rename_map[col] = "Brent_Adj_Close"
        elif "close" in col_lower and "adj" not in col_lower:
            rename_map[col] = "Brent_Close"
        elif "volume" in col_lower:
            rename_map[col] = "Brent_Volume"
        elif "date" in col_lower:
            rename_map[col] = "Date"
    df.rename(columns=rename_map, inplace=True)

    # Keep only the columns we need
    keep_cols =␣
↪["Date","Brent_Open","Brent_High","Brent_Low","Brent_Close","Brent_Adj_Close","Brent_Volume
    df = df[[c for c in keep_cols if c in df.columns]]
    return df

def rename_wti_columns(df):
    """
    Renames columns that contain 'Open', 'High', 'Low', 'Close', 'Adj Close',␣
↪'Volume'
    to 'WTI_Open', 'WTI_High', etc. Also ensures there's a 'Date' column.
    """
    rename_map = {}
    for col in df.columns:
        col_lower = col.lower()
        if "open" in col_lower:
            rename_map[col] = "WTI_Open"
        elif "high" in col_lower:
            rename_map[col] = "WTI_High"
        elif "low" in col_lower:
            rename_map[col] = "WTI_Low"
        elif "adj close" in col_lower or "adjclose" in col_lower:
            rename_map[col] = "WTI_Adj_Close"
        elif "close" in col_lower and "adj" not in col_lower:
            rename_map[col] = "WTI_Close"
        elif "volume" in col_lower:
            rename_map[col] = "WTI_Volume"
        elif "date" in col_lower:
            rename_map[col] = "Date"
    df.rename(columns=rename_map, inplace=True)

    # Keep only the columns we need
    keep_cols =␣
  ↪["Date","WTI_Open","WTI_High","WTI_Low","WTI_Close","WTI_Adj_Close","WTI_Volume"]
    df = df[[c for c in keep_cols if c in df.columns]]
    return df


# 3) DOWNLOAD & PROCESS BRENT


df_brent = yf.download(
    tickers="BZ=F",
    start=start_date,
    end=end_date,
    interval=interval
)

# Flatten columns if multi-level
df_brent = flatten_columns(df_brent)

# Convert index to a normal column
df_brent.reset_index(inplace=True)

# Rename columns for Brent
df_brent = rename_brent_columns(df_brent)
```

```
# 4) DOWNLOAD & PROCESS WTI


df_wti = yf.download(
    tickers="CL=F",
    start=start_date,
    end=end_date,
    interval=interval
)

df_wti = flatten_columns(df_wti)
df_wti.reset_index(inplace=True)
df_wti = rename_wti_columns(df_wti)


# 5) MERGE & SAVE


df_merged = pd.merge(df_brent, df_wti, on="Date", how="outer")
df_merged.sort_values("Date", inplace=True)

output_csv = os.path.join(FDS_FOLDER, "Brent_WTI_data.csv")
df_merged.to_csv(output_csv, index=False)

print(f"Merged Brent + WTI data saved to: {output_csv}")
print("Columns:", df_merged.columns.tolist())
print("Preview of the merged DataFrame:")
print(df_merged.head())
```

[ ]:
```
import pandas as pd
import pandas_datareader.data as web
import os
from datetime import datetime

# 1) Define FRED economic indicators
indicators = {
    "Inflation_Rate": "CPIAUCSL",  # Consumer Price Index (CPI)
    "Interest_Rate": "DFF",        # Effective Federal Funds Rate
    "GDP": "GDP"                   # Real Gross Domestic Product
}

# 2) Define date range
start_date = "2019-01-01"
end_date = "2025-01-31"

# 3) Fetch data from FRED
macro_data = pd.DataFrame()
```

```
for name, fred_code in indicators.items():
    macro_data[name] = web.DataReader(fred_code, "fred", start_date, end_date)


FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

csv_path = os.path.join(FDS_FOLDER, "Macroeconomic_Data.csv")
macro_data.to_csv(csv_path)

print(macro_data.head(30))
print(f"\nMacroeconomic data saved to: {csv_path}")
```

XOM NEWS DATA

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your desired date range
start_date = "2019-01-01"
end_date   = "2019-12-31"

# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
   f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2019.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your desired date range
start_date = "2020-01-01"
end_date   = "2020-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
  ↪2019
```

```
# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
    f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2020.csv")
df_xom.to_csv(csv_path, index=False)
```

```
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your desired date range
start_date = "2021-01-01"
end_date   = "2021-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
 ↪2019

# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
   f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
```

```
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2021.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your desired date range
start_date = "2022-01-01"
end_date   = "2022-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
↪2019

# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
   f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
```

```python
    f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2022.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create/ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     # 2) Define your desired date range
     start_date = "2023-01-01"
     end_date   = "2023-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
      ↪2019

     # 3) Build RSS feed URLs that incorporate the date filters
     #    For example: ExxonMobil after:2019-01-01 before:2020-01-01
     rss_feeds = [
        f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
        f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
     ]

     # 4) Fetch all articles from these feeds
     all_news = []
     for feed_url in rss_feeds:
         print(f"Fetching XOM news from: {feed_url}")
         feed = feedparser.parse(feed_url)
         time.sleep(2)  # Avoid rate-limiting
```

```
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2023.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your desired date range
start_date = "2024-01-01"
end_date   = "2024-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
 ↪2019

# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
   f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2024.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create/ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)
```

```
# 2) Define your desired date range
start_date = "2025-01-01"
end_date   = "2025-12-31"  # Note: "before:2020-01-01" will cover up to Dec 31,␣
↪2019

# 3) Build RSS feed URLs that incorporate the date filters
#    For example: ExxonMobil after:2019-01-01 before:2020-01-01
rss_feeds = [
   f"https://news.google.com/rss/search?q=ExxonMobil+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=XOM+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+quarterly+results+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
   f"https://news.google.com/rss/search?q=ExxonMobil+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Fetch all articles from these feeds
all_news = []
for feed_url in rss_feeds:
    print(f"Fetching XOM news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Avoid rate-limiting

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })
```

```
# 5) Convert to a DataFrame and remove duplicates (optional)
df_xom = pd.DataFrame(all_news)

# 6) Save to CSV (name it to reflect the date range)
csv_path = os.path.join(FDS_FOLDER, "XOM_News_2025.csv")
df_xom.to_csv(csv_path, index=False)
print(f"Saved ExxonMobil news to {csv_path}")

# 7) Quick preview
print(df_xom.head())
print("\nCount of articles:", df_xom.shape[0])
```

SHELL NEWS DATA

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create or ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     # 2) Define your date range (for the query strings)
     start_date = "2019-01-01"
     end_date   = "2020-01-01"  # "before:2020-01-01" covers 2019

     # 3) Build multiple RSS feeds for Shell
     #    We add "after:2019-01-01 before:2020-01-01" to some queries to target 2019
     rss_feeds = [
         f"https://news.google.com/rss/search?q=Shell+stock+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+oil+market+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

         # Additional queries for broader coverage
         f"https://news.google.com/rss/search?q=Shell+earnings+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+environment+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+global+markets+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+expansions+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
     ]

     # 4) Parse all feeds and collect articles
     all_news = []

     for feed_url in rss_feeds:
         print(f"Fetching Shell news from: {feed_url}")
         feed = feedparser.parse(feed_url)
         time.sleep(2)  # Sleep to avoid hitting rate limits

         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title":          entry.get("title", ""),
                 "summary":        entry.get("summary", ""),
                 "link":           entry.get("link", "")
             })

     # 5) Convert to DataFrame & remove duplicates
     df_shell = pd.DataFrame(all_news)

     # 6) Save to CSV in the FDS project folder
     csv_path = os.path.join(FDS_FOLDER, "Shell_News_2019.csv")
     df_shell.to_csv(csv_path, index=False)
     print(f"\nSaved Shell news to {csv_path}")

     # 7) Quick preview
     print(df_shell.head())
     print("\nArticle count:", len(df_shell))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)
```

```
# 2) Define your date range (for the query strings)
start_date = "2020-01-01"
end_date   = "2020-12-31"


rss_feeds = [
    f"https://news.google.com/rss/search?q=Shell+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Shell+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+expansions+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Shell news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_shell = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Shell_News_2020.csv")
df_shell.to_csv(csv_path, index=False)
print(f"\nSaved Shell news to {csv_path}")

# 7) Quick preview
print(df_shell.head())
print("\nArticle count:", len(df_shell))
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create or ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     # 2) Define your date range (for the query strings)
     start_date = "2021-01-01"
     end_date   = "2021-12-31"


     rss_feeds = [
         f"https://news.google.com/rss/search?q=Shell+stock+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+oil+market+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

         # Additional queries for broader coverage
         f"https://news.google.com/rss/search?q=Shell+earnings+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+environment+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+global+markets+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+expansions+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
     ]

     # 4) Parse all feeds and collect articles
     all_news = []

     for feed_url in rss_feeds:
         print(f"Fetching Shell news from: {feed_url}")
         feed = feedparser.parse(feed_url)
         time.sleep(2)  # Sleep to avoid hitting rate limits

         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title":          entry.get("title", ""),
                 "summary":        entry.get("summary", ""),
                 "link":           entry.get("link", "")
             })

     # 5) Convert to DataFrame & remove duplicates
     df_shell = pd.DataFrame(all_news)

     # 6) Save to CSV in the FDS project folder
     csv_path = os.path.join(FDS_FOLDER, "Shell_News_2021.csv")
     df_shell.to_csv(csv_path, index=False)
     print(f"\nSaved Shell news to {csv_path}")

     # 7) Quick preview
     print(df_shell.head())
     print("\nArticle count:", len(df_shell))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)
```

```
# 2) Define your date range (for the query strings)
start_date = "2022-01-01"
end_date   = "2022-12-31"


rss_feeds = [
    f"https://news.google.com/rss/search?q=Shell+stock+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+oil+market+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Shell+earnings+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+environment+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+global+markets+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+expansions+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Shell news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_shell = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Shell_News_2022.csv")
df_shell.to_csv(csv_path, index=False)
print(f"\nSaved Shell news to {csv_path}")

# 7) Quick preview
print(df_shell.head())
print("\nArticle count:", len(df_shell))
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create or ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     # 2) Define your date range (for the query strings)
     start_date = "2023-01-01"
     end_date   = "2023-12-31"


     rss_feeds = [
         f"https://news.google.com/rss/search?q=Shell+stock+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+oil+market+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
      ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

         # Additional queries for broader coverage
         f"https://news.google.com/rss/search?q=Shell+earnings+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+environment+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+global+markets+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
         f"https://news.google.com/rss/search?q=Shell+expansions+after:
      ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
     ]

     # 4) Parse all feeds and collect articles
     all_news = []

     for feed_url in rss_feeds:
         print(f"Fetching Shell news from: {feed_url}")
         feed = feedparser.parse(feed_url)
         time.sleep(2)  # Sleep to avoid hitting rate limits

         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title":          entry.get("title", ""),
                 "summary":        entry.get("summary", ""),
                 "link":           entry.get("link", "")
             })

     # 5) Convert to DataFrame & remove duplicates
     df_shell = pd.DataFrame(all_news)

     # 6) Save to CSV in the FDS project folder
     csv_path = os.path.join(FDS_FOLDER, "Shell_News_2023.csv")
     df_shell.to_csv(csv_path, index=False)
     print(f"\nSaved Shell news to {csv_path}")

     # 7) Quick preview
     print(df_shell.head())
     print("\nArticle count:", len(df_shell))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)
```

```
# 2) Define your date range (for the query strings)
start_date = "2024-01-01"
end_date   = "2024-12-31"


rss_feeds = [
    f"https://news.google.com/rss/search?q=Shell+stock+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+oil+market+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Shell+earnings+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+expansions+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Shell news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_shell = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Shell_News_2024.csv")
df_shell.to_csv(csv_path, index=False)
print(f"\nSaved Shell news to {csv_path}")

# 7) Quick preview
print(df_shell.head())
print("\nArticle count:", len(df_shell))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2025-01-01"
end_date   = "2025-12-31"


rss_feeds = [
    f"https://news.google.com/rss/search?q=Shell+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+news+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Royal+Dutch+Shell+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+plc+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Shell+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+energy+sector+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+finance+analysis+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+global+markets+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Shell+expansions+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Shell news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_shell = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Shell_News_2025.csv")
df_shell.to_csv(csv_path, index=False)
print(f"\nSaved Shell news to {csv_path}")

# 7) Quick preview
print(df_shell.head())
print("\nArticle count:", len(df_shell))
```

CVX NEWS DATA

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create or ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
```

```
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2019-01-01"
end_date   = "2019-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2019.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2020-01-01"
end_date   = "2020-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2020.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
```

```
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2021-01-01"
end_date   = "2021-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2021.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2022-01-01"
end_date   = "2022-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2022.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
```

```python
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2023-01-01"
end_date   = "2023-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2023.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2024-01-01"
end_date   = "2024-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
 ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
 ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2024.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create or ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
```

```
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (for the query strings)
start_date = "2025-01-01"
end_date   = "2025-12-31"  # "before:2020-01-01" covers 2019

# 3) Build multiple RSS feeds for Chevron (2019 coverage)
#    Using synonyms like "Chevron Corporation" or "CVX"
rss_feeds = [
    f"https://news.google.com/rss/search?q=Chevron+stock+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+news+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+oil+market+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+corporation+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=CVX+stock+after:{start_date}+before:
  ↪{end_date}&hl=en-US&gl=US&ceid=US:en",

    # Additional queries for broader coverage
    f"https://news.google.com/rss/search?q=Chevron+earnings+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+environment+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+energy+sector+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+finance+analysis+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+global+markets+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en",
    f"https://news.google.com/rss/search?q=Chevron+expansions+after:
  ↪{start_date}+before:{end_date}&hl=en-US&gl=US&ceid=US:en"
]

# 4) Parse all feeds and collect articles
all_news = []

for feed_url in rss_feeds:
    print(f"Fetching Chevron news from: {feed_url}")
    feed = feedparser.parse(feed_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title":          entry.get("title", ""),
            "summary":        entry.get("summary", ""),
            "link":           entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_chevron = pd.DataFrame(all_news)

# 6) Save to CSV in the FDS project folder
csv_path = os.path.join(FDS_FOLDER, "Chevron_News_2025.csv")
df_chevron.to_csv(csv_path, index=False)
print(f"\nSaved Chevron news to {csv_path}")

# 7) Quick preview
print(df_chevron.head())
print("\nArticle count:", len(df_chevron))
```

BP NEWS DATA

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     start_date = "2019-01-01"
     end_date   = "2019-12-31"

     # BP synonyms
     bp_synonyms = [
         f"BP+stock+after:{start_date}+before:{end_date}",
         f"BP+news+after:{start_date}+before:{end_date}",
         f"BP+oil+market+after:{start_date}+before:{end_date}",
         f"BP+corporation+after:{start_date}+before:{end_date}",
         f"BP+plc+after:{start_date}+before:{end_date}",
         f"BP+earnings+after:{start_date}+before:{end_date}",
         f"BP+environment+after:{start_date}+before:{end_date}",
         f"BP+energy+sector+after:{start_date}+before:{end_date}",
         f"BP+finance+analysis+after:{start_date}+before:{end_date}",
         f"BP+global+markets+after:{start_date}+before:{end_date}",
         f"BP+expansions+after:{start_date}+before:{end_date}"
     ]

     all_news = []
     for term in bp_synonyms:
```

```
    rss_url = f"https://news.google.com/rss/search?
↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching BP news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

df_bp = pd.DataFrame(all_news)

csv_path = os.path.join(FDS_FOLDER, "BP_News_2019.csv")
df_bp.to_csv(csv_path, index=False)
print(f"\nSaved BP news to {csv_path}")
print("Article count:", len(df_bp))
print(df_bp.head())
print("\nArticle count:", len(df_bp))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

start_date = "2020-01-01"
end_date   = "2020-12-31"

# BP synonyms
bp_synonyms = [
    f"BP+stock+after:{start_date}+before:{end_date}",
    f"BP+news+after:{start_date}+before:{end_date}",
    f"BP+oil+market+after:{start_date}+before:{end_date}",
    f"BP+corporation+after:{start_date}+before:{end_date}",
    f"BP+plc+after:{start_date}+before:{end_date}",
    f"BP+earnings+after:{start_date}+before:{end_date}",
    f"BP+environment+after:{start_date}+before:{end_date}",
    f"BP+energy+sector+after:{start_date}+before:{end_date}",
    f"BP+finance+analysis+after:{start_date}+before:{end_date}",
    f"BP+global+markets+after:{start_date}+before:{end_date}",
    f"BP+expansions+after:{start_date}+before:{end_date}"
]
```

```
all_news = []
for term in bp_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching BP news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

df_bp = pd.DataFrame(all_news)

csv_path = os.path.join(FDS_FOLDER, "BP_News_2020.csv")
df_bp.to_csv(csv_path, index=False)
print(f"\nSaved BP news to {csv_path}")
print("Article count:", len(df_bp))
print(df_bp.head())
print("\nArticle count:", len(df_bp))
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

start_date = "2021-01-01"
end_date   = "2021-12-31"

# BP synonyms
bp_synonyms = [
    f"BP+stock+after:{start_date}+before:{end_date}",
    f"BP+news+after:{start_date}+before:{end_date}",
    f"BP+oil+market+after:{start_date}+before:{end_date}",
    f"BP+corporation+after:{start_date}+before:{end_date}",
    f"BP+plc+after:{start_date}+before:{end_date}",
    f"BP+earnings+after:{start_date}+before:{end_date}",
    f"BP+environment+after:{start_date}+before:{end_date}",
    f"BP+energy+sector+after:{start_date}+before:{end_date}",
    f"BP+finance+analysis+after:{start_date}+before:{end_date}",
    f"BP+global+markets+after:{start_date}+before:{end_date}",
    f"BP+expansions+after:{start_date}+before:{end_date}"
]

all_news = []
for term in bp_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching BP news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

df_bp = pd.DataFrame(all_news)

csv_path = os.path.join(FDS_FOLDER, "BP_News_2021.csv")
df_bp.to_csv(csv_path, index=False)
print(f"\nSaved BP news to {csv_path}")
print("Article count:", len(df_bp))
print(df_bp.head())
print("\nArticle count:", len(df_bp))
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     start_date = "2022-01-01"
     end_date   = "2022-12-31"

     # BP synonyms
     bp_synonyms = [
         f"BP+stock+after:{start_date}+before:{end_date}",
         f"BP+news+after:{start_date}+before:{end_date}",
         f"BP+oil+market+after:{start_date}+before:{end_date}",
         f"BP+corporation+after:{start_date}+before:{end_date}",
         f"BP+plc+after:{start_date}+before:{end_date}",
         f"BP+earnings+after:{start_date}+before:{end_date}",
         f"BP+environment+after:{start_date}+before:{end_date}",
         f"BP+energy+sector+after:{start_date}+before:{end_date}",
         f"BP+finance+analysis+after:{start_date}+before:{end_date}",
         f"BP+global+markets+after:{start_date}+before:{end_date}",
         f"BP+expansions+after:{start_date}+before:{end_date}"
     ]

     all_news = []
     for term in bp_synonyms:
         rss_url = f"https://news.google.com/rss/search?
      ↪q={term}&hl=en-US&gl=US&ceid=US:en"
         print(f"Fetching BP news from: {rss_url}")
         feed = feedparser.parse(rss_url)
         time.sleep(2)
         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title": entry.get("title", ""),
                 "summary": entry.get("summary", ""),
                 "link": entry.get("link", "")
             })

     df_bp = pd.DataFrame(all_news)

     csv_path = os.path.join(FDS_FOLDER, "BP_News_2022.csv")
     df_bp.to_csv(csv_path, index=False)
     print(f"\nSaved BP news to {csv_path}")
     print("Article count:", len(df_bp))
     print(df_bp.head())
     print("\nArticle count:", len(df_bp))
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

start_date = "2023-01-01"
end_date   = "2023-12-31"

# BP synonyms
bp_synonyms = [
    f"BP+stock+after:{start_date}+before:{end_date}",
    f"BP+news+after:{start_date}+before:{end_date}",
    f"BP+oil+market+after:{start_date}+before:{end_date}",
    f"BP+corporation+after:{start_date}+before:{end_date}",
    f"BP+plc+after:{start_date}+before:{end_date}",
    f"BP+earnings+after:{start_date}+before:{end_date}",
    f"BP+environment+after:{start_date}+before:{end_date}",
    f"BP+energy+sector+after:{start_date}+before:{end_date}",
    f"BP+finance+analysis+after:{start_date}+before:{end_date}",
    f"BP+global+markets+after:{start_date}+before:{end_date}",
    f"BP+expansions+after:{start_date}+before:{end_date}"
]

all_news = []
for term in bp_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching BP news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

df_bp = pd.DataFrame(all_news)

csv_path = os.path.join(FDS_FOLDER, "BP_News_2023.csv")
df_bp.to_csv(csv_path, index=False)
print(f"\nSaved BP news to {csv_path}")
print("Article count:", len(df_bp))
print(df_bp.head())
print("\nArticle count:", len(df_bp))
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     start_date = "2024-01-01"
     end_date   = "2024-12-31"

     # BP synonyms
     bp_synonyms = [
         f"BP+stock+after:{start_date}+before:{end_date}",
         f"BP+news+after:{start_date}+before:{end_date}",
         f"BP+oil+market+after:{start_date}+before:{end_date}",
         f"BP+corporation+after:{start_date}+before:{end_date}",
         f"BP+plc+after:{start_date}+before:{end_date}",
         f"BP+earnings+after:{start_date}+before:{end_date}",
         f"BP+environment+after:{start_date}+before:{end_date}",
         f"BP+energy+sector+after:{start_date}+before:{end_date}",
         f"BP+finance+analysis+after:{start_date}+before:{end_date}",
         f"BP+global+markets+after:{start_date}+before:{end_date}",
         f"BP+expansions+after:{start_date}+before:{end_date}"
     ]

     all_news = []
     for term in bp_synonyms:
         rss_url = f"https://news.google.com/rss/search?
      ↪q={term}&hl=en-US&gl=US&ceid=US:en"
         print(f"Fetching BP news from: {rss_url}")
         feed = feedparser.parse(rss_url)
         time.sleep(2)
         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title": entry.get("title", ""),
                 "summary": entry.get("summary", ""),
                 "link": entry.get("link", "")
             })

     df_bp = pd.DataFrame(all_news)
     csv_path = os.path.join(FDS_FOLDER, "BP_News_2024.csv")
     df_bp.to_csv(csv_path, index=False)
     print(f"\nSaved BP news to {csv_path}")
     print("Article count:", len(df_bp))
     print(df_bp.head())
     print("\nArticle count:", len(df_bp))
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     start_date = "2025-01-01"
     end_date   = "2025-12-31"
```

```
# BP synonyms
bp_synonyms = [
    f"BP+stock+after:{start_date}+before:{end_date}",
    f"BP+news+after:{start_date}+before:{end_date}",
    f"BP+oil+market+after:{start_date}+before:{end_date}",
    f"BP+corporation+after:{start_date}+before:{end_date}",
    f"BP+plc+after:{start_date}+before:{end_date}",
    f"BP+earnings+after:{start_date}+before:{end_date}",
    f"BP+environment+after:{start_date}+before:{end_date}",
    f"BP+energy+sector+after:{start_date}+before:{end_date}",
    f"BP+finance+analysis+after:{start_date}+before:{end_date}",
    f"BP+global+markets+after:{start_date}+before:{end_date}",
    f"BP+expansions+after:{start_date}+before:{end_date}"
]

all_news = []
for term in bp_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching BP news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

df_bp = pd.DataFrame(all_news)

csv_path = os.path.join(FDS_FOLDER, "BP_News_2025.csv")
df_bp.to_csv(csv_path, index=False)
print(f"\nSaved BP news to {csv_path}")
print("Article count:", len(df_bp))
print(df_bp.head())
print("\nArticle count:", len(df_bp))
```

SPY NEWS DATA

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create/ensure the "FDS project" folder
```

```
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2019-01-01"
end_date   = "2019-12-31"  # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
  ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2019.csv")
```

```
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2020-01-01"
end_date   = "2020-12-31"  # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
  ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2020.csv")
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2021-01-01"
end_date   = "2021-12-31"   # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
 ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]
```

```
# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2021.csv")
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2022-01-01"
end_date   = "2022-12-31"  # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
  ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2022.csv")
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os
```

```
# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2023-01-01"
end_date   = "2023-12-31"  # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
 ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)
```

```
# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2023.csv")
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (example: only January 2019)
start_date = "2024-01-01"
end_date   = "2024-12-31"  # "before:2019-02-01" covers January 2019

# 3) Expanded synonyms for SPY
spy_synonyms = [
    f"SPY+stock+after:{start_date}+before:{end_date}",
    f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
  ↪'&'
    f"SPY+ETF+after:{start_date}+before:{end_date}",
    f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
    f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
    f"SPY+volatility+after:{start_date}+before:{end_date}",
    f"SPY+returns+after:{start_date}+before:{end_date}",
    f"SPY+performance+after:{start_date}+before:{end_date}",
    f"SPY+holdings+after:{start_date}+before:{end_date}",
    f"SPY+dividends+after:{start_date}+before:{end_date}",
    f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
    f"SPY+analysis+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in spy_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching SPY news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_spy = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "SPY_News_2024.csv")
df_spy.to_csv(csv_path, index=False)

print(f"\nSaved SPY news to {csv_path}")
print("Article count:", len(df_spy))
print(df_spy.head())
```

```
[ ]: import feedparser
     import pandas as pd
     import time
     import os

     # 1) Create/ensure the "FDS project" folder
     FDS_FOLDER = "FDS project"
     os.makedirs(FDS_FOLDER, exist_ok=True)

     # 2) Define your date range (example: only January 2019)
     start_date = "2025-01-01"
     end_date   = "2025-12-31"  # "before:2019-02-01" covers January 2019

     # 3) Expanded synonyms for SPY
     spy_synonyms = [
         f"SPY+stock+after:{start_date}+before:{end_date}",
         f"SPDR+S%26P+500+ETF+after:{start_date}+before:{end_date}",  # Use %26 for␣
       ↪'&'
         f"SPY+ETF+after:{start_date}+before:{end_date}",
         f"SPY+finance+analysis+after:{start_date}+before:{end_date}",
         f"SPY+S%26P+500+after:{start_date}+before:{end_date}",
         f"SPY+volatility+after:{start_date}+before:{end_date}",
         f"SPY+returns+after:{start_date}+before:{end_date}",
         f"SPY+performance+after:{start_date}+before:{end_date}",
         f"SPY+holdings+after:{start_date}+before:{end_date}",
         f"SPY+dividends+after:{start_date}+before:{end_date}",
         f"S%26P+500+ETF+after:{start_date}+before:{end_date}",
         f"SPY+analysis+after:{start_date}+before:{end_date}"
     ]

     # 4) Parse all feeds
     all_news = []

     for term in spy_synonyms:
         rss_url = f"https://news.google.com/rss/search?
       ↪q={term}&hl=en-US&gl=US&ceid=US:en"
         print(f"Fetching SPY news from: {rss_url}")
         feed = feedparser.parse(rss_url)
         time.sleep(2)  # Sleep to avoid hitting rate limits

         for entry in feed.entries:
             all_news.append({
                 "published_date": entry.get("published", ""),
                 "title": entry.get("title", ""),
                 "summary": entry.get("summary", ""),
                 "link": entry.get("link", "")
             })

     # 5) Convert to DataFrame & remove duplicates
     df_spy = pd.DataFrame(all_news)

     # 6) Save to CSV
     csv_path = os.path.join(FDS_FOLDER, "SPY_News_2025.csv")
     df_spy.to_csv(csv_path, index=False)

     print(f"\nSaved SPY news to {csv_path}")
     print("Article count:", len(df_spy))
     print(df_spy.head())
```

BRENT NEWS DATA

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2019-01-01"
end_date   = "2019-12-31"  # "before:2020-01-01" covers 2019
```

```
# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2019.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os
```

```python
# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2020-01-01"
end_date   = "2020-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2020.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2021-01-01"
end_date   = "2021-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })
```

```
# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2021.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2022-01-01"
end_date   = "2022-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2022.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2023-01-01"
end_date   = "2023-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []
```

```
for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2023.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2024-01-01"
end_date   = "2024-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2024.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2025-01-01"
end_date   = "2025-12-31"  # "before:2020-01-01" covers 2019
```

```
# 3) Expanded synonyms for Brent
brent_synonyms = [
    f"Brent+crude+oil+after:{start_date}+before:{end_date}",
    f"Brent+oil+price+after:{start_date}+before:{end_date}",
    f"Brent+oil+market+after:{start_date}+before:{end_date}",
    f"Brent+global+markets+after:{start_date}+before:{end_date}",
    f"Brent+energy+sector+after:{start_date}+before:{end_date}",
    f"Brent+volatility+after:{start_date}+before:{end_date}",
    f"Brent+futures+after:{start_date}+before:{end_date}",
    f"Brent+analysis+after:{start_date}+before:{end_date}",
    f"Brent+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in brent_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching Brent news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_brent = pd.DataFrame(all_news)
# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "Brent_News_2025.csv")
df_brent.to_csv(csv_path, index=False)

print(f"\nSaved Brent news to {csv_path}")
print("Article count:", len(df_brent))
print(df_brent.head())
```

[ ]:
```
WTI NEWS DATA
```

[ ]:
```
import feedparser
import pandas as pd
import time
```

```python
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2019-01-01"
end_date   = "2019-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2019.csv")
df_wti.to_csv(csv_path, index=False)
```

```
print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2020-01-01"
end_date   = "2020-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)
# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2020.csv")
df_wti.to_csv(csv_path, index=False)

print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

```
[ ]: import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2021-01-01"
end_date   = "2021-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits
    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2021.csv")
df_wti.to_csv(csv_path, index=False)

print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2022-01-01"
end_date   = "2022-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
```

```
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2022.csv")
df_wti.to_csv(csv_path, index=False)

print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2023-01-01"
end_date   = "2023-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2023.csv")
df_wti.to_csv(csv_path, index=False)

print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2024-01-01"
```

```
end_date   = "2024-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
  ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)

# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2024.csv")
df_wti.to_csv(csv_path, index=False)

print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

[ ]:
```
import feedparser
import pandas as pd
import time
```

```python
import os

# 1) Create/ensure the "FDS project" folder
FDS_FOLDER = "FDS project"
os.makedirs(FDS_FOLDER, exist_ok=True)

# 2) Define your date range (e.g., full 2019)
start_date = "2025-01-01"
end_date   = "2025-12-31"  # "before:2020-01-01" covers 2019

# 3) Expanded synonyms for WTI
wti_synonyms = [
    f"WTI+crude+oil+after:{start_date}+before:{end_date}",
    f"WTI+oil+price+after:{start_date}+before:{end_date}",
    f"WTI+oil+market+after:{start_date}+before:{end_date}",
    f"WTI+global+markets+after:{start_date}+before:{end_date}",
    f"WTI+energy+sector+after:{start_date}+before:{end_date}",
    f"WTI+volatility+after:{start_date}+before:{end_date}",
    f"WTI+futures+after:{start_date}+before:{end_date}",
    f"WTI+analysis+after:{start_date}+before:{end_date}",
    f"WTI+production+after:{start_date}+before:{end_date}"
]

# 4) Parse all feeds
all_news = []

for term in wti_synonyms:
    rss_url = f"https://news.google.com/rss/search?
 ↪q={term}&hl=en-US&gl=US&ceid=US:en"
    print(f"Fetching WTI news from: {rss_url}")
    feed = feedparser.parse(rss_url)
    time.sleep(2)  # Sleep to avoid hitting rate limits

    for entry in feed.entries:
        all_news.append({
            "published_date": entry.get("published", ""),
            "title": entry.get("title", ""),
            "summary": entry.get("summary", ""),
            "link": entry.get("link", "")
        })

# 5) Convert to DataFrame & remove duplicates
df_wti = pd.DataFrame(all_news)
# 6) Save to CSV
csv_path = os.path.join(FDS_FOLDER, "WTI_News_2025.csv")
df_wti.to_csv(csv_path, index=False)
```

```
print(f"\nSaved WTI news to {csv_path}")
print("Article count:", len(df_wti))
print(df_wti.head())
```

Section 2 : SQL Part 1

1. Installing Required Python Packages

```
[ ]: pip install pandas sqlalchemy pymysql
```

2. Load News Data into MySQL (XOM)

```
[ ]: import os
     import pandas as pd
     from sqlalchemy import create_engine

     # 1) Folder and CSV filename
     FDS_FOLDER = "FDS PROJECT"
     csv_filename = "XOM_News_cleaned.csv"  # Example CSV
     csv_path = os.path.join(FDS_FOLDER, csv_filename)

     # 2) Read the CSV into a pandas DataFrame
     df = pd.read_csv(csv_path)
     print(f"Loaded {len(df)} rows from {csv_path}")

     # 3) Create an SQLAlchemy engine for your MySQL server
     #    Adjust username, password, host, port, and database
     engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
      ↪fds_project_db")

     # 4) Write the DataFrame to a MySQL table
     #    'news_data' is the table name (change if needed)
     #    if_exists='append' adds rows to an existing table,
     #    if_exists='replace' would drop & recreate the table
     df.to_sql(name="xom_news", con=engine, if_exists="append", index=False)

     print("Data successfully loaded into MySQL!")
```

3. Load News Data into MySQL (shell)

```
[ ]: import os
     import pandas as pd
     from sqlalchemy import create_engine

     # 1) Folder and CSV filename
```

```
FDS_FOLDER = "FDS PROJECT"
csv_filename = "Shell_News_cleaned.csv"  # Example CSV
csv_path = os.path.join(FDS_FOLDER, csv_filename)

# 2) Read the CSV into a pandas DataFrame
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")

# 3) Create an SQLAlchemy engine for your MySQL server
#    Adjust username, password, host, port, and database
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

# 4) Write the DataFrame to a MySQL table
#    'news_data' is the table name (change if needed)
#    if_exists='append' adds rows to an existing table,
#    if_exists='replace' would drop & recreate the table
df.to_sql(name="shell_news", con=engine, if_exists="append", index=False)

print("Data successfully loaded into MySQL!")
```

4. Load News Data into MySQL (Chevron)

```
[ ]: import os
     import pandas as pd
     from sqlalchemy import create_engine

     # 1) Folder and CSV filename
     FDS_FOLDER = "FDS PROJECT"
     csv_filename = "Chevron_News_cleaned.csv"  # Example CSV
     csv_path = os.path.join(FDS_FOLDER, csv_filename)

     # 2) Read the CSV into a pandas DataFrame
     df = pd.read_csv(csv_path)
     print(f"Loaded {len(df)} rows from {csv_path}")

     # 3) Create an SQLAlchemy engine for your MySQL server
     #    Adjust username, password, host, port, and database
     engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
      ↪fds_project_db")

     # 4) Write the DataFrame to a MySQL table
     #    'news_data' is the table name (change if needed)
     #    if_exists='append' adds rows to an existing table,
     #    if_exists='replace' would drop & recreate the table
     df.to_sql(name="cvx_news", con=engine, if_exists="append", index=False)
```

```
print("Data successfully loaded into MySQL!")
```

5. Load News Data into MySQL (BP)

[ ]:
```
import os
import pandas as pd
from sqlalchemy import create_engine

# 1) Folder and CSV filename
FDS_FOLDER = "FDS PROJECT"
csv_filename = "BP_News_cleaned.csv"  # Example CSV
csv_path = os.path.join(FDS_FOLDER, csv_filename)

# 2) Read the CSV into a pandas DataFrame
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")

# 3) Create an SQLAlchemy engine for your MySQL server
#    Adjust username, password, host, port, and database
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

# 4) Write the DataFrame to a MySQL table
#    'news_data' is the table name (change if needed)
#    if_exists='append' adds rows to an existing table,
#    if_exists='replace' would drop & recreate the table
df.to_sql(name="bp_news", con=engine, if_exists="append", index=False)

print("Data successfully loaded into MySQL!")
```

6. Load News Data into MySQL (SPY)

[ ]:
```
import os
import pandas as pd
from sqlalchemy import create_engine

# 1) Folder and CSV filename
FDS_FOLDER = "FDS PROJECT"
csv_filename = "SPY_News_cleaned.csv"  # Example CSV
csv_path = os.path.join(FDS_FOLDER, csv_filename)

# 2) Read the CSV into a pandas DataFrame
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")

# 3) Create an SQLAlchemy engine for your MySQL server
#    Adjust username, password, host, port, and database
```

```
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

# 4) Write the DataFrame to a MySQL table
#    'news_data' is the table name (change if needed)
#    if_exists='append' adds rows to an existing table,
#    if_exists='replace' would drop & recreate the table
df.to_sql(name="spy_news", con=engine, if_exists="append", index=False)

print("Data successfully loaded into MySQL!")
```

7. Load News Data into MySQL (WTI)

```
[ ]: import os
import pandas as pd
from sqlalchemy import create_engine

# 1) Folder and CSV filename
FDS_FOLDER = "FDS PROJECT"
csv_filename = "WTI_News_cleaned.csv"  # Example CSV
csv_path = os.path.join(FDS_FOLDER, csv_filename)

# 2) Read the CSV into a pandas DataFrame
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")

# 3) Create an SQLAlchemy engine for your MySQL server
#    Adjust username, password, host, port, and database
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

# 4) Write the DataFrame to a MySQL table
#    'news_data' is the table name (change if needed)
#    if_exists='append' adds rows to an existing table,
#    if_exists='replace' would drop & recreate the table
df.to_sql(name="wti_news", con=engine, if_exists="append", index=False)

print("Data successfully loaded into MySQL!")
```

8. Load News Data into MySQL (Brent)

```
[ ]: import os
import pandas as pd
from sqlalchemy import create_engine

# 1) Folder and CSV filename
FDS_FOLDER = "FDS PROJECT"
csv_filename = "Brent_News_cleaned.csv"  # Example CSV
```

```
csv_path = os.path.join(FDS_FOLDER, csv_filename)

# 2) Read the CSV into a pandas DataFrame
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")

# 3) Create an SQLAlchemy engine for your MySQL server
#    Adjust username, password, host, port, and database
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

# 4) Write the DataFrame to a MySQL table
#    'news_data' is the table name (change if needed)
#    if_exists='append' adds rows to an existing table,
#    if_exists='replace' would drop & recreate the table
df.to_sql(name="brent_news", con=engine, if_exists="append", index=False)

print("Data successfully loaded into MySQL!")
```

9. Load Stock and Macro Data (Multiple Tables) into MySQL

```
[ ]: import os
import pandas as pd
from sqlalchemy import create_engine

# Folder where your CSV files live
FDS_FOLDER = "FDS project"

# List of (csv_filename, table_name) pairs
stock_files = [
    ("XOM_data_cleaned.csv", "xom_data"),
    ("CVX_data_cleaned.csv", "cvx_data"),
    ("BP_data_cleaned.csv",  "bp_data"),
    ("SHEL_data_cleaned.csv","shell_data"),
    ("SPY_data_cleaned.csv", "spy_data"),
    ("Brent_WTI_data.csv", "brent_wti_data"),
    ("Macroeconomic_Data.csv", "macroeconomic_data")
]

# Create an engine to your MySQL database
# Format: "mysql+pymysql://USER:PASS@HOST:PORT/DB"
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

for csv_filename, table_name in stock_files:
    csv_path = os.path.join(FDS_FOLDER, csv_filename)
    if not os.path.exists(csv_path):
        print(f"File not found: {csv_path}, skipping...")
        continue

    # Read the CSV into a DataFrame
    df = pd.read_csv(csv_path)

    # If your CSV columns differ (e.g., "Adj Close" vs. "Adj_Close"), rename␣
 ↪them
    # Example: df.rename(columns={"Adj Close": "Adj_Close"}, inplace=True)

    print(f"Loading {csv_filename} into table {table_name}...")

    # Write DataFrame to MySQL table
    # if_exists="replace" drops the table if it exists, then creates a new one
    # if_exists="append" would add rows to an existing table
    df.to_sql(name=table_name, con=engine, if_exists="replace", index=False)

    print(f"Done loading {csv_filename} into {table_name}.\n")
```

10. Analyze Correlation Between Stock Prices and Oil Prices (Includes: correlation matrix, monthly averages, rolling averages, subplots)

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     from sqlalchemy import create_engine

     #########################################################
     # 1) SETUP: MySQL Connection & Table Names
     #########################################################

     # Create the engine to your MySQL database
     engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
      ↪fds_project_db")

     # Brent/WTI table name
     oil_table = "brent_wti_data"

     # Stock tables to process
     stock_tables = [
         "xom_data",
         "cvx_data",
         "bp_data",
         "shell_data",
         "spy_data"
     ]
```

```
# Numeric columns in your stock tables
numeric_candidates = ["Open", "High", "Low", "Close", "Volume"]



df_oil = pd.read_sql(f"SELECT Date, Brent_Close, WTI_Close FROM {oil_table}",␣
 ↪engine)

# Convert 'Date' to datetime (assuming 'YYYY-MM-DD'). Adjust format if needed.
df_oil["Date"] = pd.to_datetime(df_oil["Date"], format="%Y-%m-%d",␣
 ↪errors="coerce")
df_oil.dropna(subset=["Date"], inplace=True)

# Convert Brent/WTI to numeric
df_oil["Brent_Close"] = pd.to_numeric(df_oil["Brent_Close"], errors="coerce")
df_oil["WTI_Close"]   = pd.to_numeric(df_oil["WTI_Close"],   errors="coerce")
df_oil.dropna(subset=["Brent_Close","WTI_Close"], inplace=True)

# Sort by Date
df_oil.sort_values("Date", inplace=True)

print(f"Brent/WTI table: {len(df_oil)} rows after cleaning.\n")

fig, axes = plt.subplots(nrows=len(stock_tables), ncols=2, figsize=(14, 4 *␣
 ↪len(stock_tables)))

# If there's only 1 row, unify the indexing
if len(stock_tables) == 1:
    axes = [axes]


for i, stock_table in enumerate(stock_tables):
    print(f"=== PROCESSING STOCK TABLE: {stock_table.upper()} ===")

    # 4A) Read the entire stock table
    df_stock = pd.read_sql(f"SELECT * FROM {stock_table}", engine)

    # Convert 'Date' to datetime
    if "Date" in df_stock.columns:
        df_stock["Date"] = pd.to_datetime(df_stock["Date"], format="%Y-%m-%d",␣
 ↪errors="coerce")
        df_stock.dropna(subset=["Date"], inplace=True)
    else:
        print("No 'Date' column found, skipping date parsing.")
        continue

    # Convert numeric columns
    for col in numeric_candidates:
        if col in df_stock.columns:
            df_stock[col] = pd.to_numeric(df_stock[col], errors="coerce")

    # Filter out rows with all numeric columns = NaN
    df_stock.dropna(subset=["Close"], how="any", inplace=True)

    # 4B) Rename 'Close' to <STOCK>_Close
    stock_name = stock_table.replace("_data","").upper()  # e.g. xom_data -> XOM
    close_col_name = f"{stock_name}_Close"
    if "Close" in df_stock.columns:
        df_stock.rename(columns={"Close": close_col_name}, inplace=True)
    else:
        print("No 'Close' column found, skipping stats.")
        continue

    # 4C) Merge with Brent/WTI on Date
    df_merged = pd.merge(
        df_stock[["Date", close_col_name]],
        df_oil[["Date", "Brent_Close", "WTI_Close"]],
        on="Date",
        how="inner"
    )
    df_merged.dropna(subset=[close_col_name, "Brent_Close", "WTI_Close"],␣
↪inplace=True)
    df_merged.sort_values("Date", inplace=True)

    print(f"Merged rows: {len(df_merged)}")

    # 4D) Correlation among [stock_close, Brent_Close, WTI_Close]
    corr_matrix = df_merged[[close_col_name, "Brent_Close", "WTI_Close"]].corr()
    print("\n--- Correlation Matrix ---")
    print(corr_matrix)

    ########################################################
    # 4E) MONTHLY AGGREGATES
    ########################################################
    # We'll group by Year/Month to see average
    df_merged["YearMonth"] = df_merged["Date"].dt.to_period("M")
    monthly_agg = df_merged.groupby("YearMonth")[[close_col_name,␣
↪"Brent_Close", "WTI_Close"]].mean()
    print("\n--- Monthly Averages (first 5) ---")
    print(monthly_agg.head(5))

    ########################################################
    # 4F) ROLLING AVERAGE on the stock close
    ########################################################
    # We'll do a 7-row rolling (assuming daily data)
    df_merged.set_index("Date", inplace=True, drop=False)
    rolling_col = f"rolling_7d_{stock_name.lower()}"
    df_merged[rolling_col] = df_merged[close_col_name].rolling(7).mean()
    df_merged.reset_index(drop=True, inplace=True)

    # Show a few rows
    print("\n--- Sample Rolling (7-day) on stock close ---")
    print(df_merged[[close_col_name, rolling_col]].head(10))

    ########################################################
    # 4G) PLOTTING (Subplots)
    ########################################################
    # Left col = line chart, Right col = scatter
    ax_line = axes[i][0] if len(stock_tables) > 1 else axes[0]
    ax_scatter = axes[i][1] if len(stock_tables) > 1 else axes[1]

    # (a) LINE CHART: stock vs. Brent vs. WTI
    df_line = df_merged.copy()
    df_line.set_index("Date", inplace=True)
    df_line[[close_col_name, "Brent_Close", "WTI_Close"]].plot(
        ax=ax_line,
        title=f"{stock_name} vs. Brent/WTI (Line)",
        legend=True
    )

    # (b) SCATTER: stock vs. Brent
    ax_scatter.scatter(df_merged["Brent_Close"], df_merged[close_col_name],␣
 ↪alpha=0.5)
    ax_scatter.set_xlabel("Brent_Close")
    ax_scatter.set_ylabel(close_col_name)
    ax_scatter.set_title(f"{stock_name} vs. Brent (Scatter)")

    print(f"=== FINISHED {stock_name} ===\n")

plt.tight_layout()
plt.show()
```

11. Analyze Correlation Between News Volume and Stock Close Prices (All Entities)

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     from sqlalchemy import create_engine

     ########################################################
     # 1) MySQL Connection & Table Mappings
     ########################################################
```

```
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
↪fds_project_db")

# Map each entity to its actual stock/news tables
entity_map = {
    "xom":  {"stock_table": "xom_data",    "news_table": "xom_news"},
    "cvx":  {"stock_table": "cvx_data",    "news_table": "cvx_news"},
    "bp":   {"stock_table": "bp_data",     "news_table": "bp_news"},
    "shel": {"stock_table": "shell_data",   "news_table": "shell_news"},
    "spy":  {"stock_table": "spy_data",    "news_table": "spy_news"}
}

# Create subplots: one row per entity, two columns (line chart, scatter)
fig, axes = plt.subplots(nrows=len(entity_map), ncols=2, figsize=(14, 4 *␣
↪len(entity_map)))
if len(entity_map) == 1:
    axes = [axes]  # unify indexing if only 1 row

########################################################
# 2) LOOP OVER ENTITIES
########################################################

for i, (ent, tables) in enumerate(entity_map.items()):
    stock_table = tables["stock_table"]
    news_table  = tables["news_table"]
    ent_upper   = ent.upper()

    ########################################################
    # 2A) STOCK DATA
    ########################################################
    df_stock = pd.read_sql(f"SELECT Date, Close FROM {stock_table}", engine)
    df_stock["Date"] = pd.to_datetime(df_stock["Date"], format="%Y-%m-%d",␣
↪errors="coerce")
    df_stock.dropna(subset=["Date","Close"], inplace=True)
    df_stock.sort_values("Date", inplace=True)

    ########################################################
    # 2B) NEWS DATA
    ########################################################
    df_news = pd.read_sql(f"SELECT published_date FROM {news_table}", engine)
    df_news["published_date"] = pd.to_datetime(df_news["published_date"],␣
↪errors="coerce")
    df_news.dropna(subset=["published_date"], inplace=True)

    # Group articles by day
    df_news["news_date"] = df_news["published_date"].dt.date
    df_daily_news = df_news.groupby("news_date").size().
↪reset_index(name="daily_articles")
    df_daily_news["news_date"] = pd.to_datetime(df_daily_news["news_date"])

    ########################################################
    # 2C) MERGE ON DATE
    ########################################################
    df_stock.rename(columns={"Date": "stock_date"}, inplace=True)
    df_daily_news.rename(columns={"news_date": "stock_date"}, inplace=True)

    df_merged = pd.merge(df_stock, df_daily_news, on="stock_date", how="inner")
    df_merged.dropna(subset=["stock_date","Close","daily_articles"],␣
↪inplace=True)
    df_merged.sort_values("stock_date", inplace=True)
    df_merged.rename(columns={"stock_date": "Date"}, inplace=True)

    ########################################################
    # 2D) CORRELATION
    ########################################################
    corr_val = df_merged[["daily_articles","Close"]].corr().iloc[0,1]
    print(f"{ent_upper} correlation (daily_articles vs. Close): {corr_val:.3f}")

    ########################################################
    # 2E) PLOTTING SUBPLOTS
    ########################################################
    ax_line    = axes[i][0] if len(entity_map) > 1 else axes[0]
    ax_scatter = axes[i][1] if len(entity_map) > 1 else axes[1]

    # (a) LINE CHART: bar for articles + line for Close on dual y-axis
    df_merged.set_index("Date", inplace=True, drop=False)

    ax_line_2 = ax_line.twinx()  # second y-axis
    ax_line.bar(df_merged.index, df_merged["daily_articles"], width=1,␣
↪color="gray", alpha=0.5)
    ax_line.set_ylabel("Articles", color="gray")
    ax_line.tick_params(axis="y", labelcolor="gray")

    ax_line_2.plot(df_merged.index, df_merged["Close"], color="blue")
    ax_line_2.set_ylabel("Stock Close", color="blue")
    ax_line_2.tick_params(axis="y", labelcolor="blue")

    ax_line.set_title(f"{ent_upper}: Daily Articles vs. Stock Close")

    df_merged.reset_index(drop=True, inplace=True)

    # (b) SCATTER: daily_articles vs. Close
    ax_scatter.scatter(df_merged["daily_articles"], df_merged["Close"], alpha=0.
↪5)
    ax_scatter.set_xlabel("Daily Articles")
    ax_scatter.set_ylabel("Close")
    ax_scatter.set_title(f"{ent_upper}: Scatter (Articles vs. Close)")

########################################################
# 3) SHOW PLOTS
########################################################
plt.tight_layout()
plt.show()
```

12. Analyze News Volume vs. Brent/WTI Close Prices

```
[ ]: import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine

########################################################
# 1) MySQL Connection
########################################################

engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
↪fds_project_db")

# We'll assume:
#   brent_wti_data => columns: (Date, Brent_Close, WTI_Close)
#   brent_news     => table of daily Brent articles (published_date)
#   wti_news       => table of daily WTI articles (published_date)

########################################################
# 2) READ & PREPARE BRENT/WTI DATA
########################################################

df_oil = pd.read_sql("SELECT Date, Brent_Close, WTI_Close FROM brent_wti_data",␣
↪engine)

# Convert Date to datetime
df_oil["Date"] = pd.to_datetime(df_oil["Date"], format="%Y-%m-%d",␣
↪errors="coerce")
df_oil.dropna(subset=["Date"], inplace=True)

# Convert closes to numeric
df_oil["Brent_Close"] = pd.to_numeric(df_oil["Brent_Close"], errors="coerce")
df_oil["WTI_Close"]   = pd.to_numeric(df_oil["WTI_Close"],   errors="coerce")
df_oil.dropna(subset=["Brent_Close","WTI_Close"], how="any", inplace=True)
```

```python
df_oil.sort_values("Date", inplace=True)

########################################################
# 3) FUNCTION TO PREPARE NEWS & MERGE
########################################################
def prepare_news_and_merge(df_oil, close_col, news_table):
    """
    Reads daily news from 'news_table', merges with df_oil on Date,
    focusing on 'close_col' (e.g. 'Brent_Close' or 'WTI_Close'),
    returns a merged DataFrame plus the correlation.
    """
    # Read news
    df_news = pd.read_sql(f"SELECT published_date FROM {news_table}", engine)
    df_news["published_date"] = pd.to_datetime(df_news["published_date"],␣
 ↪errors="coerce")
    df_news.dropna(subset=["published_date"], inplace=True)

    # Convert to daily date, group for article counts
    df_news["news_date"] = df_news["published_date"].dt.date
    df_daily_news = df_news.groupby("news_date").size().
 ↪reset_index(name="daily_articles")
    df_daily_news["news_date"] = pd.to_datetime(df_daily_news["news_date"])

    # Merge with df_oil on 'Date'
    df_temp = df_oil.copy()
    df_temp.rename(columns={"Date": "stock_date"}, inplace=True)
    df_temp.dropna(subset=[close_col], inplace=True)

    df_daily_news.rename(columns={"news_date": "stock_date"}, inplace=True)

    df_merged = pd.merge(
        df_temp[["stock_date", close_col]],
        df_daily_news,
        on="stock_date",
        how="inner"
    )
    df_merged.dropna(subset=["stock_date", close_col, "daily_articles"],␣
 ↪how="any", inplace=True)
    df_merged.sort_values("stock_date", inplace=True)

    # Correlation
    corr_val = df_merged[["daily_articles", close_col]].corr().iloc[0,1]

    # Rename back to 'Date'
    df_merged.rename(columns={"stock_date": "Date"}, inplace=True)

    return df_merged, corr_val
```

```
##########################################################
# 4) PREPARE SUBPLOTS: 2 rows (Brent & WTI), 2 columns (line chart, scatter)
##########################################################
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 8))
# Row 0 = Brent, Row 1 = WTI
# Col 0 = line chart, Col 1 = scatter

##########################################################
# 5) BRENT
##########################################################
df_brent, corr_brent = prepare_news_and_merge(
    df_oil, close_col="Brent_Close", news_table="brent_news"
)
print(f"Correlation (Brent daily_articles vs. Brent_Close): {corr_brent:.3f}")

ax_line_brent = axes[0][0]
ax_scatter_brent = axes[0][1]

# (a) LINE CHART
df_brent.set_index("Date", inplace=True, drop=False)
ax_line_brent_2 = ax_line_brent.twinx()
ax_line_brent.bar(df_brent.index, df_brent["daily_articles"], width=1,␣
 ↪color="gray", alpha=0.5)
ax_line_brent.set_ylabel("Articles", color="gray")
ax_line_brent.tick_params(axis="y", labelcolor="gray")

ax_line_brent_2.plot(df_brent.index, df_brent["Brent_Close"], color="blue")
ax_line_brent_2.set_ylabel("Brent Close", color="blue")
ax_line_brent_2.tick_params(axis="y", labelcolor="blue")

ax_line_brent.set_title("Brent: Daily Articles vs. Brent_Close")
df_brent.reset_index(drop=True, inplace=True)

# (b) SCATTER
ax_scatter_brent.scatter(df_brent["daily_articles"], df_brent["Brent_Close"],␣
 ↪alpha=0.5)
ax_scatter_brent.set_xlabel("Daily Articles")
ax_scatter_brent.set_ylabel("Brent_Close")
ax_scatter_brent.set_title("Brent: Scatter (Articles vs. Close)")

##########################################################
# 6) WTI
##########################################################
df_wti, corr_wti = prepare_news_and_merge(
    df_oil, close_col="WTI_Close", news_table="wti_news"
)
```

```
print(f"Correlation (WTI daily_articles vs. WTI_Close): {corr_wti:.3f}")

ax_line_wti = axes[1][0]
ax_scatter_wti = axes[1][1]

df_wti.set_index("Date", inplace=True, drop=False)
ax_line_wti_2 = ax_line_wti.twinx()
ax_line_wti.bar(df_wti.index, df_wti["daily_articles"], width=1, color="gray",␣
 ↪alpha=0.5)
ax_line_wti.set_ylabel("Articles", color="gray")
ax_line_wti.tick_params(axis="y", labelcolor="gray")

ax_line_wti_2.plot(df_wti.index, df_wti["WTI_Close"], color="blue")
ax_line_wti_2.set_ylabel("WTI Close", color="blue")
ax_line_wti_2.tick_params(axis="y", labelcolor="blue")

ax_line_wti.set_title("WTI: Daily Articles vs. WTI_Close")
df_wti.reset_index(drop=True, inplace=True)

ax_scatter_wti.scatter(df_wti["daily_articles"], df_wti["WTI_Close"], alpha=0.5)
ax_scatter_wti.set_xlabel("Daily Articles")
ax_scatter_wti.set_ylabel("WTI_Close")
ax_scatter_wti.set_title("WTI: Scatter (Articles vs. Close)")

########################################################
# 7) Show
########################################################
plt.tight_layout()
plt.show()
```

13. Cross-Entity Correlation and Monthly Average Analysis (XOM, CVX, BP, Shell, SPY)

```
[ ]: import pandas as pd
     from sqlalchemy import create_engine

     ########################################################
     # 1) MySQL Connection & Entities
     ########################################################

     engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
      ↪fds_project_db")

     # Suppose you have stock tables for XOM, CVX, BP, SHEL, SPY, etc.
     # Each has columns: Date, Close, ...
     entities = ["xom", "cvx", "bp", "shell", "spy"]

     # We'll assume table names like "xom_data", "cvx_data", etc.
     # If they differ, adjust below.
     stock_tables = {ent: f"{ent}_data" for ent in entities}

     ########################################################
     # 2) Loop Over Entities, Read & Merge
     ########################################################

     df_merged = None

     for ent in entities:
         table_name = stock_tables[ent]
         ent_upper  = ent.upper()

         # Read Date, Close from each table
         df_temp = pd.read_sql(f"SELECT Date, Close FROM {table_name}", engine)
         df_temp["Date"] = pd.to_datetime(df_temp["Date"],format="%Y-%m-%d" , ␣
      ↪errors="coerce")
         df_temp.dropna(subset=["Date","Close"], inplace=True)

         # Rename Close -> e.g. XOM_Close, CVX_Close, etc.
         close_col = f"{ent_upper}_Close"
         df_temp.rename(columns={"Close": close_col}, inplace=True)

         # Sort by Date
         df_temp.sort_values("Date", inplace=True)

         # Merge into df_merged
         if df_merged is None:
             # First entity, just set df_merged
             df_merged = df_temp
         else:
             # Outer join on Date, so we keep all dates from all entities
             df_merged = pd.merge(df_merged, df_temp, on="Date", how="outer")

     # After this loop, df_merged has columns:
     # [Date, XOM_Close, CVX_Close, BP_Close, SHEL_Close, SPY_Close, ...] (some may␣
      ↪have NaN if no data)

     ########################################################
     # 3) Cross-Entity Monthly Averages
     ########################################################

     # Convert Date to a monthly period
     df_merged["YearMonth"] = df_merged["Date"].dt.to_period("M")

     # Group by YearMonth, compute average for each entity's Close
     monthly_avg = df_merged.groupby("YearMonth")[
         [f"{e.upper()}_Close" for e in entities]
     ].mean()

     print("\n=== Monthly Averages (first 10 months) ===")
     print(monthly_avg.head(10))

     ########################################################
     # 4) Cross-Entity Correlation
     ########################################################

     # We can compute correlation among the close columns
     # e.g. XOM_Close, CVX_Close, ...
     close_cols = [f"{e.upper()}_Close" for e in entities]

     # Drop rows where all closes are NaN
     df_for_corr = df_merged[close_cols].dropna(how="all")

     corr_matrix = df_for_corr.corr()
     print("\n=== Correlation Matrix (Daily Closes) ===")
     print(corr_matrix)

     ########################################################
     # 5) (Optional) Plot Some Comparison
     ########################################################

     import matplotlib.pyplot as plt

     # Example: line chart of each entity's Close over time (some may have NaN)
     df_plot = df_merged.set_index("Date").sort_index()

     plt.figure(figsize=(10,6))
     for col in close_cols:
         plt.plot(df_plot.index, df_plot[col], label=col)

     plt.title("Cross-Entity Daily Close Comparison")
     plt.legend()
     plt.show()
```

Section 2: SQL Part 2

DESCRIPTIVE STATISTICS XOM

[ ]:
```sql
SELECT
  YEAR(date) AS yr,
  MONTH(Date) AS mn,
  MIN(Close) AS min_close,
  MAX(Close) AS max_close,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM xom_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DESCRIPTIVE STATISTICS SHELL

[ ]:
```sql
SELECT
  YEAR(date) AS yr,
  MONTH(Date) AS mn,
  MIN(Close) AS min_close,
  MAX(Close) AS max_close,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM shel_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DESCRIPTIVE STATISTICS CVX

[ ]:
```sql
SELECT
  YEAR(date) AS yr,
  MONTH(Date) AS mn,
  MIN(Close) AS min_close,
  MAX(Close) AS max_close,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM bp_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DESCRIPTIVE STATISTICS BP

[ ]:
```
SELECT
  YEAR(date) AS yr,
  MONTH(Date) AS mn,
  MIN(Close) AS min_close,
  MAX(Close) AS max_close,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM bp_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DESCRIPTIVE STATISTICS SPY

[ ]:
```
SELECT
  YEAR(date) AS yr,
  MONTH(Date) AS mn,
  MIN(Close) AS min_close,
  MAX(Close) AS max_close,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM spy_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DESCRIPTIVE STATISTICS BRENT - WTI

[ ]:
```
SELECT
  YEAR(Date) AS yr,
  MONTH(Date) AS mn,
  MIN(Brent_Close) AS min_brent,
  MAX(Brent_Close) AS max_brent,
  AVG(Brent_Close) AS avg_brent,
  STDDEV_SAMP(Brent_Close) AS std_brent,
  MIN(WTI_Close) AS min_wti,
  MAX(WTI_Close) AS max_wti,
  AVG(WTI_Close) AS avg_wti,
  STDDEV_SAMP(WTI_Close) AS std_wti
FROM brent_wti_data
WHERE Date NOT IN ('Ticker','Date','None')
  AND Date IS NOT NULL
  AND Brent_Close IS NOT NULL
  AND WTI_Close IS NOT NULL
GROUP BY YEAR(Date), MONTH(Date)
ORDER BY yr, mn;
```

DAILY MOVERS XOM

[ ]:
```
WITH daily_changes AS (
  SELECT
    Date,
    Close,
    LAG(Close) OVER (ORDER BY Date) AS prev_close,
    (Close - LAG(Close) OVER (ORDER BY Date)) AS daily_change
  FROM xom_data
  WHERE Close IS NOT NULL
    AND Date IS NOT NULL
)
SELECT
  Date,
  Close,
  prev_close,
  daily_change
FROM daily_changes
ORDER BY daily_change DESC
LIMIT 10;
```

DAILY MOVERS CVX

[ ]:
```
WITH daily_changes AS (
  SELECT
    Date,
    Close,
    LAG(Close) OVER (ORDER BY Date) AS prev_close,
    (Close - LAG(Close) OVER (ORDER BY Date)) AS daily_change
  FROM cvx_data
  WHERE Close IS NOT NULL
    AND Date IS NOT NULL
)
SELECT
  Date,
  Close,
  prev_close,
  daily_change
FROM daily_changes
ORDER BY daily_change DESC
LIMIT 10;
```

DAILY MOVERS SHELL

[ ]:
```
WITH daily_changes AS (
  SELECT
    Date,
    Close,
    LAG(Close) OVER (ORDER BY Date) AS prev_close,
    (Close - LAG(Close) OVER (ORDER BY Date)) AS daily_change
  FROM shel_data
  WHERE Close IS NOT NULL
    AND Date IS NOT NULL
)
SELECT
  Date,
  Close,
  prev_close,
  daily_change
FROM daily_changes
ORDER BY daily_change DESC
LIMIT 10;
```

DAILY MOVERS BP

[ ]:
```
WITH daily_changes AS (
  SELECT
    Date,
    Close,
    LAG(Close) OVER (ORDER BY Date) AS prev_close,
    (Close - LAG(Close) OVER (ORDER BY Date)) AS daily_change
  FROM bp_data
  WHERE Close IS NOT NULL
    AND Date IS NOT NULL
)
SELECT
  Date,
  Close,
  prev_close,
  daily_change
FROM daily_changes
ORDER BY daily_change DESC
LIMIT 10;
```

DAILY MOVERS SPY

```
[ ]: WITH daily_changes AS (
       SELECT
         Date,
         Close,
         LAG(Close) OVER (ORDER BY Date) AS prev_close,
         (Close - LAG(Close) OVER (ORDER BY Date)) AS daily_change
       FROM spy_data
       WHERE Close IS NOT NULL
         AND Date IS NOT NULL
     )
     SELECT
       Date,
       Close,
       prev_close,
       daily_change
     FROM daily_changes
     ORDER BY daily_change DESC
     LIMIT 10;
```

DAILY MOVERS BRENT

```
[ ]: WITH daily_changes AS (
       SELECT
         Date,
         Brent_Close,
         LAG(Brent_Close) OVER (ORDER BY Date) AS prev_brent,
         (Brent_Close - LAG(Brent_Close) OVER (ORDER BY Date)) AS daily_change
       FROM brent_wti_data
       WHERE Brent_Close IS NOT NULL
         AND Date IS NOT NULL
     )
     SELECT
       Date,
       Brent_Close,
       prev_brent,
       daily_change
     FROM daily_changes
     ORDER BY daily_change DESC
     LIMIT 10;
```

DAILY MOVERS WTI

```
[ ]: WITH daily_changes AS (
       SELECT
         Date,
         WTI_Close,
         LAG(WTI_Close) OVER (ORDER BY Date) AS prev_wti,
         (WTI_Close - LAG(WTI_Close) OVER (ORDER BY Date)) AS daily_change
       FROM brent_wti_data
       WHERE WTI_Close IS NOT NULL
         AND Date IS NOT NULL
     )
     SELECT
       Date,
       WTI_Close,
       prev_wti,
       daily_change
     FROM daily_changes
     ORDER BY daily_change DESC
     LIMIT 10;
```

CORRELATION CLOSE VS VOLUME XOM

[ ]:
```
WITH base AS (
  SELECT
    Close AS x,
    Volume AS y
  FROM xom_data
  WHERE Close IS NOT NULL
    AND Volume IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*) AS n,
    SUM(x) AS sum_x,
    SUM(y) AS sum_y,
    SUM(x*x) AS sum_x2,
    SUM(y*y) AS sum_y2,
    SUM(x*y) AS sum_xy
  FROM base
)
SELECT
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_close_volume
FROM calc;
```

CORRELATION CLOSE VS VOLUME CVX

[ ]:
```
WITH base AS (
  SELECT
    Close AS x,
    Volume AS y
  FROM cvx_data
  WHERE Close IS NOT NULL
    AND Volume IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*) AS n,
    SUM(x) AS sum_x,
    SUM(y) AS sum_y,
    SUM(x*x) AS sum_x2,
    SUM(y*y) AS sum_y2,
    SUM(x*y) AS sum_xy
  FROM base
)
SELECT
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_close_volume
FROM calc;
```

CORRELATION CLOSE VS VOLUME SHELL

[ ]:
```
WITH base AS (
  SELECT
    Close AS x,
    Volume AS y
  FROM shel_data
  WHERE Close IS NOT NULL
    AND Volume IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*) AS n,
    SUM(x) AS sum_x,
    SUM(y) AS sum_y,
    SUM(x*x) AS sum_x2,
    SUM(y*y) AS sum_y2,
    SUM(x*y) AS sum_xy
  FROM base
)
SELECT
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_close_volume
FROM calc;
```

CORRELATION CLOSE VS VOLUME BP

[ ]:
```
WITH base AS (
  SELECT
    Close AS x,
    Volume AS y
  FROM bp_data
  WHERE Close IS NOT NULL
    AND Volume IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*) AS n,
    SUM(x) AS sum_x,
    SUM(y) AS sum_y,
    SUM(x*x) AS sum_x2,
    SUM(y*y) AS sum_y2,
    SUM(x*y) AS sum_xy
  FROM base
)
SELECT
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_close_volume
FROM calc;
```

CORRELATION CLOSE VS VOLUME SPY

[ ]:
```
WITH base AS (
  SELECT
    Close AS x,
    Volume AS y
  FROM spy_data
  WHERE Close IS NOT NULL
    AND Volume IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*) AS n,
    SUM(x) AS sum_x,
    SUM(y) AS sum_y,
    SUM(x*x) AS sum_x2,
    SUM(y*y) AS sum_y2,
    SUM(x*y) AS sum_xy
  FROM base
)
SELECT
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_close_volume
FROM calc;
```

CORRELATION CLOSE BRENT-WTI

[ ]:
```
WITH base AS (
  SELECT
    WTI_Close AS x,
    Brent_Close AS y
  FROM brent_wti_data
  WHERE WTI_Close IS NOT NULL
    AND Brent_Close IS NOT NULL
),
calc AS (
  SELECT
    COUNT(*)      AS n,
    SUM(x)        AS sum_x,
    SUM(y)        AS sum_y,
    SUM(x*x)      AS sum_x2,
    SUM(y*y)      AS sum_y2,
    SUM(x*y)      AS sum_xy
  FROM base
)
SELECT
```

```
  (
    (calc.n * calc.sum_xy) - (calc.sum_x * calc.sum_y)
  ) / (
    SQRT(
      (calc.n * calc.sum_x2 - (calc.sum_x * calc.sum_x)) *
      (calc.n * calc.sum_y2 - (calc.sum_y * calc.sum_y))
    )
  ) AS corr_wti_brent
FROM calc;
```

Temporary news Table creation XOM

```
[ ]: -- 1A) Aggregate daily articles
CREATE TEMPORARY TABLE xom_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM xom_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

Temporary news Table creation CVX

```
[ ]: -- 1A) Aggregate daily articles
CREATE TEMPORARY TABLE cvx_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM chevron_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

Temporary news Table creation SHELL

```
[ ]: -- 1A) Aggregate daily articles
CREATE TEMPORARY TABLE shell_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM shell_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

Temporary news Table creation BP

```
[ ]: -- 1A) Aggregate daily articles
CREATE TEMPORARY TABLE bp_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM bp_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

Temporary news Table creation SPY

[ ]:
```
-- 1A) Aggregate daily articles
CREATE TEMPORARY TABLE spy_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM spy_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

Temporary news Table creation BRENT WTI

[ ]:
```
CREATE TEMPORARY TABLE brent_news_daily AS
SELECT
  DATE(published_date) AS news_day,
  COUNT(*) AS daily_articles
FROM brent_news
WHERE published_date IS NOT NULL
GROUP BY DATE(published_date);
```

NEWS ARTICLES VS STOCK XOM

[ ]:
```
SELECT
  s.Date,
  s.Close,
  n.daily_articles
FROM xom_data s
JOIN xom_news_daily n
   ON s.Date = n.news_day
WHERE s.Date IS NOT NULL
  AND s.Close IS NOT NULL
ORDER BY s.Date;
```

NEWS ARTICLES VS STOCK CVX

[ ]:
```
SELECT
  s.Date,
  s.Close,
  n.daily_articles
FROM cvx_data s
JOIN cvx_news_daily n
   ON s.Date = n.news_day
WHERE s.Date IS NOT NULL
```

```
  AND s.Close IS NOT NULL
ORDER BY s.Date;
```

NEWS ARTICLES VS STOCK SHELL

[ ]:
```
SELECT
  s.Date,
  s.Close,
  n.daily_articles
FROM shel_data s
JOIN shell_news_daily n
   ON s.Date = n.news_day
WHERE s.Date IS NOT NULL
  AND s.Close IS NOT NULL
ORDER BY s.Date;
```

NEWS ARTICLES VS STOCK BP

[ ]:
```
SELECT
  s.Date,
  s.Close,
  n.daily_articles
FROM bp_data s
JOIN bp_news_daily n
   ON s.Date = n.news_day
WHERE s.Date IS NOT NULL
  AND s.Close IS NOT NULL
ORDER BY s.Date;
```

NEWS ARTICLES VS STOCK SPY

[ ]:
```
SELECT
  s.Date,
  s.Close,
  n.daily_articles
FROM spy_data s
JOIN spy_news_daily n
   ON s.Date = n.news_day
WHERE s.Date IS NOT NULL
  AND s.Close IS NOT NULL
ORDER BY s.Date;
```

NEWS ARTICLES VS STOCK BRENT

[ ]:
```
SELECT
  b.Date,
  b.Brent_Close,
  n.daily_articles
FROM brent_wti_data b
```

```
JOIN brent_news_daily n
   ON b.Date = n.news_day
WHERE b.Date IS NOT NULL
  AND b.Brent_Close IS NOT NULL
ORDER BY b.Date;
```

NEWS ARTICLES VS STOCK WTI

```
[ ]: SELECT
       b.Date,
       b.WTI_Close,
       n.daily_articles
     FROM brent_wti_data b
     JOIN wti_news_daily n
        ON b.Date = n.news_day
     WHERE b.Date IS NOT NULL
       AND b.WTI_Close IS NOT NULL
     ORDER BY b.Date;
```

YEAR OVER YEAR XOM

```
[ ]: WITH monthly_data AS (
       SELECT
         YEAR(Date) AS yr,
         MONTH(Date) AS mn,
         AVG(Close) AS avg_close
       FROM xom_data
       WHERE Close IS NOT NULL
       GROUP BY YEAR(Date), MONTH(Date)
     ),
     monthly_yoy AS (
       SELECT
         d.yr,
         d.mn,
         d.avg_close,
         LAG(d.avg_close, 1) OVER (PARTITION BY d.mn ORDER BY d.yr) AS␣
       ↪last_year_close
       FROM monthly_data d
     )
     SELECT
       yr,
       mn,
       avg_close,
       last_year_close,
       (avg_close - last_year_close) AS yoy_diff,
       ( (avg_close - last_year_close) / last_year_close ) * 100 AS yoy_pct_change
     FROM monthly_yoy
     ORDER BY yr, mn;
```

YEAR OVER YEAR CVX

```
[ ]: WITH monthly_data AS (
       SELECT
         YEAR(Date) AS yr,
         MONTH(Date) AS mn,
         AVG(Close) AS avg_close
       FROM cvx_data
       WHERE Close IS NOT NULL
       GROUP BY YEAR(Date), MONTH(Date)
     ),
     monthly_yoy AS (
       SELECT
         d.yr,
         d.mn,
         d.avg_close,
         LAG(d.avg_close, 1) OVER (PARTITION BY d.mn ORDER BY d.yr) AS␣
      ↪last_year_close
       FROM monthly_data d
     )
     SELECT
       yr,
       mn,
       avg_close,
       last_year_close,
       (avg_close - last_year_close) AS yoy_diff,
       ( (avg_close - last_year_close) / last_year_close ) * 100 AS yoy_pct_change
     FROM monthly_yoy
     ORDER BY yr, mn;
```

YEAR OVER YEAR SHELL

```
[ ]: WITH monthly_data AS (
       SELECT
         YEAR(Date) AS yr,
         MONTH(Date) AS mn,
         AVG(Close) AS avg_close
       FROM shel_data
       WHERE Close IS NOT NULL
       GROUP BY YEAR(Date), MONTH(Date)
     ),
     monthly_yoy AS (
       SELECT
         d.yr,
         d.mn,
         d.avg_close,
         LAG(d.avg_close, 1) OVER (PARTITION BY d.mn ORDER BY d.yr) AS␣
      ↪last_year_close
```

```
  FROM monthly_data d
)
SELECT
  yr,
  mn,
  avg_close,
  last_year_close,
  (avg_close - last_year_close) AS yoy_diff,
  ( (avg_close - last_year_close) / last_year_close ) * 100 AS yoy_pct_change
FROM monthly_yoy
ORDER BY yr, mn;
```

YEAR OVER YEAR BP

[ ]:
```
WITH monthly_data AS (
  SELECT
    YEAR(Date) AS yr,
    MONTH(Date) AS mn,
    AVG(Close) AS avg_close
  FROM bp_data
  WHERE Close IS NOT NULL
  GROUP BY YEAR(Date), MONTH(Date)
),
monthly_yoy AS (
  SELECT
    d.yr,
    d.mn,
    d.avg_close,
    LAG(d.avg_close, 1) OVER (PARTITION BY d.mn ORDER BY d.yr) AS␣
  ↪last_year_close
  FROM monthly_data d
)
SELECT
  yr,
  mn,
  avg_close,
  last_year_close,
  (avg_close - last_year_close) AS yoy_diff,
  ( (avg_close - last_year_close) / last_year_close ) * 100 AS yoy_pct_change
FROM monthly_yoy
ORDER BY yr, mn;
```

YEAR OVER YEAR SPY

[ ]:
```
WITH monthly_data AS (
  SELECT
    YEAR(Date) AS yr,
    MONTH(Date) AS mn,
    AVG(Close) AS avg_close
  FROM spy_data
  WHERE Close IS NOT NULL
  GROUP BY YEAR(Date), MONTH(Date)
),
monthly_yoy AS (
  SELECT
    d.yr,
    d.mn,
    d.avg_close,
    LAG(d.avg_close, 1) OVER (PARTITION BY d.mn ORDER BY d.yr) AS␣
  ↪last_year_close
  FROM monthly_data d
)
SELECT
  yr,
  mn,
  avg_close,
  last_year_close,
  (avg_close - last_year_close) AS yoy_diff,
  ( (avg_close - last_year_close) / last_year_close ) * 100 AS yoy_pct_change
FROM monthly_yoy
ORDER BY yr, mn;
```

YEAR OVER YEAR BRENT

[ ]:
```
WITH monthly_data_brent AS (
  SELECT
    YEAR(Date) AS yr,
    MONTH(Date) AS mn,
    AVG(Brent_Close) AS avg_brent
  FROM brent_wti_data
  WHERE Brent_Close IS NOT NULL
  GROUP BY YEAR(Date), MONTH(Date)
),
monthly_yoy_brent AS (
  SELECT
    d.yr,
    d.mn,
    d.avg_brent,
    LAG(d.avg_brent, 1) OVER (
      PARTITION BY d.mn
      ORDER BY d.yr
    ) AS last_year_brent
  FROM monthly_data_brent d
)
SELECT
```

```
  yr,
  mn,
  avg_brent,
  last_year_brent,
  (avg_brent - last_year_brent) AS yoy_diff,
  CASE
    WHEN last_year_brent IS NOT NULL AND last_year_brent <> 0 THEN
      ( (avg_brent - last_year_brent) / last_year_brent ) * 100
    ELSE NULL
  END AS yoy_pct_change
FROM monthly_yoy_brent
ORDER BY yr, mn;
```

YEAR OVER YEAR WTI

[ ]:
```
WITH monthly_data_wti AS (
  SELECT
    YEAR(Date) AS yr,
    MONTH(Date) AS mn,
    AVG(WTI_Close) AS avg_wti
  FROM brent_wti_data
  WHERE WTI_Close IS NOT NULL
  GROUP BY YEAR(Date), MONTH(Date)
),
monthly_yoy_wti AS (
  SELECT
    d.yr,
    d.mn,
    d.avg_wti,
    LAG(d.avg_wti, 1) OVER (
      PARTITION BY d.mn
      ORDER BY d.yr
    ) AS last_year_wti
  FROM monthly_data_wti d
)
SELECT
  yr,
  mn,
  avg_wti,
  last_year_wti,
  (avg_wti - last_year_wti) AS yoy_diff,
  CASE
    WHEN last_year_wti IS NOT NULL AND last_year_wti <> 0 THEN
      ( (avg_wti - last_year_wti) / last_year_wti ) * 100
    ELSE NULL
  END AS yoy_pct_change
FROM monthly_yoy_wti
```

```
ORDER BY yr, mn;
```

DAY OF WEEK ANALYSIS XOM

[ ]:
```
SELECT
  DAYNAME(Date) AS day_of_week,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM xom_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYNAME(Date)
ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS CVX

[ ]:
```
SELECT
  DAYNAME(Date) AS day_of_week,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM cvx_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYNAME(Date)
ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS SHELL

[ ]:
```
SELECT
  DAYNAME(Date) AS day_of_week,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM shel_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYNAME(Date)
ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS BP

[ ]:
```
SELECT
  DAYNAME(Date) AS day_of_week,
  AVG(Close) AS avg_close,
  STDDEV_SAMP(Close) AS std_close
FROM bp_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYNAME(Date)
ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS SPY

```
[ ]: SELECT
       DAYNAME(Date) AS day_of_week,
       AVG(Close) AS avg_close,
       STDDEV_SAMP(Close) AS std_close
     FROM spy_data
     WHERE Date IS NOT NULL
       AND Close IS NOT NULL
     GROUP BY DAYNAME(Date)
     ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS BRENT

```
[ ]: SELECT
       DAYNAME(Date) AS day_of_week,
       AVG(Brent_Close) AS avg_close,
       STDDEV_SAMP(Brent_Close) AS std_close
     FROM brent_wti_data
     WHERE Date IS NOT NULL
       AND Brent_Close IS NOT NULL
     GROUP BY DAYNAME(Date)
     ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF WEEK ANALYSIS WTI

```
[ ]: SELECT
       DAYNAME(Date) AS day_of_week,
       AVG(WTI_Close) AS avg_close,
       STDDEV_SAMP(WTI_Close) AS std_close
     FROM brent_wti_data
     WHERE Date IS NOT NULL
       AND WTI_Close IS NOT NULL
     GROUP BY DAYNAME(Date)
     ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday');
```

DAY OF MONTH XOM

```
[ ]: SELECT
       DAYOFMONTH(Date) AS dom,
       AVG(Close) AS avg_close
     FROM xom_data
     WHERE Date IS NOT NULL
       AND Close IS NOT NULL
     GROUP BY DAYOFMONTH(Date)
     ORDER BY dom;
```

DAY OF MONTH CVX

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(Close) AS avg_close
FROM cvx_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

DAY OF MONTH SHELL

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(Close) AS avg_close
FROM shel_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

DAY OF MONTH BP

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(Close) AS avg_close
FROM bp_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

DAY OF MONTH SPY

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(Close) AS avg_close
FROM spy_data
WHERE Date IS NOT NULL
  AND Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

DAY OF MONTH BRENT

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(Brent_Close) AS avg_close
FROM brent_wti_data
WHERE Date IS NOT NULL
```

```
  AND Brent_Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

DAY OF MONTH WTI

[ ]:
```
SELECT
  DAYOFMONTH(Date) AS dom,
  AVG(WTI_Close) AS avg_close
FROM brent_wti_data
WHERE Date IS NOT NULL
  AND WTI_Close IS NOT NULL
GROUP BY DAYOFMONTH(Date)
ORDER BY dom;
```

Section 3: Data Cleaning & Checking

[ ]:
```python
import os
import pandas as pd

FDS_FOLDER = "FDS project"

# 1) Define the CSV files for each entity
#    Key = Entity name, Value = list of CSV filenames
entities_files = {
    "XOM": [
        "XOM_News_2019.csv",
        "XOM_News_2020.csv",
        "XOM_News_2021.csv",
        "XOM_News_2022.csv",
        "XOM_News_2023.csv",
        "XOM_News_2024.csv",
        "XOM_News_2025.csv"
    ],
    "Chevron": [
        "Chevron_News_2019.csv",
        "Chevron_News_2020.csv",
        "Chevron_News_2021.csv",
        "Chevron_News_2022.csv",
        "Chevron_News_2023.csv",
        "Chevron_News_2024.csv",
        "Chevron_News_2025.csv"
    ],
    "Shell": [
        "Shell_News_2019.csv",
        "Shell_News_2020.csv",
        "Shell_News_2021.csv",
        "Shell_News_2022.csv",
        "Shell_News_2023.csv",
        "Shell_News_2024.csv",
        "Shell_News_2025.csv"
    ],
    "BP": [
        "BP_News_2019.csv",
        "BP_News_2020.csv",
        "BP_News_2021.csv",
        "BP_News_2022.csv",
        "BP_News_2023.csv",
        "BP_News_2024.csv",
        "BP_News_2025.csv"
    ],
    "WTI": [
        "WTI_News_2019.csv",
        "WTI_News_2020.csv",
        "WTI_News_2021.csv",
        "WTI_News_2022.csv",
        "WTI_News_2023.csv",
        "WTI_News_2024.csv",
        "WTI_News_2025.csv"
    ],
    "Brent": [
        "Brent_News_2019.csv",
        "Brent_News_2020.csv",
        "Brent_News_2021.csv",
        "Brent_News_2022.csv",
        "Brent_News_2023.csv",
        "Brent_News_2024.csv",
        "Brent_News_2025.csv"
    ],
    "SPY": [
        "SPY_News_2019.csv",
        "SPY_News_2020.csv",
        "SPY_News_2021.csv",
        "SPY_News_2022.csv",
        "SPY_News_2023.csv",
        "SPY_News_2024.csv",
        "SPY_News_2025.csv"
    ]
}

# 2) A helper function to consolidate & clean files for one entity
def consolidate_and_clean(entity, file_list):
    """
    Reads all CSVs in file_list, concatenates them, removes missing &␣
 ↪duplicates,
    parses date strings, reformats to DD:MM:YYYY, and saves a cleaned file.
    Returns stats in a dictionary.
    """
    all_dfs = []
    # Read each CSV for this entity
    for filename in file_list:
        csv_path = os.path.join(FDS_FOLDER, filename)
        if os.path.exists(csv_path):
            print(f"[{entity}] Reading file: {csv_path}")
            df_temp = pd.read_csv(csv_path)
            all_dfs.append(df_temp)
        else:
            print(f"[{entity}] File not found: {csv_path} (skipping)")

    if not all_dfs:
        # No files found for this entity
        return {
            "Entity": entity,
            "Initial": 0,
            "AfterMissing": 0,
            "AfterDup": 0
        }

    # Concatenate
    df_combined = pd.concat(all_dfs, ignore_index=True)
    initial_count = len(df_combined)

    # Drop rows missing crucial columns (title, summary, link)
    df_combined.dropna(subset=["title", "summary", "link"], inplace=True)
    after_missing_count = len(df_combined)

    # Remove duplicates by (title, link)
    dup_count = df_combined.duplicated(subset=["title", "link"]).sum()
    if dup_count > 0:
        df_combined.drop_duplicates(subset=["title", "link"], inplace=True)
    after_dup_count = len(df_combined)

    # --- NEW PART: Parse and reformat the date column ---
    # We'll assume the column is named "published_date".
    # 1) Convert to datetime
    df_combined["published_date"] = pd.to_datetime(
        df_combined["published_date"],
        errors="coerce"
    )
    # 2) Reformat to DD:MM:YYYY
    #    (This will produce strings like "20:03:2020")
    df_combined["published_date"] = df_combined["published_date"].dt.
↪strftime("%Y-%m-%d")

    # Save cleaned file
    output_filename = f"{entity}_News_cleaned.csv"
    output_path = os.path.join(FDS_FOLDER, output_filename)
    df_combined.to_csv(output_path, index=False)
    print(f"[{entity}] Cleaned file saved to: {output_path}")

    # Return stats
    return {
        "Entity": entity,
        "Initial": initial_count,
        "AfterMissing": after_missing_count,
        "AfterDup": after_dup_count
    }

# 3) Consolidate and clean for each entity, collect stats
stats_list = []
for entity, file_list in entities_files.items():
    stats = consolidate_and_clean(entity, file_list)
    stats_list.append(stats)

# 4) Create a summary table (DataFrame) of the stats
df_stats = pd.DataFrame(stats_list, columns=["Entity", "Initial",␣
 ↪"AfterMissing", "AfterDup"])
print("\n=== Summary Table ===")
print(df_stats)

# 5) (Optional) Save the summary table to a CSV
summary_output = os.path.join(FDS_FOLDER, "News_Consolidation_Summary.csv")
df_stats.to_csv(summary_output, index=False)
print(f"\nSummary table saved to: {summary_output}")
```

```
[ ]: import os
import pandas as pd

# Folder with your CSV files
FDS_FOLDER = "FDS project"

# CSV files to process
csv_files = [
    "XOM_data.csv",
    "CVX_data.csv",
    "BP_data.csv",
    "SHEL_data.csv",
    "SPY_data.csv"
]

# Final column names in the exact order we want
final_columns = ["Date", "Close", "High", "Low", "Open", "Volume"]

for filename in csv_files:
    input_path = os.path.join(FDS_FOLDER, filename)
```

```python
    print(f"Processing {input_path}...")

    # 1) Read CSV, skip first 2 lines (unwanted headers)
    #    so line 3 becomes data
    # 2) Use header=None so we can rename columns ourselves
    df = pd.read_csv(input_path, skiprows=2, header=None)

    # 3) Force the columns to the final names
    #    Make sure the CSV after skipping lines has exactly 6 columns
    df.columns = final_columns

    # 4) Build output filename (e.g. "XOM_data_cleaned.csv")
    output_filename = filename.replace(".csv", "_cleaned.csv")
    output_path = os.path.join(FDS_FOLDER, output_filename)

    # 5) Save the cleaned CSV
    df.to_csv(output_path, index=False)

    print(f"Renamed columns to: {final_columns}")
    print(f"Saved cleaned file to: {output_path}\n")
```