Academic submission

FIN42110 Data Science for Trading & Risk Management

UCD Michael Smurfit Graduate Business School

Executive Report

Oil Price Volatility and Market Sentiment: A Data Science Approach to Energy Stock Analysis (2019–2025)

Section 4 - 5

PREPARED BY:

Varsha Chandrashekar Balaji (24200924)

Purva Sharma (24214393)

Group Number: 2

Table of Contents **Page No.**

1 Executive Summary……. 3

2 Section 4: Data Visualisation…….…….. 3

2.1 Understanding Market Risk: Daily Return Distributions…… 3

2.2 Smoothed Trends: Rolling and Monthly Averages……...…….. 4

2.3 Brent Oil as a Price Driver for Energy Stocks…….……... 5

2.4 News Volume and Return Volatility……...…… 5

2.5 Correlation Heatmaps: Sector Sync and Market Movers…… 6

2.6 Calendar Patterns: Is Timing Everything? ..……... 7

2.7 Returns vs Volume/Volatility……. 8

2.8 How Closely Do All Entities Move Together?…….. 8

2.9 Moving Averages and Bollinger Bands……. 9

2.10 Granger Causality and Rolling Correlation…….. 9

3 Section 5: Textual Analysis……..... 10

3.1 Sentiment Analysis and Topic Modeling……... 10

3.2 Daily Sentiment Time Series……. 11

3.3 Sentiment vs Weekly Returns……..…….. 11

3.4 OLS Regression of Sentiment vs Returns…..…… 13

3.5 Rolling Correlation: Sentiment vs Returns…… 14

3.6 Sentiment-Based Strategy vs Buy & Hold…… 14

3.7 Lagged Correlation and Mean Sentiment Summary……. 15

4 Conclusion……...…….. 16

5 References …….…..... 16

6 Acknowledgement……. 17

Appendix A…… (Continued)

Appendix B…… (Continued)

Appendix C…… (Continued)

Appendix D…… (Continued)

1. Executive Summary

This report investigates the impact of oil price volatility and news sentiment on the performance of major energy stocks and market indices from 2019 to 2025. Using a data science approach, we combined financial time series analysis with natural language processing (NLP) to uncover how structured price data and unstructured news content interact in real-world markets.

Key insights include:

- **Brent crude oil** shows strong price co-movement with upstream energy stocks like ExxonMobil (XOM) and Chevron (CVX), confirming its role as a key pricing benchmark for the sector.
- **News volume surges** often align with periods of high return volatility, indicating investor sensitivity to headline activity during market events like COVID-19 and the 2022 energy crisis.
- While **sentiment scores** alone showed limited predictive power, they added value when used alongside traditional indicators — especially for certain entities like SHELL and BP.
- A **simple sentiment-based trading strategy** outperformed buy-and-hold in a few cases but was less effective for broader indices and commodity assets.

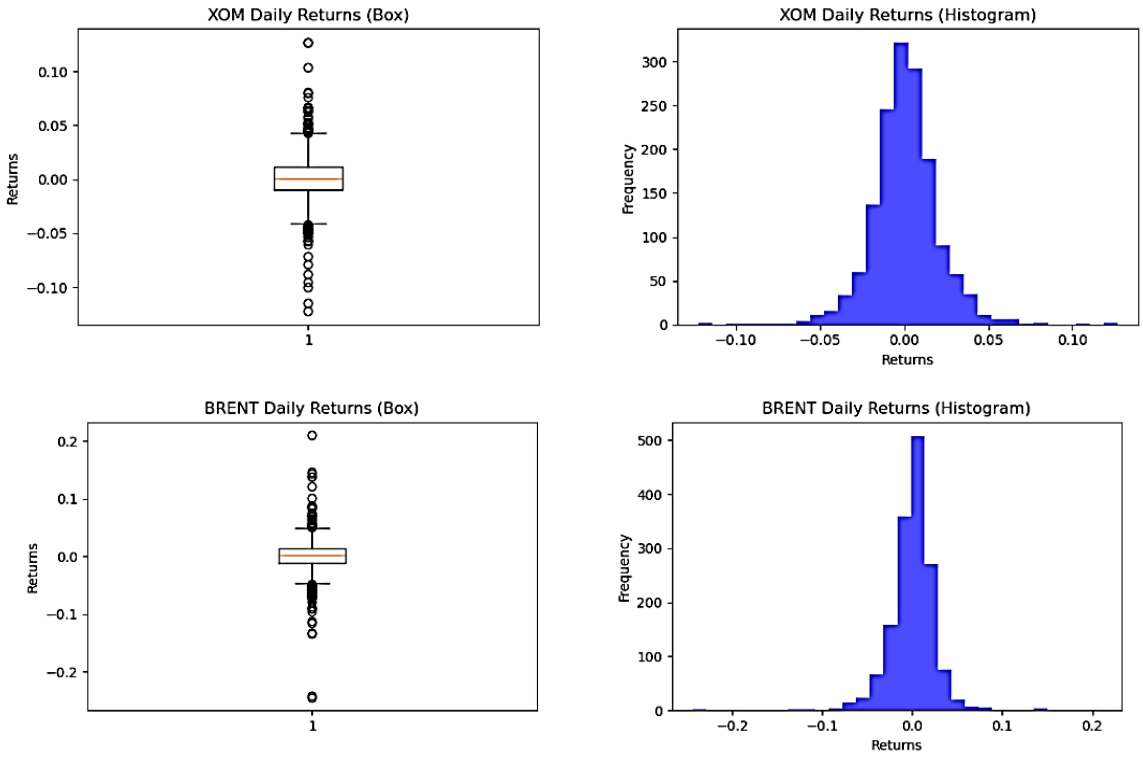
These findings suggest that combining market data with sentiment signals can enhance risk awareness and support more responsive investment strategies — particularly in sectors exposed to geopolitical, economic, or media-driven shocks.

2. Section 4: Data Visualisation

This section explores how price returns and market activity evolved over time and how they relate to news, volume, and macro events. We focus on patterns that help investors understand risks, correlations, and potential signals from oil and stock markets.

2.1 Understanding Market Risk: Daily Return Distributions

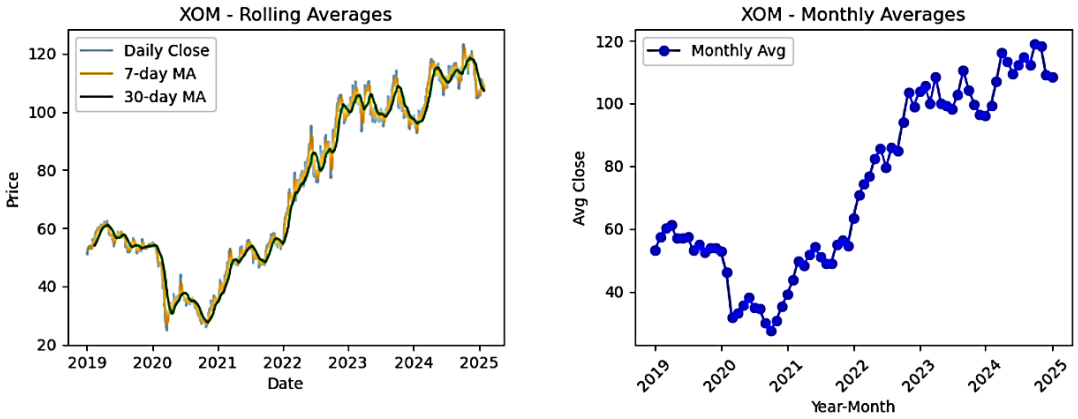
Histograms and boxplots of daily returns show that SPY and CVX have relatively stable returns, while commodities like WTI and Brent are highly volatile with extreme outliers , especially during the 2020 oil crash. The fat tails and skewed distributions in these assets indicate a greater risk of large price swings, which investors should account for in risk models.

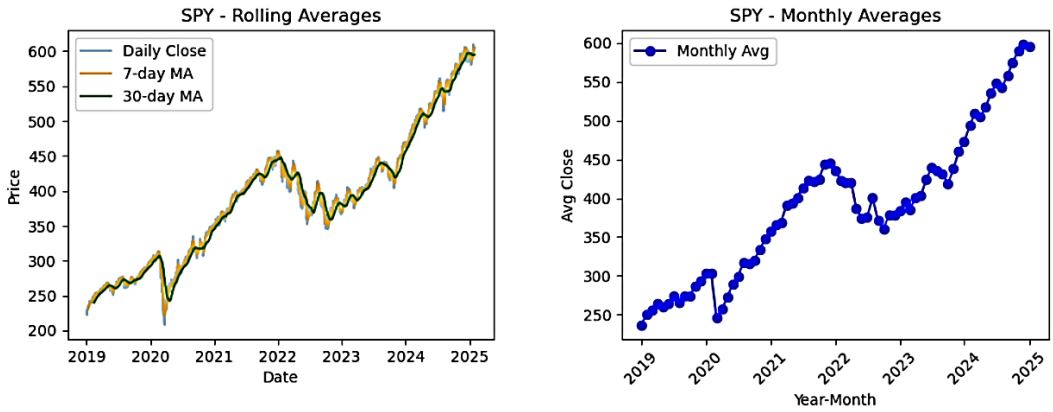


(Full visualizations for all entities are available in Appendix C.)

2.2 Smoothed Trends: Rolling and Monthly Averages

Rolling averages (7-day and 30-day) and monthly averages help us observe medium-term trends. A strong post-2021 recovery is seen across SPY, XOM, and CVX, while Brent and WTI also exhibit sharp rebounds after the 2020 crash. These smoothed trends reveal how quickly energy markets can shift , valuable for timing investment decisions or identifying turning points.

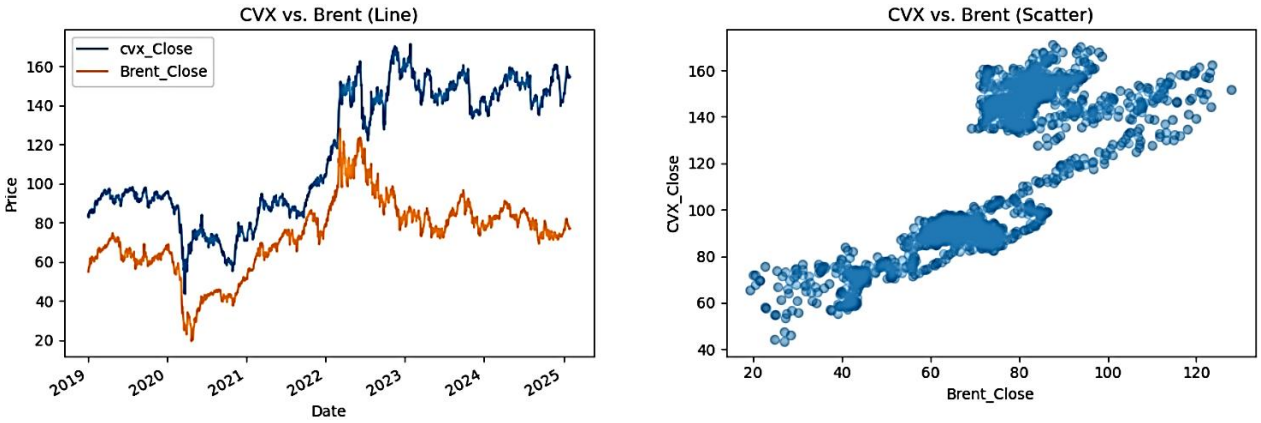




(Full visualizations for all entities are available in Appendix C.)

2.3 Brent Oil as a Price Driver for Energy Stocks

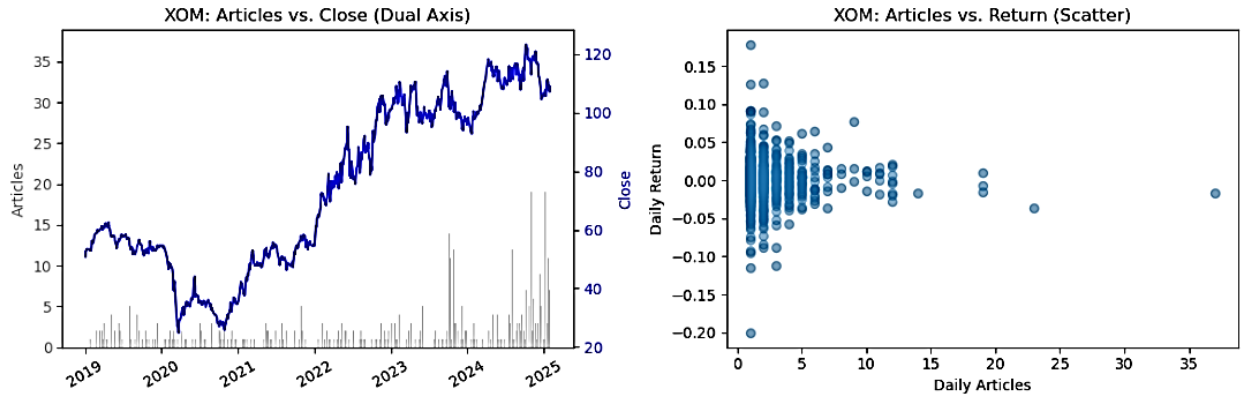
We observe strong co-movement between Brent prices and upstream oil firms (XOM, CVX, BP, SHELL). Scatter plots highlight a strong positive relationship, showing Brent's influence on their pricing. In contrast, SPY's weaker correlation reinforces its broader sector exposure. Brent can thus be used as a signal for trading or hedging energy stocks.



(Full visualizations for all entities are available in Appendix C.)

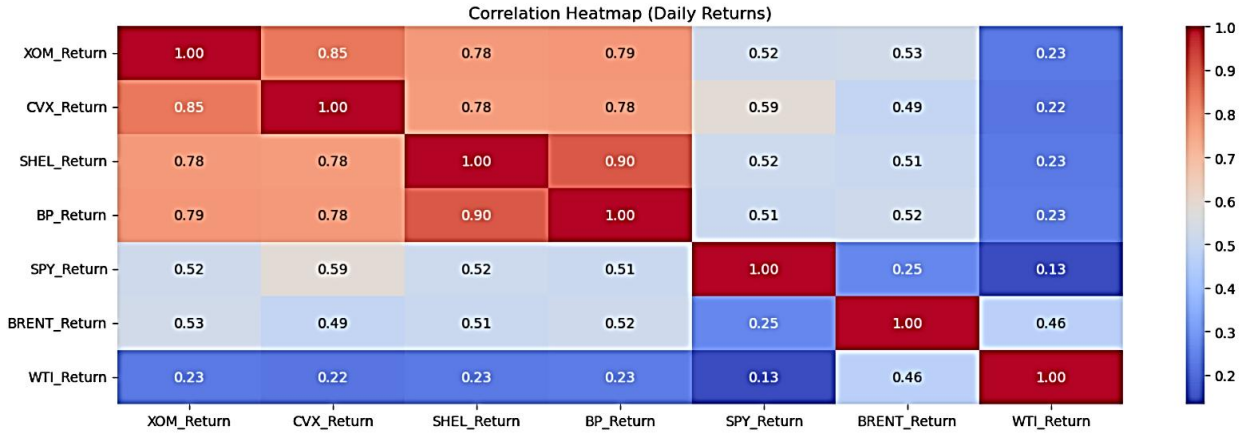
2.4 News Volume and Return Volatility

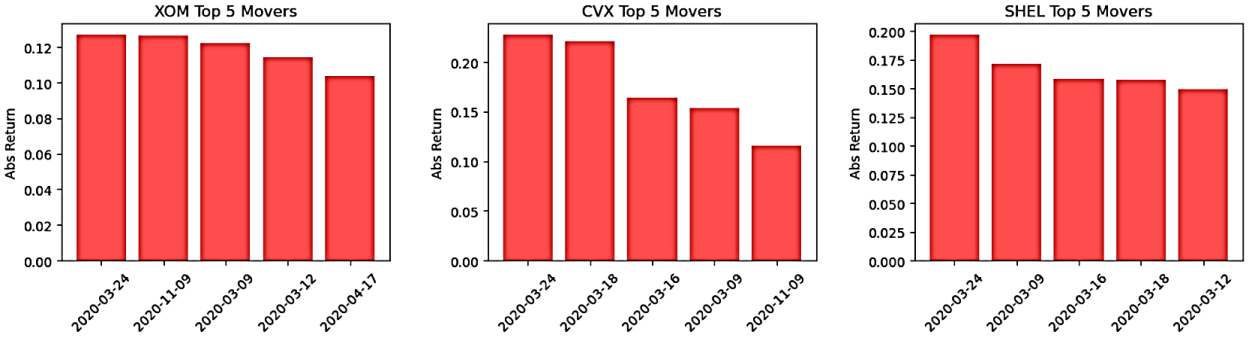
We find that spikes in news coverage align with high-return volatility , especially during crises (e.g., March 2020 or early 2022). Scatter plots confirm that higher article counts correspond with wider return ranges. This suggests news volume can be a useful early warning signal for market stress or investor overreaction.

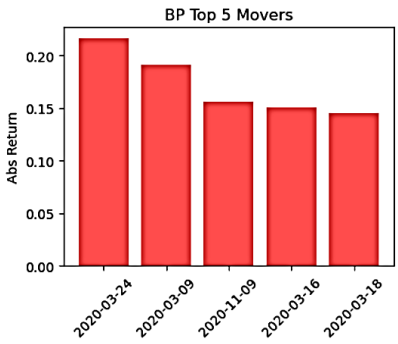


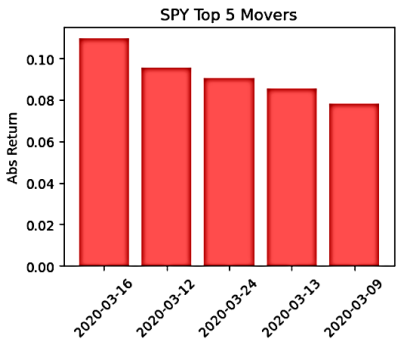
2.5 Correlation Heatmaps: Sector Sync and Market Movers

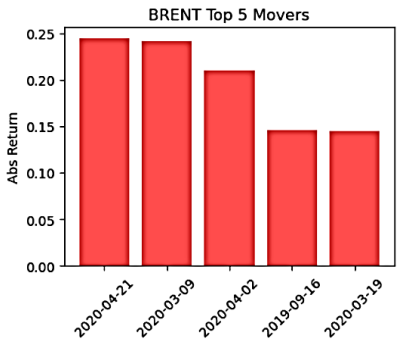
Daily return correlations show strong relationships among oil stocks (e.g., XOM–CVX > 0.75), reflecting sector synchronization. Correlations with Brent and SPY are lower, indicating independent price dynamics. Top mover charts highlight how major shocks, like COVID-19, caused sector-wide disruption, WTI even showed +300% daily return due to the oil futures collapse.

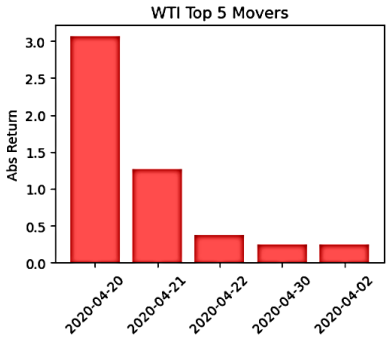






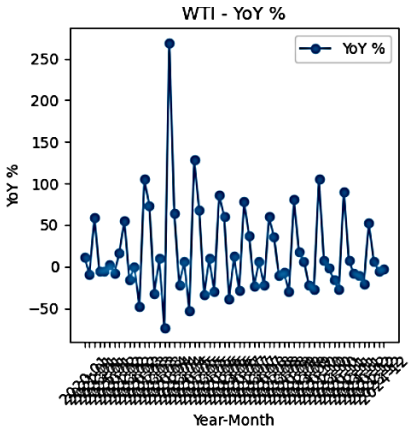


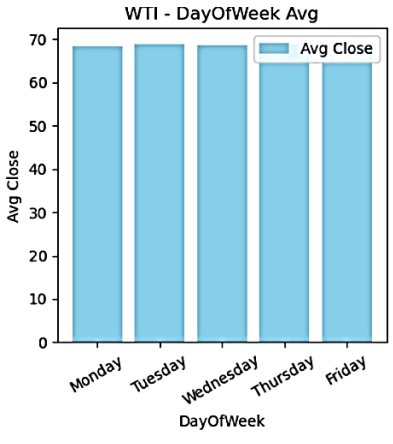


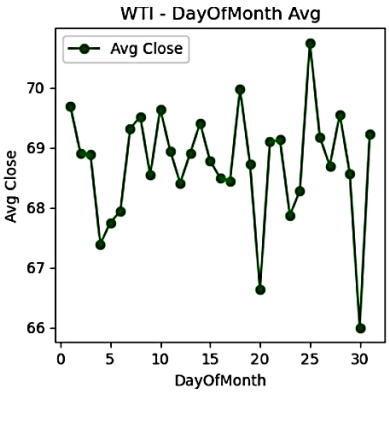


2.6 Calendar Patterns: Is Timing Everything?

Year-over-year comparisons and day-of-week/month effects show minor seasonality. WTI and Brent exhibited the highest yearly swings in 2020–2021, but weekday patterns were mostly flat. Slight month-end return bumps suggest investor positioning activity but are not statistically strong, implying these markets are more event-driven than calendar-driven.



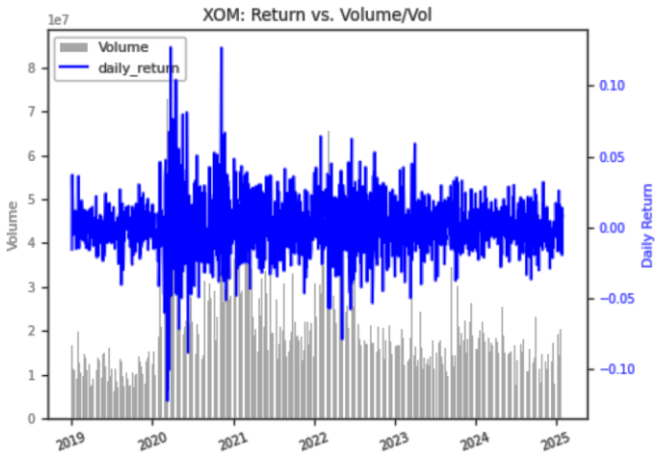




(Full visualizations for all entities are available in Appendix C.)

2.7 Returns vs Volume/Volatility

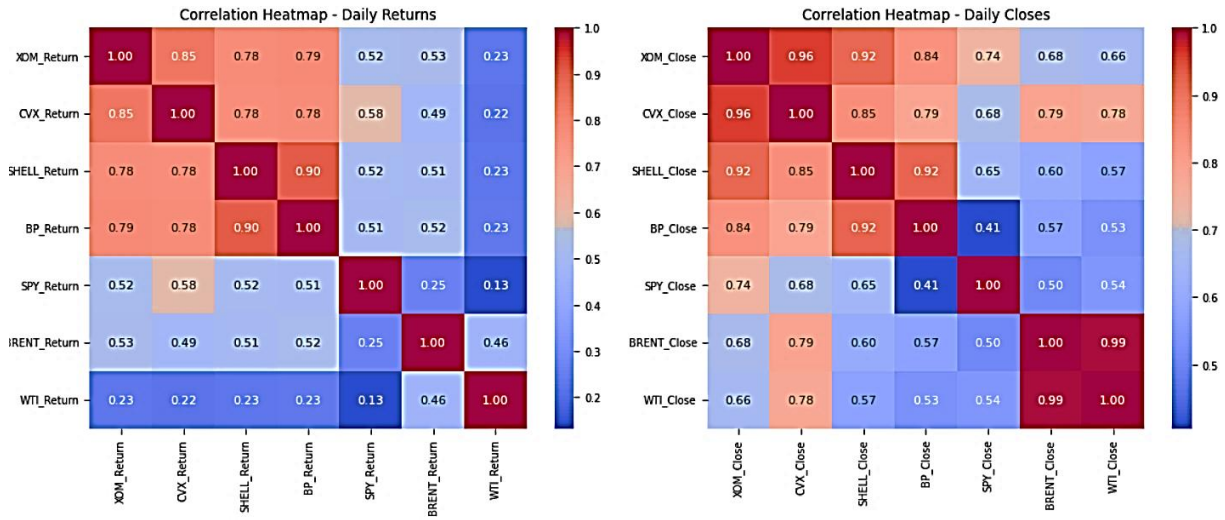
We observed a clear relationship between trading volume and return spikes in stocks like SPY and XOM, particularly during turbulent events like early COVID. However, oil commodities like WTI show volatility even without volume changes suggesting speculative forces (e.g., futures markets) may drive price swings independent of trading activity.



(Full visualizations for all entities are available in Appendix C.)

2.8 How Closely Do All Entities Move Together?

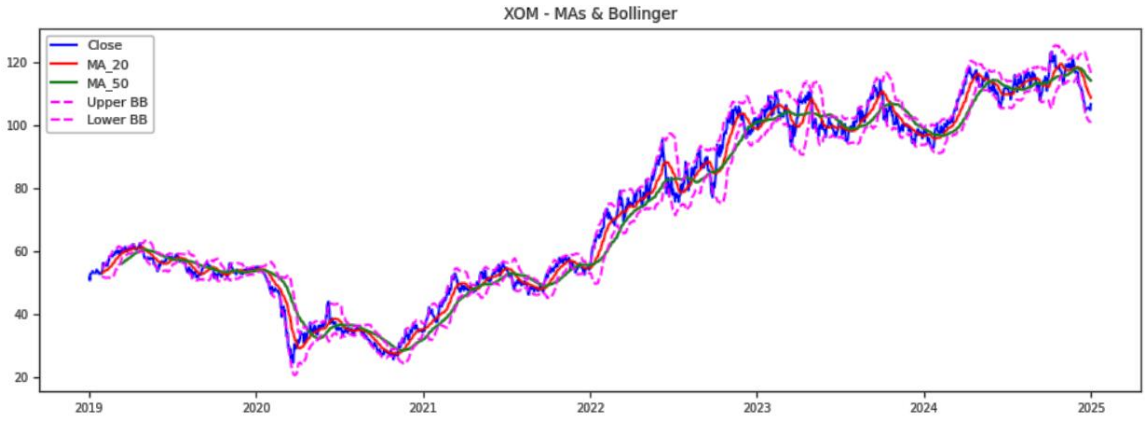
Comparing price and return correlations, we found Brent–WTI prices are nearly identical (0.99), while their returns are less aligned (0.46), suggesting similar long-term trends but different short-term dynamics. For oil stocks, XOM–CVX showed strong correlation in both price and return, reinforcing their sector-level dependency.



2.9 Moving Averages and Bollinger Bands

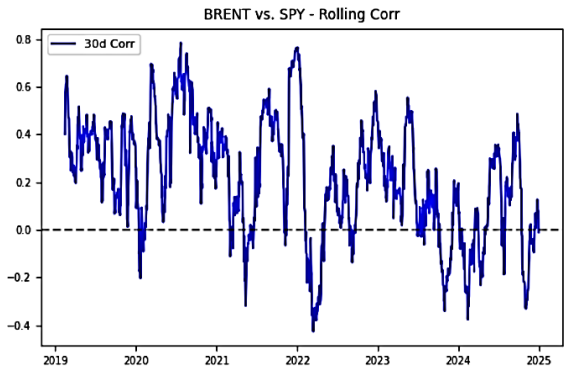
Moving averages and Bollinger Bands help assess trend direction and volatility. Most entities show mixed signals, with oil commodities leaning slightly bullish and some stocks facing downward pressure.

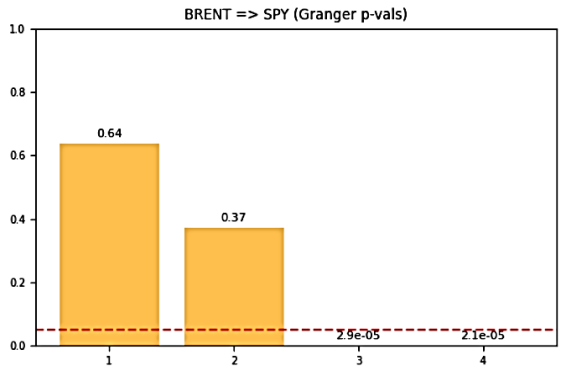
Entity	Last Close	MA 20	MA 50	Upper BB	Lower BB	Trend Comment
XOM	106.62	108.71	114.15	116.13	101.30	Below MAs; slight recovery within bands
CVX	143.25	148.45	151.53	162.08	134.81	Bearish; consistently below MAs
SHELL	61.98	62.05	63.91	65.11	58.99	Near MA20; narrowing volatility
BP	29.15	28.83	28.96	30.01	27.64	Mild bullish crossover (MA20 > MA50)
SPY	586.08	598.22	591.11	613.21	583.24	Near lower BB; post-rally pullback
Brent	74.64	73.19	73.36	75.03	71.35	Above MAs; moderately bullish
WTI	71.72	69.78	69.65	72.05	67.51	Bullish setup; hugging upper BB



2.10 Granger Causality and Rolling Correlation

Brent, CVX, and WTI showed significant Granger causality with SPY returns , meaning they may help predict SPY movements during volatile periods. Rolling 30-day correlations confirmed this, with stronger co-movement during crises (COVID, energy shocks) and decoupling during calmer markets.





(Full visualizations for all entities are available in Appendix C.)

3. Section 5: Textual Analysis

This section uses Natural Language Processing (NLP) to understand how financial news sentiment relates to stock and commodity performance , particularly in energy markets.

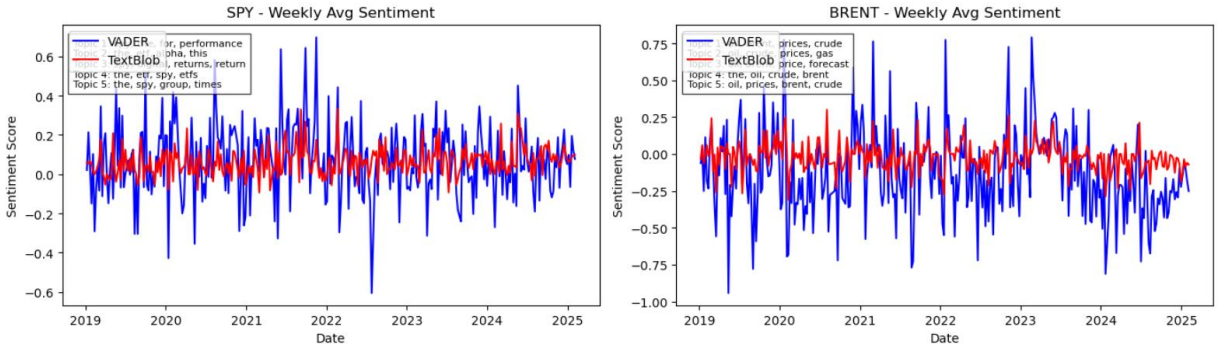
3.1 Sentiment Analysis and Topic Modeling

We analysed thousands of news headlines using VADER and TextBlob. Weekly sentiment trends showed:

- **WTI and Brent**: Extreme swings in tone during crashes or crises
- **SHELL and BP**: More stable, slightly positive tone
- **SPY**: Mild positive shift post-2022

Topic modeling using LDA revealed entity-specific keywords — for example, “earnings”, “oil”, and “climate” . offering insight into what news themes drive coverage and possibly investor attention.

Entity	Top Words (from Topics)
XOM	oil, exxonmobil, finance, guyana, climate
CVX	chevron, oil, earnings, regulation, houston
SHELL	shell, dutch, global, energy, times
BP	bp, offshore, gas, finance, field
SPY	spy, etf, investor, returns, alpha
WTI	wti, oil, crude, reuters, price
BRENT	brent, opec, forecast, sanctions, demand



(Full visualizations for all entities are available in Appendix D.)

3.2 Daily Sentiment Time Series

We include sample sentiment scores for individual articles, highlighting how different headlines are interpreted by VADER and TextBlob. We shifted to weekly sentiment later to reduce noise and align with typical investment horizons.

Entity	Example Article Title	VADER	TextBlob
XOM	ExxonMobil's 2018 performance worst since 1981	-0.8481	-0.6667
CVX	Chevron shuts down Gorgon LNG train 3 due to maintenance	0.2732	-0.1122
SHELL	Shell announces new startup cohort	0.8225	0.2857
BP	Business Efficiency Planning For Effective S&O	0.8807	0.4000
SPY	Kowtowing to the 'Prince of Evil'	0.0000	-0.6667
Brent	Oil price drop shows why reforms are urgent	-0.3400	0.0000
WTI	Crude oil prices end the year lower than they started	-0.7964	-0.3500

Insight: Sentiment signals provide early warnings of market stress or optimism. For example, **WTI and Brent** often reflect sharp negative sentiment around price crashes, while **SHELL** and **BP** tend to show more stable tone except during sectoral shocks.

3.3 Sentiment vs Weekly Returns

This section compares **weekly sentiment scores** (VADER) with **weekly returns** of each entity. The idea is to check if positive or negative sentiment tends to align with market performance over time. We computed:

- **Mean sentiment**: Overall tone (positive or negative)
- **Std Dev (sentiment)**: Volatility in sentiment

- **Mean weekly return**: How the asset performed on average each week
- **Std Dev (return)**: How volatile those weekly returns were
- **Min / Max**: The most extreme values seen over the period

Sentiment Statistics Summary

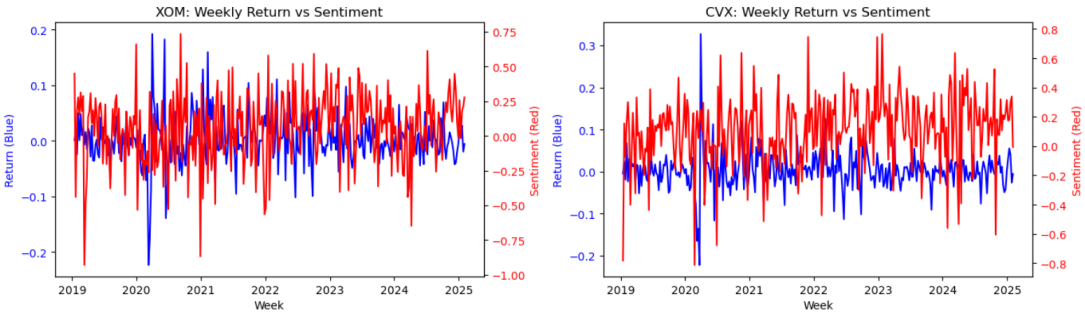
Entity	Mean Sentiment	Std Dev	Min	Max
XOM	0.058	0.25	–0.93	+0.73
CVX	0.109	0.23	–0.81	+0.76
SHELL	0.099	0.20	–0.89	+0.60
BP	0.122	0.18	–0.49	+0.76
SPY	0.069	0.18	–0.60	+0.69
Brent	–0.127	0.28	–0.94	+0.79
WTI	–0.194	0.30	–0.89	+0.78

Returns Statistics Summary

Entity	Mean Sentiment	Std Dev	Min	Max
XOM	+0.0032	0.0435	–22.31%	+19.24%
CVX	+0.0028	0.0441	–22.21%	+32.70%
SHELL	+0.0022	0.0457	–26.99%	+24.64%
BP	+0.0014	0.0482	–25.47%	+31.50%
SPY	+0.0034	0.0262	–12.54%	+17.36%
Brent	+0.0029	0.0629	–33.80%	+45.21%
WTI	–0.0074	0.1819	–267.91%	+59.55%

Comparing weekly sentiment to weekly returns, we found:

- **SHELL and BP** had positive sentiment and returns.
- **WTI and Brent** had negative tone and weak performance.
- No clear pattern emerged across all entities, suggesting sentiment **on its own** does not predict return direction consistently.



(Full visualizations for all entities are available in Appendix D.)

3.4 OLS Regression of Sentiment vs Returns

Ordinary Least Squares (OLS) regression was used to test if news sentiment can predict daily returns. Actual vs. predicted plots for each entity show a clear divergence, and the **R^2 values are low across the board**, indicating limited explanatory power.

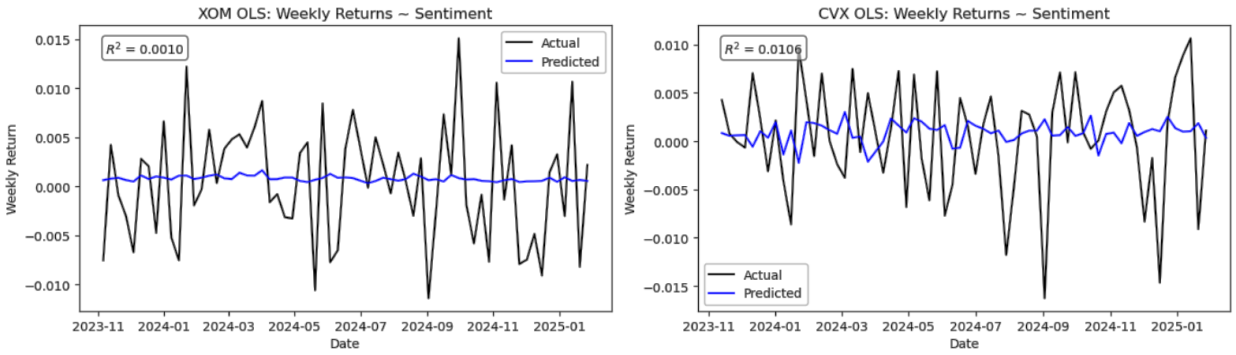
OLS regressions showed:

- Only **BP and SHELL** had statistically significant results.
- **R^2 values were low**, indicating weak explanatory power.
- For most assets, sentiment tone did not reliably predict weekly returns.

However, BP's moderate results indicate some potential in sentiment-based forecasting when combined with other signals.

OLS Regression Summary (R^2 and p-values)

Entity	R^2 Value	p-value	Interpretation
XOM	0.001	0.611	Not significant
CVX	0.011	0.104	Weak relationship, not significant
SHELL	0.016	0.044	Statistically significant, but weak
BP	0.088	0.000	Moderate and significant
SPY	0.000	0.955	No relationship
BRENT	0.001	0.723	Not significant
WTI	0.001	0.550	Very weak, not significant



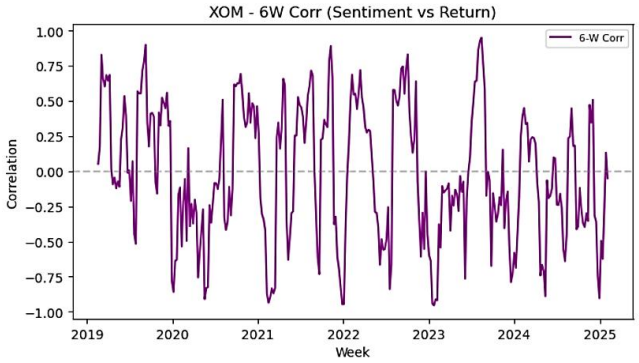
(Full visualizations for all entities are available in Appendix D.)

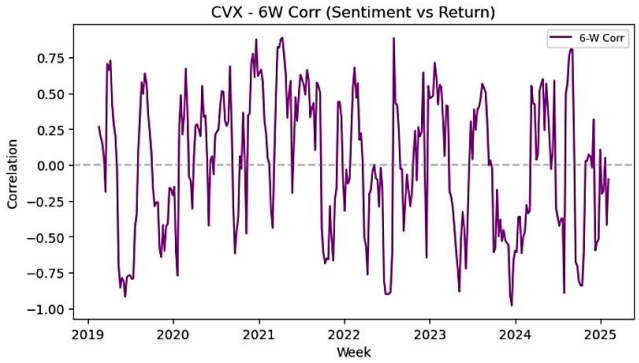
3.5 Rolling Correlation: Sentiment vs Returns

To study short-term dynamics, we calculated a **6-week rolling correlation** between sentiment and returns. These correlations are highly unstable—often flipping between positive and negative.

Rolling Correlation Summary

Entity	Mean Corr	Max	Min	Recent Trend
XOM	+0.003	+0.48	–0.41	Mild, volatile
CVX	–0.023	+0.57	–0.57	Mixed, improving
SHELL	+0.000	+0.42	–0.53	Slight positive
BP	–0.026	+0.45	–0.54	Slightly negative
SPY	+0.048	+0.47	–0.41	Increasing
Brent	–0.030	+0.39	–0.46	No clear pattern
WTI	+0.034	+0.49	–0.47	Recent positive spike





(Full visualizations for all entities are available in Appendix D.)

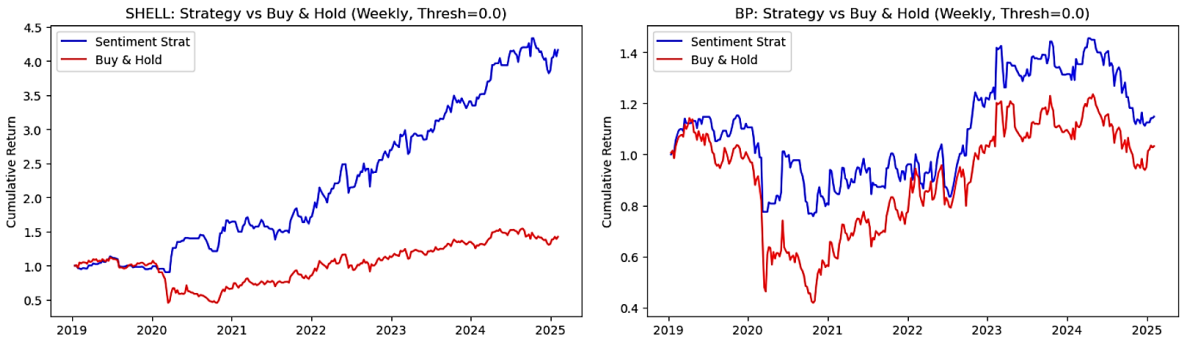
3.6 Sentiment-Based Strategy vs Buy & Hold

We created a basic **sentiment trading strategy**: The trading rule was simple: take a position if weekly sentiment was positive; otherwise, stay out of the market. This avoids shorting and mimics cautious retail investor behavior. Here's how that performed vs traditional **buy & hold**:

Strategy vs. Buy & Hold Performance

Entity	Strategy Final Value	Buy & Hold Final Value	Winner
XOM	1.71	2.35	Buy & Hold
CVX	0.72	1.84	Buy & Hold
SHELL	4.16	1.42	Sentiment Strategy
BP	1.15	1.03	Sentiment Strategy
SPY	1.62	2.54	Buy & Hold
BRENT	1.00	1.36	Buy & Hold
WTI	–1.36	1.72	Buy & Hold

*Sentiment-based strategies can outperform in **media-sensitive stocks**, but underperform in broad indices and commodities.*



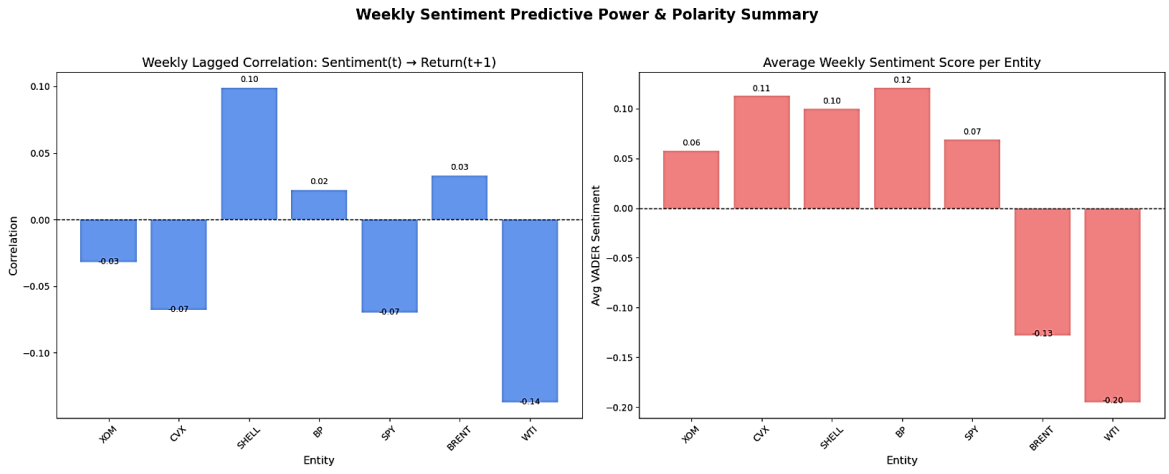
(Full visualizations for all entities are available in Appendix D.)

3.7 Lagged Correlation and Mean Sentiment Summary

We calculated correlations between sentiment this week and returns the next. Only **SHELL** showed a meaningful positive correlation (+0.10). Others were weak or negative, despite strong sentiment scores. This suggests **sentiment lags behind price**, or other factors dominate short-term movement.

Summary Table

Entity	Lagged Corr	Mean VADER Sentiment	Quick Insight
XOM	–0.03	+0.06	Slightly positive tone, weak signal
CVX	–0.07	+0.11	Optimistic tone, but poor correlation
SHELL	+0.10	+0.10	Most predictive, stable positive tone
BP	+0.02	+0.12	Positive tone, weak return signal
SPY	–0.07	+0.07	Mildly optimistic, no predictive power
BRENT	+0.03	–0.13	Negative tone, slightly positive correlation
WTI	–0.14	–0.20	Most negative tone, no predictability



Conclusion

This project explored how oil price movements and news sentiment influence the performance of major energy stocks and broader market indices between 2019 and 2025. Through data visualization and natural language processing, we uncovered several key insights:

- **Oil prices**, particularly Brent, are strongly linked to upstream energy stocks like XOM and CVX. These relationships can be used for sector-specific forecasting and hedging strategies.
- **News volume spikes** consistently aligned with periods of heightened return volatility, showing that investors react sharply to information flow — even when the sentiment tone itself is mixed.
- While **sentiment analysis** did not reliably predict stock returns across the board, it proved useful in specific cases (e.g. SHELL, BP), especially when aggregated weekly and combined with trend indicators.
- A **simple sentiment-based trading strategy** outperformed in certain media-sensitive stocks but underperformed in broad indices and commodity markets — highlighting that sentiment is most effective when used as a **filter** rather than a standalone signal.

Overall, the combination of structured market data with unstructured text-based insights provides a richer and more flexible toolkit for investors, traders, and risk managers. It enables **early identification of market shifts**, especially in high-volatility sectors like energy, and supports **more informed, data-driven investment decisions**.

5. References

[1] **Hutto, C.J., & Gilbert, E.** (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. Proceedings of the Eighth International Conference onWeblogs and Social Media (ICWSM-14).
Referenced in Section 5 for sentiment scoring.

[2] **TextBlob Documentation.** (n.d.). Simple Python Library for Text Processing. https://textblob.readthedocs.io
Used alongside VADER for polarity scoring.

[3] **Rehurek, R., & Sojka, P.** (2010). Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Underpins the LDA topic modeling done in Section 5.

[4] **Bollinger, J.** (2001). Bollinger on Bollinger Bands. McGraw-Hill. Technical foundation for volatility bands in Section 4.

[5] **Granger, C.W.J.** (1969). Investigating Causal Relations by Econometric Models and Cross-Spectral Methods. Econometrica, 37(3), 424–438.

[6] **Pandas Documentation.** (n.d.). Data Analysis and Time Series Tools.
https://pandas.pydata.org
Data manipulation, rolling averages, and resampling.

[7] **Matplotlib & Seaborn Docs.** (n.d.). Data Visualization in Python.
https://matplotlib.org,
https://seaborn.pydata.org
Used for all plots and visualizations in Section 4.

Acknowledgment of GPT Use

Portions of the code and analysis workflows presented in this report were developed with assistance from OpenAI's ChatGPT. This tool was used primarily for Python code generation, debugging, and guidance on data visualization and natural language processing techniques. All final implementations were reviewed and adapted by me and my team member.

Appendix A Section 4 - Data visualisations

April 20, 2025

Descriptive Visual Analysis: Boxplot of Daily Returns and Histogram of Close Prices for Each Entity

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


# 1) MySQL Connection & Entities


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


entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

# 2) TABLE / COLUMN MAPS


def get_table_and_col(ent):
    """
    If ent is brent or wti => brent_wti_data, with Brent_Close/WTI_Close.
    Else => <ent>_data with (Date, Close).
    """
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col


# 3) READ & PREPARE
```

```
def read_and_prepare(ent):

    table_name, col_name = get_table_and_col(ent)

    df = pd.read_sql(f"SELECT Date, {col_name} AS MyClose FROM {table_name}",␣
 ↪engine)

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

    # Convert MyClose -> numeric, rename to 'Close'
    df["MyClose"] = pd.to_numeric(df["MyClose"], errors="coerce")
    df.dropna(subset=["MyClose"], inplace=True)

    df.rename(columns={"MyClose": "Close"}, inplace=True)
    df.sort_values("Date", inplace=True)

    return df


# 4) CREATE SUBPLOTS: len(entities) rows x 2 columns


rows = len(entities)
cols = 2
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(6*cols, 4*rows))

# If there's only 1 row, unify indexing
if rows == 1:
    axes = [axes]


# 5) LOOP OVER ENTITIES


for i, ent in enumerate(entities):
    df_entity = read_and_prepare(ent)
    ent_upper = ent.upper()

    # 5A) Compute Daily Returns
    df_entity["Returns"] = df_entity["Close"].pct_change()
    df_entity.dropna(subset=["Returns"], inplace=True)

    # 5B) BOX PLOT OF DAILY RETURNS (left column)
    ax_box = axes[i][0] if rows > 1 else axes[0]
    # boxplot requires a list or array
```

```
    ax_box.boxplot(df_entity["Returns"], vert=True, showfliers=True)
    ax_box.set_title(f"{ent_upper} Daily Returns (Box)")
    ax_box.set_ylabel("Returns")

    # 5C) HISTOGRAM OF DAILY CLOSE (right column)
    ax_hist = axes[i][1] if rows > 1 else axes[1]
    ax_hist.hist(df_entity["Returns"], bins=30, color="blue", alpha=0.7)
    ax_hist.set_title(f"{ent_upper} Daily Returns (Histogram)")
    ax_hist.set_xlabel("Returns")
    ax_hist.set_ylabel("Frequency")

plt.tight_layout()
plt.subplots_adjust(hspace=0.35, wspace=0.3)
plt.show()
```

Rolling (7-day, 30-day) and Monthly Average Trend Visualization of Close Prices

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


# 1) MySQL Connection & Entities


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

entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]


# 2) TABLE / COLUMN MAPS


def get_table_and_col(ent):
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col



def read_and_prepare(ent):
```

```
    table_name, col_name = get_table_and_col(ent)

    df = pd.read_sql(f"SELECT Date, {col_name} AS MyClose FROM {table_name}",␣
 ↪engine)

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

    # Convert MyClose -> numeric, rename
    df["MyClose"] = pd.to_numeric(df["MyClose"], errors="coerce")
    df.dropna(subset=["MyClose"], inplace=True)

    df.rename(columns={"MyClose": "Close"}, inplace=True)
    df.sort_values("Date", inplace=True)

    return df


# 4) CREATE SUBPLOTS: N rows x 2 columns


rows = len(entities)
cols = 2
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(5*cols, 4*rows))

if rows == 1:
    axes = [axes]


for i, ent in enumerate(entities):
    df_entity = read_and_prepare(ent)
    ent_upper = ent.upper()

    # 5A) Rolling Averages (7-day, 30-day) => Left Subplot
    ax_roll = axes[i][0] if rows > 1 else axes[0]


    df_entity.sort_values("Date", inplace=True)

    # Create rolling columns
    df_entity["rolling_7d"]  = df_entity["Close"].rolling(7).mean()
    df_entity["rolling_30d"] = df_entity["Close"].rolling(30).mean()

    # Plot daily close
    ax_roll.plot(df_entity["Date"], df_entity["Close"], label="Daily Close",␣
 ↪alpha=0.5)
```

```
    # Plot 7-day rolling
    ax_roll.plot(df_entity["Date"], df_entity["rolling_7d"], label="7-day MA",␣
↪color="orange")
    # Plot 30-day rolling
    ax_roll.plot(df_entity["Date"], df_entity["rolling_30d"], label="30-day␣
↪MA", color="green")

    ax_roll.set_title(f"{ent_upper} - Rolling Averages")
    ax_roll.set_xlabel("Date")
    ax_roll.set_ylabel("Price")
    ax_roll.legend()

    # 5B) Monthly Averages => Right Subplot
    ax_monthly = axes[i][1] if rows > 1 else axes[1]

    # Create YearMonth
    df_entity["YearMonth"] = df_entity["Date"].dt.to_period("M")
    monthly_agg = df_entity.groupby("YearMonth")["Close"].mean().
↪reset_index(name="avg_close")

    # Convert YearMonth to a standard datetime for plotting
    monthly_agg["YM_dt"] = monthly_agg["YearMonth"].dt.to_timestamp()

    ax_monthly.plot(monthly_agg["YM_dt"], monthly_agg["avg_close"], marker="o",␣
↪label="Monthly Avg", color="blue")
    ax_monthly.set_title(f"{ent_upper} - Monthly Averages")
    ax_monthly.set_xlabel("Year-Month")
    ax_monthly.set_ylabel("Avg Close")
    ax_monthly.legend()


    for lbl in ax_monthly.get_xticklabels():
        lbl.set_rotation(45)


plt.tight_layout()
plt.subplots_adjust(hspace=0.4, wspace=0.3)
plt.show()
```

Entity vs. Brent Crude: Line and Scatter Plot Analysis of Closing 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")


stock_entities = ["xom", "cvx", "shell", "bp", "spy"]



def read_stock_merge_brent(ent):

    # 2A) Read stock data
    stock_table = f"{ent}_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) Read Brent data
    df_brent = pd.read_sql("SELECT Date, Brent_Close FROM brent_wti_data",␣
 ↪engine)
    df_brent["Date"] = pd.to_datetime(df_brent["Date"], format="%Y-%m-%d",␣
 ↪errors="coerce")
    df_brent.dropna(subset=["Date","Brent_Close"], inplace=True)
    df_brent.sort_values("Date", inplace=True)

    # 2C) Merge on Date
    df_merged = pd.merge(
        df_stock.rename(columns={"Close": f"{ent}_Close"}),
        df_brent,
        on="Date",
        how="inner"
    )
    df_merged.dropna(subset=[f"{ent}_Close","Brent_Close"], inplace=True)
    df_merged.sort_values("Date", inplace=True)

    return df_merged


# 3) CREATE SUBPLOTS: N rows x 2 columns


rows = len(stock_entities)
cols = 2
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(6*cols, 4*rows))
```

```
if rows == 1:
    axes = [axes]



for i, ent in enumerate(stock_entities):
    df_merged = read_stock_merge_brent(ent)
    ent_upper = ent.upper()

    # 4A) LEFT SUBPLOT: LINE CHART => ent_close vs. Brent_Close over time
    ax_line = axes[i][0] if rows > 1 else axes[0]
    df_line = df_merged.set_index("Date")

    df_line[[f"{ent}_Close","Brent_Close"]].plot(
        ax=ax_line,
        title=f"{ent_upper} vs. Brent (Line)",
        legend=True
    )
    ax_line.set_ylabel("Price")

    # 4B) RIGHT SUBPLOT: SCATTER => ent_close vs. Brent_Close
    ax_scatter = axes[i][1] if rows > 1 else axes[1]
    ax_scatter.scatter(df_merged["Brent_Close"], df_merged[f"{ent}_Close"],␣
 ↪alpha=0.5)
    ax_scatter.set_xlabel("Brent_Close")
    ax_scatter.set_ylabel(f"{ent_upper}_Close")
    ax_scatter.set_title(f"{ent_upper} vs. Brent (Scatter)")



plt.tight_layout()
plt.subplots_adjust(hspace=0.3, wspace=0.25)
plt.show()
```

Relationship Between Daily News Volume and Stock Returns: Dual Axis and Scatter Plot Analysis

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


# 1) MySQL Connection & Entities

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

```
entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]


# 2) STOCK/OIL: TABLE & COLUMN MAP


def get_stock_table_and_col(ent):
    """
    If ent in [brent, wti] => brent_wti_data with Brent_Close or WTI_Close
    else => <ent>_data with column Close
    """
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col


# 3) NEWS:  table = <ent>_news, with (published_date, title, summary, link)


def get_news_table(ent):
    return f"{ent}_news"


# 4) AGGREGATE NEWS -> (Date, daily_articles)
#

def aggregate_news_daily(news_table):
    """
    Reads raw news table with columns: published_date, title, summary, link
    => groups by DATE(published_date) to get daily_articles
    => returns a DataFrame with columns [Date, daily_articles]
    """
    df_news = pd.read_sql(f"SELECT published_date FROM {news_table}", engine)
    # Convert to datetime
    df_news["published_date"] = pd.to_datetime(df_news["published_date"],␣
 ↪errors="coerce")
    df_news.dropna(subset=["published_date"], inplace=True)

    # Extract date portion
    df_news["Date"] = df_news["published_date"].dt.date
    # Group
    daily_counts = df_news.groupby("Date").size().
 ↪reset_index(name="daily_articles")
```

```
    # Convert Date back to datetime for merging
    daily_counts["Date"] = pd.to_datetime(daily_counts["Date"])
    daily_counts.sort_values("Date", inplace=True)

    return daily_counts


# 5) READ STOCK/OIL => (Date, Close)


def read_stock_or_oil_data(ent):
    table_name, close_col = get_stock_table_and_col(ent)

    df = pd.read_sql(f"SELECT Date, {close_col} AS MyClose FROM {table_name}",␣
 ↪engine)
    df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d", errors="coerce")
    df.dropna(subset=["Date","MyClose"], inplace=True)

    df.rename(columns={"MyClose": "Close"}, inplace=True)
    df.sort_values("Date", inplace=True)
    return df


# 6) MERGE & Compute daily_return


def merge_news_and_close(ent):
    """
    1) Aggregate news for ent => daily_articles
    2) Read stock/oil data => (Date, Close)
    3) Merge => (Date, Close, daily_articles)
    4) Compute daily_return => drop first row if NaN
    Returns merged DataFrame
    """
    # News
    news_table = get_news_table(ent)
    df_news_daily = aggregate_news_daily(news_table)

    # Stock/Oil
    df_close = read_stock_or_oil_data(ent)

    # Merge
    df_merged = pd.merge(df_close, df_news_daily, on="Date", how="inner")
    df_merged.dropna(subset=["Close","daily_articles"], inplace=True)
    df_merged.sort_values("Date", inplace=True)
```

```
    # daily_return
    df_merged["daily_return"] = df_merged["Close"].pct_change()
    df_merged.dropna(subset=["daily_return"], inplace=True)

    return df_merged


# 7) CREATE SUBPLOTS: len(entities) rows x 2 columns


rows = len(entities)
cols = 2
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(6*cols, 4*rows))

if rows == 1:
    axes = [axes]



for i, ent in enumerate(entities):
    df_merged = merge_news_and_close(ent)
    ent_upper = ent.upper()

    # 8A) LEFT: Dual-Axis => bar(daily_articles) + line(Close)
    ax_bar = axes[i][0] if rows > 1 else axes[0]
    ax_line = ax_bar.twinx()

    ax_bar.bar(df_merged["Date"], df_merged["daily_articles"], width=1,␣
 ↪color="gray", alpha=0.5)
    ax_bar.set_ylabel("Articles", color="gray")
    ax_bar.tick_params(axis="y", labelcolor="gray")

    ax_line.plot(df_merged["Date"], df_merged["Close"], color="blue")
    ax_line.set_ylabel("Close", color="blue")
    ax_line.tick_params(axis="y", labelcolor="blue")

    ax_bar.set_title(f"{ent_upper}: Articles vs. Close (Dual Axis)")
    for lbl in ax_bar.get_xticklabels():
        lbl.set_rotation(30)

    # 8B) RIGHT: Scatter => daily_articles vs. daily_return
    ax_scat = axes[i][1] if rows > 1 else axes[1]
    ax_scat.scatter(df_merged["daily_articles"], df_merged["daily_return"],␣
 ↪alpha=0.5)
    ax_scat.set_xlabel("Daily Articles")
    ax_scat.set_ylabel("Daily Return")
    ax_scat.set_title(f"{ent_upper}: Articles vs. Return (Scatter)")
```

```
plt.tight_layout()
plt.subplots_adjust(hspace=0.35, wspace=0.3)
plt.show()
```

Correlation of Daily Returns and Top Movers: Heatmap with Individual Entity Movement Analysis

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine
import matplotlib.gridspec as gridspec

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


entities = ["xom", "cvx", "shel", "bp", "spy", "brent", "wti"]

def get_table_and_col(ent):
    """
    If ent in [brent, wti] => brent_wti_data with Brent_Close or WTI_Close
    else => <ent>_data with column Close
    """
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col


def read_all_entities(entities):
    """
    Reads daily close for each entity, merges them into a single wide DF:
      Date, XOM_Close, CVX_Close, ...
    (outer join on Date).
    """
    df_merged = None

    for ent in entities:
        table_name, close_col = get_table_and_col(ent)
        df_temp = pd.read_sql(f"SELECT Date, {close_col} AS MyClose FROM␣
 ↪{table_name}", engine)
```

```
        df_temp["Date"] = pd.to_datetime(df_temp["Date"], format="%Y-%m-%d",␣
 ↪errors="coerce")
        df_temp.dropna(subset=["Date","MyClose"], inplace=True)
        df_temp.sort_values("Date", inplace=True)

        ent_col = f"{ent.upper()}_Close"
        df_temp.rename(columns={"MyClose": ent_col}, inplace=True)

        if df_merged is None:
            df_merged = df_temp
        else:
            df_merged = pd.merge(df_merged, df_temp, on="Date", how="outer")

    df_merged.sort_values("Date", inplace=True)
    return df_merged


def compute_daily_returns(df, entities):
    """
    For each entity, compute daily returns as pct_change(fill_method=None).
    This avoids the FutureWarning about default fill_method='pad'.
    """
    df = df.copy()
    for ent in entities:
        col_close = f"{ent.upper()}_Close"
        col_ret   = f"{ent.upper()}_Return"
        if col_close in df.columns:
            df[col_ret] = df[col_close].pct_change(fill_method=None)
    return df


# 4) CORRELATION HEATMAP


def correlation_heatmap(df, entities, ax):

    return_cols = [f"{ent.upper()}_Return" for ent in entities]
    df_corr = df.dropna(subset=return_cols)[return_cols]
    corr_matrix = df_corr.corr()

    sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", ax=ax)
    ax.set_title("Correlation Heatmap (Daily Returns)")


# 5) TOP DAILY MOVERS
```

```
def plot_top_daily_movers(df, entities, ax_list, top_n=5):
    """
    For each entity, find top_n biggest absolute daily moves in its Return␣
 ↪column,
    plot them as a bar chart on the respective ax in ax_list.
    ax_list is a list of subplots, one per entity in row-major order.
    We have 7 entities => fill first 7 subplots. The last 2 we can ignore or␣
 ↪remove.
    """
    for i, ent in enumerate(entities):
        col_ret = f"{ent.upper()}_Return"
        ax = ax_list[i]
        if col_ret not in df.columns:
            ax.text(0.5, 0.5, f"No data for {ent}", ha="center", va="center")
            continue

        df_ent = df[["Date", col_ret]].dropna()
        df_ent["abs_return"] = df_ent[col_ret].abs()
        df_top = df_ent.nlargest(top_n, "abs_return")

        ax.bar(df_top["Date"].astype(str), df_top["abs_return"], color="red",␣
 ↪alpha=0.7)
        ax.set_title(f"{ent.upper()} Top {top_n} Movers")
        ax.set_ylabel("Abs Return")
        ax.tick_params(axis='x', rotation=45)



def main():
    # 6A) Read & Merge all into wide DF
    df_wide = read_all_entities(entities)

    # 6B) Compute daily returns
    df_ret = compute_daily_returns(df_wide, entities)

    # 6C) Create figure with 4 rows x 3 columns:

    import matplotlib.gridspec as gridspec

    fig = plt.figure(figsize=(14, 16))
    gs = gridspec.GridSpec(4, 3, height_ratios=[1.5, 1, 1, 1])

    # Correlation heatmap in row 0, spanning columns 0..2
    ax_corr = fig.add_subplot(gs[0, :])
    correlation_heatmap(df_ret, entities, ax_corr)
```

```
    # Next 3 rows => 3 columns each => 9 sub-subplots
    ax_subplots = []
    subplot_index = 0
    for row in [1,2,3]:
        for col in [0,1,2]:
            ax = fig.add_subplot(gs[row, col])
            ax_subplots.append(ax)
            subplot_index += 1

    # 6D) Plot top daily movers in the first 7 sub-subplots
    plot_top_daily_movers(df_ret, entities, ax_subplots, top_n=5)


    for j in range(7, 9):
        fig.delaxes(ax_subplots[j])

    plt.tight_layout()
    plt.subplots_adjust(hspace=0.4, wspace=0.3)
    plt.show()

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

Year-over-Year Trends and Calendar-Based Patterns: Analyzing Monthly, Weekly, and Daily Effects on Closing Prices

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


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

entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_col(ent):
    """
    Returns (table_name, close_col) for each entity.
    """
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col
```

```
def read_and_prepare(ent):
    table_name, col_name = get_table_and_col(ent)

    df = pd.read_sql(f"SELECT Date, {col_name} AS MyClose FROM {table_name}",␣
 ↪engine)

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

    # Convert MyClose -> numeric, rename to "Close"
    df["MyClose"] = pd.to_numeric(df["MyClose"], errors="coerce")
    df.dropna(subset=["MyClose"], inplace=True)

    df.rename(columns={"MyClose": "Close"}, inplace=True)
    df.sort_values("Date", inplace=True)

    # Extra columns
    df["Year"]       = df["Date"].dt.year
    df["Month"]      = df["Date"].dt.month
    df["DayOfWeek"]  = df["Date"].dt.day_name()     # 'Monday', 'Tuesday', ...
    df["DayOfMonth"] = df["Date"].dt.day           # 1..31

    return df


# 4) YEAR-OVER-YEAR (Manual Approach, No DeprecationWarning)

def compute_yoy(df):
    """
    1) Group by monthly average of 'Close'.
    2) For each month, shift by 1 row to get last year's data.
    3) Compute yoy_pct_change.
    Returns a DataFrame with yoy_pct_change for plotting.
    """
    # 4A) Monthly average
    df["YearMonth"] = df["Date"].dt.to_period("M")
    monthly = df.groupby("YearMonth")["Close"].mean().
 ↪reset_index(name="avg_close")

    monthly["Year"]  = monthly["YearMonth"].dt.year
    monthly["Month"] = monthly["YearMonth"].dt.month

    # Sort by (Month, Year)
    monthly.sort_values(["Month","Year"], inplace=True)
```

```
    # 4B) For each unique Month, shift avg_close by 1 row
    #     so we get last year's same month
    yoy_list = []
    for m in sorted(monthly["Month"].unique()):
        grp = monthly[monthly["Month"] == m].copy()
        grp.sort_values("Year", inplace=True)
        grp["last_year_close"] = grp["avg_close"].shift(1)
        yoy_list.append(grp)

    monthly2 = pd.concat(yoy_list, ignore_index=True)

    monthly2["yoy_diff"] = monthly2["avg_close"] - monthly2["last_year_close"]
    monthly2["yoy_pct_change"] = (monthly2["yoy_diff"] /␣
 ↪monthly2["last_year_close"]) * 100
    return monthly2.dropna(subset=["yoy_pct_change"])


# 5) DAY-OF-WEEK


def compute_day_of_week(df):
    dow = df.groupby("DayOfWeek")["Close"].mean().reset_index(name="avg_close")
    day_order =␣
 ↪["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]
    dow["DayOfWeek"] = pd.Categorical(dow["DayOfWeek"], categories=day_order,␣
 ↪ordered=True)
    dow.sort_values("DayOfWeek", inplace=True)
    return dow


# 6) DAY-OF-MONTH


def compute_day_of_month(df):
    dom = df.groupby("DayOfMonth")["Close"].mean().reset_index(name="avg_close")
    dom.sort_values("DayOfMonth", inplace=True)
    return dom


# 7) CREATE SUBPLOTS: len(entities) rows x 3 columns


rows = len(entities)
cols = 3
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(4*cols, 4*rows))

if rows == 1:
```

```
    axes = [axes]  # unify indexing if only 1 row



for i, ent in enumerate(entities):
    df_entity = read_and_prepare(ent)
    ent_upper = ent.upper()

    # 8A) YOY
    yoy_df = compute_yoy(df_entity).dropna(subset=["yoy_pct_change"])
    ax_yoy = axes[i][0] if rows > 1 else axes[0]
    yoy_df["YM_str"] = yoy_df["YearMonth"].astype(str)
    ax_yoy.plot(yoy_df["YM_str"], yoy_df["yoy_pct_change"], marker="o",␣
 ↪label="YoY %")
    ax_yoy.legend()
    ax_yoy.set_title(f"{ent_upper} - YoY %")
    ax_yoy.set_xlabel("Year-Month")
    ax_yoy.set_ylabel("YoY %")
    # rotate x-labels
    for lbl in ax_yoy.get_xticklabels():
        lbl.set_rotation(45)

    # 8B) Day-of-Week
    dow_df = compute_day_of_week(df_entity)
    ax_dow = axes[i][1] if rows > 1 else axes[1]
    ax_dow.bar(dow_df["DayOfWeek"], dow_df["avg_close"], color="skyblue",␣
 ↪label="Avg Close")
    ax_dow.legend()
    ax_dow.set_title(f"{ent_upper} - DayOfWeek Avg")
    ax_dow.set_xlabel("DayOfWeek")
    ax_dow.set_ylabel("Avg Close")
    for lbl in ax_dow.get_xticklabels():
        lbl.set_rotation(30)

    # 8C) Day-of-Month
    dom_df = compute_day_of_month(df_entity)
    ax_dom = axes[i][2] if rows > 1 else axes[2]
    ax_dom.plot(dom_df["DayOfMonth"], dom_df["avg_close"], marker="o",␣
 ↪color="green", label="Avg Close")
    ax_dom.legend()
    ax_dom.set_title(f"{ent_upper} - DayOfMonth Avg")
    ax_dom.set_xlabel("DayOfMonth")
    ax_dom.set_ylabel("Avg Close")


plt.tight_layout()
plt.subplots_adjust(hspace=0.35, wspace=0.3)
```

```
plt.show()
```

Return vs Volume/Volatility: Dual-Axis Visualization of Market Activity Across Entities

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
import matplotlib.gridspec as gridspec


# 1) Adjust Global Font Sizes

mpl.rcParams['font.size'] = 8
mpl.rcParams['axes.labelsize'] = 8
mpl.rcParams['legend.fontsize'] = 8
mpl.rcParams['xtick.labelsize'] = 7
mpl.rcParams['ytick.labelsize'] = 7


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

# Our 7 entities
entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_cols(ent):
    """
    If ent is brent/wti => brent_wti_data with prefix
    otherwise => <ent>_data with standard columns.
    """
    if ent in ["brent","wti"]:
        table_name = "brent_wti_data"
        prefix = "Brent_" if ent == "brent" else "WTI_"
        return {
            "table": table_name,
            "Open":  prefix+"Open",
            "High":  prefix+"High",
            "Low":   prefix+"Low",
            "Close": prefix+"Close",
            "Volume": prefix+"Volume"
        }
    else:
        table_name = f"{ent}_data"
        return {
```

```
            "table": table_name,
            "Open":  "Open",
            "High":  "High",
            "Low":   "Low",
            "Close": "Close",
            "Volume":"Volume"
        }

def read_and_prepare(ent):

    info = get_table_and_cols(ent)
    table = info["table"]

    col_open  = info["Open"]
    col_high  = info["High"]
    col_low   = info["Low"]
    col_close = info["Close"]
    col_vol   = info["Volume"]

    sql = f"""
    SELECT
      Date,
      {col_open}  AS Open,
      {col_high}  AS High,
      {col_low}   AS Low,
      {col_close} AS Close,
      {col_vol}   AS Volume
    FROM {table}
    """

    df = pd.read_sql(sql, engine)
    df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d", errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)

    for c in ["Open","High","Low","Close","Volume"]:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")

    # daily_return
    df["daily_return"] = df["Close"].pct_change(fill_method=None)
    # rolling_vol => 7-day std
    df["rolling_vol"]  = df["daily_return"].rolling(7).std()

    return df
```

```
# 4) Plot Return vs. Volume/Vol for All Entities


def plot_return_vs_volume_vol_all(dfs_dict):
    ent_keys = list(dfs_dict.keys())  # e.g. 7 entities
    n = len(ent_keys)                 # 7

    fig = plt.figure(figsize=(16, 12))
    gs = gridspec.GridSpec(3, 3)  # 3 rows, 3 cols => 9 sub-subplots

    ax_list = []
    subplot_index = 0
    for row in range(3):
        for col in range(3):
            ax = fig.add_subplot(gs[row, col])
            ax_list.append(ax)
            subplot_index += 1
            if subplot_index >= 9:
                break

    # Plot each entity in subplots
    for i, ent in enumerate(ent_keys):
        if i >= 9:
            break
        df = dfs_dict[ent]
        ax_left = ax_list[i]
        ax_right = ax_left.twinx()
        ent_upper = ent.upper()

        # We'll build legends
        legend_handles = []
        legend_labels  = []

        # Volume or rolling_vol on left
        if "Volume" in df.columns and df["Volume"].notna().sum() > 0:
            bars = ax_left.bar(df["Date"], df["Volume"], width=1, color="gray",␣
↪alpha=0.5, label="Volume")
            ax_left.set_ylabel("Volume", color="gray")
            ax_left.tick_params(axis="y", labelcolor="gray")
            legend_handles.append(bars[0])
            legend_labels.append("Volume")
        else:
            if df["rolling_vol"].notna().sum() > 0:
                line_vol, = ax_left.plot(df["Date"], df["rolling_vol"],␣
↪color="orange", label="rolling_vol")
                ax_left.set_ylabel("Rolling Vol", color="orange")
                ax_left.tick_params(axis="y", labelcolor="orange")
```

```
                legend_handles.append(line_vol)
                legend_labels.append("rolling_vol")
            else:
                ax_left.set_ylabel("No Vol/rolling_vol data")

        # daily_return on right
        if "daily_return" in df.columns and df["daily_return"].notna().sum() >␣
 ↪0:
            line_ret, = ax_right.plot(df["Date"], df["daily_return"],␣
 ↪color="blue", label="daily_return")
            ax_right.set_ylabel("Daily Return", color="blue")
            ax_right.tick_params(axis="y", labelcolor="blue")
            legend_handles.append(line_ret)
            legend_labels.append("daily_return")
        else:
            ax_right.set_ylabel("No daily_return", color="blue")

        ax_left.set_title(f"{ent_upper}: Return vs. Volume/Vol")

        # Combine legends
        if legend_handles and legend_labels:
            ax_left.legend(legend_handles, legend_labels, loc="upper left")

        # Rotate xlabels
        for lbl in ax_left.get_xticklabels():
            lbl.set_rotation(20)

    for j in range(n, 9):
        fig.delaxes(ax_list[j])

    fig.tight_layout()
    fig.subplots_adjust(hspace=0.3, wspace=0.2)
    return fig


def main():
    dfs_dict = {}
    for ent in entities:
        df = read_and_prepare(ent)
        dfs_dict[ent] = df

    fig_rvv = plot_return_vs_volume_vol_all(dfs_dict)
    plt.show()


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

Cross-Entity Correlation Heatmaps: Daily Returns vs Daily Closes

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine


mpl.rcParams['font.size'] = 8
mpl.rcParams['axes.labelsize'] = 8
mpl.rcParams['legend.fontsize'] = 8
mpl.rcParams['xtick.labelsize'] = 7
mpl.rcParams['ytick.labelsize'] = 7


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

# Our 7 entities
entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_cols(ent):
    """
    If ent is brent/wti => brent_wti_data with prefix
    else => <ent>_data with standard columns (Open, High, Low, Close, Volume).
    We'll just read 'Close' for correlation among daily closes.
    """
    if ent in ["brent","wti"]:
        table_name = "brent_wti_data"
        prefix = "Brent_" if ent == "brent" else "WTI_"
        return {
            "table": table_name,
            "Close": prefix+"Close"
        }
    else:
        table_name = f"{ent}_data"
        return {
            "table": table_name,
            "Close": "Close"
        }


def read_and_prepare(ent):
    info = get_table_and_cols(ent)
    table = info["table"]
```

```
    col_close = info["Close"]

    sql = f"SELECT Date, {col_close} AS Close FROM {table}"

    df = pd.read_sql(sql, engine)
    df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d", errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)

    df["Close"] = pd.to_numeric(df["Close"], errors="coerce")
    df["daily_return"] = df["Close"].pct_change(fill_method=None)

    return df


# 4) Build Wide DataFrames for Correlation


def build_wide_close_df(dfs_dict):
    """
    Merges each entity's close into a single wide DataFrame => columns␣
 ↪[XOM_Close, CVX_Close, ...].
    """
    merged = None
    for ent, df in dfs_dict.items():
        df_temp = df[["Date","Close"]].dropna(subset=["Close"]).copy()
        df_temp.rename(columns={"Close": f"{ent.upper()}_Close"}, inplace=True)
        if merged is None:
            merged = df_temp
        else:
            merged = pd.merge(merged, df_temp, on="Date", how="outer")

    return merged

def build_wide_return_df(dfs_dict):
    """
    Merges each entity's daily_return => columns [XOM_Return, CVX_Return, ...].
    """
    merged = None
    for ent, df in dfs_dict.items():
        df_temp = df[["Date","daily_return"]].dropna(subset=["daily_return"]).
 ↪copy()
        df_temp.rename(columns={"daily_return": f"{ent.upper()}_Return"},␣
 ↪inplace=True)
        if merged is None:
            merged = df_temp
        else:
```

```
            merged = pd.merge(merged, df_temp, on="Date", how="outer")

    return merged


# 5) Plot Correlation Heatmaps: Daily Return & Daily Close


def plot_correlation_heatmaps(df_close_wide, df_ret_wide):
    """
    1 row, 2 columns subplots:
      left => correlation of daily returns
      right => correlation of daily closes
    """
    fig, (ax_left, ax_right) = plt.subplots(nrows=1, ncols=2, figsize=(12,5))

    # A) Daily Return Correlation
    ret_cols = [c for c in df_ret_wide.columns if c.endswith("_Return")]
    df_ret_corr = df_ret_wide.dropna(subset=ret_cols)[ret_cols]
    if not df_ret_corr.empty:
        corr_ret = df_ret_corr.corr()
        sns.heatmap(corr_ret, annot=True, cmap="coolwarm", fmt=".2f",␣
 ↪ax=ax_left)
        ax_left.set_title("Correlation Heatmap - Daily Returns")
    else:
        ax_left.text(0.5, 0.5, "No Return Data", ha="center", va="center")

    # B) Daily Close Correlation
    close_cols = [c for c in df_close_wide.columns if c.endswith("_Close")]
    df_close_corr = df_close_wide.dropna(subset=close_cols)[close_cols]
    if not df_close_corr.empty:
        corr_close = df_close_corr.corr()
        sns.heatmap(corr_close, annot=True, cmap="coolwarm", fmt=".2f",␣
 ↪ax=ax_right)
        ax_right.set_title("Correlation Heatmap - Daily Closes")
    else:
        ax_right.text(0.5, 0.5, "No Close Data", ha="center", va="center")

    fig.tight_layout()
    return fig


def main():
    # A) Read data for each entity => store in dict
    dfs_dict = {}
    for ent in entities:
        df = read_and_prepare(ent)
```

```
        dfs_dict[ent] = df

    # B) Build wide DataFrames for daily closes & daily returns
    df_close_wide = build_wide_close_df(dfs_dict)
    df_ret_wide   = build_wide_return_df(dfs_dict)

    # C) Plot correlation heatmaps side by side
    fig_corr = plot_correlation_heatmaps(df_close_wide, df_ret_wide)
    plt.show()


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

Time Series Decomposition and Autocorrelation Analysis (ACF & PACF) of Daily Closes

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

# For time-series decomposition, ACF, PACF
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf



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

entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_col(ent):
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col


def read_and_decompose(ent):
    """
    1) Query daily close from MySQL for the entity,
    2) Set Date as index,
```

```
    3) Decompose raw_close => extract trend, seasonal, resid,
    4) Return the DataFrame with 'raw_close' and 'trend' columns
       plus the full decomposition result for plotting.
    """
    table_name, col_name = get_table_and_col(ent)

    query = f"""
    SELECT Date, {col_name} AS raw_close
    FROM {table_name}
    WHERE Date >= '2019-01-01' AND Date <= '2025-01-01'
      AND {col_name} IS NOT NULL
    ORDER BY Date
    """

    df = pd.read_sql(query, engine)

    # Convert Date
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)

    # Convert raw_close -> numeric
    df["raw_close"] = pd.to_numeric(df["raw_close"], errors="coerce")
    df.dropna(subset=["raw_close"], inplace=True)

    # Set Date as index
    df.set_index("Date", inplace=True)

    if df.empty:
        return df, None  # no data to decompose

    # Decompose (use e.g. period=252 for ~1 year if daily)
    # model="additive" is typical for stock data
    period_guess = 252
    result = seasonal_decompose(df["raw_close"], model="additive",␣
↪period=period_guess)

    # We'll store the 'trend' component in df for quick plotting
    df["trend"] = result.trend  # might have NaNs at start/end

    return df, result


# 3) Subplots Setup: one row per entity, 3 columns


nrows = len(entities)
```

```
ncols = 3
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(14, 4*nrows))

if nrows == 1:
    axes = [axes]


for i, ent in enumerate(entities):
    df, decomp_result = read_and_decompose(ent)
    if df.empty or decomp_result is None:
        print(f"[WARNING] No data or decomposition for {ent.upper()}")
        # possibly skip or continue
        continue

    ent_upper = ent.upper()

    ax0 = axes[i][0] if nrows>1 else axes[0]  # for "observed vs. trend"
    ax1 = axes[i][1] if nrows>1 else axes[1]  # for ACF
    ax2 = axes[i][2] if nrows>1 else axes[2]  # for PACF

    # 4A) Plot Observed vs. Trend in col 0
    ax0.plot(df.index, df["raw_close"], color="blue", label="Observed")
    ax0.plot(df.index, df["trend"], color="red", label="Trend")
    ax0.set_title(f"{ent_upper} - Observed vs. Trend")
    ax0.legend()

    # 4B) Plot ACF in col 1
    # statsmodels' plot_acf returns its own figure, so we pass ax=ax1
    sm.graphics.tsa.plot_acf(df["raw_close"].dropna(), ax=ax1, lags=30, alpha=0.
 ↪05)
    ax1.set_title(f"{ent_upper} - ACF")

    # 4C) Plot PACF in col 2
    sm.graphics.tsa.plot_pacf(df["raw_close"].dropna(), ax=ax2, lags=30,␣
 ↪alpha=0.05)
    ax2.set_title(f"{ent_upper} - PACF")

plt.tight_layout()
plt.subplots_adjust(hspace=0.3, wspace=0.3)
plt.show()
```

Moving Averages and Bollinger Bands Visualization for Price Trends

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

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

entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_col(ent):
    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col

def read_data(ent):
    """
    1) Query daily close from MySQL for the entity,
    2) Return a DataFrame with columns:
       [Date, Close, MA_20, MA_50, Upper_BB, Lower_BB].
    """
    table_name, col_name = get_table_and_col(ent)

    query = f"""
    SELECT Date, {col_name} AS raw_close
    FROM {table_name}
    WHERE Date >= '2019-01-01' AND Date <= '2025-01-01'
      AND {col_name} IS NOT NULL
    ORDER BY Date
    """

    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)

    # Convert raw_close -> numeric
    df["raw_close"] = pd.to_numeric(df["raw_close"], errors="coerce")
    df.dropna(subset=["raw_close"], inplace=True)
    df.rename(columns={"raw_close": "Close"}, inplace=True)

    # 1) short-term MA => 20-day
    df["MA_20"] = df["Close"].rolling(window=20).mean()
    # 2) longer-term MA => 50-day
    df["MA_50"] = df["Close"].rolling(window=50).mean()

    # Bollinger => ±2 std dev around MA_20
```

```
    df["STD_20"] = df["Close"].rolling(window=20).std()
    df["Upper_BB"] = df["MA_20"] + 2 * df["STD_20"]
    df["Lower_BB"] = df["MA_20"] - 2 * df["STD_20"]

    return df


# 3) Setup Subplots: 7 rows x 1 column

nrows = len(entities)
ncols = 1
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10, 4*nrows))
if nrows == 1:
    axes = [axes]


for i, ent in enumerate(entities):
    df_entity = read_data(ent)
    ent_upper = ent.upper()

    ax = axes[i]

    if df_entity.empty:
        ax.set_title(f"{ent_upper} - No Data")
        continue

    # Plot
    ax.plot(df_entity["Date"], df_entity["Close"], color="blue", label="Close")
    ax.plot(df_entity["Date"], df_entity["MA_20"], color="red", label="MA_20")
    ax.plot(df_entity["Date"], df_entity["MA_50"], color="green", label="MA_50")
    ax.plot(df_entity["Date"], df_entity["Upper_BB"], color="magenta",␣
 ↪linestyle="--", label="Upper BB")
    ax.plot(df_entity["Date"], df_entity["Lower_BB"], color="magenta",␣
 ↪linestyle="--", label="Lower BB")

    ax.set_title(f"{ent_upper} - MAs & Bollinger")
    ax.legend(loc="upper left")

    # Print last 5 rows so we see numeric values
    print(f"\n=== {ent_upper} - Last 5 rows of data with MAs & Bollinger ===")
    cols_to_show = ["Date","Close","MA_20","MA_50","Upper_BB","Lower_BB"]
    tail_5 = df_entity[cols_to_show].tail(5).to_string(index=False)
    print(tail_5)


plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
```

```
plt.show()
```

Granger Causality and Rolling Correlation: Entity Returns vs. SPY

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from statsmodels.tsa.stattools import grangercausalitytests


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

# We omit 'spy' so we don't do "spy vs spy"
entities = ["xom", "cvx", "shell", "bp", "brent", "wti"]

def get_table_and_col(ent):

    if ent in ["brent", "wti"]:
        table_name = "brent_wti_data"
        close_col  = "Brent_Close" if ent == "brent" else "WTI_Close"
    else:
        table_name = f"{ent}_data"
        close_col  = "Close"
    return table_name, close_col

def read_returns(ent, start_date="2019-01-01", end_date="2025-01-01"):
    """
    Query daily close from MySQL for the entity,
    compute daily returns, return DataFrame [Date, ret].
    """
    table_name, col_name = get_table_and_col(ent)

    query = f"""
    SELECT Date, {col_name} AS raw_close
    FROM {table_name}
    WHERE Date >= '{start_date}' AND Date <= '{end_date}'
      AND {col_name} IS NOT NULL
    ORDER BY Date
    """

    df = pd.read_sql(query, engine)
    # Convert Date
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)
```

```
    # Convert raw_close -> numeric
    df["raw_close"] = pd.to_numeric(df["raw_close"], errors="coerce")
    df.dropna(subset=["raw_close"], inplace=True)

    # daily returns
    df["ret"] = df["raw_close"].pct_change(fill_method=None)
    df.dropna(subset=["ret"], inplace=True)

    return df[["Date","ret"]]


def main():
    # 2A) Read SPY returns
    df_spy = read_returns("spy")
    df_spy.rename(columns={"ret":"spy_ret"}, inplace=True)

    # 2B) Setup subplots => 6 rows x 2 columns
    # (We have 6 entities in 'entities' list)
    fig, axes = plt.subplots(nrows=len(entities), ncols=2, figsize=(12,␣
 ↪4*len(entities)))

    if len(entities) == 1:
        axes = [axes]

    maxlag = 4
    lags = list(range(1, maxlag+1))

    for i, ent in enumerate(entities):
        ent_upper = ent.upper()
        ax_left  = axes[i][0]
        ax_right = axes[i][1]

        # 2C) Read entity returns
        df_ent = read_returns(ent)
        df_ent.rename(columns={"ret": f"{ent}_ret"}, inplace=True)

        # Merge with SPY
        df_merged = pd.merge(df_ent, df_spy, on="Date", how="inner").dropna()
        if len(df_merged) < 50:
            ax_left.set_title(f"{ent_upper} vs. SPY (Not enough data)")
            ax_right.set_title(f"{ent_upper} => SPY (Not enough data)")
            continue

        # 2D) Rolling correlation (30-day)
        df_merged["roll_corr_30"] = df_merged[f"{ent}_ret"].rolling(window=30).
 ↪corr(df_merged["spy_ret"])
```

```
        ax_left.plot(df_merged["Date"], df_merged["roll_corr_30"],␣
↪color="blue", label="30d Corr")
        ax_left.axhline(0, color="black", linestyle="--", alpha=0.5)
        ax_left.set_title(f"{ent_upper} vs. SPY - Rolling Corr")
        ax_left.legend(loc="upper left")

        # 2E) Granger test: ent => SPY
        # cause=ent_ret, effect=spy_ret => columns => [spy_ret, ent_ret]
        data_for_test = df_merged[["spy_ret", f"{ent}_ret"]].dropna()
        if len(data_for_test) < 30:
            ax_right.set_title(f"{ent_upper} => SPY (Too few points)")
            continue

        results = grangercausalitytests(data_for_test, maxlag=maxlag)
        pvals = []
        print(f"\n=== {ent_upper} => SPY Granger ===")
        for lag in lags:
            f_val, p_val, df_denom, df_num = results[lag][0]["ssr_ftest"]
            pvals.append(p_val)
            print(f"Lag {lag}: p-value={p_val:.4f}, F={f_val:.3f}")

        # 2F) Bar chart of p-values
        ax_right.bar(lags, pvals, color="orange", alpha=0.7)
        ax_right.axhline(0.05, color="red", linestyle="--", alpha=0.7)  #␣
↪significance line
        ax_right.set_xticks(lags)
        ax_right.set_ylim(0, 1)
        ax_right.set_title(f"{ent_upper} => SPY (Granger p-vals)")
        for j, pv in enumerate(pvals):
            ax_right.text(lags[j], pv+0.02, f"{pv:.2g}", ha="center")

    plt.tight_layout()
    plt.subplots_adjust(hspace=0.35, wspace=0.3)
    plt.show()

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

ARIMA and GARCH Modeling: Forecasting Returns and Volatility across Entities

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
from statsmodels.tsa.stattools import adfuller
```

```
import warnings
from arch.__future__ import reindexing

warnings.filterwarnings("ignore")

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

entities = ["xom", "cvx", "shell", "bp", "spy", "brent", "wti"]

def get_table_and_col(ent):
    if ent in ["brent", "wti"]:
        return "brent_wti_data", "Brent_Close" if ent == "brent" else␣
 ↪"WTI_Close"
    return f"{ent}_data", "Close"

def read_returns(ent, start_date="2019-01-01", end_date="2025-01-31"):
    table_name, col_name = get_table_and_col(ent)
    query = f"""
    SELECT Date, {col_name} AS raw_close
    FROM {table_name}
    WHERE Date >= '{start_date}' AND Date <= '{end_date}' AND {col_name} IS NOT␣
 ↪NULL
    ORDER BY Date
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date"], inplace=True)
    df.sort_values("Date", inplace=True)
    df["raw_close"] = pd.to_numeric(df["raw_close"], errors="coerce")
    df.dropna(subset=["raw_close"], inplace=True)
    df["daily_return"] = df["raw_close"].pct_change()
    df.dropna(subset=["daily_return"], inplace=True)
    return df

def prepare_for_modeling(df, scale_threshold=0.01):
    df = df.set_index("Date").asfreq("B")
    df["daily_return"] = df["daily_return"].ffill()
    ts = df["daily_return"].dropna()
    scale_factor = 1.0
    if ts.std() < scale_threshold:
        scale_factor = 100.0
        ts *= scale_factor
    return ts, scale_factor

def check_stationarity(ts):
    return adfuller(ts.dropna())[1] < 0.05
```

```
def fit_arima(ts, ent):
    try:
        if not check_stationarity(ts):
            print(f" {ent.upper()} is non-stationary. Applying differencing.")
            ts = ts.diff().dropna()
        model = ARIMA(ts, order=(1, 0, 1), enforce_stationarity=False,␣
 ↪enforce_invertibility=False)
        model_fit = model.fit(method_kwargs={"maxiter": 3000})
        forecast = model_fit.forecast(steps=5)
        print(f"\n=== {ent.upper()} ARIMA(1,0,1) ===")
        print(model_fit.pvalues)
        print("\nForecast (5 days):")
        print(forecast)
        return model_fit, forecast
    except Exception as e:
        print(f" ARIMA failed for {ent.upper()}: {e}")
        return None, None

def fit_garch(ts, ent):
    try:
        ts_resid = ts - ts.mean()
        garch_model = arch_model(ts_resid, p=1, q=1, mean='Constant',␣
 ↪vol='GARCH', dist='normal')
        garch_res = garch_model.fit(disp='off', update_freq=5)
        cond_vol = garch_res.conditional_volatility
        print(f"\n=== {ent.upper()} GARCH(1,1) P-Values ===")
        print(garch_res.pvalues)
        return garch_res, cond_vol
    except Exception as e:
        print(f" GARCH failed for {ent.upper()}: {e}")
        return None, None

def main():
    plot_start, plot_end = "2023-01-01", "2025-01-31"
    fig, axes = plt.subplots(nrows=len(entities), ncols=2, figsize=(14, 4 *␣
 ↪len(entities)))
    if len(entities) == 1:
        axes = [axes]
    for i, ent in enumerate(entities):
        ent_upper = ent.upper()
        ax_arima, ax_garch = axes[i]
        print(f"\n Processing: {ent_upper}")
        df = read_returns(ent)
        if df.empty or len(df) < 50:
            ax_arima.set_title(f"{ent_upper} - Not enough data")
            ax_garch.set_title(f"{ent_upper} - Not enough data")
```

```
            continue
        ts, scale_factor = prepare_for_modeling(df)
        if len(ts) < 50:
            ax_arima.set_title(f"{ent_upper} - Insufficient post-processed␣
↪data")
            ax_garch.set_title(f"{ent_upper} - Insufficient post-processed␣
↪data")
            continue

        model_fit, forecast = fit_arima(ts, ent)
        if model_fit:
            ts_plot = ts.loc[plot_start:plot_end]
            ax_arima.plot(ts_plot.index, ts_plot, color="gray", label="Returns")
            if forecast is not None and not forecast.empty:
                future_idx = pd.date_range(start=ts_plot.index[-1],␣
↪periods=len(forecast) + 1, freq="B")[1:]
                ax_arima.plot(future_idx, forecast, color="red",␣
↪label="Forecast")
            ax_arima.set_title(f"{ent_upper} - ARIMA(1,0,1)")
            ax_arima.legend(loc="upper left")

        garch_res, cond_vol = fit_garch(ts, ent)
        if garch_res is not None:
            cond_vol.index = ts.index
            cond_vol_plot = cond_vol.loc[plot_start:plot_end]
            ax_garch.plot(cond_vol_plot.index, cond_vol_plot, color="blue",␣
↪label="Cond. Volatility")
            ax_garch.set_title(f"{ent_upper} - GARCH(1,1)")
            ax_garch.legend(loc="upper left")

    plt.tight_layout()
    plt.subplots_adjust(hspace=0.4, wspace=0.4)
    plt.show()

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

Interactive Candlestick Charts: Visualizing OHLC Patterns for Stocks and Oil Benchmarks

[ ]:
```
import os
import pandas as pd
import plotly.graph_objects as go

# Folder where cleaned CSVs are stored
FDS_FOLDER = "FDS project"

# File mappings
```

```
files = {
    'XOM': 'XOM_data_cleaned.csv',
    'CVX': 'CVX_data_cleaned.csv',
    'BP': 'BP_data_cleaned.csv',
    'SHELL': 'SHEL_data_cleaned.csv',
    'SPY': 'SPY_data_cleaned.csv',
    'Brent': 'brent_WTI_data.csv',
    'WTI': 'brent_WTI_data.csv'
}

# Column mappings
stock_cols = {
    'open': 'Open',
    'high': 'High',
    'low': 'Low',
    'close': 'Close'
}

oil_cols = {
    'Brent': {
        'open': 'Brent_Open',
        'high': 'Brent_High',
        'low': 'Brent_Low',
        'close': 'Brent_Close'
    },
    'WTI': {
        'open': 'WTI_Open',
        'high': 'WTI_High',
        'low': 'WTI_Low',
        'close': 'WTI_Close'
    }
}


for ticker, filename in files.items():
    file_path = os.path.join(FDS_FOLDER, filename)

    if not os.path.isfile(file_path):
        print(f" File not found for {ticker}: {filename}")
        continue

    df = pd.read_csv(file_path)

    # Ensure datetime is parsed correctly
    df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d', errors='coerce')
    df.dropna(subset=['Date'], inplace=True)
    df.sort_values(by='Date', inplace=True)
```

```
    # Get correct OHLC columns
    cols = oil_cols[ticker] if ticker in oil_cols else stock_cols

    if not all(col in df.columns for col in [cols['open'], cols['high'],␣
↪cols['low'], cols['close']]):
        print(f" Missing OHLC columns in {ticker}. Skipping...")
        continue

    fig = go.Figure()

    fig.add_trace(go.Candlestick(
        x=df['Date'],
        open=df[cols['open']],
        high=df[cols['high']],
        low=df[cols['low']],
        close=df[cols['close']],
        increasing_line_color='green',
        decreasing_line_color='red',
        showlegend=False
    ))

    fig.add_trace(go.Scatter(
        x=[None], y=[None],
        mode='lines',
        line=dict(color='green', width=3),
        name='Price Up (Green Candle)'
    ))

    fig.add_trace(go.Scatter(
        x=[None], y=[None],
        mode='lines',
        line=dict(color='red', width=3),
        name='Price Down (Red Candle)'
    ))

    fig.update_layout(
        title=f'{ticker} - Candlestick Chart',
        xaxis_title='Date',
        yaxis_title='Price',
        xaxis_rangeslider_visible=False,
        template='plotly_white',
        legend=dict(
            title='Legend',
            orientation='h',
            yanchor='bottom',
            y=1.02,
```

```
            xanchor='center',
            x=0.5,
            font=dict(size=12),
            bordercolor='LightGray',
            borderwidth=1
        )
    )

    fig.show()
```

Appendix B Section 5 - Textual Analysis

April 20, 2025

Sentiment Analysis and Topic Modeling on News Articles using VADER, TextBlob, and LDA

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
from gensim import corpora, models
from nltk.corpus import stopwords
from sqlalchemy import create_engine

# Downloads
nltk.download("vader_lexicon", quiet=True)
nltk.download("stopwords", quiet=True)
stop_words = set(stopwords.words("english"))

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

# Entity to Table Map
entities_tables = {
    "xom": "xom_news", "cvx": "chevron_news", "shell": "shell_news",
    "bp": "bp_news", "spy": "spy_news", "brent": "brent_news", "wti": "wti_news"
}

start_date = "2019-01-01"
end_date = "2025-01-31"

# Initialize VADER
sia = SentimentIntensityAnalyzer()

# Read news
def read_news(table, start_date, end_date):
    query = f"""
    SELECT published_date, title, summary
```

```
    FROM {table}
    WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    ORDER BY published_date
    """
    df_news = pd.read_sql(query, engine)
    df_news["published_date"] = pd.to_datetime(df_news["published_date"],␣
 ↪errors="coerce")
    df_news.dropna(subset=["published_date"], inplace=True)
    df_news.sort_values("published_date", inplace=True)
    df_news["Date"] = df_news["published_date"].dt.date
    return df_news

# Compute sentiment scores
def compute_sentiment(df_news):
    vader_scores, textblob_scores = [], []

    for _, row in df_news.iterrows():
        text = str(row["title"]) + " " + str(row["summary"])

        # VADER
        vs = sia.polarity_scores(text)
        vader_scores.append(vs["compound"])

        # TextBlob
        try:
            polarity = TextBlob(text).sentiment.polarity
        except:
            polarity = 0.0
        textblob_scores.append(polarity)

    df_news["vader_compound"] = vader_scores
    df_news["textblob_polarity"] = textblob_scores
    return df_news

# Weekly aggregation
def compute_weekly_sentiment(df_news):
    df_news["Date"] = pd.to_datetime(df_news["Date"])
    df_news.set_index("Date", inplace=True)

    weekly_sent = df_news.resample("W-MON").agg({
        "vader_compound": "mean",
        "textblob_polarity": "mean"
    }).dropna().reset_index()

    return weekly_sent

# Topic modeling
```

```
def do_topic_modeling(df_news, text_column="title", num_topics=5):
    texts = []
    for text in df_news[text_column].fillna(""):
        text_clean = re.sub(r'[^a-zA-Z\s]', '', text.lower())
        tokens = [w for w in text_clean.split() if len(w) > 2 and w not in␣
 ↪stop_words]
        if tokens:
            texts.append(tokens)

    if len(texts) < 5:
        print("Not enough articles for topic modeling.")
        return

    dictionary = corpora.Dictionary(texts)
    dictionary.filter_extremes(no_below=2, no_above=0.8)
    corpus = [dictionary.doc2bow(t) for t in texts]

    if len(dictionary) == 0:
        print("No valid tokens after filtering. Skipping LDA.")
        return

    lda_model = models.LdaModel(corpus=corpus, id2word=dictionary,␣
 ↪num_topics=num_topics,
                                random_state=42, passes=5)

    print(f"\nTop {num_topics} topics:")
    for i in range(num_topics):
        topic = lda_model.show_topic(i, topn=5)
        terms = ", ".join([term for term, _ in topic])
        print(f"  • Topic {i+1}: {terms}")

# Main loop
def main():
    for ent, table_name in entities_tables.items():
        ent_upper = ent.upper()
        print(f"\n==============================")
        print(f" PROCESSING {ent_upper} NEWS")
        print(f"==============================")

        # Step 1: Read
        df_news = read_news(table_name, start_date, end_date)
        if df_news.empty:
            print("No news found.")
            continue

        print(f"{len(df_news)} articles loaded.")
```

```
        # Step 2: Sentiment
        df_news = compute_sentiment(df_news)

        #  Save daily sentiment
        daily_path = f"{ent}_daily_sentiment.csv"
        df_news.to_csv(daily_path, index=False)
        print(f"Daily sentiment saved to: {daily_path}")

        print("\nSample Daily Sentiment:")
        print(df_news[["published_date", "title", "vader_compound",␣
 ↪"textblob_polarity"]].head(3))

        # Step 3: Weekly sentiment
        weekly_sent = compute_weekly_sentiment(df_news)

        #  Save weekly sentiment
        weekly_path = f"{ent}_weekly_sentiment.csv"
        weekly_sent.to_csv(weekly_path, index=False)
        print(f"Weekly sentiment saved to: {weekly_path}")

        print(f"\nWeekly Sentiment Preview:")
        print(weekly_sent.head(3))

        # Step 4: Topics
        do_topic_modeling(df_news, text_column="title", num_topics=5)

        print(f"Finished {ent_upper}")

# Run it
if __name__ == "__main__":
    main()
```

Entity-wise Daily Sentiment Analysis with VADER & TextBlob and Topic Modeling Visualization

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
from sqlalchemy import create_engine

# Sentiment
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob

# Topic Modeling
from gensim import corpora, models
```

```
# Setup
nltk.download("vader_lexicon", quiet=True)

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

# Entity → News table
entities_tables = {
    "xom": "xom_news", "cvx": "chevron_news", "shell": "shell_news",
    "bp": "bp_news", "spy": "spy_news", "brent": "brent_news", "wti": "wti_news"
}

start_date = "2019-01-01"
end_date   = "2025-01-31"

sia = SentimentIntensityAnalyzer()

# --- Read News ---
def read_news(table, start_date, end_date):
    query = f"""
    SELECT published_date, title, summary
    FROM {table}
    WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    ORDER BY published_date
    """
    df = pd.read_sql(query, engine)
    df["published_date"] = pd.to_datetime(df["published_date"], errors="coerce")
    df.dropna(subset=["published_date"], inplace=True)
    df.sort_values("published_date", inplace=True)
    return df

# --- Compute Sentiment ---
def compute_sentiment(df_news):
    vader_scores = []
    textblob_scores = []

    for _, row in df_news.iterrows():
        text = str(row["title"]) + " " + str(row["summary"])

        # VADER
        vs = sia.polarity_scores(text)
        vader_scores.append(vs["compound"])

        # TextBlob (with fallback)
        try:
```

```
            polarity = TextBlob(text).sentiment.polarity
        except:
            polarity = 0.0
        textblob_scores.append(polarity)

    df_news["vader_compound"] = vader_scores
    df_news["textblob_polarity"] = textblob_scores
    return df_news

# --- Topic Modeling ---
def do_topic_modeling(df_news, text_column="title", num_topics=10, topn=5):
    texts = []
    for text in df_news[text_column].fillna(""):
        text_clean = re.sub(r'[^a-zA-Z\s]', '', text.lower())
        tokens = [w for w in text_clean.split() if len(w) > 2]
        texts.append(tokens)

    if len(texts) < 5:
        return None

    dictionary = corpora.Dictionary(texts)
    dictionary.filter_extremes(no_below=2, no_above=0.8)
    corpus = [dictionary.doc2bow(t) for t in texts]

    if len(dictionary) == 0:
        return None

    lda_model = models.LdaModel(
        corpus=corpus,
        id2word=dictionary,
        num_topics=num_topics,
        random_state=42,
        passes=5
    )

    topic_list = []
    for i in range(num_topics):
        terms = lda_model.show_topic(i, topn=topn)
        topic_terms = ", ".join([t[0] for t in terms])
        topic_list.append(f"Topic {i+1}: {topic_terms}")

    return topic_list

# --- Weekly Aggregation ---
def aggregate_weekly_sentiment(df_news):
    df_news["Date"] = df_news["published_date"].dt.date
    df_news["Date"] = pd.to_datetime(df_news["Date"])  # Ensure datetime format
```

```
    df_news.set_index("Date", inplace=True)

    weekly = df_news.resample("W-MON")[["vader_compound", "textblob_polarity"]].
 ↪mean().dropna().reset_index()
    return weekly

# --- Main ---
def main():
    rows, cols = 4, 2
    fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(14, 4 * rows))
    ax_list = axes.flatten()

    entity_keys = list(entities_tables.keys())

    for i, ent in enumerate(entity_keys):
        if i >= len(ax_list):
            break

        ax = ax_list[i]
        ent_upper = ent.upper()
        table_name = entities_tables[ent]

        print(f"\n=== PROCESSING {ent_upper} ===")

        df_news = read_news(table_name, start_date, end_date)
        if df_news.empty:
            print(f"No news for {ent_upper}.")
            ax.set_title(f"{ent_upper}: No news data")
            continue

        print(f"{len(df_news)} articles read.")

        df_news = compute_sentiment(df_news)

        # Save daily sentiment (optional)
        df_news.to_csv(f"{ent}_daily_sentiment.csv", index=False)

        # Weekly Aggregation
        weekly_sent = aggregate_weekly_sentiment(df_news)
        if weekly_sent.empty:
            print(f"No weekly sentiment for {ent_upper}")
            ax.set_title(f"{ent_upper}: No weekly sentiment data")
            continue

        # Save weekly sentiment
        weekly_sent.to_csv(f"{ent}_weekly_sentiment.csv", index=False)
        print(f"Saved weekly sentiment to {ent}_weekly_sentiment.csv")
```

```
        # Plot sentiment
        ax.plot(weekly_sent["Date"], weekly_sent["vader_compound"],␣
 ↪label="VADER", color="blue")
        ax.plot(weekly_sent["Date"], weekly_sent["textblob_polarity"],␣
 ↪label="TextBlob", color="red")
        ax.set_title(f"{ent_upper} - Weekly Avg Sentiment")
        ax.set_xlabel("Date")
        ax.set_ylabel("Sentiment Score")
        ax.legend(loc="upper left")

        # Topic Modeling
        topics = do_topic_modeling(df_news, text_column="title", num_topics=5,␣
 ↪topn=4)
        topics_str = "\n".join(topics) if topics else "No valid topics"
        ax.text(0.02, 0.95, topics_str, transform=ax.transAxes,
                fontsize=8, va='top', bbox=dict(facecolor='white', alpha=0.7))

    # Remove unused axes
    for j in range(len(entity_keys), len(ax_list)):
        fig.delaxes(ax_list[j])

    plt.tight_layout()
    plt.show()

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

Dual-Axis Visualization of Stock Price vs. News Sentiment for Each Entity

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sqlalchemy import create_engine
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Setup
nltk.download("vader_lexicon", quiet=True)
sia = SentimentIntensityAnalyzer()
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

entities_info = {
    "xom":   {"news_table": "xom_news",     "stock_table": "xom_data"},
    "cvx":   {"news_table": "chevron_news", "stock_table": "cvx_data"},
```

```
    "shell": {"news_table": "shell_news",   "stock_table": "shell_data"},
    "bp":    {"news_table": "bp_news",      "stock_table": "bp_data"},
    "spy":   {"news_table": "spy_news",     "stock_table": "spy_data"},
    "brent": {"news_table": "brent_news",   "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "Brent_Close"},
    "wti":   {"news_table": "wti_news",     "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "WTI_Close"}
}

start_date = "2019-01-01"
end_date   = "2025-01-31"

# Weekly sentiment
def read_weekly_sentiment(entity_key):
    info = entities_info[entity_key]
    query = f"""
    SELECT published_date, title, summary
    FROM {info['news_table']}
    WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
    df["published_date"] = pd.to_datetime(df["published_date"], errors="coerce")
    df.dropna(subset=["published_date"], inplace=True)
    df["text"] = df["title"].astype(str) + " " + df["summary"].astype(str)
    df["vader_compound"] = df["text"].apply(lambda x: sia.
 ↪polarity_scores(x)["compound"])
    df.set_index("published_date", inplace=True)

    weekly_sent = df["vader_compound"].resample("W-MON").mean().reset_index()
    weekly_sent.rename(columns={"published_date": "Date", "vader_compound":␣
 ↪"avg_sentiment"}, inplace=True)
    return weekly_sent

# Weekly return
def read_weekly_return(entity_key):
    info = entities_info[entity_key]
    close_col = info.get("stock_close_col", "Close")
    query = f"""
    SELECT Date, {close_col} AS Close
    FROM {info['stock_table']}
    WHERE Date BETWEEN '{start_date}' AND '{end_date}' AND {close_col} IS NOT␣
 ↪NULL
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date", "Close"], inplace=True)
    df.set_index("Date", inplace=True)
```

```
    weekly_close = df["Close"].resample("W-MON").last()
    weekly_return = weekly_close.pct_change().dropna().reset_index()
    weekly_return.rename(columns={"Close": "weekly_return"}, inplace=True)
    return weekly_return

# Main plot
def main():
    rows, cols = 4, 2
    fig, axes = plt.subplots(rows, cols, figsize=(14, 4 * rows))
    ax_list = axes.flatten()

    for i, ent in enumerate(entities_info):
        if i >= len(ax_list): break
        ax = ax_list[i]
        ent_upper = ent.upper()

        print(f"\n=== {ent_upper} ===")

        sent_df = read_weekly_sentiment(ent)
        ret_df = read_weekly_return(ent)

        if sent_df.empty or ret_df.empty:
            print("Missing data.")
            ax.set_title(f"{ent_upper}: Missing data")
            continue

        merged = pd.merge(sent_df, ret_df, on="Date", how="inner").dropna()
        if merged.empty:
            print("No overlap.")
            ax.set_title(f"{ent_upper}: No overlap")
            continue

        # Print statistics
        print("Weekly Sentiment Stats:")
        print(sent_df["avg_sentiment"].describe()[["mean", "std", "min",␣
↪"max"]])

        print("\nWeekly Return Stats:")
        print(ret_df["weekly_return"].describe()[["mean", "std", "min", "max"]])

        # Plot: Dual Axis
        ax2 = ax.twinx()
        ax.plot(merged["Date"], merged["weekly_return"], color="blue",␣
↪label="Weekly Return")
        ax2.plot(merged["Date"], merged["avg_sentiment"], color="red",␣
↪label="Sentiment")
```

```
        ax.set_ylabel("Return (Blue)", color="blue")
        ax2.set_ylabel("Sentiment (Red)", color="red")
        ax.set_title(f"{ent_upper}: Weekly Return vs Sentiment")
        ax.tick_params(axis="y", labelcolor="blue")
        ax2.tick_params(axis="y", labelcolor="red")
        ax.set_xlabel("Week")

    for j in range(len(entities_info), len(ax_list)):
        fig.delaxes(ax_list[j])

    plt.tight_layout()
    plt.show()

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

OLS Regression: Predicting returns from News Sentiment (Test Set Comparison)

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import nltk

from sqlalchemy import create_engine
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download VADER lexicon (only once)
nltk.download("vader_lexicon", quiet=True)
sia = SentimentIntensityAnalyzer()

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

# Entity configuration
entities_info = {
    "xom":   {"news_table": "xom_news",       "stock_table": "xom_data"},
    "cvx":   {"news_table": "chevron_news",   "stock_table": "cvx_data"},
    "shell": {"news_table": "shell_news",     "stock_table": "shell_data"},
    "bp":    {"news_table": "bp_news",        "stock_table": "bp_data"},
    "spy":   {"news_table": "spy_news",       "stock_table": "spy_data"},
    "brent": {"news_table": "brent_news",     "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "Brent_Close"},
    "wti":   {"news_table": "wti_news",       "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "WTI_Close"}
```

```
}

start_date = "2019-01-01"
end_date   = "2025-01-31"

# --- Read & compute weekly sentiment ---
def read_news_and_sentiment(entity_key):
    info = entities_info[entity_key]
    news_table = info["news_table"]

    query = f"""
    SELECT published_date, title, summary
    FROM {news_table}
    WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    ORDER BY published_date
    """
    df_news = pd.read_sql(query, engine)
    df_news["published_date"] = pd.to_datetime(df_news["published_date"],␣
 ↪errors="coerce")
    df_news.dropna(subset=["published_date"], inplace=True)

    if df_news.empty:
        return pd.DataFrame(columns=["Date", "avg_sentiment"])

    df_news["vader_compound"] = df_news.apply(
        lambda row: sia.polarity_scores(str(row["title"]) + " " +␣
 ↪str(row["summary"]))["compound"],
        axis=1
    )

    # Convert to week-start date
    df_news["Date"] = df_news["published_date"].dt.to_period("W").apply(lambda␣
 ↪r: r.start_time)
    weekly_sent = df_news.groupby("Date")["vader_compound"].mean().
 ↪reset_index(name="avg_sentiment")

    return weekly_sent

# --- Read & compute weekly returns ---
def read_stock_data(entity_key):
    info = entities_info[entity_key]
    stock_table = info["stock_table"]
    close_col = info.get("stock_close_col", "Close")

    query = f"""
    SELECT Date, {close_col} AS Close
    FROM {stock_table}
```

```
    WHERE Date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date", "Close"], inplace=True)
    df.sort_values("Date", inplace=True)

    # Compute returns, then convert to weekly
    df["Returns"] = df["Close"].pct_change()
    df.dropna(subset=["Returns"], inplace=True)
    df["Date"] = df["Date"].dt.to_period("W").apply(lambda r: r.start_time)
    weekly = df.groupby("Date")["Returns"].mean().reset_index()

    return weekly

# --- Main analysis + plotting ---
def main():
    rows, cols = 4, 2
    fig, axes = plt.subplots(rows, cols, figsize=(14, 4 * rows))
    ax_list = axes.flatten()

    for i, ent in enumerate(entities_info):
        if i >= len(ax_list):
            break

        ax = ax_list[i]
        ent_upper = ent.upper()
        print(f"\n=== PROCESSING {ent_upper} ===")

        # Read sentiment and stock data
        sent_df = read_news_and_sentiment(ent)
        stock_df = read_stock_data(ent)

        if sent_df.empty or stock_df.empty:
            print(f"{ent_upper}: No usable data.")
            ax.set_title(f"{ent_upper}: No usable data")
            continue

        # Merge on weekly date
        merged = pd.merge(sent_df, stock_df, on="Date", how="inner").dropna()
        if merged.empty:
            print(f"{ent_upper}: No overlapping weekly data.")
            ax.set_title(f"{ent_upper}: No overlap")
            continue

        print(f"{ent_upper} - Weekly correlation: {merged['avg_sentiment'].
↪corr(merged['Returns']):.4f}")
```

```
        # Train-test split
        train_size = int(0.8 * len(merged))
        df_train = merged.iloc[:train_size]
        df_test  = merged.iloc[train_size:]

        # OLS Regression
        X_train = sm.add_constant(df_train["avg_sentiment"])
        y_train = df_train["Returns"]
        model = sm.OLS(y_train, X_train).fit()
        print(f"{ent_upper} OLS Summary:\n{model.summary()}")

        # Predict
        X_test = sm.add_constant(df_test["avg_sentiment"])
        y_test = df_test["Returns"]
        y_pred = model.predict(X_test)

        # Plot actual vs predicted returns
        ax.plot(df_test["Date"], y_test, label="Actual", color="black")
        ax.plot(df_test["Date"], y_pred, label="Predicted", color="blue")
        ax.set_title(f"{ent_upper} OLS: Weekly Returns ~ Sentiment")
        ax.set_xlabel("Date")
        ax.set_ylabel("Weekly Return")
        ax.legend()

        # Show R² on chart
        ax.text(0.05, 0.95, f"$R^2$ = {model.rsquared:.4f}", transform=ax.
 ↪transAxes,
                verticalalignment='top', fontsize=10,
                bbox=dict(boxstyle="round", facecolor="white", alpha=0.6))

    # Clean unused plots
    for j in range(len(entities_info), len(ax_list)):
        fig.delaxes(ax_list[j])

    plt.tight_layout()
    plt.show()

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

Rolling Correlation: Sentiment vs. Returns (30-Day Window)

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
```

```
from sqlalchemy import create_engine
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Setup
nltk.download("vader_lexicon", quiet=True)
sia = SentimentIntensityAnalyzer()

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

# Entity configuration
entities_info = {
    "xom":   {"news_table": "xom_news",       "stock_table": "xom_data"},
    "cvx":   {"news_table": "chevron_news",   "stock_table": "cvx_data"},
    "shell": {"news_table": "shell_news",     "stock_table": "shell_data"},
    "bp":    {"news_table": "bp_news",        "stock_table": "bp_data"},
    "spy":   {"news_table": "spy_news",       "stock_table": "spy_data"},
    "brent": {"news_table": "brent_news",     "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "Brent_Close"},
    "wti":   {"news_table": "wti_news",       "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "WTI_Close"}
}

start_date = "2019-01-01"
end_date   = "2025-01-31"

# 1. Read weekly sentiment
def read_weekly_sentiment(entity_key):
    info = entities_info[entity_key]
    news_table = info["news_table"]

    query = f"""
        SELECT published_date, title, summary
        FROM {news_table}
        WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
    df["published_date"] = pd.to_datetime(df["published_date"], errors="coerce")
    df.dropna(subset=["published_date"], inplace=True)

    if df.empty:
        return pd.DataFrame(columns=["Date", "avg_sentiment"])

    df["vader_compound"] = df.apply(
        lambda row: sia.polarity_scores(str(row["title"]) + " " +␣
 ↪str(row["summary"]))["compound"],
```

```
        axis=1
    )

    df.set_index("published_date", inplace=True)
    weekly_sent = df["vader_compound"].resample("W-MON").mean().reset_index()
    weekly_sent.rename(columns={"published_date": "Date", "vader_compound":␣
 ↪"avg_sentiment"}, inplace=True)

    return weekly_sent

# 2. Read weekly returns
def read_weekly_returns(entity_key):
    info = entities_info[entity_key]
    stock_table = info["stock_table"]
    close_col = info.get("stock_close_col", "Close")

    query = f"""
        SELECT Date, {close_col} AS Close
        FROM {stock_table}
        WHERE Date BETWEEN '{start_date}' AND '{end_date}'
          AND {close_col} IS NOT NULL
        ORDER BY Date
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date", "Close"], inplace=True)
    df.set_index("Date", inplace=True)

    weekly_close = df["Close"].resample("W-MON").last()
    weekly_returns = weekly_close.pct_change().dropna().reset_index()
    weekly_returns.rename(columns={"Close": "weekly_return"}, inplace=True)

    return weekly_returns

# 3. Main loop with rolling correlation
def main():
    rows, cols = 4, 2
    fig, axes = plt.subplots(rows, cols, figsize=(14, 4 * rows))
    ax_list = axes.flatten()
    rolling_window = 6  # 6-week rolling correlation

    for i, ent in enumerate(entities_info):
        if i >= len(ax_list): break

        ax = ax_list[i]
        ent_upper = ent.upper()
        print(f"\n=== {ent_upper} ===")
```

```
        sent_df = read_weekly_sentiment(ent)
        ret_df = read_weekly_returns(ent)

        if sent_df.empty or ret_df.empty:
            print(f"{ent_upper}: No sentiment or return data.")
            ax.set_title(f"{ent_upper}: No data")
            continue

        # Check and log overlap
        print(f"{ent_upper} - Sentiment dates: {sent_df['Date'].min()} to␣
↪{sent_df['Date'].max()}")
        print(f"{ent_upper} - Returns dates:   {ret_df['Date'].min()} to␣
↪{ret_df['Date'].max()}")

        merged = pd.merge(sent_df, ret_df, on="Date", how="inner").dropna()
        print(f"{ent_upper} - Overlapping rows: {len(merged)}")

        if merged.empty:
            ax.set_title(f"{ent_upper}: No overlap")
            continue

        # Rolling correlation
        merged["rolling_corr"] = merged["avg_sentiment"].
↪rolling(rolling_window).corr(merged["weekly_return"])
        rolled = merged.dropna(subset=["rolling_corr"])

        if rolled.empty:
            ax.set_title(f"{ent_upper}: No valid correlation")
            continue

        # Plot
        ax.plot(rolled["Date"], rolled["rolling_corr"], color="purple",␣
↪label=f"{rolling_window}-W Corr")
        ax.axhline(0, color="gray", linestyle="--", alpha=0.5)
        ax.set_title(f"{ent_upper} - {rolling_window}W Corr (Sentiment vs␣
↪Return)")
        ax.set_xlabel("Week")
        ax.set_ylabel("Correlation")
        ax.legend(loc="upper right", fontsize=8)

    for j in range(len(entities_info), len(ax_list)):
        fig.delaxes(ax_list[j])

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

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

Sentiment-Based Trading Strategy vs. Buy & Hold Performance

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Setup
nltk.download("vader_lexicon", quiet=True)
sia = SentimentIntensityAnalyzer()

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

# Entity setup
entities_info = {
    "xom":   {"news_table": "xom_news",     "stock_table": "xom_data"},
    "cvx":   {"news_table": "chevron_news", "stock_table": "cvx_data"},
    "shell": {"news_table": "shell_news",   "stock_table": "shell_data"},
    "bp":    {"news_table": "bp_news",      "stock_table": "bp_data"},
    "spy":   {"news_table": "spy_news",     "stock_table": "spy_data"},
    "brent": {"news_table": "brent_news",   "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "Brent_Close"},
    "wti":   {"news_table": "wti_news",     "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "WTI_Close"}
}

start_date = "2019-01-01"
end_date   = "2025-01-31"

# 1. Weekly Sentiment
def read_weekly_sentiment(entity_key):
    info = entities_info[entity_key]
    news_table = info["news_table"]

    query = f"""
        SELECT published_date, title, summary
        FROM {news_table}
        WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
```

```
    df["published_date"] = pd.to_datetime(df["published_date"], errors="coerce")
    df.dropna(subset=["published_date"], inplace=True)

    if df.empty:
        return pd.DataFrame(columns=["Date", "avg_sentiment"])

    df["vader_compound"] = df.apply(
        lambda row: sia.polarity_scores(str(row["title"]) + " " +␣
 ↪str(row["summary"]))["compound"],
        axis=1
    )

    df.set_index("published_date", inplace=True)
    weekly_sent = df["vader_compound"].resample("W-MON").mean().reset_index()
    weekly_sent.rename(columns={"published_date": "Date", "vader_compound":␣
 ↪"avg_sentiment"}, inplace=True)
    return weekly_sent

# 2. Weekly Returns
def read_weekly_returns(entity_key):
    info = entities_info[entity_key]
    stock_table = info["stock_table"]
    close_col = info.get("stock_close_col", "Close")

    query = f"""
        SELECT Date, {close_col} AS Close
        FROM {stock_table}
        WHERE Date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date", "Close"], inplace=True)
    df.set_index("Date", inplace=True)

    weekly_close = df["Close"].resample("W-MON").last()
    weekly_returns = weekly_close.pct_change().dropna().reset_index()
    weekly_returns.rename(columns={"Close": "weekly_return"}, inplace=True)
    return weekly_returns

# 3. Strategy Builder
def build_sentiment_strategy(merged, threshold=0.0):
    merged = merged.copy()
    merged["signal"] = (merged["avg_sentiment"] > threshold).astype(int)
    merged["signal_shift"] = merged["signal"].shift(1).fillna(0)
    merged["strategy_return"] = merged["signal_shift"] * merged["weekly_return"]
    merged["cum_strategy"] = (1 + merged["strategy_return"]).cumprod()
    merged["cum_buy_hold"] = (1 + merged["weekly_return"]).cumprod()
```

```
    return merged

# 4. Main loop
def main():
    rows, cols = 4, 2
    fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(14, 4*rows))
    ax_list = axes.flatten()

    entity_keys = list(entities_info.keys())
    threshold = 0.0  # Signal threshold

    for i, ent in enumerate(entity_keys):
        if i >= len(ax_list): break

        ax = ax_list[i]
        ent_upper = ent.upper()
        print(f"\n=== {ent_upper} ===")

        sent_df = read_weekly_sentiment(ent)
        ret_df = read_weekly_returns(ent)

        if sent_df.empty or ret_df.empty:
            print(f"{ent_upper}: No sentiment or return data.")
            ax.set_title(f"{ent_upper}: No data")
            continue

        merged = pd.merge(sent_df, ret_df, on="Date", how="inner").dropna()
        if merged.empty:
            print(f"{ent_upper}: No overlap between sentiment and returns.")
            ax.set_title(f"{ent_upper}: No overlap")
            continue

        strategy_df = build_sentiment_strategy(merged, threshold)

        final_strat = strategy_df["cum_strategy"].iloc[-1]
        final_bh    = strategy_df["cum_buy_hold"].iloc[-1]
        print(f"{ent_upper} Strategy Final: {final_strat:.2f}, Buy & Hold:␣
↪{final_bh:.2f}")

        # Plot
        ax.plot(strategy_df["Date"], strategy_df["cum_strategy"], color="blue",␣
↪label="Sentiment Strat")
        ax.plot(strategy_df["Date"], strategy_df["cum_buy_hold"], color="red",␣
↪label="Buy & Hold")
        ax.set_title(f"{ent_upper}: Strategy vs Buy & Hold (Weekly,␣
↪Thresh={threshold})")
        ax.set_ylabel("Cumulative Return")
```

```
        ax.legend(loc="upper left")

    for j in range(len(entity_keys), len(ax_list)):
        fig.delaxes(ax_list[j])

    plt.tight_layout()
    plt.show()

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

Lagged Sentiment Correlation & Mean Sentiment Analysis per Entity

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk

# === Setup ===
nltk.download("vader_lexicon", quiet=True)
sia = SentimentIntensityAnalyzer()
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")

entities_info = {
    "xom": {"news_table": "xom_news", "stock_table": "xom_data"},
    "cvx": {"news_table": "chevron_news", "stock_table": "cvx_data"},
    "shell": {"news_table": "shell_news", "stock_table": "shell_data"},
    "bp": {"news_table": "bp_news", "stock_table": "bp_data"},
    "spy": {"news_table": "spy_news", "stock_table": "spy_data"},
    "brent": {"news_table": "brent_news", "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "Brent_Close"},
    "wti": {"news_table": "wti_news", "stock_table": "brent_wti_data",␣
 ↪"stock_close_col": "WTI_Close"},
}

start_date = "2019-01-01"
end_date = "2025-01-31"

# === Functions ===
def get_weekly_sentiment(entity_key):
    info = entities_info[entity_key]
    query = f"""
    SELECT published_date, title, summary
    FROM {info['news_table']}
```

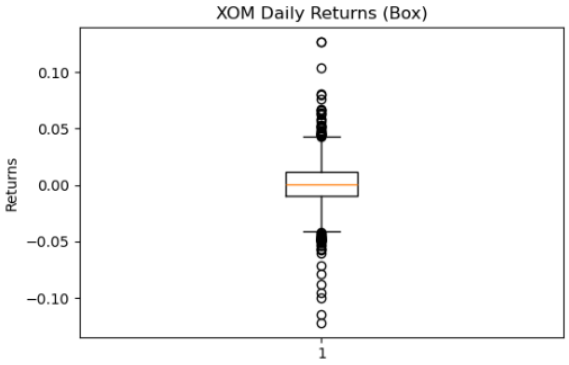
```
    WHERE published_date BETWEEN '{start_date}' AND '{end_date}'
    """
    df = pd.read_sql(query, engine)
    df["published_date"] = pd.to_datetime(df["published_date"], errors="coerce")
    df.dropna(subset=["published_date"], inplace=True)
    df["text"] = df["title"].astype(str) + " " + df["summary"].astype(str)
    df["vader_compound"] = df["text"].apply(lambda x: sia.
 ↪polarity_scores(x)["compound"])
    df.set_index("published_date", inplace=True)

    # Weekly sentiment (W-MON = week ends Monday)
    weekly_sent = df["vader_compound"].resample("W-MON").mean().reset_index()
    weekly_sent.rename(columns={"published_date": "Date", "vader_compound":␣
 ↪"avg_sentiment"}, inplace=True)
    return weekly_sent

def get_weekly_returns(entity_key):
    info = entities_info[entity_key]
    close_col = info.get("stock_close_col", "Close")
    query = f"""
    SELECT Date, {close_col} AS Close
    FROM {info['stock_table']}
    WHERE Date BETWEEN '{start_date}' AND '{end_date}' AND {close_col} IS NOT␣
 ↪NULL
    """
    df = pd.read_sql(query, engine)
    df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
    df.dropna(subset=["Date", "Close"], inplace=True)
    df.set_index("Date", inplace=True)

    # Weekly return from closing prices
    weekly_close = df["Close"].resample("W-MON").last()
    weekly_ret = weekly_close.pct_change().dropna().reset_index()
    weekly_ret.rename(columns={"Close": "weekly_return"}, inplace=True)
    return weekly_ret

# === Analysis ===
lagged_corrs = {}
mean_sentiments = {}

for ent in entities_info.keys():
    sent_df = get_weekly_sentiment(ent)
    ret_df = get_weekly_returns(ent)
    merged = pd.merge(sent_df, ret_df, on="Date", how="inner").dropna()

    if len(merged) > 10:
        merged["lagged_sentiment"] = merged["avg_sentiment"].shift(1)
```

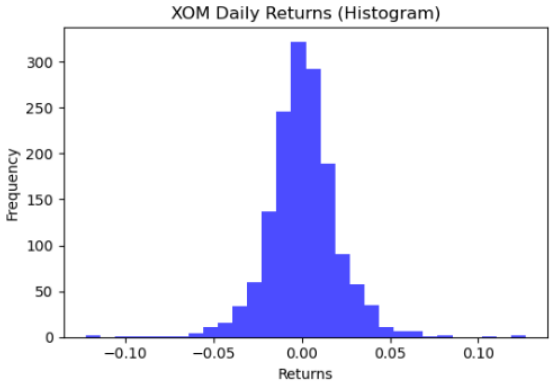
```
        merged.dropna(inplace=True)
        corr = merged["lagged_sentiment"].corr(merged["weekly_return"])
        lagged_corrs[ent.upper()] = round(corr, 3)
        mean_sentiments[ent.upper()] = round(merged["avg_sentiment"].mean(), 3)
    else:
        lagged_corrs[ent.upper()] = np.nan
        mean_sentiments[ent.upper()] = np.nan

# === Display Results ===
print("\n=== Weekly Lagged Correlation: Sentiment(t) vs Return(t+1) ===")
for k, v in lagged_corrs.items():
    print(f"{k}: {v}")

print("\n=== Weekly Mean Sentiment per Entity ===")
for k, v in mean_sentiments.items():
    print(f"{k}: {v}")

# === Plotting ===
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 7))

entities = list(lagged_corrs.keys())
corr_values = list(lagged_corrs.values())
sent_values = list(mean_sentiments.values())

# Plot 1: Lagged Correlation
bars1 = ax1.bar(entities, corr_values, color='cornflowerblue')
ax1.axhline(0, color='black', linestyle='--', linewidth=1)
ax1.set_title("Weekly Lagged Correlation: Sentiment(t) → Return(t+1)",␣
 ↪fontsize=14)
ax1.set_ylabel("Correlation", fontsize=12)
ax1.set_xlabel("Entity", fontsize=12)
ax1.tick_params(axis='x', rotation=45)

for bar in bars1:
    height = bar.get_height()
    if not np.isnan(height):
        ax1.text(bar.get_x() + bar.get_width()/2, height + (0.003 * np.
 ↪sign(height)),
                 f'{height:.2f}', ha='center', va='bottom', fontsize=9)

# Plot 2: Mean Weekly Sentiment
bars2 = ax2.bar(entities, sent_values, color='lightcoral')
ax2.axhline(0, color='black', linestyle='--', linewidth=1)
ax2.set_title("Average Weekly Sentiment Score per Entity", fontsize=14)
ax2.set_ylabel("Avg VADER Sentiment", fontsize=12)
ax2.set_xlabel("Entity", fontsize=12)
ax2.tick_params(axis='x', rotation=45)
```

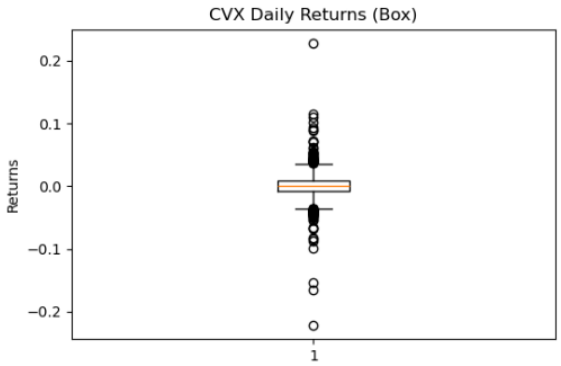
```
for bar in bars2:
    height = bar.get_height()
    if not np.isnan(height):
        ax2.text(bar.get_x() + bar.get_width()/2, height + (0.003 * np.
 ↪sign(height)),
                 f'{height:.2f}', ha='center', va='bottom', fontsize=9)

plt.suptitle("Weekly Sentiment Predictive Power & Polarity Summary",␣
 ↪fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.subplots_adjust(top=0.88)
plt.show()
```

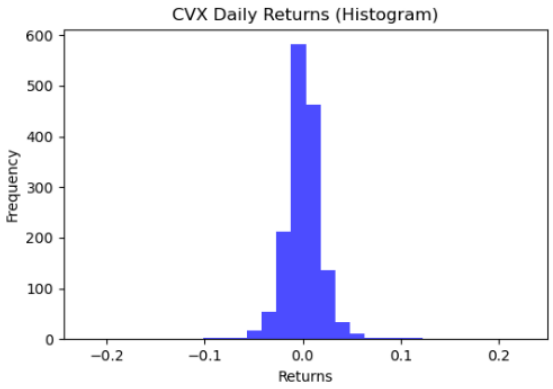
Appendix C

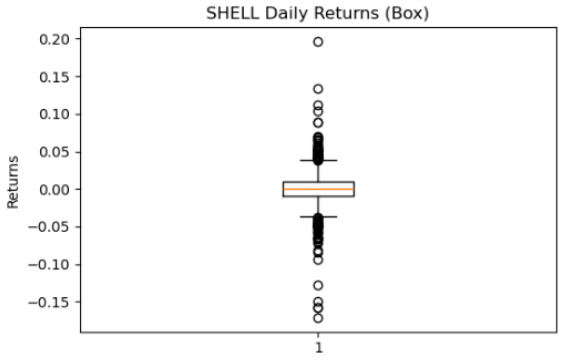
Daily Return Distributions

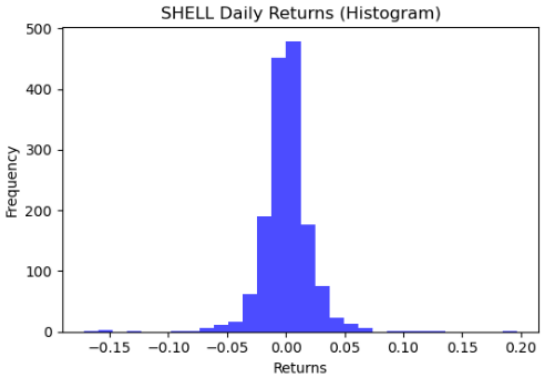


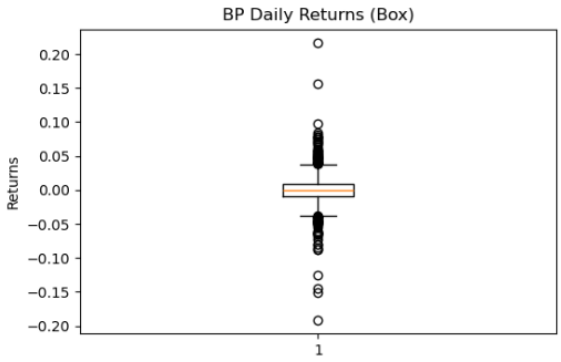


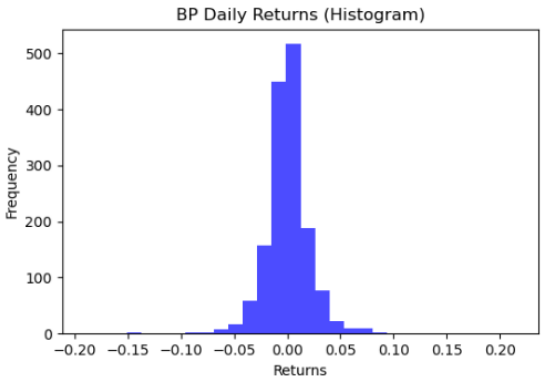


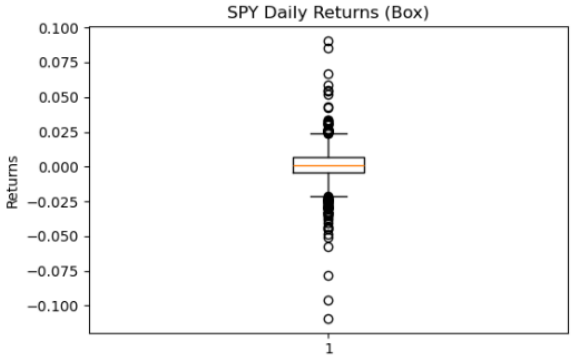


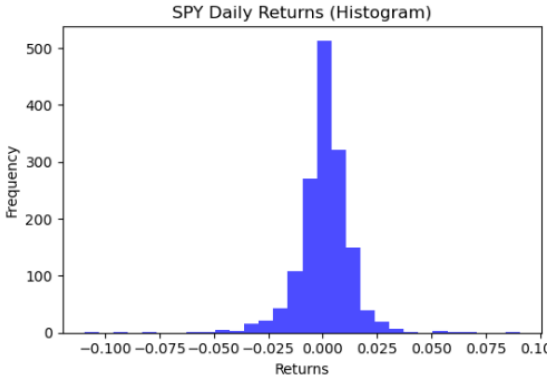


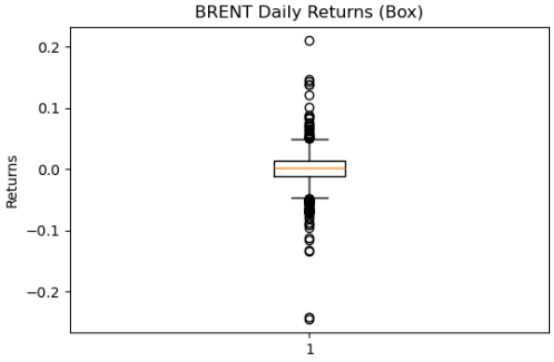


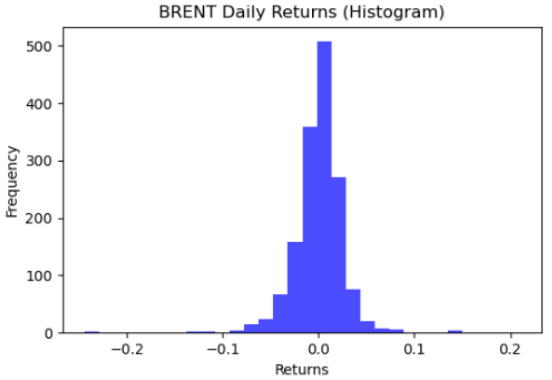


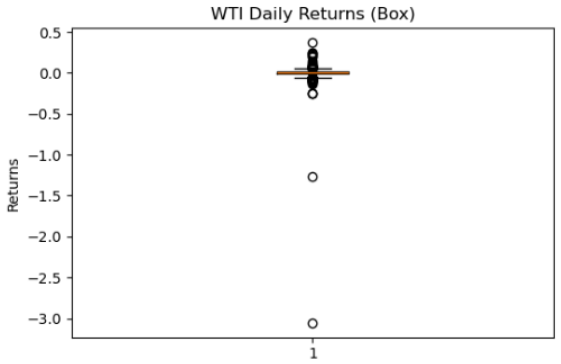


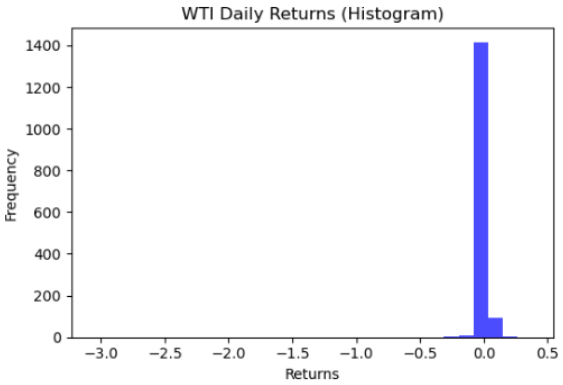

SPY Daily Returns (Box)
Returns
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
1


SPY Daily Returns (Histogram)
Frequency
500
400
300
200
100
0
−0.100 −0.075 −0.050 −0.025 0.000 0.025 0.050 0.075
Returns

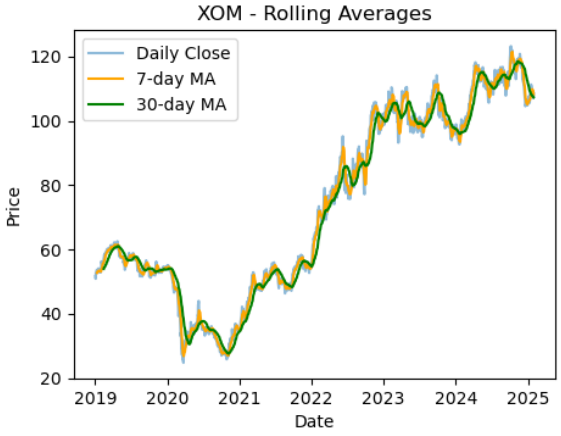

BRENT Daily Returns (Box)
Returns
0.2
0.1
0.0
−0.1
−0.2
1

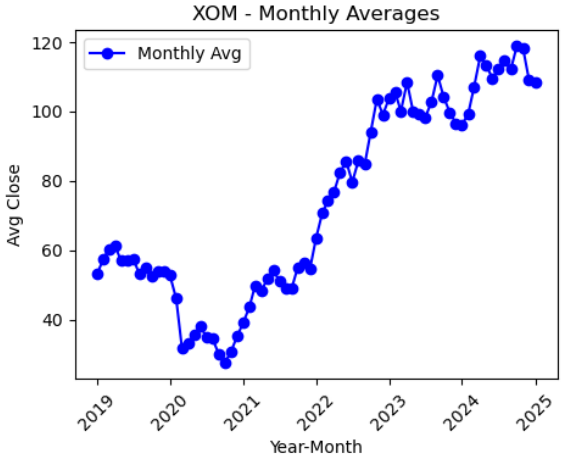

BRENT Daily Returns (Histogram)
Frequency
500
400
300
200
100
0
−0.2
−0.1
0.0
0.1
0.2
Returns

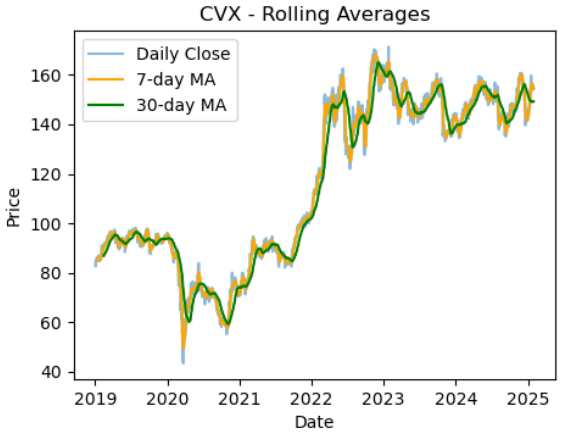

WTI Daily Returns (Box)
Returns
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
1

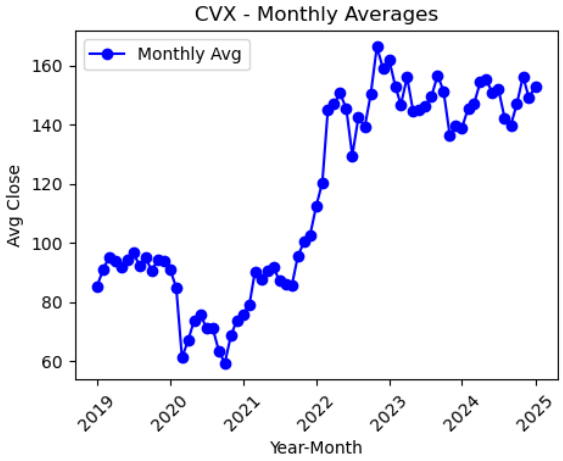

WTI Daily Returns (Histogram)
Frequency
1400
1200
1000
800
600
400
200
0
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
Returns

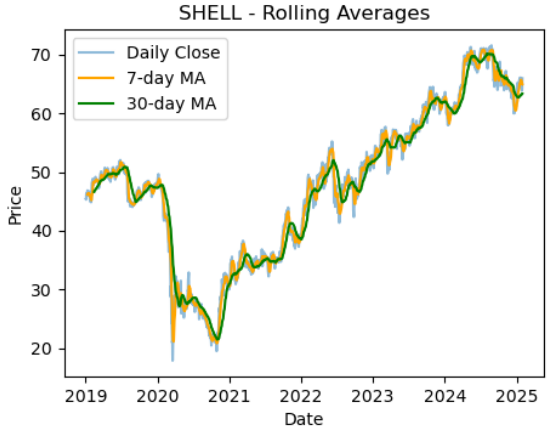
Rolling and Monthly Averages

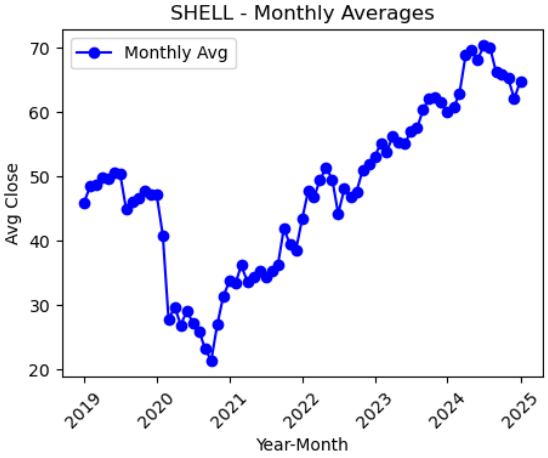


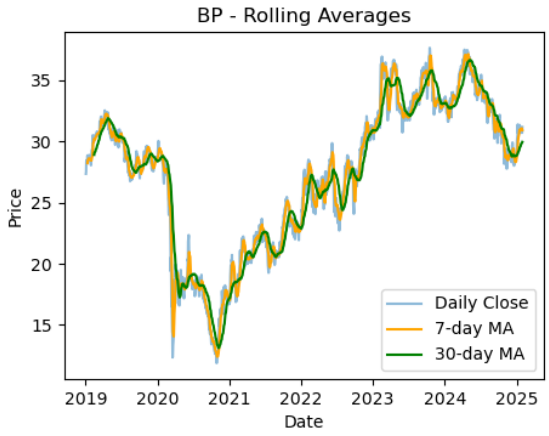


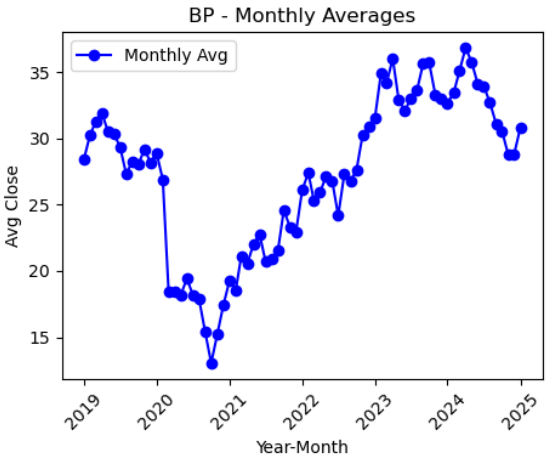


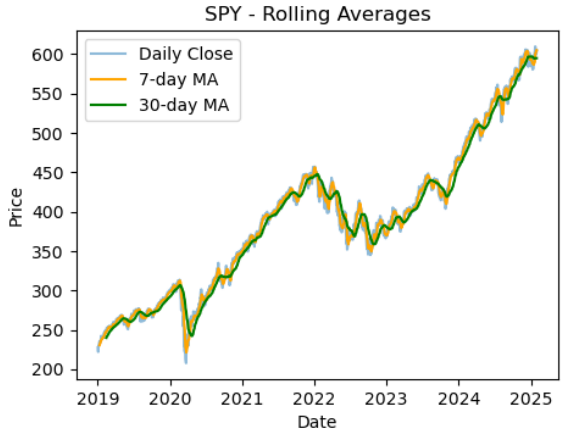


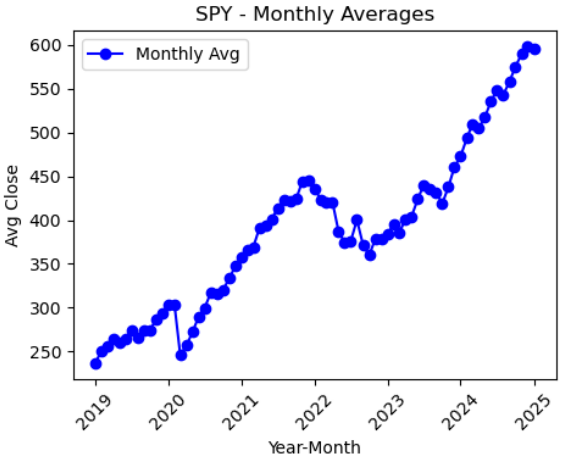


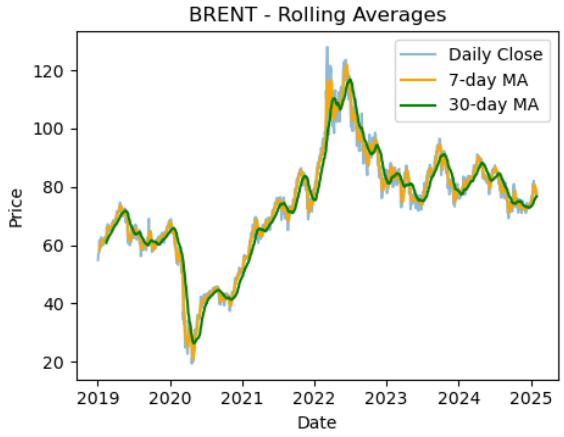


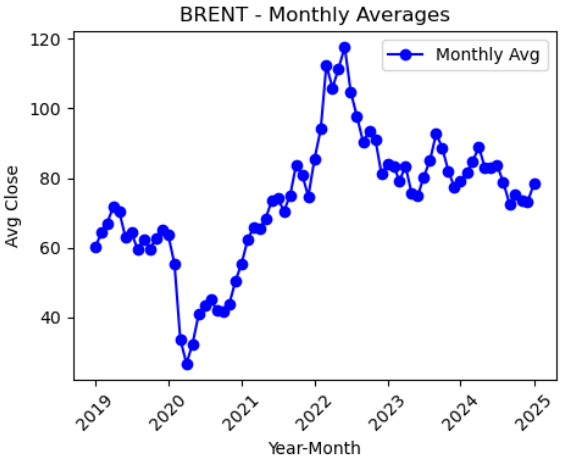


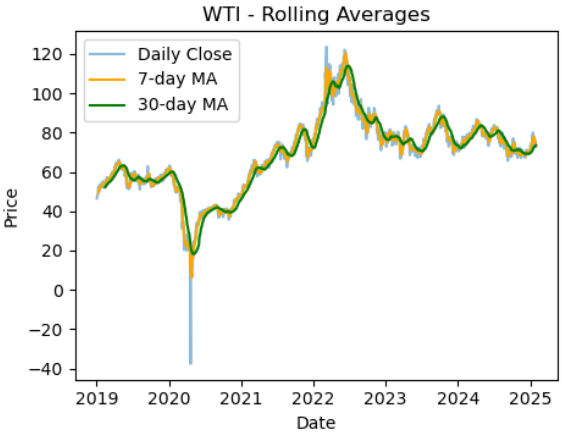


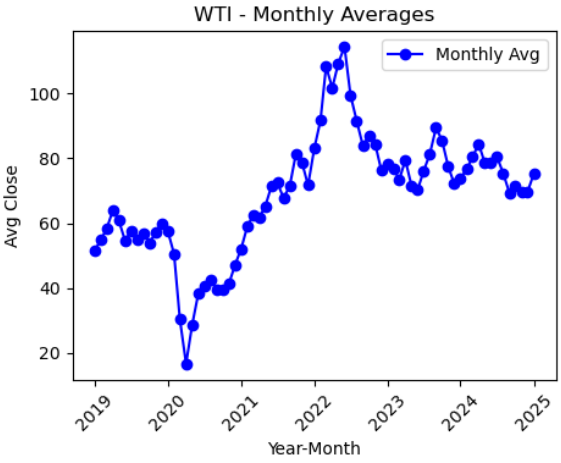

SPY - Rolling Averages
Daily Close
7-day MA
30-day MA
Price
Date


SPY - Monthly Averages
Monthly Avg
Avg Close
Year-Month

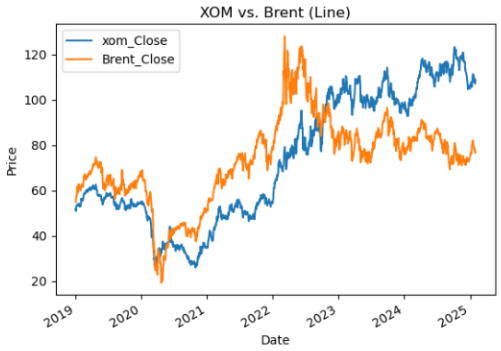

BRENT - Rolling Averages
Daily Close
7-day MA
30-day MA
Price
Date

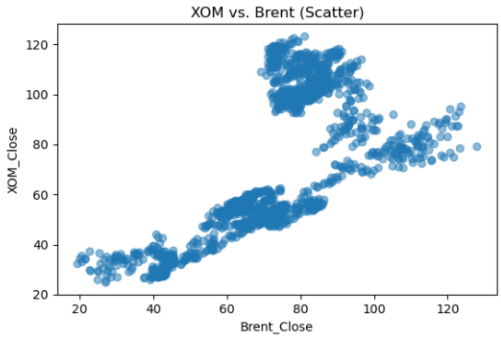

BRENT - Monthly Averages
Monthly Avg
Avg Close
Year-Month

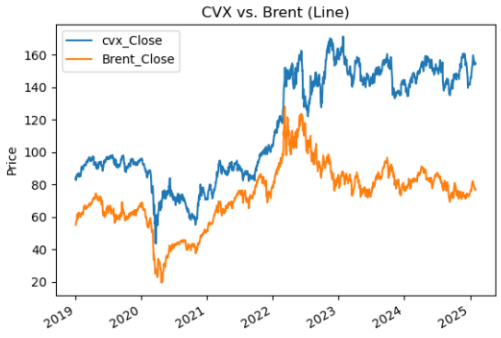

WTI - Rolling Averages
Daily Close
7-day MA
30-day MA
Price
Date

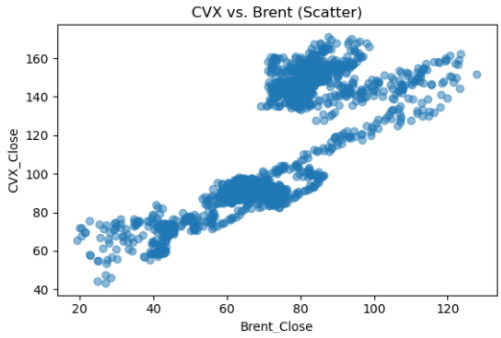

WTI - Monthly Averages
Monthly Avg
Avg Close
Year-Month

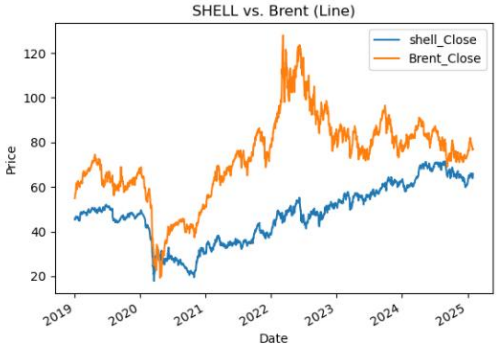
Line and Scatter Plots Energy Stocks

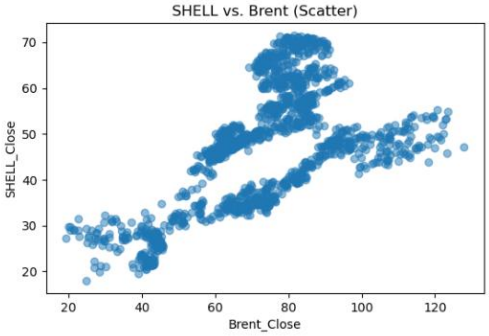


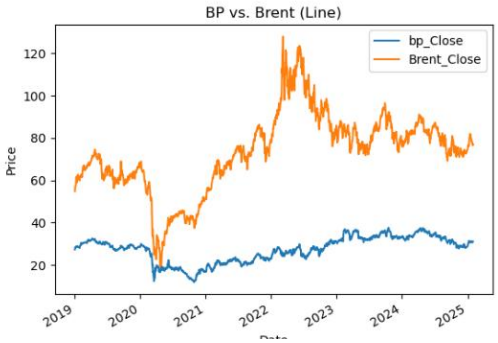


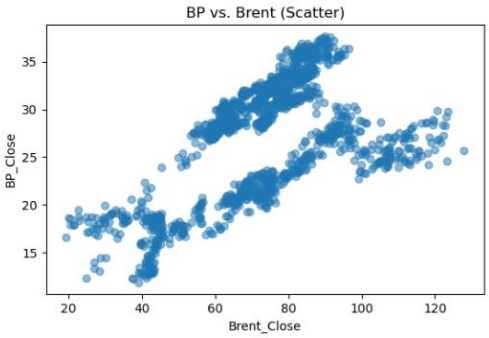


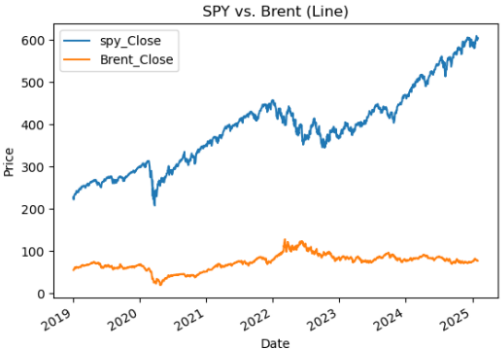


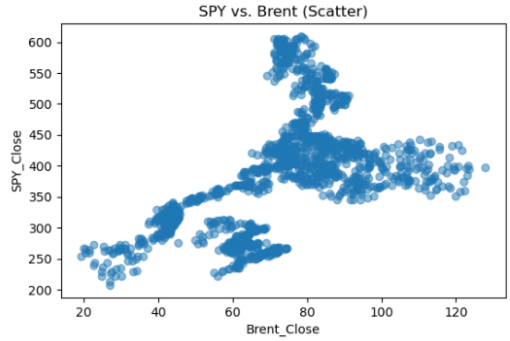




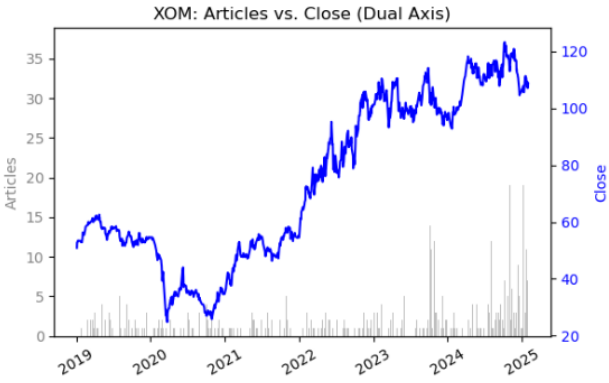


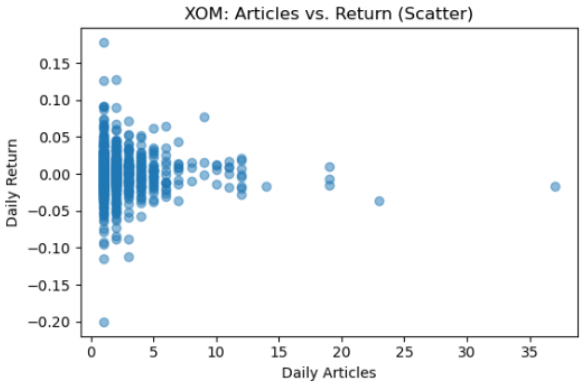


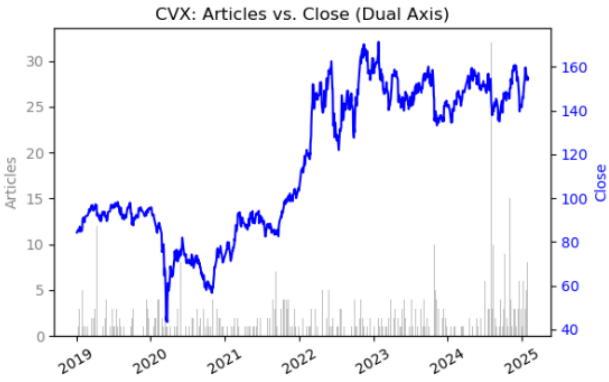


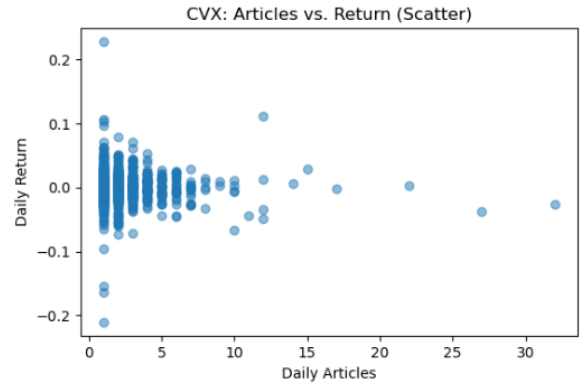


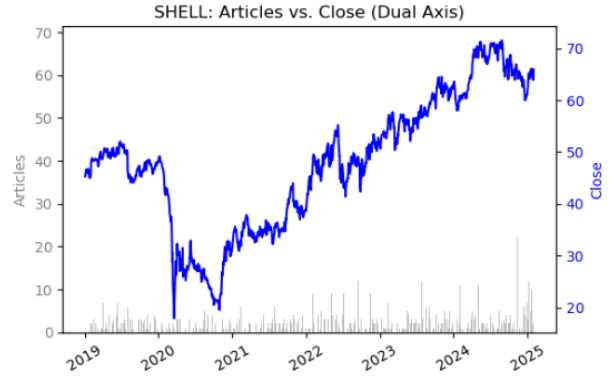
News Volume and Return Volatility

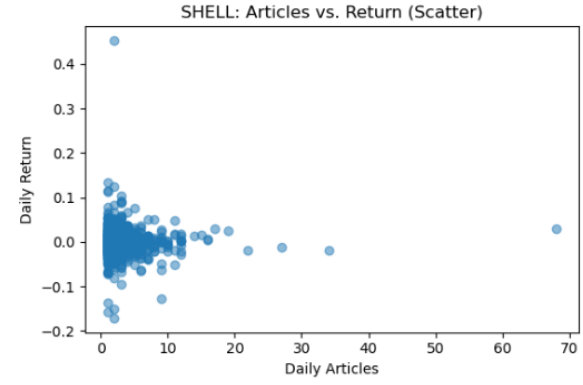


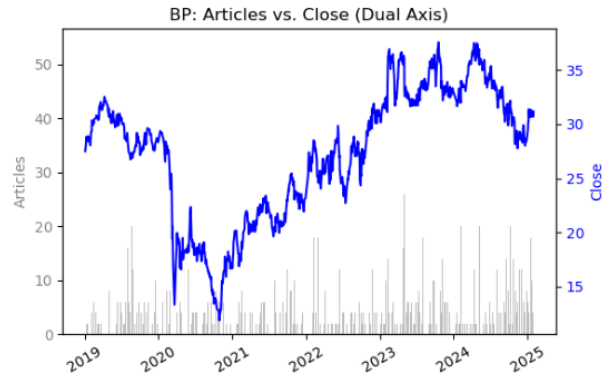


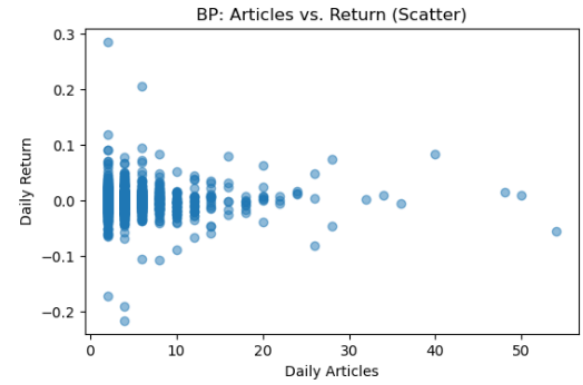


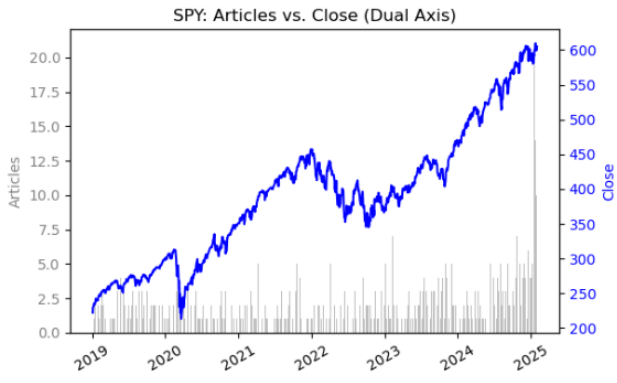


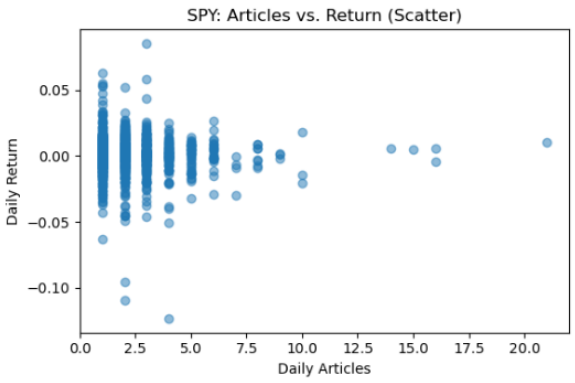


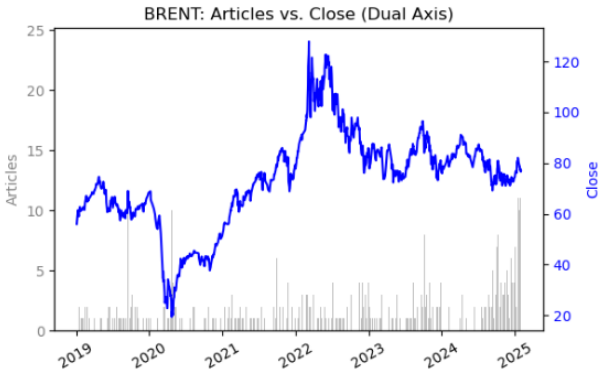


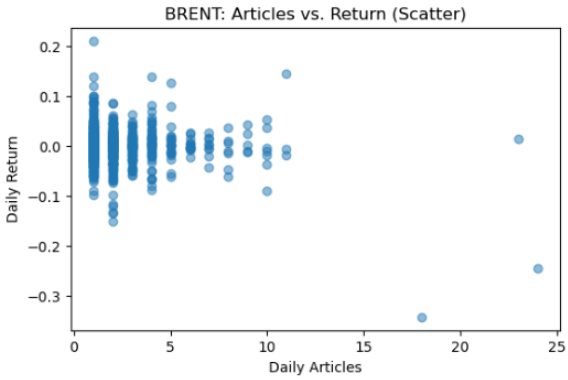


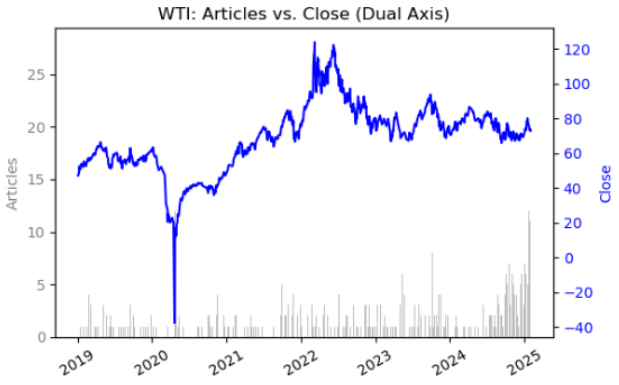


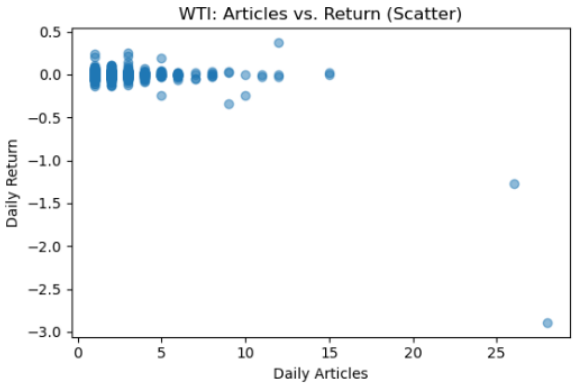

SPY: Articles vs. Close (Dual Axis)
Articles
Close
2019
2020
2021
2022
2023
2024
2025


SPY: Articles vs. Return (Scatter)
Daily Return
Daily Articles

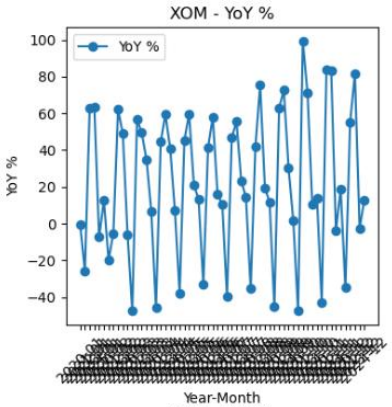

BRENT: Articles vs. Close (Dual Axis)
Articles
Close
2019
2020
2021
2022
2023
2024
2025

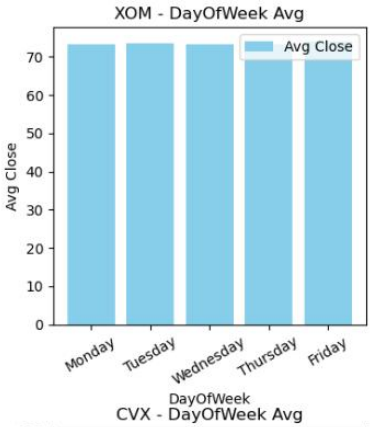

BRENT: Articles vs. Return (Scatter)
Daily Return
Daily Articles

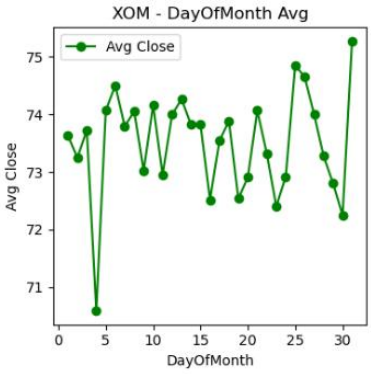

WTI: Articles vs. Close (Dual Axis)
Articles
Close
2019
2020
2021
2022
2023
2024
2025

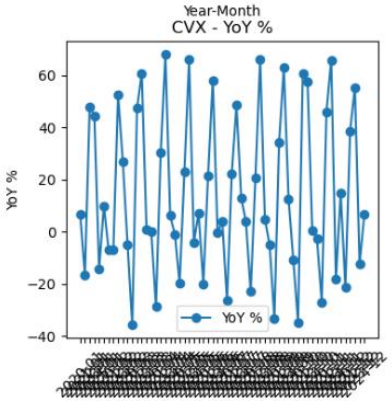

WTI: Articles vs. Return (Scatter)
Daily Return
Daily Articles

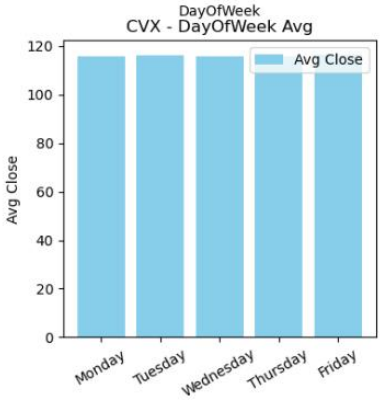
Calendar Patterns

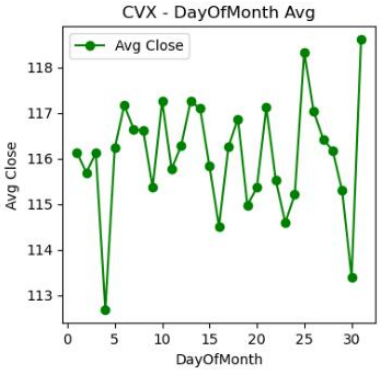


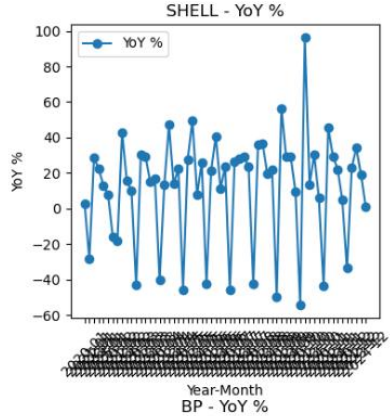


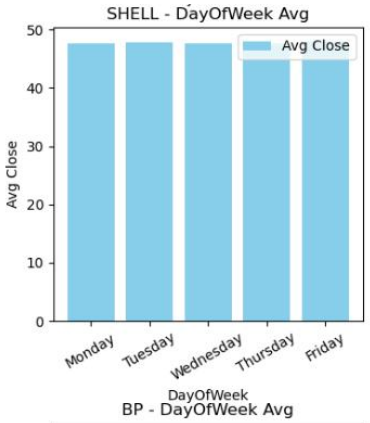


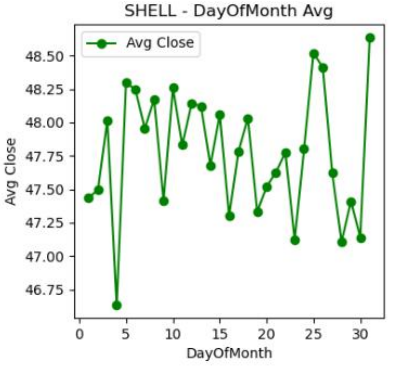


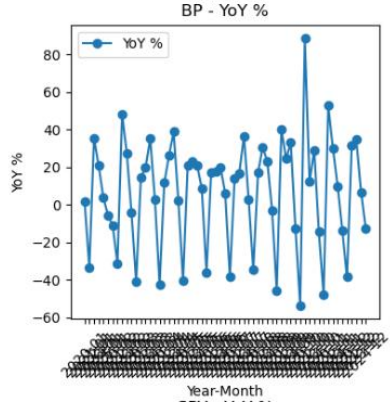


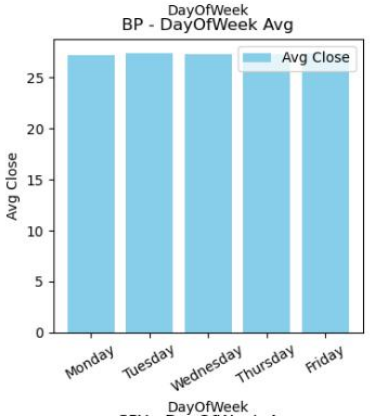


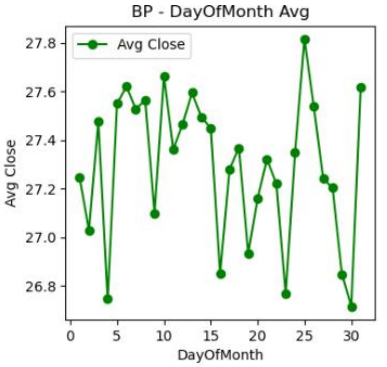


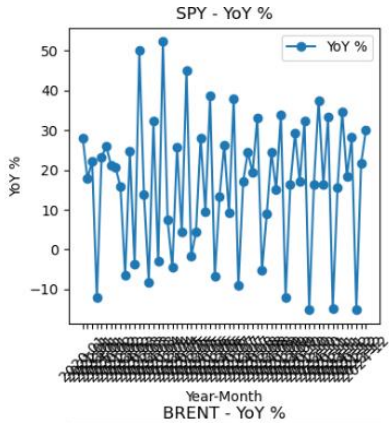


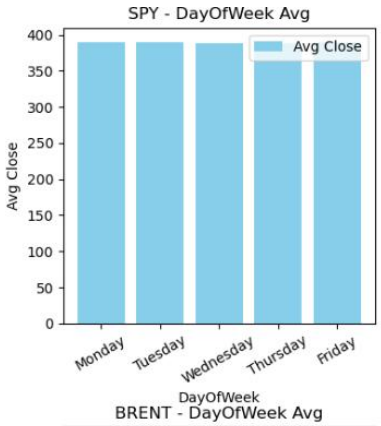


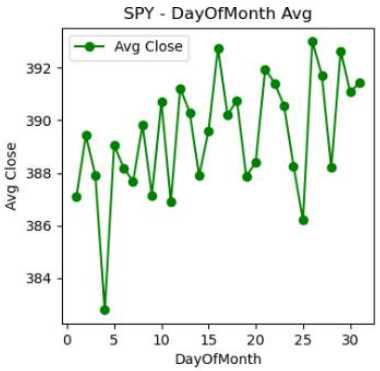


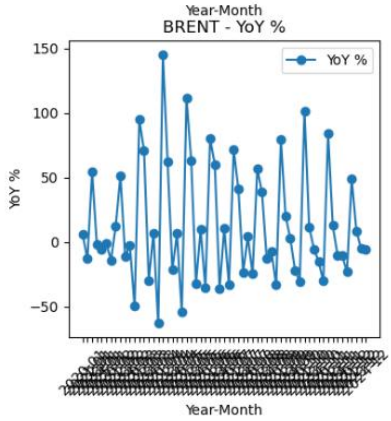


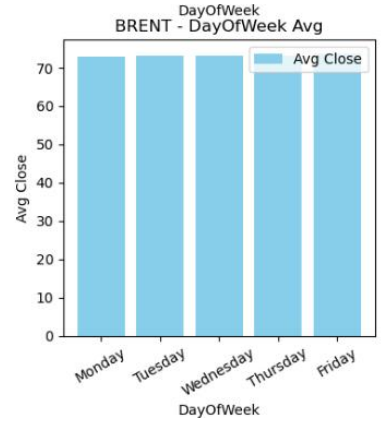


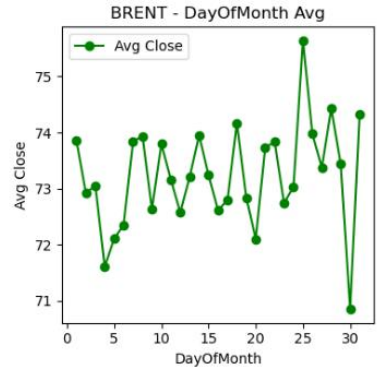

SPY - YoY %
YoY %
Year-Month

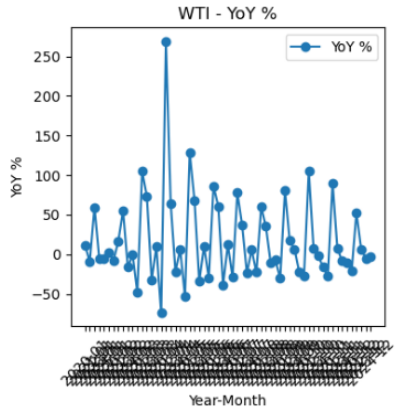

SPY - DayOfWeek Avg
Avg Close
Monday
Tuesday
Wednesday
Thursday
Friday
DayOfWeek

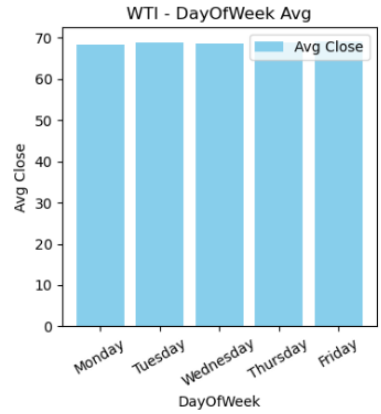

SPY - DayOfMonth Avg
Avg Close
DayOfMonth

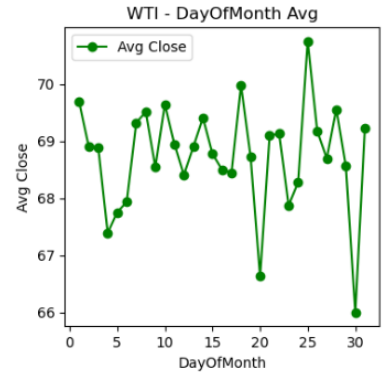

BRENT - YoY %
YoY %
Year-Month


BRENT - DayOfWeek Avg
Avg Close
Monday
Tuesday
Wednesday
Thursday
Friday
DayOfWeek

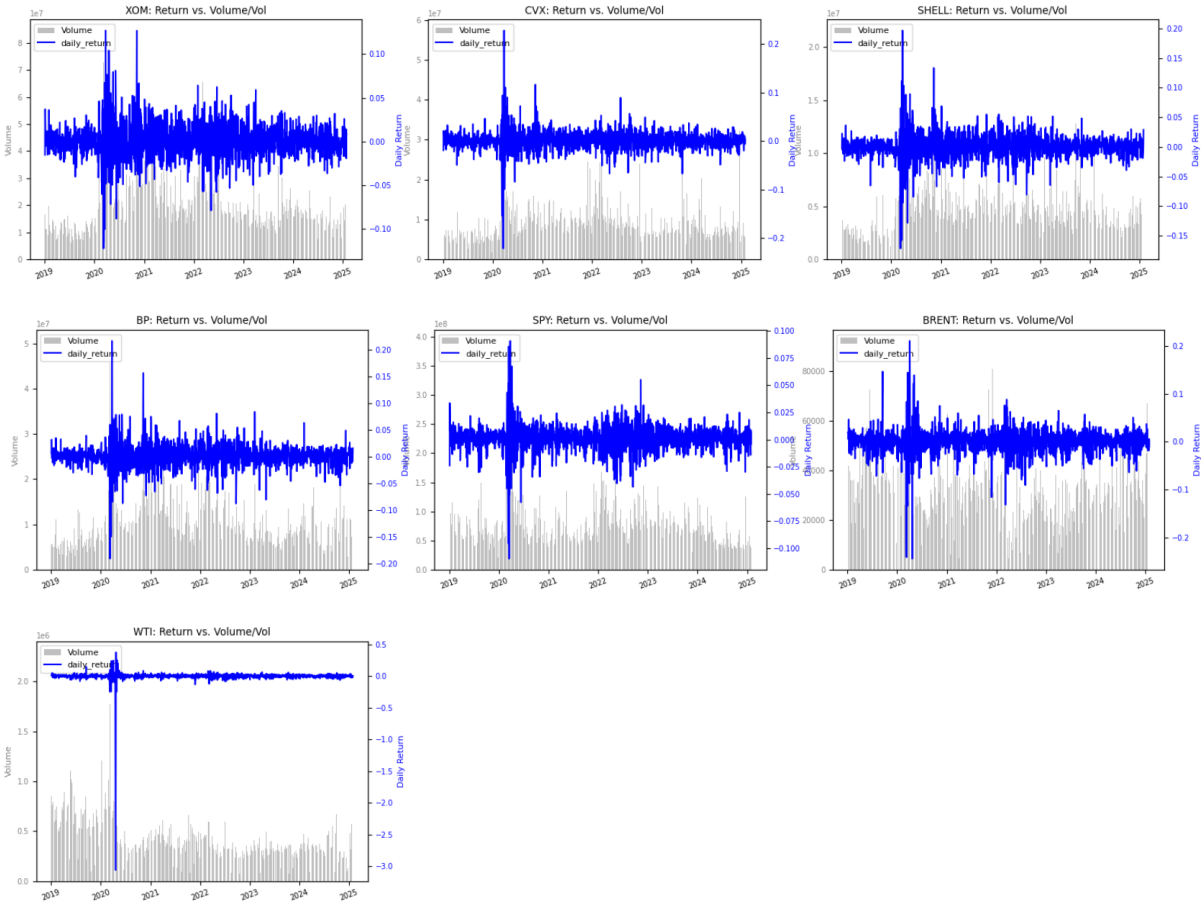

BRENT - DayOfMonth Avg
Avg Close
DayOfMonth


WTI - YoY %
YoY %
Year-Month

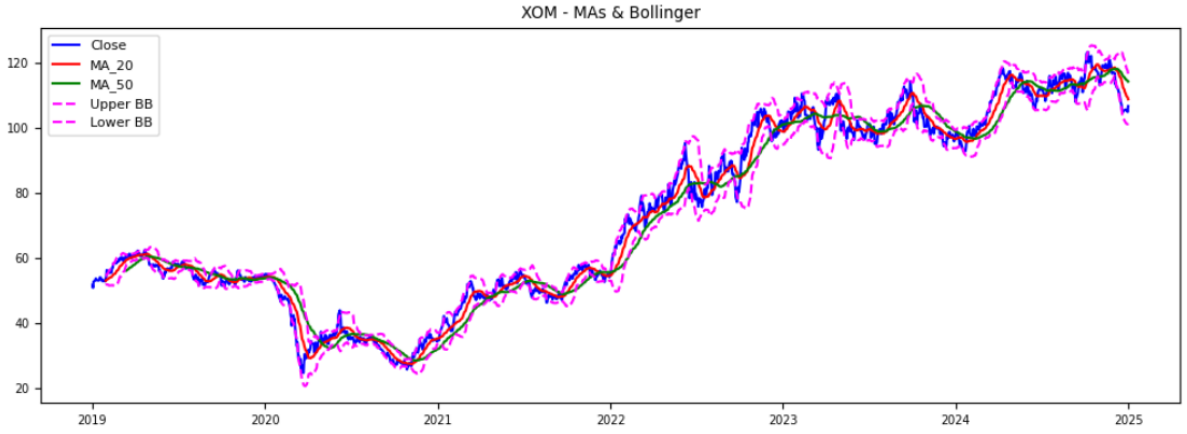

WTI - DayOfWeek Avg
Avg Close
Monday
Tuesday
Wednesday
Thursday
Friday
DayOfWeek

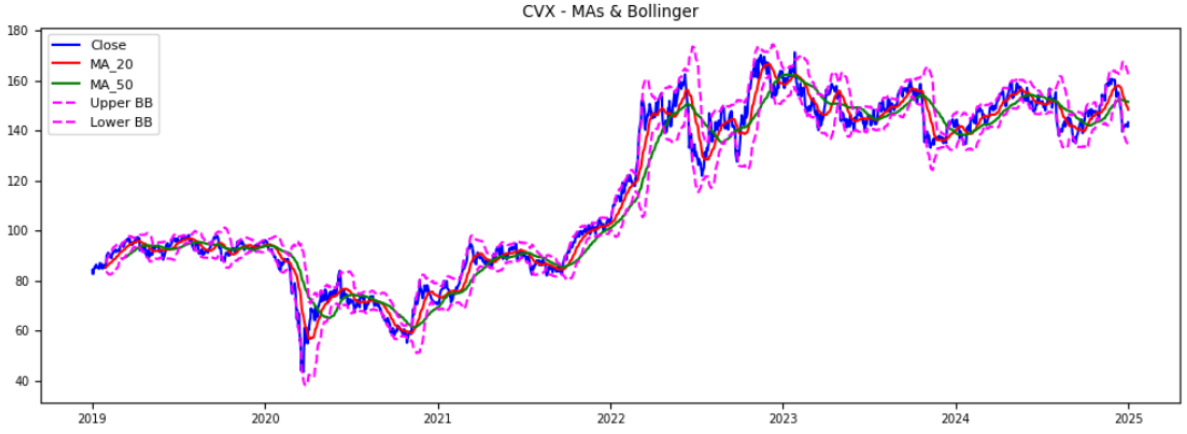

WTI - DayOfMonth Avg
Avg Close
DayOfMonth

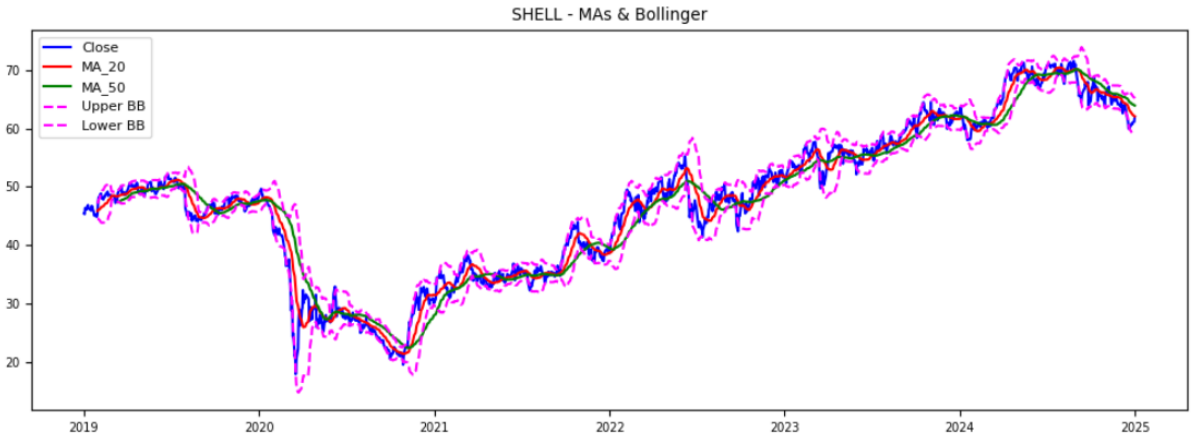
Returns vs Volume/Volatility

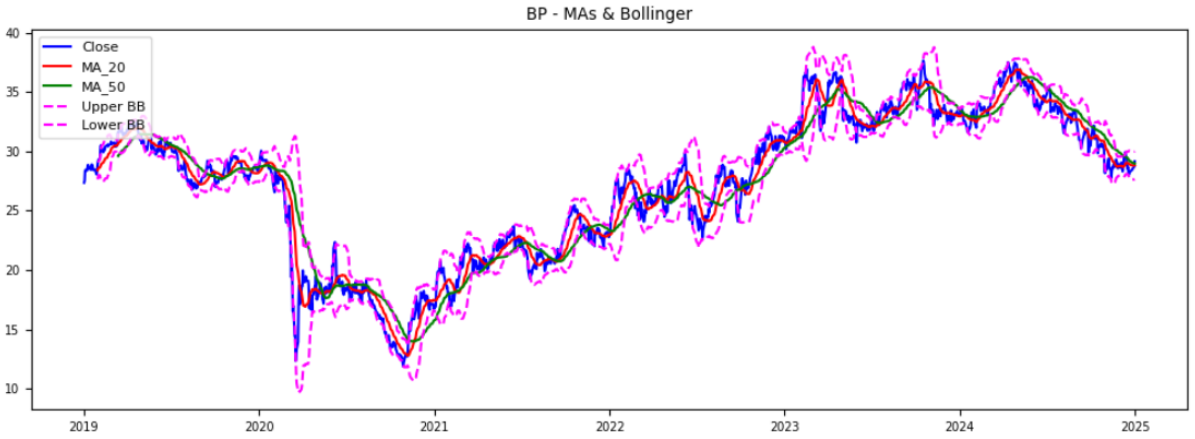


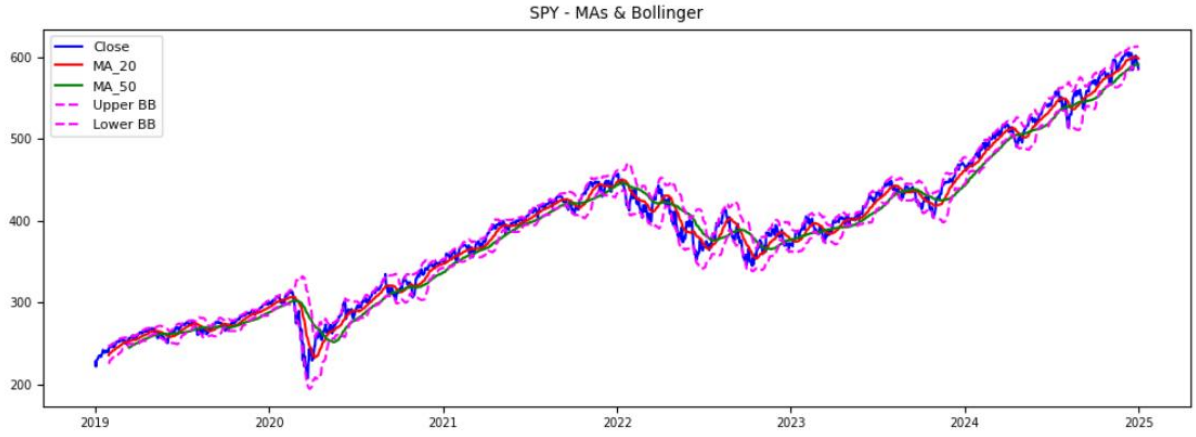
Moving Averages and Bollinger Bands

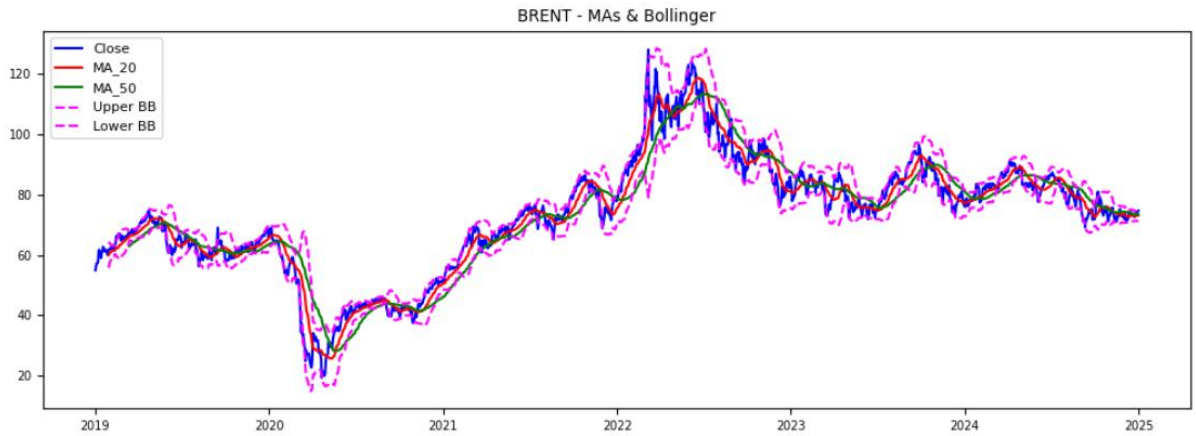


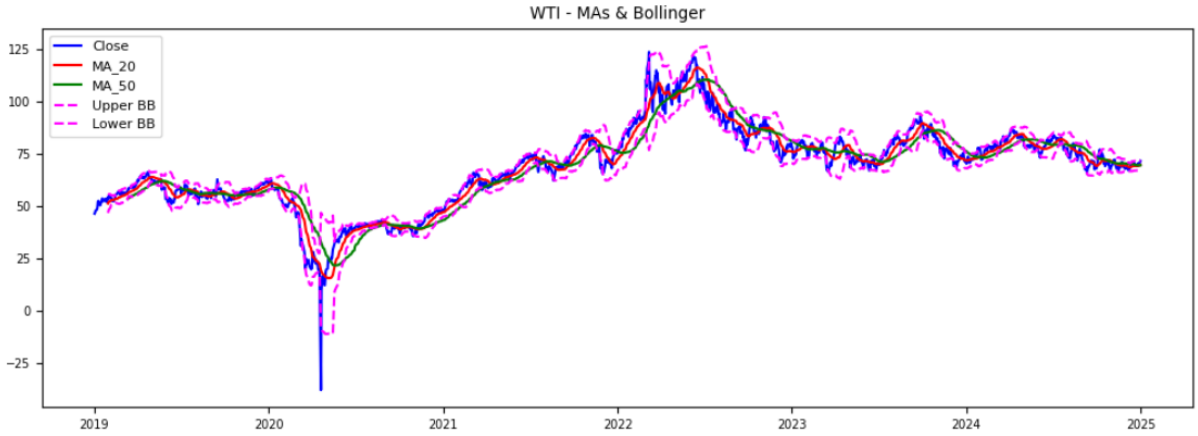



SHELL - MAs & Bollinger
Close
MA_20
MA_50
Upper BB
Lower BB
70
60
50
40
30
20
2019
2020
2021
2022
2023
2024
2025

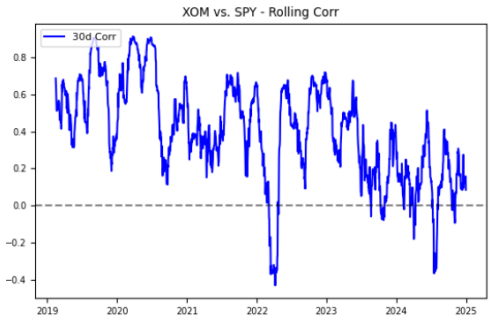

BP - MAs & Bollinger
Close
MA_20
MA_50
Upper BB
Lower BB
40
35
30
25
20
15
10
2019
2020
2021
2022
2023
2024
2025

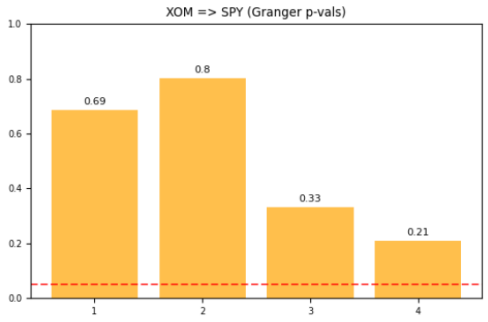

SPY - MAs & Bollinger
Close
MA_20
MA_50
Upper BB
Lower BB
600
500
400
300
200
2019
2020
2021
2022
2023
2024
2025

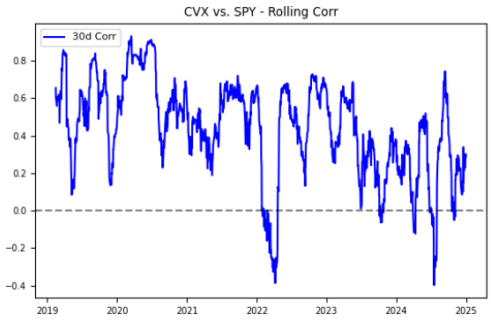

BRENT - MAs & Bollinger
Close
MA_20
MA_50
Upper BB
Lower BB
120
100
80
60
40
20
2019
2020
2021
2022
2023
2024
2025

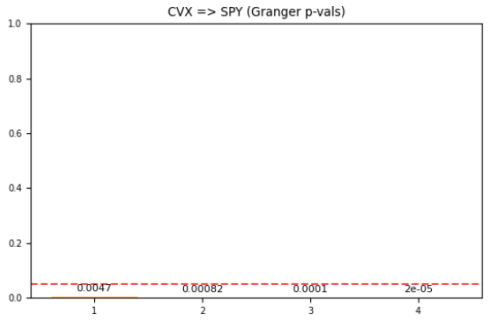


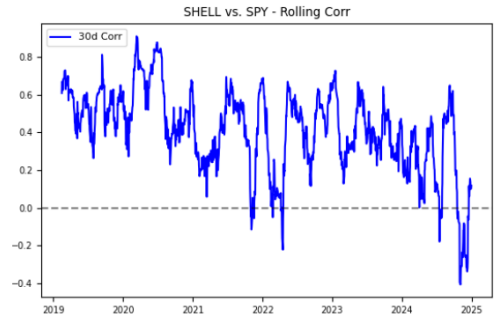
Granger Causality and Rolling Correlation

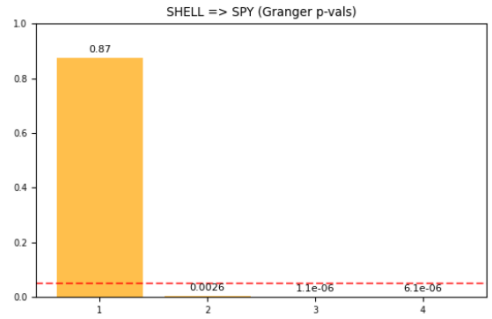


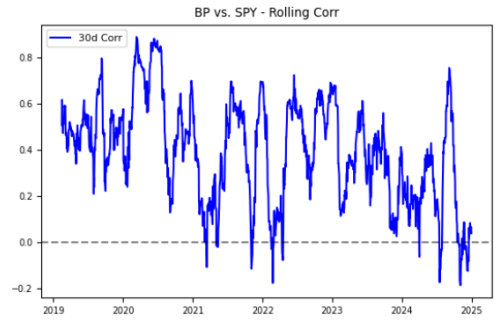


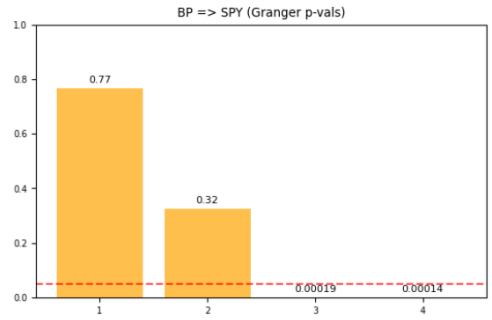


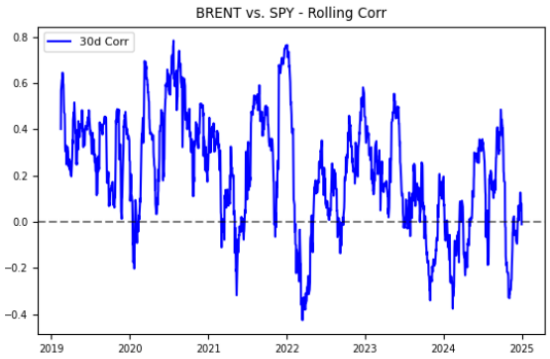


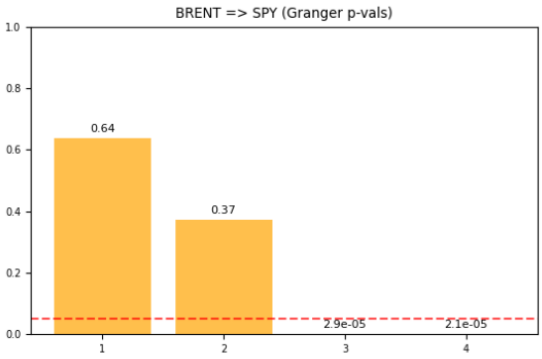


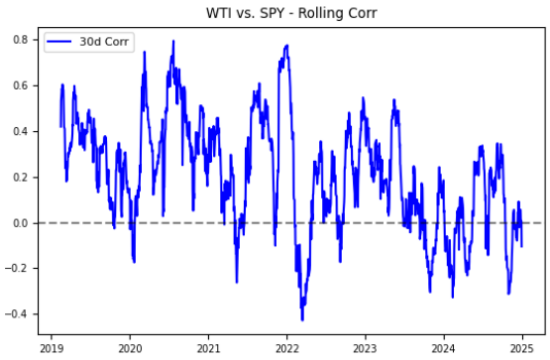


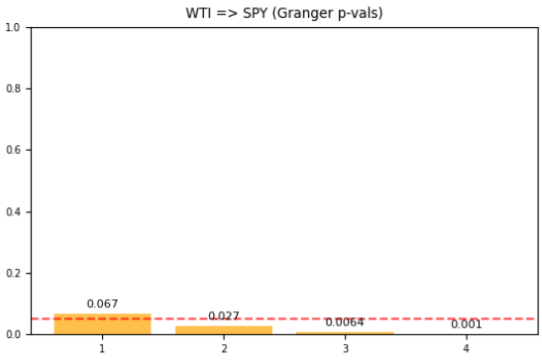





BRENT vs. SPY - Rolling Corr
30d Corr
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
2019
2020
2021
2022
2023
2024
2025

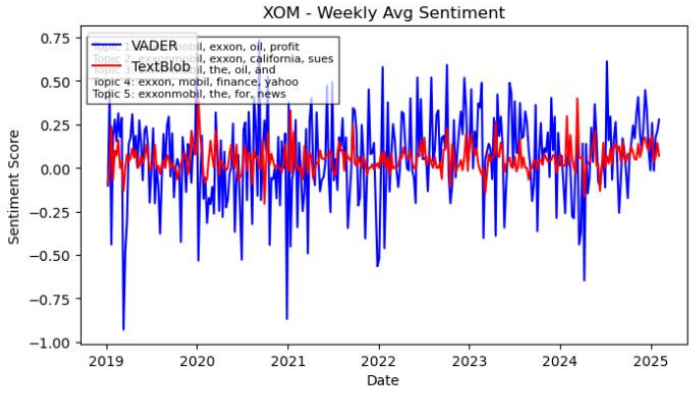

BRENT => SPY (Granger p-vals)
1.0
0.8
0.6
0.4
0.2
0.0
0.64
0.37
2.9e-05
2.1e-05
1
2
3
4

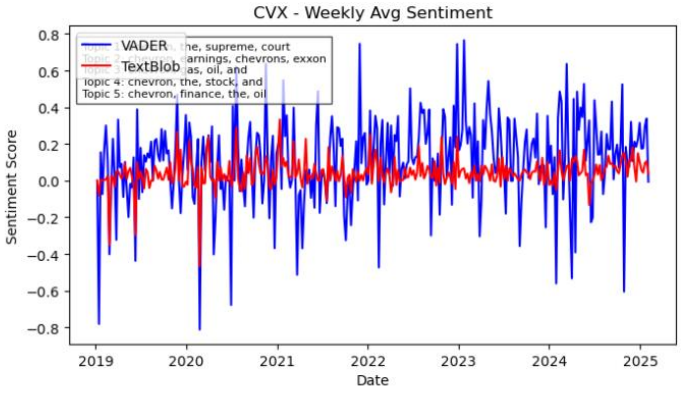

WTI vs. SPY - Rolling Corr
30d Corr
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
2019
2020
2021
2022
2023
2024
2025

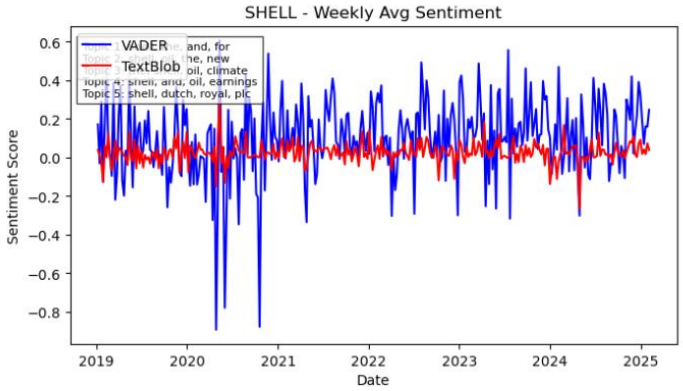

WTI => SPY (Granger p-vals)
1.0
0.8
0.6
0.4
0.2
0.0
0.067
0.027
0.0064
0.001
1
2
3
4

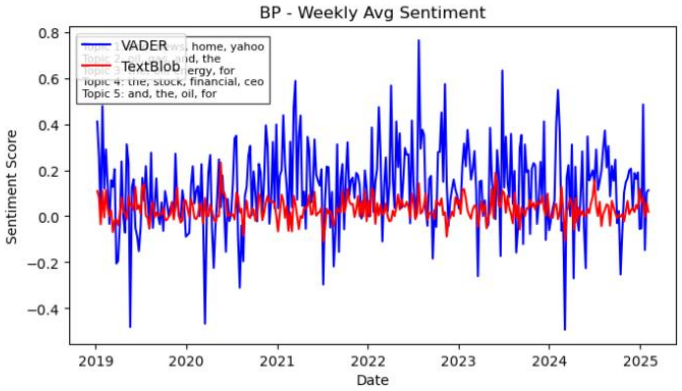
Appendix D

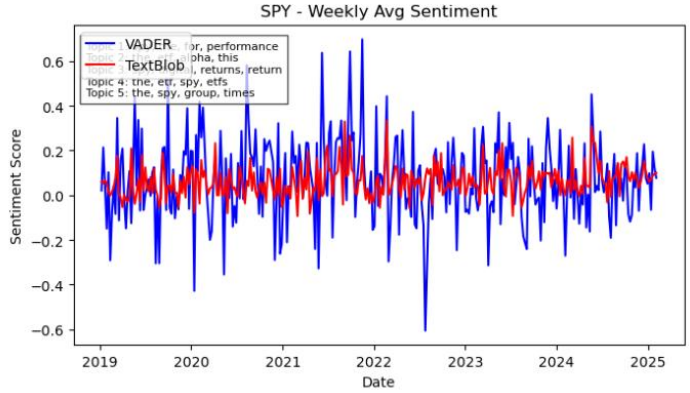
Topic Modeling

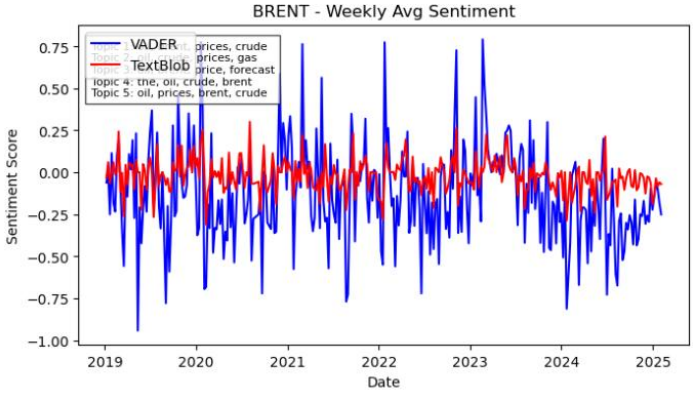


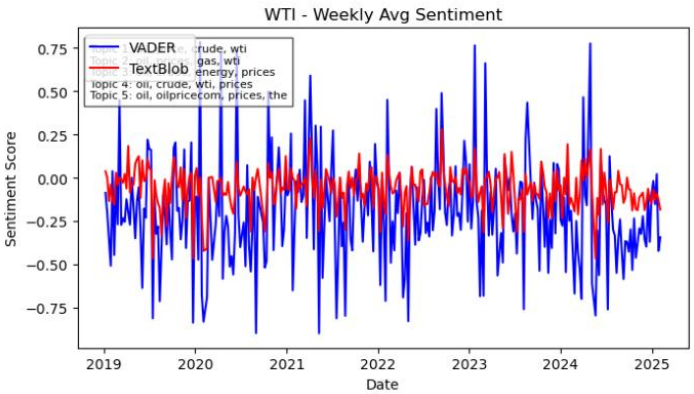




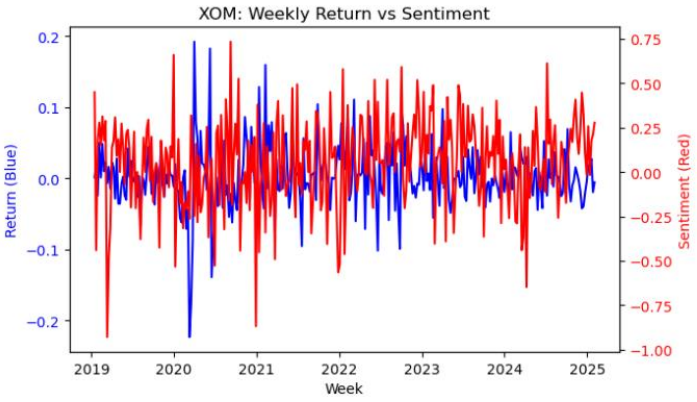


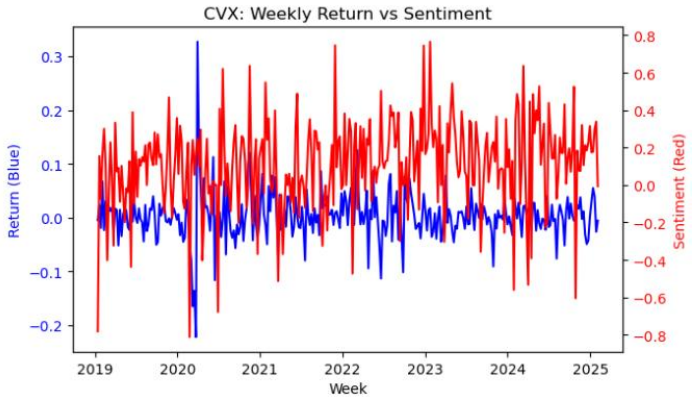


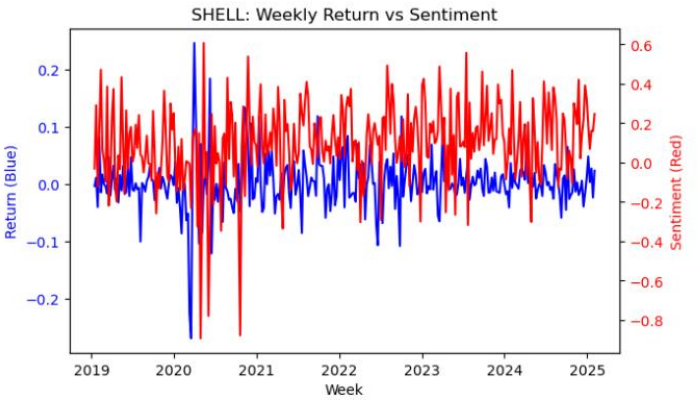


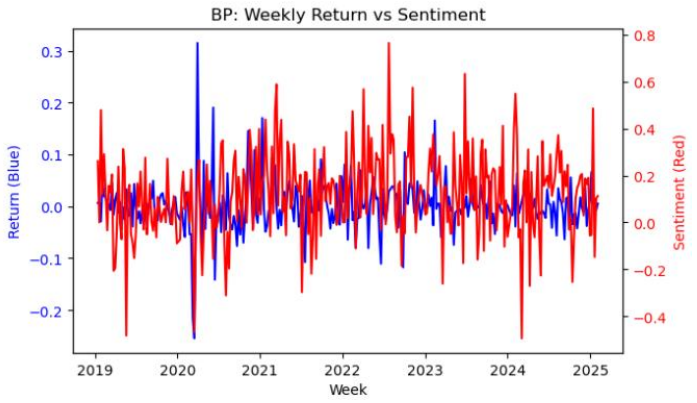


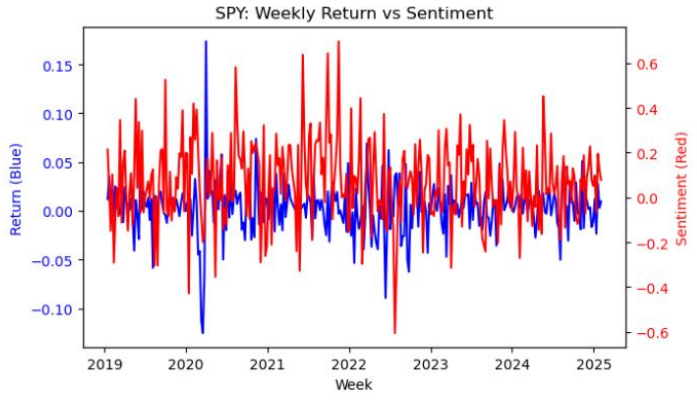
Sentiment vs Weekly Returns

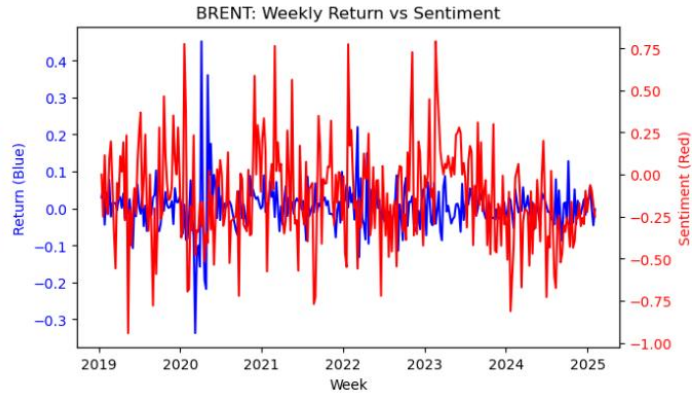


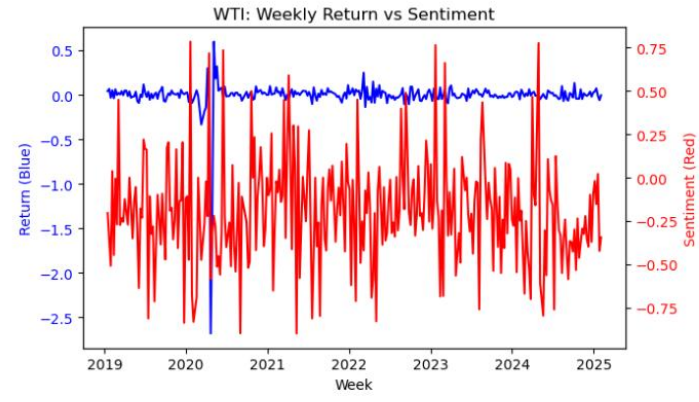




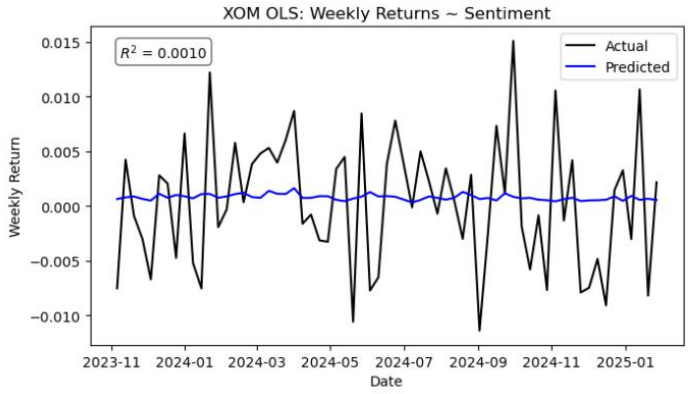


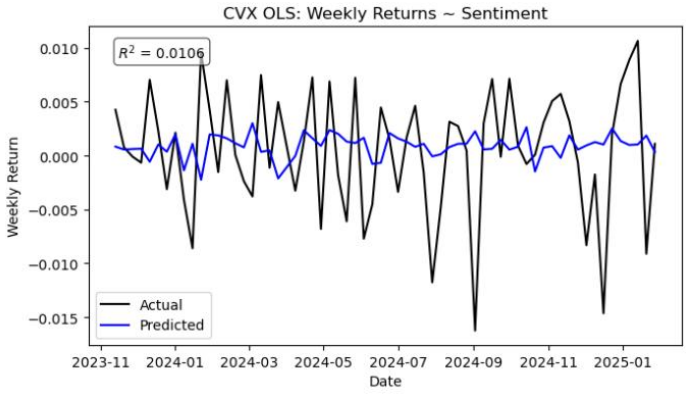


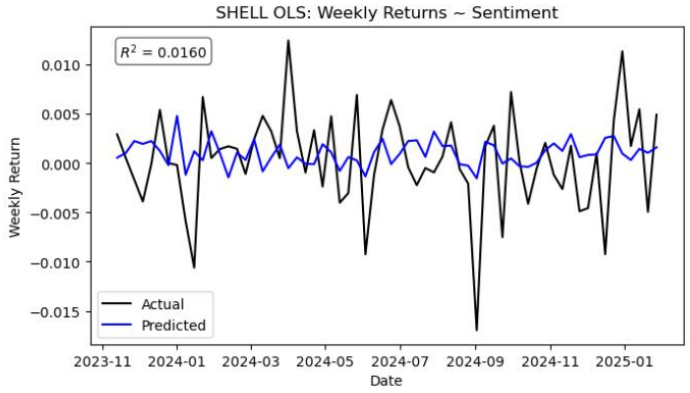


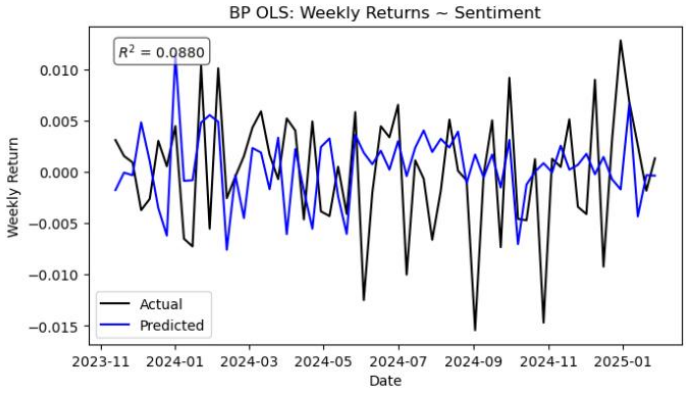


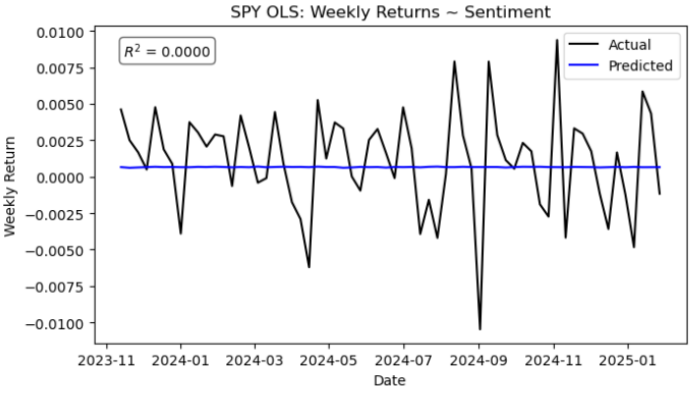
OLS Regression of Sentiment vs Returns

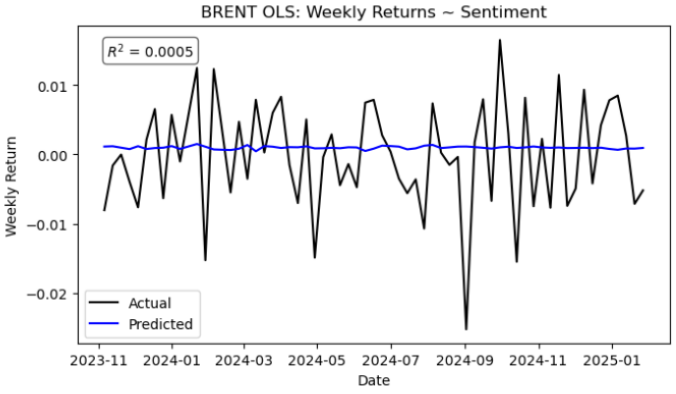


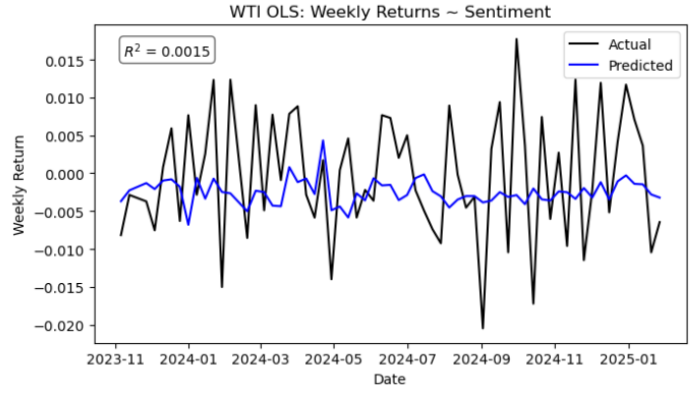




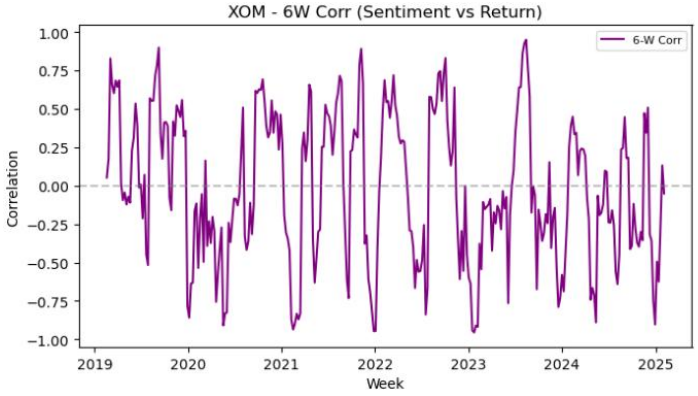


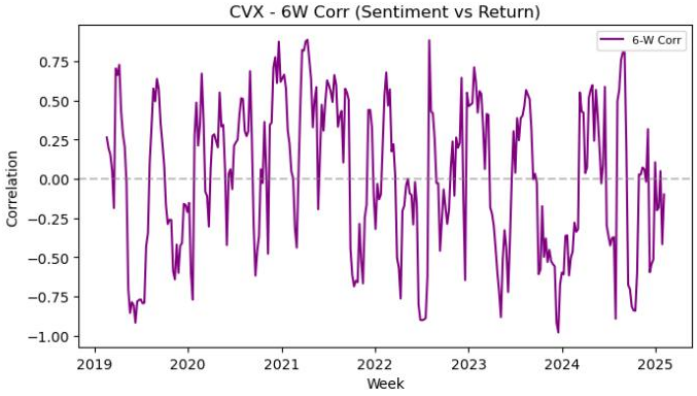


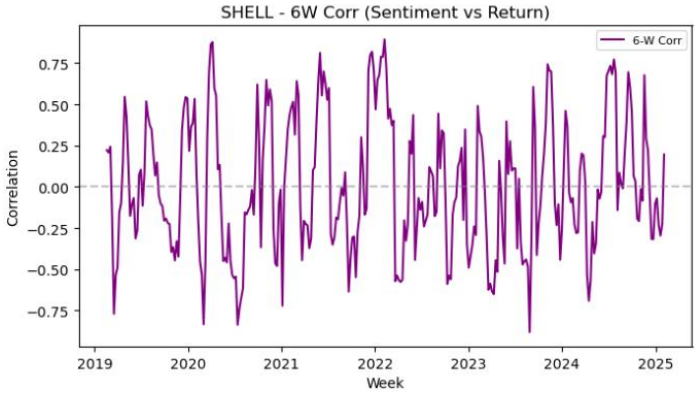


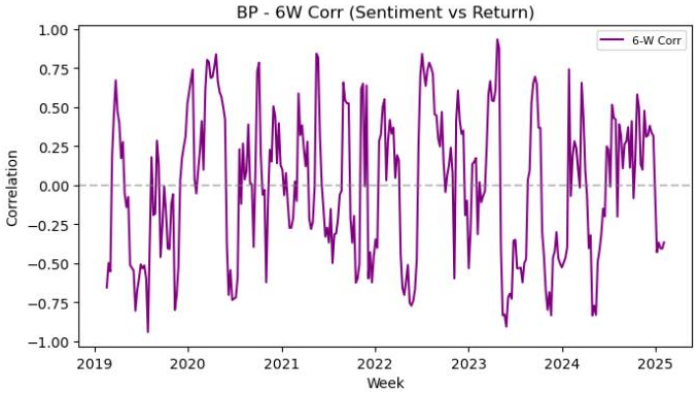


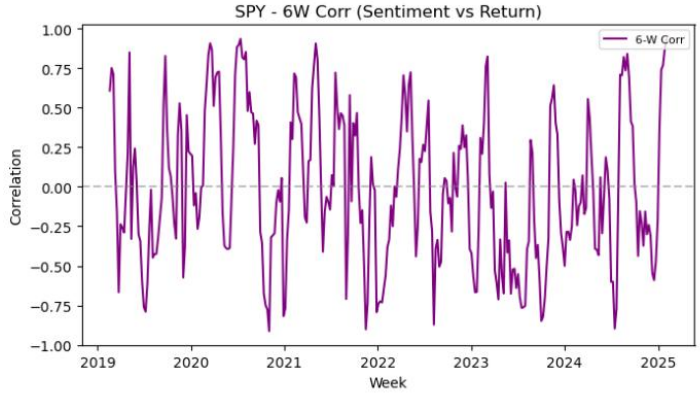
Rolling Correlation: Sentiment vs Returns

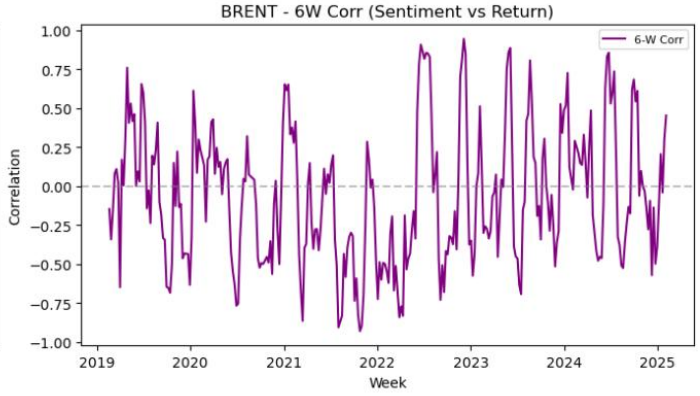


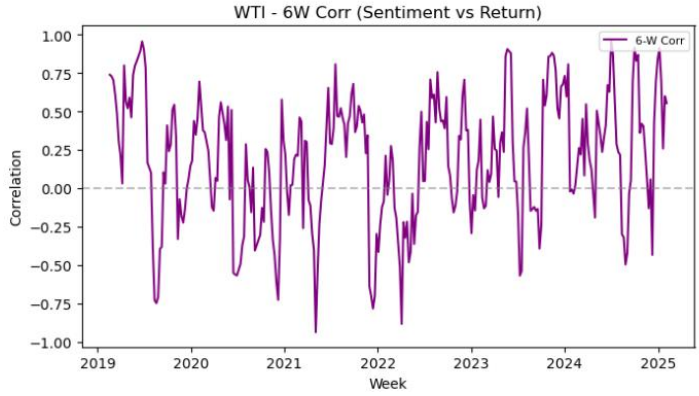




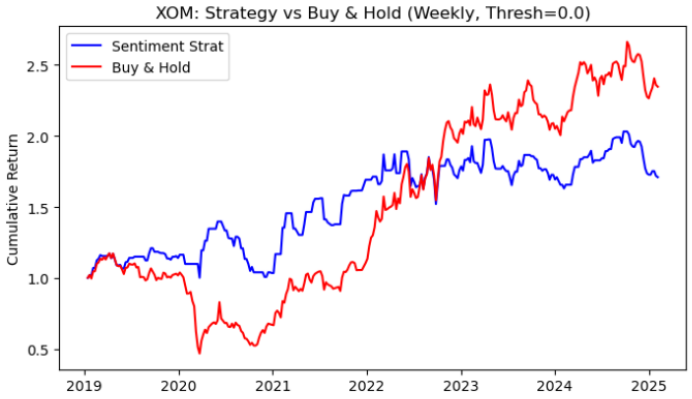


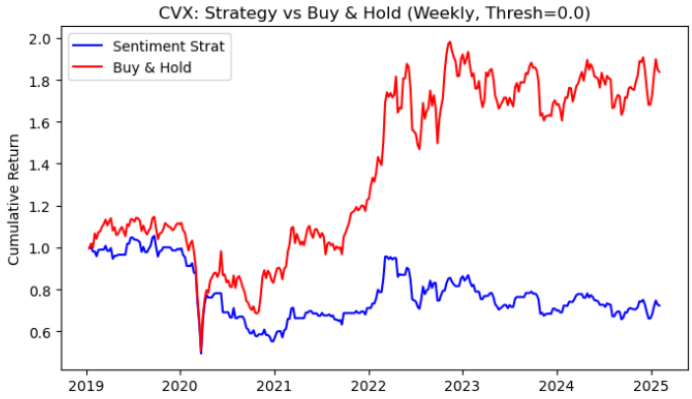


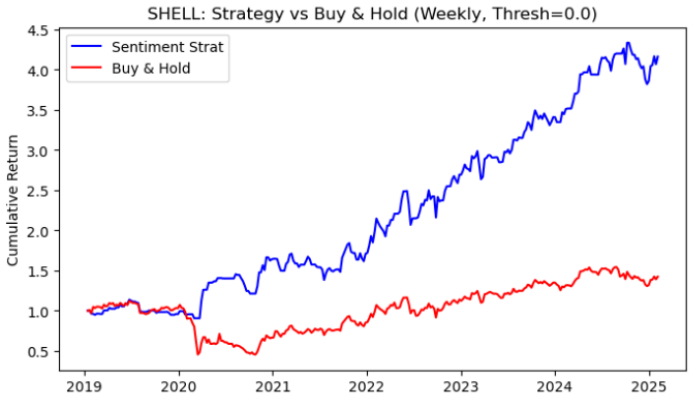


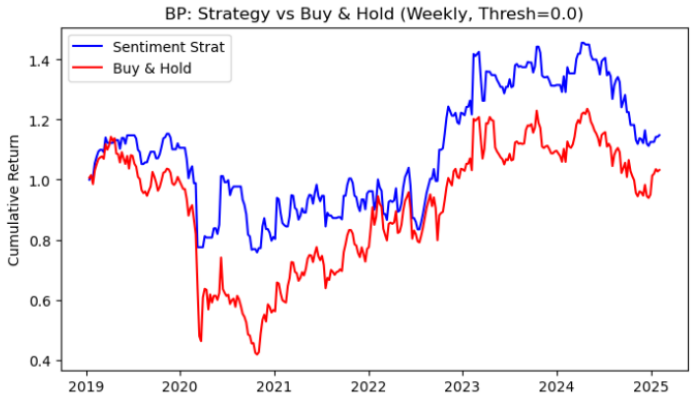


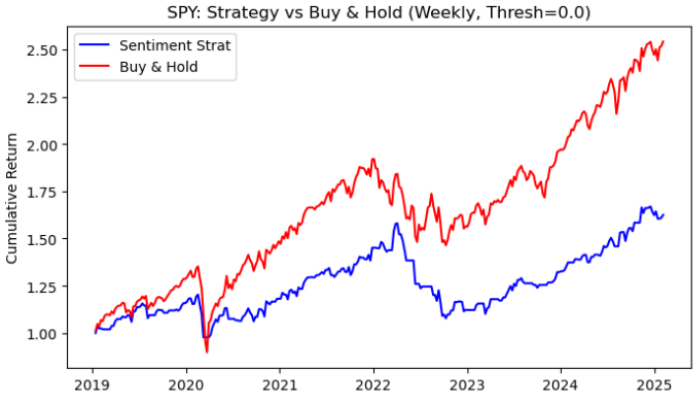
Sentiment-Based Strategy vs Buy & Hold

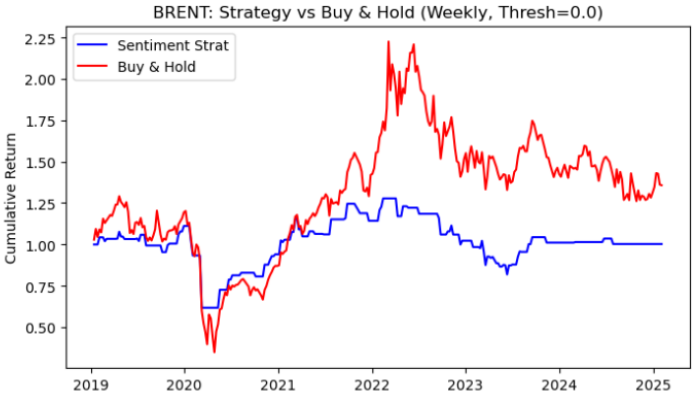


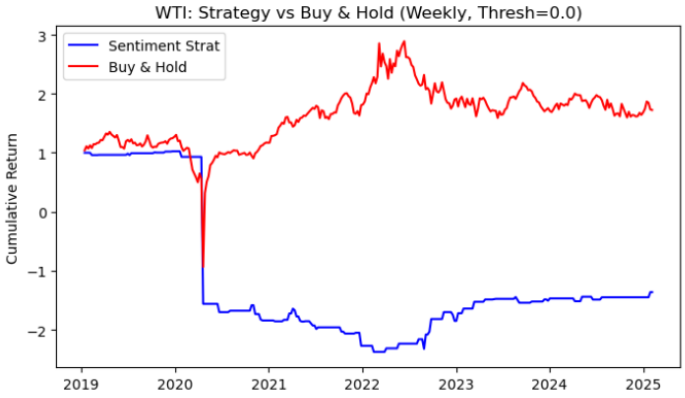












Weekly Sentiment Predictive Power & Polarity Summary