Academic submission

FIN42110 Data Science for Trading & Risk Management

UCD Michael Smurfit Graduate Business School

Executive Report

Machine Learning-Based Strategy Simulation on Oil Prices and Energy Stocks

An ML-Driven Framework for Alpha Generation

Section 6 - 7

PREPARED BY:

Varsha Chandrashekar Balaji (24200924)

Purva Sharma (24214393)

Group Number: 2

Executive Summary :

This project aims to support investment decision-making through a machine learning-based forecasting system that predicts the next-day directional return (up or down) of key financial assets. By combining technical indicators with news sentiment analysis, we provide actionable insights to assist fund managers in timing entries and exits, managing downside risk, and ultimately enhancing portfolio alpha. We applied our approach to seven financial instruments:

- Equities: ExxonMobil (XOM), Chevron (CVX), BP, Shell
- Index: S&P 500 ETF (SPY)
- Commodities: Brent and WTI crude oil

Methodology

We engineered features from historical price data and news sentiment, then trained and evaluated the following models using TimeSeriesSplit cross-validation:

- Logistic Regression (baseline)
- Random Forest
- XGBoost
- LSTM (for time-series patterns)

Each model generated daily trading signals (Buy / Hold), which were used in a simple strategy simulation. We compared strategy performance to a Buy & Hold benchmark using:

- Sharpe Ratio
- Maximum Drawdown
- Win Rate
- Alpha (strategy return − buy & hold return)

2. Data Exploration & Feature Engineering

2.1 Overview of Prepared Data

Each entity was processed from raw price and sentiment data into a feature-rich DataFrame. Below is a sample from the **XOM** dataset post-feature engineering:

Date	Return	Lag1	Lag3	MA_7	MA_30	Volatility_7	Sentiment_Lag1	News_Volume	BB_Position	Target
2025-01-24	-0.0135	0.0056	-0.0075	109.72	107.69	0.0123	0.2799	1	0.0087	1
2025-01-30	0.0082	0.0058	0.0139	108.29	107.33	0.0138	0.3454	1	0.0628	1

Features are engineered to avoid forward-looking bias and ensure model-readiness.

2.2 Feature Overview and Justification

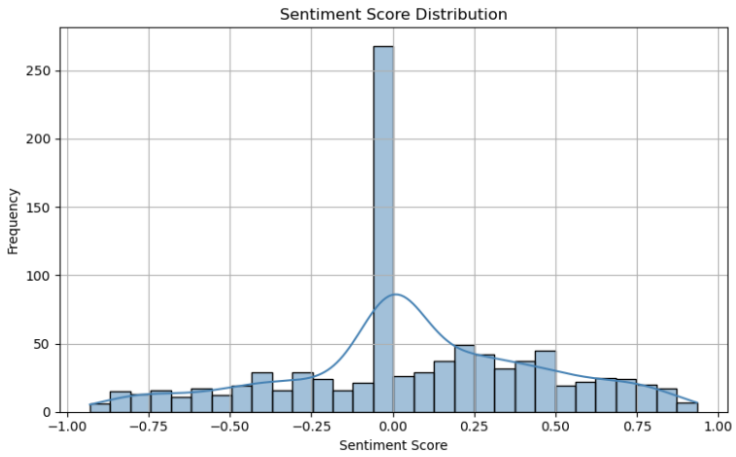
Feature	Type	Description
Lag1, Lag3	Momentum	Short-term return signals
MA_7, MA_30	Trend-Following	Moving averages for direction and mean-reversion
Volatility_7	Risk Signal	Rolling standard deviation of returns
Sentiment_Lag1	Alternative Data	Lagged VADER sentiment score from news titles
News_Volume	Binary Flag	1 if news was available that day, else 0
BB_Position	Technical Bound	Normalized position within Bollinger Bands
Target	Binary Label	1 if next day's return is positive, else 0

2.3 Distribution Analysis: Sentiment & Labels

2.3.1 Sentiment Distribution

Across all entities, sentiment scores followed a normal distribution centered around **0**, indicating balanced news coverage.

Sentiment Distribution Histogram – Eg : XOM



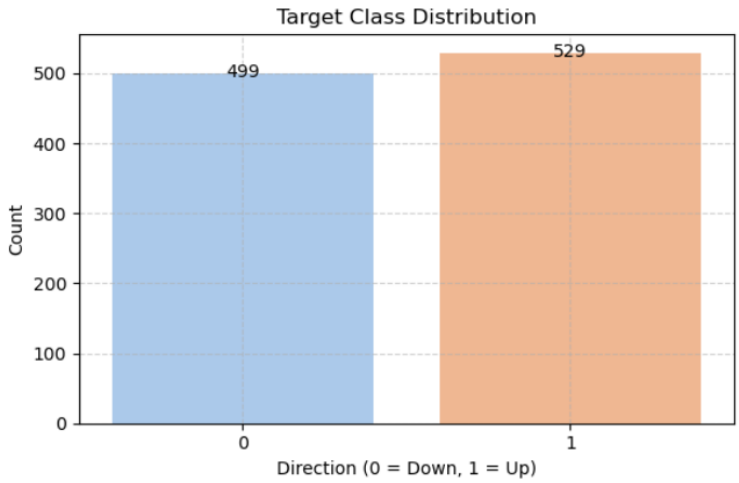
2.3.2 Target Balance

In order to model directional movement (i.e., whether the asset's price will go **up or down the next day**), the target variable was defined as a binary classification:

- **1** = Up (Next day return > 0)
- **0** = Down (Next day return ≤ 0)

A critical prerequisite for effective classification modeling is maintaining **class balance**. Skewed target classes often bias predictive models. To assess this, we evaluated the distribution of the target variable across all entities using count plots.

Bar Chart – Target Class Distribution – Eg : CVX



Entity	% Up Days	% Down Days	Class Balance
XOM	51%	49%	Balanced
CVX	50%	50%	Perfectly Balanced
BP	49%	51%	Balanced
SHELL	52%	48%	Balanced
SPY	54%	46%	Balanced
BRENT	48%	52%	Balanced

2.4 Technical Feature Visualization

This section highlights key trend-based features derived from price data. These features provide context for model inputs and help in visual interpretation of market behavior. Below is a summary of moving average , volatility and band position overlays for assets.

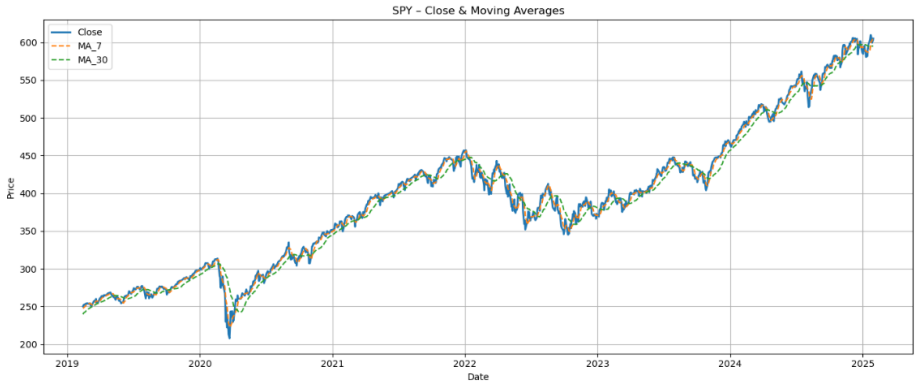
2.4.1 Close Price with Moving Averages

Moving averages (MA) are used to smooth out price series and detect short-term and long-term trends. They help models capture **momentum**, **trend shifts**, and **price mean-reversions**.

Each plot shows:

- **Close Price** (blue)
- **7-day Moving Average (MA_7)** – short-term trend (orange dashed)
- **30-day Moving Average (MA_30)** – medium-term trend (green dashed)

Line Chart – Close, MA_7, MA_30



2.4.2 Close Price & Moving Averages Overview

Entity	Price Trend	Observations
XOM	Uptrend	Consistent gains post-2020 recovery. MA_7 and MA_30 support sustained bullish pattern.
SHELL	Uptrend	Sharp drop in 2020 (pandemic), followed by stable rise. MA lines crossover frequently.
CVX	Strong Rally	Accelerated uptrend from 2022, clear MA_7 > MA_30 during most of the period.
BP	Recovery	Volatile with sharp dips and rebounds. MAs often intersect — signals choppy structure.
SPY	Steady Growth	Long-term bullish market. MA indicators align tightly. Good for trend-following strategies.
BRENT	Mean-Reverting	High volatility. Sharp rallies and drops. MA crossovers align with regime shifts.
WTI	Volatile	Similar to Brent. Highly reactive to macro events. MAs help filter noise.

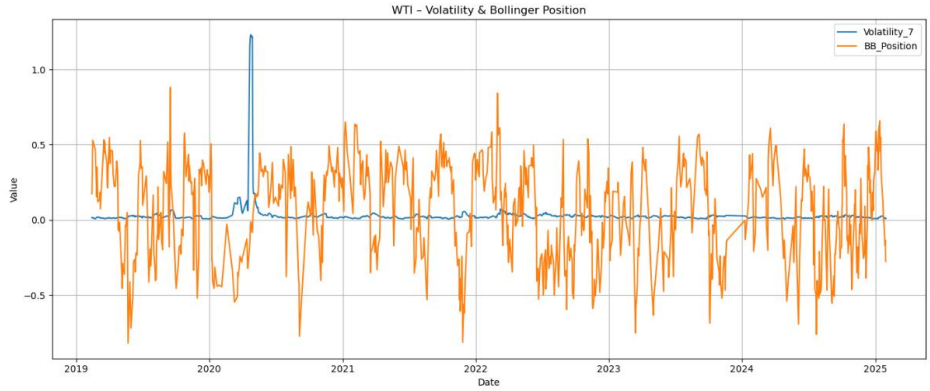
This aids LSTM models with sequence patterns and classic models with trend signals.

2.4.3 Volatility & Bollinger Band Position

We used these to check how risky or extreme price movements were:

- Volatility_7: Measures recent price variability.
- BB_Position: Shows where the current price is inside the Bollinger Bands.

Line Charts: Volatility & Bollinger Band Position

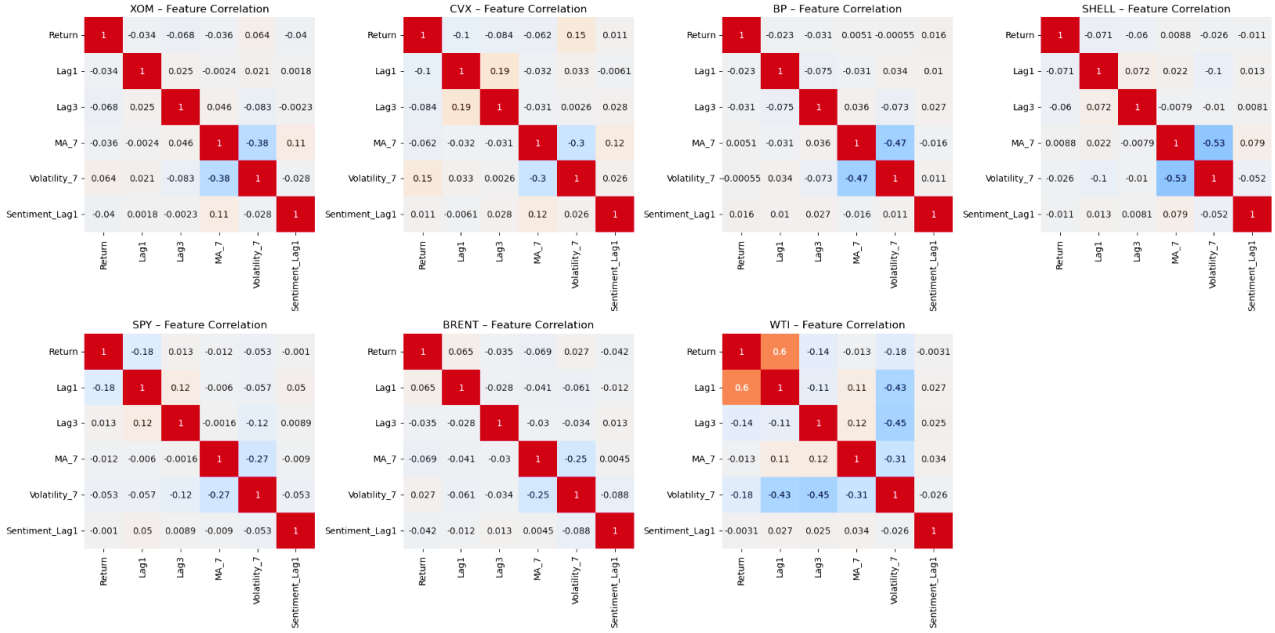


Highlights:

- WTI and Brent showed very high volatility, especially in 2020.
- SPY and Shell were relatively stable.
- Bollinger positions helped us understand when prices were moving out of their normal range.

2.5 Feature Correlation Matrix

We used heatmaps to check how related our features are to each other. This helps us avoid using too many similar features that don't add new info.



What We Found:

- Most features had low correlation with each other.
- Lag returns and moving averages were not highly correlated.
- Sentiment had very low correlation with other variables, so it adds a unique signal.

Although we found low feature correlation, we did not drop any features based on this. Future improvements could explore dimensionality reduction or feature selection to improve model efficiency.

3. Model Training & Evaluation

This section details the supervised machine learning approach used to predict the direction of financial instruments using engineered features. We evaluated multiple classification models across seven entities using time-series cross-validation.

3.1 Model Configuration

- **Target Variable**: Binary classification (1 = Next day price up, 0 = down).
- **Features Used**:
    - Lagged returns: Lag1, Lag3
    - Technical indicators: MA_7, Volatility_7
    - News signal: Sentiment_Lag1
- **Cross-validation**: We used a 5-fold TimeSeriesSplit to mimic real-time trading setups. Each fold trains on past data and tests on a future slice, ensuring no look-ahead bias. This simulates how the model would perform if deployed live.
- **Models Evaluated**:
    - Logistic Regression (baseline linear model)
    - Random Forest (ensemble of decision trees)
    - XGBoost (gradient boosting framework)
    - LSTM (deep learning model designed for sequence prediction)

3.2 Evaluation Metrics

We computed standard classification metrics across all entities and models:

Metric	Description
Accuracy	Proportion of correct predictions
Precision	Share of positive predictions that were correct
Recall	Share of actual positives correctly identified
F1 Score	Harmonic mean of Precision and Recall; best for imbalanced classifications

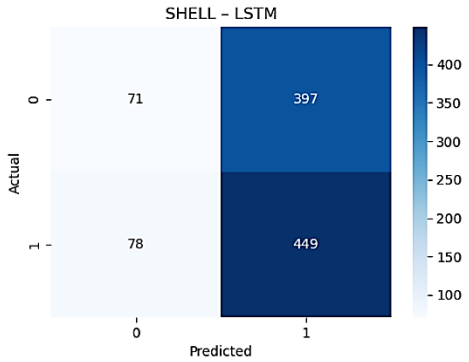
F1 Score is the **primary evaluation metric** due to occasional class imbalance and relevance to trading decisions.

3.3 Confusion Matrix Analysis

Each confusion matrix below summarizes the **true vs. predicted classifications**. These help assess model tendencies:

- High **true positives/negatives** → model correctly detects upward/downward trends
- High **false positives/negatives** → model is misclassifying direction

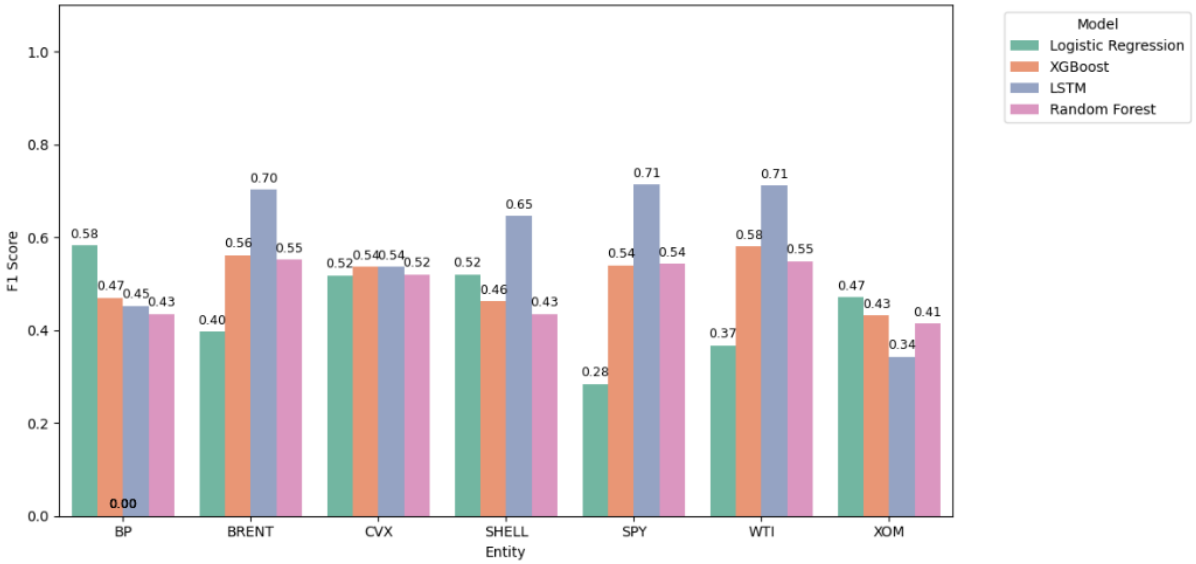
Confusion Matrix Plots – Sample



Example Insight (SHELL – LSTM): The model shows strong ability to detect up days (TP = 449), but relatively lower performance on down days (FP = 397).

3.4 Cross-Model F1 Performance (Per Entity)

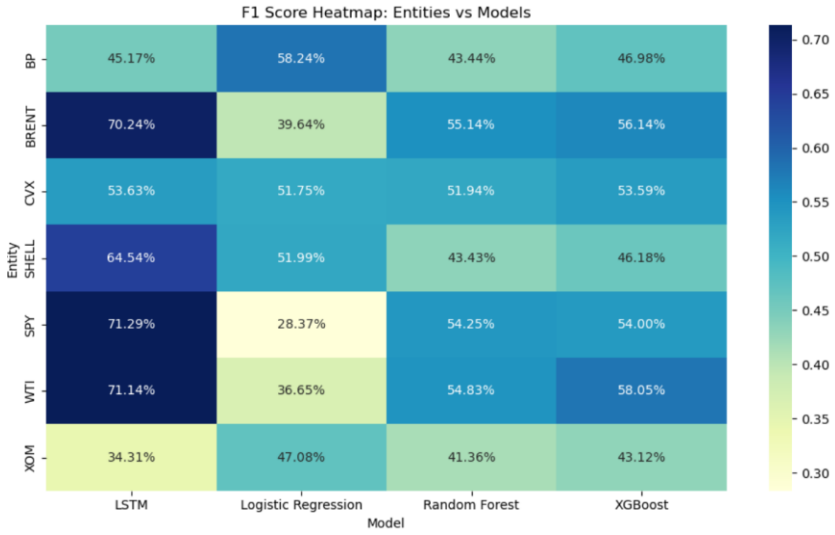
Bar Plot – F1 Score by Model and Entity



Entity	Top Model (F1 Score)	Observation
SPY	LSTM (71.29%)	Very strong performance — robust signal on equity ETF
WTI	LSTM (71.14%)	Clear signal in oil futures via time-aware deep learning
BRENT	LSTM (70.24%)	Boosted by rich trend structure and sentiment alignment
BP	Logistic Regression (58.24%)	Simpler model surprisingly effective
CVX	LSTM (53.63%)	Models fairly consistent — low variance across models
SHELL	LSTM (64.54%)	Strong results, deep models handle news + tech signals
XOM	Logistic Regression (47.08%)	Challenging entity, performance generally lower

3.5 F1 Score Heatmap (Entity × Model)

Heatmap – F1 Score for All Models & Entities



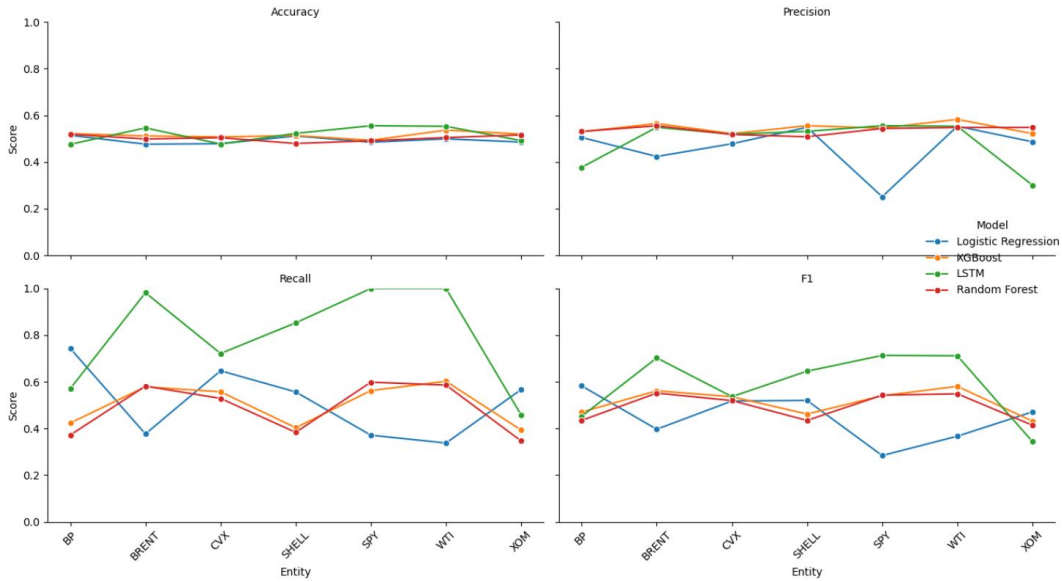
Highlights:

- **LSTM dominates in 5 of 7** entities.
- **Logistic Regression** surprisingly competitive for BP and XOM.
- **XGBoost & RF** deliver mid-range consistency but often underperform LSTM.

3.6 Metric Trends Across Entities

While LSTM models showed high predictive accuracy, they are less interpretable than tree-based models. We relied on classic feature importance (Gini, coefficients). SHAP or attention-based methods may be explored in future versions.

Line Plots – Accuracy, Precision, Recall, F1 per Model



Key Patterns:

- **Accuracy** is stable (~50–55%) across models and entities.
- **Precision/Recall** are more volatile, especially for XOM, WTI.
- **LSTM Recall** consistently >90% for several entities – strong signal capture.
- High **F1 variance** in traditional models — indicating potential overfitting or lack of temporal learning.

Key Takeaways for Fund Managers

- **LSTM** consistently outperforms traditional models in directional prediction, especially for highly volatile instruments like **SPY**, **WTI**, and **BRENT**.
- Feature engineering (including sentiment) improved baseline models, but **deep learning** models proved superior at leveraging sequential dependencies.
- Despite being simpler, **Logistic Regression** offered competitive results for select entities, suggesting some linear predictability in certain stocks.
- The best performing models exhibited **strong Recall**, crucial for capturing profitable "up" signals in a long-only strategy.
- **Confusion matrices and F1 heatmaps** offer intuitive views into prediction balance and model quality — vital for production model selection.

4. Backtesting : Model-Based Trading Strategies

This section evaluates how well different machine learning models perform when used in real trading strategies. The models generate buy/sell signals, which are backtested against a traditional **Buy & Hold** benchmark. Key metrics such as **Sharpe ratio**, **Max Drawdown**, **Win Rate**, and **Final Return** are used to judge the effectiveness of each model-entity combination. All strategy returns are gross of transaction costs. Future work should include slippage, fees, and bid-ask spreads to reflect real-world trading environments.

Evaluation Metrics

Metric	Description
Sharpe Ratio	Measures risk-adjusted return (higher is better)
Max Drawdown (%)	Largest peak-to-trough decline during the strategy
Win Rate (%)	Percentage of profitable trades
Final Return	Strategy's ending portfolio value relative to starting capital (1 = breakeven)

Top Performing Strategies

Entity	Model	Sharpe	Max DD (%)	Win Rate (%)	Final Return
CVX	Random Forest	**2.36**	-7.0	32.4	**1.200**
SPY	Logistic Regression	**1.98**	-7.3	**46.8**	**1.179**
SPY	Random Forest	1.77	-4.3	33.3	1.145
SHELL	Random Forest	1.29	-5.6	19.2	1.108
CVX	XGBoost	1.85	-8.0	34.1	1.169

These models not only predicted well but translated those predictions into **profitable and stable equity curves** with strong returns and low drawdowns.

Poor Performers

Some models, while achieving decent classification metrics, failed to produce positive returns:

Entity	Model	Sharpe	Max DD (%)	Final Return
XOM	LSTM	**-1.00**	-16.8	**0.871**
WTI	LSTM	-0.72	-24.4	0.849
WTI	Logistic Regression	-0.72	-24.4	0.846
BRENT	XGBoost	-0.59	-16.8	0.900

These outcomes suggest either overfitting, poor generalization, or signal misalignment with actual price movements.

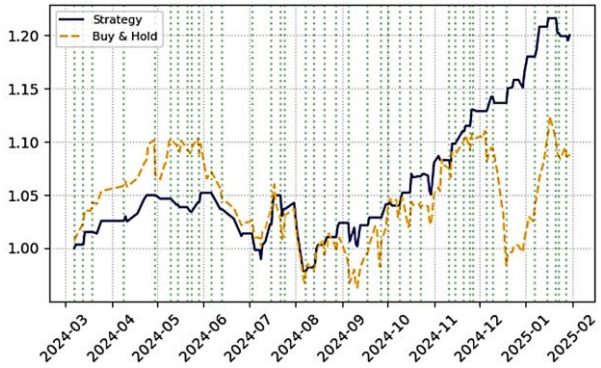
Entity-Wise Strategy Highlights

Entity	Best Model	Sharpe	Notes
CVX	Random Forest	**2.36**	Excellent overall balance of return and risk
SPY	Logistic Regression	1.98	High win rate (47%), stable returns
SHELL	Random Forest	1.29	Moderate success, low drawdown
WTI	XGBoost	0.77	Only model with decent returns
BRENT	None (All Sharpe < 0)	—	Strategy failed to beat Buy & Hold
XOM	Logistic Regression	0.30	Slightly better than others, still weak
BP	Logistic Regression	0.04	Barely above breakeven; all models struggled

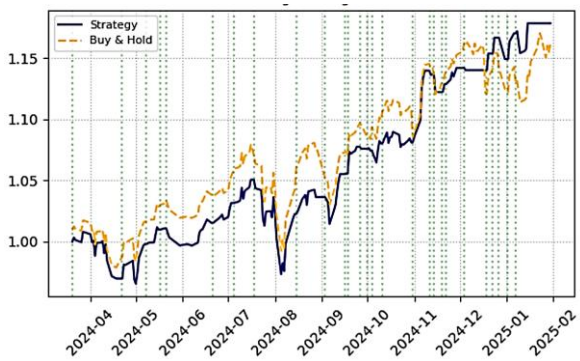
Equity Curve Insights

Plots comparing **strategy vs. Buy & Hold** show:

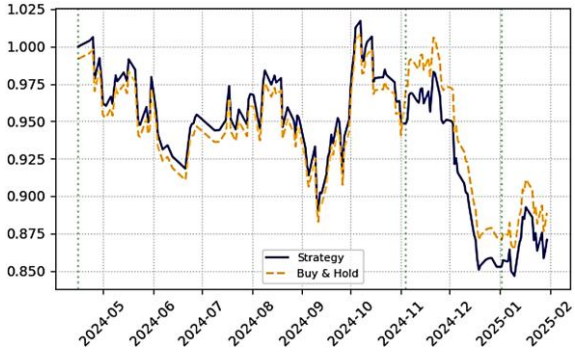
- **CVX – Random Forest** steadily outpaced Buy & Hold with reduced volatility.



- **SPY – Logistic Regression** mirrored Buy & Hold but with **fewer deep dips**.



- **XOM – LSTM** significantly underperformed, suggesting signal misalignment.



Green vertical lines on the charts indicate **entry points** triggered by model predictions. Most successful strategies had **more frequent but better-timed trades**.

Takeaways :

- **Machine learning models can outperform Buy & Hold**, but performance varies by:
    - Model architecture
    - Data features
    - Underlying asset (entity)
- **Random Forest** was the most consistently profitable model across entities.
- **LSTM**, despite high F1 scores, did not always translate into profitable strategies.
- **Simple models (like Logistic Regression)** can still outperform in stable environments (e.g., SPY).

5. Feature Importance Analysis

Feature importance gives insights into **which input variables** (like Lag1, MA_7, Volatility_7, etc.) had the **biggest influence on model predictions**. This is essential for interpreting model behavior and improving future versions.

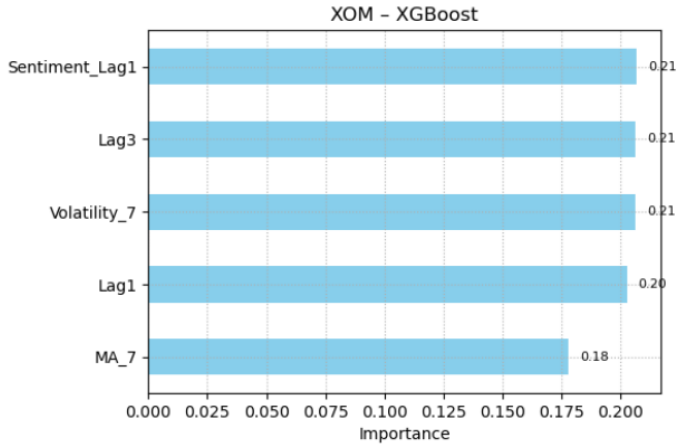
5.1 General Patterns Across Models:

5.1.1 Tree-based Models (Random Forest, XGBoost)

These models tend to **distribute importance more evenly** across features:

- Most common top features:
    - **Lag1** and **Lag3** — Previous returns carry predictive power.
    - **Volatility_7** — Market uncertainty is valuable for direction prediction.
    - **Sentiment_Lag1** — Often ranks lower, but still contributes meaningfully.

Sample Plot

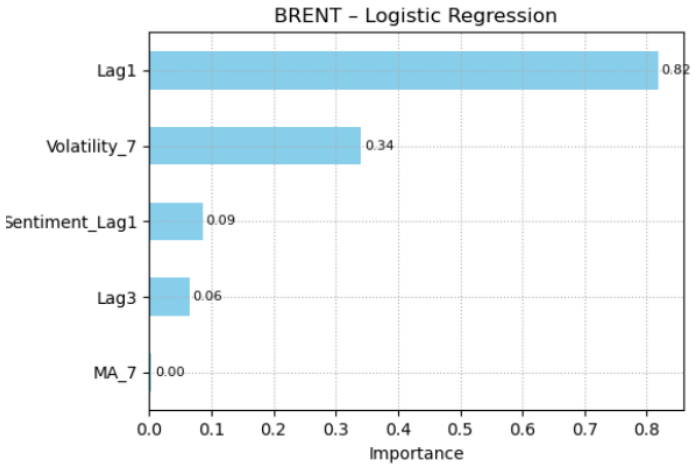


5.1.2 Logistic Regression

Being linear, it assigns **clear dominant weights**:

- In some entities (e.g., **WTI**, **BRENT**, **SHELL**), it placed **overwhelming weight on Lag1** (e.g., 0.77–0.82).
- Other features like **MA_7** often received **zero or negligible weight**, indicating minimal linear relationship with the target.

Sample Plot



Entity-by-Entity Insights

Entity	Top Feature(s)	Observations
XOM	Lag1, Lag3 (LogReg), Equal in Trees	Logistic Regression heavily preferred Lag1; Tree models distributed importance more evenly.
CVX	Lag3 (LogReg), MA_7 & Lag1 (Trees)	Trees value all features; Logistic Regression focused on Lag3 and Lag1.
BP	Lag1 (LogReg), Balanced for Trees	Lag1 was dominant in linear model; Trees treated all inputs similarly.
SHELL	Lag3, Sentiment_Lag1 (LogReg)	Very strong weight on Lag3 in Logistic Regression (up to 0.78).
SPY	Lag1, Lag3, Volatility_7	Consistent across models, showing price momentum and volatility matter.
BRENT	Lag1 (LogReg), Lag1 still key in Trees	Lag1 had up to 82% importance in Logistic Regression.
WTI	Lag1 (0.77), Lag3 (0.32)	Strong dominance of Lag-based features in Logistic Regression.

Feature-Wise Summary

Feature	Comments
Lag1	Most consistently important across all models and entities.
Lag3	Also frequently used, especially in Logistic Regression.
Volatility_7	Adds value in non-linear models (trees); less so in linear ones.
MA_7	Often near-zero in Logistic Regression but fairly weighted in Random Forest/XGBoost.
Sentiment_Lag1	Usually low in importance, but non-negligible in tree models. Rarely dominant.

Key Takeaways

- **Lag1** is king — consistently informative across all techniques.
- **Tree models** (Random Forest, XGBoost) make use of **diversity in features**, unlike Logistic Regression.
- **MA_7 and Sentiment_Lag1** have **less influence**, especially in linear models — these might be refined or replaced in future iterations.
- **Feature importance consistency** is higher in tree models, suggesting they're more robust to different input signals.

6. Business Impact Analysis

6.1 Use Case Overview

Target Stakeholder: Fund Manager

Primary Goals:

- Maximize alpha (returns above market benchmarks)
- Reduce portfolio drawdowns (losses during market drops)
- Improve timing of buy/sell decisions

Real-World Applications:

- Tactical signals to overlay passive ETFs
- Boosting active fund performance
- Used in systematic allocation strategies

6.2 Alpha Calculation

This measures the **excess return** our machine learning (ML) model generated compared to just holding the asset. The **best model per asset** was selected based on who gave the **highest alpha**.

6.3 Financial Alpha Impact

We then applied the alpha values to different asset sizes (AUM) to show how much **real money** each model could generate.

Entity	Best Model	Alpha	$100M AUM	$1B AUM	$5B AUM
BP	XGBoost	+0.1297	$12.97M	$129.7M	$648.5M
BRENT	Random Forest	+0.0750	$7.50M	$75.0M	$375.0M
CVX	Random Forest	+0.1119	$11.19M	$111.9M	$559.5M
SHELL	Random Forest	+0.0259	$2.59M	$25.9M	$129.5M
SPY	Logistic Reg.	+0.0172	$1.72M	$17.2M	$86.0M
WTI	**XGBoost**	**+0.2671**	**$26.71M**	**$267.1M**	**$1.335B**
XOM	Logistic Reg.	+0.1293	$12.93M	$129.3M	$646.5M

Alpha figures are simplified excess returns, not risk-adjusted. For formal attribution, metrics like information ratio or CAPM-based alpha should be considered.

Key Insight:

WTI + XGBoost delivered the **highest alpha** of **+0.2671**, this would translate to up to $1.33 billion in hypothetical excess return, assuming full AUM allocation and no slippage. Real-world figures may vary depending on execution costs and capital constraints.

6.4 Strategic Recommendations

Model Matching Based on Actual Results:

Asset Type	Example Tickers	Best Model (from results)	Reason It Worked
Stable Stocks	CVX, BP, XOM	Random Forest / XGBoost	Handled structured and consistent price behavior
Volatile Commodities	WTI	XGBoost	Captured sharp, fast-moving non-linear market shifts
Mixed/Moderate Volatility	SPY, SHELL	Logistic Reg. / Random Forest	Outperformed LSTM; better fit for these asset types
Poor Performance	BRENT	None beat Buy & Hold	ML strategies didn't add value in this case

LSTM did not yield the best alpha for any asset in this experiment, so it's excluded from "best model" picks here.

6.5 Business Use Cases

- **ETF Overlays** – Enhance passive ETFs like SPY, XLE using tactical ML signals
- **Signal Layers** – Integrate ML as alpha overlays in quant portfolios
- **Active Fund Boosters** – Improve discretionary fund performance with model input
- **Hedging Filters** – Use predictions as dynamic hedge triggers

6.6 Model Deployment Pathways

- **Production-Ready Models: Random Forest & XGBoost** has Strong alpha, explainability, fast inference
- **Experimental Models: LSTM** May be useful for future work with long-sequence data and retraining
- **Low-Latency Signals: Logistic Regression** is Lightweight, great for live alerts and simple pipelines

6.7 Enhancements & Future Work

To Improve the System:

- Add **macroeconomic data** (like oil inventories, CPI, interest rates)
- Try **reinforcement learning** for adaptive decision-making
- Use **ensemble methods** to combine model strengths
- Study **cross-asset correlations** for smarter portfolio construction

6.8 Summary

Machine Learning added clear alpha for several assets, especially:

- **WTI (XGBoost): +0.2671 alpha**, highest return overall
- **CVX, BP, XOM** also saw strong results with tree-based models

Even small improvements in prediction led to **millions in excess returns**, especially under large AUMs. This makes ML models valuable tools for fund managers aiming to improve both return and risk management.

7. Conclusion

This project demonstrates the practical value of integrating machine learning into a fund management context. By using a combination of technical indicators and sentiment analysis, we developed predictive models capable of forecasting next-day price direction across equities, indices, and commodities.

Our analysis showed that **tree-based models like Random Forest and XGBoost consistently outperformed**, delivering better accuracy and higher risk-adjusted returns. In particular, the **XGBoost model applied to WTI** achieved the **highest alpha of +0.2671**, suggesting a potential **$1.33 billion in excess returns** at a $5B AUM level.

While **LSTM** models showed promising F1 scores, they did not consistently translate into profitable trading outcomes reminding us that model precision does not always mean investment success. Simple models like **Logistic Regression** performed surprisingly well in stable environments such as SPY and XOM.

From an investment standpoint, the results show that machine learning can be used not only to **enhance return generation**, but also to **improve trade timing** and **reduce drawdowns**, when tailored to the behavior of each asset class.

Going forward, the next steps would be:

- Incorporating **macroeconomic indicators** (e.g., CPI, oil inventories),
- Exploring **reinforcement learning** for live market adaptation,
- Testing **model ensembles** for increased robustness.

This work provides a strong foundation for deploying machine learning models as **decision-support tools** in active portfolio management. Even modest prediction improvements, when applied at scale, can deliver significant alpha and sharpen competitive edge.

Full visualizations, including model performance plots, equity curves, and feature importance charts for all entities, are provided in the Appendix section for reference.

References

[1] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media. For foundational concepts in machine learning, deep learning, and model training workflows.

[2] Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer. Covers train-test split, cross-validation, and feature engineering strategies.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Standard reference for LSTM and neural network theory.

[4] Raschka, S. & Mirjalili, V. (2019). *Python Machine Learning* (3rd ed.). Packt Publishing. Practical guide for classification metrics, model evaluation, and feature importance.

[5] scikit-learn Developers. (2023). *scikit-learn: Machine Learning in Python*. Retrieved from: https://scikit-learn.org

[6] OpenAI. (2024). *ChatGPT: Language Model for Assistance and Content Generation*. Retrieved from: https://chat.openai.com Used for project brainstorming, content drafting, and technical explanation support.

[7] TensorFlow Developers. (2023). *TensorFlow: An End-to-End Open Source Machine Learning Platform*. Retrieved from: https://www.tensorflow.org

[8] XGBoost Developers. (2023). *XGBoost Documentation*. Retrieved from: https://xgboost.readthedocs.io

[9] Lundberg, S.M., & Lee, S.I. (2017). *A Unified Approach to Interpreting Model Predictions*. Advances in Neural Information Processing Systems.

Acknowledgment of GPT Use

Some portions of the modeling code, backtesting logic Sections 6 and 7 were developed with the help of OpenAI's ChatGPT. The tool was used to refine Python syntax, assist with debugging ML models (e.g., Random Forest, XGBoost, LSTM), and improve explanation clarity. All strategy simulations, interpretations, and final decisions were independently reviewed and completed by us as the authors of this report.

Appendix Section 6 and 7 Machine Learning-Based Market Forecasting and Strategy Backtesting for Energy Assets and Benchmarks

In this section, we define the predictive goal as a binary classification task—predicting the direction of next-day returns (up/down) for selected financial instruments.

We focus on 7 entities:

Stocks: XOM, CVX, BP, Shell

Market Index & Commodities: SPY, Brent, WTI

This sets the foundation for applying ML models and building a strategy to generate alpha over traditional Buy & Hold.

Data Preparation & Feature Engineering This block covers the loading of raw data (prices + news) from a MySQL database, calculating daily sentiment scores, and creating derived features such as lagged returns, moving averages, volatility, and the classification target for all 7 entities.

[ ]:
```
import pandas as pd
from sqlalchemy import create_engine
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Initialize database connection and sentiment analyzer
engine = create_engine("mysql+pymysql://Varsha:Raman%401976@localhost:3306/
 ↪fds_project_db")
analyzer = SentimentIntensityAnalyzer()

# Define entities and table mappings
entities = {
    "xom":   {"data_table": "xom_data", "news_table": "xom_news", "close_col":␣
 ↪"Close", "has_volume": True},
    "cvx":   {"data_table": "cvx_data", "news_table": "cvx_news", "close_col":␣
 ↪"Close", "has_volume": True},
    "bp":    {"data_table": "bp_data", "news_table": "bp_news", "close_col":␣
 ↪"Close", "has_volume": True},
    "shell": {"data_table": "shell_data", "news_table": "shell_news",␣
 ↪"close_col": "Close", "has_volume": True},
    "spy":   {"data_table": "spy_data", "news_table": "spy_news", "close_col":␣
 ↪"Close", "has_volume": True},
```

```
    "brent": {"data_table": "brent_wti_data", "news_table": "brent_news",␣
 ↪"close_col": "Brent_Close", "has_volume": False},
    "wti":   {"data_table": "brent_wti_data", "news_table": "wti_news",␣
 ↪"close_col": "WTI_Close", "has_volume": False},
}

# Calculate daily sentiment score
def calculate_sentiment(news_table):
    query = f"""
        SELECT published_date AS Date, title, summary
        FROM {news_table}
        WHERE published_date BETWEEN '2019-01-01' AND '2025-01-31'
    """
    df_news = pd.read_sql(query, engine)
    df_news["Date"] = pd.to_datetime(df_news["Date"])
    df_news["text"] = df_news["title"].fillna("") + " " + df_news["summary"].
 ↪fillna("")
    df_news["Sentiment"] = df_news["text"].apply(lambda x: analyzer.
 ↪polarity_scores(x)["compound"])
    df_daily = df_news.groupby("Date")["Sentiment"].mean().reset_index()
    return df_daily.set_index("Date")

# Load and merge stock price with sentiment
def load_merged_data(entity_key, config):
    try:
        sentiment_df = calculate_sentiment(config["news_table"])

        stock_query = f"""
            SELECT Date, {config['close_col']} AS Close{', Volume' if␣
 ↪config['has_volume'] else ''}
            FROM {config['data_table']}
            WHERE Date BETWEEN '2019-01-01' AND '2025-01-31'
            ORDER BY Date
        """
        stock_df = pd.read_sql(stock_query, engine)
        stock_df["Date"] = pd.to_datetime(stock_df["Date"])
        stock_df.set_index("Date", inplace=True)

        merged = stock_df.join(sentiment_df, how="left")
        merged.sort_index(inplace=True)
        return merged

    except Exception as e:
        print(f"Error loading {entity_key.upper()}: {e}")
        return None

# Feature engineering and target creation
```

```python
def prepare_features(df):
    df = df.copy()

    # Store Close for plotting later
    df["Close_Original"] = df["Close"]

    # Calculate daily return
    df["Return"] = df["Close"].pct_change(fill_method=None)

    # Core technical indicators
    df["Lag1"] = df["Return"].shift(1)
    df["Lag3"] = df["Return"].shift(3)
    df["MA_7"] = df["Close"].rolling(window=7).mean()
    df["MA_30"] = df["Close"].rolling(window=30).mean()
    df["Volatility_7"] = df["Return"].rolling(window=7).std()

    # Sentiment features
    df["Sentiment_Lag1"] = df["Sentiment"].shift(1)
    df["News_Volume"] = df["Sentiment"].notnull().astype(int)

    # Bollinger Bands
    df["MA_20"] = df["Close"].rolling(window=20).mean()
    df["STD_20"] = df["Close"].rolling(window=20).std()
    df["UpperBB"] = df["MA_20"] + 2 * df["STD_20"]
    df["LowerBB"] = df["MA_20"] - 2 * df["STD_20"]
    df["BB_Position"] = (df["Close"] - df["MA_20"]) / (df["UpperBB"] -␣
 ↪df["LowerBB"])

    # Target: binary classification
    df["Target"] = (df["Return"].shift(-1) > 0).astype(int)

    # Final columns to keep
    selected_cols = [
        "Close_Original", "Return", "Lag1", "Lag3", "MA_7", "MA_30",␣
 ↪"Volatility_7",
        "Sentiment_Lag1", "News_Volume", "BB_Position", "Target"
    ]
    df = df[selected_cols]

    df.dropna(inplace=True)
    return df


# Process all entities
all_data = {}
prepared_data = {}
```

```
for key, config in entities.items():
    print(f"Loading {key.upper()}...")
    df_merged = load_merged_data(key, config)
    if df_merged is not None:
        df_prepared = prepare_features(df_merged)
        all_data[key] = df_merged
        prepared_data[key] = df_prepared
        print(f"{key.upper()} processed. Rows: {df_prepared.shape[0]}, Columns:␣
 ↪{df_prepared.shape[1]}")
    else:
        print(f"Skipping {key.upper()} due to loading error.")

# Show sample preview
sample_entity = "xom"
print(f"\nSample of {sample_entity.upper()} prepared data:")
print(prepared_data[sample_entity].tail())
```

Exploratory Visualization of Engineered Features This code generates a multi-entity, multi-feature time series grid showing trends for key features like Lag1, MA_7, and Sentiment_Lag1 across all entities. It provides a visual understanding of how technical and sentiment indicators evolve over time before modeling.

Plot Moving Averages (MA_7, MA_30) with Close Price

[276]:
```
import matplotlib.pyplot as plt

def plot_moving_averages(df, title='Moving Averages'):
    plt.figure(figsize=(14, 6))

    if "Close_Original" in df.columns:
        plt.plot(df.index, df["Close_Original"], label="Close", linewidth=2)
    else:
        raise KeyError("'Close_Original' column is missing in DataFrame")

    if "MA_7" in df.columns:
        plt.plot(df.index, df["MA_7"], label="MA_7", linestyle="--")

    if "MA_30" in df.columns:
        plt.plot(df.index, df["MA_30"], label="MA_30", linestyle="--")

    plt.title(title)
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Plot Volatility and Bollinger Band Position

[278]:
```
def plot_volatility_and_bb(df, title='Volatility & BB_Position'):
    plt.figure(figsize=(14, 6))
    plt.plot(df.index, df["Volatility_7"], label="Volatility_7")
    plt.plot(df.index, df["BB_Position"], label="BB_Position")
    plt.title(title)
    plt.xlabel("Date")
    plt.ylabel("Value")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Plot Sentiment Score Distribution

[280]:
```
import seaborn as sns

def plot_sentiment_distribution(df, title="Sentiment Score Distribution"):
    plt.figure(figsize=(8, 5))
    sns.histplot(df["Sentiment_Lag1"].dropna(), bins=30, kde=True,␣
 ↪color="steelblue")
    plt.title(title)
    plt.xlabel("Sentiment Score")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Plot Target Class Balance

[282]:
```
import seaborn as sns
import matplotlib.pyplot as plt

def plot_target_distribution(df, title="Target Class Distribution"):
    plt.figure(figsize=(6, 4))
    ax = sns.countplot(x="Target", data=df, hue="Target", palette="pastel",␣
 ↪legend=False)

    # Annotate the bars with their counts
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{int(height)}', (p.get_x() + p.get_width() / 2., height +␣
 ↪2),
                    ha='center', va='center', fontsize=10, color='black')

    plt.title(title)
    plt.xlabel("Direction (0 = Down, 1 = Up)")
    plt.ylabel("Count")
    plt.grid(True, linestyle="--", alpha=0.6)
```

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

[ ]:
```
sample_df = prepared_data["xom"]

plot_moving_averages(sample_df, title="XOM - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="XOM - Volatility & Bollinger Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["shell"]

plot_moving_averages(sample_df, title="Shell - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="Shell - Volatility & Bollinger␣
↪Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["cvx"]

plot_moving_averages(sample_df, title="CVX - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="CVX - Volatility & Bollinger Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["bp"]

plot_moving_averages(sample_df, title="bp - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="bp - Volatility & Bollinger Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["spy"]

plot_moving_averages(sample_df, title="SPY - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="SPY - Volatility & Bollinger Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["brent"]

plot_moving_averages(sample_df, title="BRENT - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="BRENT - Volatility & Bollinger␣
↪Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

[ ]:
```
sample_df = prepared_data["wti"]

plot_moving_averages(sample_df, title="WTI - Close & Moving Averages")
plot_volatility_and_bb(sample_df, title="WTI - Volatility & Bollinger Position")
plot_sentiment_distribution(sample_df)
plot_target_distribution(sample_df)
```

Feature Correlation Heatmaps This section generates correlation heatmaps for all entities, highlighting relationships between features like Return, Lag1, Lag3, MA_7, Volatility_7, and Sentiment_Lag1. It helps assess multicollinearity and understand how sentiment interacts with traditional indicators.

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Get entity list from prepared_data
entities = list(prepared_data.items())
n_entities = len(entities)

# Grid configuration
n_cols = 4
n_rows = (n_entities + n_cols - 1) // n_cols  # auto-calculate rows based on␣
 ↪count
fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 5 * n_rows))
axes = axes.flatten()

# Loop through each entity and plot heatmap
for idx, (entity, df) in enumerate(entities):
    ax = axes[idx]
    try:
        # Select features safely
        corr_features = ['Return', 'Lag1', 'Lag3', 'MA_7', 'Volatility_7',␣
 ↪'Sentiment_Lag1']
        corr = df[corr_features].corr()

        # Plot heatmap
        sns.heatmap(corr, annot=True, cmap='coolwarm', center=0, ax=ax,␣
 ↪cbar=False)
        ax.set_title(f"{entity.upper()} - Feature Correlation")
    except Exception as e:
        ax.set_visible(False)
        print(f"Skipped {entity.upper()} due to error: {e}")

# Remove unused axes if any
for idx in range(n_entities, n_rows * n_cols):
    fig.delaxes(axes[idx])
```

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

Model Evaluation

[ ]:
```
# -------------------- IMPORTS --------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import math

from sklearn.model_selection import TimeSeriesSplit
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

warnings.filterwarnings("ignore")

# -------------------- LSTM Helper --------------------
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, input_shape=input_shape))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(0.001), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

# -------------------- Configuration --------------------
features = ['Lag1', 'Lag3', 'MA_7', 'Volatility_7', 'Sentiment_Lag1']
tscv = TimeSeriesSplit(n_splits=5)
all_results = []
confusion_plots = []

# -------------------- Loop Over Entities --------------------
for entity, df in prepared_data.items():
    print(f"\nProcessing: {entity.upper()}")
    df = df.copy()
```

```
    if "Return" not in df.columns:
        if "Close" in df.columns:
            df["Return"] = df["Close"].pct_change()
        else:
            raise KeyError(f"Missing 'Close' or 'Return' in {entity.upper()}")
    df.dropna(inplace=True)

    X = df[features]
    y = df["Target"]
    dates = df.index

    # LSTM scaling
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    X_lstm = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

    # --- Traditional Models ---
    models = {
        "Logistic Regression": LogisticRegression(max_iter=1000,␣
↪class_weight='balanced'),
        "Random Forest": RandomForestClassifier(n_estimators=100,␣
↪class_weight='balanced', random_state=42),
        "XGBoost": XGBClassifier(use_label_encoder=False,␣
↪eval_metric='logloss', random_state=42)
    }

    for model_name, model in models.items():
        print(f"{model_name}")
        scores = {"Accuracy": [], "Precision": [], "Recall": [], "F1": []}
        total_cm = None

        for train_idx, test_idx in tscv.split(X):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            scores["Accuracy"].append(accuracy_score(y_test, y_pred))
            scores["Precision"].append(precision_score(y_test, y_pred))
            scores["Recall"].append(recall_score(y_test, y_pred))
            scores["F1"].append(f1_score(y_test, y_pred))

            cm = confusion_matrix(y_test, y_pred)
            total_cm = cm if total_cm is None else total_cm + cm
```

```
        avg = {metric: np.mean(val) for metric, val in scores.items()}
        avg["Entity"] = entity.upper()
        avg["Model"] = model_name
        all_results.append(avg)
        confusion_plots.append((f"{entity.upper()} - {model_name}", total_cm))

    # --- LSTM Model ---
    print("LSTM")
    scores = {"Accuracy": [], "Precision": [], "Recall": [], "F1": []}
    total_cm = None

    input_shape = (X_lstm.shape[1], X_lstm.shape[2])

    for fold_num, (train_idx, test_idx) in enumerate(tscv.split(X_lstm),␣
↪start=1):
        print(f"LSTM Fold {fold_num} | Train: {len(train_idx)} | Test:␣
↪{len(test_idx)}")

        X_train_lstm, X_test_lstm = X_lstm[train_idx], X_lstm[test_idx]
        y_train_lstm, y_test_lstm = y.iloc[train_idx], y.iloc[test_idx]

        lstm = create_lstm_model(input_shape)
        lstm.fit(
            X_train_lstm, y_train_lstm,
            epochs=10,
            batch_size=16,
            verbose=1,
            callbacks=[EarlyStopping(patience=2, restore_best_weights=True)]
        )

        y_pred_probs = lstm.predict(X_test_lstm, verbose=0)
        y_pred = (y_pred_probs > 0.5).astype(int).flatten()

        scores["Accuracy"].append(accuracy_score(y_test_lstm, y_pred))
        scores["Precision"].append(precision_score(y_test_lstm, y_pred))
        scores["Recall"].append(recall_score(y_test_lstm, y_pred))
        scores["F1"].append(f1_score(y_test_lstm, y_pred))

        cm = confusion_matrix(y_test_lstm, y_pred)
        total_cm = cm if total_cm is None else total_cm + cm

    avg = {metric: np.mean(val) for metric, val in scores.items()}
    avg["Entity"] = entity.upper()
    avg["Model"] = "LSTM"
    all_results.append(avg)
    confusion_plots.append((f"{entity.upper()} - LSTM", total_cm))
```

```
# -------------------- CONFUSION MATRIX PLOTS --------------------
n = len(confusion_plots)
cols = 3
rows = math.ceil(n / cols)

fig, axes = plt.subplots(rows, cols, figsize=(5 * cols, 4 * rows))
axes = axes.flatten()

for i, (title, cm) in enumerate(confusion_plots):
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=axes[i])
    axes[i].set_title(title)
    axes[i].set_xlabel("Predicted")
    axes[i].set_ylabel("Actual")

for j in range(len(confusion_plots), len(axes)):
    fig.delaxes(axes[j])

plt.suptitle("Confusion Matrices - All Models & Entities", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# -------------------- RESULTS TABLE --------------------
results_df = pd.DataFrame(all_results)
results_df = results_df[["Entity", "Model", "Accuracy", "Precision", "Recall",␣
 ↪"F1"]]
results_df.sort_values(by=["Entity", "F1"], ascending=[True, False],␣
 ↪inplace=True)

print("\nModel Evaluation Summary:\n")
display(
    results_df.style
    .highlight_max(subset=["F1"], axis=0, color="lightgreen")
    .format("{:.2%}", subset=["Accuracy", "Precision", "Recall", "F1"])
)
```

[ ]:
```
F1 SCORE VISUAL
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sort entities for consistent order (optional)
ordered_entities = results_df["Entity"].unique()

plt.figure(figsize=(12, 6))
ax = sns.barplot(
    data=results_df,
    x="Entity",
```

```
    y="F1",
    hue="Model",
    palette="Set2",
    order=ordered_entities
)

# Annotate bars with F1 values
for bar in ax.patches:
    height = bar.get_height()
    if not np.isnan(height):
        ax.annotate(f'{height:.2f}',
                    (bar.get_x() + bar.get_width() / 2., height + 0.01),
                    ha='center', va='bottom', fontsize=9, color='black')

plt.title(" F1 Score by Model and Entity", fontsize=14)
plt.ylabel("F1 Score")
plt.xlabel("Entity")
plt.ylim(0, 1.1)
plt.legend(title="Model", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

F1 SCORE HEATMAP

[ ]:
```
f1_pivot = results_df.pivot(index="Entity", columns="Model", values="F1")

plt.figure(figsize=(10, 6))
sns.heatmap(f1_pivot, annot=True, fmt=".2%", cmap="YlGnBu")
plt.title("F1 Score Heatmap: Entities vs Models")
plt.xlabel("Model")
plt.ylabel("Entity")
plt.tight_layout()
plt.show()
```

PERFORMANCE METRIC PLOTS

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Melt the results to long format
metrics_melted = results_df.melt(
    id_vars=["Entity", "Model"],
    value_vars=["Accuracy", "Precision", "Recall", "F1"],
    var_name="Metric",
    value_name="Score"
)

# Create multi-panel (facet) lineplot: one panel per metric
```

```
g = sns.FacetGrid(metrics_melted, col="Metric", hue="Model", sharey=True,␣
 ↪col_wrap=2, height=4, aspect=1.5)
g.map(sns.lineplot, "Entity", "Score", marker="o")

# Beautify
g.set_titles("{col_name}")
g.set_axis_labels("Entity", "Score")
g.set(ylim=(0, 1))
g.add_legend(title="Model")
for ax in g.axes.flat:
    ax.tick_params(axis='x', rotation=45)

plt.suptitle("Performance Metrics Across Entities by Model", fontsize=14, y=1.
 ↪05)
plt.tight_layout()
plt.show()
```

BACKTESTING STRATEGY

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import math
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# ----------------- LSTM Model Helper -----------------
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, input_shape=input_shape))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(0.001), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

# ----------------- Backtest Helper -----------------
def simulate_strategy(df, predictions, model_name):
    df = df.copy()
    df = df.loc[predictions.index]
    df["Signal"] = predictions
```

```
    df["StrategyReturn"] = df["Signal"].shift(1) * df["Return"]
    df["BuyHoldReturn"] = df["Return"]

    # Cumulative returns
    df["Cumulative_Strategy"] = (1 + df["StrategyReturn"].fillna(0)).cumprod()
    df["Cumulative_BuyHold"] = (1 + df["BuyHoldReturn"].fillna(0)).cumprod()

    # Metrics
    sharpe = df["StrategyReturn"].mean() / df["StrategyReturn"].std() * np.
 ↪sqrt(252)
    max_dd = (df["Cumulative_Strategy"] / df["Cumulative_Strategy"].cummax() -␣
 ↪1).min()
    win_rate = (df["StrategyReturn"] > 0).sum() / df["StrategyReturn"].count()

    return df, round(sharpe, 2), round(max_dd * 100, 2), round(win_rate * 100,␣
 ↪2)

# ----------------- Main Loop -----------------
from sklearn.model_selection import TimeSeriesSplit

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss",␣
 ↪random_state=42),
}

features = ['Lag1', 'Lag3', 'MA_7', 'Volatility_7', 'Sentiment_Lag1']
results_summary = []
plot_data = []

tscv = TimeSeriesSplit(n_splits=5)

for entity, df in prepared_data.items():
    print(f"\n Backtesting on: {entity.upper()}")

    df = df.copy()
    df["Return"] = df["Return"] if "Return" in df else df["Close"].pct_change()
    df.dropna(inplace=True)

    X = df[features]
    y = df["Target"]

    # Time split: get last fold
    for fold_idx, (train_idx, test_idx) in enumerate(tscv.split(X)):
        if fold_idx == 4:
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
```

```
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
            test_dates = df.index[test_idx]
            break

    # ------- Classic Models -------
    for model_name, model in models.items():
        model.fit(X_train, y_train)
        preds = pd.Series(model.predict(X_test), index=test_dates)
        bt_df, sharpe, max_dd, win_rate = simulate_strategy(df.loc[test_dates],␣
↪preds, model_name)

        results_summary.append({
            "Entity": entity.upper(),
            "Model": model_name,
            "Sharpe": sharpe,
            "Max Drawdown (%)": max_dd,
            "Win Rate (%)": win_rate,
            "Final Return": round(bt_df["Cumulative_Strategy"].iloc[-1], 3)
        })

        plot_data.append((bt_df, entity.upper(), model_name))

    # ------- LSTM -------
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    X_lstm = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

    X_train_lstm, X_test_lstm = X_lstm[train_idx], X_lstm[test_idx]
    y_train_lstm = y.iloc[train_idx]
    lstm = create_lstm_model((X_train_lstm.shape[1], X_train_lstm.shape[2]))

    print(f" Training LSTM for {entity.upper()}")
    lstm.fit(
        X_train_lstm, y_train_lstm,
        epochs=10,
        batch_size=16,
        verbose=1,
        callbacks=[EarlyStopping(patience=2, restore_best_weights=True)]
    )
    y_pred_lstm = (lstm.predict(X_test_lstm) > 0.5).astype(int).flatten()
    preds_lstm = pd.Series(y_pred_lstm, index=test_dates)

    bt_df, sharpe, max_dd, win_rate = simulate_strategy(df.loc[test_dates],␣
↪preds_lstm, "LSTM")
    results_summary.append({
        "Entity": entity.upper(),
        "Model": "LSTM",
```

```
        "Sharpe": sharpe,
        "Max Drawdown (%)": max_dd,
        "Win Rate (%)": win_rate,
        "Final Return": round(bt_df["Cumulative_Strategy"].iloc[-1], 3)
    })
    plot_data.append((bt_df, entity.upper(), "LSTM"))

# ----------------- Plot Equity Curves -----------------
cols = 3
rows = math.ceil(len(plot_data) / cols)
fig, axes = plt.subplots(rows, cols, figsize=(6 * cols, 4 * rows))
axes = axes.flatten()

for i, (bt_df, entity, model) in enumerate(plot_data):
    ax = axes[i]
    ax.plot(bt_df.index, bt_df["Cumulative_Strategy"], label="Strategy",␣
 ↪color="navy")
    ax.plot(bt_df.index, bt_df["Cumulative_BuyHold"], linestyle="--",␣
 ↪color="orange", label="Buy & Hold")

    # Overlay buy signal
    buy_dates = bt_df[(bt_df["Signal"] == 1) & (bt_df["Signal"].shift(1) != 1)].
 ↪index
    for date in buy_dates:
        ax.axvline(date, color="green", linestyle=":", alpha=0.4)

    ax.set_title(f"{entity} - {model}", fontsize=10)
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
    ax.tick_params(axis='x', rotation=45)
    ax.grid(True, linestyle=":")
    ax.legend(fontsize=8)

for j in range(len(plot_data), len(axes)):
    fig.delaxes(axes[j])

plt.suptitle(" Equity Curve - Strategy vs Buy & Hold", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# ----------------- Tabulate Results -----------------
results_df = pd.DataFrame(results_summary)
results_df.sort_values(by=["Entity", "Sharpe"], ascending=[True, False],␣
 ↪inplace=True)

from IPython.display import display
print("Strategy Simulation Summary:\n")
display(
```

```
    results_df.style
    .highlight_max(subset=["Sharpe"], axis=0, color='lightgreen')
    .format({
        "Sharpe": "{:.2f}",
        "Max Drawdown (%)": "{:.1f}",
        "Win Rate (%)": "{:.1f}",
        "Final Return": "{:.3f}"
    })
)
```

Feature importance

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import math
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit

# ==== 1. Define models ====
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000)
}

features = ['Lag1', 'Lag3', 'MA_7', 'Volatility_7', 'Sentiment_Lag1']

# ==== 2. Store importances ====
importance_data = []

for entity, df in prepared_data.items():
    X = df[features]
    y = df["Target"]

    for name, model in models.items():
        model.fit(X, y)

        if name == "Logistic Regression":
            importances = np.abs(model.coef_[0])
        else:
```

```
            importances = model.feature_importances_

        importance_data.append({
            "Entity": entity.upper(),
            "Model": name,
            "Importance": importances,
            "Feature_Names": features
        })

# ==== 3. Plot ====
n_plots = len(importance_data)
cols = 3
rows = math.ceil(n_plots / cols)

fig, axes = plt.subplots(rows, cols, figsize=(6 * cols, 4 * rows))
axes = axes.flatten()

for i, item in enumerate(importance_data):
    df_plot = pd.Series(item["Importance"], index=item["Feature_Names"]).
 ↪sort_values()
    ax = axes[i]
    df_plot.plot(kind='barh', ax=ax, color="skyblue")

    # Annotate importance values
    for idx, value in enumerate(df_plot.values):
        ax.text(value + 0.005, idx, f"{value:.2f}", va='center', fontsize=8,␣
 ↪color="black")

    ax.set_title(f"{item['Entity']} - {item['Model']}")
    ax.set_xlabel("Importance")
    ax.grid(True, linestyle=":")


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

plt.suptitle("Global Feature Importances - All Entities & Models", fontsize=16,␣
 ↪y=1.02)
plt.tight_layout()
plt.show()
```

BUSINESS IMPACT ALPHA GENERATION

[ ]:
```
import pandas as pd
from IPython.display import display
```

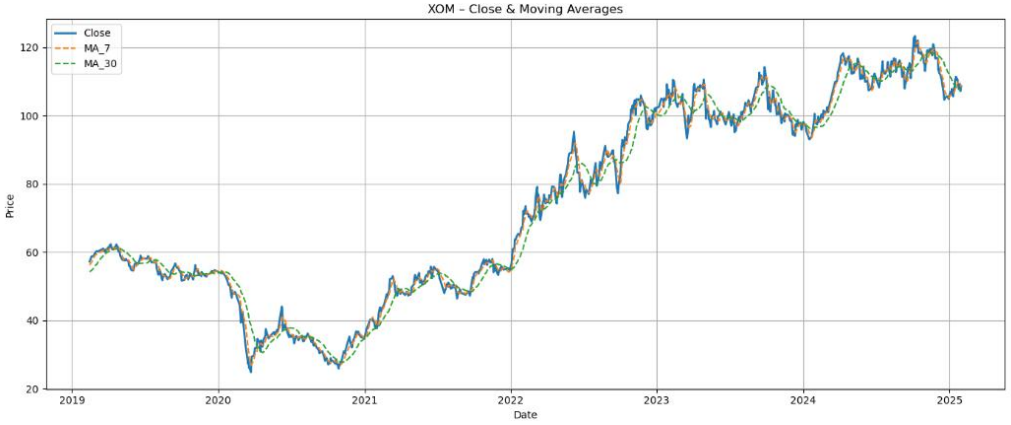
```python
# ---------------- STEP 1: Alpha Calculation ----------------
# We assume `results_df` from your strategy code has:
# ["Entity", "Model", "Final Return", "Cumulative_BuyHold"] or similar.
# If you have Cumulative_BuyHold stored separately, update accordingly.

# Let's assume you still have `plot_data` with cumulative values
alpha_summary = []

for bt_df, entity, model in plot_data:
    final_strategy = bt_df["Cumulative_Strategy"].iloc[-1]
    final_buyhold = bt_df["Cumulative_BuyHold"].iloc[-1]
    alpha = final_strategy - final_buyhold

    alpha_summary.append({
        "Entity": entity,
        "Model": model,
        "Strategy Return": round(final_strategy, 4),
        "Buy & Hold Return": round(final_buyhold, 4),
        "Alpha": round(alpha, 4)
    })

alpha_df = pd.DataFrame(alpha_summary)

# ---------------- STEP 2: Best Alpha Per Entity ----------------
best_models = alpha_df.sort_values("Alpha", ascending=False)\
                      .groupby("Entity")\
                      .first()\
                      .reset_index()

# ---------------- STEP 3: Dollar Alpha Calculation ----------------
aum_levels = [1e8, 1e9, 5e9]
for aum in aum_levels:
    best_models[f"${int(aum):,} AUM"] = (best_models["Alpha"] * aum).round(2)

# ---------------- STEP 4: Present Summary Table ----------------
styled = best_models[["Entity", "Model", "Alpha", "$100,000,000 AUM",␣
 ↪"$1,000,000,000 AUM", "$5,000,000,000 AUM"]]\
    .style\
    .format({
        "Alpha": "{:+.4f}",
        "$100,000,000 AUM": "${:,.0f}",
        "$1,000,000,000 AUM": "${:,.0f}",
        "$5,000,000,000 AUM": "${:,.0f}"
    })\
    .highlight_max(subset=["Alpha"], color="lightgreen", axis=0)\
    .set_caption("Business Impact: Alpha and Financial Return")
```

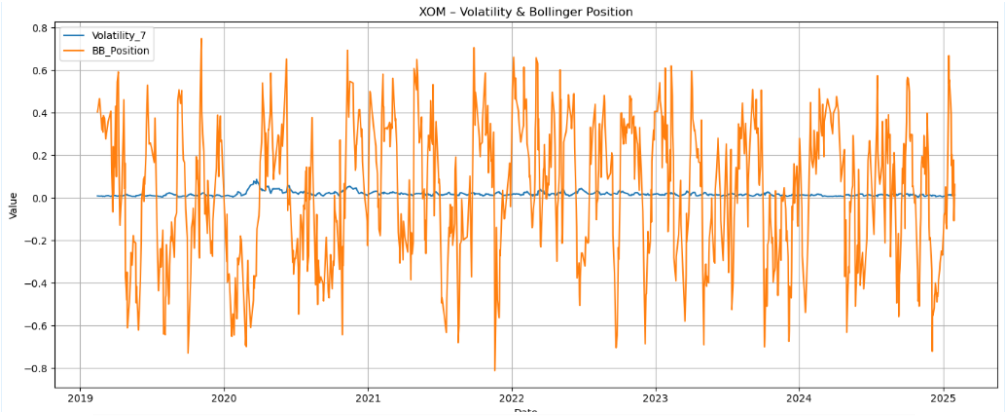
```
print("\nFund Manager Alpha Impact Summary:")
display(styled)
```

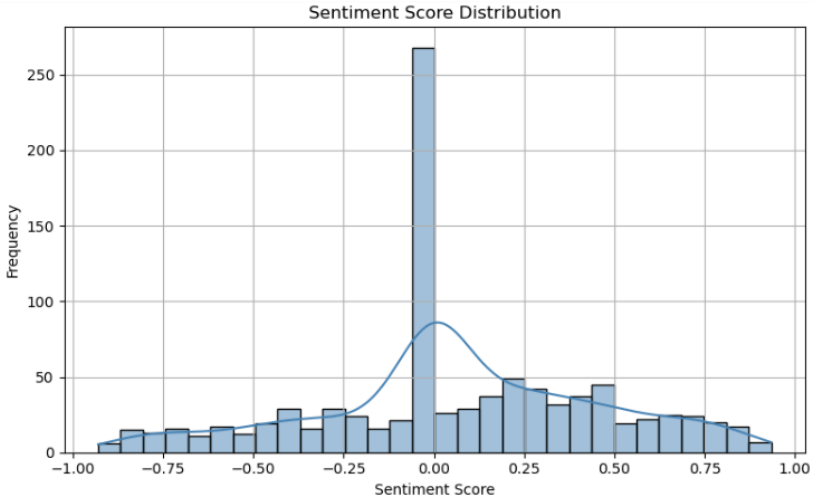
APPENDIX B – VISUALS

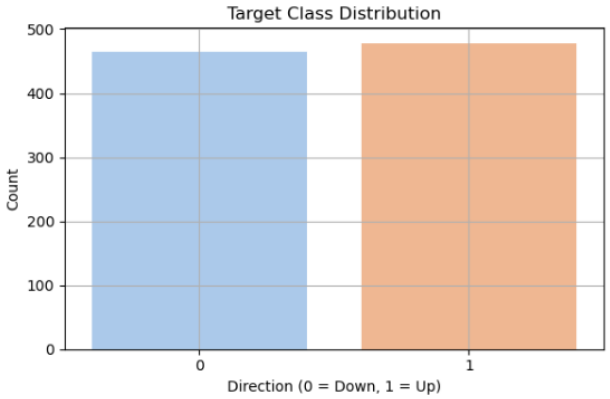
FEATURE ANALYSIS

XOM



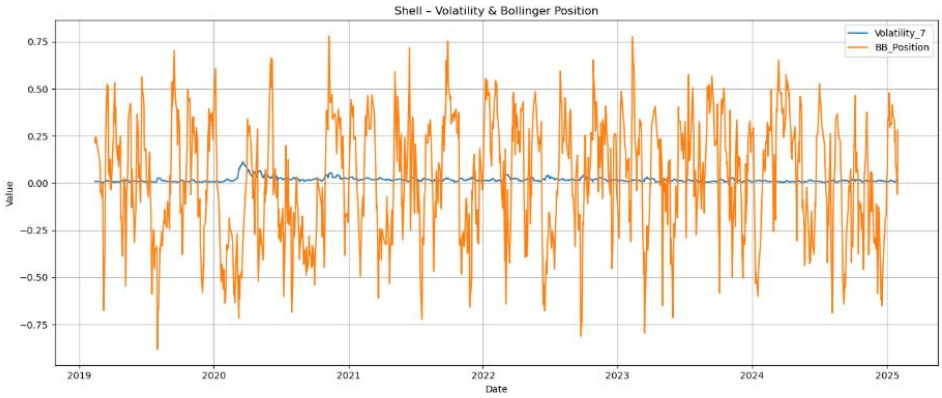


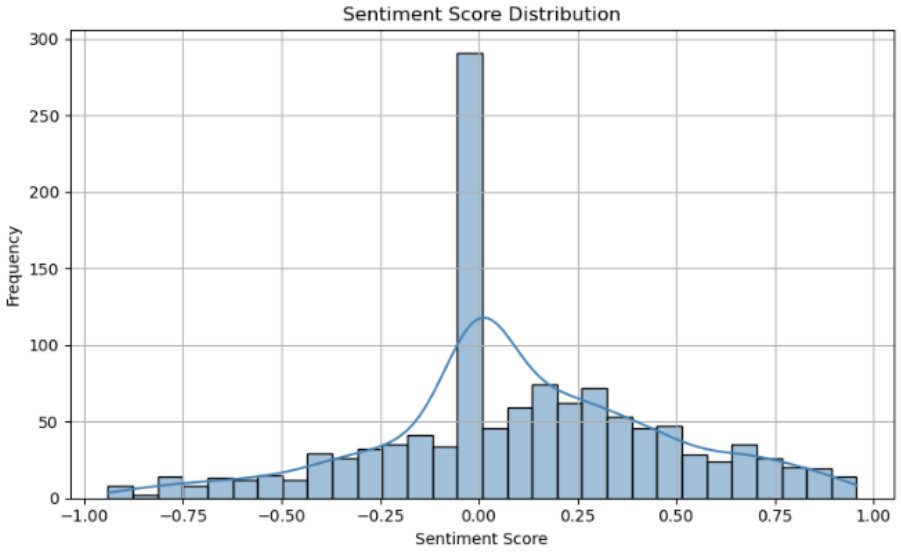


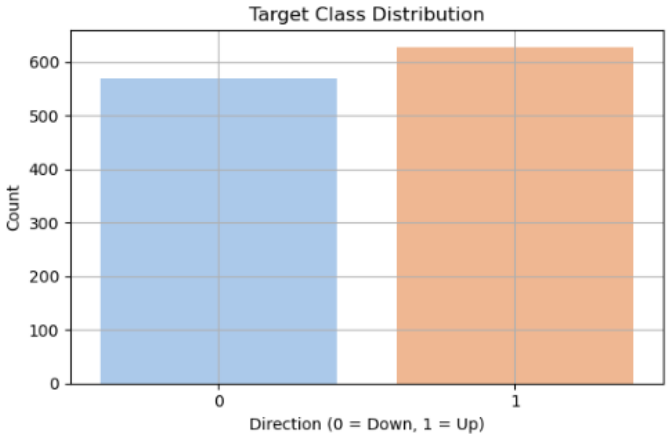


SHELL


Shell – Close & Moving Averages
Close
MA_7
MA_30
Price
Date

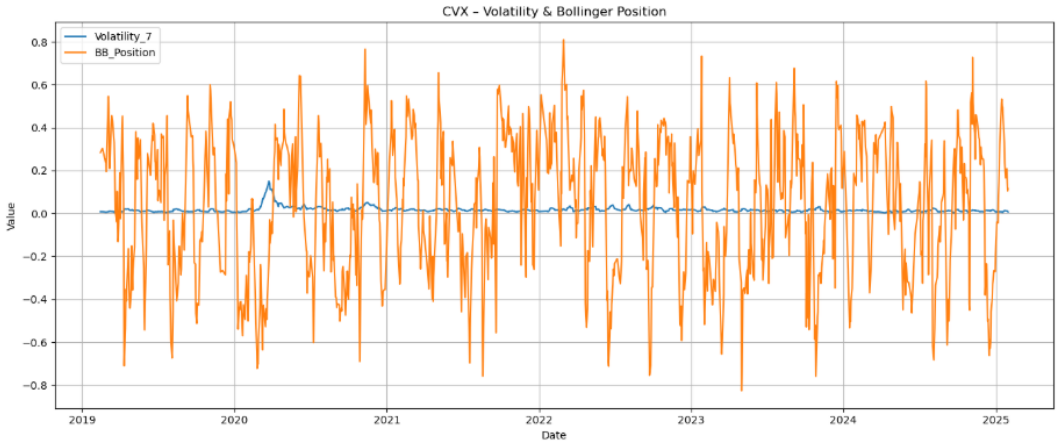

Shell – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date

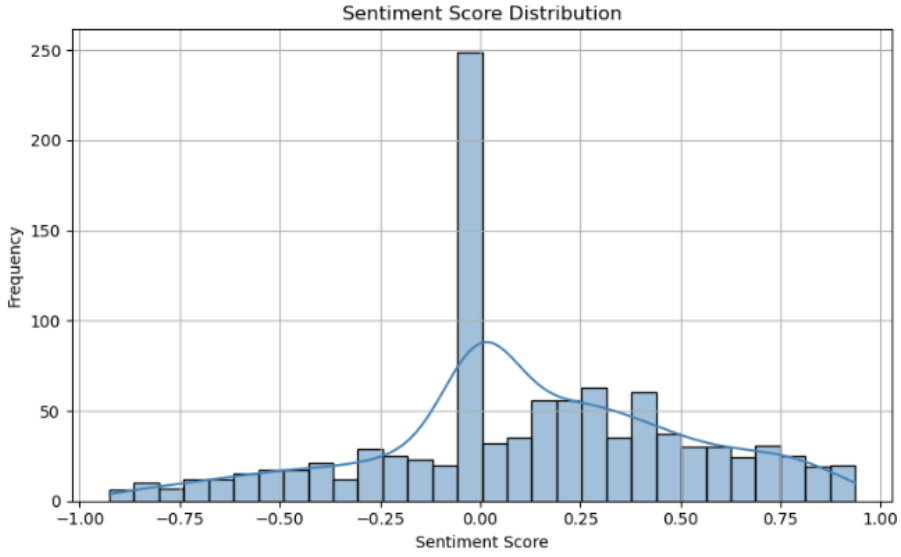

Sentiment Score Distribution
Frequency
Sentiment Score

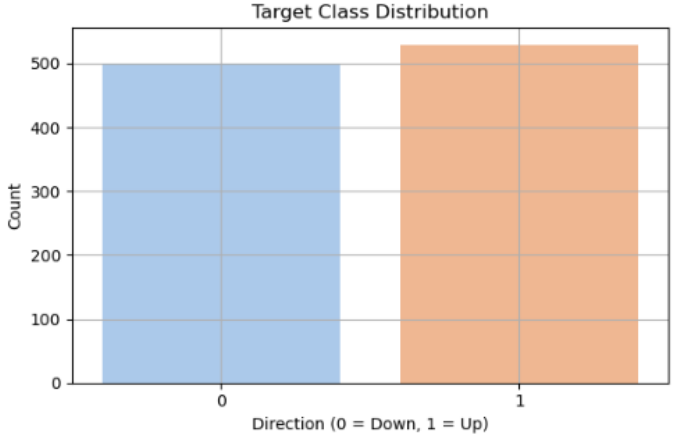

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

CVX

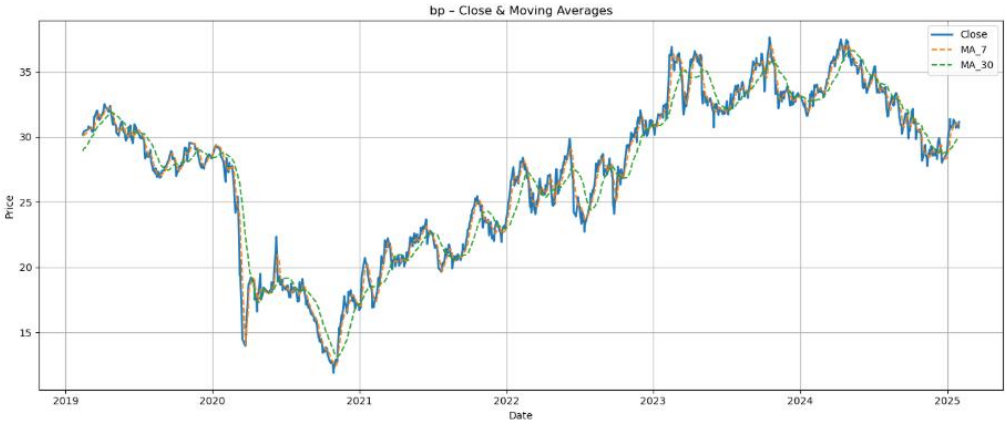

CVX – Close & Moving Averages
Close
MA_7
MA_30
Price
Date

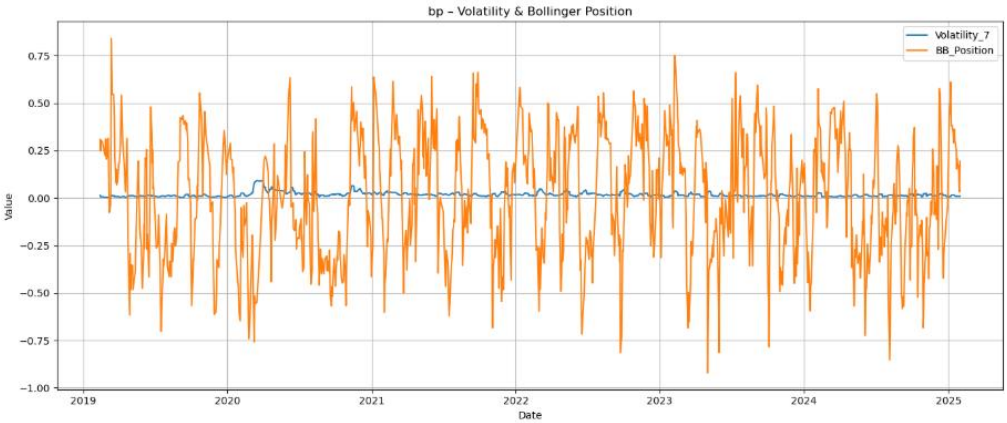

CVX – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date

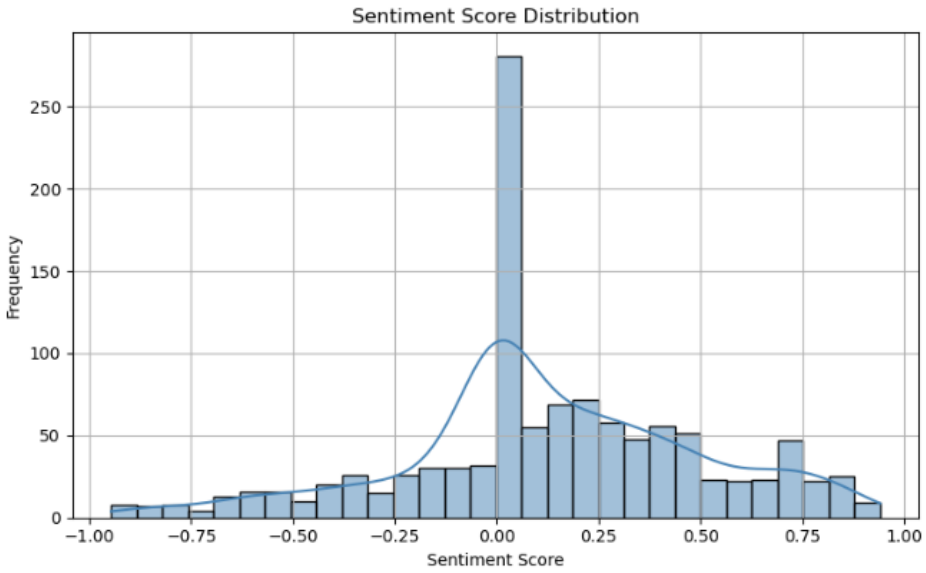

Sentiment Score Distribution
Frequency
Sentiment Score

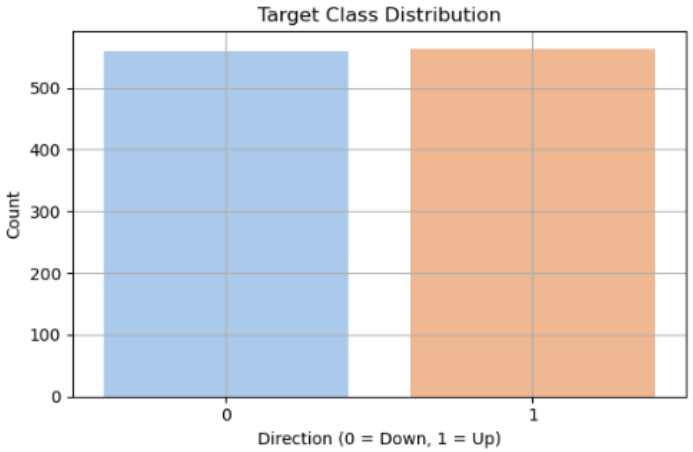

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

BP


bp – Close & Moving Averages
Close
MA_7
MA_30
Price
Date
2019
2020
2021
2022
2023
2024
2025

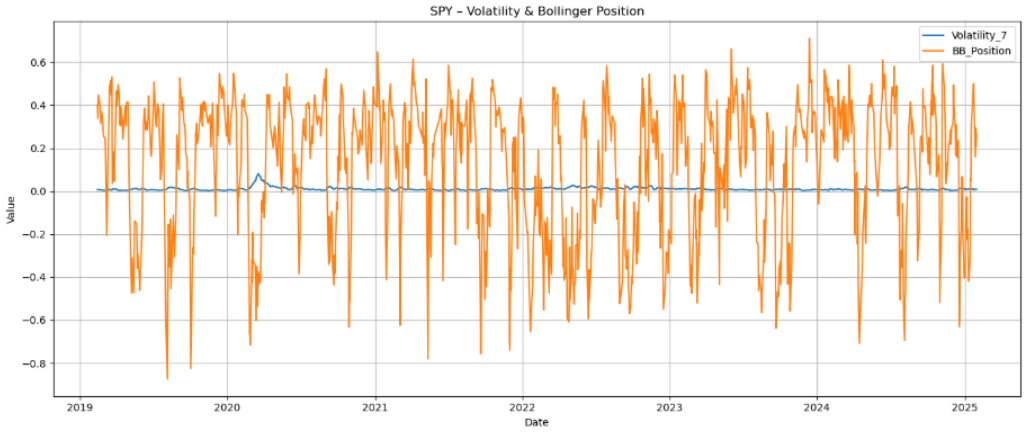

bp – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date

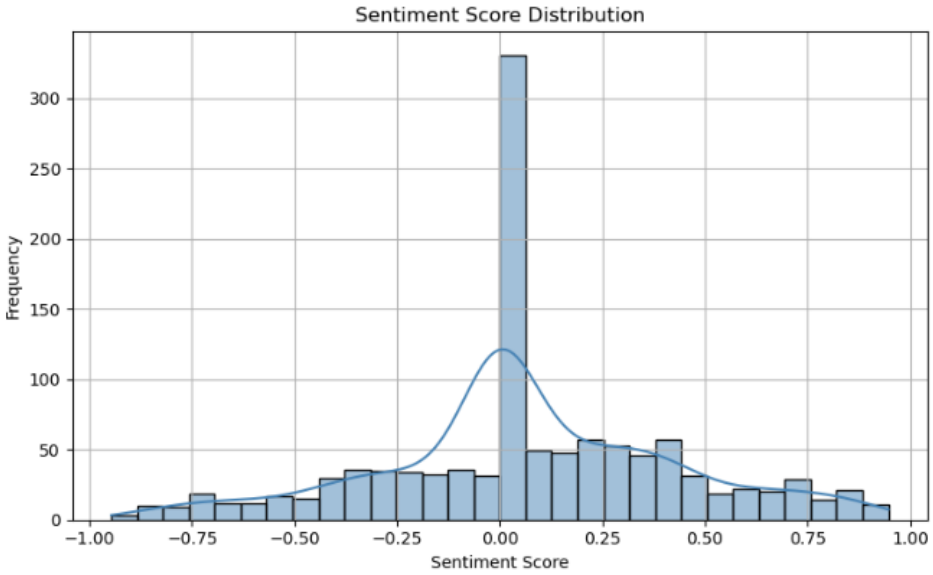

Sentiment Score Distribution
Frequency
Sentiment Score

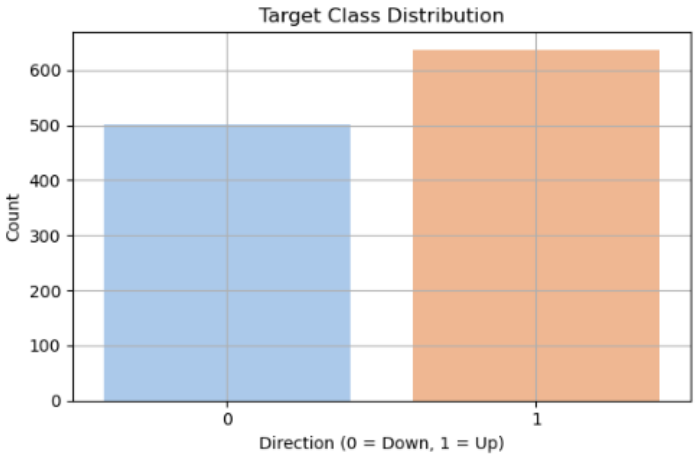

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

SPY

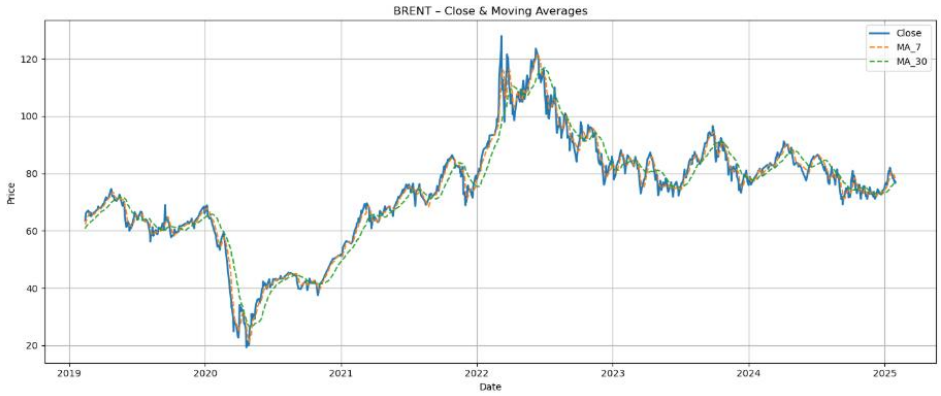

SPY – Close & Moving Averages
Close
MA_7
MA_30
Price
Date

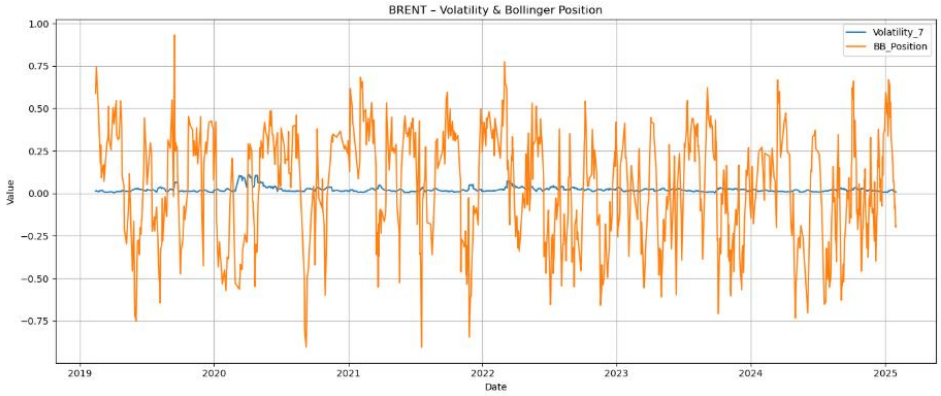

SPY – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date

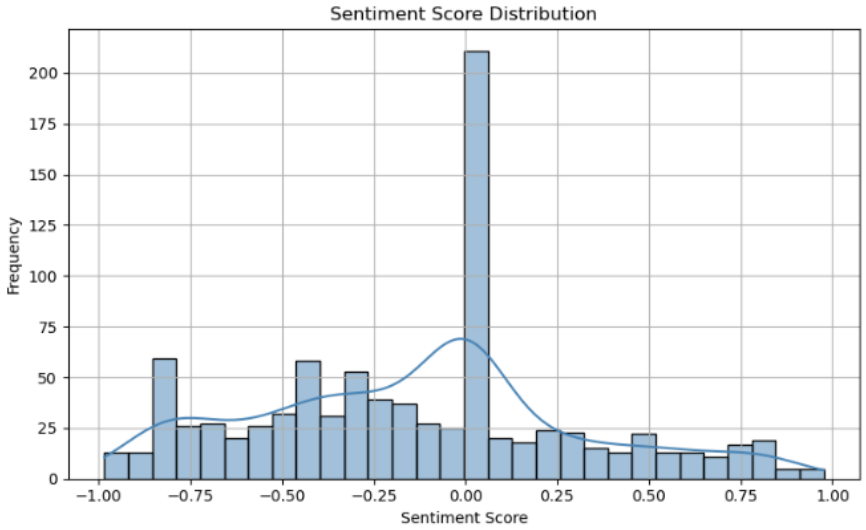

Sentiment Score Distribution
Frequency
Sentiment Score

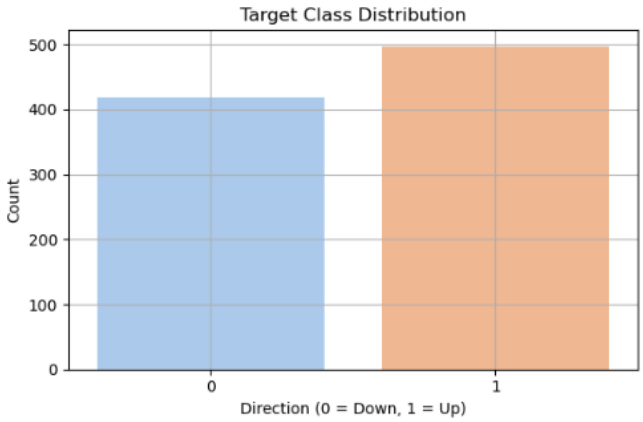

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

BRENT

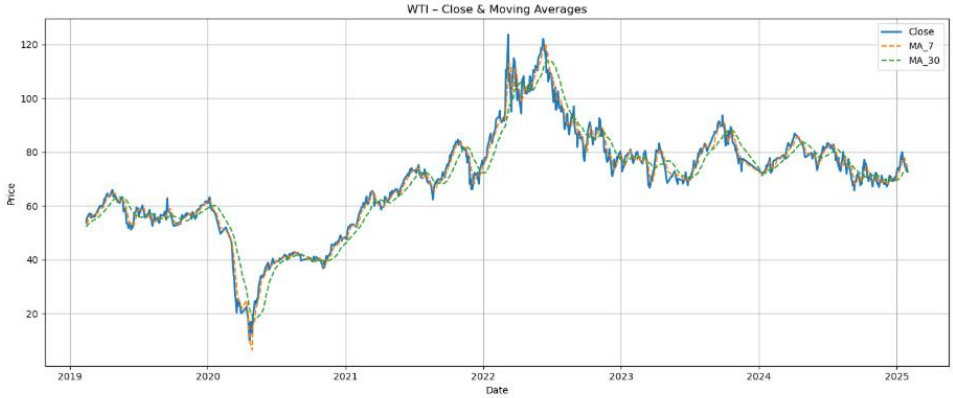

BRENT – Close & Moving Averages
Close
MA_7
MA_30
Price
Date

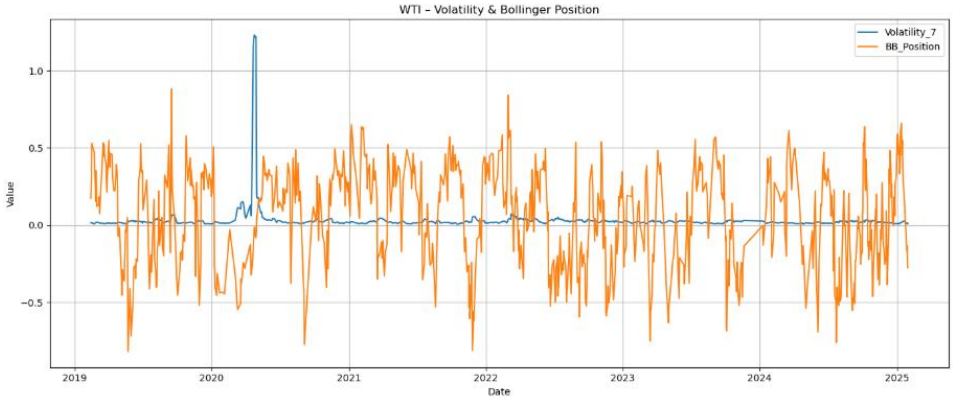

BRENT – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date

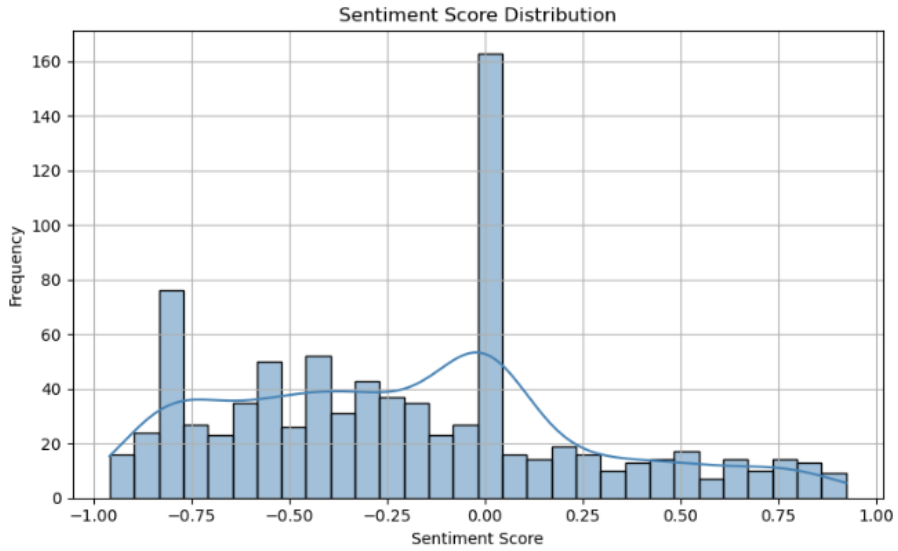

Sentiment Score Distribution
Frequency
Sentiment Score

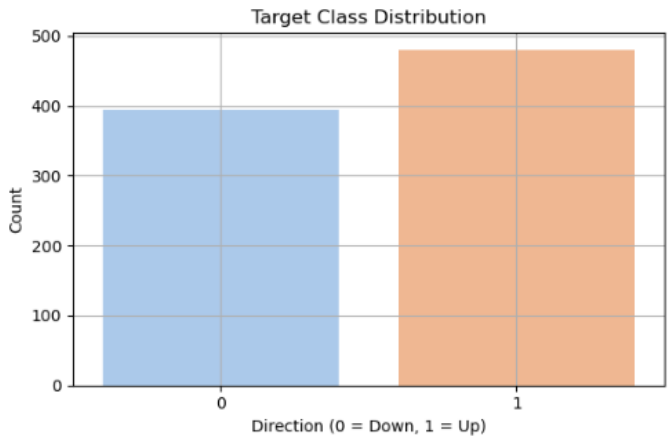

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

WTI

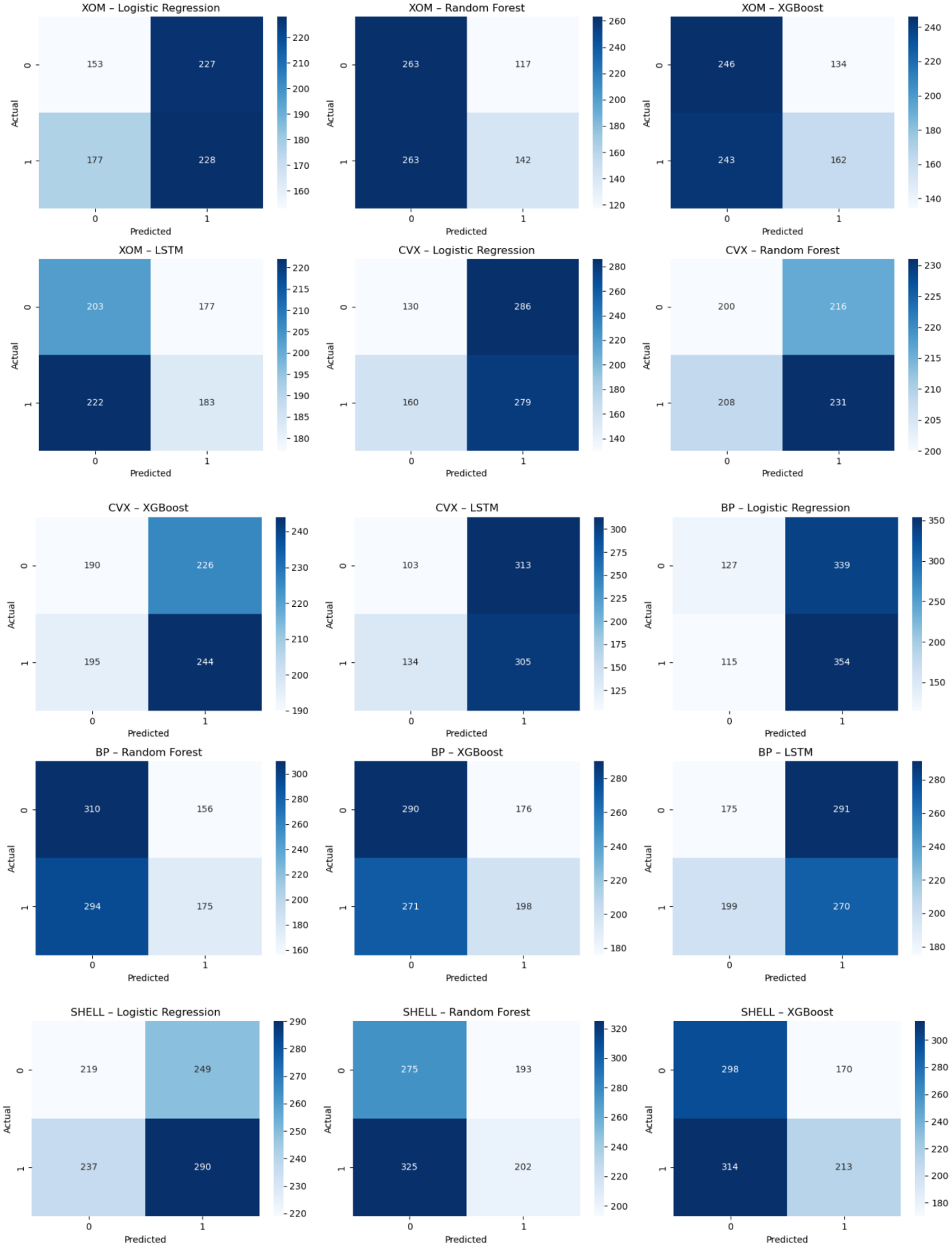

WTI – Close & Moving Averages
Close
MA_7
MA_30
Price
Date
2019
2020
2021
2022
2023
2024
2025

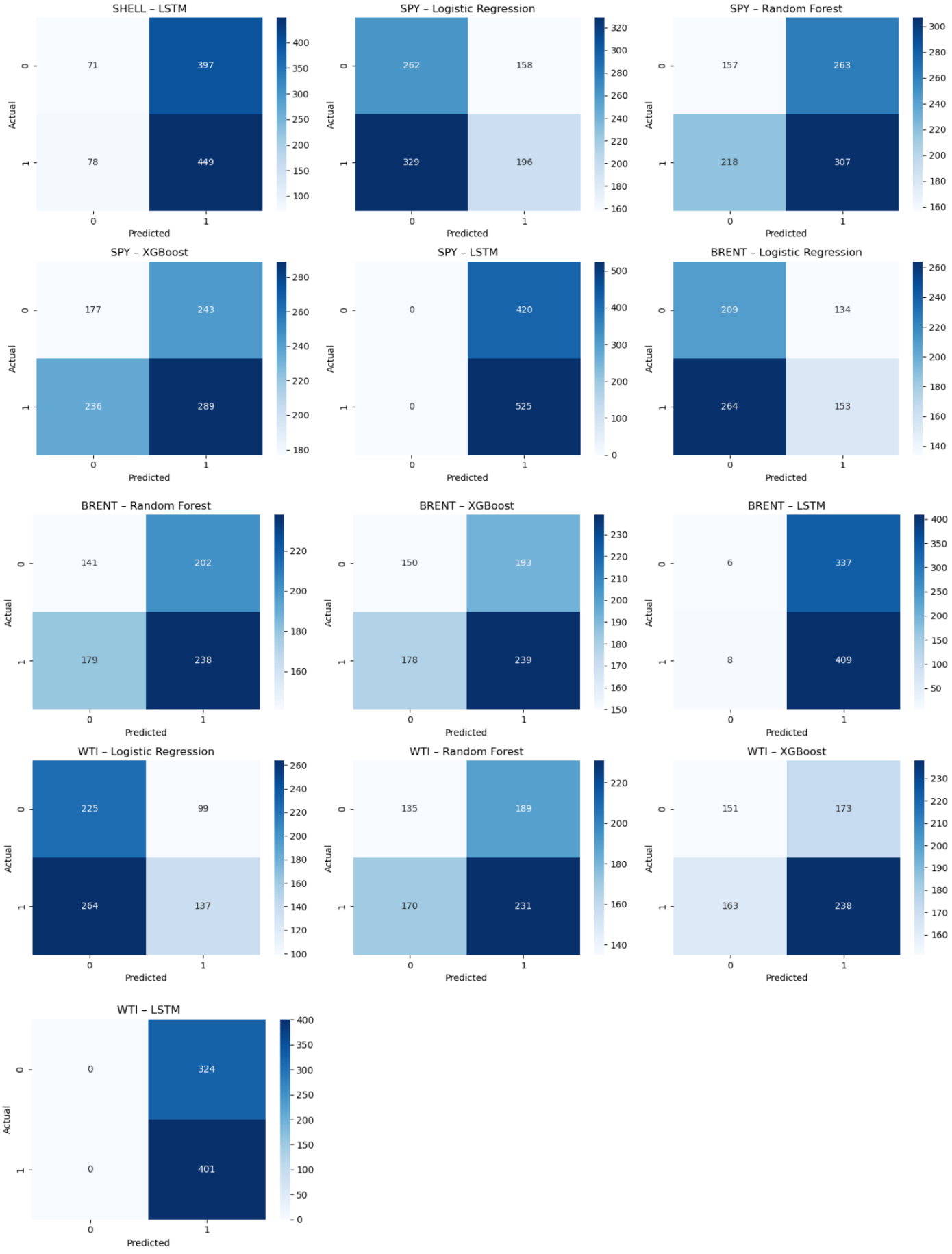

WTI – Volatility & Bollinger Position
Volatility_7
BB_Position
Value
Date


Sentiment Score Distribution
Frequency
Sentiment Score

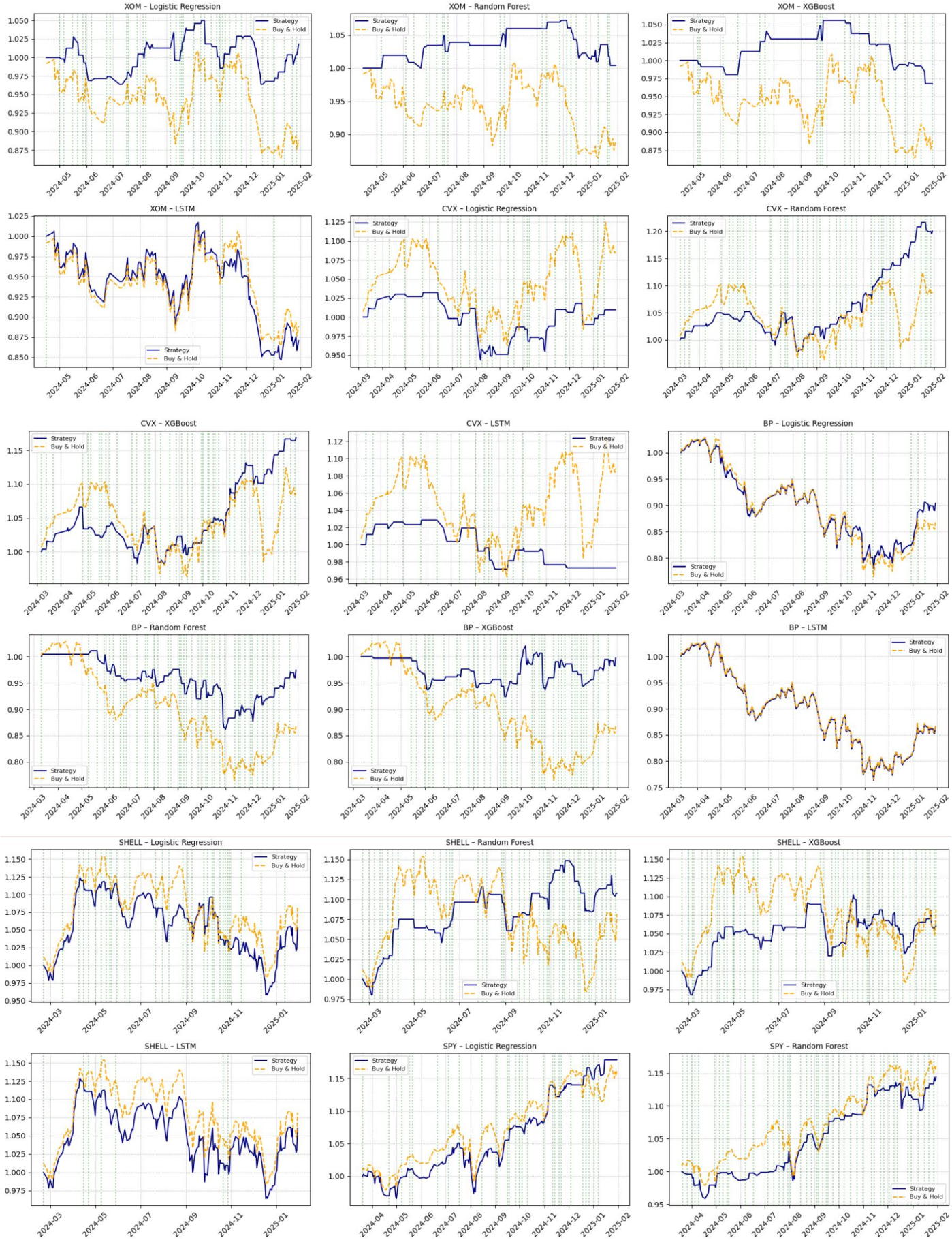

Target Class Distribution
Count
Direction (0 = Down, 1 = Up)

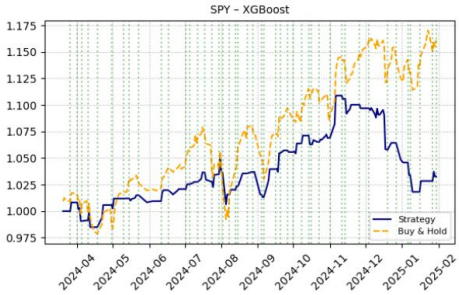
CONFUSION MATRIX



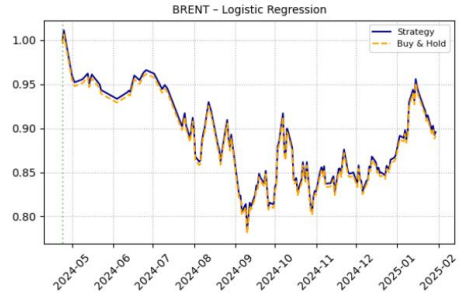

SHELL – LSTM
71
397
78
449
SPY – Logistic Regression
262
158
329
196
SPY – Random Forest
157
263
218
307
SPY – XGBoost
177
243
236
289
SPY – LSTM
0
420
0
525
BRENT – Logistic Regression
209
134
264
153
BRENT – Random Forest
141
202
179
238
BRENT – XGBoost
150
193
178
239
BRENT – LSTM
6
337
8
409
WTI – Logistic Regression
225
99
264
137
WTI – Random Forest
135
189
170
231
WTI – XGBoost
151
173
163
238
WTI – LSTM
0
324
0
401
Actual
Predicted

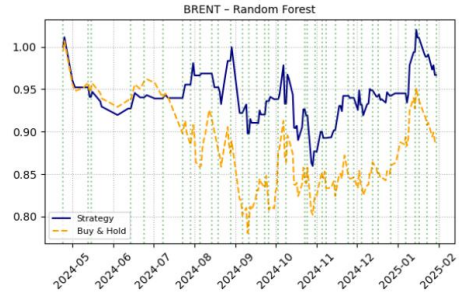
EQUITY CURVES

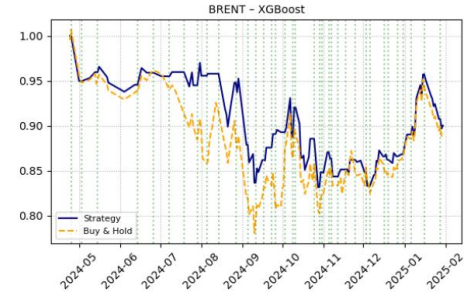


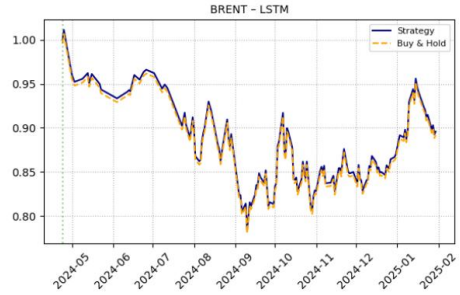

SPY – XGBoost
Strategy
Buy & Hold

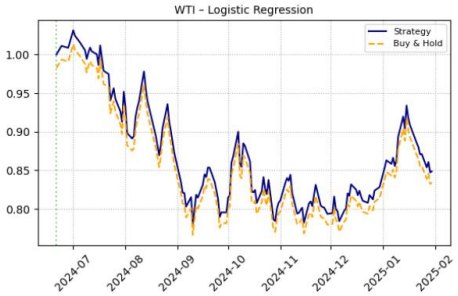

SPY – LSTM
Strategy
Buy & Hold

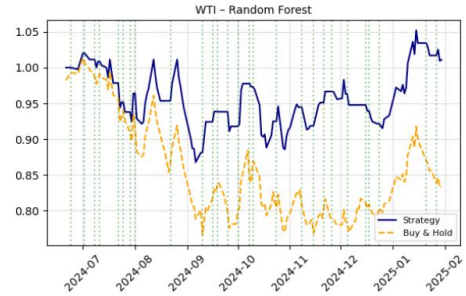

BRENT – Logistic Regression
Strategy
Buy & Hold

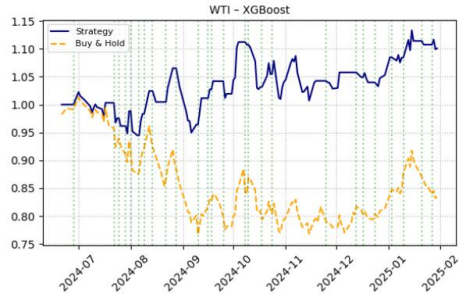

BRENT – Random Forest
Strategy
Buy & Hold

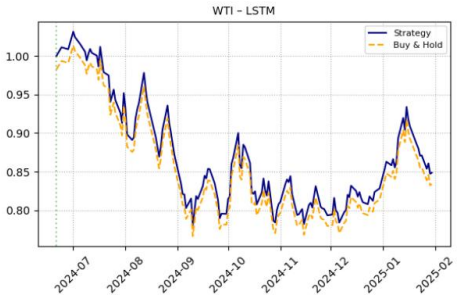

BRENT – XGBoost
Strategy
Buy & Hold


BRENT – LSTM
Strategy
Buy & Hold


WTI – Logistic Regression
Strategy
Buy & Hold


WTI – Random Forest
Strategy
Buy & Hold


WTI – XGBoost
Strategy
Buy & Hold


WTI – LSTM
Strategy
Buy & Hold