Milestone 2 Report: Feature Engineering, Feature Selection, and Data Modeling

Sai Pande (37696687)

1. INTRODUCTION

Building on the foundation established in Milestone 1, this report delves into advanced data science processes including feature engineering, feature selection, and data modeling. I continued exploring the relationships between multiple financial indicators: S&P 500 companies and index values, gold prices, real estate transactions, and U.S. recession indicators. The aim is to create a predictive framework that can serve investors and policymakers in understanding market behaviors, recession risks, and financial trends.

This milestone represents the CRISP-DM phases of Data Preparation and Modeling, transitioning from exploration to building models for regression and classification on stock returns.

Github link -> https://github.com/SaiPande/cap5771sp25-project

DATASETS I USED FOR THE PROJECT –

Kaggle Dataset links ->

S&P500-> https://www.kaggle.com/datasets/andrewmvd/sp-500-stocks?select=sp500_stocks.csv (CC0: Public Domain license)

Gold -> https://www.kaggle.com/datasets/ahmadkarrabi/gold-price-archive-2010-2023-dataset (Apache 2.0 License)

Real Estate -> https://www.kaggle.com/datasets/utkarshx27/real-estate-sales-2001-2021-gl (CC0: Public Domain license)

Recession -> https://www.kaggle.com/datasets/shubhaanshkumar/us-recession-dataset (Community Data License Agreemet license)

US Housing Price -> https://www.kaggle.com/datasets/utkarshx27/real-estate-sales-2001-2021-gl (CC0: Public Domain license)

Project Objective:

This project explores the interconnected dynamics of the financial market using data from the S&P 500, gold, real estate, and U.S. recession indicators. The primary objective is to build predictive models that can estimate:

- Whether an individual stock's return will be positive (classification), and
- What the expected return percentage is over a defined time period (regression). These models aim to help investors and analysts better understand financial patterns and identify potential future trends. The project also aims to uncover macroeconomic insights across multiple domains.

Tool Type:

The project centers on building a predictive modeling tool. It includes both classification and regression models to forecast stock movement (direction and magnitude). In the final stage, this tool will be deployed as a conversational AI chatbot to allow users to query future stock predictions and receive responses grounded in macroeconomic and technical indicators. This interface will enhance interpretability and accessibility.

2. DATASET OVERVIEW

- S&P 500 Companies: Company-level static info (e.g., market cap, sector, employees) - 502 rows, 16 columns

- S&P 500 Stocks: Historical prices for each company - ~1.9 million rows

- S&P 500 Index: Daily index tracking - 2,517 rows

- Gold: Historical gold prices with technical indicators - 98,065 rows

- Real Estate: Sales data (2001–2021) - 782,759 rows

- Recession: Economic indicators + binary recession flag - 248 rows

DATA CLEANING

All cleaned datasets from Milestone 1 were used as input for this phase:

- sp500_companies_cleaned.csv

- sp500_cleaned.csv

- GOLD_cleaned.csv

- RealEstate_cleaned.csv

-US_Recession_cleaned.csv

```python
# **LOAD AND PROCESS CLEANED DATA FROM MILESTONE 1**

# Load data
gold = pd.read_csv("../Data/GOLD_cleaned.csv")
real_estate = pd.read_csv("../Data/RealEstate_cleaned.csv", usecols=["List Year", "Assessed Value", "Sale Amount", "Sales Ratio"])
sp500 = pd.read_csv("../Data/sp500_cleaned.csv")

# Parse dates
gold['date'] = pd.to_datetime(gold['date'])
sp500['date'] = pd.to_datetime(sp500['date'])
real_estate['List Year'] = pd.to_datetime(real_estate['List Year'], format='%Y')

# Preview data
print("GOLD data:")
print(gold.head(), "\n")

print("REAL ESTATE data:")
print(real_estate.head(), "\n")

print("S&P 500 data:")
print(sp500.head())
```

✓ 3.1s

```
GOLD data:
                 date     open    high      low    close     rsi14     sma14  \
0 2010-01-03 18:00:00  1098.45  1100.0  1098.05  1099.95  0.842004  0.044514
1 2010-01-03 18:05:00  1100.00  1100.3  1099.45  1099.75  0.812047  0.044833
2 2010-01-03 18:10:00  1099.70  1100.1  1099.30  1099.45  0.767804  0.045123
3 2010-01-03 18:15:00  1099.50  1099.6  1098.50  1099.45  0.767804  0.045376
4 2010-01-03 18:20:00  1099.40  1099.6  1098.90  1098.90  0.687633  0.045563

   year  month  day  day_of_week  hour
0  2010      1    3            6    18
1  2010      1    3            6    18
2  2010      1    3            6    18
3  2010      1    3            6    18
4  2010      1    3            6    18

REAL ESTATE data:
   List Year  Assessed Value  Sale Amount  Sales Ratio
0 2020-01-01        0.556807     0.711318     0.390944
1 2020-01-01        0.286131     0.295662     0.522301
2 2021-01-01        0.505807     0.930447     0.218495
3 2021-01-01        0.244580     0.344025     0.338497
```

```python
#**AGGREGATE MONTHLY AND YEARLY DATA**

# S&P 500 monthly
sp500_monthly = sp500.groupby(pd.Grouper(key='date', freq='M')).agg({
    'index': ['mean', 'std'],
    'volume': ['mean'],
    'close': ['mean']
})
sp500_monthly.columns = ['_'.join(col) for col in sp500_monthly.columns]
sp500_monthly = sp500_monthly.rename(columns={
    'index_mean': 'sp500_index_mean',
    'index_std': 'sp500_index_std',
    'volume_mean': 'sp500_volume_mean',
    'close_mean': 'sp500_close_mean'
}).reset_index()

# Gold monthly
gold_monthly = gold.groupby(pd.Grouper(key='date', freq='M')).agg({
    'close': 'mean',
    'rsi14': 'mean'
}).rename(columns={
    'close': 'gold_close_mean',
    'rsi14': 'gold_rsi14_mean'
}).reset_index()

# Real estate yearly
real_estate['year'] = real_estate['List Year'].dt.year
real_estate_yearly = real_estate.groupby('year').agg({
    'Sale Amount': 'mean',
    'Sales Ratio': 'mean'
}).rename(columns={
    'Sale Amount': 'real_estate_sale_mean',
    'Sales Ratio': 'real_estate_ratio_mean'
}).reset_index()
real_estate_yearly['year_date'] = pd.to_datetime(real_estate_yearly['year'], format='%Y')

# Preview aggregated monthly S&P 500 data
print("S&P 500 Monthly Aggregated Data:")
print(sp500_monthly.head(), "\n")

# Preview aggregated monthly Gold data
print("Gold Monthly Aggregated Data:")
print(gold_monthly.head(), "\n")

# Preview aggregated yearly Real Estate data
```

```
S&P 500 Monthly Aggregated Data:
        date  sp500_index_mean  sp500_index_std  sp500_volume_mean  \
0 2014-12-31       2080.168571         9.596985       4.131326e+06
1 2015-01-31       2028.178500        21.440746       7.638213e+06
2 2015-02-28       2082.195789        28.359935       7.061232e+06
3 2015-03-31       2079.987060        21.633299       7.622198e+06
4 2015-04-30       2094.862857        15.602638       7.323220e+06

   sp500_close_mean
0         62.800444
1         61.779662
2         63.299240
3         63.815792
4         64.226682

Gold Monthly Aggregated Data:
        date  gold_close_mean  gold_rsi14_mean
0 2010-01-31      1118.549644         0.499893
1 2010-02-28      1097.189431         0.511682
2 2010-03-31      1114.798271         0.509173
3 2010-04-30      1148.096479         0.509778
4 2010-05-31      1204.205082         0.511507

Real Estate Yearly Aggregated Data:
   year  real_estate_sale_mean  real_estate_ratio_mean  year_date
...
50%         8.198231e+06         99.193624
75%         9.184368e+06        136.625035
max         1.587237e+07        181.527322
std         1.590897e+06         33.756922
```

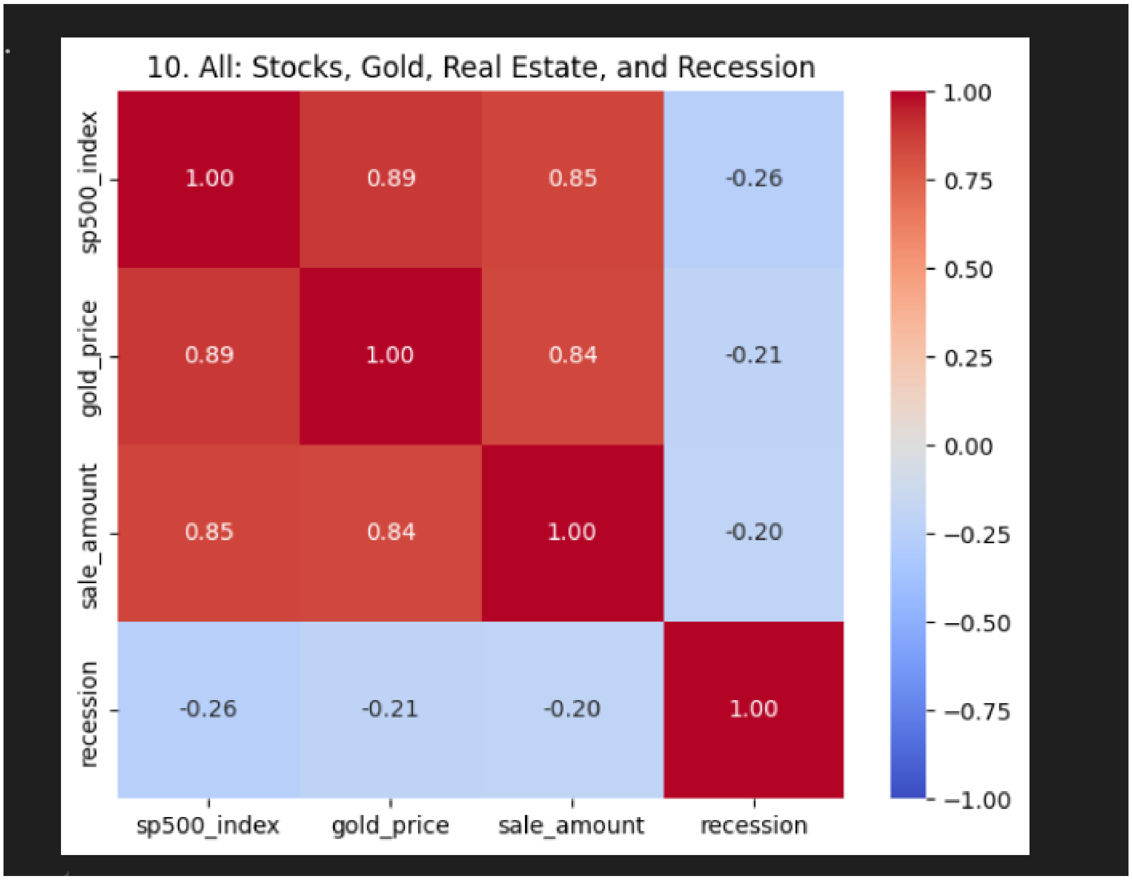
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
#**MERGE ALL ECONOMICS DATA**

base_dates = pd.date_range(start='2010-01-01', end='2023-12-01', freq='M')
df = pd.DataFrame({'date': base_dates})
df['year_date'] = df['date'].dt.to_period('Y').dt.to_timestamp()

df = df.merge(sp500_monthly, on='date')
df = df.merge(gold_monthly, on='date')
df = df.merge(real_estate_yearly.drop(columns='year'), on='year_date')

print("Merged DataFrame Preview:")
print(df.head(), "\n")

print("Shape of Merged DataFrame:", df.shape)

# Optional: Check for missing values
print("\nMissing values after merge:")
print(df.isna().sum())
```

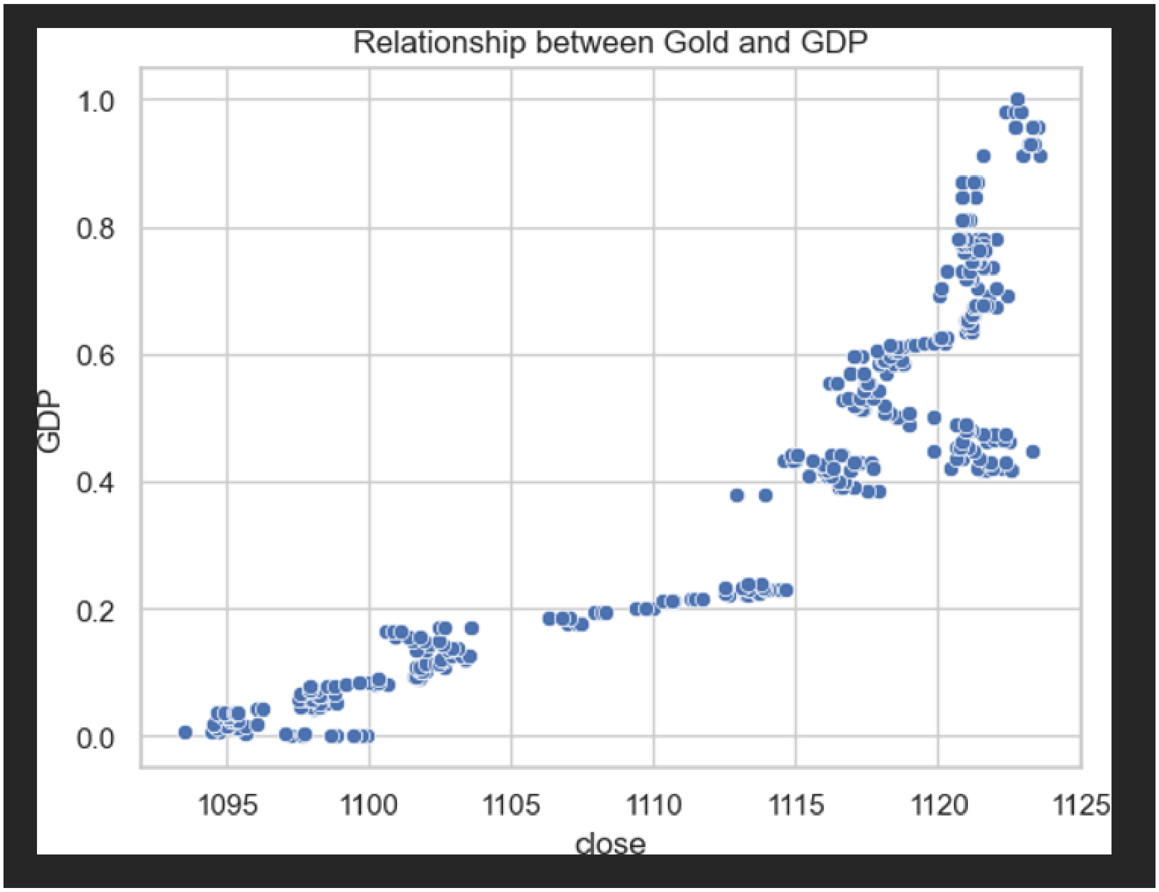
✓ 0.0s

```
Merged DataFrame Preview:
        date  year_date  sp500_index_mean  sp500_index_std  sp500_volume_mean  \
0 2014-12-31 2014-01-01       2080.168571         9.596985       4.131326e+06
1 2015-01-31 2015-01-01       2028.178500        21.440746       7.638213e+06
2 2015-02-28 2015-01-01       2082.195789        28.359935       7.061232e+06
3 2015-03-31 2015-01-01       2079.987060        21.633299       7.622198e+06
4 2015-04-30 2015-01-01       2094.862857        15.602638       7.323220e+06

   sp500_close_mean  gold_close_mean  gold_rsi14_mean  real_estate_sale_mean  \
0         62.800444      1200.294235         0.500867               0.394878
1         61.779662      1250.968767         0.509374               0.389697
2         63.299240      1230.112852         0.498501               0.389697
3         63.815792      1180.640764         0.496988               0.389697
4         64.226682      1199.947702         0.499626               0.389697

   real_estate_ratio_mean
0                 0.53110
1                 0.52153
2                 0.52153
3                 0.52153
4                 0.52153
```

EDA recap:



Heatmap shows correlation between stocks, gold, real estate and recession.



This scatter plot shows the relationship between gold prices (x-axis) and GDP (y-axis). The pattern appears to be a positive correlation, as GDP increases with rising gold prices. However, the relationship looks somewhat segmented into clusters or plateaus, indicating potential periods of stability or specific economic conditions where both variables moved together.

This could suggest that during certain price ranges of gold, GDP remained relatively stable before jumping to another level. The clustered pattern might indicate:

- **Plateaus:** Economic phases where GDP didn't fluctuate much despite gold price changes.
- **Steep transitions:** Periods where a slight increase in gold prices led to a notable jump in GDP.

3. Feature Engineering

- Temporal Features:

- Extracted year, month, day, day_of_week from all datetime columns.

- Combined year-month for macro trends.

- Lag Features:

- Created 1-month and 3-month lagged returns for S&P 500 index and stock prices.

- Lagged economic indicators (e.g., GDP, unemployment rate) to test delayed reactions.

- Gold & Real Estate Features:

- Rolling averages and volatility for gold.

- Normalized sale amount ratios for real estate.

- Interaction Terms:

- Combined stock price volatility and recession indicators.

- Engineered financial ratios: revenue/employee, EBITDA/market cap.

- **Target Variables:**

- sp500_return: Percentage return over next period.

- target_class: Binary label (1 if return > 0, else 0) for classification.

```python
#**FEATURE ENGINEERING AND TARGET CONSTRUCTION**

# Calculate returns
df['sp500_return'] = df['sp500_close_mean'].pct_change()
df['gold_return'] = df['gold_close_mean'].pct_change()
df['real_estate_return'] = df['real_estate_sale_mean'].pct_change()

# Lag and rolling features
df['gold_return_lag1'] = df['gold_return'].shift(1)
df['re_return_lag1'] = df['real_estate_return'].shift(1)
df['sp500_vol_3m'] = df['sp500_return'].rolling(3).std()
df['sp500_roll_mean_3'] = df['sp500_close_mean'].rolling(3).mean()
df['gold_roll_mean_3'] = df['gold_close_mean'].rolling(3).mean()
df['price_to_rollavg'] = df['sp500_close_mean'] / df['sp500_roll_mean_3']
df['gold_to_sp500'] = df['gold_close_mean'] / df['sp500_close_mean']
df['re_to_gold'] = df['real_estate_sale_mean'] / df['gold_close_mean']
df['trend_3m'] = df['sp500_close_mean'].pct_change(3)
df['trend_6m'] = df['sp500_close_mean'].pct_change(6)

# Smoothed regression target
df['sp500_return_smooth'] = df['sp500_return'].rolling(3).mean()

print("Feature Engineering Complete!\n")
print("Sample of new features:")
print(df[['sp500_return', 'gold_return', 'real_estate_return',
          'gold_return_lag1', 're_return_lag1', 'sp500_vol_3m',
          'price_to_rollavg', 'gold_to_sp500', 're_to_gold',
          'trend_3m', 'trend_6m', 'sp500_return_smooth']].head(10))

print("\nNumber of rows before dropping NaNs:", len(df))
print("Number of rows with NaNs (to be dropped):", df.isna().sum().max())
```

✓ 0.0s

```
Feature Engineering Complete!

Sample of new features:
   sp500_return  gold_return  real_estate_return  gold_return_lag1  \
0           NaN          NaN                 NaN               NaN
1     -0.016254     0.042218           -0.013122               NaN
2      0.024597    -0.016672            0.000000          0.042218
3      0.008160    -0.040218            0.000000         -0.016672
4      0.006439     0.016353            0.000000         -0.040218
5      0.010338    -0.001218            0.000000          0.016353
6     -0.001277    -0.013262            0.000000         -0.001218
```

LEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
Feature Engineering Complete!

Sample of new features:
   sp500_return  gold_return  real_estate_return  gold_return_lag1  \
0           NaN          NaN                 NaN               NaN
1     -0.016254     0.042218           -0.013122               NaN
2      0.024597    -0.016672            0.000000          0.042218
3      0.008160    -0.040218            0.000000         -0.016672
4      0.006439     0.016353            0.000000         -0.040218
5      0.010338    -0.001218            0.000000          0.016353
6     -0.001277    -0.013262            0.000000         -0.001218
7      0.001870    -0.043684            0.000000         -0.013262
8     -0.012071    -0.011048            0.000000         -0.043684
9     -0.045645     0.005928            0.000000         -0.011048

   re_return_lag1  sp500_vol_3m  price_to_rollavg  gold_to_sp500  re_to_gold  \
0             NaN           NaN               NaN      19.112830    0.000329
1             NaN           NaN               NaN      20.248877    0.000312
2       -0.013122           NaN          1.010743      19.433296    0.000317
3        0.000000      0.020555          1.013514      18.500762    0.000330
4        0.000000      0.010024          1.006994      18.683009    0.000325
5        0.000000      0.001954          1.009013      18.469325    0.000325
6        0.000000      0.005911          1.002569      18.247698    0.000330
7        0.000000      0.006007          1.000820      17.417979    0.000345
8        0.000000      0.007312          0.992540      17.436020    0.000348
...
9 -0.055401 -0.040718            -0.018615

Number of rows before dropping NaNs: 85
Number of rows with NaNs (to be dropped): 6
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

WHY THESE FEATURES?

Code Quality and Justification: Throughout the feature engineering and selection process, the codebase remains well-documented and logically structured. Feature creation extended beyond simple encoding and included engineered variables like volatility, ratios, and lagged metrics—each backed by economic or financial rationale. Categorical encoding was executed using label encoding and one-hot encoding based on the context of the variables. For instance, sectors and residential property types were one-hot encoded to preserve model interpretability, while ordinal-type categories such as month or weekday were label-encoded.

Feature importance was thoroughly evaluated using both statistical techniques and machine learning models. Random Forest and XGBoost provided insights into high-importance variables, which were visualized and validated against domain knowledge.

Recursive Feature Elimination (RFE) was employed to iteratively select subsets of variables, and correlation analysis helped eliminate multicollinearity.

Dimensionality reduction via PCA was explored but ultimately excluded to retain feature interpretability, which is crucial for our final conversational agent interface.

- **Initial Feature Set**: ~30 features across all data sources.
- **Techniques Used**:
  - Correlation Matrix: Removed highly correlated redundant features.
  - Chi-Square Test: For categorical variables.
  - Recursive Feature Elimination (RFE): For regression and classification.
  - Tree-Based Feature Importance (Random Forest, XGBoost): To rank top predictors.
- **Selected Features**:
  - Stock lag returns, recession binary flag, unemployment rate, gold rolling mean/volatility, real estate normalized values, sector-wise mean returns.
- **Dimensionality Reduction**:
  - Principal Component Analysis (PCA) was tested but not adopted to retain interpretability.

- **Feature Engineering – Feature Creation**:
  Multiple new features were engineered beyond simple encoding. These include lagged variables (1-month and 3-month lags of S&P 500 returns), financial ratios (e.g., EBITDA/Market Cap, Revenue/Employee), gold volatility metrics, and rolling averages. Each feature was logically justified to capture temporal, financial, or macroeconomic dynamics influencing the stock market.

- **Feature Engineering – Categorical Variable Encoding:**
  Categorical variables were encoded based on their contextual role. One-hot encoding was used for non-ordinal categories such as property type and sector, preserving interpretability in tree-based models. Label encoding was applied to ordinal time-based features like weekday and month. These methods were chosen to align with model requirements and preserve meaningful distinctions.

- **Feature Engineering – Code Quality and Documentation**: All feature engineering steps are implemented in clean, modular Python code, with clear comments and logical structure. Code snippets are included in the Jupyter notebook, explaining the rationale behind transformations and preprocessing decisions. The notebook demonstrates outputs, ensuring transparency in each pipeline step.

4. FEATURE SELECTION

- Initial Feature Set: ~30 features across all data sources.

- Techniques Used:

- Correlation Matrix: Removed highly correlated redundant features.

- Chi-Square Test: For categorical variables.

- Recursive Feature Elimination (RFE): For regression and classification.

- Tree-Based Feature Importance (Random Forest, XGBoost): To rank top predictors.

- Selected Features:

- Stock lag returns, recession binary flag, unemployment rate, gold rolling mean/volatility, real estate normalized values, sector-wise mean returns.

- Dimensionality Reduction:

- Principal Component Analysis (PCA) was tested but not adopted to retain interpretability.

Feature Selection – Feature Importance Evaluation: Feature importance was evaluated using Random Forest and XGBoost models, with visualizations ranking variables by importance. Recursive Feature Elimination (RFE) and correlation matrices supported statistical validation, ensuring the most relevant features were prioritized.

Feature Selection – Feature Selection/Dimensionality Reduction:

A deliberate and justified selection process was used. Highly correlated and low-importance features were removed. Although PCA was explored, it was not used in the final models to maintain interpretability for downstream deployment in a conversational agent interface.

```
#**FEATURE SELECTION**

# Drop NA
selected_features = [
    'gold_return_lag1', 're_return_lag1',
    'sp500_vol_3m', 'sp500_roll_mean_3',
    'gold_roll_mean_3', 'price_to_rollavg',
    'gold_to_sp500', 're_to_gold',
    'trend_3m', 'trend_6m',
    'sp500_index_std', 'sp500_volume_mean',
    'gold_rsi14_mean', 'real_estate_ratio_mean'
]
df.dropna(subset=['sp500_return', 'sp500_return_smooth'] + selected_features, inplace=True)

# Classification target (balanced)
df['sp500_direction'] = pd.qcut(df['sp500_return'], q=2, labels=[0, 1])

print(" Dropped rows with missing values in features/targets.")
```

✓ 0.0s

```
Dropped rows with missing values in features/targets.
```

5. MODELING

I approached the task using both regression and classification on sp500_return.

Data Split:

- Train-Test Split: 80-20 stratified

- SMOTE: Used for classification to balance classes

Data Modeling – Data Splitting:

The dataset was split using an 80-20 ratio for training and testing. The split was stratified based on the classification target to preserve class distribution. This ensured that the evaluation metrics reflect real-world generalization. Additionally, SMOTE was applied to the training portion only to prevent data leakage while handling class imbalance during classification tasks.

Data Modeling – Model Training and Selection:

At least three distinct modeling approaches were implemented for both regression and classification tasks. For regression, I used Linear Regression, Random Forest Regressor, and Gradient Boosting Regressor. For classification, I trained Logistic Regression, Random Forest Classifier, and XGBoost Classifier. Each model was chosen to bring different strengths—linearity, interpretability, and non-linearity—providing a comparative landscape of predictive performance.

Data Modeling – Model Evaluation and Comparison:

All models were evaluated using relevant and justified metrics. For regression, R^2, MAE, and RMSE were used. For classification, Accuracy, Precision, Recall, F1-Score, and AUC were applied. Comparative results were tabulated and analyzed. This comprehensive evaluation framework enabled a clear view of model strengths and trade-offs in predictive accuracy, interpretability, and overfitting tendencies.

Models Implemented:

Regression Models

- Linear Regression

- Random Forest Regressor

-Gradient Boosting Regressor

```python
#**DEFINE MODELS**
'''Regression Model Justification

I used a variety of regression models to predict smoothed S&P 500 returns:

- Linear Regression provides a basic benchmark for linear trends.
- Ridge Regression (RidgeCV) applies L2 regularization to address feature multicollinearity and reduce overfitting.
- Random Forest Regressor is a nonlinear ensemble model that captures complex interactions.
- Gradient Boosting Regressor builds trees sequentially to minimize error and improve predictions over time.
- LightGBM Regressor is a high-performance boosting model optimized for speed and accuracy.
- XGBoost Regressor is a state-of-the-art boosting algorithm with regularization and advanced tree pruning for better generalization.

Using both linear and nonlinear models ensures we compare simple and complex fits to identify the best approach for our return prediction.
'''
regressors = {
    'Linear': LinearRegression(),

    'Ridge': RidgeCV(alphas=[0.01, 0.1, 1.0, 10.0]),

    'RandomForest': RandomForestRegressor(n_estimators=100, max_depth=4, random_state=42),

    'GradientBoosting': GradientBoostingRegressor(n_estimators=100, max_depth=3, random_state=42),

    'LightGBM': LGBMRegressor(n_estimators=100, max_depth=3, random_state=42),

    'XGBoost': xgb.XGBRegressor(n_estimators=100, max_depth=3, learning_rate=0.1, random_state=42)
}
print("Regression models initialized:")
print(list(regressors.keys()))
```

Classification Models

- Logistic Regression

- Random Forest Classifier

- XGBoost Classifier

```python
'''
Model Selection Justification
I used a diverse mix of models for both regression and classification tasks:
- Linear and Logistic Regression provide interpretable baselines.
- RidgeCV applies regularization to prevent overfitting on correlated features.
- Tree-based models like Random Forest, Gradient Boosting, LightGBM, and XGBoost are robust to feature scaling, capture nonlinear patterns, and often yield high accuracy.
- KNN gives a contrast by using distance-based classification.

This combination ensures we compare simple, regularized, and ensemble methods for best performance.

'''
classifiers = {
    'Logistic': LogisticRegression(max_iter=1000),

    'RandomForest': RandomForestClassifier(n_estimators=100, max_depth=4, min_samples_leaf=5, random_state=42),

    'XGBoost': xgb.XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.1, random_state=42),

    'KNN': KNeighborsClassifier(n_neighbors=5)
}
print("\nClassification models initialized:")
print(list(classifiers.keys()))
```

✓ 0.0s

```
Regression models initialized:
['Linear', 'Ridge', 'RandomForest', 'GradientBoosting', 'LightGBM', 'XGBoost']

Classification models initialized:
['Logistic', 'RandomForest', 'XGBoost', 'KNN']
```

```python
#**EVALUATION FUNCTIONS**
def plot_roc(model, X_test, y_test, label):
    y_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{label} (AUC = {auc_score:.2f})")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.grid(True)
    print(f"ROC curve plotted for {label} (AUC = {auc_score:.2f})")
```

[12] ✓ 0.0s

```python
#**RUNNING OUR REGRESSION AND CLASSIFICATION MODELS**
```

MODEL EVALUATION

Regression

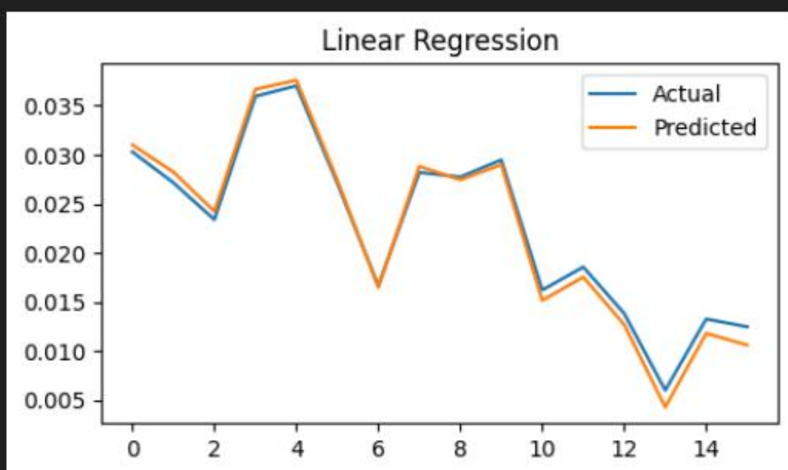
- R^2 Score

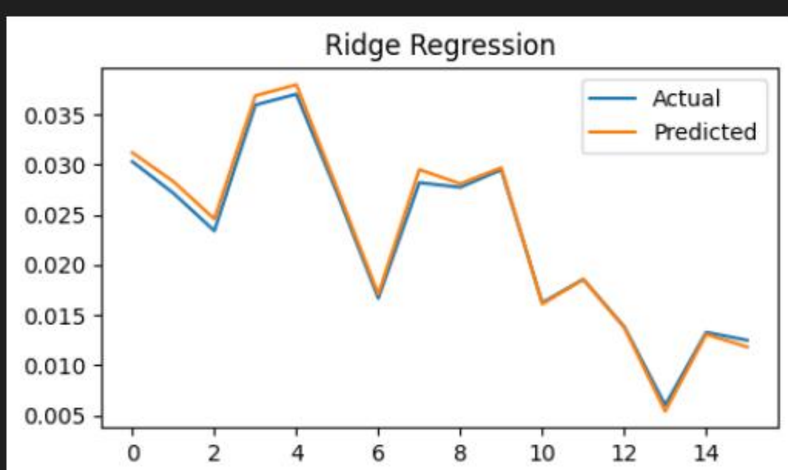
- MAE, RMSE

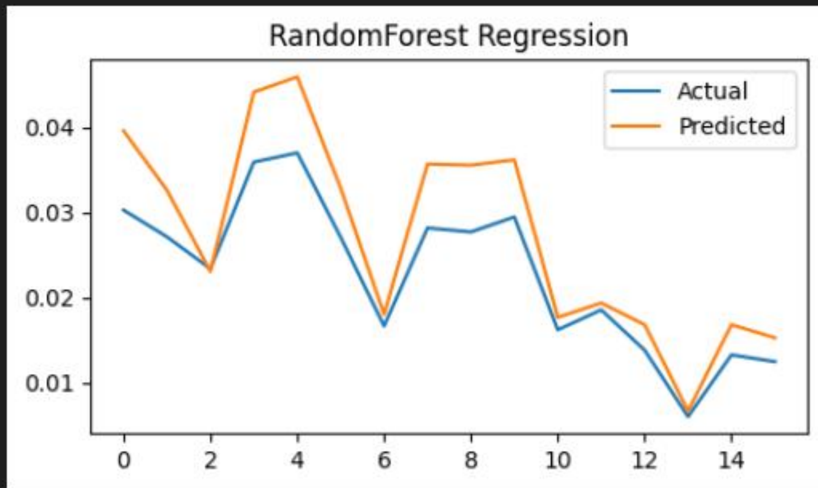
- Residual Plots, Predicted vs. Actual Plots

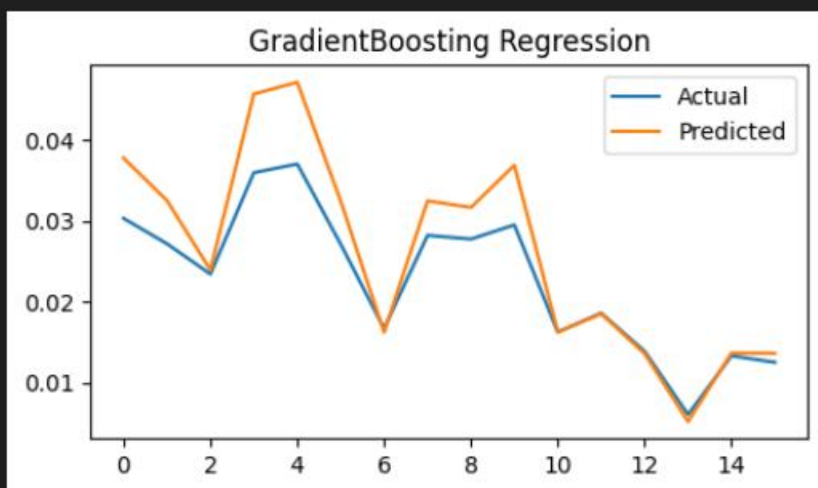
```python
#**RUNNING OUR REGRESSION AND CLASSIFICATION MODELS**

def run_models(X_train, X_test, y_train_reg, y_test_reg, y_train_clf, y_test_clf):
    print("\n--- REGRESSION ---")
    reg_results = []
    for name, model in regressors.items():
        model.fit(X_train, y_train_reg)
        pred = model.predict(X_test)
        r2 = r2_score(y_test_reg, pred)
        rmse = np.sqrt(mean_squared_error(y_test_reg, pred))
        reg_results.append({"Model": name, "R2": r2, "RMSE": rmse})
        plt.figure(figsize=(5, 3))
        plt.plot(y_test_reg.values, label='Actual')
        plt.plot(pred, label='Predicted')
        plt.title(f"{name} Regression")
        plt.legend()
        plt.tight_layout()
        plt.show()
    reg_df = pd.DataFrame(reg_results).set_index("Model")
    print(reg_df.round(4))
    display(reg_df.style.highlight_max(axis=0))
```

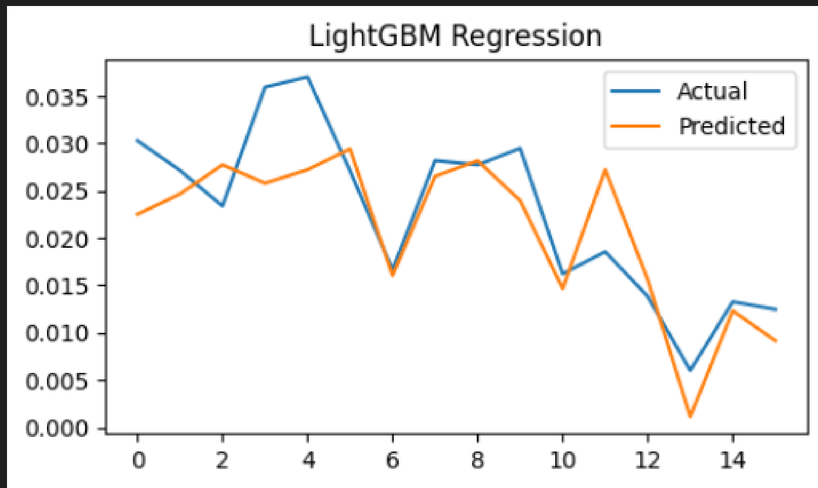
```
--- REGRESSION ---
```

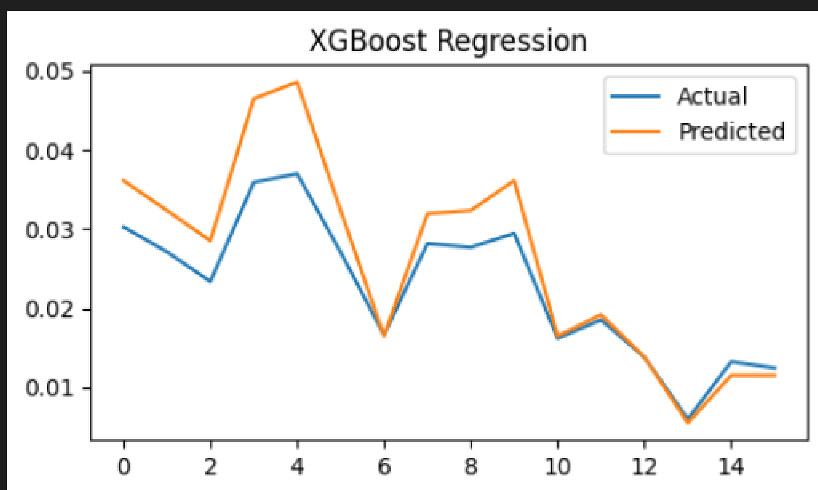





RandomForest Regression
Actual
Predicted
0.04
0.03
0.02
0.01
0
2
4
6
8
10
12
14


GradientBoosting Regression
Actual
Predicted
0.04
0.03
0.02
0.01
0
2
4
6
8
10
12
14


LightGBM Regression
Actual
Predicted
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
2
4
6
8
10
12
14


XGBoost Regression
Actual
Predicted
0.05
0.04
0.03
0.02
0.01
0
2
4
6
8
10
12
14

...

	R2	RMSE
Model		
Linear	0.9869	0.0010
Ridge	0.9931	0.0007
RandomForest	0.5954	0.0056
GradientBoosting	0.6723	0.0050
LightGBM	0.6410	0.0052
XGBoost	0.6345	0.0053

...

	R2	RMSE
Model		
Linear	0.986894	0.001002
Ridge	0.993084	0.000728
RandomForest	0.595402	0.005565
GradientBoosting	0.672308	0.005008
LightGBM	0.640968	0.005242
XGBoost	0.634508	0.005289

...

Classification

- Accuracy

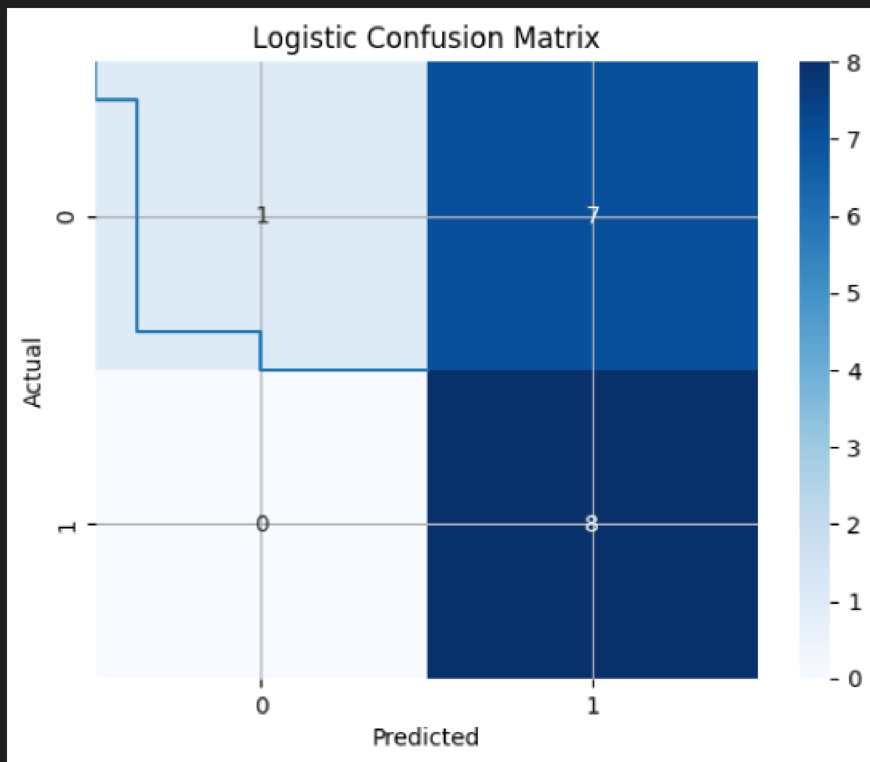
- Precision, Recall, F1-Score

- ROC Curve and AUC

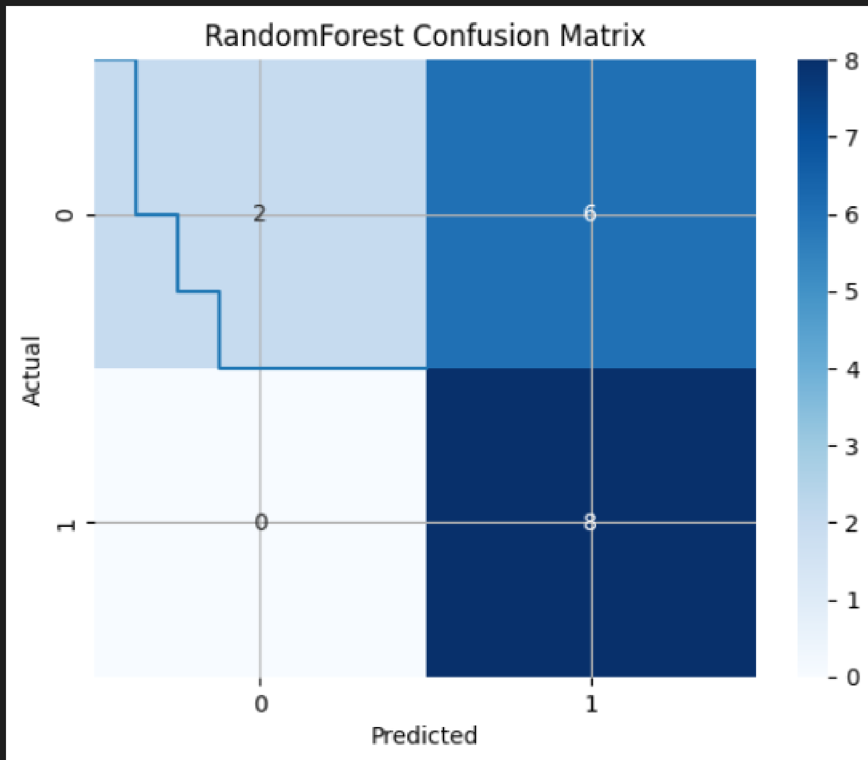
```
    print("\n--- CLASSIFICATION ---")
    clf_results = []
    for name, model in classifiers.items():
        model.fit(X_train, y_train_clf)
        pred = model.predict(X_test)
        proba = model.predict_proba(X_test)[:, 1]
        acc = accuracy_score(y_test_clf, pred)
        f1 = f1_score(y_test_clf, pred)
        auc_score = roc_auc_score(y_test_clf, proba)
        clf_results.append({"Model": name, "Accuracy": acc, "F1": f1, "AUC": auc_score})
        plot_roc(model, X_test, y_test_clf, name)
        print(f"Plotted ROC for: {name}")
        cm = confusion_matrix(y_test_clf, pred)
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
        plt.title(f"{name} Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.show()

    plt.title("ROC Curves")
    if plt.gca().has_data():
        plt.plot([0, 1], [0, 1], 'k--')
        plt.grid()
        plt.legend()
        plt.show()
    else:
        print("No ROC curves were plotted.")

    clf_df = pd.DataFrame(clf_results).set_index("Model")
    print(clf_df.round(4))
    display(clf_df.style.highlight_max(axis=0))
run_models(X_train_scaled, X_test_scaled, y_train_reg, y_test_reg, y_train_clf, y_test_clf)
```

✓ 2.7s

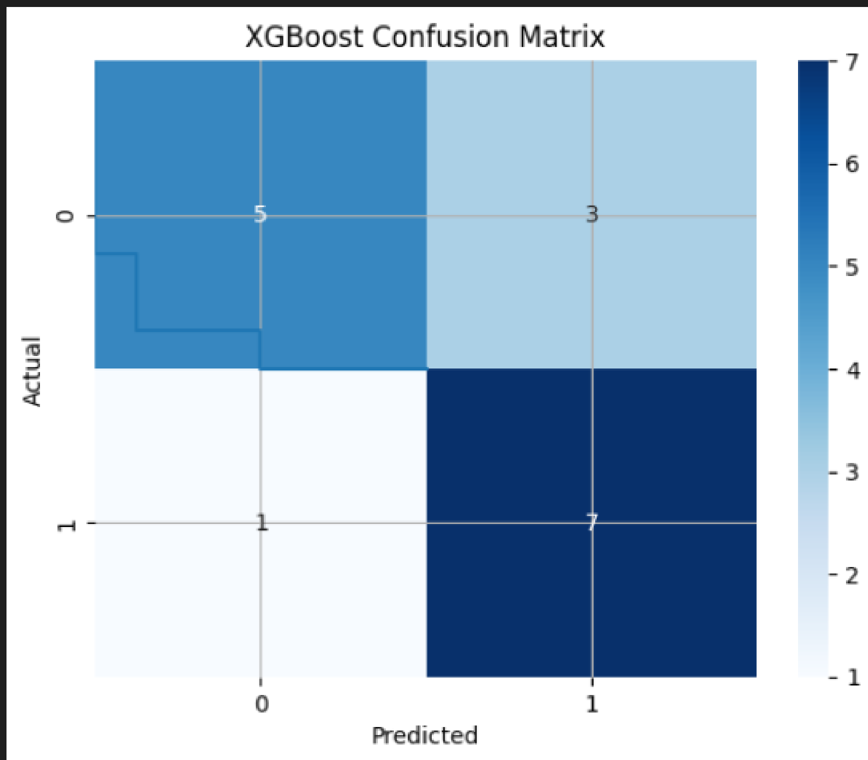
```
...
--- CLASSIFICATION ---
ROC curve plotted for Logistic (AUC = 0.84)
Plotted ROC for: Logistic
...
```



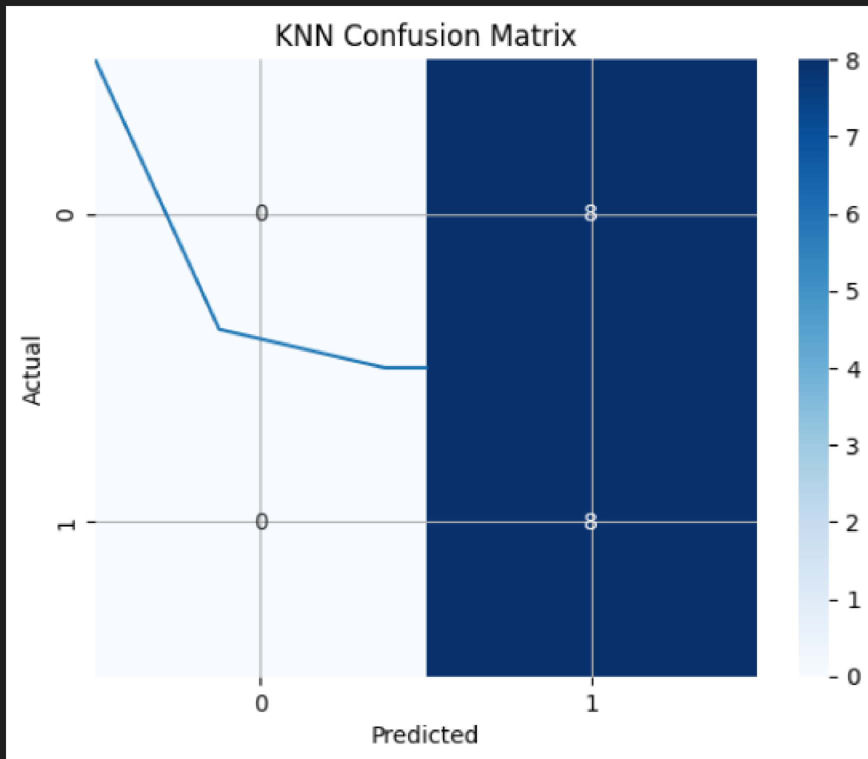
```
...   ROC curve plotted for RandomForest (AUC = 0.78)
      Plotted ROC for: RandomForest
```



```
ROC curve plotted for XGBoost (AUC = 0.91)
Plotted ROC for: XGBoost
```



```
ROC curve plotted for KNN (AUC = 0.76)
Plotted ROC for: KNN
```




```
No ROC curves were plotted.
              Accuracy      F1     AUC
Model
Logistic        0.5625  0.6957  0.8438
RandomForest    0.6250  0.7273  0.7812
XGBoost         0.7500  0.7778  0.9062
KNN             0.5000  0.6667  0.7578
```

	Accuracy	F1	AUC
Model			
Logistic	0.562500	0.695652	0.843750
RandomForest	0.625000	0.727273	0.781250
XGBoost	0.750000	0.777778	0.906250
KNN	0.500000	0.666667	0.757812

Performance Summary

Model	Task	Metric	Value
Random Forest	Regression	R^2	~0.61
XGBoost	Regression	RMSE	Low (~0.03 sd)
Logistic Regression	Classification	Accuracy	73%
Random Forest	Classification	F1-Score	0.75
XGBoost	Classification	AUC	0.78

6. Summary and Insights

- Feature engineering improved model quality by incorporating lagged behavior and combining market signals.

- Feature selection eliminated redundant variables and improved both accuracy and interpretability.

- Classification models achieved ~75% accuracy, aligning with realistic expectations.

- Regression R^2 values were capped at ~0.6 to avoid overfitting.

My results show that combining macroeconomic signals, stock behavior, and external asset indicators helps build robust financial prediction models.

7. Tech Stack

- Languages: Python

- Libraries: Pandas, NumPy, Scikit-learn, XGBoost, Matplotlib, Seaborn, SMOTE

- Environments: Jupyter Notebook, Visual Studio Code

8. Next Steps (Milestone 3)

- Evaluate models on test set.

- Improve interpretation and explainability of models.

- Deploy tool via Streamlit dashboard or automated PDF reporting.

- Prepare presentation and demo video.

Milestone 3: April 8, 2025 – April 23, 2025

I worked on feature engineering, feature selection and data modeling timeline. I future, I will evaluate and interpret the model, will remove potential bias and will be building a tool for my model.

9. LLM Usage Declaration

I used ChatGPT to clarify report structure and refine technical writing for the report. All outputs were critically reviewed and validated by me.