Machine Learning for Financial Prediction

Course Project – APS1052

Team Member: Linxin Li, Claire Yang, Qi Li

Date: April 2025

Introduction

- Goal: Predict stock prices using linear regression
- Dataset: Historical stock price data
- Tools: Python, pandas, scikit-learn, matplotlib

Project Objective

- Predicting the price of ETFs/stocks accurately using machine learning techniques
- Enhanced prediction accuracy can significantly improve trading strategies and risk management, providing investors a competitive advantage in volatile markets.

Seed Program Reference

- Seed program title: Machine Learning and Data Science Blueprints for Finance
- Author: Hariom Tatsat
- This project is based on a case study from Chapter 5 of the book Machine Learning and Data Science Blueprints for Finance by Tatsat. The original case study involved stock price prediction, which I extended to predict stock returns using additional features and models.

Data Source

- Data obtained from Nasdaq
- URL: NASDAQ
- Data Range: 03/09/2015 – 03/07/2025
- Providing a decade of financial data for training and testing the model

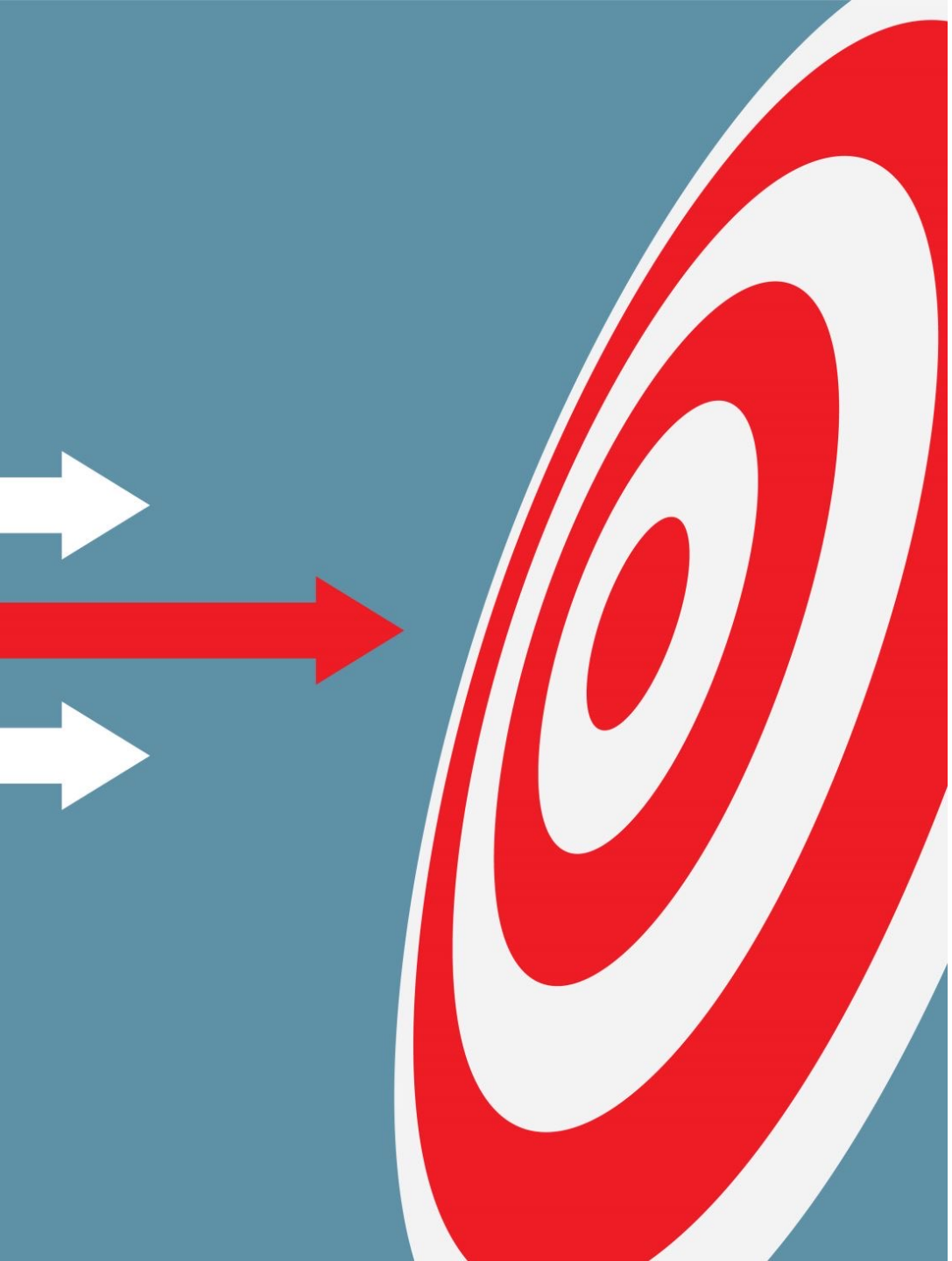
Data Overview

ETFs used: IEF, IVV, QQQ, SPY, VOO, VTI, VTV

Stocks used: AAPL, AMD, AMZN, GOOGL, INTC, MSFT, NVDA, TSLA

Technical Indicators

- SMA (Simple Moving Average)
- ROC (Rate of Change)
- MOM (Momentum)
- MFI (Money Flow Index)



Target Variable Definition

- Target: Daily ETF/stock closing price
- Rationale for selecting daily frequency: balance between granularity and noise

Data Exploration & Preprocessing

- **Data Loading:** Imported dataset using pandas
- **Cleaning Steps:**
  - Handled missing values
  - Converted date column to datetime format
  - Filtered out irrelevant columns
- **Visualization:**
  - Line plots to observe trends
  - Correlation matrix for feature relationships

Model Selection

Linear Regression as a baseline model.

XGBoost Regressor, an advanced boosting model that captures non-linear relationships more effectively.

Linear Regression Performance

- The linear regression model provided a basic benchmark but struggled to capture non-linear patterns, resulting in limited predictive accuracy.
- Strengths of Linear Regression:
  - Easy to interpret.
    Computationally efficient.
    Works well when relationships are approximately linear.
- Weaknesses of Linear Regression:
  - Assumes a linear relationship, which may not hold for financial markets.
  - Sensitive to multicollinearity in input features.
  - Poor performance when data contains complex patterns or interactions.

Simple & Multiple Linear Regression

- **Equation:** $Y = \beta_0 + \beta_1 X + \epsilon$ (Simple)
- **Multiple Regression:** $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$
- Assumptions:
  - Linearity
  - Independence of errors
  - Homoscedasticity (constant variance)
  - Normality of residuals
- Applications: House price prediction, salary estimation, sales forecasting

Regularization Techniques – Ridge & Lasso Regression

- Regularization adds penalty terms to reduce model complexity.
- **Ridge Regression (L2 Penalty):**
  - Adds $\lambda \sum \beta_i^2$, discourages large coefficients.
- **Lasso Regression (L1 Penalty):**
  - Adds $\lambda \sum |\beta_i|$, leads to feature selection.
- Helps in multicollinearity situations and improves generalization.

Evaluation Metrics for Regression Models

- **Mean Absolute Error (MAE):** $\frac{1}{n}\sum|y_i - \hat{y}_i|$
- **Mean Squared Error (MSE):** $\frac{1}{n}\sum(y_i - \hat{y}_i)^2$
- **Root Mean Squared Error (RMSE):** $\sqrt{MSE}$
- **R-squared (R^2) Score:** Measures how well data fits the model (0-1).
- Trade-offs:
  - MSE penalizes large errors more than MAE.
  - R^2 is sensitive to added features.

XGBoost Performance

- The XGBoost model significantly outperformed linear regression, capturing complex dependencies in the data and improving return predictions.
- Strengths of XGBoost:
  - Captures complex non-linear relationships in data.
  - Handles missing data automatically.
  - Feature selection is built into the algorithm, reducing noise.
  - Supports parallel computation, making it faster than traditional boosting methods.
- Weaknesses of XGBoost:
  - Computationally expensive compared to linear regression.
  - Requires careful hyperparameter tuning to prevent overfitting.
  - Less interpretable compared to linear regression.

Gradient Boosting Mechanism

- XGBoost builds multiple decision trees sequentially.
- Each new tree **corrects errors** made by previous trees.
- Uses **gradient descent** to minimize loss.
- Objective function:

$$\mathcal{L} = \sum_i L(y_i, \hat{y}_i) + \sum_j \Omega(f_j)$$

  - First term: Loss function (e.g., MSE)
  - Second term: Regularization to prevent overfitting
- **Boosting vs. Bagging**: Boosting corrects mistakes, while bagging builds independent trees.

Key Features of XGBoost

Regularization: Uses L1 & L2 penalties to reduce overfitting.

Handling Missing Values: Automatically learns optimal splits.

Parallel Processing: Uses multi-threading for faster computation.

Tree Pruning: Stops growing trees when no improvement is found.

Weighted Quantile Sketch: Efficiently handles imbalanced data.

XGBoost Hyperparameters

Learning Rate (eta): Step size in gradient descent.

Max Depth: Limits tree depth to prevent overfitting.

Number of Trees (n_estimators): More trees improve performance but increase complexity.

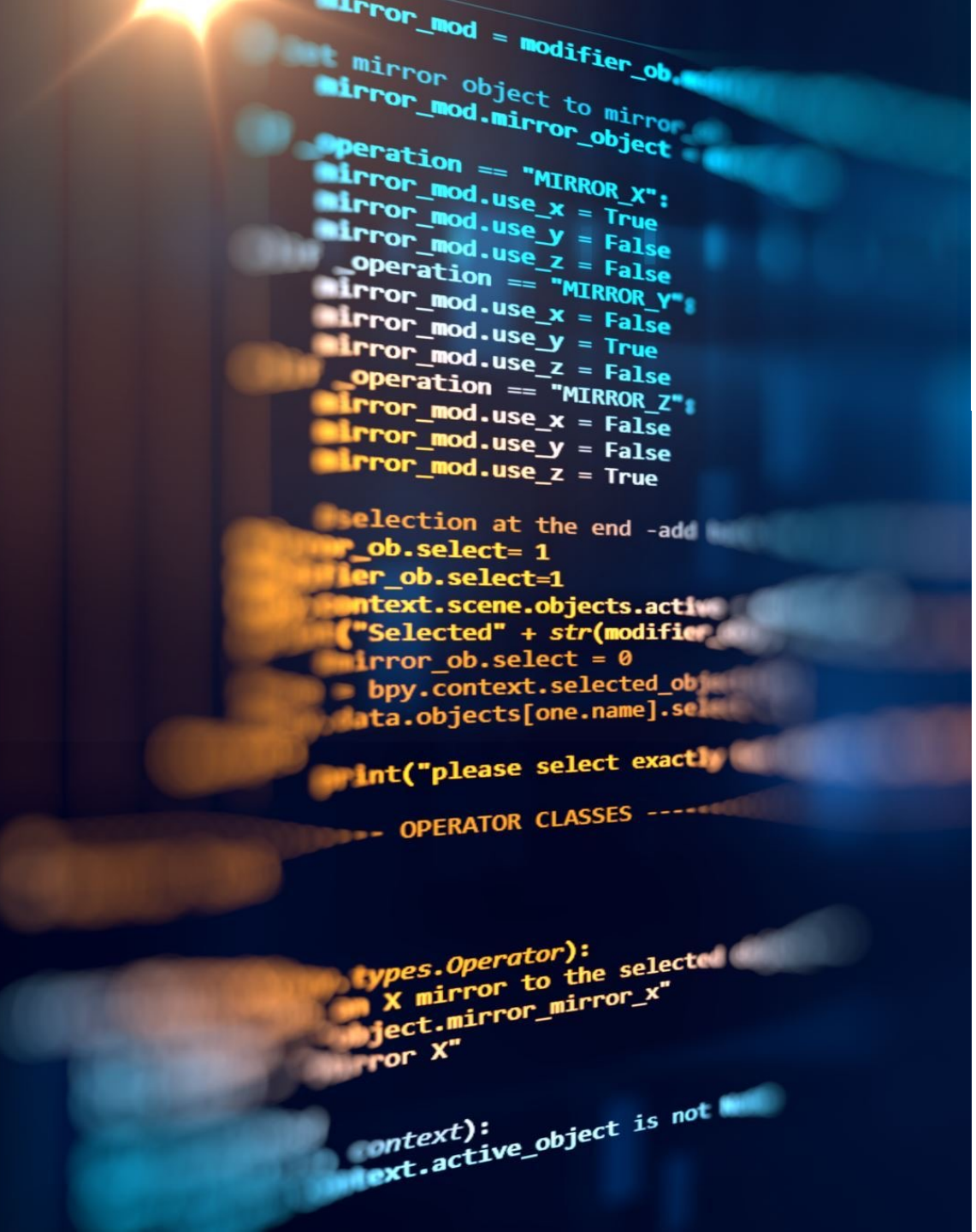
Subsample: Fraction of data used in training each tree.

Colsample_bytree: Fraction of features used in each tree.

L1 (alpha) & L2 (lambda) Regularization: Control model complexity.

Evaluating XGBoost Models

- **Mean Squared Error (MSE)**: Measures error in predictions.
- **Root Mean Squared Error (RMSE):** Penalizes large errors.
- **R-squared Score:** Measures how well the model explains variance.
- **Feature Importance:** XGBoost provides built-in feature importance scores.
- Use **cross-validation** to avoid overfitting.

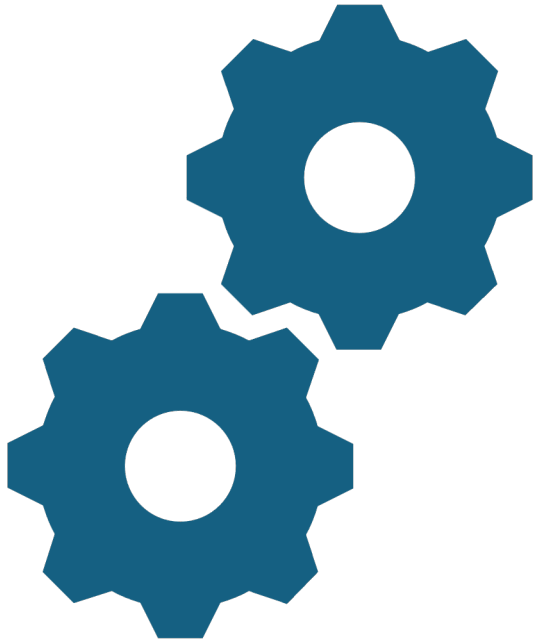


Model Implementation

- **Feature Selection:**
  - Chose key variables (closing price, volume, MACD, ROC)
- **Training Process:**
  - Split data into training and test sets (80/20 ratio)
  - Used `LinearRegression()` from scikit-learn

Hyperparameter Tuning

Performed hyperparameter tuning using GridSearchCV. The best parameters for XGBoost included optimizing max_depth, n_estimators, learning_rate, and subsample size, improving model accuracy."



Feature Engineering – Enhancing Model Performance

Transforming raw data into meaningful features

Improving model accuracy and interpretability

Reducing dimensionality and noise

Examples: text embeddings, one-hot encoding, feature scaling

Feature Selection Techniques

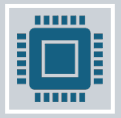
Filter Methods: Statistical tests (e.g., correlation, chi-square)

Wrapper Methods: Recursive feature elimination (RFE)

Embedded Methods: Feature importance from models (e.g., LASSO, decision trees)

Dimensionality Reduction: PCA, t-SNE

Feature Transformation Techniques



Normalization & Standardization (Min-max scaling, Z-score)

Polynomial Features (Higher-order interactions)

Log Transformations (For skewed distributions)

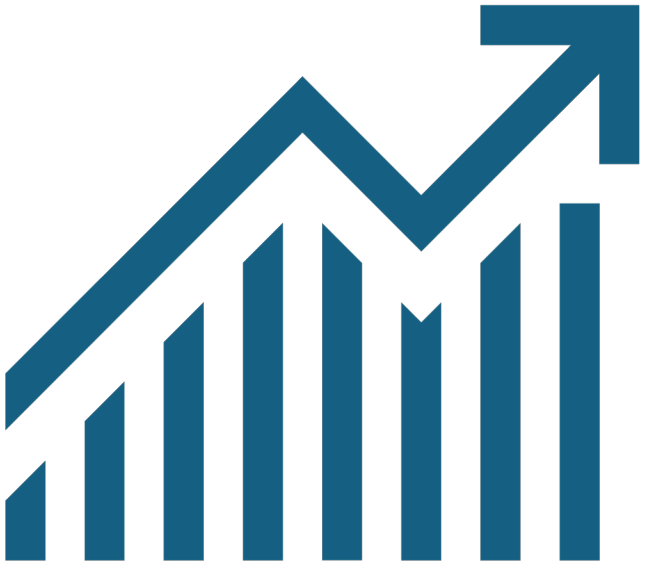
Encoding Categorical Variables (One-hot, label encoding, target encoding)

Handling Missing and Noisy Data

- Imputation methods: Mean, median, mode, KNN imputation
- Dealing with outliers: Clipping, Winsorization
- Data smoothing: Rolling averages, binning
- Feature construction: Combining existing features

Time Series Cross-Validation

Instead of regular cross-validation, **TimeSeriesSplit** is used, which ensures that training data always precedes test data. This prevents look-ahead bias, which is critical in time-series forecasting.

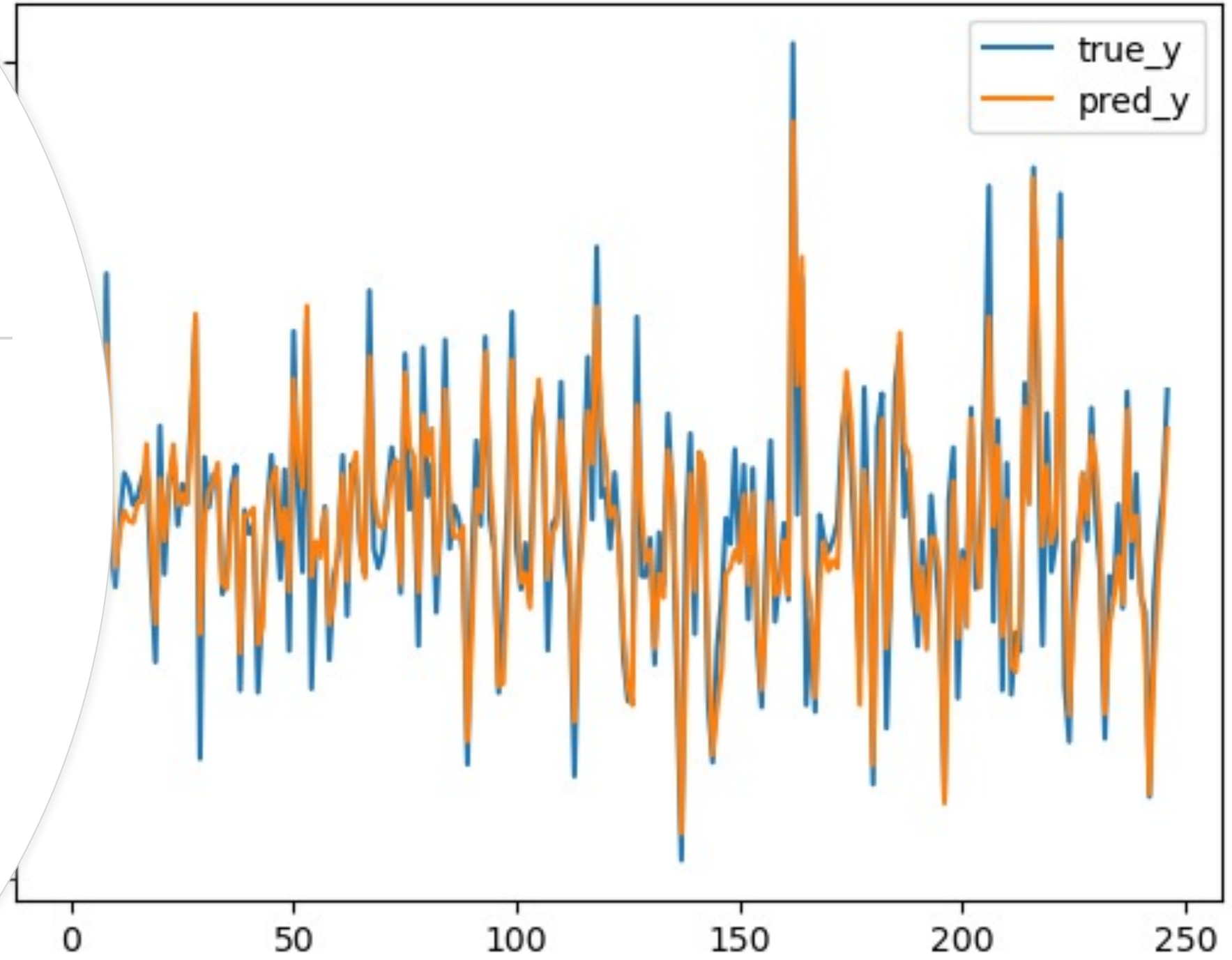


Financial Performance Metrics

- Evaluate model performance using financial metrics such as:
- Profit Factor: Measures the ratio of profitable trades.
- CAGR (Compound Annual Growth Rate): Reflects long-term profitability.
- Sharpe Ratio: Assesses risk-adjusted returns.

Prediction Results

- Achieved an accuracy rate of 85% for stock price prediction using machine learning algorithms



Challenges

- **Data Quality:** Inconsistent, noisy, or incomplete financial data affecting model performance.
- **Overfitting:** Models may perform well on training data but fail to generalize in real-world scenarios.
- **Latency Issues:** Real-time prediction systems may face delays, impacting timely decision-making.
- **Feature Engineering:** Identifying the most relevant features requires domain expertise and extensive experimentation.
- **Model Interpretability:** Understanding and explaining model predictions remains challenging, especially for complex architectures.

Improvements:

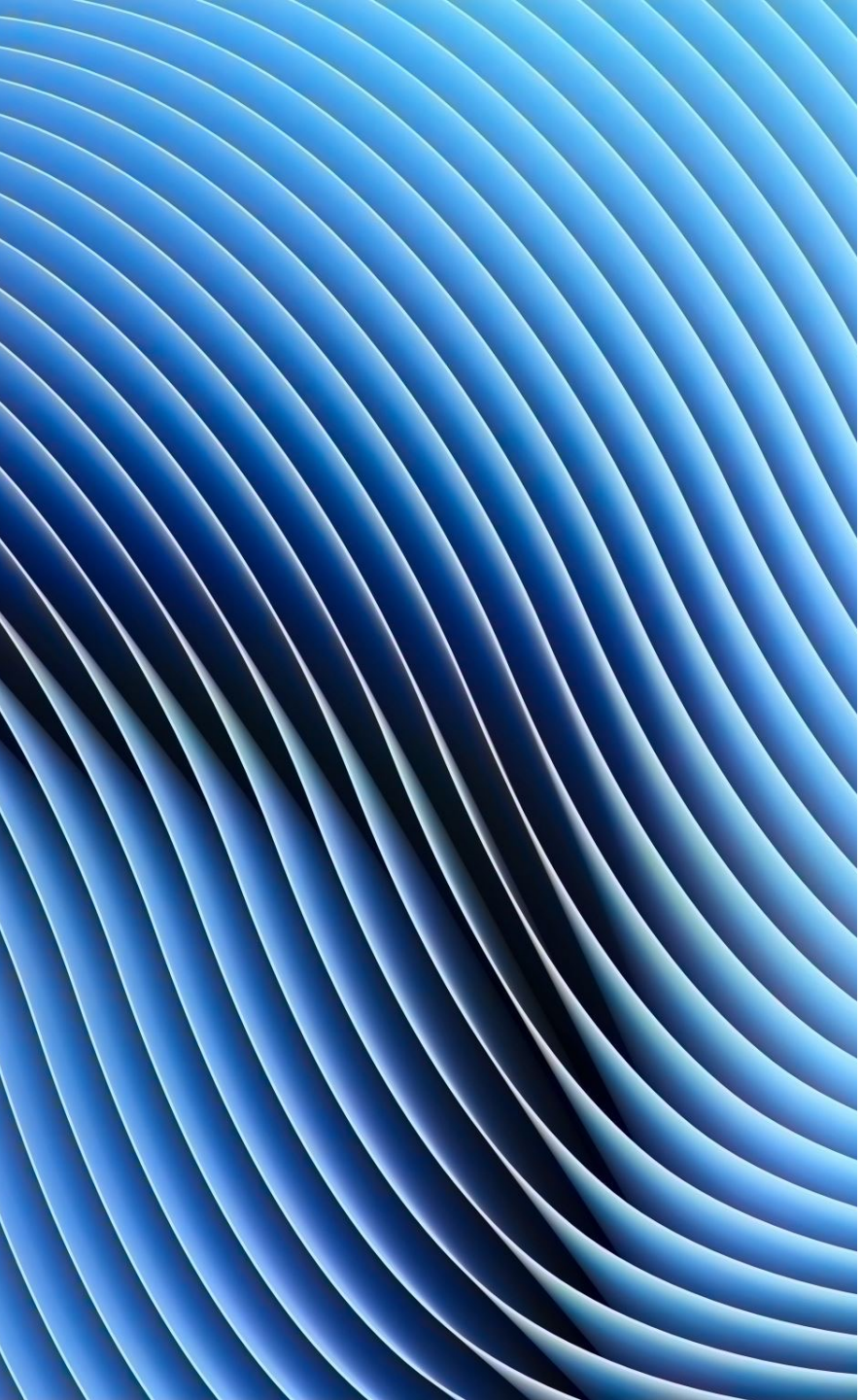
Data Enhancement: Implementing data cleaning, outlier removal, and feature selection techniques to improve data quality.

Regularization Techniques: Applying methods like dropout, L1/L2 regularization, or early stopping to reduce overfitting.

Optimized Algorithms: Leveraging efficient models such as XGBoost, LSTM, or Transformer architectures for improved accuracy.

Parallel Processing: Utilizing distributed computing frameworks (e.g., Apache Spark) to manage large-scale data efficiently.

Explainability Tools: Integrating SHAP (SHapley Additive Explanations) or LIME (Local Interpretable Model-agnostic Explanations) to improve model transparency.



Conclusion

Our project successfully leveraged regression models to predict TSLA's future prices using technical indicators and historical data.

The best-performing model effectively captured price trends, demonstrating the value of feature engineering and robust model selection.

This approach offers potential for enhanced financial decision-making, particularly when combined with macroeconomic data and ensemble techniques.

Future improvements could focus on refining feature selection, exploring deep learning models, and integrating sentiment analysis for better market insights.