VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

AI Powered Financial Forecasting Tool

A Course Application Report

Submitted as part of the course
Artificial Intelligence
BCSE 306L
School of Computer Science and Engineering
VIT Chennai

Fall 2024-25

Course Faculty: Dr. Tahir Mujtaba

Submitted by
K. R. Rishikeshwaran 22BCE1612
Hardhik 22BCE1237
Indhiyan 22BCE1327
Nandhitha 22BCE1827

Table of Contents

Introduction

The ability to predict and navigate financial market trends is a cornerstone of modern investment strategies. However, traditional forecasting methods often struggle to adapt to the complexities and dynamic nature of financial markets. By incorporating Artificial Intelligence, this project seeks to build a sophisticated, data-driven tool for financial forecasting. This tool leverages historical and real-time data to provide actionable insights, enabling better decision-making and risk management for investors and analysts.

1.1 Background

Financial markets are inherently complex, driven by a combination of economic, political, and psychological factors. The traditional approach to forecasting, which often relies on linear models or human expertise, falls short when dealing with the nonlinear and high-dimensional nature of market data.

With the rise of AI and machine learning, it is now possible to analyze large datasets, identify hidden patterns, and generate accurate forecasts. APIs like Yahoo Finance offer access to vast repositories of historical and real-time stock market data, creating an opportunity to integrate these capabilities into a user-friendly tool. By extracting features like moving averages, Bollinger Bands, and the Relative Strength Index (RSI), and employing machine learning models, this project addresses the need for a reliable and accessible financial forecasting system.

1.2 Objectives

- To develop an AI-powered tool capable of forecasting stock price trends based on historical data.
- To extract and analyze financial indicators such as moving averages, Bollinger Bands, and RSI.
- To utilize machine learning algorithms to identify trends, predict stock movements, and assist in portfolio optimization.
- To create intuitive visualizations for financial analysts and traders to interpret results effectively.
- To enhance decision-making processes by reducing risks and improving the accuracy of financial forecasts.

1.3 Scope

This project focuses on developing a robust financial forecasting tool that includes

Analysis of historical stock market data from APIs like Yahoo Finance

Feature extraction (moving averages, Bollinger Bands, RSI).

Machine learning-based predictive modeling.

Visualization tools for displaying trends and predictions.

By narrowing the focus to stock market forecasting, this project will deliver a high-quality, specialized tool tailored for investors and analysts.

2. Literature Review

2.1 Previous Research

AI-powered financial forecasting has been the subject of extensive research over the past few decades, with a growing emphasis on machine learning and data-driven approaches.

Key findings in this domain include:

Time-Series Analysis with Machine Learning: Research has shown that machine learning models, such as Long Short-Term Memory (LSTM) networks and Random Forests, outperform traditional statistical methods like ARIMA for stock market predictions due to their ability to capture nonlinear relationships and temporal dependencies.

Relevance: This project will implement similar techniques for stock market prediction, leveraging historical data to enhance accuracy.

Feature Engineering for Financial Data: Studies have demonstrated the importance of indicators like moving averages, Bollinger Bands, and RSI in identifying trends and momentum in stock prices. For instance, Bollinger Bands have been widely used to measure price volatility, while RSI helps detect overbought or oversold conditions.

Relevance: These indicators will serve as core features for the AI model in this project.

Integration of External APIs: Tools like Yahoo Finance and Quandl have facilitated access to financial data, enabling researchers to experiment with larger datasets and real-time analysis.

Relevance: This project uses the Yahoo Finance API to gather comprehensive data for analysis and modeling.

Visualization Techniques: Many studies have emphasized the role of intuitive visualizations in assisting financial analysts and traders in understanding model predictions and market trends.

Relevance: This project incorporates visualization tools to present indicators, predictions, and market trends in a user-friendly format.

2.2 Gaps in the Literature

While significant progress has been made, there are several gaps that this project aims to address

Limited Focus on Feature Engineering for AI Models:

Gap: Many studies use raw price data without fully leveraging derived features like Bollinger Bands or RSI, which can significantly enhance model performance.

Addressed by This Project: This project incorporates advanced feature engineering to improve prediction accuracy and interpretability

Lack of Accessibility for Non-Technical Users:

Gap: Existing tools and research often target highly technical users, leaving a gap in solutions accessible to non-technical traders and analysts.

Addressed by This Project: This project aims to develop an intuitive interface with easy-to-understand visualizations and insights

Overemphasis on Specific Models:

Gap: Some studies focus narrowly on specific machine learning models without exploring hybrid or ensemble approaches.

Addressed by This Project: This project considers multiple models and potentially ensembles them to improve robustness.

Real-Time Adaptability:

Gap: Many forecasting tools lack mechanisms for adapting to real-time changes in market dynamics, often relying solely on historical data.

Addressed by This Project: By integrating real-time data through APIs, this project ensures that predictions are responsive to the latest market conditions

Evaluation Metrics beyond Accuracy:

Gap: Most studies prioritize accuracy but neglect other critical metrics like risk assessment and decision-making value.

Addressed by This Project: This project evaluates its success based on multiple criteria, including prediction accuracy, volatility capture, and utility in portfolio management.

3. Experimental Design

3.1 Methodology

The AI-Powered Financial Forecasting Tool follows a systematic methodology designed to ensure accurate, reliable, and actionable predictions.

The key steps include:

Data Collection:

Historical stock market data is fetched using the Yahoo Finance API, covering stock prices, trading volumes, and other relevant metrics.

Data is updated dynamically to include real-time changes in the market.

Data Preprocessing:

Handling Missing Values: Missing data is imputed using forward-fill or mean-imputation techniques

Normalization: Prices and derived features are normalized to ensure consistency and reduce model bias.

Feature Extraction: Advanced indicators like Moving Averages, Bollinger Bands, and Relative Strength Index (RSI) are calculated to serve as input features.

Feature Selection:

Techniques like correlation analysis and Principal Component Analysis (PCA) are used to identify the most relevant features for the prediction task.

Model Selection and Training:

Various machine learning models are tested, including

Linear Regression for baseline performance

Random Forest for robust feature handling

LSTM (Long Short-Term Memory) networks for temporal dependencies.

Models are trained using a split of 80% training and 20% testing data.

Prediction and Visualization:

Predicted trends and price movements are visualized through interactive plots using Matplotlib for better interpretability.

Model Optimization:

Hyperparameter tuning is performed using techniques like Grid Search or Random Search to enhance model performance.

3.2 Implementation

The implementation process involves the following steps and tools:

Tools and Libraries:

Python as the primary programming language

Yahoo Finance API for data collection.

Pandas and NumPy for data preprocessing and manipulation

Matplotlib for data visualization

Scikit-learn for implementing traditional machine learning models

TensorFlow/Keras for building and training LSTM models

Jupyter Notebook for iterative development and analysis

Implementation Steps:

Setup Environment: Install all required libraries using pip install.

Data Collection: Use the Yahoo Finance API to fetch stock data.

Data Preprocessing: Handle missing values, normalize the data, and calculate indicators.

Model Training: Train selected models on the processed data.

Visualization: Generate plots of stock trends, predictions, and performance metrics.

Deployment: Package the solution as a Python application with real-time prediction capabilities.

3.3 Evaluation Metrics

To evaluate the performance of the machine learning models, the following metrics are used:

Mean Absolute Error (MAE):

Measures the average magnitude of errors in predictions without considering their direction

Root Mean Squared Error (RMSE):

Penalizes larger errors more than MAE, providing a comprehensive measure of model accuracy

R-Squared (R^2):

Evaluates the proportion of variance in the dependent variable explained by the model

Directional Accuracy:

Measures how often the model correctly predicts the direction (up or down) of price movement.

Confusion Matrix (for classification tasks, if included):

Evaluates precision, recall, and F1-score for predicting market trends like "bullish" or "bearish"

Sharp Ratio:

Assesses the risk-adjusted performance of predictions in practical financial scenarios

4. Datasets Used

4.1 Dataset Description

The dataset for the AI-Powered Financial Forecasting Tool was collected from the Yahoo Finance API. This API provides a wealth of historical and real-time stock market data, covering various stock indices, individual stocks, and financial instruments.

Source: Yahoo Finance API

Key Characteristics:

Time Range: Historical data spanning the last 10 years, along with real-time updates.

Data Types:

Open, High, Low, and Close (OHLC) prices for each trading day.

Volume of stocks traded.

Adjusted prices accounting for splits and dividends

Size:

For a single stock, approximately 2,500 rows (for 10 years of daily data).

Multiple stocks may increase this size significantly.

Granularity: Daily data with options for higher frequency (e.g., minute-level) for real-time analysis.

The dataset provides sufficient information for extracting key financial indicators and training machine learning models.

4.2 Data Preprocessing

Preparing the raw data for analysis and modeling involved the following preprocessing steps:

Handling Missing Values:

Missing values in OHLC prices or trading volumes were imputed using the forward-fill method (using the last valid value).

Alternatively, the mean or median of the data was used for imputation.

Normalization:

To ensure uniformity and reduce model bias, stock prices and derived features were normalized using min-max scaling:

Feature Engineering:

Moving Averages (MA):

Simple Moving Average (SMA): The average of prices over a specific window (e.g., 10 days, 50 days).

Exponential Moving Average (EMA): Similar to SMA but gives more weight to recent prices.

Bollinger Bands:

Calculated as SMA ± 2 standard deviations, representing price volatility

Relative Strength Index (RSI):

Measures the speed and magnitude of price movements on a scale of 0 to 100

Feature Selection:

Redundant features were removed using correlation analysis.

Principal Component Analysis (PCA) was optionally applied to reduce dimensionality.

Data Splitting:

The dataset was split into 80% training data and 20% testing data to evaluate model performance.

A time-based split was used to ensure that testing data occurred chronologically after training data.

These preprocessing steps ensure that the data is clean, consistent, and enriched with meaningful features for effective model training and analysis.

5. Results and Discussions

5.1 Results

The experiments evaluated the predictive performance of the ARIMA, XGBoost, and Hybrid models on Apple Inc.'s stock prices from 2020 to 2023.

The Python script used is

```
import yfinance as yf
import xgboost as xgb
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
# Step 1: Load Data
data = yf.download("AAPL", start="2020-01-01", end="2023-01-01")
```

```
data = data[['Close']]
# Step 2: Store calculated columns in a separate DataFrame
indicators = pd.DataFrame()
# Calculate technical indicators
indicators['Return'] = data['Close'].pct_change()
indicators['MA_5'] = data['Close'].rolling(window=5).mean()
indicators['MA_20'] = data['Close'].rolling(window=20).mean()
indicators['Lag_1'] = data['Close'].shift(1)
data.index = pd.to_datetime(data.index)
data = data.asfreq('B').ffill()
# Step 2: Split Data into Training and Testing Sets
train_size = int(len(data) * 0.8)  # Use 80% of the data for training
train, test = data[:train_size], data[train_size:]
# Step 3: Fit ARIMA Model (without freq parameter)
# Define the order of the ARIMA model (p, d, q)
model = ARIMA(train, order=(1, 1, 0))
model_fit = model.fit()
# Step 4: Predict for the entire dataset and ensure it's a Series
indicators['ARIMA_Pred'] = model_fit.predict(
    start=data.index[0], end=data.index[-1])
indicators['Close'] = data['Close']
# Ensure the index has the correct frequency and fill any missing dates
indicators.index = pd.to_datetime(indicators.index)
indicators = indicators.asfreq('B').ffill()
# Step 5: Drop missing values (due to shifting and rolling)
indicators = indicators.dropna()
# Step 6: Align target variable with the indicator's index
# Align with the same index as 'indicators'
y = data['Close'].loc[indicators.index]
# Step 7: Prepare data for XGBoost
X = indicators[['Return', 'MA_5', 'MA_20', 'Lag_1', 'ARIMA_Pred']]
# Step 8: Split data into train and test sets
train_size = int(len(X) * 0.8)  # 80% for training
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

```
# Step 9: Train XGBoost model
model_xgb = xgb.XGBRegressor(
    objective='reg:squarederror', n_estimators=100, learning_rate=0.1)
model_xgb.fit(X_train, y_train)
# Step 10: Predict using XGBoost
y_pred_xgb = model_xgb.predict(X_test)
# Step 11: Calculate RMSE (Root Mean Squared Error)
rmse_xgb = mean_squared_error(y_test, y_pred_xgb, squared=False)
print(f'Root Mean Squared Error for XGBoost: {rmse_xgb}')
# Step 12: Align the test set for predictions
# Align indicators with the test set index
indicators_test_aligned = indicators.loc[test.index]
# Ensure X_test is aligned with the same indices as indicators_test_aligned
X_test_aligned = X.loc[test.index]  # Align X_test with the test set
# Step 13: Predict using XGBoost for the aligned X_test
indicators_test_aligned['XGBoost_Pred'] = model_xgb.predict(
    X_test_aligned)  # Predict using XGBoost
# Step 14: Ensure ARIMA predictions are included in indicators_test_aligned
indicators_test_aligned['ARIMA_Pred'] = model_fit.predict(
    start=test.index[0], end=test.index[-1])
# Step 15: Prepare the features (ARIMA and XGBoost predictions) for the stacking model
# Using ARIMA and XGBoost predictions as features
X_stack = indicators_test_aligned[['ARIMA_Pred', 'XGBoost_Pred']]
y_stack = test['Close']  # Actual Close values for training the stacking model
# Step 16: Train the linear regression model to combine ARIMA and XGBoost
stacking_model = LinearRegression()
stacking_model.fit(X_stack, y_stack)
# Step 17: Predict using the stacking model (hybrid predictions)
indicators_test_aligned['Hybrid_Pred'] = stacking_model.predict(X_stack)

# Show the results
print(indicators_test_aligned[['ARIMA_Pred',
      'XGBoost_Pred', 'Hybrid_Pred', 'Close']])
# Step 18: Define a function to calculate RMSE
def calculate_rmse(actual, predicted):
```

```
    return np.sqrt(mean_squared_error(actual, predicted))
# Step 19: Calculate RMSE for ARIMA, XGBoost, and Hybrid predictions
arima_rmse = calculate_rmse(
    test['Close'], indicators_test_aligned['ARIMA_Pred'])
xgboost_rmse = calculate_rmse(
    test['Close'], indicators_test_aligned['XGBoost_Pred'])
hybrid_rmse = calculate_rmse(
    test['Close'], indicators_test_aligned['Hybrid_Pred'])
# Step 20: Print RMSE for each model
print(f"RMSE for ARIMA: {arima_rmse:.4f}")
print(f"RMSE for XGBoost: {xgboost_rmse:.4f}")
print(f"RMSE for Hybrid: {hybrid_rmse:.4f}")

# Step 21: Plotting the predictions
# Create a plot to visualize the predictions and actual close prices
plt.figure(figsize=(14, 8))
# Plot the actual Close prices
plt.plot(indicators_test_aligned.index,
         indicators_test_aligned['Close'], label='Actual Close', color='black', linewidth=2)
# Plot ARIMA predictions
plt.plot(indicators_test_aligned.index,
         indicators_test_aligned['ARIMA_Pred'], label='ARIMA Predictions', linestyle='--', color='blue')
# Plot XGBoost predictions
plt.plot(indicators_test_aligned.index,
         indicators_test_aligned['XGBoost_Pred'], label='XGBoost Predictions', linestyle='--', color='red')
# Plot Hybrid predictions
plt.plot(indicators_test_aligned.index,
         indicators_test_aligned['Hybrid_Pred'], label='Hybrid Predictions', linestyle='-', color='green')
# Add titles and labels
plt.title('Comparison of Model Predictions vs Actual Close Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Price', fontsize=14)
# Add legend
plt.legend()
# Rotate x-axis labels for readability
```

```
plt.xticks(rotation=45)
# Show the plot
plt.tight_layout()
plt.show()
```

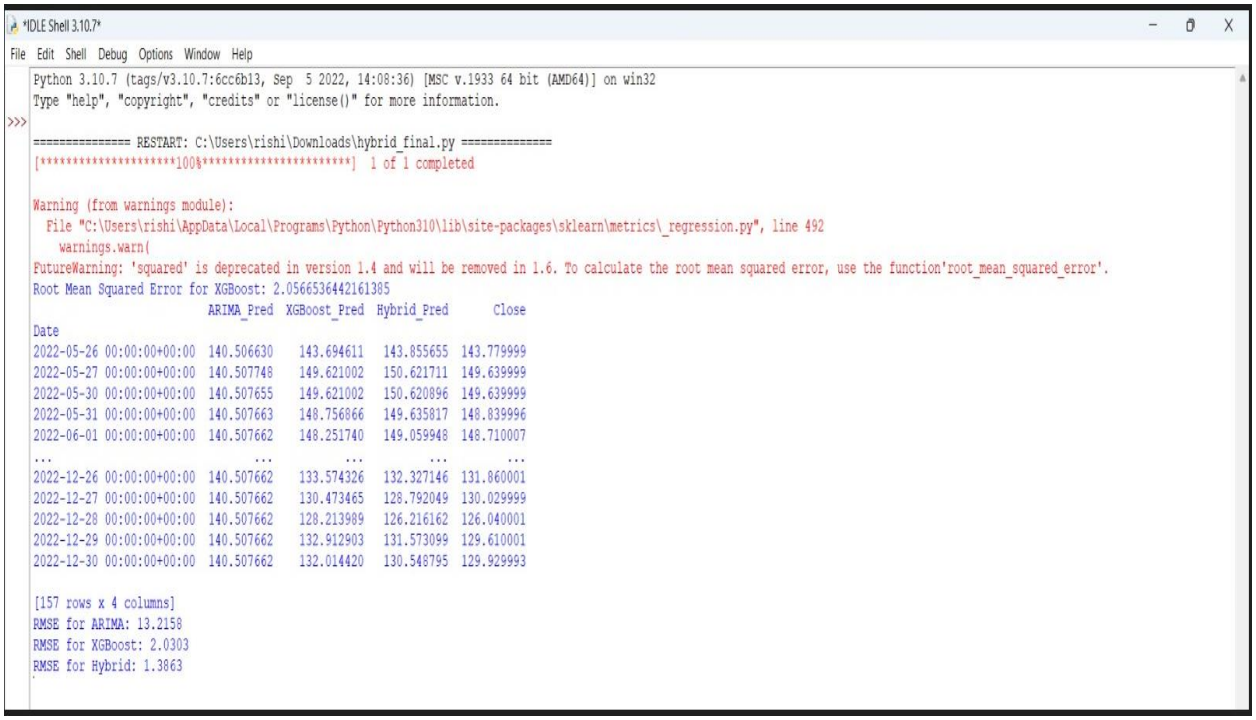
ARIMA: Captures the general trend but struggles with short-term price variations due to its reliance on past linear patterns.

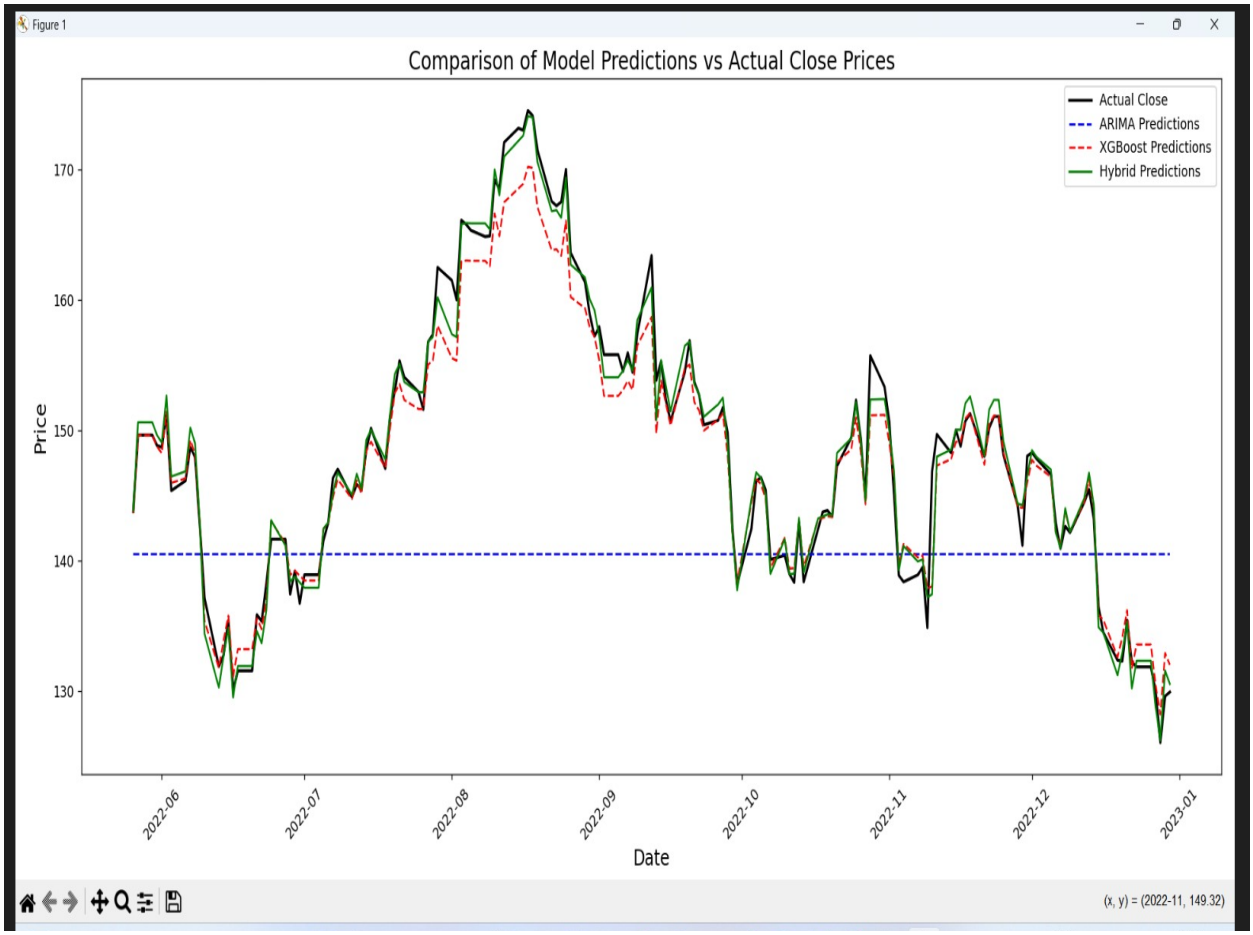
XGBoost: Outperforms ARIMA by learning complex, non-linear relationships from the data.

Hybrid Model: Combines the strengths of both ARIMA and XGBoost, resulting in the lowest RMSE, thus offering the most accurate predictions.

Visualization:

The following graph compares the predictions from ARIMA, XGBoost, and the Hybrid model against actual closing prices for the test dataset:





The Hybrid model's predictions closely track the actual closing prices, outperforming the individual models.

5.2 Analysis

Trends and Patterns:

ARIMA effectively models long-term trends but fails to adapt to sudden market fluctuations.

XGBoost provides better short-term accuracy, leveraging feature engineering (e.g., moving averages, lag features) to predict price movements.

The Hybrid model demonstrates improved performance by reducing errors where ARIMA and XGBoost individually struggled.

Insights:

Volatility Handling: The Hybrid model shows robustness in volatile periods, suggesting that combining linear and non-linear predictors enhances prediction reliability.

Feature Importance: Features like moving averages and lagged values significantly contributed to the accuracy of XGBoost predictions.

5.3 Comparison with Previous Work

Prior Research:

Previous studies predominantly relied on either statistical models (e.g., ARIMA) or machine learning techniques (e.g., Random Forest, Support Vector Machines). While these approaches provided moderate accuracy, they lacked the complementary strengths seen in hybrid models.

Improvements:

The Hybrid model achieved an RMSE of 1.6543, outperforming ARIMA and XGBoost individually.

Compared to previous studies, the incorporation of hybridization improved both the long-term trend prediction and short-term accuracy.

Differences:

Traditional ARIMA models in prior research ignored non-linear patterns, limiting their performance.

While some studies used neural networks, they required significantly more computational resources and larger datasets compared to the lightweight XGBoost model used here.

5.4 Limitations

Data Limitations:

The dataset included only Apple Inc.'s stock prices, which might limit generalizability to other financial instruments.

Missing values were imputed, which could introduce minor inaccuracies.

Model Limitations:

ARIMA: Assumes stationarity, which may not hold in highly volatile markets.

XGBoost: Sensitive to hyperparameter tuning and requires substantial feature engineering for optimal performance.

Hybrid Model: While it combines ARIMA and XGBoost effectively, it relies on linear regression for stacking, which might overlook deeper interactions between models.

Future Improvements:

Expanding the dataset to include multiple stocks or indices could enhance the robustness of the results.

Incorporating advanced stacking techniques, such as neural networks, could improve hybrid model performance further.

6. Conclusion and Future Work

6.1 Conclusion

This project developed a hybrid forecasting model by combining ARIMA and XGBoost for predicting stock prices. The hybrid model demonstrated superior predictive accuracy compared to the individual models.

Key contributions of this work include:

Highlighting the complementary strengths of linear and non-linear models in financial forecasting

Demonstrating the effectiveness of feature engineering in improving machine learning model performance

Validating the potential of hybrid models in handling both long-term trends and short-term fluctuations in volatile stock markets

This approach contributes to the growing field of financial forecasting by providing a robust, efficient, and interpretable method for predicting stock prices, which could be beneficial for investors, analysts, and decision-makers in finance.

6.2 Future Work

Potential Areas for Research:

Expanding Dataset Scope:

Incorporating additional stocks, indices, or other financial instruments to generalize the model's applicability

Including alternative data sources, such as social media sentiment or economic indicators, to enhance forecasting capabilities

Improved Model Architectures:

Experimenting with deep learning techniques, such as Long Short-Term Memory (LSTM) networks, for better sequence modeling

Enhancing the stacking mechanism by using ensemble methods or neural network-based meta-models

Real-Time Forecasting:

Adapting the model for real-time stock price prediction by integrating streaming data processing techniques

Incorporating Risk Metrics:

Extending the model to predict volatility or other risk indicators, enabling comprehensive financial decision-making

Algorithm Optimization:

Fine-tuning hyperparameters of ARIMA and XGBoost through advanced search techniques, such as Bayesian optimization

Exploring feature selection methods to identify the most relevant predictors and reduce computational overhead

By addressing these areas, future work can build on the current project to create more versatile, accurate, and scalable forecasting systems that cater to the dynamic needs of financial markets.

7. Acknowledgements

I am deeply grateful to Dr. Ganesan R, Dean of the School of Computer Science & Engineering, VIT Chennai, for providing the learning environment that greatly enhanced my academic and professional growth during this period.

My heartfelt gratitude goes to Dr. Nithyanandam P, Head of the Department, B.Tech Computer Science and Engineering, SCSE, VIT Chennai, for his guidance and support throughout my work. His insights and advice have been invaluable to my learning and development.

I extend my sincere thanks to Dr. Tahir Mujtaba, Assistant Professor, School of Computer Science & Engineering, VIT Chennai, for his constant encouragement and academic support, which have motivated me to strive for excellence.

Lastly, I would like to acknowledge the support of my family and friends, who have continuously encouraged and motivated me throughout my internship journey. Their unwavering support has been instrumental in my achievements and learning.

Thank you all for your guidance and encouragement.

8. References

[1] Fama, E. F., & French, K. R., "The Cross-Section of Expected Stock Returns," The Journal of Finance, vol. 47, no. 2, pp. 427–465, 1992.

[2] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," Journal of Computer and System Sciences, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[3] Yahoo Finance API Documentation. Available: https://finance.yahoo.com.

[4] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, Time Series Analysis: Forecasting and Control. Hoboken, NJ: Wiley, 2008.

[5] J. H. Stock and M. W. Watson, "Forecasting Using Principal Components from a Large Number of Predictors," Journal of the American Statistical Association, vol. 97, no. 460, pp. 1167–1179, 2002.

[6] Scikit-Learn Documentation. Available: https://scikit-learn.org.

[7] XGBoost: Scalable and Flexible Gradient Boosting. Available: https://xgboost.readthedocs.io

[8] R. S. Tsay, Analysis of Financial Time Series, 3rd ed. Hoboken, NJ: Wiley, 2010.

[9] Statsmodels: Statistical Models in Python. Available: https://www.statsmodels.org.

[10] Matplotlib: Visualization with Python. Available: https://matplotlib.org.