SUMMER TRAINING/INTERNSHIP

PROJECT REPORT

(Term June-July 2025)

Title Of the Project:

Stock Price Prediction and Portfolio Optimization

Submitted by

Name of Student	Registration Number
Himanshu Tomar	12313246
Manaswini Mandal	12309471
Sasmita Biswal	12323395

Course Code: PETV79

Under the Guidance of

Mr. Mahipal Singh Papola

School of Computer Science and Engineering

JULY 14

Lovely Professional University, Punjab

BONAFIDE CERTIFICATE

Certified that this project report titled **"STOCK PRICE PREDICTION AND PORTFOLIO OPTIMIZATION"** is the Bonafide work of **HIMANSHU TOMAR, MANASWINI MANDAL, and SASMITA BISWAL** who carried out the project work under my supervision as part of their academic requirements.

SIGNATURE

Name of Project Supervisor

Mr. Mahipal Singh Papola

HIMANSHU TOMAR
MANASWINI MANDAL
SASMITA BISWAL

SIGNATURES OF STUDENTS

SIGNATURE

HEAD OF THE DEPARTMENT

SIGNATURE

PROJECT SUPERVISOR

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have supported and guided us during the successful completion of our group project titled: "**Stock Price Prediction and Portfolio Optimization.**"

First and foremost, we are thankful to the Almighty for blessing us with the strength, focus, and perseverance to complete this project.

We extend our heartfelt thanks to our respected guide, Mr. Mahipal Singh Papola, for their invaluable support, expert guidance, and continuous encouragement throughout the duration of the project. Their constructive suggestions and deep knowledge have played a pivotal role in shaping the direction of our work.

We also express our gratitude to the faculty and staff of the Lovely Professional University for providing us with the academic foundation and technical resources that made this project possible.

This project has been a valuable learning experience, and we are truly thankful to everyone who contributed to its success.

Submitted By

1. Himanshu Tomar: 12313246
2. Manaswini Mandal: 12309471
3. Sasmita Biswal: 12323395

School of Computer Science and Engineering,
Lovely Professional University, Punjab

TABLE OF CONTENTS

Abstract

1. **Introduction**
   - **1.1 Objective**

2. **Technologies and Libraries**

3. **Intraday Prediction Using Random Forest**
   - **3.1 Introduction to Intraday Forecasting**
   - **3.2 Data Acquisition and Preparation**
   - **3.3 Feature Selection**
   - **3.4 Train-Test Splitting**
   - **3.5 Model Training with Random Forest**
   - **3.6 Predictions and Evaluation**
   - **3.7 Streamlit Deployment and Visualization**
   - **3.8 Advantages and Limitations**

4. **Breakout Strategy with Support/Resistance**
   - **4.1 Introduction to Breakout Strategy**
   - **4.2 Data Acquisition**
   - **4.3 Understanding Support/Resistance in Technical Analysis**
   - **4.4 Automating Support/Resistance Detection**
   - **4.5 Applying Local Extrema for Breakout Detection**
   - **4.6 Signal Generation Logic**
   - **4.7 Visualization and Interpretation**
   - **4.8 Strengths of Breakout Strategy**

5. **Stock Portfolio Optimization**
   - **5.1 Introduction**
   - **5.2 Data Acquisition and Preprocessing**
   - **5.3 Feature Engineering and RSI calculation**
   - **5.4 Predictive Modelling using XGBoost**
   - **5.5 Risk Modelling and Efficient Frontier Optimization**
   - **5.6 Discrete Assets Allocation**
   - **5.7 Visualization**
   - **5.8 Advantages**

6. **Conclusion**

ABSTRACT

This report entails the functionality and outcomes of a Streamlit-based web application designed for stock price prediction and portfolio optimization. This project proposes an integrated approach to stock market analysis and prediction by utilizing three powerful techniques: Breakout Trading Strategy using Support and Resistance levels, Portfolio Optimization with the Efficient Frontier model and allocating assests based on risks and returns percentage, and intraday price forecasting through Random Forest-based machine learning models, utilizing real-time stock data and advanced machine learning models to aid investment decisions. The application has been implemented using Python with the Streamlit framework, providing an interactive interface for dynamic stock market insights. Technical indicators, statistical modelling, and data visualization have been combined to facilitate better decision-making for investors, traders, and financial analysts. The solution enables users to forecast market movements, optimize capital allocation, and strategize intraday trading—all within a unified interface.

1. INTRODUCTION

Stock markets represent one of the most dynamic, complex, and data-intensive financial environments. For decades, analysts and researchers have attempted to model, predict, and understand the intricate behaviours of market instruments. With the democratization of data access and the advent of powerful computational tools, it has become increasingly feasible to develop sophisticated models that were once only the domain of institutional traders. This project is born out of the need for a unified tool that empowers users to gain a better understanding of market conditions through a blend of technical, quantitative, and predictive analysis.

The project integrates three popular approaches used in the field of stock market analysis: breakout strategy identification based on support and resistance, portfolio optimization using the Efficient Frontier, and intraday price forecasting using supervised machine learning models. This unified approach not only enhances flexibility and coverage for users but also provides valuable insights for long-term investors, short-term traders, and quantitative analysts alike. As such, the application reflects a convergence of traditional financial theories with modern artificial intelligence, emphasizing practical usability and academic rigor.

1.1 OBJECTIVE

The overarching goal of this project is to design and implement a unified, multi-strategy stock market prediction and analysis tool. The system is built to serve different user profiles—from casual investors to algorithmic traders—providing distinct modes of prediction and visualization. The specific objectives of the project include:

1. **To identify breakout signals using historical price patterns:** Utilizing local extrema to detect key support and resistance zones and implementing a rule-based system to generate buy, sell, or hold signals based on breakouts.
2. **To forecast optimal asset allocation through portfolio optimization techniques:** This is achieved using the Efficient Frontier model and the mean-variance framework supported by PyPortfolioOpt.

3. **To predict short-term intraday price movements through supervised learning models:** Machine learning algorithms, particularly Random Forest and XGBoost, are employed to capture non-linear patterns in high-frequency trading data.
4. **To create a user-friendly platform for financial visualization and interaction:** This enables users to input parameters, visualize data trends, and interpret model outputs seamlessly.
5. **To build a system that is scalable and modular:** The system should support future inclusion of additional strategies such as sentiment analysis or LSTM-based time series forecasting.

2. TECHNOLOGIES AND LIBRARIES

NumPy

NumPy, or Numerical Python, is one of the most essential foundational libraries for numerical computation in Python. In this project, NumPy is heavily used for handling arrays, performing mathematical operations, and generating efficient vectorized computations required in financial data analysis. NumPy provides the ability to perform complex operations such as rolling calculations for moving averages, standard deviation, and log returns, all of which are vital for understanding market trends and building machine learning features. With its fast and memory-efficient arrays, NumPy serves as the backbone for calculations involving price differences, return rates, and volatility measurements. In portfolio modelling, NumPy enables the representation of covariance matrices and expected returns in matrix form, which are critical to optimizing financial metrics like the Sharpe ratio. The use of NumPy not only accelerates computational processes but also ensures mathematical accuracy and consistency across modules. Its compatibility with other libraries like pandas and scikit-learn further enhances its utility in this multi-faceted system.

Pandas

Pandas is a powerful data manipulation and analysis library tailored for structured data. In this project, it is primarily used to handle time-series stock data obtained from Yahoo Finance. Through pandas DataFrames, historical prices including Open, High, Low, Close, and Volume are organized and processed for each ticker. It facilitates data cleaning, forward/backward filling of missing values, indexing by date, and applying rolling statistics like moving averages. More importantly, pandas supports hierarchical indexing and multi-column operations which are essential when analyzing multiple tickers simultaneously. For machine learning tasks, pandas enables seamless integration with scikit-learn by preparing feature matrices and target variables. It also provides utility functions for merging datasets, transforming columns, and summarizing data over time intervals. With its flexible syntax and built-in time-series handling capabilities, pandas acts as the data pipeline that channels raw market data into model-ready formats for technical indicators, feature engineering, and visualization components.

yfinance

yfinance is an open-source Python library that provides easy access to historical market data from Yahoo Finance. It abstracts the complexity of web scraping and API integration by offering simple functions to download stock data using ticker symbols. In this system, yfinance is used to fetch daily and intraday data for user-specified stocks such as RELIANCE.NS, AAPL, or HAL.NS, etc. The data includes essential attributes like Open, High, Low, Close, Volume, and adjusted prices. One of its key advantages is its ability to return data directly as pandas DataFrames, enabling immediate preprocessing and analysis. yfinance supports dynamic date ranges, frequency settings (e.g., 1d, 30m), and auto-adjustments for dividends and splits, making it extremely versatile for financial modeling. Its role is critical because reliable and accurate data retrieval is the first step in building predictive systems. Whether for backtesting breakout strategies or training machine learning models, yfinance ensures that the input data is current, clean, and properly formatted for further use to implement the model.

```
pip install streamlit yfinance pandas numpy matplotlib scipy scikit-learn xgboost PyPortfolioOpt
```

Python

```
import streamlit as st
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import argrelextrema
from pypfopt import EfficientFrontier, risk_models, expected_returns, plotting
from pypfopt.discrete_allocation import DiscreteAllocation, get_latest_prices
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Python

Fig 1. Installing and Importing the required libraries

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. In the context of this project, matplotlib is extensively used to visualize historical prices, support and resistance levels, buy/sell signals, portfolio allocations, and predicted versus actual price trends. It allows for fine-grained control over plot elements such as labels, legends, grid lines, and color schemes. For the Breakout Strategy module, matplotlib plots stock prices along with indicators such as local maxima/minima to illustrate key decision points. In the Portfolio Forecasting module, pie charts depict optimal capital allocation among stocks. In Intraday Prediction, it helps visualize the model's accuracy through line graphs comparing actual and predicted prices. Matplotlib's seamless integration with pandas and NumPy makes it the ideal choice for plotting time-series data. It is also highly customizable, enabling the creation of publication-ready visuals which are critical for financial reports, dashboards, or academic presentations.

scikit-learn

Scikit-learn, or sklearn, is a machine learning library in Python that provides simple and efficient tools for data mining and data analysis. In this stock market prediction project, scikit-learn plays a pivotal role in model training, evaluation, and performance measurement. Specifically, it is used to split data into training and testing sets, perform model evaluation using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2), and implement algorithms like RandomForestRegressor. Scikit-learn ensures a standardized interface for model development, enabling quick experimentation and hyperparameter tuning via tools like GridSearchCV. It facilitates robust machine learning pipelines through preprocessing, validation, and scoring mechanisms. The modular structure of scikit-learn aligns perfectly with the project's need for rapid prototyping and reproducibility. Overall, scikit-learn provides the essential tools to implement and benchmark predictive models within a consistent and reliable framework.

XGBoost

XGBoost, or Extreme Gradient Boosting, is an advanced machine learning library designed for high performance and scalability. In this project, XGBoost is employed for regression tasks, particularly in forecasting short-term returns and enhancing portfolio predictions. It implements gradient boosting algorithms that iteratively build an ensemble of weak learners, usually decision trees, to minimize prediction error. Its strength lies in regularization capabilities, handling missing data, and parallel computation, which makes it suitable for time-sensitive financial applications. XGBoost offers fine control over model complexity through hyperparameters such as learning rate, tree depth, and subsampling ratio. These features help reduce overfitting and increase predictive power, especially in volatile market conditions. In the Portfolio Forecasting module, XGBoost models are trained on technical indicators like moving averages and RSI to estimate future returns, which are then used in portfolio optimization. Its integration with pandas and scikit-learn simplifies preprocessing and evaluation, while its robust performance ensures accuracy and efficiency in prediction.

PyPortfolioOpt

PyPortfolioOpt is a Python library designed for financial portfolio optimization. It is based on modern portfolio theory and provides a suite of tools for optimizing asset allocations based on expected returns, risk models, and performance objectives such as maximizing the Sharpe ratio. In this project, PyPortfolioOpt is used in the Portfolio Forecasting module to allocate user capital across multiple stocks optimally. The library facilitates calculation of expected returns using historical data, estimation of the covariance matrix through shrinkage methods, and derivation of optimal weights via convex optimization. Additionally, PyPortfolioOpt supports discrete allocation, translating fractional weights into practical investment decisions considering real share prices. Its advanced features allow users to tailor risk preferences, rebalance strategies, and even set constraints on asset classes. The integration of PyPortfolioOpt empowers the system with institutional-grade portfolio management capabilities, making it suitable for both retail and professional investors seeking data-driven capital allocation strategies.

scipy.signal

The scipy.signal module, specifically the argrelextrema function, is utilized in this project for technical analysis, particularly in identifying local maxima and minima in time-series data. This method is pivotal in the Breakout Strategy module, where support and resistance levels are determined by locating these turning points in historical price series. Unlike simple moving average crossovers, using relative extrema allows for more adaptive and visually intuitive support/resistance zones, making the strategy more responsive to market dynamics. These identified levels are then used to generate buy or sell signals based on the breakout behaviour of the stock. The use of scipy.signal provides a mathematical and robust approach to pattern recognition in noisy financial data. It offers a significant advantage over heuristic methods by incorporating window-based extremum detection, which can be finely tuned via parameters like order size. This enhances the precision of signal generation, critical for technical traders aiming for timing accuracy.

datetime.timedelta

The timedelta class from the datetime module is used to manage date ranges and intervals in the system. Although it may appear simple, its functionality is indispensable in time-series forecasting and intraday data manipulation. For instance, when retrieving stock prices or creating training sets for machine learning, there is often a need to define time horizons—such as "10 days ago" or "next 30 minutes"—and timedelta enables such manipulations effortlessly. It also aids in aligning time frames between different datasets or when shifting target variables for supervised learning tasks. Its inclusion ensures accurate temporal calculations, making it easier to develop time-sensitive financial models. The use of timedelta thus complements libraries like yfinance and pandas by enabling date-based filtering, validation, and synchronization of datasets. This is particularly valuable in high-frequency trading and intraday prediction models, where even minor discrepancies in timestamps can lead to erroneous conclusions.

Streamlit

Streamlit is used as the interactive front-end framework to build and deploy the entire stock price prediction and portfolio optimization tool in a web-based interface. It allows users to input custom stock tickers and investment capital, and then displays real-time financial insights, machine learning model performance, optimized asset allocations, and portfolio visualizations. The app leverages Streamlit's simple syntax to integrate user input, display model metrics, and generate dynamic plots such as pie charts, the efficient frontier, and historical stock performance. This enables an intuitive and user-friendly experience for investors and data science practitioners without requiring any frontend development effort. The final application is deployed using Streamlit Community Cloud for free public access.

3. INTRADAY PREDICTION USING RANDOM FOREST

3.1 Introduction to Intraday Forecasting

Intraday trading involves the purchase and sale of financial instruments within the same trading day. Because of the high volatility and data granularity associated with this form of trading, the need for robust and timely predictive analytics becomes paramount. The module described in the code snippet focuses on leveraging machine learning—in particular, the Random Forest algorithm—for forecasting the next closing price based on recent intraday historical data. The approach aims to capture micro-level price movements that can inform short-term trading strategies.

3.2 Data Acquisition and Preparation

The prediction begins with the retrieval of historical data using the yfinance library. For intraday modeling, the code specifies a period of 10 days and an interval of 30 minutes, ensuring high-frequency data granularity. This setup allows the model to observe patterns that manifest within a single trading day or span multiple days. The data fetched includes essential market features: 'Open', 'High', 'Low', 'Close', and 'Volume'. These features are commonly used by traders for constructing strategies based on price action and liquidity dynamics.

After retrieving the data, the DataFrame is cleaned using dropna(), ensuring that missing entries do not compromise model accuracy. The 'Target' variable is created by shifting the 'Close' column by one timestep ahead. This sets up the supervised learning environment where the model learns to predict the next closing price given the current market features.

3.3 Feature Selection

The model uses four primary features: 'Open', 'High', 'Low', and 'Volume'. These are chosen based on their immediate availability and predictive strength. These features reflect market sentiment and trading activity in real time. 'Open' prices often capture the influence of overnight news and global events; 'High' and 'Low' represent intra-interval extremes that

denote volatility; 'Volume' reflects the intensity of trading activity. Together, they provide a well-rounded snapshot of market behavior for each time interval.

While more complex indicators like RSI or MACD could be added, the chosen features prioritize speed and simplicity—crucial for real-time prediction in a production setting. This simplicity also ensures that the Random Forest model remains interpretable and avoids overfitting, especially given the relatively short data span of ten days.

3.4 Train-Test Splitting

For machine learning models to be validated correctly, the dataset must be divided into training and testing sets. Here, the data is split using train_test_split() from the scikit-learn library with shuffle=False. This is important for time-series data, where the chronological order of events must be preserved. Shuffling would break temporal dependencies and introduce data leakage, thus inflating the model's performance unrealistically.

In this implementation, 80% of the data is used for training, and the remaining 20% is reserved for testing. This split is reasonable given the need to preserve sufficient examples for the model to learn from while also providing a test set that can validate the generalizability of the model's performance.

```
[ ] st.header("Intraday Prediction with Random Forest")
    ticker = st.text_input("Enter Ticker:", "HAL.NS")
    df = yf.download(ticker, period='10d', interval='30m', auto_adjust=True)
    df = df[['Open', 'High', 'Low', 'Close', 'Volume']]
    df.dropna(inplace=True)
    df['Target'] = df['Close'].shift(-1)
    df.dropna(inplace=True)

    X = df[['Open', 'High', 'Low', 'Volume']]
    y = df['Target']
    X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)
    model = RandomForestRegressor(n_estimators=100)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    st.write("### R² Score:", r2_score(y_test, y_pred))
    st.write("### MSE:", mean_squared_error(y_test, y_pred))

    fig, ax = plt.subplots(figsize=(12, 4))
    ax.plot(y_test.values, label='Actual', color='blue')
    ax.plot(y_pred, label='Predicted', color='red')
    ax.set_title(f"Intraday Price Prediction for {ticker}")
    ax.legend()
    ax.grid(True)
    st.pyplot(fig)
```

Fig 2. Code of Intraday Prediction with Random Forest

3.5 Model Training with Random Forest

Random Forest is a type of ensemble learning model based on decision trees. It operates by creating a multitude of decision trees during training and aggregating their outputs for predictions. The use of RandomForestRegressor with 100 estimators (trees) is a balance between computational efficiency and model robustness. The model is trained using the training data, learning complex, nonlinear relationships between input features and the target output. One key advantage of Random Forest is its resistance to overfitting compared to individual decision trees. This makes it particularly useful in financial forecasting where noise in data is prevalent. Moreover, Random Forests can capture feature interactions without requiring extensive feature engineering, which is an asset in intraday trading environments where time constraints are stringent.

3.6 Predictions and Evaluation

Once trained, the model is used to predict the next interval's closing prices on the test set. The predictions are then evaluated using two metrics: R^2 Score and Mean Squared Error.

- **R^2 Score** measures how well the model captures variance in the target variable. A value closer to 1 indicates a better fit.
- **MSE** calculates the average squared difference between actual and predicted values. Lower MSE values denote better predictive accuracy.

These metrics provide both a relative and absolute sense of model performance. Together, they help determine whether the model can generalize to unseen data and provide actionable predictions for intraday trading.

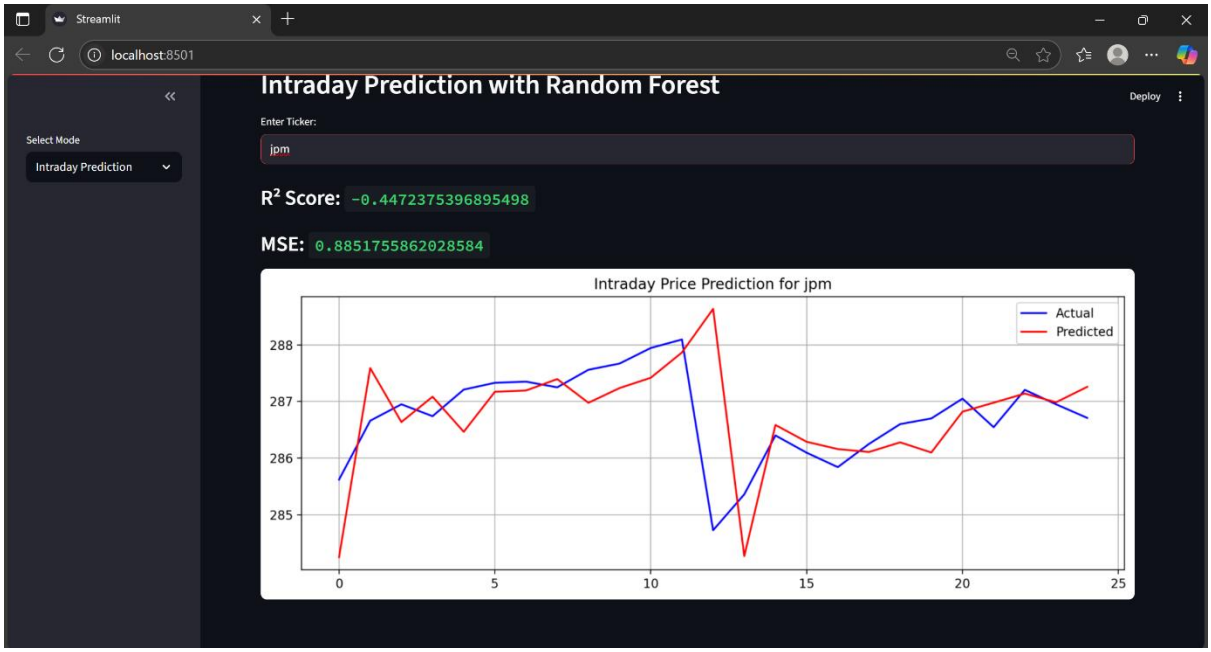
3.7 Streamlit Integration and Visualization

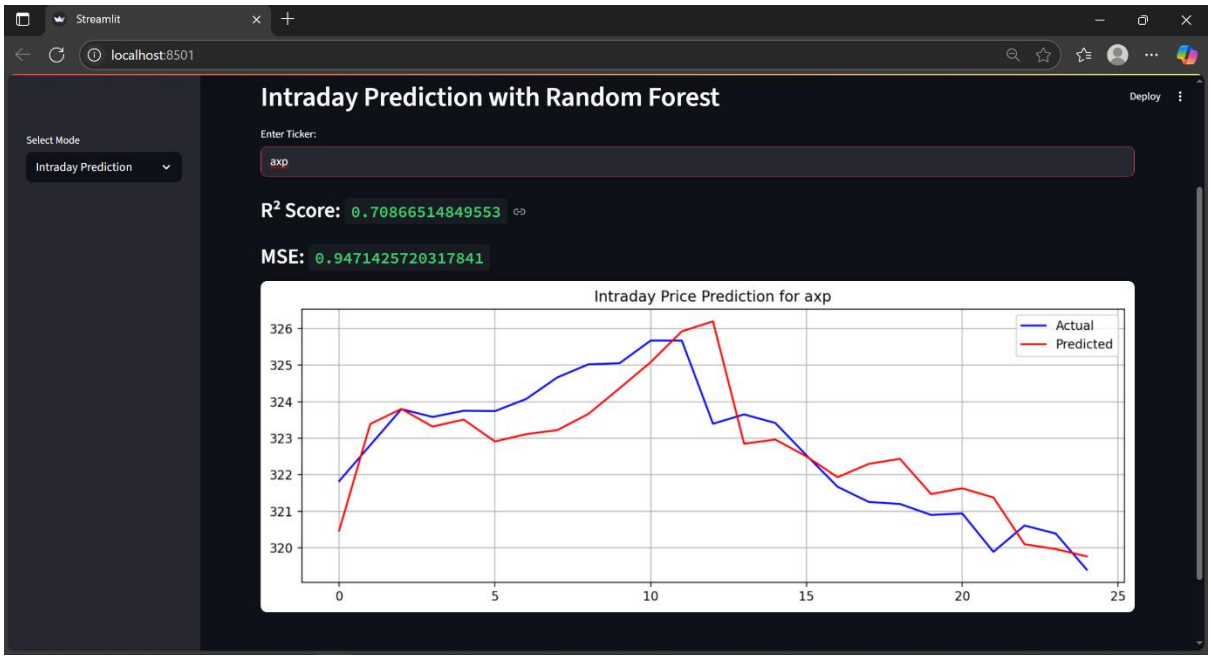
The final section of this module integrates the trained model into a user interface using Streamlit. Users can enter a specific stock ticker (default: HAL.NS), view real-time predictions, and examine model performance metrics directly on the app.

Additionally, a matplotlib-generated plot is used to visually compare actual vs predicted prices over the test set. This provides an intuitive understanding of how closely the model

tracks real market movements. A strong correlation between the two lines would indicate high model reliability, while large divergences would suggest the need for model refinement.

The graphical output aids in debugging and model tuning. It also adds value for end-users who may not have the technical expertise to interpret numerical metrics but can understand visual trends easily.





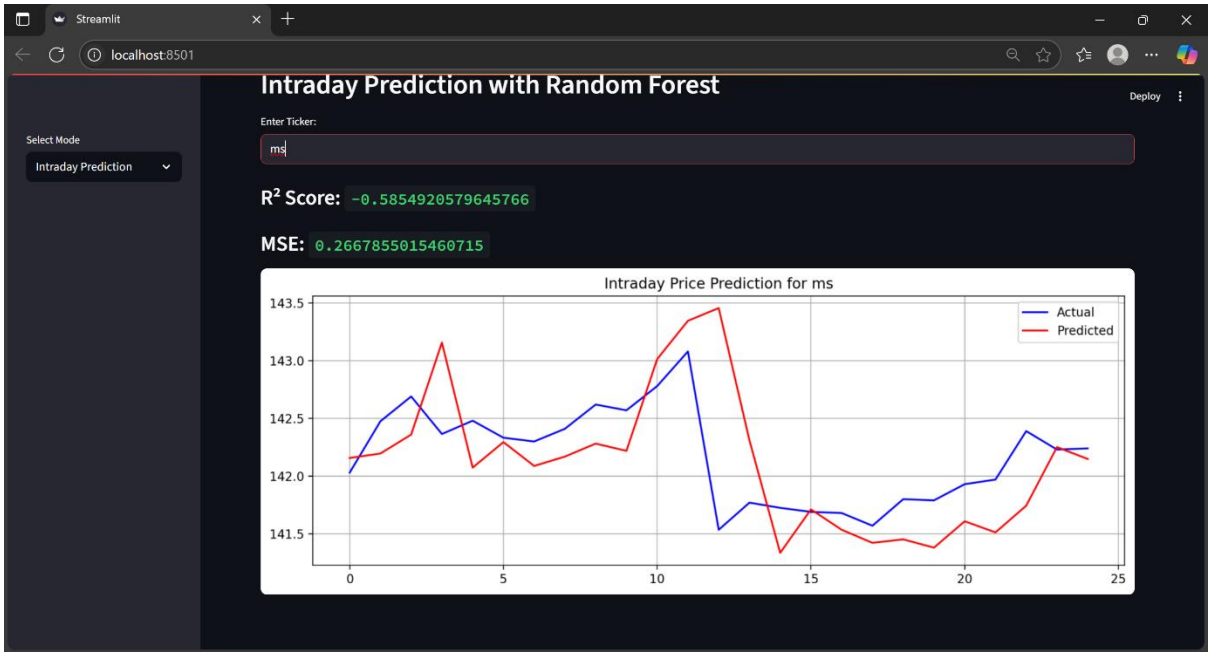


Fig 3. Output of the Intraday Prediction for some stocks

3.8 Advantages and Limitations

The intraday prediction module provides a fast, relatively accurate, and user-friendly tool for short-term price forecasting. Its modularity allows it to be extended with additional features like technical indicators or news sentiment scores. However, there are limitations. The model assumes that past price patterns contain sufficient information to predict the future, which may not always hold true in highly random or news-driven markets.

Additionally, Random Forests do not extrapolate well beyond the training data range and may struggle in completely new market conditions. Hence, regular retraining with fresh data is necessary to maintain model efficiency.

Despite these limitations, the intraday module represents a practical step towards real-time financial modelling using machine learning. It balances speed, accuracy, and usability, making it suitable for integration into trading dashboards or decision-support systems.

4. BREAKOUT STRATEGY WITH SUPPORT/RESISTANCE

4.1 Introduction to breakout strategy

In technical analysis, breakout strategies are among the most fundamental and widely used tools by traders to detect critical price movements. A breakout is said to occur when the price of a stock moves above a defined resistance level or below a defined support level with increased volume. These breakout levels represent significant psychological levels in trading, where past price action suggests major shifts in demand and supply. Identifying these levels and using them to anticipate future movements is both an art and a science. In this section, we describe how such a strategy is systematically implemented through the Breakout Strategy module using Python and Streamlit.

4.2 Data Acquisition

The foundational step in implementing the Breakout Strategy involves the systematic acquisition and preprocessing of financial time series data. Accurate and clean market data is essential for the success of any trading strategy, particularly those based on technical analysis such as support and resistance levels. In this module, we utilize the yfinance library [a Python interface for Yahoo Finance] which allows for easy access to historical and real-time market data for a wide range of financial instruments including stocks, ETFs, indices, and more.

For each stock symbol, the system downloads one year ('1y') of historical daily ('1d') data. The parameter (auto_adjust=True) ensures that price data is adjusted for dividends and stock splits, thereby preserving the integrity of time series patterns. The downloaded dataset includes the Open, High, Low, Close, and Volume (OHLCV) columns essential elements for technical analysis.

After fetching the raw data, the module selects only the relevant columns (Open, High, Low, Close, Volume) and removes any rows with missing values using dropna(). This preprocessing step is critical because missing or corrupt entries in time series data can lead to inaccurate calculation of extrema, false signals, or even runtime errors during model evaluation. By cleaning the dataset upfront, we ensure that subsequent steps in the strategy pipeline such as identifying support and resistance levels are performed on high-quality data.

4.3 Understanding Support/Resistance in Technical Analysis

Support and resistance are foundational concepts in the field of technical analysis and are extensively used by traders to make informed decisions regarding the timing of buying and selling stocks. These levels are primarily based on historical price behavior and represent psychological barriers at which price movement tends to slow down, reverse, or consolidate due to the collective behavior of market participants.

Support is defined as a price level where a stock or asset experiences increased demand, meaning buyers are more likely to purchase at that price. As a result, support levels act as a "floor" that prevents the asset's price from falling further. When prices approach this level, traders anticipate a reversal or bounce upward, often resulting in a rise in buying activity. These levels are typically tested multiple times and, if the support level holds, it confirms its strength. However, if the price falls below the support level, it may signal a bearish trend or a continuation of downward movement, often triggering stop-loss orders or panic selling.

Resistance, on the other hand, is the opposite concept. It refers to a price level where selling pressure increases, often due to traders taking profits or short sellers entering the market. Resistance acts as a "ceiling" that prevents the price from rising further. When a price approaches a resistance level, it may struggle to move past it due to a surge in sell orders. If the resistance level is eventually broken, it may indicate a bullish breakout, prompting further upward momentum.

Importantly, **support and resistance are not absolute values but rather zones or ranges** where buying and selling pressure accumulate. These areas reflect mass psychology, driven by historical memory, market sentiment, institutional behavior, and algorithmic strategies.

4.4 Automating Support/Resistance Detection

While support and resistance levels can be drawn manually by experienced traders on a price chart, automating their detection is critical for systematic and algorithmic trading systems. In this project, the identification of support and resistance zones is handled programmatically using the argrelextrema function from the scipy.signal library.

The argrelextrema() function is designed to detect **local extrema**—either minima or maxima—within a numerical sequence (like stock price data). Specifically, it identifies indices at which a given data point is greater or less than its neighboring values within a

specified range. This allows us to algorithmically determine points where the price has peaked or bottomed out relative to its surrounding points.

The function's key parameter is order=n, where n defines how many data points on either side of a given index should be used for comparison. For example:

- A value of n=10 means each data point is compared with 10 values before and 10 after it.
- If the current high price is greater than those 20 surrounding values, it's considered a **local maximum**, i.e., a candidate for resistance.
- Similarly, if a low price is lower than its neighbors, it's marked as a **local minimum**, i.e., a candidate for support.

4.5 Applying Local Extrema for Breakout Detection

After loading and preparing the data, the program calculates resistance and support levels by applying argrelextrema() to the 'High' and 'Low' price columns respectively. These functions search for local highs and lows in the context of the surrounding 10 data points (as defined by n = 10).

The result is a sparse array of resistance and support values at selected points. However, for the purposes of signal generation, continuous tracking of these levels is necessary. This is achieved by applying a forward-fill operation (ffill()), which propagates the last known resistance or support level throughout the DataFrame until a new level is detected. This creates a consistent structure that allows easy comparison of the current closing price with these benchmark levels.

4.6 Signal Generation Logic

To operationalize the breakout detection mechanism, a custom function breakout_strategy() is defined. This function iterates over each row of the dataset and assigns a trading signal:

- A Buy Signal (1) is generated if the current closing price surpasses the current resistance level.

- A Sell Signal (-1) is generated if the closing price drops below the current support level.
- A Hold Signal (0) is returned otherwise, indicating no actionable insight.

This function is vectorized and applied to the entire DataFrame, resulting in a new column called 'signal'. This column acts as the decision engine for the breakout strategy. This type of rule-based approach mirrors real-world trading systems where specific thresholds or indicators trigger automated buy/sell decisions.

```
st.header("Breakout Strategy with Support/Resistance")
    stocks = st.text_input("Enter Ticker Symbols (comma separated):")
    stock_list = [s.strip().upper() for s in stocks.split(",") if s.strip()]
    n = 10
    for symbol in stock_list:
        st.subheader(f"{symbol}")

        print(f"\nProcessing {symbol}...")
        df = yf.download(symbol, period='1y', interval='1d', auto_adjust=True)

        df = df[['Open', 'High', 'Low', 'Close', 'Volume']]
        df.dropna(inplace=True)

        df['resistance'] = df.iloc[argrelextrema(df['High'].values, np.greater_equal, order=n)[0]]['High']
        df['support'] = df.iloc[argrelextrema(df['Low'].values, np.less_equal, order=n)[0]]['Low']

        df['resistance'] = df['resistance'].ffill()
        df['support'] = df['support'].ffill()
```

```
def breakout_strategy(row):
        if row[('Close', symbol)] > row[('resistance', '')]:
            return 1
        elif row[('Close', symbol)] < row[('support', '')]:
            return -1
        else:
            return 0
    df[('signal', '')] = df.apply(breakout_strategy, axis=1)

    fig, ax = plt.subplots(figsize=(12, 4))
    ax.set_title('Support/Resistance Breakout Strategy')
    ax.set_xlabel('Date')
    ax.set_ylabel('Price')
    ax.plot(df.index, df[('Close', symbol)], label='Close', linewidth=1.5)
    ax.plot(df['resistance'], label='Resistance', linestyle='--', color='red', alpha=0.5)
    ax.plot(df['support'], label='Support', linestyle='--', color='green', alpha=0.5)
    ax.scatter(df.index[df[('signal', '')] == 1], df[('Close', symbol)][df[('signal', '')] == 1],
               marker='^', color='green', label='Buy', alpha=0.8)
    ax.scatter(df.index[df[('signal', '')] == -1], df[('Close', symbol)][df[('signal', '')] == -1],
               marker='v', color='red', label='Sell', alpha=0.8)
    ax.scatter(df.index[df[('signal', '')] == 0], df[('Close', symbol)][df[('signal', '')] == 0],
               marker='o', color='blue', label='Hold', alpha=0.5)
    ax.legend()
    ax.grid(True)
    fig.tight_layout()
    st.pyplot(fig)
```

Fig 4. Code of Breakout Strategy with Support/Resistance

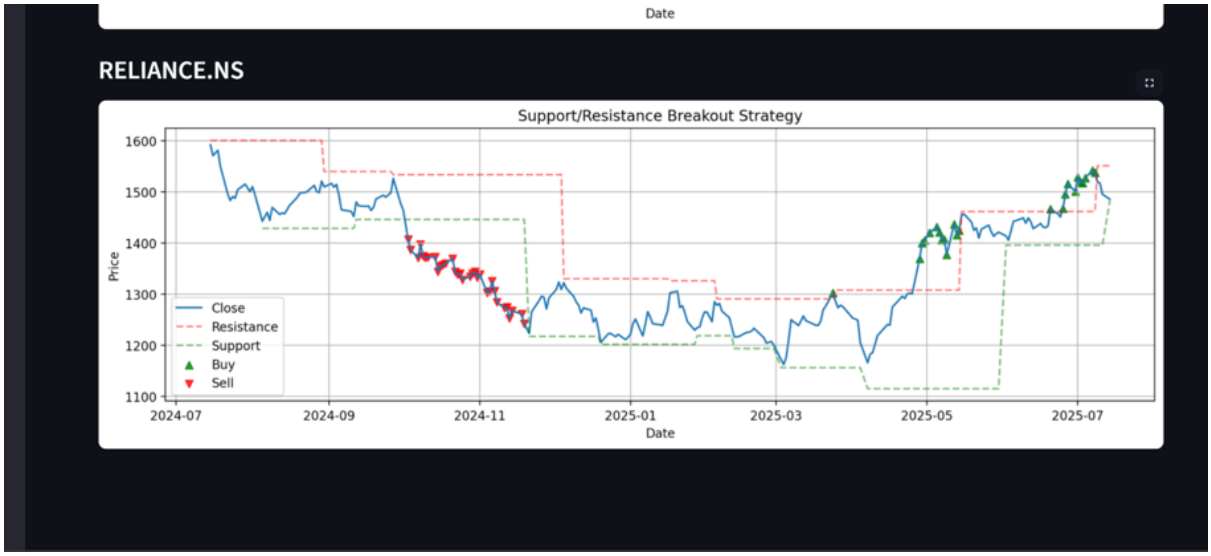
4.7 Visualization and Interpretation

The strategy's effectiveness is demonstrated through a series of visualizations using matplotlib. For each stock ticker, a time-series line graph is created showing the closing price, support and resistance levels, and trade signals. These plots are intuitive, visually appealing, and easy to interpret.

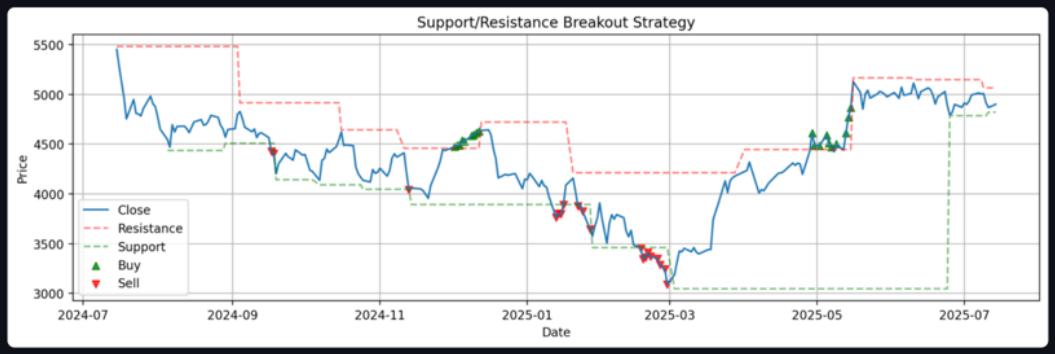
- The closing price is shown as a continuous line.
- Resistance levels are depicted with dashed red lines.
- Support levels are shown with dashed green lines.
- Buy, Sell, and Hold signals are highlighted with distinctive markers (green upward triangle, red downward triangle, blue dot respectively).

These visual outputs allow users to understand how the model interprets price action and where the buy/sell signals are triggered in historical data. Visual confirmation helps validate whether the algorithm aligns with user expectations or market theory.

Fig 5. Output of the Breakout Strategy for some stocks

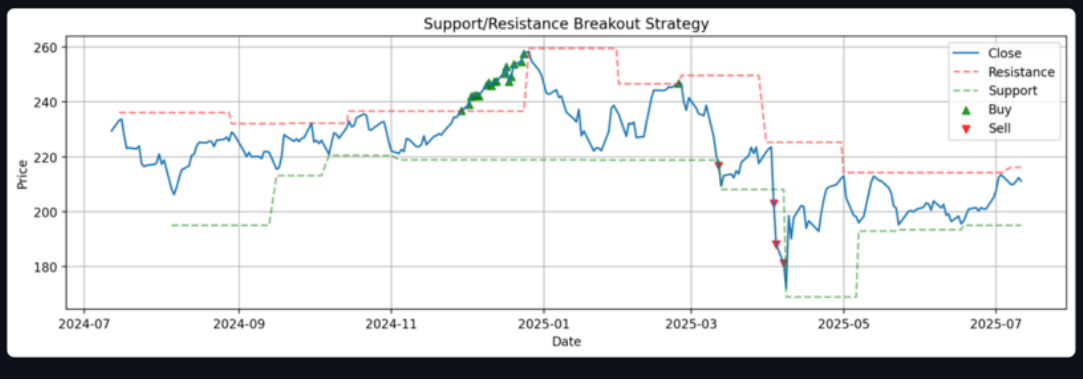


HAL.NS



MSFT

AAPL



4.8 Strengths of Breakout Strategy

- **Rule-Based Transparency:** The strategy is straightforward and explainable, using well-known technical indicators.
- **Scalable Across Stocks:** The logic works across different ticker symbols, making it useful for multi-asset analysis.
- **Customizable Thresholds:** Parameters like n can be adjusted to change the sensitivity of breakout detection.
- **Educational Tool:** Excellent for teaching new traders the fundamentals of support/resistance trading

The Breakout Strategy module serves as a powerful foundation for understanding and applying technical trading rules It transforms theory into practice. This approach enables traders, students, and data scientists to explore price dynamics, evaluate simple trading strategies, and iterate toward more complex and personalized models. Though basic in nature, the methodology has potential to evolve into a production-grade tool through systematic refinement and integration with other analytical layers.

5. STOCK PORTFOLIO OPTIMIZATION

5.1 Introduction

Portfolio forecasting is a strategic methodology that applies mathematical and statistical models to project the future performance of a group of selected assets. It is a fundamental tool in finance that empowers investors to make data-driven decisions regarding capital allocation. In our system, this module leverages the modern portfolio theory (MPT) developed by Harry Markowitz, integrating it with machine learning predictions for return estimations and advanced risk models. The resulting framework assists users in optimizing their portfolio to maximize expected return while managing risk under specified constraints.

5.2 Data Acquisition and Preprocessing

The user is first prompted to input a list of ticker symbols representing the stocks they are interested in, along with the amount of capital available for investment. The system fetches historical adjusted closing prices for the selected tickers from Yahoo Finance, covering the time period from January 1, 2020, to June 30, 2025. This provides a wide data window that captures different market regimes—bullish, bearish, and volatile—thus increasing the reliability of predictions.

To ensure numerical stability and consistency, missing values in the time series are handled through forward-fill and backward-fill operations. This guarantees that each date has complete information for all tickers, which is a prerequisite for constructing a proper covariance matrix and computing returns.

5.3 Feature Engineering and RSI Calculation

The portfolio forecasting module constructs a rich set of features for each ticker using standard technical indicators. These include:

1. **Logarithmic Returns** to normalize percentage price changes

Log returns are preferred over simple percentage returns because they are **time additive**, meaning that the sum of log returns over multiple periods equals the total log return over the combined time. This property simplifies the model in long-term returns and portfolio aggregation. Furthermore, log returns help **normalize extreme values** and reduce skewness in price distributions, which is essential when using regression models or other statistical learning algorithms.

2. **Lag Features** to represent short-term momentum

   Lag features are particularly useful in financial markets, where **momentum effects**—the tendency for assets that performed well recently to continue performing well—are commonly observed. Including multiple lag features enables the model to better recognize short-term trends, reversals, and the persistence of price movement over time.

3. **Moving Averages** (MA5 and MA10) to capture trend strength

   Moving averages (MA) are widely used indicators in technical analysis that smooth out price data to identify the direction of a trend. The two types commonly used are:

   MA5: 5-day moving average

   MA10: 10-day moving average

   In predictive modeling, moving averages help capture **trend strength** and **momentum over time**. Short-term MAs like MA5 and MA10 are particularly useful for identifying changes in direction and volatility in price series.

   Including MA values as features allows the machine learning model to understand whether the price is trending upward, downward, or moving sideways. This information complements other indicators like log returns or RSI to give a fuller picture of market behaviour.

4. **Volatility Measures** using rolling standard deviation

   Volatility refers to the degree of variation of a trading price series over time and is a core concept in risk management and financial analysis.

5. **Relative Strength Index (RSI)** to capture overbought or oversold conditions

The **Relative Strength Index (RSI)** is a momentum oscillator that measures the magnitude of recent price changes to evaluate overbought or oversold conditions.

The RSI is calculated using a 14-period lookback. It quantifies the strength of recent price changes and is a staple in momentum-based trading strategies. By combining these features, we obtain a robust representation of the underlying stock behavior for predictive modeling.

```python
# Utility Functions
def calculate_rsi(data, periods=14):
    delta = data.diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=periods).mean()
    avg_loss = loss.rolling(window=periods).mean()
    rs = avg_gain / avg_loss
    return 100 - (100 / (1 + rs))

def prepare_features(df, ticker):
    df = df[[ticker]].copy()
    df['LogReturn'] = np.log(df[ticker] / df[ticker].shift(1))
    df['Lag1'] = df['LogReturn'].shift(1)
    df['Lag5'] = df['LogReturn'].shift(5)
    df['MA5'] = df[ticker].rolling(window=5).mean()
    df['MA10'] = df[ticker].rolling(window=10).mean()
    df['Vol10'] = df['LogReturn'].rolling(window=10).std()
    df['RSI'] = calculate_rsi(df[ticker])
    df = df.dropna()
    X = df[['Lag1', 'Lag5', 'MA5', 'MA10', 'Vol10', 'RSI']]
    y = df['LogReturn']
    return X, y
```

5.4 Predictive Modeling using XGBoost

To estimate expected returns, we employ the XGBoost Regressor, a powerful ensemble machine learning technique based on gradient boosting. For each ticker, we split the dataset into training and testing subsets. Trains an XGBoost model with grid search to predict daily log returns for each stock. Parameters tuned: n_estimators, learning_rate, max_depth, reg_alpha, reg_lambda. The model learns to predict next-day log returns using the engineered features. Once trained, it forecasts the most recent data point's return, annualized by multiplying by 252 (the approximate number of trading days in a year).

This predictive capability goes beyond traditional historical mean return estimation. It incorporates recent trends and statistical patterns, improving the responsiveness and accuracy of portfolio optimization under dynamic market conditions.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
param_grid = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
    'reg_alpha': [0, 0.1],
    'reg_lambda': [1]
}
xgb = XGBRegressor(random_state=42)
grid_search = GridSearchCV(xgb, param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_xgb = grid_search.best_estimator_

y_pred = best_xgb.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
st.write(f"{ticker:<15} {mae:.6f} {rmse:.6f} {r2:.6f}")

latest_features = X.iloc[-1:].values
predicted_daily_return = best_xgb.predict(latest_features)[0]
predicted_annual_return = predicted_daily_return * 252
predicted_returns[ticker] = predicted_annual_return
valid_tickers.append(ticker)
```

5.5 Risk Modelling and Efficient Frontier Optimization

After estimating returns, the system computes the annualized covariance matrix using Ledoit-Wolf shrinkage, a robust technique from PyPortfolioOpt that ensures stability and reduces overfitting in high-dimensional data. The covariance matrix quantifies the risk associated with the selected assets and how their returns correlate with each other.

We then feed the predicted return vector (mu) and the covariance matrix (S) into the EfficientFrontier class. The optimization target is to maximize the Sharpe Ratio — a widely accepted risk-adjusted performance metric. The risk-free rate is assumed to be 6% annually. The optimizer returns the optimal weight allocation across assets that offers the best expected return for a given level of risk.

```
# Portfolio Optimization with Frontier Effect
st.subheader("Portfolio Optimization (Efficient Frontier)")
mu = expected_returns.mean_historical_return(adj_close_df[valid_tickers]) * 252
S = risk_models.CovarianceShrinkage(adj_close_df[valid_tickers]).ledoit_wolf() * 252
ef = EfficientFrontier(mu, S, weight_bounds=(0, 1))
ef.max_sharpe(risk_free_rate=0.06)
optimal_weights = ef.clean_weights()
weights_dict = {ticker: weight for ticker, weight in optimal_weights.items()}

st.write("**Optimal Weights:**")
st.json(weights_dict)

port_return, port_volatility, sharpe = ef.portfolio_performance(risk_free_rate=0.06)
st.write(f"**Expected Annual Return:** {port_return:.2%}")
st.write(f"**Expected Volatility:** {port_volatility:.2%}")
st.write(f"**Sharpe Ratio:** {sharpe:.2f}")
```

5.6 Discrete Assets Allocation

Real-world portfolios consist of whole number quantities of assets. To convert the optimized continuous weights into discrete asset allocations, we use the DiscreteAllocation module. It calculates how many shares of each asset can be purchased given the user's capital and current market prices, retrieved using get_latest_prices().

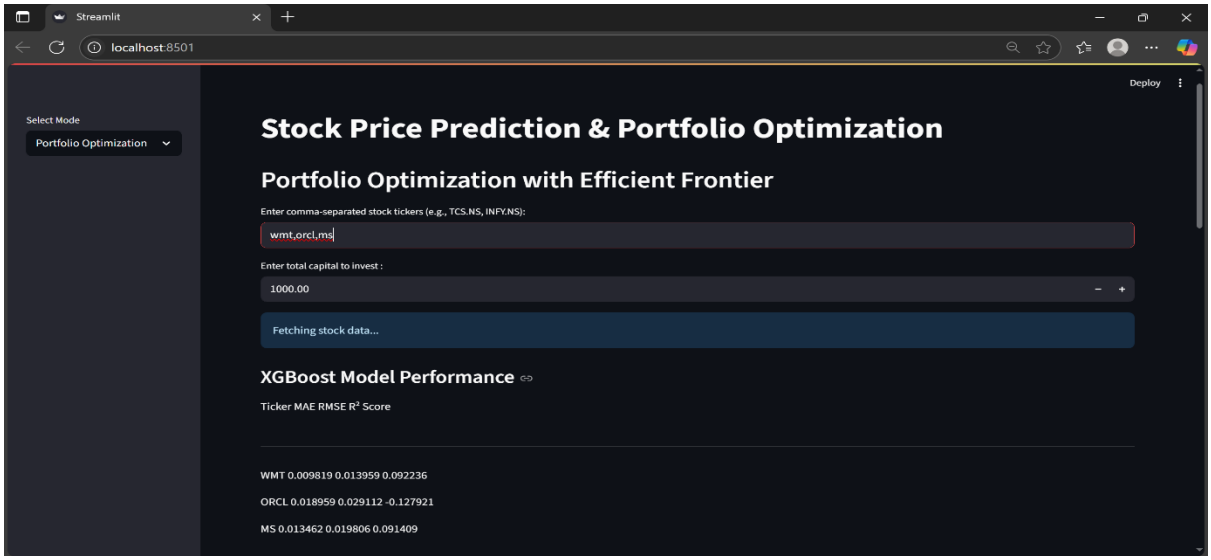
This step ensures that the portfolio recommendation is practical and implementable. It also reports leftover funds which were not sufficient to purchase an additional share of any asset, typically due to price constraints.

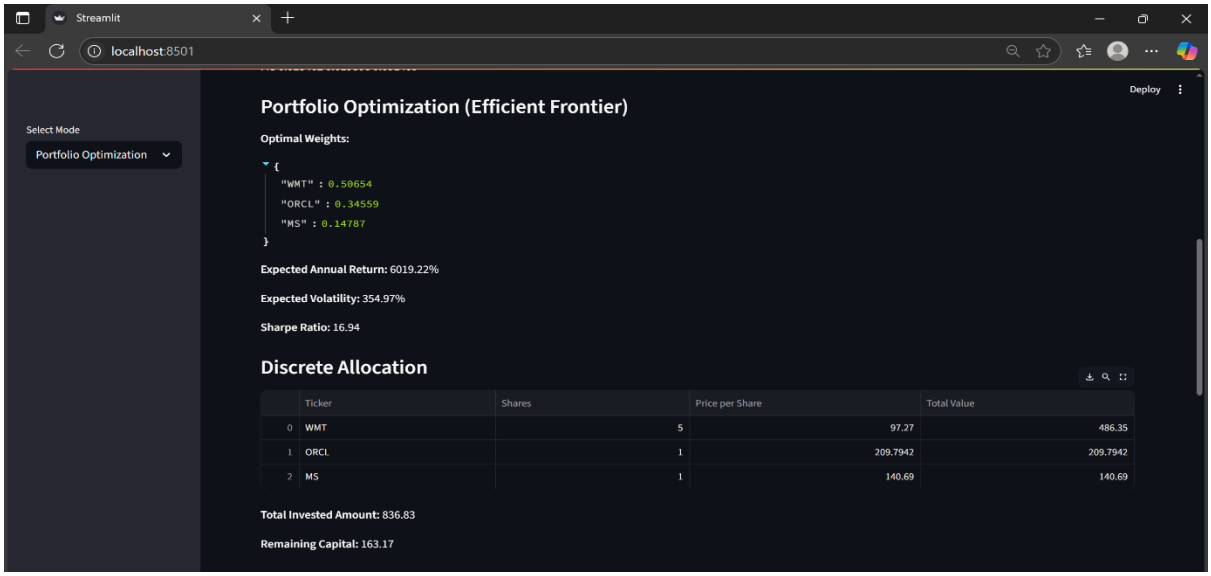
```
[ ] # Discrete Allocation
    st.subheader("Discrete Allocation")
    latest_prices = get_latest_prices(adj_close_df[valid_tickers])
    da = DiscreteAllocation(weights_dict, latest_prices, total_portfolio_value=capital)
    allocation, leftover = da.greedy_portfolio()

    alloc_df = pd.DataFrame([
        {"Ticker": k, "Shares": v, "Price per Share": latest_prices[k], "Total Value": v * latest_prices[k]}
        for k, v in allocation.items()
    ])
    st.dataframe(alloc_df)
    st.write(f"**Total Invested Amount:** {alloc_df['Total Value'].sum():,.2f}")
    st.write(f"**Remaining Capital:** {leftover:.2f}")
```

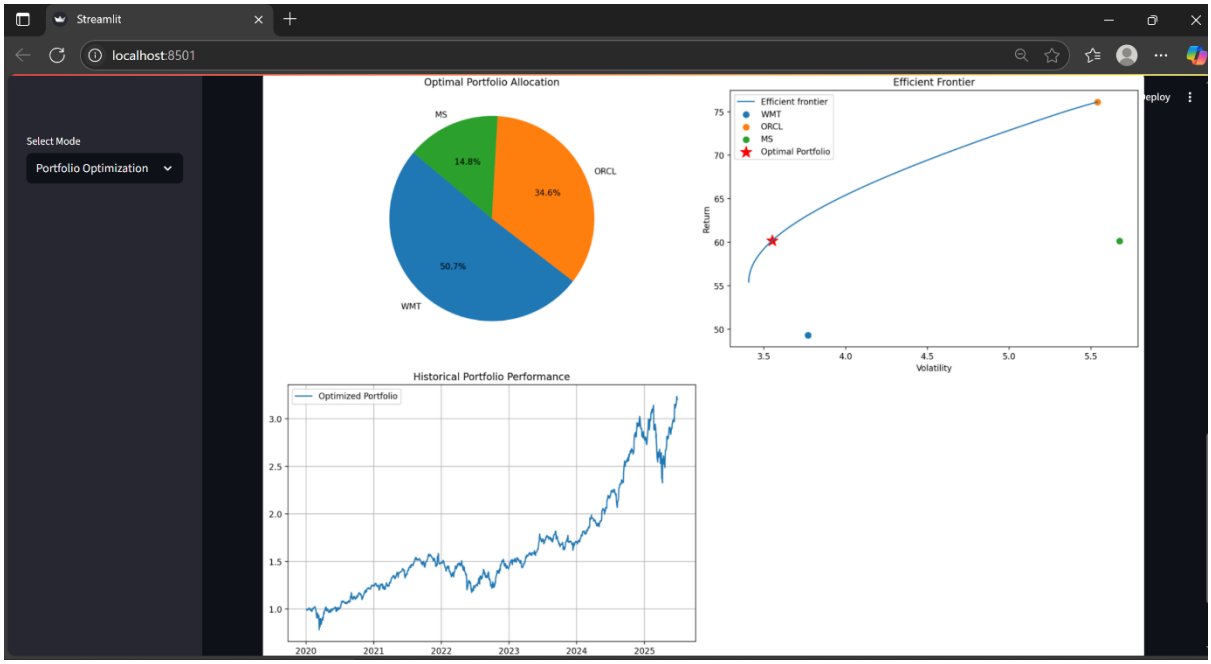
5.7 Visualization

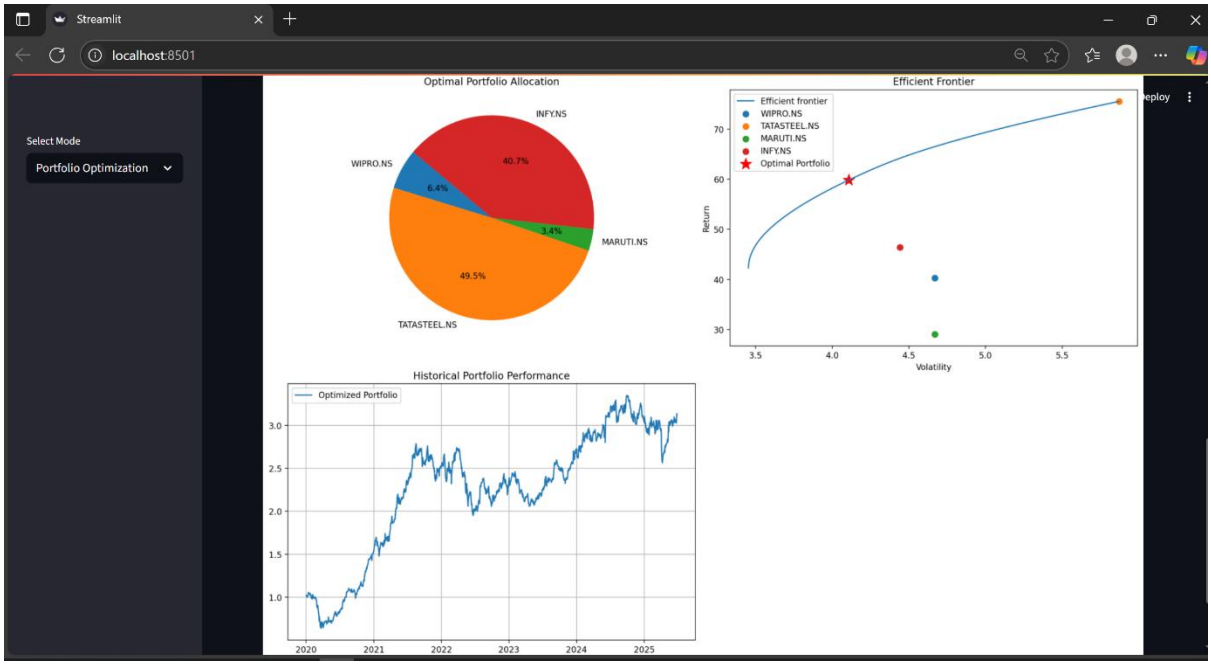
To aid interpretation, the portfolio allocation is visualized using a pie chart. Each slice represents a stock and its proportion in the total investment. The interactive Streamlit UI allows users to adjust the list of stocks or capital dynamically and view the updated portfolio in real-time.

In addition to numerical weights and allocations, this visual summary helps users intuitively understand diversification and concentration within their portfolio.


Streamlit
localhost:8501
Deploy
Select Mode
Portfolio Optimization
Stock Price Prediction & Portfolio Optimization
Portfolio Optimization with Efficient Frontier
Enter comma-separated stock tickers (e.g., TCS.NS, INFY.NS):
wmt,orcl,ms
Enter total capital to invest :
1000.00
Fetching stock data...
XGBoost Model Performance
Ticker MAE RMSE R² Score
WMT 0.009819 0.013959 0.092236
ORCL 0.018959 0.029112 -0.127921
MS 0.013462 0.019806 0.091409


Streamlit
localhost:8501
Deploy
Select Mode
Portfolio Optimization
Portfolio Optimization (Efficient Frontier)
Optimal Weights:
{
"WMT" : 0.50654
"ORCL" : 0.34559
"MS" : 0.14787
}
Expected Annual Return: 6019.22%
Expected Volatility: 354.97%
Sharpe Ratio: 16.94
Discrete Allocation
Ticker Shares Price per Share Total Value
0 WMT 5 97.27 486.35
1 ORCL 1 209.7942 209.7942
2 MS 1 140.69 140.69
Total Invested Amount: 836.83
Remaining Capital: 163.17


Streamlit
localhost:8501
Select Mode
Portfolio Optimization
Optimal Portfolio Allocation
MS
ORCL
WMT
14.8%
34.6%
50.7%
Efficient Frontier
Efficient frontier
WMT
ORCL
MS
Optimal Portfolio
Return
Volatility
Historical Portfolio Performance
Optimized Portfolio



5.8 Advantages

1. **Intelligent Return Forecasting:** Machine learning allows for more accurate and adaptive predictions than simple historical averages.
2. **Risk Management:** The use of covariance shrinkage and MPT ensures a mathematically sound foundation for decision-making.
3. **Practical Allocation:** Discrete allocation enables real-world implementation based on current asset prices.
4. **User-Friendly Interface:** Streamlit's input controls and visualization components make portfolio optimization accessible to all levels of investors.

6. CONCLUSION

This project successfully developed a unified stock market prediction system combining traditional technical analysis with modern machine learning and optimization techniques. The application allows users to interact with three core modules **Breakout Strategy**, **Intraday Prediction**, and **Portfolio Optimization,** each built using real stock data and implemented through Python and Streamlit.

The **Breakout Strategy** effectively identifies support and resistance levels to generate Buy/Sell signals. The **Intraday Prediction** module uses Random Forest to predict near-future prices based on key market features, while **Portfolio Optimization** employs XGBoost and Efficient Frontier theory to optimize asset allocation.

The system demonstrates how historical price patterns, statistical indicators, and machine learning can be integrated to guide smarter investment decisions. It provides a practical, scalable, and user-friendly platform for both learning and real-world application. Overall, the project showcases the power of combining financial domain knowledge with data science to build intelligent decision-support systems.