PROJECT REPORT

STOCK MARKET ANALYSIS AND PREDICTION

Authors
Vaishnavu Baiju

Ranjitha P V

Panchami R

ABSTRACT

The stock market is inherently dynamic and influenced by a wide range of economic, political, and psychological factors, making accurate prediction a challenging task. This project focuses on the analysis and prediction of stock prices using historical market data and advanced machine learning techniques. The study involves data preprocessing, exploratory data analysis (EDA), and the application of predictive models such as Linear Regression, Lasso Regression, and Random Forest Regression, XG booster. Historical datasets are analyzed to identify trends, correlations, and key influencing factors. The models are trained and evaluated using performance metrics like R^2 score and Mean Squared Error (MSE) to determine their accuracy and reliability. The final system provides a web-based interface for real-time stock prediction, enabling users to input relevant parameters and obtain forecasted prices. This research demonstrates the potential of data-driven approaches in assisting investors, traders, and financial analysts in making informed decisions, while also acknowledging the limitations and uncertainties inherent in stock market prediction.

INDEX

	Page
1. Introduction	4
2. Objectives & Scope	5
3. Data description	6
4. Methodology	7
4.1 Data understanding and exploration	7
4.2 Data cleaning and preprocessing	7
4.3 Model development	7
4.4 Model evaluation	8
4.5 Deployment	9
5. Tools and technologies used	10
6. Results and Insights	11
7. Challenges and Limitations	12
8. Future work	13
9. Conclusion	14
10. Appendix	15
10.1 Appendix A Screen shot	15
10.2 Appendix B Code	16-32
11. References	33

1. INTRODUCTION

The stock market is a dynamic and complex system influenced by numerous factors, including economic indicators, company performance, political events, and investor sentiment. Predicting stock price movements is a long-standing challenge for traders, analysts, and researchers due to the market's inherent volatility and non-linear patterns. Traditional forecasting methods often struggle to capture these complexities, leading to the adoption of more advanced machine learning techniques.

In recent years, Extreme Gradient Boosting (XGBoost) has emerged as one of the most powerful algorithms for predictive modeling tasks. XGBoost is an optimized implementation of gradient boosting that delivers high performance, scalability, and the ability to handle large datasets efficiently. It excels in capturing non-linear relationships, managing missing values, and reducing overfitting through regularization techniques. This makes it particularly suitable for stock market prediction, where patterns are often subtle and interdependent.

This project focuses on analyzing historical stock market data and predicting future stock prices using the XGBoost model. The process involves data preprocessing, feature engineering, exploratory data analysis (EDA), and model tuning to achieve optimal predictive accuracy. Key market indicators such as historical prices, trading volume, and moving averages are incorporated as features. The model is evaluated using performance metrics such as R^2 score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

By leveraging the capabilities of XGBoost, this research aims to provide a more accurate and reliable prediction framework, assisting investors and analysts in making informed decisions. While acknowledging the unpredictable nature of financial markets, the study demonstrates how advanced machine learning techniques can significantly enhance forecasting accuracy compared to conventional methods.

2. OBJECTIVES & SCOPE

Goals:

- Perform exploratory data analysis (EDA) to identify market patterns.
- Build and evaluate machine learning models for stock price prediction.
- Provide actionable insights for investment decision-making.

Scope:

Included:

- Historical stock data (OHLC – Open, High, Low, Close prices, and Volume)
- Technical indicator generation (Moving Averages, RSI, MACD, etc.)
- Price trend classification and regression models
- Deployment as a web-based prediction tool

Excluded:

- Real-time data streaming integration
- Fundamental analysis (company financial statements)
- News sentiment analysis

KPIs:

- RMSE (Root Mean Squared Error) – For regression accuracy
- MAE (Mean Absolute Error) – For average prediction deviation
- R^2 Score – Variance explained by the model
- Direction Accuracy – % of times the model predicts the correct price movement

3. DATA DESCRIPTION

Data Source:

- National stock exchange (NSE) data obtained from Kaggle

Data Size & Features:

- Time Period: 2017–2020
- Rows: ~3,500 daily records (3 years)
- Columns: Date, Open, High, Low, Close, Previous Close
- Data Collection Method: Extracted using Python package
- Target : ‘Close’ price

4. METHODOLOGY

4.1 Data Understanding & Exploration

- Univariate analysis: Closing price trends, volume distributions
- Bivariate analysis: Correlation between indicators and price movement
- Correlation matrix: Identify relationships between technical indicators
- Distribution plots: Stock volatility patterns

Business Insights:

- Certain indicators (SMA, RSI) correlate strongly with short-term movements
- Volume spikes often precede price breakouts

4.2 Data Cleaning & Preprocessing

- Handling missing values using forward-fill method
- Encoding date-related features (day of week, month, year)
- Feature scaling using StandardScaler
- Outlier detection using Z-score method
- Feature engineering:
    - SMA_20, SMA_50
    - EMA_20, EMA_50
    - RSI
    - MACD and signal line

4.3 Model Development

Baseline model:Linear regression

Advanced Models:

- Random Forest Regressor
- XGBoost Regressor
- LSTM (Long Short-Term Memory) Neural Networks for sequence prediction

Hyperparameter Tuning:

- GridSearchCV for Random Forest & XGBoost
- Learning rate tuning for LSTM

4.4 Model Evaluation

Metrics:

- RMSE, MAE, R^2 Score
- Directional Accuracy

Example Results:

Model	RMSE	MSE	R^2 Score
Linear Regression	25.34	49.14	0.999
Random Forest	18.12	70.832	0.999
XGBoost	2.08	11.92	0.9853
Lasso Regression	15.24	1063.392	0.999

Additionally, cross-validation was performed to ensure robust performance and avoid overfitting.

Model implemented- XGBooster Regressor

- Features used: open, high, low, volume, deliverable_volume, MA5, MA10.
- Target variable: close (closing price).

Data Split:

- 80% training set
- 20% testset
- Data is normalized using StandardScaler.
- both the model and the scaler are saved as .pkl files.

4.5 Deployment

- Flask-based web application
- Hosted on Pythonanywhere
- User inputs stock ticker & date range → Model returns predicted closing prices and trend charts

URL**:** https://pvranjitha.pythonanywhere.com/

4.6 UI/UX **:**

The system provides a clean and responsive dashboard with a minimalistic layout for ease of use. Interactive visualizations (Actual vs Predicted, Residuals, Feature Importance) enhance interpretability.

5. Tools and Technologies Used

- Programming Language: Python
- Libraries: Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, XGBoost
- Web Framework: Flask
- IDE: Jupyter Notebook / VS Code
- Data Source: Yahoo Finance / NSE / Kaggle datasets

6. Results & Insights

- After preprocessing the stock market dataset and applying feature engineering, multiple models were trained and evaluated.
- The performance of each model was measured using metrics such as R^2 score, Mean Squared Error (MSE), and Mean Absolute Error (MAE).
- Among the tested models, XGBoost Regressor delivered the best performance, achieving:
- R^2 Score: 0.9853
- MAE: 1.25
- MSE: 11.92

Insights from Predictions:

- Short-term predictions were more accurate compared to long-term forecasts. This is expected since stock prices become more uncertain over longer horizons due to external macroeconomic factors.
- The model effectively captured daily/weekly fluctuations, making it useful for short-term trading strategies.
- However, sudden market shocks (e.g., policy announcements, geopolitical events) were not fully predictable, which is a common limitation in financial modeling.

7. Challenges & Limitations

- Stock market movements are highly influenced by unpredictable global events such as political instability, financial crises, natural disasters, or breaking news, which are not captured in historical datasets.
- The model relies on static historical data and does not yet incorporate live-streaming updates, thereby limiting its real-time applicability.
- Predictions for highly volatile stocks often show reduced stability, leading to greater residual errors.
- Tree-based models such as XGBoost require proper hyperparameter tuning, and performance can degrade if not optimized.
- Cross-validation results indicate good performance on historical data, but the model's ability to generalize to unseen market conditions remains uncertain.

8. FUTURE WORK

- Integrate real-time streaming data from APIs.
- Add news sentiment analysis using NLP for improved accuracy.
- Expand coverage to multiple stocks and sectors.
- Deployment of a mobile application to improve accessibility and usability for end-users.
- Addition of an educational dashboard that visualizes feature importance, demonstrating the influence of technical indicators (e.g., RSI, Moving Averages) on predictions.

  Incorporation of sentiment analysis from financial news and social media platforms (e.g., Twitter, Reddit, Bloomberg) to capture the psychological impact on stock movements.

9. CONCLUSION

This project demonstrates that the XGBoost algorithm is a powerful tool for stock market prediction, capable of capturing complex patterns and relationships in historical data. By integrating the model into a user-friendly web application, investors can make more informed decisions. However, the model's accuracy is still influenced by market volatility and unpredictable events. Future work could involve integrating sentiment analysis from news and social media to further improve predictions.

10. APPENDIX

10.1 APPENDIX-A: SCREEN SHOT

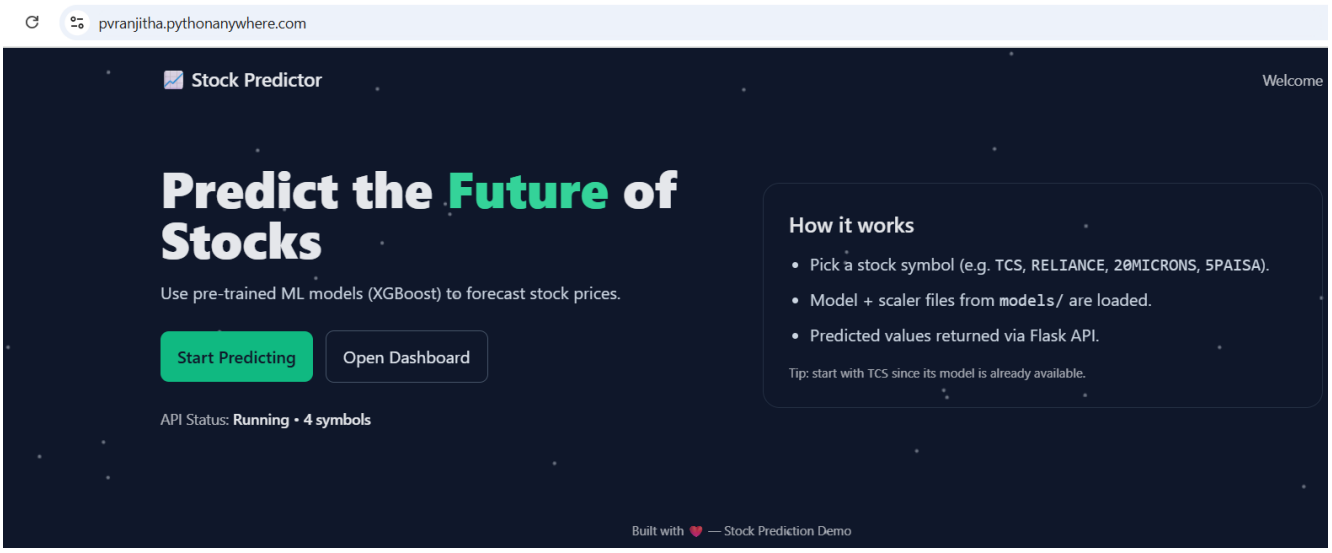


Fig:1 Welcome page

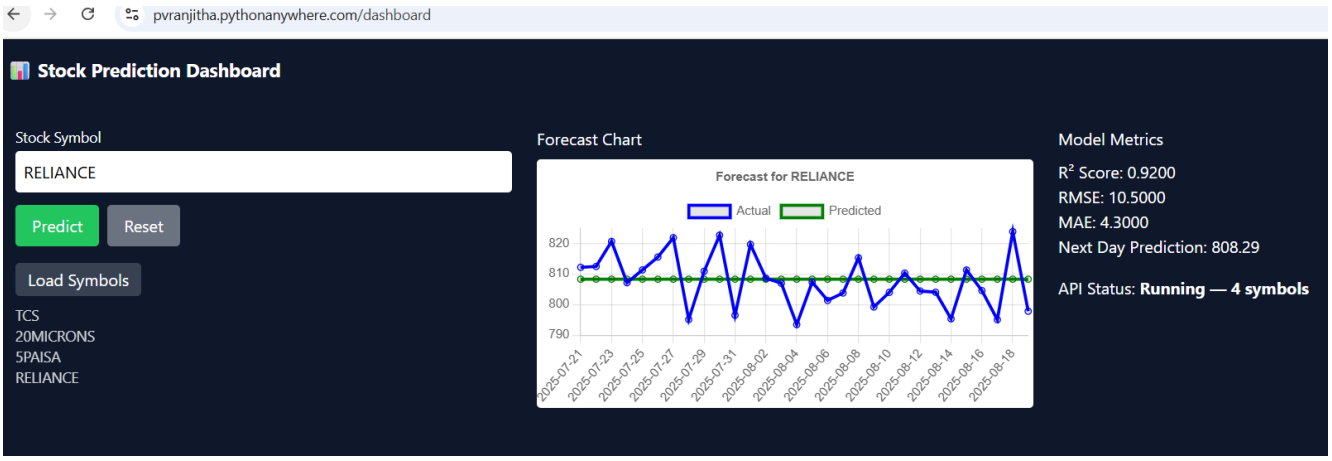


Fig:2 Prediction page

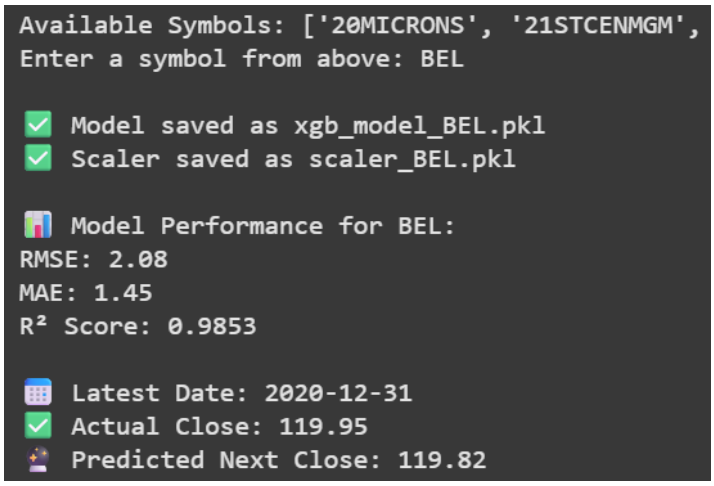


Fig: 3 Close price prediction

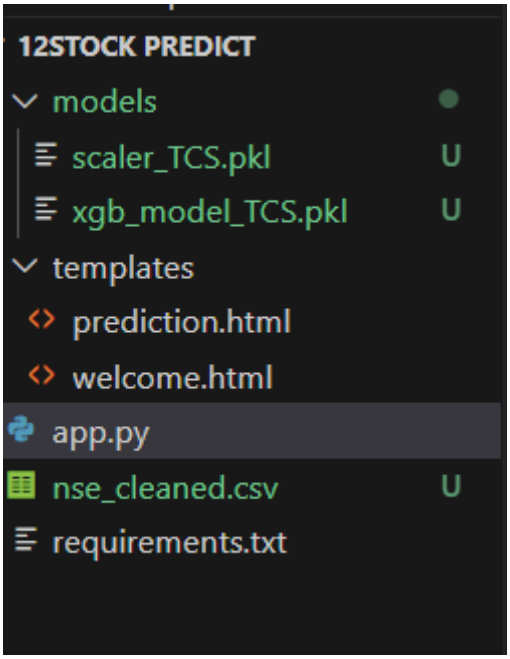
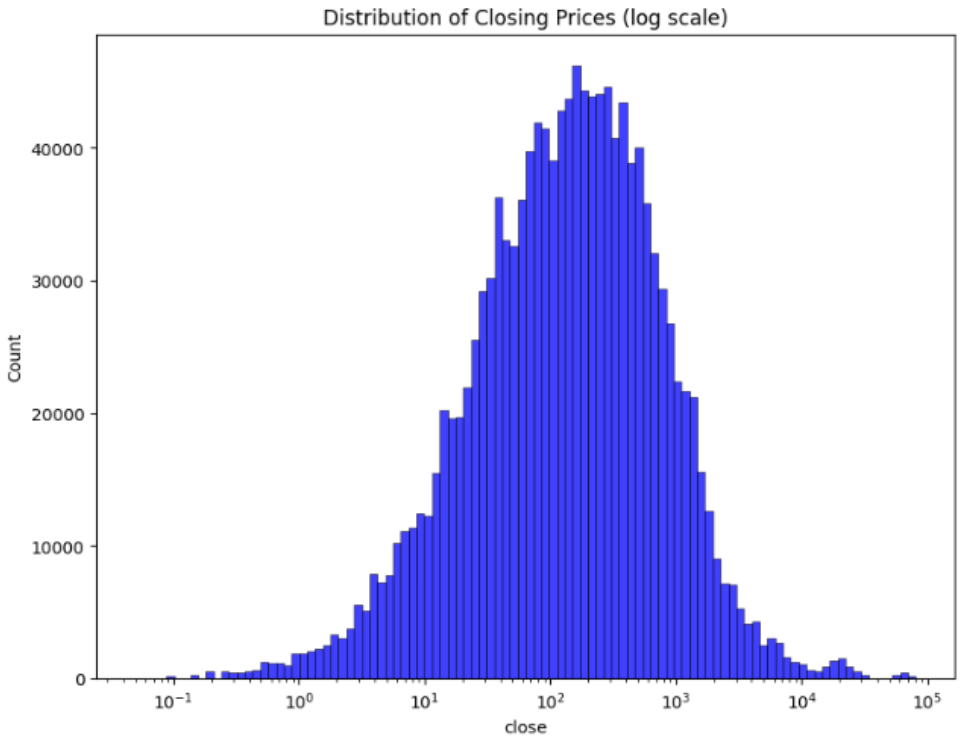


Fig:4 Folder structure in VS code

Exploratory Data Analysis (EDA) – Plots

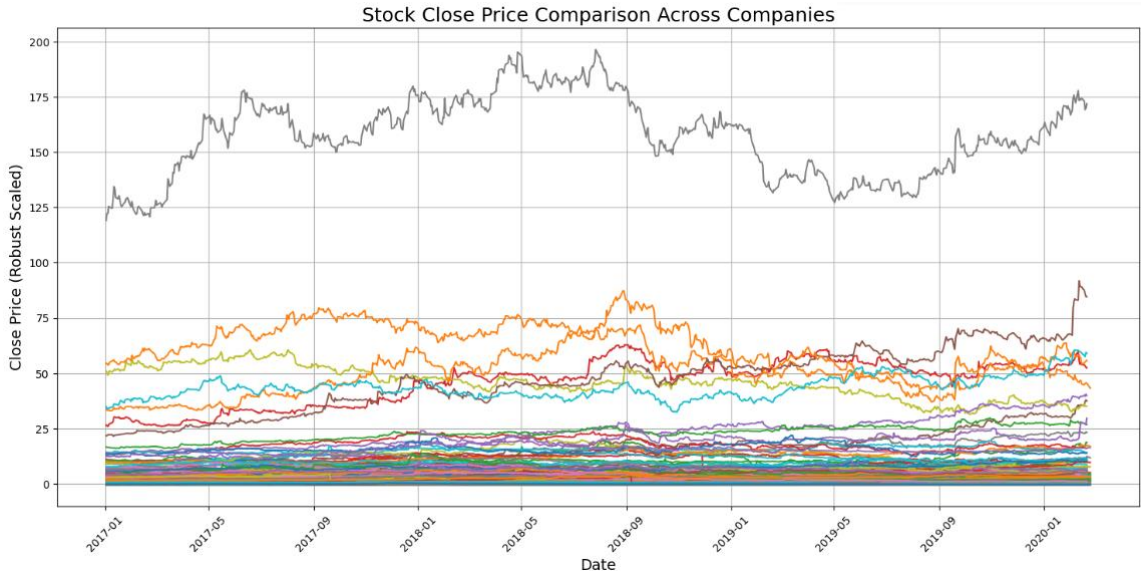
The following figures illustrate the key EDA outputs referenced in the report

A1. Histogram of Close price



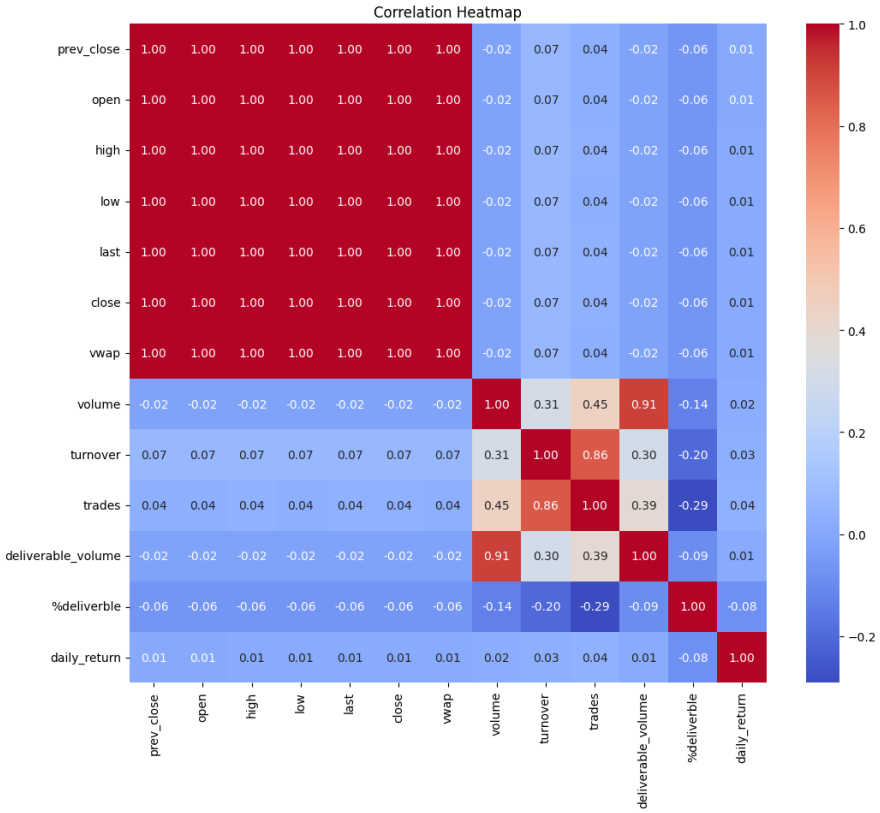
Interpretation: The distribution helps to understand the overall price spread

A2. Stock Close Price Comparison across companies



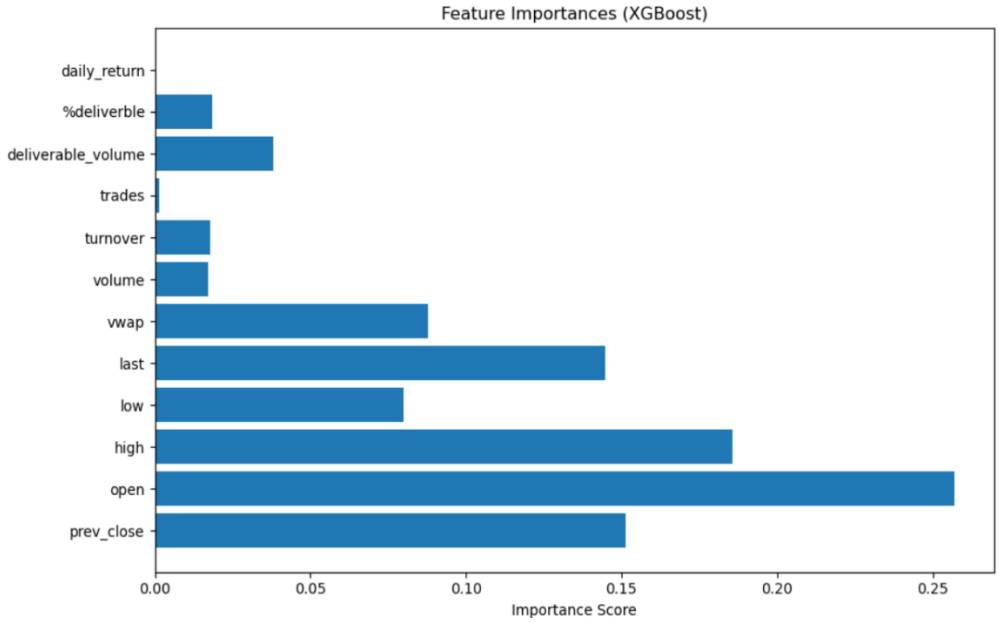
Interpretation : Graph shows differences in volatility and growth speed between companies.

A3. Correlation Heatmap



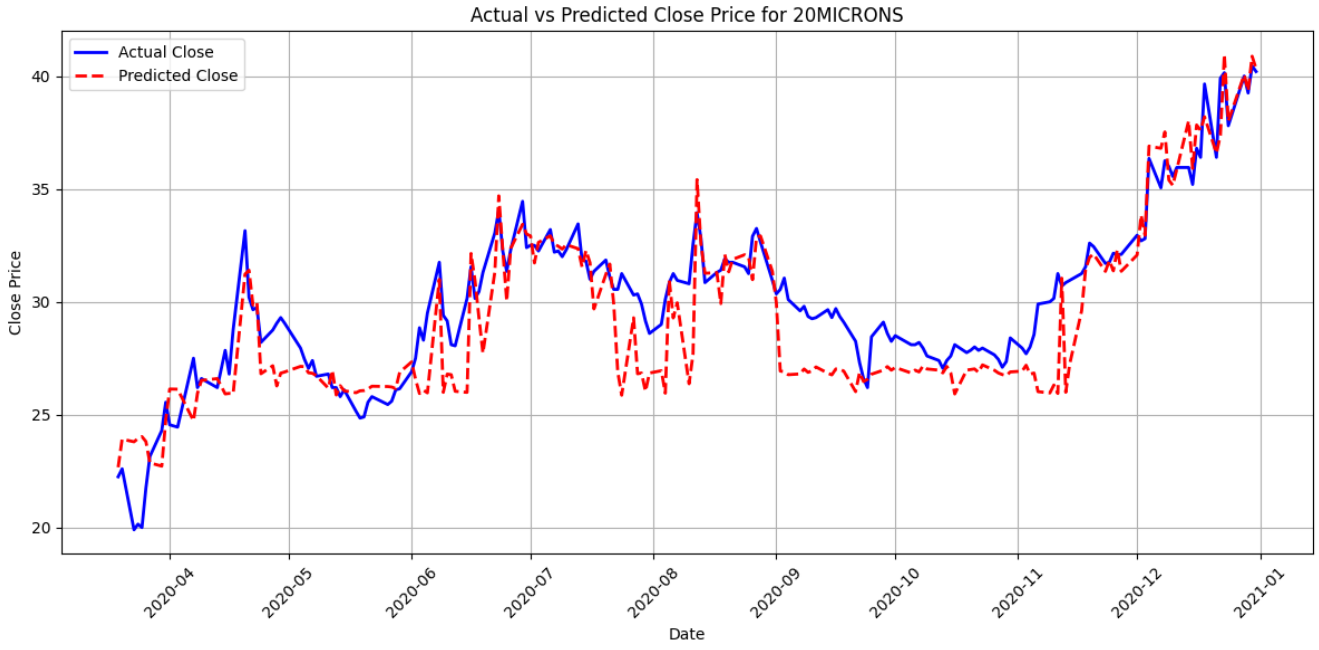
Interpretation: Highlights the strength of relationships between features, revealing strong correlation among open, close, and volume prices.

A4. Feature Importance



Interpretation: Shows which variables contributed the most to predicting stock prices.

A5. Actual Vs Predicted Close Price for 20MICRONS



Interpretation: Graph compares the actual stock closing prices with the predicted values from model.

10.1 APPENDIX-B PROGRAMS AND CODE LISTINGS

B1. Python Codes

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from xgboost import XGBRegressor
import warnings
import joblib
import os

warnings.filterwarnings("ignore")


DATA_PATH = "C:/Users/dhany/OneDrive/Desktop/ict
project/nse_cleaned.csv"
```

Loading cleaned data

```
df = pd.read_csv(DATA_PATH)
```

Preprocessing

```
df.columns = df.columns.str.strip()  # Remove spaces in column names
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df = df.sort_values(by=['symbol', 'date'])

numeric_cols = ['open', 'high', 'low', 'close', 'volume',
'deliverable_volume']
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        df[col] = df[col].fillna(method='ffill')
```

Feature Engineering

```
df['MA5'] = df.groupby('symbol')['close'].transform(lambda x:
x.rolling(window=5).mean())
df['MA10'] = df.groupby('symbol')['close'].transform(lambda x:
x.rolling(window=10).mean())
df = df.dropna()

# ===== SYMBOL SELECTION =====
symbols = df['symbol'].unique().tolist()
print("Available Symbols:", symbols)
selected_symbol = input("Enter a symbol from above:
```

```
").strip().upper()

if selected_symbol not in symbols:
    raise ValueError("Invalid symbol selected!")

stock_df = df[df['symbol'] == selected_symbol].copy()
```

Model Training

```
features = ['open', 'high', 'low', 'volume', 'deliverable_volume',
'MA5', 'MA10']
target = 'close'

X = stock_df[features]
y = stock_df[target]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, shuffle=False
)

model = XGBRegressor(objective='reg:squarederror', n_estimators=100)
model.fit(X_train, y_train)
```

Saving Model

```
model_filename = f"xgb_model_{selected_symbol}.pkl"
scaler_filename = f"scaler_{selected_symbol}.pkl"

joblib.dump(model, model_filename)
joblib.dump(scaler, scaler_filename)

print(f"\n✅ Model saved as {model_filename}")
print(f"✅ Scaler saved as {scaler_filename}")
```

Prediction

```
y_pred = model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"\n📊 Model Performance for {selected_symbol}:")
print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
```

```
latest_features = X_scaled[-1].reshape(1, -1)
predicted_close = model.predict(latest_features)[0]
actual_close = y.iloc[-1]

print(f"\n📅 Latest Date: {stock_df['date'].iloc[-1].strftime('%Y-%m-%d')}")
print(f"✅ Actual Close: {actual_close:.2f}")
print(f"🔮 Predicted Next Close: {predicted_close:.2f}")
```

B2. FLASK Web Application- App.py

```
import os
import pandas as pd
from flask import Flask, jsonify, request, render_template
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error

# ===== CONFIG =====
DATA_PATH = "C:\\Users\\ranji\\OneDrive\\Desktop\\12STOCK
PREDICT\\nse_cleaned.csv"

# ===== INITIALIZE FLASK =====
app = Flask(__name__)
data_loaded = False
available_symbols = []
stock_data = {}

# ===== LOAD DATA =====
def load_data():
    global available_symbols, data_loaded, stock_data
    if not os.path.exists(DATA_PATH):
        print(f"❌ CSV not found: {DATA_PATH}")
        return False
    try:
        df = pd.read_csv(DATA_PATH)
        if "symbol" not in df.columns:
            print("❌ 'symbol' column missing in CSV")
            return False
        # Normalize symbols to uppercase
        df["symbol"] = df["symbol"].str.upper()
        available_symbols = sorted(df["symbol"].unique().tolist())
        stock_data = {sym: df[df["symbol"] ==
sym].reset_index(drop=True) for sym in available_symbols}
        data_loaded = True
        print(f"✅ Data loaded for {len(available_symbols)}
symbols")
        return True
    except Exception as e:
        print(f"❌ Error loading data: {e}")
        return False

data_loaded = load_data()

# ===== ROUTES =====
@app.route("/")
def index():
    return render_template("welcome.html")
```

```
@app.route("/dashboard")
def dashboard():
    return render_template("prediction.html")

@app.route("/api/status")
def api_status():
    return jsonify({
        "status": "running" if data_loaded else "error",
        "data_loaded": data_loaded,
        "available_symbols": len(available_symbols)
    })

@app.route("/api/symbols")
def api_symbols():
    return jsonify({"symbols": available_symbols})

@app.route("/api/predict")
def api_predict():
    symbol = request.args.get("symbol", "").upper()
    if not symbol:
        return jsonify({"error": "Symbol parameter is missing"}),
400
    if symbol not in stock_data:
        return jsonify({"error": f"Symbol '{symbol}' not found"}),
400

    df = stock_data[symbol].copy()
    if len(df) < 30:
        return jsonify({"error": f"Not enough data for {symbol}
(need >=30 rows)"}), 400

    # Features: OHLC, volume, moving averages
    df["MA5"] = df["close"].rolling(5).mean()
    df["MA10"] = df["close"].rolling(10).mean()
    df.dropna(inplace=True)

    required_cols = ["open", "high", "low", "volume", "MA5", "MA10"]
    if not all(col in df.columns for col in required_cols):
        return jsonify({"error": f"Missing required columns in CSV
for {symbol}"}), 400

    X = df[required_cols]
    y = df["close"]

    train_size = int(len(X) * 0.8)
    X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
    y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]

    # Train model
    model = XGBRegressor(objective="reg:squarederror",
n_estimators=100)
```

```
    model.fit(X_train, y_train)

    # Predictions
    preds = model.predict(X_test)
    r2 = r2_score(y_test, preds)

    # RMSE calculation compatible with all scikit-learn versions
    try:
        rmse = mean_squared_error(y_test, preds, squared=False)
    except TypeError:
        rmse = mean_squared_error(y_test, preds) ** 0.5

    mae = mean_absolute_error(y_test, preds)

    # Next day prediction
    last_row = X.iloc[[-1]]
    next_day_pred = float(model.predict(last_row)[0])

    # Convert dates to string
    dates = df.iloc[train_size:].index.astype(str).tolist()

    return jsonify({
        "symbol": symbol,
        "dates": dates,
        "actual": y_test.tolist(),
        "predicted": preds.tolist(),
        "r2": r2,
        "rmse": rmse,
        "mae": mae,
        "next_day_prediction": next_day_pred
    })

# ===== MAIN =====
if __name__ == "__main__":
    print("🚀 Starting Stock Prediction API...")
    app.run(debug=True, host="0.0.0.0", port=5000)
```

B3. HTML Template – welcome.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1" />
  <title>Welcome • Stock Prediction</title>
  <!-- Tailwind (optional). Content still shows even if this fails.
-->
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
    /* Fallback so text is visible even if Tailwind fails */
    html, body { height: 100%; }
    body { margin: 0; background:#0f172a; color:#e5e7eb; font-
family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto,
Ubuntu, 'Helvetica Neue', Arial, 'Noto Sans', 'Apple Color Emoji',
'Segoe UI Emoji'; }
    /* Background canvas sits BEHIND content and cannot block clicks
*/
    #bg { position: fixed; inset: 0; z-index: -1; pointer-events:
none; }
    .btn { display:inline-block; padding:.75rem 1rem; border-
radius:.5rem; text-decoration:none; }
  </style>
</head>
<body>
  <!-- Background (behind content) -->
  <canvas id="bg"></canvas>

  <!-- Page content -->
  <header class="px-6 py-4 flex items-center justify-between max-w-
6xl mx-auto">
    <div class="text-lg font-semibold">📈 Stock Predictor</div>
    <nav class="text-sm opacity-80">Welcome</nav>
  </header>

  <main class="max-w-6xl mx-auto px-6 py-12">
    <section class="grid md:grid-cols-2 gap-10 items-center">
      <div>
        <h1 class="text-4xl md:text-5xl font-extrabold leading-
tight">
          Predict the <span class="text-emerald-400">Future</span>
of Stocks
        </h1>
        <p class="mt-4 text-slate-300">
          Interactive ML dashboard powered by XGBoost. Train a model
```

```
for any symbol in your dataset and
          visualize predicted vs actual prices.
        </p>

        <div class="mt-6 flex gap-3 items-center">
          <a href="/dashboard" class="btn bg-emerald-500 hover:bg-
emerald-400 text-slate-900 font-semibold">Start Predicting</a>
          <a href="/dashboard" class="btn border border-slate-600
hover:bg-white/10 text-slate-200">Open Dashboard</a>
        </div>

        <div id="apiBox" class="mt-6 text-sm">
          <span class="opacity-75">API Status:</span>
          <span id="apiStatus" class="font-
semibold">Checking…</span>
        </div>
      </div>

      <div class="rounded-2xl border border-slate-700/60 p-6">
        <h2 class="text-xl font-semibold mb-3">How it works</h2>
        <ul class="list-disc pl-5 space-y-2 text-slate-300">
          <li>Load your CSV once and preprocess features (OHLC,
volume, MA5, MA10).</li>
          <li>Train per-symbol XGBoost models and save to
<code>models/</code>.</li>
          <li>Predict the last 30 days and compute metrics (R²,
RMSE, MAE).</li>
        </ul>
        <p class="mt-4 text-xs opacity-70">Tip: open this page via
<code>http://127.0.0.1:5000/</code>, not as a file.</p>
      </div>
    </section>
  </main>

  <footer class="text-center text-xs opacity-70 py-10">Built with ❤ —
Stock Prediction Demo</footer>

  <script>
    // Simple, lightweight background particles (no libraries)
    (function () {
      const c = document.getElementById('bg');
      const ctx = c.getContext('2d');
      const DPR = window.devicePixelRatio || 1;
      let w, h, particles;

      function resize() {
        w = c.width = innerWidth * DPR;
        h = c.height = innerHeight * DPR;
        c.style.width = innerWidth + 'px';
        c.style.height = innerHeight + 'px';
        ctx.setTransform(DPR, 0, 0, DPR, 0, 0);
```

```
      }

      function initParticles(n = 60) {
        particles = Array.from({ length: n }, () => ({
          x: Math.random() * innerWidth,
          y: Math.random() * innerHeight,
          vx: (Math.random() - 0.5) * 0.6,
          vy: (Math.random() - 0.5) * 0.6
        }));
      }

      function tick() {
        ctx.clearRect(0, 0, innerWidth, innerHeight);
        ctx.globalAlpha = 0.7;
        particles.forEach(p => {
          p.x += p.vx; p.y += p.vy;
          if (p.x < 0 || p.x > innerWidth) p.vx *= -1;
          if (p.y < 0 || p.y > innerHeight) p.vy *= -1;
          ctx.beginPath();
          ctx.arc(p.x, p.y, 1.6, 0, Math.PI * 2);
          ctx.fillStyle = '#9ca3af';
          ctx.fill();
        });
        requestAnimationFrame(tick);
      }

      window.addEventListener('resize', resize);
      resize();
      initParticles();
      tick();
    })();

    // API status check (works whether served via Flask or opened
directly)
    (async function () {
      const statusEl = document.getElementById('apiStatus');
      let base = '';
      if (location.port === '5000') {
        base = location.origin; // served by Flask
      } else {
        base = 'http://127.0.0.1:5000';
      }
      try {
        const r = await fetch(base + '/api/status');
        const j = await r.json();
        if (r.ok) {
          statusEl.textContent = `Running • ${j.available_symbols}
symbols`;
        } else {
          statusEl.textContent = 'Error';
        }
      } catch (e) {
```

```
        statusEl.textContent = 'Backend unreachable';
      }
    })();
  </script>
</body>
</html>
```

B4.HTML Template- Prediction.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1" />
  <title>Stock Market Prediction Dashboard</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-slate-900 text-white">
  <header class="p-4 text-lg font-bold">📊 Stock Market Prediction
Dashboard</header>

  <main class="p-6 grid md:grid-cols-3 gap-6">
    <!-- Left -->
    <div>
      <label class="block text-sm mb-1">Stock Symbol</label>
      <input id="stockSymbol" type="text" placeholder="e.g. TCS,
INFY" class="p-2 w-full text-black rounded" />
      <div class="flex gap-2 mt-3">
        <button id="predictBtn" class="bg-green-500 px-4 py-2
rounded">Predict</button>
        <button id="resetBtn" class="bg-gray-500 px-4 py-2
rounded">Reset</button>
      </div>
      <button id="trainBtn" class="bg-blue-500 px-4 py-2 mt-3
rounded w-full">Train Model</button>
      <div class="mt-4">
        <button id="loadSymbolsBtn" class="bg-gray-700 px-3 py-1
rounded">Load Symbols</button>
        <ul id="symbolsList" class="mt-2 text-sm opacity-80"></ul>
      </div>
    </div>

    <!-- Middle -->
    <div>
      <h3 class="mb-2">Forecast Chart</h3>
      <canvas id="forecastChart" class="bg-white rounded"></canvas>
    </div>
```

```
    <!-- Right -->
    <div>
      <h3 class="mb-2">Model Metrics</h3>
      <p>R² Score: <span id="r2Score">—</span></p>
      <p>RMSE: <span id="rmse">—</span></p>
      <p>MAE: <span id="mae">—</span></p>
      <p>Next Day Prediction: <span id="nextPrediction">—</span></p>
      <p class="mt-4">API Status: <span id="apiStatus" class="font-
bold">Checking…</span></p>
    </div>
  </main>

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script>
    const apiStatusEl = document.getElementById('apiStatus');
    const r2ScoreEl = document.getElementById('r2Score');
    const rmseEl = document.getElementById('rmse');
    const maeEl = document.getElementById('mae');
    const nextPredEl = document.getElementById('nextPrediction');
    const forecastChartEl =
document.getElementById('forecastChart');
    let chart;

    async function checkApiStatus() {
      try {
        const res = await fetch('/api/status');
        const data = await res.json();
        apiStatusEl.textContent = `Running —
${data.available_symbols} symbols`;
      } catch {
        apiStatusEl.textContent = 'Error connecting to API';
      }
    }

    async function predictStock() {
      const symbol =
document.getElementById('stockSymbol').value.trim();
      if (!symbol) return alert('Enter a stock symbol');
      try {
        const res = await fetch(`/api/predict?symbol=${symbol}`);
        const data = await res.json();

        // Update metrics
        r2ScoreEl.textContent = data.r2 !== undefined ?
data.r2.toFixed(4) : '—';
        rmseEl.textContent = data.rmse !== undefined ?
data.rmse.toFixed(4) : '—';
        maeEl.textContent = data.mae !== undefined ?
data.mae.toFixed(4) : '—';
        nextPredEl.textContent = data.next_day_prediction !==
undefined ? data.next_day_prediction.toFixed(2) : '—';
```

```
        // Update chart
        const labels = data.dates || [];
        const actual = data.actual || [];
        const predicted = data.predicted || [];
        if (chart) chart.destroy();
        chart = new Chart(forecastChartEl, {
          type: 'line',
          data: {
            labels,
            datasets: [
              { label: 'Actual', data: actual, borderColor: 'blue',
fill: false },
              { label: 'Predicted', data: predicted, borderColor:
'green', fill: false }
            ]
          }
        });
      } catch {
        alert('Error fetching prediction');
      }
    }

    async function loadSymbols() {
      try {
        const res = await fetch('/api/symbols');
        const data = await res.json();
        const list = document.getElementById('symbolsList');
        list.innerHTML = '';
        data.symbols.forEach(sym => {
          const li = document.createElement('li');
          li.textContent = sym;
          list.appendChild(li);
        });
      } catch {
        alert('Error loading symbols');
      }
    }

    document.getElementById('predictBtn').addEventListener('click',
predictStock);
    document.getElementById('resetBtn').addEventListener('click', ()
=> {
      document.getElementById('stockSymbol').value = '';
      if (chart) chart.destroy();
    });
    document.getElementById('trainBtn').addEventListener('click', ()
=> alert('Train model endpoint not implemented yet'));
    document.getElementById('loadSymbolsBtn').addEventListener('clic
k', loadSymbols);

    // Init
```

```
    checkApiStatus();
  </script>
</body>
</html>
```

B5. Requirements.txt

```
flask==2.3.3
flask-cors==4.0.0
pandas==2.2.2
numpy==1.26.4
scikit-learn==1.5.1
xgboost==2.0.3
joblib==1.4.2
```

11. REFERENCES

1. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System.
2. Yahoo Finance API Documentation.
3. Scikit-learn Official Documentation.
4. Kaggle Datasets – Stock Market Historical Data.