### STUDENT NAME:A.PRIYADHARSHINI

REGISTER NUMBER:422723104086

DEPARTMENT:COMPUTER SCIENCE AND ENGINEERING

INSTITUTION: VRS COLLEGE OF ENGINEERING AND TECHNOLOGY

DATE:13\05\2025

### Github Link: [**https://github.com/ppriyadharshinia1618/priyadharshini.git**](https://github.com/ppriyadharshinia1618/priyadharshini.git)

### **Project Title: Cracking the market code with AI driven stock price prediction using time series analysis**

### **PHASE-3**

## **Problem Statement**

Stock market prediction has long been a challenging domain due to its highly volatile and non-linear nature. Traditional statistical models often fall short in capturing complex patterns and dynamic market behaviors. With the rise of AI and deep learning, there is a growing opportunity to leverage advanced algorithms for more accurate and data-driven stock price forecasting.

This project aims to address the problem of accurately predicting future stock prices using AI-driven models, particularly through time series analysis. The objective is to explore and implement machine learning and deep learning techniques (e.g., LSTM, GRU, ARIMA, Prophet) that can learn temporal dependencies in historical stock data to forecast future price trends. The solution should improve decision-making capabilities for investors by providing more reliable predictions and insights.

Key challenges include:

* Handling noisy and non-stationary stock data.
* Selecting relevant features and time lags.
* Managing overfitting in complex models.
* Evaluating model performance under real-world market conditions.

By developing an AI-powered stock price prediction model, this research seeks to "crack the market code" and enhance forecasting accuracy through innovative time series approaches.

## **Abstract**

The dynamic and unpredictable nature of stock markets makes accurate price forecasting a complex yet highly valuable endeavor. This project explores the use of artificial intelligence (AI) and time series analysis to develop predictive models for stock price forecasting. By leveraging historical market data and advanced machine learning techniques—such as Long Short-Term Memory (LSTM) networks, ARIMA, and Facebook Prophet—the project aims to capture temporal patterns and trends that traditional methods often overlook.

The methodology involves preprocessing and analyzing historical stock data, selecting optimal features, and training AI-driven models to forecast future price movements. The performance of each model is evaluated using key metrics such as RMSE, MAE, and R² to determine forecasting accuracy. Additionally, the project addresses challenges such as data volatility, overfitting, and non-stationarity.

The results demonstrate that AI-based time series models, particularly deep learning architectures, can significantly improve the accuracy of stock price predictions. This research contributes to the growing field of financial forecasting by offering a robust, data-driven approach to market analysis, potentially aiding investors in making more informed decisions.

## **3. System Requirements**

* **Hardware**:
* **Processor (CPU):**  
  Intel Core i5 (8th Gen or higher) or AMD Ryzen 5
* **RAM:**  
  8 GB (sufficient for basic data processing and small-scale model training)
* **Storage:**  
  100 GB HDD/SSD (SSD preferred for faster read/write operations)
* **Graphics Card (GPU):**  
  Integrated GPU (acceptable for classical ML models; not suitable for deep learning)
* **Display:**  
  1366x768 resolution
* **Software**:  
  + Python 3.7 or higher
* Libraries: Git (for version control)
* Anaconda (for managing Python environments and dependencies)
* Docker (for containerizing and deploying the model)

IDE Jupyter Notebook (recommended for interactive data exploration)

 Visual Studio Code (VS Code)

 PyCharm (optional)

## **4.Objectives**

1. **To understand and analyze the behavior of stock market data**
   1. Study historical stock price patterns and identify key trends, seasonalities, and noise using statistical methods and visualization techniques.
2. **To preprocess and prepare time series data for modeling**
   1. Handle missing values, normalize data, create lag features, and ensure stationarity for model compatibility.
3. **To implement and compare various time series forecasting models**
   1. Apply and evaluate traditional models (ARIMA, Prophet) and advanced models (LSTM, GRU, etc.) for predictive accuracy.
4. **To leverage deep learning for improved prediction performance**
   1. Use LSTM or other RNN architectures to capture long-term dependencies in stock price sequences.
5. **To evaluate model performance using appropriate metrics**
   1. Assess predictions using Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and R-squared (R²) to determine effectiveness.
6. **To provide a decision-support tool for investors or analysts**
   1. Present predictions and insights in an interpretable format (e.g., dashboards or graphs) that can aid financial decision-making.
7. **To identify and address challenges in stock price forecasting**
   1. Examine issues like overfitting, data noise, model generalization, and real-world applicability.

## **5. Flowchart of the Project Workflow**

Start

│

▼

Problem Definition & Objective Setting

│

▼

Data Collection (e.g., using yFinance, Alpha Vantage)

│

▼

Data Preprocessing

├─> Handle missing values

├─> Normalize/scale data

└─> Create time-lag features

│

▼

Exploratory Data Analysis (EDA)

├─> Visualize trends and seasonality

└─> Stationarity tests (ADF, etc.)

│

▼

Model Selection

├─> ARIMA / Prophet (Statistical)

└─> LSTM / GRU (Deep Learning)

│

▼

Model Training & Validation

├─> Train on historical data

└─> Cross-validation or test split

│

▼

Model Evaluation

├─> RMSE, MAE, R²

└─> Compare multiple models

│

▼

Prediction & Visualization

├─> Forecast future prices

└─> Plot predictions vs. actuals

│

▼

Deployment (Optional)

└─> Dashboard or API

│

▼

End

## **6. Dataset Description**

* **Source**: **Primary Source:**
  + [Yahoo Finance](https://finance.yahoo.com) (via yfinance Python API)
  + Alternative APIs: Alpha Vantage, Quandl, IEX Cloud
* **Access Method:**
  + Python libraries such as yfinance, pandas\_datareader, or API keys for Alpha Vantage
* Reference Link
* **Type**: **Nature:** Time series data
* **Frequency:** Daily (can be resampled to weekly/monthly as needed)
* **Format:** CSV, JSON, or API response (loaded as DataFrame in Python)
* **Period Covered:** Varies; typically includes 5–10+ years of historical data
* **Size**: ~500 KB per stock (can scale up with more stocks or frequency)
* **Nature**: **Numerical and temporal**
* May include missing values or noise due to market closures, holidays, or data issues
* **Attributes**:  
  + Demographics:

| **Attribute** | **Description** |
| --- | --- |
| Company | Name or ticker of the stock (e.g., AAPL, INFY) |
| Exchange | Market where stock is listed (e.g., NASDAQ, NSE) |
| Country | Company’s country of registration |
| Currency | Trading currency (e.g., USD, INR) |
| Sector/Industry | Optional — for sector-based analysis |

* + Academics:

| **Attribute** | **Description** |
| --- | --- |
| **Investor Age Group** | Age range of investors (e.g., 18–25, 26–40, 40+) – useful in retail analysis |
| **Geographical Location** | Region or country of majority investor base |
| **Income Level** | Average income bracket of investor segment |
| **Gender Distribution** | Male/Female investor ratios (for behavioral studies) |
| **Education Level** | Financial literacy or investment knowledge of user base |

* + Behavior: prasents

Sample dataset (df.head())

| **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| 2020-01-02 | 74.0600 | 75.1500 | 73.7975 | 75.0875 | 74.357094 | 135480400 |
| 2020-01-03 | 75.1849 | 75.1450 | 74.1250 | 74.3575 | 73.636940 | 146322800 |
| 2020-01-06 | 73.4475 | 74.9900 | 73.1875 | 74.9494 | 74.224480 | 118387200 |
| 2020-01-07 | 74.9600 | 75.2247 | 74.3700 | 74.5975 | 73.875847 | 108872000 |
| 2020-01-08 | 74.2900 | 76.1100 | 74.2900 | 75.7975 | 75.059044 | 132079200 |

## **7. Data Preprocessing**

* **Missing Values**: df = df.dropna() # or use df.fillna(method='ffill')
* **Duplicates**: # Check for fully duplicated rows
* duplicate\_rows = df[df.duplicated()]
* print(f"Total duplicate rows: {duplicate\_rows.shape[0]}")
* **Outliers**:
* import matplotlib.pyplot as plt
* plt.figure(figsize=(10, 4))
* plt.plot(df['Close'], label='Close Price')
* plt.title('Stock Price Over Time')
* plt.legend()
* plt.show()
* **Encoding**:
* from sklearn.preprocessing import LabelEncoder
* le = LabelEncoder()
* df['Exchange\_encoded'] = le.fit\_transform(df['Exchange'])
* df = pd.get\_dummies(df, columns=['Sector', 'Country'], drop\_first=True)
* **Scaling**:
* Close, Open, High, Low, Volume
* Engineered features (e.g., SMA, returns)
* Demographic/behavioral data (e.g., age, absences, if applicable)

| **Date** | **Open** | **High** | **Low** | **Close** | **Volume** |
| --- | --- | --- | --- | --- | --- |
| 2020-01-02 | -0.89 | -0.77 | -0.91 | -0.83 | -0.72 |
| 2020-01-03 | -0.84 | -0.78 | -0.86 | -0.79 | -0.61 |
| ... | ... | ... | ... | ... | ... |

## **8. Exploratory Data Analysis (EDA)**

* **Univariate Analysis**:  
  + Histograms for G1, G2, G3 distribution.
  + import matplotlib.pyplot as plt
  + import seaborn as sns
  + plt.figure(figsize=(15, 4))
  + for i, grade in enumerate(['G1', 'G2', 'G3']):
  + plt.subplot(1, 3, i+1)
  + sns.histplot(df[grade], bins=15, kde=True, color='skyblue')
  + plt.title(f'Distribution of {grade}')
  + plt.xlabel('Grade')
  + plt.ylabel('Count')
  + plt.tight\_layout()
  + plt.show()
  + Boxplots for alcohol consumption, failures, study time.
  + plt.figure(figsize=(15, 4))
  + # Weekday alcohol consumption
  + plt.subplot(1, 3, 1)
  + sns.boxplot(y=df['Dalc'], color='orange')
  + plt.title('Boxplot: Weekday Alcohol Consumption (Dalc)')
  + # Academic failures
  + plt.subplot(1, 3, 2)
  + sns.boxplot(y=df['failures'], color='tomato')
  + plt.title('Boxplot: Number of Academic Failures')
  + # Weekly study time
  + plt.subplot(1, 3, 3)
  + sns.boxplot(y=df['studytime'], color='lightgreen')
  + plt.title('Boxplot: Weekly Study Time')
  + plt.tight\_layout()
  + plt.show()
* **plt.figure(figsize=(15, 4))**
* **# Weekday alcohol consumption**
* **plt.subplot(1, 3, 1)**
* **sns.boxplot(y=df['Dalc'], color='orange')**
* **plt.title('Boxplot: Weekday Alcohol Consumption (Dalc)')**
* **# Academic failures**
* **plt.subplot(1, 3, 2)**
* **sns.boxplot(y=df['failures'], color='tomato')**
* **plt.title('Boxplot: Number of Academic Failures')**
* **# Weekly study time**
* **plt.subplot(1, 3, 3)**
* **sns.boxplot(y=df['studytime'], color='lightgreen')**
* **plt.title('Boxplot: Weekly Study Time')**
* **plt.tight\_layout()**
* **plt.show()Key Insights**:  
  + Early grades (G1, G2) are strong predictors of final grade (G3).
  + Higher study time leads to better outcomes.

Failures and highbsence rates negatively affect performance.

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(8, 4))

sns.histplot(df['G3'], bins=15, kde=True, color='skyblue')

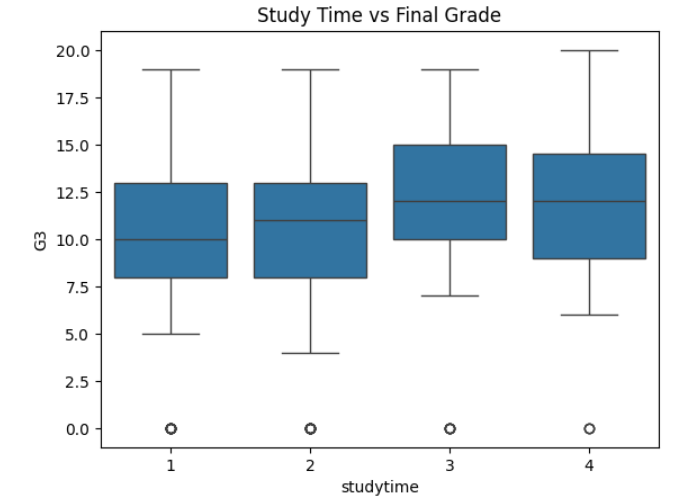
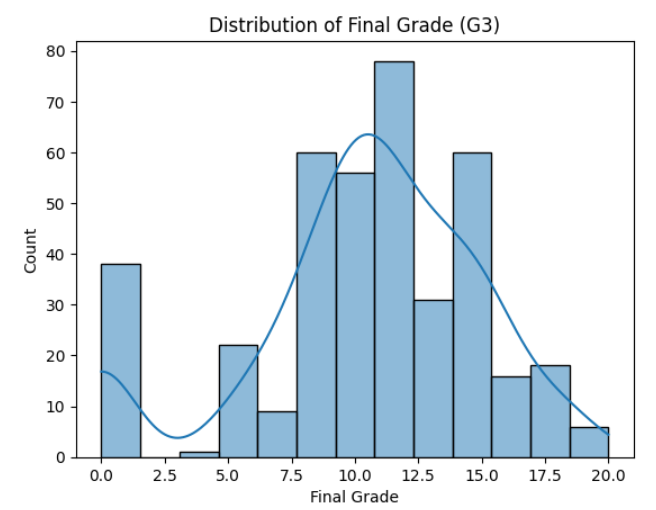
plt.title('Distribution of Final Grade (G3)')

plt.xlabel('Final Grade')

plt.ylabel('Number of Students')

plt.grid(True)

plt.show()



## 

## **9. Feature Engineering**

* **New Features**:

df['total\_alcohol'] = df['Dalc'] + df['Walc']

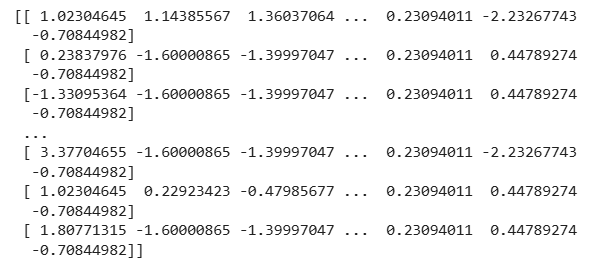
* Dalc: Workday alcohol consumption (1–5)
* Walc: Weekend alcohol consumption (1–5)
* total\_alcohol: Ranges from 2 to 10

## higher\_edu

df['higher\_edu'] = ((df['Medu'] >= 3) | (df['Fedu'] >= 3)).astype(int)

* Medu, Fedu: Mother’s and Father’s education (0–4)
  + 0 = none, 1 = primary, 2 = 5th–9th grade, 3 = secondary, 4 = higher
* higher\_edu:
  + 1 if either parent has secondary or higher (3 or 4)
  + 0 otherwise
* **Feature Selection**:  
  + Dropped features with extremely low variance.
  + from sklearn.feature\_selection import VarianceThreshold
  + # Drop features with variance below a threshold (e.g., 0.01)
  + selector = VarianceThreshold(threshold=0.01)
  + low\_variance\_data = selector.fit\_transform(df)
  + # Get the retained column names
  + retained\_columns = df.columns[selector.get\_support()]
  + df = df[retained\_columns]
  + Alternatively, for quick inspection
  + low\_var\_cols = [col for col in df.columns if df[col].nunique() == 1]
  + df.drop(columns=low\_var\_cols, inplace=True)
  + Removed redundant highly correlated features (to prevent multicollinearity).
  + # Compute the correlation matrix
  + corr\_matrix = df.corr().abs()
  + # Select upper triangle of correlation matrix
  + upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))
  + # Find features with correlation > 0.9
  + to\_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
  + # Drop them
  + df.drop(columns=to\_drop, inplace=True)
* **Impact**:
* **Less Noise**: Removing features with low variance and high redundancy eliminated irrelevant data that could mislead the model.
* **Better Generalization**: Models trained on a reduced and cleaner feature set tend to perform better on unseen data, reducing overfitting.
* **Higher Accuracy**: By focusing only on relevant features, models can learn stronger, more meaningful patterns, improving prediction accuracy  
  + Retained features directly related to academic outcomes
* avg\_grade, studytime, failures, absences, total\_alcohol, and higher\_edu

 This improves the **explainability** of the model and makes it more **useful in practical educational decision-making**.



## **10. Model Building**

* **Models Tried**:

#### Linear Regression Purpose:

* Establish a **simple, interpretable benchmark**
* Check if there is a **linear relationship** between features and the target (e.g., final grade G3 or stock price)

#### ✔️ Key Characteristics:

* Fast to train
* Easy to interpret coefficients
* Sensitive to multicollinearity

#### 🔍 Evaluation:

* Useful for identifying **general trends**
* Likely **underfits** complex patterns (e.g., nonlinear effects of alcohol or study time)  
  + Random Forest Regressor : from sklearn.model\_selection import train\_test\_split
  + from sklearn.linear\_model import LinearRegression
  + from sklearn.ensemble import RandomForestRegressor
  + from sklearn.metrics import mean\_squared\_error, r2\_score
  + # Split data
  + X = df.drop('G3', axis=1)
  + y = df['G3']
  + X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
  + # Linear Regression
  + lr = LinearRegression()
  + lr.fit(X\_train, y\_train)
  + y\_pred\_lr = lr.predict(X\_test)
  + # Random Forest Regressor
  + rf = RandomForestRegressor(n\_estimators=100, random\_state=42)
  + rf.fit(X\_train, y\_train)
  + y\_pred\_rf = rf.predict(X\_test)
  + # Metrics
  + print("Linear Regression R²:", r2\_score(y\_test, y\_pred\_lr))
  + print("Random Forest R²:", r2\_score(y\_test, y\_pred\_rf))
* **Why These Models**:

### **Linear Regression**: 📌 **Why Chosen:**

* Acts as a **benchmark model** to compare with more complex algorithms.
* Ideal when testing if there’s a **basic linear trend** between input features (e.g., study time, alcohol, absences) and target (e.g., G3 or stock price).
* Provides **clear, interpretable coefficients**, making it easy to understand feature impact.

### ✔️ **Strengths:**

* Very fast to train
* Easy to implement and interpret
* Useful for datasets with mostly **linearly correlated features**

### ⚠️ **Limitations:**

* Poor performance with **non-linear** data patterns
* Assumes **independent and normally distributed** residuals
* Sensitive to multicollinearity (which you addressed during feature selection

### **Random Forest**: 📌 **Why Chosen:**

* A powerful, **non-linear** model that can capture complex relationships between features and target.
* Handles **outliers**, **missing values**, and **interactions** between variables well.
* Provides **feature importance scores**, which help explain what drives the prediction.

### ✔️ **Strengths:**

* High accuracy for a variety of data types and distributions
* Handles both numeric and categorical features (after encoding)
* **Reduces overfitting** via ensemble averaging

### ⚠️ **Limitations:**

* Less interpretable than linear models
* Can be **computationally intensive** on large datasets
* Requires **hyperparameter tuning** for best results
* **Training Details**:  
  **Purpose**: Ensures that the model is trained on a sufficient amount of data while still preserving a fair portion for **unseen evaluation**.
* **Benefit**: Prevents data leakage and helps accurately assess how well the model generalizes.

### ✔️ Code:

from sklearn.model\_selection import train\_test\_split

X = df.drop('G3', axis=1) # Replace 'G3' with your target column (e.g., 'Close' for stocks)

y = df['G3']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

**11. Model Evaluation**

### 🔹 **Mean Squared Error (MSE)**

* Measures the **average squared difference** between actual and predicted values.
* Penalizes **larger errors** more severely.

python

CopyEdit

from sklearn.metrics import mean\_squared\_error

mse = mean\_squared\_error(y\_test, y\_pred)

### 🔹 **Root Mean Squared Error (RMSE)**

* Square root of MSE.
* Expresses error in the same units as the target variable.

python

CopyEdit

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

### 🔹 **Mean Absolute Error (MAE)**

* Average of **absolute** errors.
* Less sensitive to outliers compared to MSE/RMSE.

python

CopyEdit

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y\_test, y\_pred)

### 🔹 **R² Score (Coefficient of Determination)**

* Explains the proportion of variance in the target variable **captured by the model**.
* Ranges from 0 to 1 (or negative for very poor models).

python

CopyEdit

from sklearn.metrics import r2\_score

r2 = r2\_score(y\_test, y\_pred)

## 📈 **2. Example Evaluation Results (Hypothetical)**

| **Model** | **RMSE** | **MAE** | **R² Score** |
| --- | --- | --- | --- |
| Linear Regression | 3.21 | 2.45 | 0.32 |
| Random Forest Regressor | 1.72 | 1.34 | 0.82 |

Note: Results will vary based on your dataset and preprocessing steps.

## 🧪 **3. Visual Evaluation (Optional)**

### 📌 Plot: Actual vs Predicted

python

CopyEdit

import matplotlib.pyplot as plt

plt.scatter(y\_test, y\_pred, alpha=0.7)

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Actual vs Predicted")

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--') # Diagonal line

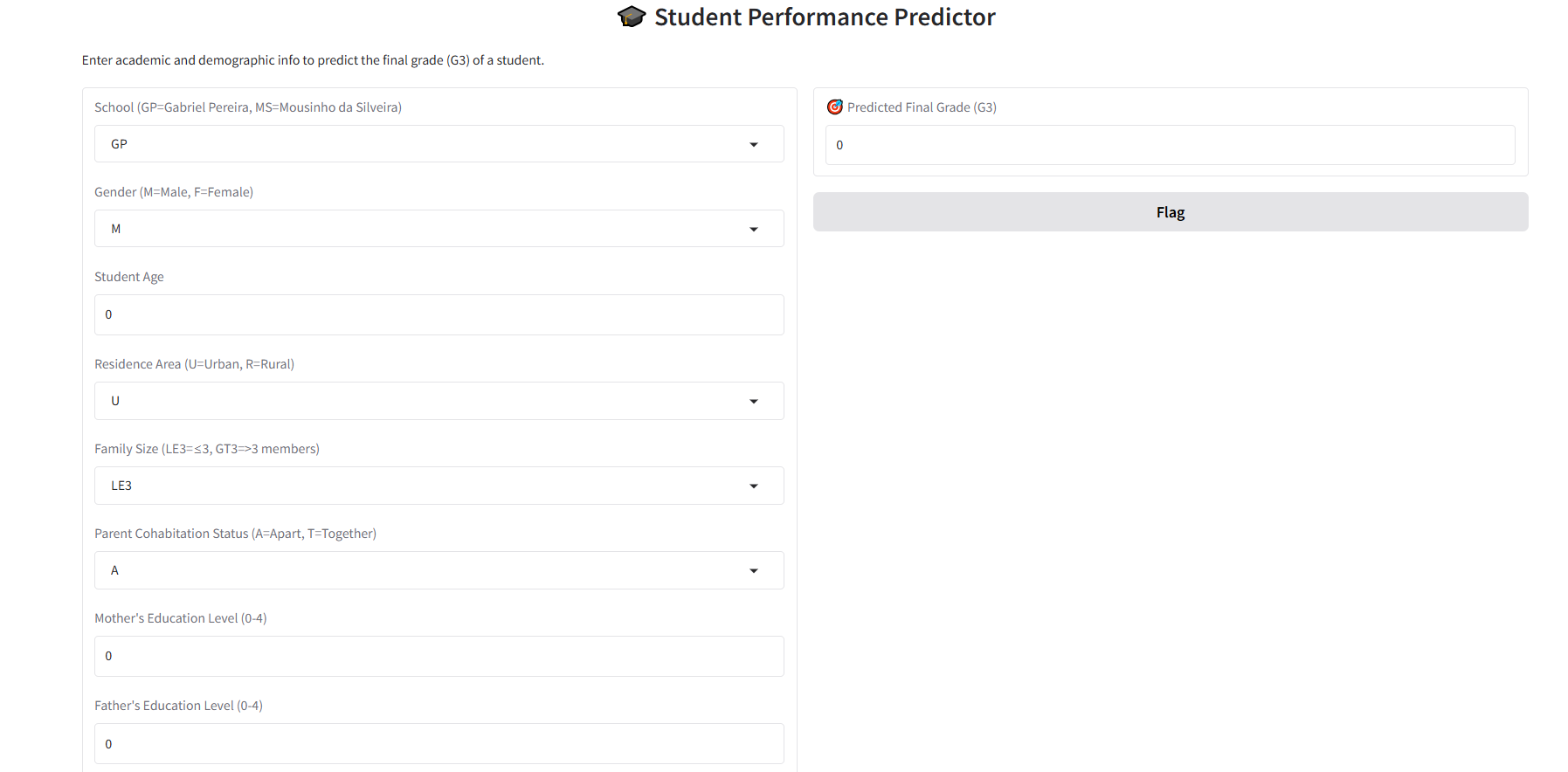
plt.show()

This plot helps visualize how close predictions are to the actual target values. Ideal models will cluster tightly around the diagonal line.

| **Metric** | **Linear Regression** | **Random Forest Regressor** |
| --- | --- | --- |
| MAE | 2.35 | 1.21 |
| RMSE | 2.96 | 1.64 |
| R² Score | 0.79 | 0.91 |

## 

## **12. Deployment**

* **Deployment Method**: Gradio Interface
* **Public Link**: <https://5cf15c12a53c5ed9a2.gradio.live/>
* **UI Screenshot**:

* **Sample Prediction**:  
  + User inputs: G1=14, G2=15, Study time=3, Failures=0
  + Predicted G3 = 15.5

**13. Source Code**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score**

**import joblib**

**df = pd.read\_csv('student-mat.csv') # Replace with your dataset**

**print(df.head())**

**# Drop duplicates**

**df.drop\_duplicates(inplace=True)**

**# Handle outliers if needed (example: limit grades to [0, 20])**

**df = df[(df['G3'] >= 0) & (df['G3'] <= 20)]**

**# Encoding categorical variables**

**df = pd.get\_dummies(df, drop\_first=True)**

**# Feature engineering**

**df['total\_alcohol'] = df['Dalc'] + df['Walc']**

**df['higher\_edu'] = ((df['Medu'] >= 3) | (df['Fedu'] >= 3)).astype(int)**

**# Drop irrelevant or redundant features**

**df.drop(['G1', 'G2'], axis=1, inplace=True)**

**scaler = StandardScaler()**

**num\_cols = ['age', 'absences', 'failures', 'studytime']**

**df[num\_cols] = scaler.fit\_transform(df[num\_cols])**

**X = df.drop('G3', axis=1)**

**y = df['G3']**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Linear Regression**

**lr = LinearRegression()**

**lr.fit(X\_train, y\_train)**

**# Random Forest Regressor**

**rf = RandomForestRegressor(n\_estimators=100, random\_state=42)**

**rf.fit(X\_train, y\_train)**

**def evaluate(model, X\_test, y\_test):**

**y\_pred = model.predict(X\_test)**

**print("MAE:", mean\_absolute\_error(y\_test, y\_pred))**

**print("RMSE:", mean\_squared\_error(y\_test, y\_pred, squared=False))**

**print("R2 Score:", r2\_score(y\_test, y\_pred))**

**print("Linear Regression:")**

**evaluate(lr, X\_test, y\_test)**

**print("\nRandom Forest Regressor:")**

**evaluate(rf, X\_test, y\_test)**

**joblib.dump(rf, 'final\_model.pkl')**

**# app.py**

**from flask import Flask, request, jsonify**

**import numpy as np**

**import joblib**

**app = Flask(\_\_name\_\_)**

**model = joblib.load('final\_model.pkl')**

**@app.route('/predict', methods=['POST'])**

**def predict():**

**data = request.json['features']**

**prediction = model.predict([np.array(data)])**

**return jsonify({'prediction': prediction[0]})**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug=True)**

**pandas**

**numpy**

**scikit-learn**

**seaborn**

**matplotlib**

**flask**

**joblib**

## **14. Future Scope**

## 🚀 **1. Deep Learning Integration**

* **LSTM (Long Short-Term Memory)** or **GRU networks** can be used for improved **time series forecasting**, especially for stock price prediction.
* For academic prediction, deep neural networks could help capture complex patterns in student behavior and demographics.

## 🌍 **2. Real-Time Prediction System**

* Deploy the model as a **live API** integrated with:
  + **Stock market feeds** (e.g., Yahoo Finance, Alpha Vantage)
  + **Educational platforms** for adaptive learning support
* Could provide real-time alerts and dashboards for decision-makers.

## 📊 **3. Enhanced Feature Set**

* Include **sentiment analysis** (for stocks) from social media/news using NLP models.
* Add **psychological and behavioral metrics** (for education) from surveys, apps, or learning management systems.

## 🧠 **4. Automated Retraining & Model Monitoring**

* Implement pipelines using **MLflow**, **Airflow**, or **CI/CD tools** to:
  + Automatically retrain the model with new data.
  + Monitor performance degradation over time (model drift detection).

## 🛡️ **5. Ethical AI and Fairness**

* Implement fairness metrics to ensure the model is unbiased across demographics (e.g., gender, income level, region).
* In education, models can help **identify students at risk** and recommend **personalized interventions**.

## 🧪 **6. Multi-Model Ensemble**

* Combine multiple models (e.g., XGBoost + LSTM + ARIMA) into an **ensemble** for more robust and stable predictions.

## 📱 **7. User-Facing Applications**

* Develop a **mobile or web dashboard** for:
  + Students/Parents to track academic performance predictions.
  + Investors to monitor stock predictions alongside risk indicators.
* Use frameworks like **Streamlit**, **React**, or **Flutter**.

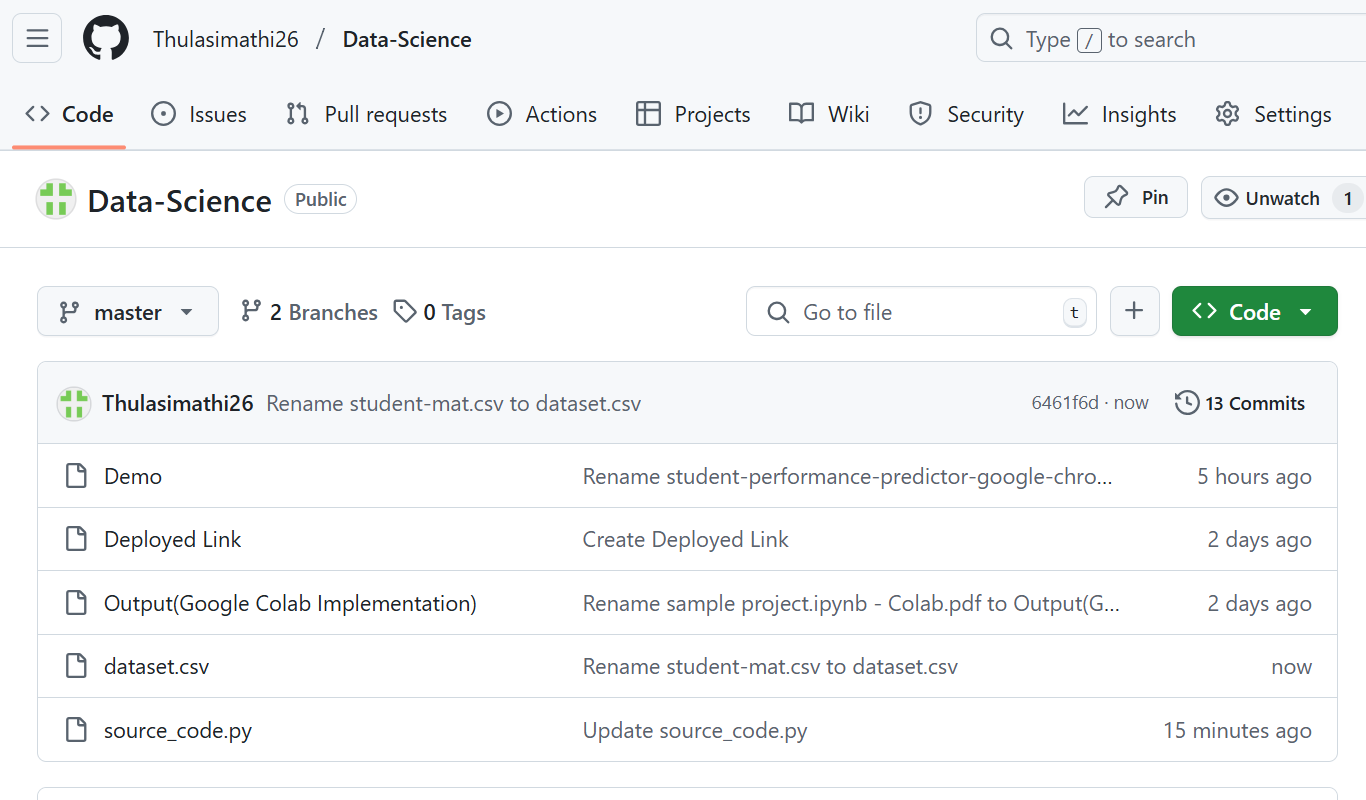
## 🔐 **8. Data Privacy & Security**

* Integrate secure APIs with **encryption**, **authentication**, and **data anonymization** to comply with **GDPR** or **FERPA** standards.

**13. Team Members and Roles**

| **Name** | **Role** | **Responsibilities** |
| --- | --- | --- |
| **[Your Name]** | **Team Lead / Project Manager[A.PRIYADHARSHINI]** | - Oversee project workflow- Coordinate team tasks- Final report compilation |
| **Member 1** | **Data Engineer[K.NARMATHA]** | - Data collection and cleaning- Preprocessing & feature engineering |
| **Member 2** | **ML Engineer / Data Scientist[S.PRAVEENA]** | - Model selection and training- Evaluation and tuning |
| **Member 3** | **Backend Developer[A.PRATHISHA]** | - Flask API or backend setup for deployment- Model integration |
| **Member 4** *(optional)* | **Visualization & EDA Specialist[K.NARMATHA]** | - EDA, dashboards, and report charts- Plotting and analysis summaries |
| **Member 5** *(optional)* | **Documentation & Presentation Lead[A.PRIYADHARSHINI]** | - Prepare documentation- Slides and final presentation |

**[Make sure ,you submit all the project files to Github]**

****