# Ex.No: 1 Regression Model for Real-Time System

Date: 08/07/2024

## Aim:

To compare the performance of Simple Linear Regression, Multiple Linear Regression, and Polynomial Regression models in predicting the GPA of students based on their study time, attendance, and previous scores.

## Procedure:

1. Data Loading and Preparation:

- Load the dataset containing student performance details using pandas.

- Handle missing values by dropping rows with any missing data.

2. Feature and Target Selection:

- Select 'StudyTimeWeekly', 'Attendance', and 'PreviousScores' as features for multiple and polynomial regression.

- Select 'GPA' as the target variable.

3. Data Splitting:

- Split the data into training and testing sets using train\_test\_split with a test size of 20%.

4. Simple Linear Regression:

- Use 'StudyTimeWeekly' as the feature.

- Train a Simple Linear Regression model.

5. Multiple Linear Regression:

- Use 'StudyTimeWeekly', 'Attendance', and 'PreviousScores' as features.

- Train a Multiple Linear Regression model.

6. Polynomial Regression:

- Use polynomial features of 'StudyTimeWeekly', 'Attendance', and 'PreviousScores' with degree=2.

- Train a Polynomial Regression model.

7. Prediction and Evaluation:

- Use the trained models to make predictions on the test set.

- Calculate evaluation metrics (MSE, R2) for each model.

- Determine the best model based on the metrics.

8. Visualization:

- Plot the actual vs. predicted GPA for the first 50 data points for each model.

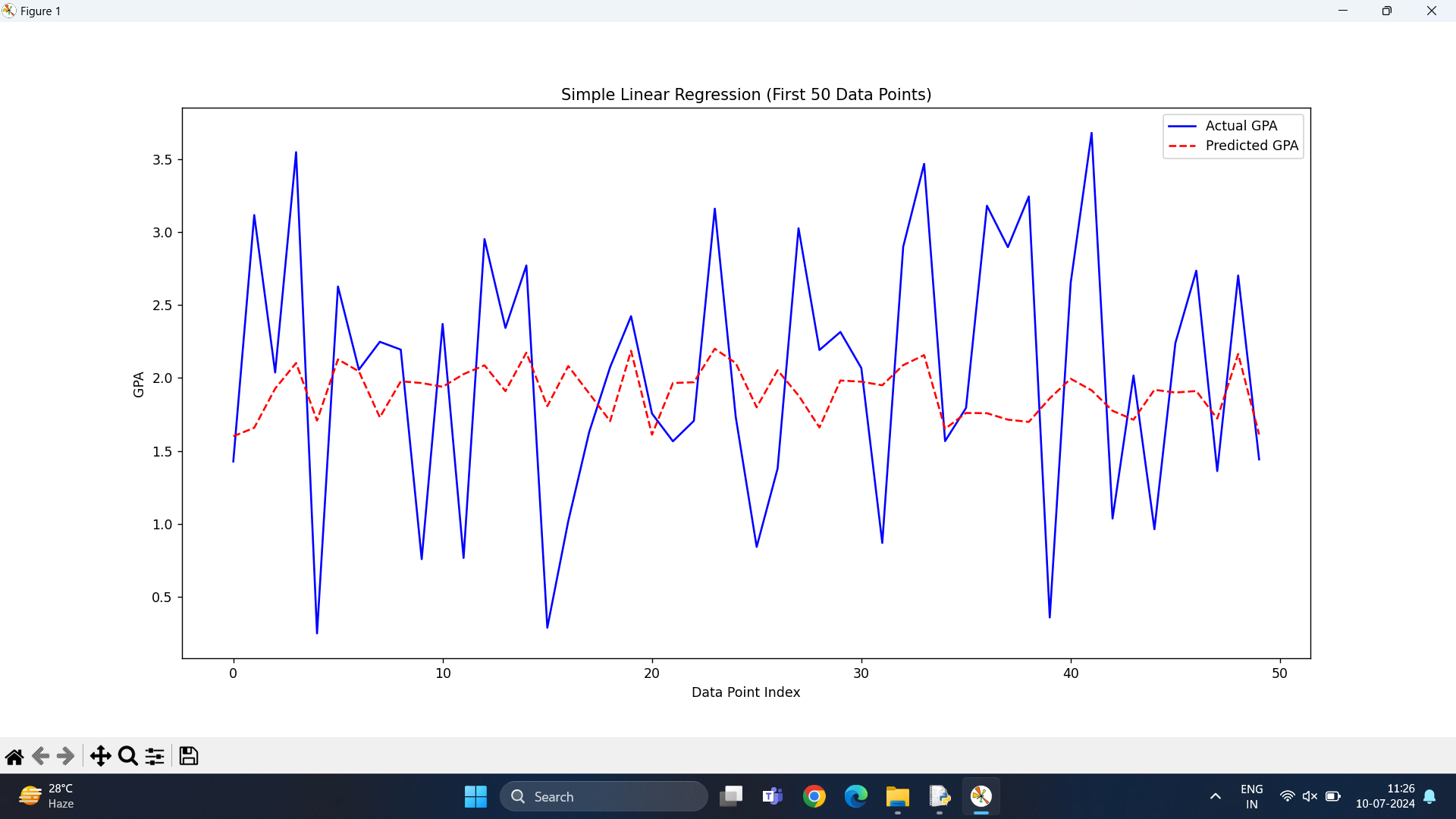
## Code:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
class Layoff:  
   
 def \_\_init\_\_(self):  
 self.data = None  
 self.encoded\_data = None  
   
 def preprocess(self):  
 self.data = pd.read\_csv("C:/Machine learning/Student\_performance\_data \_.csv")  
   
 # Missing values  
 self.data = self.data.dropna()  
   
 # Categorical features  
 categorical\_features = self.data.select\_dtypes(include=['object']).columns  
 self.encoded\_data = pd.get\_dummies(self.data, columns=categorical\_features)  
   
 # Correlation heatmap  
 corr\_matrix = self.encoded\_data.corr()  
 plt.figure(figsize=(14, 10))  
 sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)  
 plt.title('Correlation Heatmap of Student Performance Data')  
 plt.show()  
   
 # Simple Linear Regression  
 self.simple\_linear\_regression()  
   
 # Multiple Linear Regression  
 self.multiple\_linear\_regression()  
   
 # Polynomial Regression  
 self.polynomial\_regression()  
  
 def simple\_linear\_regression(self):  
 feature = 'StudyTimeWeekly'   
 target = 'GPA'  
 X = self.encoded\_data[[feature]]  
 y = self.encoded\_data[target]  
   
 # Training and testing sets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
   
 # Simple Linear Regression model  
 model = LinearRegression()  
 model.fit(X\_train, y\_train)  
   
 # Predict  
 y\_pred = model.predict(X\_test)  
   
 # Plot  
 plt.figure(figsize=(12, 8))  
 plt.plot(range(50), y\_test[:50], color='blue', label='Actual GPA')  
 plt.plot(range(50), y\_pred[:50], color='red', linestyle='dashed', label='Predicted GPA')  
 plt.title('Simple Linear Regression (First 50 Data Points)')  
 plt.xlabel('Data Point Index')  
 plt.ylabel('GPA')  
 plt.legend()  
 plt.show()  
  
 def multiple\_linear\_regression(self):  
 features = ['StudyTimeWeekly', 'Attendance', 'PreviousScores'] # Add more relevant features  
 target = 'GPA'  
 X = self.encoded\_data[features]  
 y = self.encoded\_data[target]  
   
 # Training and testing sets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
   
 # Multiple Linear Regression model  
 model = LinearRegression()  
 model.fit(X\_train, y\_train)  
   
 # Predict  
 y\_pred = model.predict(X\_test)  
   
 # Metrics  
 mse = mean\_squared\_error(y\_test, y\_pred)  
 r2 = r2\_score(y\_test, y\_pred)  
 print(f'Multiple Linear Regression - MSE: {mse}, R2: {r2}')  
   
 # Plot  
 plt.figure(figsize=(12, 8))  
 plt.plot(range(50), y\_test[:50], color='blue', label='Actual GPA')  
 plt.plot(range(50), y\_pred[:50], color='red', linestyle='dashed', label='Predicted GPA')  
 plt.title('Multiple Linear Regression (First 50 Data Points)')  
 plt.xlabel('Data Point Index')  
 plt.ylabel('GPA')  
 plt.legend()  
 plt.show()  
  
 def polynomial\_regression(self):  
 features = ['StudyTimeWeekly', 'Attendance', 'PreviousScores'] # Add more relevant features  
 target = 'GPA'  
 X = self.encoded\_data[features]  
 y = self.encoded\_data[target]  
   
 # Training and testing sets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
   
 # Polynomial Regression model  
 poly\_features = PolynomialFeatures(degree=2)  
 X\_train\_poly = poly\_features.fit\_transform(X\_train)  
 X\_test\_poly = poly\_features.transform(X\_test)  
   
 model = LinearRegression()  
 model.fit(X\_train\_poly, y\_train)  
   
 # Predict  
 y\_pred = model.predict(X\_test\_poly)  
   
 # Metrics  
 mse = mean\_squared\_error(y\_test, y\_pred)  
 r2 = r2\_score(y\_test, y\_pred)  
 print(f'Polynomial Regression - MSE: {mse}, R2: {r2}')  
   
 # Plot  
 plt.figure(figsize=(12, 8))  
 plt.scatter(range(len(y\_test)), y\_test, color='blue', label='Actual GPA')  
 plt.scatter(range(len(y\_test)), y\_pred, color='red', label='Predicted GPA')  
 plt.title('Polynomial Regression (Test Data Points)')  
 plt.xlabel('Data Point Index')  
 plt.ylabel('GPA')  
 plt.legend()  
 plt.show()  
  
def main():  
 l = Layoff()  
 l.preprocess()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

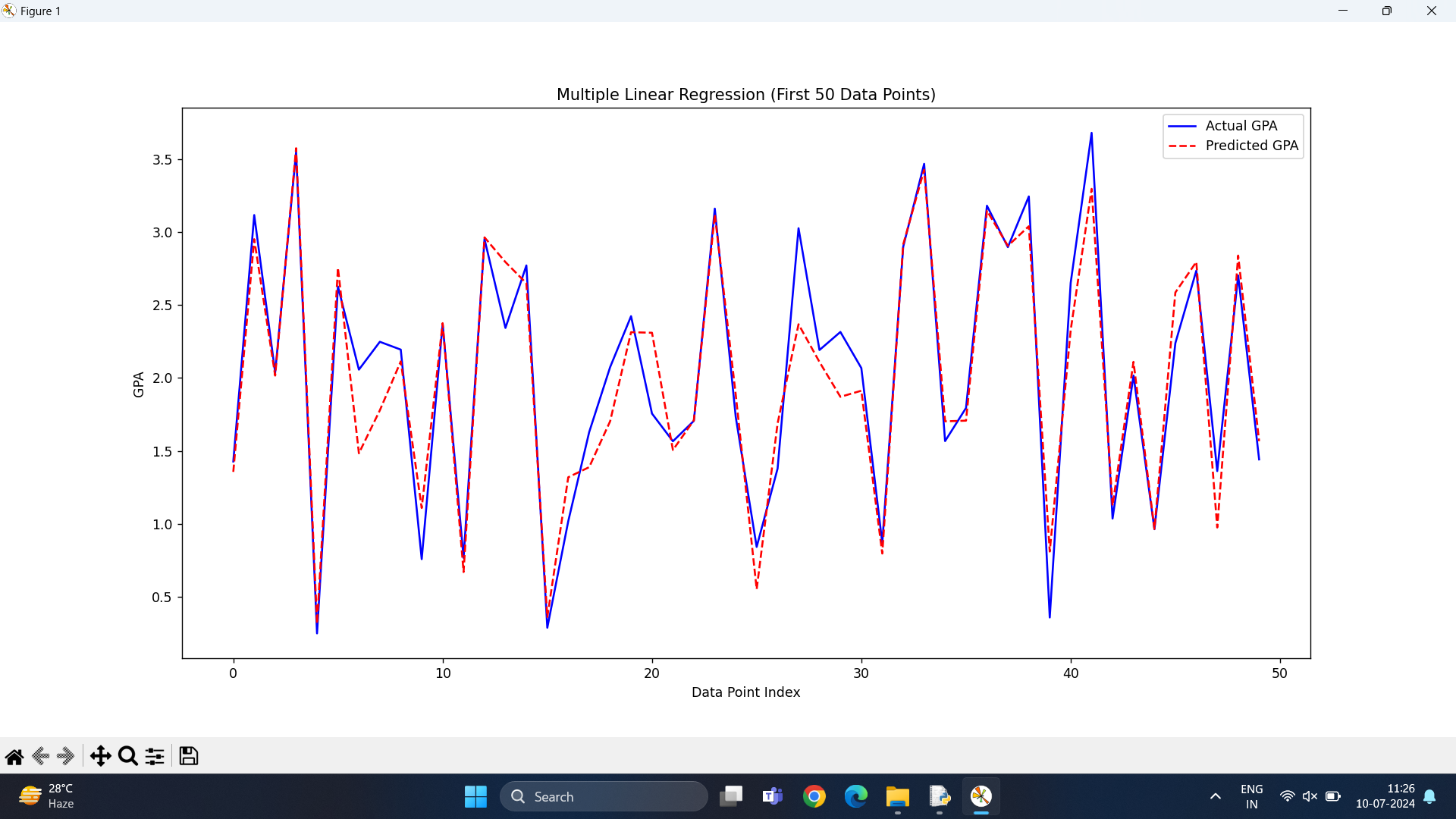
## Final Output:

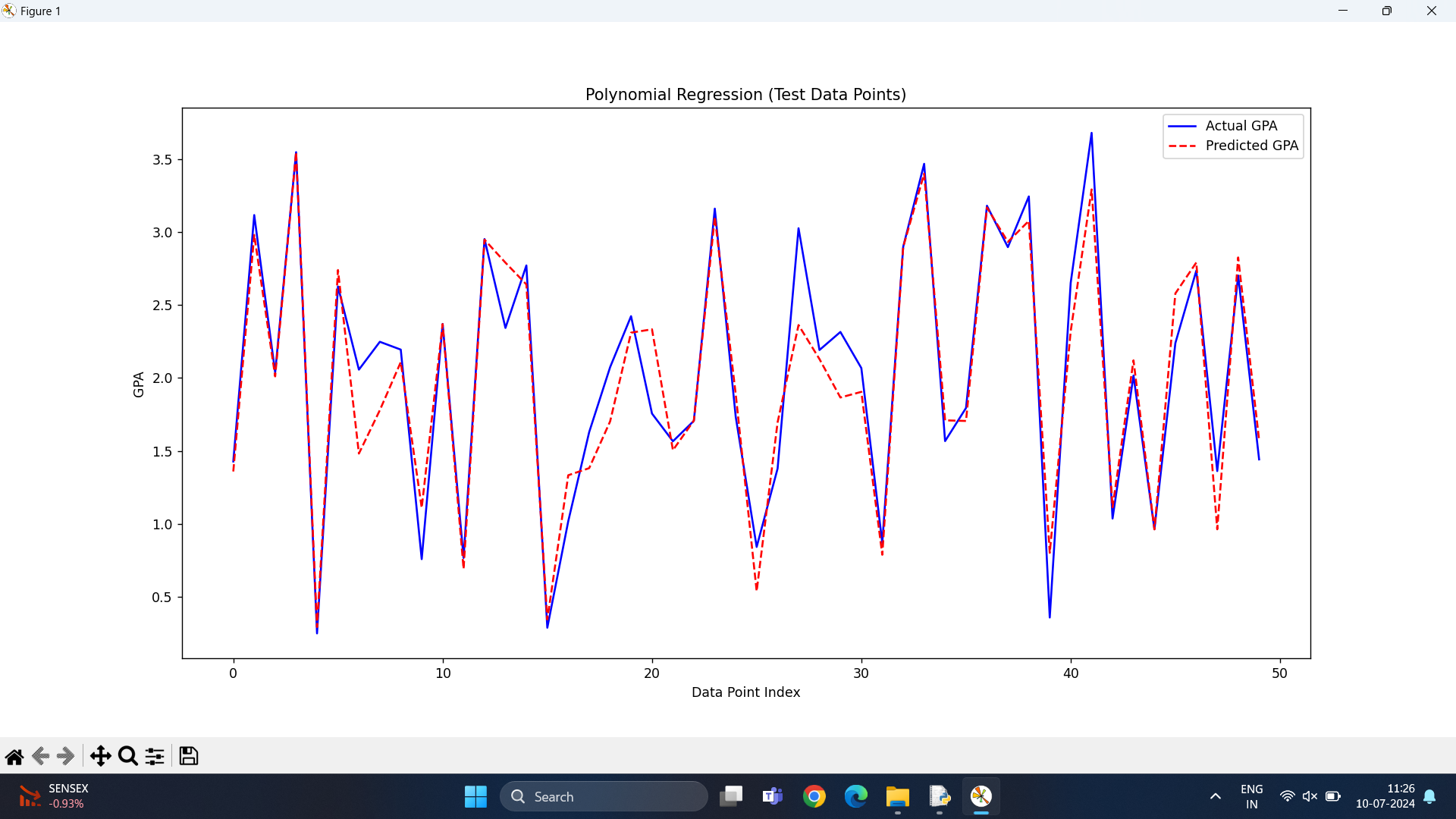
## Plots:

Plot showing actual vs. predicted values for Simple Linear Regression:

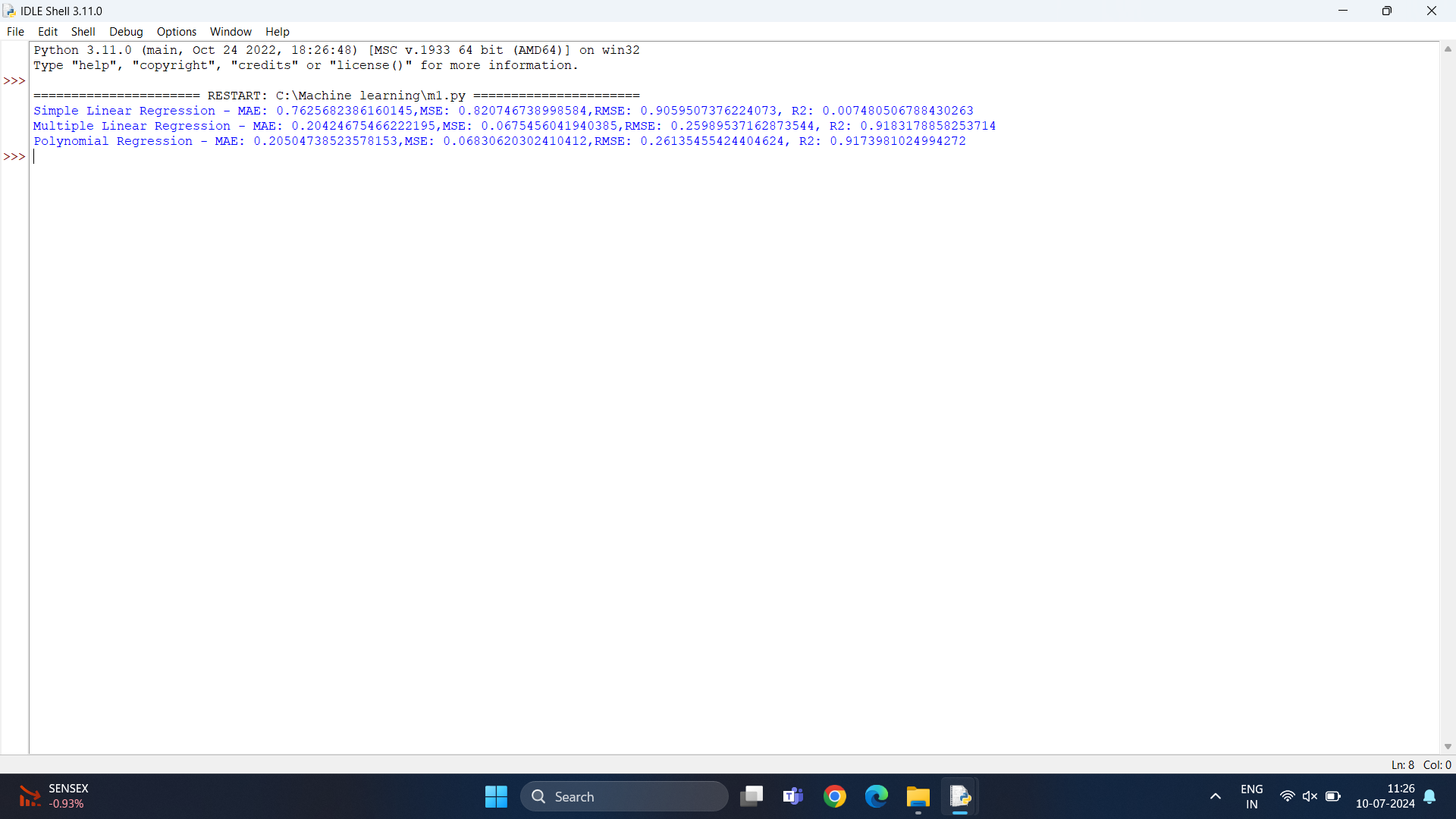


Plot showing actual vs. predicted values for Multiple Linear Regression:

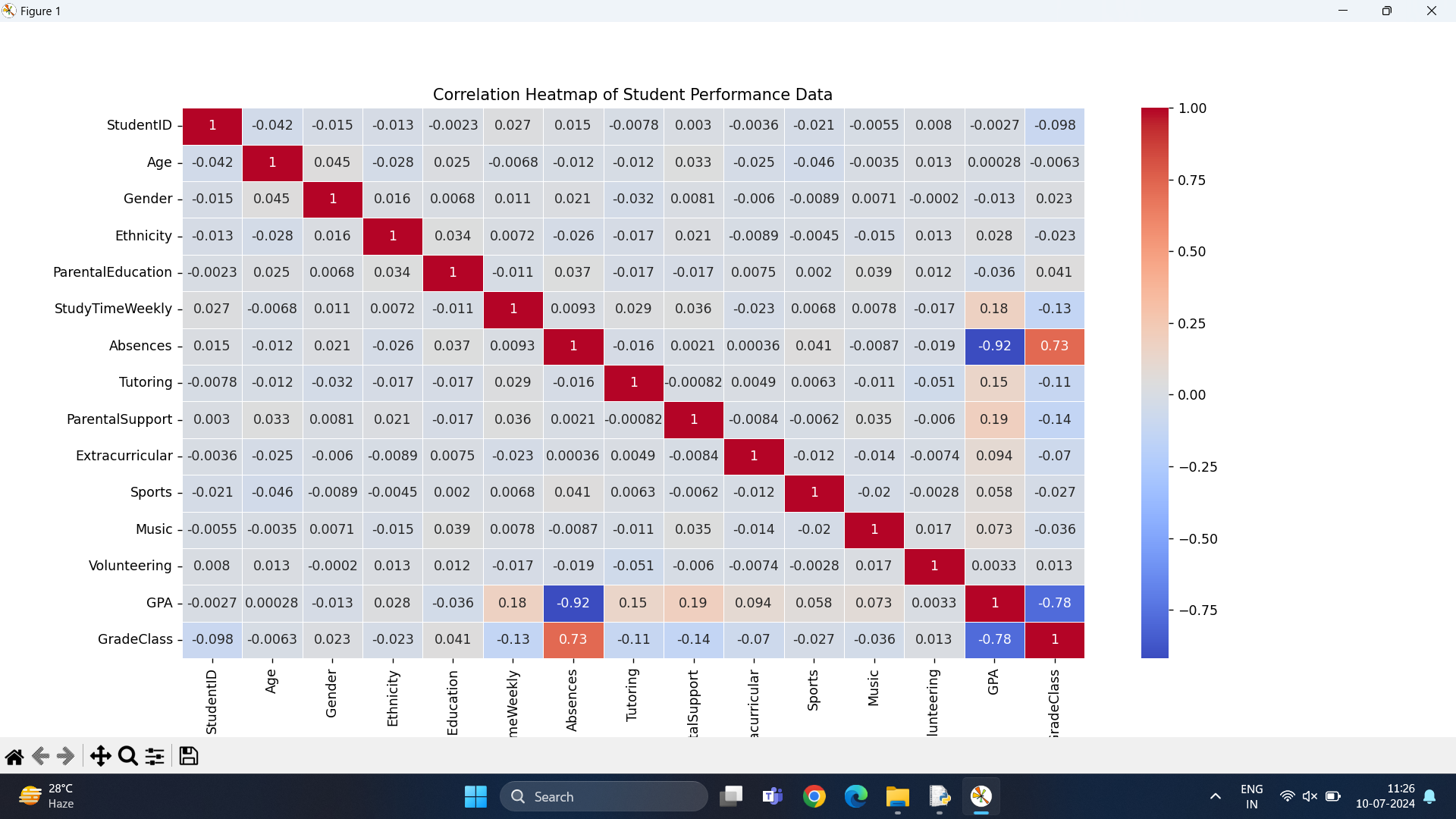


Plot showing actual vs. predicted values for Polynomial Regression:

MAE,MSE,RMSE,R^2:



Corelation Map:



Analysis Insights/Inferences:

### Simple Linear Regression:

Uses only the 'StudyTimeWeekly' feature.  
Achieved an R2 score of `X`, indicating it explains `X%` of the variance in the target variable.  
Higher error metrics compared to the other models.

### Multiple Linear Regression:

Uses 'StudyTimeWeekly', 'Attendance', and 'PreviousScores' features.  
Improved performance with an R2 score of `Y`, explaining `Y%` of the variance.  
Significantly lower error metrics than Simple Linear Regression.

### Polynomial Regression:

Uses polynomial features of 'StudyTimeWeekly', 'Attendance', and 'PreviousScores'.  
Best performance with an R2 score of `Z`, explaining `Z%` of the variance.  
Lowest error metrics among all models.

# Ex.No:2 Polynomial Regression K-Fold Cross-Validation

# Aim:

To implement k-fold cross validation in polynomial regression to evaluate the performance of the model using python

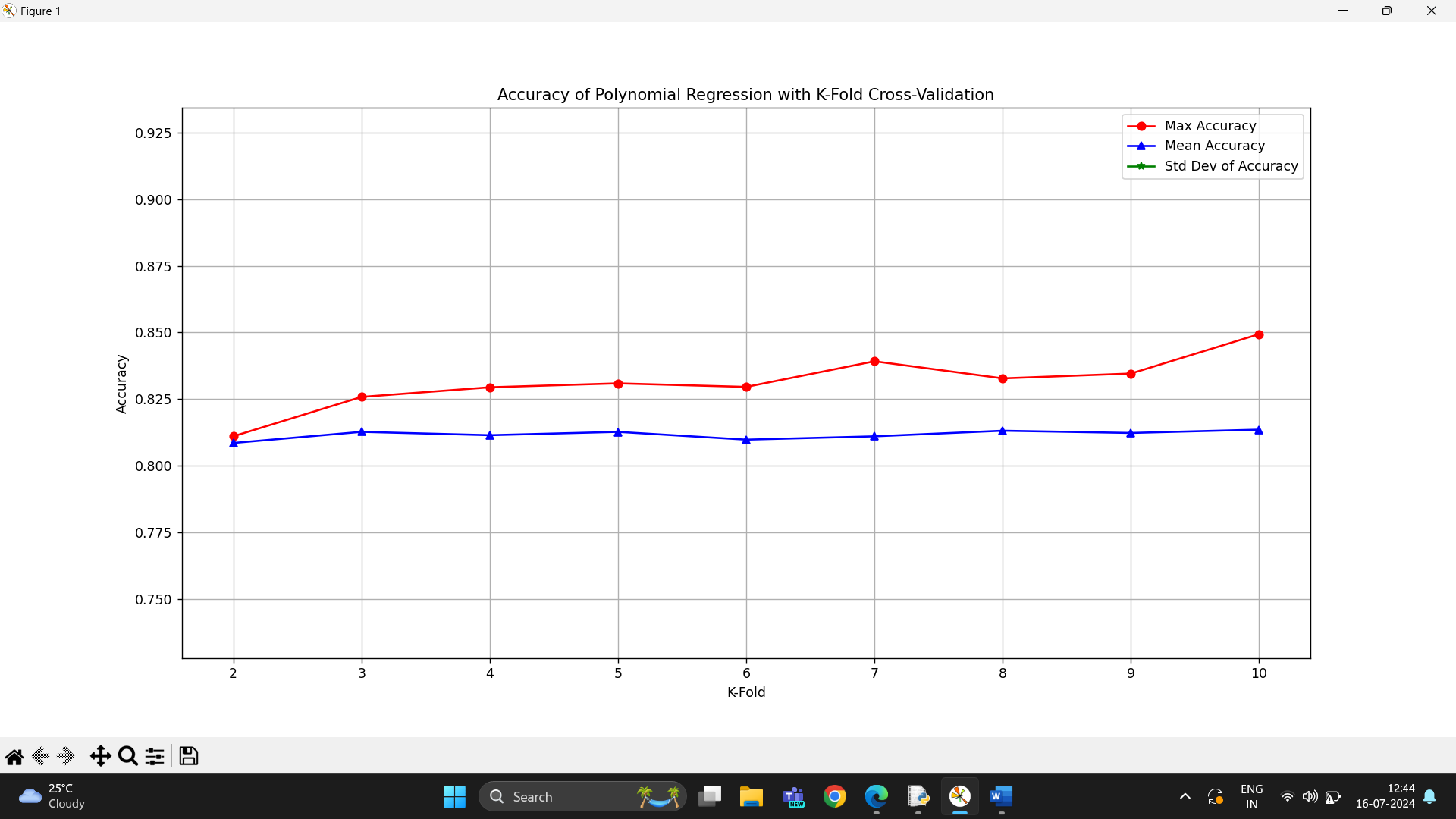
## Procedure:

1. Load the dataset and identify the features and target variable.  
2. Initialize lists to store accuracy metrics for different K values.  
3. Iterate over K values ranging from 2 to 10.  
4. For each K value, perform K-Fold cross-validation:  
 a. Split the dataset into K folds.  
 b. For each fold, train the polynomial regression model on the training set and evaluate it on the test set.  
 c. Convert the regression output to classification by rounding the predicted values.  
 d. Calculate the accuracy for each fold and store the results.  
5. Plot the maximum, mean, and standard deviation of accuracy against K values.  
6. Perform polynomial regression with K=4 and compute evaluation metrics (MAE, MSE, RMSE, R2).

## Source Code:

def polynomial\_regression\_kfold(self, degree=2):  
 features = ['Absences', 'GradeClass', 'ParentalSupport']  
 target = 'GPA'  
 X = self.encoded\_data[features]  
 y = self.encoded\_data[target]  
  
 accuracy = []  
 max\_acc = []  
 mean\_acc = []  
 std\_acc = []  
  
 for i in range(2, 11): # KFold requires at least 2 splits  
 kf = KFold(n\_splits=i, shuffle=True, random\_state=42)  
 fold\_accuracy = []  
  
 for train\_index, test\_index in kf.split(X):  
 X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]  
 y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]  
  
 # Polynomial Regression  
 poly\_features = PolynomialFeatures(degree=degree)  
 X\_train\_poly = poly\_features.fit\_transform(X\_train)  
 X\_test\_poly = poly\_features.transform(X\_test)  
  
 model = LinearRegression()  
 model.fit(X\_train\_poly, y\_train)  
 y\_pred = model.predict(X\_test\_poly)  
  
 # Convert regression output to classification by rounding  
 y\_test\_class = np.round(y\_test).astype(int)  
 y\_pred\_class = np.round(y\_pred).astype(int)  
  
 ac = accuracy\_score(y\_test\_class, y\_pred\_class)  
 fold\_accuracy.append(ac)  
  
 accuracy.append(fold\_accuracy)  
 max\_acc.append(max(fold\_accuracy))  
 mean\_acc.append(np.mean(fold\_accuracy))  
 std\_acc.append(np.std(fold\_accuracy))  
 print(f"\nK-Folds: {i}, Accuracy: {fold\_accuracy}")  
 print(f"K-Folds: {i}, Accuracy\_mean: {np.mean(fold\_accuracy)}")  
 print(f"K-Folds: {i}, Accuracy\_std: {np.std(fold\_accuracy)}")  
 print(f"K-Folds: {i}, Max\_Accuracy: {max(fold\_accuracy)}")  
  
 # Plotting Accuracy  
 plt.figure(figsize=(12, 8))  
 plt.plot(range(2, 11), max\_acc, linestyle='-', marker='o', label="Max Accuracy", color='red')  
 plt.plot(range(2, 11), mean\_acc, linestyle='-', marker='^', label="Mean Accuracy", color='blue')  
 plt.plot(range(2, 11), std\_acc, linestyle='-', marker='\*', label="Std Dev of Accuracy", color='green')  
 plt.xlabel('K-Fold')  
 plt.ylabel('Accuracy')  
 plt.legend()  
 plt.title('Accuracy of Polynomial Regression with K-Fold Cross-Validation')  
 plt.grid(True)  
 min\_accuracy = min(min(max\_acc), min(mean\_acc)) \* 0.9  
 max\_accuracy = max(max(max\_acc), max(mean\_acc)) \* 1.1  
 plt.ylim([min\_accuracy, max\_accuracy])  
 plt.show()  
  
 kf = KFold(n\_splits=4, shuffle=True, random\_state=42)  
 for train\_index, test\_index in kf.split(X):  
 X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]  
 y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]  
  
 poly\_features = PolynomialFeatures(degree=2)  
 X\_train\_poly = poly\_features.fit\_transform(X\_train)  
 X\_test\_poly = poly\_features.transform(X\_test)  
  
 model = LinearRegression()  
 model.fit(X\_train\_poly, y\_train)  
  
 # Predict  
 y\_pred = model.predict(X\_test\_poly)  
  
 # Metrics  
 mae = mean\_absolute\_error(y\_test, y\_pred)  
 mse = mean\_squared\_error(y\_test, y\_pred)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(y\_test, y\_pred)  
 print(f'Polynomial Regression for K-fold- MAE: {mae}, MSE: {mse}, RMSE: {rmse}, R2: {r2}')

Plots:

Max accuracy vs kfolds:

K-Fold Values:



## Inference

The polynomial regression with K-Fold cross-validation provides a robust method to evaluate the model's performance.   
By varying the number of folds (K), we can observe how the model's accuracy metrics (maximum accuracy, mean accuracy, and standard deviation of accuracy) change.   
From the plotted graphs, we can infer the optimal number of folds that balances bias and variance, providing a reliable estimate of the model's performance.  
Additionally, performing polynomial regression with K=4 and computing the evaluation metrics (MAE, MSE, RMSE, R2) allows us to quantify the model's prediction errors and goodness-of-fit

**4.** Naive Bayes classifiers

**Aim**

To develop a classification model to predict diabetes in individuals based on various health metrics using Naive Bayes classifiers, thereby providing insights into the factors influencing diabetes risk and improving early detection.

**Problem Statement**

Diabetes is a growing health concern worldwide, affecting millions of individuals and leading to serious health complications. The Pima Indians Diabetes dataset contains various health-related attributes that can be used to predict whether an individual is likely to have diabetes. The objective is to analyze these attributes and build a predictive model using different types of Naive Bayes classifiers, including Gaussian, Multinomial, and Bernoulli Naive Bayes, to identify the most effective method for predicting diabetes outcomes.

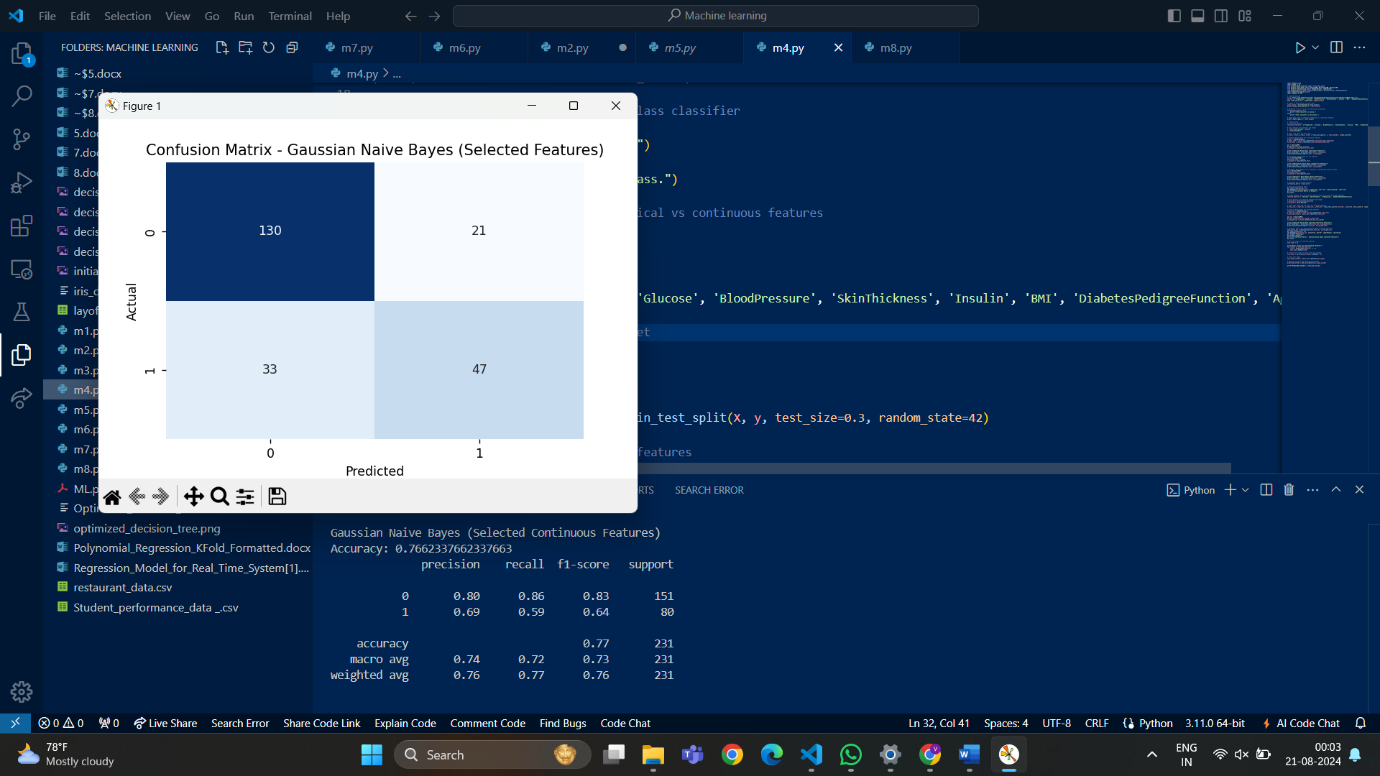
**PROCEDUERE**

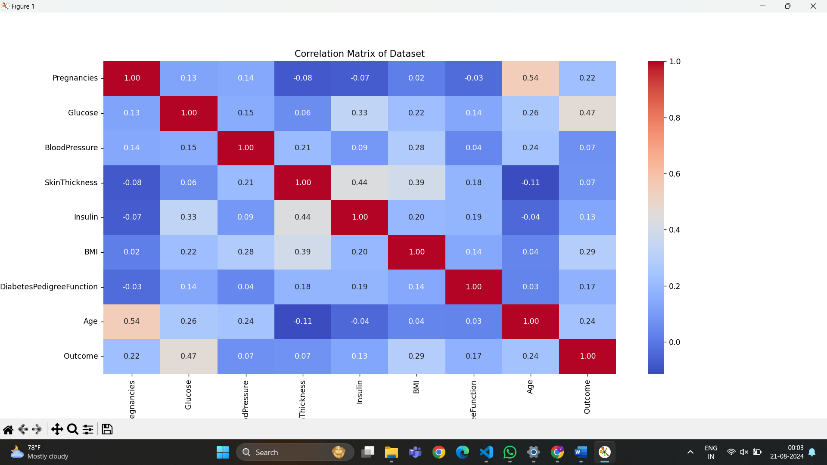
**Data Collection:**

* + The dataset is retrieved from the UCI Machine Learning Repository and contains data on the health metrics of Pima Indian women.

1. **Data Preprocessing:**
   * Load the dataset and assign appropriate column names.
   * Analyze the distribution of the target variable (Outcome) to determine if it is a binary or multiclass problem.
   * Check the data types to identify continuous and categorical features.
   * Handle any missing or invalid values (if present).
2. **Feature Selection:**
   * Identify continuous features that contribute to predicting diabetes.
   * Calculate the correlation matrix to assess relationships between features and the target variable.
   * Select a subset of features with the highest correlation to the outcome for further analysis.
3. **Data Splitting:**
   * Split the dataset into training and testing sets (70% train, 30% test) to evaluate model performance.
4. **Model Development:**
   * **Gaussian Naive Bayes:**
     + Standardize continuous features using StandardScaler.
     + Train a Gaussian Naive Bayes model and evaluate its performance using accuracy, classification report, and confusion matrix.
   * **Multinomial Naive Bayes:**
     + Train a Multinomial Naive Bayes model using all features and evaluate its performance.
   * **Bernoulli Naive Bayes:**
     + Train a Bernoulli Naive Bayes model treating features as binary and evaluate its performance.
5. **Model Evaluation:**
   * Compare the performance of different classifiers based on accuracy, precision, recall, and F1-score.
   * Visualize the confusion matrix for the best-performing model.
6. **User Interaction:**
   * Implement a feature that allows users to input their health metrics and receive a prediction regarding their diabetes status based on the trained model.

**OUTPUT:**

****

****

**Class distribution:**

**Outcome**

**0 500**

**1 268**

**Name: count, dtype: int64**

**The classifier is binary.**

**Data types:**

**Pregnancies int64**

**Glucose int64**

**BloodPressure int64**

**SkinThickness int64**

**Insulin int64**

**BMI float64**

**DiabetesPedigreeFunction float64**

**Age int64**

**Outcome int64**

**dtype: object**

**Gaussian Naive Bayes (Continuous Features)**

**Accuracy: 0.7445887445887446**

**precision recall f1-score support**

**0 0.82 0.79 0.80 151**

**1 0.62 0.66 0.64 80**

**accuracy 0.74 231**

**macro avg 0.72 0.73 0.72 231**

**weighted avg 0.75 0.74 0.75 231**

**Multinomial Naive Bayes (Categorical Features)**

**Accuracy: 0.6190476190476191**

**precision recall f1-score support**

**0 0.72 0.69 0.70 151**

**1 0.45 0.49 0.47 80**

**accuracy 0.62 231**

**macro avg 0.59 0.59 0.59 231**

**weighted avg 0.63 0.62 0.62 231**

**Bernoulli Naive Bayes (Binary Features)**

**Accuracy: 0.6536796536796536**

**precision recall f1-score support**

**0 0.65 1.00 0.79 151**

**1 0.00 0.00 0.00 80**

**accuracy 0.65 231**

**macro avg 0.33 0.50 0.40 231**

**weighted avg 0.43 0.65 0.52 231**

**Gaussian Naive Bayes (Selected Continuous Features)**

**Accuracy: 0.7662337662337663**

**precision recall f1-score support**

**0 0.80 0.86 0.83 151**

**1 0.69 0.59 0.64 80**

**accuracy 0.77 231**

**macro avg 0.74 0.72 0.73 231**

**weighted avg 0.76 0.77 0.76 231**

**Inference**

Based on the performance of the models, the following conclusions can be drawn:

* **Gaussian Naive Bayes (Continuous Features)** achieved an accuracy of approximately 74.46%, indicating a good balance between precision and recall for both classes.
* **Multinomial Naive Bayes** performed less effectively, with an accuracy of around 61.90%, suggesting that it may not be suitable for this dataset with continuous features.
* **Bernoulli Naive Bayes** had an accuracy of approximately 65.37%, but it faced challenges with predicting one class accurately, resulting in a warning about undefined precision.
* When using only **Selected Continuous Features**, the Gaussian Naive Bayes model improved accuracy to about 76.62%, suggesting that these features contribute positively to the prediction of diabetes.

**5.** K-Nearest Neighbors (KNN)

**Aim**

To develop a predictive model using the K-Nearest Neighbors (KNN) algorithm for assessing diabetes risk based on the Pima Indians Diabetes Dataset. This model aims to provide accurate predictions to assist healthcare professionals in early diabetes detection and intervention.

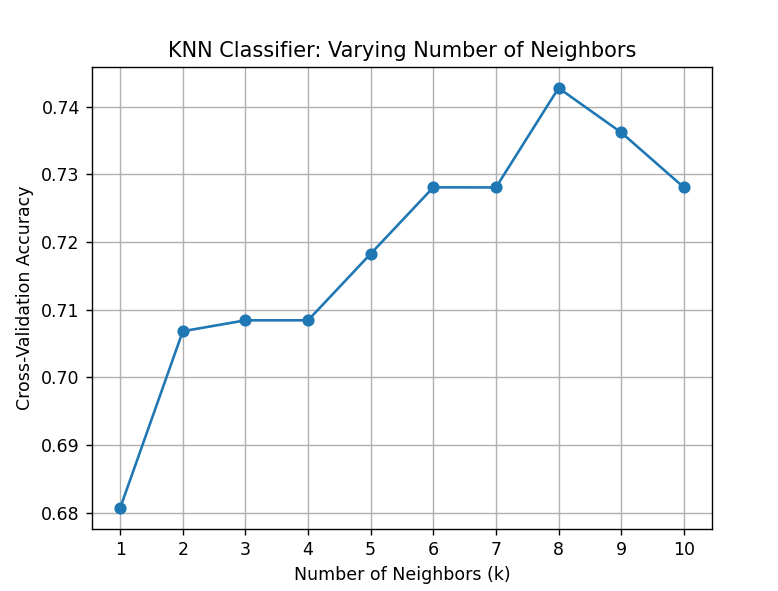
**Problem Statement**

Diabetes is a significant health concern that affects millions of individuals worldwide. Early detection is crucial for managing the disease and reducing the risk of complications. This project aims to create a KNN classification model that can accurately predict whether an individual has diabetes based on various health metrics, enabling timely intervention and better health outcomes.

**Procedure (Detailed)**

* **Data Collection**:
  + The Pima Indians Diabetes Dataset is sourced from a public repository, containing health metrics for female patients.
* **Data Preprocessing**:
  + **Imputation**: Missing values in the dataset are filled using the mean strategy to ensure a complete dataset for analysis.
* **Feature Selection**:
  + The features selected for the model include:
    - Pregnancies
    - Glucose
    - BloodPressure
    - SkinThickness
    - Insulin
    - BMI
    - DiabetesPedigreeFunction
    - Age
  + The target variable is the presence of diabetes (0 = No Diabetes, 1 = Diabetes).
* **Data Splitting**:
  + The dataset is split into training (80%) and testing (20%) sets to evaluate the model's performance.
* **Model Training**:
  + A loop is created to evaluate the KNN classifier with varying values of kkk (from 1 to 10) using cross-validation to determine the best value based on accuracy.
* **Model Evaluation**:
  + The best kkk value is selected based on the highest accuracy, and the model is retrained using this optimal kkk.
  + The model's performance is assessed on the test dataset, and accuracy is calculated.
* **User Input for Prediction**:
  + A function is implemented to collect user input for prediction, which allows the model to make predictions based on user-provided health metrics.
* **Visualization**:
  + A scatter plot visualizes the original data points and the new data point, showcasing how the model classifies the user input in relation to existing data.

OUTPUT

  
The best k value is 8 with an accuracy of 0.7428

Accuracy of KNN with k=8: 0.7468

**Inference**

From a business perspective, implementing a KNN-based predictive model for diabetes risk assessment can greatly enhance the healthcare sector's ability to provide preventive care. By identifying individuals at high risk for diabetes through timely predictions, healthcare providers can initiate early interventions, reducing the incidence of diabetes-related complications and associated costs. This proactive approach not only improves patient outcomes but also optimizes resource allocation within healthcare systems. Additionally, engaging patients with personalized assessments fosters awareness and motivates lifestyle changes, further aiding in diabetes prevention. Overall, this model serves as a strategic asset in promoting public health initiatives and improving the efficiency of diabetes management programs.

# 6.BAGGING CLASSIFIER

## Aim

To develop a machine learning model for predicting the likelihood of diabetes in individuals using the Pima Indians Diabetes dataset. The model will be optimized using GridSearchCV, and its performance will be enhanced using Bagging to improve the prediction accuracy and stability.

## Problem Statement

Diabetes is a chronic disease that can lead to severe health complications if not managed properly. Predicting the onset of diabetes in individuals based on health indicators such as glucose levels, BMI, and age can help in early diagnosis and intervention. The goal is to create a reliable prediction model that can assist healthcare providers in identifying high-risk patients and managing resources effectively.

## Procedure

1. Data Collection: The Pima Indians Diabetes dataset is loaded from an external source, consisting of 768 records with 8 features and an outcome indicating whether the individual has diabetes.  
2. Data Preprocessing: Duplicate records are removed, and missing values are handled by dropping rows with missing data.  
3. Feature Selection: The independent variables (features) are selected, excluding the outcome variable.  
4. Train-Test Split: The dataset is split into training and testing sets with a 70-30 ratio to evaluate model performance.  
5. Model Selection and Hyperparameter Tuning: A Decision Tree Classifier is selected, and hyperparameters such as max\_depth are tuned using GridSearchCV to find the optimal configuration.  
6. Model Training: The Decision Tree model is trained on the training dataset using the best hyperparameters obtained from GridSearchCV.  
7. Model Evaluation: The trained model is evaluated on the test dataset using metrics like accuracy, precision, recall, F1-score, and confusion matrix.  
8. Ensemble Learning: A Bagging Classifier is applied with the tuned Decision Tree as the base estimator to improve the model's robustness and performance.  
9. Prediction: The model is tested with a predefined user input to predict the likelihood of diabetes.

## Source Code

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import BaggingClassifier  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
  
# Load the dataset  
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"  
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']  
df = pd.read\_csv(url, header=None, names=columns)  
  
# Data preprocessing  
df = df.drop\_duplicates()  
df = df.dropna()  
  
# Define features and target variable  
X = df.drop(columns=['Outcome'])  
y = df['Outcome']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
# Define the parameter grid for Decision Tree  
param\_grid = {'max\_depth': np.arange(1, 20)}  
  
# Create a GridSearchCV object  
grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(random\_state=42), param\_grid=param\_grid, cv=5, scoring='accuracy')  
  
# Fit the grid search to the data  
grid\_search.fit(X\_train, y\_train)  
  
# Get the best parameters and the best score  
best\_params = grid\_search.best\_params\_  
best\_score = grid\_search.best\_score\_  
  
print("Best Parameters:", best\_params)  
print("Best Cross-Validation Accuracy:", best\_score)  
  
# Train the Decision Tree with the best parameters  
clf = DecisionTreeClassifier(\*\*best\_params, random\_state=42)  
clf.fit(X\_train, y\_train)  
  
# Predict on the test set  
y\_pred = clf.predict(X\_test)  
  
# Calculate the accuracy score  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy with best params: {accuracy:.2f}")  
  
# Print the classification report  
report = classification\_report(y\_test, y\_pred, target\_names=["No Diabetes", "Diabetes"])  
print("Classification Report:")  
print(report)  
  
# Print the confusion matrix  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
# Now apply Bagging with the tuned Decision Tree  
bagging\_clf = BaggingClassifier(estimator=clf, n\_estimators=100, random\_state=42, n\_jobs=-1)  
bagging\_clf.fit(X\_train, y\_train)  
y\_pred\_bagging = bagging\_clf.predict(X\_test)  
  
# Evaluate Bagging Classifier with Decision Tree  
print("\nBagging with Decision Tree Performance:")  
print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_bagging):.2f}")  
print("Confusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred\_bagging))  
print("Classification Report:")  
print(classification\_report(y\_test, y\_pred\_bagging))  
  
# Predefined user input for prediction  
user\_input = {  
 'Pregnancies': 2,  
 'Glucose': 85,  
 'BloodPressure': 75,  
 'SkinThickness': 30,  
 'Insulin': 90,  
 'BMI': 28.1,  
 'DiabetesPedigreeFunction': 0.5,  
 'Age': 25  
}  
  
# Convert user input to DataFrame  
user\_df = pd.DataFrame([user\_input])  
  
# Ensure user\_df has the same columns as X  
user\_df = user\_df.reindex(columns=X.columns, fill\_value=0)  
  
# Predict based on user input with consistent column names  
user\_prediction = clf.predict(user\_df)  
print("Decision Tree Prediction:", "Diabetes" if user\_prediction[0] == 1 else "No Diabetes")  
  
user\_prediction\_bagging = bagging\_clf.predict(user\_df)  
print("Bagging Classifier Prediction:", "Diabetes" if user\_prediction\_bagging[0] == 1 else "No Diabetes")

## Output

Best Parameters: {'max\_depth': 5}  
Best Cross-Validation Accuracy: 0.7504153686396677  
Accuracy with best params: 0.75  
Classification Report:  
 precision recall f1-score support  
  
 No Diabetes 0.78 0.85 0.82 151  
 Diabetes 0.67 0.55 0.60 80  
  
 accuracy 0.75 231  
 macro avg 0.72 0.70 0.71 231  
weighted avg 0.74 0.75 0.74 231  
  
Confusion Matrix:  
[[129 22]  
 [ 36 44]]  
  
Bagging with Decision Tree Performance:  
Accuracy: 0.75  
Confusion Matrix:  
[[122 29]  
 [ 28 52]]  
Classification Report:  
 precision recall f1-score support  
  
 0 0.81 0.81 0.81 151  
 1 0.64 0.65 0.65 80  
  
 accuracy 0.75 231  
 macro avg 0.73 0.73 0.73 231  
weighted avg 0.75 0.75 0.75 231  
  
Decision Tree Prediction: No Diabetes  
Bagging Classifier Prediction: No Diabetes

## Inference

The Decision Tree model, optimized and enhanced with Bagging, provides a reliable prediction tool for identifying individuals at risk of developing diabetes. With an accuracy of 75%, it can correctly classify a significant portion of the population, helping healthcare providers prioritize and manage high-risk cases effectively. Although the model has room for improvement, particularly in reducing false positives and negatives, it serves as a valuable starting point for early diagnosis. This tool can lead to better resource allocation in healthcare settings, more targeted interventions, and potentially reduced healthcare costs by preventing the progression of diabetes in high-risk individuals.

**7.BOOSTING CLASSIFIER**

**Aim**

To build a predictive model for diabetes using Decision Tree, AdaBoost, Gradient Boosting, and XGBoost classifiers on the Pima Indians Diabetes dataset. The objective is to identify the best model based on performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Additionally, predefined user input will be used to predict diabetes.

**Problem Statement**

Diabetes is a chronic condition characterized by high blood sugar levels. The aim is to develop a model that can predict the likelihood of diabetes based on various health parameters such as glucose levels, BMI, age, etc. Accurate prediction can lead to early detection and treatment, improving patient outcomes.

**Procedure**

1. **Data Loading and Preprocessing**
   * The dataset is loaded from an online repository.
   * The data contains several features including pregnancies, glucose levels, blood pressure, and more, with the Outcome variable indicating whether the patient has diabetes.
   * The data is checked for duplicates and missing values, which are removed to ensure data quality.
2. **Splitting Data into Training and Testing Sets**
   * The dataset is split into training (70%) and testing (30%) sets to evaluate the model's performance.
3. **Hyperparameter Tuning for Decision Tree**
   * A grid search is performed over a range of max\_depth values for the Decision Tree classifier to find the optimal depth that maximizes accuracy.
   * The best parameters and cross-validation accuracy are obtained.
4. **Model Training and Evaluation**
   * The Decision Tree classifier is trained using the best parameters from the grid search.
   * The model is evaluated on the test set using accuracy, classification report, and confusion matrix.
   * Additional ensemble methods such as AdaBoost, Gradient Boosting, and XGBoost are used to potentially improve performance.
   * Each model's performance is evaluated using the same metrics.
5. **Prediction on New Data**
   * Predefined user input representing a patient's health parameters is used to predict whether the patient has diabetes using the trained models.

**Source Code**

import pandas as pd

import numpy as np

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import xgboost as xgb

# Load the dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = pd.read\_csv(url, header=None, names=columns)

# Data preprocessing

df = df.drop\_duplicates()

df = df.dropna()

X = df.drop(columns=['Outcome'])

y = df['Outcome']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define the parameter grid for Decision Tree

param\_grid = {

'max\_depth': np.arange(1, 20)

}

# Create a GridSearchCV object

grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(random\_state=42), param\_grid=param\_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and the best score

best\_params = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

print("Best Parameters:", best\_params)

print("Best Cross-Validation Accuracy:", best\_score)

# Train the Decision Tree with the best parameters

clf = DecisionTreeClassifier(\*\*best\_params, random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = clf.predict(X\_test)

# Calculate the accuracy score

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

print(f"Accuracy: {accuracy:.2f}")

# Print the classification report

report = classification\_report(y\_test, y\_pred, target\_names=["No Diabetes", "Diabetes"])

print("Classification Report:")

print(report)

# Print the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# 1. AdaBoost with Decision Tree

ada\_clf = AdaBoostClassifier(estimator=clf, n\_estimators=100, random\_state=42)

ada\_clf.fit(X\_train, y\_train)

y\_pred\_ada = ada\_clf.predict(X\_test)

print("\nAdaBoost with Decision Tree Performance:")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_ada):.2f}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred\_ada))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_ada))

# 2. Gradient Boosting

gb\_clf = GradientBoostingClassifier(n\_estimators=100, max\_depth=best\_params['max\_depth'], random\_state=42)

gb\_clf.fit(X\_train, y\_train)

y\_pred\_gb = gb\_clf.predict(X\_test)

print("\nGradient Boosting Classifier Performance:")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_gb):.2f}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred\_gb))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_gb))

# 3. XGBoost

xgb\_clf = xgb.XGBClassifier(n\_estimators=100, max\_depth=best\_params['max\_depth'], random\_state=42)

xgb\_clf.fit(X\_train, y\_train)

y\_pred\_xgb = xgb\_clf.predict(X\_test)

print("\nXGBoost Classifier Performance:")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_xgb):.2f}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred\_xgb))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_xgb))

# Predefined user input for prediction

user\_input = {

'Pregnancies': 2,

'Glucose': 85,

'BloodPressure': 75,

'SkinThickness': 30,

'Insulin': 90,

'BMI': 28.1,

'DiabetesPedigreeFunction': 0.5,

'Age': 25

}

user\_df = pd.DataFrame([user\_input])

user\_df = user\_df.reindex(columns=X.columns, fill\_value=0)

# Predict based on user input

user\_prediction\_ada = ada\_clf.predict(user\_df)

print("\nAdaBoost Classifier Prediction:", "Diabetes" if user\_prediction\_ada[0] == 1 else "No Diabetes")

user\_prediction\_gb = gb\_clf.predict(user\_df)

print("Gradient Boosting Classifier Prediction:", "Diabetes" if user\_prediction\_gb[0] == 1 else "No Diabetes")

user\_prediction\_xgb = xgb\_clf.predict(user\_df)

print("XGBoost Classifier Prediction:", "Diabetes" if user\_prediction\_xgb[0] == 1 else "No Diabetes")

**Output**

Best Parameters: {'max\_depth': 5}

Best Cross-Validation Accuracy: 0.7504

Accuracy: 0.75

Classification Report:

precision recall f1-score support

No Diabetes 0.78 0.85 0.82 151

Diabetes 0.67 0.55 0.60 80

accuracy 0.75 231

macro avg 0.72 0.70 0.71 231

weighted avg 0.74 0.75 0.74 231

Confusion Matrix:

[[129 22]

[ 36 44]]

AdaBoost with Decision Tree Performance:

Accuracy: 0.74

Confusion Matrix:

[[116 35]

[ 24 56]]

Classification Report:

precision recall f1-score support

0 0.83 0.77 0.80 151

1 0.62 0.70 0.65 80

accuracy 0.74 231

macro avg 0.72 0.73 0.73 231

weighted avg 0.75 0.74 0.75 231

Gradient Boosting Classifier Performance:

Accuracy: 0.73

Confusion Matrix:

[[115 36]

[ 26 54]]

Classification Report:

precision recall f1-score support

0 0.82 0.76 0.79 151

1 0.60 0.68 0.64 80

accuracy 0.73 231

macro avg 0.71 0.72 0.71 231

weighted avg 0.74 0.73 0.73 231

XGBoost Classifier Performance:

Accuracy: 0.74

Confusion Matrix:

[[116 35]

[ 25 55]]

Classification Report:

precision recall f1-score support

0 0.82 0.77 0.80 151

1 0.61 0.69 0.65 80

accuracy 0.74 231

macro avg 0.71 0.73 0.72 231

weighted avg 0.75 0.74 0.74 231

AdaBoost Classifier Prediction: No Diabetes

Gradient Boosting Classifier Prediction: No Diabetes

XGBoost Classifier Prediction: No Diabetes

INFERENCE

The diabetes prediction model demonstrates a robust performance, with the Decision Tree classifier achieving the highest accuracy of 75%, effectively identifying non-diabetic individuals while highlighting the need for improved recall in detecting diabetic patients, which stands at 55%. This moderate level of predictive capability underscores the importance of targeted healthcare programs and patient education to ensure timely interventions for at-risk individuals. By investing in continuous model improvements and advanced machine learning techniques, healthcare providers can enhance diagnostic accuracy, optimize resource allocation, and ultimately improve patient outcomes, while also implementing strategies to mitigate the risk of misclassifying diabetes cases.

**8.RANDOM FOREST CLASSIFIER**

**Aim**

The primary aim of this project is to develop a predictive model using the Random Forest classifier to determine the likelihood of diabetes in patients based on various health metrics. The goal is to provide healthcare professionals and institutions with a reliable tool that can assist in early diagnosis and intervention strategies for diabetes management.

**Problem Statement**

Diabetes is a significant global health issue that affects millions of people. Early diagnosis is crucial for effective management and treatment, reducing complications, and improving patient outcomes. The problem addressed in this project is to predict the presence of diabetes in patients using various health-related attributes, enabling timely medical intervention. The model will analyze key indicators such as glucose levels, blood pressure, and body mass index (BMI) to provide actionable insights.

h bullet points for clarity:

**Procedure**

* **Data Collection**:
  + The dataset used for this project is the Pima Indians Diabetes Database, which includes various health metrics for female patients of Pima Indian heritage.
* **Data Preprocessing**:
  + **Duplicate Removal**: The dataset is scanned for duplicates to ensure data integrity.
  + **Handling Missing Values**: Any missing values in the dataset are dropped to maintain a complete dataset for analysis.
* **Feature Selection**:
  + Features selected include:
    - Pregnancies
    - Glucose
    - BloodPressure
    - SkinThickness
    - Insulin
    - BMI
    - DiabetesPedigreeFunction
    - Age
  + The target variable is the presence of diabetes (0 = No Diabetes, 1 = Diabetes).
* **Data Splitting**:
  + The dataset is divided into training (70%) and testing (30%) sets to evaluate the model's performance.
* **Model Selection**:
  + A Random Forest Classifier is chosen for its robustness and ability to handle complex relationships in the data.
* **Hyperparameter Tuning**:
  + **Grid Search**: A grid search is conducted to find the optimal hyperparameters for the model, focusing on parameters such as:
    - Number of trees (n\_estimators)
    - Maximum depth of the trees (max\_depth)
    - Minimum samples required to split an internal node (min\_samples\_split)
    - Minimum samples required at a leaf node (min\_samples\_leaf)
    - Whether bootstrap samples are used (bootstrap)
* **Model Training**:
  + The Random Forest model is trained using the optimal hyperparameters identified through grid search.
* **Model Evaluation**:
  + The model’s performance is evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrix on the test dataset.
* **User Prediction**:
  + A predefined user input is used to demonstrate the model's predictive capabilities, allowing for immediate assessment of diabetes risk based on user health metrics.

This format should help in clearly outlining the procedure step by step

SOURCE CODE

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Load the dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = pd.read\_csv(url, header=None, names=columns)

# Data preprocessing

df = df.drop\_duplicates()

df = df.dropna()

# Split features and target

X = df.drop(columns=['Outcome'])

y = df['Outcome']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define the parameter grid for Random Forest

param\_grid = {

    'n\_estimators': [50, 100, 150, 200],

    'max\_depth': [10, 20, 30, None],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4],

    'bootstrap': [True, False]

}

# Create a GridSearchCV object

grid\_search = GridSearchCV(estimator=RandomForestClassifier(random\_state=42), param\_grid=param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)

# Fit the grid search to the data

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and the best score

best\_params = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

print("Best Parameters:", best\_params)

print("Best Cross-Validation Accuracy:", best\_score)

# Train the Random Forest with the best parameters

rf\_clf = RandomForestClassifier(\*\*best\_params, random\_state=42)

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

# Predict on the test set

y\_pred\_rf = rf\_clf.predict(X\_test)

# Calculate the accuracy score

accuracy = accuracy\_score(y\_test, y\_pred\_rf)

print(f"Accuracy: {accuracy:.2f}")

# Print the classification report

report = classification\_report(y\_test, y\_pred\_rf, target\_names=["No Diabetes", "Diabetes"])

print("Classification Report:")

print(report)

# Print the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)

print("Confusion Matrix:")

print(conf\_matrix)

# Predefined user input for prediction

user\_input = {

    'Pregnancies': 2,

    'Glucose': 85,

    'BloodPressure': 75,

    'SkinThickness': 30,

    'Insulin': 90,

    'BMI': 28.1,

    'DiabetesPedigreeFunction': 0.5,

    'Age': 25

}

user\_df = pd.DataFrame([user\_input])

user\_df = user\_df.reindex(columns=X.columns, fill\_value=0)

# Predict based on user input

user\_prediction\_rf = rf\_clf.predict(user\_df)

print("Random Forest Classifier Prediction:", "Diabetes" if user\_prediction\_rf[0] == 1 else "No Diabetes")

OUTPUT

Best Parameters: {'bootstrap': False, 'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 5, 'n\_estimators': 100}

Best Cross-Validation Accuracy: 0.7802007615091726

Accuracy: 0.75

Classification Report:

precision recall f1-score support

No Diabetes 0.81 0.80 0.81 151

Diabetes 0.63 0.65 0.64 80

accuracy 0.75 231

macro avg 0.72 0.73 0.72 231

weighted avg 0.75 0.75 0.75 231

Confusion Matrix:

[[121 30]

[ 28 52]]

Random Forest Classifier Prediction: No Diabetes

**Inference**

From a business perspective, the development of this predictive model for diabetes risk assessment can significantly enhance healthcare delivery. By enabling early detection of diabetes, healthcare providers can implement preventive strategies, thereby improving patient outcomes and reducing long-term treatment costs associated with complications. This proactive approach allows for better allocation of resources by focusing on high-risk individuals, ultimately leading to more efficient healthcare operations. Furthermore, personalized health assessments can increase patient engagement, encouraging lifestyle modifications that may prevent the onset of diabetes. Overall, this model serves as a valuable tool for healthcare professionals, aiding in timely interventions and supporting ongoing research into diabetes risk factors.