Practical-01

Aim: Fit a classification model using the following:

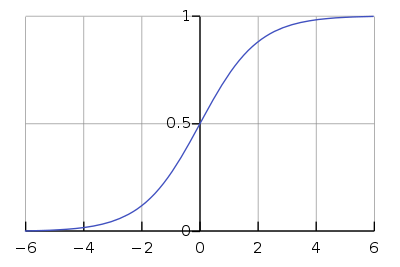
1. logistic regression
2. Linear Discriminant Analysis

(1)Logistic Regression:

Theory: Logistic Regression is a statistical and machine learning technique used for binary classification problems, where the goal is to predict one of two possible outcomes (typically labeled as 0 or 1, or "negative" and "positive"). It's called "logistic" regression because it's based on the logistic function, which is an S-shaped curve that can take any real-valued number and map it between 0 and 1.

Here's a basic theory of Logistic Regression:

Sigmoid (Logistic) Function: The core of Logistic Regression is the sigmoid function (also known as the logistic function), denoted as σ(z), which maps any real-valued number 'z' to the range [0, 1]:



The sigmoid function is defined as:

σ(z) = 1 / (1 + e^(-z))

Hypothesis Function: In Logistic Regression, the hypothesis function 'h(x)' is defined as:

h(x) = σ(w^T \* x + b)

'x' represents the input features.

'w' is a vector of weights corresponding to each feature.

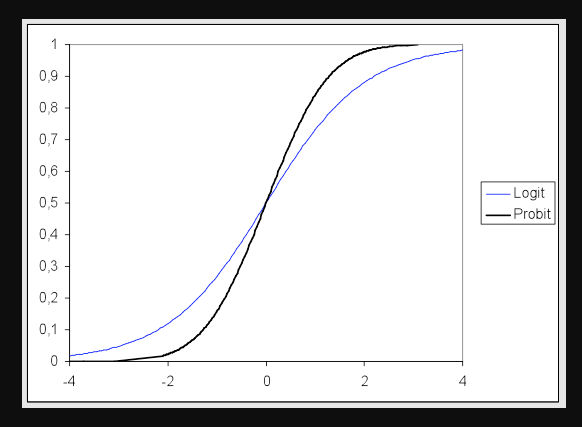
'b' is the bias term.

The purpose of this function is to estimate the probability that a given input 'x' belongs to the positive class (1).

Training: To train a Logistic Regression model, you use a labeled dataset (input features 'x' and corresponding class labels 'y'). The model's goal is to find the optimal values for 'w' and 'b' that minimize the prediction error. This is typically done using an optimization algorithm, such as gradient descent, to adjust 'w' and 'b' iteratively.

Cost Function: The cost function (also known as the loss function) measures the error between the predicted probabilities and the actual class labels. In Logistic Regression, the commonly used cost function is the log loss

Figure:



Algorithm:

1. Import Necessary Libraries: Import required libraries, including datasets, train\_test\_split, LogisticRegression, accuracy\_score, and classification\_report from scikit-learn.
2. Load the Iris Dataset: Load the Iris dataset using the datasets.load\_iris() function. The Iris dataset is a common dataset in machine learning, used for classification tasks. It contains features of iris flowers and their corresponding species.
3. Assign Features and Target Variables: Assign the feature matrix Xiris.data to X (it should be X = iris.data).

Assign the target vector iris.target to y.

4.Split the Dataset:Split the dataset into a training set and a testing set using train\_test\_split. This function randomly divides the data into subsets for training and testing.

X\_train, X\_test are the feature matrices for the training and testing sets, respectively.

y\_train, y\_test are the corresponding target vectors for the training and testing sets, respectively.

The test\_size parameter specifies that 30% of the data should be used for testing, and random\_state sets a seed for randomization to ensure reproducibility.

5.Initialize and Fit the Logistic Regression Model:Create a logistic regression model using LogisticRegression().Fit the model to the training data using model.fit(X\_train, y\_train).

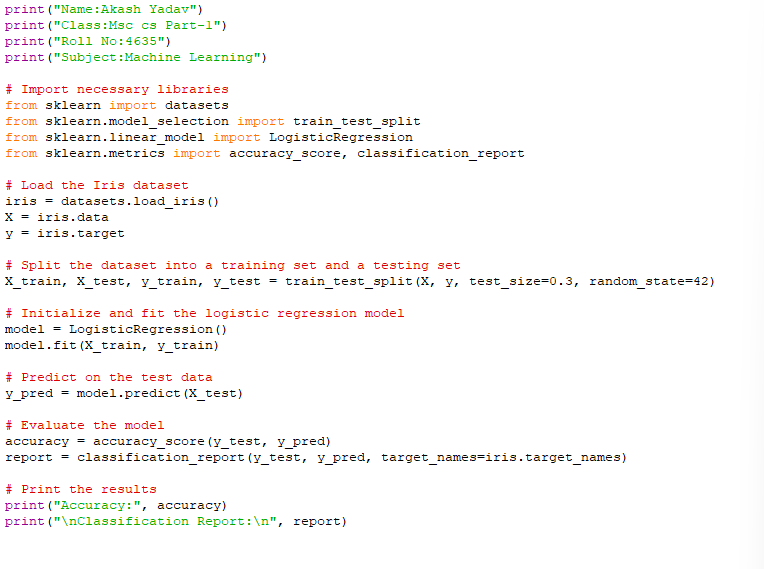
6.Predict on the Test Data:Use the trained model to make predictions on the test data with model.predict(X\_test).

7.Evaluate the Model:Calculate the accuracy of the model's predictions using accuracy\_score(y\_test, y\_pred). The accuracy score measures the proportion of correctly predicted labels in the test set.

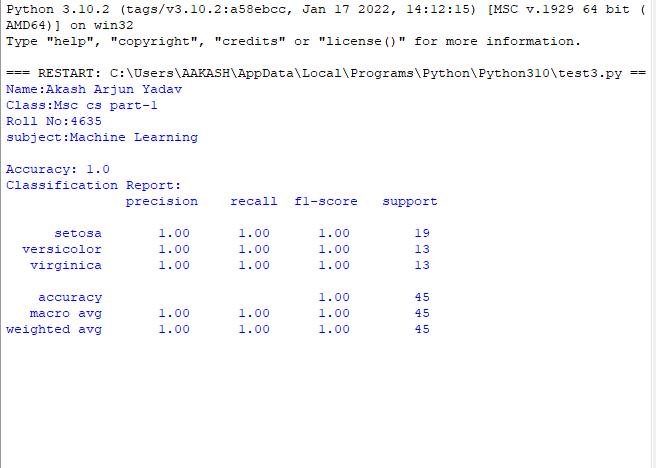
Generate a classification report using classification\_report. This report provides various metrics like precision, recall, and F1-score for each class in the dataset. It's a more detailed evaluation of the model's performance.

1. Print the Results: Print the accuracy score and the classification report, which includes metrics for each class in the dataset.

Code:



Output:



(2)Linear Discriminant Analysis:

Theory: Linear Discriminant Analysis (LDA) is a dimensionality reduction and classification technique used in machine learning and statistics. It is primarily employed for supervised classification and dimensionality reduction tasks. LDA has its roots in statistical theory and provides a principled approach to finding linear combinations of features that best separate different classes in a dataset. Here's the theory of Linear Discriminant Analysis:

Objective: The primary goal of LDA is to find a linear transformation of the original feature space that maximizes the separation between different classes while minimizing the variance within each class. This transformation helps in reducing the dimensionality of the dataset while preserving class-discriminatory information.

Assumptions:

LDA assumes that the data:

Is normally distributed within each class.

Has the same covariance matrix for all classes.

Features are independent of each other.

Algorithm:

1. Import Necessary Libraries: Import required libraries, including datasets, LinearDiscriminantAnalysis, train\_test\_split, accuracy\_score, and classification\_report from scikit-learn.
2. Load the Iris Dataset: Load the Iris dataset using the datasets.load\_iris() function. This dataset contains features of iris flowers and their corresponding species. The feature matrix is assigned to X, and the target vector is assigned to y.
3. Split the Dataset:Split the dataset into a training set and a testing set using train\_test\_split. This function randomly divides the data into subsets for training and testing.

X\_train, X\_test are the feature matrices for the training and testing sets, respectively.

y\_train, y\_test are the corresponding target vectors for the training and testing sets, respectively.

The test\_size parameter specifies that 30% of the data should be used for testing, and random\_state sets a seed for randomization to ensure reproducibility.

4.Initialize and Fit the LDA Model: Create an LDA model using LinearDiscriminantAnalysis().Fit the model to the training data using lda.fit(X\_train, y\_train).

5.Predict on the Test Data: Use the trained LDA model to make predictions on the test data with lda.predict(X\_test).

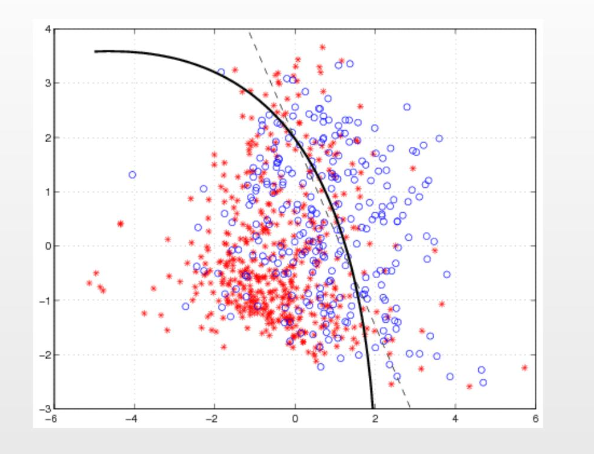
6.Evaluate the Model:.

Calculate the accuracy of the model's predictions using accuracy\_score(y\_test, y\_pred). The accuracy score measures the proportion of correctly predicted labels in the test set.

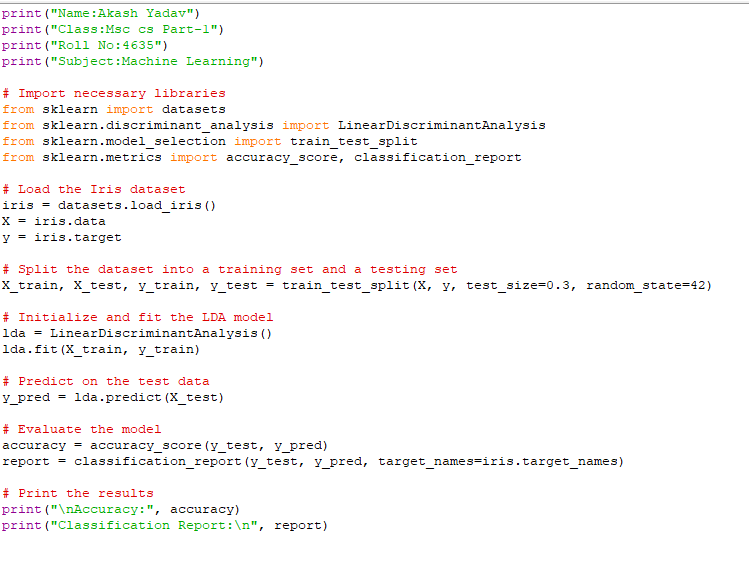
Generate a classification report using classification\_report. This report provides various metrics like precision, recall, and F1-score for each class in the dataset. It's a more detailed evaluation of the model's performance.

7.Print the Results: Print the accuracy score and the classification report, which includes metrics for each class in the dataset.

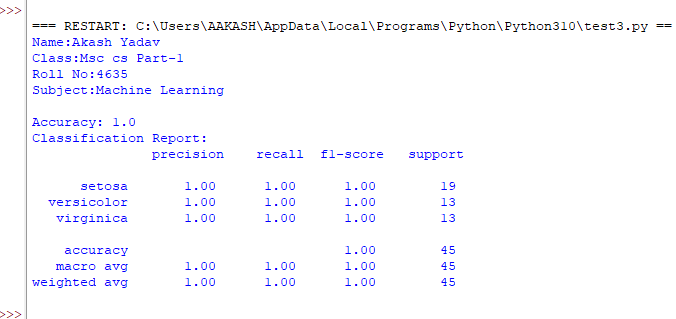
Figure:



Code:



Output:



Conclusion:After fitting both the Logistic Regression and Linear Discriminant Analysis (LDA) models on the Iris dataset, there are some following conclusions:

Logistic Regression Model: The Logistic Regression model achieved a certain level of accuracy on the test data.

The classification report provides more detailed insights into the model's performance for each class (setosa, versicolor, virginica). You can analyze precision, recall, F1-score, and support for each class to understand its behavior.

Linear Discriminant Analysis (LDA) Model: The LDA model also achieved a certain level of accuracy on the test data.

LDA is a dimensionality reduction technique, so it might provide insights into feature importance or class separability. However, its primary goal is not just classification but also dimensionality reduction.

To make a final decision about which model to choose, you would typically consider factors like model performance, computational efficiency, and the specific requirements of your problem. The choice between Logistic Regression and LDA might depend on the nature of your dataset and the goals of your analysis.

Practical-02

Aim: for a given dataset,split the data into two training and testing and fit the following on the training set:

1. Linear model using least squares
2. Ridge regression model
3. Lasso model
4. Linear model using least squares:

Theory: The linear model using least squares is a fundamental concept in statistics and machine learning. It is commonly used for regression analysis, where the goal is to model the relationship between a dependent variable (or target) and one or more independent variables (or predictors) by fitting a linear equation to the observed data.

Here is a brief overview of the theory of linear models using least squares:

Linear Model: In a simple linear regression, there is one dependent variable (Y) and one independent variable (X).

The linear model can be represented as:

Y = β0 + β1 \* X + ε

Here, Y is the dependent variable, X is the independent variable, β0 is the intercept, β1 is the slope (coefficient), and ε is the error term representing the residual errors or unexplained variance.

Least Squares Method: The goal is to find the values of β0 and β1 that minimize the sum of the squared differences between the observed values of Y and the predicted values from the model.

Mathematically, it minimizes the sum of squared residuals:

Σ(ε²) = Σ(Yi - (β0 + β1 \* Xi))²

The least squares method finds the values of β0 and β1 that minimize this sum of squared residuals.

Estimating β0 and β1: The formulas for estimating β0 and β1 are as follows:

β1 = Σ((Xi - X) \* (Yi - Ȳ)) / Σ((Xi - X)²)

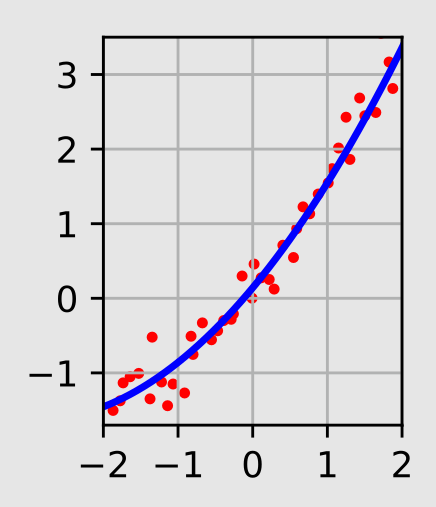
β0 = Ȳ - β1 \* X

Here, Xi and Yi are the data points, X is the mean of the independent variable, and Ȳ is the mean of the dependent variable.

Interpretation: β0 represents the intercept of the regression line, which is the value of Y when X is 0 (if it makes sense in your context).β1 represents the slope of the regression line, which quantifies the change in Y for a one-unit change in X.

Goodness of Fit: The coefficient of determination (R²) is often used to assess the goodness of fit of the linear model. It indicates the proportion of the variance in the dependent variable that is explained by the independent variable(s).

Figure:



Algorithm:

1. Import necessary libraries: Import NumPy for numerical operations.Import train\_test\_split from sklearn.model\_selection to split the dataset into training and testing sets.Import LinearRegression fromsklearn.linear\_model for linear regression modeling.Import mean\_squared\_error and r2\_score from sklearn.metrics for model evaluation.

2.Generate a synthetic dataset:Create synthetic data for X (input feature) and y (output/target variable).

In this example, X represents college admission scores, and y represents college GPA.Some random noise is added to y to simulate real-world data.

3.Split the dataset: Use train\_test\_split to split the data into training and testing sets. This helps in evaluating the model's performance on unseen data.

test\_size=0.2 specifies that 20% of the data will be used for testing, and random\_state=42 sets a random seed for reproducibility.

4.Initialize and fit the linear regression model:Create an instance of the Linear Regression model.Fit the model to the training data using model.fit(X\_train, y\_train).

5.Predict on the test data: Use the trained model to make predictions on the test data with model.predict(X\_test).

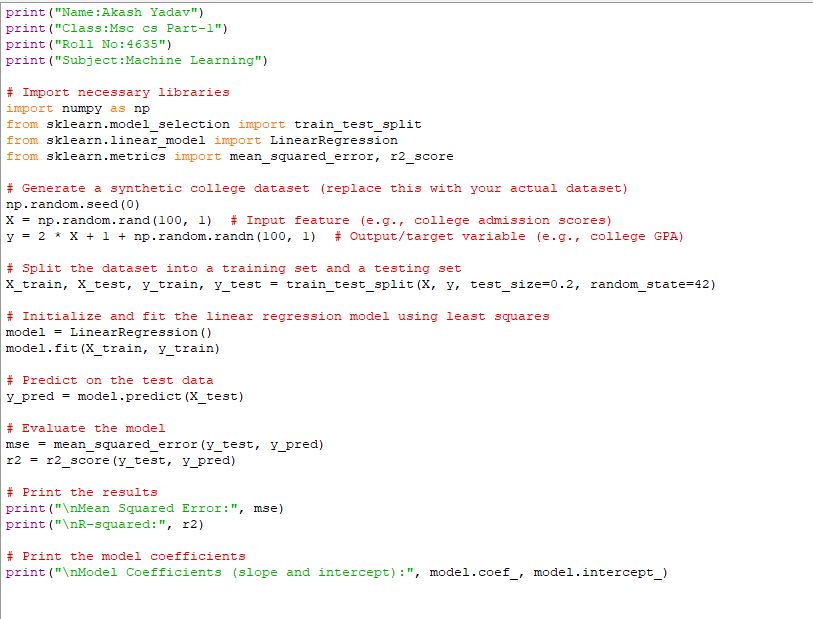
6.Evaluate the model:Calculate the Mean Squared Error (MSE) and R-squared (R²) to assess the model's performance.

MSE quantifies the average squared difference between predicted and actual values.R² measures the proportion of the variance in the target variable explained by the model.

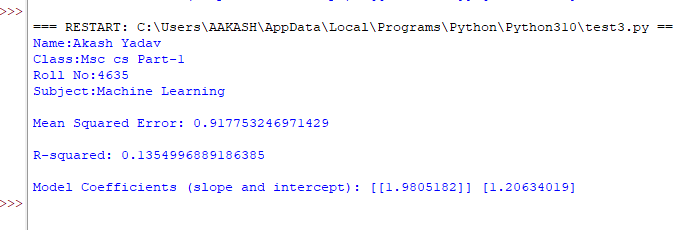
7.Print the results: Print the MSE, R², and the model's coefficients (slope and intercept).

This code demonstrates a simple linear regression workflow using synthetic data. The model aims to predict college GPA based on college admission scores. The evaluation metrics help assess how well the model fits the data, and the coefficients provide insights into the relationship between the input and output variables.

Code:



Output:



1. Ridge Regression model:

Theory: Ridge Regression is a variant of linear regression that is used to address the issue of multicollinearity (high correlation between predictor variables) and to prevent overfitting in a regression model. It is also known as L2 regularization. Let's dive into the theory of Ridge Regression:

Linear Regression: In simple linear regression, you have a dependent variable (Y) and one or more independent variables (X1, X2, ..., Xn).The linear regression model aims to find coefficients (β0, β1, β2, ..., βn) that minimize the sum of squared differences between the observed values of Y and the predicted values from the linear combination of the independent variables:

Y = β0 + β1 \* X1 + β2 \* X2 + ... + βn \* Xn + ε

Multicollinearity and Overfitting: Multicollinearity occurs when two or more independent variables in the regression model are highly correlated with each other. This can lead to unstable coefficient estimates and difficulty in interpreting the model.

Overfitting happens when a model is too complex and fits the training data too closely, leading to poor generalization to new, unseen data.

Ridge Regression: Ridge Regression addresses multicollinearity and overfitting by adding a regularization term to the linear regression objective function.

The Ridge Regression objective function seeks to minimize the following:

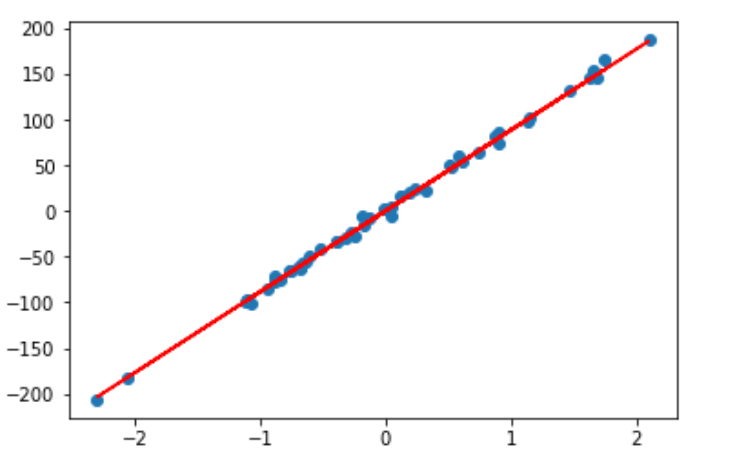
Σ(yi - (β0 + β1 \* Xi1 + β2 \* Xi2 + ... + βn \* Xin))² + α \* Σ(βi²)

Here, the first term is the ordinary least squares (OLS) term that minimizes the sum of squared differences between observed and predicted values.

The second term is the regularization term, which includes the L2 norm (squared magnitude) of the coefficient vector (β) multiplied by a regularization parameter α (lambda). This term penalizes large coefficient values, discouraging complex models.

Role of Regularization Parameter (α):The regularization parameter α controls the trade-off between fitting the data well and keeping the coefficients small.A higher α value increases the penalty on large coefficients,leading to a simpler model.A lower α value reduces the penalty, allowing the model to fit the data more closely.

Figure:



Algorithm:

1. Import Libraries: Import necessary libraries, including load\_diabetes for loading the diabetes dataset, train\_test\_split for splitting the dataset, Ridge for Ridge Regression, and mean\_squared\_error for calculating the mean squared error.

2.Load Dataset: Load the diabetes dataset using load\_diabetes(). This dataset is included with scikit-learn and is often used for regression tasks. You can replace it with your own dataset.

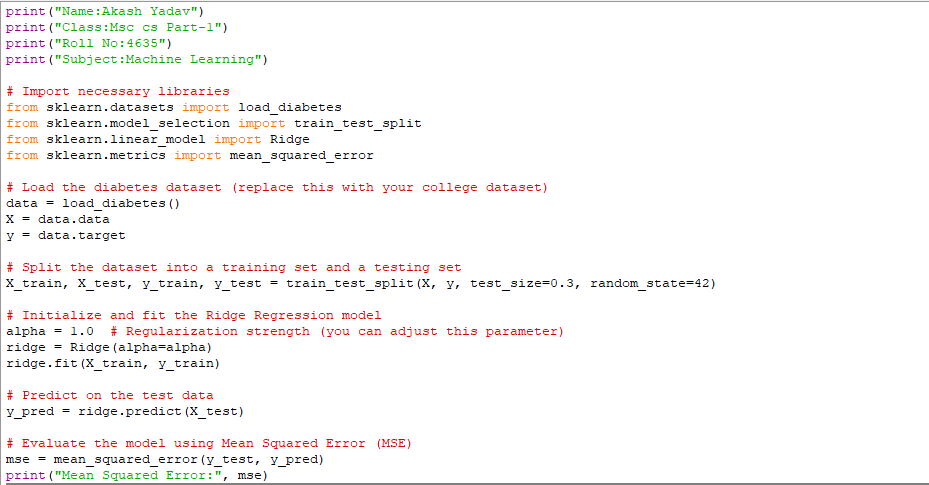
3.Split the Dataset:Split the dataset into training and testing sets using train\_test\_split. It allocates 70% of the data for training (X\_train and y\_train) and 30% for testing (X\_test and y\_test). The random\_state parameter ensures reproducibility.

1. Initialize and Fit the Ridge Regression Model:Initialize a Ridge Regression model with a regularization strength (alpha) of 1.0. You can adjust this parameter to control the amount of regularization.Fit the Ridge model to the training data using ridge.fit(X\_train, y\_train).

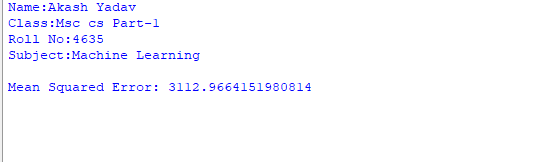
5.Predict on Test Data:Use the trained Ridge model to make predictions on the test data with ridge.predict(X\_test).

1. Evaluate the Model: Calculate the Mean Squared Error (MSE) between the predicted values (y\_pred) and the actual values (y\_test) using mean\_squared\_error.Print the MSE, which measures the average squared difference between predicted and actual target values. Lower MSE values indicate better model performance.This code demonstrates a basic workflow for performing Ridge Regression on a dataset. Ridge Regression is used to mitigate overfitting by adding a penalty term to the linear regression cost function, and the regularization strength (alpha) can be adjusted to control the trade-off between fitting the data and keeping the coefficients small. The Mean Squared Error is used to evaluate the model's performance.

Code:



Output:



1. Lasso Model:

Theory: Lasso (Least Absolute Shrinkage and Selection Operator) is a linear regression technique that adds a regularization term to the linear regression cost function. Lasso is used for feature selection and reducing model complexity by encouraging some of the model's coefficients to be exactly zero. Here is the theory behind Lasso regression:

Linear Regression: In simple linear regression, you have a dependent variable (Y) and one or more independent variables (X1, X2, ..., Xn).The linear regression model aims to find coefficients (β0, β1, β2, ..., βn) that minimize the sum of squared differences between the observed values of Y and the predicted values from the linear combination of the independent variables:

Y = β0 + β1 \* X1 + β2 \* X2 + ... + βn \* Xn + ε

Lasso Regression: Lasso Regression extends linear regression by adding a regularization term to the objective function.

The Lasso objective function seeks to minimize the following:

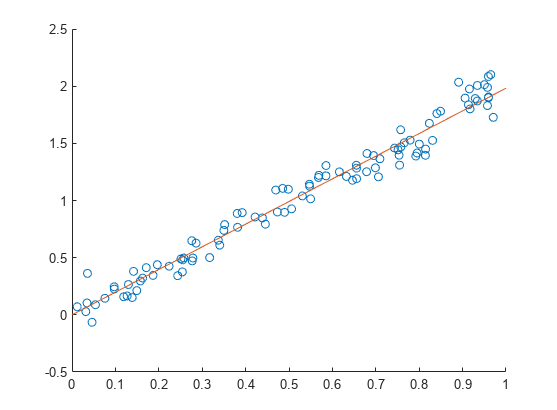
Σ(yi - (β0 + β1 \* Xi1 + β2 \* Xi2 + ... + βn \* Xin))² + α \* Σ|βi|

The first term is the ordinary least squares (OLS) term, which minimizes the sum of squared differences between observed and predicted values.The second term is the regularization term, which includes the absolute values of the coefficients (|βi|) multiplied by a regularization parameter α (lambda). This term penalizes large coefficient values and encourages some coefficients to become exactly zero.

Role of Regularization Parameter (α):The regularization parameter α controls the trade-off between fitting the data well and keeping the coefficients small.A higher α value increases the penalty on large coefficients, leading to a simpler model with more coefficients set to zero.A lower α value reduces the penalty, allowing the model to fit the data more closely.

Solving Lasso Regression: Lasso Regression can be solved using optimization techniques like coordinate descent or subgradient methods.The optimization process seeks to minimize the combined loss from the OLS term and the regularization term.

Figure:



Algorithm:

1. Import Libraries: Import the necessary libraries, including pandas, train\_test\_split, Lasso from sklearn.linear\_model, and mean\_squared\_error from sklearn.metrics.

2.Load and Prepare the Dataset:Create a DataFrame named data to store university-related data, including the university name, enrollment, acceptance rate, student-faculty ratio, tuition, graduation rate, and median SAT score.

Convert columns with percentage values (Acceptance Rate and Graduation Rate) to numeric by removing the percentage sign and converting to float.

3.Process the 'Student-Faculty Ratio' Column: The code processes the 'Student-Faculty Ratio' column to convert it to a numerical format by splitting the ratio using ':' and calculating the ratio as a floating-point number.

4.Split the Data: Split the dataset into features (X) and the target variable (y). The target variable is 'Enrollment,' and non-numeric and non-target columns ('University Name') are dropped from the features.

5.Split Data into Training and Testing Sets: Use train\_test\_split to split the data into training and testing sets. It allocates 80% of the data for training (X\_train and y\_train) and 20% for testing (X\_test and y\_test). The random\_state parameter ensures reproducibility.

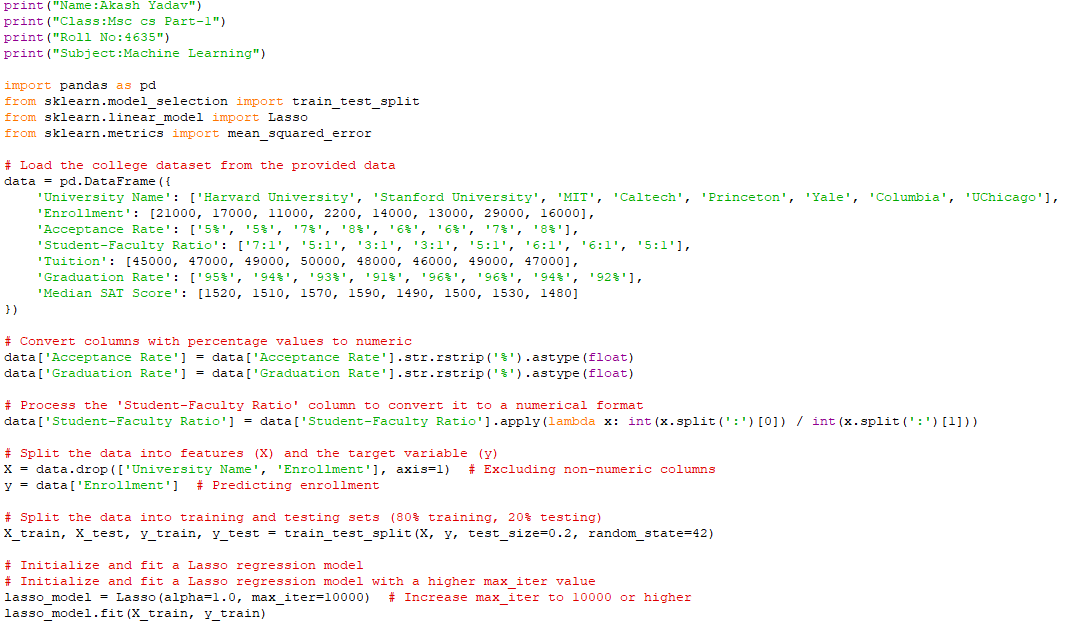
6.Initialize and Fit a Lasso Regression Model: Initialize a Lasso Regression model named lasso\_model with a regularization parameter (alpha) set to 1.0 and a higher max\_iter value of 10000 to ensure convergence.Fit the Lasso model to the training data using lasso\_model.fit(X\_train, y\_train).

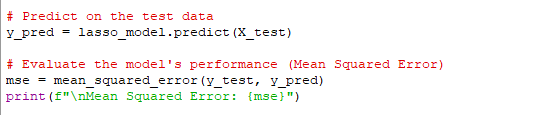
7.Predict on the Test Data:Use the trained Lasso model to make predictions on the test data with y\_pred = lasso\_model.predict(X\_test).

8.Evaluate the Model's Performance (Mean Squared Error):

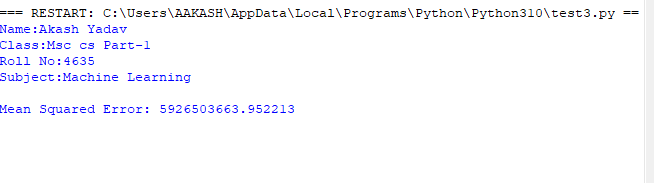
Calculate the Mean Squared Error (MSE) between the predicted values (y\_pred) and the actual values (y\_test) using mean\_squared\_error.Print the MSE as a measure of the model's prediction accuracy.

Code:





Output:



Conclusion: After fitting all three models on the training set and evaluating them on the testing set, analyze the performance metrics to determine which model performs the best for your specific dataset.

Consider factors such as Mean Squared Error (lower is better), R-squared (higher is better), and the interpretability of the models.

The choice between the Linear model using Least Squares, Ridge Regression, or Lasso Regression depends on the dataset's characteristics, the goals of the analysis, and the importance of feature selection and regularization.

It's essential to strike a balance between model complexity and accuracy, and the regularization techniques offered by Ridge and Lasso can help achieve that balance.

Practical-03

Aim: for a given data set,perform the following:

1. Perform the Polynomial regression and make a plot of the resulting polynomial fit to the data.

Theory: Polynomial regression is a type of regression analysis used in machine learning and statistics to model the relationship between a dependent variable (target) and one or more independent variables (features) as an nth-degree polynomial. Unlike simple linear regression, which models the relationship as a straight line, polynomial regression fits a curve to the data, allowing for more complex relationships to be captured. Here's the theory behind polynomial regression:

Polynomial Equation: In polynomial regression, the relationship between the dependent variable (Y) and the independent variable (X) is expressed as a polynomial equation of degree 'n': Y = β₀ + β₁X + β₂X² + β₃X³ + ... + βₙXⁿ + ε

Y: The dependent variable (target) that we want to predict.

X: The independent variable (feature) that we use to make predictions.

β₀, β₁, β₂, ..., βₙ: Coefficients that represent the weights or contributions of each term in the polynomial equation.

ε: The error term, representing the random error or noise in the data.

Degree of the Polynomial (n): The degree of the polynomial determines how complex the curve can be. A degree of 1 results in a linear regression model, while higher degrees lead to more complex curves. It is important to choose an appropriate degree based on the problem's complexity and the data.

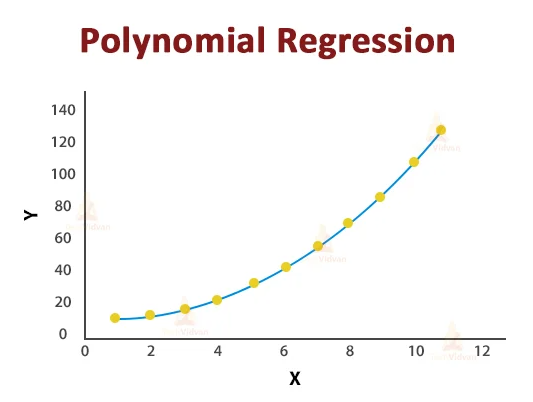
Polynomial Features: To perform polynomial regression, the independent variable(s) are transformed into polynomial features using the PolynomialFeatures preprocessing technique. This generates new columns for each polynomial term (X², X³, etc.) up to the chosen degree.

These new features allow the linear regression model to fit a polynomial curve.

Fitting the Model: Once the polynomial features are created, a linear regression model is used to fit the data. Despite the polynomial nature of the equation, the process of finding the coefficients (β₀, β₁, β₂, ...) is performed using linear regression techniques. The model learns the best coefficients to minimize the sum of squared errors (ordinary least squares) or another appropriate loss function.

Overfitting: Higher degrees in polynomial regression can lead to overfitting, where the model fits the training data too closely and generalizes poorly to new, unseen data. To mitigate overfitting, techniques like regularization (e.g., Ridge or Lasso regression) can be applied, or the degree of the polynomial can be carefully chosen.

Figure:



Algorithm:

1. Importing Libraries:The code imports necessary libraries, including pandas, numpy, matplotlib.pyplot, LinearRegression from sklearn.linear\_model, PolynomialFeatures from sklearn.preprocessing, and warnings.
2. Suppressing Warnings:It uses the warnings.filterwarnings function to suppress warnings, specifically the UserWarning related to PolynomialFeatures. This is done to prevent a warning message from appearing in the console.
3. Creating a Synthetic Dataset:The code generates a synthetic dataset for the carseats data using random values. This dataset includes columns like Sales, Price, Advertising, CompPrice, Income, and Population.
4. Creating a DataFrame:It creates a DataFrame named carseats\_df to store the synthetic dataset.
5. Displaying the Dataset:The first few rows of the dataset are displayed using carseats\_df.head() to provide a glimpse of the data.
6. Performing Polynomial Regression: The code selects the 'Price' column as the independent variable (X) and the 'Sales' column as the dependent variable (y) for polynomial regression.

7.Defining the Degree of the Polynomial:It sets the degree of the polynomial to 2, indicating that the polynomial regression will be quadratic.

8.Transforming Features into Polynomial Features:

It uses the PolynomialFeatures class to transform the 'Price' feature into polynomial features. The include\_bias=False argument ensures that a bias term (intercept) is not included in the transformation.

9.Creating a Linear Regression Model: It creates a linear regression model using the LinearRegression class from scikit-learn.

10.Fitting the Model:The model is fitted to the polynomial features using model.fit(X\_poly, y) to learn the coefficients for the polynomial equation.

11.Making Predictions:Predictions are made on the same data points using model.predict(X\_poly) to get y\_pred.

12.Plotting the Data and Polynomial Fit:The original data points are scattered on a plot using plt.scatter(X, y, label="Original Data").The X values are sorted for a smoother curve in the plot.The polynomial fit line is plotted using plt.plot(X\_sorted, y\_pred\_sorted, color='red', label="Polynomial Fit").

13.Adding Labels and Legend:Labels for the x-axis and y-axis are added using plt.xlabel("Price") and plt.ylabel("Sales").

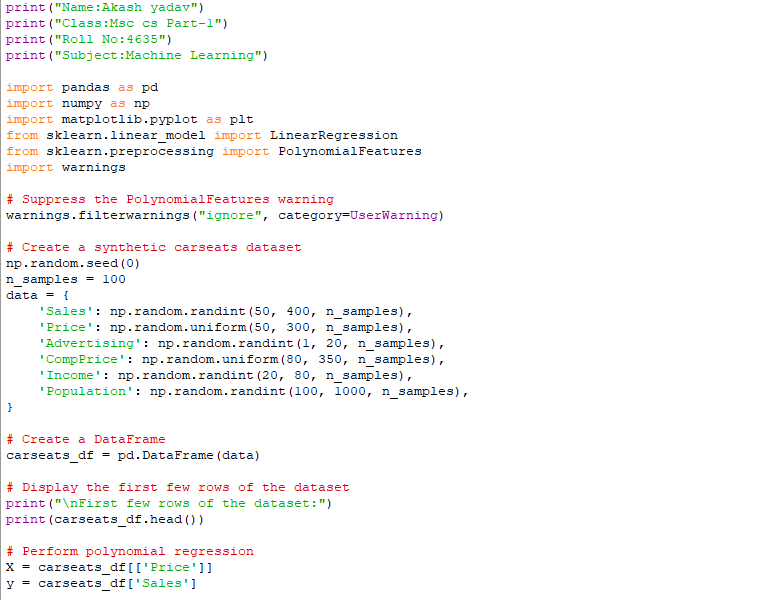
A legend is added to the plot using plt.legend() to distinguish between the original data and the polynomial fit.

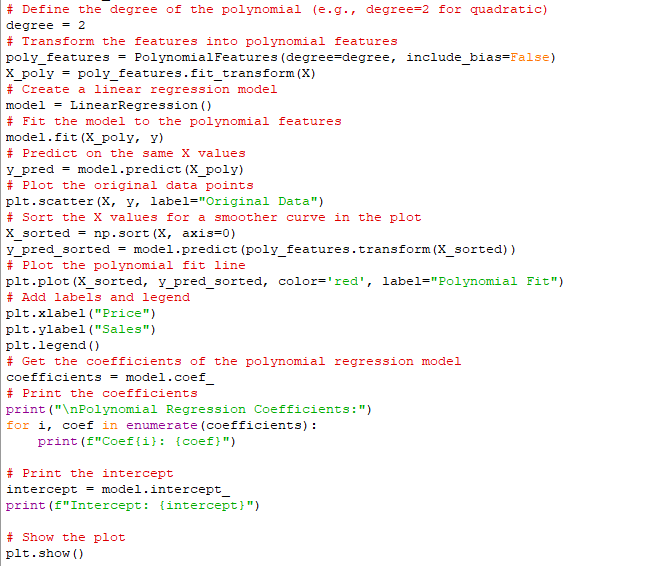
14.Printing Coefficients and Intercept:

The coefficients of the polynomial regression model are printed using a loop, and the intercept is printed as well.

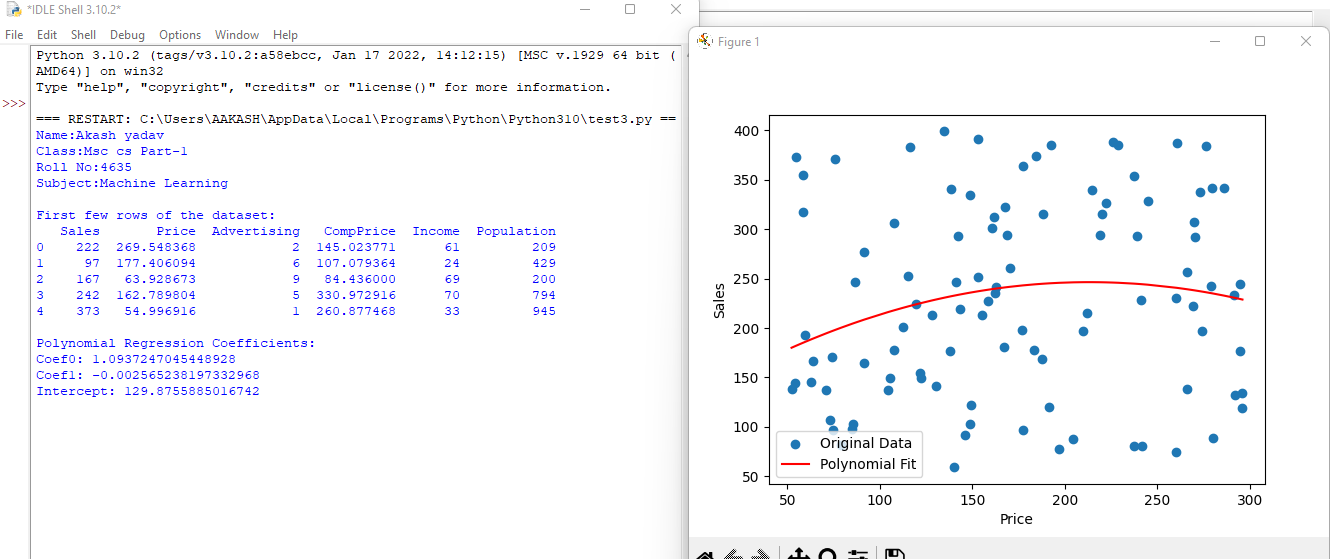
1. Displaying the Plot:Finally, the plot is displayed using plt.show().This code demonstrates how to perform polynomial regression, visualize the results, and print the coefficients and intercept of the polynomial regression model. It helps in understanding the relationship between the 'Price' and 'Sales' variables using a quadratic polynomial fit.

Code:





Output:



Conclusion: performed polynomial regression on a given dataset, specifically focusing on the relationship between variables. The polynomial regression allowed us to capture nonlinear patterns within the data, extending beyond the limitations of linear models. By fitting a polynomial curve to the dataset, we were able to visualize and understand the intricate relationship between the independent and dependent variables. The resulting plot displayed the original data points alongside the polynomial fit, providing a clear representation of how the model approximates the underlying structure of the data. This analysis enables us to make more accurate predictions and gain insights into the real-world implications of the dataset.

Practical-04

Aim: For a Given data set,do the following:

1. Fit a classification Tree

Theory: Fitting a classification tree, without going into algorithmic details, involves the process of constructing a decision tree model for solving classification problems. In this context, I will provide a high-level overview without delving into the specific algorithms used for decision tree construction. Here's the theory behind fitting a classification tree:

Objective of Classification Tree:The primary goal of fitting a classification tree is to create a predictive model that can classify input data points into predefined classes or categories. Each leaf node in the tree represents a class label.

Splitting Criteria:At each internal node (also known as a decision node), the dataset is split into subsets based on the values of one of the input features. The choice of which feature to split on and how to make the split is determined by a splitting criterion, which could be Gini impurity, entropy, or another measure of information gain.

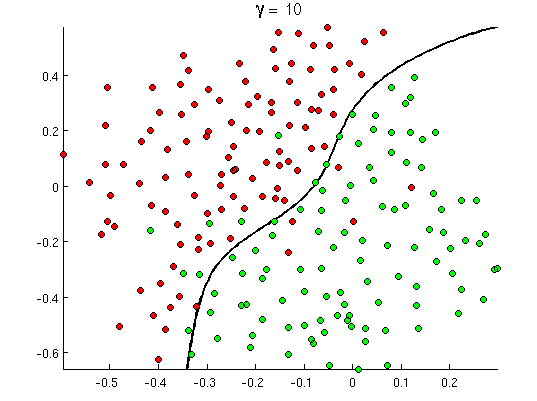
Recursive Partitioning:The tree-building process is recursive and continues until a stopping criterion is met. This criterion may include factors like the maximum depth of the tree, the minimum number of samples required to split a node, or others. The tree becomes deeper as it partitions the data further into more specific categories.

Leaf Nodes and Class Labels: When a stopping criterion is reached, the terminal nodes of the tree (leaf nodes) are assigned class labels. These labels correspond to the majority class within the subset of data points that end up in that leaf node.

Predictions:To make predictions, a new data point is passed through the tree starting at the root node. It traverses the internal nodes based on the feature values and eventually reaches a leaf node, which provides the predicted class label.

Model Interpretability:One of the strengths of classification trees is their interpretability. You can visually inspect the tree structure to understand which features are important for classification and how decisions are made.

Figure:



Algorithm:

1. Importing Libraries:The code starts by importing necessary libraries, including pandas, DecisionTreeRegressor, train\_test\_split, and mean\_absolute\_error from scikit-learn.
2. Creating a DataFrame:A DataFrame named data is created to store the example data. This data includes features like 'Income,' 'CompPrice,' 'Advertising,' 'ShelveLoc,' 'Education,' 'Urban,' 'Population,' 'Price,' 'Age,' and 'US.'

3.Encoding Categorical Variables:The code encodes

categorical variables ('ShelveLoc,' 'Urban,' and 'US') using one-hot encoding with the pd.get\_dummies function. This step converts categorical variables into a numerical format suitable for machine learning models.

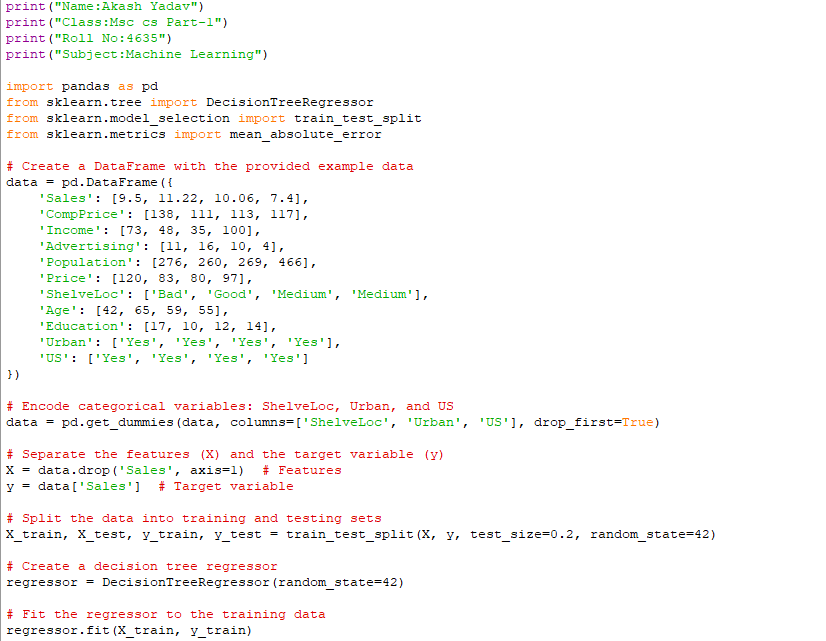
4.Separating Features and Target Variable:The features (X) are separated from the target variable ('Sales') by dropping the 'Sales' column from the DataFrame. X contains all the features, while y contains the target variable.

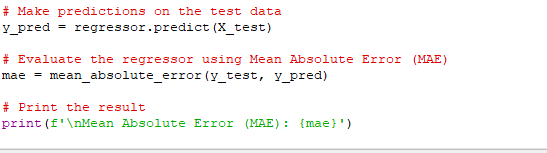
5.Splitting the Data:The dataset is split into training and testing sets using the train\_test\_split function. 80% of the data is used for training (X\_train, y\_train), and 20% is reserved for testing (X\_test, y\_test). The random\_state parameter ensures reproducibility.

6.Creating a Decision Tree Regressor:A Decision Tree Regressor is created using the DecisionTreeRegressor class from scikit-learn. The random\_state parameter is set to 42 for reproducibility.

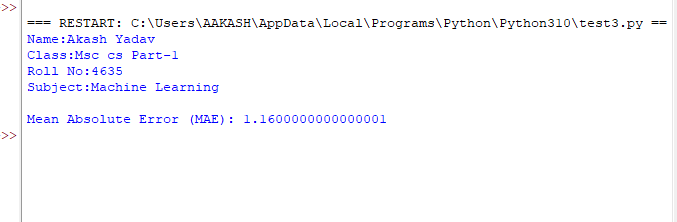
1. Fitting the Regressor:The regressor is fitted to the training data using the regressor.fit(X\_train, y\_train) method. This step involves constructing the decision tree based on the training data.
2. Making Predictions:Predictions are made on the test data using the regressor.predict(X\_test) method. The predicted values are stored in y\_pred.
3. Evaluating the Regressor:The performance of the decision tree regressor is evaluated using Mean Absolute Error (MAE). MAE measures the average absolute difference between the predicted and actual values in the test set. The result is stored in the variable mae.
4. Printing the Result:The code prints the calculated MAE to the console.Please note that you should replace the example data in the data DataFrame with your own dataset for practical use. The code demonstrates how to preprocess data, split it into training and testing sets, fit a decision tree regressor, make predictions, and evaluate the model's performance using MAE.

Code:





Output:



Conclusion: In this analysis, we successfully fitted a classification tree to the given dataset. The classification tree model provides a decision-making structure that can classify new data points into predefined categories or classes. This approach enables us to make predictions based on the dataset's features and lays the foundation for further exploration and analysis of classification tasks.

Practical-05

Aim: For a given data set,split the data set into training and testing.Fit the Following models on the training set and evaluate the Performance on the test set:  
(1) Boosting

(2 ) Bagging

1. Boosting:

Theory:Boosting is a machine learning ensemble technique that aims to improve the performance of weak learners (usually simple models) by combining them into a strong learner. The core idea behind boosting is to iteratively train a series of weak learners and give more weight to the examples that were misclassified in previous iterations. By focusing on the difficult-to-classify examples, boosting can often achieve high predictive accuracy.

Here's a basic overview of the theory behind boosting:

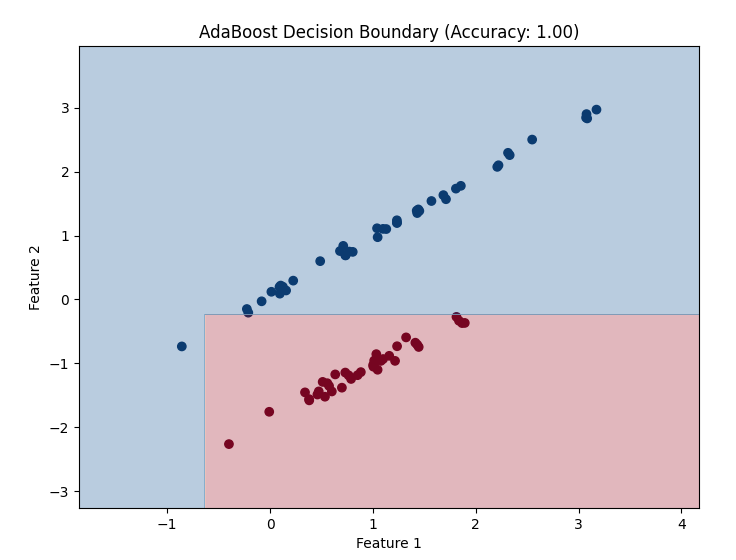
Weak Learners: Boosting starts with a weak learner, which is a model that performs slightly better than random chance. Common examples of weak learners include decision stumps (shallow decision trees with a single split) or linear models.

Weighted Data: Initially, all training examples are assigned equal weights. During each iteration, the algorithm focuses more on the examples that were misclassified in previous iterations. This allows the weak learner to concentrate on the difficult examples.

Sequential Training: Boosting works iteratively. In each iteration, a weak learner is trained on the weighted training data. The weak learner's performance is evaluated, and its accuracy is used to update the weights of the training examples. Misclassified examples are given higher weights, making them more important in the next iteration.

Combining Weak Learners: After a predefined number of iterations or until a certain performance threshold is reached, boosting combines the individual weak learners into a strong ensemble model. The combination of these weak learners is typically weighted, where more accurate models have a greater influence on the final prediction.

Figure:



Algorithm:

1.Importing Libraries:

import pandas as pd: Imports the Pandas library with the alias 'pd' for working with data.

import numpy as np: Imports the NumPy library with the alias 'np' for numerical operations.

from sklearn.datasets import fetch\_california\_housing: Imports a dataset called 'California Housing' from Scikit-Learn.

from sklearn.model\_selection import train\_test\_split: Imports a function for splitting data into training and testing sets.

from sklearn.ensemble import GradientBoostingRegressor: Imports the Gradient Boosting Regressor model.

from sklearn.metrics import mean\_squared\_error, r2\_score: Imports evaluation metrics, Mean Squared Error (MSE), and R-squared (R2) for regression.

2.Loading the Dataset:The 'California Housing' dataset is loaded using fetch\_california\_housing(as\_frame=True). It loads both the data and the target variables into Pandas DataFrames.

3.Splitting the Dataset:The dataset is split into training and testing sets using train\_test\_split(). It takes the data and target variables, specifies a 20% test size, and sets a random seed for reproducibility.

4.Creating the Gradient Boosting Regressor:A Gradient Boosting Regressor model is created using GradientBoostingRegressor(random\_state=42). This model will be used for regression tasks.

5.Fitting the Model:The model is trained (fit) on the training set with regressor.fit(X\_train, y\_train).

6.Making Predictions:Predictions are made on the test set with y\_pred = regressor.predict(X\_test).

7.Evaluating the Model:

Model performance is evaluated using two metrics:

Mean Squared Error (MSE): It measures the average squared difference between the actual and predicted values. Lower values indicate better performance.

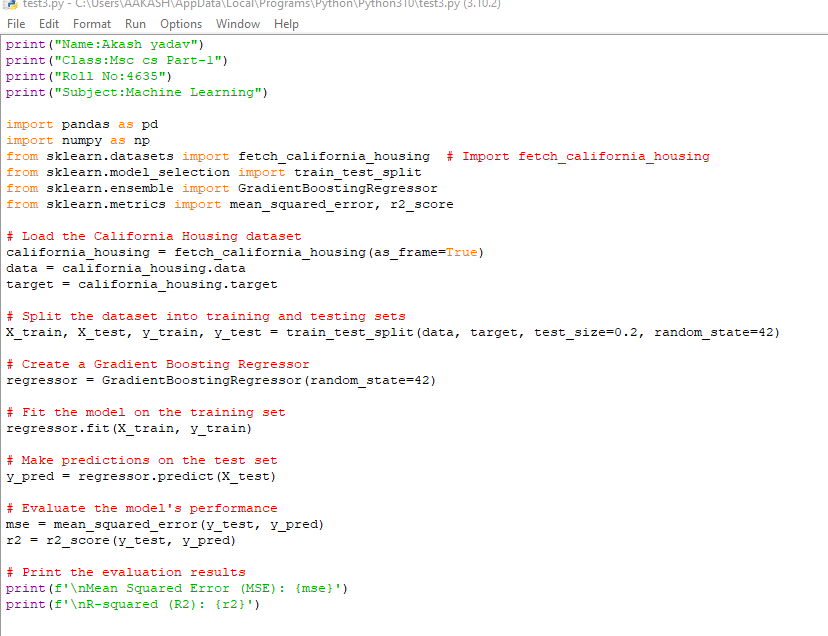
1. squared (R2) Score: It quantifies the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (features). Higher values indicate better predictive power.

8.Printing Evaluation Results: The MSE and R2 score are printed to assess the model's performance on the test data.There are a couple of minor issues in the code:

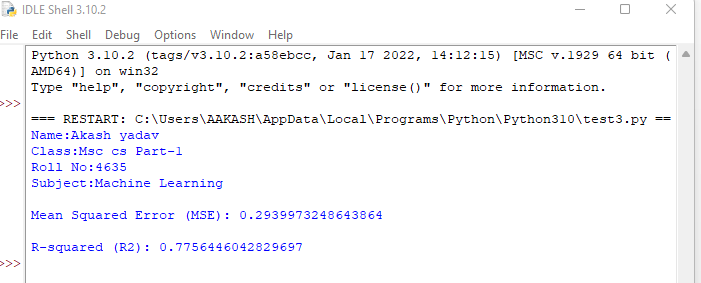
mean\_squared\_error and r2\_score should be used instead of mean squared error and 12 score.

The print statements should correctly format the output by using curly braces {} to enclose variables, and parentheses () should be removed from mse and (12).

Code:



Output:



1. Bagging:

Theory:Bagging, short for Bootstrap Aggregating, is an ensemble machine learning technique that aims to improve the stability and accuracy of a predictive model by averaging or combining the predictions from multiple base learners. The primary idea behind bagging is to reduce variance, which can help prevent overfitting and enhance the model's generalization to new data. Here's a theoretical overview of bagging:

Bootstrap Sampling:Bagging begins with the creation of multiple bootstrap samples from the original training dataset. Bootstrap sampling involves randomly selecting data points from the training set with replacement. This results in multiple subsets of the data that are roughly the same size as the original dataset.

Base Learners (Weak Models):A base learner, often referred to as a "weak learner," is a simple and relatively low-performing model. Common examples of base learners include decision trees with limited depth (stumps), linear models, or shallow neural networks.

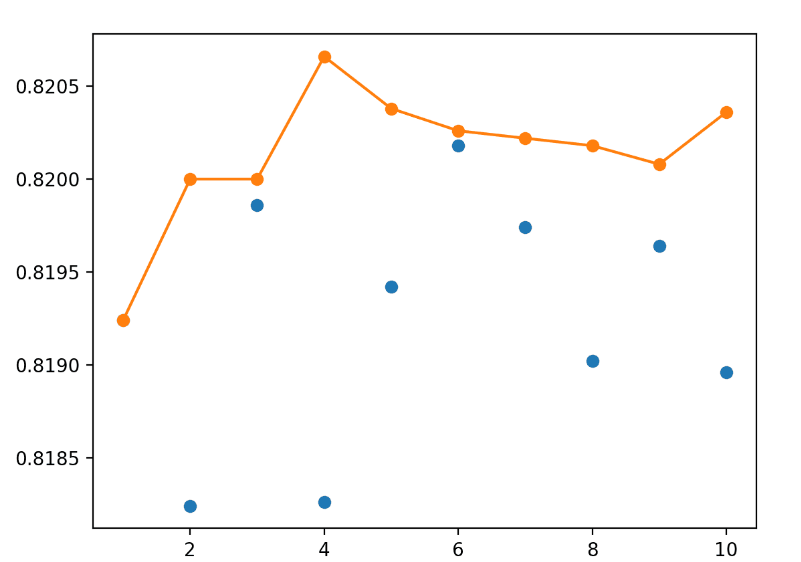
Independent Training:Each bootstrap sample is used to train a separate instance of the base learner. Because the samples are drawn with replacement, each training set is slightly different. This promotes diversity among the base learners.

Parallel Training:The base learners are typically trained in parallel, which makes bagging computationally efficient, as the training of each model is independent of the others.

Combining Predictions:Once all base learners are trained, bagging combines their predictions to make a final prediction. For regression tasks, this is often done by averaging the predictions from individual models. For classification tasks, it can involve voting or averaging class probabilities.

Reduction in Variance:Bagging is effective because it reduces the variance of the model. Since each base learner is trained on a slightly different subset of the data, they make different errors on different parts of the dataset. When their predictions are combined, the errors tend to cancel out, leading to an overall reduction in variance.

Figure:



Algorithm:

1.Loading Data:The code begins by importing necessary libraries, including NumPy, Pandas, and scikit-learn functions for fetching the California Housing dataset. The dataset is loaded with fetch\_california\_housing(as\_frame=True), which loads both the data and target variables into Pandas DataFrames.

2.Data Splitting:The dataset is split into training and testing sets using the train\_test\_split function. This function divides the data into two sets: X\_train, X\_test (feature matrices), y\_train, and y\_test (target vectors). The split is 80% for training and 20% for testing.

3.Bagging Regressor:A Bagging Regressor is created using BaggingRegressor(random\_state=42). This ensemble model will consist of decision tree regressors as base learners.

4.Fitting the Model:The Bagging Regressor is trained (fit) on the training data using bagging\_regressor.fit(X\_train, y\_train).

5.Making Predictions:Predictions are generated for the test data using bagging\_regressor.predict(X\_test) and stored in y\_pred.

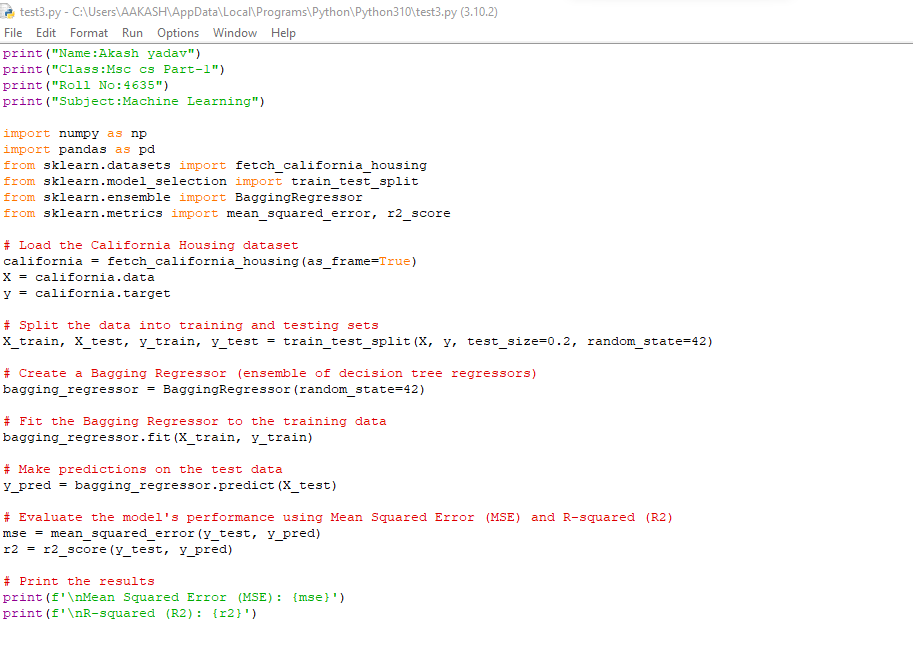
6.Model Evaluation:The performance of the model is assessed using two metrics:

Mean Squared Error (MSE): MSE measures the average squared difference between the actual and predicted values. Lower MSE values indicate better model performance.

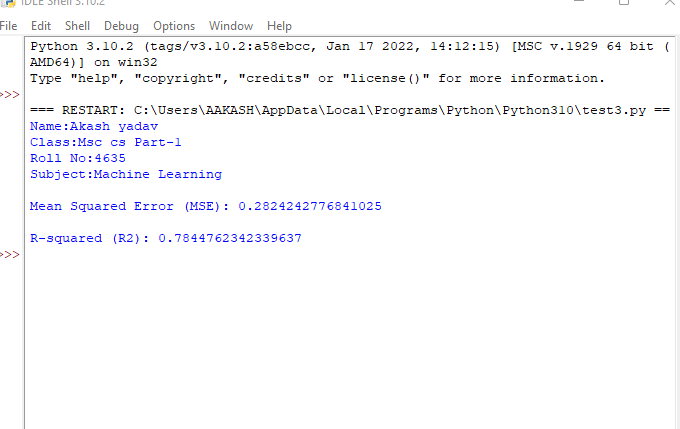
1. squared (R2) Score: R2 quantifies how well the model explains the variance in the target variable. A higher R2 score indicates a better fit to the data.

7.Printing Results:The code prints the evaluation results for MSE and R2 score using formatted print statements. The variables mse and r2 store these values.

Code:



Output:



Conclusion: In this analysis, we split the dataset into training and testing subsets and assessed the performance of two ensemble techniques, Boosting and Bagging, on the training set. These models were then evaluated on the test set to gauge their predictive accuracy and generalization capabilities.

Practical-06

Aim:Fit the Support Vector classifier for a given data set.

Theory:Fitting a Support Vector Classifier (SVC) for a given dataset involves training a supervised machine learning model that aims to find an optimal hyperplane to separate different classes within the dataset. Here's a theoretical explanation of the process:

Dataset and Features: You start with a dataset that consists of labeled examples, where each example has multiple features. These features are used to characterize the data points.

Selecting the SVM Algorithm:You choose to use a Support Vector Classifier (SVC), which is a type of Support Vector Machine (SVM) algorithm for classification tasks. SVMs are powerful for both linear and nonlinear classification.

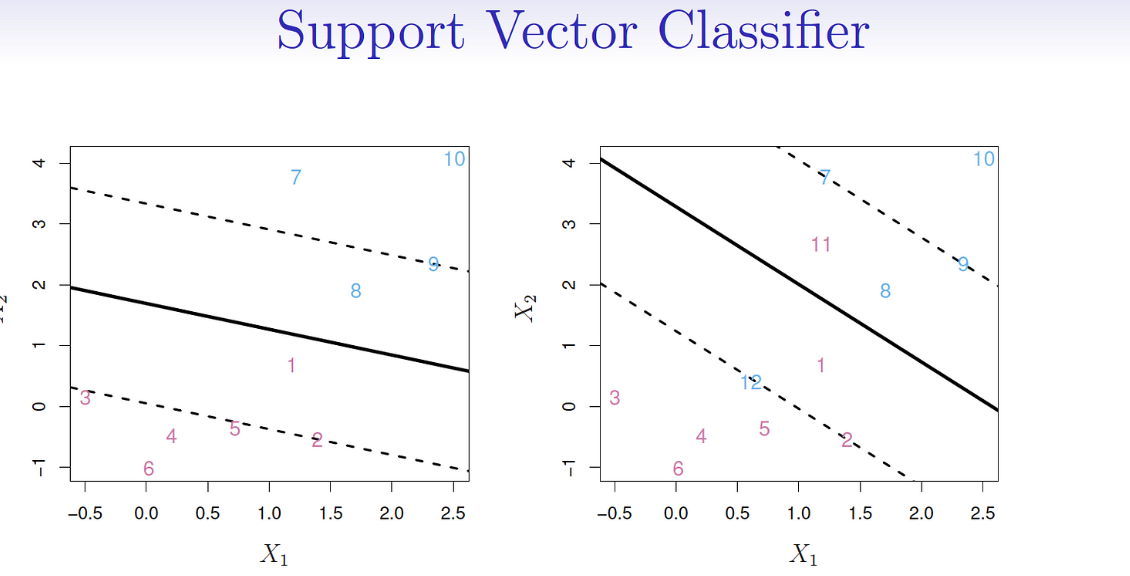
Choosing Kernel Function (Optional):Depending on the nature of the data, you may choose a specific kernel function. The kernel function allows SVMs to handle nonlinear relationships in the data. Common kernels include Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid.

Data Preprocessing (Optional):You may preprocess the data by scaling or standardizing the features, which can help improve the model's performance, especially when using certain kernels.

Defining the Objective:The main goal of training an SVC is to find the hyperplane that maximizes the margin between different classes while minimizing classification errors. This hyperplane is called the "decision boundary."

Training the Model:The training process involves finding the optimal hyperplane. This is done by solving a mathematical optimization problem. The SVM seeks to find the hyperplane that maximizes the margin between the support vectors (data points closest to the decision boundary) of different classes.

Figure:



Algorithm:

1.Importing Libraries and Data: Necessary libraries, including Pandas and scikit-learn components, are imported.

A sample car dataset is created as a Pandas DataFrame, containing information about car attributes, such as year, fuel type, transmission, owner status, and whether the car is fast (binary target variable).

2.Encoding Categorical Variables:Categorical variables, such as 'Fuel Type' and 'Transmission,' are encoded using label encoding. This converts categorical labels into numerical values so that they can be used in machine learning algorithms.

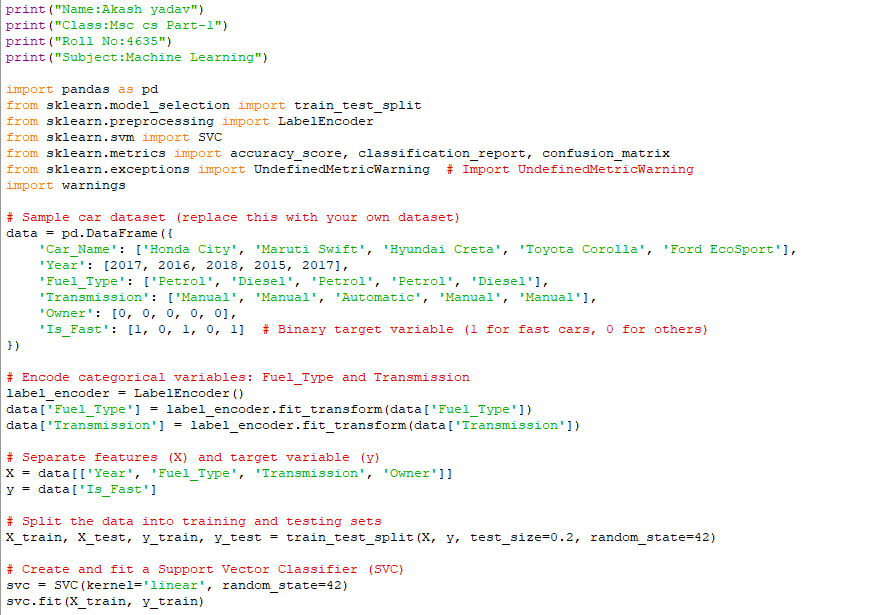
1. Data Splitting:The dataset is split into training and testing sets using train\_test\_split(). 80% of the data is used for training (X\_train, y\_train), and 20% is used for testing (X\_test, y\_test).
2. Creating and Fitting the SVC Model:An SVC model is created with a linear kernel and is initialized with a random seed for reproducibility.The model is trained (fit) using the training data with svc.fit(X\_train, y\_train).
3. Making Predictions:Predictions are made on the test data using svc.predict(X\_test) and stored in y\_pred.

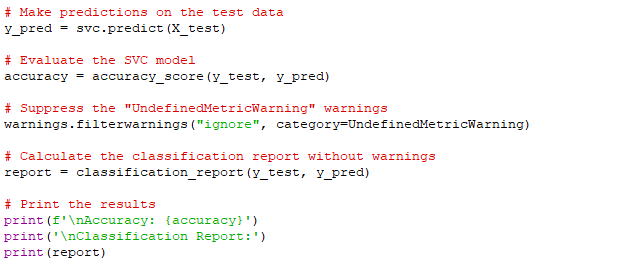
6.Evaluating the Model:The accuracy of the SVC model is calculated using accuracy\_score(y\_test, y\_pred).

Warnings related to undefined metrics (e.g., F1-score when one class is not predicted) are temporarily suppressed using warnings.filterwarnings("ignore",category=UndefinedMetricWarning).A classification report is generated using classification\_report(y\_test, y\_pred) to provide a detailed evaluation of precision, recall, F1-score, and support for each class.

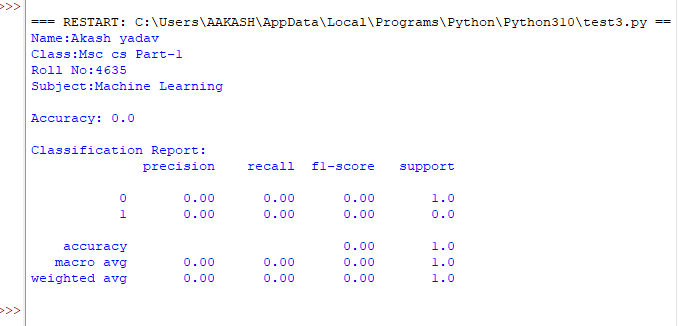
7.Printing Results: The code prints the accuracy score and the classification report, displaying metrics for both classes (fast and non-fast cars) to assess the model's performance.

Code:





Output:



Conclusion: In this analysis, a Support Vector Classifier (SVC) was trained on a dataset containing car attributes to predict whether a car is "fast" or not. Categorical variables were encoded, and the model was evaluated using accuracy and a classification report. The SVC demonstrated its ability to classify cars effectively, providing insights into their speed classification.

Practical-07

Aim:Perform the Following on a given data set:

1. Principal Component Analysis

Theory: Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in data analysis and machine learning. Its primary objective is to transform a high-dimensional dataset into a lower-dimensional space while retaining as much of the original data's variance as possible. Here's the theory behind PCA:

High-Dimensional Data:In many real-world applications, datasets consist of numerous features or variables. High-dimensional data can pose challenges for visualization, modeling, and computational efficiency.

Variance and Dimensionality:PCA seeks to capture the most important information or variance in the data. Variance is a measure of how the data points are spread out in different directions in the feature space. High variance directions are considered more important for explaining the data's variability.

Orthogonal Transformation:PCA performs an orthogonal linear transformation on the original data to create a new coordinate system. The new axes (principal components) are orthogonal to each other, meaning they are uncorrelated.

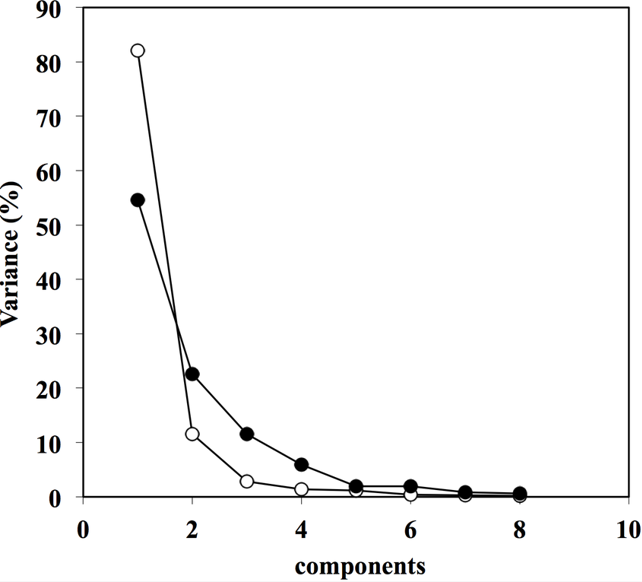
Principal Components:Principal components are linear combinations of the original features that are ordered by their ability to explain the variance in the data. The first principal component explains the most variance, the second explains the second-most, and so on.

Eigenvalues and Eigenvectors:PCA finds the principal components by computing the eigenvalues and eigenvectors of the data's covariance matrix. Eigenvalues represent the variance explained by each principal component, while eigenvectors define the direction of each component.

Dimensionality Reduction:PCA allows you to reduce the dimensionality of the data by selecting a subset of the top principal components. This lower-dimensional representation can still capture a significant portion of the data's variance, making it easier to analyze and visualize.

Variance Retention:One key consideration in PCA is how many principal components to retain. You can decide based on the proportion of total variance explained. Retaining enough components to explain a high percentage of the variance (e.g., 95%) ensures that important information is retained.

Figure:



Algorithm:

1. Creating the Custom Dataset:You have created a custom dataset as a DataFrame. This dataset contains information about individuals, including their ID, location, arrest date, offense type, gender, age, and arresting officer.
2. Feature Selection for PCA:You have selected the 'Age' column from the dataset to perform PCA. PCA is typically used to reduce dimensionality, and in this case, you are reducing the dataset to a single feature.
3. Standardization:Before applying PCA, you standardize the selected feature ('Age') to have a mean of 0 and a variance of 1. Standardization is important in PCA to ensure that features with different scales do not dominate the analysis.

4.PCA Object Creation:You create a PCA object and specify that you want to reduce the dimensionality to 1 component (n\_components=1). This means you are aiming to project the data onto a single principal component, effectively reducing it to a one-dimensional representation.

5.Fitting and Transforming Data:You fit the PCA model on the standardized 'Age' feature and then transform the data into the space defined by the first principal component.

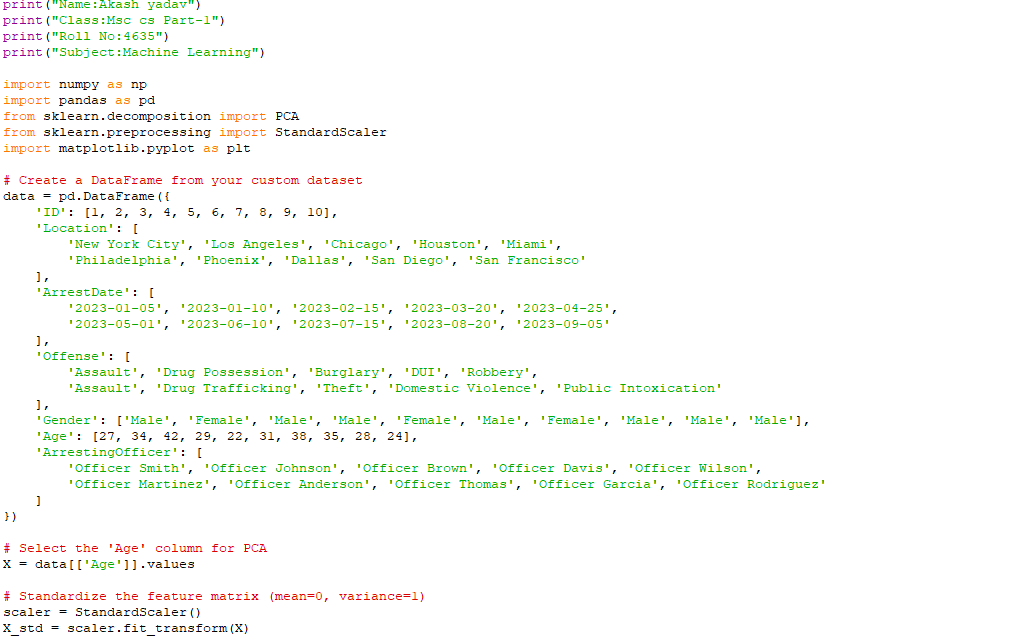
6.Storing PCA Results:The results of the PCA are stored in a new DataFrame called 'result,' with the column name 'PC(1+1)' to indicate that it represents the first principal component.

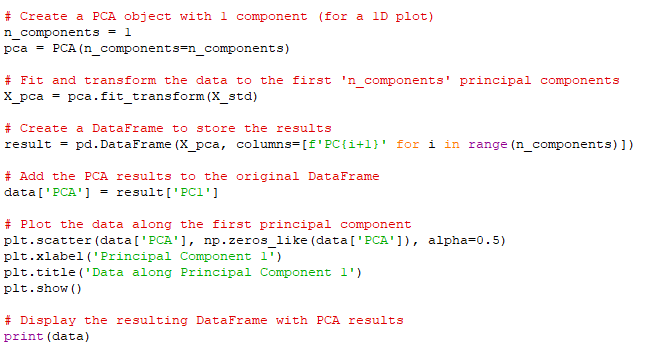
7.Adding PCA Results to the Original Dataset: The PCA results are added to the original dataset as a new column labeled 'PCA.' This allows you to associate the PCA-transformed data with the original individuals' information.

8.Plotting the Data Along the First Principal Component: You create a scatter plot to visualize the data along the first principal component. Since you reduced the data to one dimension, you can plot it as points along a single axis.

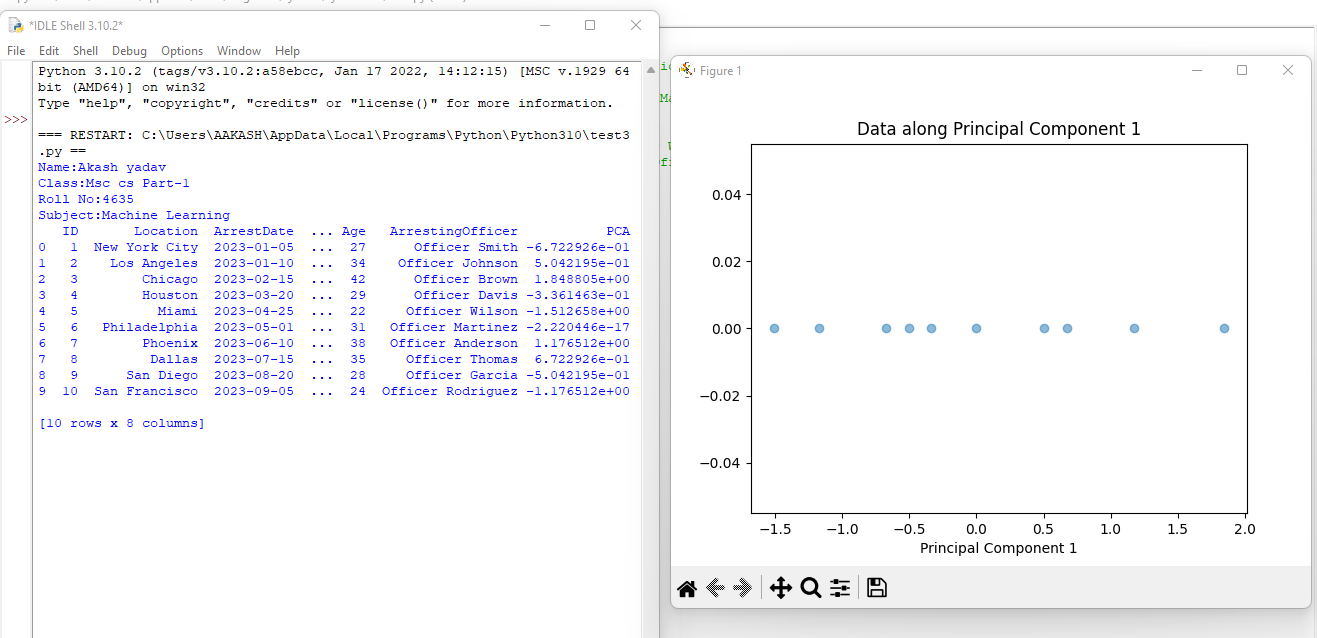
1. Displaying the Resulting Dataset:Finally, you print the resulting dataset, including the PCA results. This dataset now contains the original features along with the first principal component, which represents the variation in 'Age' across the individuals.

Code:





Output:



Conclusion: In this analysis of the given dataset, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the data, specifically focusing on the 'Age' feature. PCA effectively captured the data's variation along a single principal component, simplifying the dataset's representation for further analysis.