****

PROGRAM 1: Linear Regression for the diabetes dataset

REGNO:24251115

DATE: 23-05-2025

****

```
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_df



# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
```

r2 = r2_score(y_test, y_pred)

Output results

print("Coefficients:", model.coef_)

print("Intercept:", model.intercept_)

print("Mean Squared Error:", mse)

print("R² Score:", r2)

OUTPUT:

```
model = LinearRegression()
```

```
model
```

```
▾ LinearRegression
LinearRegression()
```

	Actual	Predicted
304	253.0	117.614637
358	90.0	48.721858
128	115.0	90.586456
373	168.0	143.619232
9	310.0	212.757922
...	...	...
129	268.0	214.506193
63	128.0	96.055855
342	178.0	159.934376
291	248.0	194.817561
329	135.0	104.155230

89 rows × 2 columns

```
print("Coefficients:", model.coef_)
```

```
Coefficients: [  37.90402135 -241.96436231  542.42875852  347.703
84391 -931.48884588
  518.06227698  163.41998299  275.31790158  736.1988589    48.670
65743]
```

```
print("Intercept:", model.intercept_)
```

```
Intercept: 151.34560453985995
```

```
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 2900.1936284934804
```

```
print("R² Score:", r2)
```

```
R² Score: 0.4526027629719196
```

****
PROGRAM 2: Logistic Regression for the breast cancer dataset

REGNO: 24251115

DATE: 23-05-2025
****

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load the dataset
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
model = LogisticRegression(max_iter=10000)  # max_iter increased to ensure convergence
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
comparison_df = pd.DataFrame(
    {
        'Actual': y_test,
        'Predicted': y_pred
    }
)
```

comparison_df

Evaluate the model

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))

OUTPUT:

```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

```
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

	Actual	Predicted
0	1	1
1	0	0
2	0	0
3	1	1
4	1	1
...	...	...
109	1	1
110	0	0
111	1	1
112	1	0
113	0	0

114 rows × 2 columns

```
Confusion Matrix:
 [[39  4]
 [ 1 70]]

Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Accuracy Score: 0.956140350877193
```

****
PROGRAM 3: Classification matrix for the Decision Tree classifier for iris dataset

REGNO:24251115

DATE: : 23-05-2025
****

```
sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Evaluate using confusion matrix
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target_names)
accuracy = accuracy_score(y_test, y_pred)
```

Output

print("Confusion Matrix:\n", cm)

print("\nClassification Report:\n", report)

print("Accuracy Score:", accuracy)

OUTPUT:

```
clf = DecisionTreeClassifier(random_state=42)
```

```
clf
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Accuracy Score: 1.0
```

****

PROGRAM 4: Classification model for Wine dataset using RandomForestClassifier

REGNO: 24251115

DATE: : 23-05-2025

****

```
from sklearn.datasets import load_wine
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
wine = load_wine()
X = wine.data
y = wine.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=wine.target_names)
```

conf_matrix = confusion_matrix(y_test, y_pred)

Display results

print("Accuracy:", accuracy)

print("Classification Report:\n", report)

print("Confusion Matrix:\n", conf_matrix)

OUTPUT:

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
clf
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        14
     class_1       1.00      1.00      1.00        14
     class_2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36

Confusion Matrix:
 [[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
```

****
PROGRAM 5: Classification model for Wine dataset using K-Nearest Neighbors Classifier.

REGNO: 24251115

DATE: : 23-05-2025
****

```
# Import necessary libraries
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Step 1: Load the dataset
wine = load_wine()
X = wine.data
y = wine.target

# Step 2: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
```

```
# Step 5: Make predictions
y_pred = knn.predict(X_test_scaled)

# Step 6: Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

OUTPUT:

```
knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
Confusion Matrix:
[[14  0  0]
 [ 1 12  1]
 [ 0  0  8]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.97        14
           1       1.00      0.86      0.92        14
           2       0.89      1.00      0.94         8

    accuracy                           0.94        36
   macro avg       0.94      0.95      0.94        36
weighted avg       0.95      0.94      0.94        36

Accuracy Score: 0.9444444444444444
```

****

PROGRAM 6: Regression model for Fetch House Pricing using RandomForestRegressor

REGNO: 24251115

DATE: : 23-05-2025

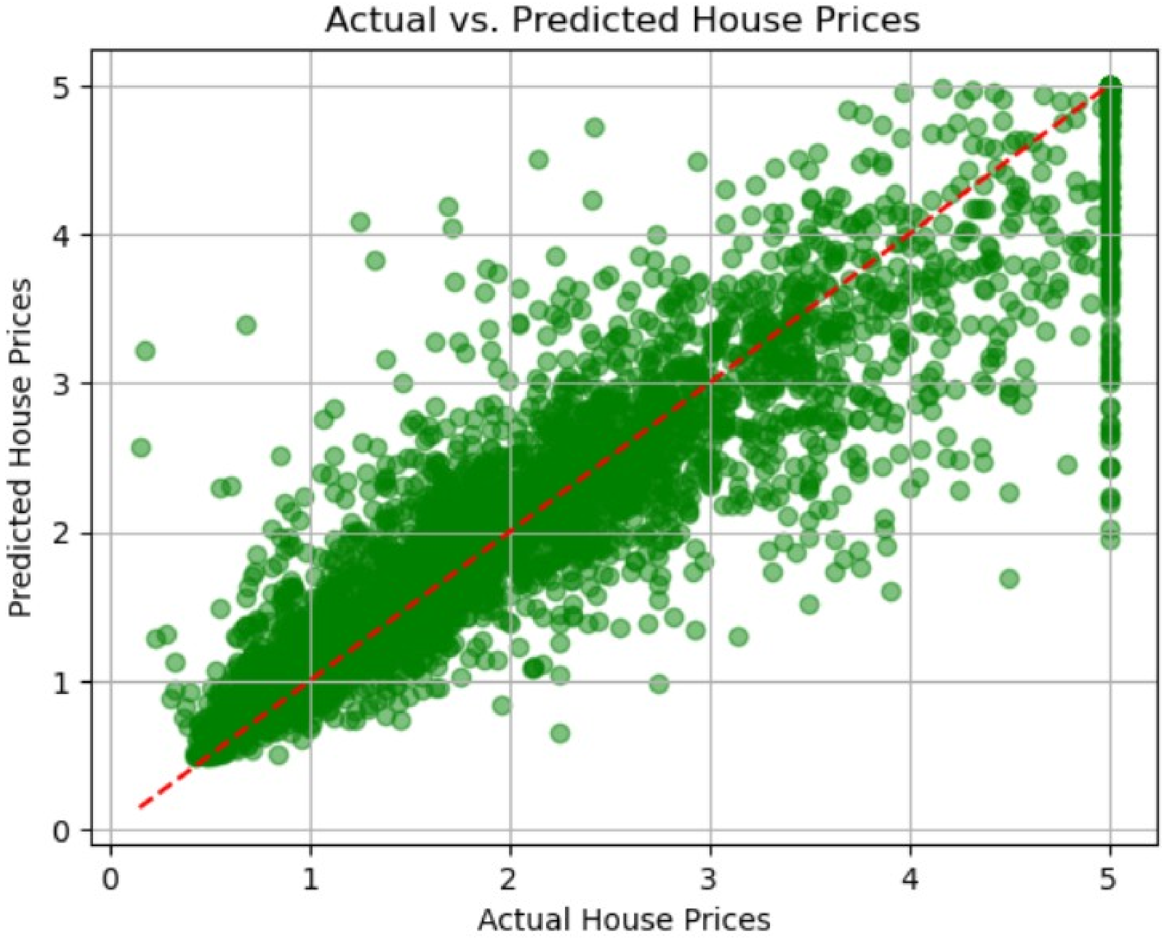
****

```
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Load the dataset

housing = fetch_california_housing()

X = housing.data

y = housing.target


# Step 2: Split into training and testing data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Step 3: Standardize the data

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Step 4: Train the Random Forest Regressor

model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train_scaled, y_train)
```

```
# Step 5: Predict
y_pred = model.predict(X_test_scaled)

# Step 6: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", round(mse, 4))
print("R-squared Score:", round(r2, 4))

# Step 7: Plot actual vs predicted values
plt.scatter(y_test, y_pred, alpha=0.5, color='green')
plt.xlabel("Actual House Prices")
plt.ylabel("Predicted House Prices")
plt.title("Actual vs. Predicted House Prices")
plt.grid(True)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')  # Diagonal line
plt.show()
```

OUTPUT:

```
Mean Squared Error: 0.2552
R-squared Score: 0.8053
```



****

PROGRAM 7: Regression model for Fetch House Prizing using KneighborRegressor

REGNO: 24251115

DATE: 23-05-2025

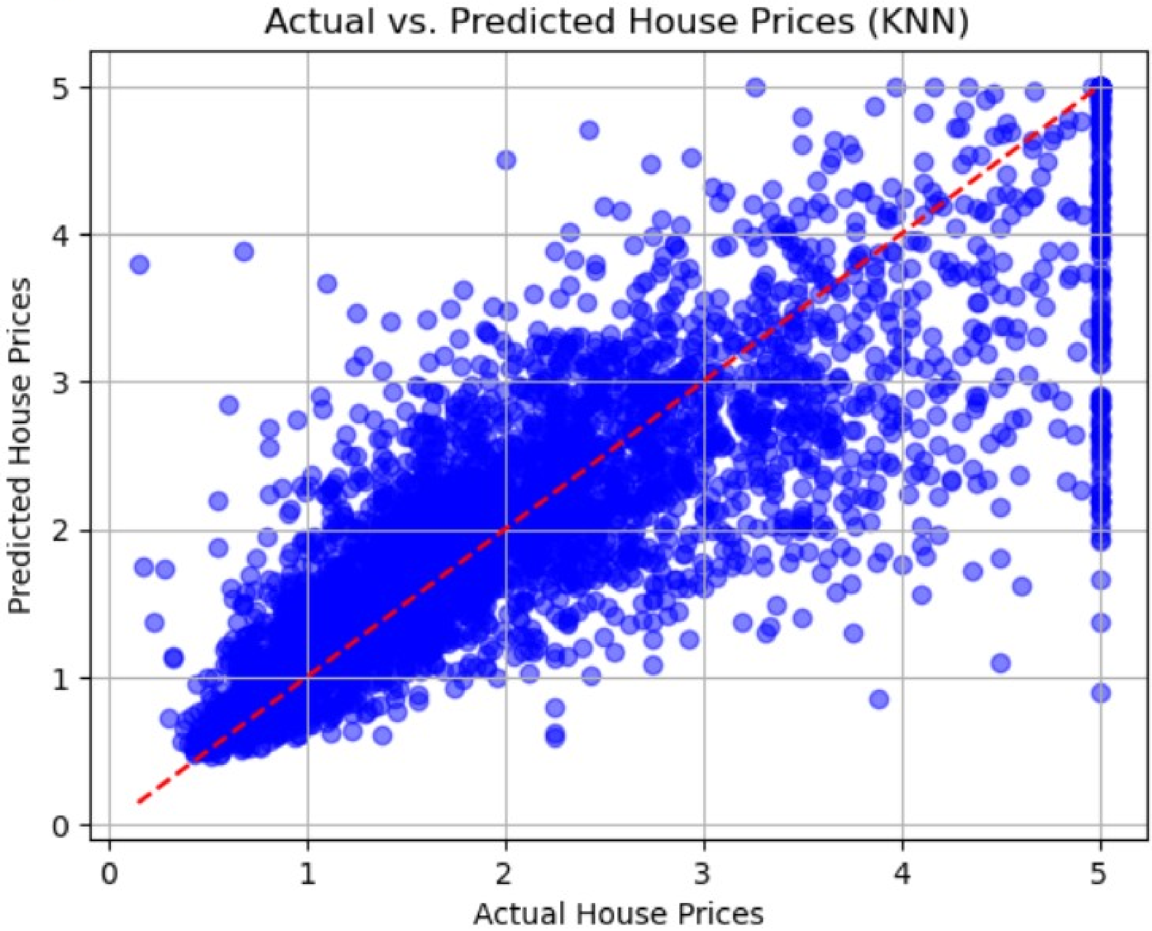
****

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load the dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target

# Step 2: Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train the K-Nearest Neighbors Regressor
model = KNeighborsRegressor(n_neighbors=5)
model.fit(X_train_scaled, y_train)

# Step 5: Predict
y_pred = model.predict(X_test_scaled)
```

```
# Step 6: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", round(mse, 4))
print("R-squared Score:", round(r2, 4))

# Step 7: Plot actual vs predicted values
plt.scatter(y_test, y_pred, alpha=0.5, color='blue')
plt.xlabel("Actual House Prices")
plt.ylabel("Predicted House Prices")
plt.title("Actual vs. Predicted House Prices (KNN)")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')  # Diagonal line
plt.grid(True)
plt.show()
```

OUTPUT:

```
Mean Squared Error: 0.4324
R-squared Score: 0.67
```



****

PROGRAM 8: Preprocessing missing value treatment

REGNO: 24251115

DATE: 23-05-2025

****

```
import matplotlib.pyplot 8. Preprocessing missing value treatment
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Sample data with missing values
data = {
    'Age': [25, 30, None, 35, 40],
    'Salary': [50000, 60000, 52000, None, 58000],
    'HousePrice': [200000, 250000, 220000, 240000, 230000]
}

df = pd.DataFrame(data)

# Separate features and target
X = df[['Age', 'Salary']]
y = df['HousePrice']

# Fill missing values with mean
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", round(mse, 2))
```

OUTPUT:

```
model
```

```
▾ LinearRegression
LinearRegression()
```

```
X = df[['Age', 'Salary']]
```

```
X
```

	Age	Salary
0	25.0	50000.0
1	30.0	60000.0
2	NaN	52000.0
3	35.0	NaN
4	40.0	58000.0

```
y = df['HousePrice']
```

```
y
```

```
0    200000
1    250000
2    220000
3    240000
4    230000
Name: HousePrice, dtype: int64
```

****

PROGRAM 9: Preprocessing, Scaling and Encoding

REGNO: 24251115

DATE: 23-05-2025

****

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = {
    'Age': [25, 30, None, 35, 40],
    'Salary': [50000, 60000, 52000, None, 58000],
    'City': ['New York', 'Paris', 'London', None, 'Paris'],
    'HousePrice': [200000, 250000, 220000, 240000, 230000]
}

df = pd.DataFrame(data)
X = df.drop('HousePrice', axis=1)
y = df['HousePrice']

# Missing value treatment
num_cols = ['Age', 'Salary']
cat_cols = ['City']

imputer_num = SimpleImputer(strategy='mean')
X[num_cols] = imputer_num.fit_transform(X[num_cols])
```

```
imputer_cat = SimpleImputer(strategy='most_frequent')
X[cat_cols] = imputer_cat.fit_transform(X[cat_cols])

# Encoding categorical variables
X = pd.get_dummies(X, drop_first=True)

# Scaling numeric features
scaler = StandardScaler()
X[num_cols] = scaler.fit_transform(X[num_cols])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
```

OUTPUT

```
df
```

	Age	Salary	City	HousePrice
0	25.0	50000.0	New York	200000
1	30.0	60000.0	Paris	250000
2	NaN	52000.0	London	220000
3	35.0	NaN	None	240000
4	40.0	58000.0	Paris	230000

```
model
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred
```

```
array([237174.72118959])
```

```
mse
```

```
164487776.5647242
```

```
r2
```

```
nan
```

****

PROGRAM 10: SVM model with preprocessing for the Gender classification dataset

REGNO: 24251115

DATE: 23-05-2025

****

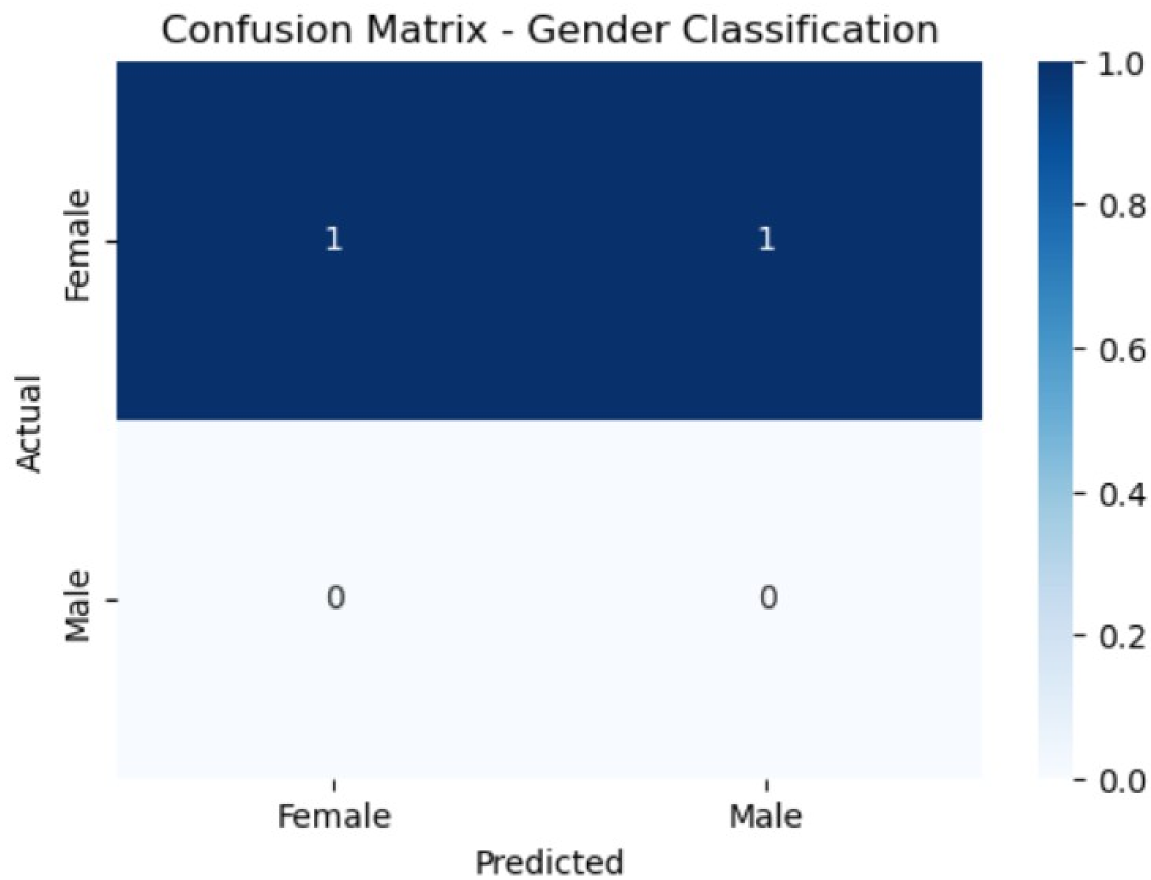
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Sample dataset
data = {
   'Height': [170, 160, 180, 175, 155, 165, 172, 158],
   'Weight': [65, 55, 80, 70, 50, 60, 68, 54],
   'Age': [25, 22, 28, 30, 20, 27, 24, 21],
   'Gender': ['Male', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Female']
}

df = pd.DataFrame(data)

# Features and target
X = df.drop('Gender', axis=1)
y = df['Gender']

# Encode target
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # Male=1, Female=0 or vice versa
```

```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.25,
random_state=42)

# Train SVM
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

# Predict
y_pred = svm_model.predict(X_test)

# Evaluate
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.2f}")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gender Classification')
plt.show()
```

OUTPUT:

```
Accuracy: 0.50
              precision    recall  f1-score   support

      Female       1.00      0.50      0.67         2
        Male       0.00      0.00      0.00         0

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2
```



****

PROGRAM 11: Gradient boosting classification with preprocessing for the Student Marks Dataset

REGNO: 24251115

DATE: : 23-05-2025

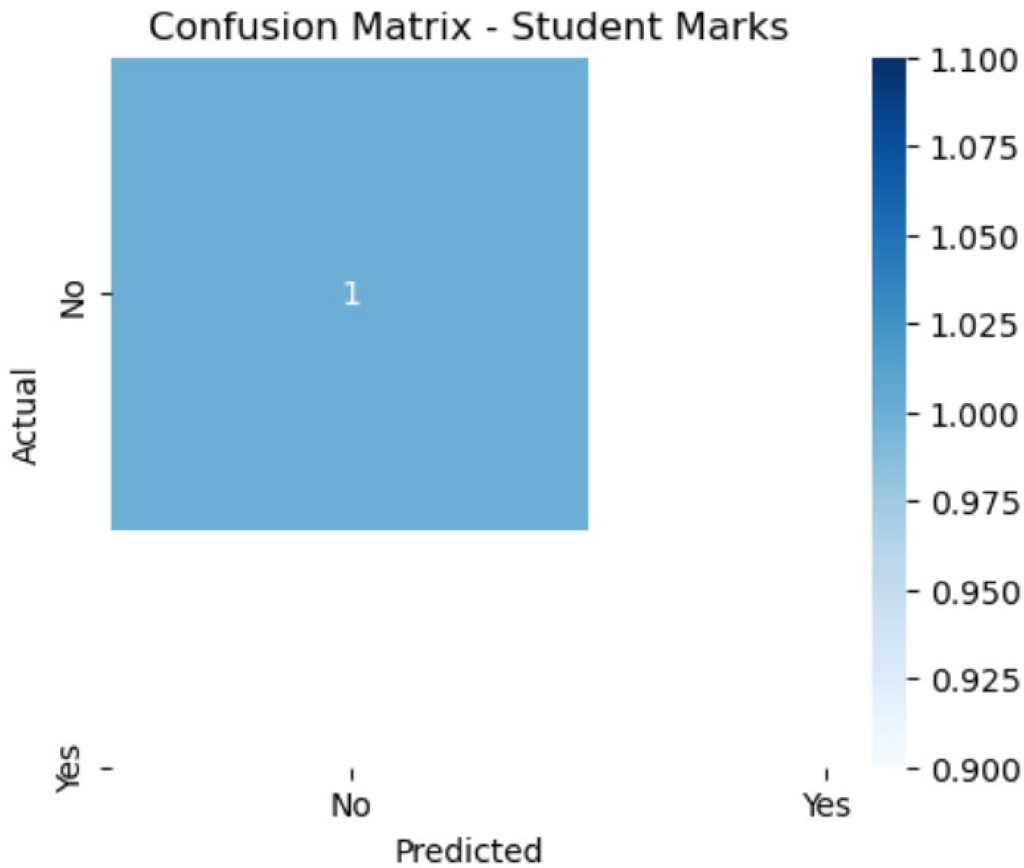
****

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Sample data
data = {
    'Math': [85, 70, 90, None, 60],
    'Science': [90, 65, None, 80, 70],
    'English': [78, 80, 85, 75, None],
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Female'],
    'Passed': ['Yes', 'No', 'Yes', 'No', 'No']
}

df = pd.DataFrame(data)

# 2. Separate features and target
X = df.drop('Passed', axis=1)
y = df['Passed']
```

```
# 3. Fill missing numeric values with mean
imputer = SimpleImputer(strategy='mean')
X[['Math', 'Science', 'English']] = imputer.fit_transform(X[['Math', 'Science', 'English']])

# 4. Encode Gender column
X = pd.get_dummies(X, columns=['Gender'], drop_first=True)

# 5. Scale numeric columns
scaler = StandardScaler()
X[['Math', 'Science', 'English']] = scaler.fit_transform(X[['Math', 'Science', 'English']])

# 6. Encode target variable
le = LabelEncoder()
y = le.fit_transform(y)  # Yes=1, No=0

# 7. Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 8. Train Gradient Boosting Classifier
model = GradientBoostingClassifier(random_state=42)
model.fit(X_train, y_train)

# 9. Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", round(accuracy, 2))

# 10. Confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
# 11. Plot confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Student Marks')
plt.show()
```

OUTPUT:




```
]: accuracy
```

```
]: 1.0
```

****

PROGRAM 12: . Visualization using matplotlib for iris dataset

- **Bar chart**
- **Pie chart**
- **Histogram**

REGNO: 24251115

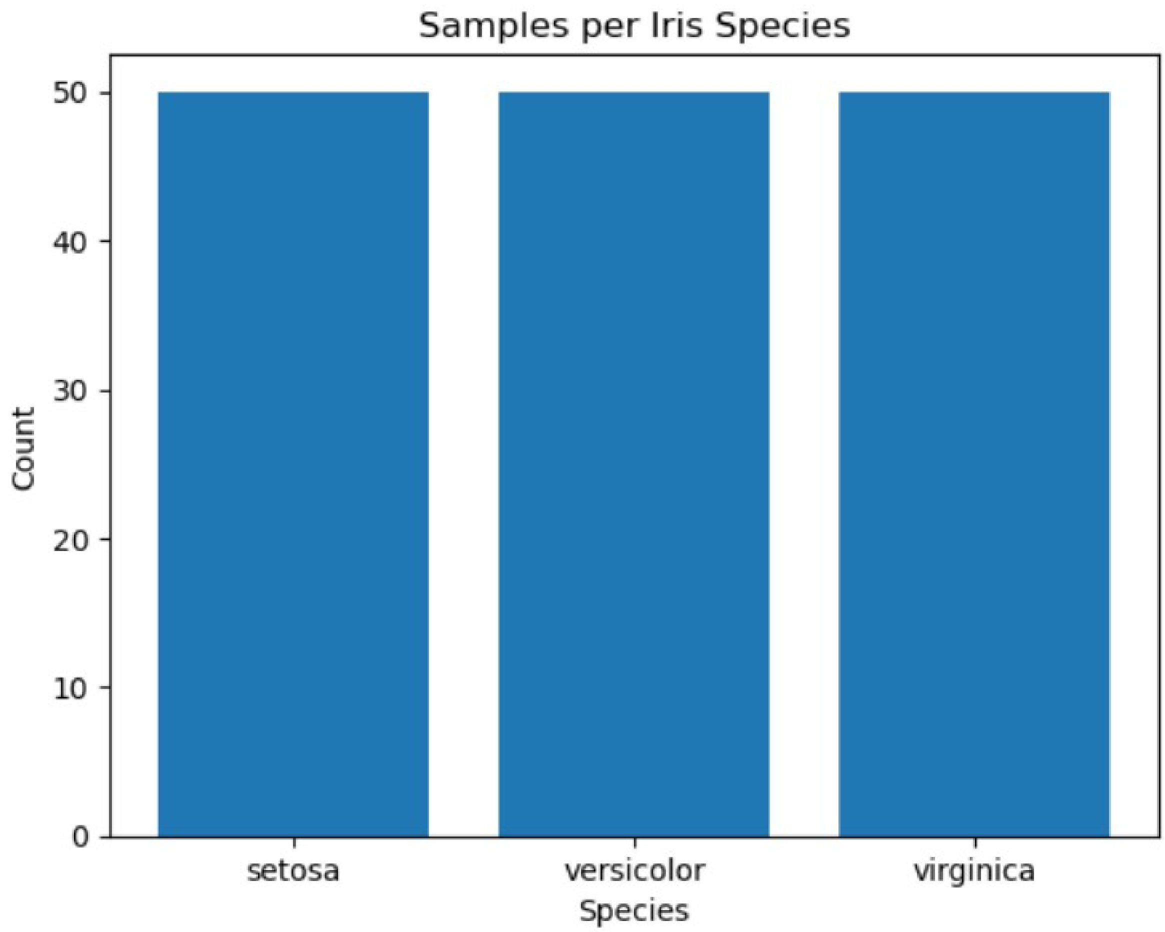
DATE: : 23-05-2025

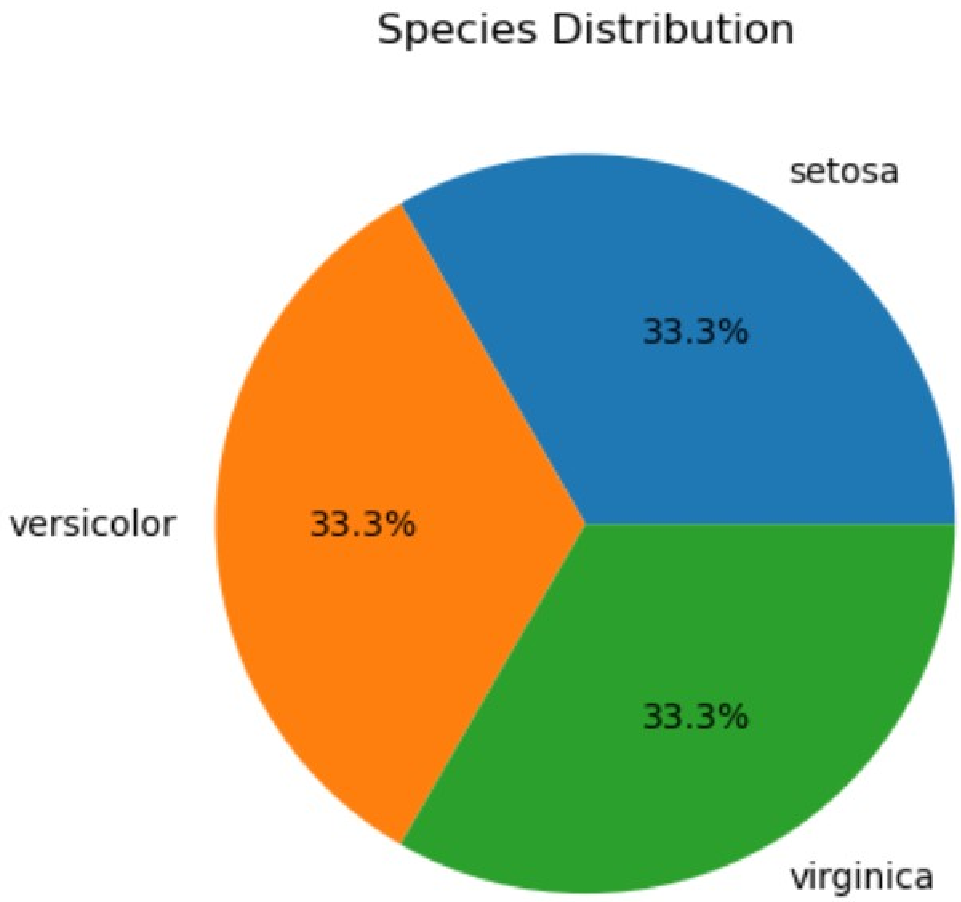
****

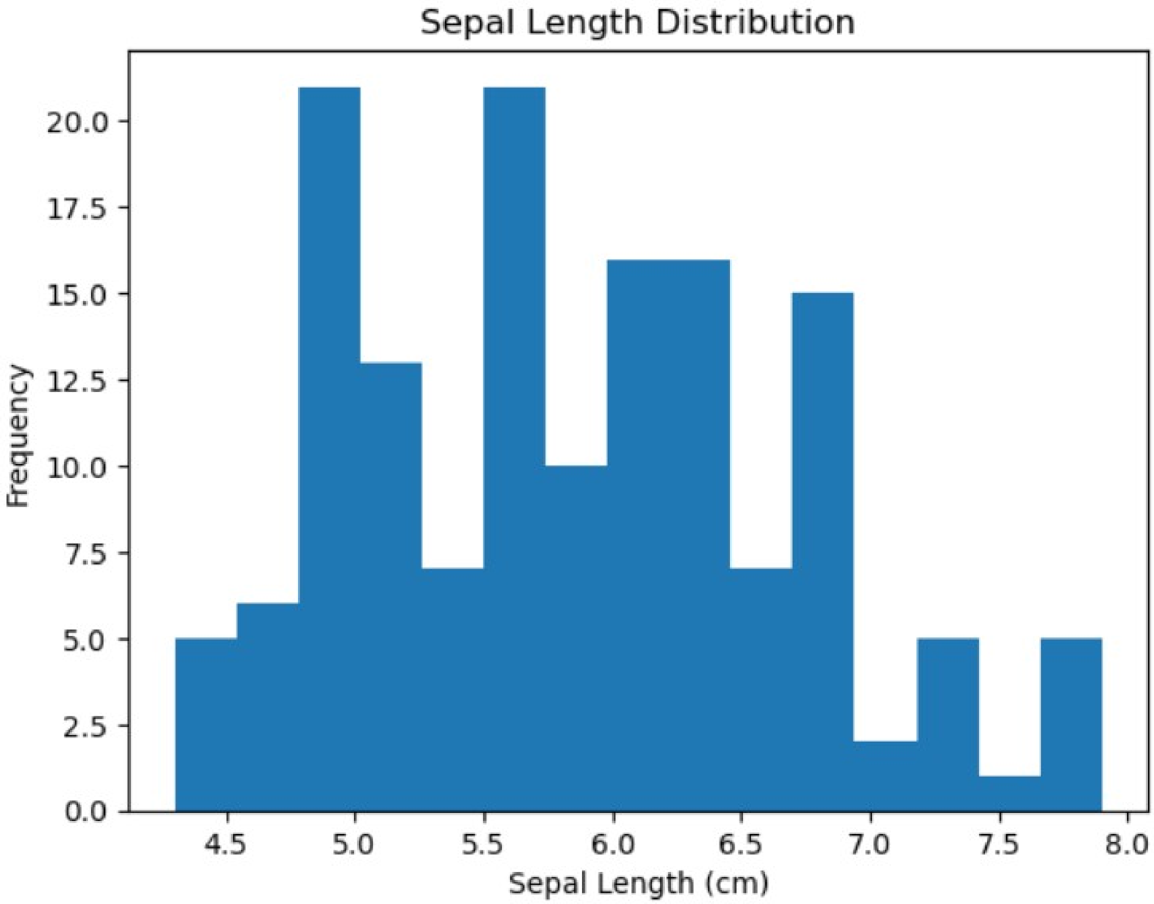
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
target = iris.target
target_names = iris.target_names

# Count samples per species
counts = [sum(target == i) for i in range(len(target_names))]

# 1. Bar chart
plt.bar(target_names, counts)
plt.title('Samples per Iris Species')
plt.xlabel('Species')
plt.ylabel('Count')
plt.show()

# 2. Pie chart
plt.pie(counts, labels=target_names, autopct='%1.1f%%')
plt.title('Species Distribution')
plt.show()
```

```
# 3. Histogram (Sepal Length)
plt.hist(iris.data[:, 0], bins=15)
plt.title('Sepal Length Distribution')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Frequency')
plt.show()
```

OUTPUT:




Species Distribution
setosa
33.3%
versicolor
33.3%
33.3%
virginica


Sepal Length Distribution
Frequency
Sepal Length (cm)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

****

PROGRAM 13: Pickle File for digits dataset with GausianNB model

REGNO: 24251115

DATE: 23-05-2025

****

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# 1. Load digits dataset
digits = load_digits()
X = digits.data
y = digits.target

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Train Gaussian Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# 4. Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# 5. Confusion matrix plot
cm = confusion_matrix(y_test, y_pred)
```

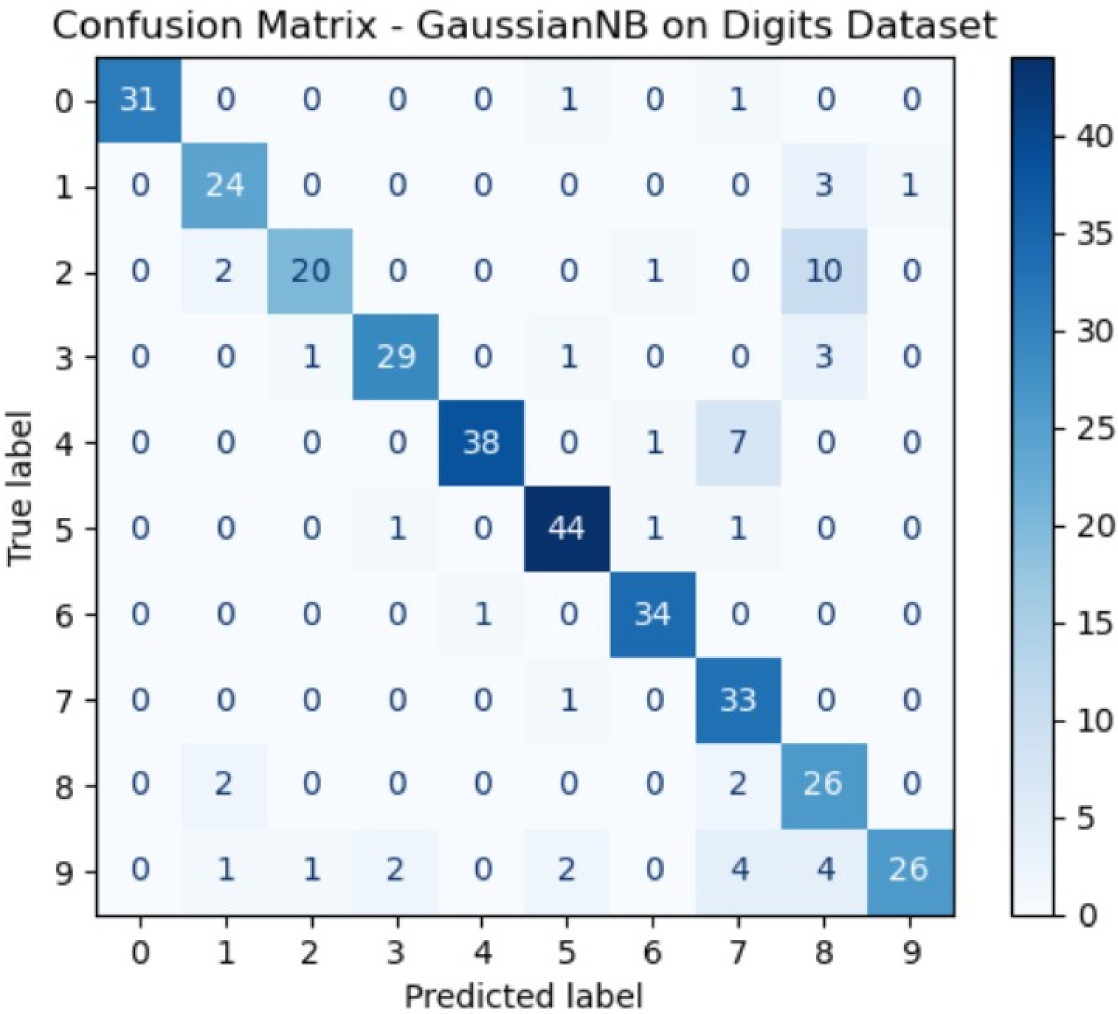
```python
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=digits.target_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix - GaussianNB on Digits Dataset")
plt.show()

# 6. Save model to pickle file
with open('gaussian_nb_digits.pkl', 'wb') as f:
    pickle.dump(model, f)
print("Model saved to 'gaussian_nb_digits.pkl'")

# 7. Load the model (example)
with open('gaussian_nb_digits.pkl', 'rb') as f:
    loaded_model = pickle.load(f)

# 8. Predict using loaded model (example)
sample_preds = loaded_model.predict(X_test[:5])
print("Sample predictions from loaded model:", sample_preds)
```

OUTPUT:

```
model
```

```
▾ GaussianNB
GaussianNB()
```

```
Accuracy: 0.85
```




```
Model saved to 'gaussian_nb_digits.pkl'
Sample predictions from loaded model: [6 9 3 7 2]
```

****
PROGRAM 14: **Advanced visualization and interpretation using seaborn for titanic dataset**

- **Stacked bar chart**
- **KDE plot**
- **Violin plot**
- **Heatmap**
- **Swarm plot**

REGNO: 24251115

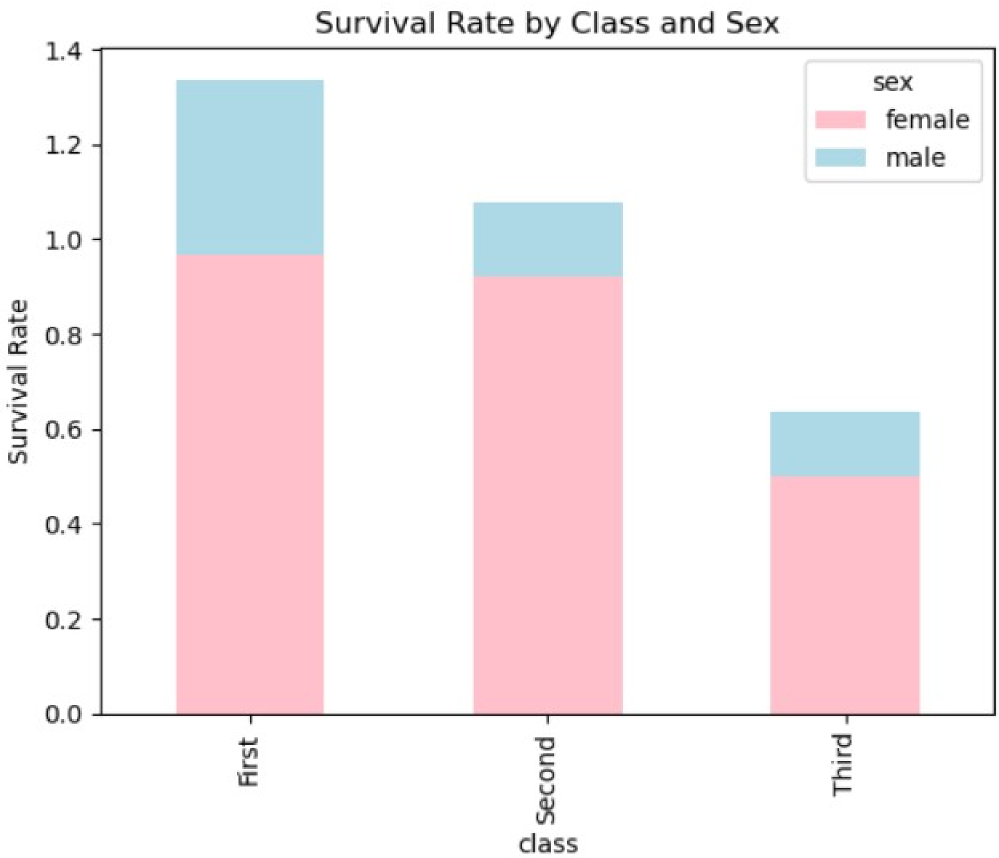
DATE: 23-05-2025

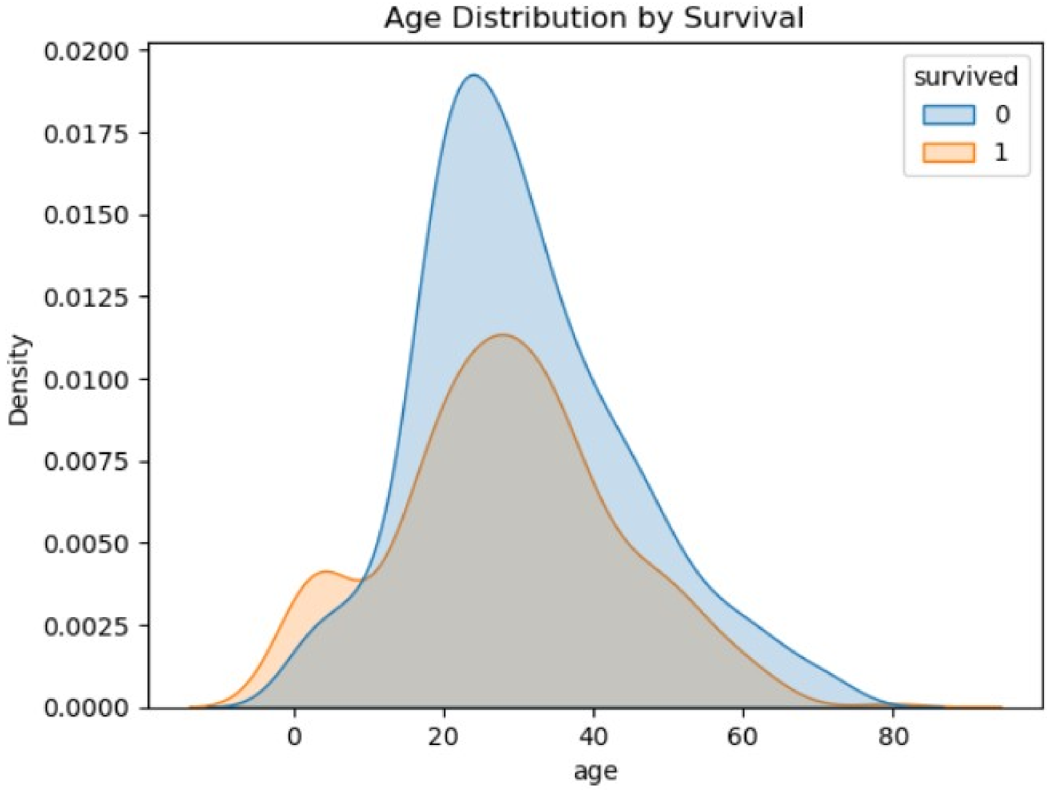
****

```
Import matplotlib.pyplot as plt

# Load dataset
titanic = sns.load_dataset('titanic')

# 1. Stacked Bar Chart (using pandas plot)
survival = titanic.groupby(['class', 'sex'])['survived'].mean().unstack()
survival.plot(kind='bar', stacked=True, color=['pink', 'lightblue'])
plt.title('Survival Rate by Class and Sex')
plt.ylabel('Survival Rate')
plt.show()

# 2. KDE Plot for Age by Survival
sns.kdeplot(data=titanic, x='age', hue='survived', fill=True)
plt.title('Age Distribution by Survival')
plt.show()

# 3. Violin Plot for Age by Class
sns.violinplot(x='class', y='age', data=titanic)
```

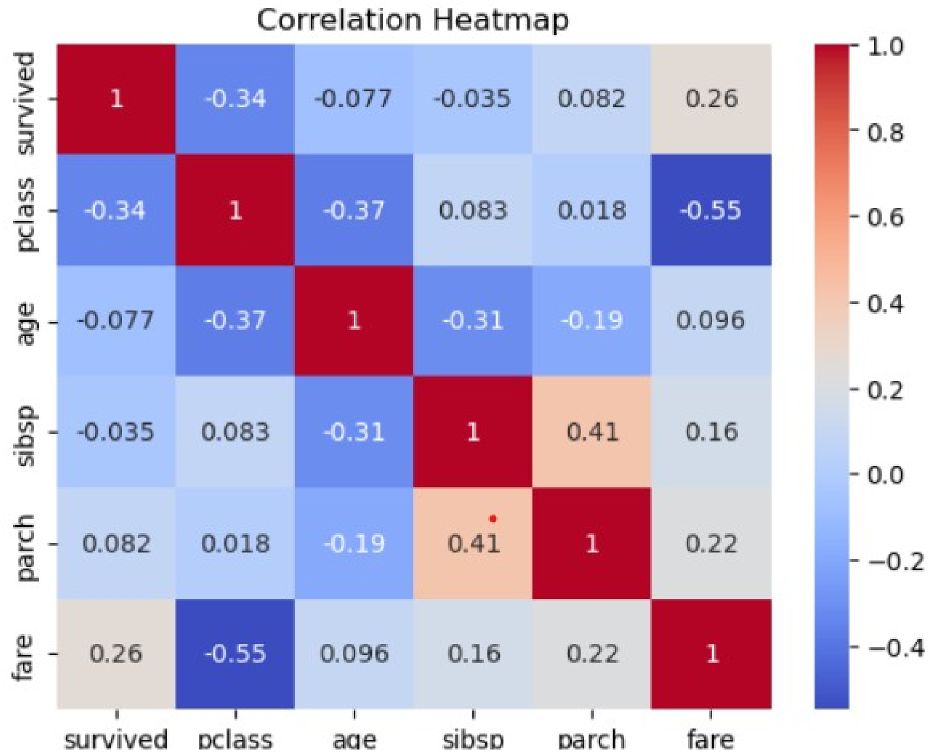
```
plt.title('Age Distribution by Class')
plt.show()

# 4. Heatmap for Correlation
numeric_data = titanic.select_dtypes(include=['float64', 'int64'])
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

# 5. Swarm Plot for Fare by Survival
sns.swarmplot(x='survived', y='fare', data=titanic)
plt.title('Fare Paid by Survival')
plt.show()
```

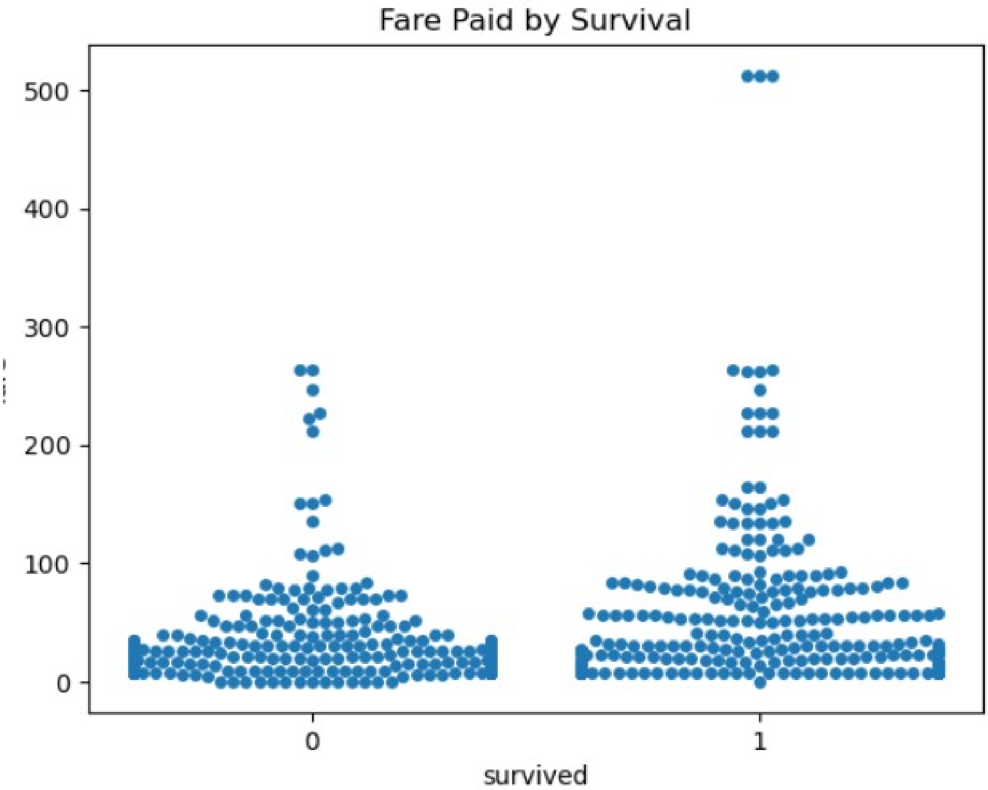
OUTPUT:




Age Distribution by Survival
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Density
survived
0
1
0
20
40
60
80
age


Age Distribution by Class
80
60
40
20
0
age
First
Second
Third
class


Correlation Heatmap
survived
pclass
age
sibsp
parch
fare
1
-0.34
-0.077
-0.035
0.082
0.26
-0.34
1
-0.37
0.083
0.018
-0.55
-0.077
-0.37
1
-0.31
-0.19
0.096
-0.035
0.083
-0.31
1
0.41
0.16
0.082
0.018
-0.19
0.41
1
0.22
0.26
-0.55
0.096
0.16
0.22
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Fare Paid by Survival
500
400
300
200
100
0
0
1
survived

****

PROGRAM 15: KMeans Cluster analysis

REGNO: 24251115

DATE: 23-05-2025

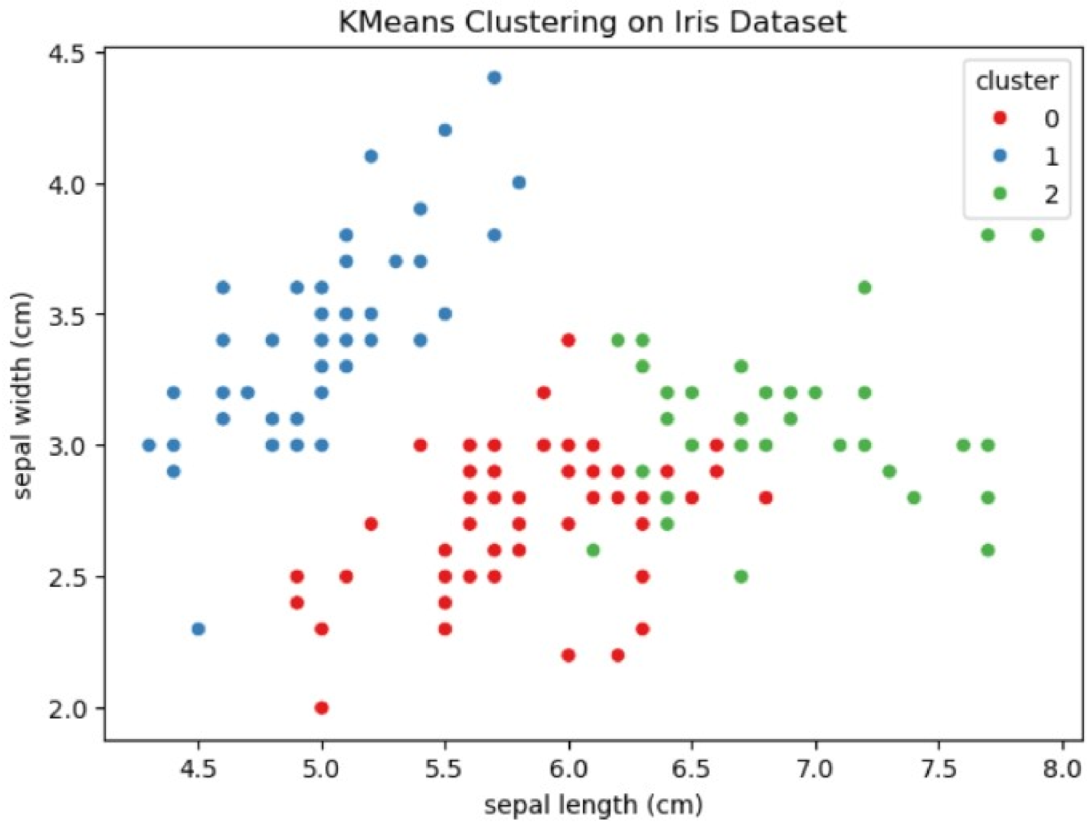
****

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=0)
df['cluster'] = kmeans.fit_predict(df)

# Add species for comparison (optional)
df['species'] = iris.target

# Plot clusters (using first two features for simplicity)
plt.figure(figsize=(7,5))
sns.scatterplot(x=df.iloc[:, 0], y=df.iloc[:, 1], hue=df['cluster'], palette='Set1')
plt.title('KMeans Clustering on Iris Dataset')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.show()
```

OUTPUT:

KMeans

```
KMeans(n_clusters=3, random_state=0)
```



****

PROGRAM 16: . Program to build a machine learning model to predict whether the person is diabetic or not using Bagging Classifier

REGNO:24251115

DATE: 23-05-2025
****

```
import pandas as pd

from sklearn.datasets import load_diabetes

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error,
r2_score

import seaborn as sns

import matplotlib.pyplot as plt


# Load dataset

data = load_diabetes()

X = pd.DataFrame(data.data, columns=data.feature_names)


# Convert target to binary classification (1 = diabetic, 0 = not)

y = (data.target > data.target.mean()).astype(int)


# Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Scale features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

```
# Train Bagging Classifier
model = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=50,
random_state=42)
model.fit(X_train_scaled, y_train)

# Predict
y_pred = model.predict(X_test_scaled)

# --- Metrics ---
accuracy = accuracy_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

# --- Print Metrics ---
print("Accuracy:", round(accuracy, 4))
print("Mean Squared Error:", round(mse, 4))
print("R-squared Score:", round(r2, 4))

# --- Plot Confusion Matrix ---
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Diabetic', 'Diabetic'],
            yticklabels=['Not Diabetic', 'Diabetic'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

OUTPUT:

```
model
```

```
▸ BaggingClassifier
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
accuracy
```

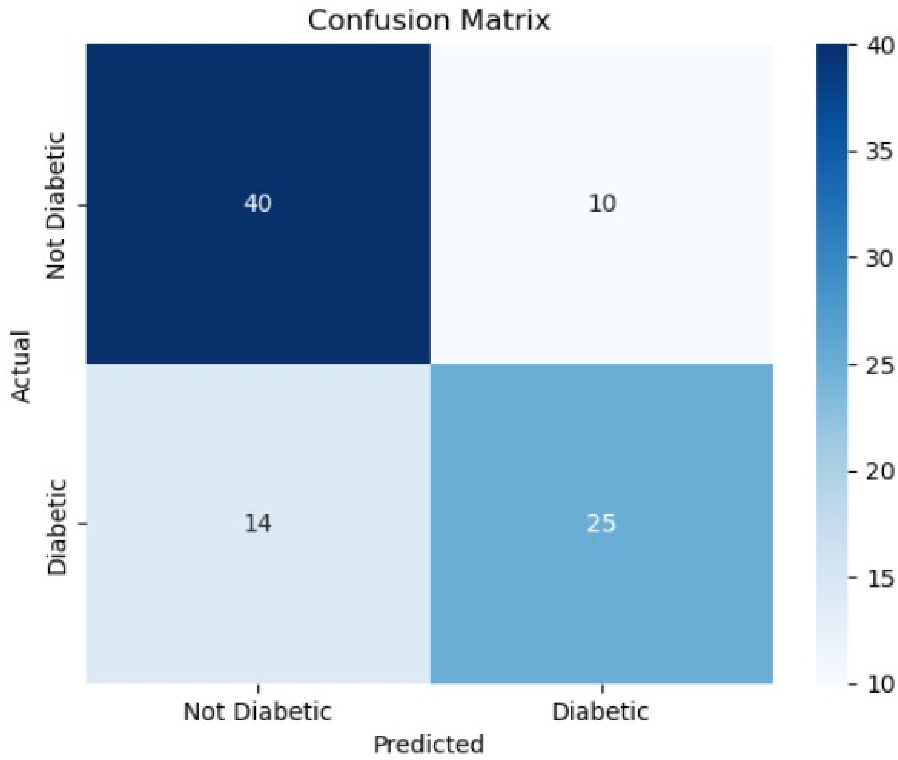
```
0.7303370786516854
```

```
mse
```

```
0.2696629213483146
```

```
r2
```

```
-0.0953846153846154
```



****

PROGRAM 17: Program to demonstrate Principal Component Analysis(PCA)

REGNO:24251115

DATE: 23-05-2025

****

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data                    # Features (sepal & petal measurements)
y = iris.target                  # Target (species)
target_names = iris.target_names

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA to reduce dimensions (from 4 to 2)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Create a DataFrame for PCA results
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['Species'] = y

# Step 5: Plot the 2 principal components
plt.figure(figsize=(7, 5))
for i, label in enumerate(target_names):
```

```
    plt.scatter(pca_df[pca_df['Species'] == i]['PC1'],
                pca_df[pca_df['Species'] == i]['PC2'],
                label=label)

plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()Program to demonstrate Principal Component Analysis(PCA)
```

OUTPUT:

```
scaler
```

```
StandardScaler
StandardScaler()
```

****

PROGRAM 18: .Program to demonstrate K Nearest Neighbours Classification(KNN).

REGNO:24251102

DATE: 23-05-2025

****

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error
import seaborn as sns

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=3)
```

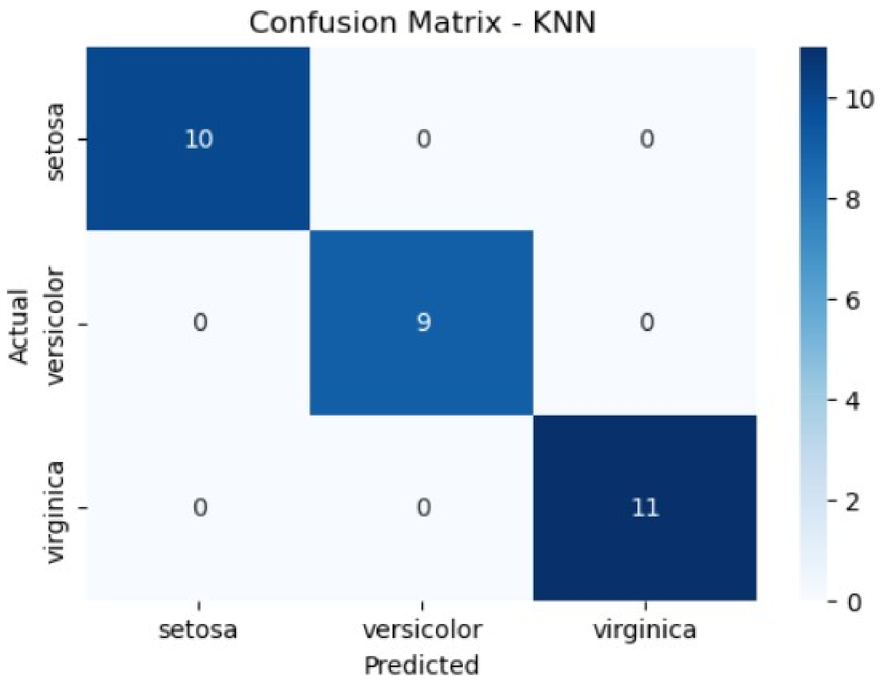
```
knn.fit(X_train_scaled, y_train)

# Predict
y_pred = knn.predict(X_test_scaled)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Mean Squared Error: {mse:.4f}")

# Confusion matrix plot
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=target_names, yticklabels=target_names)
plt.title("Confusion Matrix - KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

OUTPUT:

```
knn
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
Accuracy: 1.0000
Mean Squared Error: 0.0000
```



****
PROGRAM 19: Program to demonstrate Neural Networks and deep Learning

REGNO: 24251115

DATE: 23-05-2025
****

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load data
iris = load_iris()
X = iris.data
y = iris.target

# 2. Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# 4. Build and train the MLP neural network
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)
```

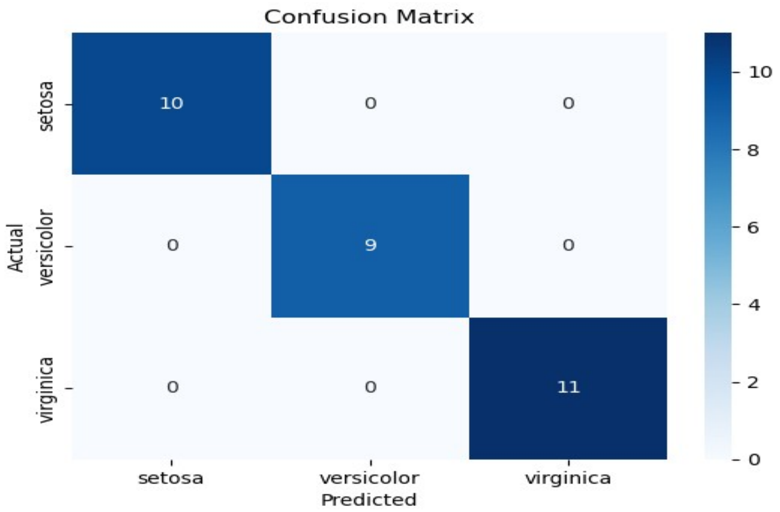
```python
# 5. Predict
y_pred = mlp.predict(X_test)

# 6. Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Test accuracy: {acc:.4f}")

# 7. Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:

```
scaler

StandardScaler
StandardScaler()
```

```
accuracy

1.0

y_pred

array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2
       0, 2, 2, 2, 2, 2, 0, 0])
```

	actual	predicted
0	1	1
1	0	0
2	2	2
3	1	1
4	1	1
5	0	0
6	1	1
7	2	2
8	1	1
9	1	1
10	2	2
11	0	0
12	0	0
13	0	0
14	0	0
15	1	1
16	2	2
17	1	1
18	1	1
19	2	2
20	0	0
21	2	2
22	0	0
23	2	2
24	2	2
25	2	2
26	2	2
27	2	2
28	0	0
29	0	0



****

PROGRAM 20: . Visualization using matplotlib for iris dataset

- **Scatter plot**
- **Box plot**
- **Line chart**

REGNO: 24251115

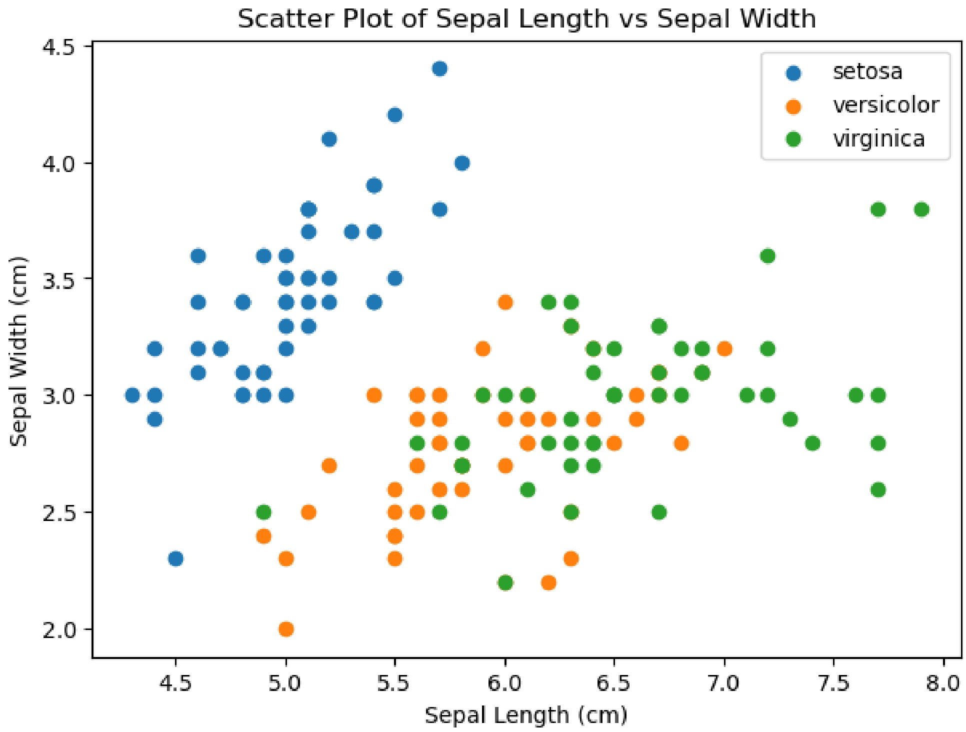
DATE: 23-05-2025

****

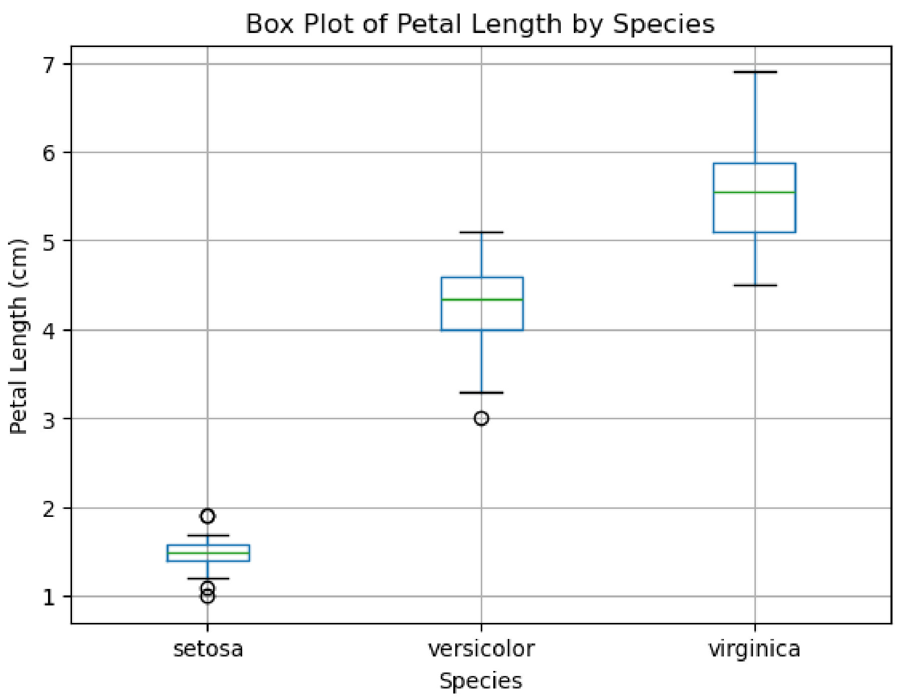
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd

# Load Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

# 1. Scatter plot: Sepal Length vs Sepal Width colored by species
plt.figure(figsize=(7,5))
for species in iris.target_names:
    subset = df[df['species'] == species]
    plt.scatter(subset['sepal length (cm)'], subset['sepal width (cm)'], label=species)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Scatter Plot of Sepal Length vs Sepal Width')
plt.legend()
plt.show()

# 2. Box plot: Distribution of Petal Length by Species
plt.figure(figsize=(7,5))
```

```
df.boxplot(column='petal length (cm)', by='species')
plt.title('Box Plot of Petal Length by Species')
plt.suptitle('')  # Removes default subtitle
plt.xlabel('Species')
plt.ylabel('Petal Length (cm)')
plt.show()

# 3. Line chart: Mean sepal length for each species
mean_sepal_length = df.groupby('species')['sepal length (cm)'].mean()
plt.figure(figsize=(7,5))
plt.plot(mean_sepal_length.index, mean_sepal_length.values, marker='o')
plt.title('Mean Sepal Length by Species')
plt.xlabel('Species')
plt.ylabel('Mean Sepal Length (cm)')
plt.grid(True)
plt.show()
```

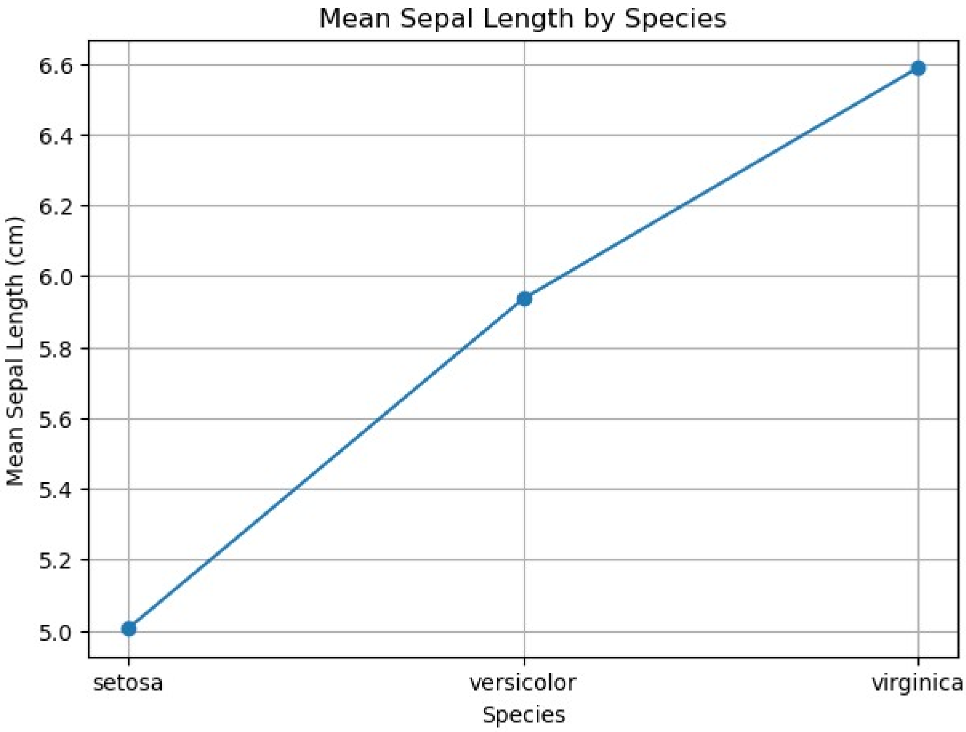
OUTPUT:



Scatter Plot of Sepal Length vs Sepal Width
setosa
versicolor
virginica
Sepal Width (cm)
Sepal Length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



Box Plot of Petal Length by Species
Petal Length (cm)
7
6
5
4
3
2
1
setosa
versicolor
virginica
Species


Mean Sepal Length by Species
Mean Sepal Length (cm)
6.6
6.4
6.2
6.0
5.8
5.6
5.4
5.2
5.0
setosa
versicolor
virginica
Species