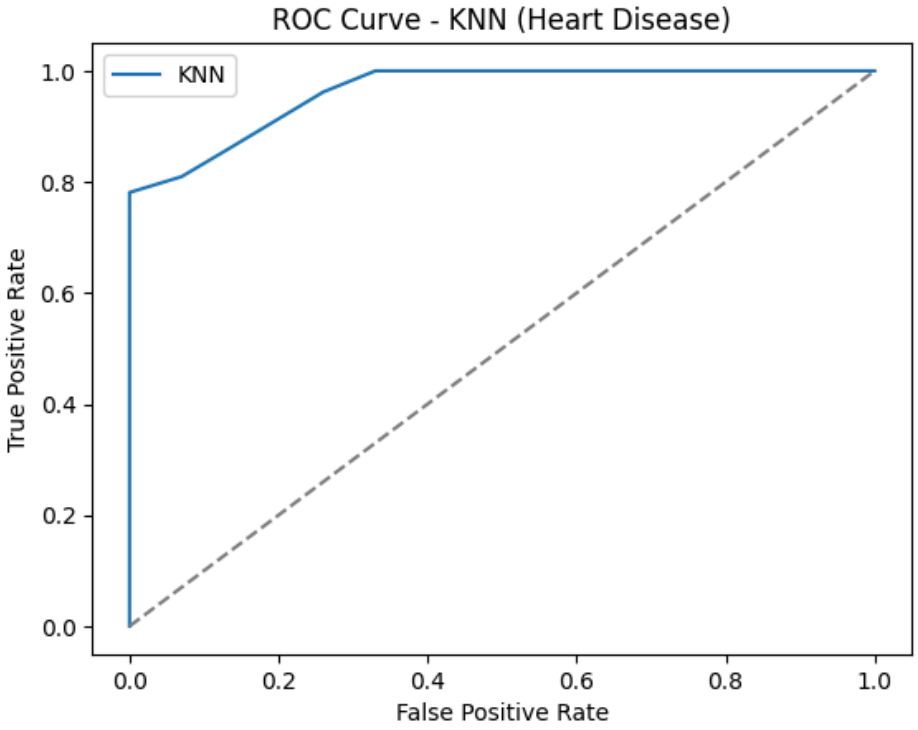
```
# Predicting Employee Attrition using Logistic Regression

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# 1. Load dataset
df = pd.read_csv("/content/Employee Attrition.csv")

# 2. Handle missing values
df.fillna(df.median(numeric_only=True), inplace=True)
df.fillna(df.mode().iloc[0], inplace=True)

# 3. Encode categorical variables
le = LabelEncoder()
for col in df.select_dtypes(include=['object']).columns:
    df[col] = le.fit_transform(df[col])

# 4. Split features and target
X = df.drop("Attrition", axis=1)
y = df["Attrition"]

# 5. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 7. Train logistic regression
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)

# 8. Predictions and evaluation
y_pred = model.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, digits=3))
```

```
Confusion Matrix:
 [[186  61]
 [ 11  36]]

Classification Report:
              precision    recall  f1-score   support

           0      0.944     0.753     0.838       247
           1      0.371     0.766     0.500        47

    accuracy                          0.755       294
   macro avg      0.658     0.759     0.669       294
weighted avg      0.853     0.755     0.784       294
```

```
# Predicting Heart Disease using KNN

import pandas as pd
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# 1. Load dataset
df = pd.read_csv("/content/heart.csv")  # replace with actual file name if different

# 2. Handle missing values (mean for numeric, mode for categorical just in case)
for col in df.columns:
    if df[col].dtype in ['int64', 'float64']:
        df[col].fillna(df[col].mean(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)

# 3. Encode categorical variables
le = LabelEncoder()
for col in df.select_dtypes(include=['object']).columns:
    df[col] = le.fit_transform(df[col])

# 4. Feature / Target split
X = df.drop("target", axis=1)   # assume target column is named "target"
y = df["target"]

# 5. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 7. Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# 8. Predictions
y_pred = knn.predict(X_test)
y_prob = knn.predict_proba(X_test)[:,1]

# 9. Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

# 10. ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label="KNN")
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - KNN (Heart Disease)")
plt.legend()
plt.show()
```

```
/tmp/ipython-input-1753650042.py:16: FutureWarning: A value is trying to be set on a copy of a DataFram
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df[col].fillna(df[col].mean(), inplace=True)
Accuracy: 0.8634146341463415
ROC-AUC: 0.9625714285714285
```



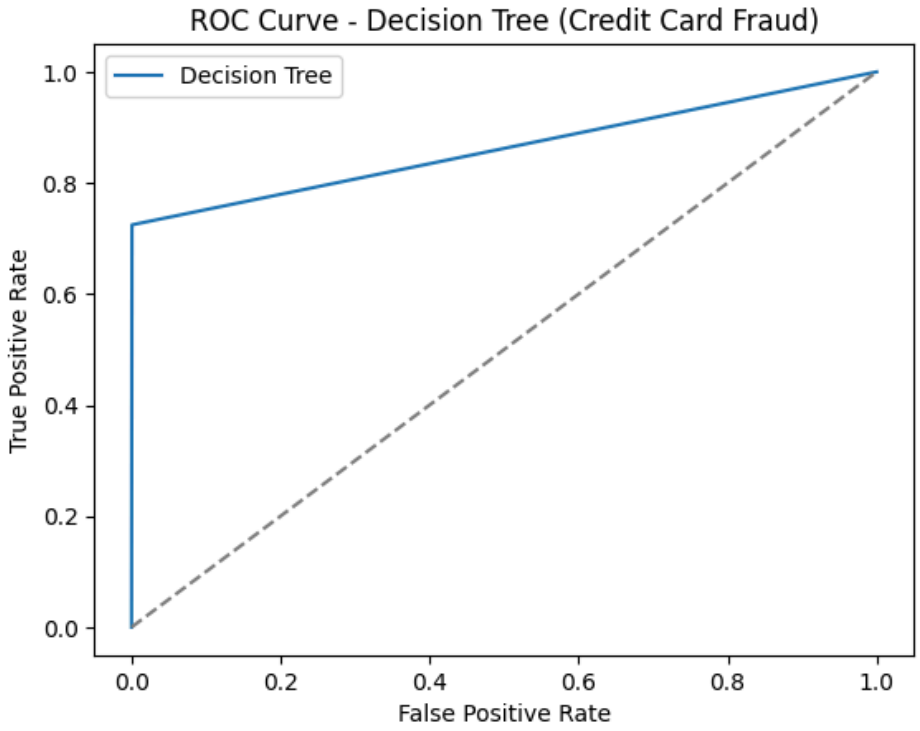

```
# Predicting Hospital Readmission using Logistic Regression

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# 1. Load dataset
df = pd.read_csv("/content/hospital_readmissions.csv")  # replace with actual dataset filename

# 2. Handle missing values (mode for categorical, mean for numeric just in case)
for col in df.columns:
    if df[col].dtype in ['int64', 'float64']:
        df[col].fillna(df[col].mean(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)

# 3. Encode categorical variables
le = LabelEncoder()
for col in df.select_dtypes(include=['object']).columns:
    df[col] = le.fit_transform(df[col])

# 4. Feature / Target split
X = df.drop("Readmitted", axis=1)   # assume target column is "Readmitted"
y = df["Readmitted"]
```

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

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 7. Train Logistic Regression
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)

# 8. Predictions & Evaluation
y_pred = model.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report (Precision, Recall, F1-Score):\n", classification_report(y_test, y_pred
```

```
# Classifying Credit Card Fraud using Decision Trees

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# 1. Load dataset
df = pd.read_csv("/content/creditcard.csv")  # replace with actual dataset filename

# 2. Handle missing values (fill numeric with mean, categorical with mode if any)
for col in df.columns:
    if df[col].dtype in ['int64', 'float64']:
        df[col].fillna(df[col].mean(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)

# 3. Feature / Target split
X = df.drop("Class", axis=1)   # assume fraud label column is named "Class" (0 = legit, 1 = fraud)
y = df["Class"]

# 4. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 6. Train Decision Tree model
dt = DecisionTreeClassifier(random_state=42, class_weight="balanced")
dt.fit(X_train, y_train)

# 7. Predictions & Evaluation
y_pred = dt.predict(X_test)
y_prob = dt.predict_proba(X_test)[:,1]

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

```
# 8. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label="Decision Tree")
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree (Credit Card Fraud)")
plt.legend()
plt.show()
```

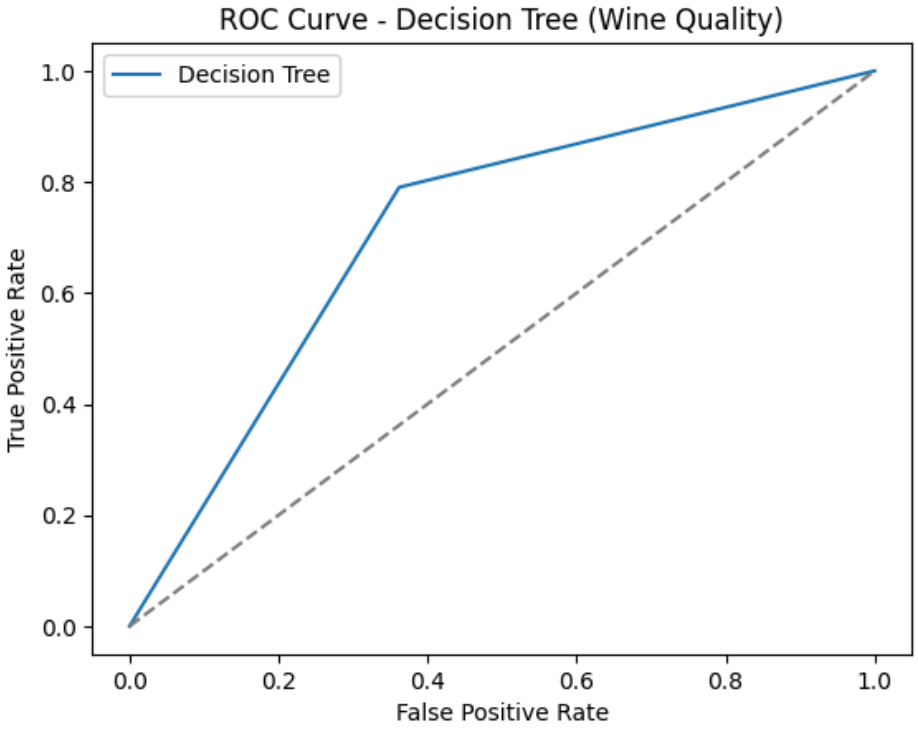
```
/tmp/ipython-input-3443208046.py:16: FutureWarning: A value is trying to be set on a copy of a DataFram
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df[col].fillna(df[col].mean(), inplace=True)
Confusion Matrix:
 [[56830    34]
 [   27    71]]

ROC-AUC Score: 0.8619459390396564
```




```
# Classifying Wine Quality using Decision Trees

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# 1. Load dataset
df = pd.read_csv("/content/WineQT.csv")  # replace with actual dataset filename

# 2. Handle missing values (numeric -> mean, categorical -> mode)
```

```
for col in df.columns:
    if df[col].dtype in ['int64', 'float64']:
        df[col].fillna(df[col].mean(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)

# 3. Encode categorical variables if any
le = LabelEncoder()
for col in df.select_dtypes(include=['object']).columns:
    df[col] = le.fit_transform(df[col])

# 4. Define target (binarize quality: good=1 if >=6, else 0)
df["QualityLabel"] = (df["quality"] >= 6).astype(int)
X = df.drop(["quality", "QualityLabel"], axis=1)
y = df["QualityLabel"]

# 5. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 7. Train Decision Tree model
dt = DecisionTreeClassifier(random_state=42, class_weight="balanced")
dt.fit(X_train, y_train)

# 8. Predictions & Evaluation
y_pred = dt.predict(X_test)
y_prob = dt.predict_proba(X_test)[:,1]

print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

# 9. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label="Decision Tree")
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree (Wine Quality)")
plt.legend()
plt.show()
```

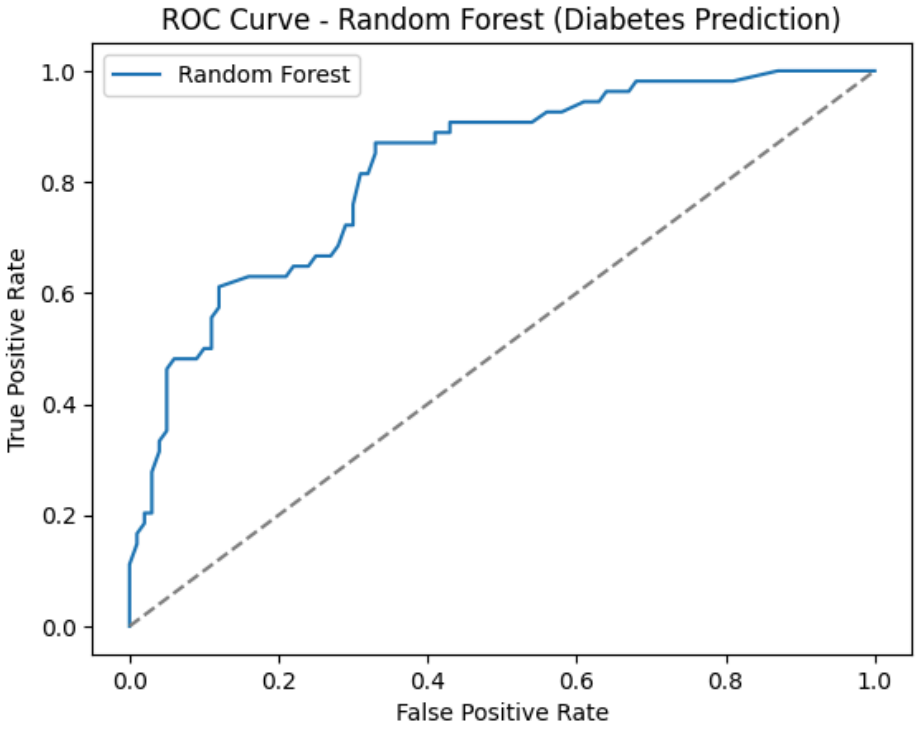
```
/tmp/ipython-input-2451597122.py:16: FutureWarning: A value is trying to be set on a copy of a DataFram
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df[col].fillna(df[col].mean(), inplace=True)
Accuracy: 0.7205240174672489
ROC-AUC: 0.7142089093701995
```

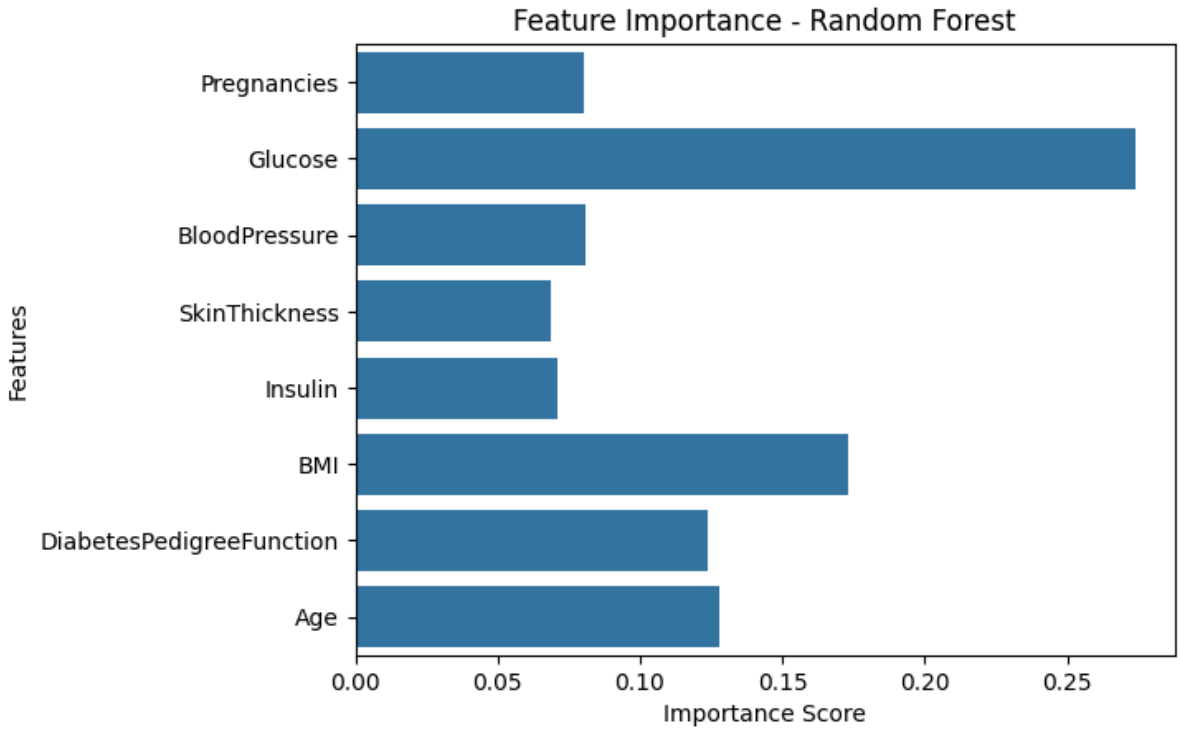



```
# Predicting Diabetes using Random Forest

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Load dataset
df = pd.read_csv("diabetes.csv")  # PIMA Indians Diabetes dataset

# 2. Handle missing values (fill with mean)
df.fillna(df.mean(numeric_only=True), inplace=True)

# 3. Features and target
X = df.drop("Outcome", axis=1)  # 'Outcome' is the target column
y = df["Outcome"]

# 4. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
```

```
)

# 6. Train Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight="balanced")
rf.fit(X_train, y_train)

# 7. Predictions & Evaluation
y_pred = rf.predict(X_test)
y_prob = rf.predict_proba(X_test)[:, 1]

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

# 8. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label="Random Forest")
plt.plot([0, 1], [0, 1], "--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Random Forest (Diabetes Prediction)")
plt.legend()
plt.show()

# 9. Feature Importance
importances = rf.feature_importances_
feat_names = X.columns

sns.barplot(x=importances, y=feat_names)
plt.title("Feature Importance - Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.show()
```

```
Confusion Matrix:
 [[88 12]
 [21 33]]

Accuracy: 0.7857142857142857
ROC-AUC: 0.8233333333333335
```






```
# Classifying Iris Flowers Using SVM

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report

# 1. Load Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target

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

# 3. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

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

# 5. Predictions
y_pred = svm.predict(X_test)

# 6. Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision (macro):", precision_score(y_test, y_pred, average="macro"))
print("Recall (macro):", recall_score(y_test, y_pred, average="macro"))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_nam
```

```
Accuracy: 1.0
Precision (macro): 1.0
Recall (macro): 1.0

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

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

```
# Classifying Breast Cancer Using KNN

import pandas as pd
from sklearn.datasets import load_breast_cancer
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
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# 2. Handle missing values (fill with mean if any)
```

```
X.fillna(X.mean(numeric_only=True), inplace=True)

# 3. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 5. Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# 6. Predictions
y_pred = knn.predict(X_test)

# 7. Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# 8. Confusion Matrix Heatmap
plt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
            xticklabels=data.target_names, yticklabels=data.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - KNN (Breast Cancer)")
plt.show()
```

```
Accuracy: 0.9649122807017544
Confusion Matrix:
 [[39  3]
 [ 1 71]]
```