**KNN :**

# Importing necessary libraries for visualization

import numpy as np

import matplotlib.pyplot as plt

from sklearn import neighbors, datasets

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

from sklearn.metrics import accuracy\_score

# Load the iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2] # Using only the first two features for 2D visualization

y = iris.target

# 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)

# Create an instance of KNeighborsClassifier with weights='distance' for weighted neighbors

knn = neighbors.KNeighborsClassifier(n\_neighbors=5, weights='distance')

# Fit the model on the training data

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

# Predict on the test data

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

# Calculate accuracy

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

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

# Create color maps

cmap\_light = plt.cm.Paired

cmap\_bold = ['darkorange', 'c', 'darkblue']

# Generate a mesh grid for plotting decision boundaries

h = .02 # Step size in the mesh

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

# Predict the class for each point in the mesh grid

Z = knn.predict(np.c\_[xx.ravel(), yy.ravel()])

# Reshape the result back into a mesh grid shape

Z = Z.reshape(xx.shape)

# Plot the decision boundaries

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, cmap=cmap\_light)

# Plot the training points

scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolor='k', s=20)

# Choose a test point (you can change the coordinates)

test\_point = np.array([[5.0, 3.5]])

# Predict the class of the test point

test\_prediction = knn.predict(test\_point)

# Find the nearest neighbors of the test point

distances, indices = knn.kneighbors(test\_point)

# Plot the test point

plt.scatter(test\_point[0][0], test\_point[0][1], c='red', s=100, edgecolor='k', label='Test Point')

# Plot the nearest neighbors

nearest\_neighbors = X[indices.flatten()]

plt.scatter(nearest\_neighbors[:, 0], nearest\_neighbors[:, 1], c='yellow', s=100, edgecolor='k', marker='X', label='Nearest Neighbors')

# Annotate the nearest neighbors

for i in range(len(indices.flatten())):

plt.annotate(f"{y[indices.flatten()[i]]}", (nearest\_neighbors[i, 0], nearest\_neighbors[i, 1]),

textcoords="offset points", xytext=(0, 10), ha='center', fontsize=9)

# Add labels, title, and legend

plt.xlabel(iris.feature\_names[0])

plt.ylabel(iris.feature\_names[1])

plt.title(f"K-NN Classification with Weighted Neighbors\nAccuracy: {accuracy:.2f}")

plt.legend()

# Show the plot

plt.show()

**APRIORITY:**  
import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

def main():

    transactions = [

        ["milk", "bread", "eggs"],

        ["bread", "butter"],

        ["milk", "bread", "butter", "eggs"],

        ["bread", "eggs"],

        ["milk", "eggs"],

    ]

    te = TransactionEncoder()

    te\_ary = te.fit(transactions).transform(transactions)

    df = pd.DataFrame(te\_ary, columns=te.columns\_)

    frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True)

    rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

    print("Frequent Itemsets:\n", frequent\_itemsets)

    print("\nAssociation Rules:\n", rules)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**PRE PROCESSING**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data["species"] = iris.target

data = data.fillna(data.mean())

# Remove duplicates

data = data.drop\_duplicates()

# Encode categorical data

data = pd.get\_dummies(data, columns=["species"])

# Feature scaling

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

# Save preprocessed data

data\_preprocessed = pd.DataFrame(data\_scaled, columns=data.columns)

print(data\_preprocessed.head())

**RFE:**

# Import necessary libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

data = load\_iris()

x = data.data

y = data.target

feature\_names = data.feature\_names

model = LogisticRegression(max\_iter=500)

selector = RFE(estimator=model, n\_features\_to\_select=2)

selector.fit(x, y)

print("Feature ranking:", selector.ranking\_)

plt.figure(figsize=(13, 5))

plt.barh(feature\_names, selector.ranking\_)

plt.xlabel("Feature Ranking")

plt.ylabel("Features")

plt.title("Feature Importance Ranking using RFE")

plt.show()

**MLR**

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import load\_diabetes

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

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

def main():

    # Load the diabetes dataset

    diabetes = load\_diabetes()

    X = diabetes.data  # Multiple features

    y = diabetes.target

    # 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=50

    )

    # Initialize and train the model

    model = LinearRegression()

    model.fit(X\_train, y\_train)

    # Make predictions on the testing set

    y\_pred = model.predict(X\_test)

    # Output the results

    print("Multiple Linear Regression Results")

    print("-------------------------------")

    print(f"Coefficient: {model.coef\_}")

    print(f"Intercept: {model.intercept\_:.2f}")

    print(f"Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred):.2f}")

    print(f"Coefficient of Determination (R²): {r2\_score(y\_test, y\_pred):.2f}")

    # Plotting Actual vs Predicted

    plt.scatter(y\_test, y\_pred)

    plt.plot([y.min(), y.max()], [y.min(), y.max()], color="red", linewidth=2)

    plt.xlabel("Actual Target")

    plt.ylabel("Predicted Target")

    plt.title("Actual vs Predicted Targets")

    plt.legend(["Perfect Prediction", "Predictions"])

    plt.show()

main()

**PCA:**

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

iris = load\_iris()

X = iris.data

y = iris.target

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

plt.figure(figsize=(14, 6))

# Before PCA

plt.subplot(1, 2, 1)

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=y, cmap="viridis")

plt.title("Before PCA")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

# After PCA

plt.subplot(1, 2, 2)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap="viridis")

plt.title("After PCA")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.show()