|  |  |
| --- | --- |
| # Import necessary libraries  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  # Load the dataset from local storage  data = pd.read\_csv("california\_housing.csv")  # Display the shape of the dataset  print("Dataset shape:", data.shape)  # Display column names  print("Column Names:", data.columns.tolist())  # Display basic information about the dataset  print("\nDataset Information:\n")  print(data.info())  # Print the first 10 rows of the dataset  print("\nFirst 10 Rows of the Dataset:\n")  print(data.head(10))  # Generate Histograms for all numerical features  plt.figure(figsize=(12, 8))  data.hist(bins=30, figsize=(12, 8), edgecolor='black', color='skyblue')  plt.suptitle("Histograms of All Numerical Features", fontsize=16)  plt.show()  # Generate Box Plots to identify outliers  plt.figure(figsize=(12, 8))  for i, col in enumerate(data.columns):  plt.subplot(3, 3, i+1)  sns.boxplot(y=data[col], color='lightcoral')  plt.title(col)  plt.ylabel('')  plt.tight\_layout()  plt.suptitle("Box Plots of All Numerical Features", fontsize=16)  plt.show() | import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  # Load the dataset  df = pd.read\_csv("california\_housing.csv")  # Display basic information about the dataset  print("Dataset Overview:")  print(df.info())  print("\nFirst 5 Rows:")  print(df.head())  # Compute the correlation matrix  correlation\_matrix = df.corr()  # Visualize the correlation matrix using a heatmap  Machine Learning Lab-Program2- Heat Map and Pair Plot  Dept. of CSE (AI), MIT Mysore 4  plt.figure(figsize=(10, 6))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)  plt.title("Correlation Matrix Heatmap")  plt.show()  # Create a pair plot to visualize relationships between features  sns.pairplot(df, diag\_kind='kde')  plt.show() |

|  |  |
| --- | --- |
| LAB 3: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality  of the Iris dataset from 4 features to 2.  import pandas as pd  from sklearn.decomposition import PCA  import matplotlib.pyplot as plt  # Read the Iris dataset  data = pd.read\_csv("Iris.csv")  # Select the 4 feature columns  X = data[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]  # Apply PCA to reduce to 2 components  pca = PCA(n\_components=2)  X\_pca = pca.fit\_transform(X)  # Print the shape and the first few rows of the reduced data  print("Reduced data shape:", X\_pca.shape)  print(X\_pca[:5])  # Plot the 2D PCA result  plt.scatter(X\_pca[:, 0], X\_pca[:, 1])  plt.xlabel("Principal Component 1")  plt.ylabel("Principal Component 2")  plt.title("PCA of Iris Dataset")  plt.show() | LAB 4: For a given set of training data examples stored in a .csv file, implement and demonstrate the Find-S  algorithm to output a description of the set of all hypotheses consistent with the training examples.  import csv  # Load CSV directly into data  with open('enjoysport.csv', 'r') as file:  data = list(csv.reader(file))  # Initialize hypothesis with the first positive example  hypothesis = []  for example in data[1:]: # Skip header row  if example[-1].lower() == 'yes':  hypothesis = example[:-1]  break  # Generalize hypothesis based on other positive examples  for example in data[1:]:  if example[-1].lower() == 'yes':  for i in range(len(hypothesis)):  if hypothesis[i] != example[i]:  hypothesis[i] = '?'  print("Final Hypothesis:", hypothesis) |

|  |  |
| --- | --- |
| import numpy as np  from collections import Counter  # Generate 100 random values in [0, 1]  np.random.seed(42)  data = np.random.rand(100)  # Split into training (first 50) and test (last 50)  train\_x = data[:50]  test\_x = data[50:]  # Label training data  train\_y = ['Class1' if x <= 0.5 else 'Class2' for x in train\_x]  def knn\_predict(train\_x, train\_y, test\_val, k):  distances = np.abs(train\_x - test\_val)  sorted\_idx = np.argsort(distances)  nearest\_labels = np.array(train\_y)[sorted\_idx][:k]  return Counter(nearest\_labels).most\_common(1)[0][0]  k\_values = [1, 2, 3, 4, 5, 20, 30]  for k in k\_values:  print(f"--- For k = {k} ---")  predictions = []  for i, x in enumerate(test\_x, start=51):  pred = knn\_predict(train\_x, train\_y, x, k)  predictions.append(pred)  print(f"Point x{i}: {x:.4f} -> Predicted: {pred}")  counts = Counter(predictions)  print(f"Summary for k = {k}: Class1 = {counts['Class1']}, Class2 = {counts['Class2']}\n") | import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  def kernel(point, xmat, k):  m,n = np.shape(xmat)  weights = np.mat(np.eye((m)))  for j in range(m):  diff = point - X[j]  weights[j,j] = np.exp(diff\*diff.T/(-2.0\*k\*\*2))  return weights  def localWeight(point, xmat, ymat, k):  wei = kernel(point,xmat,k)  W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T))  return W  def localWeightRegression(xmat, ymat, k):  m,n = np.shape(xmat)  ypred = np.zeros(m)  for i in range(m):  ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k)  return ypred  # load data points  data = pd.read\_csv('tips.csv')  bill = np.array(data.total\_bill)  tip = np.array(data.tip)  #preparing the data  mbill = np.mat(bill)  mtip = np.mat(tip)  m= np.shape(mbill)[1]  one = np.mat(np.ones(m))  X = np.hstack((one.T,mbill.T))  #set k here  ypred = localWeightRegression(X,mtip,0.5)  SortIndex = X[:,1].argsort(0)  xsort = X[SortIndex][:,0]  fig = plt.figure()  ax = fig.add\_subplot(1,1,1)  ax.scatter(bill,tip, color='yellow')  ax.plot(xsort[:,1],ypred[SortIndex], color = 'black', linewidth=2)  plt.xlabel('Total bill')  plt.ylabel('Tip')  plt.show(); |

|  |  |
| --- | --- |
| # ------------------- LINEAR REGRESSION: Boston Housing Dataset -------------------  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score  # Load Boston Housing data from local CSV  boston\_df = pd.read\_csv("BostonHousing.csv")  # Feature and target  X = boston\_df[['RM']] # Average number of rooms,  # uses double square brackets to return the result as a DataFrame, not a Series.  y = boston\_df['MEDV'] # Median house price  # Split the data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Train the model  model = LinearRegression()  model.fit(X\_train, y\_train)  # Predict  y\_pred = model.predict(X\_test)  # Plot Linear Regression output  plt.scatter(X\_test, y\_test, color='blue', label='Actual Prices')  plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted Prices')  plt.xlabel("Average number of rooms (RM)")  plt.ylabel("Median house value (MEDV)")  plt.title("Linear Regression - Boston Housing Dataset")  plt.legend()  plt.grid(True)  plt.show()  # Evaluate  print("\nLinear Regression Evaluation:")  print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))  print("R2 Score:", r2\_score(y\_test, y\_pred))  # ------------------ POLYNOMIAL REGRESSION: Auto MPG Dataset ----------------------  from sklearn.preprocessing import PolynomialFeatures  from sklearn.pipeline import make\_pipeline  # Load Auto MPG dataset from local CSV  auto\_df = pd.read\_csv("auto\_mpg.csv")  auto\_df = auto\_df.dropna() # Drop missing values  # Feature and target  X = auto\_df[['horsepower']]  y = auto\_df['mpg']  # Polynomial regression pipeline  poly\_model = make\_pipeline(PolynomialFeatures(degree=2), LinearRegression())  poly\_model.fit(X, y)  # Create input range with correct feature name to avoid warning  x\_range = pd.DataFrame(np.linspace(X.min().values[0], X.max().values[0], 100),  columns=["horsepower"])  # Predict  y\_poly\_pred = poly\_model.predict(x\_range)  # Plot Polynomial Regression output  plt.scatter(X, y, color='gray', alpha=0.5, label='Actual MPG')  plt.plot(x\_range, y\_poly\_pred, color='green', linewidth=2, label='Polynomial Fit')  plt.xlabel("Horsepower")  plt.ylabel("Miles Per Gallon (MPG)")  plt.title("Polynomial Regression - Auto MPG Dataset")  plt.legend()  plt.grid(True)  plt.show() | # Import necessary libraries  from sklearn.datasets import load\_breast\_cancer # To load dataset  from sklearn.tree import DecisionTreeClassifier # Decision Tree classifier  from sklearn.model\_selection import train\_test\_split # To split data  from sklearn.metrics import accuracy\_score # To check accuracy  # Step 1: Load the Breast Cancer dataset  data = load\_breast\_cancer()  # Step 2: Display the features and target names  print("Feature Names:\n", data.feature\_names)  print("\nTarget Names:\n", data.target\_names)  # Step 3: Separate features (X) and labels (y)  X = data.data # Features  y = data.target # Labels (0 = malignant, 1 = benign)  # Step 4: Split the data into training and testing sets (80% train, 20% test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,  random\_state=42)  # Step 5: Create and train the Decision Tree classifier  model = DecisionTreeClassifier(random\_state=42)  model.fit(X\_train, y\_train)  # Step 6: Test the model on the test data  y\_pred = model.predict(X\_test)  # Step 7: Print the accuracy  accuracy = accuracy\_score(y\_test, y\_pred)  print("\nAccuracy on test data:", round(accuracy \* 100, 2), "%")  # Step 8: Classify a new (unseen) sample from the test set  new\_sample = X\_test[0].reshape(1, -1) # Truly unseen sample  predicted\_class = model.predict(new\_sample)  # Step 9: Display the result  print("\nNew Sample Prediction (from test set):")  print("Predicted Class:", predicted\_class[0])  print("Meaning:", "Malignant" if predicted\_class[0] == 0 else "Benign") |

|  |  |
| --- | --- |
| # Import necessary libraries  import matplotlib.pyplot as plt  from sklearn.datasets import fetch\_olivetti\_faces  from sklearn.model\_selection import train\_test\_split  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import accuracy\_score  # Load the Olivetti Faces dataset  faces = fetch\_olivetti\_faces(shuffle=True, random\_state=42)  X = faces.data  y = faces.target  # Display 10 sample face images with their labels  fig, axes = plt.subplots(2, 5, figsize=(10, 5))  for i, ax in enumerate(axes.flat):  ax.imshow(X[i].reshape(64, 64), cmap='gray')  ax.set\_title(f"Person {y[i]}")  ax.axis('off')  plt.suptitle("Sample Images from Olivetti Faces Dataset")  plt.tight\_layout()  plt.show()  # Split the data into training and testing sets (80%-20%)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,  stratify=y, random\_state=42)  # Train the Naive Bayes classifier  model = GaussianNB()  model.fit(X\_train, y\_train)  # Make predictions and calculate accuracy  y\_pred = model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print("\nNaive Bayes Classifier Accuracy: {:.2f}%".format(accuracy \* 100))  # Visualize predictions  fig, axes = plt.subplots(2, 5, figsize=(10, 5))  for i, ax in enumerate(axes.flat):  ax.imshow(X\_test[i].reshape(64, 64), cmap='gray')  ax.set\_title(f"Pred: {y\_pred[i]}\nActual: {y\_test[i]}")  ax.axis('off')  plt.suptitle("Predictions on Test Images")  plt.tight\_layout()  plt.show() | import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.datasets import load\_breast\_cancer  from sklearn.cluster import KMeans  from sklearn.decomposition import PCA  from sklearn.preprocessing import StandardScaler  # Step 1: Load the Wisconsin Breast Cancer dataset  data = load\_breast\_cancer()  X = data.data # Feature matrix (569 samples × 30 features)  # Step 2: Standardize the features (mean=0, std=1)  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # Step 3: Apply K-Means clustering with 2 clusters  kmeans = KMeans(n\_clusters=2, random\_state=42, n\_init=10)  clusters = kmeans.fit\_predict(X\_scaled)  # clusters is an array of 0s and 1s assigning each sample to one of two clusters  # Step 4: Reduce data to 2 principal components for visualization  pca = PCA(n\_components=2)  X\_pca = pca.fit\_transform(X\_scaled)  # X\_pca now has shape (569, 2), ready for 2D plotting  # Step 5: Transform cluster centroids into the same 2D PCA space  centroids\_original = kmeans.cluster\_centers\_  centroids\_pca = pca.transform(centroids\_original)  # Step 6: Plot the clustered data in 2D  plt.figure(figsize=(8, 6))  # Define user-friendly names for the two clusters  cluster\_names = {0: "Cluster 1", 1: "Cluster 2"}  colors = ["yellow", "blue"]  for cluster\_id, color in zip(cluster\_names, colors):  mask = (clusters == cluster\_id)  plt.scatter(  X\_pca[mask, 0], # x = PC1  X\_pca[mask, 1], # y = PC2  c=color,  alpha=0.6,  edgecolor='k',  label=cluster\_names[cluster\_id]  )  # Step 7: Plot the centroids as black 'X' markers  plt.scatter(centroids\_pca[:, 0],  centroids\_pca[:, 1],  marker='X',  s=200,  c='black',  label="Centroids"  )  # Step 8: Add titles and labels  plt.title("K-Means Clustering on Breast Cancer Data")  plt.xlabel("Principal Component 1")  plt.ylabel("Principal Component 2")  plt.legend(loc="upper right")  # Step 9: Display the plot  plt.tight\_layout()  plt.show() |