**P1**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**housing\_df = pd.read\_csv('California housing dataset.csv')**

**numerical\_features = housing\_df.select\_dtypes(include=[np.number]).columns**

**plt.figure(figsize=(15, 10))**

**for i, feature in enumerate(numerical\_features):**

**plt.subplot(3, 3, i + 1)**

**sns.histplot(housing\_df[feature], kde=True, bins=30, color='blue')**

**plt.title(f'Distribution of {feature}')**

**plt.tight\_layout()**

**plt.show()**

**plt.figure(figsize=(15, 10))**

**for i, feature in enumerate(numerical\_features):**

**plt.subplot(3, 3, i + 1)**

**sns.boxplot(x=housing\_df[feature], color='orange')**

**plt.title(f'Box Plot of {feature}')**

**plt.tight\_layout()**

**plt.show()**

**print("Outliers Detection:")**

**outliers\_summary = {}**

**for feature in numerical\_features:**

**Q1 = housing\_df[feature].quantile(0.25)**

**Q3 = housing\_df[feature].quantile(0.75)**

**IQR = Q3 - Q1**

**lower\_bound = Q1 - 1.5 \* IQR**

**upper\_bound = Q3 + 1.5 \* IQR**

**outliers = housing\_df[(housing\_df[feature] < lower\_bound) | (housing\_df[feature] > upper\_bound)]**

**outliers\_summary[feature] = len(outliers)**

**print(f"{feature}: {len(outliers)} outliers")**

**print("\nDataset Summary:")**

**print(housing\_df.describe())**

**P2**

**import pandas as pd**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**data = pd.read\_csv("California housing dataset.csv")**

**numeric\_data = data.select\_dtypes(include=[float, int])**

**correlation\_matrix = numeric\_data.corr()**

**plt.figure(figsize=(10, 8))**

**sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)**

**plt.title('Correlation Matrix of California Housing Features')**

**plt.show()**

**sns.pairplot(data, diag\_kind='kde', plot\_kws={'alpha': 0.5})**

**plt.suptitle('Pair Plot of California Housing Features', y=1.02)**

**plt.show()**

**P3**

**from sklearn .datasets import load\_iris**

**from sklearn.decomposition import PCA**

**from sklearn.preprocessing import StandardScaler**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**iris = load\_iris()**

**feature = iris.data**

**target = iris.target**

**scaler = StandardScaler()**

**feature\_standardized = scaler.fit\_transform(feature)**

**pca = PCA(n\_components=2)**

**feature\_pca = pca.fit\_transform(feature\_standardized)**

**pca\_df = pd.DataFrame(data=feature\_pca, columns=['PC1', 'PC2'])**

**pca\_df['Target'] = target**

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

**for label, color in zip(iris.target\_names, ['red', 'blue', 'green']):**

**plt.scatter(pca\_df.loc[pca\_df['Target'] == list(iris.target\_names).index(label), 'PC1'],**

**pca\_df.loc[pca\_df['Target'] == list(iris.target\_names).index(label), 'PC2'],**

**label=label,**

**alpha=0.7,**

**c=color)**

**plt.title("pca on iris dataset[4 features to 2 features]", fontsize=14)**

**plt.xlabel("PC1", fontsize=12)**

**plt.ylabel("PC2", fontsize=12)**

**plt.legend(title="species")**

**plt.grid()**

**plt.show()**

**explained\_variance = pca.explained\_variance\_ratio\_**

**print("Explained Variance by each Principal Component:")**

**print("Principal Component 1: ",explained\_variance[0])**

**print("Principal Component 2: ",explained\_variance[1])**

**print("Total Variance Retained: ",sum(explained\_variance))**

**P4**

**import pandas as pd**

**data = pd.read\_csv("Dataset.csv")**

**print(data)**

**def find\_s\_algorithm(data):**

**"""Implements the Find-S algorithm to find the most specific hypothesis."""**

**attributes = data.iloc[:, :-1].values**

**target = data.iloc[:, -1].values**

**for i in range(len(target)):**

**if target[i] == "Yes":**

**hypothesis = attributes[i].copy()**

**break**

**for i in range(len(target)):**

**if target[i] == "Yes":**

**for j in range(len(hypothesis)):**

**if hypothesis[j] != attributes[i][j]:**

**hypothesis[j] = '?'**

**return hypothesis**

**final\_hypothesis = find\_s\_algorithm(data)**

**print("Most Specific Hypothesis:", final\_hypothesis)**

**P7**

**import pandas as pd**

**import matplotlib.pyplot as plt**

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

**from sklearn.linear\_model import LinearRegression**

**from sklearn.preprocessing import PolynomialFeatures, StandardScaler**

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

**from sklearn.pipeline import make\_pipeline**

**def linear\_regression\_boston\_housing():**

**housing = pd.read\_csv('housing.csv')**

**X = housing["total\_rooms"].values.reshape(-1, 1)**

**y = housing["median\_house\_value"].values**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**plt.scatter(X\_test, y\_test, color="blue", label="Actual")**

**plt.plot(X\_test, y\_pred, color="red", label="Predicted")**

**plt.xlabel("Total Rooms")**

**plt.ylabel("Median House Value")**

**plt.title("Linear Regression - Housing Dataset")**

**plt.legend()**

**plt.show()**

**print("Linear Regression - Housing Dataset")**

**print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))**

**print("R^2 Score:", r2\_score(y\_test, y\_pred))**

**def polynomial\_regression\_auto\_mpg():**

**data = pd.read\_csv('auto-mpg.csv')**

**data = data.dropna()**

**X = data["displacement"].values.reshape(-1, 1)**

**y = data["mpg"].values**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**poly\_model = make\_pipeline(PolynomialFeatures(degree=2), StandardScaler(), LinearRegression())**

**poly\_model.fit(X\_train, y\_train)**

**y\_pred = poly\_model.predict(X\_test)**

**plt.scatter(X\_test, y\_test, color="blue", label="Actual")**

**plt.scatter(X\_test, y\_pred, color="red", label="Predicted")**

**plt.xlabel("Displacement")**

**plt.ylabel("Miles per gallon (mpg)")**

**plt.title("Polynomial Regression - Auto MPG Dataset")**

**plt.legend()**

**plt.show()**

**print("Polynomial Regression - Auto MPG Dataset")**

**print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))**

**print("R^2 Score:", r2\_score(y\_test, y\_pred))**

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

**linear\_regression\_boston\_housing()**

**polynomial\_regression\_auto\_mpg()**

**P8**

**import pandas as pd**

**import matplotlib.pyplot as plt**

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

**from sklearn.tree import DecisionTreeClassifier, plot\_tree**

**from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix**

**from sklearn.tree import export\_graphviz**

**from IPython.display import Image**

**import pydotplus**

**import warnings**

**warnings.filterwarnings('ignore')**

**!pip install graphviz**

**!apt-get install graphviz -y**

**import graphviz**

**data = pd.read\_csv('Breast Cancer Dataset.csv')**

**pd.set\_option('display.max\_columns', None)**

**data.diagnosis.unique()**

**df = data.drop(['id'], axis=1)**

**df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})**

**X = df.drop('diagnosis', axis=1)**

**y = df['diagnosis']**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)**

**model = DecisionTreeClassifier(criterion='entropy')**

**model.fit(X\_train, y\_train)**

**model**

**import math**

**def entropy(column):**

**counts = column.value\_counts()**

**probabilities = counts / len(column)**

**return -sum(probabilities \* probabilities.apply(math.log2))**

**def conditional\_entropy(data, X, target):**

**feature\_values = data[X].unique()**

**weighted\_entropy = 0**

**for value in feature\_values:**

**subset = data[data[feature] == value]**

**weighted\_entropy += (len(subset) / len(data)) \* entropy(subset[target])**

**return weighted\_entropy**

**def information\_gain(data, X, target):**

**total\_entropy = entropy(data[target])**

**feature\_conditional\_entropy = conditional\_entropy(data, X, target)**

**return total\_entropy - feature\_conditional\_entropy**

**for feature in X:**

**ig = information\_gain(df,feature,'diagnosis')**

**print(f"Information Gain for {feature}: {ig}")**

**dot\_data = export\_graphviz(model, out\_file=None,feature\_names=X\_train.columns,rounded=True, proportion=False,precision=2, filled=True)**

**graph = pydotplus.graph\_from\_dot\_data(dot\_data)**

**Image(graph.create\_png())**

**plt.figure(figsize=(12, 8))**

**plot\_tree(model, filled=True, feature\_names=X.columns, class\_names=['Benign', 'Malignent'])**

**plt.show()**

**y\_pred = model.predict(X\_test)**

**y\_pred**

**accuracy = accuracy\_score(y\_test, y\_pred) \* 100**

**classification\_rep = classification\_report(y\_test, y\_pred)**

**print("Accuracy:", accuracy)**

**print("Classification Report:\n", classification\_rep)**

**df.head(1)**

**new = [[12.5, 19.2, 80.0, 500.0, 0.085, 0.1, 0.05, 0.02, 0.17, 0.06,0.4, 1.0, 2.5, 40.0, 0.006, 0.02, 0.03, 0.01, 0.02, 0.003,16.0, 25.0, 105.0, 900.0, 0.13, 0.25, 0.28, 0.12, 0.29, 0.08]]**

**y\_pred = model.predict(new)**

**if y\_pred[0] == 0:**

**print("Prediction: Benign")**

**else:**

**print("Prediction: Malignant")**

**P9**

**import numpy as np**

**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 confusion\_matrix, accuracy\_score, roc\_auc\_score**

**from sklearn.preprocessing import label\_binarize**

**data = fetch\_olivetti\_faces()**

**X, y = data.data, data.target**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**nb = GaussianNB()**

**nb.fit(x\_train, y\_train)**

**y\_pred = nb.predict(x\_test)**

**accuracy = round(accuracy\_score(y\_test, y\_pred) \* 100, 2)**

**print(f"Naive Bayes Accuracy: {accuracy}%")**

**print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))**

**mis\_idx = np.where(y\_pred != y\_test)[0]**

**print(f"Number of misclassified images: {len(mis\_idx)}")**

**plt.figure(figsize=(10, 2))**

**for i, idx in enumerate(mis\_idx[:5]):**

**plt.subplot(1, 5, i+1)**

**plt.imshow(x\_test[idx].reshape(64, 64), cmap='gray')**

**plt.title(f"T:{y\_test[idx]},P:{y\_pred[idx]}")**

**plt.axis('off')**

**plt.show()**

**y\_test\_bin = label\_binarize(y\_test, classes=np.unique(y\_test))**

**y\_pred\_prob = nb.predict\_proba(x\_test)**

**for i in range(y\_test\_bin.shape[1]):**

**auc = roc\_auc\_score(y\_test\_bin[:, i], y\_pred\_prob[:, i])**

**print(f"Class {i} AUC: {auc:.2f}")**

**P10**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.cluster import KMeans**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.decomposition import PCA**

**data = pd.read\_csv("Wisconsin Breast Cancer dataset.csv")**

**df = data.drop(['id', 'Unnamed: 32'], axis=1)**

**df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})**

**X = df.drop(columns=["diagnosis"])**

**X\_scaled = StandardScaler().fit\_transform(X)**

**X\_pca = PCA(n\_components=2).fit\_transform(X\_scaled)**

**wcss = []**

**for k in range(1, 11):**

**kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)**

**kmeans.fit(X\_pca)**

**wcss.append(kmeans.inertia\_)**

**plt.plot(range(1, 11), wcss, marker="o")**

**plt.xlabel("Number of Clusters (k)")**

**plt.ylabel("WCSS")**

**plt.title("Elbow Method")**

**plt.show()**

**optimal\_k = 2**

**kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42, n\_init=10)**

**clusters = kmeans.fit\_predict(X\_pca)**

**plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=clusters, cmap="viridis", alpha=0.6)**

**plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=200, c="red", label="Centroids")**

**plt.xlabel("Principal Component 1")**

**plt.ylabel("Principal Component 2")**

**plt.title("K-Means Clustering after PCA")**

**plt.legend()**

**plt.show()**