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()
```