Program 1

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os

# Step 1: Load the California Housing dataset from directory
file_path = r"C:/Users/Admin/Desktop/datasets/housing (1).csv"  # Update this path to your local file
if os.path.exists(file_path):
    housing_df = pd.read_csv(file_path)
    print("\nDataset loaded successfully!")
else:
    print(f"\nError: File not found at {file_path}")
    exit()

# Step 2: Create histograms for numerical features
numerical_features = housing_df.select_dtypes(include=[np.number]).columns

# Plot histograms
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()

# Step 3: Generate box plots for numerical features
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()

# Step 4: Identify outliers using the IQR method
print("\nOutliers 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")

# Step 5: Print a summary of the dataset
print("\nDataset Summary:")
print(housing_df.describe())
```

program 2

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
# Step 1: Load the California Housing Dataset
california_data = fetch_california_housing(as_frame=True)
```

```
data = california_data.frame
# Step 2: Compute the correlation matrix
correlation_matrix = data.corr()
# Step 3: Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of California Housing Features')
plt.show()
# Step 4: Create a pair plot to visualize pairwise relationships
sns.pairplot(data, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pair Plot of California Housing Features', y=1.02)
plt.show()
```

program 3

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = load_iris()
data = iris.data
labels = iris.target
label_names = iris.target_names
# Convert to a DataFrame for better visualization
iris_df = pd.DataFrame(data, columns=iris.feature_names)
# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(data)
# Create a DataFrame for the reduced data
reduced_df = pd.DataFrame(data_reduced,columns=['Principal Component 1','Principal Component 2'])
```

```
reduced_df['Label'] = labels
# Plot the reduced data
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b']
for i, label in enumerate(np.unique(labels)):plt.scatter(
 reduced_df[reduced_df['Label'] == label]['Principal Component 1'],
 reduced_df[reduced_df['Label'] == label]['Principal Component 2'],
 label=label_names[label],
 color=colors[i]
 )
plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

Program 4

```
import pandas as pd

def find_s_algorithm(file_path):
    data = pd.read_csv(file_path)

    print("Training data:")
    print(data)

    attributes = data.columns[:-1]
    class_label = data.columns[-1]

    hypothesis = ['?' for _ in attributes]
```

```
    for index, row in data.iterrows():
        if row[class_label] == 'Yes':
            for i, value in enumerate(row[attributes]):
                if hypothesis[i] == '?' or hypothesis[i] == value:
                    hypothesis[i] = value
                else:
                    hypothesis[i] = '?'

    return hypothesis

file_path = 'C:/Users/Admin/Desktop/datasets/data1.csv'
hypothesis = find_s_algorithm(file_path)
print("\nThe final hypothesis is:", hypothesis)
```

Program 5

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

data = np.random.rand(100)

labels = ["Class1" if x <= 0.5 else "Class2" for x in data[:50]]


def euclidean_distance(x1, x2):
    return abs(x1 - x2)


def knn_classifier(train_data, train_labels, test_point, k):
    distances = [(euclidean_distance(test_point, train_data[i]), train_labels[i]) for i in
range(len(train_data))]
```

```
    distances.sort(key=lambda x: x[0])
    k_nearest_neighbors = distances[:k]

    k_nearest_labels = [label for _, label in k_nearest_neighbors]

    return Counter(k_nearest_labels).most_common(1)[0][0]



train_data = data[:50]
train_labels = labels

test_data = data[50:]

k_values = [1, 2, 3, 4, 5, 20, 30]

print("--- k-Nearest Neighbors Classification ---")
print("Training dataset: First 50 points labeled based on the rule (x <= 0.5 -> Class1, x > 0.5 -> Class2)")
print("Testing dataset: Remaining 50 points to be classified\n")

results = {}

for k in k_values:
    print(f"Results for k = {k}:")
    classified_labels = [knn_classifier(train_data, train_labels, test_point, k) for test_point in test_data]
    results[k] = classified_labels

    for i, label in enumerate(classified_labels, start=51):
        print(f"Point x{i} (value: {test_data[i - 51]:.4f}) is classified as {label}")
    print("\n")
```

```
print("Classification complete.\n")


for k in k_values:
    classified_labels = results[k]
    class1_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] == "Class1"]
    class2_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] == "Class2"]


    plt.figure(figsize=(10, 6))
    plt.scatter(train_data, [0] * len(train_data), c=["blue" if label == "Class1" else "red" for label in
train_labels],
                label="Training Data", marker="o")
    plt.scatter(class1_points, [1] * len(class1_points), c="blue", label="Class1 (Test)", marker="x")
    plt.scatter(class2_points, [1] * len(class2_points), c="red", label="Class2 (Test)", marker="x")


    plt.title(f"k-NN Classification Results for k = {k}")
    plt.xlabel("Data Points")
    plt.ylabel("Classification Level")
    plt.legend()
    plt.grid(True)
    plt.show()
```

Program 6

```
import numpy as np
import matplotlib.pyplot as plt
def gaussian_kernel(x,xi,tau):
    return np.exp(-np.sum((x - xi)*2)/(2*tau*2))
def locally_weighted_regression(x,X,y,tau):
    m=X.shape[0]
    weights=np.array([gaussian_kernel(x,X[i],tau) for i in range(m)])
    W=np.diag(weights)
```

```
    X_transpose_W=X.T @W
    theta=np.linalg.inv(X_transpose_W@X)@X_transpose_W@y
    return x @ theta
np.random.seed(42)
X=np.linspace(0,2*np.pi,100)
y=np.sin(X)+0.1*np.random.randn(100)
X_bias=np.c_[np.ones(X.shape),X]
x_test=np.linspace(0,2*np.pi,200)
x_test_bias= np.c_[np.ones(x_test.shape),x_test]
tau=0.5
y_pred=np.array([locally_weighted_regression(xi, X_bias, y, tau)
                 for xi in x_test_bias])
plt.figure(figsize=(10,6))
plt.scatter(X,y,color='red',label='Training Data',alpha=0.7)
plt.plot(x_test,y_pred,color='blue',label=f'LWR Fit(tau={tau})',linewidth=2)
plt.xlabel('X',fontsize=12)
plt.ylabel('y',fontsize=12)
plt.title('Locally Weighted Regression',fontsize=14)
plt.legend(fontsize=10)
plt.grid(alpha=0.3)
plt.show()
```

Program 7

```
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
from sklearn.preprocessing import PolynomialFeatures
```

```
# ---------------------- Part 1: Linear Regression on Boston Housing ----------------------

# Load Boston Housing Dataset
column_names = [
    'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
    'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV'
]
boston_data = pd.read_csv("C:/Users/Admin/Desktop/3vc22cs077/hello/BostonHousing.csv",
header=None, names=column_names)

# Feature and target
X = boston_data[['RM']]  # Average number of rooms
y = boston_data['MEDV']  # Median value of homes

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Plot
plt.figure(figsize=(8, 5))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted', linewidth=2)
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median Value (MEDV)")
plt.title("Linear Regression on Boston Housing Data")
```

```
plt.legend()
plt.grid(True)
plt.show()

# Evaluation
print("Linear Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))



# ---------- Part 2: Polynomial Regression on Auto MPG ----------

# Load Auto MPG dataset (Make sure the correct file is used here)
auto_data = pd.read_csv("C:/Users/Admin/Desktop/3vc22cs077/hello/auto-mpg.csv")

# Clean data
auto_data.replace('?', np.nan, inplace=True)
auto_data.dropna(inplace=True)
auto_data['horsepower'] = auto_data['horsepower'].astype(float)

# Feature and target
X_auto = auto_data[['horsepower']].values
y_auto = auto_data['mpg'].values

# Polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_auto)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_auto, test_size=0.2, random_state=42)
```

```
# Train model
poly_reg = LinearRegression()
poly_reg.fit(X_train, y_train)

# Predict
y_pred = poly_reg.predict(X_test)

# Plot sorted results
sort_idx = X_test[:, 1].argsort()
X_sorted = X_test[sort_idx][:, 1]
y_sorted = y_pred[sort_idx]

plt.figure(figsize=(8, 5))
plt.scatter(X_test[:, 1], y_test, color='green', label='Actual')
plt.plot(X_sorted, y_sorted, color='orange', linewidth=2, label='Predicted')
plt.title("Polynomial Regression - Auto MPG (Horsepower vs MPG)")
plt.xlabel("Horsepower")
plt.ylabel("Miles per Gallon (MPG)")
plt.legend()
plt.grid(True)
plt.show()

# Evaluation
print("Polynomial Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```

Program 8

```
# -- coding: utf-8 --
"""
Created on Fri Apr 25 14:53:20 2025
```

@author: Admin

"""

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
data = load_breast_cancer()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
new_sample = np.array([X_test[0]])
prediction = clf.predict(new_sample)
prediction_class = "Benign" if prediction == 1 else "Malignant"
print(f"Predicted Class for the new sample: {prediction_class}")
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.title("Decision Tree - Breast Cancer Dataset")
plt.show()
```

Program 9

```
# -- coding: utf-8 --
```

```
"""
Created on Fri Apr 25 14:56:47 2025

@author: Admin
"""

import numpy as np
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

data = fetch_olivetti_faces(shuffle=True, random_state=42)
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=1))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
cross_val_accuracy = cross_val_score(gnb, X, y, cv=5, scoring='accuracy')
print(f'\nCross-validation accuracy: {cross_val_accuracy.mean() * 100:.2f}%')

fig, axes = plt.subplots(3, 5, figsize=(12, 8))
for ax, image, label, prediction in zip(axes.ravel(), X_test, y_test, y_pred):
    ax.imshow(image.reshape(64, 64), cmap=plt.cm.gray)
    ax.set_title(f"True: {label}, Pred: {prediction}")
    ax.axis('off')
plt.show()
```

Program 10

```
# -- coding: utf-8 --
"""
Created on Fri Apr 25 15:06:51 2025

@author: Admin
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report

data = load_breast_cancer()
X = data.data
```

```
y = data.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=2, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)

print("Confusion Matrix:")
print(confusion_matrix(y, y_kmeans))
print("\nClassification Report:")
print(classification_report(y, y_kmeans))

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df['Cluster'] = y_kmeans
df['True Label'] = y

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100, edgecolor='black', alpha=0.7)
plt.title('K-Means Clustering of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="Cluster")
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='True Label', palette='coolwarm', s=100, edgecolor='black', alpha=0.7)
```

```
plt.title('True Labels of Breast Cancer Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(title="True Label")

plt.show()


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

sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100, edgecolor='black', alpha=0.7)

centers = pca.transform(kmeans.cluster_centers_)

plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='Centroids')

plt.title('K-Means Clustering with Centroids')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(title="Cluster")

plt.show()
```