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

# Create the dataset
data = pd.DataFrame({
    'Age': [23, 23, 27, 27, 39, 41, 47, 49, 50, 52, 54, 54, 56, 57, 58, 58, 60,
61],
    '%Fat': [7.8, 9.5, 17.8, 25.9, 26.5, 27.2, 27.4, 28.8, 30.2, 31.2, 31.4, 32.9,
             33.4, 34.1, 34.6, 35.7, 41.2, 42.5]
})

# Save dataset
data.to_csv('Age_Fat.csv', index=False)

# Load dataset
data = pd.read_csv('Age_Fat.csv')
print("Dataset Preview:\n", data.head())

# ---------- Numerical Column: %Fat ----------
numerical_column = '%Fat'
data_num = data[numerical_column]

# Compute statistics
print("\nStatistics:")
print("Mean:", data_num.mean())
print("Median:", data_num.median())
print("Mode:", data_num.mode()[0])
print("Standard Deviation:", data_num.std())
print("Variance:", data_num.var())
print("Range:", data_num.max() - data_num.min())

# Histogram
plt.figure(figsize=(8, 5))
plt.hist(data_num, bins=10, color='skyblue', edgecolor='black')
plt.title("Histogram of %Fat")
plt.xlabel("%Fat")
plt.ylabel("Frequency")
plt.show()

# Boxplot
plt.figure(figsize=(8, 5))
sns.boxplot(x=data_num, color='lightgreen')
plt.title("Boxplot of %Fat")
plt.show()

# Outlier detection using IQR
q1 = data_num.quantile(0.25)
q3 = data_num.quantile(0.75)
```

```
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
outliers = data_num[(data_num < lower_bound) | (data_num > upper_bound)]
print("\nOutliers Detected:\n", outliers)

# ---------- Categorical Column: Age Group ----------
def age_group(age):
    if age < 30:
        return 'Young'
    elif 30 <= age <= 50:
        return 'Middle-aged'
    else:
        return 'Older'

data['Age Group'] = data['Age'].apply(age_group)
category_counts = data['Age Group'].value_counts()

print("\nCategory Frequencies:\n", category_counts)

# Bar Chart
plt.figure(figsize=(8, 5))
category_counts.plot(kind='bar', color='coral', edgecolor='black')
plt.title("Bar Chart of Age Group")
plt.xlabel("Age Group")
plt.ylabel("Frequency")
plt.show()

# Pie Chart
plt.figure(figsize=(8, 5))
category_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90,
                     colors=sns.color_palette('pastel'))
plt.title("Pie Chart of Age Group")
plt.ylabel("")
plt.show()
```

2.

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

df = pd.read_csv('Sample_Marks.csv')
x, y = 'Test Score', 'Video Game'
selected = df[[x, y]]

plt.scatter(selected[x], selected[y], alpha=0.7)
plt.title(f"{x} vs {y}")
```

```
plt.show()

print("Pearson Correlation:", np.corrcoef(selected[x], selected[y])[0, 1])
print("Covariance Matrix:\n", np.cov(selected[x], selected[y]))
print("Correlation Matrix:\n", selected.corr())

sns.heatmap(selected.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

3.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

file_path = "10-dataset.csv"
tips = pd.read_csv(file_path)

X = tips["total_bill"].values
y = tips["tip"].values

def locally_weighted_regression(x_query, X, y, tau):
    m = len(X)
    weights = np.exp(-((X - x_query) ** 2) / (2 * tau ** 2))
    X_b = np.c_[np.ones(m), X]
    W = np.diag(weights)
    theta = np.linalg.pinv(X_b.T @ W @ X_b) @ (X_b.T @ W @ y)
    x_query_b = np.array([1, x_query])
    return x_query_b @ theta

tau = 10
x_query = 30
predicted_tip = locally_weighted_regression(x_query, X, y, tau)
print(f"Predicted Tip for a total bill of $30: {predicted_tip:.2f}")

X_range = np.linspace(X.min(), X.max(), 100)
y_pred = np.array([locally_weighted_regression(x, X, y, tau) for x in X_range])

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', alpha=0.5, label="Actual Data")
plt.plot(X_range, y_pred, color='blue', label="Locally Weighted Regression")
plt.scatter([x_query], [predicted_tip], color='green',
            marker='o', label="Prediction ($30 bill)")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.legend()
plt.title("Locally Weighted Regression on Tips Dataset")
plt.grid(True)
plt.show()
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder

file_path = 'IRIS.csv'
df = pd.read_csv(file_path)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

def evaluate_knn(X_train, X_test, y_train, y_test, k_values, weighted=False):
    results = {}
    for k in k_values:
        if weighted:
            knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
        else:
            knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        conf_matrix = confusion_matrix(y_test, y_pred)
        results[k] = {'accuracy': accuracy, 'f1_score': f1, 'conf_matrix':
conf_matrix}
    return results

k_values = [1, 3, 5]

regular_knn_results = evaluate_knn(X_train, X_test, y_train,
                                   y_test, k_values, weighted=False)
print("Regular k-NN Results:")
for k, metrics in regular_knn_results.items():
    print(f"\nk={k}: Accuracy={metrics['accuracy']:.4f},
          F1 Score={metrics['f1_score']:.4f}")
    print("Confusion Matrix:")
    print(metrics['conf_matrix'])
```

```
weighted_knn_results = evaluate_knn(X_train, X_test,
                                    y_train, y_test, k_values, weighted=True)
print("\nWeighted k-NN Results:")
for k, metrics in weighted_knn_results.items():
    print(f"\nk={k}: Accuracy={metrics['accuracy']:.4f},
          F1 Score={metrics['f1_score']:.4f}")
    print("Confusion Matrix:")
    print(metrics['conf_matrix'])
```

```
5.import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('iris.csv')
print("Dataset preview:")
print(df.head())

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

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

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

unique_labels = np.unique(y)

plt.figure(figsize=(8, 6))
for target in unique_labels:
    plt.scatter(X_pca[y == target, 0], X_pca[y == target,
                                             1], label=target, alpha=0.7)

print("\n\nOriginal dataset shape:", X_scaled.shape)
print("Reduced dataset shape:", X_pca.shape)

plt.title('PCA of Dataset (Reduced to 2D)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
print("Explained variance ratio:", pca.explained_variance_ratio_)
==================================================================================
================
6.import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

# ------------------ Linear Regression (Boston Housing) ------------------

# Load Boston Housing dataset
boston_data = pd.read_csv("BostonHousing.csv")

# Feature and target
X_boston = boston_data[['rm']]   # Average number of rooms
y_boston = boston_data['medv']   # Median home value

# Train-test split
X_train_b, X_test_b, y_train_b, y_test_b = train_test_split(
    X_boston, y_boston, test_size=0.2, random_state=42)

# Train Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train_b, y_train_b)

# Predict
y_pred_b = lin_reg.predict(X_test_b)

# Plot Linear Regression results
plt.figure(figsize=(8, 6))
plt.scatter(X_test_b, y_test_b, color='blue',
            label='Actual Prices', alpha=0.6)
plt.plot(X_test_b, y_pred_b, color='red',
         linewidth=2, label='Regression Line')
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median Value of Homes (MEDV)")
plt.title("Linear Regression: RM vs MEDV")
plt.legend()
plt.grid(True)
plt.show()


# ---------------- Polynomial Regression (Auto MPG) ---------------------

# Load Auto MPG dataset
auto_data = pd.read_csv("auto-mpg.csv")

# Clean 'horsepower' column
auto_data.replace({'horsepower': {'?': np.nan}}, inplace=True)
auto_data.dropna(subset=['horsepower'], inplace=True)
auto_data['horsepower'] = auto_data['horsepower'].astype(float)
```

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

# Train-test split
X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(
    X_auto, y_auto, test_size=0.2, random_state=42)

# Polynomial features (degree 2)
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_a)
X_test_poly = poly.transform(X_test_a)

# Train Polynomial Regression model
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train_a)

# Predict
y_pred_a = poly_reg.predict(X_test_poly)

# Plot Polynomial Regression results
plt.figure(figsize=(8, 6))
plt.scatter(X_test_a, y_test_a, color='green',
            label='Actual MPG', alpha=0.6)
sorted_idx = X_test_a['horsepower'].argsort()
plt.plot(X_test_a.iloc[sorted_idx], y_pred_a[sorted_idx],
         color='orange', label='Polynomial Fit')
plt.xlabel("Horsepower")
plt.ylabel("MPG")
plt.title("Polynomial Regression: Horsepower vs MPG")
plt.legend()
plt.grid(True)
plt.show()
===================================================================================
===================
7.import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

df = pd.read_csv('titanic.csv')
df = df[['Survived', 'Pclass', 'Sex', 'Age', 'Fare']].dropna()
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

X = df[['Pclass', 'Sex', 'Age', 'Fare']]
y = df['Survived']

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)

plt.figure(figsize=(20, 12))
plot_tree(clf, feature_names=X.columns, class_names=["Not Survived", "Survived"],
          filled=True, rounded=True, fontsize=12)
plt.title("Decision Tree - Titanic Survival Prediction", fontsize=16)
plt.show()

y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
***********************************************************************************
*******************
8.import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = pd.read_csv('iris.csv')

X = data.iloc[:, :-1]
y = data.iloc[:, -1]

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

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Naive Bayes Classifier: {accuracy * 100:.2f}%")
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
-_-_-_-_-_-_-_-_
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.stats import mode

df = pd.read_csv("Breast Cancer Wisconsin.csv")
df = df.drop(columns=['id', 'Unnamed: 32'], errors='ignore')
```

```
if 'diagnosis' in df.columns:
    diagnosis = df['diagnosis'].map({'M': 1, 'B': 0})
    df = df.drop(columns=['diagnosis'])
else:
    diagnosis = None

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(scaled_data)
labels = kmeans.labels_

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

vis_df = pd.DataFrame({
    'PCA1': pca_data[:, 0],
    'PCA2': pca_data[:, 1],
    'Cluster': labels
})

if diagnosis is not None:
    vis_df['Actual'] = diagnosis
    cluster_map = {}
    for cluster in [0, 1]:
        majority = mode(diagnosis[labels == cluster], keepdims=True).mode[0]
        cluster_map[cluster] = 'Predicted Malignant'
        if majority == 1 else 'Predicted Benign'
    vis_df['Cluster_Label'] = vis_df['Cluster'].map(cluster_map)
    vis_df['Actual_Label'] = vis_df['Actual']
    .map({1: 'Actual Malignant', 0: 'Actual Benign'})

plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=vis_df,
    x='PCA1', y='PCA2',
    hue='Cluster_Label',
    palette={'Predicted Malignant': 'red', 'Predicted Benign': 'blue'},
    s=100,
    alpha=0.6,
    legend='full'
)
sns.scatterplot(
    data=vis_df,
    x='PCA1', y='PCA2',
    style='Actual_Label',
    markers={'Actual Malignant': 'X', 'Actual Benign': 'o'},
    color='black',
    s=50,
```

```
    alpha=0.5,
    legend='brief'
)
plt.title("K-Means Clustering on Breast Cancer Dataset (PCA Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Legend", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^
_^_
```