ML- Meghana

April 1, 2025

Meghana Korimi

<center>22WU0101130</center>
<center>CSE-B</center>
<center>Machine Learning Lab Mnual</center>

TOC

Lab 1

Lab 2

Lab 3

Lab 4

Lab 5

Lab 6

Lab 7

Lab 8

Lab 9

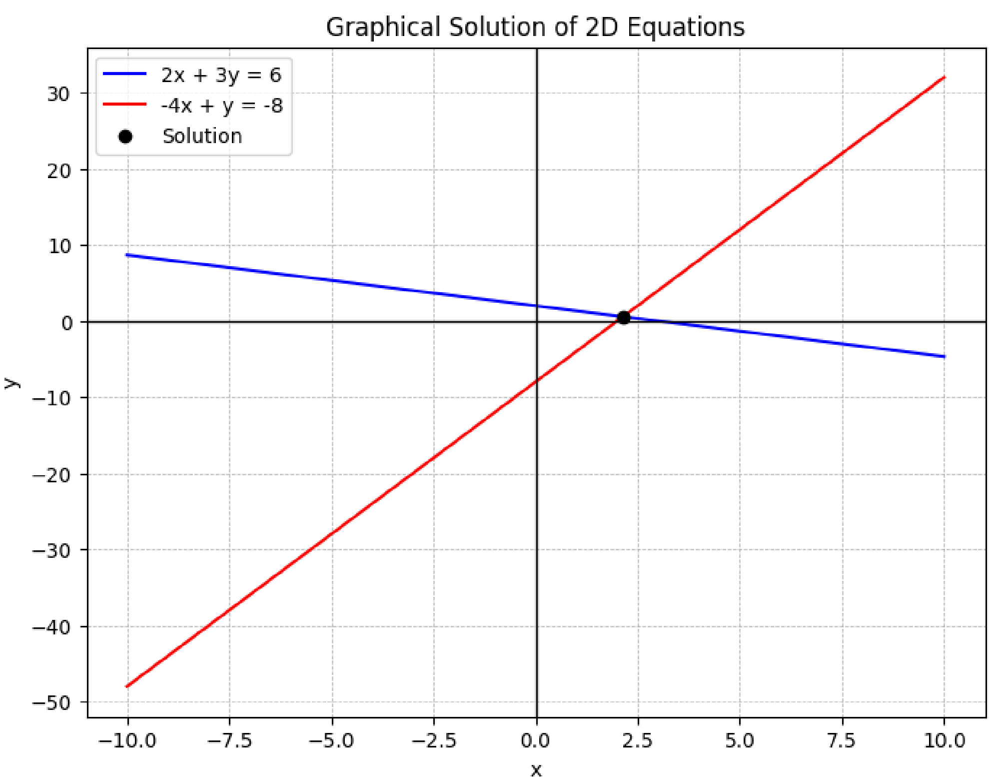
Lab 1

1 Lab 1 Solving Systems of Equations using Matrix Method and Effect of Outliers on Mean and Median

[50]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

A = np.array([[2, 3], [-4, 1]])
B = np.array([6, -8])

solution = np.linalg.solve(A, B)
x_sol, y_sol = solution

print(f"2D Solution: x = {x_sol}, y = {y_sol}")
```

```
x_vals = np.linspace(-10, 10, 400)
y1 = (6 - 2*x_vals) / 3
y2 = (-8 + 4*x_vals)

plt.figure(figsize=(8,6))
plt.plot(x_vals, y1, label="2x + 3y = 6", color="blue")
plt.plot(x_vals, y2, label="-4x + y = -8", color="red")

plt.scatter(x_sol, y_sol, color="black", zorder=3, label="Solution")

plt.xlabel("x")
plt.ylabel("y")
plt.axhline(0, color="black", linewidth=1)
plt.axvline(0, color="black", linewidth=1)
plt.grid(True, linestyle="--", linewidth=0.5)
plt.legend()
plt.title("Graphical Solution of 2D Equations")
plt.show()
```

```
2D Solution: x = 2.142857142857143, y = 0.5714285714285714
```

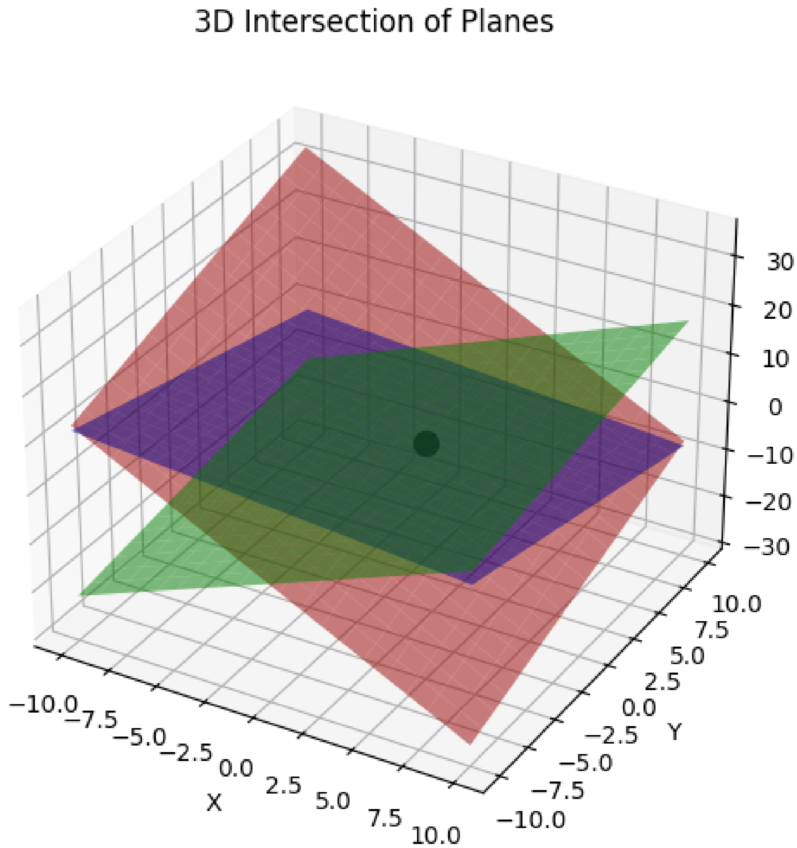


[3]:

```
A_3D = np.array([[1, 2, 3], [2, -1, 1], [3, 1, -2]])
B_3D = np.array([6, 3, 4])

try:
    solution_3D = np.linalg.solve(A_3D, B_3D)
    x_3D, y_3D, z_3D = solution_3D
    print(f"3D Unique Solution: x = {x_3D}, y = {y_3D}, z = {z_3D}")
except np.linalg.LinAlgError:
    print("3D System has no unique solution.")

det_A = np.linalg.det(A_3D)

if det_A != 0:
    print("Unique solution exists.")
elif np.linalg.matrix_rank(A_3D) == np.linalg.matrix_rank(np.
 ↪column_stack((A_3D, B_3D))):
    print("Infinitely many solutions exist.")
else:
    print("No solution exists.")

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')

x_range = np.linspace(-10, 10, 20)
y_range = np.linspace(-10, 10, 20)
X, Y = np.meshgrid(x_range, y_range)

Z1 = (6 - X - 2*Y) / 3
Z2 = (3 - 2*X + Y)
Z3 = (4 - 3*X - Y) / -2

# Plot planes
ax.plot_surface(X, Y, Z1, color='blue', alpha=0.5, label="Plane 1")
ax.plot_surface(X, Y, Z2, color='red', alpha=0.5, label="Plane 2")
ax.plot_surface(X, Y, Z3, color='green', alpha=0.5, label="Plane 3")

if det_A != 0:
    ax.scatter(x_3D, y_3D, z_3D, color="black", s=100, label="Intersection␣
 ↪Point")

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title("3D Intersection of Planes")
plt.show()
```

```
3D Unique Solution: x = 1.5666666666666667, y = 0.9666666666666667, z =
0.8333333333333335
```

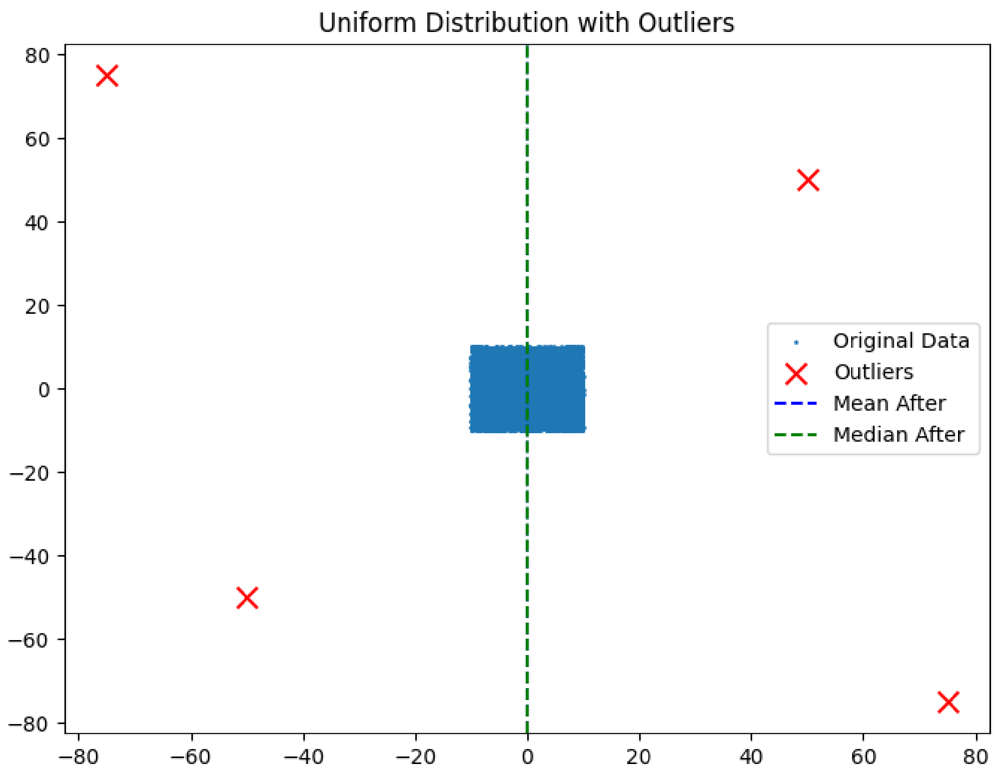
```
Unique solution exists.
```



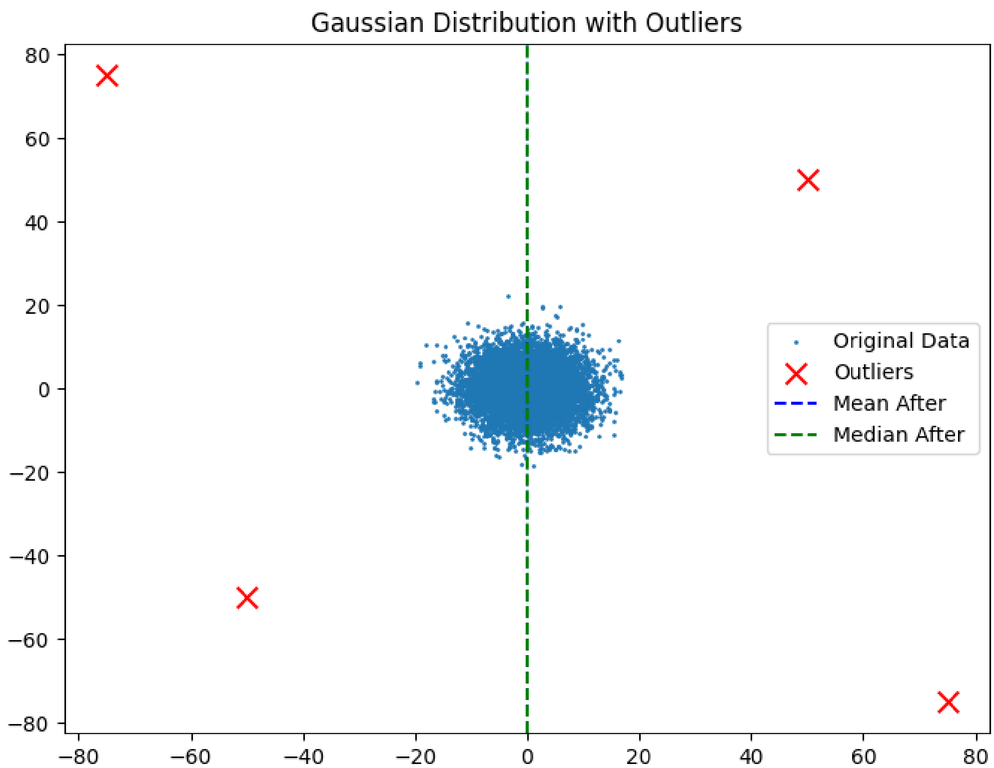
[4]:
```python
import numpy as np
import matplotlib.pyplot as plt

def generate_and_plot(distribution="uniform"):
    np.random.seed(42)

    if distribution == "uniform":
        data = np.random.uniform(-10, 10, size=(10000, 2))
    else:
        data = np.random.normal(0, 5, size=(10000, 2))

    mean_before = np.mean(data, axis=0)
    median_before = np.median(data, axis=0)

    outliers = np.array([[50, 50], [-50, -50], [75, -75], [-75, 75]])
```

```
    data_with_outliers = np.vstack([data, outliers])

    mean_after = np.mean(data_with_outliers, axis=0)
    median_after = np.median(data_with_outliers, axis=0)

    print(f"\n{distribution.capitalize()} Distribution:")
    print(f"Mean before outliers: {mean_before}")
    print(f"Median before outliers: {median_before}")
    print(f"Mean after outliers: {mean_after}")
    print(f"Median after outliers: {median_after}")

    plt.figure(figsize=(8, 6))
    plt.scatter(data[:, 0], data[:, 1], s=1, label="Original Data")
    plt.scatter(outliers[:, 0], outliers[:, 1], color='red', label="Outliers",␣
↪marker='x', s=100)
    plt.axvline(mean_after[0], color='blue', linestyle='--', label="Mean After")
    plt.axvline(median_after[0], color='green', linestyle='--', label="Median␣
↪After")
    plt.title(f"{distribution.capitalize()} Distribution with Outliers")
    plt.legend()
    plt.show()

generate_and_plot("uniform")
generate_and_plot("gaussian")
```

```
Uniform Distribution:
Mean before outliers: [-0.03052838  0.00431707]
Median before outliers: [-0.03280336  0.00543043]
Mean after outliers: [-0.03051617  0.00431534]
Median after outliers: [-0.03280336  0.00543043]
```




```
Gaussian Distribution:
Mean before outliers: [0.0179704  0.03901995]
Median before outliers: [0.02403672 0.0563448 ]
Mean after outliers: [0.01796321 0.03900435]
Median after outliers: [0.02403672 0.0563448 ]
```



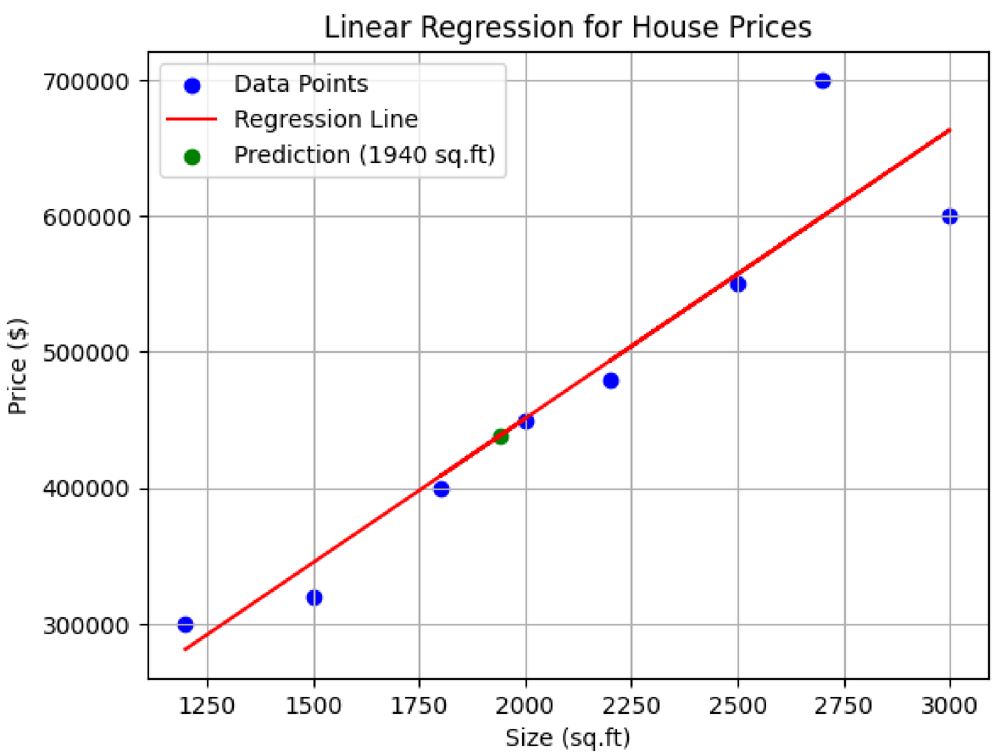
Lab 2

2 Linear Regression for House Price Prediction

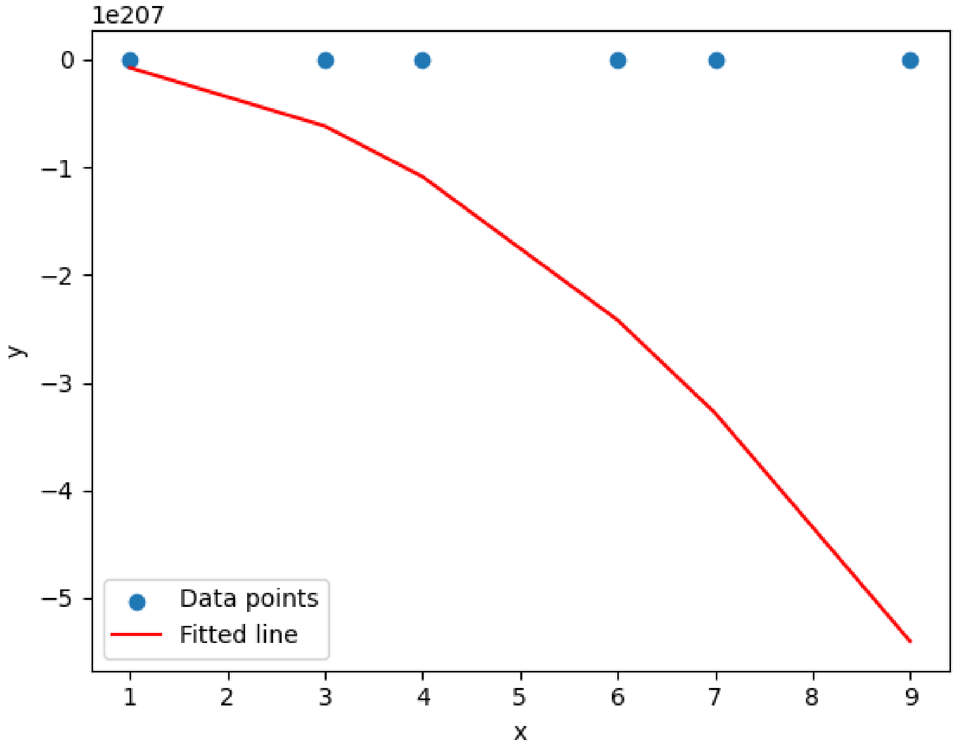
[51]:
```
import numpy as np
import matplotlib.pyplot as plt
X = np.array([[1200], [1500]])
y = np.array([300000, 320000])
X_new = np.array([[2000], [1800], [2500], [3000], [2200], [2700]])
y_new = np.array([450000, 400000, 550000, 600000, 480000, 700000])
X_combined = np.vstack([X, X_new])
y_combined = np.hstack([y, y_new])
X_combined_bias = np.hstack([np.ones((X_combined.shape[0], 1)), X_combined])

# Normal equation: theta = (X^T X)^(-1) X^T y
theta = np.linalg.inv(X_combined_bias.T @ X_combined_bias) @ X_combined_bias.T
 ↪@ y_combined

# Predicted house price for a house size of 1940
```

```python
size_to_predict = 1940
X_predict = np.array([1, size_to_predict])  # Add bias term
predicted_price = X_predict @ theta

print(f"Linear Regression Coefficients (theta): {theta}")
print(f"Predicted price for a 1940 sq.ft house: ${predicted_price:.2f}")

# Visualization
plt.scatter(X_combined, y_combined, color='blue', label='Data Points')
plt.plot(X_combined, X_combined_bias @ theta, color='red', label='Regression␣
 ↪Line')
plt.scatter(size_to_predict, predicted_price, color='green', label='Prediction␣
 ↪(1940 sq.ft)')
plt.xlabel('Size (sq.ft)')
plt.ylabel('Price ($)')
plt.title('Linear Regression for House Prices')
plt.legend()
plt.grid()
plt.show()
```

```
Linear Regression Coefficients (theta): [26789.65021562   212.17057978]
Predicted price for a 1940 sq.ft house: $438400.57
```



[52]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Data points
x = np.array([1, 3,4, 6, 7,9])
y = np.array([5,2, 5, 4, 1, 7])

# Initial parameters
a, b, c = 0, 0, 0
learning_rate = 0.001

iterations = 500

for _ in range(iterations):
    y_pred = a * x**2 + b * x + c
    da = -2 * np.sum(x**2 * (y - y_pred)) / len(x)
    db = -2 * np.sum(x * (y - y_pred)) / len(x)
    dc = -2 * np.sum(y - y_pred) / len(x)
    a -= learning_rate * da
    b -= learning_rate * db
    c -= learning_rate * dc
plt.scatter(x, y, label='Data points')
plt.plot(x, a*(x**2)+b*x+c, color='red', label='Fitted line')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

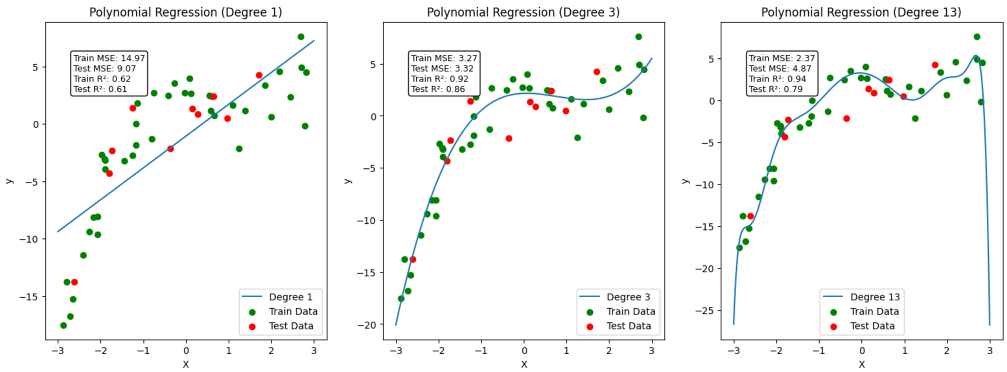


Lab 3

3 Linear Regression using Gradient Descent

[33]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline

# Generate synthetic data
np.random.seed(42)
X = np.random.uniform(-3, 3, 50).reshape(-1, 1)
y = 0.5 * X**3 - X**2 + 2 + np.random.normal(0, 2, size=X.shape)

# Function to perform Polynomial Regression
def polynomial_regression(degree, ax):
```

```
    model = Pipeline([('poly', PolynomialFeatures(degree)), ('regressor',␣
↪LinearRegression())])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    x_range = np.linspace(-3, 3, 1000).reshape(-1, 1)
    ax.plot(x_range, model.predict(x_range), label=f'Degree {degree}')
    ax.scatter(X_train, y_train, color='green', label='Train Data')
    ax.scatter(X_test, y_test, color='red', label='Test Data')

    # Add text with box around it
    ax.text(0.1, 0.9, f'Train MSE: {train_mse:.2f}\nTest MSE: {test_mse:.
↪2f}\nTrain R²: {train_r2:.2f}\nTest R²: {test_r2:.2f}',
            fontsize=9, ha='left', va='top', transform=ax.transAxes,
            bbox=dict(facecolor='white', edgecolor='black',␣
↪boxstyle='round,pad=0.3'))

    ax.set_title(f'Polynomial Regression (Degree {degree})')
    ax.set_xlabel('X')
    ax.set_ylabel('y')
    ax.legend()

# Plotting for different degrees
fig, axs = plt.subplots(1, 3, figsize=(18, 6))
for i, degree in enumerate([1, 3, 13]):
    polynomial_regression(degree, axs[i])

plt.show()
```



Lab 4

4 Multiple and Polynomial Regression using the California Housing Dataset

[34]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression

# Load dataset
data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Preprocessing: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply polynomial features (Degree = 2)
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)
```

```python
# Train Linear Regression model with polynomial features
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Predict on test data
y_pred = model.predict(X_test_poly)

# Calculate Mean Squared Error manually
mse = np.sum((y_test - y_pred) ** 2) / len(y_test)

# Calculate R² Score manually
y_mean = np.mean(y_test)
ss_total = np.sum((y_test - y_mean) ** 2)
ss_residual = np.sum((y_test - y_pred) ** 2)
r2 = 1 - (ss_residual / ss_total)

print(f"Mean Squared Error (Polynomial Degree=2): {mse:.4f}")
print(f"R² Score (Polynomial Degree=2): {r2:.4f}")

### **1. Linear Regression**
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
y_pred_linear = linear_model.predict(X_test_scaled)

# Calculate MSE for Linear Regression manually
mse_linear = np.sum((y_test - y_pred_linear) ** 2) / len(y_test)

# Calculate R² Score for Linear Regression manually
ss_residual_linear = np.sum((y_test - y_pred_linear) ** 2)
r2_linear = 1 - (ss_residual_linear / ss_total)

# Variance of Residuals (Linear Regression)
residuals_linear = y_test - y_pred_linear
variance_linear = np.sum((residuals_linear - np.mean(residuals_linear)) ** 2) /
 ↪len(residuals_linear)

print("\n### Linear Regression Results ###")
print(f"Mean Squared Error: {mse_linear:.4f}")
print(f"R² Score: {r2_linear:.4f}")
print(f"Variance of Residuals: {variance_linear:.4f}")

### **2. Polynomial Regression (Degree = 4)**
poly = PolynomialFeatures(degree=4)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

poly_model = LinearRegression()
```

```
poly_model.fit(X_train_poly, y_train)
y_pred_poly = poly_model.predict(X_test_poly)

# Calculate MSE for Polynomial Regression manually
mse_poly = np.sum((y_test - y_pred_poly) ** 2) / len(y_test)

# Calculate R² Score for Polynomial Regression manually
ss_residual_poly = np.sum((y_test - y_pred_poly) ** 2)
r2_poly = 1 - (ss_residual_poly / ss_total)

# Variance of Residuals (Polynomial Regression)
residuals_poly = y_test - y_pred_poly
variance_poly = np.sum((residuals_poly - np.mean(residuals_poly)) ** 2) /␣
 ↪len(residuals_poly)

print("\n### Polynomial Regression (Degree=4) Results ###")
print(f"Mean Squared Error: {mse_poly:.4f}")
print(f"R² Score: {r2_poly:.4f}")
print(f"Variance of Residuals: {variance_poly:.4f}")
```

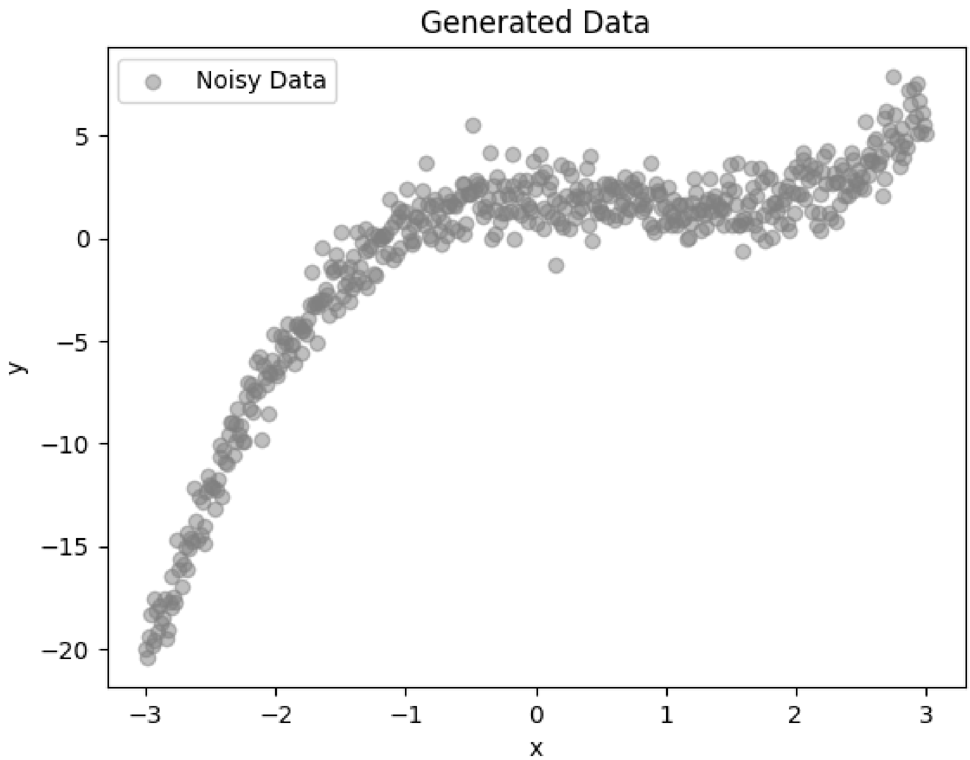
```
Mean Squared Error (Polynomial Degree=2): 0.4643
R² Score (Polynomial Degree=2): 0.6457

### Linear Regression Results ###
Mean Squared Error: 0.5559
R² Score: 0.5758
Variance of Residuals: 0.5559

### Polynomial Regression (Degree=4) Results ###
Mean Squared Error: 15039.7004
R² Score: -11476.1042
Variance of Residuals: 15032.9489
```

Lab 5

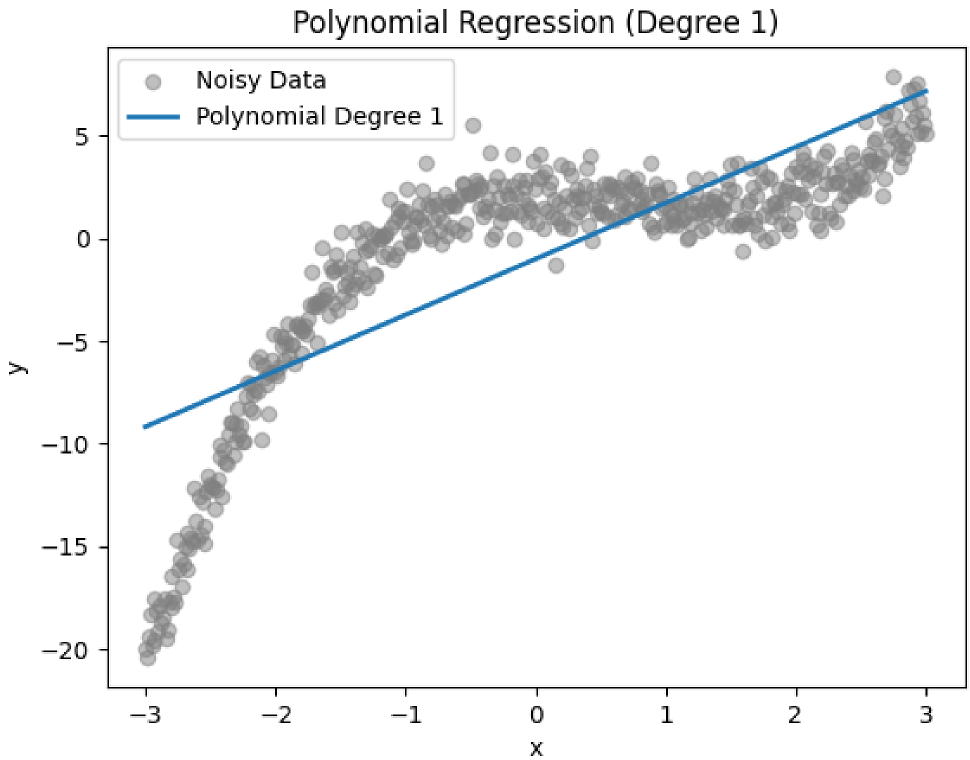
5 Overfitting and Underfitting in Polynomial Regression

[42]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Task 1: Data Generation
np.random.seed(42)
x = np.linspace(-3, 3, 500).reshape(-1, 1)
y_true = 0.5 * x**3 - x**2 + 2
```

```
noise = np.random.normal(0, 1, y_true.shape)
y = y_true + noise

# Visualizing generated data
plt.scatter(x, y, label='Noisy Data', color='gray', alpha=0.5)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Generated Data')
plt.legend()
plt.show()

# Function to fit polynomial regression models
def fit_and_evaluate_polynomial(degree):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(x)
    model = LinearRegression()
    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)

    mse = mean_squared_error(y, y_pred)
    mae = mean_absolute_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    print(f'Polynomial Degree: {degree}')
    print(f'MSE: {mse:.4f}, MAE: {mae:.4f}, R^2: {r2:.4f}\n')

    plt.scatter(x, y, label='Noisy Data', color='gray', alpha=0.5)
    plt.plot(x, y_pred, label=f'Polynomial Degree {degree}', linewidth=2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title(f'Polynomial Regression (Degree {degree})')
    plt.show()
fit_and_evaluate_polynomial(1)
```

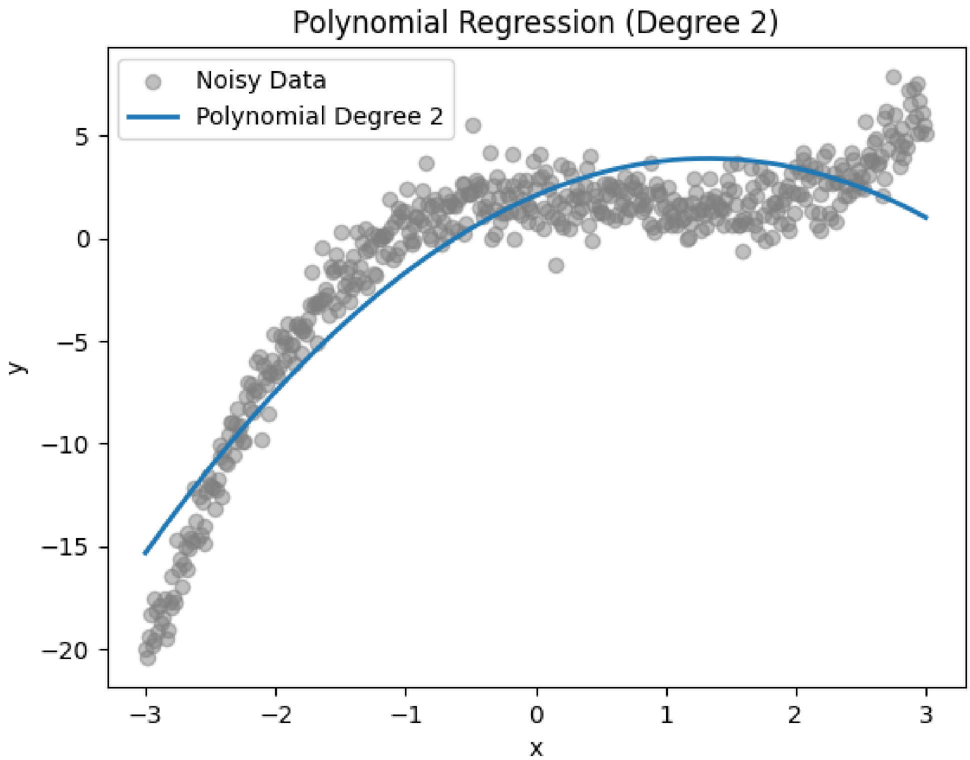



```
Polynomial Degree: 1
MSE: 12.8013, MAE: 2.8570, R^2: 0.6357
```

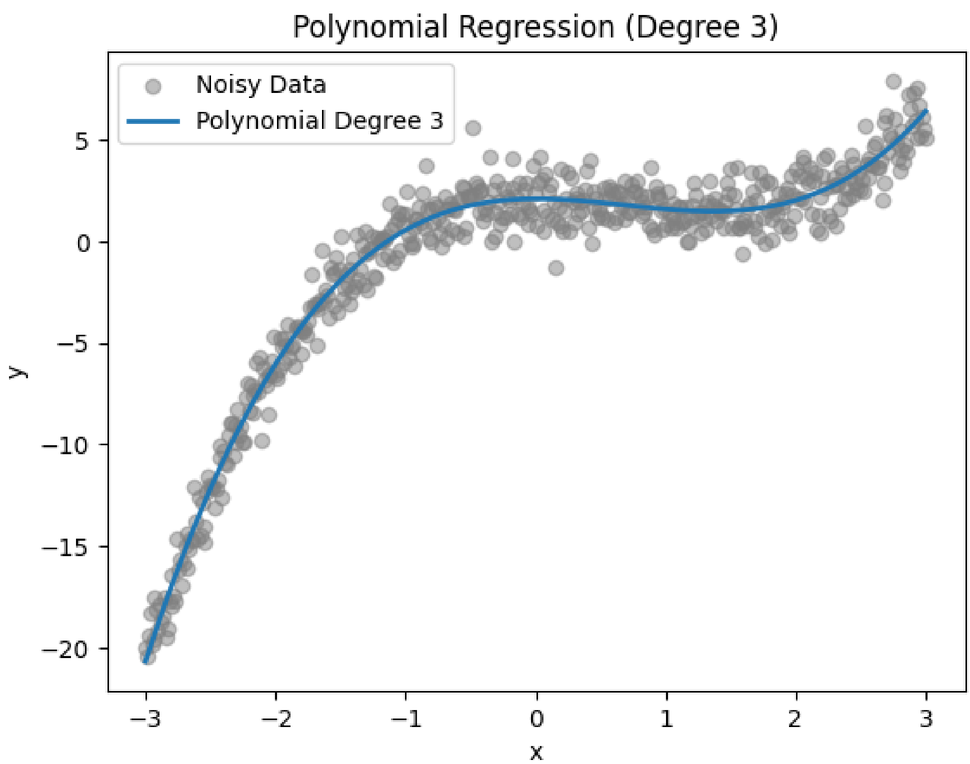


[43]:
```
fit_and_evaluate_polynomial(2)
fit_and_evaluate_polynomial(3)
```

```
Polynomial Degree: 2
MSE: 5.1621, MAE: 1.9004, R^2: 0.8531
```




```
Polynomial Degree: 3
MSE: 0.9556, MAE: 0.7745, R^2: 0.9728
```

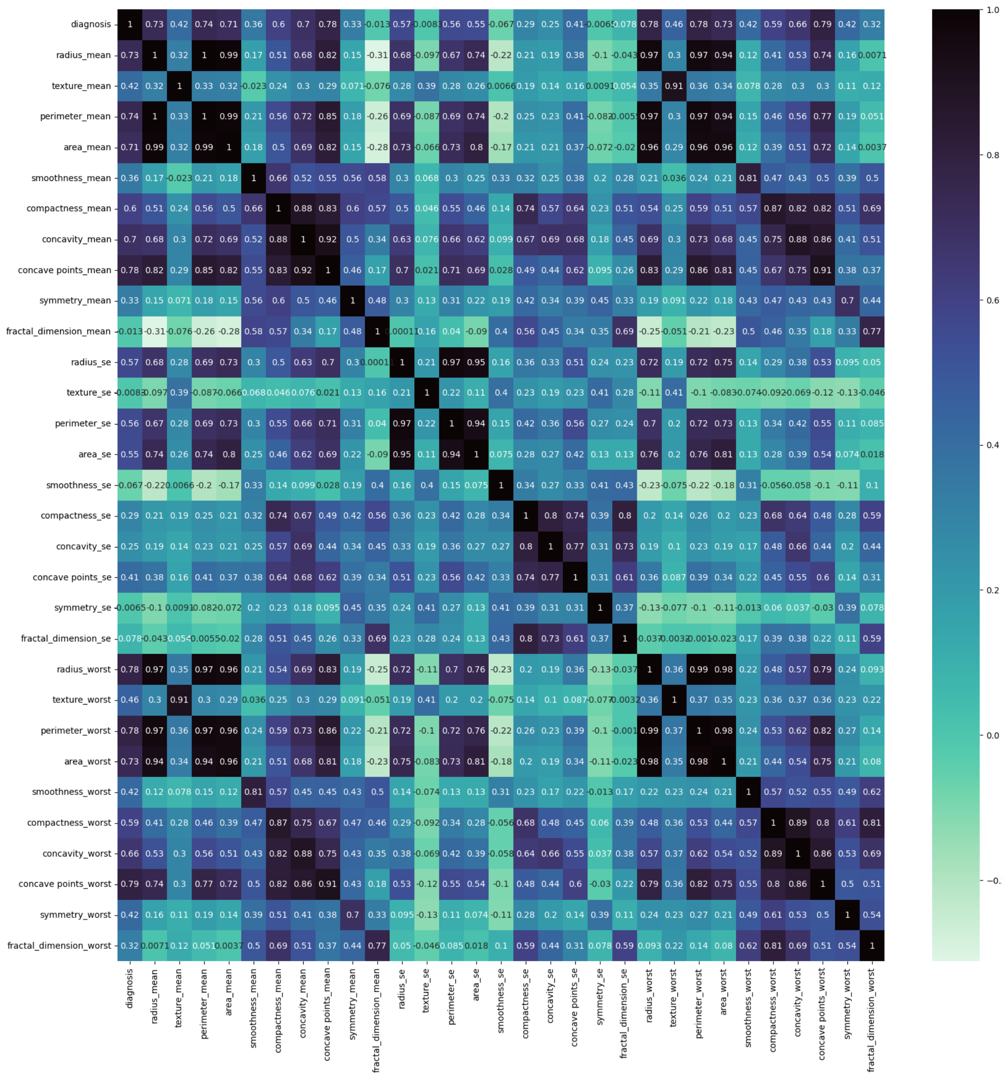


Lab 6

6 Logistic Regression on Breast Cancer Dataset

[53]:
```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

df = pd.read_csv('breast-cancer.csv')
df.columns
df.drop('id', axis=1, inplace=True) #drop redundant columns
df.loc[:,'diagnosis']
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/
↪0
```

```
corr = df.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corr, cmap='mako_r',annot=True)
plt.show()
# Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (thresold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)
X = df[names]
y = df['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) #split the  data into traing and validating
scaler = StandardScaler() #create an instance of standard scaler
scaler.fit(X_train) # fit it to the training data

X_train = scaler.transform(X_train) #transform training data
X_test = scaler.transform(X_test) #transform validation data
model = LogisticRegression() #create logistic regression instance
model.fit(X_train, y_train)  #fit the model instance
predictions = model.predict(X_test) # calculate predictions
accuracy = accuracy_score(y_test, predictions)
print(f'the model accuracy: {accuracy}')
```




```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_se',
'concavity_se', 'concave points_se', 'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst',
'concavity_worst', 'concave points_worst', 'symmetry_worst',
'fractal_dimension_worst']
the model accuracy: 0.9736842105263158
```

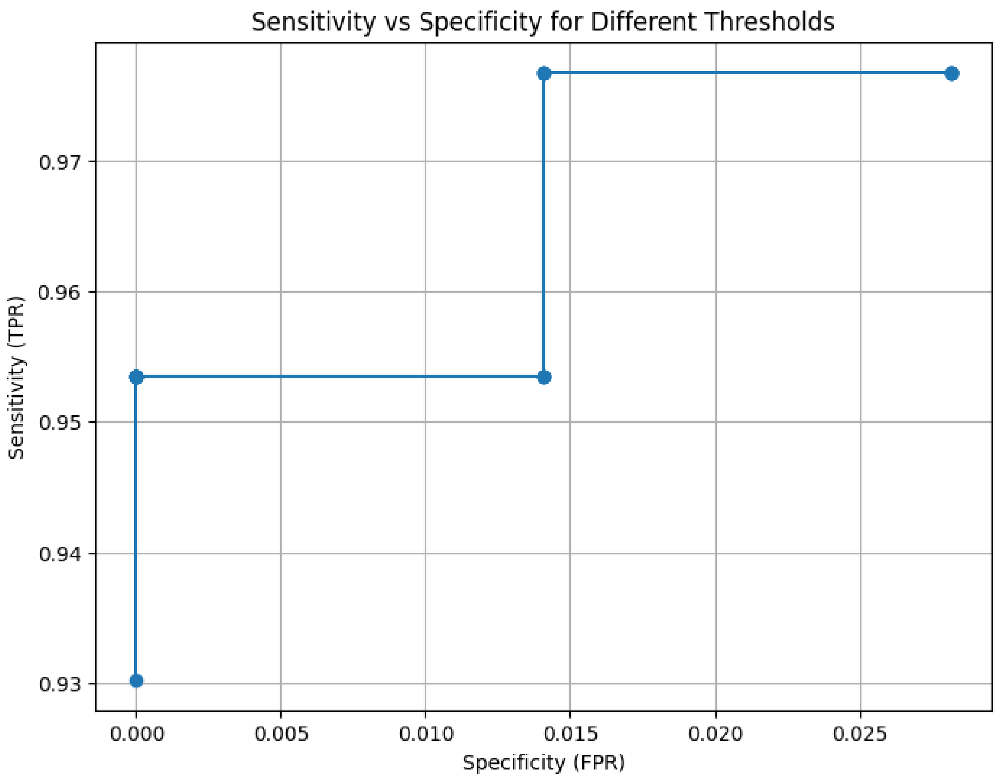
[54]:
```
def evaluate_metrics(y_true, y_pred):
    """
```

```
    Calculate accuracy, precision, recall, sensitivity, and specificity.

    :param y_true: List or array of actual class labels (0 or 1)
    :param y_pred: List or array of predicted class labels (0 or 1)
    :return: Dictionary containing accuracy, precision, recall, sensitivity,␣
↪and specificity
    """

    # Convert Pandas Series to NumPy arrays to use integer indexing if necessary
    y_true = y_true.to_numpy() if isinstance(y_true, pd.Series) else y_true
    y_pred = y_pred.to_numpy() if isinstance(y_pred, pd.Series) else y_pred

    # True Positives (TP): Correctly predicted positive cases
    TP = sum((y_true[i] == 1 and y_pred[i] == 1) for i in range(len(y_true)))

    # True Negatives (TN): Correctly predicted negative cases
    TN = sum((y_true[i] == 0 and y_pred[i] == 0) for i in range(len(y_true)))

    # False Positives (FP): Incorrectly predicted as positive
    FP = sum((y_true[i] == 0 and y_pred[i] == 1) for i in range(len(y_true)))

    # False Negatives (FN): Incorrectly predicted as negative
    FN = sum((y_true[i] == 1 and y_pred[i] == 0) for i in range(len(y_true)))

    # Accuracy
    accuracy = (TP + TN) / (TP + TN + FP + FN) if (TP + TN + FP + FN) != 0 else␣
↪0

    # Precision (Positive Predictive Value)
    precision = TP / (TP + FP) if (TP + FP) != 0 else 0

    # Recall (Sensitivity / True Positive Rate)
    recall = TP / (TP + FN) if (TP + FN) != 0 else 0

    # Sensitivity (Same as Recall)
    sensitivity = recall

    # Specificity (True Negative Rate)
    specificity = TN / (TN + FP) if (TN + FP) != 0 else 0

    print(f'Accuracy: {accuracy}')
    print(f'precision: {precision}')
    print(f' recall : { recall}')
    print(f'sensitivity:{sensitivity}')
    print(f'specificity: {specificity}')

evaluate_metrics(y_test, predictions)
```

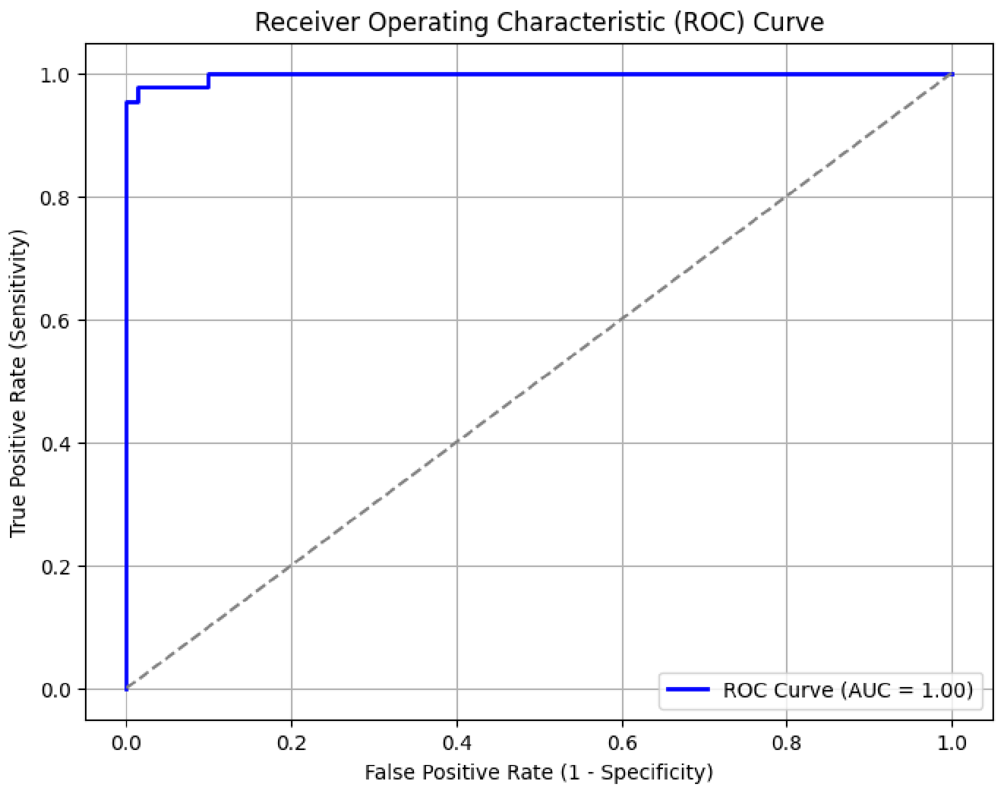
```
Accuracy: 0.9736842105263158
precision: 0.9761904761904762
 recall : 0.9534883720930233
sensitivity:0.9534883720930233
specificity: 0.9859154929577465
```

[55]:

```
from sklearn.metrics import confusion_matrix,roc_curve,auc
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y_true_encoded=le.fit_transform(y_test)
pred=model.predict(X_test)
pred_probs=model.predict_proba(X_test)[:,1]
sens=[]
oneminspec=[]
for t in np.linspace(0.3,0.8,20):
  class_pred=pred_probs>t
  tn,fp,fn,tp=confusion_matrix(y_true_encoded,class_pred).ravel()
  tpr=tp/(tp+fn)
  fpr=fp/(fp+tn)
  sens.append(tpr)
  oneminspec.append(fpr)

plt.figure(figsize=(8, 6))
plt.plot(oneminspec, sens, marker='o', linestyle='-')

# Labels and title
plt.xlabel("Specificity (FPR)")
plt.ylabel("Sensitivity (TPR)")
plt.title("Sensitivity vs Specificity for Different Thresholds")
plt.grid(True)

# Show the plot
plt.show()
```



[56]:
```
# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_true_encoded, pred_probs)

# Compute the AUC (Area Under the Curve)
roc_auc = auc(fpr, tpr)

# Plot the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (random␣
↪model)
plt.xlabel("False Positive Rate (1 - Specificity)")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Lab 7

7 Hyperparameter Tuning of Support Vector Machine (SVM)

[57]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# Load dataset
data = pd.read_csv('heart.csv')
```

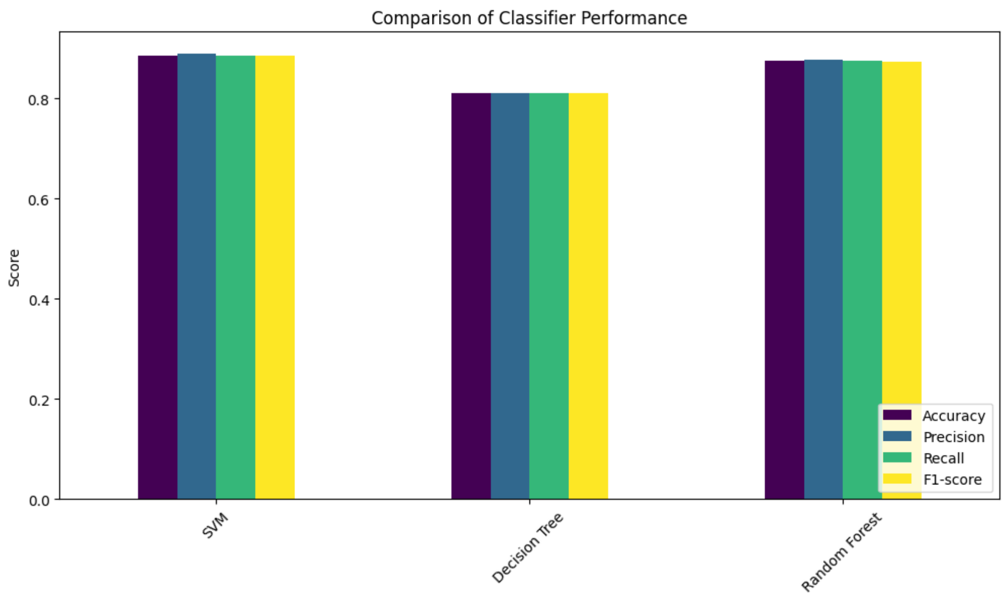
```python
# Separate features and target
X, y = data.drop(columns=['HeartDisease']), data['HeartDisease']

# Identify categorical columns
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',␣
↪'ST_Slope']
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# Standard Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define parameter grids for tuning
svm_params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}

dt_params = {
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10]
}
rf_params = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 8]
}

models = {
    "SVM": (SVC(), svm_params),
    "Decision Tree": (DecisionTreeClassifier(random_state=42), dt_params),
    "Random Forest": (RandomForestClassifier(random_state=42), rf_params),
}
results = {}
for name, (model, params) in models.items():
    print(f"Tuning {name}...")
    grid = GridSearchCV(model, params, cv=5, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train_scaled, y_train)
    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test_scaled)
```

```python
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    results[name] = {
        "Best Params": grid.best_params_,
        "Accuracy": acc,
        "Precision": report["weighted avg"]["precision"],
        "Recall": report["weighted avg"]["recall"],
        "F1-score": report["weighted avg"]["f1-score"]
    }
    print(f"\n===== {name} Best Parameters =====")
    print(grid.best_params_)

    print(f"\n===== {name} Classification Report =====")
    print(classification_report(y_test, y_pred))
df_results = pd.DataFrame(results).T
print("\nOverall Performance Summary:\n", df_results)
df_results.drop(columns=["Best Params"]).plot(kind="bar", figsize=(12, 6),␣
 ↪colormap="viridis")
plt.title("Comparison of Classifier Performance")
plt.xticks(rotation=45)
plt.ylabel("Score")
plt.legend(loc="lower right")
plt.show()
```

```
Tuning SVM…

===== SVM Best Parameters =====
{'C': 1, 'kernel': 'rbf'}

===== SVM Classification Report =====
              precision    recall  f1-score   support

           0       0.92      0.82      0.86        82
           1       0.86      0.94      0.90       102

    accuracy                           0.89       184
   macro avg       0.89      0.88      0.88       184
weighted avg       0.89      0.89      0.88       184

Tuning Decision Tree…

===== Decision Tree Best Parameters =====
{'max_depth': 5, 'min_samples_split': 2}

===== Decision Tree Classification Report =====
              precision    recall  f1-score   support

           0       0.79      0.78      0.79        82
```

```
           1       0.83      0.83      0.83       102

    accuracy                           0.81       184
   macro avg       0.81      0.81      0.81       184
weighted avg       0.81      0.81      0.81       184
```

```
Tuning Random Forest…

===== Random Forest Best Parameters =====
{'max_depth': 5, 'min_samples_split': 8, 'n_estimators': 50}

===== Random Forest Classification Report =====
              precision    recall  f1-score   support

           0       0.90      0.80      0.85        82
           1       0.86      0.93      0.89       102

    accuracy                           0.88       184
   macro avg       0.88      0.87      0.87       184
weighted avg       0.88      0.88      0.87       184


Overall Performance Summary:
                                                      Best Params  Accuracy  \
SVM                                       {'C': 1, 'kernel': 'rbf'}   0.88587
Decision Tree           {'max_depth': 5, 'min_samples_split': 2}  0.809783
Random Forest  {'max_depth': 5, 'min_samples_split': 8, 'n_es…     0.875

              Precision    Recall  F1-score
SVM            0.888459   0.88587  0.884967
Decision Tree  0.809592  0.809783  0.809663
Random Forest   0.87736     0.875  0.874012
```



Lab 8

8 Optimization of a Random Forest Classifier using GridSearchCV

[58]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

# Step 1: Load Dataset
data = load_iris()
X, y = data.data, data.target  # Features and labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Step 4: Perform Grid Search with 5-Fold Cross-Validation
rf_model = RandomForestClassifier(random_state=42)
```

```
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy',␣
↪n_jobs=-1)
grid_search.fit(X_train, y_train)

# Step 5: Get Best Parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Step 6: Train Best Model on Entire Training Data
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

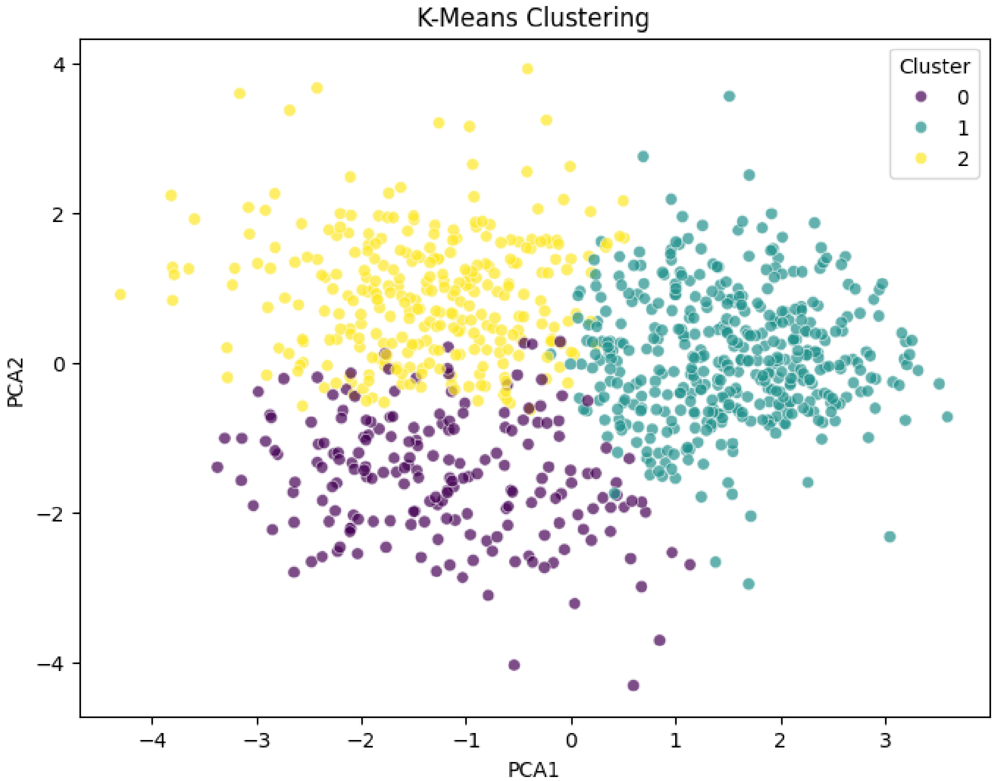
```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 200}
Test Accuracy: 1.0000
```

Lab 9

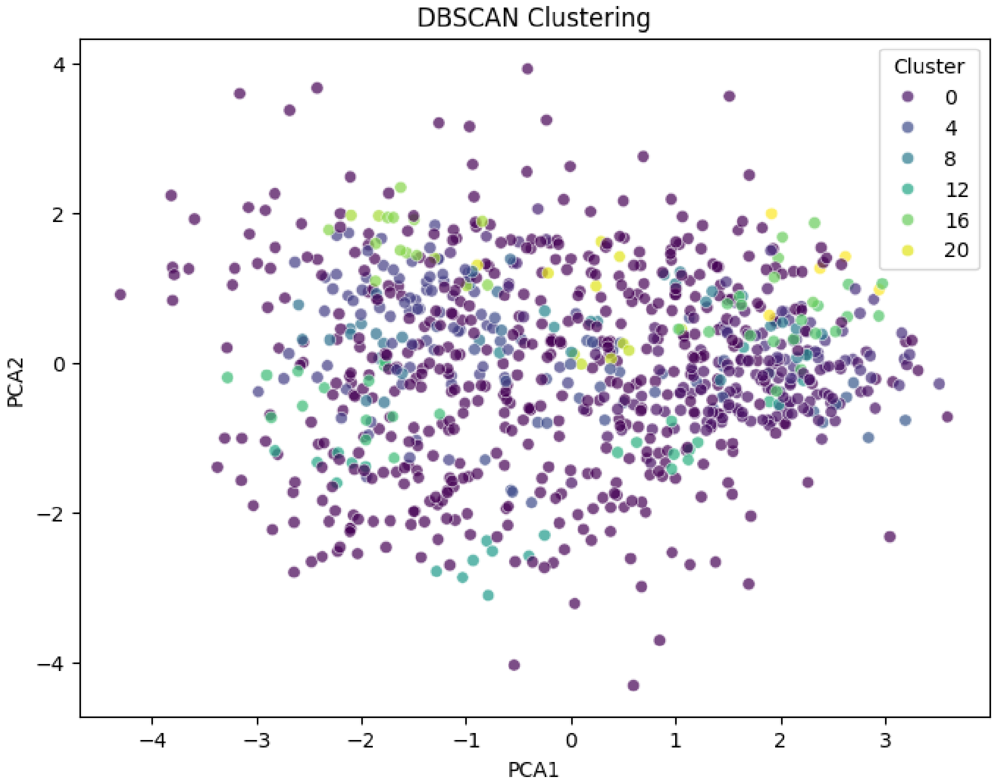
9 Clustering Analysis Using Python

[59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans, DBSCAN, OPTICS
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import linkage, fcluster, dendrogram

# Load dataset
file_path = "heart.csv"
df = pd.read_csv(file_path)

# Encode categorical features
categorical_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina",␣
↪"ST_Slope"]
df_encoded = df.copy()

for col in categorical_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df[col])

# Prepare features
features = df_encoded.drop(columns=["HeartDisease"])
```

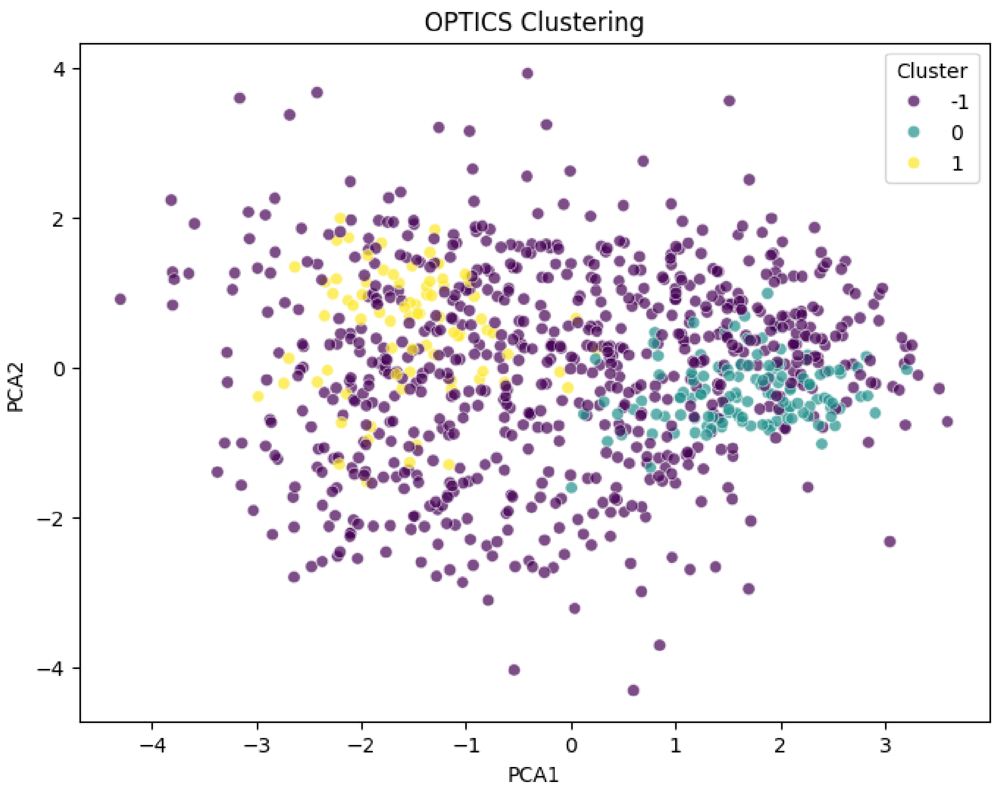
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_encoded["KMeans_Cluster"] = kmeans.fit_predict(scaled_features)

# Hierarchical Clustering
hierarchical_linkage = linkage(scaled_features, method="ward")
df_encoded["Hierarchical_Cluster"] = fcluster(hierarchical_linkage, 3,␣
↪criterion="maxclust")

# DBSCAN Clustering
dbscan = DBSCAN(eps=1.5, min_samples=5)
df_encoded["DBSCAN_Cluster"] = dbscan.fit_predict(scaled_features)

# OPTICS Clustering
optics = OPTICS(min_samples=5, xi=0.05, min_cluster_size=0.05)
df_encoded["OPTICS_Cluster"] = optics.fit_predict(scaled_features)

# Dimensionality Reduction with PCA
pca = PCA(n_components=2)
reduced_features = pca.fit_transform(scaled_features)
df_encoded["PCA1"] = reduced_features[:, 0]
df_encoded["PCA2"] = reduced_features[:, 1]

# Function to Plot Clusters
def plot_clusters(data, cluster_col, title):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x="PCA1", y="PCA2", hue=cluster_col, palette="viridis",␣
↪data=data, alpha=0.7)
    plt.title(title)
    plt.legend(title="Cluster")
    plt.show()

# Plot results
plot_clusters(df_encoded, "KMeans_Cluster", "K-Means Clustering")
plot_clusters(df_encoded, "DBSCAN_Cluster", "DBSCAN Clustering")
plot_clusters(df_encoded, "OPTICS_Cluster", "OPTICS Clustering")

# Plot Hierarchical Clustering Dendrogram
plt.figure(figsize=(10, 5))
plt.title("Hierarchical Clustering Dendrogram")
dendrogram(hierarchical_linkage, truncate_mode="level", p=5)
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
```

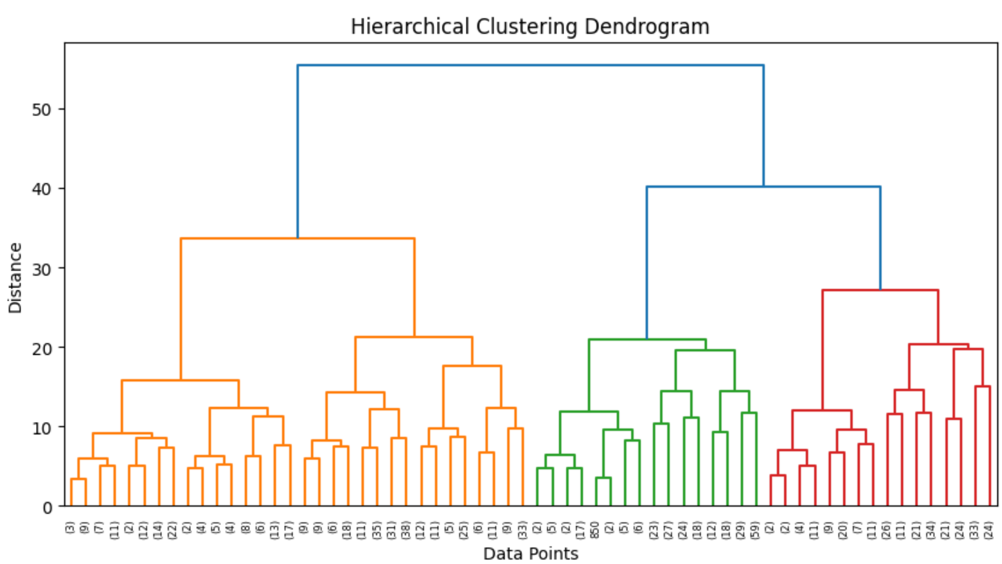


K-Means Clustering
Cluster
0
1
2
PCA2
PCA1



DBSCAN Clustering
Cluster
0
4
8
12
16
20
PCA2
PCA1
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3


OPTICS Clustering
Cluster
-1
0
1
PCA1
PCA2
−4
−3
−2
−1
0
1
2
3
4


Hierarchical Clustering Dendrogram
Distance
Data Points
0
10
20
30
40
50

```
[ ]:
```

```
[ ]:
```