K. SAI VIGNESH
22WU0101044

MACHINE LEARNING LAB MANUAL

EXPERIMENT 1

#EXPERIMENT 1 task 1-1

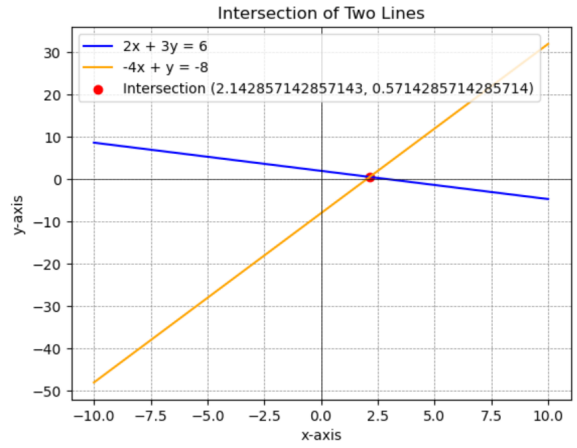
```
import numpy as np
import matplotlib.pyplot as plt
def line1(x):
    return (6 - 2 * x) / 3  # Rearrange 2x + 3y = 6
def line2(x):
    return -8 + 4 * x  # Rearrange -4x + y = -8
A = np.array([[2, 3], [-4, 1]])
B = np.array([6, -8])
det_A = (A[0, 0] * A[1, 1]) - (A[0, 1] * A[1, 0])
adj_A = np.array([[A[1, 1], -A[0, 1]], [-A[1, 0], A[0, 0]]])
inv_A = (1 / det_A) * adj_A
intersection = np.dot(inv_A, B)
x_values = np.linspace(-10, 10, 400)
plt.plot(x_values, line1(x_values), label="2x + 3y = 6", color='blue')
plt.plot(x_values, line2(x_values), label="-4x + y = -8", color='orange')
plt.scatter(intersection[0], intersection[1], color='red', label=f"Intersection
{tuple(intersection)}")
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
plt.legend()
plt.title("Intersection of Two Lines")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()

print(f"The intersection point is: {tuple(intersection)}")
```

#EXPERIMENT 1 task 1-2

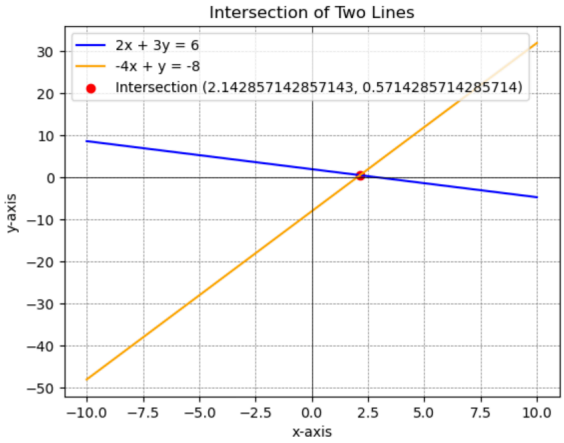
```
import numpy as np
import matplotlib.pyplot as plt

def line1(x):
    return (6 - 2 * x) / 3  # Rearrange 2x + 3y = 6

def line2(x):
    return -8 + 4 * x  # Rearrange -4x + y = -8
A = np.array([[2, 3], [-4, 1]])
B = np.array([6, -8])
intersection = np.linalg.solve(A, B)

x_values = np.linspace(-10, 10, 400)
plt.plot(x_values, line1(x_values), label="2x + 3y = 6", color='blue')
plt.plot(x_values, line2(x_values), label="-4x + y = -8", color='orange')
plt.scatter(intersection[0], intersection[1], color='red', label=f"Intersection
{tuple(intersection)}")
```

MACHINE LEARNING LAB MANUAL

```
# Formatting the plot
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
plt.legend()
plt.title("Intersection of Two Lines")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()

print(f"The intersection point is: {tuple(intersection)}")
```



The intersection point is: (2.142857142857143, 0.5714285714285714)



The intersection point is: (2.142857142857143, 0.5714285714285714)

#EXPERIMENT 1 task -2 1000 points

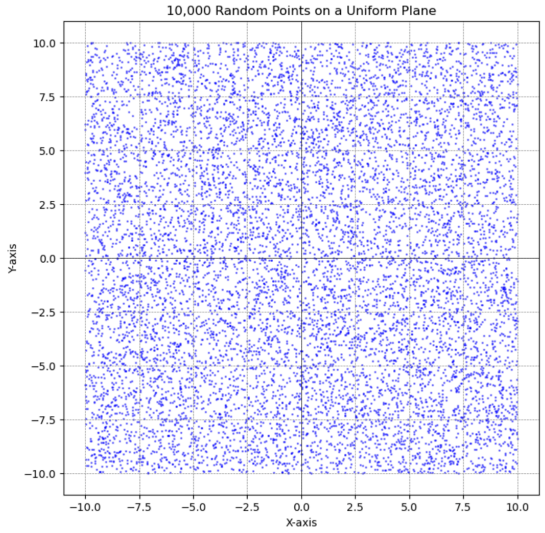
```
import numpy as np
import matplotlib.pyplot as plt

# Define the range for the uniform plane
x_min, x_max = -10, 10
y_min, y_max = -10, 10
x_points = np.random.uniform(x_min, x_max, 10000)
y_points = np.random.uniform(y_min, y_max, 10000)

plt.figure(figsize=(8, 8))
plt.scatter(x_points, y_points, s=1, alpha=0.5, color='blue')
plt.axhline(0, color='black', linewidth=0.5)  # x-axis
plt.axvline(0, color='black', linewidth=0.5)  # y-axis
plt.title("10,000 Random Points on a Uniform Plane")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.show()
```

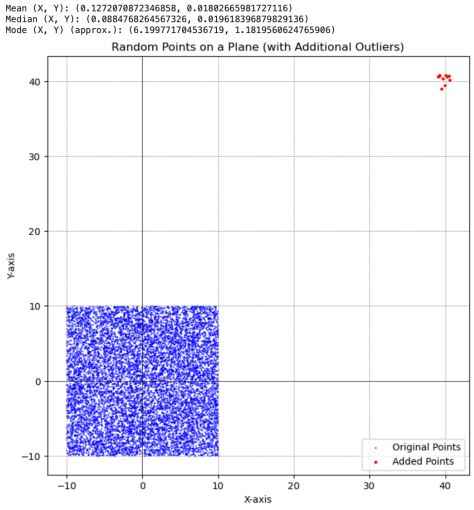
b 1000 points with outliers

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

MACHINE LEARNING LAB MANUAL

```
x_min, x_max = -10, 10
y_min, y_max = -10, 10
x_points = np.random.uniform(x_min, x_max, 10000)
y_points = np.random.uniform(y_min, y_max, 10000)
additional_x = np.random.uniform(39, 41, 10)
additional_y = np.random.uniform(39, 41, 10)
x_points = np.append(x_points, additional_x)
y_points = np.append(y_points, additional_y)

mean_x = np.mean(x_points)
mean_y = np.mean(y_points)
median_x = np.median(x_points)
median_y = np.median(y_points)
bin_count_x, bin_edges_x = np.histogram(x_points, bins=50)
bin_count_y, bin_edges_y = np.histogram(y_points, bins=50)
mode_x = bin_edges_x[np.argmax(bin_count_x)]
mode_y = bin_edges_y[np.argmax(bin_count_y)]

print("Mean (X, Y):", (mean_x, mean_y))
print("Median (X, Y):", (median_x, median_y))
print("Mode (X, Y) (approx.):", (mode_x, mode_y))

plt.figure(figsize=(8, 8))
plt.scatter(x_points, y_points, s=1, alpha=0.5, label='Original Points',
color='blue')
plt.scatter(additional_x, additional_y, s=5, color='red', label='Added Points')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.title("Random Points on a Plane (with Additional Outliers)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.legend()
plt.show()
```





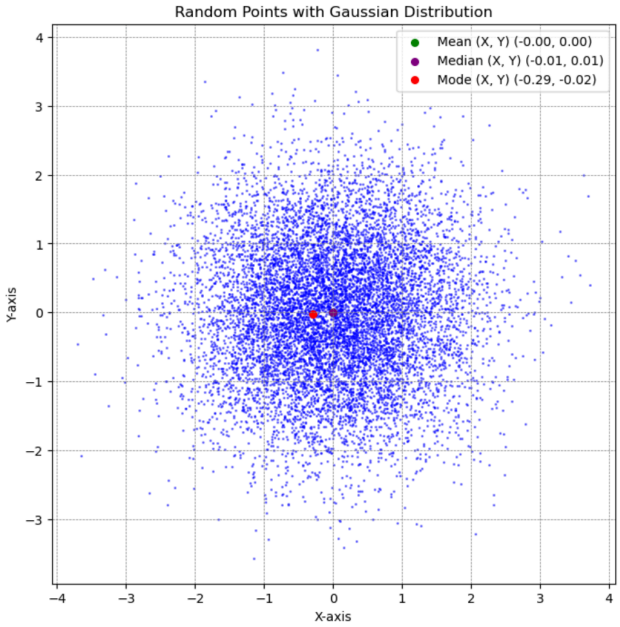
b 1000 points using gausian

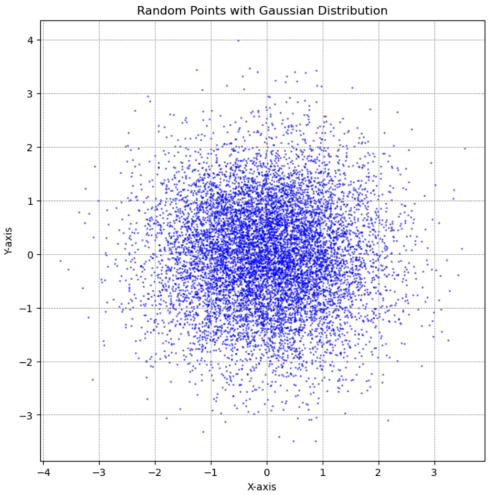
```
import numpy as np
import matplotlib.pyplot as plt
mean = 0
std_dev = 1
x_points = np.random.normal(loc=mean, scale=std_dev, size=10000)
plt.figure(figsize=(8, 8))
plt.scatter(x_points, y_points, s=1, alpha=0.5, color='blue')
plt.title("Random Points with Gaussian Distribution")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.show()
```

b 1000 points using gausian and outliers

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
mean = 0
std_dev = 1
x_points = np.random.normal(loc=mean, scale=std_dev, size=10000)
y_points = np.random.normal(loc=mean, scale=std_dev, size=10000)
mean_y = np.mean(y_points)
median_x = np.median(x_points)
median_y = np.median(y_points)
bin_count_x, bin_edges_x = np.histogram(x_points, bins=50)
bin_count_y, bin_edges_y = np.histogram(y_points, bins=50)
mode_x = bin_edges_x[np.argmax(bin_count_x)]
mode_y = bin_edges_y[np.argmax(bin_count_y)]

# Print the results
print("Mean (X, Y):", (mean_x, mean_y))
print("Median (X, Y):", (median_x, median_y))
print("Mode (X, Y) (approx.):", (mode_x, mode_y))

# Plotting the points
plt.figure(figsize=(8, 8))
plt.scatter(x_points, y_points, s=1, alpha=0.5, color='blue')
plt.scatter(mean_x, mean_y, color='green', s=30, label=f'Mean (X, Y) ({mean_x:.2f},
{mean_y:.2f})')
plt.scatter(median_x, median_y, color='purple', s=30, label=f'Median (X, Y)
({median_x:.2f}, {median_y:.2f})')
plt.scatter(mode_x, mode_y, color='red', s=30, label=f'Mode (X, Y) ({mode_x:.2f},
{mode_y:.2f})')
plt.title("Random Points with Gaussian Distribution")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.legend()
plt.show()
```

```
Mean (X, Y): (-0.002161843129209182, 0.002185062765345363)
Median (X, Y): (-0.005422377391600 38, 0.009776895605524085)
Mode (X, Y) (approx.): (-0.2892352379005141, -0.020802668662864843)
```





EXPERIMENT 2

#EXPERIMENT 2-1 using formula

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

# Given data points
x = np.array([2000, 1800, 2500, 3000, 2200, 2700])
y = np.array([450000, 400000, 550000, 600000, 480000, 700000])

# Calculate means of x and y
x_mean = np.mean(x)
y_mean = np.mean(y)

# Calculate B (slope)
numerator = np.sum((x * y) - (y_mean * x))
denominator = np.sum((x**2) - (x_mean * x))
M = numerator / denominator

# Calculate a (intercept)
C = y_mean - (M * x_mean)

# Print the results
print(f"Slope (M): {M}")
print(f"Intercept (C): {C}")

# Plot the points and the line of best fit
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, C + M * x, color='red', label='Best Fit Line')  # Line of best fit

# Add labels and legend
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression Line')
plt.legend()

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

#EXPERIMENT 2-2,5

```
import numpy as np

# Updated data points
x_updated = np.array([1200, 1500, 2000, 1800, 2500, 3000, 2200, 2700])
y_updated = np.array([300000, 320000, 450000, 400000, 550000, 600000, 480000,
700000])
```

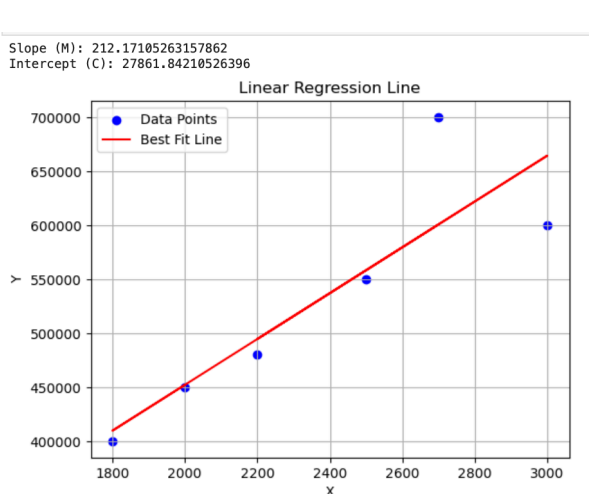
```
# Prepare the design matrix X (including bias term for the intercept)
X_updated = np.vstack((np.ones_like(x_updated), x_updated)).T

# Convert y to a column vector
Y_updated = y_updated.reshape(-1, 1)

# Calculate theta = (X^T X)^-1 X^T Y
theta_updated = np.linalg.inv(X_updated.T @ X_updated) @ X_updated.T @ Y_updated

C_updated, M_updated = theta_updated.flatten()

predicted_price_updated = C_updated + M_updated * 1940

print(f"Slope (M): {M_updated}")
print(f"Intercept (C): {C_updated}")
print(f"Predicted price at 1940 sq. ft with updated data:
${predicted_price_updated}")
```

#EXPERIMENT 2-4

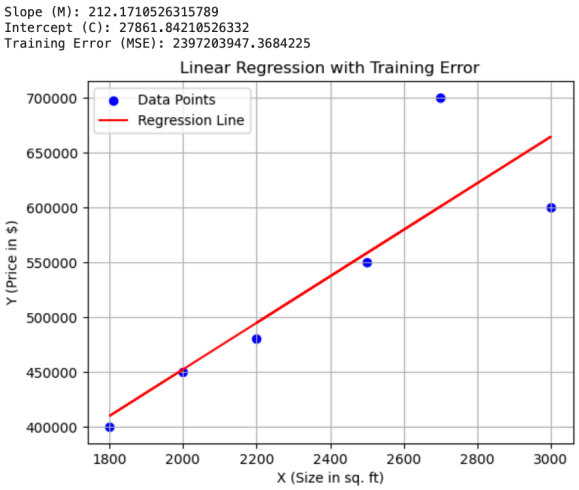
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Given data points
x = np.array([2000, 1800, 2500, 3000, 2200, 2700]).reshape(-1, 1)  # Reshape for
sklearn
y = np.array([450000, 400000, 550000, 600000, 480000, 700000])

# Create and fit the model
model = LinearRegression()
model.fit(x, y)

# Coefficients
slope = model.coef_[0]
intercept = model.intercept_
print(f"Slope (M): {slope}")
print(f"Intercept (C): {intercept}")

# Predictions
y_pred = model.predict(x)

# Calculate Mean Squared Error
mse = mean_squared_error(y, y_pred)
print(f"Training Error (MSE): {mse}")

# Plot the points and the regression line
import matplotlib.pyplot as plt

plt.scatter(x, y, color='blue', label='Data Points')
```

MACHINE LEARNING LAB MANUAL

```
plt.plot(x, y_pred, color='red', label='Regression Line')
plt.xlabel('X (Size in sq. ft)')
plt.ylabel('Y (Price in $)')
plt.title('Linear Regression with Training Error')
plt.legend()
plt.grid()
plt.show()
```



Using formula



Using .fit

```
Slope (M): 212.17057977958774
Intercept (C): 26789.650215621426
Predicted price at 1940 sq. ft with updated data: $438400.57498802163
```

Using closed form solution

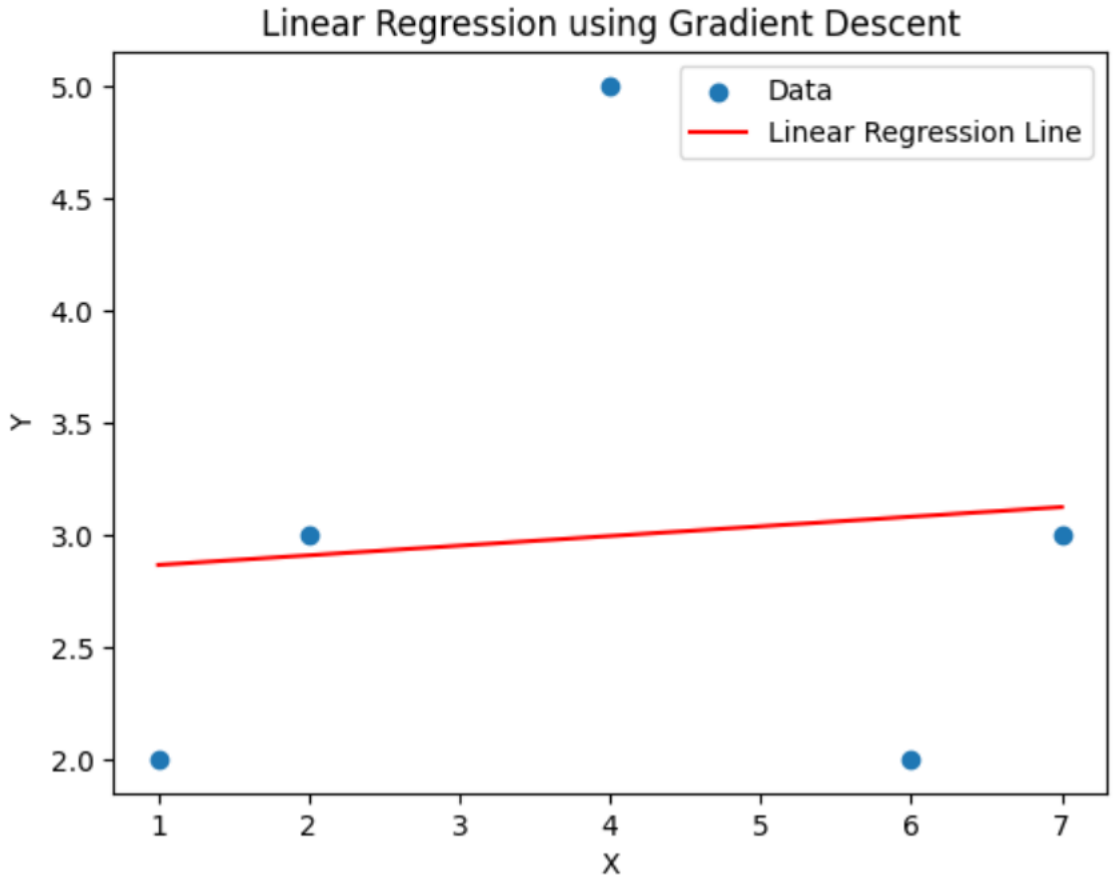
MACHINE LEARNING LAB MANUAL

EXPERIMENT 3

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the data sets
X = np.array([1, 2, 4, 6, 7])
Y = np.array([2, 3, 5, 2, 3])

# Add a column of ones to X for the bias term
X = np.vstack((np.ones(len(X)), X)).T

# Initialize the coefficients (slope and intercept)
theta = np.array([0, 0])

# Define the learning rate and number of iterations
alpha = 0.01
num_iterations = 1000

# Define the cost function (mean squared error)
def cost_function(X, Y, theta):
    predictions = np.dot(X, theta)
    return np.mean((predictions - Y) ** 2)

# Define the gradient descent function
def gradient_descent(X, Y, theta, alpha, num_iterations):
    m = len(Y)
    for _ in range(num_iterations):
        predictions = np.dot(X, theta)
        errors = predictions - Y
        gradient = (2 / m) * np.dot(X.T, errors)
        theta = theta - alpha * gradient
    return theta

# Run the gradient descent algorithm
theta = gradient_descent(X, Y, theta, alpha, num_iterations)

# Print the coefficients (slope and intercept)
print("Coefficients (slope and intercept):", theta)

# Generate predictions
predictions = np.dot(X, theta)

# Visualize the result
plt.scatter(X[:, 1], Y, label="Data")
plt.plot(X[:, 1], predictions, label="Linear Regression Line", color="red")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Linear Regression using Gradient Descent")
plt.legend()
```

```
plt.show()
```

```
Coefficients (slope and intercept): [2.82237136 0.04299948]
```



MACHINE LEARNING LAB MANUAL

EXPERIMENT 4

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

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

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit multiple linear regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predictions
y_pred_linear = linear_model.predict(X_test)

# Evaluate
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print("Multiple Linear Regression Results:")
print(f"MSE: {mse_linear:.4f}")
print(f"R² Score: {r2_linear:.4f}")

# Plot predictions vs actual
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_linear, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Multiple Linear Regression: Actual vs Predicted')
plt.grid(True)
plt.show()
```

4:2

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create polynomial features (degree=2)
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
```

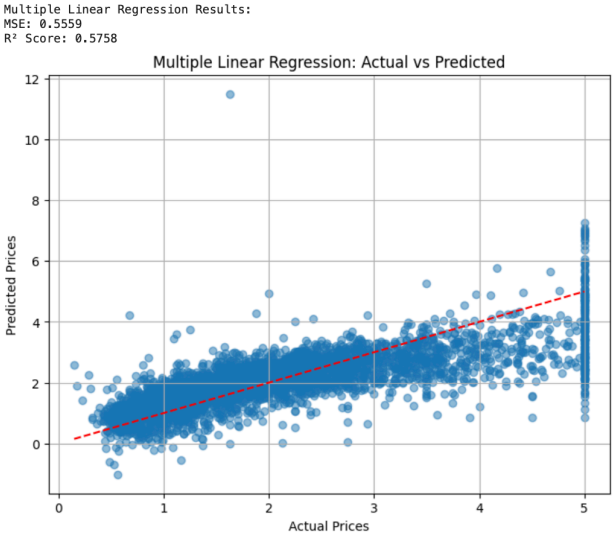
```
X_test_poly = poly_features.transform(X_test)

# Fit polynomial regression
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)

# Predictions
y_pred_poly = poly_model.predict(X_test_poly)

# Evaluate
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print("\nPolynomial Regression (Degree=2) Results:")
print(f"MSE: {mse_poly:.4f}")
print(f"R² Score: {r2_poly:.4f}")

# Plot predictions vs actual
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_poly, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Polynomial Regression: Actual vs Predicted')
plt.grid(True)
plt.show()
```

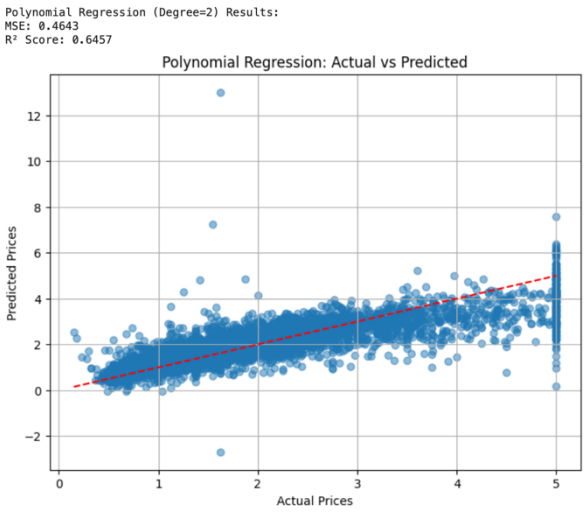
4:3

```
# Task 1: Multiple Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

# Evaluate linear regression
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print("Multiple Linear Regression Results:")
print(f"MSE: {mse_linear:.4f}")
print(f"R^2 Score: {r2_linear:.4f}")

# Task 2: Polynomial Regression (degree=2)
poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_poly_train, y_train)
y_pred_poly = poly_model.predict(X_poly_test)
```

MACHINE LEARNING LAB MANUAL

```python
# Evaluate polynomial regression
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print("\nPolynomial Regression (degree=2) Results:")
print(f"MSE: {mse_poly:.4f}")
print(f"R^2 Score: {r2_poly:.4f}")

# Compare the results
print("\nComparison:")
print(f"Linear Regression -> MSE: {mse_linear:.4f}, R^2: {r2_linear:.4f}")
print(f"Polynomial Regression -> MSE: {mse_poly:.4f}, R^2: {r2_poly:.4f}")
```






```
Multiple Linear Regression Results:
MSE: 0.5559
R^2 Score: 0.5758

Polynomial Regression (degree=2) Results:
MSE: 0.4643
R^2 Score: 0.6457

Comparison:
Linear Regression -> MSE: 0.5559, R^2: 0.5758
Polynomial Regression -> MSE: 0.4643, R^2: 0.6457
```

MACHINE LEARNING LAB MANUAL

EXPERIMENT 5

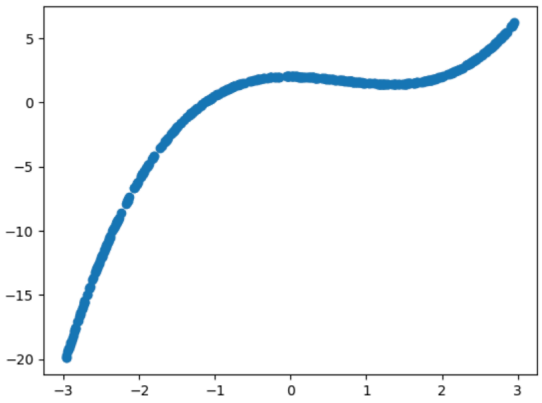
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Task 1: Data Generation
np.random.seed(42)
x = np.random.uniform(-3, 3, 500).reshape(-1,1)
y = 0.5 * x**3 - x**2 + 2 + np.random.normal(0, 1, size=X.shape)
y1 = 0.5 * x**3 - x**2 + 2
```

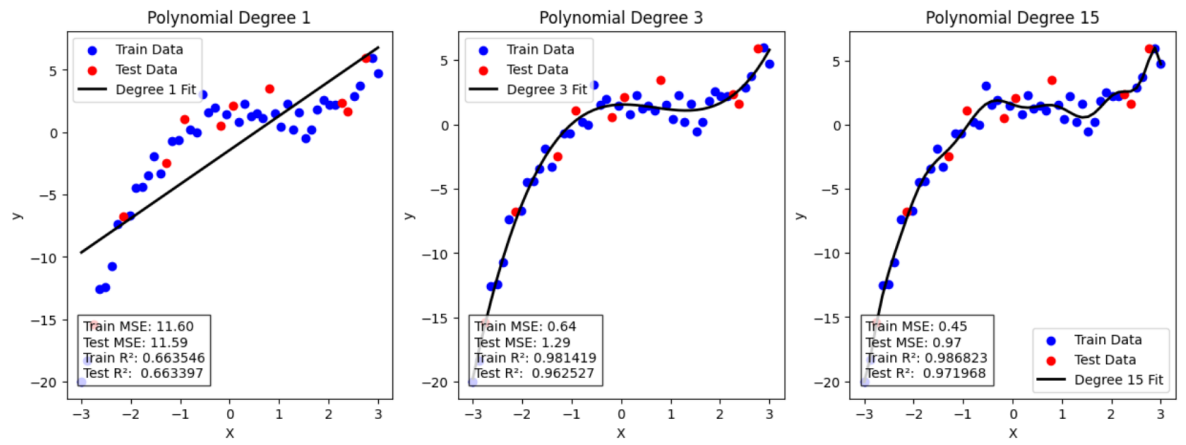
Experiment 5:2

```
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
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=21)
degrees = [1, 3, 15]
models = {}

plt.figure(figsize=(15, 5))
for i, degree in enumerate(degrees):
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X_train,y_train)
    models[degree] = model

    y_pred = model.predict(X)
    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)

    plt.subplot (1, 3, i + 1)
    plt.scatter (X_train, y_train, color="blue", label="Train Data")
```

MACHINE LEARNING LAB MANUAL

```
    plt.scatter (X_test, y_test, color="red", label="Test Data")
    plt.plot(X, y_pred, color="black", linewidth=2, label=f"Degree {degree} Fit")
    plt.xlabel ("X")
    plt.ylabel("y")
    plt.title(f"Polynomial Degree {degree}")
    plt.legend()

# Annotate the figure with MSE and R2
    text = f"Train MSE: {train_mse:.2f}\nTest MSE: {test_mse:.2f}\nTrain R²:
{train_r2:2f}\nTest R²: {test_r2: 2f}"
    plt.text(0.05, 0.05, text, transform=plt.gca().transAxes, fontsize=10,
verticalalignment='bottom', bbox=dict (facecolor='white', alpha=0.8))
plt.show()
```





MACHINE LEARNING LAB MANUAL

EXPERIMENT 6

1,2,3

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
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('/Users/apple/Documents/ml lab/breast-cancer.csv')

df.head()
df.drop('id', axis=1, inplace=True) #drop redundant columns
df.describe().T
px.histogram(data_frame=df, x='diagnosis',
color='diagnosis',color_discrete_sequence=['#05445E','#75E6DA'])
px.histogram(data_frame=df,x='area_mean',color='diagnosis',color_discrete_sequence=
['#05445E','#75E6DA'])
px.histogram(data_frame=df,x='perimeter_mean',color='diagnosis',color_discrete_sequ
ence=['#05445E','#75E6DA'])
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>0.6]

# 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}')
```

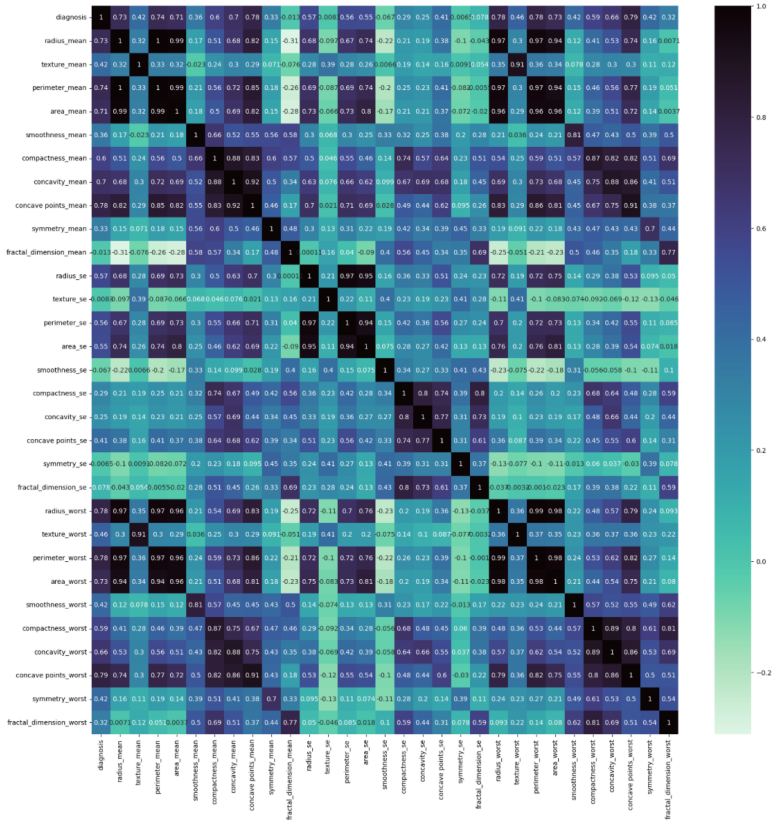
4,5

```
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
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import time  # Import time module

# Load dataset
df = pd.read_csv('/Users/apple/Documents/ml lab/breast-cancer.csv')
df.drop('id', axis=1, inplace=True)  # Drop redundant column
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})  # Convert categorical
target to numerical

# Compute correlation with target
correlation = df.corr()['diagnosis'].abs()

# Define feature selection thresholds
thresholds = [0.3, 0.6, 0]  # Thresholds for feature selection (0 means using all
features)

results = []

for threshold in thresholds:
    if threshold:
        # Select features with correlation greater than the threshold
        selected_features = correlation[correlation > threshold].index.tolist()
        selected_features.remove('diagnosis')  # Remove target variable
    else:
        # Use all features if threshold is 0
        selected_features = df.columns.tolist()
        selected_features.remove('diagnosis')

    # Prepare data
    X = df[selected_features]
```

```
    y = df['diagnosis']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Scale data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Initialize models
    models = {
        'Logistic Regression': LogisticRegression(),
        'SVM': SVC(kernel='linear', C=1.0)  # You can change kernel to 'rbf',
'poly', etc.
    }

    for model_name, model in models.items():
        # Train model and measure training time
        start_time = time.time()  # Record start time
        model.fit(X_train, y_train)  # Train model
        end_time = time.time()  # Record end time
        training_time = end_time - start_time  # Calculate training time

        # Predict on test set
        predictions = model.predict(X_test)
        train_predictions = model.predict(X_train)

        # Compute metrics for test set
        accuracy = accuracy_score(y_test, predictions)
        precision = precision_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        f1 = f1_score(y_test, predictions)
        tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
        specificity = tn / (tn + fp)
        sensitivity = recall  # Recall is same as sensitivity

        # Compute training set accuracy
        train_accuracy = accuracy_score(y_train, train_predictions)

        # Store results
        results.append([
            model_name, threshold, len(selected_features), training_time,
            accuracy, train_accuracy, precision, recall, f1, specificity
        ])

# Convert results to DataFrame
columns = ["Model", "Threshold", "FeaturesUsed", "Training Time (s)", "Test
Accuracy", "Train Accuracy",
           "Precision", "Sensitivity", "F1-score", "Specificity"]
results_df = pd.DataFrame(results, columns=columns)
```

MACHINE LEARNING LAB MANUAL

```
# Display results in table format
print("\n Model Performance Summary ")
print(results_df.to_string(index=False))
```




```
 Model Performance Summary
                Model  Threshold  FeaturesUsed  Training Time (s)  Test Accuracy  Train Accuracy  Precision  Sensitivity  F1-score  Specificity
  Logistic Regression        0.3            23           0.004584       0.973684        0.984615   0.976190     0.953488  0.964706     0.985915
                  SVM        0.3            23           0.002650       0.973684        0.980220   0.976190     0.953488  0.964706     0.985915
  Logistic Regression        0.6            10           0.001789       0.973684        0.960440   0.954545     0.976744  0.965517     0.971831
                  SVM        0.6            10           0.001640       0.982456        0.962637   0.976744     0.976744  0.976744     0.985915
  Logistic Regression        0.0            30           0.001659       0.973684        0.986813   0.976190     0.953488  0.964706     0.985915
                  SVM        0.0            30           0.001479       0.956140        0.986813   0.931818     0.953488  0.942529     0.957746
```

MACHINE LEARNING LAB MANUAL

EXPERIMENT 7

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# Step 2: Preprocess the data (Normalize features)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Split data into train (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Define the hyperparameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'rbf'],  # Kernel type
    'gamma': ['scale', 'auto', 0.01, 0.1]  # Kernel coefficient (for 'rbf')
}

# Step 5: Perform GridSearchCV with cross-validation (cv=5)
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

# Step 6: Evaluate the best model on the test set
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Step 7: Report results
print("\n=== Best Hyperparameters ===")
print(grid_search.best_params_)

print("\n=== Best Model Accuracy (Test Set) ===")
print(f"Accuracy: {accuracy:.4f}")

print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits

=== Best Hyperparameters ===
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}

=== Best Model Accuracy (Test Set) ===
Accuracy: 0.9667

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

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.89      0.94         9
   virginica       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

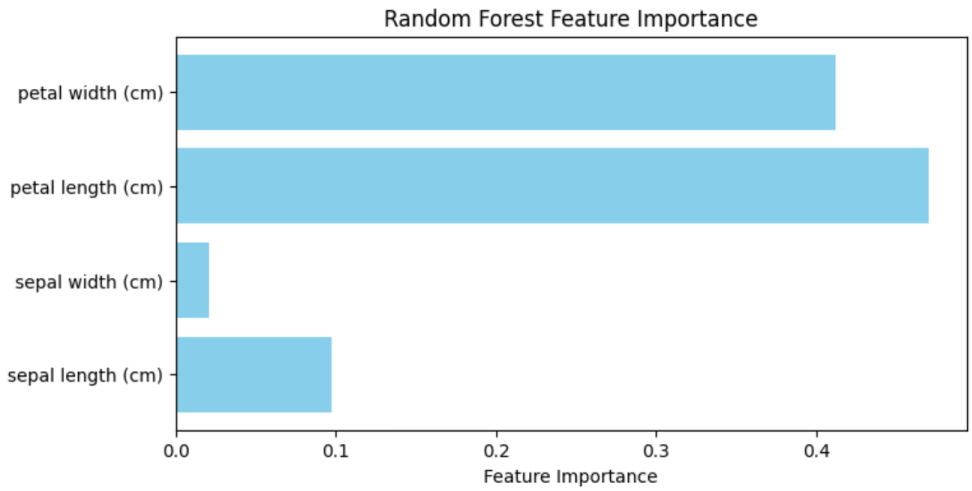
MACHINE LEARNING LAB MANUAL

EXPERIMENT 8

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

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# Step 2: Preprocess the data (if needed)
# (No preprocessing required for Random Forest, but scaling can be tested if
needed)

# Step 3: Split data into train (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 4: Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees
    'max_depth': [None, 10, 20, 30],  # Maximum tree depth
    'min_samples_split': [2, 5, 10],  # Minimum samples to split a node
    'min_samples_leaf': [1, 2, 4]  # Minimum samples per leaf
}

# Step 5: Perform GridSearchCV with cross-validation (cv=5)
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

# Step 6: Evaluate the best model on the test set
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Step 7: Report results
print("\n=== Best Hyperparameters ===")
print(grid_search.best_params_)

print("\n=== Best Model Accuracy (Test Set) ===")
print(f"Accuracy: {accuracy:.4f}")

print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# (Optional) Feature Importance Plot
```

```
import matplotlib.pyplot as plt
importances = best_rf.feature_importances_
features = iris.feature_names
plt.figure(figsize=(8, 4))
plt.barh(features, importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.show()
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

=== Best Hyperparameters ===
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

=== Best Model Accuracy (Test Set) ===
Accuracy: 1.0000

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

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



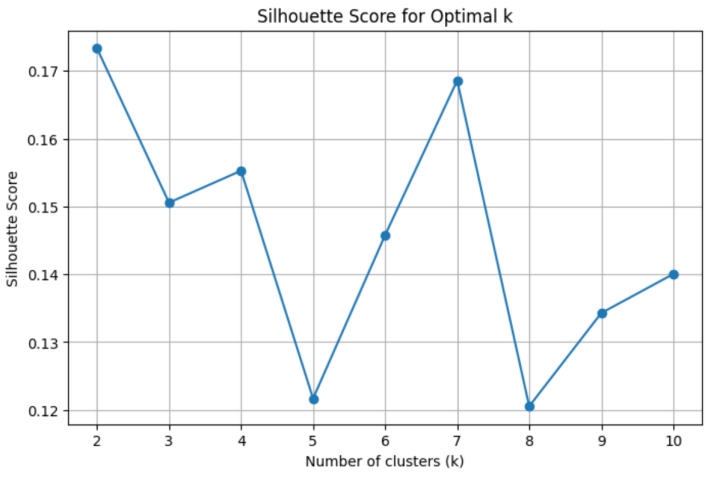
MACHINE LEARNING LAB MANUAL

EXPERIMENT 9

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('heart.csv')

# Display basic info
print(df.info())
print(df.head())

# Preprocessing
# Convert categorical variables to numerical
df = pd.get_dummies(df, columns=['Sex', 'ChestPainType', 'RestingECG',
'ExerciseAngina', 'ST_Slope'])

# Separate features and target
X = df.drop('HeartDisease', axis=1)
y = df['HeartDisease']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Determine optimal number of clusters
silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    score = silhouette_score(X_scaled, kmeans.labels_)
    silhouette_scores.append(score)
    print(f"For k={k}, Silhouette Score={score:.4f}")

# Plot silhouette scores
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Optimal k')
plt.grid(True)
plt.show()

# Choose optimal k (highest silhouette score)
optimal_k = silhouette_scores.index(max(silhouette_scores)) + 2  # +2 because we
started from k=2
print(f"\nOptimal number of clusters: {optimal_k}")
```

```
# Perform K-Means with optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame with the PCA results and cluster assignments
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['Cluster'] = clusters
pca_df['HeartDisease'] = y  # Add the target variable for coloring
# Plot the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Cluster', palette='viridis',
s=100, alpha=0.7)
plt.title(f'K-Means Clustering (k={optimal_k}) with PCA Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

# Optional: Plot with HeartDisease as color to see if clusters align with actual
labels
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='HeartDisease',
palette='coolwarm', s=100, alpha=0.7)
plt.title('PCA Visualization Colored by Heart Disease Status')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

```
For k=2, Silhouette Score=0.1733
For k=3, Silhouette Score=0.1506
For k=4, Silhouette Score=0.1553
For k=5, Silhouette Score=0.1217
For k=6, Silhouette Score=0.1458
For k=7, Silhouette Score=0.1686
For k=8, Silhouette Score=0.1205
For k=9, Silhouette Score=0.1343
For k=10, Silhouette Score=0.1400
```




```
Optimal number of clusters: 2
```

MACHINE LEARNING LAB MANUAL