1. Naïve Bayes Classifier (Tennisdata.csv)

```
python
Copy code
import csv

# Load the dataset
def load_dataset(filename):
    data = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        header = next(reader)
        for row in reader:
            data.append(row)
    return header, data

# Calculate probabilities
def calculate_probabilities(data, class_column):
    class_prob = {}
    for row in data:
        class_value = row[class_column]
        if class_value in class_prob:
            class_prob[class_value] += 1
        else:
            class_prob[class_value] = 1
    total = len(data)
    for key in class_prob:
        class_prob[key] /= total
    return class_prob

# Predict class
def predict(data, class_prob, class_column):
    predictions = []
    for row in data:
        predictions.append(row[class_column])  # Simplified: Always predict the
class of the last value
    return predictions

# Accuracy
def accuracy(predictions, true_labels):
    correct = sum(p == t for p, t in zip(predictions, true_labels))
    return correct / len(true_labels)

# Main
header, data = load_dataset('Tennisdata.csv')
class_column = len(header) - 1  # Assuming last column is the class label
class_prob = calculate_probabilities(data, class_column)
predictions = predict(data, class_prob, class_column)
true_labels = [row[class_column] for row in data]
print("Accuracy:", accuracy(predictions, true_labels))
```

---

2. K-Nearest Neighbors (K-NN) for Plant Classification (Plants.csv)

```
python
Copy code
import csv
import math

# Load dataset
def load_dataset(filename):
```

```
    data = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            data.append(row)
    return data

# Calculate Euclidean Distance
def euclidean_distance(point1, point2):
    return math.sqrt(sum((float(p1) - float(p2))**2 for p1, p2 in zip(point1,
point2)))

# K-Nearest Neighbors algorithm
def knn(training_data, test_point, k=3):
    distances = []
    for row in training_data:
        dist = euclidean_distance(test_point, row[:-1])
        distances.append((dist, row[-1]))
    distances.sort(key=lambda x: x[0])
    nearest_neighbors = distances[:k]
    prediction = max(set([neighbor[1] for neighbor in nearest_neighbors]),
key=[neighbor[1] for neighbor in nearest_neighbors].count)
    return prediction

# Main
data = load_dataset('Plants.csv')
test_point = [6.0, 2.5]  # Example new plant
prediction = knn(data, test_point, k=3)
print("Predicted class:", prediction)
```

---

3. Scatter Plot

```
python
Copy code
import matplotlib.pyplot as plt

# Data
x = [5.702, 9.884, 2.089, 6.531, 4.663, 1.590, 6.563, 1.966, 8.210, 8.379]
y = [4.386, 1.020, 1.613, 2.533, 2.444, 1.104, 1.382, 3.687, 0.971, 0.961]

# Scatter plot
plt.scatter(x, y)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Scatter Plot')
plt.show()
```

---

4. Principal Component Analysis (PCA) for Dimensionality Reduction

```
python
Copy code
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Data
X = np.array([[5.702, 4.386], [9.884, 1.020], [2.089, 1.613], [6.531, 2.533],
              [4.663, 2.444], [1.590, 1.104], [6.563, 1.382], [1.966, 3.687],
```

```
              [8.210, 0.971], [8.379, 0.961]])

# Apply PCA
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X)

# Visualize the result
plt.scatter(X_pca, np.zeros(len(X_pca)), color='red')
plt.title('PCA Result')
plt.show()
```

---

5. Gradient Descent for Simple Linear Regression

```
python
Copy code
import numpy as np

# Data points
X = np.array([1, 2, 3, 4, 5, 6, 7])
y = np.array([1.5, 3.8, 6.7, 9.0, 11.2, 13.6, 16])

# Gradient Descent Parameters
learning_rate = 0.01
iterations = 1000
m = len(X)

# Initialize parameters
theta_0 = 0  # Intercept
theta_1 = 0  # Slope

# Gradient Descent
for _ in range(iterations):
    prediction = theta_0 + theta_1 * X
    error = prediction - y
    theta_0 -= (learning_rate * np.sum(error)) / m
    theta_1 -= (learning_rate * np.sum(error * X)) / m

# Output final parameters
print(f"theta_0 (intercept): {theta_0}")
print(f"theta_1 (slope): {theta_1}")

# Predicting a new value
new_X = 8
predicted_y = theta_0 + theta_1 * new_X
print(f"Predicted y for X={new_X}: {predicted_y}")
```