1. Implement the k-Nearest Neighbor (k-NN) algorithm to classify a given dataset using Minkowski distance for p=3. Evaluate the accuracy of the classifier.
CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"
data = pd.read_csv(file_path)
# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target

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

# k-NN with Minkowski distance (p=3)
knn = KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=3)
knn.fit(X_train, y_train)

# Predictions and accuracy
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) * 100
print(f"Accuracy: {accuracy:.2f}%")
```

If only some selected features have to be included in the code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e0
7d5ae3 (1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"
data = pd.read_csv(file_path)
print(data.columns.tolist())
```

```
# Select specific features (replace with the actual column names or
indices)
selected_features = ['feature_1', 'feature_2', 'feature_3']  #
Replace with your column names
X = data[selected_features]

# Target column
y = data['target']  # Replace with the actual target column name

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

# k-NN with Minkowski distance (p=3)
knn = KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=3)
knn.fit(X_train, y_train)

# Predictions and accuracy
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) * 100
print(f"Accuracy: {accuracy:.2f}%")
```

2. Implement the Iterative Dichotomiser (ID3) algorithm with entropy as the criterion to build a decision tree using a given dataset. Evaluate the classifier by computing its accuracy.

CODE:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)

# Prepare features and target
```

```
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Decision Tree with entropy
dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train, y_train)

# Evaluate
y_pred = dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree (Entropy) Accuracy: {accuracy:.2f}")
```

3.Implement feature reduction using Principal Component Analysis by at least one dimension for a given dataset. Evaluate the performance of Logistic regression before and after applying PCA.

CODE:

```
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)

# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression before PCA
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
y_pred = lr.predict(X_test)
accuracy_before = accuracy_score(y_test, y_pred)

# PCA
pca = PCA(n_components=X_train.shape[1] - 1)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Logistic Regression after PCA
lr_pca = LogisticRegression()
lr_pca.fit(X_train_pca, y_train)
y_pred_pca = lr_pca.predict(X_test_pca)
accuracy_after = accuracy_score(y_test, y_pred_pca)

print(f"Logistic Regression Accuracy Before PCA: {accuracy_before:.2f}")
print(f"Logistic Regression Accuracy After PCA: {accuracy_after:.2f}")
```

4. Implement the na¨ive Bayesian classifier for a given data set. Compute the accuracy of the classifier,considering few test data. Sets.

CODE:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import pandas as pd

file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)

# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)
```

```
# Evaluate
y_pred = nb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Naive Bayes Accuracy: {accuracy:.2f}")
```

5. Implement Support Vector Machine (SVM) model for a given data for Kernels :'linear', 'poly', 'rbf','sigmoid'. Plot the support vectors and with regions of classes in each case. Evaluate their performance on the test data and suggest the best fitting kernel.

CODE:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import pandas as pd

file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target


# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"SVM Accuracy with {kernel} kernel: {accuracy:.2f}")

    # Optional: Plot decision regions if dataset is 2D
    if X_train.shape[1] == 2:
        plt.figure()
        plt.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=y_train, cmap='viridis', s=50, alpha=0.5)
```

```
    plt.title(f"SVM Decision Regions ({kernel})")
    plt.show()
```

6. Implement Rosenblatt's perceptron model for the Boolean expression ((p ∧ q) ∨ r) → (p∧ ~ r) and evaluate its accuracy.

CODE

```
import numpy as np
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

# Define the truth table for the Boolean expression ((p ∧ q) ∨ r) → (p ∧ ~r)
data = []
for p in [0, 1]:
    for q in [0, 1]:
        for r in [0, 1]:
            # Calculate ((p ∧ q) ∨ r) and (p ∧ ~r)
            left = (p and q) or r
            right = p and not r
            result = int(left <= right)  # Implication
            data.append([p, q, r, result])

data = np.array(data)

# Input (X) and target (y)
X = data[:, :-1]
y = data[:, -1]

# Train Rosenblatt's Perceptron
perceptron = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perceptron.fit(X, y)

# Evaluate
y_pred = perceptron.predict(X)
accuracy = accuracy_score(y, y_pred)
print(f"Rosenblatt's Perceptron Accuracy: {accuracy:.2f}")
```

7. Implement polynomial regression using Stochastic Gradient Descent for the given dataset, plot the accuracy for different degrees and conclude the best fit.

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Specify the file path for the Iris dataset (CSV file)
file_path = r"C:\Users\tanuj\OneDrive\Desktop\iris.csv"  # Replace with the actual path to your
CSV file

# Load dataset
data = pd.read_csv(file_path)

# Select features and target columns
# Assuming Sepal Length (1st column) as X and Petal Length (3rd column) as y
X = data.iloc[:, 0].values.reshape(-1, 1)  # Feature
y = data.iloc[:, 2].values.reshape(-1, 1)  # Target

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

# Evaluate polynomial regression for different degrees
degrees = range(1, 10)
train_errors = []
test_errors = []

for degree in degrees:
    # Polynomial feature transformation
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # SGD Regressor
    sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
    sgd_reg.fit(X_train_poly, y_train.ravel())

    # Predictions and errors
    y_train_pred = sgd_reg.predict(X_train_poly)
    y_test_pred = sgd_reg.predict(X_test_poly)
    train_errors.append(mean_squared_error(y_train, y_train_pred))
    test_errors.append(mean_squared_error(y_test, y_test_pred))
```

```
# Plot the errors for different polynomial degrees
plt.figure(figsize=(10, 6))
plt.plot(degrees, train_errors, label="Train Error", marker='o')
plt.plot(degrees, test_errors, label="Test Error", marker='o')
plt.xlabel("Polynomial Degree")
plt.ylabel("Mean Squared Error")
plt.title("Polynomial Degree vs. Error")
plt.legend()
plt.grid(True)
plt.show()

# Best degree conclusion
best_degree = degrees[np.argmin(test_errors)]
print(f"The best polynomial degree is {best_degree} with test error {min(test_errors):.2f}.")
```

8.Implement multiple linear regression using Stochastic Gradient Descent for the given dataset and compute the accuracy on the test data.

CODE:

```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)


# Split dataset 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)

# Train Multiple Linear Regression using SGD
sgd = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
sgd.fit(X_train, y_train)

# Predict on test data
y_pred = sgd.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Output results
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R2) Score: {r2:.4f}")
```

9. Implement the Iterative Dichotomiser (ID3) algorithm with Gini as the criterion to build a decision tree for a given dataset. Evaluate the classifier by computing its accuracy.

CODE:

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt

file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
# Split dataset 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)

# Train Decision Tree using Gini criterion
dt = DecisionTreeClassifier(criterion='gini', random_state=42)
dt.fit(X_train, y_train)

# Predict on test data
y_pred = dt.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree (Gini) Accuracy: {accuracy:.4f}")

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dt, feature_names=X.columns, class_names=dt.classes_.astype(str), filled=True,
rounded=True)
plt.title("Decision Tree Visualization (Gini)")
plt.show()
```

10. Implement the ADALINE model using the Delta Rule for binary classification for a given dataset.Evaluate the classifier by computing its accuracy.

CODE:

```
import numpy as np
import pandas as pd  # Import pandas for reading datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

class Adaline:
    def __init__(self, lr=0.01, epochs=100):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, y):
        self.weights = np.zeros(X.shape[1])
        self.bias = 0
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                y_pred = np.dot(xi, self.weights) + self.bias  # Linear output
                error = target - y_pred
                self.weights += self.lr * error * xi  # Weight update using Delta Rule
                self.bias += self.lr * error  # Bias update

    def predict(self, X):
        # Apply the sign function for binary classification
        return (np.dot(X, self.weights) + self.bias >= 0).astype(int)

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

```
# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Ensure the target is binary (convert if necessary)
#y = (y == some_condition).astype(int)  # Replace `some_condition` with appropriate logic if needed

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the ADALINE model
model = Adaline(lr=0.01, epochs=1000)
model.fit(X_train, y_train)

# Evaluate the model
accuracy = np.mean(model.predict(X_test) == y_test)
print("Accuracy:", accuracy)
```

11. Implement ADALINE for a regression task using sigmoid activation and delta rule, where the model predicts continuous values for a given dataset. Compute the mean squared error on the test data.

CODE:

```
#11
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
```

```
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
# Split dataset into training and testing sets
# Split dataset into training and testing sets
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train ADALINE model using SGD (Delta Rule for regression)
adaline = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
adaline.fit(X_train, y_train)

# Predict using the trained model
y_pred_linear = adaline.predict(X_test)

# Apply sigmoid activation to predictions
y_pred_sigmoid = sigmoid(y_pred_linear)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred_sigmoid)
print(f"ADALINE Regression MSE (with Sigmoid Activation): {mse:.4f}")
```

12. Implement the MADALINE model for a regression task, where multiple ADALINE neurons are used in the hidden layer to predict continuous values with ReLU activations. Compute the mean squared error on the test data.

CODE:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np

# ReLU activation function
```

```
def relu(x):
    return np.maximum(0, x)

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)


# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# First Layer: Hidden layer with ReLU
hidden_layer_weights = np.random.randn(X_train.shape[1], 64)  # Random weights
hidden_layer_bias = np.random.randn(64)
X_train_hidden = relu(np.dot(X_train, hidden_layer_weights) + hidden_layer_bias)
X_test_hidden = relu(np.dot(X_test, hidden_layer_weights) + hidden_layer_bias)

# Output Layer: Linear Regression
regressor = LinearRegression()
regressor.fit(X_train_hidden, y_train)

# Predict and evaluate
y_pred = regressor.predict(X_test_hidden)
mse = mean_squared_error(y_test, y_pred)
print(f"MADALINE Regression MSE (Alternative Approach): {mse:.4f}")
```

13. Implement the MADALINE model for a binary classification task, where multiple ADALINE neurons are used in the hidden layers with ReLU activations. Compute the accuracy on the test data.

CODE

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np

# ReLU activation function
def relu(x):
    return np.maximum(0, x)

# Load dataset
file_path =
r"C:\Users\jslam\Downloads\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3
(1)\8836201-6f9306ad21398ea43cba4f7d537619d0e07d5ae3\iris.csv"  # Replace with your
dataset file path
data = pd.read_csv(file_path)



# Prepare features and target
X = data.iloc[:, :-1]  # Assuming last column is the target
y = data.iloc[:, -1]   # Assuming last column is the target
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
# Split dataset 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)

# First Layer: Hidden layer with ReLU activation
hidden_layer_weights = np.random.randn(X_train.shape[1], 64)  # Random weights
hidden_layer_bias = np.random.randn(64)
X_train_hidden = relu(np.dot(X_train, hidden_layer_weights) + hidden_layer_bias)
X_test_hidden = relu(np.dot(X_test, hidden_layer_weights) + hidden_layer_bias)

# Scale the hidden layer output
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_hidden)
X_test_scaled = scaler.transform(X_test_hidden)

# Logistic Regression: Output layer
classifier = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=42)  # Increased
max_iter
classifier.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = classifier.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nMADALINE Classification Accuracy: {accuracy:.4f}")
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

14.Apply Linear Discriminant Analysis (LDA) for feature reduction to improve classification performance of Logistic regression on a given dataset.

CODE:

```
 import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load dataset from a user-defined file path
file_path = r"C:\Users\tanuj\OneDrive\Desktop\iris.csv"  # Replace with the actual path to your CSV file
data = pd.read_csv(file_path)

# Extract features (X) and target (y)
X = data.iloc[:, :-1].values  # Features (all columns except the last)
y = data.iloc[:, -1].values   # Target (last column)

# Encode target variable if it is categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

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

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Apply Linear Discriminant Analysis (LDA)
lda = LDA(n_components=None)  # Retain all components initially
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
```

```
# Check the explained variance ratio to determine retained components
explained_variance_ratio = lda.explained_variance_ratio_
print("Explained Variance Ratio for each LDA component:", explained_variance_ratio)

# Train Logistic Regression on reduced features
log_reg = LogisticRegression()
log_reg.fit(X_train_lda, y_train)

# Predict and evaluate
y_train_pred = log_reg.predict(X_train_lda)
y_test_pred = log_reg.predict(X_test_lda)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
print("\nClassification Report (Testing):")
print(classification_report(y_test, y_test_pred))
```

15.15. Implement the K-means clustering algorithm for a given dataset. Plot the performance graph Inertia vs k.

CODE:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic dataset
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0, random_state=42)

# Compute inertia for k = 1 to 10
inertias = [KMeans(n_clusters=k, random_state=42).fit(X).inertia_ for k in range(1, 11)]

# Plot Inertia vs k
plt.plot(range(1, 11), inertias, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.title("K-means: Inertia vs k")
plt.show()
```

16.

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
clustering = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
y_pred = clustering.fit_predict(X_scaled)
plt.figure(figsize=(10, 7))
sch.dendrogram(sch.linkage(X_scaled, method='ward'))
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
sil_score = silhouette_score(X_scaled, y_pred)
print(f"Silhouette Score: {sil_score:.2f}")
```