Assignment 1

Name: Pasupulety Chethan Krishna Venkat

Roll Number: 21CS30036

```python
# import all the necessary libraries here
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

data_df = 
pd.read_excel('../../dataset/logistic-regression/Pumpkin_Seeds_Dataset
.xlsx')


# Applying logistic regression on the dataset

# Split the dataset into training, validation, and test sets
train_data_df, temp_data_df = train_test_split(data_df, test_size=0.5, 
random_state=42)
validation_data_df, test_data_df = train_test_split(temp_data_df, 
test_size=0.4, random_state=42)



# Convert DataFrame to numpy arrays
train_data = train_data_df.to_numpy()
validation_data = validation_data_df.to_numpy()
test_data = test_data_df.to_numpy()

# Split the data into X and y
X_train = train_data[:, :-1]
y_train = train_data[:, -1]

X_validation = validation_data[:, :-1]
y_validation = validation_data[:, -1]

X_test = test_data[:, :-1]
y_test = test_data[:, -1]



# normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_validation = scaler.transform(X_validation)
X_test = scaler.transform(X_test)
```

```
# Add a column of ones to X_train, X_validation, and X_test
X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
X_validation = np.hstack((np.ones((X_validation.shape[0], 1)),
X_validation))
X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))

# Changing data in Y_train, Y_validation, and Y_test to 0 and 1
# 0 for Çerçevelik and 1 for Kabakçık
y_train = np.where(y_train == 'Çerçevelik', 0, 1)
y_validation = np.where(y_validation == 'Çerçevelik', 0, 1)
y_test = np.where(y_test == 'Çerçevelik', 0, 1)

# Initialize theta to zeros
theta = np.zeros(X_train.shape[1])
```

Preprocessing of data done. Moving on to the calculations

```
# define the sigmoid function
# returns matrix of the same shape as z with values between 0 and 1
def sigmoid(z):
    return 1 / (1 + np.exp(-z))



# Define the gradient descent function for logistic regression
def gradient_descent(X_train, y_train, theta, alpha, iterations):

    # m= number of training examples
    m = X_train.shape[0]
    for i in range(iterations):

        # calculating the gradient term as an (n+1 X 1) matrix
        # gradient matrix= (X^T)*(h(X*theta)-Y)

        # h(X*theta) is a (m X 1) matrix
        h = sigmoid(np.dot(X_train, theta))

        # loss matrix= (h(X*theta)-Y) is a (m X 1) matrix
        loss = h-y_train

        # gradient matrix= (X^T)*(h(X*theta)-Y) is a (n+1 X 1) matrix
        gradient = np.dot(X_train.T, loss)

        # updating theta
        theta = theta - (alpha / m) * gradient

    return theta
```

```
# Predict the labels for the test set using the optimal theta
def predict(X, theta):
    # if the value of sigmoid function is greater than 0.5, then the label is 1
    # if the value of sigmoid function is less than 0.5, then the label is 0
    return np.round(sigmoid(np.dot(X,theta)))
```

Training the model

```

# setting hyperparameters
alpha = 0.1
iterations = 1000


# Run gradient descent algorithm to get the optimal theta
theta = gradient_descent(X_train, y_train, theta,alpha,iterations)



# Predict the labels for the test set using the optimal theta
y_pred= predict(X_test, theta)


# Defining accuracy_score function
def accuracy_score(y_test, y_pred):

    # calculating the number of true positives, true negatives, false positives, and false negatives
    return np.sum(y_test == y_pred) / len(y_test)

# Defining presicion_score function
def presicion_score(y_test, y_pred):

    # calculating the number of true positives and false positives

    TP = np.sum((y_test == 1) & (y_pred == 1))
    FP = np.sum((y_test == 0) & (y_pred == 1))
    return TP / (TP + FP)

# Defining recall_score function
def recall_score(y_test, y_pred):
    # calculating the number of true positives and false negatives
    TP = np.sum((y_test == 1) & (y_pred == 1))
    FN = np.sum((y_test == 1) & (y_pred == 0))
    return TP / (TP + FN)
```

```python
# Print the accuracy, precision, and recall
print('Logistic regression model:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', presicion_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
```

```
Logistic regression model:
Accuracy: 0.87
Precision: 0.8632478632478633
Recall: 0.8595744680851064
```