Assignment - 3

Name: Pola Gnana Shekar

Roll No: 21CS10052

```
import pandas as pd
import numpy as np

# Load the dataset using Pandas
data = pd.read_csv("./housing.csv")

# Display the first few rows of the dataset to get an overview
print(data.head())

# Check for missing values (if any)
print("\n\nMissing Values:")
print(data.isnull().sum())
```

```
      RM  LSTAT  PTRATIO      MEDV
0  6.575   4.98     15.3  504000.0
1  6.421   9.14     17.8  453600.0
2  7.185   4.03     17.8  728700.0
3  6.998   2.94     18.7  701400.0
4  7.147   5.33     18.7  760200.0


Missing Values:
RM         0
LSTAT      0
PTRATIO    0
MEDV       0
dtype: int64
```

```
data = np.array(data)
m, n= data.shape
np.random.shuffle(data)

#split the data set
data_train=data[0:400].T
X_train=data_train[0:n-1].T
Y_train=data_train[n-1].T

data_test=data[400:m].T
X_test=data_test[0:n-1].T
Y_test=data_test[n-1].T

print("Training data set shapes:")
```

```python
print("X_train shape:",X_train.shape)
print("Y_train shape:",Y_train.shape)

print("Test data set shapes:")
print("X_test shape:",X_test.shape)
print("Y_test shape:",Y_test.shape)
```

```
Training data set shapes:
X_train shape: (400, 3)
Y_train shape: (400,)
Test data set shapes:
X_test shape: (89, 3)
Y_test shape: (89,)
```

```python
# Standardize the input features
mean_input = np.mean(X_train, axis=0)
std_input = np.std(X_train, axis=0)
X_train_standardized = (X_train - mean_input) / std_input

# Standardize the target values
mean_target = np.mean(Y_train, axis=0)
std_target = np.std(Y_train, axis=0)
Y_train_standardized = (Y_train - mean_target) / std_target

# Standardize the target values
mean_test_input = np.mean(X_test, axis=0)
std_test_input = np.std(X_test, axis=0)
X_test_standardized = (X_test - mean_test_input) / std_test_input

class NeuralNetwork:
    def __init__(self, input_layer_size, hidden_layer_size,
output_layer_size, learning_rate, epochs):
        self.input_size = input_layer_size
        self.hidden_size = hidden_layer_size
        self.output_size = output_layer_size
        self.learning_rate = learning_rate
        self.epochs = epochs

        # Initialize weights and biases randomly
        self.weights_input_hidden = np.random.randn(self.input_size,
self.hidden_size)
        self.bias_hidden = np.random.randn(1, self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size,
self.output_size)
        self.bias_output = np.random.randn(1, self.output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)
```

```
    def forward(self, X):
        # Forward propagation
        self.hidden_input = np.dot(X, self.weights_input_hidden) +
self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        self.output = np.dot(self.hidden_output,
self.weights_hidden_output) + self.bias_output
        return self.output

    def backward(self, X, y, output):
        # Backpropagation
        y=y.reshape(-1,1)
        error = y - output
        d_output = error
        error_hidden = d_output.dot(self.weights_hidden_output.T)
        d_hidden = error_hidden *
self.sigmoid_derivative(self.hidden_output)

        # Update weights and biases
        self.weights_hidden_output +=
self.hidden_output.T.dot(d_output) * self.learning_rate
        self.bias_output += np.sum(d_output, axis=0, keepdims=True) *
self.learning_rate
        self.weights_input_hidden += X.T.dot(d_hidden) *
self.learning_rate
        self.bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) *
self.learning_rate

    def train(self, X, y, loss_threshold=1e5):
        for epoch in range(self.epochs):
            output = self.forward(X)
            self.backward(X, y, output)
            loss = np.mean(np.square(y - output))
            if loss > loss_threshold:
                print(f"Loss is diverging at epoch {epoch}. Stopping
training.")
                break

    def predict(self, X):
        return self.forward(X)

# Define the network parameters for each case
input_layer_size = 3
output_layer_size = 1
epochs = 1000

# Create and train the neural network for case (a)
hidden_layer_size=3
learning_rate=0.01
```

```
nn_a = NeuralNetwork(input_layer_size, hidden_layer_size,
output_layer_size, learning_rate, epochs)
nn_a.train(X_train_standardized, Y_train_standardized)
predictions_a = nn_a.predict(X_test_standardized)
predictions_original_scale_a = (predictions_a * std_target) +
mean_target

# Create and train the neural network for case (b)
hidden_layer_size=4
learning_rate=0.001
nn_b = NeuralNetwork(input_layer_size, hidden_layer_size,
output_layer_size, learning_rate, epochs)
nn_b.train(X_train_standardized, Y_train_standardized)
predictions_b = nn_b.predict(X_test_standardized)
predictions_original_scale_b = (predictions_b * std_target) +
mean_target

# Create and train the neural network for case (c)
hidden_layer_size=5
learning_rate=0.0001
nn_c = NeuralNetwork(input_layer_size, hidden_layer_size,
output_layer_size, learning_rate, epochs)
nn_c.train(X_train_standardized, Y_train_standardized)
predictions_c = nn_c.predict(X_test_standardized)
predictions_original_scale_c = (predictions_c * std_target) +
mean_target

# Store and print predictions for case (a)
# predictions_a = nn_a.predict(X_test_scaled)
print("Predictions for case (a):")
print(predictions_original_scale_a)

# Store and print predictions for case (b)
# predictions_b = nn_b.predict(X_test_scaled)
print("Predictions for case (b):")
print(predictions_original_scale_b)

# Store and print predictions for case (c)
# predictions_c = nn_c.predict(X_test_scaled)
print("Predictions for case (c):")
print(predictions_original_scale_c)
```

```
C:\Users\NAIDU\AppData\Local\Temp\ipykernel_8396\3807626742.py:16:
RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))

Loss is diverging at epoch 5. Stopping training.
Predictions for case (a):
[[2.82447039e+08]
 [2.43603877e+08]
```

```
 [2.43603878e+08]
 [2.43603877e+08]
 [2.82447038e+08]
 [2.43603879e+08]
 [3.30096841e+08]
 [2.43603877e+08]
 [2.43603877e+08]
 [2.82447037e+08]
 [2.82447041e+08]
 [2.43603868e+08]
 [2.43603873e+08]
 [2.43603873e+08]
 [2.43603874e+08]
 [2.43603876e+08]
 [2.43603873e+08]
 [2.43603944e+08]
 [2.43603872e+08]
 [3.30096845e+08]
 [2.43603872e+08]
 [2.43603878e+08]
 [2.43603877e+08]
 [2.43603878e+08]
 [2.43603880e+08]
 [3.30096842e+08]
 [2.43603873e+08]
 [2.43603873e+08]
 [2.82447038e+08]
 [2.43603871e+08]
 [2.43603869e+08]
 [2.43603874e+08]
 [2.43603873e+08]
 [2.43603871e+08]
 [2.43603880e+08]
 [3.30096871e+08]
 [3.30096870e+08]
 [2.82447041e+08]
 [2.82447041e+08]
 [2.82447040e+08]
 [2.43603874e+08]
 [2.43603877e+08]
 [2.43603874e+08]
 [2.43603877e+08]
 [2.43603877e+08]
 [2.43603874e+08]
 [3.30096840e+08]
 [2.43603871e+08]
 [2.43603874e+08]
 [3.30096859e+08]
 [2.82447040e+08]
```

```
 [2.43603872e+08]
 [2.43603874e+08]
 [2.82447041e+08]
 [3.30096866e+08]
 [2.43603869e+08]
 [2.43603887e+08]
 [2.43604441e+08]
 [2.82447039e+08]
 [2.43603874e+08]
 [2.43603878e+08]
 [2.43603876e+08]
 [2.43603870e+08]
 [2.43603870e+08]
 [2.82447039e+08]
 [2.43603873e+08]
 [2.43603881e+08]
 [3.30060645e+08]
 [2.82446986e+08]
 [2.43603871e+08]
 [2.82447041e+08]
 [2.43603874e+08]
 [2.43603875e+08]
 [2.43604393e+08]
 [2.43603877e+08]
 [2.82447041e+08]
 [2.43603874e+08]
 [2.43603878e+08]
 [2.43603873e+08]
 [2.43603876e+08]
 [3.30096843e+08]
 [2.43603876e+08]
 [2.43603872e+08]
 [2.43603870e+08]
 [2.82447040e+08]
 [2.43603875e+08]
 [2.82447041e+08]
 [3.30096836e+08]
 [2.43603873e+08]]
Predictions for case (b):
[[640578.11213107]
 [301344.53227121]
 [318557.46497404]
 [256113.72818869]
 [858013.92250846]
 [307536.09394734]
 [253202.220861  ]
 [348810.77877001]
 [455364.53434954]
 [861140.52025754]
```

```
 [601500.53588072]
 [487123.69268706]
 [363517.90993997]
 [451062.93034755]
 [457610.81655794]
 [406552.56725432]
 [398588.79952226]
 [548517.40850708]
 [339704.58432242]
 [321165.68212019]
 [472599.02171541]
 [316631.501488  ]
 [446934.44933559]
 [379328.31216375]
 [398495.71689045]
 [240480.86313021]
 [472494.08677467]
 [383180.82533476]
 [812843.71300549]
 [505318.42744845]
 [528850.87866953]
 [332221.15516913]
 [513975.76691359]
 [498885.80419586]
 [334359.73869032]
 [174216.76904168]
 [358461.86673272]
 [601803.82217311]
 [595496.23098018]
 [709436.04372472]
 [416955.8882481 ]
 [395399.86485938]
 [313653.8546492 ]
 [327082.09830966]
 [332729.68042411]
 [453016.17288837]
 [235675.66724585]
 [507368.45128301]
 [469972.22006677]
 [424685.08093202]
 [572695.16339948]
 [438033.77145085]
 [452213.20127665]
 [637892.77585886]
 [171785.7126994 ]
 [486285.0111629 ]
 [423239.45358359]
 [519112.05088659]
 [712832.95097883]
```

```
 [469863.67444815]
 [452270.92622408]
 [422070.45160397]
 [584736.38526259]
 [337701.62250032]
 [887471.64812692]
 [445441.72562682]
 [406020.74496037]
 [420858.8512789 ]
 [537430.44230961]
 [559032.55195323]
 [633396.62468371]
 [420238.62712916]
 [484120.36611921]
 [509442.90908624]
 [453308.27628129]
 [601763.90602399]
 [294499.72872263]
 [423763.95592791]
 [452322.77502371]
 [399192.44121114]
 [295413.41781747]
 [348228.50482905]
 [487712.23531562]
 [412459.04417128]
 [541385.75268602]
 [468543.37901197]
 [673421.15927495]
 [217220.47524266]
 [391500.89855923]]
Predictions for case (c):
[[605599.50055724]
 [316196.19445179]
 [323496.23862207]
 [308860.14057999]
 [778053.33701269]
 [343138.0727025 ]
 [301478.45969664]
 [341053.85711215]
 [439852.4976292 ]
 [774308.15626783]
 [628652.11456841]
 [500098.14224373]
 [358766.66916394]
 [438748.89894885]
 [450953.5235993 ]
 [366316.39945054]
 [373811.34710042]
 [570173.8380853 ]
```

```
 [371105.07846755]
 [282869.72950804]
 [478885.33965408]
 [325724.17112282]
 [454408.25465102]
 [354180.87214066]
 [342161.38549562]
 [272830.35720565]
 [495749.15226165]
 [365852.17185987]
 [749935.32979778]
 [534498.29899327]
 [522283.05324442]
 [343117.3462179 ]
 [564340.01581131]
 [548765.40640022]
 [342949.45650983]
 [237402.83255615]
 [264480.11376924]
 [612261.94219168]
 [628193.85584485]
 [714646.65995658]
 [381836.72212125]
 [349276.15703001]
 [325768.89552667]
 [344948.0555233 ]
 [332493.5905622 ]
 [441445.84644934]
 [294091.98067797]
 [567581.96888278]
 [504944.9122663 ]
 [237960.01282706]
 [556433.5249191 ]
 [411760.52238274]
 [440486.6384137 ]
 [681418.42492643]
 [249786.99623962]
 [485891.98852934]
 [393313.49440662]
 [586402.7585148 ]
 [705391.85948558]
 [511474.54564448]
 [469710.82860918]
 [384803.93982157]
 [606468.53922086]
 [397337.85663723]
 [775800.6518203 ]
 [428538.6248117 ]
 [346979.67227937]
```

```
 [386846.14740526]
 [552248.03823191]
 [589844.4801505 ]
 [665724.09253388]
 [393908.2348902 ]
 [515531.55269012]
 [572857.90868213]
 [436891.9419655 ]
 [638677.31395045]
 [323821.58903338]
 [375300.17389197]
 [428692.64341478]
 [358706.45431661]
 [298639.60569066]
 [358265.9528242 ]
 [488670.30359718]
 [428853.13191356]
 [531321.74336068]
 [501916.49590113]
 [682340.96966877]
 [287273.47894341]
 [379009.19108208]]
```

```python
# Define the custom accuracy function with the specified threshold
def calculate_accuracy(y_true, y_pred, threshold=300000):
    within_threshold = np.abs(y_true - y_pred) <= threshold
    accuracy = np.mean(within_threshold)
    return accuracy

# Calculate and print the accuracy for case (a)
accuracy_a = calculate_accuracy(Y_test, predictions_original_scale_a)
print("Accuracy for case (a):", accuracy_a)

# Calculate and print the accuracy for case (b)
accuracy_b = calculate_accuracy(Y_test, predictions_original_scale_b)
print("Accuracy for case (b):", accuracy_b)

# Calculate and print the accuracy for case (c)
accuracy_c = calculate_accuracy(Y_test, predictions_original_scale_c)
print("Accuracy for case (c):", accuracy_c)
```

```
Accuracy for case (a): 0.0
Accuracy for case (b): 0.8950890039136473
Accuracy for case (c): 0.8967302108319657
```

```python
#data set for the cross validation

dataT=data.T
X_validation=dataT[0:n-1].T
Y_validation=dataT[n-1].T
```

```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import warnings

# To filter out all warnings
warnings.filterwarnings("ignore")

# Define a custom accuracy function for regression (e.g., within a
threshold)
def calculate_accuracy(y_true, y_pred, threshold=300000):
    within_threshold = np.abs(y_true - y_pred) <= threshold
    accuracy = np.mean(within_threshold)
    return accuracy

# Define the number of folds for cross-validation
kf_values = [5, 10]

# Define specific pairs of hidden layer sizes and learning rates
pairs = [(3, 0.01), (4, 0.001), (5, 0.0001)]

for kf in kf_values:
    print(f"{kf}-Fold Cross-Validation:")
    for hidden_layer_size, learning_rate in pairs:
        print(f"For Hidden Layer Size: {hidden_layer_size}, Learning
Rate: {learning_rate}")

        # Initialize lists to store the metrics for each fold
        mae_scores = []
        mse_scores = []
        r2_scores = []
        accuracy_scores = []

        # Create a KFold cross-validation iterator
        kfold = KFold(n_splits=kf, shuffle=True, random_state=42)

        for train_index, test_index in kfold.split(X_validation):
            # Standardize the target values
            mean_X_val = np.mean(X_validation, axis=0)
            std_X_val = np.std(X_validation, axis=0)
            X_val_stand = (X_validation - mean_X_val) / std_X_val

            # Standardize the target values
            mean_Y_val = np.mean(Y_validation, axis=0)
            std_Y_val = np.std(Y_validation, axis=0)
            Y_val_stand = (Y_validation - mean_Y_val) / std_Y_val

            X_train_fold, X_val_fold = X_val_stand[train_index],
X_val_stand[test_index]
```

```
            Y_train_fold, Y_val_fold = Y_val_stand[train_index],
Y_val_stand[test_index]

            Y_val_original=(Y_val_fold * std_Y_val) + mean_Y_val

            try:
                # Create and train the neural network for the current
fold
                nn = NeuralNetwork(input_layer_size,
hidden_layer_size, output_layer_size, learning_rate, epochs)
                nn.train(X_train_fold, Y_train_fold)
                predictions_val = nn.predict(X_val_fold)
                predictions_original_scale=(predictions_val *
std_Y_val) + mean_Y_val

                # Calculate regression metrics and accuracy for the
current fold
                mae = mean_absolute_error(Y_val_original,
predictions_original_scale)
                mse = mean_squared_error(Y_val_original,
predictions_original_scale)
                r2 = r2_score(Y_val_original,
predictions_original_scale)
                accuracy = calculate_accuracy(Y_val_original,
predictions_original_scale)

                mae_scores.append(mae)
                mse_scores.append(mse)
                r2_scores.append(r2)
                accuracy_scores.append(accuracy)
            except Exception as e:
                print(f"Error in this fold: {str(e)}")

        # Calculate and print the average metrics across all folds
        average_mae = np.nanmean(mae_scores)
        average_mse = np.nanmean(mse_scores)
        average_r2 = np.nanmean(r2_scores)
        average_accuracy = np.nanmean([acc for acc in
accuracy_scores])

        print(f"Average MAE: {average_mae:.4f}")
        print(f"Average MSE: {average_mse:.4f}")
        print(f"Average R2 Score: {average_r2:.4f}")
        print(f"Average Accuracy: {average_accuracy:.4f}")
        print()
```

```
5-Fold Cross-Validation:
For Hidden Layer Size: 3, Learning Rate: 0.01
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 4. Stopping training.
```

```
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 6. Stopping training.
Average MAE: 283069044.6514
Average MSE: 89186966798501408.0000
Average R2 Score: -3441711.6590
Average Accuracy: 0.0000

For Hidden Layer Size: 4, Learning Rate: 0.001
Average MAE: 52321.8489
Average MSE: 4654140301.0202
Average R2 Score: 0.8252
Average Accuracy: 0.8309

For Hidden Layer Size: 5, Learning Rate: 0.0001
Average MAE: 63834.6849
Average MSE: 7252618969.8667
Average R2 Score: 0.7342
Average Accuracy: 0.8454

10-Fold Cross-Validation:
For Hidden Layer Size: 3, Learning Rate: 0.01
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Loss is diverging at epoch 5. Stopping training.
Loss is diverging at epoch 4. Stopping training.
Average MAE: 478922656.3396
Average MSE: 286992541926160832.0000
Average R2 Score: -11748311.8699
Average Accuracy: 0.0000

For Hidden Layer Size: 4, Learning Rate: 0.001
Average MAE: 52016.8906
Average MSE: 4629495380.2278
Average R2 Score: 0.8199
Average Accuracy: 0.8337

For Hidden Layer Size: 5, Learning Rate: 0.0001
Average MAE: 61048.9047
Average MSE: 6483581028.9230
Average R2 Score: 0.7552
Average Accuracy: 0.8394
```