**Question 1**

Use Batch, Mini-Batch and Stochastic Gradient Descent to train a model on the Iris dataset

**Source Code**

**Batch Gradient Descent**  
import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

dataset\_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'class']

data = pd.read\_csv(dataset\_url, header=None, names=columns)

print("Dataset Preview:")

print(data.head())

X = data[['sepal\_length', 'sepal\_width', 'petal\_width']].values

y = data['petal\_length'].values.reshape(-1, 1)

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

m = len(y\_train) # Number of training samples

for epoch in range(epochs):

predictions = np.dot(X\_train, theta)

error = predictions - y\_train

gradient = (1 / m) \* np.dot(X\_train.T, error)

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

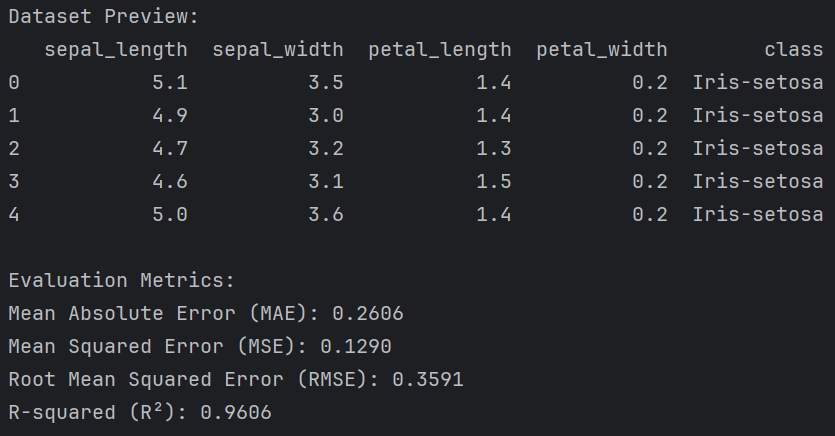
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*

**Mini-Batch Gradient Descent**  
import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

dataset\_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'class']

data = pd.read\_csv(dataset\_url, header=None, names=columns)

print("Dataset Preview:")

print(data.head())

X = data[['sepal\_length', 'sepal\_width', 'petal\_width']].values

y = data['petal\_length'].values.reshape(-1, 1)

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

batch\_size = 16

# Mini-Batch Gradient Descent

m = len(y\_train)

for epoch in range(epochs):

shuffled\_indices = np.random.permutation(m)

X\_train\_shuffled = X\_train[shuffled\_indices]

y\_train\_shuffled = y\_train[shuffled\_indices]

for i in range(0, m, batch\_size):

X\_batch = X\_train\_shuffled[i:i + batch\_size]

y\_batch = y\_train\_shuffled[i:i + batch\_size]

predictions = np.dot(X\_batch, theta)

error = predictions - y\_batch

gradient = (1 / batch\_size) \* np.dot(X\_batch.T, error)

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

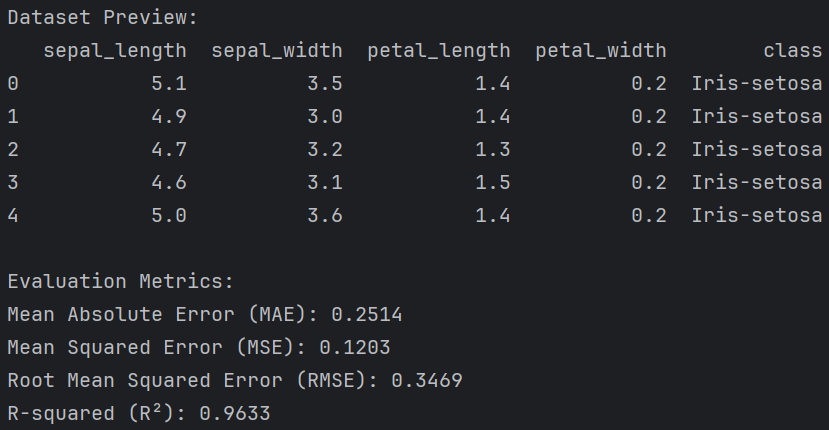
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*

**Stochastic Gradient Descent**  
import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

dataset\_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'class']

data = pd.read\_csv(dataset\_url, header=None, names=columns)

print("Dataset Preview:")

print(data.head())

X = data[['sepal\_length', 'sepal\_width', 'petal\_width']].values

y = data['petal\_length'].values.reshape(-1, 1)

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

m = len(y\_train)

for epoch in range(epochs):

for i in range(m):

random\_index = np.random.randint(0, m)

x\_i = X\_train[random\_index:random\_index + 1]

y\_i = y\_train[random\_index:random\_index + 1]

prediction = np.dot(x\_i, theta)

error = prediction - y\_i

gradient = x\_i.T \* error

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

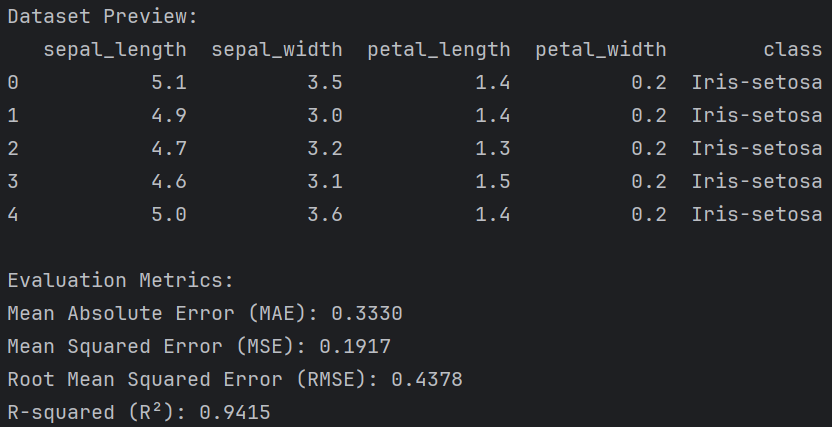
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*

**Question 2**

Use Batch, Mini-Batch and Stochastic Gradient Descent to train a model on the Diabetes dataset.

**Source Code**

**Batch Gradient Descent**  
import numpy as np

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target.reshape(-1, 1)

print("Dataset Preview:")

print(diabetes)

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

m = len(y\_train)

for epoch in range(epochs):

predictions = np.dot(X\_train, theta)

error = predictions - y\_train

gradient = (1 / m) \* np.dot(X\_train.T, error)

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

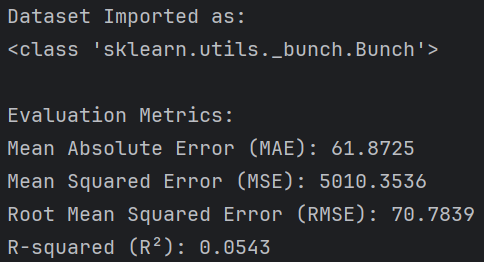
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*

**Mini-Batch Gradient Descent**  
import numpy as np

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target.reshape(-1, 1)

print("Dataset Imported as:")

print(type(diabetes))

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

batch\_size = 32

m = len(y\_train)

for epoch in range(epochs):

indices = np.random.permutation(m)

X\_train\_shuffled = X\_train[indices]

y\_train\_shuffled = y\_train[indices]

for start\_idx in range(0, m, batch\_size):

end\_idx = start\_idx + batch\_size

X\_batch = X\_train\_shuffled[start\_idx:end\_idx]

y\_batch = y\_train\_shuffled[start\_idx:end\_idx]

predictions = np.dot(X\_batch, theta)

error = predictions - y\_batch

gradient = (1 / len(y\_batch)) \* np.dot(X\_batch.T, error)

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

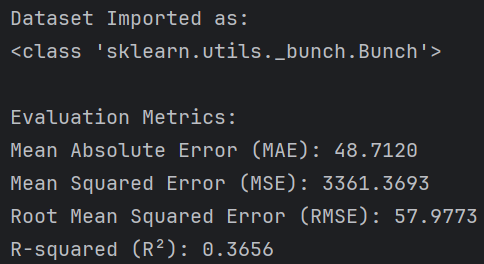
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*

**Stochastic Gradient Descent**  
import numpy as np

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target.reshape(-1, 1)

print("Dataset Imported as:")

print(type(diabetes))

X = np.hstack([np.ones((X.shape[0], 1)), X])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_features = X\_train.shape[1]

theta = np.zeros((n\_features, 1))

learning\_rate = 0.01

epochs = 1000

m = len(y\_train)

for epoch in range(epochs):

for i in range(m):

random\_index = np.random.randint(m)

X\_sample = X\_train[random\_index:random\_index + 1]

y\_sample = y\_train[random\_index:random\_index + 1]

prediction = np.dot(X\_sample, theta)

error = prediction - y\_sample

gradient = np.dot(X\_sample.T, error)

theta -= learning\_rate \* gradient

y\_pred = np.dot(X\_test, theta)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print("\nEvaluation Metrics:")

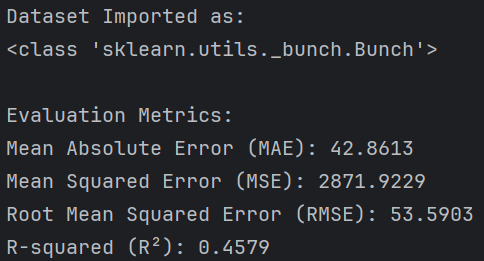
print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

**Output**

*Terminal*