EXPERIMRNT 7A::

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data

np.random.seed(0)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Initialize parameters

m = np.random.randn()

b = np.random.randn()

learning\_rate = 0.1

n\_iterations = 1000

# Gradient Descent

for i in range(n\_iterations):

y\_pred = m \* X + b

error = y\_pred - y

cost = (error \*\* 2).mean()

# Compute gradients

m\_gradient = (2 / len(X)) \* np.dot(X.T, error).sum()

b\_gradient = (2 / len(X)) \* error.sum()

# Update parameters

m -= learning\_rate \* m\_gradient

b -= learning\_rate \* b\_gradient

if i % 100 == 0:

print(f"Iteration {i}: m={m:.3f}, b={b:.3f}, cost={cost:.3f}")

print(f"Final parameters: m={m:.3f}, b={b:.3f}")

# Plot the data and the fitted line

plt.scatter(X, y, color='blue')

plt.plot(X, m \* X + b, color='red')

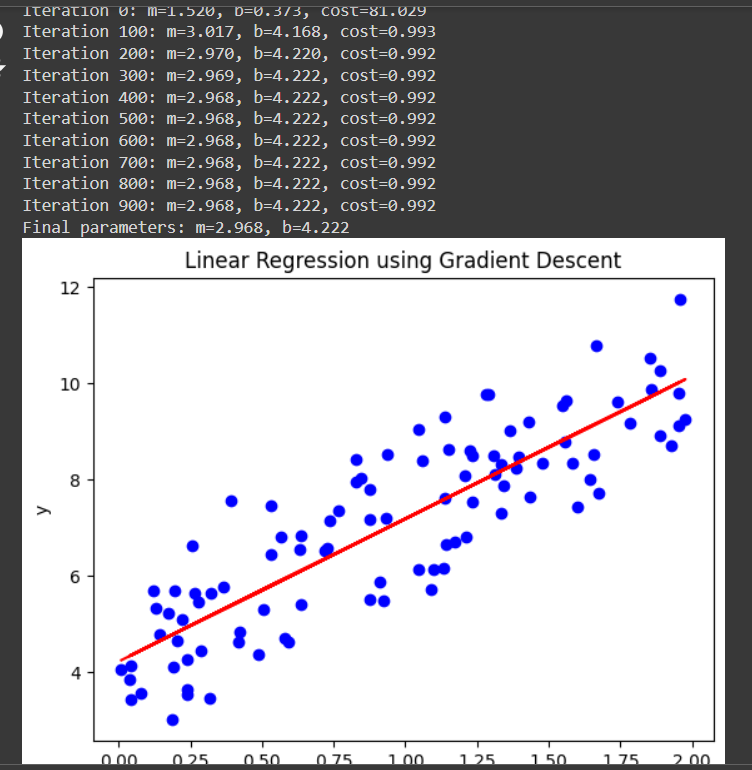
plt.xlabel('X')

plt.ylabel('y')

plt.title('Linear Regression using Gradient Descent')

plt.show()

OUTPUT:



B)

import numpy as np

import matplotlib.pyplot as plt

# Generate modified synthetic data

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1) \* 2 # Added more noise

# Initialize parameters

m = np.random.randn()

b = np.random.randn()

learning\_rate = 0.01

n\_iterations = 2000

# Gradient Descent

for i in range(n\_iterations):

y\_pred = m \* X + b

error = y\_pred - y

cost = (error \*\* 2).mean()

# Compute gradients

m\_gradient = (2 / len(X)) \* np.dot(X.T, error).sum()

b\_gradient = (2 / len(X)) \* error.sum()

# Update parameters

m -= learning\_rate \* m\_gradient

b -= learning\_rate \* b\_gradient

if i % 100 == 0:

print(f"Iteration {i}: m={m:.3f}, b={b:.3f}, cost={cost:.3f}")

print(f"Final parameters: m={m:.3f}, b={b:.3f}")

# Plot the data and the fitted line

plt.scatter(X, y, color='blue', label='Data points')

plt.plot(X, m \* X + b, color='red', label='Fitted line')

plt.xlabel('X')

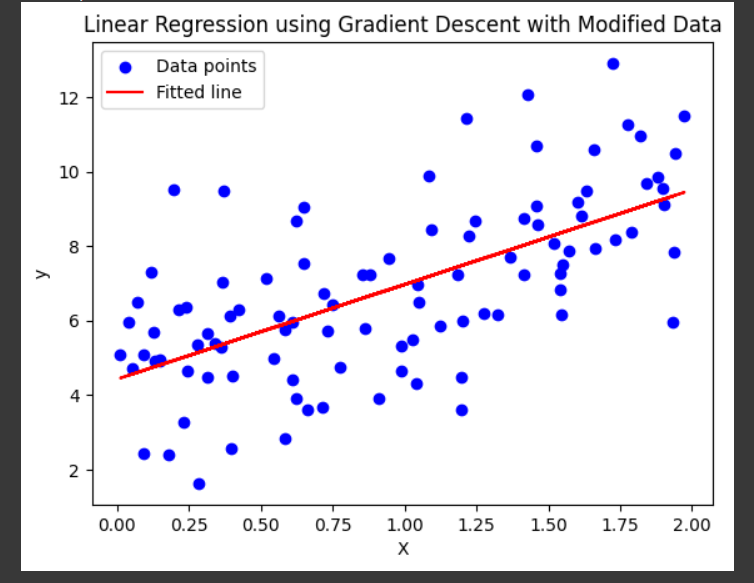
plt.ylabel('y')

plt.title('Linear Regression using Gradient Descent with Modified Data')

plt.legend()

plt.show()

OUTPUT:



EXPERIMENT 8::

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import numpy as np

# Load and preprocess the dataset

(X\_train, y\_train), (X\_test, y\_test) = datasets.cifar10.load\_data()

# Normalize the images to the range [0, 1]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Verify the data

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(X\_train[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[y\_train[i][0]])

plt.show()

# Build the CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10)

])

# Compile the model

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10,

validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"\nTest accuracy: {test\_acc}")

# Make predictions

y\_pred = model.predict(X\_test)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Classification report

print("\nClassification Report:\n")

print(classification\_report(y\_test, y\_pred\_classes, target\_names=class\_names))

# Confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_classes)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', xticklabels=class\_names, yticklabels=class\_names, cmap=plt.cm.Blues)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# Visualize some predictions

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i + 1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(X\_test[i], cmap=plt.cm.binary)

plt.xlabel(f"True: {class\_names[y\_test[i][0]]}\nPred: {class\_names[y\_pred\_classes[i]]}")

plt.show()

OUTPUT:

