**Slip 8**

Q.1 Write a python program to Implement Naïve Bayes. [10M]

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix df = pd.read\_csv('iris.csv')

print("Dataset:")

print(df.head())

X = df.drop('Species', axis=1)

y = df['Species']

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

model.fit(X\_train, y\_train)

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

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

print('\nAccuracy:', accuracy)

print('\nClassification Report:')

print(classification\_report(y\_test, y\_pred))

print('\nConfusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

OR

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

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

# Load the dataset

dataset = pd.read\_csv('social-network-ads.csv')

# Selecting relevant columns

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Naive Bayes to the Training set

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

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

# Making the Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

# Printing the accuracy

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

print("\nAccuracy:", accuracy)

# Printing the classification report

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

Q.2 Write a python program to implement Multiple Linear Regression for given [20M]

dataset.(Position\_Salaries Dataset)

def mse(coef, x, y):

    return np.mean((np.dot(x, coef) - y)\*\*2)/2

def gradients(coef, x, y):

    return np.mean(x.transpose()\*(np.dot(x, coef) - y), axis=1)

def multilinear\_regression(coef, x, y, lr, b1=0.9, b2=0.999, epsilon=1e-8):

    prev\_error = 0

    m\_coef = np.zeros(coef.shape)

    v\_coef = np.zeros(coef.shape)

    moment\_m\_coef = np.zeros(coef.shape)

    moment\_v\_coef = np.zeros(coef.shape)

    t = 0

    while True:

        error = mse(coef, x, y)

        if abs(error - prev\_error) <= epsilon:

            break

        prev\_error = error

        grad = gradients(coef, x, y)

        t += 1

        m\_coef = b1 \* m\_coef + (1-b1)\*grad

        v\_coef = b2 \* v\_coef + (1-b2)\*grad\*\*2

        moment\_m\_coef = m\_coef / (1-b1\*\*t)

        moment\_v\_coef = v\_coef / (1-b2\*\*t)

        delta = ((lr / moment\_v\_coef\*\*0.5 + 1e-8) \*

                 (b1 \* moment\_m\_coef + (1-b1)\*grad/(1-b1\*\*t)))

        coef = np.subtract(coef, delta)

    return coef

coef = np.array([0, 0, 0])

c = multilinear\_regression(coef, x, y, 1e-1)

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(x[:, 1], x[:, 2], y, label='y',

           s=5, color="dodgerblue")

ax.scatter(x[:, 1], x[:, 2], c[0] + c[1]\*x[:, 1] + c[2]\*x[:, 2],

           label='regression', s=5, color="orange")

ax.view\_init(45, 0)

ax.legend()

plt.show()

OR

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

return np.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.add\_subplot(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

**Slip 9**

Q.1 Write a python program to Prepare Scatter Plot (Use Iris Dataset).

=>

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Load the Iris dataset from Seaborn

iris = sns.load\_dataset('iris')

# Data preprocessing

# Separate features and target variable

X = iris.drop('species', axis=1)

y = iris['species']

# Data processing

# Perform any necessary data processing steps here

# Data analysis

# Calculate summary statistics

summary\_stats = X.describe()

print("Summary Statistics:")

print(summary\_stats)

# Plot the graph using Seaborn and Matplotlib

sns.set(style="ticks")

sns.pairplot(iris, hue="species")

plt.show()

OR

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset from seaborn library

iris = sns.load\_dataset("iris")

# Prepare scatter plot

sns.scatterplot(x="sepal\_length", y="sepal\_width", hue="species", style="species", data=iris)

# Add title and labels

plt.title("Scatter Plot of Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

# Show the plot

plt.show()

Q.2 Write a python program to implement Naive Bayes. (social-network-ads)

🡺

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix df = pd.read\_csv('iris.csv')

print("Dataset:")

print(df.head())

X = df.drop('Species', axis=1)

y = df['Species']

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

model.fit(X\_train, y\_train)

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

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

print('\nAccuracy:', accuracy)

print('\nClassification Report:')

print(classification\_report(y\_test, y\_pred))

print('\nConfusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

OR

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

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

# Load the dataset

dataset = pd.read\_csv('social-network-ads.csv')

# Selecting relevant columns

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Naive Bayes to the Training set

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

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

# Making the Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

# Printing the accuracy

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

print("\nAccuracy:", accuracy)

# Printing the classification report

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

**Slip 10**

Q.1 Get total profit of all months and show line plot with the following Style properties. [10M]

(create sales\_data.csv file)

To generate line plot must include following Style properties: –Line Style dotted and Line-color

should be red, Show legend at the lower right location , label name = Month Number ,

label name = Sold units number, Add a circle marker, Line marker color as read, Line width

should be 3

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv("D:\\Python\\Articles\\matplotlib\\sales\_data.csv")

profitList = df ['total\_profit'].tolist()

monthList = df ['month\_number'].tolist()

plt.plot(monthList, profitList, label = 'Profit data of last year',

color='r', marker='o', markerfacecolor='k',

linestyle='--', linewidth=3)

plt.xlabel('Month Number')

plt.ylabel('Profit in dollar')

plt.legend(loc='lower right')

plt.title('Company Sales data of last year')

plt.xticks(monthList)

plt.yticks([100000, 200000, 300000, 400000, 500000])

plt.show()

OR

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

df = pd.read\_csv("D:\\Python\\Articles\\matplotlib\\sales\_data.csv")

profitList = df ['total\_profit'].tolist()

monthList = df ['month\_number'].tolist()

plt.plot(monthList, profitList, label = 'Month-wise Profit data of last year')

plt.xlabel('Month number')

plt.ylabel('Profit in dollar')

plt.xticks(monthList)

plt.title('Company profit per month')

plt.yticks([100000, 200000, 300000, 400000, 500000])

plt.show()

Q.2 Write a python program to implement Multiple Linear Regression for given dataset. [20M]

(Create Own Dataset)

dataset.(Position\_Salaries Dataset)

def mse(coef, x, y):

    return np.mean((np.dot(x, coef) - y)\*\*2)/2

def gradients(coef, x, y):

    return np.mean(x.transpose()\*(np.dot(x, coef) - y), axis=1)

def multilinear\_regression(coef, x, y, lr, b1=0.9, b2=0.999, epsilon=1e-8):

    prev\_error = 0

    m\_coef = np.zeros(coef.shape)

    v\_coef = np.zeros(coef.shape)

    moment\_m\_coef = np.zeros(coef.shape)

    moment\_v\_coef = np.zeros(coef.shape)

    t = 0

    while True:

        error = mse(coef, x, y)

        if abs(error - prev\_error) <= epsilon:

            break

        prev\_error = error

        grad = gradients(coef, x, y)

        t += 1

        m\_coef = b1 \* m\_coef + (1-b1)\*grad

        v\_coef = b2 \* v\_coef + (1-b2)\*grad\*\*2

        moment\_m\_coef = m\_coef / (1-b1\*\*t)

        moment\_v\_coef = v\_coef / (1-b2\*\*t)

        delta = ((lr / moment\_v\_coef\*\*0.5 + 1e-8) \*

                 (b1 \* moment\_m\_coef + (1-b1)\*grad/(1-b1\*\*t)))

        coef = np.subtract(coef, delta)

    return coef

coef = np.array([0, 0, 0])

c = multilinear\_regression(coef, x, y, 1e-1)

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(x[:, 1], x[:, 2], y, label='y',

           s=5, color="dodgerblue")

ax.scatter(x[:, 1], x[:, 2], c[0] + c[1]\*x[:, 1] + c[2]\*x[:, 2],

           label='regression', s=5, color="orange")

ax.view\_init(45, 0)

ax.legend()

plt.show()

OR

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

return np.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.add\_subplot(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

**Slip 11**

Q.1 Read all product sales data and show it  using a multiline plot . [10M]

(create sales\_data.csv file)

Display the number of units sold per month for each product using multiline plots. (i.e., Separate

Plotline for each product ).The graph should look like this.

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv("D:\\Python\\Articles\\matplotlib\\sales\_data.csv")

monthList = df ['month\_number'].tolist()

faceCremSalesData = df ['facecream'].tolist()

faceWashSalesData = df ['facewash'].tolist()

toothPasteSalesData = df ['toothpaste'].tolist()

bathingsoapSalesData = df ['bathingsoap'].tolist()

shampooSalesData = df ['shampoo'].tolist()

moisturizerSalesData = df ['moisturizer'].tolist()

plt.plot(monthList, faceCremSalesData, label = 'Face cream Sales Data', marker='o', linewidth=3)

plt.plot(monthList, faceWashSalesData, label = 'Face Wash Sales Data', marker='o', linewidth=3)

plt.plot(monthList, toothPasteSalesData, label = 'ToothPaste Sales Data', marker='o', linewidth=3)

plt.plot(monthList, bathingsoapSalesData, label = 'ToothPaste Sales Data', marker='o', linewidth=3)

plt.plot(monthList, shampooSalesData, label = 'ToothPaste Sales Data', marker='o', linewidth=3)

plt.plot(monthList, moisturizerSalesData, label = 'ToothPaste Sales Data', marker='o', linewidth=3)

plt.xlabel('Month Number')

plt.ylabel('Sales units in number')

plt.legend(loc='upper left')

plt.xticks(monthList)

plt.yticks([1000, 2000, 4000, 6000, 8000, 10000, 12000, 15000, 18000])

plt.title('Sales data')

plt.show()

OR

import pandas as pd

import matplotlib.pyplot as plt

# Read the sales data from the CSV file

sales\_data = pd.read\_csv('sales\_data.csv')

# Convert the 'Month' column to datetime format

sales\_data['Month'] = pd.to\_datetime(sales\_data['Month'])

# Set the 'Month' column as the index

sales\_data.set\_index('Month', inplace=True)

# Plotting

plt.figure(figsize=(10, 6))

# Iterate over each product and plot its sales data

for product in sales\_data.columns:

plt.plot(sales\_data.index, sales\_data[product], label=product)

plt.xlabel('Month')

plt.ylabel('Number of Units Sold')

plt.title('Product Sales Data')

plt.legend()

plt.grid(True)

plt.xticks(rotation=45)

plt.tight\_layout()

# Show the plot

plt.show()

Q.2 Write a python program to implement k-means algorithm on a synthetic [20M]

Dataset.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

data, labels = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60,  random\_state=42)

kmeans = KMeans(n\_clusters=4)

kmeans.fit(data)

centers = kmeans.cluster\_centers\_

predicted\_labels = kmeans.labels\_

plt.scatter(data[:, 0], data[:, 1], c=predicted\_labels, cmap='viridis', edgecolors='k') plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Clustering with K-Means')

plt.show()

OR

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

X,y = make\_blobs(n\_samples = 500,n\_features = 2,centers = 3,random\_state = 23)

fig = plt.figure(0)

plt.grid(True)

plt.scatter(X[:,0],X[:,1])

plt.show()

**Slip 12**

Q.1 Write a Python program to implement to find all null values in a given [10M]

Data set and remove them.(Use air quality dataset.)

import pandas as pd

# Sample data set

data = {

    'Name': ['John', 'Amy', 'Bob', None],

    'Age': [25, None, 35, None],

    'Gender': ['Male', 'Female', None, 'Male']

}

# Creating a pandas DataFrame

df = pd.DataFrame(data)

# Displaying the original data set

print("\nOriginal Data Set:")

print(df)

# Removing null values

df.dropna(inplace=True)

# Displaying the updated data set without null values

print("\nData Set without Null Values:")

print(df)

OR

import pandas as pd

# Example dataset

data = pd.DataFrame({'category': ['A', 'B', 'C', 'A', 'B', 'B', 'C']})

# Map categorical values to numeric format

data['category\_numeric'] = data['category'].astype('category').cat.codes

print(data)

Q.2 Write a python program to implement Agglomerative clustering on a synthetic [20M]

Dataset.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import AgglomerativeClustering

data, true\_labels = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60,  random\_state=42)

agglomerative = AgglomerativeClustering(n\_clusters=4)

agglomerative.fit(data)

predicted\_labels = agglomerative.labels\_

plt.scatter(data[:, 0], data[:, 1], c=predicted\_labels, cmap='viridis', edgecolors='k') plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Agglomerative Clustering')

plt.show()

OR

# agglomerative clustering

from numpy import unique

from numpy import where

from sklearn.datasets import make\_classification

from sklearn.cluster import AgglomerativeClustering

from matplotlib import pyplot

# define dataset

X, \_ = make\_classification(n\_samples=1000, n\_features=2, n\_informative=2, n\_redundant=0, n\_clusters\_per\_class=1, random\_state=4)

# define the model

model = AgglomerativeClustering(n\_clusters=2)

# fit model and predict clusters

yhat = model.fit\_predict(X)

# retrieve unique clusters

clusters = unique(yhat)

# create scatter plot for samples from each cluster

for cluster in clusters:

# get row indexes for samples with this cluster

row\_ix = where(yhat == cluster)

# create scatter of these samples

pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

# show the plot

pyplot.show()

**Slip 13**

Q1

Read face cream and facewash product sales data and show it using the bar chart. [10M]

The bar chart should display the number of units sold per month for each product. Add a separate

bar for each product in the same chart. (create sales\_data.csv file)

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

df = pd.read\_csv("D:\\Python\\Articles\\matplotlib\\sales\_data.csv")

monthList = df ['month\_number'].tolist()

faceCremSalesData = df ['facecream'].tolist()

faceWashSalesData = df ['facewash'].tolist()

plt.bar([a-0.25 **for** a **in** monthList], faceCremSalesData, width= 0.25, label = 'Face Cream sales data', align='edge')

plt.bar([a+0.25 **for** a **in** monthList], faceWashSalesData, width= -0.25, label = 'Face Wash sales data', align='edge')

plt.xlabel('Month Number')

plt.ylabel('Sales units in number')

plt.legend(loc='upper left')

plt.title(' Sales data')

plt.xticks(monthList)

plt.grid(True, linewidth= 1, linestyle="--")

plt.title('Facewash and facecream sales data')

plt.show()

OR

import numpy as np

import matplotlib.pyplot as plt

X = ['Group A','Group B','Group C','Group D']

Ygirls = [10,20,20,40]

Zboys = [20,30,25,30]

X\_axis = np.arange(len(X))

plt.bar(X\_axis - 0.2, Ygirls, 0.4, label = 'Girls')

plt.bar(X\_axis + 0.2, Zboys, 0.4, label = 'Boys')

plt.xticks(X\_axis, X)

plt.xlabel("Groups")

plt.ylabel("Number of Students")

plt.title("Number of Students in each group")

plt.legend()

plt.show()

Q.2 Write a python program to implement Multiple Linear Regression for given dataset. [20M]

(Use Position\_Salaries Dataset)

dataset.(Position\_Salaries Dataset)

def mse(coef, x, y):

    return np.mean((np.dot(x, coef) - y)\*\*2)/2

def gradients(coef, x, y):

    return np.mean(x.transpose()\*(np.dot(x, coef) - y), axis=1)

def multilinear\_regression(coef, x, y, lr, b1=0.9, b2=0.999, epsilon=1e-8):

    prev\_error = 0

    m\_coef = np.zeros(coef.shape)

    v\_coef = np.zeros(coef.shape)

    moment\_m\_coef = np.zeros(coef.shape)

    moment\_v\_coef = np.zeros(coef.shape)

    t = 0

    while True:

        error = mse(coef, x, y)

        if abs(error - prev\_error) <= epsilon:

            break

        prev\_error = error

        grad = gradients(coef, x, y)

        t += 1

        m\_coef = b1 \* m\_coef + (1-b1)\*grad

        v\_coef = b2 \* v\_coef + (1-b2)\*grad\*\*2

        moment\_m\_coef = m\_coef / (1-b1\*\*t)

        moment\_v\_coef = v\_coef / (1-b2\*\*t)

        delta = ((lr / moment\_v\_coef\*\*0.5 + 1e-8) \*

                 (b1 \* moment\_m\_coef + (1-b1)\*grad/(1-b1\*\*t)))

        coef = np.subtract(coef, delta)

    return coef

coef = np.array([0, 0, 0])

c = multilinear\_regression(coef, x, y, 1e-1)

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(x[:, 1], x[:, 2], y, label='y',

           s=5, color="dodgerblue")

ax.scatter(x[:, 1], x[:, 2], c[0] + c[1]\*x[:, 1] + c[2]\*x[:, 2],

           label='regression', s=5, color="orange")

ax.view\_init(45, 0)

ax.legend()

plt.show()

OR

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

return np.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.add\_subplot(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

**Slip 14**

Q.1 Write a python program to Prepare Scatter Plot (Use Iris Dataset). [10M]

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Load the Iris dataset from Seaborn

iris = sns.load\_dataset('iris')

# Data preprocessing

# Separate features and target variable

X = iris.drop('species', axis=1)

y = iris['species']

# Data processing

# Perform any necessary data processing steps here

# Data analysis

# Calculate summary statistics

summary\_stats = X.describe()

print("Summary Statistics:")

print(summary\_stats)

# Plot the graph using Seaborn and Matplotlib

sns.set(style="ticks")

sns.pairplot(iris, hue="species")

plt.show()

OR

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset from seaborn library

iris = sns.load\_dataset("iris")

# Prepare scatter plot

sns.scatterplot(x="sepal\_length", y="sepal\_width", hue="species", style="species", data=iris)

# Add title and labels

plt.title("Scatter Plot of Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

# Show the plot

plt.show()

Q.2 Write a python program to implement Multiple Linear Regression for given dataset. [20M]

(Use Position\_Salaries Dataset)

dataset.(Position\_Salaries Dataset)

def mse(coef, x, y):

    return np.mean((np.dot(x, coef) - y)\*\*2)/2

def gradients(coef, x, y):

    return np.mean(x.transpose()\*(np.dot(x, coef) - y), axis=1)

def multilinear\_regression(coef, x, y, lr, b1=0.9, b2=0.999, epsilon=1e-8):

    prev\_error = 0

    m\_coef = np.zeros(coef.shape)

    v\_coef = np.zeros(coef.shape)

    moment\_m\_coef = np.zeros(coef.shape)

    moment\_v\_coef = np.zeros(coef.shape)

    t = 0

    while True:

        error = mse(coef, x, y)

        if abs(error - prev\_error) <= epsilon:

            break

        prev\_error = error

        grad = gradients(coef, x, y)

        t += 1

        m\_coef = b1 \* m\_coef + (1-b1)\*grad

        v\_coef = b2 \* v\_coef + (1-b2)\*grad\*\*2

        moment\_m\_coef = m\_coef / (1-b1\*\*t)

        moment\_v\_coef = v\_coef / (1-b2\*\*t)

        delta = ((lr / moment\_v\_coef\*\*0.5 + 1e-8) \*

                 (b1 \* moment\_m\_coef + (1-b1)\*grad/(1-b1\*\*t)))

        coef = np.subtract(coef, delta)

    return coef

coef = np.array([0, 0, 0])

c = multilinear\_regression(coef, x, y, 1e-1)

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(x[:, 1], x[:, 2], y, label='y',

           s=5, color="dodgerblue")

ax.scatter(x[:, 1], x[:, 2], c[0] + c[1]\*x[:, 1] + c[2]\*x[:, 2],

           label='regression', s=5, color="orange")

ax.view\_init(45, 0)

ax.legend()

plt.show()

OR

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

return np.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.add\_subplot(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()