# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load dataset

# Replace 'headbrain.csv' with your dataset path

data = pd.read\_csv('headbrain.csv')

# Extract features and target

X = data['Head Size(cm^3)'].values

Y = data['Brain Weight(grams)'].values

# Calculate means

X\_mean = np.mean(X)

Y\_mean = np.mean(Y)

# Calculate coefficients

numerator = sum((X - X\_mean) \* (Y - Y\_mean))

denominator = sum((X - X\_mean) \*\* 2)

b1 = numerator / denominator

b0 = Y\_mean - b1 \* X\_mean

# Predict values

Y\_pred = b0 + b1 \* X

# Calculate R^2 score

SS\_total = sum((Y - Y\_mean) \*\* 2)

SS\_residual = sum((Y - Y\_pred) \*\* 2)

R2 = 1 - (SS\_residual / SS\_total)

# Display results

print(f"Coefficient b1: {b1}, Intercept b0: {b0}, R^2 Score: {R2:.4f}")

# Plot data points and regression line

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

plt.plot(X, Y\_pred, color='red', label='Regression line')

plt.xlabel('Head Size (cm³)')

plt.ylabel('Brain Weight (grams)')

plt.legend()

plt.title('Simple Linear Regression (HeadBrain Dataset)')

plt.show()

# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load dataset

# Replace 'headbrain.csv' with your dataset path

data = pd.read\_csv('housing\_prices\_SLR.csv')

# Extract features and target

X = data['AREA'].values

Y = data['PRICE'].values

# Calculate means

X\_mean = np.mean(X)

Y\_mean = np.mean(Y)

# Calculate coefficients

numerator = sum((X - X\_mean) \* (Y - Y\_mean))

denominator = sum((X - X\_mean) \*\* 2)

b1 = numerator / denominator

b0 = Y\_mean - b1 \* X\_mean

# Predict values

Y\_pred = b0 + b1 \* X

# Calculate R^2 score

SS\_total = sum((Y - Y\_mean) \*\* 2)

SS\_residual = sum((Y - Y\_pred) \*\* 2)

R2 = 1 - (SS\_residual / SS\_total)

# Display results

print(f"Coefficient b1: {b1}, Intercept b0: {b0}, R^2 Score: {R2:.4f}")

# Plot data points and regression line

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

plt.plot(X, Y\_pred, color='red', label='Regression line')

plt.xlabel('Head Size (cm³)')

plt.ylabel('Brain Weight (grams)')

plt.legend()

plt.title('Simple Linear Regression (HeadBrain Dataset)')

plt.show()

# Import libraries

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

# Load dataset

# Replace 'housing\_prices.csv' with your dataset path

data = pd.read\_csv('housing\_prices\_SLR.csv')

# Extract features and target

X = data[['AREA']] # Ensure it's 2D

Y = data['PRICE']

# Split into training and testing sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

# Create linear regression model

model = LinearRegression()

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

# Predict values

Y\_train\_pred = model.predict(X\_train)

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

# Calculate R^2 scores

R2\_train = r2\_score(Y\_train, Y\_train\_pred)

R2\_test = r2\_score(Y\_test, Y\_test\_pred)

# Display results

print(f"R^2 Score (Training): {R2\_train:.4f}, R^2 Score (Testing): {R2\_test:.4f}")

# Plot data points and regression line

plt.scatter(X\_train, Y\_train, color='blue', label='Data points')

plt.scatter(X\_test, Y\_test, color='green', label='Data points')

plt.plot(X, model.predict(X) , color='red', label='Regression line')

plt.xlabel('Area')

plt.ylabel('Price')

plt.legend()

plt.title('Simple Linear Regression (Housing Prices Dataset)')

plt.show()

##2.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

data = pd.read\_csv('student.csv')

print(data.shape)

data.head()

math = data['Math'].values

read = data['Reading'].values

write = data['Writing'].values

# Ploting the scores as scatter plot

fig = plt.figure()

ax = fig.add\_subplot(111,projection='3d')

ax.scatter(math, read, write, color='#ef1234')

plt.legend()

plt.show()

import numpy as np

# Number of data points

m = len(math)

# Adding x0 (bias term)

x0 = np.ones(m)

X = np.array([x0, math, read]).T

# Initial coefficients

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

Y = np.array(write)

# Learning rate

alpha = 0.0001

# Cost function

def cost\_function(X, Y, B):

"""

Calculate the cost for given X, Y, and coefficients B using Mean Squared Error.

"""

m = len(Y) # Number of data points

J = np.sum((X.dot(B) - Y) \*\* 2) / (2 \* m)

return J

# Initial cost

initial\_cost = cost\_function(X, Y, B)

print("Initial Cost:", initial\_cost)

# Gradient descent function

def gradient\_descent(X, Y, B, alpha, iterations):

"""

Perform gradient descent to minimize the cost function.

Returns updated coefficients and the history of the cost function.

"""

cost\_history = [0] \* iterations

m = len(Y)

for iteration in range(iterations):

# Hypothesis values

h = X.dot(B)

# Difference between hypothesis and actual values

loss = h - Y

# Gradient calculation

gradient = X.T.dot(loss) / m

# Updating coefficients

B = B - alpha \* gradient

# Cost after updating coefficients

cost = cost\_function(X, Y, B)

cost\_history[iteration] = cost

return B, cost\_history

# Performing gradient descent with 100,000 iterations

iterations = 100000

newB, cost\_history = gradient\_descent(X, Y, B, alpha, iterations)

# Displaying new coefficients and final cost

print("New Coefficients [b0, b1, b2]:", newB)

print("Final Cost:", cost\_history[-1])

# Model evaluation - RMSE

def rmse(Y, Y\_pred):

"""

Calculate the Root Mean Square Error (RMSE).

"""

rmse = np.sqrt(np.sum((Y - Y\_pred) \*\* 2) / len(Y))

return rmse

# Model evaluation - R2 score

def r2\_score(Y, Y\_pred):

"""

Calculate the R-squared score.

"""

mean\_y = np.mean(Y)

ss\_tot = np.sum((Y - mean\_y) \*\* 2)

ss\_res = np.sum((Y - Y\_pred) \*\* 2)

r2 = 1 - (ss\_res / ss\_tot)

return r2

# Predictions using the optimized coefficients

Y\_pred = X.dot(newB)

# Evaluating the model

print("R2 Score:", r2\_score(Y, Y\_pred))

print("RMSE:", rmse(Y, Y\_pred))

#2b.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

df=pd.read\_csv("housing\_prices.csv")

df.head()

x=df.iloc[:,:3].values

y=df.PRICE.values

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=100)

# print(mlr\_model.intercept\_)

# print(mlr\_model.coef\_)

from sklearn.linear\_model import LinearRegression

model= LinearRegression(fit\_intercept=True)

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

print(model.score(x\_train,y\_train))

print(model.score(x\_test,y\_test))

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('breast\_cancer.csv')

df=df.iloc[:,:-1]

df.head()

x=df.iloc[:,2:].values

y=df.diagnosis.values

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2)

from sklearn.tree import DecisionTreeClassifier

dt\_classifier.fit(x\_train, y\_train)

predictions = dt\_classifier.predict(x\_test)

prob\_predictions = dt\_classifier.predict\_proba(x\_test)

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

print("Training accuracy Score is : ", accuracy\_score(y\_train,

dt\_classifier.predict(x\_train)))

print("Training Confusion Matrix is : \n", confusion\_matrix(y\_train,

dt\_classifier.predict(x\_train)))

print("Testing Confusion Matrix is : \n", confusion\_matrix(y\_test,

dt\_classifier.predict(x\_test)))

print(classification\_report(y\_test,dt\_classifier.predict(x\_test)))

# Importing required packages

import numpy as np

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.impute import SimpleImputer

# Step 1: Load the dataset

df = pd.read\_csv("breast\_cancer.csv")

# Step 2: Drop irrelevant columns and clean the data

df = df.drop(columns=["id", "Unnamed: 32"]) # Remove ID and unnamed columns

# Step 3: Handle missing values

imputer = SimpleImputer(strategy="mean") # Impute missing values with the mean

df.iloc[:, 1:] = imputer.fit\_transform(df.iloc[:, 1:])

# Step 4: Create Feature Matrix (X) and Target Vector (y)

X = df.iloc[:, 1:].values # Exclude diagnosis column

y = df.diagnosis.values # Only diagnosis column

# Step 5: Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=500)

# Step 6: Baseline model

baseline\_pred = ["B"] \* len(y\_train) # Predict benign for all

baseline\_accuracy = accuracy\_score(y\_train, baseline\_pred)

print(f"Baseline model accuracy: {baseline\_accuracy}")

baseline\_confusion = confusion\_matrix(y\_train, baseline\_pred)

print(f"Baseline Confusion Matrix (Training):\n{baseline\_confusion}")

# Step 7: Train a Gaussian Naive Bayes model

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

# Evaluate the model on training and testing sets

train\_accuracy = nb\_model.score(X\_train, y\_train)

test\_accuracy = nb\_model.score(X\_test, y\_test)

train\_confusion = confusion\_matrix(y\_train, nb\_model.predict(X\_train))

test\_confusion = confusion\_matrix(y\_test, nb\_model.predict(X\_test))

print(f"Training accuracy: {train\_accuracy}")

print(f"Testing accuracy: {test\_accuracy}")

print(f"Training Confusion Matrix:\n{train\_confusion}")

print(f"Testing Confusion Matrix:\n{test\_confusion}")

# Classification reports

train\_report = classification\_report(y\_train, nb\_model.predict(X\_train))

test\_report = classification\_report(y\_test, nb\_model.predict(X\_test))

print("Training Classification Report:")

print(train\_report)

print("Testing Classification Report:")

print(test\_report)

import pandas as pd

df = pd.read\_csv('ch1ex1.csv')

points = df.values

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=3)

model.fit(points)

labels = model.predict(points)

import matplotlib.pyplot as plt

xs = points[:,0]

ys = points[:,1]

plt.scatter(xs, ys, c=labels)

plt.show()

centroids = model.cluster\_centers\_

centroids\_x = centroids[:,0]

centroids\_y = centroids[:,1]

plt.scatter(xs, ys, c=labels)

plt.scatter(centroids\_x, centroids\_y, marker='X', s=200)

plt.show()

import pandas as pd

seeds\_df = pd.read\_csv('seeds-less-rows.csv')

# remove the grain species from the DataFrame, save for later

varieties = list(seeds\_df.pop('grain\_variety'))

# extract the measurements as a NumPy array

samples = seeds\_df.values

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

mergings = linkage(samples, method='complete')

dendrogram(mergings,labels=varieties,leaf\_rotation=90,leaf\_font\_size=6)

plt.show()

#import fcluster from scipy.cluster.hierarchy

from scipy.cluster.hierarchy import fcluster

labels = fcluster(mergings, 6, criterion='distance')

df = pd.DataFrame({'labels': labels, 'varieties': varieties})

ct = pd.crosstab(df['labels'], df['varieties'])

ct

import numpy as np

import pandas as pd

from keras import models

from keras.models import Sequential

from keras.layers import Dense

from keras import layers

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

from sklearn import preprocessing

import matplotlib.pyplot as plt

dataframe = pd.read\_csv('pima-indians-diabetes.csv', delimiter=',')

dataframe.head()

X = dataframe.iloc[:,:8]

y = dataframe.iloc[:,8]

dataframe.shape

features\_train,features\_test,target\_train,target\_test=train\_test\_split(X,y,test\_size=0.33,random\_state=0)

network=models.Sequential()

network.add(Dense(units=8,activation="relu",input\_shape=(features\_train.shape[1],)))

network.add(Dense(units=8,activation="relu"))

network.add(Dense(units=1,activation="sigmoid"))

network.compile(loss='binary\_crossentropy', optimizer='adam',

metrics=['accuracy'])

history=network.fit(features\_train,target\_train,epochs=20,verbose=1,batch\_size=100,validation\_data=(features\_test,target\_test))

training\_loss=history.history["loss"]

test\_loss=history.history["val\_loss"]

epoch\_count=range(1,len(training\_loss)+1)

plt.plot(epoch\_count,test\_loss,"b-")

plt.legend(["Training Loss","Test Loss"])

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.show()

\_, accuracy = network.evaluate(features\_train,target\_train)

print('Accuracy: %.2f' % (accuracy\*100))

predicted\_target= network.predict(features\_test)

\_, accuracy = network.evaluate(features\_test,target\_test)

print('Accuracy: %.2f' % (accuracy\*100))

for i in range(10):

print(predicted\_target[i])

training\_accuracy=history.history["accuracy"]

test\_accuracy=history.history["val\_accuracy"]

plt.plot(epoch\_count,training\_accuracy,"r--")

plt.plot(epoch\_count,test\_accuracy,"b-")

plt.legend(["Training Accuracy","Test Accuracy"])

plt.xlabel("Epoch")

plt.ylabel("Accuracy Score")

plt.show()

from keras.models import Sequential

from keras.layers import Embedding, SimpleRNN

from keras.datasets import imdb

from keras\_preprocessing import sequence

from keras.layers import Dense

max\_features = 10000

maxlen = 500

batch\_size = 32

print('Loading data...')

(input\_train, y\_train), (input\_test, y\_test) = imdb.load\_data(

num\_words=max\_features)

print(len(input\_train), 'train sequences')

print(len(input\_test), 'test sequences')

print('Pad sequences (samples x time)')

input\_train = sequence.pad\_sequences(input\_train, maxlen=maxlen)

input\_test = sequence.pad\_sequences(input\_test, maxlen=maxlen)

print('input\_train shape:', input\_train.shape)

print('input\_test shape:', input\_test.shape)

model = Sequential()

model.add(Embedding(max\_features, 32)) #max\_feature=10,000 so, 320,000

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))#(32+1)\*1=33

model.summary()

model.compile(optimizer='rmsprop', loss='binary\_crossentropy',metrics=['acc'])

history = model.fit(input\_train, y\_train,epochs=5, batch\_size=128,

validation\_split=0.2)

predicted\_classes = model.predict(input\_test)

import numpy as np

predicted\_classes = np.argmax(np.round(predicted\_classes),axis=1)

predicted\_classes.shape, y\_test.shape

correct = np.where(predicted\_classes==y\_test)[0]

print ("Found %d correct labels" % len(correct))

incorrect = np.where(predicted\_classes!=y\_test)[0]

print ("Found %d incorrect labels" % len(incorrect))

from sklearn.metrics import classification\_report

num\_classes=2

target\_names = ["Class {}".format(i) for i in range(num\_classes)]

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

import matplotlib.pyplot as plt

acc = history.history['acc']

val\_acc = history.history['val\_acc']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

import numpy as np

from keras.datasets import mnist

from keras.utils import to\_categorical

import matplotlib.pyplot as plt

import keras

from keras.models import Sequential,Input,Model

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

from keras.layers.advanced\_activations import LeakyReLU

(train\_X,train\_Y), (test\_X,test\_Y) = mnist.load\_data()

print('Training data shape : ', train\_X.shape, train\_Y.shape)

print('Testing data shape : ', test\_X.shape, test\_Y.shape)

classes = np.unique(train\_Y)

nClasses = len(classes)

print('Total number of outputs : ', nClasses)

print('Output classes : ', classes)

plt.subplot(121)

plt.imshow(train\_X[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(train\_Y[0]))

plt.subplot(122)

plt.imshow(test\_X[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(test\_Y[0]))

Text(0.5, 1.0, 'Ground Truth : 7')

train\_X = train\_X.reshape(-1, 28,28, 1)

test\_X = test\_X.reshape(-1, 28,28, 1)

train\_X.shape, test\_X.shape

train\_X = train\_X.astype('float32')

test\_X = test\_X.astype('float32')

train\_X = train\_X / 255

test\_X = test\_X / 255

train\_Y\_one\_hot = to\_categorical(train\_Y)

test\_Y\_one\_hot = to\_categorical(test\_Y)

print('Original label:', train\_Y[0])

print('After conversion to one-hot:', train\_Y\_one\_hot[0])

Original label: 5

After conversion to one-hot: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

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

train\_X,valid\_X,train\_label,valid\_label = train\_test\_split(train\_X,

train\_Y\_one\_hot, test\_size=0.2, random\_state=13)

train\_X.shape,valid\_X.shape,train\_label.shape,valid\_label.shape

batch\_size = 64

epochs = 3

num\_classes = 10

m\_model = Sequential()

m\_model.add(Conv2D(32, kernel\_size=(3,3),activation='linear',input\_shape=(28,28,1),padding='same'))

m\_model.add(LeakyReLU(alpha=0.1))

m\_model.add(MaxPooling2D((2, 2),padding='same'))

m\_model.add(Flatten())

m\_model.add(Dense(128, activation='linear'))

m\_model.add(LeakyReLU(alpha=0.1))

m\_model.add(Dense(num\_classes, activation='softmax'))

m\_model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

m\_model.summary()

m\_train = m\_model.fit(train\_X, train\_label,

batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(valid\_X,

valid\_label))

test\_eval = m\_model.evaluate(test\_X, test\_Y\_one\_hot, verbose=0)

print('Test loss:', test\_eval[0])

print('Test accuracy:', test\_eval[1])

accuracy = m\_train.history['accuracy']

val\_accuracy = m\_train.history['val\_accuracy']

loss = m\_train.history['loss']

val\_loss = m\_train.history['val\_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, '--', label='Training accuracy')

plt.plot(epochs, val\_accuracy, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, '--', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

epochs=1

m\_model = Sequential()

m\_model.add(Conv2D(32, kernel\_size=(3,3),activation='linear',padding='same',input\_shape=(28,28,1)))

m\_model.add(LeakyReLU(alpha=0.1))

m\_model.add(MaxPooling2D((2, 2),padding='same'))

m\_model.add(Dropout(0.25))

m\_model.add(Flatten())

m\_model.add(Dense(128, activation='linear'))

m\_model.add(LeakyReLU(alpha=0.1))

m\_model.add(Dropout(0.3))

m\_model.add(Dense(num\_classes, activation='softmax'))

m\_model.summary()

Model: "sequential\_2"

m\_model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

m\_train\_dropout = m\_model.fit(train\_X, train\_label,

batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(valid\_X,

valid\_label))

print('Test loss:', test\_eval[0])

print('Test accuracy:', test\_eval[1])

accuracy = m\_train\_dropout.history['accuracy']

val\_accuracy = m\_train\_dropout.history['val\_accuracy']

loss = m\_train\_dropout.history['loss']

val\_loss = m\_train\_dropout.history['val\_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training accuracy')

plt.plot(epochs, val\_accuracy, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

predicted\_classes = m\_model.predict(test\_X)

predicted\_classes = np.argmax(np.round(predicted\_classes),axis=1)

predicted\_classes.shape, test\_Y.shape

correct = np.where(predicted\_classes==test\_Y)[0]

print ("Found %d correct labels" % len(correct))

for i, correct in enumerate(correct[:9]):

plt.subplot(3,3,i+1)

plt.imshow(test\_X[correct].reshape(28,28), cmap='gray', interpolation='none')

plt.title("Predicted {}, Class {}".format(predicted\_classes[correct], test\_Y[correct]))

plt.tight\_layout()

Found 9680 correct labels

incorrect = np.where(predicted\_classes!=test\_Y)[0]

print ("Found %d incorrect labels" % len(incorrect))

for i, incorrect in enumerate(incorrect[:9]):

plt.subplot(3,3,i+1)

plt.imshow(test\_X[incorrect].reshape(28,28), cmap='gray', interpolation='none')

plt.title("Predicted {}, Class {}".format(predicted\_classes[incorrect], test\_Y[incorrect]))

plt.tight\_layout()

from sklearn.metrics import classification\_report

target\_names = ["Class {}".format(i) for i in range(num\_classes)]

print(classification\_report(test\_Y, predicted\_classes, target\_names=target\_names))