**Advanced Machine and Deep Learning**

**Name : Shilpa Dinesh Singh**

**Roll No : 545**

**Class : Msc.Cs. Part 1**

**Practical No 1**

**Title : Implemented Simple Linear Regression**

import numpy as np

import matplotlib.pyplot as plt

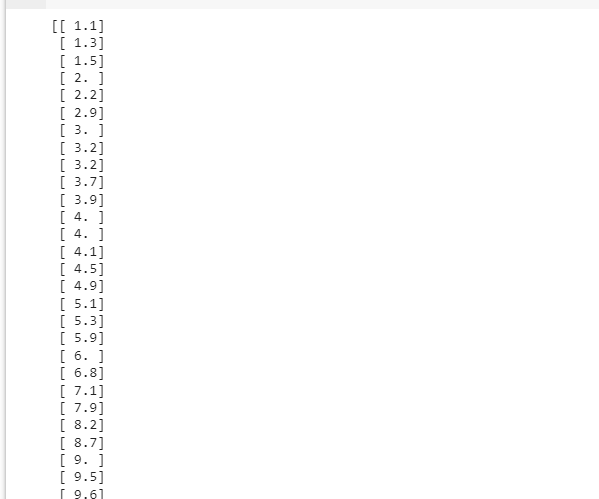
import pandas as pd

dataset=pd.read\_csv("/content/Salary\_Data.csv")

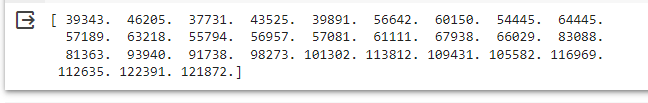
x=dataset.iloc[:, :-1].values

y=dataset.iloc[:, -1].values

print(x)



print(y)



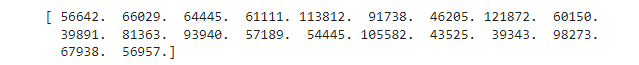
from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size=1/3, random\_state=0)

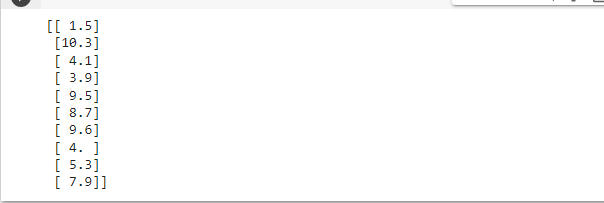
print(x\_train)



print(y\_train)



print(x\_test)



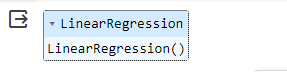
print(y\_test)



from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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



y\_pred = regressor.predict(x\_test)

plt.scatter(x\_train, y\_train,color='red')

plt.plot(x\_train,regressor.predict(x\_train),color='blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel ('Year of Experiene')

plt.ylabel ('Salary')

plt.show()



plt.scatter(x\_test, y\_test,color='red')

plt.plot(x\_test,regressor.predict(x\_test),color='blue')

plt.title('Salary vs Experience (Testing set)')

plt.xlabel ('Year of Experiene')

plt.ylabel ('Salary')

plt.show()



**Practical No 2**

**Title : Implement Multiple Linear Regression**

import numpy as np

import matplotlib.pyplot as plt

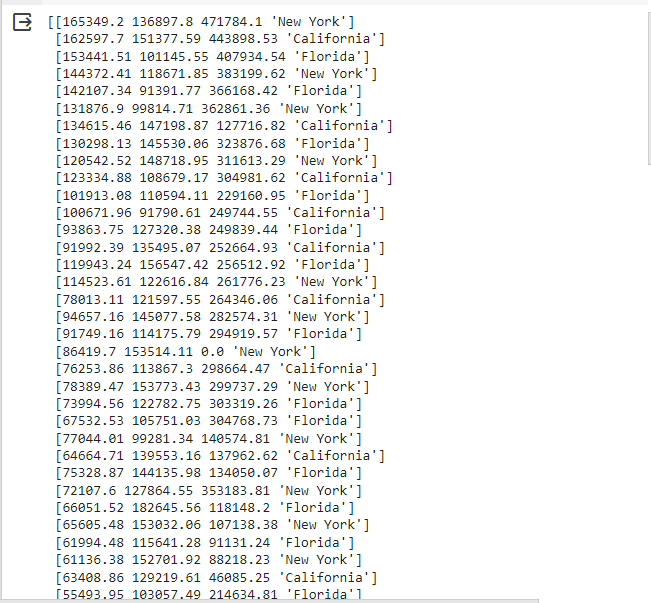
import pandas as pd

dataset = pd.read\_csv("50\_Startups.csv")

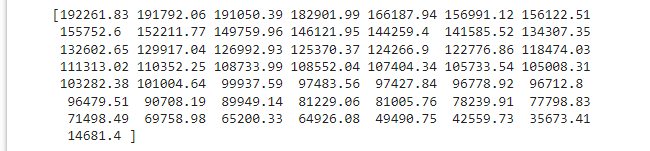
x=dataset.iloc[:, :-1].values

y=dataset.iloc[:, -1].values

print(x)



print(y)



from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

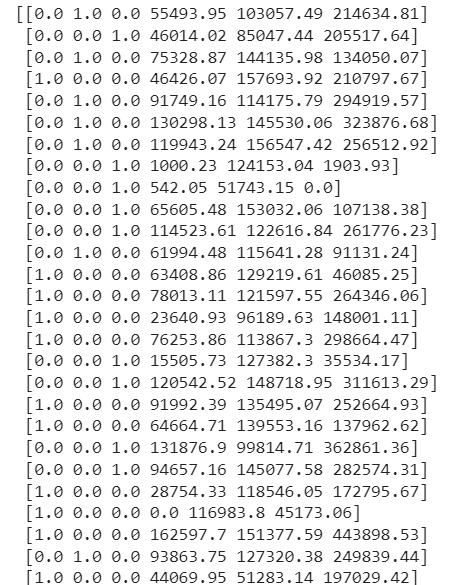
ct = ColumnTransformer(transformers=[('encoded',OneHotEncoder(), [3])],remainder='passthrough')

x=np.array(ct.fit\_transform(x))

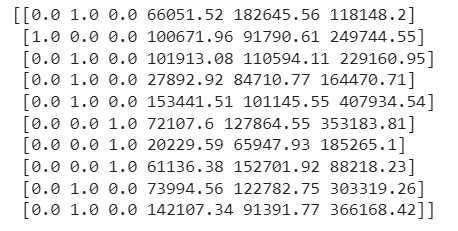
from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size=1/3, random\_state=0)

print(x\_train)



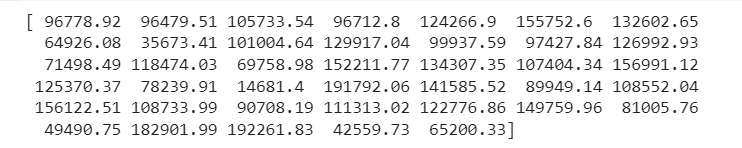
print (x\_test)



print(y\_test)



print(Y\_train)



from sklearn.linear\_model import LinearRegression

regressor=LinearRegression()

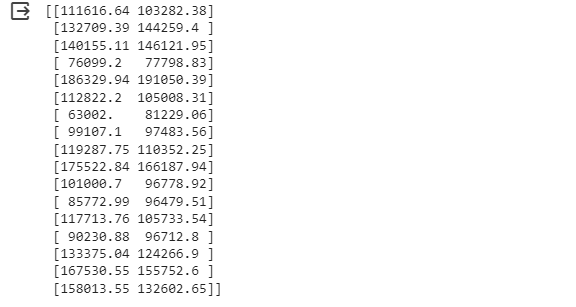
regressor.fit(x\_train,y\_train)



y\_pred = regressor.predict(x\_test)

np.set\_printoptions(precision=2)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1 ))



**Practical No 3**

**Title : Implement Logistic Regression For Classification On Iris Dataset**

import pandas as pd

iris\_data = pd.read\_csv('/content/Iris .csv')

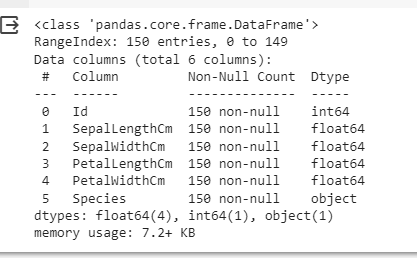
#print(iris\_data)

#iris\_data.sample(5)

iris\_data.head()



iris\_data.info()



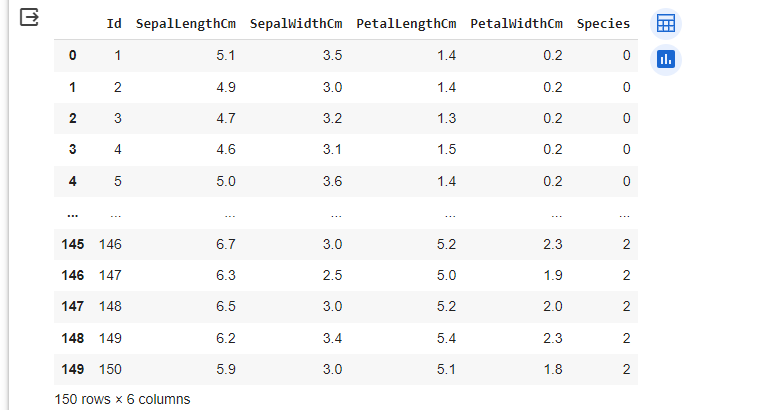
# Iris-setosa = 0 , Iris-versicolor = 1, Iris-virginica = 2

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

iris\_data['Species'] = encoder.fit\_transform(iris\_data['Species'])

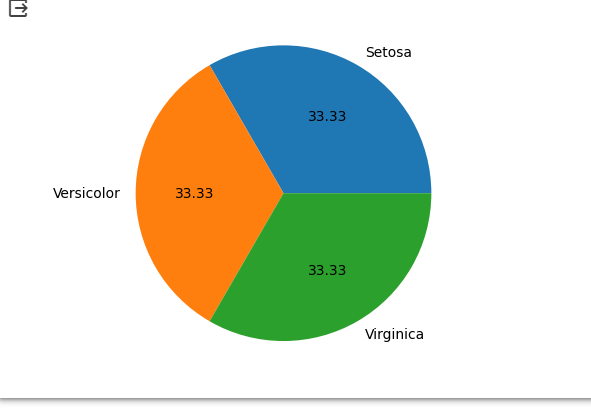
iris\_data.head(150)



import matplotlib.pyplot as plt

plt.pie(iris\_data['Species'].value\_counts(),labels=['Setosa','Versicolor','Virginica'],autopct='%0.2f')

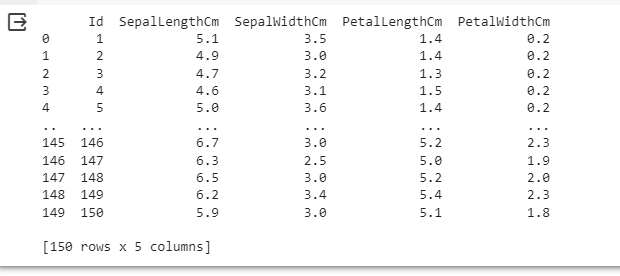
plt.show()



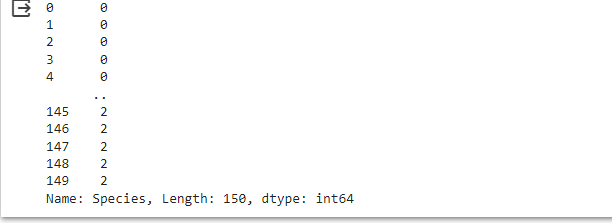
x = iris\_data.drop('Species',axis=1)

y = iris\_data['Species']

print(x)



print(y)



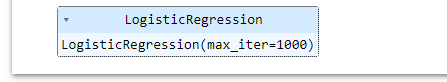
from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=2)

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(max\_iter = 1000)

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



pred\_train = model.predict(x\_train)

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

accuracy\_score(y\_train,pred\_train)

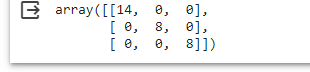


pred\_test = model.predict(x\_test)

accuracy\_score(y\_test,pred\_test)



confusion\_matrix(y\_test,pred\_test)



**Practical No 4**

**Title : Implement Social Media Ads Classification Using Support Vector Machine**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

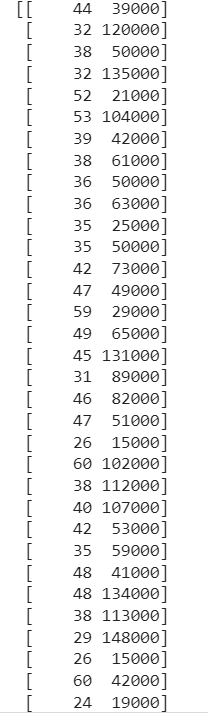
X = dataset.iloc[:, :-1].values

Y = dataset.iloc[:, -1].values

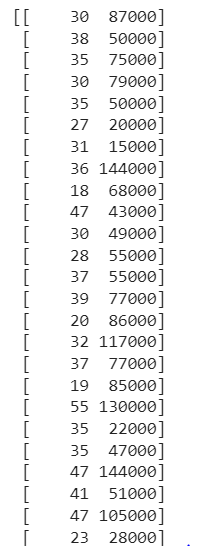
from sklearn.model\_selection import train\_test\_split

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

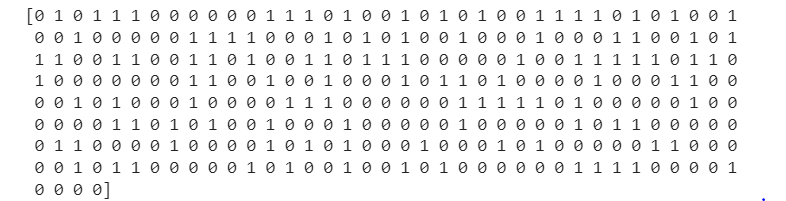
print(X\_train)



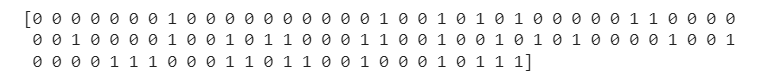
print(X\_test)



print(Y\_train)



print(Y\_test)



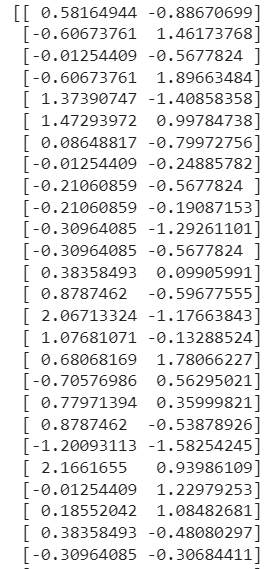
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

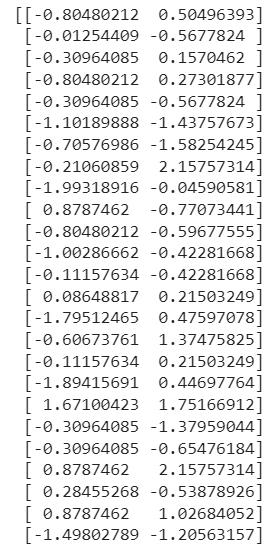
X\_train = sc.fit\_transform(X\_train)

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

print (X\_train)



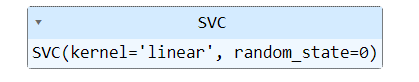
print(X\_test)



from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

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

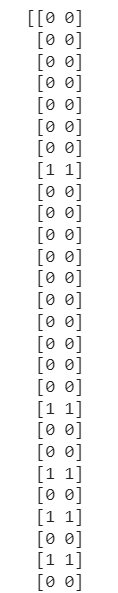


print(classifier.predict(sc.transform([[30,87000]])))



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

print(np.concatenate((Y\_pred.reshape(len(Y\_pred),1),Y\_test.reshape(len(Y\_test),1)),1))

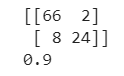


from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

accuracy\_score(Y\_test, Y\_pred)



**Practical No 5**

**Title : Implement Social Media Ads Classification Using KNN**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

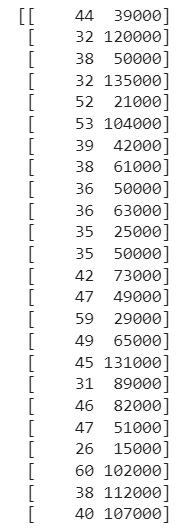
X = dataset.iloc[:, :-1].values

Y = dataset.iloc[:, -1].values

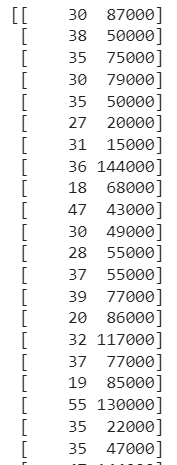
from sklearn.model\_selection import train\_test\_split

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

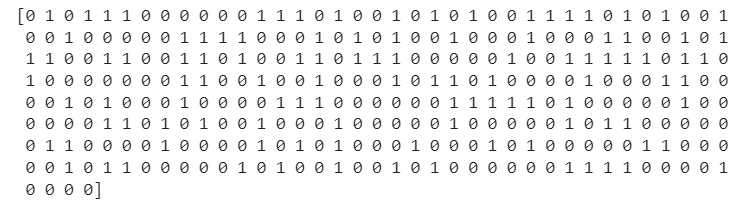
print(X\_train)



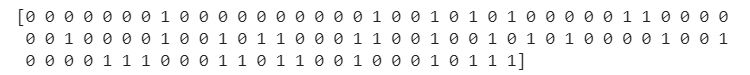
print(X\_test)



print(Y\_train)



print(Y\_test)



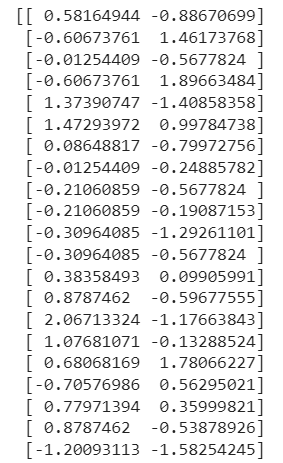
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

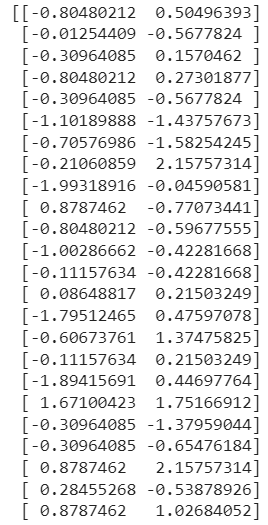
X\_train = sc.fit\_transform(X\_train)

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

print(X\_train)



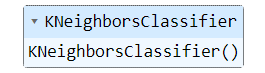
print(X\_test)



from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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

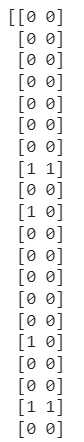


print(classifier.predict(sc.transform([[40,200000]])))



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

print(np.concatenate((Y\_pred.reshape(len(Y\_pred), 1), Y\_test.reshape(len(Y\_test), 1)), 1))

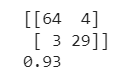


from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

accuracy\_score(Y\_test, Y\_pred)



**Practical No 6**

**Title : Implement K–means Clustering**

import numpy as np

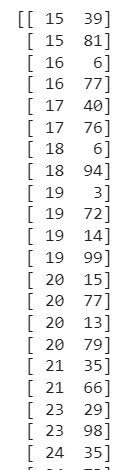
import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('/content/Mall\_Customers.csv')

x = dataset.iloc[:, [3, 4]].values

print(x)



from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

  kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

  kmeans.fit(x)

  wcss.append(kmeans.inertia\_)

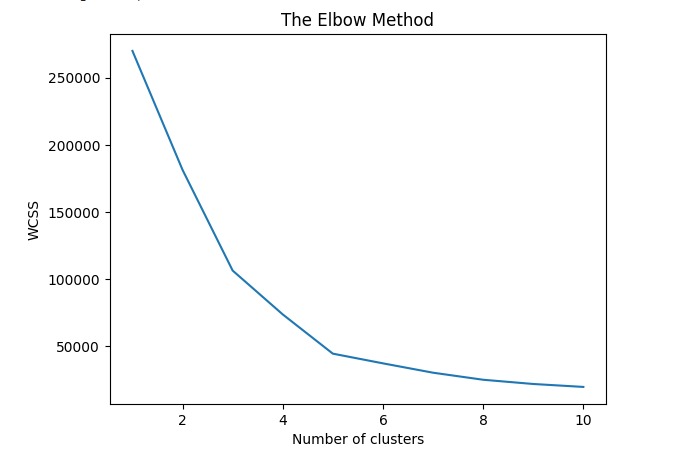
plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()



from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(x)

print(y\_kmeans)

plt.scatter(x[y\_kmeans == 0,0], x[y\_kmeans == 0,1], s = 100, c= 'red', label = 'Cluster 1')

plt.scatter(x[y\_kmeans == 1,0], x[y\_kmeans == 1,1], s = 100, c= 'blue', label = 'Cluster 2')

plt.scatter(x[y\_kmeans == 2,0], x[y\_kmeans == 2,1], s = 100, c= 'green', label = 'Cluster 3')

plt.scatter(x[y\_kmeans == 3,0], x[y\_kmeans == 3,1], s = 100, c= 'cyan', label = 'Cluster 4')

plt.scatter(x[y\_kmeans == 4,0], x[y\_kmeans == 4,1], s = 100, c= 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

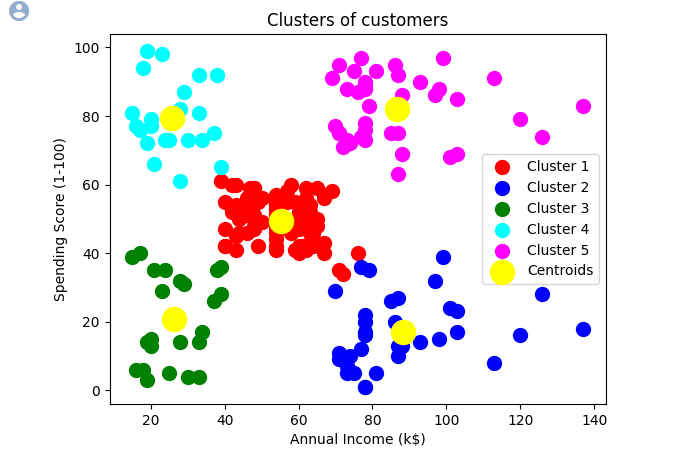
plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()



**Practical No 7**

**Title : Implement Hierarchical Clustering**

import numpy as np

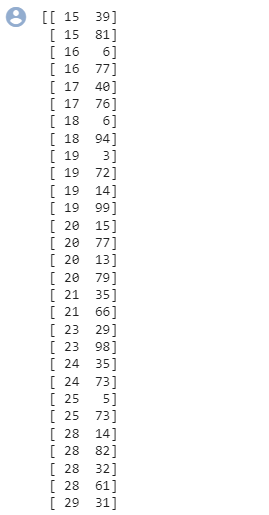
import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('/content/Mall\_Customers.csv')

x = dataset.iloc[:, [3, 4]].values

print(x)



import scipy.cluster.hierarchy as sch

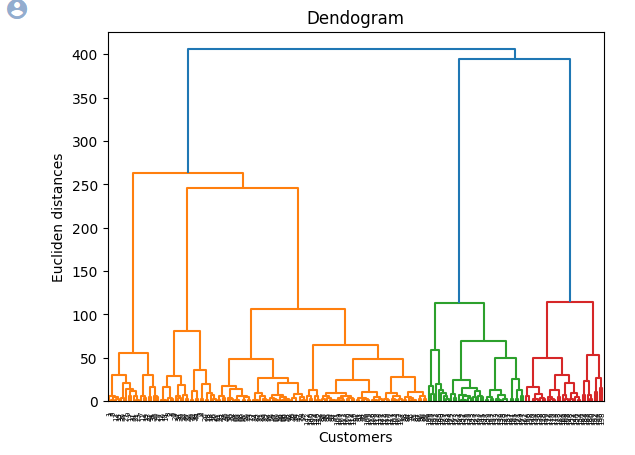
dendrogram = sch.dendrogram(sch.linkage(x, method = 'ward'))

plt.title('Dendogram')

plt.xlabel('Customers')

plt.ylabel('Eucliden distances')

plt.show()

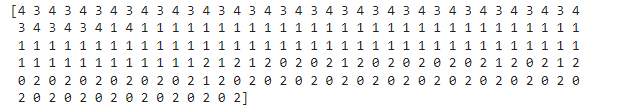


from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(x)

print(y\_hc)



plt.scatter(x[y\_hc == 0,0], x[y\_hc == 0,1], s = 100, c= 'red', label = 'Cluster 1')

plt.scatter(x[y\_hc == 1,0], x[y\_hc == 1,1], s = 100, c= 'blue', label = 'Cluster 2')

plt.scatter(x[y\_hc == 2,0], x[y\_hc == 2,1], s = 100, c= 'green', label = 'Cluster 3')

plt.scatter(x[y\_hc == 3,0], x[y\_hc == 3,1], s = 100, c= 'cyan', label = 'Cluster 4')

plt.scatter(x[y\_hc == 4,0], x[y\_hc == 4,1], s = 100, c= 'magenta', label = 'Cluster 5')

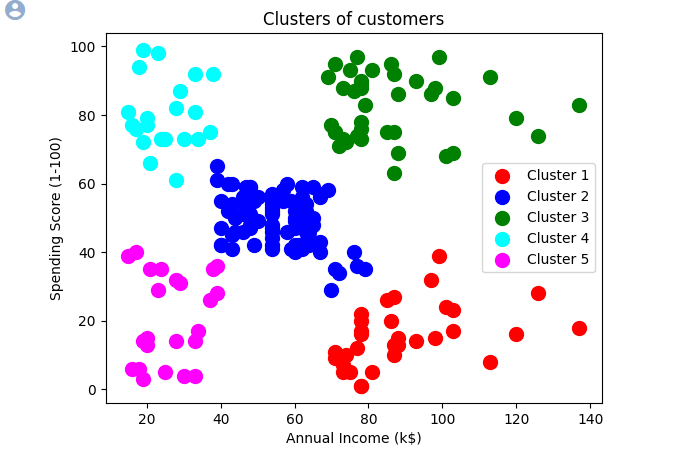
plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()



**Practical No 8**

**Title : Write A Program To Generate A Few Activation Function That Are Being Used In Neural Network**

#sigmoid function

import numpy as np

import matplotlib.pyplot as plt

import numpy as np

plt.style.use('seaborn')

plt.figure(figsize=(8,4))

def SigmoidBinary(t):

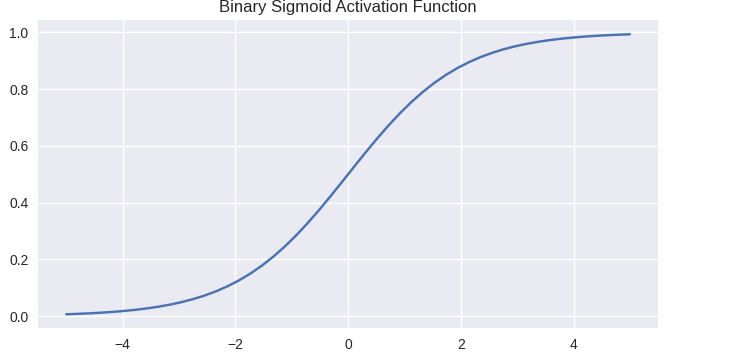
    return 1/(1+np.exp(-t))

t = np.linspace(-5, 5)

plt.plot(t, SigmoidBinary(t))

plt.title('Binary Sigmoid Activation Function')

plt.show()



#Hyperbolic tangent function

plt.style.use('seaborn')

plt.figure(figsize=(8,4))

def HyperbolicTan(t):

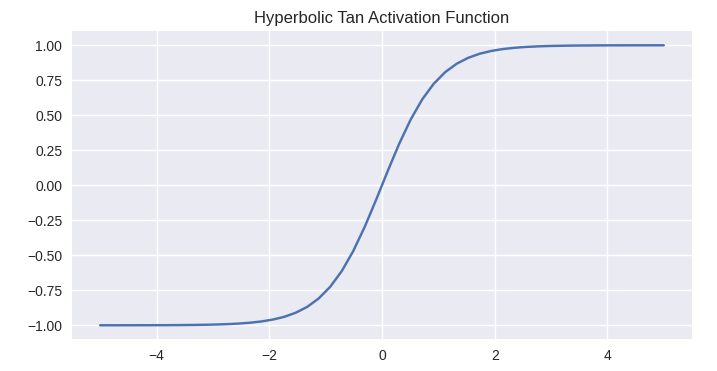
    return np.tanh(t)

t = np.linspace(-5, 5)

plt.plot(t, HyperbolicTan(t))

plt.title('Hyperbolic Tan Activation Function')

plt.show()



#Relu(Rectified linear unit)

plt.style.use('seaborn')

plt.figure(figsize=(8,4))

def RectifiedLinearUnit(t):

    lst=[]

    for i in t:

        if i>=0:

            lst.append(i)

        else:

            lst.append(0)

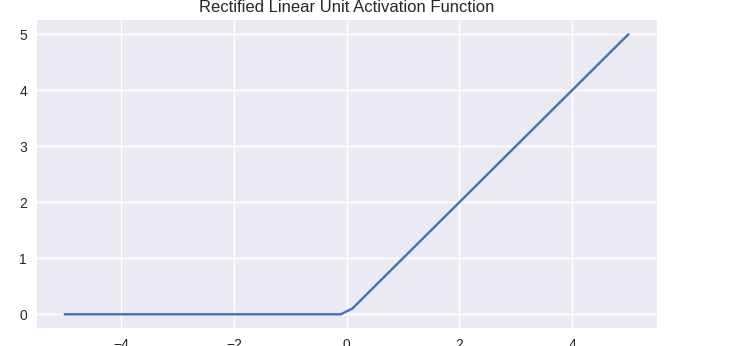
    return lst

arr = np.linspace(-5, 5)

plt.plot(arr, RectifiedLinearUnit(arr))

plt.title('Rectified Linear Unit Activation Function')

plt.show()



#Binary step function

plt.style.use('seaborn')

plt.figure(figsize=(8,4))

def binaryStep(x):

    lst=[]

    for i in t:

        if i>=0:

            lst.append(1)

        else:

            lst.append(0)

    return lst

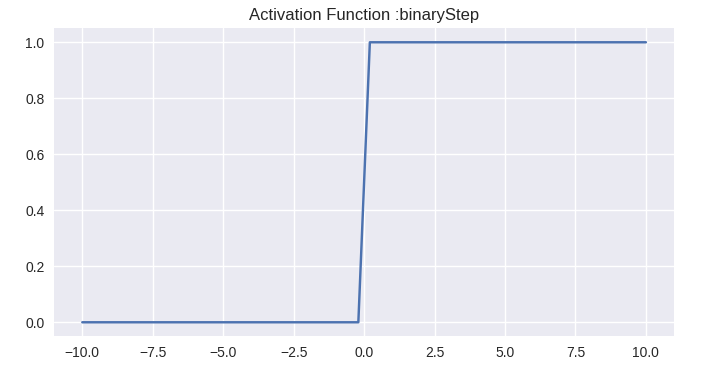
x = np.linspace(-10, 10)

plt.plot(x, binaryStep(x))

plt.axis('tight')

plt.title('Activation Function :binaryStep')

plt.show()



#linear activation function

def linear(x):

    ''' y = f(x) It returns the input as it is'''

    return x

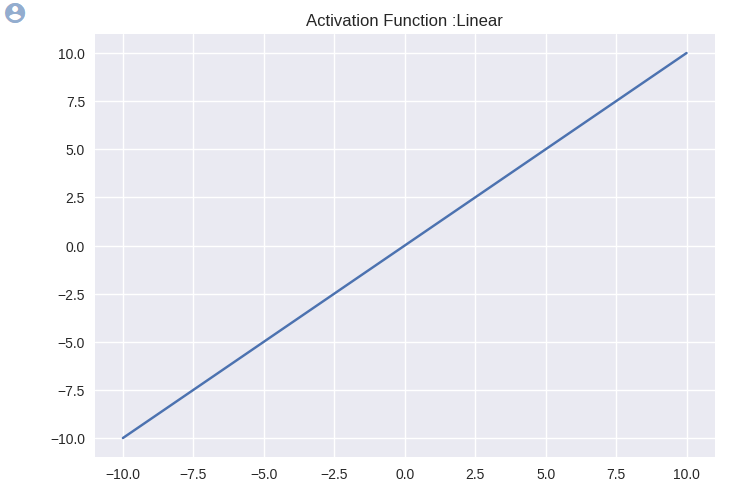
x = np.linspace(-10, 10)

plt.plot(x, linear(x))

plt.axis('tight')

plt.title('Activation Function :Linear')

plt.show()



#softmax activation function

import numpy as np

import matplotlib.pyplot as plt

import numpy as np

plt.style.use('seaborn')

plt.figure(figsize=(8,4))

def softmax(t):

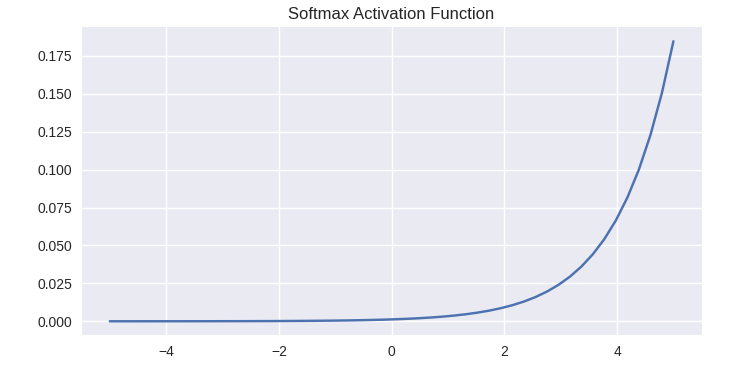
    return np.exp(t) / np.sum(np.exp(t))

t = np.linspace(-5, 5)

plt.plot(t, softmax(t))

plt.title('Softmax Activation Function')

plt.show()



**Practical No 9**

**Title : Implement Tensor Operation(Add, Sub, Mul, Div & Less)**

import numpy as np

tensor\_id = np.array([1.3, 1, 4.0, 23.99])

print(tensor\_id)



print(tensor\_id[0])



print(tensor\_id[2])

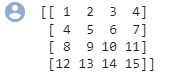


**CREATE TWO DIMENSIONAL TENSOR**

import numpy as np

tensor\_2d = np.array([(1, 2, 3, 4), (4, 5, 6, 7), (8, 9, 10, 11), (12, 13, 14, 15)]

print(tensor\_2d)



tensor\_2d[3] [2]



**TENSOR HANDLING AND MANIPULATION**

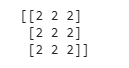
import tensorflow as tf

import numpy as np

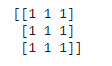
matrix1 = np.array([(2, 2, 2), (2, 2, 2), (2, 2, 2)], dtype = 'int32')

matrix2 = np.array([(1, 1, 1), (1, 1, 1), (1, 1, 1)], dtype = 'int32')

print(matrix1)



print(matrix2)



matrix1 = tf.constant(matrix1)

matrix2 = tf.constant(matrix2)

matrix\_product = tf.matmul(matrix1, matrix2)

matrix\_sum = tf.add(matrix1, matrix2)

matrix\_sub = tf.subtract(matrix1, matrix2)

matrix\_div = tf.divide(matrix1, matrix2)

matrix\_3 = np.array([(2, 7, 2), (1, 4, 2), (9, 0, 2)], dtype = 'float32')

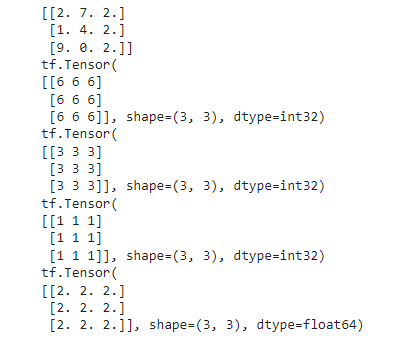
print(matrix\_3)

print(matrix\_product)

print(matrix\_sum)

print(matrix\_sub)

print(matrix\_div)



#importing the library

import tensorflow as tf

#initializing the input tensor

a = tf.constant([7, 8, 13, 11], dtype = tf.float64)

b = tf.constant([2, 13, 14, 5], dtype = tf.float64)

#printing the input tensor

print('a:', a)

print('b:', b)



res = tf.math.less(x = a, y = b)

print('Result:', res)



**Practical No 10**

**Title : Artificial Neural Network**

import numpy as np

import pandas as pd

import tensorflow as tf

tf.\_\_version\_\_

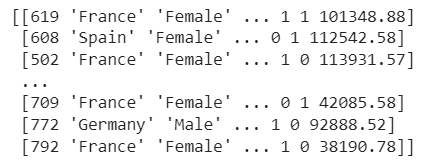


dataset = pd.read\_csv('/content/Churn\_Modelling.csv')

X = dataset.iloc[:, 3:-1].values

Y = dataset.iloc[:, -1].values

print(X)



print(Y)

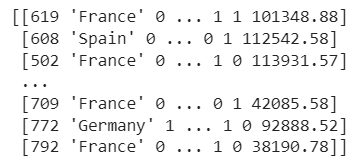


from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X[:, 2] = le.fit\_transform(X[:, 2])

print(X)



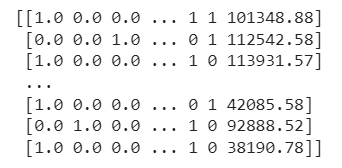
from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)



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 = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

ann = tf.keras.models.Sequential()

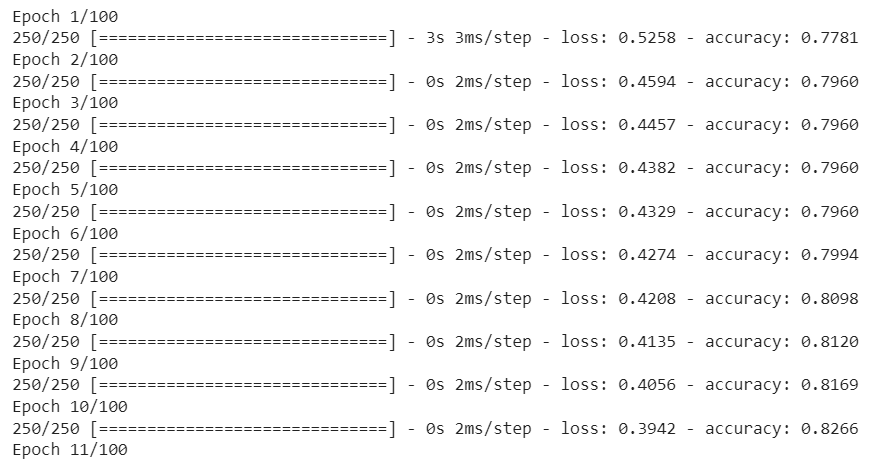
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

ann.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

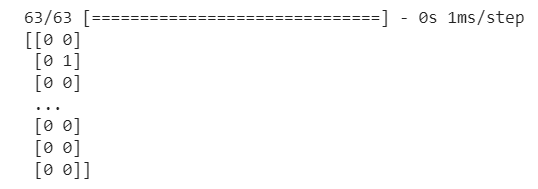
ann.fit(X\_train, Y\_train, batch\_size = 32, epochs = 100)



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

Y\_pred = (Y\_pred > 0.5)

print(np.concatenate((Y\_pred.reshape(len(Y\_pred),1), Y\_test.reshape(len(Y\_test),1)),1))



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

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

print(cm)

accuracy\_score(Y\_test, Y\_pred)

