## Lab Project - 1

*Normalizing Data*

Write a code which works exactly like normalize(data).

| dataset1 = iris\_df['sepal length (cm)'] min\_val= min(dataset1) max\_val= max(dataset1)  def normalize (a\_df, datapoint):  maximum = a\_df.max()  minimum = a\_df.min()  norm\_dp (datapoint-minimum)/(maximum-minimum)  return norm\_dp  normalize(dataset1, dataset1[1]) |
| --- |

## Lab Project - 2

*Cost Function*

Based on the given cost function, write a code for implementing it.

| def cost function(theta,x,y):  n = len(y)  h\_theta = np.dot(X, theta)  squared\_errors= (h\_theta y) \*\* 2  cost = (1 / (2n)) np. sum (squared\_errors) return cost  theta = np.array([8.5, 8.7]) X = np.array([[1, 2], [1, 3], [1, 4]]) y = np.array([2, 3.5, 4.5]) cost cost function(theta, x, y) print(cost) |
| --- |

## Lab Project - 3

*Mapping Variables*

| def mapping (name, response):  resp =[]  for i in range(0, len(response)):  if (response[i] == 'No'):  resp.append(0.0)  elif(response[i] == 'Yes'):  resp.append(1.8)  else:  resp.append(0.5)  data={"Name": name, "Response": resp}  df = pd.DataFrame(data)  print(df) name = ['Reetesh', 'Shruti', 'Kaustubh, 'Vikas', 'Mahima', 'Akshay'] response = ['No', 'Maybe', 'yes', 'Yes', 'maybe', 'Yes'] mapping(name, response) |
| --- |

## Lab Project - 4

*Sigmoid Function*

| def sigmoid(x):  result = 1 / (1 + np.exp(-x))  return result def calculate\_probability(x1, x2):  z = 0,005 \* x1 + 0.5\* x2  probability sigmoid(z)  return probability x1 = 80 x2 = 1 probability calculate\_probability(x1, x2) print(probability) |
| --- |

## Lab Project - 5

*Gradient Descent with Two Variables*

| alpha = 0.1 w1\_t1 = 4 w2\_t1 -3.2 w2\_t2 = w2\_t1 - alpha \* 2 \* w2\_t1 print("w2 at the next iteration:", w2 t2) |
| --- |

## Lab Project - 8

*Evaluation Metrics*

| def stats(df):  tp = df[0][0]  fn=df[0][1]  fp=df[1][0]  tn=df[1][1]  accuracy (tp+tn)/+(tp+tn+fp+fn)  specificity tn/(fp+tn)  sensitivity = tp/(tp+fn)  return max([accuracy, specificity, sensitivity]) df = ((80,40), (30,50)) print(stats (df)) |
| --- |

## Lab Project - 9

*F1 - Score*

| def fStat(df):  tp = df[0][0]  fo = df[0][1]  fp = df[1][8]  tn = df [1][1]  accuracy = (tp+tn)/+(tp+tn+fp+fn)  precision = tp/(tp+tn)  recall tp/(tp+fn)  f1 = 2\*((precision recall)/(precision+recall))  return f1 df = ((100, 400), (150,50)) print(fStat(df)) |
| --- |

## Lab Project - 6 *Weights Calculation*

| import numpy as np # Given data X = np.array([  [1, 2, 3],  [10, -0.5, 1],  [0, 55, 0.3], ]) y = np.array([1, 0, 1]) # Initial weights (20th iteration) w\_t = np.array([0.04, -1, 0.4]) # Learning rate alpha = 0.2 # Sigmoid function def sigmoid(z):  return 1 / (1 + np.exp(-z)) # Calculate the average gradient m = len(y) gradient\_avg = np.mean((sigmoid(np.dot(X, w\_t)) - y) \* X.T, axis=1) # Update weights for the 21st iteration w\_t1 = w\_t - alpha \* gradient\_avg # Print the updated weights print(f"Updated Weights for the 21st iteration: {w\_t1}") |
| --- |

## Lab Project - 7 *Predicting Class Label*

| import numpy as np # Given weights w0 = -0.025 w1 = -0.957 w2 = 0.322 # Given features of the data point x1 = 40 x2 = 0.6 x3 = 1 # Linear combination of features and weights z = w0 + w1 \* x1 + w2 \* x2 + w3 \* x3 # Sigmoid function def sigmoid(z):  return 1 / (1 + np.exp(-z)) sigmoid\_output = sigmoid(z) threshold = 0.6 predicted\_class = 1 if sigmoid\_output > threshold else 0 # Print the results print(f"Sigmoid Output: {sigmoid\_output}") print(f"Predicted Class: {predicted\_class}") |
| --- |

## Lab Project - 10

*Bayes Theorem*

| # Given values  # Prior probability of selecting Bag A randomly p\_a = 1/2  # Probability of drawing a Green ball given Bag A was selected p\_b\_given\_a = 4/7 # Calculate P(B), the total probability of drawing a Green ball p\_b = (1/2 \* 4/7) + (1/2 \* 6/10) # Calculate P(A|B) using Bayes' Theorem p\_a\_given\_b = (p\_b\_given\_a \* p\_a) / p\_b # Print the result print(f"The probability that the bag chosen was A given a Green ball was drawn: {p\_a\_given\_b:.2%}") |
| --- |

## Lab Project - 11

*Distance Measures*

| import math  # Coordinates of points A and B x1, y1 = 7, 50 x2, y2 = 23, 34  # Euclidean distance euclidean\_distance = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)  # Manhattan distance manhattan\_distance = abs(x2 - x1) + abs(y2 - y1)  # Round off the distances to two decimal places euclidean\_distance = round(euclidean\_distance, 2) manhattan\_distance = round(manhattan\_distance, 2)  # Print the results print(f"Euclidean Distance: {euclidean\_distance}") print(f"Manhattan Distance: {manhattan\_distance}") |
| --- |

## Lab Project - 12

*Hopkins Statistics*

| import pandas as pd import numpy as np from sklearn.neighbors import NearestNeighbors  def hopkins\_statistic(data, sample\_size=1000):  """  Calculate the Hopkins statistic for clustering tendency assessment.  Parameters:  - data: Pandas DataFrame, the dataset for which clustering tendency is to be  assessed.  - sample\_size: int, the number of randomly selected points for comparison  (default is 1000).  Returns:  - Hopkins statistic value.  """  # Function to generate random data points for comparison  def random\_points(n, d):  return pd.DataFrame(np.random.rand(n, d), columns=data.columns)   # Number of features in the dataset  d = data.shape[1]   # Generate random data points  random\_data = random\_points(sample\_size, d)   # Fit nearest neighbors on the original and random data  nn\_original = NearestNeighbors(n\_neighbors=1).fit(data)  nn\_random = NearestNeighbors(n\_neighbors=1).fit(random\_data)   # Calculate nearest neighbor distances for both datasets  d1, \_ = nn\_original.kneighbors(data, 2, return\_distance=True)  d2, \_ = nn\_random.kneighbors(random\_data, 2, return\_distance=True)   # Calculate the Hopkins statistic  hopkins\_stat = np.sum(d1[:, 1] / (d1[:, 0] + d1[:, 1])) /   (np.sum(1 / d2[:, 0]))   return hopkins\_stat |
| --- |

## 

## Lab Project - 13

*Cricket Clustering*

Q. Given a dataset here about the batting figures of batsmen in ODI matches. Choose the number of clusters as four. Does ​SR Tendulkar fall in the same cluster as Virat Kohli?

A. Yes, SR Tendulkar does fall in the same cluster as Virat Kohli.

## Lab Project - 14

*Cricket Clustering*

Q. Given a dataset here about the batting figures of batsmen in ODI matches. Based on the clustering, given that the clusters formed are (high SR, high Ave) - A, (low SR, low Ave) - B, (High SR, Low Ave) - C, (Low SR, High Ave) - D. Who all fall in group A?

A. Viv Richards and SR Tendulkar both belong to group A