**Lab 1**

1. **Write a Python program to implement Standard Scalar**.

import numpy as np

import pandas as pd

class StandardNorm:

def scale(self, df):

for i in df.columns:

mean = df[i].mean()

sd = df[i].std()

df[i] = (df[i] - mean) / sd

return df

df = pd.DataFrame(

[[45000, 42], [32000, 26], [58000, 48], [37000, 32]], columns=["Salary", "Age"]

)

print("Original Data")

print(df)

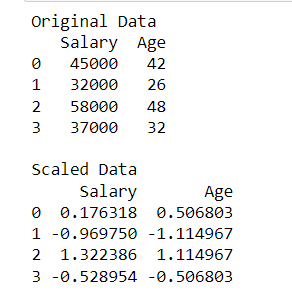
s = StandardNorm()

df\_scaled = s.scale(df)

print("\nScaled Data")

print(df\_scaled)

**Output:**

****

1. **Write a Python program to implement Min-max Scalar.**

import numpy as np

import pandas as pd

class MinMaxNorm:

def scale(self, df):

for c in df.columns:

min = df[c].min()

max = df[c].max()

df[c] = (df[c] - min) / (max - min)

return df

df = pd.DataFrame(

[[45000, 42], [32000, 26], [58000, 48], [37000, 32]], columns=["Salary", "Age"]

)

print("Original Data")

print(df)

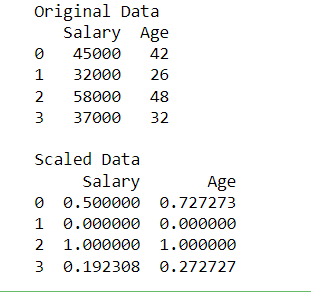
s = MinMaxNorm()

df\_scaled = s.scale(df)

print("\nScaled Data")

print(df\_scaled)

**Output:**

****

**Lab2**

1. **Write a python program to implement K-means Clustering algorithm**

**Generate 1000 2-D data points in the range 0-100 randomly.**

**Divide data points into 3 clusters.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = np.random.rand(1000, 2) \* 100

km = KMeans(n\_clusters=3, init="random")

km.fit(data)

centers = km.cluster\_centers\_

labels = km.labels\_

print("Cluser centers: ", \*centers)

# print("Cluser Labels: ", \*labels)

colors = ["r", "g", "b"]

markers = ["+", "x", "\*"]

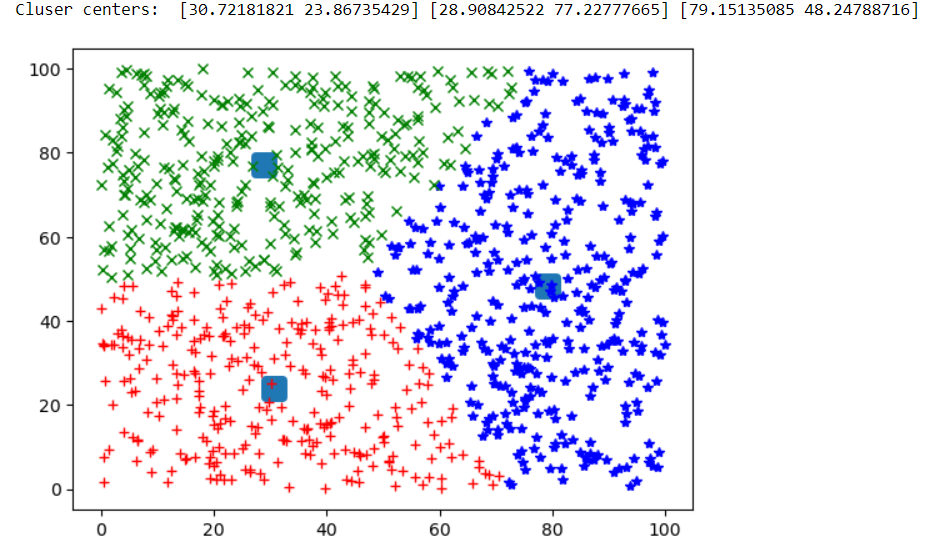
for i in range(len(data)):

plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0], centers[:, 1], marker="s", s=100, linewidths=5)

plt.show()

**Output:**

****

**2.** **Write a python program to implement K-means++ Clustering algorithm.**

**Generate 1000 2-D data points in the range 0-200 randomly.**

**Divide data points into 4 clusters.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = np.random.rand(1000, 2) \* 200

km = KMeans(n\_clusters=4, init="k-means++")

km.fit(data)

centers = km.cluster\_centers\_

labels = km.labels\_

print("Cluser centers: ", \*centers)

# print("Cluser Labels: ", \*labels)

colors = ["r", "g", "b", "y"]

markers = ["+", "x", "\*", "."]

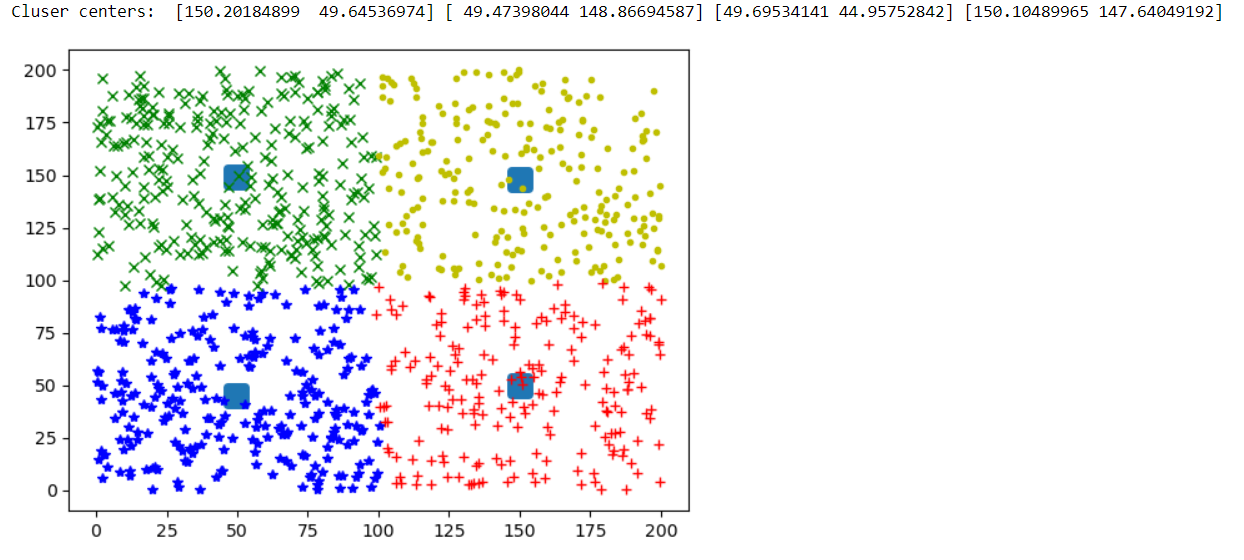
for i in range(len(data)):

plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0], centers[:, 1], marker="s", s=100, linewidths=5)

plt.show()

**Output:**

****

**Lab3**

1. **Write a python program to implement K-means Clustering algorithm**

**Generate 10000 2-D data points in the range 0-100 randomly**

**Divide data points into 5 clusters**

**Find time taken by the algorithm to find clusters**

import time

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = np.random.rand(10000, 2) \* 100

km = KMeans(n\_clusters=5, init="random")

t0 = time.process\_time()

km.fit(data)

t1 = time.process\_time()

tt = t1 - t0

print("Total Time:", tt)

centers = km.cluster\_centers\_

labels = km.labels\_

print("Cluster Centers:", centers)

# print("Cluster Labels:", \*labels)

colors = ["g", "r", "b", "y", "m"]

markers = ["+", "x", "\*", ".", "d"]

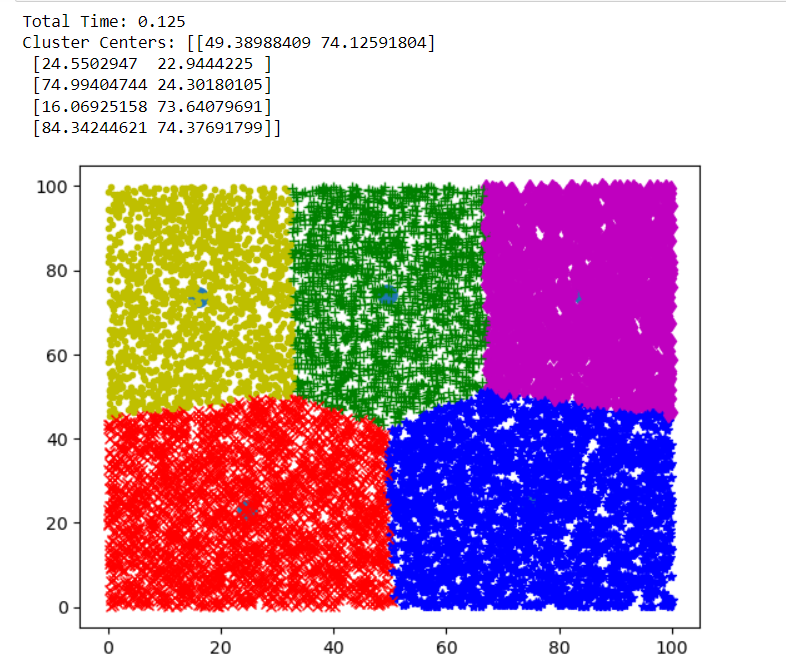
for i in range(len(data)):

plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0], centers[:, 1], marker="o", s=50, linewidths=5)

plt.show()

**Output:**

****

**2.** **Write a python program to implement Mini-batch K-means Clustering algorithm**

**Generate 10000 2-D data points in the range 0-100 randomly**

**Divide data points into 5 clusters**

**Find time taken by the algorithm to find clusters**

**Vary the batch size from 100 to 1500, find time taken by the algorithm in**

**each case and find best value of the batch size.**

import time

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import MiniBatchKMeans

data = np.random.rand(10000, 2) \* 100

mbk = MiniBatchKMeans(n\_clusters=5, init="random", batch\_size=500)

t0 = time.time()

mbk.fit(data)

t1 = time.time()

tt = t1 - t0

print("Total Time: ", tt)

centers = mbk.cluster\_centers\_

labels = mbk.labels\_

print("Cluster Centers:", centers)

# print("Labels:", labels)

colors = ["g", "r", "b", "y", "m"]

markers = ["+", "x", "\*", ".", "d"]

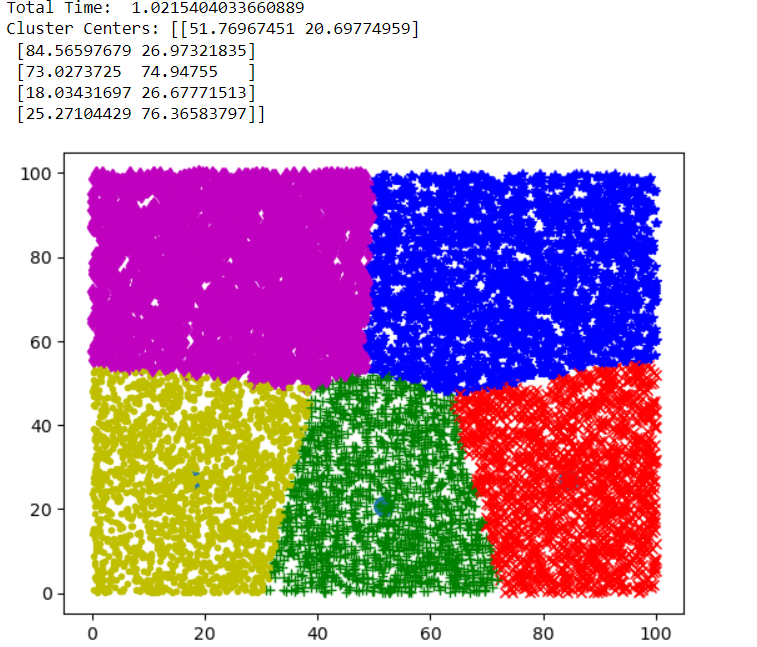
for i in range(len(data)):

plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0], centers[:, 1], marker="o", s=50, linewidths=5)

plt.show()

**Output:**

**­­­­­**

**Lab4**

1. **Write a python program to find clusters of Iris Dataset using KMedoids Algorithm**

!pip install scikit-learn-extra

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn\_extra.cluster import KMedoids

from sklearn import metrics

import matplotlib.pyplot as plt

iris\_data = load\_iris()

x = iris\_data.data

y = iris\_data.target

# print(x[:5])

# print(y[:5])

sc = StandardScaler().fit(x)

sx = sc.transform(x)

km = KMedoids(n\_clusters=3)

km.fit(sx)

py = km.fit\_predict(sx)

# print("Predicted: ", py)

fig = plt.figure(figsize=(12, 8))

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

colors = ["g", "r", "b"]

markers = ["+", "x", "\*"]

for i in range(len(sx)):

ax.scatter(sx[i][0], sx[i][1], sx[i][2], color=colors[py[i]], marker=markers[py[i]])

plt.show()

ri = metrics.rand\_score(y, py)

print("Rand Index:", ri)

hs = metrics.homogeneity\_score(y, py)

print("Homogeniety Score:", hs)

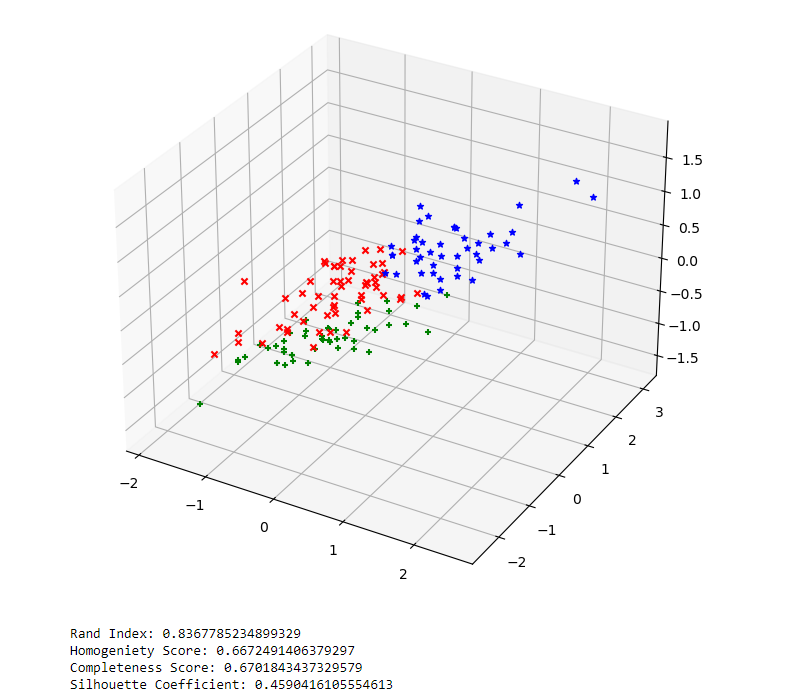
cs = metrics.completeness\_score(y, py)

print("Completeness Score:", cs)

sc = metrics.silhouette\_score(sx, py, metric="euclidean")

print("Silhouette Coefficient:", sc)

**Output:**

****

1. **Write a python program to find clusters of Iris Dataset using Agglomerative Clustering Algorithm.**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering

from sklearn import metrics

import matplotlib.pyplot as plt

iris\_data = load\_iris()

x = iris\_data.data

y = iris\_data.target

# print(x[:5])

# print(y[:5])

sc = StandardScaler().fit(x)

sx = sc.transform(x)

ac = AgglomerativeClustering(n\_clusters=3)

ac.fit(sx)

py = ac.fit\_predict(sx)

# print("Predicted: ", py)

fig = plt.figure(figsize=(12, 8))

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

colors = ["g", "r", "b"]

markers = ["+", "x", "\*"]

for i in range(len(sx)):

ax.scatter(sx[i][0], sx[i][1], sx[i][2], color=colors[py[i]], marker=markers[py[i]])

plt.show()

ri = metrics.rand\_score(y, py)

print("Rand Index:", ri)

hs = metrics.homogeneity\_score(y, py)

print("Homogeniety Score:", hs)

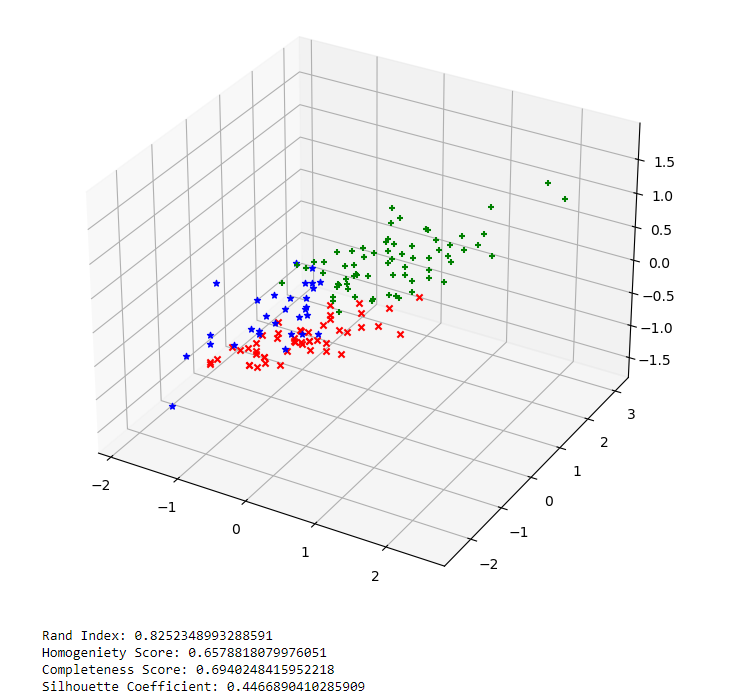
cs = metrics.completeness\_score(y, py)

print("Completeness Score:", cs)

sc = metrics.silhouette\_score(sx, py, metric="euclidean")

print("Silhouette Coefficient:", sc)

**Output:**

****

**Lab 5**

1. **Write a python program to predict diabetes using Naive Bayes Classification.**

import pandas as pd

from sklearn import metrics

from sklearn.naive\_bayes import GaussianNB

dataset = pd.read\_csv("Diabetes.csv")

print("Dataset Size: ", len(dataset))

split = int(len(dataset) \* 0.7)

train, test = dataset.iloc[:split], dataset.iloc[split:]

p = train["Pragnency"].values

g = train["Glucose"].values

bp = train["Blod Pressure"].values

st = train["Skin Thikness"].values

ins = train["Insulin"].values

bmi = train["BMI"].values

dpf = train["DFP"].values

a = train["Age"].values

d = train["Diabetes"].values

trainfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)

traininput = list(trainfeatures)

# print(traininput)

model = GaussianNB()

model.fit(traininput, d)

p = test["Pragnency"].values

g = test["Glucose"].values

bp = test["Blod Pressure"].values

st = test["Skin Thikness"].values

ins = test["Insulin"].values

bmi = test["BMI"].values

dpf = test["DFP"].values

a = test["Age"].values

d = test["Diabetes"].values

testfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)

testinput = list(testfeatures)

predicted = model.predict(testinput)

# print('Actual Class:', \*d)

# print('Predicted Class:', \*predicted)

print("Confusion Matrix:")

print(metrics.confusion\_matrix(d, predicted))

print("\nClassification Measures:")

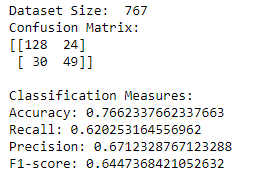
print("Accuracy:", metrics.accuracy\_score(d, predicted))

print("Recall:", metrics.recall\_score(d, predicted))

print("Precision:", metrics.precision\_score(d, predicted))

print("F1-score:", metrics.f1\_score(d, predicted))

**Output:**

****

1. **Write a python program to predict diabetes using ID3 Decision Tree Classifier.**

import pandas as pd

from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier

dataset = pd.read\_csv("Diabetes.csv")

print("Dataset Size: ", len(dataset))

split = int(len(dataset) \* 0.7)

train, test = dataset.iloc[:split], dataset.iloc[split:]

p = train["Pragnency"].values

g = train["Glucose"].values

bp = train["Blod Pressure"].values

st = train["Skin Thikness"].values

ins = train["Insulin"].values

bmi = train["BMI"].values

dpf = train["DFP"].values

a = train["Age"].values

d = train["Diabetes"].values

trainfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)

traininput = list(trainfeatures)

# print(traininput)

model = DecisionTreeClassifier(criterion="entropy", max\_depth=4)

model.fit(traininput, d)

p = test["Pragnency"].values

g = test["Glucose"].values

bp = test["Blod Pressure"].values

st = test["Skin Thikness"].values

ins = test["Insulin"].values

bmi = test["BMI"].values

dpf = test["DFP"].values

a = test["Age"].values

d = test["Diabetes"].values

testfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)

testinput = list(testfeatures)

predicted = model.predict(testinput)

# print('Actual Class:', \*d)

# print('Predicted Class:', \*predicted)

print("Confusion Matrix:")

print(metrics.confusion\_matrix(d, predicted))

print("\nClassification Measures:")

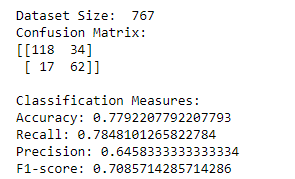
print("Accuracy:", metrics.accuracy\_score(d, predicted))

print("Recall:", metrics.recall\_score(d, predicted))

print("Precision:", metrics.precision\_score(d, predicted))

print("F1-score:", metrics.f1\_score(d, predicted))

Output:



Lab6

1. Write a python program to implement Apriori algorithm to find association rules.

!pip install apyori

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

dataset = pd.read\_csv("store\_data.csv", header=None)

# print(dataset)

records = []

for i in range(0, 7501):

test = []

data = dataset.iloc[i]

data = data.dropna()

for j in range(0, len(data)):

test.append(str(dataset.values[i, j]))

records.append(test)

# print(records)

association\_rules = apriori(

records, min\_support=0.005, min\_confidence=0.2, min\_lift=3, min\_length=2

)

association\_results = list(association\_rules)

for item in association\_results:

# print(item)

# print(item[2])

# print(item[2][0])

print(list(item[2][0][0]), '->', list(item[2][0][1]))

**Output:**

