**APRIORI ALGORITHM**

from efficient\_apriori import apriori

transactions = [

['butter', 'milk', 'bread'],

['butter', 'milk', 'apple'],

['bread', 'milk', 'banana'],

['milk', 'bread', 'butter']

]

itemsets, rules = apriori(transactions, min\_support=0.3, min\_confidence=0.8)

print("Frequent Itemsets:")

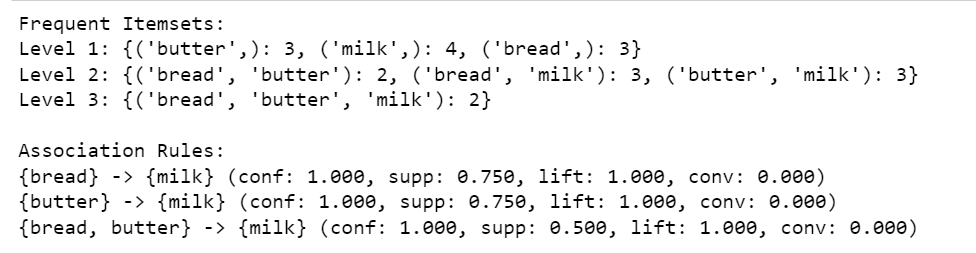
for k, v in itemsets.items():

print(f"Level {k}: {v}")

print("\nAssociation Rules:")

for rule in rules:

print(rule)



**K-MEANS ALGORITHM**

from sklearn.cluster import KMeans

data\_features = ["Hieght", "Age"]

X = [[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32],

[166,6],[128,32],[179,10],[136,34],[186,20],[126,25],[176,28],[112,38],

[169,9],[171,36],[116,25],[196,25]]

model = KMeans(n\_clusters=3)

model.fit(X)

cluster\_labels = model.predict(X)

print(cluster\_labels)

x1 = [] # hieght

x2 = [] # age

for item in X:

x1.append(item[0])

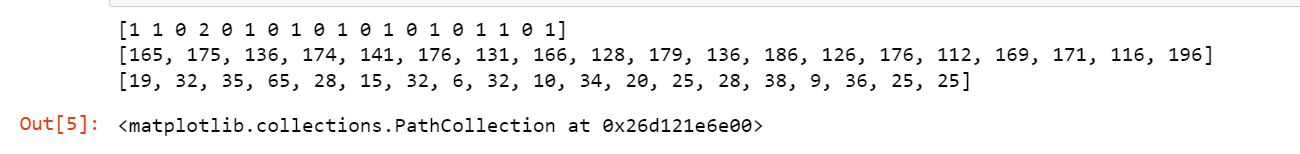
x2.append(item[1])

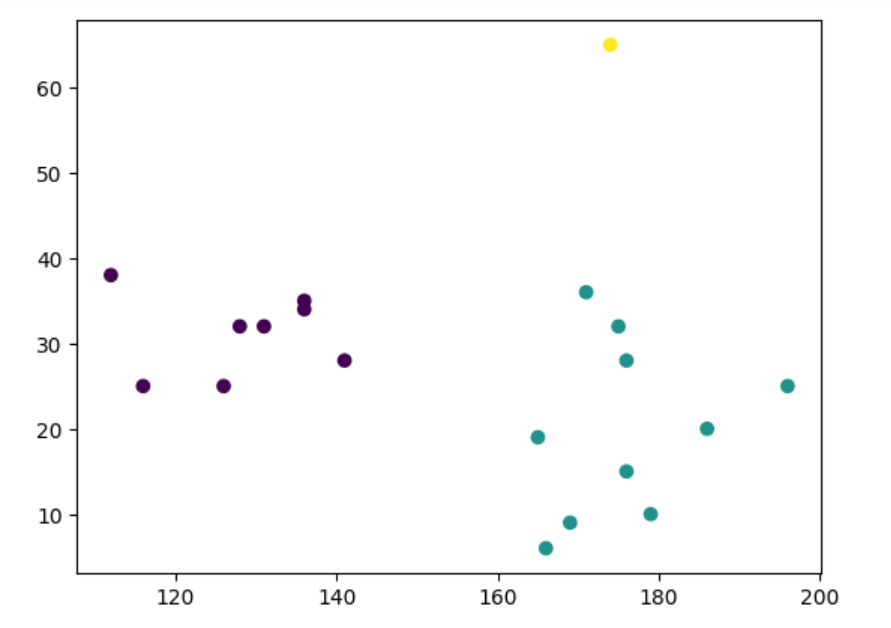
print(x1)

print(x2)

import matplotlib.pyplot as plt

plt.scatter(x1,x2, c=model.labels\_)





**KNN ALGORITHM**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import NearestNeighbors

A = np.array(

[

[3.1, 2.3],

[2.3, 4.2],

[3.9, 3.5],

[3.7, 6.4],

[4.8, 1.9],

[8.3, 3.1],

[5.2, 7.5],

[4.8, 4.7],

[3.5, 5.1],

[4.4, 2.9],

]

)

plt.figure()

plt.title('Input data')

plt.scatter(A[:,0], A[:,1], marker = 'x', s = 50, color = 'red')

test\_data = [5.2, 2.9]

knn\_model = NearestNeighbors(n\_neighbors = 3, algorithm = 'auto')

knn\_model.fit(A)

distances, indices = knn\_model.kneighbors([test\_data])

print(distances)

print(indices)

print("\nK Nearest Neighbors:")

for rank, index in enumerate(indices[0][:3], start = 1):

print(str(rank) + " is", A[index])

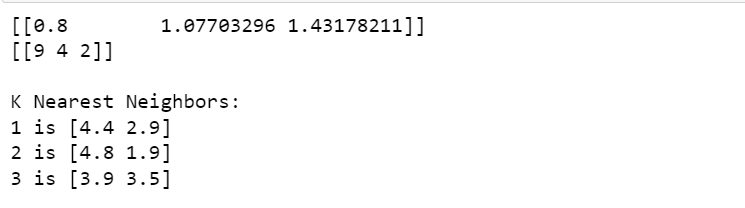
plt.figure()

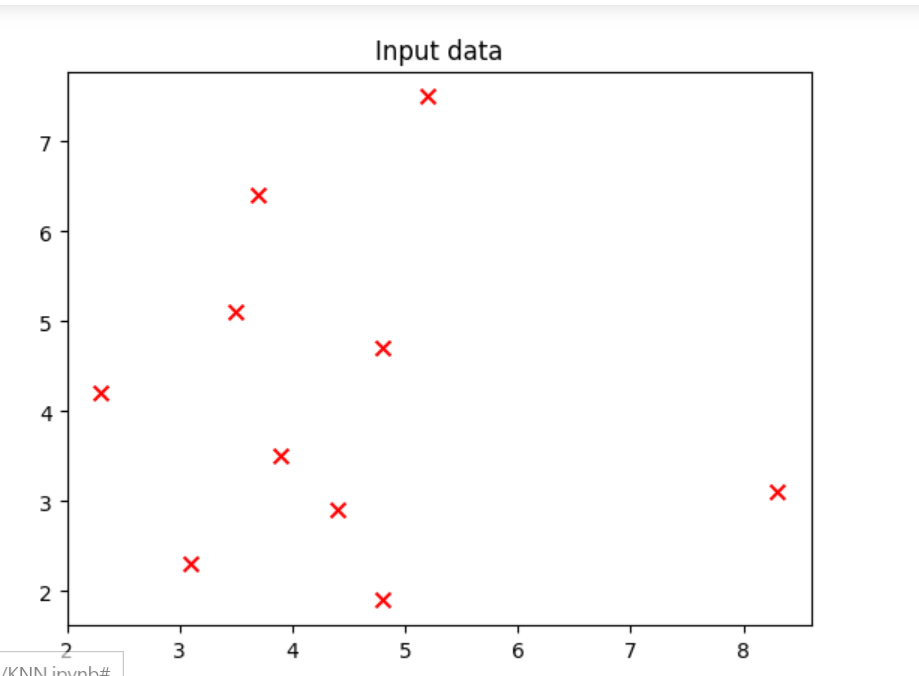
plt.title('Nearest neighbors')

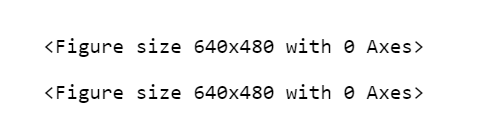
plt.scatter(A[:, 0], A[:, 1], marker = 'x', s = 100, color = 'red')

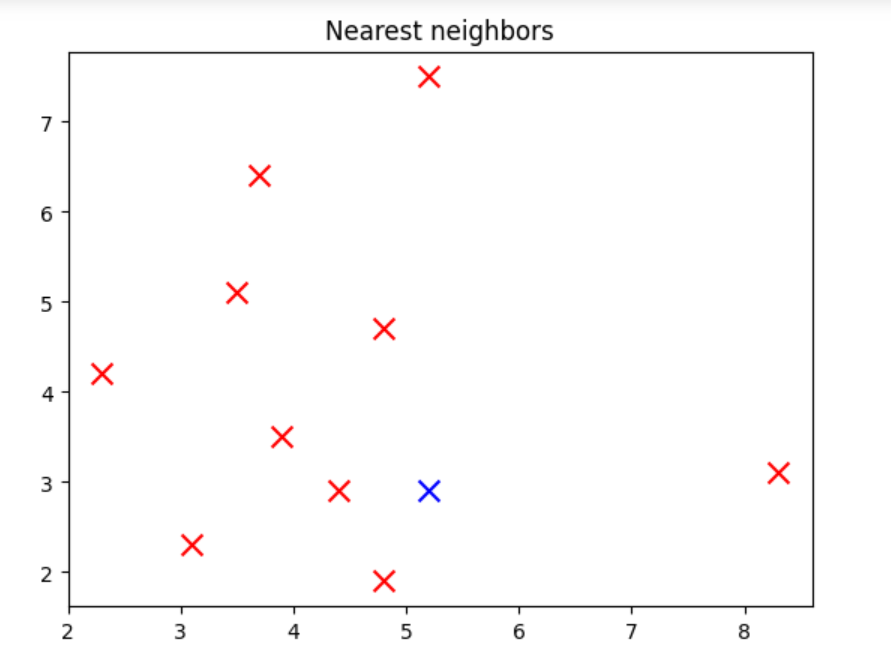
plt.scatter(test\_data[0], test\_data[1],marker = 'x', s = 100, color = 'blue')

plt.show()









**SVM**

from sklearn.svm import SVC

data\_feature\_names = ['height','age']

X = [[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32],

[166,6],[128,32],[179,10],[136,34],[186,2],[126,25],[176,28],[112,38],

[169,9],[171,36],[116,25],[196,25]]

Y = ['Man','Woman','Woman','Man','Woman','Man','Woman','Man','Woman',

'Man','Woman','Man','Woman','Woman','Woman','Man','Woman','Woman','Man']

SVC\_model = SVC(gamma='auto')

SVC\_model.fit(X, Y)

print(SVC\_model.predict([[156, 53]]))

print('Accuracy on the training subset:',format(SVC\_model.score(X,Y)))

**LOGISTIC REGRESSION**

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as pl

X = [[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32],

[166,6],[128,32],[179,10],[136,34],[186,2],[126,25],[176,28],[112,38],

[169,9],[171,36],[116,25],[196,25]]

Y = ['Man','Woman','Woman','Man','Woman','Man','Woman','Man','Woman',

'Man','Woman','Man','Woman','Woman','Woman','Man','Woman','Woman','Man']

data\_feature\_names = ['height','age']

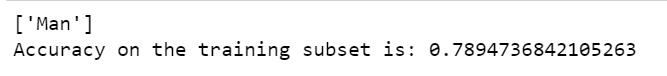
LR\_model = linear\_model.LogisticRegression()

LR\_model.fit(X, Y)

prediction = LR\_model.predict([[169,19]])

print(prediction)

print('Accuracy on the training subset is:',format(LR\_model.score(X,Y)))



**LINEAR REGRESSION**

from sklearn.linear\_model import LinearRegression

X = [[2001,5.2],[2002,5.1],[2003,5.1],[2004,4.9],[2005,5.0],[2006,5.1],[2007,5.4],[2008,5.6],[2009,5.9],[2010,5.8],[2011,6.2],

[2012,6.0],[2013,5.8],[2014,6.1],[2015,6.4],[2016,6.6],[2017,6.6],[2018,6.8],[2019,6.85],[2020,5.9]]

Y = [2.5,2.52,2.54,2.48,2.52,2.54,2.55,2.7,2.9,3.2,3.16,3.28,3.2,3.15,3.26,3.29,3.17,3.25,3.29,3.18]

len(X), len(Y)

LinR\_model = LinearRegression()

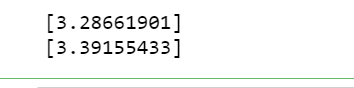
LinR\_model.fit(X, Y)

prediction = LinR\_model.predict([[2021,6.1]])

print(prediction)

prediction\_2022 = LinR\_model.predict([[2022,6.4]])

print(prediction\_2022)



**NAIVE BAYES ALGORITHM**

from sklearn.naive\_bayes import GaussianNB

import numpy as np

X = np.array([[-3,7],[1,5], [1,2], [-2,0], [2,3], [-4,0], [-1,1], [1,1], [-2,2], [2,7], [-4,1], [-2,7]])

Y = np.array([3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 4, 4])

model = GaussianNB()

model.fit(X, Y)

prediction= model.predict([[-2,0]])

print(prediction)



**DATA PRE-PROCESSING**

import pandas as pd

import numpy as np

# Create the DataFrame

data = {

"ID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],

"Name": ["Alice", "Bob", "Charlie", None, "David", "Eve", "Alice", "Frank", "Grace", "Charlie", "Charlie"],

"Age": [25, 29, 28, np.nan, 32, 35, 25, 100, 26, 28, 28],

"Salary": [50000, 60000, 55000, 70000, None, 80000, 50000, 100000, 65000, 60000, 60000],

"Department": ["HR", "IT", "IT", "HR", "IT", "Sales", "HR", "HR", "IT", "IT", "IT"],

"Join Date": ["01-01-2020", "15-05-2020", "20-03-2021", "25-11-2020", "30-08-2021", None, "01-01-2020", "18-06-2022", "11-07-2021", "20-03-2021", "20-03-2021"],

"Bonus": [5000, 6000, 5500, 7000, None, 8000, 5000, 10000, 6500, 5500, 6000]

}

df = pd.DataFrame(data)

# 1. Display Original DataFrame

print("Original DataFrame:")

print(df)

# 2. Remove rows with missing values

df\_no\_missing = df.dropna()

print("\nDataFrame after removing rows with missing values:")

print(df\_no\_missing)

# 3. Remove duplicate rows

df\_no\_duplicates = df\_no\_missing.drop\_duplicates(subset=['Name','Age'])

print("\nDataFrame after removing duplicate rows:")

print(df\_no\_duplicates)

# 4. Remove outliers based on Age and Salary

Q1\_age, Q3\_age = df\_no\_duplicates["Age"].quantile([0.25, 0.75])

Q1\_salary, Q3\_salary = df\_no\_duplicates["Salary"].quantile([0.25, 0.75])

IQR\_age = Q3\_age - Q1\_age

IQR\_salary = Q3\_salary - Q1\_salary

# Define lower and upper bounds

lower\_age = Q1\_age - 1.5 \* IQR\_age

upper\_age = Q3\_age + 1.5 \* IQR\_age

lower\_salary = Q1\_salary - 1.5 \* IQR\_salary

upper\_salary = Q3\_salary + 1.5 \* IQR\_salary

# Filter out the outliers

df\_no\_outliers = df\_no\_duplicates[

(df\_no\_duplicates["Age"] >= lower\_age) & (df\_no\_duplicates["Age"] <= upper\_age) &

(df\_no\_duplicates["Salary"] >= lower\_salary) & (df\_no\_duplicates["Salary"] <= upper\_salary)

]

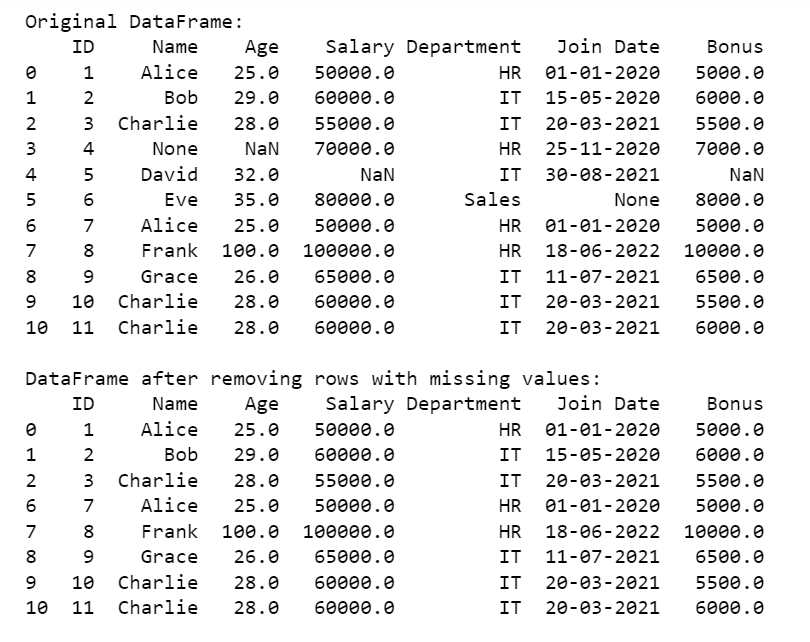
print("\nDataFrame after removing outliers:")

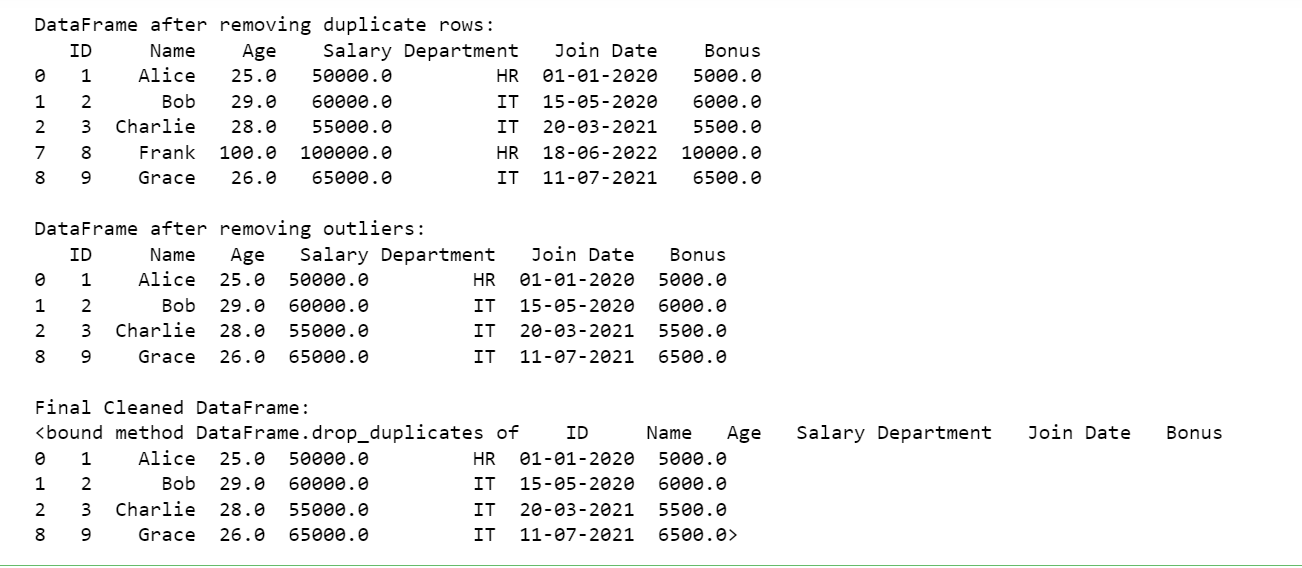
print(df\_no\_outliers)

# 5. Final Cleaned DataFrame

print("\nFinal Cleaned DataFrame:")

print(df\_no\_outliers.drop\_duplicates)

****

****