Breadth first search

def bfs(visited, graph, node):

    visited.append(node)

    queue.append(node)

    while queue:

        v = queue.pop(0)

        print(v)

        for u in graph[v]:

            if u not in visited:

                visited.append(u)

                queue.append(u)

g = {

    '1': ('2', '3'),

    '2': ('4', '5'),

    '3': ('6'),

    '4': (),

    '5': ('6'),

    '6': ()

}

visited = []

queue = []

bfs(visited, g, '1')

Depth search

def dfs(visited, graph, node):

  if node not in visited:

    print(node)

    visited.add(node)

    for n in graph[node]:

      dfs(visited,graph,n)

graph = {

  '1' : ['2','3'],

  '2' : ['4','5'],

  '3' : ['6'],

  '4' : [],

  '5' : ['6'],

  '6' : []

}

visited = set()

dfs(visited, graph, '5')

Best first

from queue import PriorityQueue

v = 14

graph = [[] for \_ in range(v)]

def add\_edge(x, y, cost):

    graph[x].append((y, cost))

    graph[y].append((x, cost))

add\_edge(0, 1, 3)

add\_edge(0, 2, 6)

add\_edge(0, 3, 5)

add\_edge(1, 4, 9)

def best\_first\_search(src, target, n):

    visited = [False] \* n

    pq = PriorityQueue()

    pq.put((0, src))

    visited[src] = True

    while not pq.empty():

        u = pq.get()[1]

        print(u)

        if u == target:

            break

        for v, c in graph[u]:

            if not visited[v]:

                visited[v] = True

                pq.put((c, v))

source = 0

target = 9

print("Best-First Search Result, starting from vertex 1:")

best\_first\_search(source, target, v)

Linear / decision/ Logistic

import pandas as pd

from sklearn.preprocessing import LabelEncoder

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

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.linear\_model import LinearRegression

df = pd.read\_csv("data.csv")

print(df)

df.head

y = df.loc[:,"diagnosis"].values

X = df.drop(["diagnosis","id","Unnamed: 32"],axis=1).values

le = LabelEncoder()

y = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,stratify=y,random\_state=0)

# Logistic

model = LogisticRegression()

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

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

print(y\_pred)

# Linear

model = LinearRegression()

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

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

y\_pred\_binary = [1 if pred > 0.5 else 0 for pred in y\_pred]

accuracy = accuracy\_score(y\_test, y\_pred\_binary)

print(f"Accuracy: {accuracy}")

# Decisiion

model = DecisionTreeClassifier(random\_state = 42)

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

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

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

tree\_train = accuracy\_score(y\_train, y\_train\_pred)

tree\_test = accuracy\_score(y\_test, y\_test\_pred)

print(f'Decision tree train/test accuracies: {tree\_train:.3f}/{tree\_test:.3f}')

K – means

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

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

print(dataset)

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

from sklearn.cluster import KMeans

wcss\_list= []

for i in range(1, 11):

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

    kmeans.fit(x)

    wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

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

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

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

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

mtp.scatter(x[y\_predict== 2, 0], x[y\_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

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

mtp.title('Clusters of customers')

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

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

mtp.legend()

mtp.show()

Naïve base :-

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred = gnb.predict(X\_test)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test,y\_pred)\*100)

Apriori

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

import pandas as pd

transactions = [

    ['bread', 'milk', 'eggs'],

    ['bread', 'butter', 'jelly'],

    ['milk', 'coffee'],

    ['bread', 'milk', 'butter'],

    ['bread', 'coffee']

]

te = TransactionEncoder()

te\_ary = te.fit(transactions).transform(transactions)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)