INDEX

	Evaluate the Accuracy of the model on test data analyze results.	
8	Implement the K-NN algorithm for classification and regression.	
	Apply the K-NN algorithm to a given dataset and predict the class or value for test data.	
	Evaluate the accuracy or error of the predictions and analyze the results.	
9	Implement the Association Rule Mining algorithm to find frequent item sets.	
	Generate association rule from frequent item sets and calculate their support and confidence.	
	Interpret and analyze the discovered association rules.	

pract1a

October 15, 2023

1 Practical 1A

2 Aim : Implement the breadth first search algorithm to solve a given problem

[5]:
```
import queue as Q
```

3 Data

[6]:
```
data = {
    'dombivali': {'thane': 80, 'diva': 53,'mulund': 105},
    'thane': {'dombivali': 80, 'mulund': 114},
    'diva':{'dombivali':75,'mulund':53},
    'mulund':{'diva':53,'thane':114,'dombivali':105,'mumbai':40},
    'mumbai':{'mulund':40}

}
```

[7]:
```
def BFS(start, goal):
    distance = {key: float('inf') for key in data}
    distance[start] = 0
    q = Q.Queue()
    q.put(start)
    while not q.empty():
        current = q.get()
        for neighbor, dist in data[current].items():
            if distance[neighbor] > distance[current] + dist:
                distance[neighbor] = distance[current] + dist
                q.put(neighbor)
    return distance[goal], [start] + [neighbor for neighbor in distance if␣
 ↪distance[neighbor] == distance[goal]]
```

[8]:
```
def main():
    start = input("Enter the starting city: ").lower()
    goal = input("Enter the target city: ").lower()
```

```
    shortest_distance, shortest_path = BFS(start, goal)

    if shortest_distance is None:
        print(f"There is no path from {start} to {goal}")
    else:
        print("Output:")
        path_string = " -> ".join(shortest_path)
        print(f"The shortest distance from {start} to {goal} is:␣
 ↪{shortest_distance}")
        print(f"The shortest path is: {path_string}")

main()
```

```
Output:
The shortest distance from diva to mumbai is: 93
The shortest path is: diva -> mumbai
```

pract1b

October 15, 2023

1 Practical 1B

2 Aim : Implement the iterative Depth first search algorithm to solve a the same problem

[15]:
```
import queue as Q
```

Data

[16]:
```
data = {
    'dombivali': {'thane': 80, 'diva': 53,'mulund': 105},
    'thane': {'dombivali': 80, 'mulund': 114},
    'diva':{'dombivali':75,'mulund':53},
    'mulund':{'diva':53,'thane':114,'dombivali':105,'mumbai':40},
    'mumbai':{'mulund':40}

}
```

3 input data and result

[17]:
```
start="dombivali"
goal="mumbai"
result=""
```

[18]:
```
def DFS(city,visitstack,startlimit,endlimit):
    global result
    found=0
    result=result+city+" "
    visitstack.append(city)
    if city==goal:
        return 1
    if startlimit==endlimit:
        return 0
    for eachcity in data[city].keys():
        if eachcity not in visitstack:
            found=DFS(eachcity,visitstack,startlimit+1,endlimit)
```

```
        if found:
            return found
```

[19]:
```
def IDDFS(city,visitstack,endlimit):
    global result
    for i in range(0,endlimit):
        print("srearching at limit:",i)
        found=DFS(city,visitstack,0,i)
        if found:
            print(found)
            break
        else:
            print("Not Found")
            print(result)
            print("      ")
            result=""
            visitstack=[]
```

[20]:
```
def main():
    visitstack=[]
    IDDFS(start,visitstack,9)
    print("IDDFS Traversal from ",start," to ",goal," is:")
    print(result)

main()
```

```
srearching at limit: 0
Not Found
dombivali

srearching at limit: 1
Not Found
dombivali thane diva mulund

srearching at limit: 2
Not Found
dombivali thane mulund diva

srearching at limit: 3
1
IDDFS Traversal from  dombivali  to  mumbai  is:
dombivali thane mulund diva mumbai
```

pract2a

October 15, 2023

1 Practical 2A

2 Aim : Implement the A* Search algorithm to solve a pathfinding problem

[2]:
```
import queue as Q
```

3 Data

[3]:
```
dist={
    "dombivali":260,"thane":194,"diva":128,"mulund":312,"mumbai":40
}

data = {
    'dombivali': {'thane': 80, 'diva': 75,'mulund': 105},
    'thane': {'dombivali': 80, 'mulund': 114},
    'diva':{'dombivali':75,'mulund':53},
    'mulund':{'diva':53,'thane':114,'dombivali':105,'mumbai':40},
    'mumbai':{'mulund':40}
}
```

4 input data and result

[4]:
```
start="dombivali"
goal="mulund"
result = ""
```

[5]:
```
def get_city(citystr):
    cities = citystr.split(",")
    hn = gn = 0
    for ctr in range(len(cities)):
        if ctr == 0:
            hn = gn + dist[cities[ctr]]
        else:
            gn = gn + data[cities[ctr]][cities[ctr - 1]]
```

```
            hn = gn + dist[cities[ctr]]
    return hn + gn
```

[6]:
```
def expand(cityq):
    global result

    tot, citystr, thiscity = cityq.get()

    if thiscity == goal:
        result = citystr + "::" + str(tot)
        return
    for city in data[thiscity]:
        cityq.put((get_city(citystr + "," + city), citystr + "," + city, city))
    expand(cityq)
```

[7]:
```
def main():
    cityq = Q.PriorityQueue()
    thiscity = start
    cityq.put((get_city(start), start, thiscity))
    expand(cityq)
    print("Output:")
    print("This A* path with the total is :")
    print(result)

main()
```

```
Output:
This A* path with the total is :
dombivali,mulund::522
```

pract2b

October 15, 2023

1 Practical 2B

2 Aim : Implement the Recursive Best-first Search algorithm to solve a same problem

3 Data

[5]:
```
dist= {
    "dombivali": 238, "thane": 194, "diva": 128, "mulund": 312, "mumbai": 40
}

data = {
    'dombivali': {'thane': 80, 'diva': 53, 'mulund': 105},
    'thane': {'dombivali': 80, 'mulund': 114},
    'diva': {'dombivali': 75, 'mulund': 53},
    'mulund': {'diva': 53, 'thane': 114, 'dombivali': 105, 'mumbai': 40},
    'mumbai': {'mulund': 40}
}
```

4 input data and result

[6]:
```
start = "dombivali"
goal = "mumbai"
result = ""
```

[7]:
```
def get_city(citystr):
    cities = citystr.split(",")
    hn = gn = 0
    for ctr in range(len(cities)):
        if ctr == 0:
            hn = dist[cities[ctr]]  # Heuristic value for the starting city
        else:
            gn = gn + data[cities[ctr]][cities[ctr - 1]]
            hn = dist[cities[ctr]]
    return hn + gn
```

[8]:
```
def printout(cityq):
    for item in cityq:
        print(item)
```

[9]:
```
def recursive_bfs(cityq):
    global result
    if not cityq:
        return

    tot, citystr, thiscity = cityq.pop(0)

    if thiscity == goal:
        result = citystr + "::" + str(tot)
        return

    print("Expanded city -------", thiscity)

    for city in data[thiscity]:
        new_citystr = citystr + "," + city
        new_fn = get_city(new_citystr)
        cityq.append((new_fn, new_citystr, city))

    printout(cityq)
    recursive_bfs(cityq)
```

[10]:
```
def main():
    cityq = [(get_city(start), start, start)]
    recursive_bfs(cityq)
    print(result)

main()
```

```
Expanded city ------- dombivali
(274, 'dombivali,thane', 'thane')
(203, 'dombivali,diva', 'diva')
(417, 'dombivali,mulund', 'mulund')
Expanded city ------- thane
(203, 'dombivali,diva', 'diva')
(417, 'dombivali,mulund', 'mulund')
(398, 'dombivali,thane,dombivali', 'dombivali')
(506, 'dombivali,thane,mulund', 'mulund')
Expanded city ------- diva
(417, 'dombivali,mulund', 'mulund')
(398, 'dombivali,thane,dombivali', 'dombivali')
(506, 'dombivali,thane,mulund', 'mulund')
(366, 'dombivali,diva,dombivali', 'dombivali')
(440, 'dombivali,diva,mulund', 'mulund')
```

```
Expanded city ------- mulund
(398, 'dombivali,thane,dombivali', 'dombivali')
(506, 'dombivali,thane,mulund', 'mulund')
(366, 'dombivali,diva,dombivali', 'dombivali')
(440, 'dombivali,diva,mulund', 'mulund')
(286, 'dombivali,mulund,diva', 'diva')
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
Expanded city ------- dombivali
(506, 'dombivali,thane,mulund', 'mulund')
(366, 'dombivali,diva,dombivali', 'dombivali')
(440, 'dombivali,diva,mulund', 'mulund')
(286, 'dombivali,mulund,diva', 'diva')
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
Expanded city ------- mulund
(366, 'dombivali,diva,dombivali', 'dombivali')
(440, 'dombivali,diva,mulund', 'mulund')
(286, 'dombivali,mulund,diva', 'diva')
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
Expanded city ------- dombivali
(440, 'dombivali,diva,mulund', 'mulund')
(286, 'dombivali,mulund,diva', 'diva')
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
(402, 'dombivali,diva,dombivali,thane', 'thane')
```

```
(331, 'dombivali,diva,dombivali,diva', 'diva')
(545, 'dombivali,diva,dombivali,mulund', 'mulund')
Expanded city ------- mulund
(286, 'dombivali,mulund,diva', 'diva')
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
(402, 'dombivali,diva,dombivali,thane', 'thane')
(331, 'dombivali,diva,dombivali,diva', 'diva')
(545, 'dombivali,diva,dombivali,mulund', 'mulund')
(309, 'dombivali,diva,mulund,diva', 'diva')
(436, 'dombivali,diva,mulund,thane', 'thane')
(471, 'dombivali,diva,mulund,dombivali', 'dombivali')
(208, 'dombivali,diva,mulund,mumbai', 'mumbai')
Expanded city ------- diva
(413, 'dombivali,mulund,thane', 'thane')
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
(402, 'dombivali,diva,dombivali,thane', 'thane')
(331, 'dombivali,diva,dombivali,diva', 'diva')
(545, 'dombivali,diva,dombivali,mulund', 'mulund')
(309, 'dombivali,diva,mulund,diva', 'diva')
(436, 'dombivali,diva,mulund,thane', 'thane')
(471, 'dombivali,diva,mulund,dombivali', 'dombivali')
(208, 'dombivali,diva,mulund,mumbai', 'mumbai')
(449, 'dombivali,mulund,diva,dombivali', 'dombivali')
(523, 'dombivali,mulund,diva,mulund', 'mulund')
Expanded city ------- thane
(448, 'dombivali,mulund,dombivali', 'dombivali')
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
```

```
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
(402, 'dombivali,diva,dombivali,thane', 'thane')
(331, 'dombivali,diva,dombivali,diva', 'diva')
(545, 'dombivali,diva,dombivali,mulund', 'mulund')
(309, 'dombivali,diva,mulund,diva', 'diva')
(436, 'dombivali,diva,mulund,thane', 'thane')
(471, 'dombivali,diva,mulund,dombivali', 'dombivali')
(208, 'dombivali,diva,mulund,mumbai', 'mumbai')
(449, 'dombivali,mulund,diva,dombivali', 'dombivali')
(523, 'dombivali,mulund,diva,mulund', 'mulund')
(537, 'dombivali,mulund,thane,dombivali', 'dombivali')
(645, 'dombivali,mulund,thane,mulund', 'mulund')
Expanded city ------- dombivali
(185, 'dombivali,mulund,mumbai', 'mumbai')
(434, 'dombivali,thane,dombivali,thane', 'thane')
(363, 'dombivali,thane,dombivali,diva', 'diva')
(577, 'dombivali,thane,dombivali,mulund', 'mulund')
(375, 'dombivali,thane,mulund,diva', 'diva')
(502, 'dombivali,thane,mulund,thane', 'thane')
(537, 'dombivali,thane,mulund,dombivali', 'dombivali')
(274, 'dombivali,thane,mulund,mumbai', 'mumbai')
(402, 'dombivali,diva,dombivali,thane', 'thane')
(331, 'dombivali,diva,dombivali,diva', 'diva')
(545, 'dombivali,diva,dombivali,mulund', 'mulund')
(309, 'dombivali,diva,mulund,diva', 'diva')
(436, 'dombivali,diva,mulund,thane', 'thane')
(471, 'dombivali,diva,mulund,dombivali', 'dombivali')
(208, 'dombivali,diva,mulund,mumbai', 'mumbai')
(449, 'dombivali,mulund,diva,dombivali', 'dombivali')
(523, 'dombivali,mulund,diva,mulund', 'mulund')
(537, 'dombivali,mulund,thane,dombivali', 'dombivali')
(645, 'dombivali,mulund,thane,mulund', 'mulund')
(484, 'dombivali,mulund,dombivali,thane', 'thane')
(413, 'dombivali,mulund,dombivali,diva', 'diva')
(627, 'dombivali,mulund,dombivali,mulund', 'mulund')
dombivali,mulund,mumbai::185
```

pract3

October 15, 2023

1 Practical 3

2 Aim : Decision Tree Learning

3 Implement the Decision Tree Learning algorithm to build a decision tree for given dataset.

4 Evaluate the Accuracy and Effectiveness of the decision tree on test data.

5 Vizualize and interpret the generated decision tree.

[17]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
import matplotlib.pyplot as plt
import graphviz
```

[18]:
```
df=pd.read_csv("D:/dataset/PlayTennis.csv")
df
```

[18]:

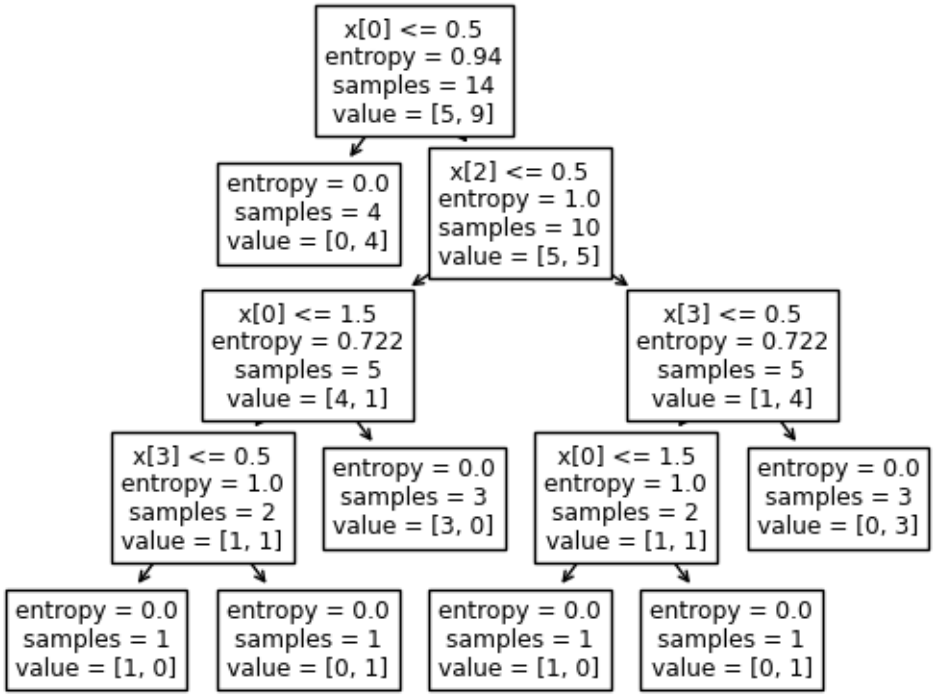
	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

[19]:
```
le=LabelEncoder()

df['Outlook']=le.fit_transform(df['Outlook'])
df['Temperature']=le.fit_transform(df['Temperature'])
df['Humidity']=le.fit_transform(df['Humidity'])
df['Wind']=le.fit_transform(df['Wind'])
df['Play Tennis']=le.fit_transform(df['Play Tennis'])

df
```

[19]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

[20]:
```
x=df.drop(['Play Tennis'],axis=1)
y=df['Play Tennis']

clf=tree.DecisionTreeClassifier(criterion="entropy")
clf=clf.fit(x,y)
tree.plot_tree(clf)
plt.show()
```



[21]:
```
accuracy=clf.score(x,y)
print("Accuracy :",accuracy)
```

```
Accuracy : 1.0
```

[22]:
```
x_pred=clf.predict(x)
x_pred==y
```

[22]:
```
0     True
1     True
2     True
3     True
4     True
5     True
6     True
7     True
8     True
9     True
10    True
11    True
12    True
13    True
Name: Play Tennis, dtype: bool
```

[23]:
```
dot_data=tree.export_graphviz(clf,out_file=None)
graph=graphviz.Source(dot_data)
print(graph)
```

```
digraph Tree {
node [shape=box, fontname="helvetica"] ;
edge [fontname="helvetica"] ;
0 [label="x[0] <= 0.5\nentropy = 0.94\nsamples = 14\nvalue = [5, 9]"] ;
1 [label="entropy = 0.0\nsamples = 4\nvalue = [0, 4]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="x[2] <= 0.5\nentropy = 1.0\nsamples = 10\nvalue = [5, 5]"] ;
0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
3 [label="x[0] <= 1.5\nentropy = 0.722\nsamples = 5\nvalue = [4, 1]"] ;
2 -> 3 ;
4 [label="x[3] <= 0.5\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]"] ;
3 -> 4 ;
5 [label="entropy = 0.0\nsamples = 1\nvalue = [1, 0]"] ;
4 -> 5 ;
6 [label="entropy = 0.0\nsamples = 1\nvalue = [0, 1]"] ;
4 -> 6 ;
7 [label="entropy = 0.0\nsamples = 3\nvalue = [3, 0]"] ;
3 -> 7 ;
8 [label="x[3] <= 0.5\nentropy = 0.722\nsamples = 5\nvalue = [1, 4]"] ;
2 -> 8 ;
9 [label="x[0] <= 1.5\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]"] ;
8 -> 9 ;
10 [label="entropy = 0.0\nsamples = 1\nvalue = [1, 0]"] ;
9 -> 10 ;
11 [label="entropy = 0.0\nsamples = 1\nvalue = [0, 1]"] ;
9 -> 11 ;
12 [label="entropy = 0.0\nsamples = 3\nvalue = [0, 3]"] ;
8 -> 12 ;
}
```

pract4

October 15, 2023

1 Practical 4

2 Aim : implement the Feed Forward Backpropagation algorithm to train neural network

3 use given dataset to train neural network for specific task.

4 Evaluate the performance of the trained network on test data.

[6]:
```
import numpy as np
```

[7]:
```
class NeuralNetwork:

    def __init__(self):
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def train(self, training_input, training_output, training_iterations):
        for i in range(training_iterations):
            output = self.sigmoid(np.dot(training_input, self.synaptic_weights))
            error = training_output - output
            self.synaptic_weights += np.dot(training_input.T, error * output *␣
 ↪(1 - output))

    def predict(self, inputs):
        return self.sigmoid(np.dot(inputs, self.synaptic_weights))
```

[8]:
```
if __name__ == "__main__":
    neural_network = NeuralNetwork()
    print("Beginning Randomly Generated Weight:")
    print(neural_network.synaptic_weights)

    training_inputs = np.array([[0, 0, 1],
                                [1, 1, 1],
```

```
                                [1, 0, 1],
                                [0, 0, 1]])

    training_outputs = np.array([[0, 1, 1, 0]]).T

    neural_network.train(training_inputs, training_outputs, 15000)
    print("Ending Weight after training:")
    print(neural_network.synaptic_weights)

    user_inputs = np.array([float(input("User input one: ")),
                            float(input("User input two: ")),
                            float(input("User input three: "))])

    print("Considering new situation:", user_inputs)
    print("New output data:")
    result = neural_network.predict(user_inputs)
    print(result)
```

```
Beginning Randomly Generated Weight:
[[-0.23134391]
 [-0.07425989]
 [ 0.60549569]]
Ending Weight after training:
[[ 9.73965592]
 [ 1.53415317]
 [-4.93073304]]
Considering new situation: [1. 2. 3.]
New output data:
[0.12086791]
```

pract5

October 15, 2023

1 Practical 5

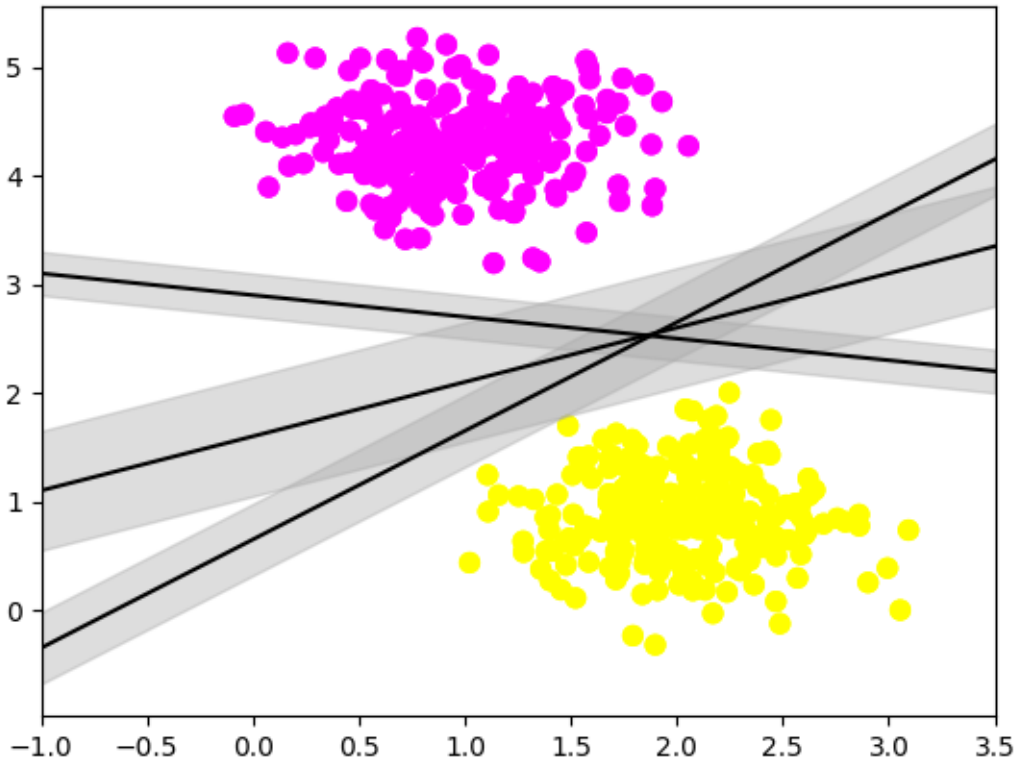
2 Aim : implement the SVM algorithm for binary classifications.

3 Train SVM model using given dataset and optimize its parameters.

4 Evaluate the performance of the SVM model on dataset and analyze result.

[ ]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
X, Y = make_blobs(n_samples=500, centers=2,
                                 random_state=0, cluster_std=0.40)

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

xfit = np.linspace(-1, 3.5)

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
        yfit = m * xfit + b
        plt.plot(xfit, yfit, '-k')
        plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
        color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5)
plt.show()
```


5
4
3
2
1
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

pract6

October 15, 2023

1 Practical 6

2 Aim : Implement the Adaboost algorithm to create an ensemble of weak classifiers.

3 Train the ensemble model on a given dataset and evaluate its performance.

4 Compare the results with individual weak classifiers.

[1]:
```
import pandas as pd
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
```

[2]:
```
path="./pima-indians-diabetes.csv"
names=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigr
df=pd.read_csv(path,names=names)
df
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
..	…	…	…	…	…	…	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1

```
3                       0.167   21         0
4                       2.288   33         1
..                        …   …        …
763                     0.171   63         0
764                     0.340   27         0
765                     0.245   30         0
766                     0.349   47         1
767                     0.315   23         0

[768 rows x 9 columns]
```

[3]:
```
array=df.values
x=array[:,0:8]
y=array[:,8]
seed=7
num_tress=30
```

[4]:
```
model=AdaBoostClassifier(n_estimators=num_tress,random_state=seed)
results=model_selection.cross_val_score(model,x,y)
results.mean()
```

[4]:
```
0.7617774382480265
```

pract7

October 15, 2023

1 Practical 7

2 Aim : Implement the Naive Bayes algorithm classification.

3 Train a Naive Bayes model using a given dataset calculate class probabilities.

4 Evaluate the Accuracy of the model on test data analyze results.

[ ]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
import seaborn
```

[36]:
```
df=pd.read_csv("./disease.csv")
df
```

[36]:

	Sore throat	Fever	Swollen glands	Congestion	Headache	Diagnosis
0	Yes	Yes	Yes	Yes	Yes	Strep throat
1	No	No	No	Yes	Yes	Allergy
2	Yes	Yes	No	Yes	No	Cold
3	Yes	No	Yes	No	No	Strep throat
4	No	Yes	No	Yes	No	Cold
5	No	No	No	Yes	No	Allergy
6	No	No	Yes	No	No	Strep throat
7	Yes	No	No	Yes	Yes	Allergy
8	No	Yes	No	Yes	Yes	Cold
9	Yes	Yes	No	Yes	Yes	Cold

[37]:
```
df.head()
```

[37]:

	Sore throat	Fever	Swollen glands	Congestion	Headache	Diagnosis
0	Yes	Yes	Yes	Yes	Yes	Strep throat
1	No	No	No	Yes	Yes	Allergy
2	Yes	Yes	No	Yes	No	Cold
3	Yes	No	Yes	No	No	Strep throat
4	No	Yes	No	Yes	No	Cold

[38]:
```
df.tail()
```

[38]:

	Sore throat	Fever	Swollen glands	Congestion	Headache	Diagnosis
5	No	No	No	Yes	No	Allergy
6	No	No	Yes	No	No	Strep throat
7	Yes	No	No	Yes	Yes	Allergy
8	No	Yes	No	Yes	Yes	Cold
9	Yes	Yes	No	Yes	Yes	Cold

[39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sore throat     10 non-null     object
 1   Fever           10 non-null     object
 2   Swollen glands  10 non-null     object
 3   Congestion      10 non-null     object
 4   Headache        10 non-null     object
 5   Diagnosis       10 non-null     object
dtypes: object(6)
memory usage: 608.0+ bytes
```

[40]:
```
l=LabelEncoder()

df['Congestion']=l.fit_transform(df['Congestion'])
df['Diagnosis']=l.fit_transform(df['Diagnosis'])
df['Fever']=l.fit_transform(df['Fever'])
df['Headache']=l.fit_transform(df['Headache'])
df['Sore throat']=l.fit_transform(df['Sore throat'])
df['Swollen glands']=l.fit_transform(df['Swollen glands'])
df
```

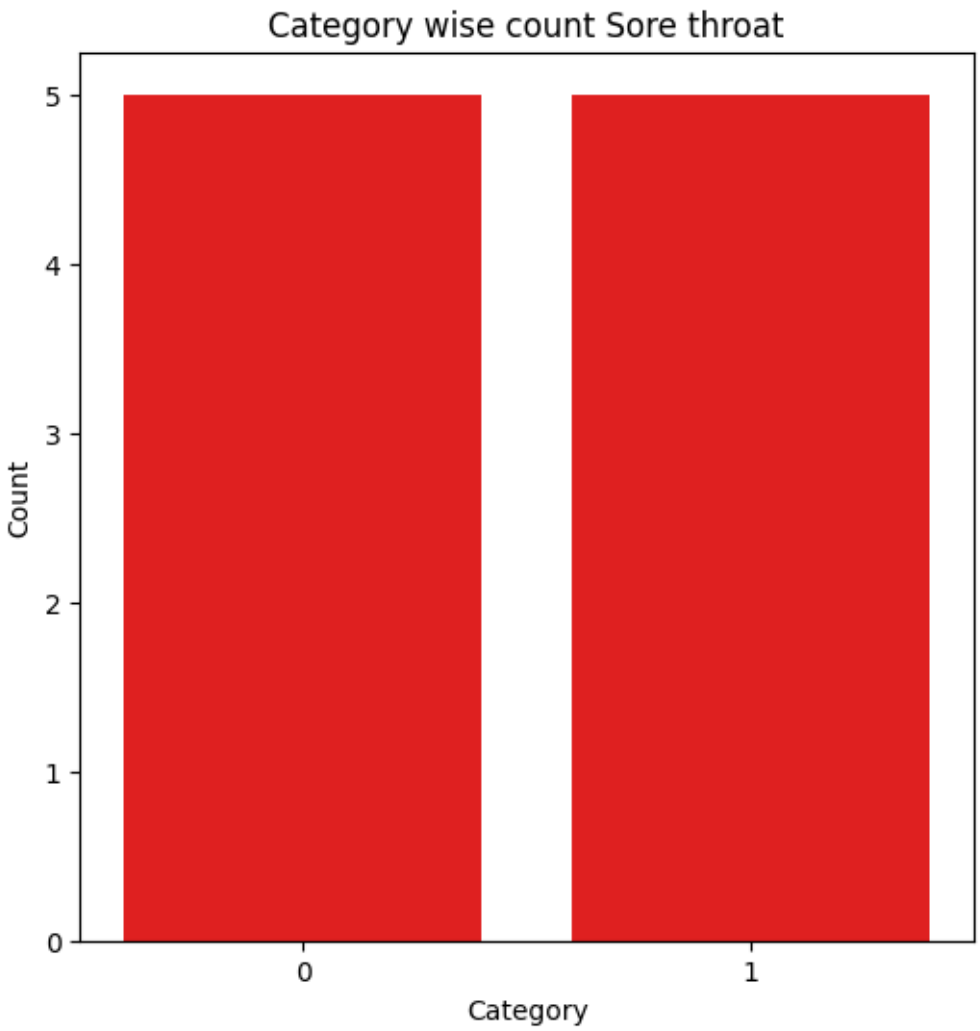
[40]:

	Sore throat	Fever	Swollen glands	Congestion	Headache	Diagnosis
0	1	1	1	1	1	2
1	0	0	0	1	1	0
2	1	1	0	1	0	1
3	1	0	1	0	0	2
4	0	1	0	1	0	1
5	0	0	0	1	0	0
6	0	0	1	0	0	2
7	1	0	0	1	1	0
8	0	1	0	1	1	1
9	1	1	0	1	1	1

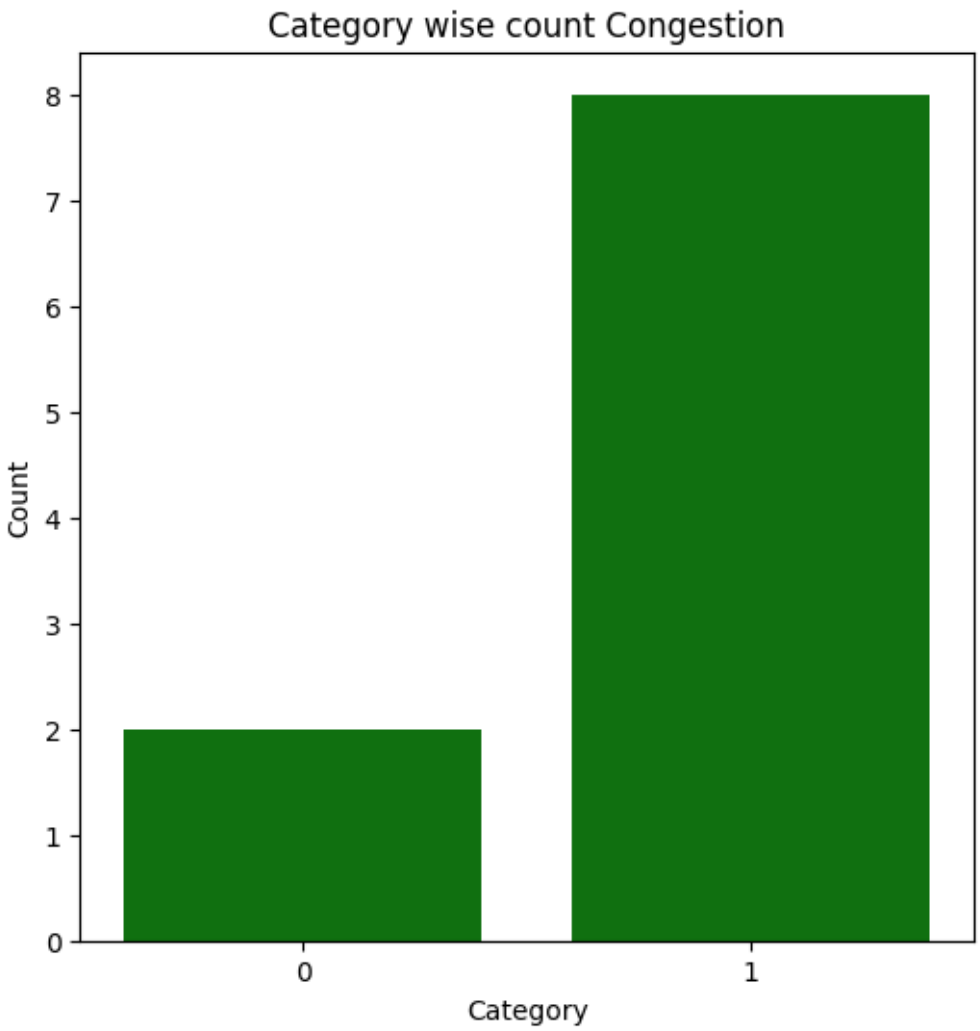
[41]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sore throat     10 non-null     int32
 1   Fever           10 non-null     int32
 2   Swollen glands  10 non-null     int32
 3   Congestion      10 non-null     int32
 4   Headache        10 non-null     int32
 5   Diagnosis       10 non-null     int32
dtypes: int32(6)
memory usage: 368.0 bytes
```

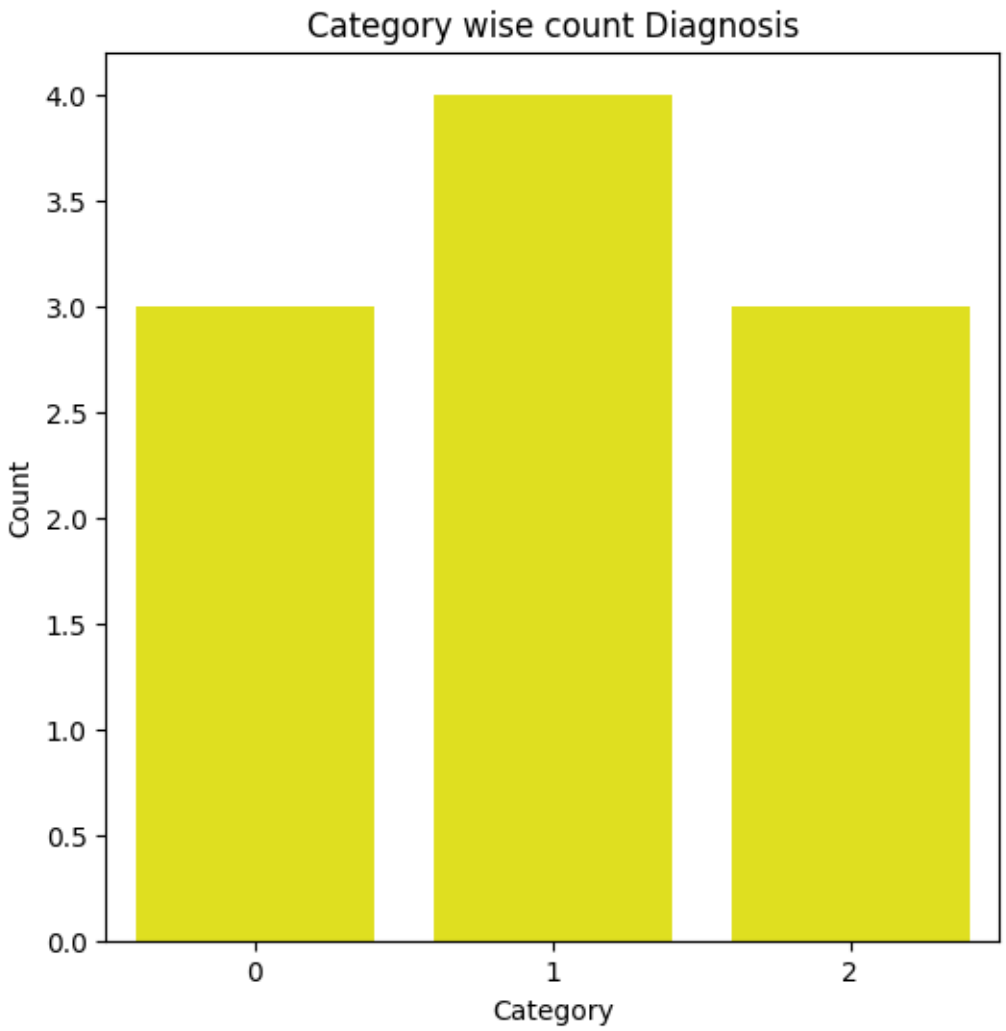
[42]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Sore throat'],data=df ,color='red')
plt.title("Category wise count Sore throat")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



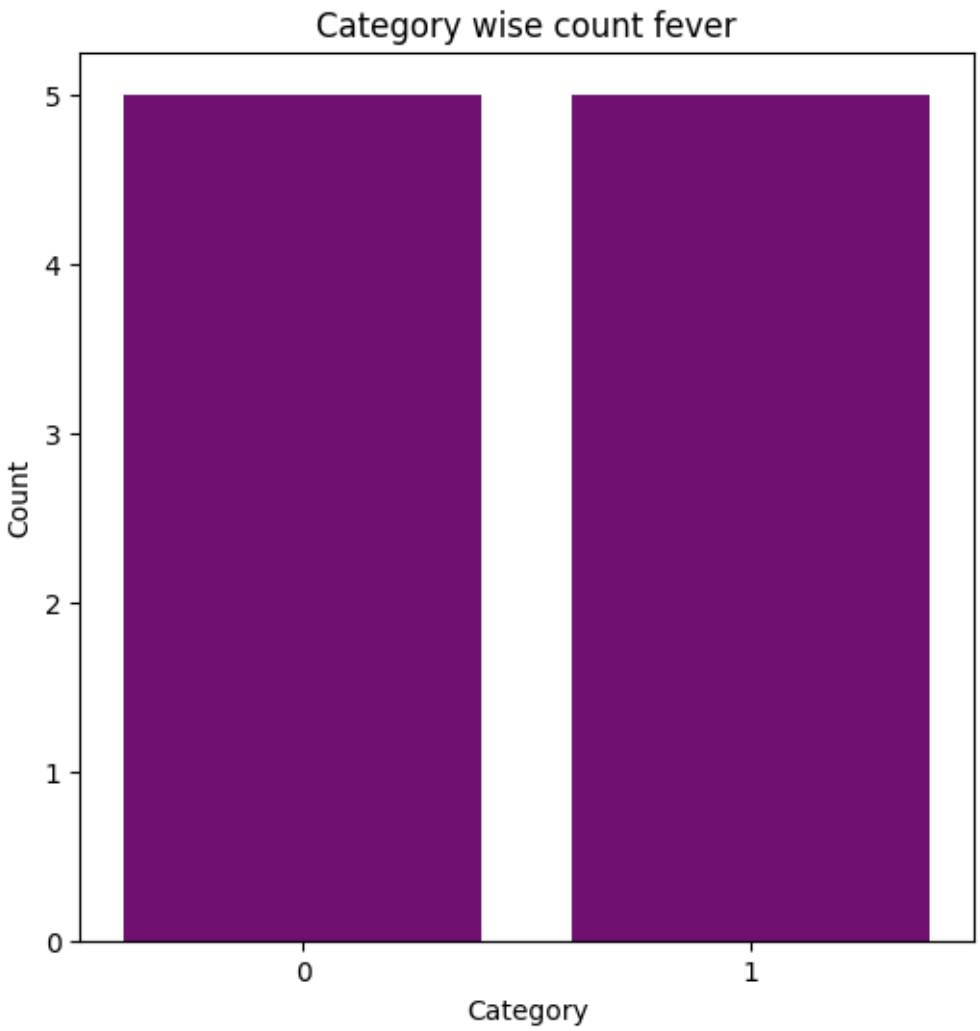
[43]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Congestion'],data=df,color='green')
plt.title("Category wise count Congestion")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



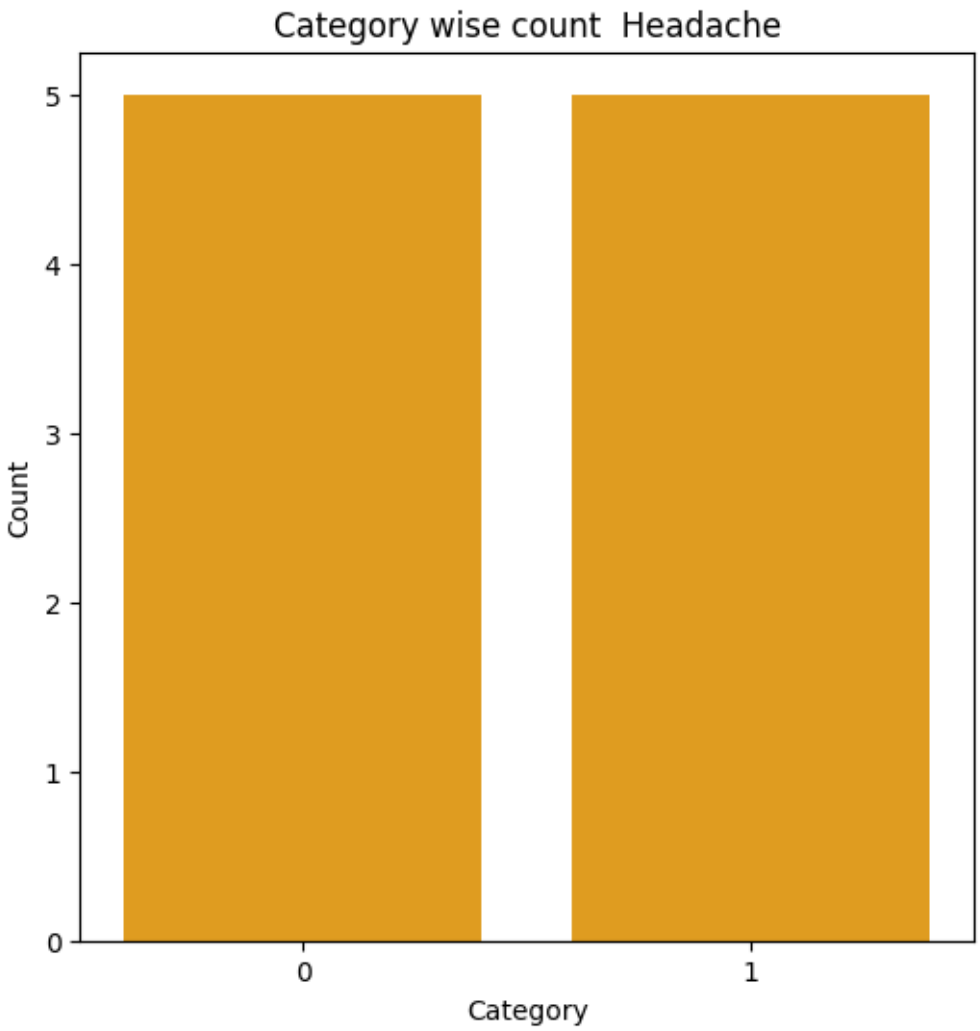
[44]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Diagnosis'],data=df,color='yellow')
plt.title("Category wise count Diagnosis")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



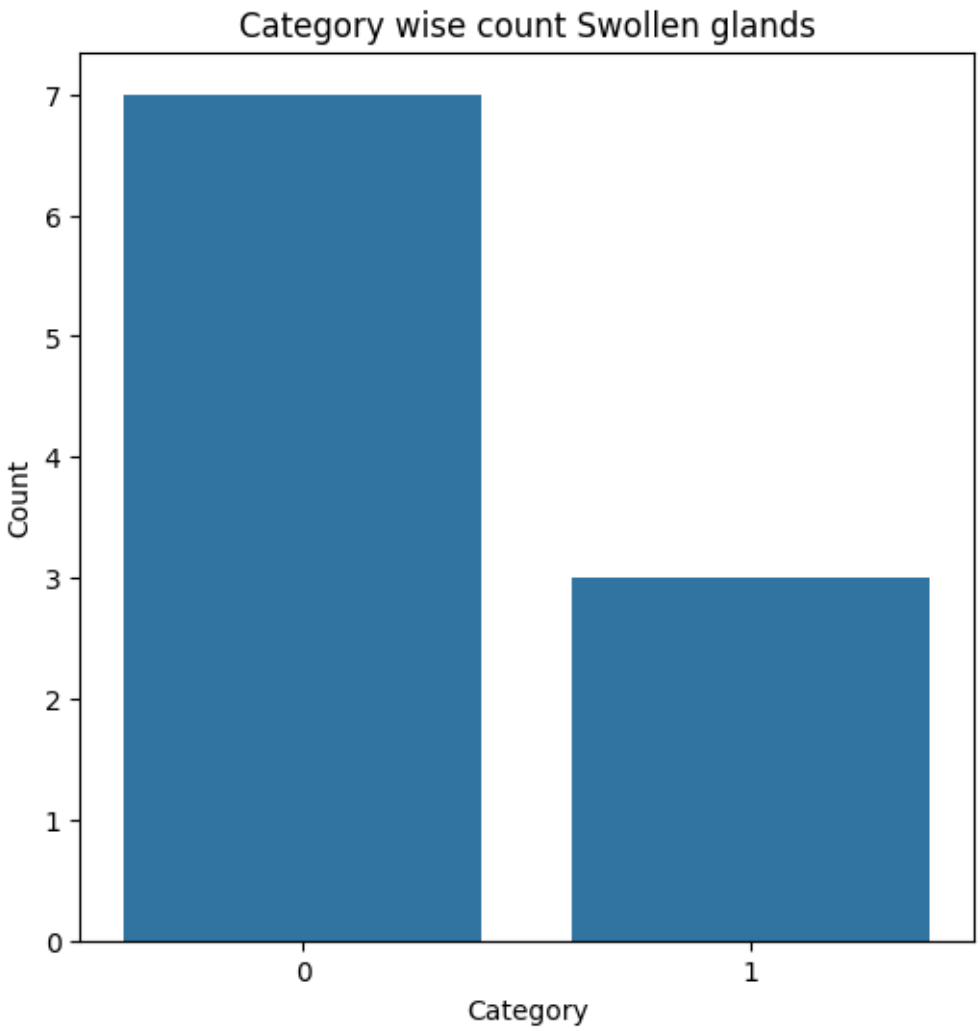
[45]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Fever'],data=df,color='purple')
plt.title("Category wise count fever")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



[46]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Headache'],data=df,color='orange')
plt.title("Category wise count  Headache")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



[47]:
```
plt.subplots(figsize=(6,6))
seaborn.countplot(x=df['Swollen glands'],data=df)
plt.title("Category wise count Swollen glands")
plt.xlabel("Category")
plt.ylabel("Count")
plt.show()
```



[48]:
```
from sklearn.naive_bayes import MultinomialNB,CategoricalNB,GaussianNB
y=df['Diagnosis']
x=df.drop("Diagnosis",axis=1)

clf=MultinomialNB()
clf.fit(x,y)
```

[48]: MultinomialNB()

[49]:
```
clf=CategoricalNB()
clf.fit(x,y)
```

[49]: CategoricalNB()

[50]:
```
clf=GaussianNB()
clf.fit(x,y)
```

[50]:
```
GaussianNB()
```

[52]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report,precision_score,recall_score,f1_score

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)


clf=MultinomialNB()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Confusion matrix :\n",confusion_matrix(y_test,y_pred))
print("Accuracy :",accuracy_score(y_test,y_pred))
print("Precision :",precision_score(y_test,y_pred))
print("Recall :",recall_score(y_test,y_pred))
print("f1 score :",f1_score(y_test,y_pred))
print("classification report :\n",classification_report(y_test,y_pred))
```

```
Confusion matrix :
 [[1 0]
 [0 1]]
Accuracy : 1.0
Precision : 1.0
Recall : 1.0
f1 score : 1.0
classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

pract8

October 15, 2023

1 Practical 8

2 Aim : Implement the K-NN algorithm for classification and regression.

3 Apply the K-NN algorithm to a given dataset and predict the class or value for test data.

4 Evaluate the accuracy or error of the predictions and analyze the results.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("ggplot")
```

[204]:
```
path="./pima-indians-diabetes.csv"
names=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigr
df=pd.read_csv(path,names=names)
df
```

[204]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
..	…	…	…	…	…	…	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1

```
1                         0.351   31        0
2                         0.672   32        1
3                         0.167   21        0
4                         2.288   33        1
..                          …   …       …
763                       0.171   63        0
764                       0.340   27        0
765                       0.245   30        0
766                       0.349   47        1
767                       0.315   23        0

[768 rows x 9 columns]
```

[205]:
```
df.shape
```

[205]:
```
(768, 9)
```

[206]:
```
df.dtypes
```

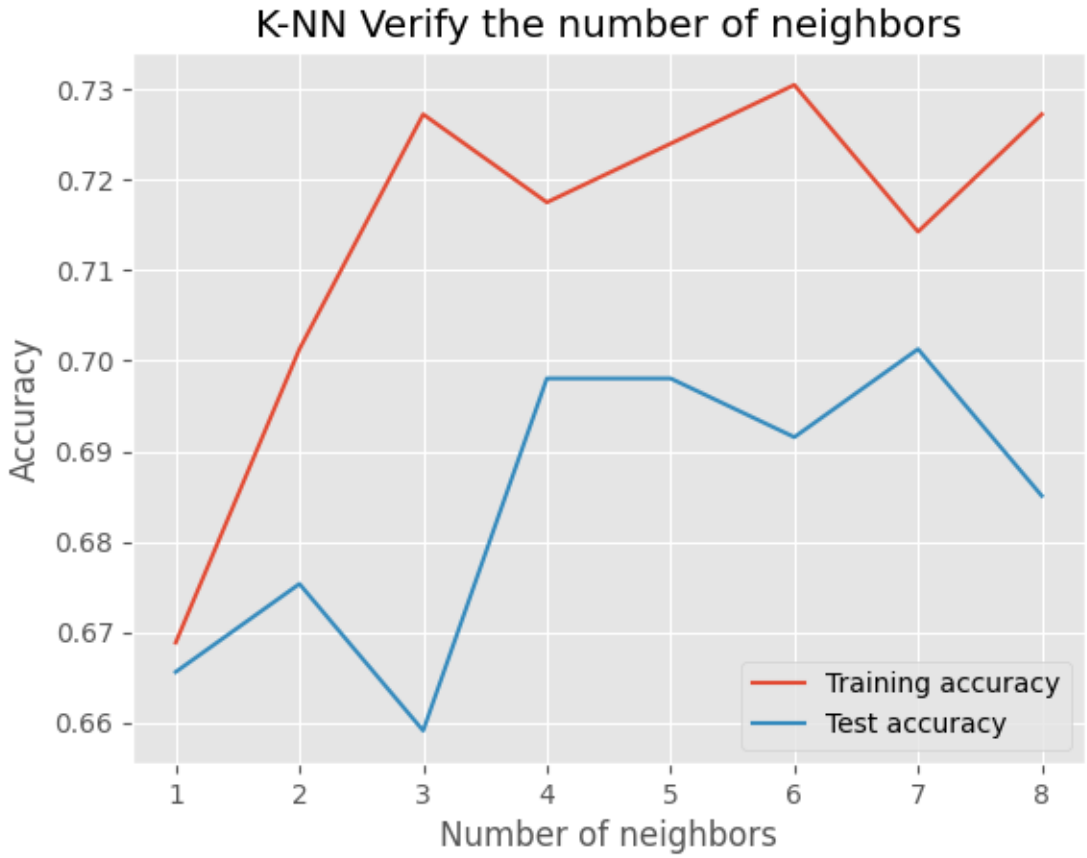
[206]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

[207]:
```
x=df.drop('Outcome',axis=1)
y=df['Outcome']
```

[208]:
```
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)
```

[209]:
```
from sklearn.neighbors import KNeighborsClassifier
neighors=np.arange(1,9)
train_accuracy=np.empty(len(neighors))
test_accuracy=np.empty(len(neighors))

for i,k in enumerate(neighors):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    train_accuracy[i]=knn.score(x_train,y_train)
    train_accuracy[i]=knn.score(x_test,y_test)
```

[210]:
```
plt.title("K-NN Verify the number of neighbors")
plt.plot(neighors,train_accuracy,label="Training accuracy")
plt.plot(neighors,test_accuracy,label="Test accuracy")
plt.legend()
plt.xlabel("Number of neighbors")
plt.ylabel("Accuracy")
plt.show()
```



[211]:
```
knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train,y_train)
```

[211]:
```
KNeighborsClassifier(n_neighbors=7)
```

[212]:
```
knn.score(x_test,y_test)
```

[212]:
```
0.7142857142857143
```

[213]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

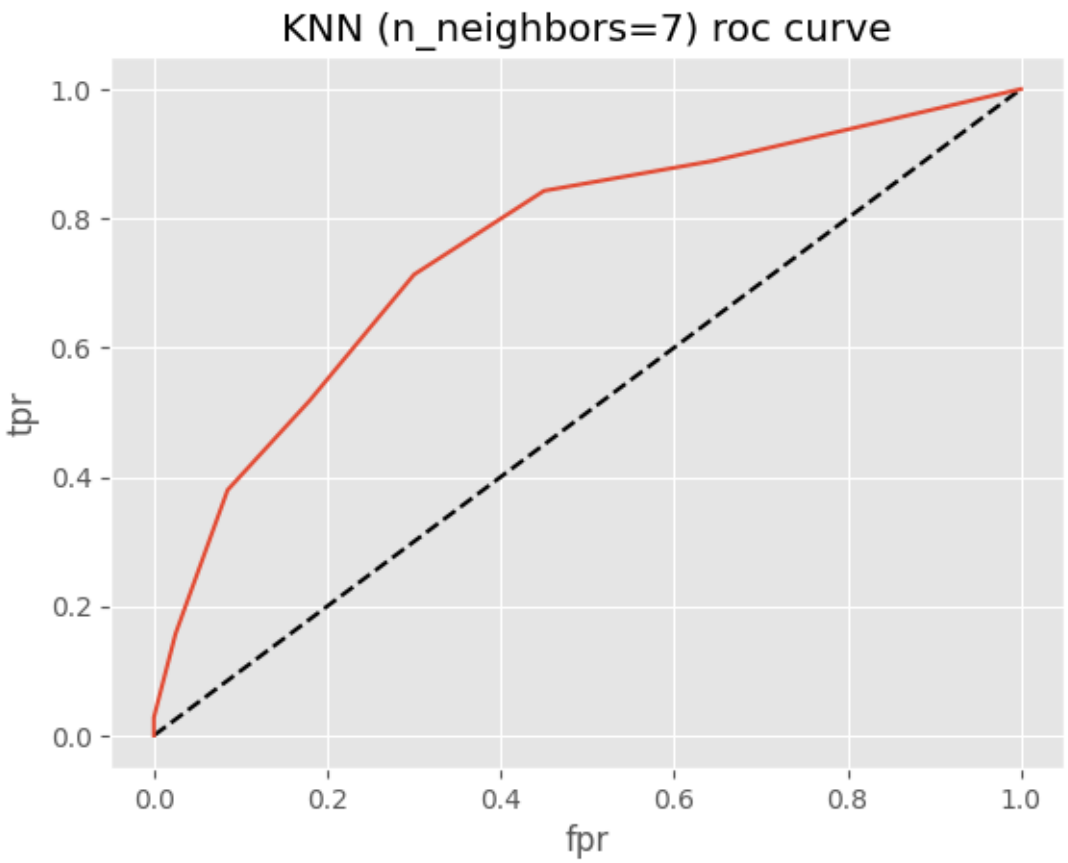
```
y_pred=knn.predict(x_test)
confusion_matrix(y_test,y_pred)
```

[213]:
```
array([[164,  36],
       [ 52,  56]], dtype=int64)
```

[214]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.82      0.79       200
           1       0.61      0.52      0.56       108

    accuracy                           0.71       308
   macro avg       0.68      0.67      0.67       308
weighted avg       0.71      0.71      0.71       308
```

[215]:
```
from sklearn.metrics import roc_curve,roc_auc_score

y_pred_prob=knn.predict_proba(x_test)[:,1]

fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label="knn")
plt.title("KNN (n_neighbors=7) roc curve")
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.show()
```



[216]:
```
roc_auc_score(y_test,y_pred_prob)
```

[216]:
```
0.7557407407407407
```

[217]:
```
from numpy import array
from sklearn.model_selection import GridSearchCV

param_grid={"n_neighbors":np.arange(1,50)}

knn=KNeighborsClassifier()
knn_cv=GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x,y)
GridSearchCV(cv=5,estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors':
 ↪array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,

 ↪21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
                                                 41,42,43,44,45,46,47,48,49
                                                 ])})
```

```
knn_cv.best_score_
```

[217]: 0.7578558696205755

[218]:
```
knn_cv.best_params_
```

[218]: {'n_neighbors': 14}

pract9

October 15, 2023

1 Practical 9

2 Aim : Implement the Association Rule Mining algorithm to find frequent itemsets.

3 Generate association rule from frequent itemsets and calculate their support and confidence.

4 Interpret and analyze the discovered association rules.

[ ]:
```
import pandas as pd
```

5 Loding data

[202]:
```
df=pd.read_csv("./Groceries_dataset.csv")
df
df.head()
```

[202]:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

[203]:
```
# Any null Values
df.isnull().any()
```

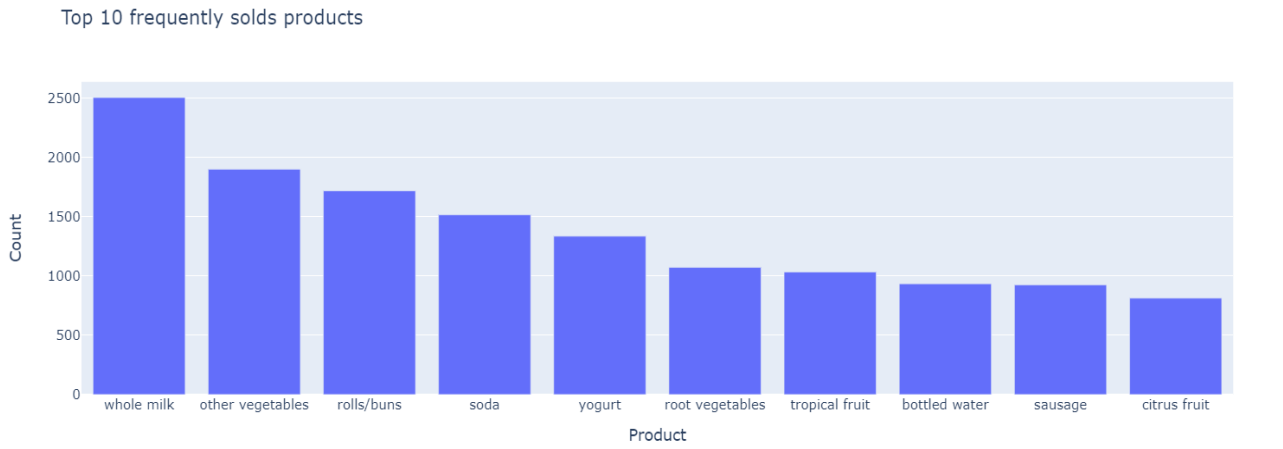
[203]:
```
Member_number      False
Date               False
itemDescription    False
dtype: bool
```

[204]:
```
allproducts=df['itemDescription'].unique()
print("Total product :",len(allproducts))
```

Total product : 167

[205]:
```
import plotly.graph_objects as go

def distribution_plot(x,y,name=None,xaxis=None,yaxis=None):
    fig=go.Figure([go.Bar(x=x,y=y)])
    fig.update_layout(
        title_text=name,
        xaxis_title=xaxis,
        yaxis_title=yaxis
    )

    fig.show()
```

[206]:
```
x=df['itemDescription'].value_counts()
x=x.sort_values(ascending=False)
x=x[:10]

distribution_plot(x=x.index,y=x.values,yaxis="Count",xaxis="Product",name="Top 10 frequently solds products")
```



#One hot representation of products purchased

[207]:
```
one_hot=pd.get_dummies(df['itemDescription'])
df.drop('itemDescription',inplace=True,axis=1)
df=df.join(one_hot)
df.head()
```

[207]:

	Member_number	Date	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	...
0	1808	21-07-2015	False	False	False	False	False	False	False	False	...
1	2552	05-01-2015	False	False	False	False	False	False	False	False	...
2	2300	19-09-2015	False	False	False	False	False	False	False	False	...
3	1187	12-12-2015	False	False	False	False	False	False	False	False	...
4	3037	01-02-2015	False	False	False	False	False	False	False	False	...

5 rows × 169 columns

#Transactions

```
records=df.groupby(['Member_number','Date'])[allproducts[:]].apply(sum)
records=records.reset_index()[allproducts]

def get_pname(x):
    for product in allproducts:
        if x[product]>0:
            x[product]=product
    return x
records=records.apply(get_pname,axis=1)
records.head(10)
```

[208]:

	tropical fruit	whole milk	pip fruit	other vegetables	rolls/buns	\
0	0	whole milk	0	0	0	
1	0	whole milk	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	
5	0	0	0	0	0	
6	0	whole milk	0	0	rolls/buns	
7	0	whole milk	0	0	0	
8	0	0	0	0	0	
9	0	0	0	0	0	

	pot plants	citrus fruit	beef	frankfurter	chicken	...	flower (seeds)	rice	\
0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	...	0	0	
2	0	0	0	0	0	...	0	0	
3	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	...	0	0	
5	0	0	0	frankfurter	0	...	0	0	
6	0	0	0	0	0	...	0	0	
7	0	0	0	0	0	...	0	0	
8	0	0	beef	0	0	...	0	0	
9	0	0	0	frankfurter	0	...	0	0	

	tea	salad	dressing	specialty	vegetables	pudding	powder	ready soups	\
0	0		0		0		0	0	
1	0		0		0		0	0	
2	0		0		0		0	0	
3	0		0		0		0	0	
4	0		0		0		0	0	
5	0		0		0		0	0	
6	0		0		0		0	0	
7	0		0		0		0	0	
8	0		0		0		0	0	
9	0		0		0		0	0	

	make up remover	toilet cleaner	preservation products
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0

[10 rows x 167 columns]

[209]:
```
print("Total transactions :",len(records))
```

Total transactions : 14963

[210]:
```
#remove zero

x=records.values

x=[sub[~(sub==0)].tolist()
   for sub in x
   if sub[sub!=0].tolist()]
transactions=x
```

#Example transactions

[211]:
```
transactions[:10]
```

[211]:
```
[['whole milk', 'yogurt', 'sausage', 'semi-finished bread'],
 ['whole milk', 'pastry', 'salty snack'],
 ['canned beer', 'misc. beverages'],
 ['sausage', 'hygiene articles'],
```

```
['soda', 'pickled vegetables'],
['frankfurter', 'curd'],
['whole milk', 'rolls/buns', 'sausage'],
['whole milk', 'soda'],
['beef', 'white bread'],
['frankfurter', 'soda', 'whipped/sour cream']]
```

[212]:
```
from apyori import apriori

rules=apriori(transactions,min_support=0.00030,min_confidance=0.
↪05,min_left=3,min_length=2,target="rules")
associations_results=list(rules)
```

[213]:
```
for item in associations_results:

    items = [str(x) for x in item]   # Convert items to strings

    print("Rules: "+items[0] + " -> " + items[1])


    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("*********************************************")
```

```
Rules: frozenset({'Instant food products'}) -> 0.004009891064626078
Support: 0.004009891064626078
Confidence: 0.004009891064626078
Lift: 1.0
*********************************************
Rules: frozenset({'UHT-milk'}) -> 0.021386085678005748
Support: 0.021386085678005748
Confidence: 0.021386085678005748
Lift: 1.0
*********************************************
Rules: frozenset({'abrasive cleaner'}) -> 0.0014702933903628951
Support: 0.0014702933903628951
Confidence: 0.0014702933903628951
Lift: 1.0
*********************************************
Rules: frozenset({'artif. sweetener'}) -> 0.0019381140145692708
Support: 0.0019381140145692708
Confidence: 0.0019381140145692708
Lift: 1.0
*********************************************
Rules: frozenset({'baking powder'}) -> 0.008086613646995923
Support: 0.008086613646995923
```

Confidence: 0.008086613646995923
Lift: 1.0
****
Rules: frozenset({'bathroom cleaner'}) -> 0.0011361358016440553
Support: 0.0011361358016440553
Confidence: 0.0011361358016440553
Lift: 1.0
****
Rules: frozenset({'beef'}) -> 0.03395041101383412
Support: 0.03395041101383412
Confidence: 0.03395041101383412
Lift: 1.0
****
Rules: frozenset({'berries'}) -> 0.021787074784468355
Support: 0.021787074784468355
Confidence: 0.021787074784468355
Lift: 1.0
****
Rules: frozenset({'beverages'}) -> 0.016574216400454454
Support: 0.016574216400454454
Confidence: 0.016574216400454454
Lift: 1.0
****
Rules: frozenset({'bottled beer'}) -> 0.04531176903027468
Support: 0.04531176903027468
Confidence: 0.04531176903027468
Lift: 1.0
****
Rules: frozenset({'bottled water'}) -> 0.06068301811134131
Support: 0.06068301811134131
Confidence: 0.06068301811134131
Lift: 1.0
****
Rules: frozenset({'brandy'}) -> 0.0025395976742631824
Support: 0.0025395976742631824
Confidence: 0.0025395976742631824
Lift: 1.0
****
Rules: frozenset({'brown bread'}) -> 0.03762614448974136
Support: 0.03762614448974136
Confidence: 0.03762614448974136
Lift: 1.0
****
Rules: frozenset({'butter'}) -> 0.03522020985096572
Support: 0.03522020985096572
Confidence: 0.03522020985096572
Lift: 1.0
****