T.Y.B.Sc. COMPUTER SCIENCE

SEMESTER: V

Lab Manual

Subject Code: USCSP501

Subject Name: Artificial Intelligence

Course Writer:

Prof. Hasan Phudinawala

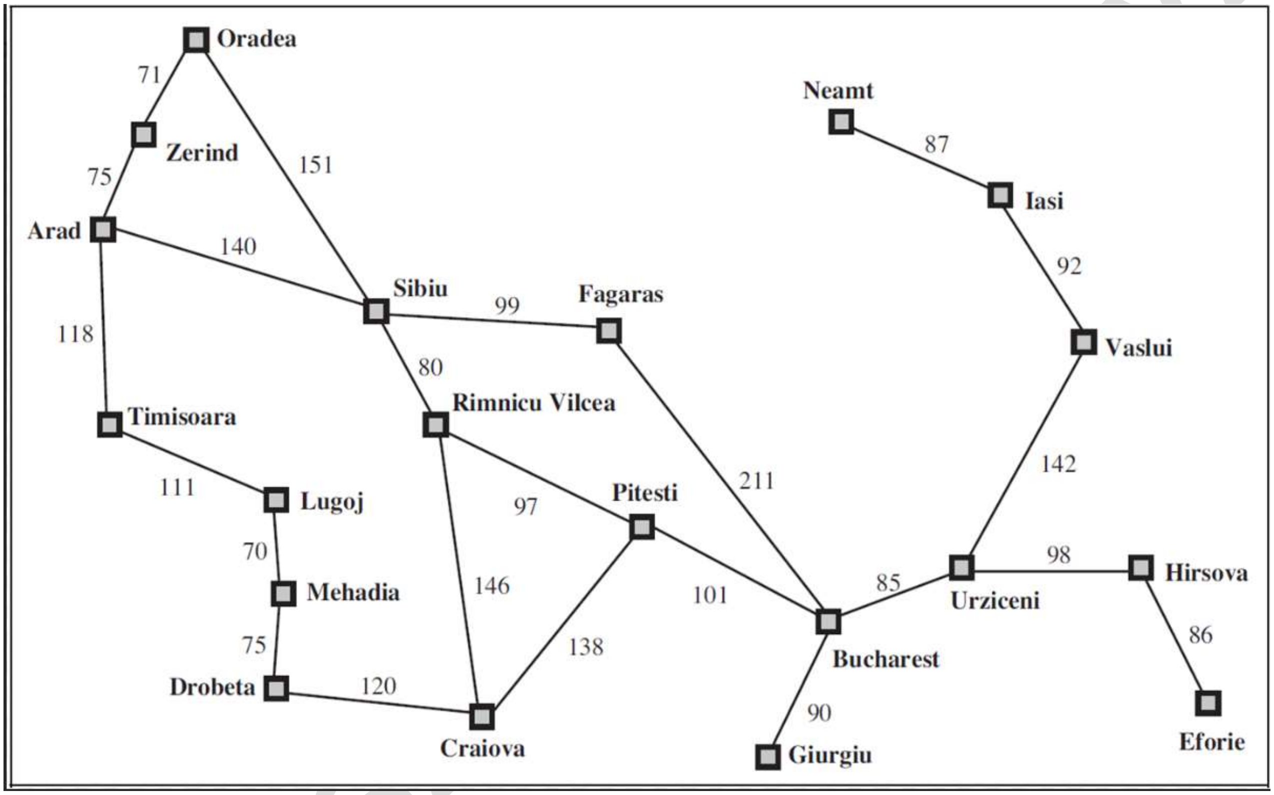
PRACTICAL 1

A. **Aim: Implement Breadth First Search Algorithm**

Dataset: RMP.py File

Requirement: RMP.py, Python IDLE

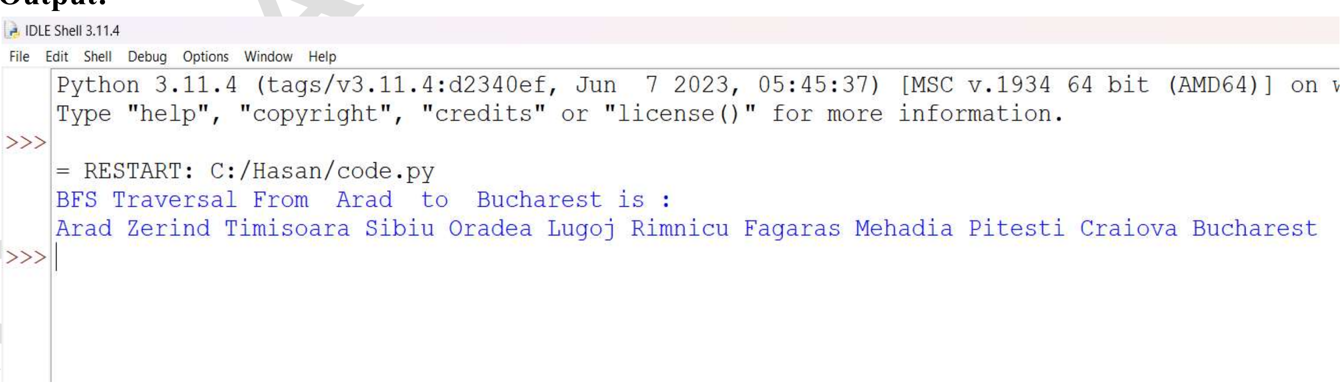
Diagram:



A. Code: Implement the Breadth First Search algorithm to solve a given problem.

```
import queue as Q
from RMP import dict_gn
start = 'Arad'
goal = "Bucharest"
result=''
def BFS(city,cityq,visitedq):
    global result
    if city==start:
        result = result + "" + city
    for eachcity in dict_gn[city].keys():
        if eachcity==goal:
            result = result + " " + eachcity
            return
        if eachcity not in cityq.queue and eachcity not in visitedq.queue:
            cityq.put(eachcity)
            result = result + " " + eachcity
    visitedq.put(city)
    BFS(cityq.get(),cityq,visitedq)
def main():
    cityq = Q.Queue()
    visitedq = Q.Queue()
    BFS(start,cityq,visitedq)
    print("BFS Traversal From ", start," to " , goal, "is :")
    print(result)
main()
```

Output:



```
IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on w
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Hasan/code.py
BFS Traversal From  Arad  to  Bucharest is :
Arad Zerind Timisoara Sibiu Oradea Lugoj Rimnicu Fagaras Mehadia Pitesti Craiova Bucharest
>>>
```

B. Code: Implement the Iterative Depth First Search algorithm to solve the same problem.

```
import queue as Q
from RMP import dict_gn
start = "Arad"
goal = "Bucharest"
result = ""

def DLS(city,visitedstack,startlimit,endlimit):
    global result
    found = 0
    result = result + city + " "
    visitedstack.append(city)
    if city == goal:
        return 1
    if startlimit == endlimit:
        return 0
    for eachcity in dict_gn[city].keys():
        if eachcity not in visitedstack:
            found = DLS(eachcity,visitedstack,startlimit+1,endlimit)
        if found:
            return found

def IDDFS(city,visitedstack,endlimit):
    global result
    for i in range(0,endlimit):
        print("Seaching at Limit:", i)
        found = DLS(city,visitedstack, 0 , i)
        if found:
            print("Found")
            break
        else:
            print("Not Found!")
            print(result)
            print("______")
            result=""
            visitedstack = []
```

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

Output:

```
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: C:\Hasan\code.py
    Seaching at Limit: 0
    Not Found!
    Arad
    ______
    Seaching at Limit: 1
    Not Found!
    Arad Zerind Timisoara Sibiu
    ______
    Seaching at Limit: 2
    Not Found!
    Arad Zerind Oradea Timisoara Lugoj Sibiu Rimnicu Fagaras
    ______
    Seaching at Limit: 3
    Not Found!
    Arad Zerind Oradea Sibiu Timisoara Lugoj Mehadia
    ______
    Seaching at Limit: 4
    Not Found!
    Arad Zerind Oradea Sibiu Rimnicu Fagaras Timisoara Lugoj Mehadia Drobeta
    ______
    Seaching at Limit: 5
    Found
    IDDFS Traversal from  Arad  to  Bucharest  is:
    Arad Zerind Oradea Sibiu Rimnicu Pitesti Craiova Fagaras Bucharest
>>>
```

PRACTICAL 2

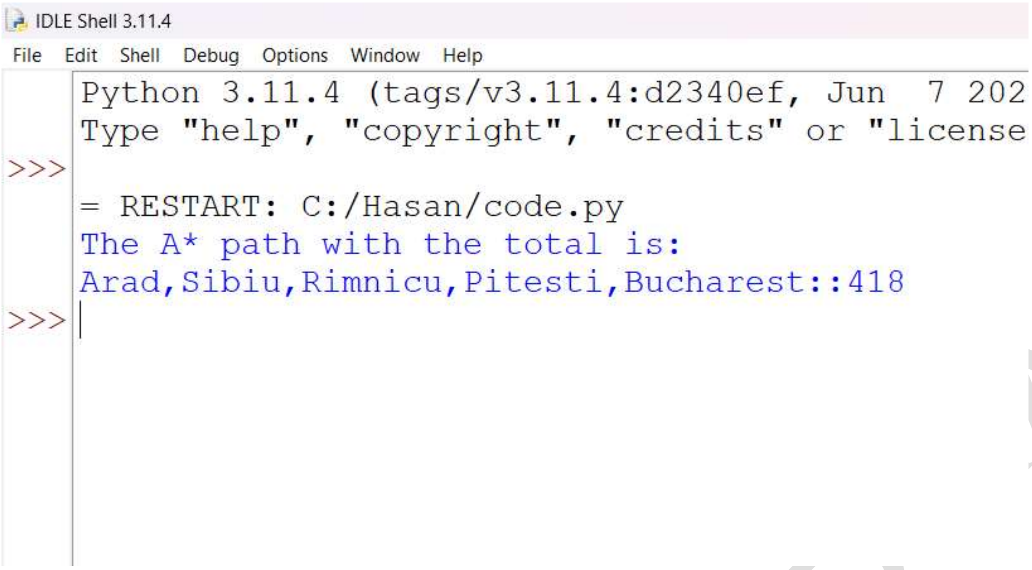
AIM: A* Search and Recursive Best-First Search

Dataset: [RMP.py File]

Code: Implement the A* Search algorithm for solving a pathfinding problem.

```
import queue as Q
from RMP import dict_gn
from RMP import dict_hn

start = 'Arad'
goal = 'Bucharest'
result = ''

def get_fn(citystr):
    cities=citystr.split(",")
    hn=gn=0
    for ctr in range(0, len(cities)-1):
        gn=gn+dict_gn[cities[ctr]][cities[ctr+1]]
    hn=dict_hn[cities[len(cities)-1]]
    return(hn+gn)
def expand(cityq):
    global result
    tot, citystr, thiscity=cityq.get()
    if thiscity==goal:
        result=citystr+"::"+str(tot)
        return
    for cty in dict_gn[thiscity]:
        cityq.put((get_fn(citystr+","+cty),citystr+","+cty,cty))
    expand(cityq)
def main():
    cityq=Q.PriorityQueue()
    thiscity=start
    cityq.put((get_fn(start),start,thiscity))
    expand(cityq)
    print("The A* path with the total is: ")
    print(result)
main()
```

Output:



Code: Implement the Recursive Best-First Search algorithm for the same problem.

```
import queue as Q
from RMP import dict_gn
from RMP import dict_hn

start = 'Arad'
goal = 'Bucharest'
result = ''
def get_fn(citystr):
    cities=citystr.split(",")
    hn=gn=0
    for ctr in range(0, len(cities)-1):
        gn=gn+dict_gn[cities[ctr]][cities[ctr+1]]
    hn=dict_hn[cities[len(cities)-1]]
    return(hn+gn)
def printout(cityq):
    for i in range(0, cityq.qsize()):
        print(cityq.queue[i])
def expand(cityq):
    global result
    tot, citystr, thiscity = cityq.get()
    nexttot = 999
    if not cityq.empty():
        nexttot,nextcitystr,nextthiscity=cityq.queue[0]
    if thiscity== goal and tot < nexttot:
```

```
        result = citystr + "::" + str(tot)
        return
    print("Expaded city ---------", thiscity)
    print("second best f(n)---------", nexttot)
    tempq = Q.PriorityQueue()
    for cty in dict_gn[thiscity]:
        tempq.put((get_fn(citystr+','+cty), citystr+','+cty, cty))
    for ctr in range(1,3):
        ctrtot, ctrcitystr ,ctrthiscity = tempq.get()
        if ctrtot < nexttot:
            cityq.put((ctrtot, ctrcitystr,ctrthiscity))
        else:
            cityq.put((ctrtot, citystr, thiscity))
            break
    printout(cityq)
    expand(cityq)

def main():
    cityq=Q.PriorityQueue()
    thiscity=start
    cityq.put((999, "NA", "NA"))
    cityq.put((get_fn(start), start, thiscity))
    expand(cityq)
    print(result)
main()
```

Output:

```
>>>
= RESTART: C:\Hasan\code.py
Expaded city --------- Arad
second best f(n)--------- 999
(393, 'Arad,Sibiu', 'Sibiu')
(999, 'NA', 'NA')
(447, 'Arad,Timisoara', 'Timisoara')
Expaded city --------- Sibiu
second best f(n)--------- 447
(413, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(415, 'Arad,Sibiu,Fagaras', 'Fagaras')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Rimnicu
second best f(n)--------- 415
(415, 'Arad,Sibiu,Fagaras', 'Fagaras')
(417, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Fagaras
second best f(n)--------- 417
(417, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Rimnicu
second best f(n)--------- 447
(417, 'Arad,Sibiu,Rimnicu,Pitesti', 'Pitesti')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(526, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
Expaded city --------- Pitesti
second best f(n)--------- 447
(418, 'Arad,Sibiu,Rimnicu,Pitesti,Bucharest', 'Bucharest')
(447, 'Arad,Timisoara', 'Timisoara')
(607, 'Arad,Sibiu,Rimnicu,Pitesti', 'Pitesti')
(526, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(999, 'NA', 'NA')
Arad,Sibiu,Rimnicu,Pitesti,Bucharest::418
>>>
```

PRACTICAL NO: 3

Aim: Implement the decision tree learning algorithm to build a decision tree for a given dataset. Evaluate the accuracy and efficiency on the test data set.

Implementing Decision Tree using Scikit Learn

This notebook is a reference notebook to a blog, Decision Tree for Beginers.

In [1]:
```
#numpy and pandas initialization
import numpy as np
import pandas as pd
```

In [2]:
```
#Loading the PlayTennis data
PlayTennis = pd.read_csv("../input/PlayTennis.csv")
```

In [3]:
```
PlayTennis
```

Out[3]:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

It is easy to implement Decision Tree with numerical values. We can convert all the non numerical values into numerical values using LabelEncoder

In [4]:
```
from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])
```

In [5]:
```
PlayTennis
```

Out[5]:

	outlook	temp	humidity	windy	play
0	2	1	0	0	0
1	2	1	0	1	0
2	0	1	0	0	1
3	1	2	0	0	1
4	1	0	1	0	1
5	1	0	1	1	0
6	0	0	1	1	1
7	2	2	0	0	0
8	2	0	1	0	1
9	1	2	1	0	1
10	2	2	1	1	1
11	0	2	0	1	1
12	0	1	1	0	1
13	1	2	0	1	0

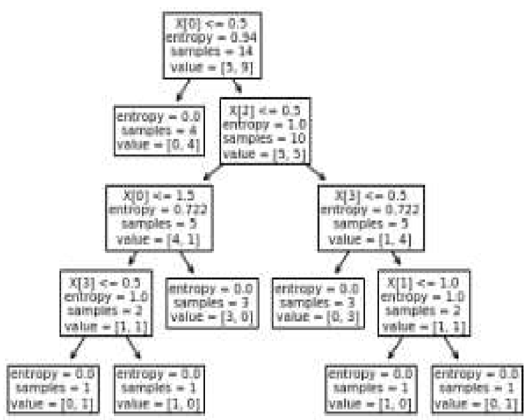
- Lets split the training data and its coresponding prediction values.
- y - holds all the decisions.
- X - holds the training data.

In [6]:
```
y = PlayTennis['play']
X = PlayTennis.drop(['play'],axis=1)
```

In [7]:
```
# Fitting the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(X, y)
```

In [8]:
```
# We can visualize the tree using tree.plot_tree
tree.plot_tree(clf)
```

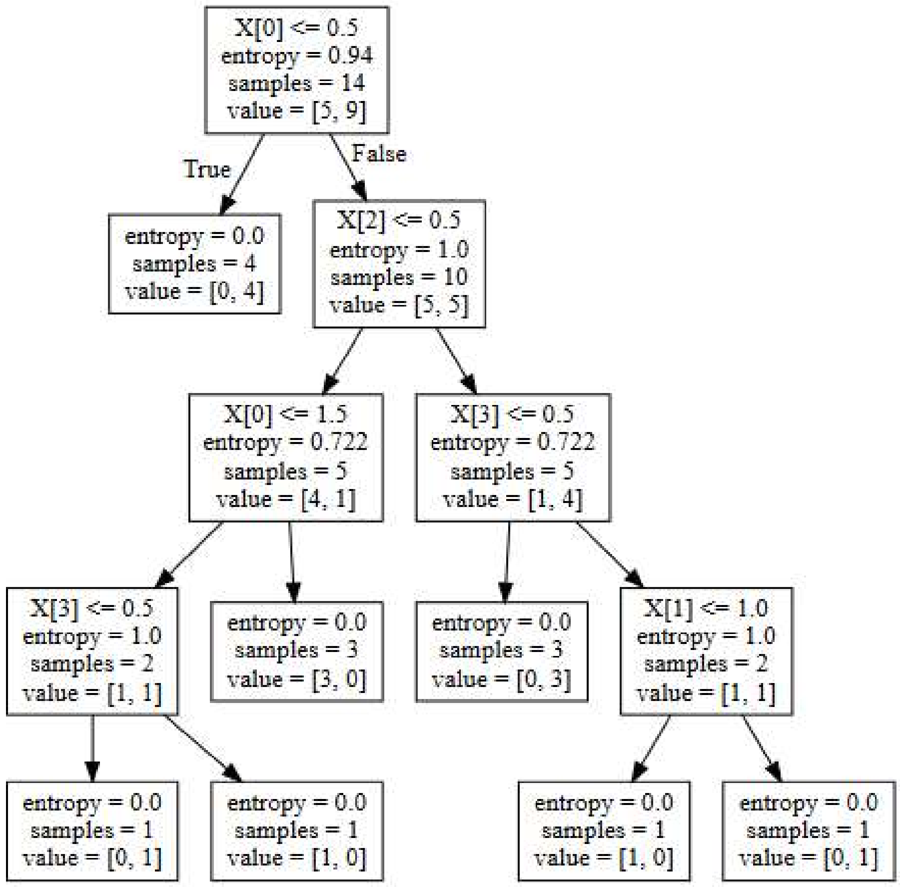
```
Out[8]: [Text(133.92000000000002, 195.696, 'X[0] <= 0.5\nentropy = 0.94\nsamples = 14\n
value = [5, 9]'),
 Text(100.44000000000001, 152.208, 'entropy = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(167.40000000000003, 152.208, 'X[2] <= 0.5\nentropy = 1.0\nsamples = 10\nv
alue = [5, 5]'),
 Text(100.44000000000001, 108.72, 'X[0] <= 1.5\nentropy = 0.722\nsamples = 5\nv
alue = [4, 1]'),
 Text(66.96000000000001, 65.232, 'X[3] <= 0.5\nentropy = 1.0\nsamples = 2\nvalu
e = [1, 1]'),
 Text(33.480000000000004, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(100.44000000000001, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(133.92000000000002, 65.232, 'entropy = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(234.36, 108.72, 'X[3] <= 0.5\nentropy = 0.722\nsamples = 5\nvalue = [1,
4]'),
 Text(200.88000000000002, 65.232, 'entropy = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(267.84000000000003, 65.232, 'X[1] <= 1.0\nentropy = 1.0\nsamples = 2\nval
ue = [1, 1]'),
 Text(234.36, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(301.32000000000005, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [0,
1]')]
```



GraphViz gives a better and clearer Graph.

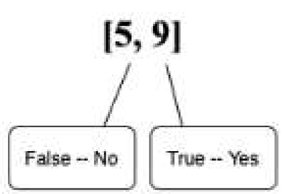
```
In [9]: import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph
```

Out[9]:



In the above graph,

- X[0] -> Outlook
- X[1] -> Temperature
- X[2] -> Humidity
- X[3] -> Wind



values

Since we dont have any data to test. we can just make the model to predict our train data.

In [10]:
```
# The predictions are stored in X_pred
X_pred = clf.predict(X)
```

In [11]:
```
# verifying if the model has predicted it all right.
X_pred == y
```

Out[11]:
```
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, dtype: bool
```

PRACTICAL NO: 4

AIM: Feed Forward Back propagation Neural Network

- Implement the Feed Forward Back propagation algorithm to train a neural network
- Use a given dataset to train the neural network for a specific task

Requirement: Python IDLE

Code:

```
from doctest import OutputChecker
import numpy as np
class NeuralNetwork():
def __init__(self):
np.random.seed()
self.synaptic_weights=2*np.random.random((3,1))-1
def sigmoid(self,x):
return 1/(1+np.exp(-x))
def sigmoid_derivative(self,x):
return x*(1-x)
def train(self,training_inputs,training_outputs,training_iterations):
for iteration in range(training_iterations):
output=self.think(training_inputs)
error = training_outputs-output
adjustments=np.dot(training_inputs.T,error*self.sigmoid_derivative(output))
self.synaptic_weights +=adjustments
def think(self,inputs):
inputs=inputs.astype(float)
output=self.sigmoid(np.dot(inputs,self.synaptic_weights))
return output

if __name__ == "__main__":
#initializing the neuron class
neural_network = NeuralNetwork()
print("Beginning Randomly Generated Weights: ")
print(neural_network.synaptic_weights)
#training data consisting of 4 examples--3 input values and 1 output
training_inputs = np.array([[0,0,1],
[1,1,1],
[1,0,1],
[0,1,1]])
```

```
training_outputs = np.array([[0,1,1,0]]).T
#training taking place
neural_network.train(training_inputs, training_outputs, 15000)
print("Ending Weights After Training: ")
print(neural_network.synaptic_weights)
user_input_one = str(input("User Input One: "))
user_input_two = str(input("User Input Two: "))
user_input_three = str(input("User Input Three: "))
print("Considering New Situation: ", user_input_one, user_input_two, user_input_three)
print("New Output data: ")
print(neural_network.think(np.array([user_input_one, user_input_two, user_input_three])))
```

Output:

```
Beginning Randomly Generated Weights:
[[0.18138631]
 [0.03957296]
 [0.68171289]]
Ending Weights After Training:
[[10.08723627]
 [-0.20745403]
 [-4.83719347]]
User Input One: 2
User Input Two: 3
User Input Three: 2
Considering New Situation:  2 3 2
New Output data:
[0.9999487]
```

PRACTICAL NO: 5

Aim: Implement the SVM algorithm for binary classification. Train a SVM Model using the given dataset. Evaluate the performance on test data and analyze the results.

Importing Libraries

In [1]:
```
from warnings import filterwarnings
filterwarnings("ignore")
```

In [2]:
```
pip install skompiler
```

```
Collecting skompiler
  Downloading SKompiler-0.6.tar.gz (45 kB)
     | 45 kB 388 kB/s
Requirement already satisfied: scikit-learn>=0.22 in /opt/conda/lib/python3.7/sit
e-packages (from skompiler) (0.23.2)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-pack
ages (from scikit-learn>=0.22->skompiler) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.7/site-pac
kages (from scikit-learn>=0.22->skompiler) (1.6.3)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.7/site-pac
kages (from scikit-learn>=0.22->skompiler) (1.19.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/s
ite-packages (from scikit-learn>=0.22->skompiler) (2.1.0)
Building wheels for collected packages: skompiler
  Building wheel for skompiler (setup.py) ... -⊠ ⊠\⊠ ⊠done
  Created wheel for skompiler: filename=SKompiler-0.6-py3-none-any.whl size=54265
sha256=940fba6a64063cef9232f47a844962bf8f3f142124b4629cea6a480f4af9acbc
  Stored in directory: /root/.cache/pip/wheels/47/1c/59/b80a730f4afd2144bad854df4
b167b812486c9d4c1bd4cf4c5
Successfully built skompiler
Installing collected packages: skompiler
Successfully installed skompiler-0.6
WARNING: Running pip as root will break packages and permissions. You should inst
all packages reliably by using venv: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split,cross_val_score,cross_val_p
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,BaseEnsemble,GradientBoostin
from sklearn.svm import SVC,LinearSVC
import time
from matplotlib.colors import ListedColormap
from xgboost import XGBRegressor
```

```
from skompiler import skompile
from lightgbm import LGBMRegressor
```

In order to see all rows and columns, we will increase max display numbers of dataframe.

In [4]:
```
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 1000)
pd.set_option('display.width', 1000)
```

Support Vector Machines - Classifier(SVM) - Linear Kernel

Illustrative example:

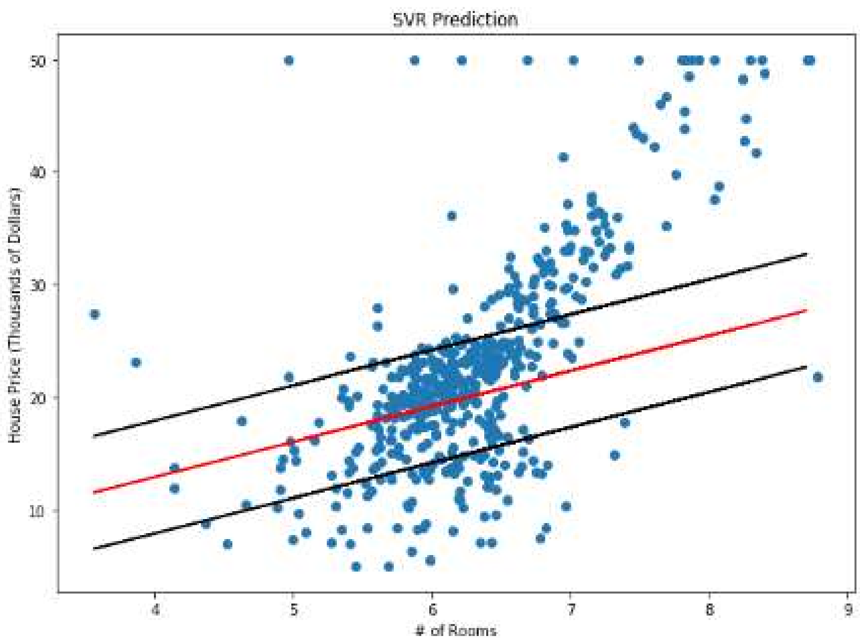


Photo is cited by:https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

In [5]:
```
df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
df.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
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	

In [6]:
```
df.shape
```
Out[6]: (768, 9)

In [7]:
```
df.describe()
```
Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

In [8]:
```
X = df.drop("Outcome",axis=1)
y= df["Outcome"] #We will predict Outcome(diabetes)
```

Now we're going to split our dataset to train and test set. We will choose almost 20% of dataset as test size.

In [9]:
```
X_train = X.iloc[:600]
X_test = X.iloc[600:]
y_train = y[:600]
y_test = y[600:]

print("X_train Shape: ",X_train.shape)
print("X_test Shape: ",X_test.shape)
print("y_train Shape: ",y_train.shape)
print("y_test Shape: ",y_test.shape)
```
```
X_train Shape:  (600, 8)
X_test Shape:  (168, 8)
y_train Shape:  (600,)
y_test Shape:  (168,)
```

In [10]:
```
support_vector_classifier = SVC(kernel="linear").fit(X_train,y_train)
```

We also could use *svm.LinearSVC()* function directly.

In [11]:
```
support_vector_classifier
```

Out[11]:
```
SVC(kernel='linear')
```

In [12]:
```
# Default C
support_vector_classifier.C
```

Out[12]:
```
1.0
```

Prediction

In [13]:
```
support_vector_classifier
```

Out[13]:
```
SVC(kernel='linear')
```

Because we are doing a classification case, we will create a **confusion matrix** in order to evaluate out model.

In [14]:
```
y_pred = support_vector_classifier.predict(X_test)
```

In [15]:
```
cm = confusion_matrix(y_test,y_pred)
```

In [16]:
```
cm
```

Out[16]:
```
array([[96, 12],
       [27, 33]])
```

- **true positive**: for correctly predicted event values.
- **false positive**: for incorrectly predicted event values.
- **true negative**: for correctly predicted no-event values.
- **false negative**: for incorrectly predicted no-event values.

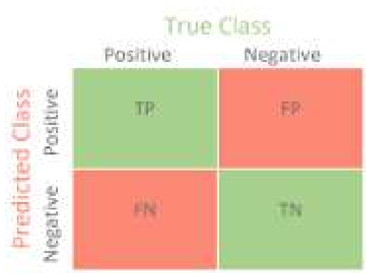


Photo is cited by here.

In [17]:
```
print("Our Accuracy is: ", (cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1]
```

```
Our Accuracy is:  0.7678571428571429
```

In [18]:
```
accuracy_score(y_test,y_pred)
```

Out[18]:
```
0.7678571428571429
```

In [19]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       108
           1       0.73      0.55      0.63        60

    accuracy                           0.77       168
   macro avg       0.76      0.72      0.73       168
weighted avg       0.76      0.77      0.76       168
```

Model Tuning & Validation

In [20]:
```
support_vector_classifier
```

Out[20]:
```
SVC(kernel='linear')
```

Now we will try to tune our model by using **K-Fold Cross Validation.**

In [21]:
```
accuracies= cross_val_score(estimator=support_vector_classifier,
                            X=X_train,y=y_train,
                            cv=10)
print("Average Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standart Deviation of Accuracies: {:.2f} %".format(accuracies.std()*100))
```

```
Average Accuracy: 77.33 %
Standart Deviation of Accuracies: 4.90 %
```

In [22]:
```
support_vector_classifier.predict(X_test)[:10]
```

Out[22]:
```
array([0, 0, 0, 1, 1, 0, 1, 0, 1, 0])
```

Now we will tune our model with GridSearch.

In [23]:
```
svm_params ={"C":np.arange(1,20)}
```

In [24]:
```
svm = SVC(kernel="linear")
svm_cv = GridSearchCV(svm,svm_params,cv=8)
```

In [25]:
```
start_time = time.time()

svm_cv.fit(X_train,y_train)

elapsed_time = time.time() - start_time

print(f"Elapsed time for Support Vector Regression cross validation: "
      f"{elapsed_time:.3f} seconds")
```

```
Elapsed time for Support Vector Regression cross validation: 4095.631 seconds
```

In [26]:
```
#best score
svm_cv.best_score_
```

```
Out[26]: 0.7716666666666667

In [27]: #best parameters
         svm_cv.best_params_

Out[27]: {'C': 2}

In [28]: svm_tuned = SVC(kernel="linear",C=2).fit(X_train,y_train)

In [29]: svm_tuned

Out[29]: SVC(C=2, kernel='linear')

In [30]: y_pred = svm_tuned.predict(X_test)

In [31]: cm = confusion_matrix(y_test,y_pred)

In [32]: cm

Out[32]: array([[96, 12],
               [27, 33]])
```

- **true positive**: for correctly predicted event values.
- **false positive**: for incorrectly predicted event values.
- **true negative**: for correctly predicted no-event values.
- **false negative**: for incorrectly predicted no-event values.

```
In [33]: print("Our Accuracy is: ", (cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1]

        Our Accuracy is:  0.7678571428571429

In [34]: accuracy_score(y_test,y_pred)

Out[34]: 0.7678571428571429

In [35]: print(classification_report(y_test,y_pred))

                  precision    recall  f1-score   support

               0       0.78      0.89      0.83       108
               1       0.73      0.55      0.63        60

        accuracy                           0.77       168
       macro avg       0.76      0.72      0.73       168
    weighted avg       0.76      0.77      0.76       168
```

PRACTICAL NO: 6

AIM: Adaboost Ensemble Learning

- Implement the Adaboost algorithm to create an ensemble of weak classifiers.
- Train the ensemble model on a given dataset and evaluate its performance
- Compare the results with individual weak classifiers

Requirement:

Code:

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
#kfold makes trees with split number.
#kfold = model_selection.KFold(n_splits=10, random_state=seed)
#n_estimators : This is the number of trees you want to build before predictions.
#Higher number of trees give you better voting optionsand perfomance performance
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
#cross_val_score method is used to calculate the accuracy of model sliced into x, y
#cross validator cv  is optional cv=kfold
results = model_selection.cross_val_score(model, X, Y)
print(results.mean())
```

Output:

```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
========================== RESTART: C:\Hasan\code.py ======
0.7617774382480265
>>> |
```

PRACTICAL NO: 7

AIM: Naive Bayes' Classifier

- Implement the Naive Bayes algorithm for classification.
- Trin a Naive Bayes' model using a given dataset and calculate class probabilities.
- Evaluate the accuracy of the model on test data and analyze the results.

Requirement: disease dataset

Code:

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import MultinomialNB, CategoricalNB, GaussianNB
        from sklearn.metrics import accuracy_score
        import seaborn as sns
```

```
In [2]: # Load the disease dataset
        df = pd.read_csv('disease.csv')
```

```
In [4]: df.head(11)
```

Out[4]:

	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

In [5]:
```
df.tail()
```

Out[5]:

	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

In [6]:
```
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
```

In [7]:
```
#Changing the Datatypes of all the columns from object to int
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Sore Throat']=le.fit_transform(df['Sore Throat'])
df['Fever']=le.fit_transform(df['Fever'])
df['Swollen Glands']=le.fit_transform(df['Swollen Glands'])
df['Congestion']=le.fit_transform(df['Congestion'])
df['Headache']=le.fit_transform(df['Headache'])
df['Diagnosis']=le.fit_transform(df['Diagnosis'])
```

In [8]:
```
df.info()
```

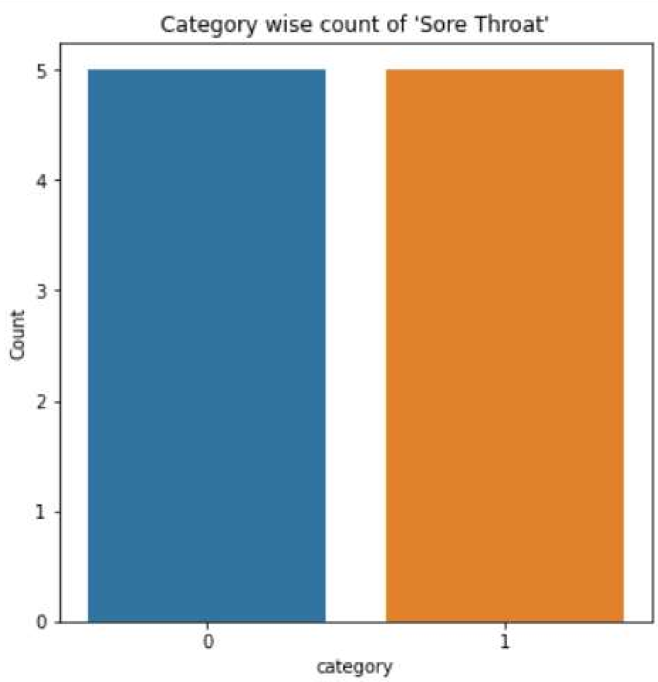
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sore Throat     10 non-null     int32
 1   Fever           10 non-null     int32
 2   Swollen Glands  10 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
```

In [9]:
```
df.head(11)
```

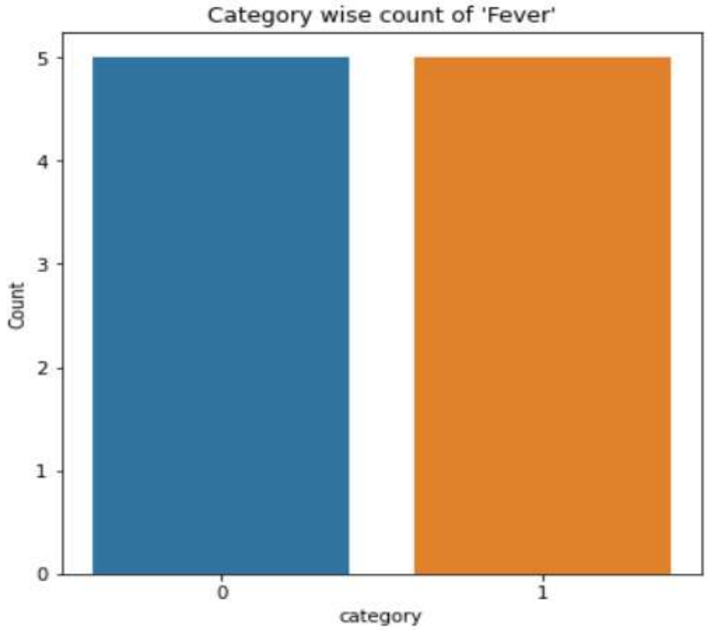
Out[9]:

	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

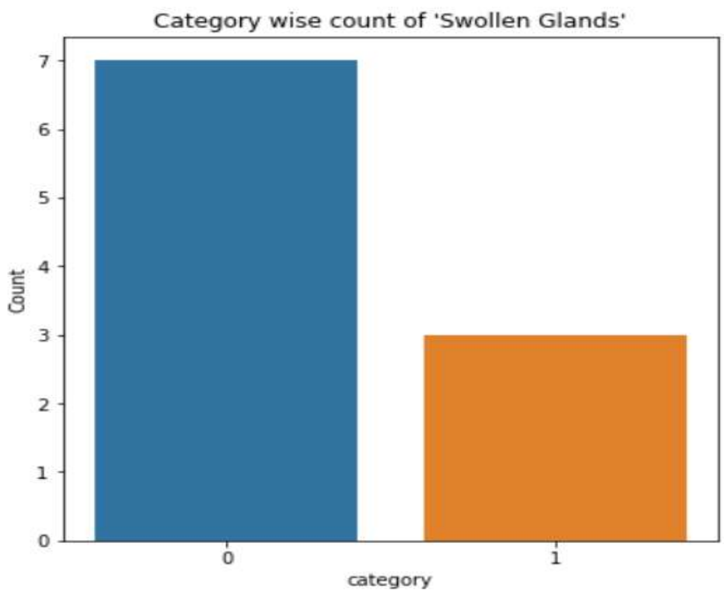
In [13]:
```
#setting the dimenions of the plot
fig,ax=plt.subplots(figsize=(6,6))
sns.countplot(x=df['Sore Throat'],data=df)
plt.title("Category wise count of 'Sore Throat'")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



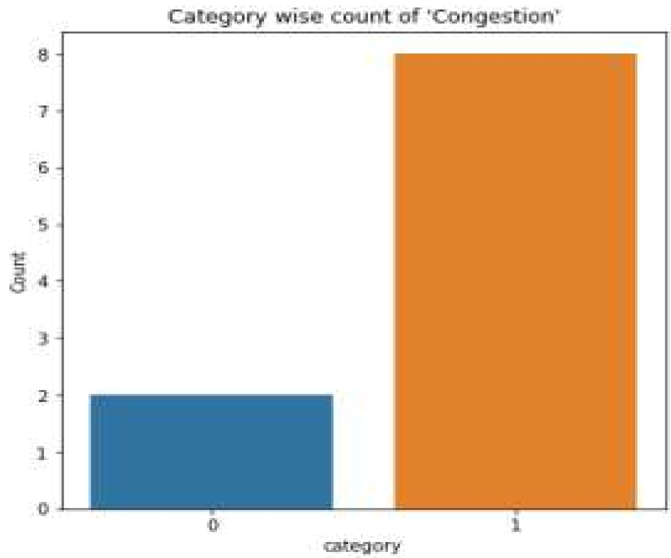
```
In [14]: fig,ax=plt.subplots(figsize=(6,6))
         sns.countplot(x=df['Fever'],data=df)
         plt.title("Category wise count of 'Fever'")
         plt.xlabel("category")
         plt.ylabel("Count")
         plt.show()
```



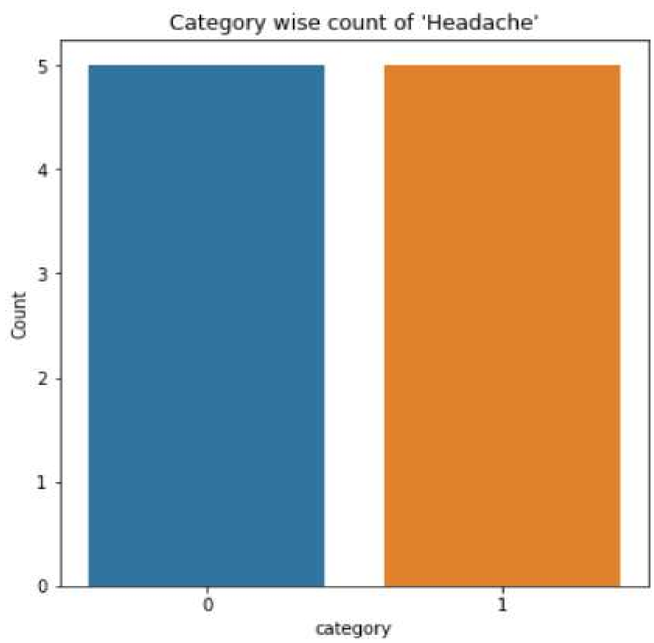

```
In [15]: fig,ax=plt.subplots(figsize=(6,6))
         sns.countplot(x=df['Swollen Glands'],data=df)
         plt.title("Category wise count of 'Swollen Glands'")
         plt.xlabel("category")
         plt.ylabel("Count")
         plt.show()
```



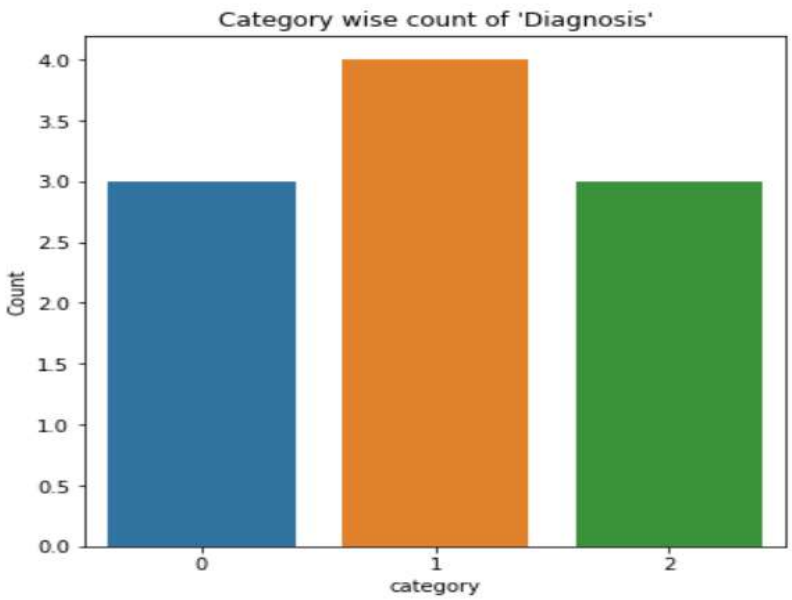
```
In [16]: fig,ax=plt.subplots(figsize=(6,6))
         sns.countplot(x=df['Congestion'],data=df)
         plt.title("Category wise count of 'Congestion'")
         plt.xlabel("category")
         plt.ylabel("Count")
         plt.show()
```




```
In [17]: fig,ax=plt.subplots(figsize=(6,6))
         sns.countplot(x=df['Headache'],data=df)
         plt.title("Category wise count of 'Headache'")
         plt.xlabel("category")
         plt.ylabel("Count")
         plt.show()
```



In [18]:
```
fig,ax=plt.subplots(figsize=(6,6))
sns.countplot(x=df['Diagnosis'],data=df)
plt.title("Category wise count of 'Diagnosis'")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



In [19]:
```
X=df.drop('Diagnosis',axis=1)
y=df['Diagnosis']
```

In [21]:
```
#Training algorithm
classifier=MultinomialNB()
classifier.fit(X,y)
```

Out[21]: MultinomialNB()

In [54]:
```
#Training algorithm
classifier=CategoricalNB()
classifier.fit(X,y)
```

Out[54]: CategoricalNB()

In [27]:
```
#Training algorithm
classifier=GaussianNB()
classifier.fit(X,y)
```

Out[27]: GaussianNB()

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
```

In [56]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

```
In [57]: classifier=MultinomialNB()
         classifier.fit(X_train,y_train)
         y_pred=classifier.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:]n              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1
           2       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
```

PRACTICAL NO: 8

Aim:- Implement the K-NN Algorithm for classification or regression.
Apply K-NN Algorithm on the given dataset & predict the class or value for test data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [2]:
```
df = pd.read_csv('C:/Users/RDNC/Desktop/diabetes.csv')
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [3]:
```
df.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
df.dtypes
```

Out[4]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [9]:
```
x= df.drop('Outcome',axis=1).values
y = df['Outcome'].values
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

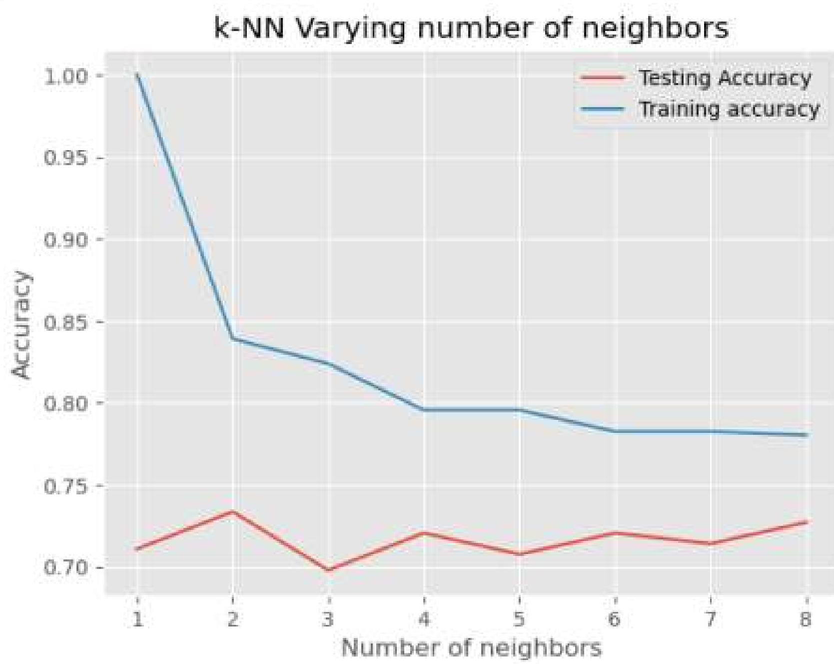
In [19]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.4,random_state=42, st
```

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
neighbors = np.arange(1,9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
#Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)
#Fit the model
    knn.fit(X_train, y_train)
#Compute accuracy on the training set
```

```
    train_accuracy[i] = knn.score(X_train, y_train)
#Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

In [25]:
```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

file:///C:/Users/Lenovo/Downloads/Practical no 8- K-NEAREST NEIGHBOUR.html 4/36



In [27]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

In [30]:
```
knn.fit(X_train,y_train)
```

Out[30]: KNeighborsClassifier(n_neighbors=7)

In [32]:
```
knn.score(X_test,y_test)
```

```
C:\Users\RDNC\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: F
utureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default
behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become False, the `axis` o
ver which the statistic is taken will be eliminated, and the value None will no longe
r be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[32]: 0.7142857142857143

In [36]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:

In [37]:
```
y_pred = knn.predict(X_test)
```

In [39]:
```
confusion_matrix(y_test,y_pred)
```

Out[39]:
```
array([[163,  43],
       [ 45,  57]], dtype=int64)
```

In [43]:
```
from sklearn.metrics import classification_report
```

In [ ]:

In [44]:
```
print(classification_report(y_test,y_pred))
```

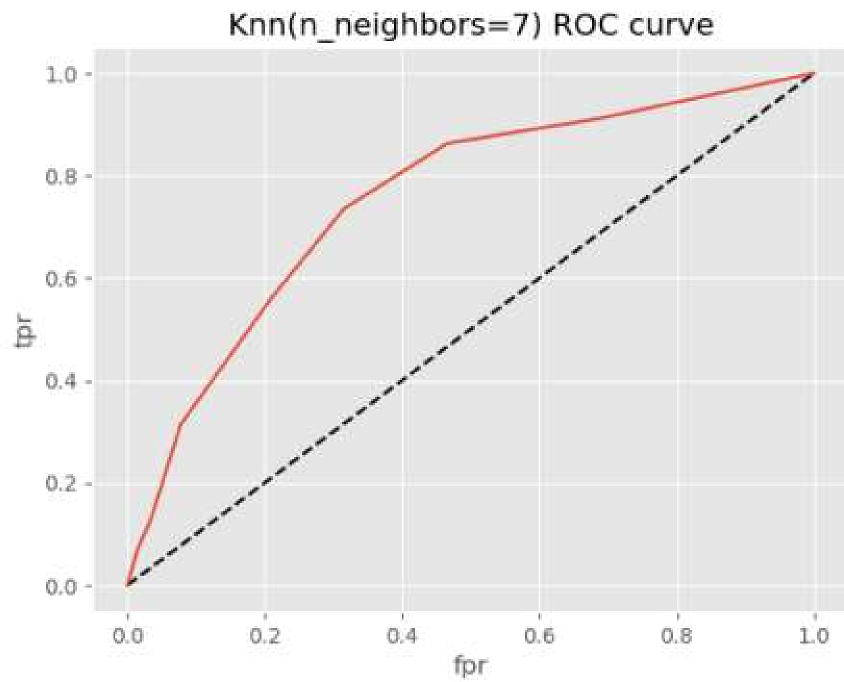
```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79       206
           1       0.57      0.56      0.56       102

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

In [46]:
```
y_pred_proba = knn.predict_proba(X_test)[:,1]
```

In [48]:
```
from sklearn.metrics import roc_curve
```

In [50]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [52]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=7) ROC curve')
plt.show()
```



In [54]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

Out[54]:
```
0.7536645726251665
```

In [56]:
```
from sklearn.model_selection import GridSearchCV
```

In [58]:
```
param_grid = {'n_neighbors':np.arange(1,50)}
```

In [61]:
```
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x,y)
```

Out[61]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 1
0, 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])})
```

In [63]:
```
knn_cv.best_score_
```

Out[63]:
```
0.7578558696205755
```

In [64]:
```
knn_cv.best_params_
```

Out[64]:
```
{'n_neighbors': 14}
```

PRACTICAL No: 9

Aim: Implement the Association Rule Mining algorithm (e.g. Apriori) to find frequent dataset. Generate association rules from the frequent item set and calculate their support.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyt
hon
# For example, here's several helpful packages to load

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fil
es under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserv
ed as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
of the current session
```

```
/kaggle/input/groceries-dataset/Groceries_dataset.csv
```

Importing libraries

In [2]:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px


try:
    import apyori
except:
    !pip install apyori

from apyori import apriori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... -□ □\□ □done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=59
74 sha256=5819b318291b268ba58838396c358d49aa0ca7e69e4568102921188619c581d4
  Stored in directory: /root/.cache/pip/wheels/cb/f6/e1/57973c631d27efd1a2
f375bd6a83b2a616c4021f24aab84080
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

Loading Dataset

In [3]:

```
df = pd.read_csv('../input/groceries-dataset/Groceries_dataset.csv', parse_dates=['Dat
e'])
df.head()
```

Out[3]:

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

Any null values

In [4]:

```
df.isnull().any()
```

Out[4]:

```
Member_number      False
Date               False
itemDescription    False
dtype: bool
```

Total Products

In [5]:

```
all_products = df['itemDescription'].unique()
print("Total products: {}".format(len(all_products)))
```

```
Total products: 167
```

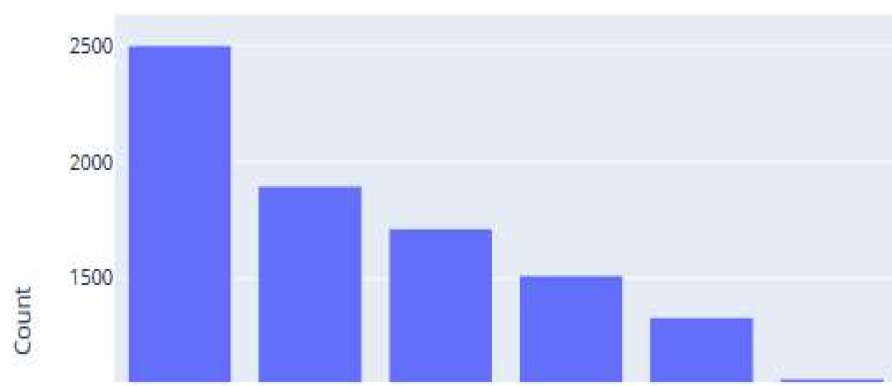
Top 10 frequently sold products

In [6]:

```
def ditribution_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()
```

In [7]:

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

ditribution_plot(x=x.index, y=x.values, yaxis="Count", xaxis="Products")
```



One-hot representation of products purchased

In [8]:

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

Out[8]:

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

5 rows × 169 columns

Transactions

Note: if a customer bought multiple products on same day, We will consider it one transaction

In [9]:

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

In [10]:

```
## Replacing non-zero values with product names
def get_Pnames(x):
    for product in all_products:
        if x[product] > 0:
            x[product] = product
    return x

records = records.apply(get_Pnames, axis=1)
records.head()
```

Out[10]:

	tropical fruit	whole milk	pip fruit	other vegetables	rolls/buns	pot plants	citrus fruit	beef	frankfurter	chicken	...
0	0	whole milk	0	0	0	0	0	0	0	0	...
1	0	whole milk	0	0	0	0	0	0	0	0	...
2	0	0	0	0	0	0	0	0	0	0	...
3	0	0	0	0	0	0	0	0	0	0	...
4	0	0	0	0	0	0	0	0	0	0	...

5 rows × 167 columns

In [11]:

```
print("total transactions: {}".format(len(records)))
```

```
total transactions: 14963
```

In [12]:

```
## Removing zeros
x = records.values
x = [sub[~(sub == 0)].tolist() for sub in x if sub[sub != 0].tolist()]
transactions = x
```

Example transactions

In [13]:

```
transactions[0:10]
```

Out[13]:

```
[['whole milk', 'pastry', 'salty snack'],
 ['whole milk', 'yogurt', 'sausage', 'semi-finished bread'],
 ['soda', 'pickled vegetables'],
 ['canned beer', 'misc. beverages'],
 ['sausage', 'hygiene articles'],
 ['whole milk', 'rolls/buns', 'sausage'],
 ['whole milk', 'soda'],
 ['frankfurter', 'soda', 'whipped/sour cream'],
 ['frankfurter', 'curd'],
 ['beef', 'white bread']]
```

Association Rules

In [14]:

```
rules = apriori(transactions,min_support=0.00030,min_confidance=0.05,min_lift=3,min_len
gth=2,target="rules")
association_results = list(rules)
```

In [15]:

```
for item in association_results:

    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    print("Support: " + str(item[1]))

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

```
Rule: specialty chocolate -> frozen fish
Support: 0.0003341575887188398
Confidence: 0.049019607843137254
Lift: 3.0689556157190907
=====================================
Rule: liver loaf -> fruit/vegetable juice
Support: 0.00040098910646260775
Confidence: 0.011787819253438114
Lift: 3.52762278978389
=====================================
Rule: pickled vegetables -> ham
Support: 0.0005346521419501437
Confidence: 0.03125
Lift: 3.4895055970149254
=====================================
Rule: roll products  -> meat
Support: 0.0003341575887188398
Confidence: 0.019841269841269844
Lift: 3.620547812620984
=====================================
Rule: misc. beverages -> salt
Support: 0.0003341575887188398
Confidence: 0.0211864406779661
Lift: 3.5619405827461437
=====================================
Rule: spread cheese -> misc. beverages
Support: 0.0003341575887188398
Confidence: 0.0211864406779661
Lift: 3.170127118644068
=====================================
Rule: soups -> seasonal products
Support: 0.0003341575887188398
Confidence: 0.04716981132075471
Lift: 14.704205974842766
=====================================
Rule: spread cheese -> sugar
Support: 0.00040098910646260775
Confidence: 0.06
Lift: 3.3878490566037733
=====================================
Rule: sausage -> butter
Support: 0.0003341575887188398
Confidence: 0.007374631268436578
Lift: 3.8050554368833285
=====================================
Rule: whole milk -> hard cheese
Support: 0.0003341575887188398
Confidence: 0.007374631268436578
Lift: 3.9409502739148756
=====================================
Rule: frozen vegetables -> canned beer
Support: 0.0003341575887188398
Confidence: 0.008880994671403198
Lift: 6.644316163410303
=====================================
Rule: sausage -> canned beer
Support: 0.00040098910646260775
Confidence: 0.010657193605683837
Lift: 4.309826700590467
=====================================
Rule: butter -> frankfurter
```

```
Support: 0.0003341575887188398
Confidence: 0.009487666034155597
Lift: 3.086172758023265
=====================================
Rule: yogurt -> canned beer
Support: 0.0003341575887188398
Confidence: 0.019011406844106463
Lift: 4.9046151829028455
=====================================
Rule: sausage -> canned beer
Support: 0.0003341575887188398
Confidence: 0.007122507122507123
Lift: 3.437873357228196
=====================================
Rule: whole milk -> canned beer
Support: 0.00040098910646260775
Confidence: 0.008547008547008546
Lift: 4.918803418803418
=====================================
Rule: chewing gum -> yogurt
Support: 0.00040098910646260775
Confidence: 0.03333333333333333
Lift: 5.732950191570881
=====================================
Rule: pork -> citrus fruit
Support: 0.00040098910646260775
Confidence: 0.004669260700389105
Lift: 3.4933073929961087
=====================================
Rule: rolls/buns -> frankfurter
Support: 0.0003341575887188398
Confidence: 0.008849557522123895
Lift: 3.6782202556538843
=====================================
Rule: frankfurter -> soda
Support: 0.0003341575887188398
Confidence: 0.010570824524312896
Lift: 3.438505377332475
=====================================
Rule: sausage -> pastry
Support: 0.0003341575887188398
Confidence: 0.010570824524312896
Lift: 3.2952343199436225
=====================================
Rule: sausage -> curd
Support: 0.0003341575887188398
Confidence: 0.009920634920634922
Lift: 5.497868900646679
=====================================
Rule: sausage -> curd
Support: 0.0003341575887188398
Confidence: 0.009920634920634922
Lift: 5.301516439909298
=====================================
Rule: sausage -> curd
Support: 0.00046782062420637575
Confidence: 0.007751937984496124
Lift: 3.4115367077063383
=====================================
Rule: sausage -> hard cheese
Support: 0.0003341575887188398
```

```
Confidence: 0.022727272727272728
Lift: 3.7785353535353536
=====================================
Rule: pip fruit -> ice cream
Support: 0.0003341575887188398
Confidence: 0.022026431718061675
Lift: 4.453804024288606
=====================================
Rule: shopping bags -> margarine
Support: 0.0003341575887188398
Confidence: 0.01037344398340249
Lift: 3.1043568464730287
=====================================
Rule: sausage -> margarine
Support: 0.00040098910646260775
Confidence: 0.0066445182724252485
Lift: 3.106935215946843
=====================================
Rule: sausage -> pastry
Support: 0.0003341575887188398
Confidence: 0.006459948320413437
Lift: 3.580007656235047
=====================================
Rule: onions -> yogurt
Support: 0.0003341575887188398
Confidence: 0.016501650165016504
Lift: 3.1655665566556657
=====================================
Rule: sausage -> waffles
Support: 0.0003341575887188398
Confidence: 0.002736726874657909
Lift: 3.4124703521255246
=====================================
Rule: yogurt -> other vegetables
Support: 0.0003341575887188398
Confidence: 0.03333333333333333
Lift: 4.122038567493113
=====================================
Rule: pork -> sausage
Support: 0.00040098910646260775
Confidence: 0.004669260700389105
Lift: 3.037658602605312
=====================================
Rule: whole milk -> pastry
Support: 0.0003341575887188398
Confidence: 0.006459948320413437
Lift: 5.685894512843897
=====================================
Rule: sausage -> whole milk
Support: 0.0003341575887188398
Confidence: 0.005537098560354374
Lift: 4.142580287929125
=====================================
```