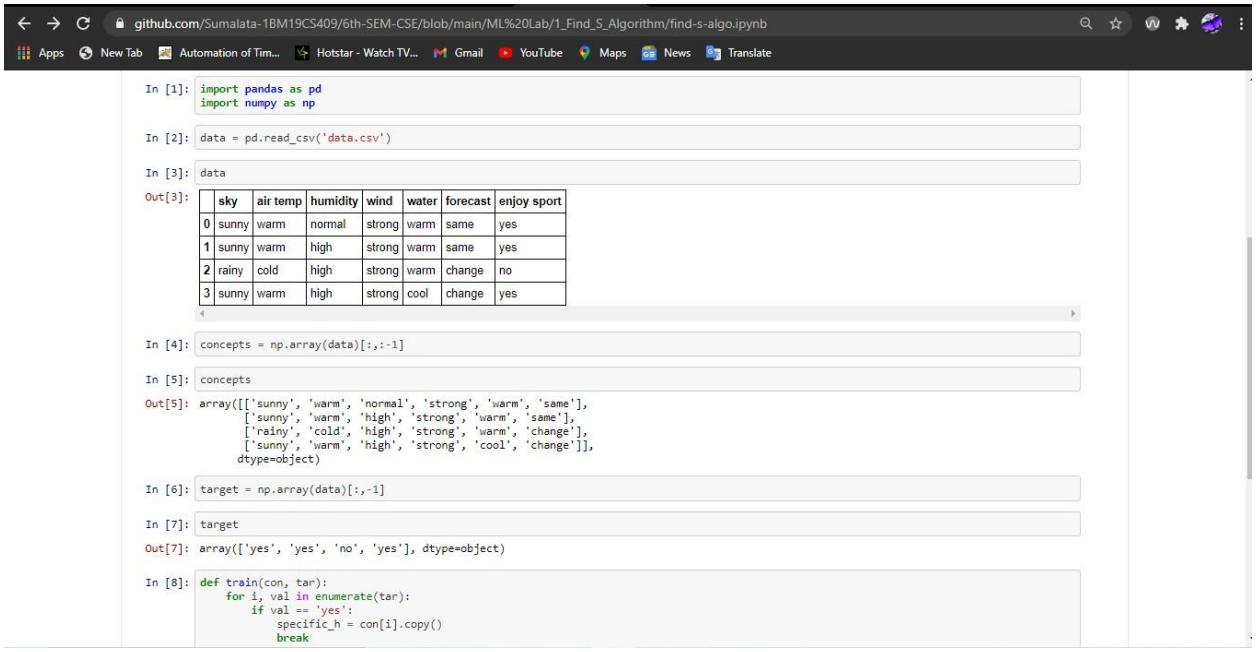
Sowmya D U
1BM19CS408

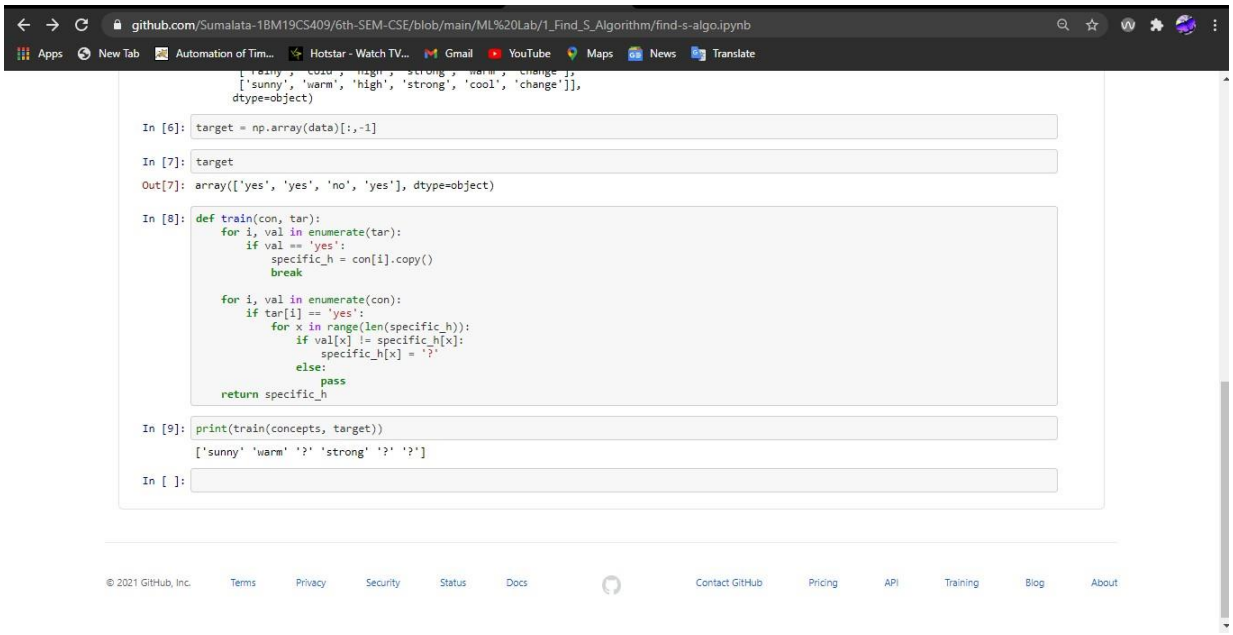
1. Fins – S algorithm

```
import pandas as pd
import numpy as np
data = pd.read_csv('data.csv')
data
concepts = np.array(data)[:,:-1]
concepts
target = np.array(data)[:,-1]
target
def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h
print(train(concepts, target))
```

Output



Sowmya D U
1BM19CS408



github.com/Sumalata-1BM19CS409/6th-SEM-CSE/blob/main/ML%20Lab/1_Find_S_Algorithm/find-s-algo.ipynb

Apps New Tab Automation of Tim... Hotstar - Watch TV... Gmail YouTube Maps News Translate

```
['sunny', 'warm', 'high', 'strong', 'cool', 'change']],
dtype=object)
```

In [6]:
```
target = np.array(data)[:,-1]
```

In [7]:
```
target
```

Out[7]:
```
array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

In [8]:
```
def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h
```

In [9]:
```
print(train(concepts, target))
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

In [ ]:

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

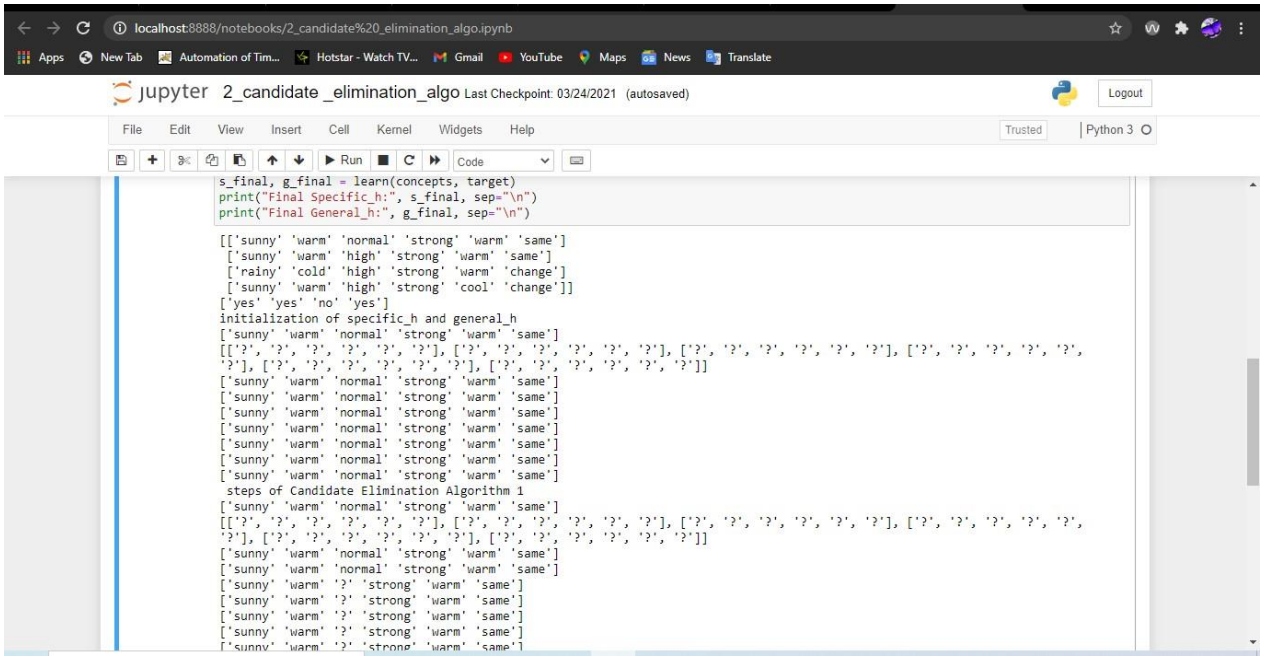
Sowmya D U
1BM19CS408

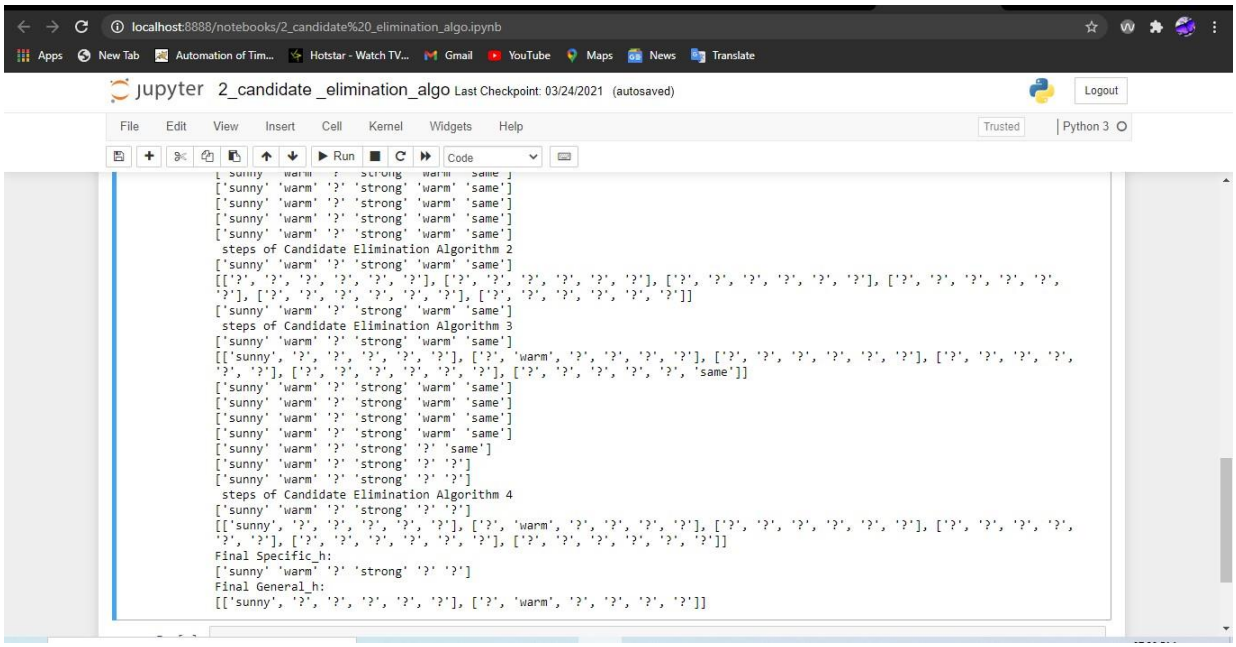
2. Candidate elimination algorithm

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                print(specific_h)
        print(specific_h)
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Sowmya D U
1BM19CS408

Output





Sowmya D U
1BM19CS408

3. ID3 algorithm

```
import math
import csv
```

In [2]:

```
def load_csv(filename):
    lines = csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers
```

In [3]:

```
class Node:
    def __init__(self,attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""
```

In [4]:

```
def subtables(data,col,delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))

    counts=[0]*len(attr)
    r = len(data)
    c = len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] for j in range(counts[x])]
        pos = 0
        for y in range(r):
            if data[y][col] == attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos] = data[y]
                pos+=1
    return attr, dic
```

In [5]:

```
def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0,0]
    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x])/(len(S)*1.0)
```

Sowmya D U
1BM19CS408

```
    sums = 0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums
```

In [6]:

```
def compute_gain(data,col):
    attr,dic = subtables(data, col, delete = False)

    total_size = len(data)
    entropies = [0]*len(attr)
    ratio = [0]*len(attr)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]])/(total_size*1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x]*entropies[x]
    return total_entropy
```

In [7]:

```
def build_tree(data,  features):
    lastcol = [row[-1] for row in data]
    if(len(set(lastcol))) == 1:
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0])-1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split]+features[split+1:]

    attr, dic = subtables(data, split, delete = True)
    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))
    return node
```

In [8]:

```
def print_tree(node, level):
    if node.answer != "":
        print(" "*level, node.answer)
        return

    print(" "*level, node.attribute)
    for value,n in node.children:
        print(" "*(level+1), value)
        print_tree(n, level+2)
```

In [9]:

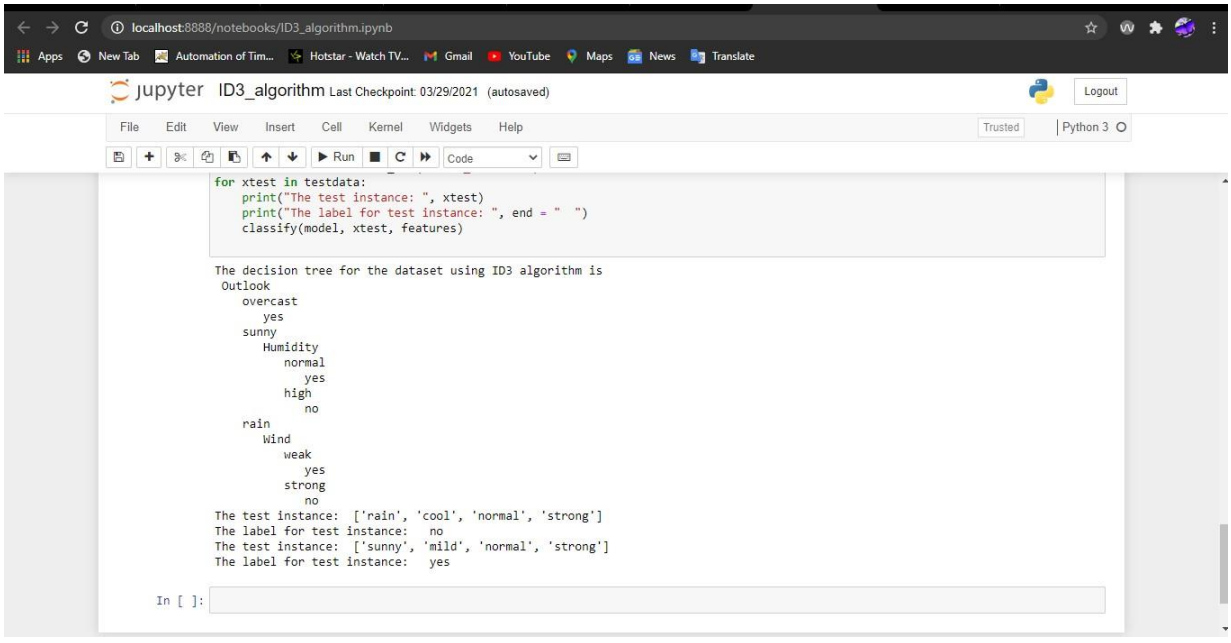
Sowmya D U
1BM19CS408

```
def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if    x_test[pos]    ==    value:
            classify(n, x_test, features)
```

In [10]:

```
'''Main Program'''
dataset, features = load_csv("data3.csv")
model = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(model, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance: ", xtest)
    print("The label for test instance: ", end = "  ")
    classify(model, xtest, features)
```

Output



Sowmya D U
1BM19CS408

4. Naïve Bayesian algorithm

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())
```

In [2]:

```
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
```

In [3]:

```
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
```

In [4]:

```
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

In [5]:

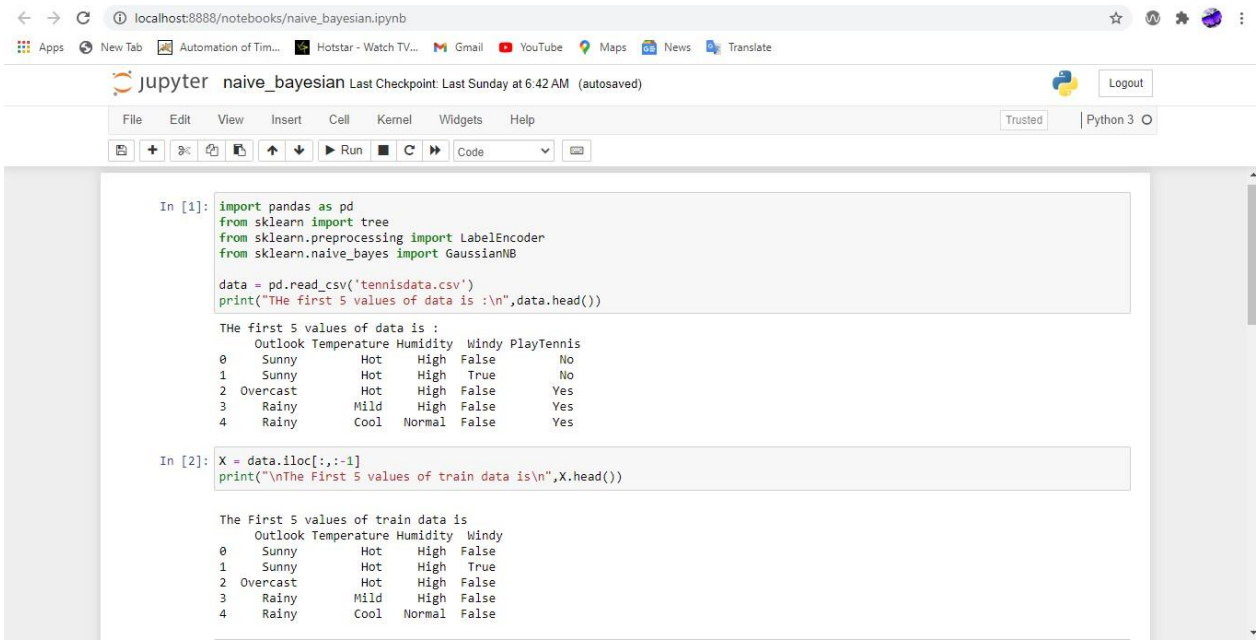
```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

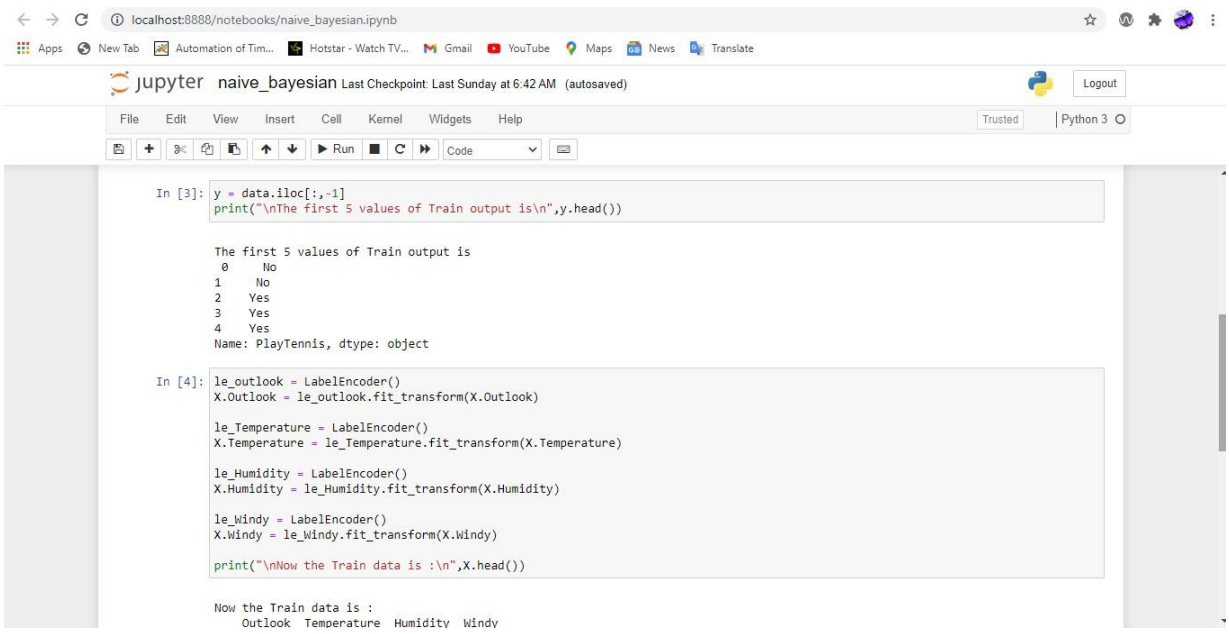
In [6]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

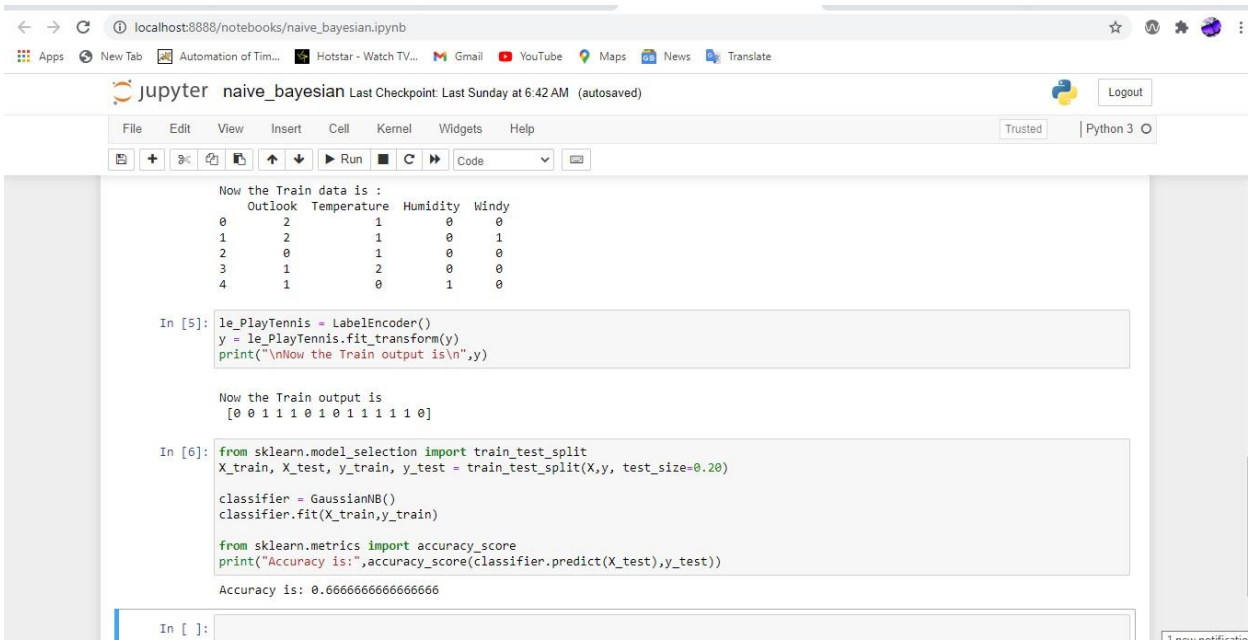
Sowmya D U
1BM19CS408

Output





Sowmya D U
1BM19CS408



Sowmya D U
1BM19CS408

5. Bayesian Network Classifier algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [3]:

```
df = pd.read_csv('pima_indian.csv')
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
```

In [4]:

```
X = df[feature_col_names].values # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict
```

In [5]:

```
#splitting the dataset into train and test data
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
```

In [6]:

```
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
```

In [7]:

```
# Training Naive Bayes (NB) classifier on training data.

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [8]:

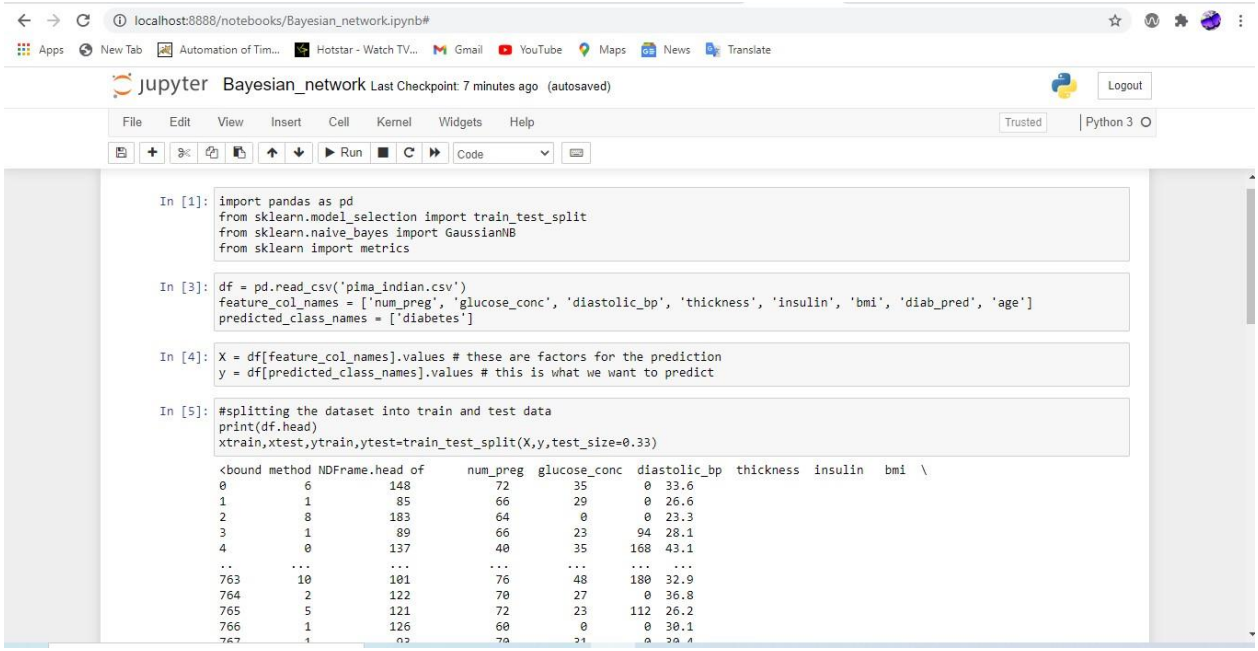
```
#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

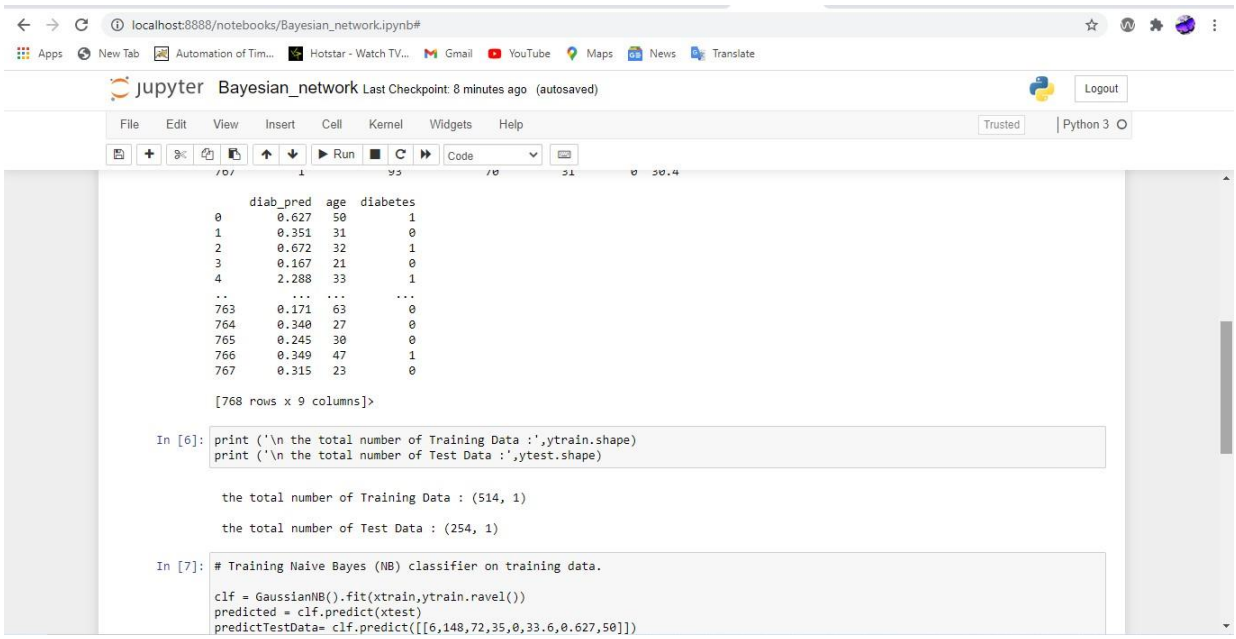
In [9]:

```
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

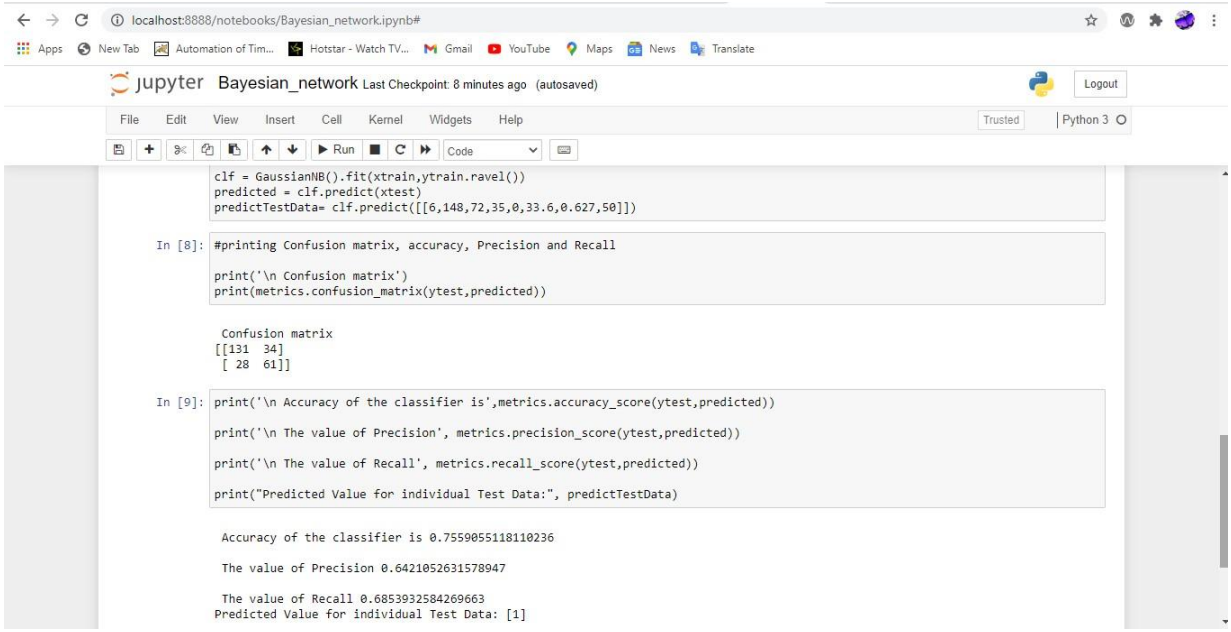
Sowmya D U
1BM19CS408

Output





Sowmya D U
1BM19CS408



Sowmya D U
1BM19CS408

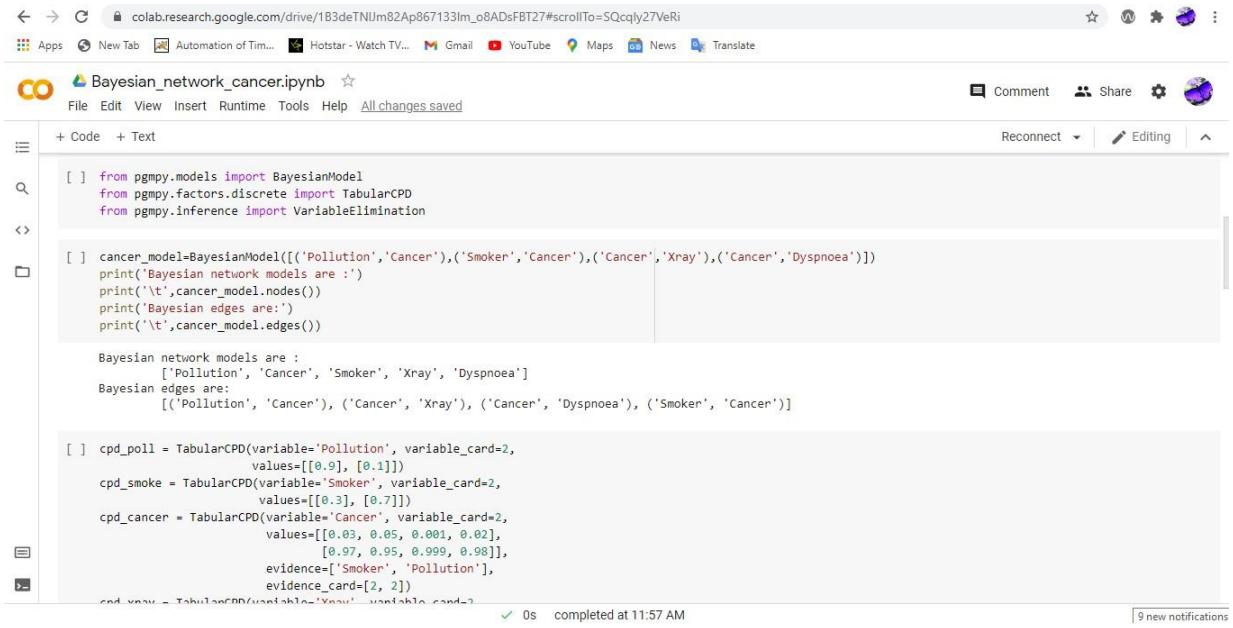
6. Bayesian Network using cancer dataset

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

cancer_model=BayesianModel([('Pollution','Cancer'),('Smoker','Cancer'),('Cancer','Xray'),('Cancer','Dyspnoea')
])
print('Bayesian network models are :')
print('\t',cancer_model.nodes())
print('Bayesian edges are:')
print('\t',cancer_model.edges())


cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
                      values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
                       values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
                        values=[[0.03, 0.05, 0.001, 0.02],
                                [0.97, 0.95, 0.999, 0.98]],
                        evidence=['Smoker', 'Pollution'],
                        evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
                      values=[[0.9, 0.2], [0.1, 0.8]],
                      evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
                      values=[[0.65, 0.3], [0.35, 0.7]],
                      evidence=['Cancer'], evidence_card=[2])

# Associating the parameters with the model structure.
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)

# Checking if the cpds are valid for the model.
cancer_model.check_model()

cancer_infer=VariableElimination(cancer_model)

print('All local independecies are as follows')
cancer_model.get_independencies()
print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
```

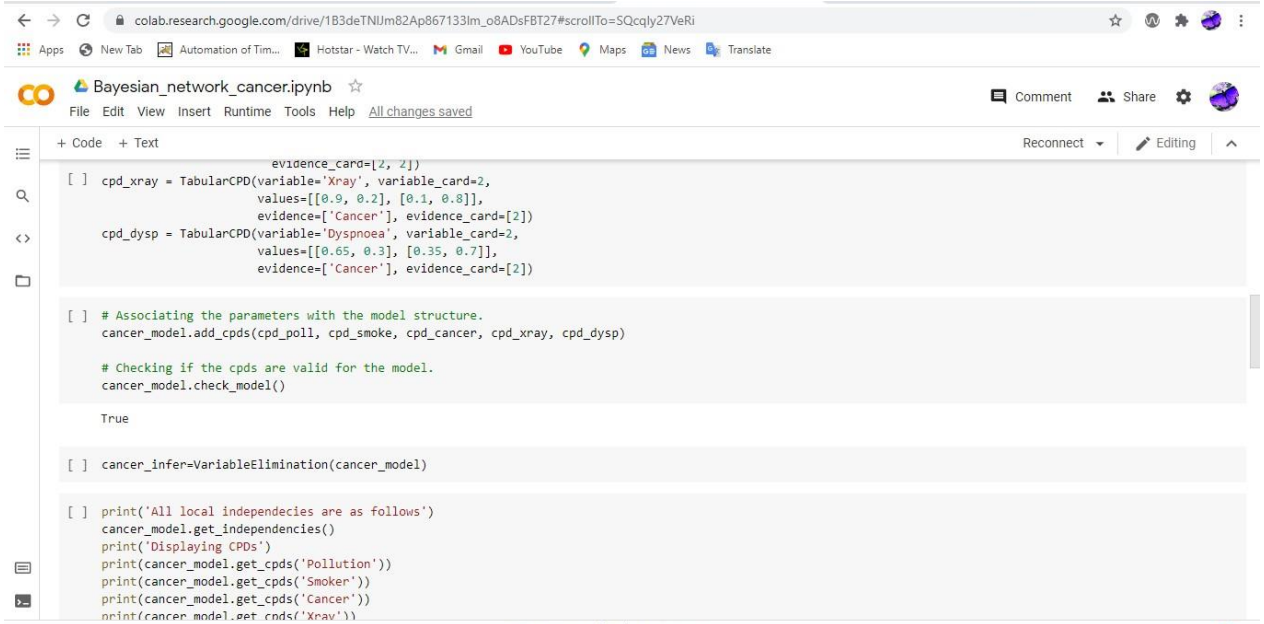
Sowmya D U
1BM19CS408

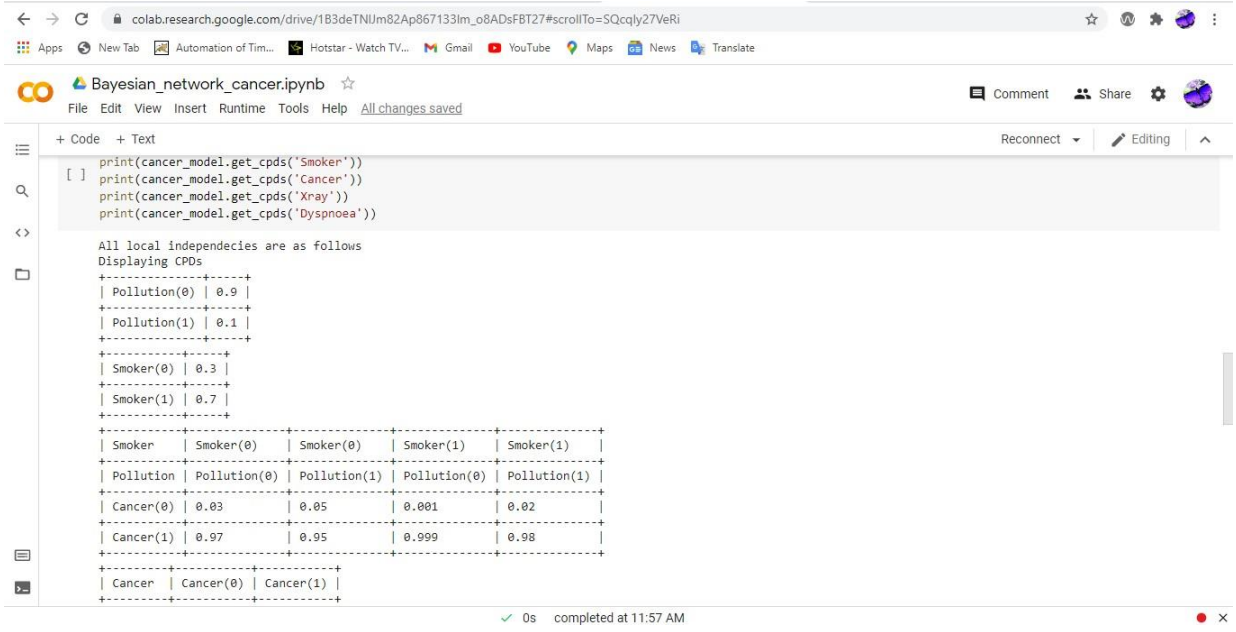
```
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))

print('\n Probablity of Cancer given smoker')
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1})
print(q)

print('\n Probablity of Cancer given smoker, pollution')
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1,'Pollution':1})
print(q)
```

Output



Sowmya D U
1BM19CS408





Sowmya D U
1BM19CS408