Regd No: Pg No:

Exercise-1

Q)Write a program to implement data cleaning techniques

Aim: To Write a program to implement data cleaning techniques

a)Creating DataFrame ¶

In [1]:
```
import numpy as np
import pandas as pd
data=pd.DataFrame(np.random.randn(4,3),index=[1,3,5,7],columns=("A","B","C"))
print(data)
```

```
          A         B         C
1 -0.505106  0.557892  0.147493
3  0.584394 -0.416774  0.793128
5  1.216465 -0.551766 -1.919672
7  1.218970 -0.076131 -0.286438
```

b)Re-indexing DataFrame

In [2]:
```
import numpy as np
import pandas as pd
data=pd.DataFrame(np.random.randn(4,3),index=[1,3,5,7],columns=("A","B","C"))
data=data.reindex([1,2,3,4,5,6,7,8])
print(data)
```

```
          A         B         C
1  0.012678 -0.232131 -0.707552
2       NaN       NaN       NaN
3 -0.935170  0.079498  1.060499
4       NaN       NaN       NaN
5  1.634519  1.730596  0.191508
6       NaN       NaN       NaN
7 -0.585106  0.218143 -0.371988
8       NaN       NaN       NaN
```

c)Checking For Missing Values

In [3]:
```
print(data["A"].isnull())
```

```
1    False
2     True
3    False
4     True
5    False
6     True
7    False
8     True
Name: A, dtype: bool
```

d)Filling Missing Values

(i) With Specific Value

In [4]:
```
print(data.fillna(619))
```

```
             A           B           C
1     0.012678   -0.232131   -0.707552
2   619.000000  619.000000  619.000000
3    -0.935170    0.079498    1.060499
4   619.000000  619.000000  619.000000
5     1.634519    1.730596    0.191508
6   619.000000  619.000000  619.000000
7    -0.585106    0.218143   -0.371988
8   619.000000  619.000000  619.000000
```

(ii) Forward Method

In [5]:
```
print(data.fillna(method='pad'))
```

```
          A         B         C
1  0.012678 -0.232131 -0.707552
2  0.012678 -0.232131 -0.707552
3 -0.935170  0.079498  1.060499
4 -0.935170  0.079498  1.060499
5  1.634519  1.730596  0.191508
6  1.634519  1.730596  0.191508
7 -0.585106  0.218143 -0.371988
8 -0.585106  0.218143 -0.371988
```

(iii) Backward Method

In [6]:
```
print(data.fillna(method='bfill'))
```

```
          A         B         C
1  0.012678 -0.232131 -0.707552
2 -0.935170  0.079498  1.060499
3 -0.935170  0.079498  1.060499
4  1.634519  1.730596  0.191508
5  1.634519  1.730596  0.191508
6 -0.585106  0.218143 -0.371988
7 -0.585106  0.218143 -0.371988
8       NaN       NaN       NaN
```

e)Drop Missing Values

In [7]:
```
print(data.dropna())
```

```
          A         B         C
1  0.012678 -0.232131 -0.707552
3 -0.935170  0.079498  1.060499
5  1.634519  1.730596  0.191508
7 -0.585106  0.218143 -0.371988
```

f)Replacing Generic Value with Specific Value

In [8]:
```
df=pd.DataFrame({"One":[1,2,3,4],"Two":[5,6,7,8]})
print(df.replace({4:10,8:12}))
```

```
   One  Two
0    1    5
1    2    6
2    3    7
3   10   12
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise-2

Q)Write a program to implement data preprocessing techniques

Aim: To Write a program to implement data preprocessing techniques

a)Data preprocessing using Onehotencoding

In [1]:
```
import pandas as pd
book_data={'empid':[200,500,100,750],'emploc':['Delhi','Pune','Goa','Bangalore'],
           'salary':[30000,35000,55000,43000]}
print(book_data)
df=pd.DataFrame(data=book_data)
print(df)
from sklearn.feature_extraction.text import CountVectorizer
v=CountVectorizer()
v.fit(df['emploc'].values)
x=v.transform(df['emploc'].values)
print(x)
print('Extracting EMP LOCATONS:',v.get_feature_names())
print(x.toarray())
```

```
{'empid': [200, 500, 100, 750], 'emploc': ['Delhi', 'Pune', 'Goa', 'Bangalore'], 'salary': [30000, 35000, 55000, 43000]}
   empid     emploc  salary
0    200      Delhi   30000
1    500       Pune   35000
2    100        Goa   55000
3    750  Bangalore   43000
  (0, 1)        1
  (1, 3)        1
  (2, 2)        1
  (3, 0)        1
Extracting EMP LOCATONS: ['bangalore', 'delhi', 'goa', 'pune']
[[0 1 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [1 0 0 0]]
```

b)Data preprocessing using Bag of words

In [2]:
```
#Bag of Words preprocessing technique
import pandas as pd #sklearn library contains a lot of efficient tools
# for machine learning and statistical modeling including classification,
# regression, clustering and dimensionality reduction.

#feature_extraction used to extract features in a format
# supported by machine learning algorithms
# from datasets consisting of formats such as text and image.
from sklearn.feature_extraction.text import CountVectorizer
#CountVectorizer means breaking down a sentence or any text into words by
# performing preprocessing tasks like converting all words to lowercase,
# thus removing special characters
data=pd.read_csv('bag.csv')
f=[]
for i in data['Review1'].values:
    sentence = ' '.join(s for s in i.split())
    f.append(sentence.upper().strip())
print(f)
v=CountVectorizer(ngram_range=(1,3))#generating combination of 1&2&3 word list  ngram_range=(1,3)
BoW=v.fit_transform(f)
print('extracting features:',v.get_feature_names())
print('Shape of matrix',BoW.toarray())
```

```
['THIS BOOK IS GOOD THIS BOOK IS FUNNY AND BORING THIS BOOK IS FULL OF FUN']
extracting features: ['and', 'and boring', 'and boring this', 'book', 'book is', 'book is full', 'book is funny', 'book is goo
d', 'boring', 'boring this', 'boring this book', 'full', 'full of', 'full of fun', 'fun', 'funny', 'funny and', 'funny and bori
ng', 'good', 'good this', 'good this book', 'is', 'is full', 'is full of', 'is funny', 'is funny and', 'is good', 'is good thi
s', 'of', 'of fun', 'this', 'this book', 'this book is']
Shape of matrix [[1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 3 3 3]]
```

c)Data preprocessing using Stemming

In [3]:
```
from nltk.stem.snowball import SnowballStemmer
s=SnowballStemmer('english')
p=['singing','singer','cooked','cooking','reader','reading','university']
uni=[s.stem(i) for i in p]
print(' '.join(uni))
```

```
sing singer cook cook reader read univers
```

Result: Process Terminated Successfully

d)Min-max scaling

In [1]:
```
from sklearn.preprocessing import MinMaxScaler
d=[[2,10],[3,2],[3.2,2],[0,18]]
s=MinMaxScaler()
s.fit(d)
print(s.fit_transform(d))
```

```
[[0.625  0.5   ]
 [0.9375 0.    ]
 [1.     0.    ]
 [0.     1.    ]]
```

Result: Process Terminated Successfully

e)Standardization

In [2]:
```
from sklearn.preprocessing import StandardScaler
d=[[0,6],[0,5],[1,4],[1,3]]
s=StandardScaler()
s.fit(d)
print(s.fit_transform(d))
```

```
[[-1.          1.34164079]
 [-1.          0.4472136 ]
 [ 1.         -0.4472136 ]
 [ 1.         -1.34164079]]
```

Result: Process Terminated Successfully

f) Normalization

In [3]:
```
from sklearn.preprocessing import Normalizer
d=[[-2,6,2,4],[1,3,5,2],[2,5,2,1],[1,1,2,3]]
s=Normalizer()
s.fit(d)
print(s.fit_transform(d))
```

```
[[-0.25819889  0.77459667  0.25819889  0.51639778]
 [ 0.16012815  0.48038446  0.80064077  0.32025631]
 [ 0.34299717  0.85749293  0.34299717  0.17149859]
 [ 0.25819889  0.25819889  0.51639778  0.77459667]]
```

Result: Process Terminated Successfully

Additional Program

Linear Regression using Gradient descent:

In [7]:
```
import numpy as np
def gradient_descent(x,y):
    m=0
    b=0
    iterations=10
    n=len(x)
    learning_rate=0.0001
    for i in range(iterations):
        ypredict= m * x + b
        cost=(1/n) * sum([val **2 for val in (y-ypredict)])
        m_derivative=-(2/n)*sum(x*(y-ypredict))
        b_derivative=-(2/n)*sum(y-ypredict)
        m=m - learning_rate * m_derivative
        b=b - learning_rate * b_derivative
        print('m {}, b {},iteration {} ,cost{}'.format(m,b,i,cost))
x=np.array([1,2,3,4,5])
y=np.array([5,7,9,11,13])
gradient_descent(x,y)
```

```
m 0.006200000000000001, b 0.0018000000000000002,iteration 0 ,cost89.0
m 0.012385280000000002, b 0.00359592,iteration 1 ,cost88.58369304
m 0.018555874832000004, b 0.005387769648,iteration 2 ,cost88.16935357626447
m 0.024711819245580805, b 0.0071755585691712,iteration 3 ,cost87.75697230888764
m 0.030853147908099024, b 0.008959296365910018,iteration 4 ,cost87.34653998192246
m 0.03697989540488166, b 0.010738992617891976,iteration 5 ,cost86.93804738317266
m 0.04309209623942019, b 0.012514656882125469,iteration 6 ,cost86.53148534398603
m 0.04918978483356419, b 0.014286298693005392,iteration 7 ,cost86.12684473904856
m 0.055272995527714544, b 0.016053927562366652,iteration 8 ,cost85.72411648617962
m 0.06134176258101615, b 0.01781755297953755,iteration 9 ,cost85.32329154612809
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise-3

Q) Make your data ready for model training

Aim: To make data ready for model training

a)Multi linear:

In [3]:
```
import pandas as pd
df=pd.read_csv('multilinear.csv')
print(df)
X=df[['distance','years']]
y=df['price']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=10)
from sklearn.linear_model import LinearRegression
a=LinearRegression()
a.fit(X_train,y_train)
print(a.predict(X_test))
print(a.score(X_train,y_train))
print(a.intercept_)
print(a.coef_)
print(a.coef_[0]*10+a.coef_[1]*3+a.intercept_)
```

```
   distance  years  price
0        20      5  10000
1        10      3   9000
2        50      8  40000
3        35      6  25000
4        42      7  35000
[41500. 30500.]
1.0
17499.999999999993
[ 2000. -9500.]
9000.0
```

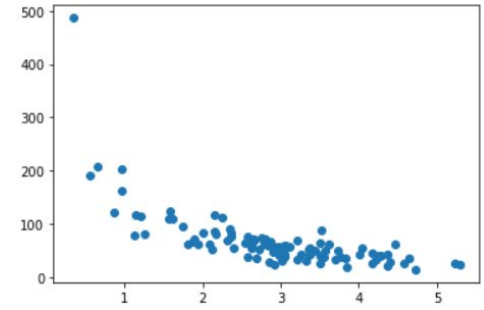
multilinear.csv:

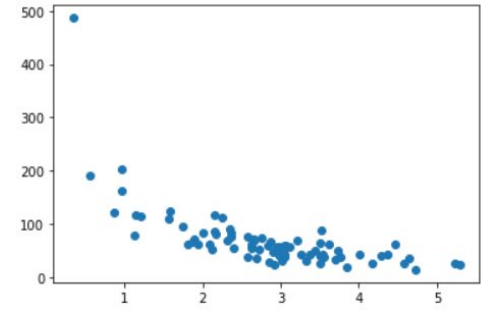
```
distance,years,price
20,5,10000
10,3,9000
50,8,40000
35,6,25000
42,7,35000
```

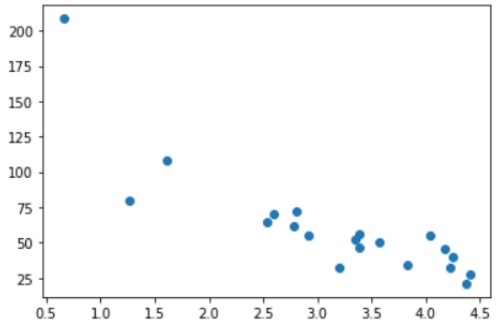
Result: Process Terminated Successfully

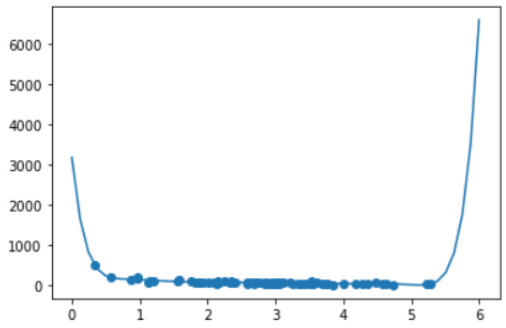
b) Training data (manual training):

In [4]:
```
import numpy
import matplotlib.pyplot as plt
import sys
numpy.random.seed(2)
x = numpy.random.normal(3, 1,100)
y = numpy.random.normal(150, 40,100) / x
plt.scatter(x, y)
plt.show()
train_x = x[:80]
train_y = y[:80]
test_x = x[80:]
test_y = y[80:]
plt.scatter(train_x, train_y)
plt.show()
plt.scatter(test_x, test_y)
plt.show()
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y,10))
myline = numpy.linspace(start=0, stop=6, num=50)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```







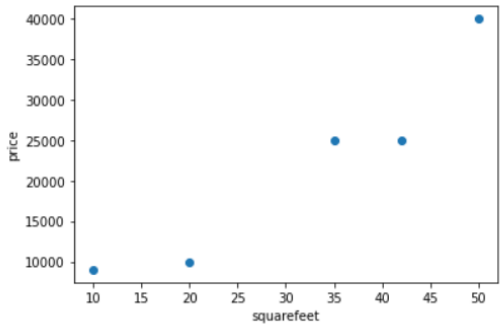


Result: Process Terminated Successfully

c) Simple Linear Regression:

In [5]:
```
import pandas as pd
df=pd.read_csv('simplelinearregression.csv')
print(df)
import matplotlib.pyplot as plt
plt.scatter(df['sft'],df['price'])
plt.xlabel('squarefeet')
plt.ylabel('price')
plt.show()
from sklearn.linear_model import LinearRegression
a=LinearRegression()
x=df[['sft']]
y=df[['price']]
reg=LinearRegression()
reg.fit(x,y)
#print(reg.predict([[35]]))
print(reg.coef_)
print(reg.intercept_)
print(35*848.18731118-2833.081571)
```

```
   sft  price
0   20  10000
1   10   9000
2   50  40000
3   35  25000
4   42  25000
```




```
[[748.11178248]]
[-1690.70996979]
26853.474320300003
```

simplelinearregression.csv:

```
sft,price
20,10000
10,9000
50,40000
35,25000
42,25000
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise-4

Q) Train, Validate and test KNN model

Aim: To Train, Validate and Test KNN model

In [3]:
```
import pandas as pd
df=pd.read_csv('gender.csv')
df.head()
x=df.drop(columns=['gender'])
x.head()
y=df['gender'].values
y[0:5]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train,y_train)
knn.predict(x_test)[0:5]
knn.score(x_test,y_test)
from sklearn.model_selection import cross_val_score
import numpy as np
knn_cv=KNeighborsClassifier(n_neighbors=3)
cv_scores=cross_val_score(knn_cv,x,y,cv=3)
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[0.5 0.5 0. ]
cv_scores mean:0.3333333333333333
```

gender.csv:

```
s.no,long_hair,forehead_width_cm,forehead_height_cm,nose_width,nose_long, lips_thin,distance_nose_to_lip_long,gender
0,1,11.8,6.1,1,0,1,1,male
1,0,14.5,4,0,0,1,0,female
2,0,11.8,6.3,1,1,1,1,male
3,0,14.4,6.1,0,1,1,1,male
4,1,13.5,5.9,0,0,0,0,female
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise-5

Q) Train, Validate and test Naïve Bayes model

Aim: To Train, Validate and Test KNN model

In [6]:
```
import pandas as pd
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy','Sunny','Overcast','Overcast'
         ,'Rainy']
temp=['Hot','Ho6t','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(weather)
print("Weather:",wheather_encoded)
# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print("Temp:",temp_encoded)
print("Play:",label)
#Combining weather and temp into single listof tuples
features=list(zip(wheather_encoded,temp_encoded))
print(features)
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
model = GaussianNB()
# Train the model using the training sets
model.fit(features,label)
#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

```
Weather: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]
Temp: [2 1 2 3 0 0 0 3 0 3 3 3 2 3]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
[(2, 2), (2, 1), (0, 2), (1, 3), (1, 0), (1, 0), (0, 0), (2, 3), (2, 0), (1, 3), (2, 3), (0, 3), (0, 2), (1, 3)]
Predicted Value: [1]
```

Regd No: Pg No:

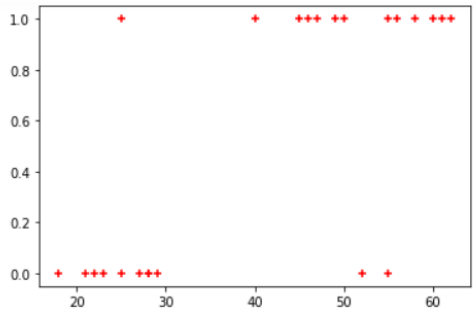
Exercise-6

Q) Train and Test Logistic Regression

Aim: To Train and Test Logistic Regression

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
df=pd.read_csv('logistic.csv')
print(df.head())
plt.scatter(df.age,df.insurance,marker='+',color='red')
plt.show()
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df[['age']],df.insurance,test_size=0.1)
print(len(X_train))
print(len(X_test))
print('training data is:',X_train)
print('testing data is',X_test)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train,y_train)
print(X_test)
print(model.predict(X_test))
print(model.predict_proba(X_test))
print(model.score(X_test,y_test))
```

```
   age  insurance
0   22          0
1   25          0
2   47          1
3   52          0
4   46          1
```




```
21
3
training data is:     age
5    56
14   49
19   28
11   28
6    55
13   29
23   23
8    62
10   18
9    61
4    46
18   21
7    60
2    47
1    25
```

```
17   58
15   55
12   27
21   45
22   50
0    22
testing data is     age
16   25
3    52
20   40
    age
16   25
3    52
20   40
[0 1 0]
[[0.97004422 0.02995578]
 [0.09134512 0.90865488]
 [0.56692583 0.43307417]]
0.0
```

logistic.csv:

```
age,insurance
22,0
25,0
47,1
52,0
46,1
56,1
55,0
60,1
62,1
61,1
18,0
28,0
27,0
29,0
49,1
55,1
25,1
58,1
21,0
28,0
40,1
45,1
50,1
23,0
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise-7

Q) Train, Validate and Test Logistic Regression

Aim: To Train, Validate and Test Logistic Regression

In [3]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('logistic.csv')
dataset=pd.DataFrame(data)
dataset.info()
dataset.describe().transpose()
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
target=dataset['insurance']
features=dataset.drop(['insurance'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(features,target,test_size=0.2)
from sklearn.svm import SVC
svc_model=SVC(C=1,kernel='linear',gamma=1)
svc_model.fit(x_train,y_train)
prediction=svc_model.predict(x_test)
print(svc_model.score(x_train,y_train))
print(svc_model.score(x_test,y_test))
print('confusion matrix: \n',confusion_matrix(prediction,y_test))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        24 non-null     int64
 1   insurance  24 non-null     int64
dtypes: int64(2)
memory usage: 512.0 bytes
0.8947368421052632
0.8
confusion matrix:
 [[0 0]
 [1 4]]
```

Result: Process Terminated Successfully

b)

In [4]:
```
from sklearn.svm import SVC
svc_model=SVC(C=1,kernel='rbf',gamma=1)
svc_model.fit(x_train,y_train)
prediction=svc_model.predict(x_test)
print(svc_model.score(x_train,y_train))
print(svc_model.score(x_test,y_test))
print('Confusion matrix:\n',confusion_matrix(prediction,y_test))
```

```
0.8947368421052632
0.8
Confusion matrix:
 [[0 0]
 [1 4]]
```

logistic.csv:

```
age,insurance
22,0
25,0
47,1
52,0
46,1
56,1
55,0
60,1
62,1
61,1
18,0
28,0
27,0
29,0
49,1
55,1
25,1
58,1
21,0
28,0
40,1
45,1
50,1
23,0
```

Result: Process Terminated Successfully

Regd No: Pg No:

Exercise -8

Q. Train Validate and test random forest ensemble.

Aim: To write a program to train validate and test random forest ensemble.

In [2]:
```
import pandas as pd
#from matplotlib import pyplot as plt
#import numpy as np
df=pd.read_csv("naivebayes.csv")
print(df.head())
sizes=df['playTennis'].value_counts(sort=1)
print(sizes)
df.drop(['Humidity'],axis=1,inplace=True)
df.drop(['wind'],axis=1,inplace=True)
print(df.head())
#convert non numeric to numeric
df.playTennis[df.playTennis=='yes']=1
df.playTennis[df.playTennis=='no']=2
df.outlook[df.outlook=='sunny']=1
df.outlook[df.outlook=='overcast']=2
df.outlook[df.outlook=='rain']=3
df.temperature[df.temperature=='hot']=1
df.temperature[df.temperature=='mild']=2
df.temperature[df.temperature=='cool']=3
print(df.head())
#define dependent variable
y=df['playTennis'].values
y=y.astype('int')
#define independent variable
X=df.drop(labels=['playTennis'],axis=1)
#split data
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=
10)
print(X_test)
print(X_train)
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=10,random_state=20)
model.fit(X_train,y_train)
predict_test =model.predict(X_test)
from sklearn import metrics
print("Accuracy = ",metrics.accuracy_score(y_test,predict_test))
```

```
    outlook temperature Humidity    wind playTennis
0     sunny         hot     high    week         no
1     sunny         hot     high    week         no
2  overcast        mild     high    week        yes
3      rain        cool   normal  strong        yes
4      rain        cool   normal  strong         no
yes    6
no     4
Name: playTennis, dtype: int64
    outlook temperature playTennis
0     sunny         hot         no
1     sunny         hot         no
2  overcast        mild        yes
3      rain        cool        yes
4      rain        cool         no
```

```
  outlook temperature playTennis
0       1           1          2
1       1           1          2
2       2           2          1
3       3           3          1
4       3           3          2
  outlook temperature
8       1           1
2       2           2
5       2           3
6       2           2
  outlook temperature
3       3           3
1       1           1
0       1           1
7       1           2
4       3           3
9       3           2
Accuracy =  1.0
```

naivebayes.csv:

```
outlook,temperature,Humidity,wind,playTennis
sunny,hot,high,week,no
sunny,hot,high,week,no
overcast,mild,high,week,yes
rain,cool,normal,strong,yes
rain,cool,normal,strong,no
overcast,cool,high,week,yes
overcast,mild,high,week,yes
sunny,mild,normal,strong,yes
sunny,hot,high,strong,no
rain,mild,normal,strong,yes
```

Result: Process Terminated Successfully

Regd No: Pg No:

Additional Program:

Word 2 vec:

In [3]:
```
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
#nltk.download('stopwords')
import re
p='''When it comes to modeling text,
there is a complicated problem as it is
not straight forward to convert it into
something that machine learning algorithms
that need specific inputs and outputs can understand.
There is no way the machine learning algorithms can
work with raw text as there is it not in a format that can directly be fed.
Machines deal with numbers like 45,78 etc, and the text must first
be converted to numbers, more precisely, vectors of numbers.
The vectors obtained from converting textual data can reflect the
properties of the text from a linguistic perspective.
In Natural Language Processing,
this is called feature encoding or feature extraction.
The bag of words model is a simple and popular feature extraction method.
'''
#preprocessing the data
t=re.sub(r'\[[0-9]*\)',' ',p)
t=re.sub(r'\s+',' ',t)
t=t.lower()
t=re.sub(r'\d',' ',t)
print(t)
#preparing the data set
ss=nltk.sent_tokenize(t)
ss=[nltk.word_tokenize(s) for s in ss]
for i in range(len(ss)):
 ss[i]=[word for word in ss[i] if word not in stopwords.words('english')]
print(ss[i])
#preparing the word2vec model
model=Word2Vec(ss,min_count=3)
words=model.wv.key_to_index
print(words)
#finding word vectors
vector=model.wv['text']
print(vector)
#finding similar words
similar=model.wv.most_similar('text')
print(similar)
```

Regd No: Pg No:

```
when it comes to modeling text, there is a complicated problem as it is not straight forward to convert it into something that
machine learning algorithms that need specific inputs and outputs can understand. there is no way the machine learning algorith
ms can work with raw text as there is it not in a format that can directly be fed. machines deal with numbers like   ,   etc, a
nd the text must first be converted to numbers, more precisely, vectors of numbers. the vectors obtained from converting textua
l data can reflect the properties of the text from a linguistic perspective. in natural language processing, this is called fea
ture encoding or feature extraction. the bag of words model is a simple and popular feature extraction method.
['bag', 'words', 'model', 'simple', 'popular', 'feature', 'extraction', 'method', '.']
{'.': 0, ',': 1, 'text': 2, 'feature': 3, 'numbers': 4}
[ 9.4563962e-05  3.0773187e-03 -6.8126465e-03 -1.3754654e-03
  7.6685809e-03  7.3464084e-03 -3.6732983e-03  2.6427007e-03
 -8.3171297e-03  6.2054847e-03 -4.6373224e-03 -3.1641079e-03
  9.3113566e-03  8.7338447e-04  7.4907015e-03 -6.0740639e-03
  5.1605059e-03  9.9228211e-03 -8.4573915e-03 -5.1356913e-03
 -7.0648384e-03 -4.8626517e-03 -3.7785650e-03 -8.5362010e-03
  7.9556061e-03 -4.8439382e-03  8.4236125e-03  5.2625705e-03
 -6.5500261e-03  3.9578700e-03  5.4701497e-03 -7.4265362e-03
 -7.4057197e-03 -2.4752307e-03 -8.6257271e-03 -1.5815735e-03
 -4.0343284e-04  3.2996845e-03  1.4418793e-03 -8.8142155e-04
 -5.5940580e-03  1.7303658e-03 -8.9737179e-04  6.7936899e-03
  3.9735888e-03  4.5294715e-03  1.4343048e-03 -2.6998566e-03
 -4.3668128e-03 -1.0320758e-03  1.4370275e-03 -2.6460099e-03
 -7.0737838e-03 -7.8053069e-03 -9.1217877e-03 -5.9351707e-03
 -1.8474245e-03 -4.3238713e-03 -6.4606713e-03 -3.7173224e-03
  4.2891572e-03 -3.7390448e-03  8.3781742e-03  1.5339922e-03
 -7.2423196e-03  9.4337985e-03  7.6312111e-03  5.4932809e-03
 -6.8488456e-03  5.8226776e-03  4.0090918e-03  5.1853680e-03
  4.2559002e-03  1.9397545e-03 -3.1701636e-03  8.3538434e-03
```

Result: Process Terminated Successfully