Machine Learning
Practical # 1

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Design the Machine Learning Model	**Batch**	Batch 2

Topic Design the Machine Learning Model

DESCRIPTION:- Degree Of Polynomial.

The “degree” of the polynomial is used to control the number of features added, e.g. a degree of 3 will add two new variables for each input variable. Typically a small degree is used such as 2 or 3.

a) AIM: Design a simple machine learning model to train the training instances and test the same.

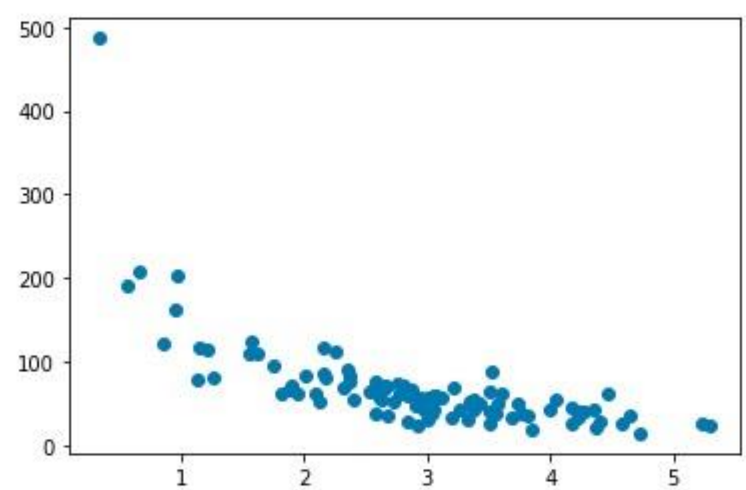
```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3,1,100)
print(x)
y = numpy.random.normal(150,40,100) /x
print(y)
```

```
[2.58324215 2.94373317 0.8638039  4.64027081 1.20656441 2.15825263
 3.50288142 1.75471191 1.94204778 2.09099239 3.55145404 5.29220801
 3.04153939 1.88207455 3.53905832 2.4038403  2.9808695  4.17500122
 2.25212905 3.00902525 2.12189211 2.84356583 3.25657045 2.01122095
 2.66117803 2.76381597 2.36234499 1.81238771 1.57878277 2.8465048
 2.73094304 5.23136679 0.56523242 3.1127265  3.37044454 4.35963386
 3.50185721 2.1557863  3.00000976 3.54235257 2.6864918  3.77101174
 1.13190935 4.73118467 4.46767801 2.66432266 3.61134078 3.04797059
 2.17086471 3.08771022 4.00036589 2.61890748 2.62433058 2.92552924
 3.43349633 4.27837923 2.36532069 3.50839624 3.21611601 1.14138761
 2.58068352 2.8676711  2.96042976 3.32600343 0.95967695 3.04625552
 2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
 2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
 3.69511961 0.96653345 2.81053074 2.92278133 3.82470301 4.24821292
 2.59610773 1.61548133 4.36723542 4.21788563 2.53799465 3.35088849
 3.38186623 3.56627544 3.20420798 4.40669624 1.2620405  4.04082395
 3.38047197 2.78286473 4.1735315  0.65639681]
[ 76.05204933  56.20180641 121.17874037  36.05903817 114.23885932
 117.41526024  63.77986643  95.52998052  62.4237197   60.57574247
```

```
 2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
 2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
 3.69511961 0.96653345 2.81053074 2.92278133 3.82470301 4.24821292
 2.59610773 1.61548133 4.36723542 4.21788563 2.53799465 3.35088849
 3.38186623 3.56627544 3.20420798 4.40669624 1.2620405  4.04082395
 3.38047197 2.78286473 4.1735315  0.65639681]
[ 76.05204933  56.20180641 121.17874037  36.05903817 114.23885932
 117.41526024  63.77986643  95.52998052  62.4237197   60.57574247
  38.57519009  24.10914678  37.45148182  67.13926856  39.26265343
  53.79918302  40.94657678  27.02857247 111.90190427  30.26663537
  51.4368334   58.83311239  42.08623741  83.01076429  68.37843898
  72.54627253  76.22874513  60.83111238 123.11113005  27.89501382
  53.25015791  24.86406278 190.30762228  55.79245737  42.32964984
  43.76381026  25.90093643  85.28325651  56.63901768  43.77321677
  34.70979433  37.10649687  77.86225629  14.09666443  62.93869329
  70.87521926  61.39097018  43.58292288  81.92492065  57.61442568
  43.5111781   57.3316853   53.67848811  22.97550427  50.79538368
  39.01941998  82.32095959  39.62788318  68.30365792 115.73628743
  38.66530343  65.39332448  44.34023444  30.00934597 161.50533328
  59.1743156   68.74904453 108.8692008   89.19445659  48.95077634
  90.02681869  18.36485932  62.86162946  59.01318439  71.22685026
  25.07604874 487.03726791  47.24533754  34.16662793 202.76589695
  72.37873053  55.46264153  34.46826737  40.15213735  70.55883508
 108.46604975  21.035144    32.35727584  64.76189111  52.19177448
  55.71813453  50.5667094   32.65308038  27.61777936  80.14230427
  54.98360439  46.50723143  61.85229524  45.84155234 208.47130994]
```

```python
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
from sklearn.model_selection import train_test_split

x = numpy.random.normal(3,1,100)
print(x)
y = numpy.random.normal(150,40,100) /x
print(y)
```

```python
plt.scatter(x,y)
plt.show()
```



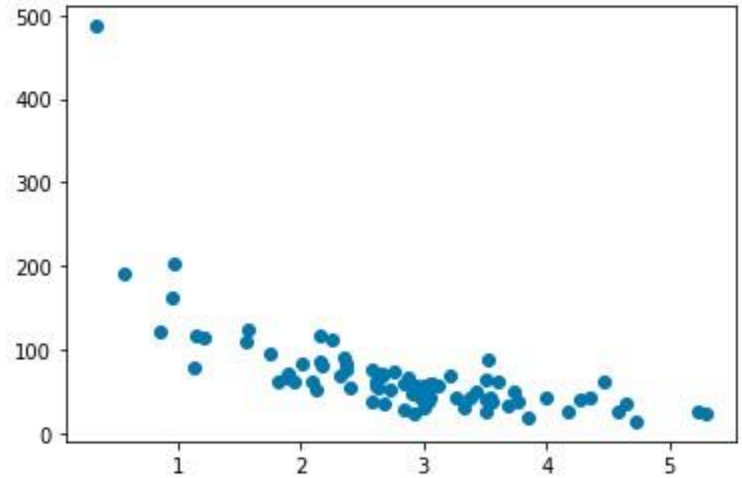
1. **Training data:-** Training data is the data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict.
2. **Test data:-** Test data is used to measure the performance, such as accuracy or efficiency, of the algorithm you are using to train the machine.

Code and output:

```
[4] train_x = x[:80]
    train_y = y[:80]

    test_x = x[:20]
    test_y = y[:20]

    print (train_x, train_y, test_x, test_y)
```

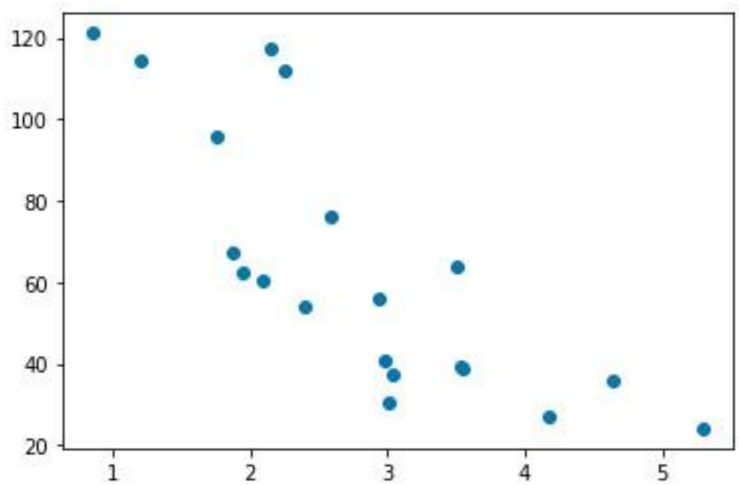
```
[2.58324215 2.94373317 0.8638039  4.64027081 1.20656441 2.15825263
 3.50288142 1.75471191 1.94204778 2.09099239 3.55145404 5.29220801
 3.04153939 1.88207455 3.53905832 2.4038403  2.9808695  4.17500122
 2.25212905 3.00902525 2.12189211 2.84356583 3.25657045 2.01122095
 2.66117803 2.76381597 2.36234499 1.81238771 1.57878277 2.8465048
 2.73094304 5.23136679 0.56523242 3.1127265  3.37044454 4.35963386
 3.50185721 2.1557863  3.00000976 3.54235257 2.6864918  3.77101174
 1.13190935 4.73118467 4.46767801 2.66432266 3.61134078 3.04797059
 2.17086471 3.08771022 4.00036589 2.61890748 2.62433058 2.92552924
 3.43349633 4.27837923 2.36532069 3.50839624 3.21611601 1.14138761
 2.58068352 2.8676711  2.96042976 3.32600343 0.95967695 3.04625552
 2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
 2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
 3.69511961 0.96653345] [ 76.05204933  56.20180641 121.17874037  36.05903817 114.23885932
 117.41526024  63.77986643  95.52998052  62.4237197   60.57574247
  38.57519009  24.10914678  37.45148182  67.13926856  39.26265343
  53.79918302  40.94657678  27.02857247 111.90190427  30.26663537
  51.4368334   58.83311239  42.08623741  83.01076429  68.37843898
  72.54627253  76.22874513  60.83111238 123.11113005  27.89501382
  53.25015791  24.86406278 190.30762228  55.79245737  42.32964984
  43.76381026  25.90093643  85.28325651  56.63901768  43.77321677
```

```
[5] plt.scatter(train_x,train_y)
    plt.show()
```

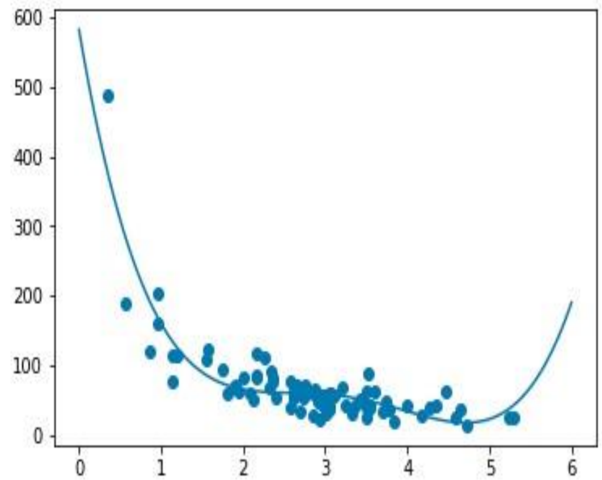



```
[25] train_x, test_x, train_y, test_y = train_test_split(x,y,test_size=0.3)
```

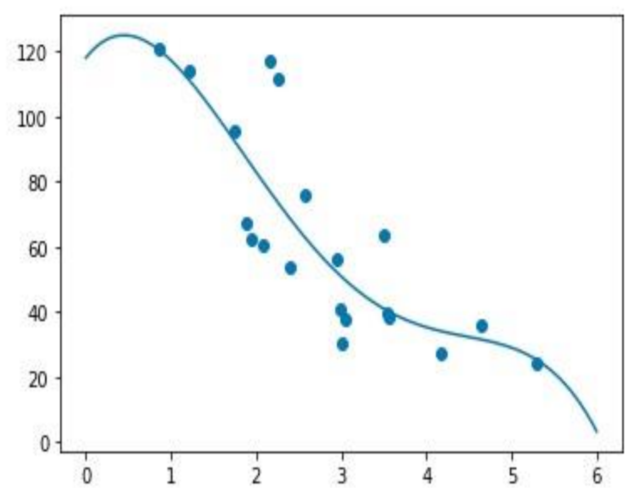
```
plt.scatter(test_x,test_y)
plt.show()
```



```
# Draw a Polynomial Regression line through the data points with training data
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0,6,100)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```




```
# Draw a Polynomial Regression line through the data points with test data
mymodel = numpy.poly1d(numpy.polyfit(test_x, test_y, 4))
myline = numpy.linspace(0,6,100)
plt.scatter(test_x, test_y)
plt.plot(myline, mymodel(myline))
plt.show()
```



```python
#It measures the relationship between the x axis and y axis
#where 0 means no relationship, and 1 means totally related.

import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)
r2 = r2_score(train_y, mymodel(train_x))
print(r2)
```

```
0.79886455446298
```

```python
#Predict the values
#Now that we have established that our model is OK, we can start prediction

print(mymodel(5))  #check with graph
```

```
22.8796259181172
```

Machine Learning
Practical # 2

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Concept Learning	**Batch**	Batch 2

Topic: Concept Learning / two-way classification / binary classification

a) AIM: Implement and demonstrate the find-s algorithm for finding the most specific.

DESCRIPTION:

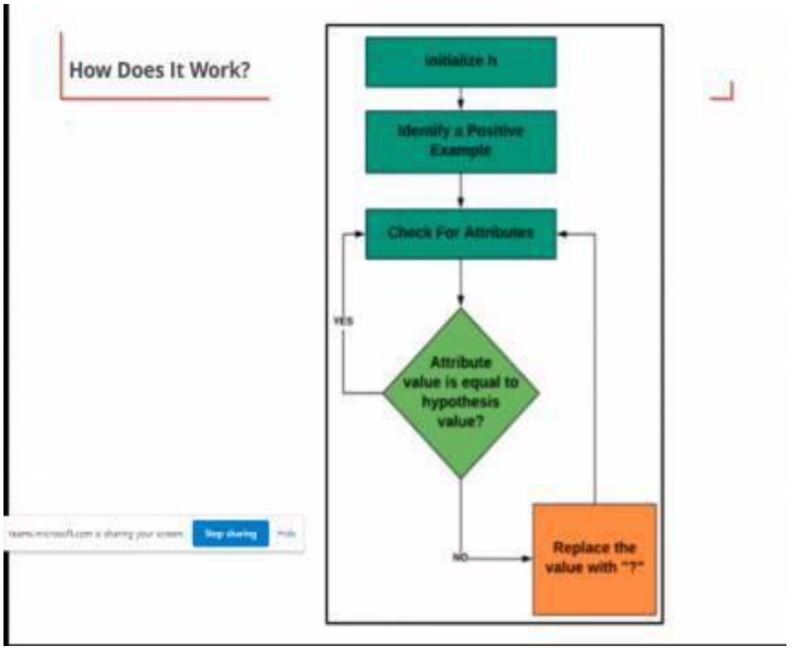
1. Training dataset table (input data):

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

2.: Write the right hypothesis/function from historical data

- The hypothesis is one of the commonly used concepts of statistics in Machine Learning.
- It is specifically used in Supervised Machine learning, where an ML model learns a function that best maps the input to corresponding outputs with the help of an available dataset.

3. How Does It Work?



4: Code and output:

```
import csv
num_attributes = 6
a = []
```

```
[2] print("\n The Given Training Data Set \n")
    with open('enjoysport.csv','r') as csvfile:
      reader = csv.reader(csvfile)
      count=0
      for row in reader:
        if count == 0:
          print (row) #heading
          count+=1;
        else:
          a.append (row)
          print(row)
          count+=1;
```

```
The Given Training Data Set

['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

```
[3] print("\n The initial value of hypothesis: ")
    hypothesis = ['0'] * num_attributes
    print(hypothesis)
```

```
The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
```

```
[4] for j in range(0,num_attributes):
      hypothesis[j] = a[0][j];
      print(hypothesis)
```

```
['Sunny', '0', '0', '0', '0', '0']
['Sunny', 'Warm', '0', '0', '0', '0']
['Sunny', 'Warm', 'Normal', '0', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

```
[6] print("\n find S: finding a Maximally Specific Hypothesis\n")
    for i in range(0,len(a)):
      if a[i][num_attributes]=='Yes':
        for j in range(0,num_attributes):
          if a[i][j]!=hypothesis[j]:
            hypothesis[j]='?'
          else:
            hypothesis[j]=a[i][j]
      print(" For training Example No :{0} the hypothesis is".format(i),hypothesis)
```

```
find S: finding a Maximally Specific Hypothesis

For training Example No :0 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For training Example No :1 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For training Example No :2 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For training Example No :3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
print(hypothesis)
```

```
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Machine Learning
Practical # 3

Name	Vaishnavi Pangam	Roll Number	21306A1074
Subject/Course:	Machine Learning	Class	M.Sc. IT – Sem III
Topic	Concept Learning - PCA	Batch	Batch 1 & 2

Topic: Feature Selection

a)

Aim: Data loading, feature scoring and ranking, feature selection (principal component analysis).

Description:

Principal Component Analysis:-

Principal component analysis, or PCA, is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected on the basis of variance that they cause in the output. The feature that causes highest variance is the first principal component will be ignored. The feature that is responsible for second highest variance is considered the second principal component, and so on. It is important to mention that principal components do not have any correlation with each other.

Code and output:

```
import numpy as np
import pandas as pd
url = https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

names = ['sepal-length','sepal-width','petal-length','petal-width','Class']
dataset = pd.read_csv(url, names=names)
dataset.head()
```

```
dataset.head()
```

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
#store the features sets into X variables and
# the series of corresponding variables in y
x=dataset.drop('Class',axis=1)
y=dataset['Class']
x.head()
```

```
[7] x.head()
```

	sepal-length	sepal-width	petal-length	petal-width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

```
y.head()
```

```
[8] y.head()

    0    Iris-setosa
    1    Iris-setosa
    2    Iris-setosa
    3    Iris-setosa
    4    Iris-setosa
    Name: Class, dtype: object
```

```
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.preprocessing import StandardScaler
sc = StandardScaler()
x_train1 = sc.fit_transform(x_train)
x_test1 = sc.transform(x_test)
y_train1 = y_train
y_test1 = y_test

from sklearn.decomposition import PCA
pca=PCA()
x_train1=pca.fit_transform(x_train1)
x_test1=pca.transform(x_test1)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.72226528 0.23974795 0.03338117 0.0046056 ]
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
x_train1 = pca.fit_transform(x_train1)
x_test1 = pca.transform(x_test1)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(x_train1, y_train1)
y_pred=classifier.predict(x_test1)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm=confusion_matrix(y_test,y_pred)
print(cm)
print('Accuracy',accuracy_score(y_test,y_pred))
```

```
[[11  0  0]
 [ 0 12  1]
 [ 0  1  5]]
Accuracy 0.9333333333333333
```

Machine Learning
Practical # 4

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	candidate-elimination algorithm	**Batch**	2

Topic: Candidate-elimination algorithm

a)

Aim: For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Description:

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example. You can consider this as an extended form of Find-S algorithm.

- Consider both positive and negative examples.
- Actually, positive examples are used here as Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified from generalize form.

Terms :-

General Hypothesis: Not Specifying features to learn the machine. **G = {'?', '?','?','?'…}:** Number of attributes.

Specific Hypothesis: Specifying features to learn machine (Specific feature). **S= {'pi','pi','pi'…}:** Number of pi depends on number of attributes.

Version Space: It is intermediate of general hypothesis and Specific hypothesis. It not only just written one hypothesis but a set of all possible hypothesis based on training data-set.

Candidate-elimination algorithm :-

```
Step1: Load Data set
Step2: Initialize General Hypothesis  and Specific  Hypothesis.
Step3: For each training example
Step4: If example is positive example
          if attribute_value == hypothesis_value:
             Do nothing
          else:
             replace attribute value with '?' (Basically generalizing it)
Step5: If example is Negative example
          Make generalize hypothesis more specific.
```

Code and output :-

```python
import numpy as np
import pandas as pd

#Loading data from a csv file.
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
print(data)
```

```
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes
```

```python
#Separating concept features from Target
concepts = np.array(data.iloc[:,0:6])
print(concepts)
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

```python
#Isolating target into a separate DataFrame
#Copying last column to target  array
target = np.array(data.iloc[:,6])
print(target)
```

```
['Yes' 'Yes' 'No' 'Yes']
```

```python
def learn(concepts, target):
#Initialise S0 with the first instance from concepts.
```

```
#.copy()makes sure a new list is created instead of just pointing to the same memory
location.
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h)
)]
    print("\nGeneric Boundary: ",general_h)
# The learning iterations.     for
i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
# Checking if the hypothesis has a positive target.
if target[i] == "yes":
            print("Instance is Positive ")
for x in range(len(specific_h)):  # Change
values in S & G only if values change.
                if h[x]!= specific_h[x]:
specific_h[x] ='?'
general_h[x][x] ='?'
# Checking if the hypothesis has a positive target.
if target[i] == "no":
print("Instance is Negative ")              for x in
range(len(specific_h)):  # For negative hypothesis
change values only in G.                  if h[x]!=
specific_h[x]:
general_h[x][x] = specific_h[x]
else:
general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
print("Generic Boundary after ", i+1, "Instance is ", general_h)         print("\n")
# find indices where we have empty rows, meaning those that are unchanged.
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?
', '?']]         for
i in indices:
# remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])
# Return final values
    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")
```

```
Initialization of specific_h and genearal_h

Specific Boundary:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?'

Instance 1 is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Specific Bundary after  1 Instance is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',



Instance 2 is  ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Specific Bundary after  2 Instance is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',



Instance 3 is  ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Specific Bundary after  3 Instance is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Generic Boundary after  3 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '


Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Machine Learning
Practical # 5

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Naïve Bayesian Classifier	**Batch**	2

Topic: Naïve Bayesian Classifier

a)

Aim: Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Description:

Naïve Bayesian Classifier : -

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

To start with, let us consider a dataset -

For Example,consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for playing golf.

Gaussian Classifier : -

The Gaussian Processes Classifier is **a classification machine learning algorithm**. Gaussian Processes are a generalization of the Gaussian probability distribution and can be used as the basis for sophisticated non-parametric machine learning algorithms for classification and regression.

Code and output:

```
#Import important libraries.
!pip install sklearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from sklearn) (1.0.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->sklearn) (1.7.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->sklearn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->sklearn) (1.21.6)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1310 sha256=3847e42102e63a5972cdbb8d7ef316140682e54862d48b1653d471e14d8881c4
  Stored in directory: /root/.cache/pip/wheels/46/ef/c3/157e41f5ee1372d1be90b09f74f82b10e391eaacca8f22d33e
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0
```

```
import numpy as np
import pandas as pd

#Import dataset
from sklearn import datasets

#Load dataset
wine = datasets.load_wine()

#print (wine) #if you want to see the data you can print data
#print the name of the 13 features

print("Features: ", wine.feature_names)
```

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od28
```

```
#print the label type of wine
print("Labels: ", wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head())
print(wine.data.shape)
```

```
     Labels:  ['class_0' 'class_1' 'class_2']
           0     1     2     3      4     5     6     7     8     9    10    11  \
     0  14.23  1.71  2.43  15.6  127.0  2.80  3.06  0.28  2.29  5.64  1.04  3.92
     1  13.20  1.78  2.14  11.2  100.0  2.65  2.76  0.26  1.28  4.38  1.05  3.40
     2  13.16  2.36  2.67  18.6  101.0  2.80  3.24  0.30  2.81  5.68  1.03  3.17
     3  14.37  1.95  2.50  16.8  113.0  3.85  3.49  0.24  2.18  7.80  0.86  3.45
     4  13.24  2.59  2.87  21.0  118.0  2.80  2.69  0.39  1.82  4.32  1.04  2.93

            12
     0  1065.0
     1  1050.0
     2  1185.0
     3  1480.0
     4   735.0
     (178, 13)

#print the wine labels (0:Class_0, 1:class_2, 2:class_2)
y=print (wine.target)

#import train_test_split function
from sklearn.model_selection import train_test_split


    [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
     2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

#split dataset into training set and test set.
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size
=0.30,random_state=10)
#import gaussian naive bayes model.
from sklearn.naive_bayes import GaussianNB

#create a gaussian classifier
gnb = GaussianNB()

#train the model using the training sets
gnb.fit(X_train,y_train)

   GaussianNB()


#predict the response for test dataset
y_pred = gnb.predict(X_test)
print(y_pred)

    [1 1 0 1 0 1 1 0 2 0 0 0 0 2 1 1 2 1 1 2 0 2 0 0 1 2 1 2 1 1 2 2 2 2 1 0 0
     1 0 1 1 0 2 1 0 1 2 1 1 1 2 1 2 0]
```

```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8888888888888888
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm
```

```
array([[14,  1,  0],
       [ 2, 22,  3],
       [ 0,  0, 12]])
```

Machine Learning
Practical # 6

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Decision Tree Classifier & Random Forest Classifier	**Batch**	2

Topic: Decision Tree Classifier & Random Forest Classifier.

a)

Aim: Write a program to implement the Decision Tree Classifier & Random Forest Classifier with prediction, test score and confusion matrix.

Description:

Decision Tree Classifier :-

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a treestructured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.

Random Forest Classifier :-

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- As the name suggests, **"*Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.*"**

Code and output :-

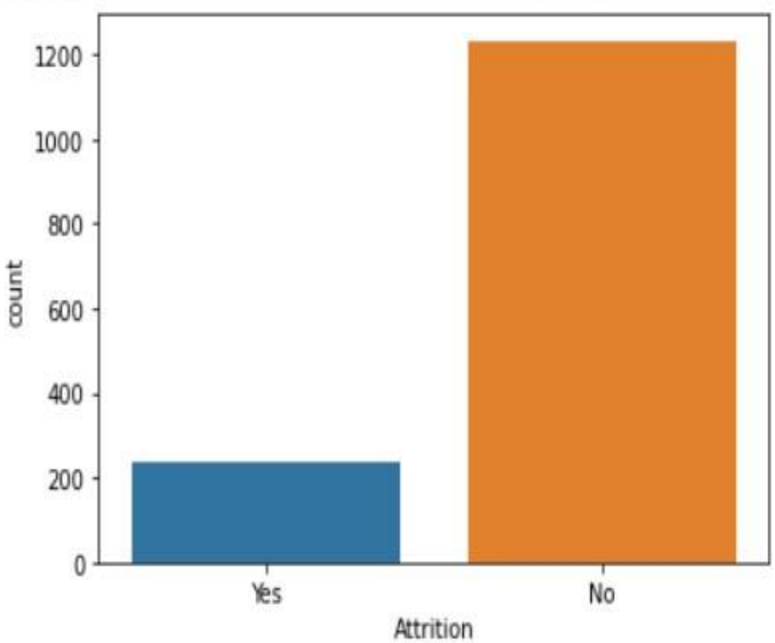
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df = pd.read_csv("WA_Fn-UseC_-HR-Employee-
Attrition.csv")  #Keeping emp position unaffects.
df.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationshipSatisfaction	StandardHours
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	1	80
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	4	80
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	2	80
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	3	80
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	4	80

5 rows × 35 columns

```
#Exploratory Data Analysis
sns.countplot(x='Attrition', data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ffb94f102d0>



```
from pandas.core.arrays import categorical
df.drop(['EmployeeCount','EmployeeNumber', 'Over18', 'StandardHours'], axis="columns"
, inplace=True)
categorical_col = [] for
column in df.columns:
  if df[column].dtype == object:
categorical_col.append(column)

df['Attrition'] = df.Attrition.astype("category").cat.codes

from sklearn.preprocessing import LabelEncoder for
column in categorical_col:
  df[column] = LabelEncoder().fit_transform(df[column]) from
sklearn.model_selection import train_test_split
X = df.drop('Attrition', axis=1) y
= df.Attrition
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
=42)


def print_score(clf, X_train, y_train, X_test, y_test, train=True):
   if
train:

    pred = clf.predict(X_train)

    clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))

    print("Train Result:\n=======================================")

    print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")

    print("____________________________________")

    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("____________________________________")

    print(f"Confusion Matrix: \n{confusion_matrix(y_train, pred)}\n")
   elif
train==False:

    pred = clf.predict(X_test)

    clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))

    print("Test Result:\n=======================================")

    print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
```

```
    print("____________________________________")

    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("____________________________________")

    print(f"Confusion Matrix: \n{confusion_matrix(y_test, pred)}\n")
```

1.Decision Tree Classifier :-

```
from sklearn.tree import DecisionTreeClassifier


from pickle import TRUE

from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(random_state=42)

tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train,y_train, X_test, y_test, train=True)

print_score(tree_clf, X_train,y_train, X_test, y_test, train=False)
```

```
Train Result:
========================================
Accuracy Score: 100.00%
_______________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    853.0  176.0       1.0     1029.0        1029.0
_______________________________________
Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
========================================
Accuracy Score: 77.78%
_______________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.887363   0.259740  0.777778    0.573551      0.800549
recall       0.850000   0.327869  0.777778    0.588934      0.777778
f1-score     0.868280   0.289855  0.777778    0.579067      0.788271
support    380.000000  61.000000  0.777778  441.000000    441.000000
_______________________________________
Confusion Matrix:
[[323  57]
 [ 41  20]]
```

2. Random Forest Classifier:-

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
========================================
Accuracy Score: 100.00%
________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    853.0  176.0       1.0     1029.0        1029.0
________________________________
Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
========================================
Accuracy Score: 86.17%
________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.871795   0.500000  0.861678    0.685897      0.820367
recall       0.984211   0.098361  0.861678    0.541286      0.861678
f1-score     0.924598   0.164384  0.861678    0.544491      0.819444
support    380.000000  61.000000  0.861678  441.000000    441.000000
________________________________
Confusion Matrix:
[[374   6]
 [ 55   6]]
```

➢ Confusion Matrix for Decision Tree :-

* Confusion Matrix :-

→ Decision Tree classifier :-

$$\begin{array}{cc} TP & FN \\ \end{array}$$

$$\begin{bmatrix} 323 & 57 \\ 41 & 20 \end{bmatrix}$$

$$\begin{array}{cc} FP & TN \\ \end{array}$$

$$TP = \frac{TP}{TP + FN} = \frac{323}{323+57} = \frac{323}{380} = 0.85.$$

$$TN = \frac{TN}{FP + TN} = \frac{20}{41+20} = \frac{20}{61} = 0.32.$$

$$FP = \frac{FP}{FP + TN} = \frac{41}{41+20} = \frac{41}{61} = 0.67.$$

$$FN = \frac{FN}{TN + FN} = \frac{57}{20+57} = \frac{57}{77} = 0.74.$$

→ Precision :- $\frac{TP}{TP+FP} = \frac{323}{323+41} = \frac{323}{364} = 0.88.$

→ Recall :- $\frac{TP}{TP+FN} = \frac{323}{323+57} = \frac{323}{380} = 0.85.$

- **Confusion Matrix for Random Forest:-**

★. Confusion Matrix :-

→ Random Forest Classifier :-

$$\begin{array}{cc} TP & FN \\ \left[\begin{array}{cc} 374 & 6 \\ 55 & 6 \end{array}\right] & \\ FP & TN \end{array}$$

$$TP = \frac{TP}{TP+FN} = \frac{374}{374+6} = \frac{374}{380} = 0.98$$

$$TN = \frac{TN}{FP+TN} = \frac{6}{55+6} = \frac{6}{61} = 0.09$$

$$FP = \frac{FP}{FP+TN} = \frac{55}{55+6} = \frac{55}{61} = 0.90$$

$$FN = \frac{FN}{TN+FN} = \frac{6}{6+6} = \frac{6}{12} = 0.28$$

→ Precision :- $\frac{TP}{TP+FP} = \frac{374}{374+55} = \frac{374}{429} = 0.87$

→ Recall :- $\frac{TP}{TP+FN} = \frac{374}{374+6} = \frac{374}{380} = 0.98$

VSIT

Machine Learning
Practical # 7

Name	Vaishnavi Pangam	Roll Number	21306A1074
Subject/Course:	Machine Learning	Class	M.Sc. IT – Sem III
Topic	Linear and Logistic Regression	Batch	2

Topic: Linear and Logistic Regression

Title: Least Square Regression

7.a) Aim: For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

Description:

Least Squares method :- The least squares method is a form of mathematical regression analysis used to determine the line of best fit for a set of data, providing a visual demonstration of the relationship between the data points. Each point of data represents the relationship between a known independent variable and an unknown dependent variable.

Code and output :-

```
# Making imports import
pandas as pd import
numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0,9.0)

# pre - processing input data data
= pd.read_csv('data.csv')
X = data.iloc[:,0] Y
= data.iloc[:,1]
plt.scatter(X,Y)
plt.show()
```


120

```
plt.scatter(X, Y)  # actual
plt.plot([min(X),max(X)],[min(Y_pred), max(Y_pred)], color='red') #prediction
plt.show()
```

Code and output :-

```
# Importing the libraries import
numpy as np
import matplotlib.pyplot as plt import
pandas as pd

# Importing the dataset
dataset = pd.read_csv('DMVWrittenTests.csv')
X = dataset.iloc[:, [0,1]].values Y
= dataset.iloc[:,2].values
dataset.head(5)
```

```
# Splitting the dataset into the training set and test set. from
sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.25, random_sta
te = 0)

# Feature scaling
# is used to normalize the data within a particular range.
# as the data is widely varies.
# a small limit (-2,2).
# the score 96.51142588 is normalized to 1.55187648.
# are normalized to a smaller range.

from sklearn.preprocessing import StandardScaler sc
= StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# Training the logistic regression model on the training set
```

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, Y_train)
```

```
LogisticRegression()

# Predicting the test set results.
y_pred = classifier.predict(X_test)
y_pred

array([1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0])

# Confusion Matrix and Accuracy.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy:", accuracy_score(Y_test, y_pred))
cm

Accuracy: 0.88
array([[11,  0],
       [ 3, 11]])
```

VSIT Machine Learning Practical # 8

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	K – Nearest Neighbour	**Batch**	2

Topic: K – Nearest Neighbour

a)

Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data-set.

Description:

The k-nearest neighbours algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

Code and output :-

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier import
numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()

print("\n IRIS TARGET NAMES: \n", iris_dataset.target_names) for
i in range(len(iris_dataset.target_names)):
print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))
```

```
In [6]: runfile('C:/Users/admin/Desktop/Practical 8 KNN.py', wdir='C:/Users/admin/Desktop')

 IRIS TARGET NAMES:
 ['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]
```

print("\n IRIS DATA :\n",iris_dataset["data"])

```
IRIS DATA :
[[5.1 3.5 1.4 0.2]
[4.9 3.  1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5.  3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
```

```
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"],
iris_dataset["target"],random_state=0)

print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)
```

```
Y TRAIN
[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1
0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0
1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2
0]

Y TEST
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
1]
```

```
print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(X_train, y_train)
```

```
Target :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2]

X TRAIN
[[5.9 3.  4.2 1.5]
[5.8 2.6 4.  1.2]
[6.8 3.  5.5 2.1]
[4.7 3.2 1.3 0.2]
[6.9 3.1 5.1 2.3]
[5.  3.5 1.6 0.6]
[5.4 3.7 1.5 0.2]
[5.  2.  3.5 1. ]
[6.5 3.  5.5 1.8]
[6.7 3.3 5.7 2.5]
[6.  2.2 5.  1.5]
[6.7 2.5 5.8 1.8]
[5.6 2.5 3.9 1.1]
[7.7 3.  6.1 2.3]
[6.3 3.3 4.7 1.6]
[5.5 2.4 3.8 1.1]
[6.3 2.7 4.9 1.8]
```

```
print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=5)
kn.fit(X_train, y_train)
```

```
Target :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

X TRAIN
[[5.9 3.  4.2 1.5]
 [5.8 2.6 4.  1.2]
 [6.8 3.  5.5 2.1]
 [4.7 3.2 1.3 0.2]
 [6.9 3.1 5.1 2.3]
 [5.  3.5 1.6 0.6]
 [5.4 3.7 1.5 0.2]
 [5.  2.  3.5 1. ]
 [6.5 3.  5.5 1.8]
 [6.7 3.3 5.7 2.5]
 [6.  2.2 5.  1.5]
```

```
for i in range (len(X_test)):
   X=X_test[i]
   X_new=np.array([X])
   prediction = kn.predict(X_new)
   print("\n Actual : {0}{1}, Predicted:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset ["target_names"][prediction]))
```

```
Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 2virginica, Predicted:[2]['virginica']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 0setosa, Predicted:[0]['setosa']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 1versicolor, Predicted:[1]['versicolor']

Actual : 0setosa, Predicted:[0]['setosa']

Actual : 0setosa, Predicted:[0]['setosa']
```

```
print("\n TEST SCORE[ACCURACY]: {:.2F}\n".format(kn.score(X_test,y_test)))
```

```
TEST SCORE[ACCURACY]: 0.97
```

Machine Learning
Practical # 9

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Euclidean Distance	**Batch**	2

Topic: Euclidean Distance Methods.

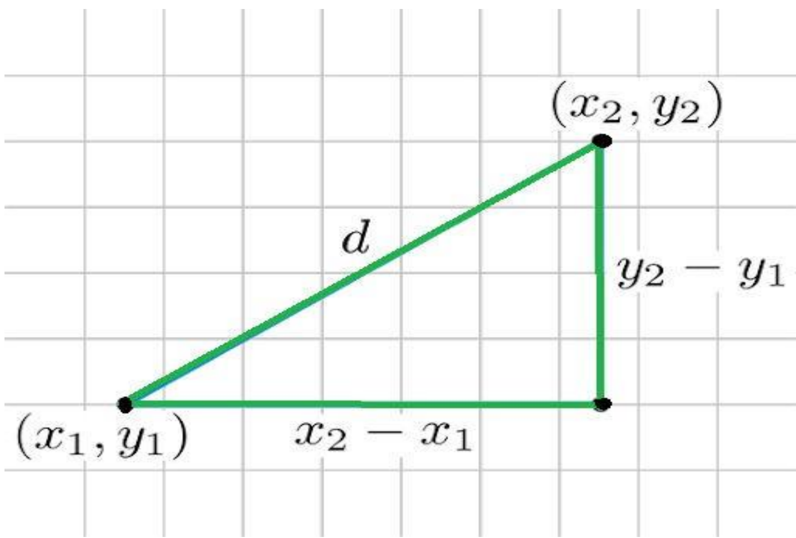
a)

Aim: Implement the different Distance methods (Euclidean) with Prediction, Test score and Confusion Matrix.

Description:

Euclidean distance is considered the traditional metric for problems with geometry. It can be simply explained as the ordinary distance between two points. It is one of the most used algorithms in the cluster analysis. One of the algorithms that use this formula would be K-mean. Mathematically it computes the root of squared differences between the coordinates between two objects.

$$
\begin{aligned}
d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) &= \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} \\
&= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}
\end{aligned}
$$



Code and output :-

```
#Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Mat
rix
#Import required libraries

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

#Load the dataset
df = pd.read_csv("IRIS.csv")

#quick look into the data
df.head(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
#Separate data and label
x = df.drop(['species'], axis=1)
y = df['species']

#Prepare data for classification process
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

#Create a model , p = 2 => Euclidean Distance:
knn = KNeighborsClassifier(n_neighbors = 6, p = 2, metric='minkowski')

#Train the model
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=6)
```

Calculate the accuracy of the model print(knn.score(x_test,
y_test))
y_pred = knn.predict(x_test)

```
0.9777777777777777
```

#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred)) cm

```
array([[16,  0,  0],
       [ 0, 17,  1],
       [ 0,  0, 11]])
```

#Create a model , p = 1 => Manhattan Distance
knn = KNeighborsClassifier(n_neighbors = 6, p = 1, metric='minkowski')

#Train the model
knn.fit(x_train, y_train)

```
KNeighborsClassifier(n_neighbors=6, p=1)
```

Calculate the accuracy of the model print(knn.score(x_test,
y_test))
y_pred = knn.predict(x_test)

```
0.9555555555555556
```

#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred)) cm

```
array([[16,  0,  0],
       [ 0, 17,  1],
       [ 0,  1, 10]])
```

#Create a model ,p = ∞, Chebychev Distance
#let ∞ = 10000
knn = KNeighborsClassifier(n_neighbors = 6, p = 10000, metric='minkowski')

#Train the model
knn.fit(x_train, y_train)

```
KNeighborsClassifier(n_neighbors=6, p=10000)
```

# Calculate the accuracy of the model
print(knn.score(x_test, y_test))
y_pred = knn.predict(x_test)

```
0.8
```

#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm

```
array([[16,  0,  0],
       [ 0, 18,  0],
       [ 0,  9,  2]])
```

Topic	K – Means Clustering	Batch	2

Topic: K – Means Clustering.

a)

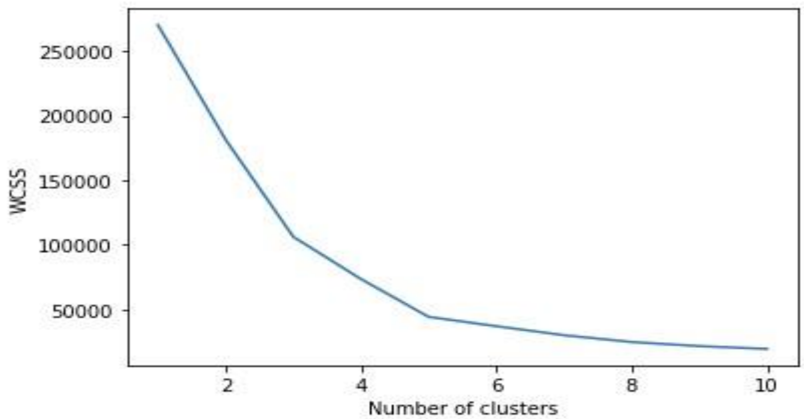
Aim: Implement the classification model using K-means clustering with Prediction, Test score and Confusion Matrix.

Description:

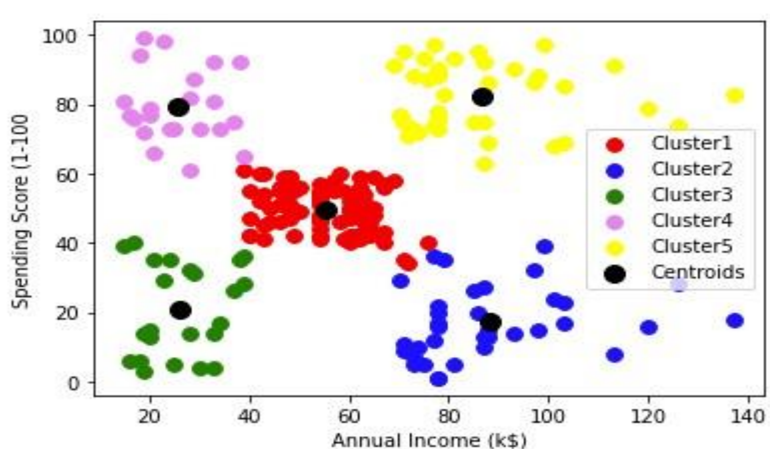
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

Code and output :-

```
import numpy as np
import matplotlib.pyplot as plt import
pandas as pd
import sklearn

#Import the dataset and slice the important features
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3,4]].values

#Find the optimal k value for clustering the data. from
sklearn.cluster import KMeans wcss = [] for i in range(1,11):
kmeans = KMeans(n_clusters=i, init='k-means++',random_state=42)
kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') plt.show()
```




```
#The point at which the elbow shape is created is 5.
kmeans = KMeans(n_clusters=5,init="k-means++",random_state=42)
y_kmeans = kmeans.fit_predict(X)

plt.scatter(X[y_kmeans == 0,0], X[y_kmeans == 0,1], s = 60, c = 'red', label = 'Clust
er1')
plt.scatter(X[y_kmeans == 1,0], X[y_kmeans == 1,1], s = 60, c = 'blue', label = 'Clus
ter2')
plt.scatter(X[y_kmeans == 2,0], X[y_kmeans == 2,1], s = 60, c = 'green', label = 'Clu
ster3')
plt.scatter(X[y_kmeans == 3,0], X[y_kmeans == 3,1], s = 60, c = 'violet', label = 'Cl
uster4')
plt.scatter(X[y_kmeans == 4,0], X[y_kmeans == 4,1], s = 60, c = 'yellow', label = 'Cl
uster5')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],s=100,c='black
',label='Centroids')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100')
plt.legend()
plt.show()
```



Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Text pre-processing , Text clustering	**Batch**	2

Topic: Text pre-processing , Text clustering.

a)

Aim: Text pre-processing, text clustering, classification with prediction, test score and confusion matrix.

Description:

Text pre-processing is an approach for cleaning and preparing text data for use in a specific context. Developers use it in almost all natural language processing (NLP) pipelines, including voice re… Noise Removal. Text cleaning is a technique that developers use in a variety of domains.

Text clustering is to automatically group textual documents (for example, documents in plain text, web pages, emails and etc) into clusters based on their content similarity.

Code and output :-

```
import numpy as np
import matplotlib.pyplot as plt import
pandas as pd

dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)

import re import
nltk
nltk.download('stopwords') from
nltk.corpus import stopwords from
nltk.stem.porter import PorterStemmer
corpus = []
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
for i in range(0,1000):    review = re.sub('[^a-zA-
Z]','',dataset['Review'][i])    review = review.lower()
review = review.split()
```

```
  ps = PorterStemmer()
  review = [ps.stem(word) for word in review if not word in set(stopwords.words('engl
ish'))]
  review = ''.join(review)
  corpus.append(review)


#Creating the bag of words model
from sklearn.feature_extraction.text import CountVectorizer cv
= CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()
Y = dataset.iloc[:,1].values

#Splitting the dataset into the training set and test set from
sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.25, random_sta
te=100)


#Fitting naive bayes to the training set. from
sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
```

```
GaussianNB()
```

```
# Predicting the test set results.
Y_pred = classifier.predict(X_test)

#Model Accuracy
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))
```

```
Accuracy: 0.54
```

```
#Making the confusion matrix
from sklearn.metrics import confusion_matrix cm
= confusion_matrix(Y_test, Y_pred) print(cm)
```

```
[[  1 115]
 [  0 134]]
```

Machine Learning

Name	Vaishnavi Pangam	**Roll Number**	21306A1074
Subject/Course:	Machine Learning	**Class**	M.Sc. IT – Sem III
Topic	Multiclass classification - SVM	**Batch**	2

Topic: Multiclass classification (Problem based Learning)

AIM: Support vector machine (SVM) algorithm for multiclass classification

Description: Calculate the TP, TN, FP, FN values for the class Setosa using the confusion matrix / contingency table and also calculate precision and recall for the same:

Calculate the TP, TN, FP, FN values for the class setosa using the confusion matrix and also calculate precision and recall for the same.

Confusion Matrix

Actual values			
setosa →	16 (TP)	0 (FN)	0 (FN)
versicolor →	0 (FP)	18 (TN)	0 (TN)
virginica →	0 (FP)	0 (TN)	11 (TN)
	↑ setosa	↑ versicolor	↑ virginica

$$TP = \frac{TP}{TP+FN} = \frac{16}{16+0+0} = \frac{16}{16} = 1$$

$$TN = \frac{TN}{FP+TN} = \frac{18+11}{0+0+18+11+0+0} = \frac{29}{29} = 1$$

$$FP = \frac{FP}{FP+TN} = \frac{0}{0+18+11+0+0} = \frac{0}{29} = 0$$

$$FN = \frac{FN}{TN+FN} = \frac{0+0}{18+11+0+0+0+0} = \frac{0}{29} = 0$$

$$Precision = \frac{TP}{TP+FP} = \frac{16}{16+0} = \frac{16}{16} = 1$$

$$Recall = \frac{TP}{TP+FN} = \frac{16}{16+0+0} = \frac{16}{16} = 1$$

Code and output:

+ Code + Text

```
[1] #importing packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[2] #importing of dataset to dataframe
df = pd.read_csv("Iris.csv")
```

```
[3] #To see first 5 rows of the dataset
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
[5] #To know the data type of the variables.
df.dtypes
```

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

```
[8] #Speceis is the output class, to know the count of each class we use value_counts()
df['Species'].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

```
[11] #Separate independent variable and dependent variable("Species")
     x = df.drop(['Species'], axis=1)
     y = df['Species']
     # print(x.head())
     print(x.shape)
     #print(y.head())
     print(y.shape)
```

```
(150, 5)
(150,)
```

```
[12] #Splitting the dataset to Train and Test
     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=0)
```

```
[13] #To know the shape of the train and test dataset.
     print(x_train.shape)
     print(y_train.shape)
     print(x_test.shape)
     print(y_test.shape)
```

```
(105, 5)
(105,)
(45, 5)
(45,)
```

```
[14] #We use support Vector classifier as a classifier
     from sklearn.svm import SVC
     from sklearn.metrics import confusion_matrix
```

```
[16] #Training the classifier using x_Train and x_train
     clf = SVC(kernel ='linear').fit(x_train,y_train)
     clf.predict(x_train)
```

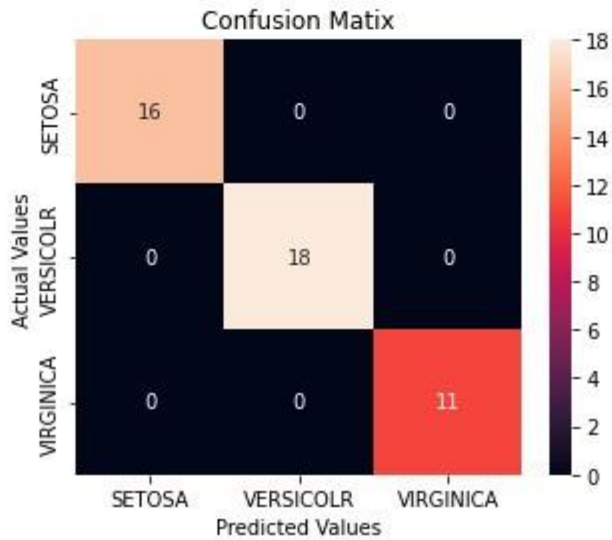
```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
```

```
[17] #Testing the model using x_test and storing the output in y_pred
     y_pred = clf.predict(x_test)
```

```
[18] #Creating a confusion matrix, which compares the y_test and y_pred
     cm = confusion_matrix(y_test, y_pred)
```

```
[19] cm_df = pd.DataFrame(cm,
                          index = ['SETOSA','VERSICOLR','VIRGINICA'],
                          columns = ['SETOSA','VERSICOLR','VIRGINICA'])
```

```
#plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



Confusion Matrix for Banksheet data

```
[2] df = pd.read_csv("/content/banksheet.csv")
```

```
[3] df.head()
```

	age	balance	day	duration	campaign	pdays	previous	poutcome
0	45	0	2	570	2	-1	0	unknown
1	38	485	2	252	1	-1	0	unknown
2	29	642	2	156	3	-1	0	unknown
3	32	2142	2	406	1	-1	0	unknown
4	41	840	2	938	1	-1	0	unknown

```
[4] df.dtypes
```

```
age            int64
balance        int64
day            int64
duration       int64
campaign       int64
pdays          int64
previous       int64
poutcome      object
dtype: object
```

```
[5] df['poutcome'].value_counts()
```

```
unknown    174
failure     58
other       26
success      2
Name: poutcome, dtype: int64
```

```
[6] x = df.drop(['poutcome'], axis=1)
    y = df['poutcome']
    # print(x.head())
    print(x.shape)
    #print(y.head())
    print(y.shape)
```

```
(260, 7)
(260,)
```

```
[7] 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=0)
```

```
[8] print(x_train.shape)
    print(y_train.shape)
    print(x_test.shape)
    print(y_test.shape)
```

```
(182, 7)
(182,)
(78, 7)
(78,)
```

```
[9] from sklearn.svm import SVC
    from sklearn.metrics import confusion_matrix
```

```
[10] clf = SVC(kernel ='linear').fit(x_train,y_train)
     clf.predict(x_train)
```

```
array(['failure', 'unknown', 'unknown', 'failure', 'unknown', 'unknown',
       'unknown', 'unknown', 'failure', 'unknown', 'unknown', 'failure',
       'unknown', 'failure', 'unknown', 'unknown', 'unknown', 'unknown',
       'other', 'other', 'unknown', 'unknown', 'failure', 'unknown',
       'unknown', 'unknown', 'unknown', 'unknown', 'unknown', 'unknown',
       'unknown', 'other', 'unknown', 'unknown', 'unknown', 'unknown',
       'failure', 'unknown', 'unknown', 'unknown', 'failure', 'failure',
       'failure', 'unknown', 'unknown', 'failure', 'unknown', 'failure',
       'failure', 'unknown', 'unknown', 'unknown', 'unknown', 'unknown',
       'unknown', 'unknown', 'failure', 'unknown', 'unknown', 'failure',
       'failure', 'unknown', 'failure', 'unknown', 'unknown', 'other',
       'failure', 'unknown', 'unknown', 'other', 'unknown', 'unknown',
       'unknown', 'unknown', 'failure', 'failure', 'unknown', 'unknown',
       'unknown', 'unknown', 'unknown', 'unknown', 'unknown', 'unknown',
       'unknown', 'unknown', 'unknown', 'unknown', 'unknown', 'unknown',
       'unknown', 'unknown', 'unknown', 'failure', 'unknown', 'failure',
       'unknown', 'unknown', 'failure', 'unknown', 'unknown', 'unknown',
       'failure', 'unknown', 'failure', 'unknown', 'unknown', 'other',
```

```
[11] y_pred = clf.predict(x_test)
```

```
[12] cm = confusion_matrix(y_test, y_pred)
```

```
[13] cm_df = pd.DataFrame(cm,
                          index = ['unknown','failure','other'],
                          columns = ['unknown','failure','other'])
```

```
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```