Roll No.**24** Exam Seat No._____

VIVEKANANDEDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg,Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that Mr. **NARENDER KESWANI [ROLL NO: 24]** of **FYMCA-1B** has satisfactorily completed a course of the necessary experiments in **MCAL21 - Artificial Intelligence and Machine Learning Lab** under the supervision of **Dr.Meenakshi Garg** in the Institute of Technology in the academic year **2021- 2022.**

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony, Chembur, Mumbai

Department of M.C.A

AI/ML INDEX

Sr. No	Contents	Date Of Preparation	Date Of Submission	Marks	Sign
1	Introduction to Python Programming				
2	Implementation of Logic programming using PROLOG- Basic of Prolog DFS for water jug				
3	Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn				
4	Implementation of Linear Regression, Logistic regression, KNN- classification				
5	Implementation of Decision Tree Using an Algorithm(ID3,C4.5,Gini) And Naïve Bayes Classfier				
6	Implementation and analysis of clustering algorithms like K-Means , K-medoid				
7	Implementation of Classifying data using Linear Support Vector Machines (SVMs).				
8	Implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).				
9	Implementation of Bagging Algorithm: Decision Tree, Random Forest.				

10	**Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.**				
11	**Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.**				
12	**Deployment of Machine Learning Models**				

OBJECTIVE:

INTRODUCTION TO PYTHON

COLAB:

Colaboratory by Google (Google Colab in short) is a Jupyter notebook based runtime environment which allows you to run code entirely on the cloud.

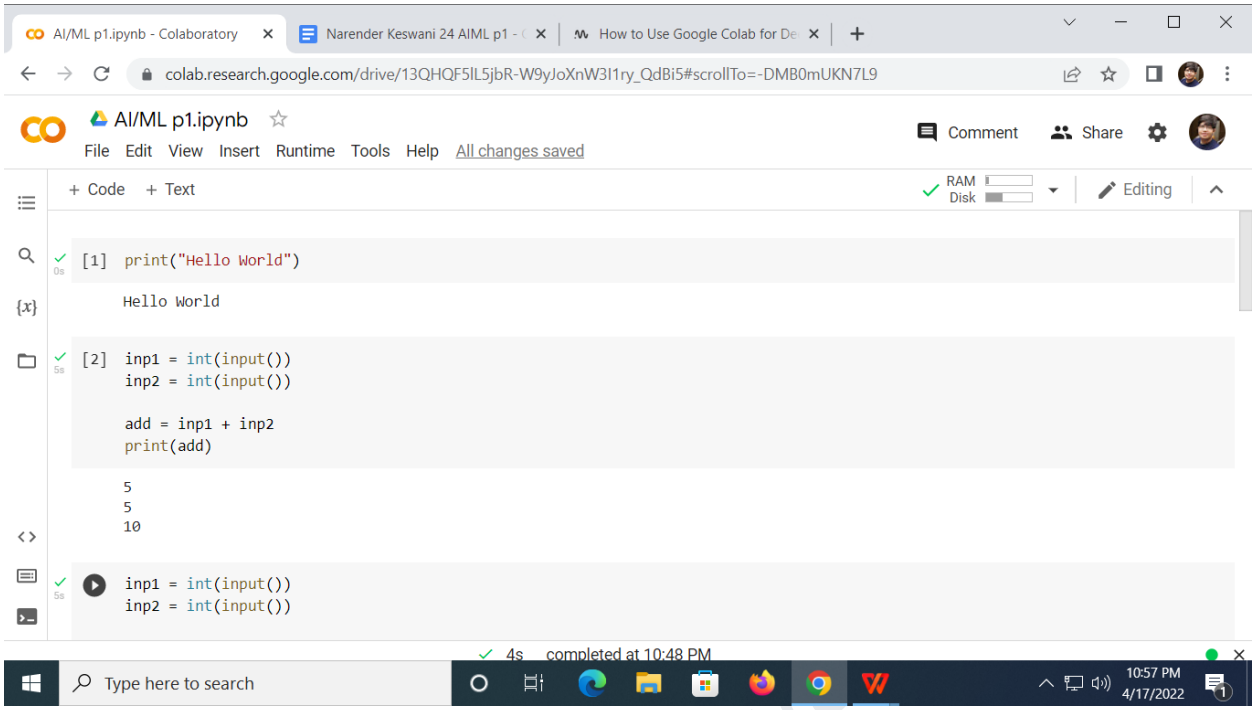
This is necessary because it means that you can train large scale ML and DL models even if you don't have access to a powerful machine or a high speed internet access.

Google Colab supports both GPU and TPU instances, which makes it a perfect tool for deep learning and data analytics enthusiasts because of computational limitations on local machines.

Since a Colab notebook can be accessed remotely from any machine through a browser, it's well suited for commercial purposes as well.

Creating your first .ipynb notebook in colab

Open a browser of your choice and go to colab.research.google.com and sign in using your Google account. Click on a new notebook to create a new runtime instance.



Pros:

- Pre-built with lots of python library
- Quick Start of python learning
- No infra setup required
- No charges for GPU usage
- Can run your code for 24 hrs without interruptions but not more than that
- Your notebooks is saved in google drive only

Cons:

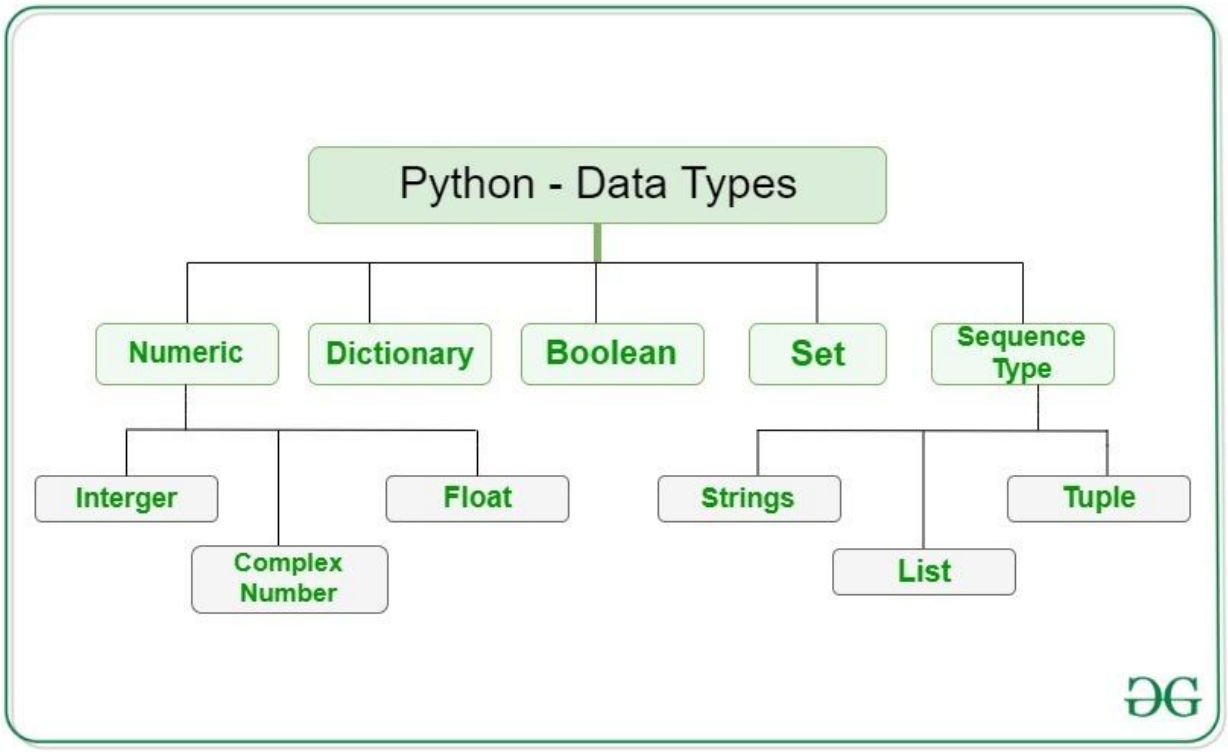
- Need to install all specific libraries which does not come with standard python (Need to repeat this with every session)
- Google Drive is your source and target for Storage, there are other like local (which eats your bandwidth if dataset is big)
- Google provided the code to connect and use with google drive but that will not work with lots of other data format
- Google Storage is used with current session, so if you have downloaded some file and want to use it later, better save it before closing the session.
- Difficult to work with BIGGER datasets as you have to download and store them in Google drive (15 GB Free space with gmail id, additional required a payment towards google)

DATA TYPES IN PYTHON:

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instances (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



1) Numeric

In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part)* + *(imaginary part)j*. For example – 2+3j

Note – type() function is used to determine the type of data type.

2) Sequence Type

In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

A) String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Creating String

Strings in Python can be created using single quotes or double quotes or even triple quotes.

B) List

Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating List

Lists in Python can be created by just placing the sequence inside the square brackets[].

C) Tuple

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple

In Python, tuples are created by placing a sequence of values separated by ‘comma’ with or without the use of parentheses for grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

Note: Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing ‘comma’ to make it a tuple.

Note – Creation of Python tuple without the use of parentheses is known as Tuple Packing.

3) Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

4) Set

In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by 'comma'. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

5) Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{}.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

A) HELLO WORLD:

```
print("Hello World")
```

```
Hello World
```

B) ARITHMETIC OPERATIONS:

1) ADDITION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

add = inp1 + inp2
print("Addition of two numbers is " + str(add))
```

```
Enter number 1: 5
Enter number 2: 10
Addition of two numbers is 15
```

2) SUBTRACTION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

sub = inp1 - inp2
print("Subtraction of two numbers is " + str(sub))
```

```
Enter number 1: 10
Enter number 2: 5
Subtraction of two numbers is 5
```

3) MULTIPLICATION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

mul = inp1 * inp2
print("Multiplication of two numbers is " + str(mul))
```

```
Enter number 1: 5
Enter number 2: 5
Multiplication of two numbers is 25
```

4) DIVISION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

div = inp1 / inp2
print("Division of two numbers is " + str(div))
```

```
Enter number 1: 10
Enter number 2: 5
Division of two numbers is 2.0
```

C) AREA OF CIRCLE:

```
import math
radius = int(input("Enter radius for circle: "))
areaOfCircle = math.pi * radius * radius
print("Area of Circle is ",areaOfCircle)
```

```
Enter radius for circle: 5
Area of Circle is  78.53981633974483
```

D) AREA OF TRIANGLE:

```
b = int(input("Enter breadth for triangle: "))
h = int(input("Enter height for triangle: "))

areaOfTriangle = 0.5 * b * h
print("Area of Triangle is ",areaOfTriangle)
```

```
Enter breadth for triangle: 5
Enter height for triangle: 10
Area of Triangle is  25.0
```

E) EVEN NUMBERS:

```
evenTerms = int(input(""))
print("Following are the even terms from 0 to", str(evenTerms))
for i in range((evenTerms*2)+1):
  if(i%2)==0:
    print(i)
```

```
10
Following are the even terms from 0 to 10
0
2
4
6
8
10
12
14
16
18
20
```

F) ODD NUMBERS:

```
oddTerms = int(input())
print("Following are the odd terms from 0 to", str(oddTerms))
for i in range((oddTerms*2)+1):
  if(i%2)!=0:
    print(i)
```

```
10
Following are the odd terms from 0 to 10
1
3
5
7
9
11
13
15
17
19
```

G) GENERATE MULTIPLICATION TABLE USING LOOPS:

```
print("Multiplication Table:")
tableToPrint = int(input("Enter number "))
for i in range(11):
  print(str(tableToPrint) + " x " + str(i) + " = " + str(i * tableToPrint))
```

```
Multiplication Table:
10
10 x 0 = 0
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

H) WHILE LOOP:

WHILE LOOP

```
num_of_times = 1

while num_of_times <= 7:
  print("Narender Keswani")
  num_of_times += 1
```

```
Narender Keswani
Narender Keswani
Narender Keswani
Narender Keswani
Narender Keswani
Narender Keswani
Narender Keswani
```

I) PYRAMID:

```
[ ] rows = int(input("Enter number of rows: "))

    for i in range(rows):
        for j in range(i+1):
            print("Narender ", end="")
        print("\n")
```

```
Enter number of rows: 5
Narender

Narender Narender

Narender Narender Narender

Narender Narender Narender Narender

Narender Narender Narender Narender Narender
```

J) PRIME NUMBER CHECK:

10 PRIME NUMBER CHECK

```
[ ] num = int(input("Enter a number: "))

    flag = False

    if num > 1:
        for i in range(2, num):
            if (num % i) == 0:
                flag = True
                break

    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")
```

```
Enter a number: 5
5 is a prime number
```

K) GENERATE PRIME NUMBERS:

11 GENERATE PRIME NUMBERS

```
[1] lower_value = int(input ("Please, Enter the Lowest Range Value: "))
    upper_value = int(input ("Please, Enter the Upper Range Value: "))

    print ("The Prime Numbers in the range are: ")
    for number in range (lower_value, upper_value + 1):
        if number > 1:
            for i in range (2, number):
                if (number % i) == 0:
                    break
            else:
                print (number)
```

```
Please, Enter the Lowest Range Value: 5
Please, Enter the Upper Range Value: 15
The Prime Numbers in the range are:
5
7
11
13
```

L) PALINDROME NUMBER CHECK:

12 PALINDROME NUMBER CHECK

```
def isPalindrome(s):
  return s == s[::-1]

s = "malayalam"
ans = isPalindrome(s)

if ans:
  print("Yes")
else:
  print("No")
```

```
Yes
```

M) TAKE INPUT FROM USER(ROLL, NAME, DOB, ADDRESS) AND STORE IN ARRAY DATATYPE

13 TAKE INPUT FROM USER (ROLL, NAME, DOB,ADDRESS) AND STORE IN ARRAY DATATYPE

```
roll = int(input("Enter roll "))
name = input("Enter name ")
dob = input("Enter dob ")
address = input("Enter address ")

thisdict = {
  "roll": roll,
  "name": name,
  "dob": dob,
  "address":address
}

for x, y in thisdict.items():
  print("The value of ",x, "is ", y)
```

```
Enter roll 24
Enter name Narender Keswani
Enter dob 10/11/1999
Enter address Ulhasnagar
The value of  roll is  24
The value of  name is  Narender Keswani
The value of  dob is  10/11/1999
The value of  address is  Ulhasnagar
```

PYTHON EXERCISE CRASH COURSE:

A) What is 7 to the power of 4?

```
7**4
```

```
2401
```

B) Split this string

```
[2] s = "Hi there Narender Keswani!"
    x = s.split()
    print(x)
```

```
['Hi', 'there', 'Narender', 'Keswani!']
```

C) Use .format() to print the following string

```
[13] planet = "Earth"
     diameter = 12742
```

```
[14] print("The diameter of {} is {} kilometers.".format(planet, diameter))
```

```
The diameter of Earth is 12742 kilometers.
```

D) Given this nested list, use indexing to grab the word "hello"

```
[7] lst = [1,2,[3,4],[5,[100,200,['hello']],23,11],1,7]
```

```
[11] print(lst[3][1][2])
```

```
['hello']
```

E) Given this nested dictionary grab the word "hello". Be prepared, this will be annoying/tricky

```
d = {'k1':[1,2,3,{'tricky':['oh','man','inception',{'target':[1,2,3,'hello']}]}]}
```

```
[19] d['k1'][3]['tricky'][3]['target'][3]

'hello'
```

F) Difference between list and tuple:

SR.NO.	LIST	TUPLE
1	Lists are mutable	Tuples are immutable
2	Implication of iterations is Time-consuming	The implication of iterations is comparatively Faster
3	The list is better for performing operations, such as insertion and deletion.	Tuple data type is appropriate for accessing the elements
4	Lists consume more memory	Tuple consume less memory as compared to the list
5	Lists have several built-in methods	Tuple does not have many built-in methods.
6	The unexpected changes and errors are more likely to occur	In tuple, it is hard to take place.

G) Create a function that grabs the email website domain from a string in the form

```
[22] def domainGet(email):
       a = email.split('@')
       return a[1]
```

```
[23] domainGet('user@domain.com')
```

```
'domain.com'
```

H) Create a basic function that returns True if the word 'dog' is contained in the input string. Don't worry about edge cases like a punctuation being attached to the word dog, but do account for capitalization.

```
[39] def findDog(dog):
       if "Dog" in dog or "dog" in dog:
         return True
       else:
         return False
```

```
findDog('Is there a dog here?')
```

```
True
```

I) Create a function that counts the number of times the word "dog" occurs in a string. Again ignore edge cases.

```
[31] def countDog(c):
       return c.count('dog')
```

```
[32] countDog('This dog runs faster than the other dog dude!')
```

```
2
```

J) Use lambda expressions and the filter() function to filter out words from a list that don't start with the letter 's'.

```
[33] seq = ['soup','dog','salad','cat','great']
```

```
[35] list(filter(lambda item: item.startswith('s'),seq))
```

```
['soup', 'salad']
```

K) You are driving a little too fast, and a police officer stops you. Write a function to return one of 3 possible results: "No ticket", "Small ticket", or "Big Ticket". If your speed is 60 or less, the result is "No Ticket". If speed is between 61 and 80 inclusive, the result is "Small Ticket". If speed is 81 or more, the result is "Big Ticket". Unless it is your birthday (encoded as a boolean value in the parameters of the function) -- on your birthday, your speed can be 5 higher in all cases.

```
[36] def caught_speeding(speed, is_birthday):
       if is_birthday:
         speed = speed - 5
       if speed > 80:
         return 'Big Ticket'
       elif speed > 60:
         return 'Small Ticket'
       else:
         return 'No Ticket'
```

```
[37] caught_speeding(81,True)
```

```
'Small Ticket'
```

```
[38] caught_speeding(81,False)
```

```
'Big Ticket'
```

Aim: Write down to libraries used for the following machine learning applications

1. Scikit-learn(sklearn):

- Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
- In python, sklearn is a machine learning package which includes a lot of ML algorithms.

```
[9] import sklearn as sc
    print(sc)

    <module 'sklearn' from '/usr/local/lib/python3.7/dist-packages/sklearn/__init__.py'>
```

2. NumPy:

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.
- It is a numeric python module which provides fast maths functions for calculations.
- It is used to read data in numpy arrays and for manipulation purpose.

```
[2] import numpy as np
    print(np.pi)

    3.141592653589793
```

3. Pandas:

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Used to read and write different files such as csv.
- Data manipulation can be done easily with dataframes.

```
import pandas as pd
print(pd)

<module 'pandas' from '/usr/local/lib/python3.7/dist-packages/pandas/__init__.py'>
```

+ Code + Text

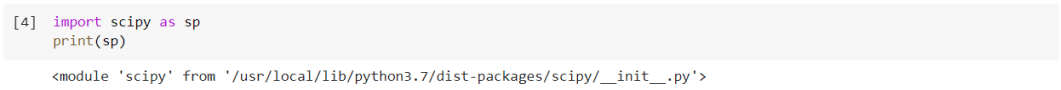
4. Matplotlib:

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

- One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc

5. SciPy:

- SciPy is an open-source Python library which is used to solve scientific and mathematical problems. It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.



ALGORITHM NAME	NUMPY	PANDAS	SCIKT-LEARN (SKLEARN)	MATPLOTLIB	XGBOOST
REGRESSION					
LINEAR	Y	Y	Y	Y	
LOGISTIC	Y	Y	Y	Y	
KNN	Y	Y	Y	Y	
CLASSIFICATION					
ID3	Y	Y	Y	Y	
C4.5	Y	Y	Y	Y	
NAÏVE BAYES	Y	Y	Y	Y	
CLUSTERING					
K-MEANS	Y	Y	Y	Y	
K-MEDOID	Y	Y	Y	Y	
SVM	Y	Y	Y	Y	
RANDOM FOREST	Y	Y	Y	Y	
XGBOOST	Y	Y	Y	Y	Y

Numpy Applications:

1. An alternative for lists and arrays in Python

Arrays in Numpy are equivalent to lists in python. Like lists in python, the Numpy arrays are homogenous sets of elements. The most important feature of NumPy arrays is they are homogenous in nature.

This differentiates them from python arrays. It maintains uniformity for mathematical operations that would not be possible with heterogeneous elements. Another benefit of using NumPy arrays is there are a large number of functions that are applicable to these arrays.

These functions could not be performed when applied to python arrays due to their heterogeneous nature.

2. NumPy maintains minimal memory

Arrays in NumPy are objects. Python deletes and creates these objects continually, as per the requirements. Hence, the memory allocation is less as compared to Python lists. NumPy has features to avoid memory wastage in the data buffer.

It consists of functions like copies, view, and indexing that helps in saving a lot of memory. Indexing helps to return the view of the original array, that implements reuse of the data. It also specifies the data type of the elements which leads to code optimization.

3. Using NumPy for multi-dimensional arrays

We can also create multi-dimensional arrays in NumPy.These arrays have multiple rows and columns. These arrays have more than one column that makes these multi-dimensional. Multi-dimensional array implements the creation of matrices.

These matrices are easy to work with. With the use of matrices the code also becomes memory efficient. We have a matrix module to perform various operations on these matrices.

4. Mathematical operations with NumPy

Working with NumPy also includes easy to use functions for mathematical computations on the array data set. We have many modules for performing basic and special mathematical functions in NumPy.

There are functions for Linear Algebra, bitwise operations, Fourier transform, arithmetic operations, string operations, etc.

Numpy Array Applications

1. Shape Manipulations

Users can change array dimensions at runtime if the output produces the same number of elements. We apply np.reshape(…)function on the array. The reshape function is useful for performing various operations. For eg, we use it when we want to broadcast two dissimilar arrays.

2. Array Generation

We can generate array data set for implementing various functions. We can also generate a predefined set of numbers for the array elements using the np.arrange(…)function. Reshape function is useful to generate a different set of dimensions.

We can also use the random function to generate an array having random values. Similarly, we can use linspace function to generate arrays having similar spacing in elements.

We can create arrays with pre-filled ones or zeroes. The default data type is set to be float64 but we can edit the data type using dtype option.

3. Array Dimensions

Numpy consists of both one and multidimensional arrays. Some functions have restrictions on multidimensional arrays. It is then necessary to transform those arrays into one-dimensional arrays. We can transform multi-dimensional to single dimension using np.ravel(..)

Numpy Applications with Other Libraries

1. NumPy with Pandas

Pandas is one of the most important libraries in python for data analysis. Pandas provide high performance, fast analysis, and data cleaning. We use it to manipulate data structures and have data analysis tools.

It consists of a data frame object. It interoperates with NumPy for faster computations. When we use both the libraries together it is a very helpful resource for scientific computations.

2. NumPy with Matplotlib

Matplotlib is a module in NumPy. It is a very helpful tool to work with graphical representations. It consists of a wide range of functions to plot graphs and also manipulate them.

This combination can replace the functionalities of MatLab. It is used to generate the graphs of the results. We enhance it further with the use of graphic toolkits like PyQt and wxPython.

3. NumPy with SciPy

Scipy is an open-source library in Python. It is the most important scientific library in python. It has been built upon the functionalities of NumPy.There are advanced functionalities in SciPy for scientific computations.

We can combine it with NumPy for greater mathematical performance. The combination helps in the implementation of complex scientific operations.

ALGORITHMS:

A) Linear Regression:

It is one of the best statistical models that studies the relationship between a dependent variable (Y) with a given set of independent variables (X). The relationship can be established with the help of fitting a best line.

sklearn.linear_model.LinearRegression is the module used to implement linear regression.

B) Logistic Regression:

Logistic regression, despite its name, is a classification algorithm rather than regression algorithm. Based on a given set of independent variables, it is used to estimate discrete value (0 or 1, yes/no, true/false). It is also called logit or MaxEnt Classifier.

Basically, it measures the relationship between the categorical dependent variable and one or more independent variables by estimating the probability of occurrence of an event using its logistics function.

sklearn.linear_model.LogisticRegression is the module used to implement logistic regression.

C) K-NN:

k-NN (k-Nearest Neighbor), one of the simplest machine learning algorithms, is non-parametric and lazy in nature. Non-parametric means that there is no assumption for the underlying data distribution i.e. the model structure is determined from the dataset. Lazy or instance-based learning means that for the purpose of model generation, it does not require any training data points and whole training data is used in the testing phase.

sklearn.neighbors.NearestNeighbors is the module used to implement unsupervised nearest neighbor learning. It uses specific nearest neighbor algorithms named BallTree, KDTree or Brute Force. In other words, it acts as a uniform interface to these three algorithms.

D) Decision Tree:

Decisions tress (DTs) are the most powerful non-parametric supervised learning method. They can be used for the classification and regression tasks. The main goal of DTs is to create a model predicting target variable value by learning simple decision rules deduced from the data features. Decision trees have two main entities; one is root node, where the data splits, and other is decision nodes or leaves, where we got final output.

Decision Tree Algorithms

Different Decision Tree algorithms are explained below −

ID3 algorithm stands for Iterative Dichotomiser 3

It was developed by Ross Quinlan in 1986. It is also called Iterative Dichotomiser 3. The main goal of this algorithm is to find those categorical features, for every node, that will yield the largest information gain for categorical targets.

It lets the tree to be grown to their maximum size and then to improve the tree's ability on unseen data, applies a pruning step. The output of this algorithm would be a multiway tree.

C4.5

It is the successor to ID3 and dynamically defines a discrete attribute that partition the continuous attribute value into a discrete set of intervals. That's the reason it removed the restriction of categorical features. It converts the ID3 trained tree into sets of 'IF-THEN' rules.

In order to determine the sequence in which these rules should applied, the accuracy of each rule will be evaluated first.

C5.0

It works similar as C4.5 but it uses less memory and build smaller rulesets. It is more accurate than C4.5.

CART

It is called Classification and Regression Trees alsgorithm. It basically generates binary splits by using the features and threshold yielding the largest information gain at each node (called the Gini index).

Homogeneity depends upon Gini index, higher the value of Gini index, higher would be the homogeneity. It is like C4.5 algorithm, but, the difference is that it does not compute rule sets and does not support numerical target variables (regression) as well.

Classification with decision trees

In this case, the decision variables are categorical.

Sklearn Module − The Scikit-learn library provides the module name **DecisionTreeClassifier** for performing multiclass classification on dataset.

1	***criterion*** *− string, optional default= "gini"* It represents the function to measure the quality of a split. Supported criteria are "gini" and "entropy". The default is gini which is for Gini impurity while entropy is for the information gain.

E) KMeans:

This algorithm computes the centroids and iterates until it finds optimal centroid. It requires the number of clusters to be specified that's why it assumes that they are already known. The main logic of this algorithm is to cluster the data separating samples in n number of groups of equal variances by minimizing the criteria known as the inertia. The number of clusters identified by algorithm is represented by 'K. Scikit-learn have **sklearn.cluster.KMeans** module to perform K-Means clustering. While computing cluster centers and value of inertia, the parameter

named **sample_weight** allows **sklearn.cluster.KMeans** module to assign more weight to some samples.

F) SVM:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine

G) Random Forest:

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

H) XGBoost:

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

CONCLUSION:

From this practical, I have successfully learned about libraries which are used for the machine learning applications.

A) Import NumPy as np

```
[1]  import numpy as np
```

B) Create an array of 10 zeros

```
[2]  np.zeros(10)

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

C) Create an array of 10 ones

```
[3]  np.ones(10)

array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

D) Create an array of 10 fives

```
np.ones(10)*5

array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

E) Create an array of the integers from 10 to 50

```
[5]  np.arange(10,51)

array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
       44, 45, 46, 47, 48, 49, 50])
```

F) Create an array of all the even integers from 10 to 50

```
np.arange(10,51,2)

array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48, 50])
```

G) Create a 3x3 matrix with values ranging from 0 to 8

```
np.arange(0, 9).reshape(3,3)
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

H) Create a 3x3 identity matrix

```
[8] np.eye(3)
```

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

I) Use NumPy to generate a random number between 0 and 1

```
[9] np.random.rand(1)
```

```
array([0.63995357])
```

J) Use NumPy to generate an array of 25 random numbers sampled from a standard normal distribution

```
[10] np.random.randn(5,5)
```

```
array([[ 0.00917627,  2.54296586,  0.05459417, -0.16396732,  0.48492437],
       [ 2.59090973, -0.4354634 ,  0.18801321,  1.21906203, -0.09548617],
       [ 0.10950666,  0.73144329,  1.55376488, -1.65816763, -0.30470597],
       [-0.66117415, -0.14516945, -0.00943456, -0.45216566,  1.15621409],
       [-0.13117192,  3.63209751,  0.67736721, -0.06525428,  0.5618076 ]])
```

K) Create the following matrix:

```
[11] np.arange(0.01, 1.01, 0.01).reshape(10,10)
```

```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

L) Create an array of 20 linearly spaced points between 0 and 1:

```
[12] np.linspace(0,1,20)

array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

Numpy Indexing and Selection

M) Now you will be given a few matrices, and be asked to replicate the resulting matrix outputs:

```
[13] mat = np.arange(1,26).reshape(5,5)
     mat

array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
[15] mat1 = np.array([12,13,14,15,17,18,19,20,22,23,24,25]).reshape(3,4)
     mat1

array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

```
[17] mat1[1][3]

20
```

```
[19] mat2 = np.arange(2,13,5).reshape(3,1)
     mat2

array([[ 2],
       [ 7],
       [12]])
```

```
[21] mat3 = np.arange(21,26)
     mat3
```

```
array([21, 22, 23, 24, 25])
```

```
[23] mat4 = np.arange(16,26).reshape(2,5)
     mat4
```

```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

N) Get the sum of all the values in mat

```
[24] np.sum(mat)
```

```
325
```

O) Get the standard deviation of the values in mat

```
[25] np.std(mat)
```

```
7.211102550927978
```

P) Get the sum of all the columns in mat

```
[26] np.sum(mat,axis=0)
```

```
array([55, 60, 65, 70, 75])
```

CONCLUSION:

From this practical, I have successfully learned about numpy library in python.

PANDAS:

A) IMPORTING PANDAS & READING DATASET:

```
import pandas as pd
```

```
sal = pd.read_csv('Salaries.csv')
```

```
sal.head()
```

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay	TotalPayBenefits	Year	Notes	Agency	Status
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.00	400184.25	NaN	567595.43	567595.43	2011	NaN	San Francisco	NaN
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	538909.28	538909.28	2011	NaN	San Francisco	NaN
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.60	NaN	335279.91	335279.91	2011	NaN	San Francisco	NaN
3	4	CHRISTOPHER CHONG	WIRE ROPE CABLE MAINTENANCE MECHANIC	77916.00	56120.71	198306.90	NaN	332343.61	332343.61	2011	NaN	San Francisco	NaN
4	5	PATRICK GARDNER	DEPUTY CHIEF OF DEPARTMENT,(FIRE DEPARTMENT)	134401.60	9737.00	182234.59	NaN	326373.19	326373.19	2011	NaN	San Francisco	NaN

B) Use the .info() method to find out how many entries there are

```
sal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to 148653
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Id                148654 non-null  int64
 1   EmployeeName      148654 non-null  object
 2   JobTitle          148654 non-null  object
 3   BasePay           148045 non-null  float64
 4   OvertimePay       148650 non-null  float64
 5   OtherPay          148650 non-null  float64
 6   Benefits          112491 non-null  float64
 7   TotalPay          148654 non-null  float64
 8   TotalPayBenefits  148654 non-null  float64
 9   Year              148654 non-null  int64
 10  Notes             0 non-null       float64
 11  Agency            148654 non-null  object
 12  Status            0 non-null       float64
dtypes: float64(8), int64(2), object(3)
memory usage: 14.7+ MB
```

C) **What is the average BasePay ?**

```
sal['BasePay'].mean()
```

```
66325.4488404877
```

D) **What is the highest amount of OvertimePay in the dataset ?**

```
sal['OvertimePay'].max()
```

```
245131.88
```

E) **What is the job title of JOSEPH DRISCOLL ? Note: Use all caps, otherwise you may get an answer that doesn't match up (there is also a lowercase Joseph Driscoll).**

```
sal[sal['EmployeeName']=='JOSEPH DRISCOLL']['JobTitle']
```

```
24    CAPTAIN, FIRE SUPPRESSION
Name: JobTitle, dtype: object
```

F) **How much does JOSEPH DRISCOLL make (including benefits)?**

```
sal[sal['EmployeeName']=='JOSEPH DRISCOLL']['TotalPayBenefits']
```

```
24    270324.91
Name: TotalPayBenefits, dtype: float64
```

G) **What is the name of highest paid person (including benefits)?**

```
ind = sal['TotalPayBenefits'].idxmax()
sal.loc[ind]['EmployeeName']
```

```
'NATHANIEL FORD'
```

H) **What is the name of lowest paid person (including benefits)? Do you notice something strange about how much he or she is paid?**

```
ind = sal['TotalPayBenefits'].idxmin()
sal.iloc[ind]
```

```
Id                                   148654
EmployeeName                      Joe Lopez
JobTitle           Counselor, Log Cabin Ranch
BasePay                                 0.0
OvertimePay                             0.0
OtherPay                             -618.13
Benefits                                0.0
TotalPay                             -618.13
TotalPayBenefits                     -618.13
Year                                    2014
Notes                                    NaN
Agency                         San Francisco
Status                                   NaN
Name: 148653, dtype: object
```

I) What was the average (mean) BasePay of all employees per year? (2011-2014) ?

```
sal.groupby('Year').mean()['BasePay']
```

```
Year
2011    63595.956517
2012    65436.406857
2013    69630.030216
2014    66564.421924
Name: BasePay, dtype: float64
```

J) How many unique job titles are there?

```
sal['JobTitle'].nunique()
```

```
2159
```

K) What are the top 5 most common jobs?

```
sal['JobTitle'].value_counts().head()
```

```
Transit Operator                7036
Special Nurse                   4389
Registered Nurse                3736
Public Svc Aide-Public Works    2518
Police Officer 3                2421
Name: JobTitle, dtype: int64
```

L) How many Job Titles were represented by only one person in 2013? (e.g. Job Titles with only one occurence in 2013?)

```
(sal[sal['Year']==2013]['JobTitle'].value_counts()==1).sum()
```

```
202
```

M) How many people have the word Chief in their job title?

```
def chief_string(title):
    if 'chief' in title.lower().split():
        return True
    else:
        return False
sum(sal['JobTitle'].apply(lambda x:chief_string(x)))
```

```
477
```

N) Is there a correlation between length of the Job Title string and Salary?

```
sal['title_len']=sal['JobTitle'].apply(len)
sal[['TotalPayBenefits','title_len']].corr()
```

	TotalPayBenefits	title_len
TotalPayBenefits	1.000000	-0.036878
title_len	-0.036878	1.000000

CONCLUSION:

From this practical, I have successfully learned about pandas library in python.

SCIKIT-LEARN:

A) IMPORTING LIBRARIES, DATASET & READING DATASET:

```
import sklearn
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# store the feature and target names
feature_names = iris.feature_names
target_names = iris.target_names

# printing features and target names of our dataset
print("Feature names:", feature_names)
print("Target names:", target_names)

# X and y are numpy arrays
print("\nType of X is:", type(X))

# printing first 5 input rows
print("\nFirst 5 rows of X:\n", X[:5])
```

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

Type of X is: <class 'numpy.ndarray'>

First 5 rows of X:
 [[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]]
```

B) SPLITTING DATASET INTO TRAINING & TESTING DATASET:

```
# splitting X and y into training and testing sets
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=1)

# printing the shapes of the new X objects
print('X Train Data: ',X_train.shape)
print('X Test Data: ',X_test.shape)

# printing the shapes of the new y objects
print('Y Train Data: ',y_train.shape)
print('Y Test Data: ',y_test.shape)
```

```
X Train Data:  (90, 4)
X Test Data:  (60, 4)
Y Train Data:  (90,)
Y Test Data:  (60,)
```

C) NORMALIZATION:

```
# import module
from sklearn.preprocessing import StandardScaler

# create data
data = [[11, 2], [3, 7], [0, 10], [11, 8]]

# compute required values
scaler = StandardScaler()
model = scaler.fit(data)
scaled_data = model.transform(data)

# print scaled data
print(scaled_data)
```

```
[[ 0.97596444 -1.61155897]
 [-0.66776515  0.08481889]
 [-1.28416374  1.10264561]
 [ 0.97596444  0.42409446]]
```

SCIPY:

A) SPARSE MATRIX:

```
# import necessary modules
from scipy import sparse
# Row-based linked list sparse matrix
A = sparse.lil_matrix((1000, 1000))
print(A)

A[0,:100] = np.random.rand(100)
A[1,100:200] = A[0,:100]
A.setdiag(np.random.rand(1000))
print(A)
```

```
(0, 0)        0.19036855447545786
(0, 1)        0.33776231739432405
(0, 2)        0.9749038090665604
(0, 3)        0.8942583911186659
(0, 4)        0.08515061548006031
(0, 5)        0.21431379796449723
(0, 6)        0.8676941823129108
(0, 7)        0.07258083664498649
(0, 8)        0.5557847427639602
(0, 9)        0.7459339793334242
(0, 10)       0.3857750726237723
(0, 11)       0.8832801032093758
(0, 12)       0.6813002996002154
(0, 13)       0.6296641368561785
(0, 14)       0.16101650416423774
(0, 15)       0.4873200634402388
(0, 16)       0.1806188884055946
(0, 17)       0.28177347650237183
(0, 18)       0.19815175912132565
(0, 19)       0.4342894381136917
(0, 20)       0.6364320992619823
(0, 21)       0.21080807130051238
(0, 22)       0.020340881969629243
(0, 23)       0.6116070621487552
(0, 24)       0.9183747764032624
(0, 25)       0.031451450420754035
(0, 26)       0.3871320802870829
```

B) <u>CONVERT THIS MATRIX TO COMPRESSED SPARSE ROW FORMAT</u>

```
from scipy.sparse import linalg

# Convert this matrix to Compressed Sparse Row format.
A.tocsr()

A = A.tocsr()
b = np.random.rand(1000)
ans = linalg.spsolve(A, b)
# it will print ans array of 1000 size
print(ans)
```

```
[-2.35319469e+02 -1.49842949e+02  9.73797985e-01  8.01662749e-01
  2.56669454e-01  3.23166234e-01  2.86738795e-01  1.84657087e+00
  4.48405102e-01  9.84638653e-02  4.44847188e-01  2.88717900e-01
  4.44918552e-02  4.58318163e-01  1.43384483e+00  1.99350174e+00
  1.21888495e+00  1.57496174e+00  6.70136481e-01  3.59897327e+01
  1.28297548e+00  1.96483683e+00  3.49994061e+00  5.23922832e-01
  2.06604672e+00  1.84576603e+00  1.56911917e-01  1.64625356e+00
  1.54586678e+00  3.70630437e-01  9.22407476e-01  1.25307475e+01
  1.27201263e+00  8.69771911e-01  5.71008317e-01  2.36020605e+00
  2.52519006e+00  2.86415071e-01  4.45305112e-01  1.88676711e+00
  9.66215505e-01  5.54360157e-01  5.13487903e+00  1.35910055e+00
  1.65365834e+00  6.86115676e+00  1.43536015e+00  1.22939297e+00
  3.39476772e-01  1.33685644e+00  3.62356948e+00  2.53105349e+00
  1.06194083e+02  4.53304440e+00  3.69673226e-01  1.44402506e+00
  9.76192380e-01  7.12327777e-01  1.37028728e+00  9.17760860e-01
  6.89348469e-01  2.13957351e+00  9.28235879e+00  9.67851414e-01
  2.79828865e-01  4.13236913e-01  1.06858116e+00  1.79529384e+00
  6.55611974e+00  8.26922782e-01  3.55375831e+01  5.22998358e-02
  8.53147778e-01  6.09557326e-01  1.60171443e+00  1.13057652e+00
  1.99948661e+00  3.45236527e-01  5.45060752e+00  9.86027796e-01
  5.54949985e-02  1.34151479e+00  6.30784392e-01  1.93189946e+00
  2.39506893e+01  1.89743431e-01  3.16347084e-01  8.24358555e-02
  4.38889999e-01  2.73188918e+00  7.30921368e-01  1.07383848e+00
  4.00896310e-02  1.03398614e+00  7.16866466e-01  3.02780953e-02
  2.44974913e+00  1.46451142e+00  2.79801309e-01  2.32484949e+00
  3.28744365e-01  1.37912374e-01  1.71207446e-01  1.45246897e-01
  9.09435792e-01  1.22430643e+00  2.83132823e+00  7.70920938e-01
```

C) <u>DETEMINANT:</u>

```
 # import numpy library
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,8]])
# importing linalg function from scipy
from scipy import linalg

# Compute the determinant of a matrix
linalg.det(A)
```

```
3.0
```

D) <u>DOT PRODUCT:</u>

```
P, L, U = linalg.lu(A)
print(P)
print(L)
print(U)
# print LU decomposition
print(np.dot(L,U))
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
[[1.         0.         0.        ]
 [0.14285714 1.         0.        ]
 [0.57142857 0.5        1.        ]]
[[7.         8.         8.        ]
 [0.         0.85714286 1.85714286]
 [0.         0.         0.5       ]]
[[7. 8. 8.]
 [1. 2. 3.]
 [4. 5. 6.]]
```

E) EIGEN VECTOR:

```
eigen_values, eigen_vectors = linalg.eig(A)
print(eigen_values)
print(eigen_vectors)
```

```
[15.55528261+0.j -1.41940876+0.j -0.13587385+0.j]
[[-0.24043423 -0.67468642  0.51853459]
 [-0.54694322 -0.23391616 -0.78895962]
 [-0.80190056  0.70005819  0.32964312]]
```

F) INTEGRATION:

```
from scipy import integrate
f = lambda y, x: x*y**2
i = integrate.dblquad(f, 0, 2, lambda x: 0, lambda x: 1)
# print the results
print(i)
```

```
(0.6666666666666667, 7.401486830834377e-15)
```

CONCLUSION:

From this practical, I have learned and implemented scikit-learn & scipy libraries in python.

MATHPLOTLIB:

A) IMPORT MATHPLOTLIB:

```
import numpy as np
x = np.arange(0,100)
y = x*2
z = x**2
```

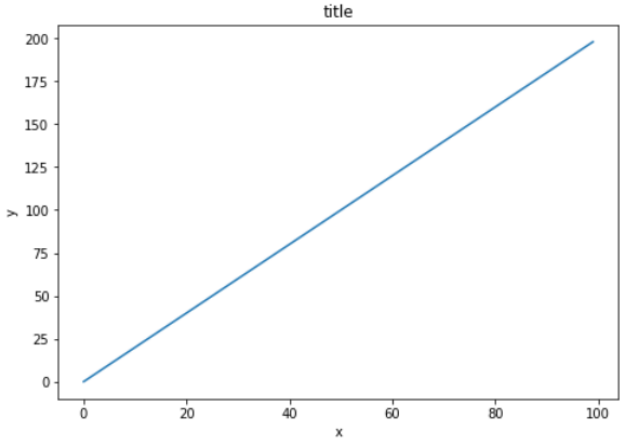
```
import matplotlib.pyplot as plt
%matplotlib inline
```

B) Follow along with these steps:

Create a figure object called fig using plt.figure()
Use add_axes to add an axis to the figure canvas at [0,0,1,1]. Call this new axis ax.
Plot (x,y) on that axes and set the labels and titles to match the plot below:

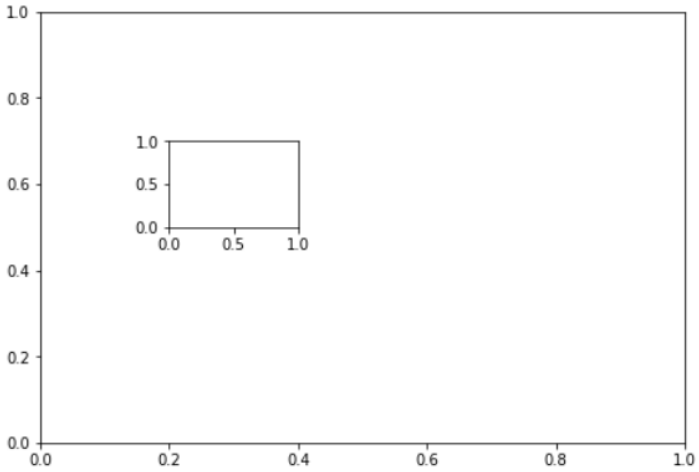
```
fig = plt.figure()
ax= fig.add_axes([0,0,1,1])
ax.plot(x,y)
ax.set_title('title')
ax.set_xlabel('x')
ax.set_ylabel('y')
```

```
Text(0, 0.5, 'y')
```



C) Create a figure object and put two axes on it, ax1 and ax2. Located at [0,0,1,1] and [0.2,0.5,.2,.2] respectively.

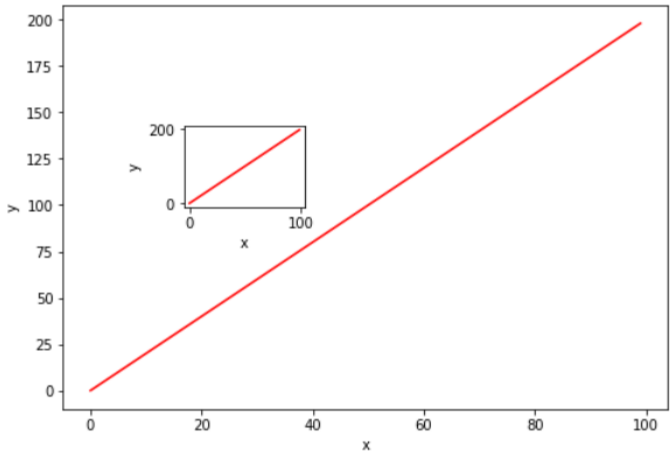
```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax2 = fig.add_axes([0.2,0.5,.2,.2])
```



D) Now plot (x,y) on both axes. And call your figure object to show it.

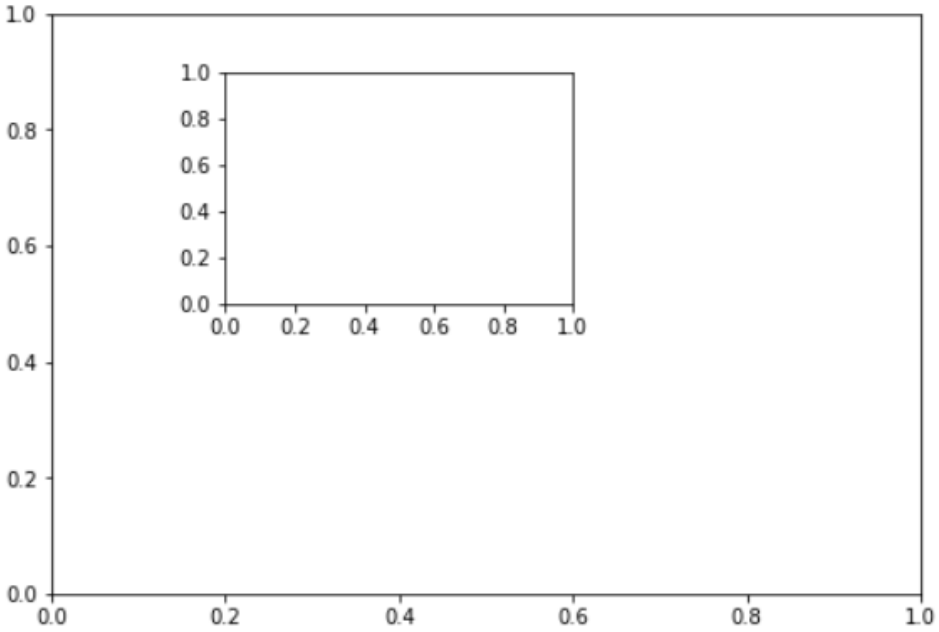
```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax1.plot(x,y,'red')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax2 = fig.add_axes([0.2,0.5,.2,.2])
ax2.plot(x,y,'red')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
```

```
Text(0, 0.5, 'y')
```



E) Create the plot below by adding two axes to a figure object at [0,0,1,1] and [0.2,0.5,.4,.4]

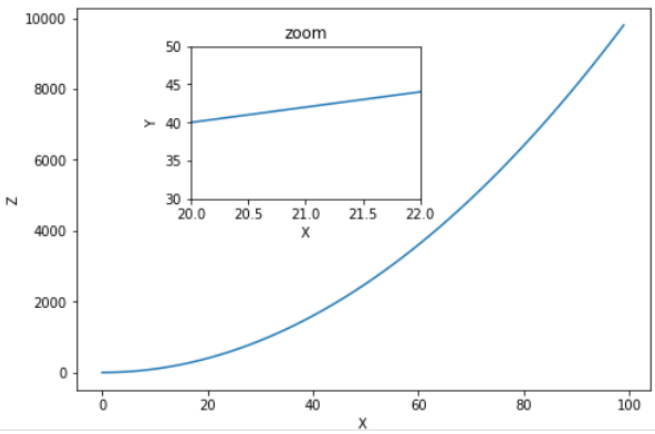
```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax2 = fig.add_axes([0.2,0.5,.4,.4])
```



F) **Now use x,y, and z arrays to recreate the plot below. Notice the xlimits and y limits on the inserted plot:**

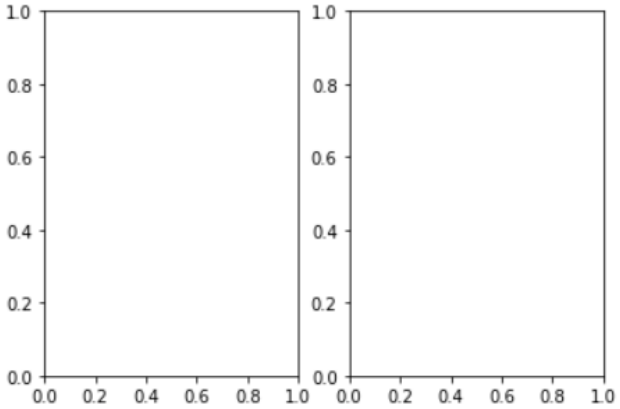
```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax1.plot(x,z)
ax1.set_xlabel('X')
ax1.set_ylabel('Z')
ax2 = fig.add_axes([0.2,0.5,.4,.4])
ax2.plot(x,y)
ax2.set_title('zoom')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_xlim(20,22)
ax2.set_ylim(30,50)
```

```
(30.0, 50.0)
```



G) Use plt.subplots(nrows=1, ncols=2) to create the plot below.

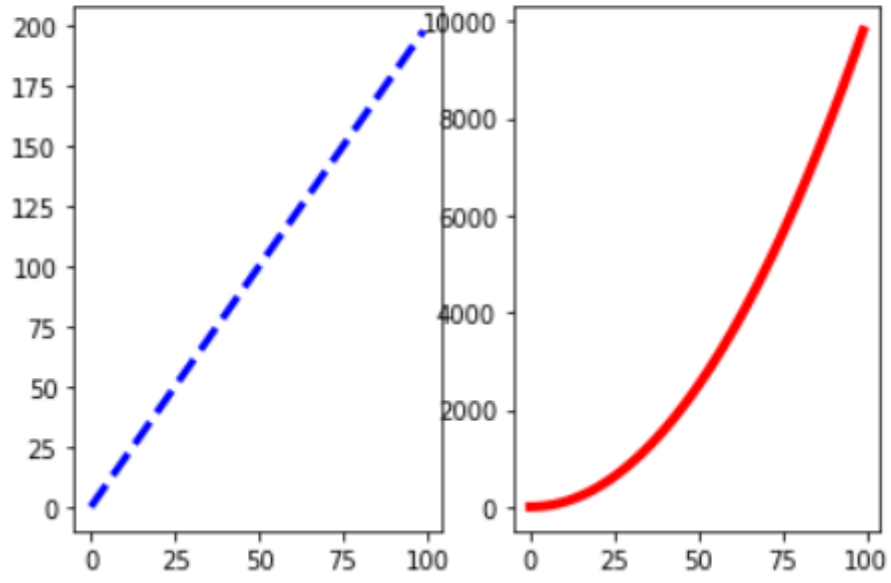
```
fig, axes = plt.subplots(nrows=1, ncols=2)
```



H) Now plot (x,y) and (x,z) on the axes. Play around with the linewidth and style

```
fig, axes = plt.subplots(nrows=1, ncols=2)
axes[0].plot(x,y,color='blue',lw=3,ls='--')
axes[1].plot(x,z,color='red',lw=4)
```

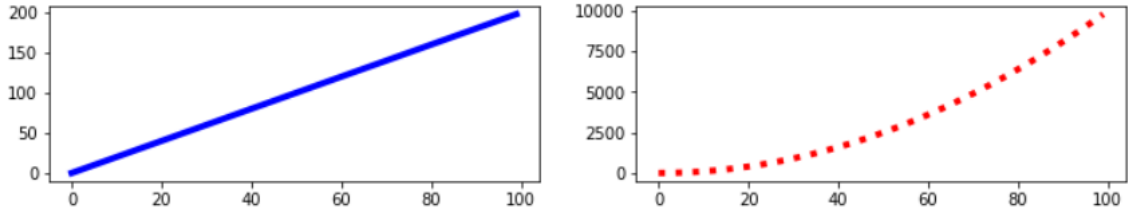
```
[<matplotlib.lines.Line2D at 0x7f24a91d71d0>]
```



I) See if you can resize the plot by adding the figsize() argument in plt.subplots() are copying and pasting your previous code.

```
fig, axes = plt.subplots(figsize=(12,2),nrows=1, ncols=2)
axes[0].plot(x,y,color='blue',lw=4,ls='-')
axes[1].plot(x,z,color='red',lw=4,ls=':')
```

```
[<matplotlib.lines.Line2D at 0x7f24a88f6590>]
```



CONCLUSION:

From this practical, I have successfully learned about matholib library in python.

SEABORN:

A) IMPORTING LIBRARIES AND DATA:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
sns.set_style('whitegrid')
```

```
titanic = sns.load_dataset('titanic')
```

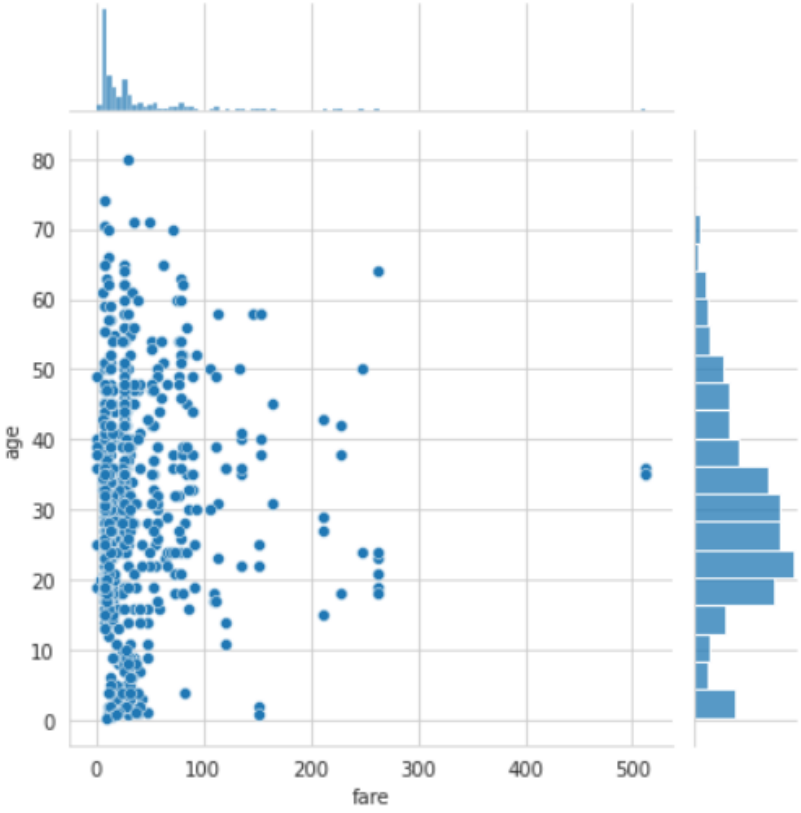
```
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

B) JOINPLOT [FARE V/S AGE]:

```
sns.jointplot(x='fare',y='age',data=titanic)
```

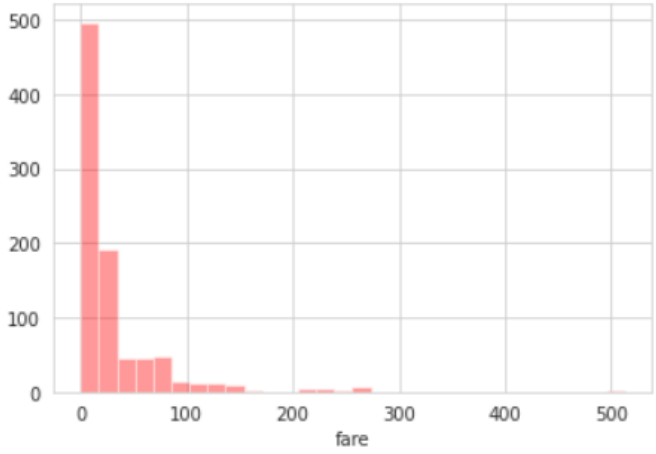
```
<seaborn.axisgrid.JointGrid at 0x7fd921864090>
```



C) **DISPLOT:**

```
sns.distplot(titanic['fare'],bins=30,kde=False,color='red')
```

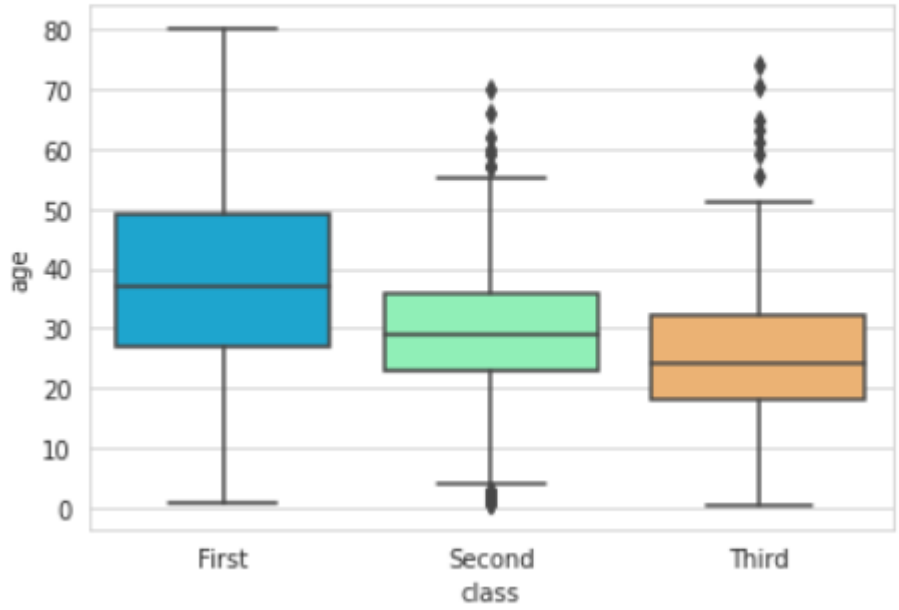
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.p
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fd91d018550>
```



D) **BOXPLOT:**

```
sns.boxplot(x='class',y='age',data=titanic,palette='rainbow' )
```

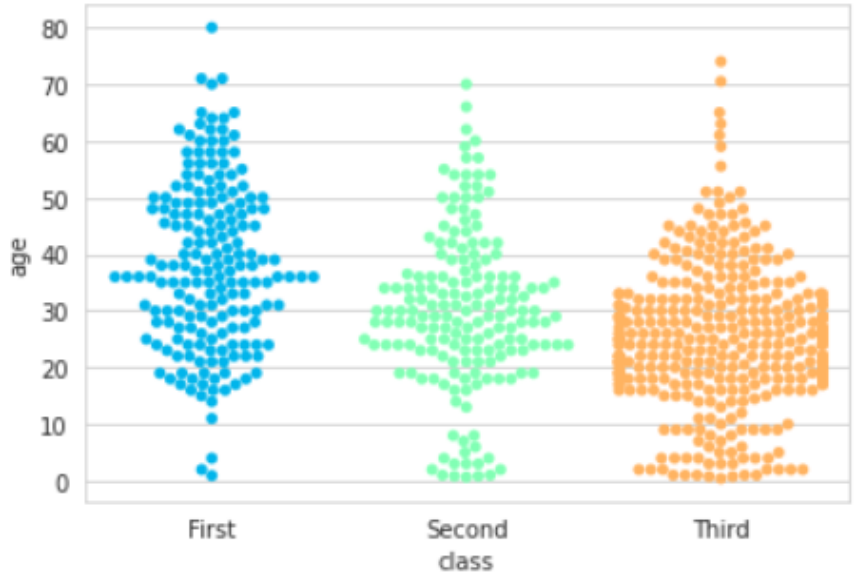
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd91cf76cd0>
```



E) SWARMPLOT [CLASS V/S AGE]:

```
sns.swarmplot(x='class',y='age',data=titanic,palette='rainbow')
```

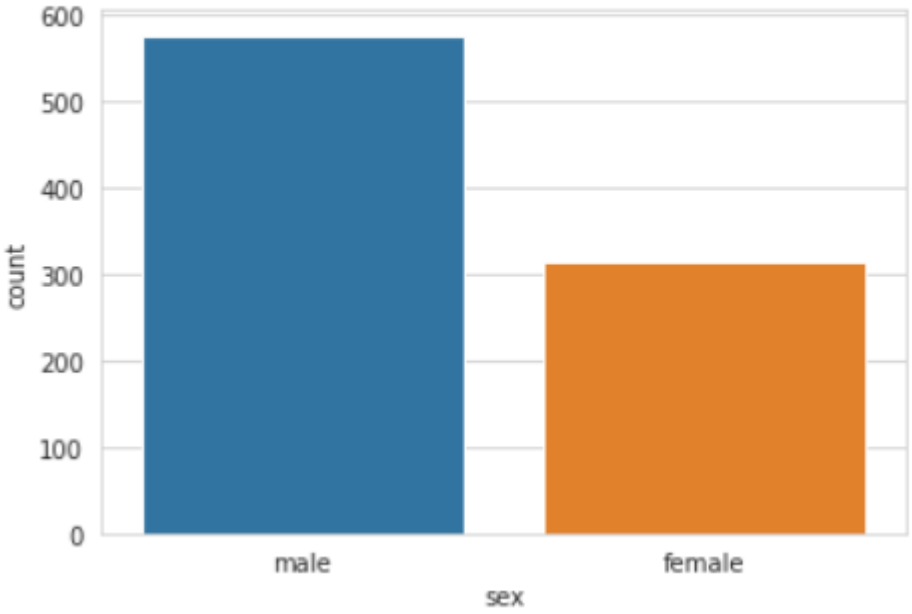
```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296:
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fd91ce78f90>
```



F) COUNTPLOT:

```
sns.countplot(x='sex',data=titanic)
```

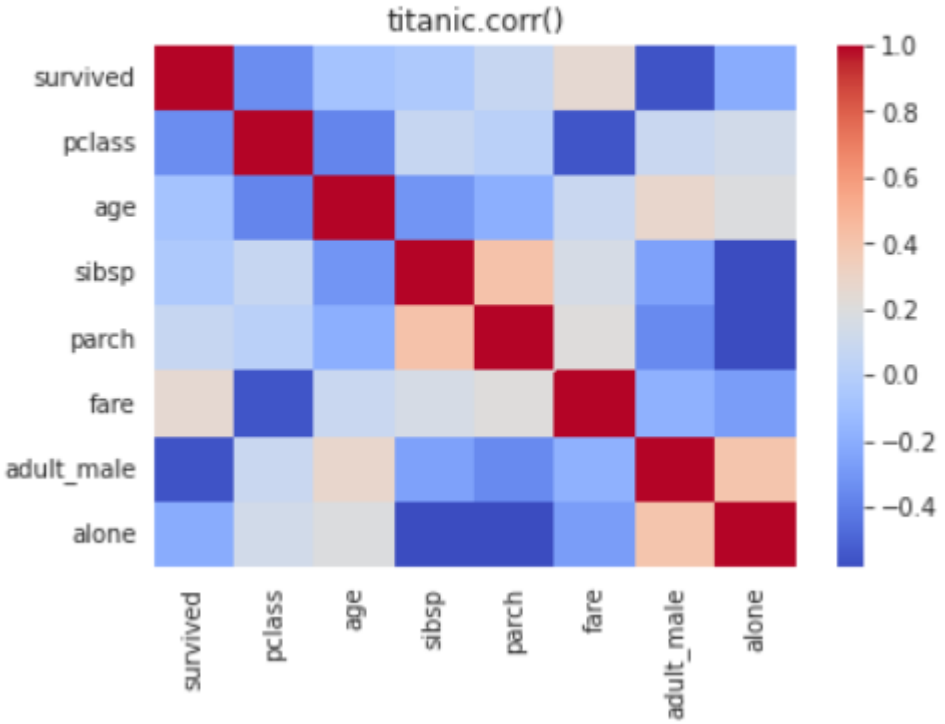
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd91ce78fd0>
```



G) HEATMAP:

```
sns.heatmap(titanic.corr(),cmap='coolwarm')
plt.title('titanic.corr()')
```

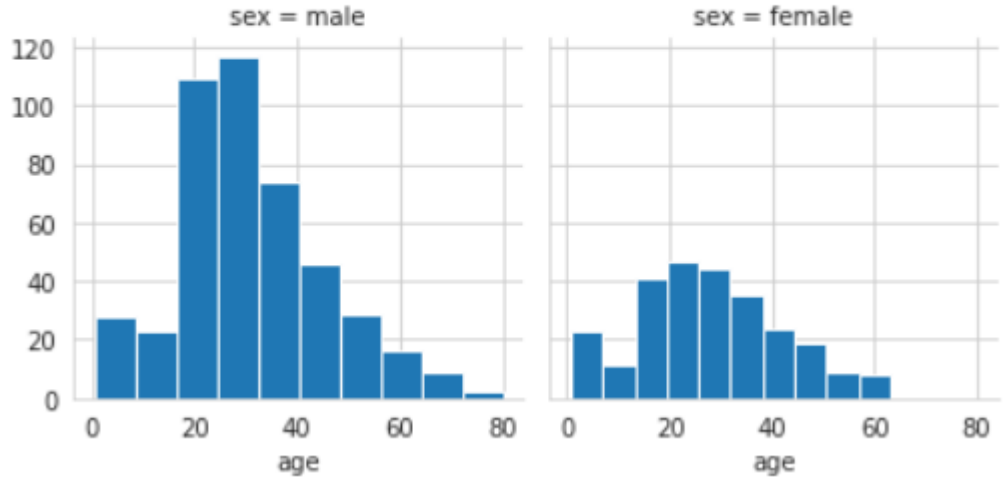
```
Text(0.5, 1.0, 'titanic.corr()')
```



H) FACEGRID:

```
g = sns.FacetGrid(data=titanic,col='sex')
g.map(plt.hist,'age')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fd91cd04450>
```



CONCLUSION:

From this practical, I have successfully learned about seaborn library in python.

AIM: Implementation of Logic programming using PROLOG-DFS for water jug problem.

THEORY:

This is the jug problem using simple depth-first search of a graph. The modified water-jug problem is as follows: Jug A holds 4 liters, and jug B holds 3 liters. There is a pump, which can be used to fill either Jug. How can you get exactly 2 liters of water into the 4-liter jug?

ASSUMPTIONS:

• We can fill a jug from the pump

• We can pour water out of the jug onto the ground

• We can pour water from one jug to another

• There are no other measuring devices available

• To solve the water jug problem, apart from problem statement we also need a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to state is made as described in corresponding right side and the resulting state is checked to see if it corresponds to a goal state. As long as it does not the cycle continues.

SOURCE CODE:

```
move(jugs(J1,J2), jugs(0,J2)) :- J1 > 0. % 3. empty large
move(jugs(J1,J2), jugs(J1,0)) :- J2 > 0. % 4. empty small
move(jugs(J1,J2), jugs(4,N)) :- % 5. fill large from small until large is full
J2 > 0,
J1 + J2 >= 4, % small must contain enough liquidto fill up large
N is J2 - (4 - J1).
move(jugs(J1,J2), jugs(N,3)) :- % 6. fill small from large until small is full
J1 > 0,
J1 + J2 >= 3, % large must contain enough liquid to fill up small
N is J1 - (3 - J2).
move(jugs(J1,J2), jugs(N,0)) :- % 7. empty small into large
J2 > 0,
J1 + J2 < 4, % all liquid in small must fit in large
N is J1 + J2.
move(jugs(J1,J2), jugs(0,N)) :- % 8. empty large into small
J1 > 0,
J1 + J2 < 3, % all liquid in large must fit in small
N is J1 + J2.
dfs_no_cycles_deep(State, Goal, Path) :-
dfs_deep_help([State], Goal, RevPath, 1), % start with maximum depth 1
reverse(RevPath, Path).
dfs_deep_help(PathSoFar, Goal, Path, DepthBound) :- % helper to try increasingly larger depths
dfs_bounded(PathSoFar, Goal, Path, DepthBound). % until a solution is found
dfs_deep_help(PathSoFar, Goal, Path, DepthBound) :-
NewDepth is DepthBound + 1, % add 1 to the maximum depth allowed
   NewDepth < 40, % uncomment to set an absolute limit on depth
dfs_deep_help(PathSoFar, Goal, Path, NewDepth). % try again with the increased maximum depth
dfs_bounded([Goal|PathSoFar], Goal, [Goal|PathSoFar], 1).
```

```
dfs_bounded([State|PathSoFar], Goal, Path, DepthBound) :-
DepthBound > 0, % check so maximum depth is not exceeded
move(State, NextState),not(member(NextState, [State|PathSoFar])), % check against cycles in our traversal
NewDepth is DepthBound - 1,
dfs_bounded([NextState,State|PathSoFar], Goal, Path, NewDepth).
% dfs_no_cycles_deep(jugs(0,0), jugs(2,0), Path).
% this works: 6 transitions from the initial state
% dfs_no_cycles_deep(jugs(0,0), jugs(3,0), Path).
% this works: 2 transitions from the initial state
% dfs_no_cycles_deep(jugs(0,0), jugs(2,2), Path).
% this has no solution
```

OUTPUT:

```
% c:/Users/NARENDER KESWANI/Documents/Prolog/water_jug_prob.pl compiled 0.00 sec, 13 clauses
?-
|    dfs_no_cycles_deep(jugs(0,0), jugs(2,0), Path).
Path = [jugs(0, 0), jugs(0, 3), jugs(3, 0), jugs(3, 3), jugs(4, 2), jugs(0, 2), jugs(2, 0)]
```

CONCLUSION:

From this practical, I have learned about Prolog and implantation of water jug problem.

AIM: IMPLEMENTATION OF LINEAR REGRESSION & LOGISTIC REGRESSION

A) LINEAR REGRESSION:

THEORY:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things:

1. does a set of predictor variables do a good job in predicting an outcome (dependent) variable?
2. Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable?

These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable

SOURCE CODE & OUTPUT:

1) IMPORTING LIBRARIES:

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

2) COLLECTING DATA:

```
url = "http://bit.ly/w-data"
data = pd.read_csv(url)
print("Data imported successfully")
data
```

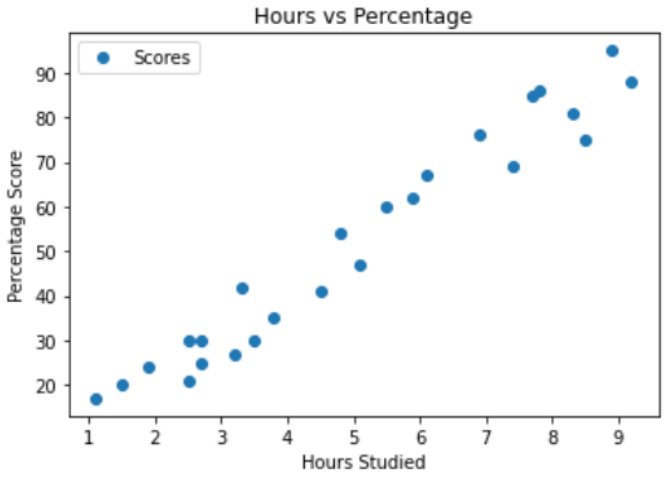
```
Data imported successfully
```

1 to 25 of 25 entries Filter

index	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17

3) Plotting the distribution of scores:

```
data.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



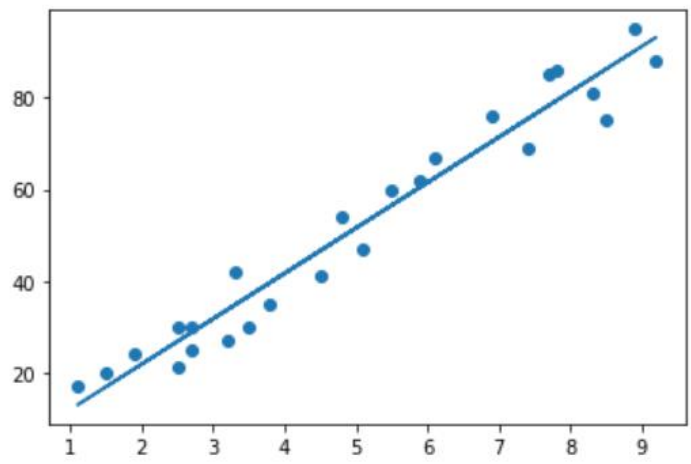
4) TRAINING DATA:

```
#Training Data
X = data.iloc[:, :-1].values
y = data.iloc[:, 1].values
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.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
print("Training complete.")
```

```
Training complete.
```

5) PLOTTING THE REGRESSION LINE:

```
line = regressor.coef_*X+regressor.intercept_
# Plotting for the test data
plt.scatter(X, y)
plt.plot(X, line);
plt.show()
```



6) PREDICTING THE SCORES:

```
y_pred = regressor.predict(X_test)
y_pred
```

```
array([16.88414476, 33.73226078, 75.357018  , 26.79480124, 60.49103328])
```

7) COMPARING ACTUAL VS PREDICTED:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Index	Actual	Predicted
0	20	16.884144762398023
1	27	33.732260779489835
2	69	75.35701799818725
3	30	26.79480124304026
4	62	60.491033277223885

Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

8) EVALUATING THE ALGORITHM:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred))
)
```

```
Mean Absolute Error: 4.183859899002982
Mean Squared Error: 21.598769307217456
Root Mean Squared Error: 4.647447612100373
```

B) LOGISTIC REGRESSION:

THEORY:

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1, with a sum of one.
Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression[1] (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

SOURCE CODE & OUTPUT:

1) COLLECTING DATA:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set= pd.read_csv('suv_data.csv')
data_set
```

1 to 25 of 400 entries Filter

index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

2) EXTRACTING INDEPENDENT AND DEPENDENT VARIABLE:

```
x = data_set.iloc[:, [2,3]].values
y = data_set.iloc[:, 4].values
```

3) SPLITTING THE DATASET INTO TRAINING 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_state=0)
```

4) FEATURE SCALING:

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

5) FITTING LOGISTIC REGRESSION TO THE TRAINING SET:

```
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

6) PREDICTING THE TEST SET RESULT:

```
y_pred= classifier.predict(x_test)
y_pred
```

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

7) CREATING THE CONFUSION MATRIX:

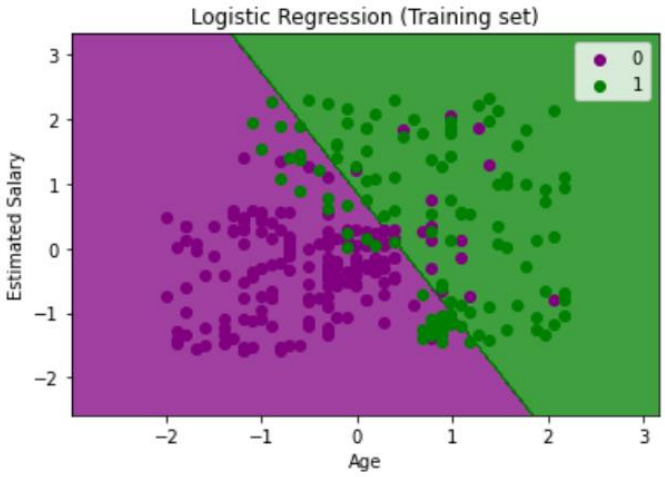
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[65,  3],
       [ 8, 24]])
```

8) VISUALIZING THE TRAINING SET RESULT:

```
from matplotlib.colors import ListedColormap
```

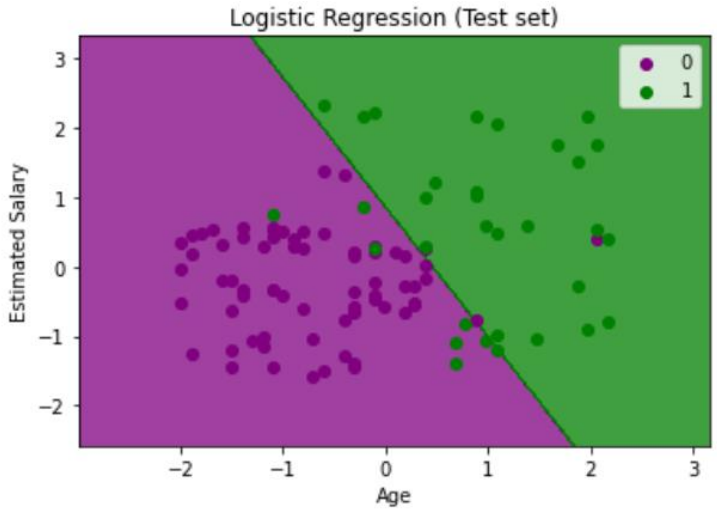
```
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
tep =0.01),
                nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01)
)

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shap
e),
         alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
          c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



9) <u>VISULAIZING THE TEST SET RESULT:</u>

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
 step =0.01),
                nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.0
1))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.
shape), alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
          c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
```

```
mtp.show()
```



CONCLUSION:

From this practical, I have learned about Implementation of Linear and logistic regression in python.

AIM: Implementation of ID3

THEORY:

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node. Most generally ID3 is only used for classification problems with nominal features only.

1) IMPORTING LIBRARIES:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder #if data is string thn use it to preprocess the data
from sklearn.tree import DecisionTreeClassifier
```

2) READING DATASET:

```
df=pd.read_csv('glass.csv')
df
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 10 columns

3) EXTRACTING FEATURES:

```
x=df.iloc[:,:-1]
```

```
y=df.iloc[:,9]
```

4) **MAKING MODEL:**

```
dt=DecisionTreeClassifier(criterion="entropy")
dt
```

5) **SPLITTING DATASET INTO TRAINING , TESTING & PREDICTING VALUES:**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = .3, random_state=0)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
y_pred
```

```
array([7, 1, 2, 6, 5, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 7, 3, 2, 3, 2, 5, 1,
       7, 7, 7, 1, 7, 1, 1, 2, 1, 3, 2, 7, 2, 1, 1, 3, 1, 7, 2, 6, 2, 1,
       2, 1, 1, 2, 1, 2, 1, 7, 7, 1, 2, 1, 1, 2, 1, 3, 1, 1, 1, 6, 1])
```

6) **CONFUSION MATRIX:**

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[17  3  1  0  0  0]
 [ 6 15  1  0  1  3]
 [ 2  2  3  0  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  2  0]
 [ 0  0  0  0  0  7]]
```

CONCLUSION:

From this practical, I have learned about implementation of ID3 algorithm in python.

AIM: Implementation of C4.5

THEORY:

The C4.5 algorithm is used in Data Mining as a Decision Tree Classifier which can be employed to generate a decision, based on a certain sample of data (univariate or multivariate predictors). C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. This accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

SOURCE CODE:

1) INSTALLING CHEFBOOST FOR C4.5:

```
!pip install chefboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting chefboost
  Downloading chefboost-0.0.17-py3-none-any.whl (26 kB)
Requirement already satisfied: pandas>=0.22.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (1.3.5)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (1.21.6)
Requirement already satisfied: psutil>=5.4.3 in /usr/local/lib/python3.7/dist-packages (from chefboost) (5.4.8)
Requirement already satisfied: tqdm>=4.30.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (4.64.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22.0->chefboost
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22.0->chefboost) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.22.0-
Installing collected packages: chefboost
Successfully installed chefboost-0.0.17
```

2) IMPORTING LIBRARIES AND READING DATA:

```
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/serengil/chefboost/master/tests/dataset/golf.txt')
data
```

	Outlook	Temp.	Humidity	Wind	Decision
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes

3) **BUILDING MODEL:**

```
from chefboost import Chefboost as chef
config = {'algorithm': 'C4.5'}
model = chef.fit(data, config = config, target_label = 'Decision')
```

```
[INFO]:  1 CPU cores will be allocated in parallel running
C4.5  tree is going to be built...
-------------------------
finished in  0.5118496417999268  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  100.0 % on  14  instances
Labels:  ['No' 'Yes']
Confusion matrix:  [[5, 0], [0, 9]]
Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
```

4) **PREDICTING VALUES:**

```
for i in range(data.shape[0]):
  prediction = chef.predict(model, param = data.iloc[i])
  print(prediction)
```

```
No
No
Yes
Yes
Yes
No
Yes
No
Yes
Yes
Yes
Yes
Yes
No
```

CONCLUSUION:

From this practical, I have learned the implementation of C4.5 in python.

AIM: Implementation of Naïve Bayes Classifier

THEORY:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

1) **IMPORTING LIBRARIES:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
```

2) **DATA PREPROCESSING:**

```
Dataframe = pd.read_csv('winequalityN.csv')
# getting info.
Dataframe.info()
Dataframe.describe()
# null value check
Dataframe.isnull().sum()
Dataframe = Dataframe.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
Dataframe.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   type                  6497 non-null   object
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

3) DATA SPLITTING INTO TRAINING DATASET & TESTING DATASET & CREATING MODEL:

```
x = Dataframe.drop(columns = ['quality','type'])
y = Dataframe['quality']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=1)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train,y_train)
```

```
GaussianNB()
```

4) PREDICTING QUALITY OF THE WINE:

```
model.score(x_test,y_test)
y_pred = model.predict(x_test)
np.set_printoptions(threshold=np.inf)
y_pred
```

```
array([6, 6, 6, 7, 7, 6, 7, 7, 7, 6, 6, 7, 6, 5, 6, 7, 6, 5, 5, 5, 7, 5,
       6, 5, 6, 7, 5, 7, 5, 5, 7, 6, 5, 6, 6, 7, 7, 5, 6, 8, 5, 7, 7, 7,
       6, 5, 6, 7, 6, 6, 5, 4, 6, 6, 7, 7, 5, 5, 5, 6, 5, 5, 7, 7, 7, 6,
       7, 7, 6, 7, 6, 7, 5, 7, 6, 5, 6, 6, 4, 3, 7, 5, 3, 5, 6, 5, 7, 6,
       6, 5, 6, 5, 6, 7, 7, 5, 7, 6, 7, 7, 5, 6, 6, 3, 6, 5, 5, 7, 6, 6,
       5, 5, 6, 6, 5, 5, 7, 6, 7, 5, 6, 4, 6, 7, 6, 6, 6, 5, 5, 7, 5, 5,
       5, 5, 6, 4, 5, 7, 6, 7, 6, 5, 7, 6, 5, 5, 5, 5, 7, 7, 6, 7, 7, 7,
       5, 6, 6, 7, 6, 7, 5, 7, 6, 5, 7, 7, 7, 7, 7, 5, 6, 7, 7, 5, 5, 6,
       7, 5, 4, 6, 6, 5, 7, 7, 5, 6, 6, 7, 5, 5, 6, 7, 5, 5, 7, 6, 5, 5,
       5, 5, 6, 5, 7, 6, 6, 5, 6, 5, 6, 6, 5, 7, 7, 7, 6, 5, 5, 5, 5, 6,
       7, 5, 7, 7, 5, 6, 5, 7, 5, 5, 5, 6, 6, 7, 6, 6, 7, 6, 7, 6, 7, 7,
       5, 6, 5, 5, 7, 7, 8, 7, 6, 6, 6, 5, 6, 6, 7, 6, 5, 5, 5, 6, 5, 5,
       5, 6, 5, 7, 5, 5, 6, 5, 7, 7, 7, 6, 5, 5, 5, 5, 6, 7, 6, 4, 5, 7,
       5, 6, 6, 7, 6, 7, 7, 4, 7, 7, 7, 3, 6, 6, 5, 7, 7, 7, 7, 4, 6, 7,
       7, 5, 6, 5, 6, 7, 6, 6, 7, 7, 6, 5, 5, 5, 7, 7, 7, 5, 5, 6, 5, 7,
       5, 5, 5, 6, 5, 5, 5, 7, 5, 6, 4, 5, 5, 6, 7, 6, 5, 7, 5, 7, 5, 3,
       6, 7, 6, 7, 6, 6, 6, 6, 5, 5, 6, 5, 6, 6, 5, 5, 5, 7, 5, 4, 6, 7,
```

5, 5, 6, 5, 5, 5, 6, 4, 6, 3, 6, 7, 5, 6, 6, 5, 5, 5, 5, 6, 5, 6,
5, 6, 6, 5, 7, 6, 6, 4, 6, 6, 7, 5, 6, 5, 6, 4, 5, 5, 6, 5, 5, 6,
5, 5, 6, 6, 5, 6, 7, 6, 5, 5, 7, 5, 5, 6, 6, 7, 5, 5, 7, 7, 7, 7,
5, 6, 7, 6, 6, 6, 7, 6, 5, 6, 7, 5, 5, 7, 6, 5, 6, 5, 7, 6, 7, 7,
7, 4, 7, 7, 5, 6, 7, 7, 6, 5, 6, 4, 5, 5, 5, 6, 6, 5, 7, 6, 6, 6,
6, 6, 5, 5, 7, 6, 5, 6, 6, 7, 6, 5, 5, 7, 5, 5, 5, 6, 5, 5, 6, 6,
5, 6, 5, 6, 5, 7, 7, 6, 4, 6, 6, 6, 7, 6, 5, 6, 5, 7, 5, 6, 5, 6,
6, 7, 7, 6, 7, 6, 6, 6, 5, 7, 5, 5, 6, 6, 6, 5, 5, 6, 7, 7, 5, 7,
6, 4, 5, 6, 6, 7, 6, 5, 5, 7, 7, 7, 7, 5, 5, 6, 6, 4, 6, 5, 4, 5,
6, 7, 4, 7, 7, 6, 6, 7, 5, 5, 5, 7, 7, 7, 6, 5, 4, 6, 5, 5, 5, 6,
6, 5, 7, 6, 5, 5, 7, 6, 7, 7, 6, 6, 6, 6, 5, 6, 6, 7, 7, 6, 3, 7,
5, 5, 5, 7, 6, 5, 3, 5, 7, 5, 7, 6, 6, 5, 7, 7, 5, 7, 6, 5, 6, 5,
6, 5, 5, 6, 7, 4, 7, 5, 6, 6, 7, 5, 7, 6, 6, 6, 5, 5, 6, 6, 5, 5,
5, 5, 7, 7, 3, 4, 7, 6, 5, 7, 5, 5, 5, 7, 6, 7, 7, 5, 5, 6, 5, 5,
6, 6, 7, 6, 7, 6, 6, 7, 7, 7, 7, 5, 5, 5, 5, 6, 7, 7, 6, 6, 6, 5,
7, 6, 6, 7, 5, 7, 5, 6, 7, 7, 6, 7, 6, 4, 5, 3, 4, 5, 7, 6, 6, 5,
5, 7, 5, 7, 7, 5, 6, 6, 6, 5, 7, 6, 5, 5, 7, 6, 5, 6, 7, 7, 6, 5,
5, 6, 6, 5, 5, 5, 5, 7, 5, 6, 6, 6, 6, 3, 6, 7, 6, 5, 6, 6, 7, 5,
5, 7, 7, 5, 6, 5, 5, 3, 4, 7, 5, 7, 7, 7, 6, 7, 7, 6, 4, 6, 6, 7,
5, 6, 6, 5, 7, 5, 5, 7, 6, 6, 5, 5, 7, 5, 7, 5, 7, 7, 6, 3, 6, 6,
7, 5, 6, 7, 5, 7, 5, 7, 6, 5, 4, 5, 5, 7, 5, 6, 6, 3, 4, 7, 6, 6,
5, 4, 6, 7, 5, 5, 6, 7, 7, 7, 6, 7, 7, 5, 5, 4, 7, 6, 6, 6, 7, 7,
7, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6, 6, 5, 7, 6, 6, 5, 7, 5, 7, 6, 7,
5, 7, 5, 7, 5, 5, 6, 7, 7, 6, 5, 6, 5, 6, 5, 4, 6, 7, 7, 5, 6, 5,
6, 5, 6, 6, 6, 6, 6, 7, 6, 6, 6, 4, 6, 6, 5, 5, 6, 4, 6, 6, 7, 5,
6, 6, 7, 5, 7, 6, 7, 7, 6, 7, 5, 7, 6, 5, 7, 7, 7, 7, 5, 6, 5, 7,
5, 6, 6, 7, 5, 7, 7, 7, 6, 5, 5, 5, 7, 5, 7, 6, 6, 7, 7, 6, 6, 5,
6, 5, 5, 6, 7, 7, 7, 9, 6, 5, 6, 6, 6, 5, 7, 5, 6, 6, 6, 5, 5, 6,
5, 7, 7, 4, 7, 6, 7, 6, 7, 5, 6, 6, 5, 6, 5, 7, 5, 7, 5, 5, 6, 7,
5, 6, 7, 6, 5, 6, 5, 6, 6, 5, 7, 5, 5, 6, 6, 6, 5, 7, 5, 6, 5, 6,
5, 7, 6, 6, 7, 6, 4, 4, 4, 6, 7, 6, 5, 6, 7, 6, 5, 7, 6, 7, 5, 5,
6, 5, 7, 5, 7, 7, 7, 7, 6, 6, 7, 7, 6, 5, 7, 6, 7, 7, 6, 6, 5, 5,
5, 6, 6, 7, 7, 5, 6, 7, 6, 6, 7, 7, 7, 6, 6, 5, 5, 7, 7, 5, 5, 7,
7, 5, 6, 5, 6, 5, 7, 5, 6, 6, 6, 6, 6, 5, 5, 4, 4, 6, 6, 6, 6, 7,
6, 6, 7, 7, 5, 7, 6, 6, 4, 4, 5, 5, 6, 6, 5, 8, 5, 4, 5, 6, 5, 8,
5, 4, 6, 3, 7, 6, 6, 6, 4, 5, 7, 5, 7, 7, 7, 6, 6, 6, 5, 6, 6, 6,
6, 6, 7, 7, 5, 7, 5, 5, 6, 4, 5, 7, 7, 6, 5, 6, 6, 6, 6, 7, 6, 6,
4, 7, 7, 5, 5, 7, 7, 5, 5, 5, 8, 6, 5, 7, 6, 5, 5, 6, 7, 6, 7, 6,
5, 4, 7, 5, 7, 5, 5, 6, 6, 5, 6, 7, 5, 5, 5, 5, 7, 6, 5, 5, 6, 6,
5, 5, 6, 5, 5, 5, 7, 7, 6, 5, 5, 6, 5, 5, 7, 5, 7, 6, 6, 6, 5, 6,
5, 7, 5, 5, 6, 5, 6, 7, 7, 7, 6, 7, 6, 6, 5, 7, 7, 6, 6, 5, 7, 5,
6, 5, 7, 7, 6, 5, 5, 6, 7, 6, 6, 6, 5, 5, 5, 5, 5, 7, 7, 7, 6, 5,
5, 6, 5, 5, 6, 5, 7, 7, 6, 5, 6, 7, 7, 6, 6, 6, 5, 5, 5, 6, 5, 6,
6, 6, 7, 6, 5, 6, 6, 3, 7, 6, 5, 6, 5, 6, 6, 6, 6, 7, 6, 7, 5, 7,
6, 5, 6, 5, 5, 6, 6, 5, 6, 6, 7, 7, 5, 5, 5, 5, 6, 5, 5, 7, 6, 5,
5, 5, 6, 5, 5, 6, 5, 4, 7, 5, 6, 5, 4, 6, 5, 7, 6, 5, 6, 5, 5, 6,
5, 7, 5, 6, 7, 5, 7, 7, 7, 4, 7, 5, 5, 5, 6, 7, 6, 7, 5, 5, 5, 7,
7, 6, 6, 6, 6, 6, 6, 5, 5, 6, 5, 7, 5, 6, 6, 6, 7, 7, 7, 5, 5, 7,
7, 7, 6, 6, 7, 6, 7, 5, 5, 6, 7, 7, 5, 6, 6, 6, 5, 6, 7, 5, 6, 6,
7, 5, 7, 7, 5, 5, 6, 5, 7, 6, 6, 5, 6, 6, 5, 5, 7, 6, 5, 7, 7, 5,
7, 7, 7, 5, 7, 7, 7, 6, 6, 5, 5, 7, 7, 7, 5, 5, 6, 6, 5, 7, 5, 7,

7, 4, 5, 6, 6, 6, 5, 5, 6, 7, 6, 5, 6, 6, 6, 5, 3, 6, 7, 6, 7, 9,
6, 7, 5, 5, 6, 5, 6, 7, 3, 7, 4, 7, 5, 5, 5, 5, 5, 6, 7, 5, 7, 6,
4, 7, 6, 6, 6, 7, 7, 5, 5, 5, 5, 5, 7, 5, 6, 6, 7, 5, 5, 5, 5, 5,
7, 5, 6, 7, 6, 5, 6, 5, 5, 6, 7, 7, 5, 7, 5, 6, 6, 6, 5, 7, 6, 5,
7, 7, 5, 5, 5, 7, 5, 5, 6, 6, 5, 7, 5, 7, 7, 6, 6, 7, 6, 5, 5, 5,
7, 5, 5, 6, 7, 7, 7, 7, 6, 5, 7, 5, 5, 5, 7, 5, 7, 7, 7, 6, 7, 7,
7, 7, 5, 5, 5, 6, 6, 7, 3, 6, 7, 5, 6, 6, 7, 7, 4, 5, 7, 6, 6, 5,
5, 7, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 5, 7, 7, 7, 5, 6,
7, 6, 4, 5, 6, 5, 6, 7, 7, 6, 5, 6, 6, 6, 5, 6, 5, 7, 6, 5, 5, 5,
7, 5, 6, 6, 6, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 7, 5, 5, 6, 6, 6,
7, 7, 5, 5, 6, 6, 7, 7, 6, 6, 6, 7, 6, 6, 7, 6, 7, 7, 6, 6, 7, 5,
5, 5, 5, 5, 7, 5, 5, 6, 5, 4, 7, 6, 5, 6, 6, 5, 5, 5, 5, 5, 5, 7,
6, 7, 7, 6, 7, 5, 6, 7, 7, 3, 7, 5, 6, 6, 7, 6, 6, 6, 6, 5, 4, 7,
7, 5, 7, 6, 6, 6, 6, 7, 5, 7, 5, 7, 6, 5, 5, 6, 6, 5, 5, 7, 6, 7,
5, 5, 5, 6, 6, 7, 5, 5, 6, 4, 6, 6, 7, 7, 5, 6, 4, 7, 7, 7, 6, 6,
7, 7, 7, 5, 6, 7, 7, 6, 7, 5, 5, 5, 6, 6, 7, 7, 6, 5, 7, 6, 6, 7,
7, 3, 7, 5, 7, 6, 7, 6, 6, 5, 6, 6, 7, 4, 6, 6, 9, 5, 6, 5, 5, 5,
6, 5, 7, 5, 4, 4, 6, 5, 5, 6, 7, 6, 7, 4, 5, 5, 5, 5, 5, 6, 6, 5,
7, 6, 7, 5, 6, 5, 5, 7, 6, 6, 6, 7, 6, 7, 5, 6, 6, 7, 6, 5, 6, 7,
6, 5, 7, 5, 7, 5, 6, 7, 5, 7, 7, 6, 7, 7, 5, 6, 6, 5, 5, 7, 7, 7,
7, 6, 5, 5, 7, 6, 5, 5, 7, 5, 5, 5, 6, 6])

5) CONFUSION MATRIX:

metrics.confusion_matrix(y_test,y_pred)

```
array([[  0,   2,   2,   1,   0,   0,   0],
       [  0,   4,  30,  16,  10,   0,   0],
       [  8,  26, 353, 207,  53,   0,   0],
       [  9,  26, 238, 319, 245,   0,   0],
       [  3,   6,  30, 105, 194,   3,   3],
       [  1,   0,   3,  12,  38,   2,   0],
       [  0,   0,   0,   0,   1,   0,   0]])
```

CONCLUSION:

From this practical, I have learned the implementation of naïve bayes classifier in python.

AIM: Implementation of ID3

THEORY:

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node. Most generally ID3 is only used for classification problems with nominal features only.

1) IMPORTING LIBRARIES:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder #if data is string thn use it to preprocess the data
from sklearn.tree import DecisionTreeClassifier
```

2) READING DATASET:

```
df=pd.read_csv('glass.csv')
df
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 10 columns

3) EXTRACTING FEATURES:

```
x=df.iloc[:,:-1]
```

```
y=df.iloc[:,9]
```

4) **MAKING MODEL:**

```
dt=DecisionTreeClassifier(criterion="entropy")
dt
```

5) **SPLITTING DATASET INTO TRAINING , TESTING & PREDICTING VALUES:**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = .3, random_state=0)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
y_pred
```

```
array([7, 1, 2, 6, 5, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 7, 3, 2, 3, 2, 5, 1,
       7, 7, 7, 1, 7, 1, 1, 2, 1, 3, 2, 7, 2, 1, 1, 3, 1, 7, 2, 6, 2, 1,
       2, 1, 1, 2, 1, 2, 1, 7, 7, 1, 2, 1, 1, 2, 1, 3, 1, 1, 1, 6, 1])
```

6) **CONFUSION MATRIX:**

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[17  3  1  0  0  0]
 [ 6 15  1  0  1  3]
 [ 2  2  3  0  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  2  0]
 [ 0  0  0  0  0  7]]
```

CONCLUSION:

From this practical, I have learned about implementation of ID3 algorithm in python.

AIM: Implementation of C4.5

THEORY:

The C4.5 algorithm is used in Data Mining as a Decision Tree Classifier which can be employed to generate a decision, based on a certain sample of data (univariate or multivariate predictors). C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. This accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

SOURCE CODE:

1) INSTALLING CHEFBOOST FOR C4.5:

```
!pip install chefboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting chefboost
  Downloading chefboost-0.0.17-py3-none-any.whl (26 kB)
Requirement already satisfied: pandas>=0.22.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (1.3.5)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (1.21.6)
Requirement already satisfied: psutil>=5.4.3 in /usr/local/lib/python3.7/dist-packages (from chefboost) (5.4.8)
Requirement already satisfied: tqdm>=4.30.0 in /usr/local/lib/python3.7/dist-packages (from chefboost) (4.64.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22.0->chefboost
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22.0->chefboost) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.22.0-
Installing collected packages: chefboost
Successfully installed chefboost-0.0.17
```

2) IMPORTING LIBRARIES AND READING DATA:

```
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/serengil/chefboost/master/tests/dataset/golf.txt')
data
```

	Outlook	Temp.	Humidity	Wind	Decision
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes

3) **BUILDING MODEL:**

```
from chefboost import Chefboost as chef
config = {'algorithm': 'C4.5'}
model = chef.fit(data, config = config, target_label = 'Decision')
```

```
[INFO]:  1 CPU cores will be allocated in parallel running
C4.5  tree is going to be built...
-------------------------
finished in  0.5118496417999268  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  100.0 % on  14  instances
Labels:  ['No' 'Yes']
Confusion matrix:  [[5, 0], [0, 9]]
Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
```

4) **PREDICTING VALUES:**

```
for i in range(data.shape[0]):
  prediction = chef.predict(model, param = data.iloc[i])
  print(prediction)
```

```
No
No
Yes
Yes
Yes
No
Yes
No
Yes
Yes
Yes
Yes
Yes
No
```

CONCLUSUION:

From this practical, I have learned the implementation of C4.5 in python.

AIM: Implementation of Naïve Bayes Classifier

THEORY:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

1) **IMPORTING LIBRARIES:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
```

2) **DATA PREPROCESSING:**

```
Dataframe = pd.read_csv('winequalityN.csv')
# getting info.
Dataframe.info()
Dataframe.describe()
# null value check
Dataframe.isnull().sum()
Dataframe = Dataframe.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
Dataframe.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   type                  6497 non-null   object
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

3) DATA SPLITTING INTO TRAINING DATASET & TESTING DATASET & CREATING MODEL:

```
x = Dataframe.drop(columns = ['quality','type'])
y = Dataframe['quality']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=1)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train,y_train)
```

```
GaussianNB()
```

4) PREDICTING QUALITY OF THE WINE:

```
model.score(x_test,y_test)
y_pred = model.predict(x_test)
np.set_printoptions(threshold=np.inf)
y_pred
```

```
array([6, 6, 6, 7, 7, 6, 7, 7, 7, 6, 6, 7, 6, 5, 6, 7, 6, 5, 5, 5, 7, 5,
       6, 5, 6, 7, 5, 7, 5, 5, 7, 6, 5, 6, 6, 7, 7, 5, 6, 8, 5, 7, 7, 7,
       6, 5, 6, 7, 6, 6, 5, 4, 6, 6, 7, 7, 5, 5, 5, 6, 5, 5, 7, 7, 7, 6,
       7, 7, 6, 7, 6, 7, 5, 7, 6, 5, 6, 6, 4, 3, 7, 5, 3, 5, 6, 5, 7, 6,
       6, 5, 6, 5, 6, 7, 7, 5, 7, 6, 7, 7, 5, 6, 6, 3, 6, 5, 5, 7, 6, 6,
       5, 5, 6, 6, 5, 5, 7, 6, 7, 5, 6, 4, 6, 7, 6, 6, 6, 5, 5, 7, 5, 5,
       5, 5, 6, 4, 5, 7, 6, 7, 6, 5, 7, 6, 5, 5, 5, 5, 7, 7, 6, 7, 7, 7,
       5, 6, 6, 7, 6, 7, 5, 7, 6, 5, 7, 7, 7, 7, 7, 5, 6, 7, 7, 5, 5, 6,
       7, 5, 4, 6, 6, 5, 7, 7, 5, 6, 6, 7, 5, 5, 6, 7, 5, 5, 7, 6, 5, 5,
       5, 5, 6, 5, 7, 6, 6, 5, 6, 5, 6, 6, 5, 7, 7, 7, 6, 5, 5, 5, 5, 6,
       7, 5, 7, 7, 5, 6, 5, 7, 5, 5, 5, 6, 6, 7, 6, 6, 7, 6, 7, 6, 7, 7,
       5, 6, 5, 5, 7, 7, 8, 7, 6, 6, 6, 5, 6, 6, 7, 6, 5, 5, 5, 6, 5, 5,
       5, 6, 5, 7, 5, 5, 6, 5, 7, 7, 7, 6, 5, 5, 5, 5, 6, 7, 6, 4, 5, 7,
       5, 6, 6, 7, 6, 7, 7, 4, 7, 7, 7, 3, 6, 6, 5, 7, 7, 7, 7, 4, 6, 7,
       7, 5, 6, 5, 6, 7, 6, 6, 7, 7, 6, 5, 5, 5, 7, 7, 7, 5, 5, 6, 5, 7,
       5, 5, 5, 6, 5, 5, 5, 7, 5, 6, 4, 5, 5, 6, 7, 6, 5, 7, 5, 7, 5, 3,
       6, 7, 6, 7, 6, 6, 6, 6, 5, 5, 6, 5, 6, 6, 5, 5, 5, 7, 5, 4, 6, 7,
```

5, 5, 6, 5, 5, 5, 6, 4, 6, 3, 6, 7, 5, 6, 6, 5, 5, 5, 5, 6, 5, 6,
5, 6, 6, 5, 7, 6, 6, 4, 6, 6, 7, 5, 6, 5, 6, 4, 5, 5, 6, 5, 5, 6,
5, 5, 6, 6, 5, 6, 7, 6, 5, 5, 7, 5, 5, 6, 6, 7, 5, 5, 7, 7, 7, 7,
5, 6, 7, 6, 6, 6, 7, 6, 5, 6, 7, 5, 5, 7, 6, 5, 6, 5, 7, 6, 7, 7,
7, 4, 7, 7, 5, 6, 7, 7, 6, 5, 6, 4, 5, 5, 5, 6, 6, 5, 7, 6, 6, 6,
6, 6, 5, 5, 7, 6, 5, 6, 6, 7, 6, 5, 5, 7, 5, 5, 5, 6, 5, 5, 6, 6,
5, 6, 5, 6, 5, 7, 7, 6, 4, 6, 6, 6, 7, 6, 5, 6, 5, 7, 5, 6, 5, 6,
6, 7, 7, 6, 7, 6, 6, 6, 5, 7, 5, 5, 6, 6, 6, 5, 5, 6, 7, 7, 5, 7,
6, 4, 5, 6, 6, 7, 6, 5, 5, 7, 7, 7, 7, 5, 5, 6, 6, 4, 6, 5, 4, 5,
6, 7, 4, 7, 7, 6, 6, 7, 5, 5, 5, 7, 7, 7, 6, 5, 4, 6, 5, 5, 5, 6,
6, 5, 7, 6, 5, 5, 7, 6, 7, 7, 6, 6, 6, 6, 5, 6, 6, 7, 7, 6, 3, 7,
5, 5, 5, 7, 6, 5, 3, 5, 7, 5, 7, 6, 6, 5, 7, 7, 5, 7, 6, 5, 6, 5,
6, 5, 5, 6, 7, 4, 7, 5, 6, 6, 7, 5, 7, 6, 6, 6, 5, 5, 6, 6, 5, 5,
5, 5, 7, 7, 3, 4, 7, 6, 5, 7, 5, 5, 5, 7, 6, 7, 7, 5, 5, 6, 5, 5,
6, 6, 7, 6, 7, 6, 6, 7, 7, 7, 7, 5, 5, 5, 5, 6, 7, 7, 6, 6, 6, 5,
7, 6, 6, 7, 5, 7, 5, 6, 7, 7, 6, 7, 6, 4, 5, 3, 4, 5, 7, 6, 6, 5,
5, 7, 5, 7, 7, 5, 6, 6, 6, 5, 7, 6, 5, 5, 7, 6, 5, 6, 7, 7, 6, 5,
5, 6, 6, 5, 5, 5, 5, 7, 5, 6, 6, 6, 6, 3, 6, 7, 6, 5, 6, 6, 7, 5,
5, 7, 7, 5, 6, 5, 5, 3, 4, 7, 5, 7, 7, 7, 6, 7, 7, 6, 4, 6, 6, 7,
5, 6, 6, 5, 7, 5, 5, 7, 6, 6, 5, 5, 7, 5, 7, 5, 7, 7, 6, 3, 6, 6,
7, 5, 6, 7, 5, 7, 5, 7, 6, 5, 4, 5, 5, 7, 5, 6, 6, 3, 4, 7, 6, 6,
5, 4, 6, 7, 5, 5, 6, 7, 7, 7, 6, 7, 7, 5, 5, 4, 7, 6, 6, 6, 7, 7,
7, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6, 6, 5, 7, 6, 6, 5, 7, 5, 7, 6, 7,
5, 7, 5, 7, 5, 5, 6, 7, 7, 6, 5, 6, 5, 6, 5, 4, 6, 7, 7, 5, 6, 5,
6, 5, 6, 6, 6, 6, 6, 7, 6, 6, 6, 4, 6, 6, 5, 5, 6, 4, 6, 6, 7, 5,
6, 6, 7, 5, 7, 6, 7, 7, 6, 7, 5, 7, 6, 5, 7, 7, 7, 7, 5, 6, 5, 7,
5, 6, 6, 7, 5, 7, 7, 7, 6, 5, 5, 5, 7, 5, 7, 6, 6, 7, 7, 6, 6, 5,
6, 5, 5, 6, 7, 7, 7, 9, 6, 5, 6, 6, 6, 5, 7, 5, 6, 6, 6, 5, 5, 6,
5, 7, 7, 4, 7, 6, 7, 6, 7, 5, 6, 6, 5, 6, 5, 7, 5, 7, 5, 5, 6, 7,
5, 6, 7, 6, 5, 6, 5, 6, 6, 5, 7, 5, 5, 6, 6, 6, 5, 7, 5, 6, 5, 6,
5, 7, 6, 6, 7, 6, 4, 4, 4, 6, 7, 6, 5, 6, 7, 6, 5, 7, 6, 7, 5, 5,
6, 5, 7, 5, 7, 7, 7, 7, 6, 6, 7, 7, 6, 5, 7, 6, 7, 7, 6, 6, 5, 5,
5, 6, 6, 7, 7, 5, 6, 7, 6, 6, 7, 7, 7, 6, 6, 5, 5, 7, 7, 5, 5, 7,
7, 5, 6, 5, 6, 5, 7, 5, 6, 6, 6, 6, 6, 5, 5, 4, 4, 6, 6, 6, 6, 7,
6, 6, 7, 7, 5, 7, 6, 6, 4, 4, 5, 5, 6, 6, 5, 8, 5, 4, 5, 6, 5, 8,
5, 4, 6, 3, 7, 6, 6, 6, 4, 5, 7, 5, 7, 7, 7, 6, 6, 6, 5, 6, 6, 6,
6, 6, 7, 7, 5, 7, 5, 5, 6, 4, 5, 7, 7, 6, 5, 6, 6, 6, 6, 7, 6, 6,
4, 7, 7, 5, 5, 7, 7, 5, 5, 5, 8, 6, 5, 7, 6, 5, 5, 6, 7, 6, 7, 6,
5, 4, 7, 5, 7, 5, 5, 6, 6, 5, 6, 7, 5, 5, 5, 5, 7, 6, 5, 5, 6, 6,
5, 5, 6, 5, 5, 5, 7, 7, 6, 5, 5, 6, 5, 5, 7, 5, 7, 6, 6, 6, 5, 6,
5, 7, 5, 5, 6, 5, 6, 7, 7, 7, 6, 7, 6, 6, 5, 7, 7, 6, 6, 5, 7, 5,
6, 5, 7, 7, 6, 5, 5, 6, 7, 6, 6, 6, 5, 5, 5, 5, 5, 7, 7, 7, 6, 5,
5, 6, 5, 5, 6, 5, 7, 7, 6, 5, 6, 7, 7, 6, 6, 6, 5, 5, 5, 6, 5, 6,
6, 6, 7, 6, 5, 6, 6, 3, 7, 6, 5, 6, 5, 6, 6, 6, 6, 7, 6, 7, 5, 7,
6, 5, 6, 5, 5, 6, 6, 5, 6, 6, 7, 7, 5, 5, 5, 5, 6, 5, 5, 7, 6, 5,
5, 5, 6, 5, 5, 6, 5, 4, 7, 5, 6, 5, 4, 6, 5, 7, 6, 5, 6, 5, 5, 6,
5, 7, 5, 6, 7, 5, 7, 7, 7, 4, 7, 5, 5, 5, 6, 7, 6, 7, 5, 5, 5, 7,
7, 6, 6, 6, 6, 6, 6, 5, 5, 6, 5, 7, 5, 6, 6, 6, 7, 7, 7, 5, 5, 7,
7, 7, 6, 6, 7, 6, 7, 5, 5, 6, 7, 7, 5, 6, 6, 6, 5, 6, 7, 5, 6, 6,
7, 5, 7, 7, 5, 5, 6, 5, 7, 6, 6, 5, 6, 6, 5, 5, 7, 6, 5, 7, 7, 5,
7, 7, 7, 5, 7, 7, 7, 6, 6, 5, 5, 7, 7, 7, 5, 5, 6, 6, 5, 7, 5, 7,

7, 4, 5, 6, 6, 6, 5, 5, 6, 7, 6, 5, 6, 6, 6, 5, 3, 6, 7, 6, 7, 9,
6, 7, 5, 5, 6, 5, 6, 7, 3, 7, 4, 7, 5, 5, 5, 5, 5, 6, 7, 5, 7, 6,
4, 7, 6, 6, 6, 7, 7, 5, 5, 5, 5, 5, 7, 5, 6, 6, 7, 5, 5, 5, 5, 5,
7, 5, 6, 7, 6, 5, 6, 5, 5, 6, 7, 7, 5, 7, 5, 6, 6, 6, 5, 7, 6, 5,
7, 7, 5, 5, 5, 7, 5, 5, 6, 6, 5, 7, 5, 7, 7, 6, 6, 7, 6, 5, 5, 5,
7, 5, 5, 6, 7, 7, 7, 7, 6, 5, 7, 5, 5, 5, 7, 5, 7, 7, 7, 6, 7, 7,
7, 7, 5, 5, 5, 6, 6, 7, 3, 6, 7, 5, 6, 6, 7, 7, 4, 5, 7, 6, 6, 5,
5, 7, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 5, 7, 7, 7, 5, 6,
7, 6, 4, 5, 6, 5, 6, 7, 7, 6, 5, 6, 6, 6, 5, 6, 5, 7, 6, 5, 5, 5,
7, 5, 6, 6, 6, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 7, 5, 5, 6, 6, 6,
7, 7, 5, 5, 6, 6, 7, 7, 6, 6, 6, 7, 6, 6, 7, 6, 7, 7, 6, 6, 7, 5,
5, 5, 5, 5, 7, 5, 5, 6, 5, 4, 7, 6, 5, 6, 6, 5, 5, 5, 5, 5, 5, 7,
6, 7, 7, 6, 7, 5, 6, 7, 7, 3, 7, 5, 6, 6, 7, 6, 6, 6, 6, 5, 4, 7,
7, 5, 7, 6, 6, 6, 6, 7, 5, 7, 5, 7, 6, 5, 5, 6, 6, 5, 5, 7, 6, 7,
5, 5, 5, 6, 6, 7, 5, 5, 6, 4, 6, 6, 7, 7, 5, 6, 4, 7, 7, 7, 6, 6,
7, 7, 7, 5, 6, 7, 7, 6, 7, 5, 5, 5, 6, 6, 7, 7, 6, 5, 7, 6, 6, 7,
7, 3, 7, 5, 7, 6, 7, 6, 6, 5, 6, 6, 7, 4, 6, 6, 9, 5, 6, 5, 5, 5,
6, 5, 7, 5, 4, 4, 6, 5, 5, 6, 7, 6, 7, 4, 5, 5, 5, 5, 5, 6, 6, 5,
7, 6, 7, 5, 6, 5, 5, 7, 6, 6, 6, 7, 6, 7, 5, 6, 6, 7, 6, 5, 6, 7,
6, 5, 7, 5, 7, 5, 6, 7, 5, 7, 7, 6, 7, 7, 5, 6, 6, 5, 5, 7, 7, 7,
7, 6, 5, 5, 7, 6, 5, 5, 7, 5, 5, 5, 6, 6])

5) CONFUSION MATRIX:

metrics.confusion_matrix(y_test,y_pred)

```
array([[  0,   2,   2,   1,   0,   0,   0],
       [  0,   4,  30,  16,  10,   0,   0],
       [  8,  26, 353, 207,  53,   0,   0],
       [  9,  26, 238, 319, 245,   0,   0],
       [  3,   6,  30, 105, 194,   3,   3],
       [  1,   0,   3,  12,  38,   2,   0],
       [  0,   0,   0,   0,   1,   0,   0]])
```

CONCLUSION:

From this practical, I have learned the implementation of naïve bayes classifier in python.

AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

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.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

1) **IMPORTING LIBRARIES:**

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

2) **DATA PREPROCESSING:**

```
data = pd.read_csv('Countrries.csv')
data = data.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
data
```

	country	latitude	longitude	name
0	AD	42.546245	1.601554	Andorra
1	AE	23.424076	53.847818	United Arab Emirates
2	AF	33.939110	67.709953	Afghanistan
3	AG	17.060816	-61.796428	Antigua and Barbuda
4	AI	18.220554	-63.068615	Anguilla
...	...	...	...	...
240	YE	15.552727	48.516388	Yemen
241	YT	-12.827500	45.166244	Mayotte
242	ZA	-30.559482	22.937506	South Africa
243	ZM	-13.133897	27.849332	Zambia
244	ZW	-19.015438	29.154857	Zimbabwe

245 rows × 4 columns

3) PLOTTING GRAPH:

```
plt.scatter(data['longitude'],data['latitude'])
plt.xlim(-180,180)
plt.ylim(-90,90)
plt.show()
```

4) DATA SPLITTING INTO TRAINING DATASET & TESTING DATASET & CREATING CLUSTERS:

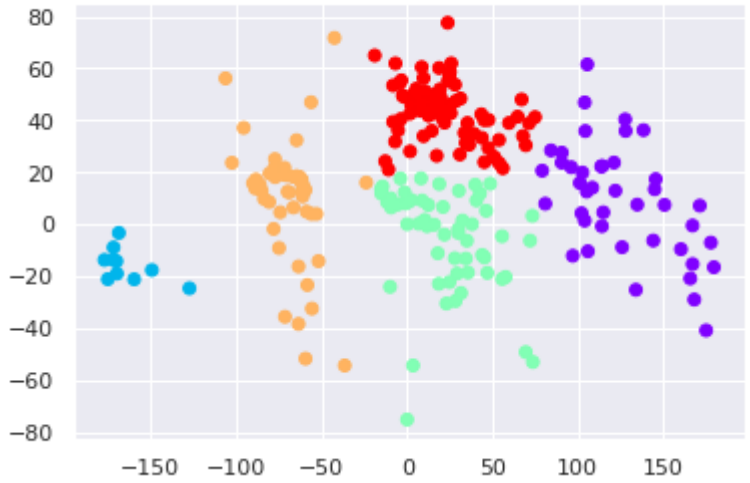
```
x = data.iloc[:,1:3]
kmeans = KMeans(5)
kmeans.fit(x)
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
array([2, 2, 2, 1, 1, 2, 2, 1, 4, 4, 1, 3, 2, 0, 1, 2, 2, 1, 0, 2, 4, 2,
       2, 4, 4, 1, 0, 1, 1, 1, 0, 4, 4, 2, 1, 1, 0, 4, 4, 4, 2, 4, 3, 1,
       4, 0, 1, 1, 1, 1, 0, 2, 2, 2, 4, 2, 1, 1, 2, 1, 2, 2, 2, 4, 2, 4,
       2, 0, 1, 0, 2, 2, 4, 2, 1, 2, 1, 2, 4, 2, 1, 4, 4, 1, 4, 2, 1, 1,
       0, 4, 1, 2, 0, 4, 1, 2, 1, 2, 0, 2, 2, 2, 0, 4, 2, 2, 2, 2, 2, 1,
       2, 0, 4, 2, 0, 3, 4, 1, 0, 0, 2, 1, 2, 0, 2, 1, 2, 0, 4, 4, 2, 2,
       2, 2, 2, 2, 2, 2, 4, 0, 2, 4, 0, 0, 0, 0, 1, 2, 1, 2, 4, 0, 4, 1,
       0, 4, 4, 0, 4, 0, 4, 1, 2, 2, 0, 0, 3, 0, 2, 1, 1, 3, 0, 0, 2, 2,
       1, 3, 1, 2, 2, 0, 1, 2, 4, 2, 2, 0, 4, 2, 0, 4, 4, 2, 0, 4, 2, 2,
       2, 4, 2, 4, 4, 1, 4, 1, 2, 4, 1, 4, 4, 4, 0, 2, 3, 0, 2, 2, 3, 2,
       1, 0, 0, 4, 2, 4, 4, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 3, 3, 2, 4, 4,
       4, 4, 4], dtype=int32)
```

5) PLOTTING CLUSTERS:

```
data_with_clusters = data.copy()
data_with_clusters['clusters'] = identified_clusters
plt.scatter(data_with_clusters['longitude'],data_with_clusters['latitude'],c=data_with_cluste
rs['clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7fa01d1aa110>
```



CONCLUSION:

From this practical, I have learned the implementation of k-means in python.

AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

K-medoids clustering is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points.

The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center. It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as 'exemplars' or 'medoids'). K-Medoids also differ in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres

SOURCE CODE:

1) **IMPORTING LIBRARIES & READING DATASET:**

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

    df = pd.read_csv('breast-cancer-wisconsin.data')
    df.head()
```

	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4

2) DATA PREPROCESSING:

```
dummy = pd.DataFrame(df.columns).T
for col in dummy.columns:
    dummy[col]=dummy[col].astype(float)
    dummy[col]=dummy[col].astype(int)

df.columns=range(0,df.shape[1])
df[6] = df[6].str.extract('(\d+)').astype(float)
for col in df.columns:
    df[col]=df[col].astype(float)
df = pd.concat([dummy, df],axis=0)
df.reset_index(inplace=True, drop=True)

column_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromati',
                'Normal Nucleoli', 'Mitoses', 'Class']

df.columns=column_names
df['Class'] = df['Class'].replace(to_replace={2:1, 4:0})
df.tail()
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromati	Normal Nucleoli	Mitoses	Class
694	776715.0	3.0	1.0	1.0	1.0	3.0	2.0	1.0	1.0	1.0	1.0
695	841769.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0	1.0
696	888820.0	5.0	10.0	10.0	3.0	7.0	3.0	8.0	10.0	2.0	0.0
697	897471.0	4.0	8.0	6.0	4.0	3.0	4.0	10.0	6.0	1.0	0.0
698	897471.0	4.0	8.0	8.0	5.0	4.0	5.0	10.0	4.0	1.0	0.0

3) COUNT OF NULL VALUES:

```
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                  16
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

4) DROPPING NULL VALUES:

```
df = df.dropna(axis=0)
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                   0
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

5) NORMALIZATION:

```
[8] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler

    X = df.iloc[:,:-1].values
    y = df.iloc[:,-1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    scaler = MinMaxScaler()

    X_train = scaler.fit_transform(X_train)
    X_test  = scaler.transform(X_test)
```

6) INSTALLING & IMPORTING LIBRARIES FOR KMEDIOD:

```
!pip install scikit-learn-extra
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.2.0-cp37-cp37m-manylinux2010_x86_64.whl (1.7 MB)
     |████████████████████████████████| 1.7 MB 3.8 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.21.6)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.0.2)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.1.0)
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.2.0
```

7) BUILDING MODEL OF K-MEDIOD & PREDICTION OF BREAST CANCER:

```
0s
from sklearn_extra.cluster import KMedoids

kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X_train)
y_pred = kmedoids.predict(X_test)
y_pred
```

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

8) CONFUSION MATRIX:

```
0s
[20] from sklearn.metrics import classification_report,confusion_matrix
     print(confusion_matrix(y_test, y_pred))
```

```
[[43  7]
 [ 1 86]]
```

CONCLUSION:

From this practical, I have learned the implementation of k-medoid in python.

AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

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.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

1) **IMPORTING LIBRARIES:**

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

2) **DATA PREPROCESSING:**

```
data = pd.read_csv('Countrries.csv')
data = data.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
data
```

	country	latitude	longitude	name
0	AD	42.546245	1.601554	Andorra
1	AE	23.424076	53.847818	United Arab Emirates
2	AF	33.939110	67.709953	Afghanistan
3	AG	17.060816	-61.796428	Antigua and Barbuda
4	AI	18.220554	-63.068615	Anguilla
...	...	...	...	...
240	YE	15.552727	48.516388	Yemen
241	YT	-12.827500	45.166244	Mayotte
242	ZA	-30.559482	22.937506	South Africa
243	ZM	-13.133897	27.849332	Zambia
244	ZW	-19.015438	29.154857	Zimbabwe

245 rows × 4 columns

3) PLOTTING GRAPH:

```
plt.scatter(data['longitude'],data['latitude'])
plt.xlim(-180,180)
plt.ylim(-90,90)
plt.show()
```

4) DATA SPLITTING INTO TRAINING DATASET & TESTING DATASET & CREATING CLUSTERS:

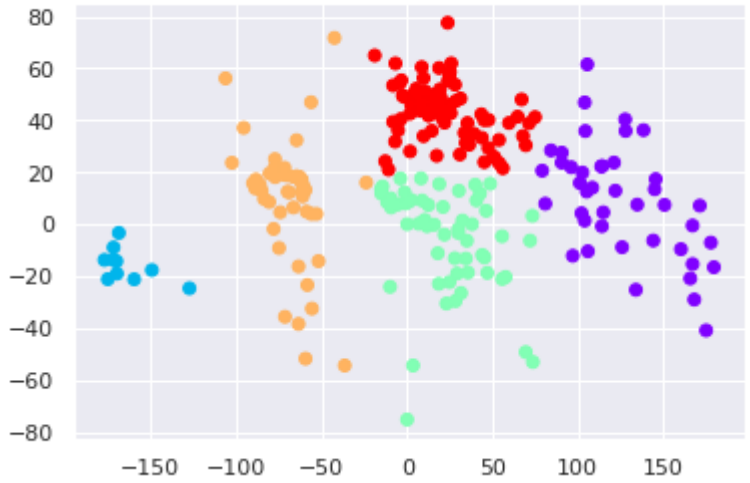
```
x = data.iloc[:,1:3]
kmeans = KMeans(5)
kmeans.fit(x)
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
array([2, 2, 2, 1, 1, 2, 2, 1, 4, 4, 1, 3, 2, 0, 1, 2, 2, 1, 0, 2, 4, 2,
       2, 4, 4, 1, 0, 1, 1, 1, 0, 4, 4, 2, 1, 1, 0, 4, 4, 4, 2, 4, 3, 1,
       4, 0, 1, 1, 1, 1, 0, 2, 2, 2, 4, 2, 1, 1, 2, 1, 2, 2, 2, 4, 2, 4,
       2, 0, 1, 0, 2, 2, 4, 2, 1, 2, 1, 2, 4, 2, 1, 4, 4, 1, 4, 2, 1, 1,
       0, 4, 1, 2, 0, 4, 1, 2, 1, 2, 0, 2, 2, 2, 0, 4, 2, 2, 2, 2, 2, 1,
       2, 0, 4, 2, 0, 3, 4, 1, 0, 0, 2, 1, 2, 0, 2, 1, 2, 0, 4, 4, 2, 2,
       2, 2, 2, 2, 2, 2, 4, 0, 2, 4, 0, 0, 0, 0, 1, 2, 1, 2, 4, 0, 4, 1,
       0, 4, 4, 0, 4, 0, 4, 1, 2, 2, 0, 0, 3, 0, 2, 1, 1, 3, 0, 0, 2, 2,
       1, 3, 1, 2, 2, 0, 1, 2, 4, 2, 2, 0, 4, 2, 0, 4, 4, 2, 0, 4, 2, 2,
       2, 4, 2, 4, 4, 1, 4, 1, 2, 4, 1, 4, 4, 4, 0, 2, 3, 0, 2, 2, 3, 2,
       1, 0, 0, 4, 2, 4, 4, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 3, 3, 2, 4, 4,
       4, 4, 4], dtype=int32)
```

5) PLOTTING CLUSTERS:

```
data_with_clusters = data.copy()
data_with_clusters['clusters'] = identified_clusters
plt.scatter(data_with_clusters['longitude'],data_with_clusters['latitude'],c=data_with_cluste
rs['clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7fa01d1aa110>
```



CONCLUSION:

From this practical, I have learned the implementation of k-means in python.

AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

K-medoids clustering is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points.

The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center. It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as 'exemplars' or 'medoids'). K-Medoids also differ in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres

SOURCE CODE:

1) **IMPORTING LIBRARIES & READING DATASET:**

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

    df = pd.read_csv('breast-cancer-wisconsin.data')
    df.head()
```

	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4

2) DATA PREPROCESSING:

```
dummy = pd.DataFrame(df.columns).T
for col in dummy.columns:
    dummy[col]=dummy[col].astype(float)
    dummy[col]=dummy[col].astype(int)

df.columns=range(0,df.shape[1])
df[6] = df[6].str.extract('(\d+)').astype(float)
for col in df.columns:
    df[col]=df[col].astype(float)
df = pd.concat([dummy, df],axis=0)
df.reset_index(inplace=True, drop=True)

column_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromati',
                'Normal Nucleoli', 'Mitoses', 'Class']

df.columns=column_names
df['Class'] = df['Class'].replace(to_replace={2:1, 4:0})
df.tail()
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromati	Normal Nucleoli	Mitoses	Class
694	776715.0	3.0	1.0	1.0	1.0	3.0	2.0	1.0	1.0	1.0	1.0
695	841769.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0	1.0
696	888820.0	5.0	10.0	10.0	3.0	7.0	3.0	8.0	10.0	2.0	0.0
697	897471.0	4.0	8.0	6.0	4.0	3.0	4.0	10.0	6.0	1.0	0.0
698	897471.0	4.0	8.0	8.0	5.0	4.0	5.0	10.0	4.0	1.0	0.0

3) COUNT OF NULL VALUES:

```
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                  16
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

4) DROPPING NULL VALUES:

```
df = df.dropna(axis=0)
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                   0
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

5) NORMALIZATION:

```
[8] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler

    X = df.iloc[:,:-1].values
    y = df.iloc[:,-1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    scaler = MinMaxScaler()

    X_train = scaler.fit_transform(X_train)
    X_test  = scaler.transform(X_test)
```

6) INSTALLING & IMPORTING LIBRARIES FOR KMEDIOD:

```
!pip install scikit-learn-extra
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.2.0-cp37-cp37m-manylinux2010_x86_64.whl (1.7 MB)
     |████████████████████████████████| 1.7 MB 3.8 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.21.6)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.0.2)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.1.0)
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.2.0
```

7) BUILDING MODEL OF K-MEDIOD & PREDICTION OF BREAST CANCER:

```
0s
from sklearn_extra.cluster import KMedoids

kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X_train)
y_pred = kmedoids.predict(X_test)
y_pred
```

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

8) CONFUSION MATRIX:

```
0s
[20] from sklearn.metrics import classification_report,confusion_matrix
     print(confusion_matrix(y_test, y_pred))
```

```
[[43  7]
 [ 1 86]]
```

CONCLUSION:

From this practical, I have learned the implementation of k-medoid in python.

AIM: IMPLEMENTATION OF CLASSIFYING DATA USING SUPPORT VECTOR MACHINES (SVMS).

THEORY:

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well . Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyperplane/ line) In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in nonlinear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined. In Python, scikit-learn is a widely used library for implementing machine learning algorithms. SVM is also available in the scikit-learn library and we follow the same structure for using it(Import library, object creation, fitting model and prediction)

Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

SOURCE CODE:

1) **IMPORTING LIBRARIES:**

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

2) **READING DATASET:**

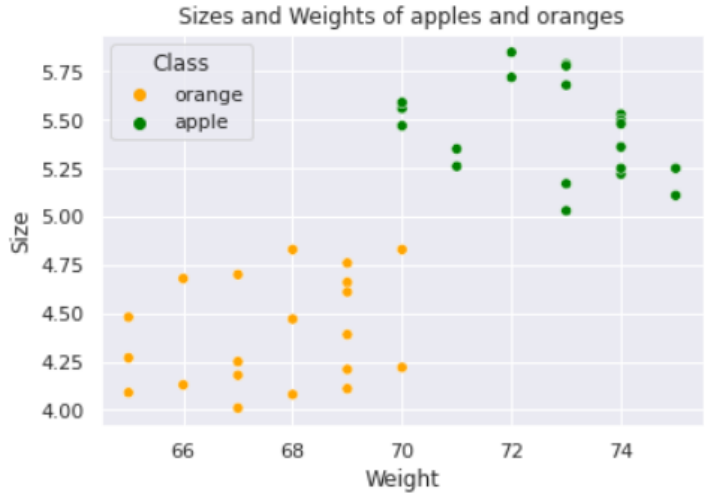
```
data=pd.read_csv('apples_and_oranges.csv')
data.head(5)
```

	Weight	Size	Class
0	69	4.39	orange
1	69	4.21	orange
2	65	4.09	orange
3	72	5.85	apple
4	67	4.70	orange

3) PLOTTING GRAPH:

```
# create a dictionary to colour classes
color_dict = dict({'orange':'orange',
                   'apple':'green'})
# scatterplot
plt.title('Sizes and Weights of apples and oranges')
sns.scatterplot(data=data, x="Weight", y="Size", hue="Class", palette = color_dict)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa1132502d0>
```



4) CREATING SVM LINEAR MODEL:

```
# define input data
X = data[["Weight", "Size"]]
# define target
y = data.Class
# fitting the support vector machine using a linear kernel
from sklearn import svm
clf = svm.SVC(kernel = 'linear', C=10)
clf.fit(X, y)
```

```
SVC(C=10, kernel='linear')
```

5) OBTAINING HYPERPLANE:

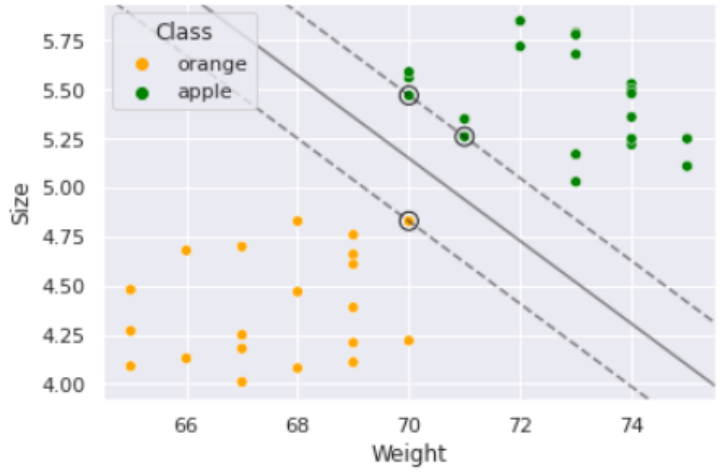
```
#Obtain hyperplane
b = clf.intercept_
w_1 = clf.coef_[0][0]
w_2 = clf.coef_[0][1]
b, w_1, w_2
```

```
(array([62.09725159]), -0.6575905440881797, -3.1196742877366077)
```

6) PLOTTING HYPERPLANE AND SVM:

```
# plotting the hyperplane and support vector lines
ax = plt.gca()
sns.scatterplot(data=data, x="Weight", y="Size", hue="Class", palette = color_dict)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
  "X does not have valid feature names, but"
```



7) OBTAINING SUPPORT VECTORS:

```
# obtain support vectors
clf.support_vectors_
```

```
array([[70.  ,  5.47],
       [71.  ,  5.26],
       [70.  ,  4.83]])
```

8) PREDICTING VALUES:

```
clf.predict([[70, 4.6]])
```

```
/usr/local/lib/python3.7/dist-pac
  "X does not have valid feature
array(['orange'], dtype=object)
```

```
clf.predict([[74, 5.3]])
```

```
/usr/local/lib/python3.7/dist-pac
  "X does not have valid feature
array(['apple'], dtype=object)
```

```
clf.predict([[62,3.0]])
```

```
/usr/local/lib/python3.7/dist-pac
  "X does not have valid feature
array(['orange'], dtype=object)
```

CONCLUSION:

From this practical, I have learned and implemented Linear Support Vector Machine(SVM) in python.

AIM: IMPLEMENTATION OF CLASSIFYING DATA USING NON-LINEAR KERNELS SUPPORT VECTOR MACHINES (SVMS).

THEORY:

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well . Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyperplane/ line) In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in nonlinear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined. In Python, scikit-learn is a widely used library for implementing machine learning algorithms. SVM is also available in the scikit-learn library and we follow the same structure for using it(Import library, object creation, fitting model and prediction)

Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

SOURCE CODE:

1) **IMPORTING LIBRARIES & READING DATASET:**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
dataset.head(8)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1

2) SPLITTING DATASET INTO TRAIN & TEST DATA:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

3) NORMALIZATION:

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

4) BUILDING MODEL [Radial Basis Function(rbf)]:

```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf',random_state=0)
classifier.fit(X_train, y_train)
```

```
SVC(random_state=0)
```

5) PREDICTION:

```
y_pred = classifier.predict(X_test)
y_pred
```

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

6) CONFUSION MATRIX:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[64,  4],
       [ 3, 29]])
```

7) ACCURACY:

```
from sklearn import metrics
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.93
```

8) POLUNOMIAL KERNEL:

```
from sklearn.svm import SVC
pclassifier = SVC(kernel='poly',random_state=0)
pclassifier.fit(X_train, y_train)
```

```
SVC(kernel='poly', random_state=0)
```

```
py_pred = pclassifier.predict(X_test)
py_pred
```

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

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, py_pred)
cm
```

```
array([[67,  1],
       [13, 19]])
```

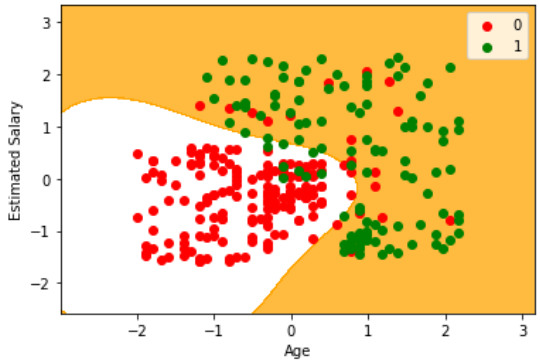
```
from sklearn import metrics
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=py_pred))
```

```
accuracy: 0.86
```

9) PLOT:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('white', 'orange')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
```



CONCLUSION:

From this practical, I have learned and implemented Non-Linear Support Vector Machine(SVM) in python.

AIM: Implementation of Bagging Algorithm: Decision Tree, Random Forest

THEORY:

1) Decision Tree:

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules. Below diagram illustrate the basic flow of decision tree for decision making with labels (Rain(Yes), No Rain(No)).

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a nonparametric supervised learning method used for both classification and regression tasks.

Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

2) Random Forest:

Random forest is an ensemble machine learning algorithm.

It is perhaps the most popular and widely used machine learning algorithm given its good or excellent performance across a wide range of classification and regression predictive modeling problems.

It is also easy to use given that it has few key hyperparameters and sensible heuristics for configuring these hyperparameters.

It is an extension of bootstrap aggregation (bagging) of decision trees and can be used for classification and regression problems.

In bagging, a number of decision trees are created where each tree is created from a different bootstrap sample of the training dataset. A bootstrap sample is a sample of the training dataset where a sample may appear more than once in the sample, referred to as sampling with replacement.

Bagging is an effective ensemble algorithm as each decision tree is fit on a slightly different training dataset, and in turn, has a slightly different performance. Unlike normal decision tree models, such as classification and regression trees (CART), trees used in the ensemble are unpruned, making them slightly overfit to the training dataset. This is desirable as it helps to make each tree more different and have less correlated predictions or prediction errors.

Predictions from the trees are averaged across all decision trees resulting in better performance than any single tree in the model.

1) IMPORTING LIBRARIES:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
```

2) READING DATASET [TRAINING & TESTING]:

```
train_df = pd.read_csv("Disease Training.csv")
test_df = pd.read_csv("Disease Testing.csv")
train_df.head()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_peeling	silver_like_dusting	small_dents_in_nails
0	1	1	1	0	0	0	0	0	0	0	...	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	...	0	0	0	0
4	1	1	1	0	0	0	0	0	0	0	...	0	0	0	0

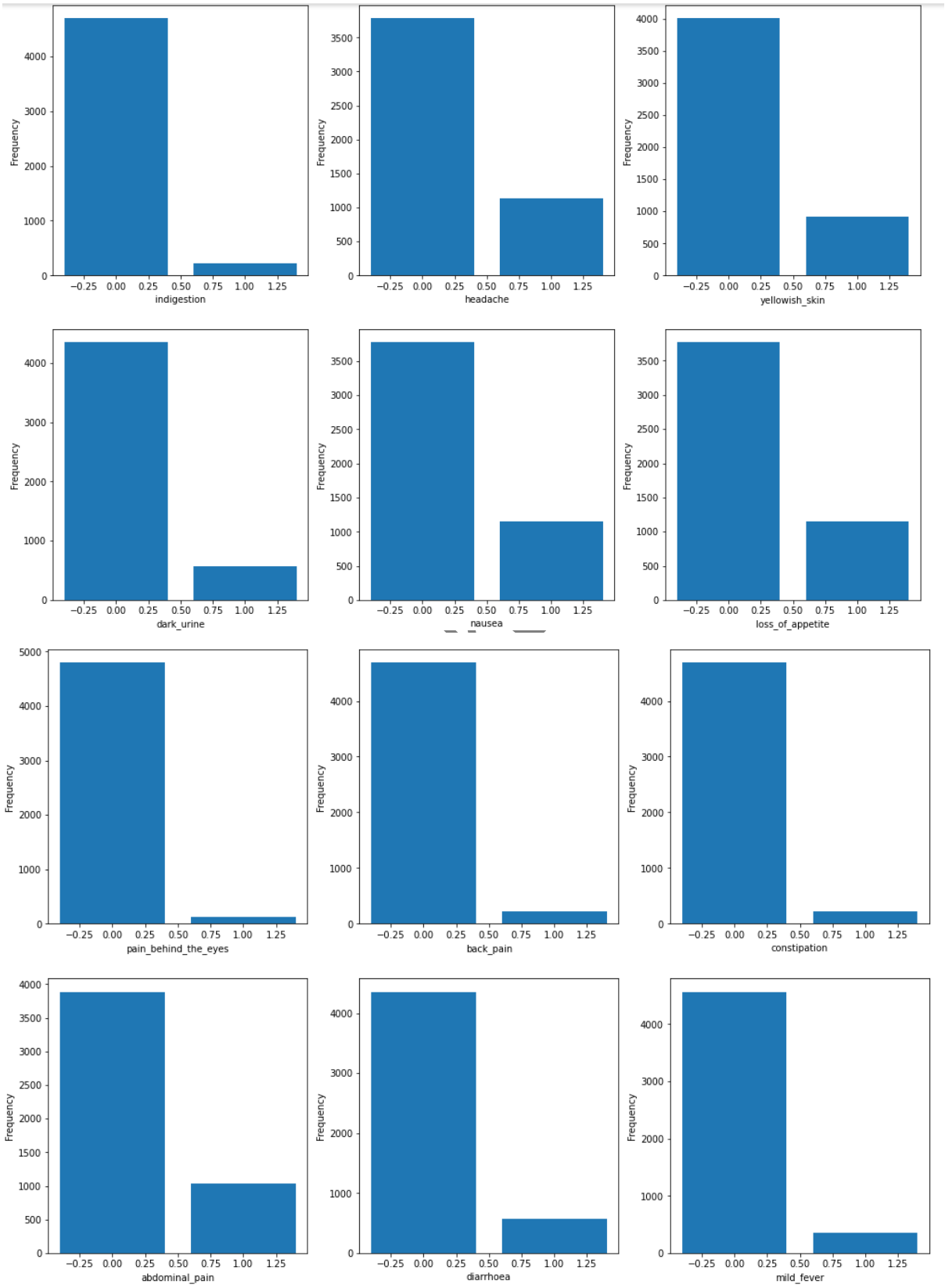
5 rows × 134 columns

3) DATA CLEANING:

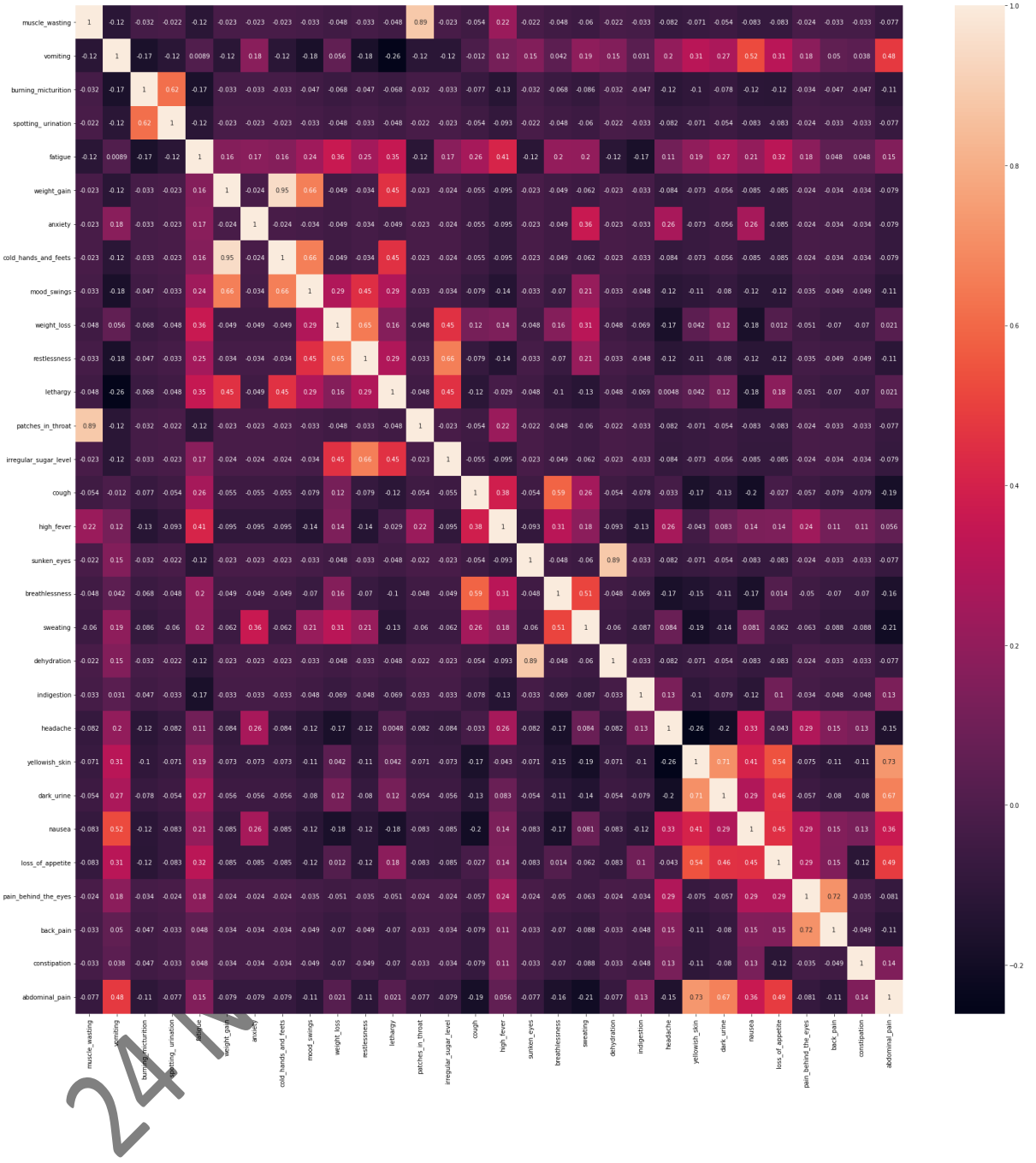
```
# drop unnamed feature from train data
train_df.drop("Unnamed: 133", axis = 1, inplace = True)
# train_df["Unnamed: 133"]  # it's not here anymore
```

4) FEATURES VISULALIZATION:

```
# lets visualize some of features
features = ['indigestion', 'headache', 'yellowish_skin', 'dark_urine', 'nausea',
       'loss_of_appetite', 'pain_behind_the_eyes', 'back_pain', 'constipation',
       'abdominal_pain', 'diarrhoea', 'mild_fever']

plt.figure(figsize = (17,25))
for i, feature in enumerate(features):
    plt.subplot(4,3,i+1)
    plt.bar(train_df[feature].value_counts().index.to_numpy(), train_df[feature].value_counts().values)
    plt.xlabel(feature)
    plt.ylabel("Frequency")
plt.show()
```




```
# linear relationships between some of features using correlation heatmap: for example which symptoms occur together?
df_corr = train_df.iloc[:, 10:40]
plt.figure(figsize = (30, 30))
sns.heatmap(df_corr.corr(), annot = True)
plt.show()
```


muscle_wasting
vomiting
burning_micturition
spotting_ urination
fatigue
weight_gain
anxiety
cold_hands_and_feets
mood_swings
weight_loss
restlessness
lethargy
patches_in_throat
irregular_sugar_level
cough
high_fever
sunken_eyes
breathlessness
sweating
dehydration
indigestion
headache
yellowish_skin
dark_urine
nausea
loss_of_appetite
pain_behind_the_eyes
back_pain
constipation
abdominal_pain
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

5) BUILDING RANDOM FOREST MODEL:

```
from sklearn.ensemble import RandomForestClassifier
x_train, y_train = train_df.loc[:,train_df.columns != "prognosis"], train_df.loc[:,"prognosis"]
x_test, y_test = test_df.loc[:,train_df.columns != "prognosis"], test_df.loc[:,"prognosis"]
rfc = RandomForestClassifier(random_state = 42, n_estimators = 100)
rfc.fit(x_train, y_train)
rfc.predict(x_test)
```

```
array(['Fungal infection', 'Allergy', 'GERD', 'Chronic cholestasis',
       'Drug Reaction', 'Peptic ulcer diseae', 'AIDS', 'Diabetes ',
       'Gastroenteritis', 'Bronchial Asthma', 'Hypertension ', 'Migraine',
       'Cervical spondylosis', 'Paralysis (brain hemorrhage)', 'Jaundice',
       'Malaria', 'Chicken pox', 'Dengue', 'Typhoid', 'hepatitis A',
       'Hepatitis B', 'Hepatitis C', 'Hepatitis D', 'Hepatitis E',
       'Alcoholic hepatitis', 'Tuberculosis', 'Common Cold', 'Pneumonia',
       'Dimorphic hemmorhoids(piles)', 'Heart attack', 'Varicose veins',
       'Hypothyroidism', 'Hyperthyroidism', 'Hypoglycemia',
       'Osteoarthristis', 'Arthritis',
       '(vertigo) Paroymsal  Positional Vertigo', 'Acne',
       'Urinary tract infection', 'Psoriasis', 'Impetigo', 'Impetigo'],
      dtype=object)
```

6) CHECKING RANDOM FOREST SCORE:

```
rfc.score(x_test, y_test)
```

```
0.9761904761904762
```

7) BUILDING BAGGING CLASSIFIER:

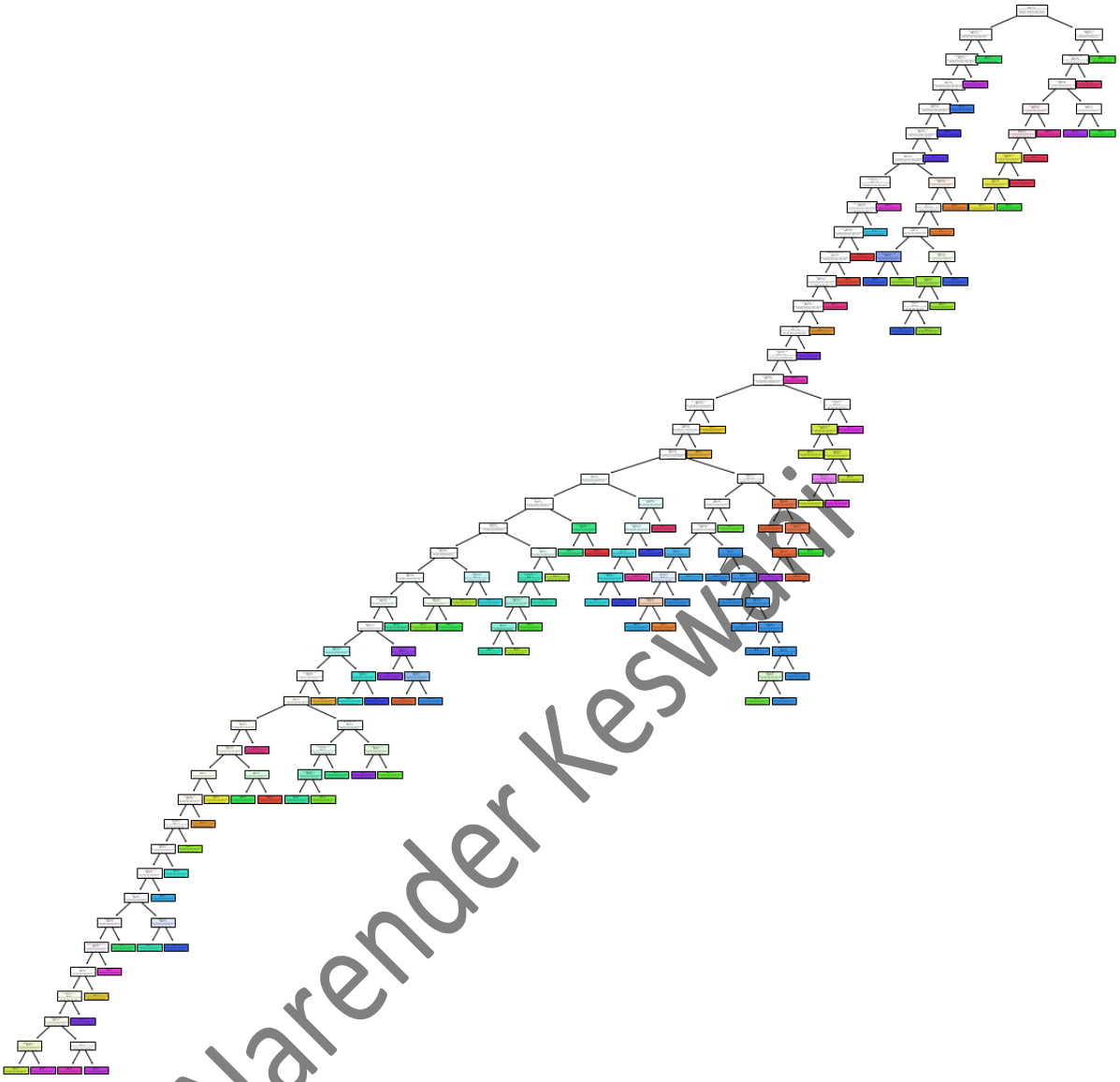
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bagging_clf = BaggingClassifier(base_estimator=tree, n_estimators=1500,
random_state=42)
bagging_clf.fit(x_train, y_train)
y_test_pred = bagging_clf.predict(x_test)
y_train_pred = bagging_clf.predict(x_train)
print("TRAINIG RESULTS: \n===============================")
clf_report = pd.DataFrame(classification_report(y_train, y_train_pred,
output_dict=True))
print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
print(f"CLASSIFICATION REPORT:\n{clf_report}")
print("TESTING RESULTS: \n===============================")
clf_report = pd.DataFrame(classification_report(y_test, y_test_pred,
output_dict=True))
print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```
TRAINIG RESULTS:
================================
CONFUSION MATRIX:
[[120   0   0 ...   0   0   0]
 [  0 120   0 ...   0   0   0]
 [  0   0 120 ...   0   0   0]
 ...
 [  0   0   0 ... 120   0   0]
 [  0   0   0 ...   0 120   0]
 [  0   0   0 ...   0   0 120]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
           (vertigo) Paroymsal  Positional Vertigo   AIDS   Acne  \
precision                                    1.0    1.0    1.0
recall                                       1.0    1.0    1.0
f1-score                                     1.0    1.0    1.0
support                                    120.0  120.0  120.0

           Alcoholic hepatitis  Allergy  Arthritis  Bronchial Asthma  \
precision                  1.0      1.0        1.0               1.0
recall                     1.0      1.0        1.0               1.0
f1-score                   1.0      1.0        1.0               1.0
support                  120.0    120.0      120.0             120.0

           Cervical spondylosis  Chicken pox  Chronic cholestasis  ...  \
precision                   1.0          1.0                  1.0  ...
recall                      1.0          1.0                  1.0  ...
f1-score                    1.0          1.0                  1.0  ...
support                   120.0        120.0                120.0  ...

           Pneumonia  Psoriasis  Tuberculosis  Typhoid  \
precision        1.0        1.0           1.0      1.0
recall           1.0        1.0           1.0      1.0
f1-score         1.0        1.0           1.0      1.0
support        120.0      120.0         120.0    120.0

           Urinary tract infection  Varicose veins  hepatitis A  accuracy  \
precision                      1.0             1.0          1.0       1.0
recall                         1.0             1.0          1.0       1.0
f1-score                       1.0             1.0          1.0       1.0
support                      120.0           120.0        120.0       1.0

           macro avg  weighted avg
precision        1.0           1.0
recall           1.0           1.0
f1-score         1.0           1.0
support       4920.0        4920.0

[4 rows x 44 columns]
```

```
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[1 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
ACCURACY SCORE:
0.9762
CLASSIFICATION REPORT:
           (vertigo) Paroymsal  Positional Vertigo  AIDS  Acne  \
precision                                    1.0   1.0   1.0
recall                                       1.0   1.0   1.0
f1-score                                     1.0   1.0   1.0
support                                      1.0   1.0   1.0

           Alcoholic hepatitis  Allergy  Arthritis  Bronchial Asthma  \
precision                  1.0      1.0        1.0               1.0
recall                     1.0      1.0        1.0               1.0
f1-score                   1.0      1.0        1.0               1.0
support                    1.0      1.0        1.0               1.0

           Cervical spondylosis  Chicken pox  Chronic cholestasis  ...  \
precision                   1.0     0.500000                  1.0  ...
recall                      1.0     1.000000                  1.0  ...
f1-score                    1.0     0.666667                  1.0  ...
support                     1.0     1.000000                  1.0  ...

           Pneumonia  Psoriasis  Tuberculosis  Typhoid  \
precision        1.0        1.0           1.0      1.0
recall           1.0        1.0           1.0      1.0
f1-score         1.0        1.0           1.0      1.0
support          1.0        1.0           1.0      1.0

           Urinary tract infection  Varicose veins  hepatitis A  accuracy  \
precision                      1.0             1.0          1.0   0.97619
recall                         1.0             1.0          1.0   0.97619
f1-score                       1.0             1.0          1.0   0.97619
support                        1.0             1.0          1.0   0.97619

           macro avg  weighted avg
precision   0.987805      0.988095
recall      0.987805      0.976190
f1-score    0.983740      0.976190
support    42.000000     42.000000

[4 rows x 44 columns]
```

8) PLOTTING TREE:

```
!pip install dtreeviz
from dtreeviz.trees import dtreeviz # will be used for tree visualization
from matplotlib import pyplot as plt
from sklearn import tree
plt.figure(figsize=(20,20))
X = pd.DataFrame(train_df, columns=train_df.columns)
_ = tree.plot_tree(rfc.estimators_[0], feature_names=X.columns, filled=True)
```



CONCLUSION:

From this practical, I have learned and implemented the random forest algorithm in python.

Aim: Implementation of AdaBoost Classifier, Gradient Descent Algorithm, Voting Ensemble Algorithm.

THEORY:

A) ADABOOST CLASSIFIER:

- AdaBoost or Adaptive Boosting is one of the ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996.
- It combines multiple weak classifiers to increase the accuracy of classifiers.
- AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier.
- The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations.
- Any machine learning algorithm can be used as base classifier if it accepts weights on the training set.

AdaBoost should meet two conditions :-

1. The classifier should be trained interactively on various weighed training examples.
2. In each iteration, it tries to provide an excellent fit for these examples by minimizing training error.

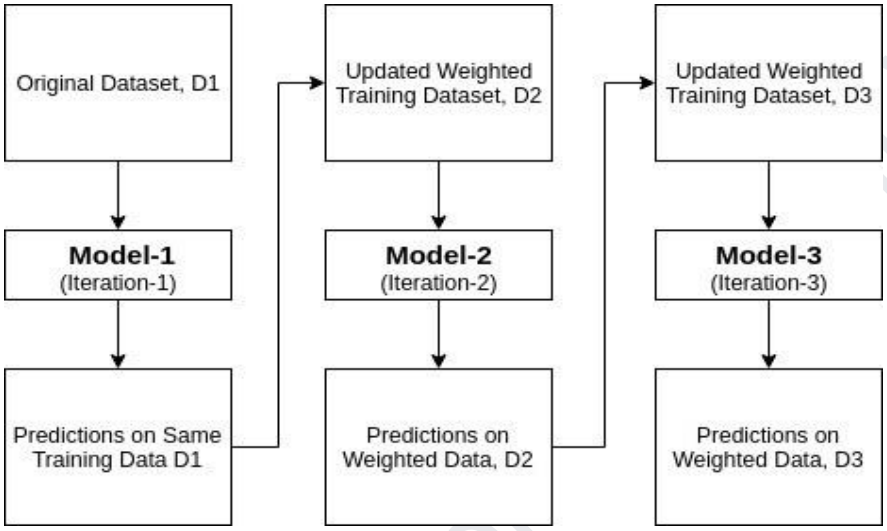
- To build a AdaBoost classifier, imagine that as a first base classifier we train a Decision Tree algorithm to make predictions on our training data.
- Now, following the methodology of AdaBoost, the weight of the misclassified training instances is increased.
- The second classifier is trained and acknowledges the updated weights and it repeats the procedure over and over again.
- At the end of every model prediction we end up boosting the weights of the misclassified instances so that the next model does a better job on them, and so on.
- AdaBoost adds predictors to the ensemble gradually making it better. The great disadvantage of this algorithm is that the model cannot be parallelized since each predictor can only be trained after the previous one has been trained and evaluated.

Below are the steps for performing the AdaBoost algorithm :-

1. Initially, all observations are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, the higher the error the more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached. **AdaBoost algorithm intuition :-**

It works in the following steps:

1. Initially, Adaboost selects a training subset randomly.
2. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
3. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.
4. Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.
5. This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators.
6. To classify, perform a "vote" across all of the learning algorithms you built. The intuition can be depicted with the following diagram



SOURCE CODE:

```
[1] import numpy as nm
    import pandas as pd
    from sklearn import datasets
    data = df = pd.read_csv("iris.data", names=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Species'])
    data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

x=data[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
y=data['Species']
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=1)
adaboost = AdaBoostClassifier(n_estimators=100, base_estimator=None,learning_rate=1)
model = adaboost.fit(x_train,y_train)
y_pred = model.predict(x_test)
print(y_test.values)
print(y_pred)
print("The accuracy of the model on validation set is", accuracy_score(y_test,y_pred))
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-setosa'
 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor']
['Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-setosa'
 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor']
The accuracy of the model on validation set is 0.9777777777777777
```

B) GRADIENT DESCENT ALGORITHM TO OPTIMIZE VARIOUS MODELS.

Gradient Descent is the most common optimization algorithm in *machine learning* and *deep learning*. It is a first-order optimization algorithm. This means it only takes into account the first derivative when performing the updates on the parameters. On each iteration, we update the parameters in the opposite direction of the gradient of the objective function *J(w)* w.r.t the parameters where the gradient gives the direction of the steepest ascent. The size of the step we take on each iteration to reach the local minimum is determined by the learning rate α. Therefore, we follow the direction of the slope downhill until we reach a local minimum.

Types of Gradient Descent

There are three popular types of gradient descent that mainly differ in the amount of data they use:

- **Batch gradient descent**

Batch gradient descent, also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated does the model get updated. This whole process is like a cycle and it's called a training epoch.

- **Stochastic gradient descent**

By contrast, stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for each training example one by one. Depending on the problem, this can make SGD faster than batch gradient descent. One advantage is the frequent updates allow us to have a pretty detailed rate of improvement.
The frequent updates, however, are more computationally expensive than the batch gradient descent approach. Additionally, the frequency of those updates can result in noisy gradients, which may cause the error rate to jump around instead of slowly decreasing.

- **Mini-batch gradient descent**

Mini-batch gradient descent is the go-to method since it’s a combination of the concepts of SGD and batch gradient descent. It simply splits the training dataset into small batches and performs an update for each of those batches. This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Common mini-batch sizes range between 50 and 256, but like any other machine learning technique, there is no clear rule because it varies for different applications. This is the go-to algorithm when training a neural network and it is the most common type of gradient descent within deep learning.

SOURCE CODE:

```
import pandas as pd
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
#Loading Data
mydata= pd.read_csv('pima-indians-diabetes.csv',delimiter=",")
print(mydata)
#spliting data into independent and dependent features
x= mydata.iloc[:,0:8].values
y= mydata.iloc[:,8].values
```

```
       6  148  72  35    0  33.6  0.627  50  1
0      1   85  66  29    0  26.6  0.351  31  0
1      8  183  64   0    0  23.3  0.672  32  1
2      1   89  66  23   94  28.1  0.167  21  0
3      0  137  40  35  168  43.1  2.288  33  1
4      5  116  74   0    0  25.6  0.201  30  0
..    ..  ...  ..  ..  ...   ...    ...  .. ..
762   10  101  76  48  180  32.9  0.171  63  0
763    2  122  70  27    0  36.8  0.340  27  0
764    5  121  72  23  112  26.2  0.245  30  0
765    1  126  60   0    0  30.1  0.349  47  1
766    1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]
```

```
#split data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=1)
#Running various models
models = []
models.append(('LogisticRegression', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVM', SVC()))
models.append(('XGB',XGBClassifier(eta=0.01, gamma=10))) #eta = 0.01, gamma=10
import time
# evaluate each model in turn
results = []
names = []
scoring ='accuracy'

for name, model in models:
 start_time= time.time()
 model.fit(x_train, y_train)
 y_pred = model.predict (x_test)
 predictions = [round(value) for value in y_pred]

 # evaluate predictions
 accuracy = accuracy_score(y_test, predictions)
 print("Accuracy: %.2f%%" % (accuracy * 100.0), name)
 print("--- %s seconds ---"% (time.time() - start_time))
print()
```

```
Accuracy: 75.97% LogisticRegression
--- 0.04490494728088379 seconds ---
Accuracy: 74.68% KNN
--- 0.009744167327880886 seconds ---
Accuracy: 78.57% SVM
--- 0.022034883499145508 seconds ---
Accuracy: 74.68% XGB
--- 0.15872812271118164 seconds ---

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

C) VOTING ENSEMBLE ALGORITHM USING IRIS DATASET:

- Voting is an ensemble machine learning algorithm.
- For regression, a voting ensemble involves making a prediction that is the average of multiple other regression models.
- A voting ensemble (or a "majority voting ensemble") is an ensemble machine learning model that combines the predictions from multiple other models.
- It is a technique that may be used to improve model performance, ideally achieving better performance than any single model used in the ensemble.
- A voting ensemble works by combining the predictions from multiple models. It can be used for classification or regression. In the case of regression, this involves calculating the average of the predictions from the models. In the case of classification, the predictions for each label are summed and the label with the majority vote is predicted.

- **Regression Voting Ensemble**: Predictions are the average of contributing models.
- **Classification Voting Ensemble**: Predictions are the majority vote of contributing models.

There are two approaches to the majority vote prediction for classification; they are hard voting and soft voting.

Hard voting involves summing the predictions for each class label and predicting the class label with the most votes. Soft voting involves summing the predicted probabilities (or probability-like scores) for each class label and predicting the class label with the largest probability.

- **Hard Voting**. Predict the class with the largest sum of votes from models
- **Soft Voting**. Predict the class with the largest summed probability from models.

SOURCE CODE:

```
# importing libraries
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
#loading iris dataset
iris = load_iris()
X = iris.data[:, :4]
Y = iris.target
# train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, Y, test_size=0.20, random_state = 42)
# group / ensemble of models
estimator = []
estimator.append(('LR',LogisticRegression (solver ='lbfgs', multi_class='multinomial', max_iter = 200)))
estimator.append(('SVC', SVC(gamma='auto', probability = True)))
estimator.append(('DTC', DecisionTreeClassifier()))
# Voting Classifier with hard voting
vot_hard = VotingClassifier(estimators = estimator, voting ='hard')
vot_hard.fit(X_train, y_train)
y_pred = vot_hard.predict(X_test)
# using accuracy_score metric to predict accuracy
score = accuracy_score(y_test, y_pred)
print("Hard Voting Score %d" % score)
# Voting Classifier with soft voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(X_train, y_train)
y_pred = vot_soft.predict(X_test)
# using accuracy_score
score = accuracy_score(y_test, y_pred)
print("Soft Voting Score %d" % score)
```

```
Hard Voting Score 1
Soft Voting Score 1
```

CONCLUSION:

From this practical, I have successfully learned & implemented boosting algorithms such as: AdaBoost Classifier, Gradient Descent Algorithm, Voting Ensemble Algorithm

Aim: Implementation of dimensionality reduction techniques: Normalization, Transformation, Principal Components Analysis.

A) NORMALIZATION:-

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modeling.

Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

SOURCE CODE:

1) Importing Datasets:

```
[2] from sklearn import preprocessing
    import pandas as pd
    housing = pd.read_csv("/content/sample_data/california_housing_train.csv")
    housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

2) Normalizing- MinMaxScalar Transformation of Dataset:

```
[27] scaler = preprocessing.MinMaxScaler()
     names = housing.columns
     d = scaler.fit_transform(housing)
     scaled_df = pd.DataFrame(d, columns=names)
     scaled_df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	1.000000	0.175345	0.274510	0.147885	0.198945	0.028364	0.077454	0.068530	0.107012
1	0.984064	0.197662	0.352941	0.201608	0.294848	0.031559	0.075974	0.091040	0.134228
2	0.975100	0.122210	0.313725	0.018927	0.026847	0.009249	0.019076	0.079378	0.145775
3	0.974104	0.116897	0.254902	0.039515	0.052142	0.014350	0.037000	0.185639	0.120414
4	0.974104	0.109458	0.372549	0.038276	0.050435	0.017405	0.042921	0.098281	0.104125

B) Principal Component Analysis:-

Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations

among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

SOURCE CODE:

1) Importing Dataset:

```
import pandas as pd
iris_df = pd.read_csv("iris.data",names=['sepal length','sepal width','petal length','petal width','target'])
iris_df.head()
```

	sepal length	sepal width	petal length	petal width	target
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

2) Standardize the Data:

```
from sklearn.preprocessing import StandardScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Separating out the features
x = iris_df.loc[:, features].values
# Separating out the target
y = iris_df.loc[:,['target']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
print("Standardized features:-\n", x)
```

```
Standardized features:-
 [[-9.00681170e-01  1.03205722e+00 -1.34127240e+00 -1.31297673e+00]
 [-1.14301691e+00 -1.24957601e-01 -1.34127240e+00 -1.31297673e+00]
 [-1.38535265e+00  3.37848329e-01 -1.39813811e+00 -1.31297673e+00]
 [-1.50652052e+00  1.06445364e-01 -1.28440670e+00 -1.31297673e+00]
 [-1.02184904e+00  1.26346019e+00 -1.34127240e+00 -1.31297673e+00]
 [-5.37177559e-01  1.95766909e+00 -1.17067529e+00 -1.05003079e+00]
 [-1.50652052e+00  8.00654259e-01 -1.34127240e+00 -1.18150376e+00]
 [-1.02184904e+00  8.00654259e-01 -1.28440670e+00 -1.31297673e+00]
 [-1.74885626e+00 -3.56360566e-01 -1.34127240e+00 -1.31297673e+00]
 [-1.14301691e+00  1.06445364e-01 -1.28440670e+00 -1.44444970e+00]
 [-5.37177559e-01  1.49486315e+00 -1.28440670e+00 -1.31297673e+00]
 [-1.26418478e+00  8.00654259e-01 -1.22754100e+00 -1.31297673e+00]
 [-1.26418478e+00 -1.24957601e-01 -1.34127240e+00 -1.44444970e+00]
 [-1.87002413e+00 -1.24957601e-01 -1.51186952e+00 -1.44444970e+00]
 [-5.25060772e-02  2.18907205e+00 -1.45500381e+00 -1.31297673e+00]
 [-1.73673948e-01  3.11468391e+00 -1.28440670e+00 -1.05003079e+00]
 [-5.37177559e-01  1.95766909e+00 -1.39813811e+00 -1.05003079e+00]
 [-9.00681170e-01  1.03205722e+00 -1.34127240e+00 -1.18150376e+00]
 [-1.73673948e-01  1.72626612e+00 -1.17067529e+00 -1.18150376e+00]
 [-9.00681170e-01  1.72626612e+00 -1.28440670e+00 -1.18150376e+00]
 [-5.37177559e-01  8.00654259e-01 -1.17067529e+00 -1.31297673e+00]
 [-9.00681170e-01  1.49486315e+00 -1.28440670e+00 -1.05003079e+00]
 [-1.50652052e+00  1.26346019e+00 -1.56873522e+00 -1.31297673e+00]
```

3) PCA Projection to 2D:

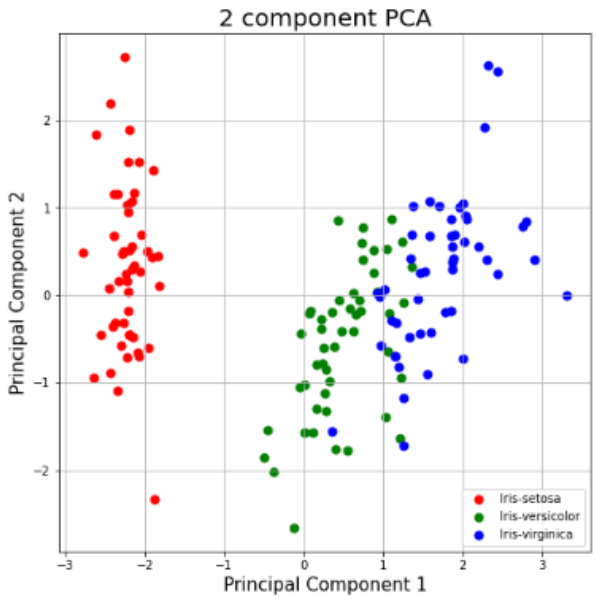
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, iris_df[['target']]], axis = 1)
finalDf
```

	principal component 1	principal component 2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica
148	1.376391	1.016362	Iris-virginica
149	0.959299	-0.022284	Iris-virginica

150 rows × 3 columns

4) Visualize 2D Projection:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
 indicesToKeep = finalDf['target'] == target
 ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
 , finalDf.loc[indicesToKeep, 'principal component 2']
 , c = color
 , s = 50)
ax.legend(targets)
ax.grid()
```



CONCLUSION:

From this practical, I have learned and implemented dimensionality reduction techniques: Normalization, Transformation, Principal Components Analysis.

AIM: DEPLOYMENT OF MACHINE LEARNING MODELS

THEORY:

ML-Model-Flask-Deployment

This is a mini project to elaborate how Machine Learn Models are deployed on production using Flask API

Prerequisites

You must have Numpy, Scikit Learn, Pandas (for Machine Learning Model) and Flask (for API) installed.

This project has four major parts :

model.py - This contains code for our Machine Learning model to predict employee salaries based on training data in 'hiring.csv' file.

app.py - This contains Flask APIs that receives employee details through GUI, computes the predicted value based on our model and returns it.

request.py - This uses a requests module to call APIs already defined in app.py and displays the returned value.

templates - This folder contains the HTML template to allow users to enter employee detail and displays the predicted employee salary.

Running the project

Create the machine learning model by running below command -

python model.py

This would create a serialized version of our model into a file

model.pkl Run app.py using below command to start Flask API *python*

app.py

By default, flask will run on port 5000.

Navigate to URL http://localhost:5000

You should be able to view the homepage.

Enter valid numerical values in all 3 input boxes and hit Predict.

You will be able to see the predicted salary value on the HTML page.

I) CREATING MODEL:

A) IMPORTING LIBRARIES:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

B) READING DATASET:

```
df = pd.read_csv('parkinsons.data')
```

```
df.head()
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	Shimmer:APQ3	Shimmer:APQ5	MDVP:APQ	Shimmer:DDA
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	0.426	0.02182	0.03130	0.02971	0.06545
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	0.626	0.03134	0.04518	0.04368	0.09403
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	0.482	0.02757	0.03858	0.03590	0.08270
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	0.517	0.02924	0.04005	0.03772	0.08771
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	0.584	0.03490	0.04825	0.04465	0.10470

C) LISITING ALL COLUMNS:

```
df.columns
```

```
Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)',
       'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
       'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

D) DEPTH INFO OF DATASET:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64  
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

E) <u>COUNTING NULL VALUES:</u>

```
df.isnull().sum()
```

```
name              0
MDVP:Fo(Hz)       0
MDVP:Fhi(Hz)      0
MDVP:Flo(Hz)      0
MDVP:Jitter(%)    0
MDVP:Jitter(Abs)  0
MDVP:RAP          0
MDVP:PPQ          0
Jitter:DDP        0
MDVP:Shimmer      0
MDVP:Shimmer(dB)  0
Shimmer:APQ3      0
Shimmer:APQ5      0
MDVP:APQ          0
Shimmer:DDA       0
NHR               0
HNR               0
status            0
RPDE              0
DFA               0
spread1           0
spread2           0
D2                0
PPE               0
dtype: int64
```

F) DIMENSIONS OF DATASET:

```
df.shape
```

```
(195, 24)
```

G) SPLITTING DATASET:

```
X = df.drop(['name'], 1)
X = X.drop(['status'], 1)
y = df['status']
```

```
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)
```

H) NORMALIZATION:

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

I) BUILDING XGBOOST MODEL:

```
from xgboost import XGBClassifier
model = XGBClassifier().fit(X_train, y_train)
```

```
predictions = model.predict(X_test)
```

J) CHECKING ACCURACY OF THE MODEL:

```
from sklearn.metrics import accuracy_score, f1_score
```

```
accuracy_score(y_test, predictions)
```

```
0.8974358974358975
```

```
f1_score(y_test, predictions)
```

```
0.9354838709677419
```

K) EXPORTING MODEL:

```
import pickle
# Writing different model files to file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(model,f)

with open('standardScalar.sav', 'wb') as f:
    pickle.dump(sc,f)
```

II) DEPLOYMENT OF MODEL:

FRONTEND:
HTML:
Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Parkinson's Diseases Prediction</title>
<link
href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display=swap"
rel="stylesheet" />
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"
  integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
  crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
  integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
  crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
  integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
  crossorigin="anonymous"></script>
</head>
<body>
    <header>
      <div class="container">
```

```
                <h1>
                  <p style="text-align: center">Parkinson's Diseases Prediction</p>
                </h1>
              </div>
            </header>

          <div class="container">

              <form action="/predict" method="POST">
                <div class="mb-3">
                <input type="float" name="mdvp_fo"  placeholder="MDVP:Fo(Hz)
range(88,260)" class="form-control" required/><br/>
                <input type="float" name="mdvp_fhi"  placeholder="MDVP:Fhi(Hz)
range(102,592)" class="form-control" required><br/>
                <input type="float" name="mdvp_flo" id="mdvp_flo"
placeholder="MDVP:Flo(Hz) range(65,240)" class="form-control" required><br />
                <input type="float" name="mdvp_jitper" id="mdvp_jitper"
placeholder="MDVP:Jitter(%) range(0.001, 0.033)" class="form-control" required><br />
                <input type="float" name="mdvp_jitabs" id="mdvp_jitabs"
placeholder="MDVP:Jitter(Abs) range(0.00002, 0.0002)" class="form-control"
required><br />
                <input type="float" name="mdvp_jitabs" id="mdvp_jitabs"
placeholder="MDVP:Jitter(Abs) range(0.00002, 0.0002)" class="form-control"
required><br />
                <input type="float" name="mdvp_rap" id="mdvp_rap"
placeholder="MDVP:RAP range(0.0006, 0.02)" class="form-control" required><br />
                <input type="float" name="mdvp_ppq" id="mdvp_ppq"
placeholder="MDVP:PPQ range(0.0009, 0.02)" class="form-control" required><br />
                <input type="float" name="jitter_ddp" id="jitter_ddp" placeholder="Jitter:DDP
range(0.002, 0.065)" class="form-control" required><br />
                <input type="float" name="mdvp_shim" id="mdvp_shim"
placeholder="MDVP:Shimmer range(0.009, 0.12)" class="form-control" required><br />
                <input type="float" name="mdvp_shim_db" id="mdvp_shim_db"
placeholder="MDVP:Shimmer(dB) range(0.085, 1.302)" class="form-control"
required><br />
                <input type="float" name="shimm_apq3" id="shimm_apq3"
placeholder="Shimmer:APQ3 range(0.004, 0.056)" class="form-control" required><br />
                <input type="float" name="shimm_apq5" id="shimm_apq5"
placeholder="Shimmer:APQ5 range(0.005, 0.08)" class="form-control" required><br />
                <input type="float" name="mdvp_apq" id="mdvp_apq"
placeholder="MDVP:APQ range(0.007, 0.14)" class="form-control" required><br />
                <input type="float" name="shimm_dda" id="shimm_dda"
placeholder="Shimmer:DDA range(0.013, 0.17)" class="form-control" required><br />
                <input type="float" name="nhr" id="nhr" placeholder="NHR range(0.0006,
0.31)" class="form-control" required><br />
                <input type="float" name="hnr" id="hnr" placeholder="HNR range(8, 33)"
class="form-control" required><br />
                <input type="float" name="rpde" id="rpde" placeholder="RPDE range(0.25,
0.68)" class="form-control" required><br />
                <input type="float" name="dfa" id="dfa" placeholder="DFA range(0.57, 0.82)"
class="form-control" required><br />
```

```
            <input type="float" name="spread1" id="spread1" placeholder="Spread1
  range(-7, -2)" class="form-control" required><br />
            <input type="float" name="spread2" id="spread2" placeholder="Spread2
  range(0.006, 0.45)" class="form-control" required><br />
            <input type="float" name="d2" id="d2" placeholder="D2 range(1.42, 3.67)"
  class="form-control" required><br />
            <input type="float" name="ppe" id="ppe" placeholder="PPE range(0.04, 0.5)"
  class="form-control" required><br />

            <button type="submit" class="btn btn-primary"
  value="Predict">Predict</button>
          </div>
        </form>
      </div>
  </body>
  </html>
```

Result.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Predicted Result</title>
  <link
href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display=swap"
rel="stylesheet" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"
    integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
    integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
    integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
    crossorigin="anonymous"></script>
</head>
<body>
  <div class="container">
    <h2>Predicted Result</h2>
    <p>{{prediction}}</p>
  </div>
```

```
</body>
</html>
```

BACKEND:

PYTHON:

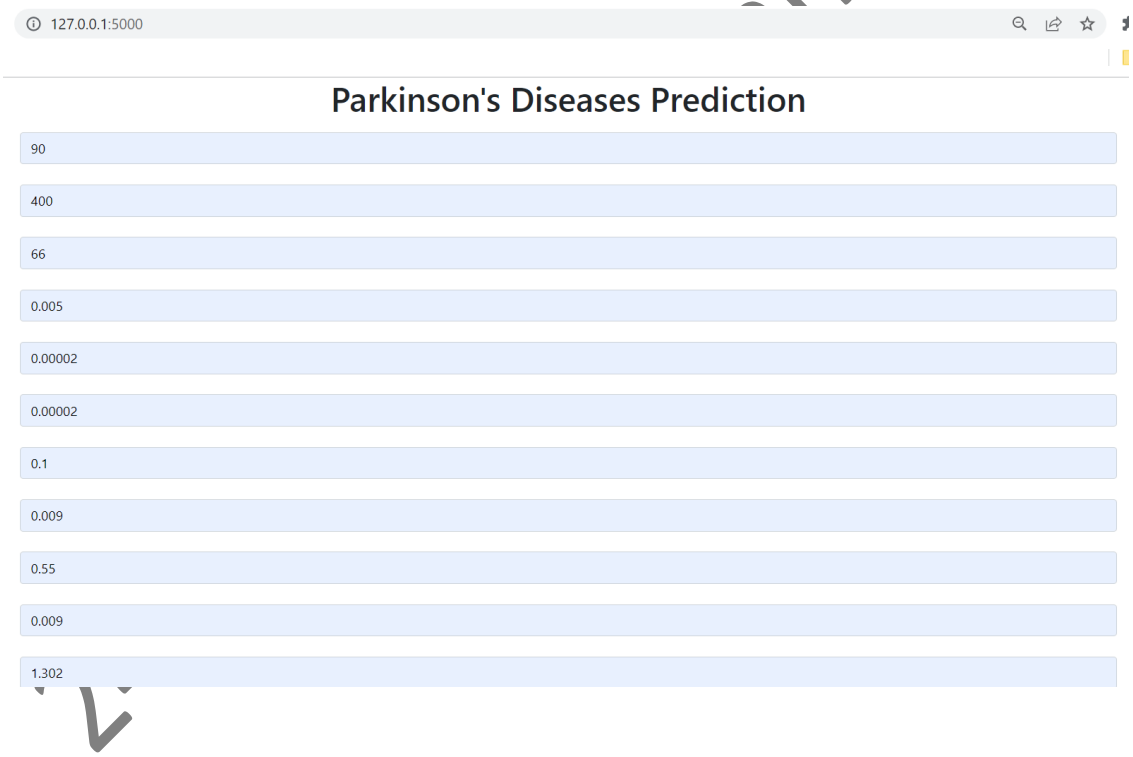
app.py:

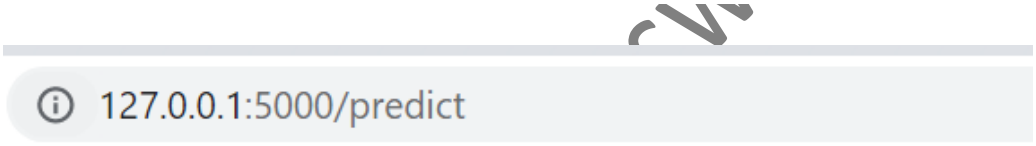
```
# importing the necessary dependencies
from flask import Flask, render_template, request
from flask_cors import CORS,cross_origin
import pickle

app = Flask(__name__) # initializing a flask app

@app.route('/',methods=['GET'])  # route to display the home page
@cross_origin()
def homePage():
    return render_template("index.html")

@app.route('/predict',methods=['POST','GET']) # route to show the predictions in a web
UI
@cross_origin()
def index():
    if request.method == 'POST':
        #  reading the inputs given by the user
        mdvp_fo=float(request.form['mdvp_fo'])
        mdvp_fhi=float(request.form['mdvp_fhi'])
        mdvp_flo=float(request.form['mdvp_flo'])
        mdvp_jitper=float(request.form['mdvp_jitper'])
        mdvp_jitabs=float(request.form['mdvp_jitabs'])
        mdvp_rap=float(request.form['mdvp_rap'])
        mdvp_ppq=float(request.form['mdvp_ppq'])
        jitter_ddp=float(request.form['jitter_ddp'])
        mdvp_shim=float(request.form['mdvp_shim'])
        mdvp_shim_db=float(request.form['mdvp_shim_db'])
        shimm_apq3=float(request.form['shimm_apq3'])
        shimm_apq5=float(request.form['shimm_apq5'])
        mdvp_apq=float(request.form['mdvp_apq'])
        shimm_dda=float(request.form['shimm_dda'])
        nhr=float(request.form['nhr'])
        hnr=float(request.form['hnr'])
        rpde=float(request.form['rpde'])
        dfa=float(request.form['dfa'])
        spread1=float(request.form['spread1'])
        spread2=float(request.form['spread2'])
        d2=float(request.form['d2'])
        ppe=float(request.form['ppe'])

        filename = 'modelForPrediction.sav'
        loaded_model = pickle.load(open(filename, 'rb')) # loading the model file from the
storage
```

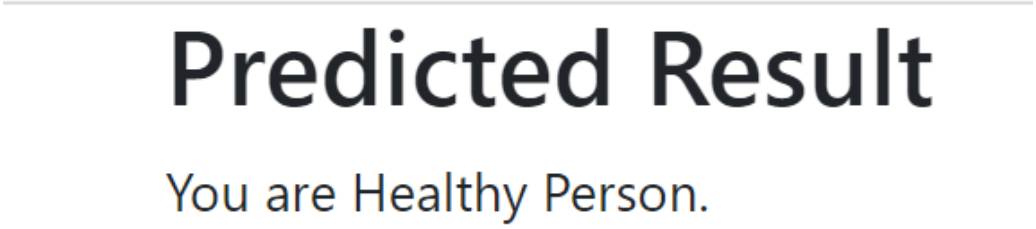
```
        # predictions using the loaded model file
    scaler = pickle.load(open('standardScalar.sav', 'rb'))
    prediction=loaded_model.predict(scaler.transform([[mdvp_fo,mdvp_fhi,mdvp_flo,
mdvp_jitper, mdvp_jitabs,
          mdvp_rap,mdvp_ppq, jitter_ddp, mdvp_shim,
mdvp_shim_db,shimm_apq3,shimm_apq5,mdvp_apq,shimm_dda,nhr,hnr,rpde,dfa,spr
ead1,spread2,d2,ppe]]))
    print('prediction is', prediction)
    if prediction == 1:
      pred = "You have Parkinson's Disease. Please consult a specialist."
      return render_template('results.html', prediction=pred)
    else:
      pred = "You are Healthy Person."
      # showing the prediction results in a UI
      return render_template('results.html',prediction=pred)
  else:
    return render_template('index.html')


if __name__ == "__main__":
  #app.run(host='127.0.0.1', port=8001, debug=True)
  app.run(debug=False) # running the app
```

OUTPUT:


MDVP:APQ range(0.007, 0.14)
Shimmer:DDA range(0.013, 0.17)
NHR range(0.0006, 0.31)
HNR range(8, 33)
RPDE range(0.25, 0.68)
DFA range(0.57, 0.82)
Spread1 range(-7, -2)
Spread2 range(0.006, 0.45)
D2 range(1.42, 3.67)
PPE range(0.04, 0.5)
Predict


127.0.0.1:5000
Parkinson's Diseases Prediction
90
400
66
0.005
0.00002
0.00002
0.1
0.009
0.55
0.009
1.302





CONCLUSION:

From this practical, I have learned how to deploy machine learning applications using flask python framework.