EXPERIMENT-1:

Aim: Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation

Mean:

```
# Python program to print the mean of elements

# List of elements to calculate mean

n_num = [1, 2, 3, 4, 5]

# Calculate the number of elements

n = len(n_num)

# Calculate the sum of elements

get_sum = sum(n_num)

# Calculate the mean

mean = get_sum / n

# Print the result

print("Mean / Average is: " + str(mean))
```

Output:

Mean / Average is: 3.0

Median:

```
# Python program to print the median of elements
# List of elements to calculate the median
n_num = [1, 2, 3, 4, 5]
# Number of elements in the list
n = len(n_num)
# Sort the list
n_num.sort()
# Calculate median
if n % 2 == 0:
    median1 = n_num[n // 2]
    median2 = n_num[n // 2 - 1]
    median = (median1 + median2) / 2
else:
    median = n_num[n // 2]
print("Median is:", median)
```

Output:
Median is: 3

Mode:

```
# Python program to print the mode of elements
from collections import Counter
# List of elements to calculate mode
n_num = [1, 2, 3, 4, 5, 5]
# Count the frequency of each element
data = Counter(n_num)
# Get the mode(s)
get_mode = dict(data)
mode = [k for k, v in get_mode.items() if v == max(list(data.values()))]
# Check if there is a mode
if len(mode) == len(n_num):
   get_mode = "No mode found"
else:
   get_mode = "Mode is/are: " + ', '.join(map(str, mode))
# Print the result
print(get_mode)
```

Output:
Mode is/are: 5

Variance:

```
# Python program to get the variance of a list
# Importing the NumPy module
import numpy as np
# Taking a list of elements
numbers = [2, 4, 4, 4, 5, 5, 7, 9]
# Calculating variance using var()
variance = np.var(numbers)
# Printing the variance
print("Variance of the list is:", variance)
```

Output:
Variance of the list is: 4.0

Standard Deviation:

```
# Python program to get the standard deviation of a list
# Importing the NumPy module
import numpy as np
# Taking a list of elements
numbers = [2, 4, 4, 4, 5, 5, 7, 9]
# Calculating standard deviation using std()
std_deviation = np.std(numbers)
# Printing the standard deviation
print("Standard Deviation of the list is:", std_deviation)
```

Output:
Standard Deviation of the list is: 2.0

Sample Viva Questions

1. What are Central Tendency Measures: Mean, Median, Mode

- **Central Tendency Measures** are statistical tools used to summarize a set of data points. These measures include:
  - **Mean**: The average of all numbers in a dataset.
  - **Median**: The middle value when the data points are ordered in ascending or descending order.
  - **Mode**: The number that appears most frequently in a dataset.

2. Define Mean, Mode, and Median

- **Mean**: The mean is the average of all numbers in a dataset. It is calculated by summing all the values and dividing by the total number of values.
- **Median**: The median is the middle number in a set of ordered numbers. If the dataset has an odd number of values, the median is the middle value. If even, it is the average of the two middle values.
- **Mode**: The mode is the number that occurs most frequently in a dataset. If no number repeats, the dataset is said to have no mode. If multiple numbers repeat, it is multimodal.

3. Define Variance, Standard Deviation

- **Variance**: Variance is a measure of how much the numbers in a dataset differ from the mean. It is calculated by finding the average of the squared differences from the mean.
  - Formula for variance:

    $$\text{Variance} = (1/n) * \Sigma (x_i - \mu)^2$$

- **Standard Deviation**: Standard deviation is the square root of the variance and provides an indication of how spread out the data is from the mean.

4. What is the syntax for input and output in Python?

Input: In Python, we use the `input()` function to take input from the user. It always returns the input as a string by default python

5. Which python version your using?

To check your Python version, run python --version in your command line (Windows), shell (Mac), or terminal (Linux/Ubuntu).

EXPERIMENT-2:

Aim: Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

i) Statistics

Python's statistics is a built-in Python library for descriptive statistics. You can use it if your datasets are not too large or if you can't rely on importing other libraries. NumPy is a third-party library for numerical computing, optimized for working with single- and multi- dimensional arrays.

Understanding Descriptive Statistics

Descriptive statistics is about describing and summarizing data. It uses two main approaches:

- The quantitative approach describes and summarizes data
- The visual approach illustrates data with charts, plots, histograms, and other graphs.

You can apply descriptive statistics to one or many datasets or variables. When you describe and summarize a single variable, you're performing univariate analysis. When you search for statistical relationships among a pair of variables, you're doing a bivariate analysis. Similarly, a multivariate analysis is concerned with multiple variables at once.

There are many Python statistics libraries out there for you to work with, but in this tutorial, you'll be learning about some of the most popular and widely used ones:

- Python's statistics is a built-in Python library for descriptive You can use it if your datasets are not too large or if you can't rely on importing other libraries.
- NumPy is a third-party library for numerical computing, optimized for working with single- and multi-dimensional Its primary type is the array type called ndarray. This library contains many routines for statistical analysis.
- SciPy is a third-party library for scientific computing based on NumPy. It offers additional functionality compared to NumPy, including scipy.stats for statistical

- pandas is a third-party library for numerical computing based on It excels in handling labeled one-dimensional (1D) data with Series objects and two- dimensional (2D) data with DataFrame objects.
- Matplotlib is a third-party library for data visualization. It works well in combination with NumPy, SciPy, and pandas.

Note that, in many cases, Series and DataFrame objects can be used in place of NumPy arrays. Often, you might just pass them to a NumPy or SciPy statistical function. In addition, you can get the unlabeled data from a Series or DataFrame as a np.ndarray object by calling .values or .to_numpy().

Getting Started With Python Statistics Libraries

The built-in Python statistics library has a relatively small number of the most important statistics functions. The official documentation is a valuable resource to find the details. If you're limited to pure Python, then the Python statistics library might be the right choice.

A good place to start learning about NumPy is the official User Guide, especially the quickstart and basics sections. The official reference can help you refresh your memory on specific NumPy concepts. While you read this tutorial, you might want to check out the statistics section and the official scipy.stats reference as well.

ii)Math

To carry out calculations with real numbers, the Python language contains many additional functions collected in a library (module) called math.

To use these functions at the beginning of the program, you need to connect the math library, which is done by the command

import math

Python provides various operators for performing basic calculations, such as * for multiplication,% for a module, and / for the division. If you are developing a program in Python to perform certain tasks, you need to work with trigonometric functions, as well as complex numbers. Although you cannot use these functions directly, you can

access them by turning on the math module math, which gives access to hyperbolic, trigonometric and logarithmic functions for real numbers. To use complex numbers, you can use the math module cmath. When comparing *math* vs *numpy*, a *math* library is more lightweight and can be used for extensive computation as well.

The Python Math Library is the foundation for the rest of the math libraries that are written on top of its functionality and functions defined by the C standard. Please refer to the python math examples for more information.

Number-theoretic and representation functions

This part of the mathematical library is designed to work with numbers and their representations. It allows you to effectively carry out the necessary transformations with support for NaN (not a number) and infinity and is one of the most important sections of the Python math library. Below is a short list of features for Python 3rd version. A more detailed description can be found in the documentation for the math library.

math.ceil(x) – return the ceiling of x, the smallest integer greater than or equal to x

math.comb(n, k) – return the number of ways to choose k items from n items without repetition and without order

math.copysign(x, y) – return float with the magnitude (absolute value) of x but the sign of

1. On platforms that support signed zeros, copysign (1.0, -0.0) returns -1.0

math.fabs(x) – return the absolute value of x

math.factorial(x) – return x factorial as an integer. Raises ValueError if x is not integral or is negative

math.floor(x) – return the floor of x, the largest integer less than or equal to x

math.fmod(x, y) – return fmod(x, y), as defined by the platform C library

math.frexp(x) – return the mantissa and exponent of x as the pair (m, e). m is a float and e is an integer such that x == m * 2**e exactly

math.fsum(iterable) – return an accurate floating-point sum of values in the iterable

math.gcd(a, b) – return the greatest common divisor of the integers a and b

math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0) – return True if the values a and b are close to each other and False otherwise

math.isfinite(x) – return True if x is neither infinity nor a NaN, and False otherwise (note that 0.0 is considered finite)

math.isinf(x) – return True if x is positive or negative infinity, and False otherwise

math.isnan(x) – return True if x is a NaN (not a number), and False otherwise

math.isqrt(n) – return the integer square root of the nonnegative integer n. This is the floor of the exact square root of n, or equivalently the greatest integer a such that $a^2 \leq n$

math.ldexp(x, i) – return x * (2i).** This is essentially the inverse of function frexp()

math.modf(x) – return the fractional and integer parts of x. Both results carry the sign of x and are floats

math.perm(n, k=None) – return the number of ways to choose k items from n items without repetition and with order

math.prod(iterable, *, start=1) – calculate the product of all the elements in the input iterable. The default start value for the product is 1

math.remainder(x, y) – return the IEEE 754-style remainder of x with respect to y

math.trunc(x) – return the Real value x truncated to an Integral (usually an integer)

Power and logarithmic functions

The power and logarithmic functions section are responsible for exponential calculations, which is important in many areas of mathematics, engineering, and statistics. These

functions can work with both natural logarithmic and exponential functions, logarithms modulo two, and arbitrary bases.

math.exp(x) – return e raised to the power x, where e = 2.718281… is the base of natural logarithms

math.expm1(x) – return e raised to the power x, minus 1. Here e is the base of natural logarithms. math.log(x[, base]) – With one argument, return the natural logarithm of x (to base e). With two arguments, return the logarithm of x to the given base, calculated as log(x)/log(base)

math.log1p(x) – return the natural logarithm of 1+x (base e). The result is calculated in a way that is accurate for x near zero

math.log2(x) – return the base-2 logarithm of x. This is usually more accurate than log(x, 2)

math.log10(x) – return the base-10 logarithm of x. This is usually more accurate than log(x, 10)

math.pow(x, y) – return x raised to the power y

math.sqrt(x) – return the square root of x

Trigonometric functions

Trigonometric functions, direct and inverse, are widely represented in the Python Mathematical Library. They work with radian values, which is important. It is also possible to carry out calculations with Euclidean functions.

math.acos(x) – return the arc cosine of x, in radians

math.asin(x) – return the arc sine of x, in radians

math.atan(x) – return the arctangent of x, in radians

math.atan2(y, x) – return atan(y / x), in radians. The result is between -pi and pi

math.cos(x) – return the cosine of x radians

math.dist(p, q) – return the Euclidean distance between two points p and q, each given as a sequence (or iterable) of coordinates. The two points must have the same dimension

math.hypot(*coordinates) – return the Euclidean norm, sqrt(sum(x**2 for x in coordinates)). This is the length of the vector from the origin to the point given by the coordinates

math.sin(x) – return the sine of x radians

math.tan(x) – return the tangent of x radians

Angular conversion

Converting degrees to radians and vice versa is a fairly common function and therefore the developers have taken these actions to the Python library. This allows you to write compact and understandable code.

math.degrees(x) – convert angle x from radians to degrees

math.radians(x) – convert angle x from degrees to radians

Hyperbolic functions

Hyperbolic functions are analogs of trigonometric functions that are based on hyperbolas instead of circles.

math.acosh(x) – return the inverse hyperbolic cosine of x

math.asinh(x) – return the inverse hyperbolic sine of x

math.atanh(x) – return the inverse hyperbolic tangent of x

math.cosh(x) – return the hyperbolic cosine of x

math.sinh(x) – return the hyperbolic sine of x

math.tanh(x) – return the hyperbolic tangent of x

Special functions

The special functions section is responsible for error handling and gamma functions. This is a necessary function and it was decided to implement it in the standard Python mathematical library.

math.erf(x) – Return the error function at x

math.erfc(x) – Return the complementary error function at x

math.gamma(x) – Return the Gamma function at x

math.lgamma(x) – Return the natural logarithm of the absolute value of the Gamma function at x

Constants

The constant section provides ready-made values for basic constants and writes them with the necessary accuracy for a given hardware platform, which is important for Python's portability as a cross-platform language. Also, the very important values infinity and "not a number" are defined in this section of the Python library.

math.pi – the mathematical constant π = 3.141592…, to available precision

math.e – the mathematical constant e = 2.718281…, to available precision

math.tau – the mathematical constant τ = 6.283185…, to available precision. Tau is a circle constant equal to 2π, the ratio of a circle's circumference to its radius

math.inf – a floating-point positive infinity. (For negative infinity, use -math.inf.) Equivalent to the output of float('inf')

math.nan – a floating-point "not a number" (NaN) value. Equivalent to the output of float('nan')

iii)Scipy

SciPy is a library for the open-source Python programming language, designed to perform scientific and engineering calculations.

The capabilities of this library are quite wide:

- Search for minima and maxima of functions
- Calculation of function integrals
- Support for special functions
- Signal processing
- Image processing
- Work with genetic algorithms
- Solving ordinary differential equations

SciPy in Python is a collection of mathematical algorithms and functions built as a Numpy extension. It greatly extends the capabilities of an interactive Python session by providing the user with high-level commands and classes for managing and visualizing data. With SciPy, an interactive Python session becomes a data processing and prototyping system competing with systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

An additional advantage of Python-based SciPy is that it is also a fairly powerful programming language used in the development of complex programs and specialized applications. Scientific applications also benefit from the development of additional modules in numerous software niches by developers around the world. Everything from parallel programming for the web to routines and database classes

is available to the Python programmer. All of these features are available in addition to the SciPy math library.

Packages for mathematical methods

SciPy is organized into sub-packages covering various scientific computing areas:

cluster – Clustering Algorithms

constants – physical and mathematical constants

fftpack – Fast Fourier Transform subroutines

integrate – integration and solution of ordinary differential equations

Interpolate – interpolation and smoothing splines

io – input and output

linalg – linear algebra

ndimage – n-dimensional image processing

odr -orthogonal regression distance multiplexing

optimize – root structure optimization and search

signal – signal processing

sparse – sparse matrices and related procedures

spatial – spatial Data Structures and Algorithms

special – special functions

stats – statistical Distributions and Functions

weave – C / C ++ integration

The SciPy ecosystem includes general and specialized tools for data management and computation, productive experimentation, and high-performance computing. Below, we overview some key packages, though there are many more relevant packages.

Main components of ScyPy

Data and computation:

pandas, providing high-performance, easy-to-use data structures

SymPy, for symbolic mathematics and computer algebra

scikit-image is a collection of algorithms for image processing

scikit-learn is a collection of algorithms and tools for machine learning **h5py and PyTables** can both access data stored in the HDF5 format

Productivity and high-performance computing:

IPython, a rich interactive interface, letting you quickly process data and test ideas

The Jupyter notebook provides IPython functionality and more in your web browser, allowing you to document your computation in an easily reproducible form

Cython extends Python syntax so that you can conveniently build C extensions, either to speed up critical code or to integrate with C/C++ libraries

Dask, Joblib or IPyParallel for distributed processing with a focus on numeric data

Quality assurance:

nose, a framework for testing Python code, being phased out in preference for pytest numpydoc, a standard, and library for documenting Scientific Python libraries SciPy provides a very wide and sought-after feature set:

Clustering package **(scipy.cluster)**

Constants **(scipy.constants)**

Discrete Fourier transforms **(scipy.fftpack)**

Integration and ODEs **(scipy.integrate)**

Interpolation **(scipy.interpolate)**

Input and output **(scipy.io)**

Linear algebra **(scipy.linalg)**

Miscellaneous routines **(scipy.misc)**

Multi-dimensional image processing **(scipy.ndimage)**

Orthogonal distance regression **(scipy.odr)**

Optimization and Root Finding **(scipy.optimize)**

Signal processing **(scipy.signal)**

Sparse matrices **(scipy.sparse)**

Sparse linear algebra **(scipy.sparse.linalg)**

Compressed Sparse Graph Routines **(scipy.sparse.csgraph)**

Spatial algorithms and data structures **(scipy.spatial)**

Special functions **(scipy.special)**

Statistical functions **(scipy.stats)**

Statistical functions for masked arrays **(scipy.stats.mstats)**

Low-level callback functions

An example of how to calculate effectively on SciPy

In this tutorial, Basic functions — SciPy v1.4.1 Reference Guide, you can find how to calculate polynomials, their derivatives, and integrals. Yes, by one line of code SciPy

calculates derivative and integral in symbolic form. Imagine how many lines of code you would need to do this without SciPy. This is why this library is valuable in Python:

>>> p = poly1d([3,4,5])

>>> print(p) 2

3 x + 4 x + 5

>>> print(p*p)

4 3 2

9 x + 24 x + 46 x + 40 x + 25

>>> print(p.integ(k=6))

3 2

1 x + 2 x + 5 x + 6

>>> print(p.deriv()) 6 x + 4

>>> p([4, 5])

array([69, 100])

Applications:

- Multidimensional image operations
- Solving differential equations and the Fourier transform
- Optimization algorithms
- Linear algebra

iv)Numpy

In early 2005, programmer and data scientist Travis Oliphant wanted to unite the community around one project and created the NumPy library to replace the Numeric and NumArray libraries. NumPy was created based on the Numeric code. The Numeric code was rewritten to be easier to maintain, and new features could be added to the library. NumArray features have been added to NumPy. NumPy was originally part of the SciPy library. To allow other projects to use the NumPy library, its code was placed in a separate package.

The source code for NumPy is publicly available. NumPy is licensed under the BSD license.

Purpose of the NumPy library

Mathematical algorithms implemented in interpreted languages, for example, Python, often work much slower than the same algorithms implemented in compiled languages (for example, Fortran, C, and Java). The NumPy library provides implementations of computational algorithms in the form of functions and operators, optimized for working with multidimensional arrays. As a result, any algorithm that can be expressed as a sequence of operations on arrays (matrices) and implemented using NumPy works as fast as the equivalent code executed in MATLAB. If we compare *numpy* vs *math, we quickly find thatnumpy* has more advantages for computation methods compared to *math*.

Here are some of the features of Numpy:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

What’s the difference between a Python list and a NumPy array?

As described in the NumPy documentation, “NumPy gives you an enormous range of fast and efficient ways of creating arrays and manipulating numerical data inside them. While a Python list can contain different data types within a single list, all of the

elements in a NumPy array should be homogenous. The mathematical operations that are meant to be performed on arrays would be extremely inefficient if the arrays weren't homogenous." **Numpy provides the following features to the user:**

- Array objects
- Constants
- Universal functions (ufunc)
- Routine
- Packaging (numpy.distutils)
- NumPy Distutils – Users Guide
- NumPy C-API
- NumPy internals
- NumPy and SWIG

NumPy basics:

- Data types
- Array creation
- I/O with NumPy
- Indexing
- Broadcasting
- Byte-swapping
- Structured arrays
- Writing custom array containers
- Subclassing ndarray

One of the main objects of NumPy is ndarray. It allows you to create multidimensional data arrays of the same type and perform operations on them with great speed. Unlike sequences in Python, arrays in NumPy have a fixed size, the elements of the array must be of the same type. You can apply various mathematical operations to arrays, which are performed more efficiently than for Python sequences. The next example shows how to work with linear algebra with NumPy. It is really simple and easy-to-understand for Python users.

```
>>> import numpy as np

>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])

>>> print(a) [[ 1. 2.]
```

[3. 4.]]

>>> a.transpose() array([[1., 3.],

[2., 4.]])

>>> np.linalg.inv(a) array([[-2. , 1.],

[1.5, -0.5]])

>>> u = np.eye(2) # unit 2×2 matrix; “eye” represents “I”

>>> u

array([[1., 0.],

[0., 1.]])

>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j

matrix product array([[-1., 0.],

[0., -1.]])

>>> np.trace(u) # trace 2.0

>>> y = np.array([[5.], [7.]])

>>> np.linalg.solve(a, y) array([[-3.],

[4.]])

>>> np.linalg.eig(j)

(array([0.+1.j, 0.-1.j]), array([[0.70710678+0.j,0.70710678-0.j], [0.00000000-0.70710678j, 0.00000000+0.70710678j]]))

Numpy allows processing information without cycles. Please take a look at this article published by Brad solomon about the advantages of Numpy: “It is sometimes said that Python, compared to low-level languages such as C++, improves development time at the expense of runtime. Fortunately, there are a handful of ways to speed up operation runtime in Python without sacrificing ease of use. One option

suited for fast numerical operations is NumPy, which deservedly bills itself as the fundamental package for scientific computing with Python.” It makes computation in Python really fast.

Applications:

- Extensively used in data analysis
- Creates powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Replacement of MATLAB when used with scipy and matplotlib

EXPERIMENT-3:

Aim: Study of Python Libraries for ML application such as Pandas and Matplotlib

i) Pandas

Pandas is an open-source Python library that provides high-performance, easy-to-use data structure, and data analysis tools for the Python programming language.

Python with pandas is used in a wide range of fields, including academics, retail, finance, economics, statistics, analytics, and many others.

Python pandas is well suited for different kinds of data, such as:

- Ordered and unordered time series data
- Unlabeled data
- Any other form of observational or statistical data sets

Pandas is a powerful and versatile library that simplifies tasks of data manipulation in Python . Pandas is built on top of the NumPy library and is particularly well-suited for working with tabular data, such as spreadsheets or SQL tables. Its versatility and ease of use make it an essential tool for data analysts, scientists, and engineers working with structured data in Python.

Installing Pandas

The first step of working in pandas is to ensure whether it is installed in the system or not. If not then we need to install it in our system using the **pip command**. Type the cmd command in the search box and locate the folder using the cd command

where **python-pip file** has been installed. After locating it, type the command:

pip install pandas

For more reference take a look at this article on installing pandas follows.

Importing Pandas

After the pandas have been installed into the system, you need to import the library. This module is generally imported as follows:

import pandas as pd

What can you do using Pandas?

Pandas are generally used for data science but have you wondered why? This is because pandas are used in conjunction with other libraries that are used for data science. It is built on the top of the **NumPy** library which means that a lot of structures of NumPy are used or replicated in Pandas. The data produced by Pandas are often used as input for plotting functions of **Matplotlib**, statistical analysis in **SciPy**, and machine learning algorithms in **Scikit-learn**. Here is a list of things that we can do using Pandas.

- Data set cleaning, merging, and
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
- Columns can be inserted and deleted from DataFrame and higher dimensional
- Powerful group by functionality for performing split-apply-combine operations on data
- Data Visulaization

Getting Started with Pandas Installing Pandas

The first step of working in pandas is to ensure whether it is installed in the system or not. If not then we need to install it in our system using the **pip command**. Type the cmd command in the search box and locate the folder using the cd command where **python- pip file** has been installed. After locating it, type the command:

pip install pandas

For more reference take a look at this article on installing pandas follows. Importing Pandas

After the pandas have been installed into the system, you need to import the library. This module is generally imported as follows:

import pandas as pd

Here, pd is referred to as an alias to the Pandas. However, it is not necessary to import the library using the alias, it just helps in writing less amount code every time a method or property is called.

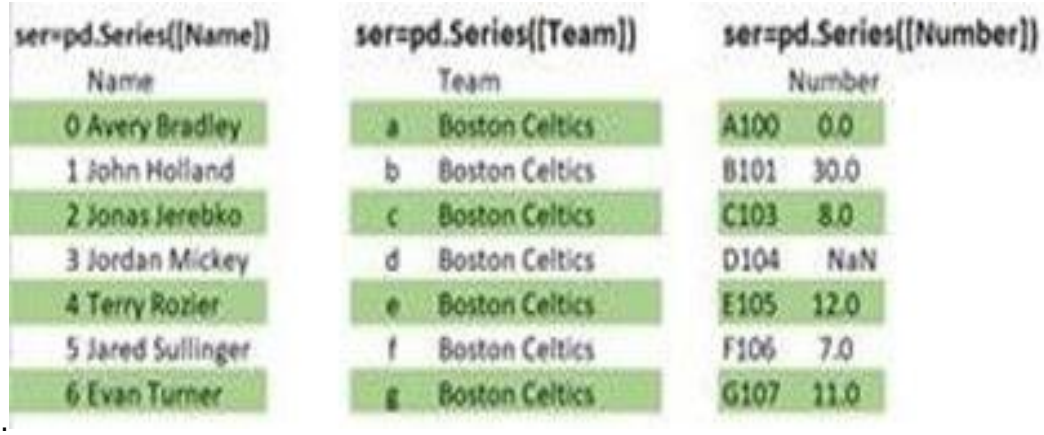
Pandas Data Structures

Pandas generally provide two data structures for manipulating data, They are:

- **Series**
- **DataFrame**

Pandas Series

A Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an Excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.



ser=pd.Series([Name])

	Name
0	Avery Bradley
1	John Holland
2	Jonas Jerebko
3	Jordan Mickey
4	Terry Rozier
5	Jared Sullinger
6	Evan Turner

ser=pd.Series([Team])

	Team
a	Boston Celtics
b	Boston Celtics
c	Boston Celtics
d	Boston Celtics
e	Boston Celtics
f	Boston Celtics
g	Boston Celtics

ser=pd.Series([Number])

	Number
A100	0.0
B101	30.0
C103	8.0
D104	NaN
E105	12.0
F106	7.0
G107	11.0

Creating a Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file. Pandas Series can be created from lists, dictionaries, and from scalar values, etc.

Pandas Series Example:

```
import pandas as pd
import numpy as np
# Creating empty series
ser = pd.Series()
print("Pandas Series: ", ser)
# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])
ser = pd.Series(data)
print("Pandas Series:\n", ser)
```

Output:
Pandas Series: Series([], dtype: float64) Pandas Series: 0 g 1 e 2 e 3 k 4 s dtype: object

DataFrame

Pandas DataFrame is a two-dimensional data structure with labeled axes (rows and columns).

Note: For more information, refer to Python | Pandas DataFrame Creating Data Frame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file. Pandas DataFrame can be created from lists, dictionaries, and from a list of dictionaries, etc.

Pandas DataFrame Example:

```
import pandas as pd
# Calling DataFrame constructor
df = pd.DataFrame()
print(df)
# list of strings
lst = ['CMR', 'CMRTC', 'SET', 'CMREC', 'ENGG', 'CMR1', 'CMR2']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

Output:

Empty DataFrame
Columns: [ ]
Index: [ ]

0
0 CMR
1 CMRTC
2 SET
3 CMREC
4 ENGG
5 CMR1
6 CMR2

How to run Pandas Program in Python?

Pandas program can be run from any text editor but it is recommended to use Jupyter Notebook for this as Jupyter gives the ability to execute code in a particular cell rather than executing the entire file. Jupyter also provides an easy way to visualize pandas data frames and plots.

ii) Matplotlib

Neuroscientist John D. Hunter began developing matplotlib in 2003, mainly inspired by the emulation of Mathworks MATLAB software teams. Matplotlib is today a whole product of the community: it is developed and supported by many people. John talked about the evolution of matplotlib at the SciPy conference in 2012. Learning matplotlib at times can be a difficult process. The problem is not the lack of documentation (which is very extensive, by the way). Difficulties may arise with the following:

- The size of the library is huge in itself, about 70,000 lines of code
- Matplotlib contains several different interfaces (ways to build a figure) and can interact with a large number of backends. (The backends are responsible for how in fact the diagrams will be displayed, not only for the internal structure)
- Despite the vastness, some of the matplotlib's own documentation is seriously outdated. The library is still evolving, and many old examples on the web may include 70% less code than in their current version.

Understanding that matplotlib roots grow from MATLAB helps explain the existence of pylab. pylab is a module inside the matplotlib library that has been built in to emulate the overall MATLAB style. It exists only for introducing a number of class functions from NumPy and matplotlib into the namespace, which simplifies the transition of MATLAB users who did not encounter the need for import statements. Former MATLAB users love its functionality, because with from pylab import * they can simply call plot() or array() directly, just like they did in MATLAB.

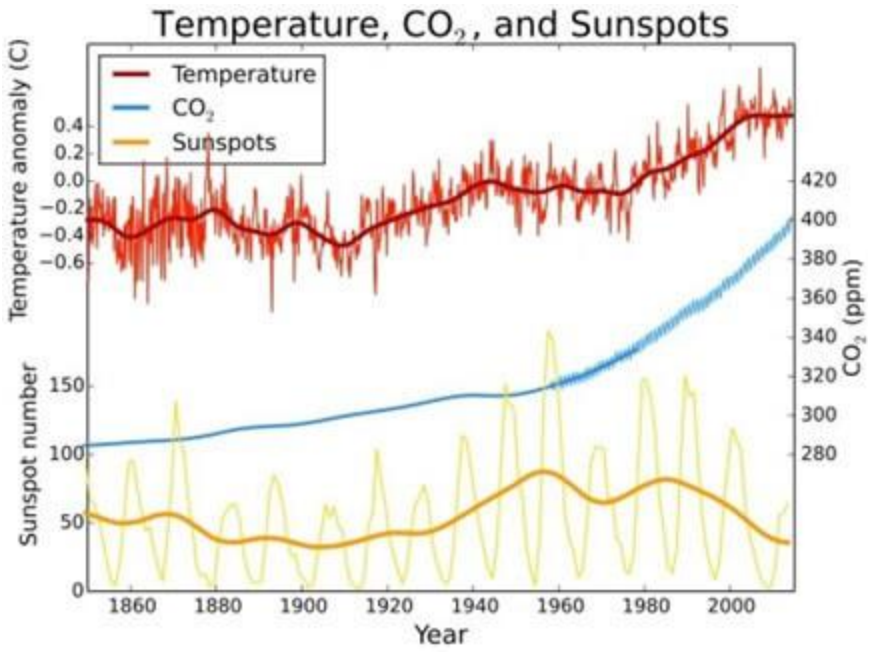
Key features of Matplotlib

One of the business cards of matplotlib is the hierarchy of its objects. If you have already worked with the matplotlib introductory manual, you may have already called something like plt.plot ([1, 2, 3]). This one line indicates that the graph is actually a hierarchy of Python objects. By "hierarchy" we mean that each chart is based on a tree-like structure of matplotlib objects.

The Figure object is the most important external container for matplotlib graphics, which can include several Axes objects. The reason for the difficulty in understanding may be

the name: Axes (axes), in fact, turn into what we mean by an individual graph or chart (rather than the plural “axis”, as you might expect).

You can think of the Figure object as a box-like container containing one or more Axes objects (real graphs). Below Axes objects, in a hierarchical order, are smaller objects such as individual lines, elevations, legends, and text boxes. Almost every “element” of a diagram is its own manipulated Python object, right up to labels and markers. An example chart on the matplot is located below.

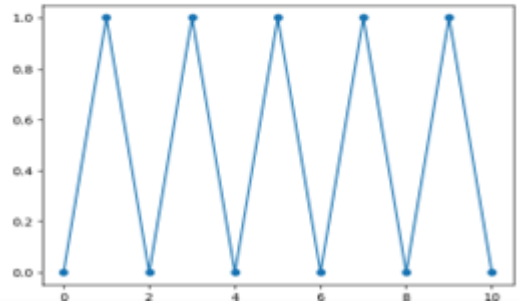


Matplotlib is a flexible, easily configurable package that, along with NumPy, SciPy, and IPython, provides features similar to MATLAB. The package currently works with several graphics libraries, including wxWindows and PyGTK.

Python code example for plotting

The Python code itself is quite simple and straightforward. Here's an example of a simple plot:

```
import matplotlib.pyplot as plt

x = [0,1,2,3,4,5,6,7,8,9,10]
y = [0,1,0,1,0,1,0,1,0,1,0]
plt.plot(x, y, marker="o")
plt.show()
```



Types of graphs and charts

The package supports many types of graphs and charts:

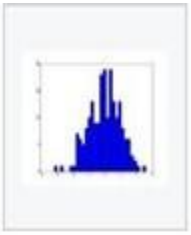
1. Charts (line plot)
2. Scatter plot
3. Bar charts and histograms
4. Pie Chart
5. Chart trunk (stem plot)
6. Contour plots
7. Gradient Fields (quiver)
8. Spectrograms

The user can specify the coordinate axis, grid, add labels and explanations, use a logarithmic scale or polar coordinates

Simple 3D graphics can be built using the mplot3d toolkit. There are other toolkits: for mapping, for working with Excel, utilities for GTK and others. With Matplotlib, you can make animated images.

Matplotlib can be technically and syntactically complex. To create a ready-made diagram, it can take half an hour to google search alone and combine all this hash to fine-tune the graph. However, understanding how matplotlib interfaces interact with each other is an investment that can pay off.

Line plot



Histogram

Scatter plot

3D plot

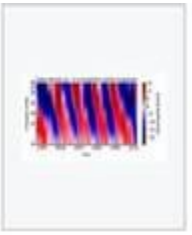
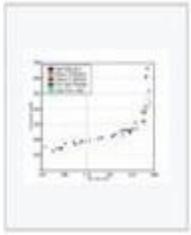


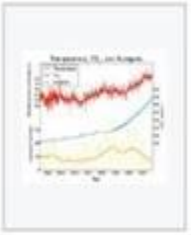
Image plot

Contour plot



Scatter plot

Polar plot



Line plot

3-D plot

Image plot

EXPERIMENT-4:

Aim: Write a Python program to implement Simple Linear Regression

Linear Regression Example:

```
# Importing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
# Data preprocessing
X = dataset.iloc[:, :-1].values  # Independent variable array
y = dataset.iloc[:, 1].values  # Dependent variable vector
# Splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
# Fitting the regression model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)  # Actually produces the linear equation
for the data
# Predicting the test set results
y_pred = regressor.predict(X_test)
y_pred
y_test
# Visualizing the results: Training set
plt.scatter(X_train, y_train, color='red')  # Plotting the observation
points
plt.plot(X_train, regressor.predict(X_train), color='blue')  # Plotting
the regression line
plt.title("Salary vs Experience (Training set)")  # Adding a title to the
graph
plt.xlabel("Years of Experience")  # Labeling x-axis
plt.ylabel("Salaries")  # Labeling y-axis
plt.show()  # Display the graph
```

```
# Visualizing the results: Testing set
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')  # Plotting
the regression line
plt.title("Salary vs Experience (Testing set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salaries")
plt.show()
```

Output:

Sample Viva Questions :

1.What is linear regression, and how does it work?

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It is called “linear” because the model assumes a linear relationship between the dependent and independent variables.

2. What are the assumptions of a linear regression model?

The assumptions of a linear regression model are:

- The relationship between the independent and dependent variables is
- The residuals, or errors, are normally distributed with a mean of zero and a constant
- The independent variables are not correlated with each other (i.e. they are not collinear).
- The residuals are independent of each other (i.e. they are not autocorrelated).
- The model includes all the relevant independent variables needed to accurately predict the dependent variable.

3. What are outliers?

A value that is significantly different from the mean or the median is considered to be an outlier in statistics. There is a possibility of erroneous results due to measurement errors. There is also the possibility of an experimental error being indicated.

4. What are the common types of errors in linear regression analysis?

There are several types of errors that can occur in linear regression analysis. Some of the most common include:

Overestimating or underestimating the relationship between the variables: This can happen if the model is too complex or if important variables are left out of the model.

Incorrect functional form: The chosen functional form (e.g. linear, log-linear, etc.) may not accurately represent the relationship between the variables.

Non-linearity of the residuals: The residuals (the difference between the observed values and the predicted values) should be randomly distributed around zero if the model is correct. If the residuals exhibit non-linear patterns, it may indicate that the chosen model is not the best fit for the data.

Multicollinearity: This occurs when two or more predictor variables in a model are highly correlated, which can cause unstable coefficient estimates and make it difficult to interpret the results.

Outliers: Outliers, or extreme values in the data, can have a significant impact on the fitted values and coefficients in a linear regression model. It is important to identify and address any potential outliers in the data before fitting a model.

5. What is the difference between biased and unbiased estimates in linear regression?

Biased estimates in linear regression refer to estimates that consistently over- or underestimate the true population parameter. This can occur due to various factors, such as missing information or incorrect assumptions about the data. Unbiased estimates, on the other hand, accurately reflect the true population parameter, without any systematic over- or underestimation.

EXPERIMENT-5:

Aim: Implementation of Multiple Linear Regression for House Price Prediction using sklearn

Real Estate Price Prediction Using Linear Regression:

```
# Importing modules and packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Importing the dataset
df = pd.read_csv('Real-estate1.csv')
df.drop('No', inplace=True, axis=1)  # Dropping unnecessary column
print(df.head())  # Display the first few rows of the dataset
print(df.columns)  # Display column names

# Plotting a scatterplot
sns.scatterplot(
    x='X4 number of convenience stores',
    y='Y house price of unit area',
    data=df
)
plt.title('Convenience Stores vs House Price')
plt.xlabel('Number of Convenience Stores')
plt.ylabel('House Price per Unit Area')
plt.show()
# Creating feature and target variables
X = df.drop('Y house price of unit area', axis=1)  # Feature variables
y = df['Y house price of unit area']  # Target variable
print(X.head())  # Display the first few rows of features
print(y.head())  # Display the first few rows of the target variable
```

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=101
)
# Creating and fitting the regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
predictions = model.predict(X_test)
# Model evaluation
print('Mean Squared Error:', mean_squared_error(y_test, predictions))
print('Mean Absolute Error:', mean_absolute_error(y_test, predictions))
```

Output:

X1 transaction date X2 house age … X6 longitude Y house price of unit area

0 2012.917 32.0 … 121.54024 37.9

1 2012.917 19.5 … 121.53951 42.2

2 2013.583 13.3 … 121.54391 47.3

3 2013.500 13.3 … 121.54391 54.8

4 2012.833 5.0 … 121.54245 43.1

[5 rows x 7 columns]

Index([‘X1 transaction date’, ‘X2 house age’,

‘X3 distance to the nearest MRT station’,

‘X4 number of convenience stores’, ‘X5 latitude’, ‘X6 longitude’,

‘Y house price of unit area’],

dtype=’object’)

X1 transaction date X2 house age … X5 latitude X6 longitude

0 2012.917 32.0 … 24.98298 121.54024

1 2012.917 19.5 … 24.98034 121.53951

2 2013.583 13.3 … 24.98746 121.54391

3 2013.500 13.3 … 24.98746 121.54391

4 2012.833 5.0 … 24.97937 121.54245

.. … … … … …

409 2013.000 13.7 … 24.94155 121.50381

410 2012.667 5.6 … 24.97433 121.54310

411 2013.250 18.8 … 24.97923 121.53986

412 2013.000 8.1 … 24.96674 121.54067

413 2013.500 6.5 … 24.97433 121.54310

[414 rows x 6 columns]

0 37.9

1 42.2

2 47.3

3 54.8

4 43.1

…

409 15.4

410 50.0

411 40.6

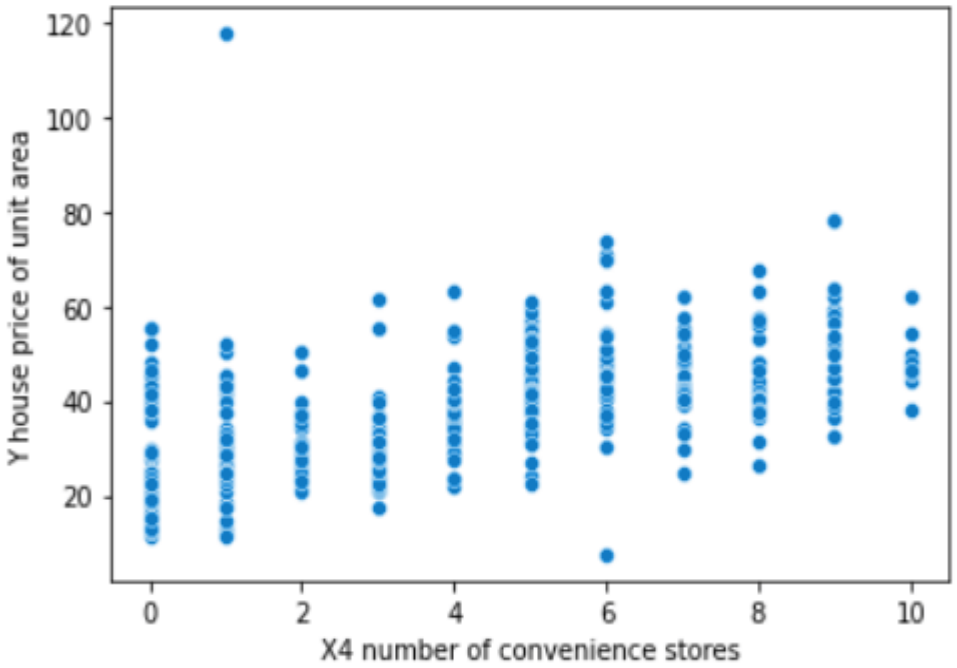
412 52.5

413 63.9

Name: Y house price of unit area, Length: 414, dtype: float64

mean_squared_error : 46.21179783493418

mean_absolute_error : 5.392293684756571



Sample Viva Questions:

1.What Is Multiple Linear Regression (MLR)?

Multiple linear regression (MLR) is used to determine a mathematical relationship among several random variables.1 In other terms, MLR examines how multiple independent variables are related to one dependent variable.

2. What are the different visualization libraries in python for multiple linear regression?

Some of the commonly used visualization libraries for Multiple Linear Regression in Python are Matplotlib, Seaborn, Plotly, and ggplot. These libraries can be used to create a range of plots (like the scatter plot) and charts, to better understand relationships between variables, detect patterns and trends, and communicate results to stakeholders.

3. What is the difference between linear and multiple regression?

A Linear regression is a statistical method used to analyze the relationship between two continuous variables. On the other hand, multiple regression is a statistical method used to analyze the relationship between one dependent variable and two or more independent variables.

4. How to use scikit-learn linear regression in Python?

Follow the steps below to use scikit-learn's linear regression in Python:

1. First, import the LinearRegression module from scikit-learn's linear_model
2. Then, create an instance of the LinearRegression object and fit your data to the model using the fit() method.
3. Once the model is trained, you can make predictions on new data using the predict()
4. Finally, you can evaluate the performance of the model using various metrics, such as R-squared, mean squared error, or mean absolute error.

EXPERIMENT-6:

Aim: Implementation of Decision tree using sklearn and its parameter tuning

Program:

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Function to import the dataset
def importdata():
    balance_data = pd.read_csv(
        'https://archive.ics.uci.edu/ml/machine-learning-databases/balance-
scale/balance-scale.data',
        sep=',', header=None
    )
    # Displaying dataset information
    print("Dataset Length: ", len(balance_data))
    print("Dataset Shape: ", balance_data.shape)
    print("Dataset: ", balance_data.head())
    return balance_data

# Function to split the dataset into features and target variables
def splitdataset(balance_data):
    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
random_state=100)
    return X, Y, X_train, X_test, y_train, y_test
```

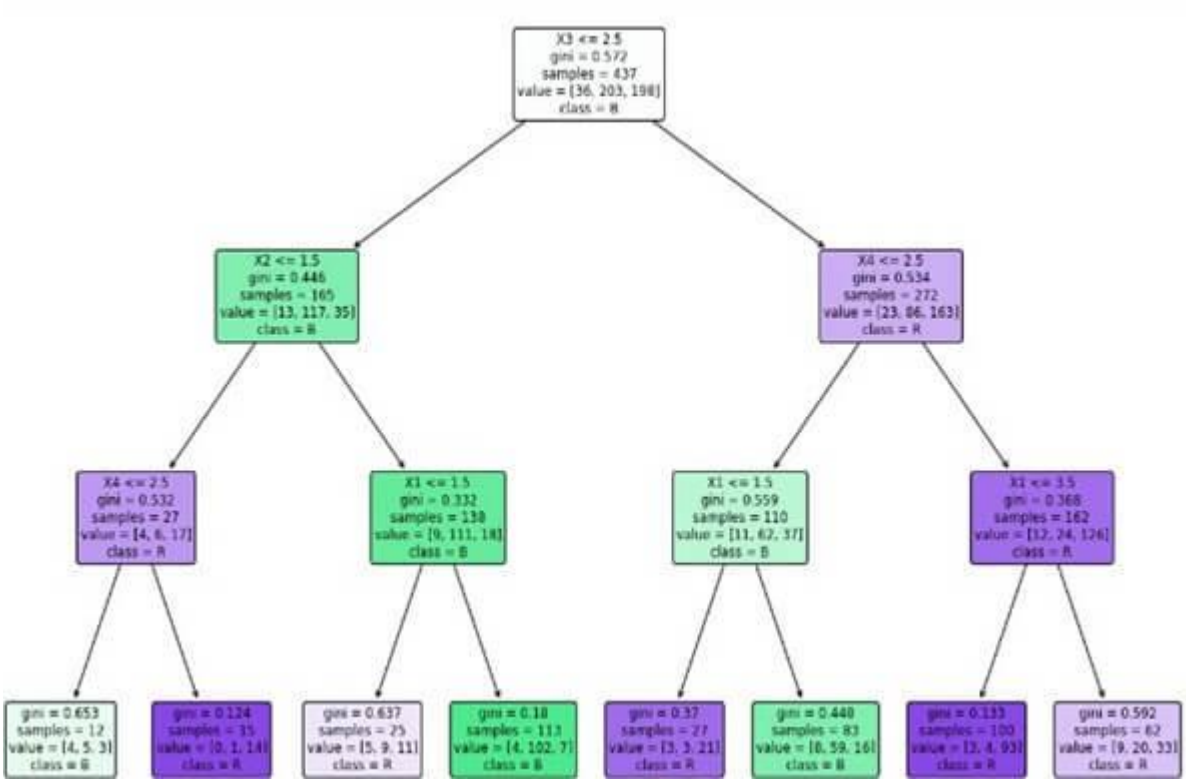
```
# Function to train the model using Gini index
def train_using_gini(X_train, X_test, y_train):
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion="gini", max_depth=3,
min_samples_leaf=5)
    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to train the model using Entropy
def train_using_entropy(X_train, X_test, y_train):
    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
        criterion="entropy", random_state=100, max_depth=3, min_samples_leaf=5
    )
    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
    print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
    print("Report : ", classification_report(y_test, y_pred))

# Function to plot the decision tree
def plot_decision_tree(clf_object, feature_names, class_names):
    plt.figure(figsize=(15, 10))
    plot_tree(clf_object, filled=True, feature_names=feature_names,
class_names=class_names, rounded=True)
```

```
    plt.show()

# Main execution
if __name__ == "__main__":
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)

    # Training using Gini Index
    clf_gini = train_using_gini(X_train, X_test, y_train)
    # Training using Entropy
    clf_entropy = train_using_entropy(X_train, X_test, y_train)

    # Visualizing the Decision Trees
    plot_decision_tree(clf_gini, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])
    plot_decision_tree(clf_entropy, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])

    # Operational Phase
    print("Results Using Gini Index:")
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
```

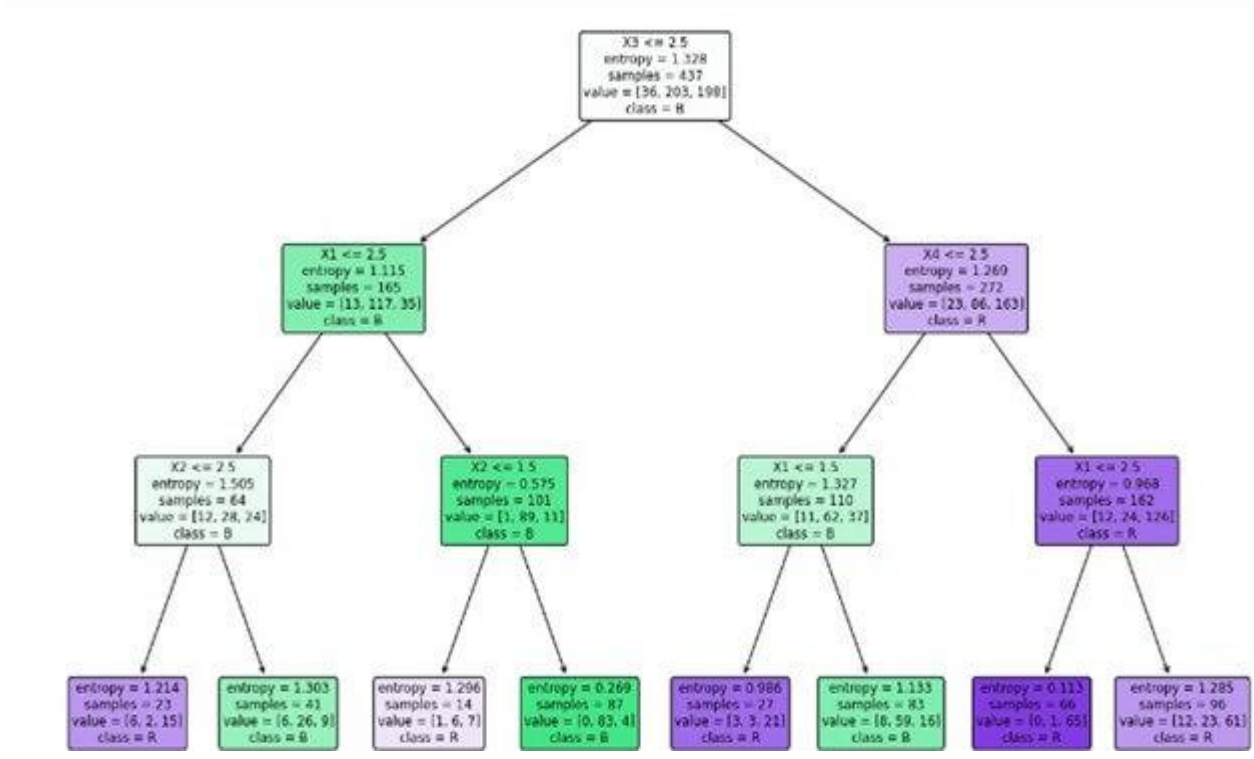
Output:

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

Using Gini Index



Using Entrop



```
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
Report :                precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```

Sample Viva Questions:

1.What are decision trees?

Decision trees are a type of machine learning algorithm that can be used for both classification and regression tasks. They work by partitioning the data into smaller and smaller subsets based on certain criteria. The final decision is made by following the path through the tree that is most likely to lead to the correct outcome.

2. How do decision trees work?

Decision trees work by recursively partitioning the data into smaller subsets. At each partition, a decision is made based on a certain criterion, such as the value of a particular feature. The data is then partitioned again based on the value of a different feature, and so on. This process continues until the data is divided into a number of subsets that are each relatively homogeneous.

3. How do you implement a decision tree in Python?

There are several libraries available for implementing decision trees in Python. One popular library is scikit-learn. To implement a decision tree in scikit-learn, you can use the DecisionTreeClassifier class. This class has several parameters that you can set, such as the criterion for splitting the data and the maximum depth of the tree.

4. How do you evaluate the performance of a decision tree?

There are several metrics that you can use to evaluate the performance of a decision tree. One common metric is accuracy, which is the proportion of predictions that the tree makes correctly. Another common metric is precision, which is the proportion of positive predictions that are actually correct. And another common metric is recall, which is the proportion of actual positives that the tree correctly identifies as positive

.

5. What are some of the challenges of using decision trees?

Some of the challenges of using decision trees include:

- Overfitting: Decision trees can be prone to overfitting, which means that they can learn the training data too well and not generalize well to new data.
- Interpretability: Decision trees can be difficult to interpret when they are very deep or complex.
- Feature selection: Decision trees can be sensitive to the choice of
- Pruning: Decision trees can be difficult to prune, which means that it can be difficult to remove irrelevant branches from the tree.

EXPERIMENT-7:

Aim: Implementation of KNN using sklearn

Program:

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

# URL for Iris dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"

# Assign column names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'Class']

# Read in the dataset
df = pd.read_csv(url, names=names)
df

# Preprocessing of Dataset
X = df.iloc[:, :-1].values
y = df.iloc[:, 4].values

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

# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
```

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

# Make Your Prediction
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Evaluating Your Algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Output:

```
[[10 0 0]
[ 0 8 0]
[ 0 0 12]]
       precision  recall  f1-score  support

Setosa      1.00    1.00   1.00     10
Versicolor    1.00    1.00   1.00     8
Virginica     1.00    1.00   1.00     12

accuracy                  1.00      30
macro avg   1.00    1.00   1.00      30
weighted avg  1.00    1.00   1.00      30
```

Sample Viva Questions:

1.Define KNN

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning,

2.Why is KNN a non-parametric Algorithm?

The term “non-parametric” refers to not making any assumptions on the underlying data distribution. These methods do not have any fixed numbers of parameters in the model. Similarly, in KNN, the model parameters grow with the training data by considering each training case as a parameter of the model. So, KNN is a non-parametric algorithm.

3.What is “K” in the KNN Algorithm?

K represents the number of nearest neighbours you want to select to predict the class of a given item, which is coming as an unseen dataset for the model.

4.Why is the odd value of “K” preferred over even values in the KNN Algorithm?

The odd value of K should be preferred over even values in order to ensure that there are no ties in the voting. If the square root of a number of data points is even, then add or subtract 1 to it to make it odd.

5. Why is it recommended not to use the KNN Algorithm for large datasets? The Problem in processing the data:

KNN works well with smaller datasets because it is a lazy learner. It needs to store all the data and then make a decision only at run time. It includes the computation of

distances for a given point with all other points. So if the dataset is large, there will be a lot of processing which may adversely impact the performance of the algorithm.

Sensitive to noise:

Another thing in the context of large datasets is that there is more likely a chance of noise in the dataset which adversely affects the performance of the KNN algorithm since the KNN algorithm is sensitive to the noise present in the dataset.

EXPERIMENT-8:

Aim: Implementation of Logistic Regression using sklearn

Program:

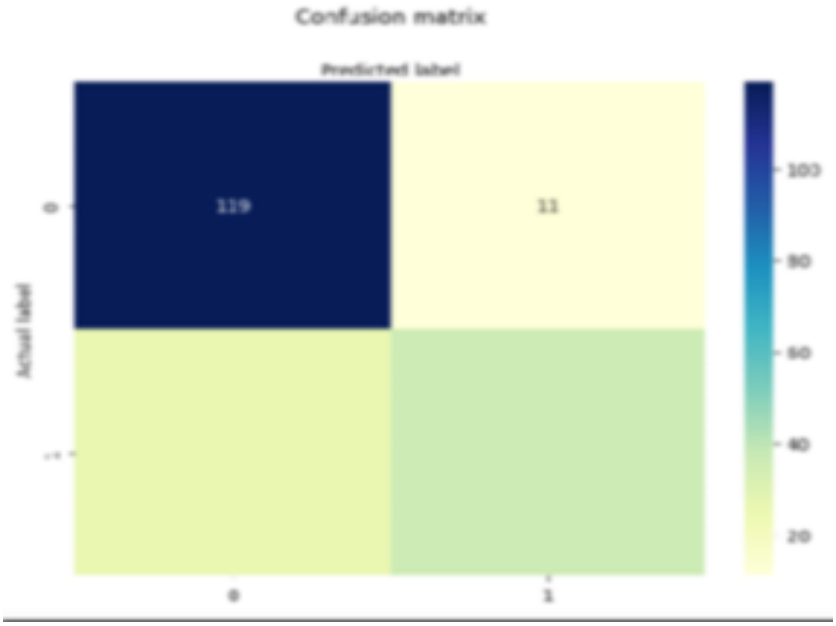
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
diab_df = pd.read_csv("diabetes.csv")
diab_df.head()

# Split dataset into features and target variable
diab_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']

X = diab_df[diab_cols]  # Features
y = diab_df.Outcome  # Target variable

# Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

# Model Development and Prediction
logreg = LogisticRegression(solver='liblinear')  # Instantiate the model
logreg.fit(X_train, y_train)  # Fit the model with data
y_pred = logreg.predict(X_test)  # Predicting y_pred

# Model Evaluation using Confusion Matrix
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
# Visualizing Confusion Matrix using Heatmap
class_names = [0, 1]  # Name of classes
fig, ax = plt.subplots()

tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# Create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1)
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

# Confusion Matrix Evaluation Metrics
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))
```

Output:
Accuracy: 0.8072916666666666

Precision: 0.7659574468085106

Recall: 0.5806451612903226



Sample Viva Questions:

1.What is classification?

Classification is a supervised machine learning problem of predicting which category or class a particular observation belongs to based on its features.Some examples of classification algorithms:

- Logistic regression
- Decision trees
- Random forest
- Artificial neural networks
- XGBoost

2. What is Logistic regression?

Logistic regression is a supervised classification model known as the logit model. It estimates the probability of something occurring, like ‘will buy’ or ‘will not buy,’ based on a dataset of independent variables. The outcome should be a categorical or a discrete value. The outcome can be either a 0 and 1, true and false, yes and no, and so on.

3.Types of logistic regression

So far, we have discussed one type of binary type of logistic regression where the outcome is a 0/1, True/False, and so on. There are two more types:

- Multinomial logistic regression: This type of regression has three or more unordered types of dependent variables, such as cats/dogs/donkeys.
- Ordinal logistic regression: Has three or more ordered dependent variables such as poor/average/ good or high/medium/average.

4. Assumptions of logistic regression

Logistic regression assumes that:

- The response variable is binary or
- The observations or independent variables have very little or no
- There are no extreme
- There is a linear relationship between the predictor variables and the log- odds of the response variable.

5. Logistic regression with Scikit-learn

To implement logistic regression with Scikit-learn, you need to understand the Scikit-learn modeling process and linear regression.

The steps for building a logistic regression include:

- Import the packages, classes, and
- Load the
- Exploratory Data Analysis(EDA).
- Transform the data if
- Fit the classification
- Evaluate the performance

EXPERIMENT-9:

Aim: Implementation of K-Means Clustering

Program:

```
import numpy as np
from sklearn.datasets import make_blobs
class KMeans:
    def __init__(self, n_clusters, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
    def fit(self, X):
        # Initialize centroids randomly
        self.centroids = X[np.random.choice(X.shape[0],
self.n_clusters, replace=False)]
        for _ in range(self.max_iters):
            # Assign each data point to the nearest centroid
            labels = self._assign_labels(X)
            # Update centroids
            new_centroids = self._update_centroids(X, labels)
            # Check for convergence
            if np.all(self.centroids == new_centroids):
                break
            self.centroids = new_centroids
    def _assign_labels(self, X):
        # Compute distances from each data point to centroids
        distances = np.linalg.norm(X[:, np.newaxis] -
self.centroids, axis=2)
        # Assign labels based on the nearest centroid
        return np.argmin(distances, axis=1)

    def _update_centroids(self, X, labels):
        new_centroids = np.array([X[labels == i].mean(axis=0) for i
in range(self.n_clusters)])
```

```
        return new_centroids

# Generate synthetic data using sklearn.datasets.make_blobs
X, _ = make_blobs(n_samples=300, centers=3, random_state=42)

# Create a K-Means instance with 3 clusters
kmeans = KMeans(n_clusters=3)
# Fit the model to the data
kmeans.fit(X)
# Get cluster assignments for each data point
labels = kmeans._assign_labels(X)

print("Cluster Assignments:", labels)
print("Final Centroids:", kmeans.centroids)
```

Output:

Cluster Assignments:

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

Final Centroids:

[[-7.36858243 -7.40836171]] # Centroid for Cluster 0
[[-6.05855368 -6.26139533]] # Centroid for Cluster 1
[[1.05693535 5.52708203]] # Centroid for Cluster 2

Sample Viva Questions:

1.What is K-Means clustering?

K-Means clustering is the most popular unsupervised machine learning algorithm. K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. In this kernel, I implement K-Means clustering to find intrinsic groups within the dataset that display the same status_type behaviour.

2. Understanding the K-Means Algorithm

The K-Means algorithm is relatively simple yet effective. Here are the basic steps:

- Choose the number of clusters, k, that you want to
- Initialize k cluster centroids
- Assign each data point to the nearest centroid, creating k
- Recalculate the centroids as the mean of all data points in each
- Repeat steps 3 and 4 until convergence (centroids no longer change significantly) or for a specified number of iterations.

3. Write the Applications of clustering ?

K-Means clustering is the most common unsupervised machine learning algorithm. It is widely used for many applications which include-

- Image segmentation
- Customer segmentation
- Species clustering
- Anomaly detection
- Clustering languages

4. K-Means Clustering intuition

K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. It is based on centroid-based clustering.

Centroid – A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. It is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. K-Means clustering works as follows: – The K-Means clustering algorithm uses an iterative procedure to deliver a final result. The algorithm requires number of

clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids. The algorithm then iterates between two steps: –

Data assignment step: Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if ci is the collection of centroids in set C, then each data point is assigned to a cluster based on minimum Euclidean distance. **Centroid update step :**In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid's cluster.

5. What is the objective of K-Means clustering?

K-means clustering begins with the description of a cost function over a parameterized set of possible clustering, and the objective of the clustering algorithm is to find a minimum cost partitioning (clustering). The clustering function is turned into an optimization problem under this model.