UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION &TECHNOLOGY

IT-761 Data Analytics Lab

Practical File

Submitted to	Submitted By
Prof. Anjana Gusain (Professor)	Mehak Sharma Enrolment No: 05416404521 MCA(SE) 3rd Semester

Content

Practical 1

Introduction to Data Analytics using Python

- **Six Steps of Data Analysis Process**
- **Different Sources of Data for Data Analysis**

Data Analysis is the technique to collect, transform, and organize data to make future predictions, and make informed data-driven decisions. It also helps to find possible solutions for a business problem. There are six steps for Data Analysis. They are:

1. Ask or Specify Data Requirements
2. Prepare or Collect Data
3. Clean and Process
4. Analyze
5. Share
6. Act or Report

Each step has its own process and tools to make overall conclusions based on the data.

1. Ask

The first step in the process is to Ask. The data analyst is given a problem/business task. The analyst has to understand the task and the stakeholder's expectations for the solution. Questions to ask yourself for the Ask phase are:

What are the problems that are being mentioned by my stakeholders?

What are their expectations for the solutions?

2. Prepare

The second step is to Prepare or Collect the Data. This step includes collecting data and storing it for further analysis. The analyst has to collect the data based on the task given from multiple sources. The data has to be collected from various sources, internal or external sources. Internal data is the data available in the organization that you work for while external data is the data available in sources other than your organization. The data that is collected by an individual from their own resources is called first-party data. The data that is collected and sold is called second-party data. Data that is collected from outside sources is called third-party data.

3. Clean and Process Data

The third step is Process. After the data is collected from multiple sources, it is time to clean the data. Clean data means data that is free from misspellings, redundancies, and irrelevance. Clean data largely depends on data integrity. There might be duplicate data or the data might not be in a format, therefore the unnecessary data is removed and cleaned. this is one of the most important steps in Data Analysis as clean and formatted data helps in finding trends and solutions. The most important part of the Process phase is to check whether your data is biased or not. Bias is an act of favoring a particular group/community while ignoring the rest.

4. **Analyse**

The fourth step is to Analyse. The cleaned data is used for analysing and identifying trends. It also performs calculations and combines data for better results. The most widely used programming languages for data analysis are R and Python.

5. **Share**

The fifth step is Share. Nothing is more compelling than a visualization. The data now transformed has to be made into a visual(chart, graph). The reason for making data visualizations is that there might be people, mostly stakeholders that are non-technical. Visualizations are made for a simple understanding of complex data. R and Python have some packages that provide beautiful data visualizations.

6. **Act or Report**

The final/sixth step is Act. After a presentation is given based on your findings, the stakeholders discuss whether to move forward or not. If they agreed to your recommendations, they move further with your solutions. If they don't agree with your findings, you will have to dig deeper to find more possible solutions. Every step has to be re-organized. We have to repeat every step to see whether there are any gaps in there.

Different sources of data for data analysis

Data can be gathered from two places: internal and external sources. The information collected from internal sources is called "primary data," while the information gathered from outside references is called "secondary data."

Data analysis must be collected through primary or secondary research. A data source is a pool of statistical facts and non-statistical facts that a researcher or analyst can use to do more work on their research.

There are mostly two kinds of data sources:

- Statistical Data sources
  Statistical data sources are surveys and other statistical reports used for official purposes.

- Census Data Sources
  According to this method, the data are taken from the census report that was published earlier. It's the opposite of statistical surveys. The Census method closely examines all parts of the population during the research process.

Researchers use both data sources a lot in their work. The data is collected from these using either primary or secondary research methods.

Additional sources of data

1. Internal sources of data
   These types of data can easily be found within the organization such as market record, a sales record, transactions, customer data, accounting resources, etc.
2. External sources of data
   The data which can't be found at internal organizations and can be gained through external third party resources is external source data. The cost and time consumption is more because this contains a huge amount of data.Examples of external sources are Government publications, news publications, Registrar General of India etc.

Practical 2

Introduction to Python Libraries: NumPy, Pandas, SciPy, Scikit-learn, Matplotlib, Seaborn.

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Pandas is an open-source Python package that provides high-performance, easy-to-use data structures and data analysis tools for the labelled data in Python programming language. Pandas stand for Python Data Analysis Library. Pandas take data in a CSV or TSV file or a SQL database and create a Python object with rows and columns called a data frame.

The **SciPy** library is one of the core packages that make up the SciPy stack. SciPy library contains modules for efficient mathematical routines as linear algebra, interpolation, optimization, integration, and statistics. The main functionality of the SciPy library is built upon NumPy and its arrays. SciPy uses arrays as its basic data structure.

Scikit Learn is a robust machine learning library for Python. It features ML algorithms like SVMs, random forests, k-means clustering, spectral clustering, mean shift, cross-validation and more. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. Scikit-learn has Supervised learning models like Naive Bayes and use for grouping unlabelled data such as KMeans.

Matplotlib is the plotting library for Python that provides an object-oriented API for embedding plots into applications. It is a close resemblance to MATLAB embedded in Python programming language. Histogram, bar plots, scatter plots, area plot to pie plot, Matplotlib can depict a wide range of visualizations.

Seaborn is defined as the data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. Matplotlib is used for basic plotting; bars, pies, lines, scatter plots and stuff whereas, seaborn provides a variety of visualization patterns with less complex and fewer syntax.

Practical 3

Introduction to Python Programming

- **Datatypes**
- **Operators**
- **Loops**
- **Central Tendency Measures**
- **MATRIX OPERATIONS:**
  Write a Python program to do the following operations:
  o Library: NumPy
  o a) Create multi-dimensional arrays and find its shape and dimension
  o b) Create a matrix full of zeros and ones
  o c) Reshape and flatten data in the array
  o d) Append data vertically and horizontally
  o e) Apply indexing and slicing on array
  o f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation
- **LINEAR ALGEBRA ON MATRICES**
  Write a Python program to do the following operations:
  o Library: NumPy
  o a) Dot and matrix product of two arrays
  o b) Compute the Eigen values of a matrix
  o c) Solve a linear matrix equation such as 3 * x 0 + x 1 = 9, x 0 + 2 * x 1 = 8
  o d) Compute the multiplicative inverse of a matrix
  o e) Compute the rank of a matrix
  o f) Compute the determinant of an array

There are different types of data types in Python. Some built-in Python data types are:

- **Numeric data types**: *int, float, complex*
- **String data types**: *str*
- **Sequence types**: *list, tuple, range*
- **Binary types**: *bytes, bytearray, memoryview*
- **Mapping data type**: *dict*
- **Boolean type**: *bool*
- **Set data types**: *set, frozenset*

Python divides the **operators** in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python has following types of **loops**

1. while loop

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

2. for loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

3. Nested loops

We can use one or more loop inside any another while, for or do..while loop.

Central Tendency

A measure of central tendency (also referred to as measures of centre or central location) is a summary measure that attempts to describe a whole set of data with a single value that represents the middle or centre of its distribution.

There are three main measures of central tendency: the mode, the median and the mean. Each of these measures describes a different indication of the typical or central value in the distribution.

The mode is the most commonly occurring value in a distribution.

The median is the middle value in distribution when the values are arranged in ascending or descending order.

The mean is the sum of the value of each observation in a dataset divided by the number of observations. This is also known as the arithmetic average.

```python
import numpy as np
arr = np.array([[ 1, -2, 3],
                 [ 4, 2, 9],
                 [0, 7, 8]])
print("Shape of the array : ", arr.shape)
print("Dimension of the array : ", arr.ndim)

arr1 = np.zeros((2,2))
print(arr1)

arr2 = np.ones((2,2))
print(arr2)

print('\n')
reshaped = arr.reshape(3,3,1)
print("Reshaped array \n",reshaped);

flattenarr=arr.flatten();
print ("Flattened arrray",flattenarr)

arr3 = np.array([1,2])
#temp=np.append(arr1, arr3, axis=1)

arr3=np.vstack((arr1,arr3))
print("Appending row to an array : \n", arr3)

arr5 = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

sliced = arr5[:2, ::2]

print("Sliced : \n",sliced)
print ("Row-wise maximum elements:",arr5.max(axis = 1))
print ("Column-wise minimum elements:",arr5.min(axis = 0))
r1 = np.mean(arr)
print("\nMean: ", r1)

r2 = np.std(arr)
print("\nstd: ", r2)

print(np.median(arr, axis=0))
```

```python

import numpy as np
mat1 = np.array([[4, 6],
                 [6, 4]])

mat2 = np.array([[ 8,  5],
                 [20, 13]])

print("Dot Product : ",mat1.dot(mat2))

print("Element-wise Multiplication : ", mat1*mat2)

w, v = np.linalg.eig(mat1)
print("Eigen values : ", w)

#3*x0 + x1 = 9, x0 +2*x1 = 8


a = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
x = np.linalg.solve(a, b)
print("Solution : ",x)

print("Multiplicative Inverse : ", np.linalg.inv(a))

print("Determinant : ", np.linalg.det(a))

print("Ran of Matrix : ", np.linalg.matrix_rank(a))
```

OUTPUT

```
In [25]: runfile('C:/Users/Dell/Desktop/PYTHON/q1.py', wdir='C:/Users/Dell/Desktop/PYTHON')
Shape of the array :  (3, 3)
Dimension of the array :  2
[[0. 0.]
 [0. 0.]]
[[1. 1.]
 [1. 1.]]


Reshaped array
 [[[ 1]
  [-2]
  [ 3]]

 [[ 4]
  [ 2]
  [ 9]]

 [[ 0]
  [ 7]
  [ 8]]]
Flattened arrray [ 1 -2  3  4  2  9  0  7  8]
Appending row to an array :
 [[0. 0.]
 [0. 0.]
 [1. 2.]]
Sliced :
 [[-1.  0.]
 [ 4.  6.]]
Row-wise maximum elements: [4. 6. 8. 4.]
Column-wise minimum elements: [-1. -7.  0.  0.]

Mean:  3.5555555555555554

std:  3.5624932315291993
[1. 2. 8.]

In [26]:
```

```
In [27]: runfile('C:/Users/Dell/Desktop/PYTHON/q2.py', wdir='C:/Users/Dell/Desktop/PYTHON')
Dot Product :  [[152  98]
 [128  82]]
Element-wise Multiplication :  [[ 32  30]
 [120  52]]
Eigen values :  [10. -2.]
Solution :  [2. 3.]
Multiplicative Inverse :  [[ 0.4 -0.2]
 [-0.2  0.6]]
Determinant :  5.000000000000001
Ran of Matrix :  2

In [28]:
```

Practical 4

UNDERSTANDING DATA

Write a Python program to do the following operations:

Data set: brain_size.csv

Library: Pandas

a) Loading data from CSV file

b) Compute the basic statistics of given data - shape, no. of columns, mean

c) Splitting a data frame on values of categorical variables

d) Visualize data using Scatter plot

Pandas groupby is used for grouping the data according to the categories and apply a function to the categories. It also helps to aggregate data efficiently.

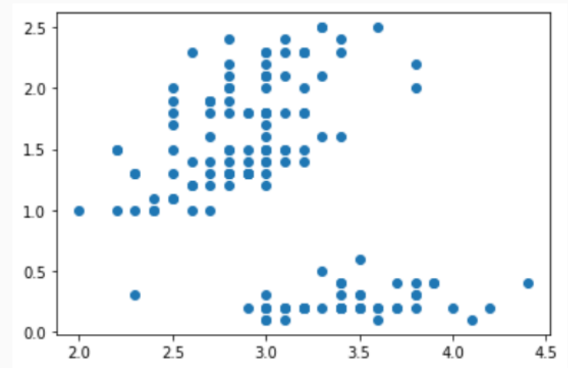
Pandas dataframe.groupby() function is used to split the data into groups based on some criteria. pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

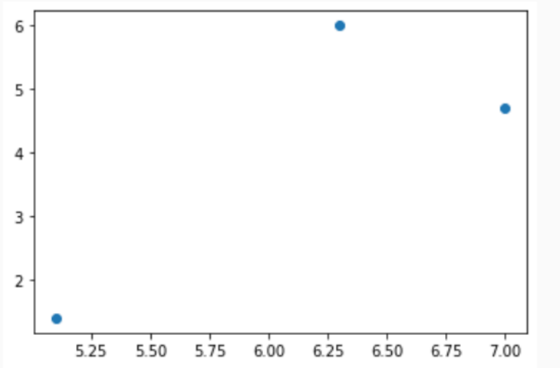
The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

```python

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

# Reading the CSV file
df = pd.read_csv("Iris.csv")

# Printing top 5 rows
print(df.head())

print("Shape = ",np.shape(df))

a = np.ndim(df)
print("dimensions = ",a)


print("mean = ", statistics.mean(df['sepal.length']))

gk = df.groupby('variety')

plt.scatter(df['sepal.width'], df['petal.width'])
plt.show()

print(gk.first())
gkk = gk.first()
plt.scatter(gkk['sepal.length'], gkk['petal.length'])
```

OUTPUT

```
In [23]: runfile('C:/Users/Dell/Desktop/PYTHON/q4_data_analytics.py', wdir='C:/Users/Dell/Desktop/PYTHON')
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
Shape =  (150, 5)
dimensions =  2
mean =  5.843333333333334
            sepal.length  sepal.width  petal.length  petal.width
variety
Setosa               5.1          3.5           1.4          0.2
Versicolor           7.0          3.2           4.7          1.4
Virginica            6.3          3.3           6.0          2.5
```





Practical 5

CORRELATION MATRIX

Write a python program to load the dataset and understand the input data

Dataset : Pima Indians Diabetes Dataset

Library : Scipy

a) Load data, describe the given data and identify missing, outlier data items

b) Find correlation among all attributes

c) Visualize correlation matrix

Outliers: Outliers are those data points that are significantly different from the rest of the dataset. They are often abnormal observations that skew the data distribution, and arise due to inconsistent data entry, or erroneous observations.

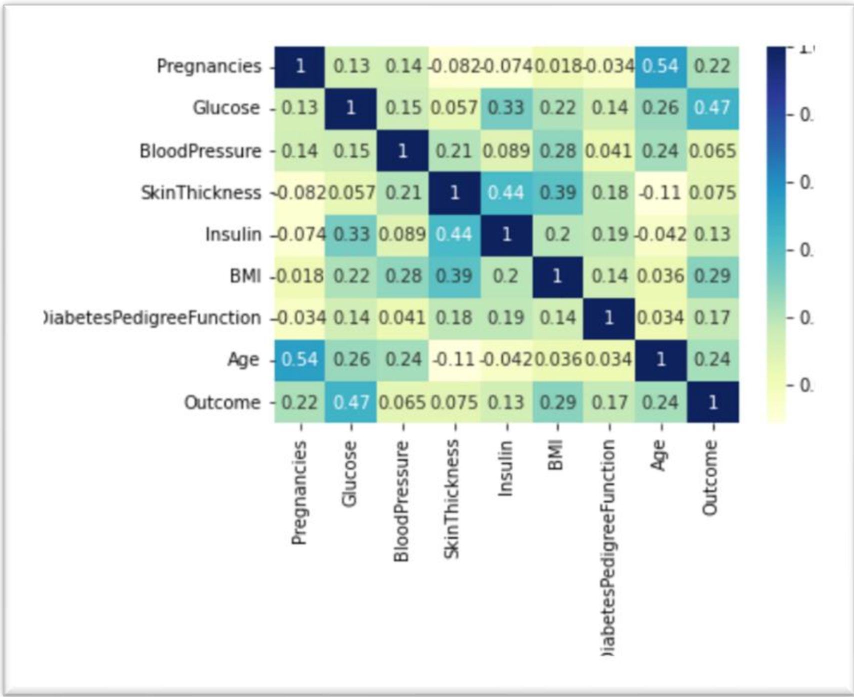
A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

```
import pandas as pd
import numpy as np
import statistics as st
import matplotlib.pyplot as mp
from scipy import stats
import seaborn as sb

# Reading the CSV file
df = pd.read_csv(r"C:\Users\DTL-317\Desktop\Mehak\Pima.csv")

# Printing top 5 rows
print(df[:10])
print(df.describe())
print(df.isnull().sum())

#plt.scatter(df['Pregnancies'], df['Glucose'])

mean = np.mean(df)
sd = np.std(df)

#print(mean["Pregnancies"])
#ul = np.array(mean+3*sd)
#ll = np.array(mean+3*sd)
#print(ul)

z = np.abs(stats.zscore(df['Glucose']))
idx_outliers = np.where(z>1, True, False)
ans = pd.Series(idx_outliers, index = df['Glucose'].index)
print(ans)
print(np.sum(ans))

print(df.corr(method='pearson'))

# plotting correlation heatmap
dataplot = sb.heatmap(df.corr(), cmap="YlGnBu", annot=True)

# displaying heatmap
```

OUTPUT

```
In [10]: runfile('C:/Users/Dell/Desktop/PYTHON/q5.py', wdir='C:/Users/Dell/Desktop/PYTHON')
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1
5            5      116             74  ...                     0.201   30        0
6            3       78             50  ...                     0.248   26        1
7           10      115              0  ...                     0.134   29        0
8            2      197             70  ...                     0.158   53        1
9            8      125             96  ...                     0.232   54        1

[10 rows x 9 columns]
       Pregnancies     Glucose  ...         Age     Outcome
count   768.000000  768.000000  ...  768.000000  768.000000
mean      3.845052  120.894531  ...   33.240885    0.348958
std       3.369578   31.972618  ...   11.760232    0.476951
min       0.000000    0.000000  ...   21.000000    0.000000
25%       1.000000   99.000000  ...   24.000000    0.000000
50%       3.000000  117.000000  ...   29.000000    0.000000
75%       6.000000  140.250000  ...   41.000000    1.000000
max      17.000000  199.000000  ...   81.000000    1.000000

[8 rows x 9 columns]
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
0      False
1       True
2       True
3      False
4      False
       ...
763    False
764    False
765    False
766    False
```

```
766    False
767    False
Length: 768, dtype: bool
228
                          Pregnancies   Glucose  ...       Age   Outcome
Pregnancies                  1.000000  0.129459  ...  0.544341  0.221898
Glucose                      0.129459  1.000000  ...  0.263514  0.466581
BloodPressure                0.141282  0.152590  ...  0.239528  0.065068
SkinThickness               -0.081672  0.057328  ... -0.113970  0.074752
Insulin                     -0.073535  0.331357  ... -0.042163  0.130548
BMI                          0.017683  0.221071  ...  0.036242  0.292695
DiabetesPedigreeFunction    -0.033523  0.137337  ...  0.033561  0.173844
Age                          0.544341  0.263514  ...  1.000000  0.238356
Outcome                      0.221898  0.466581  ...  0.238356  1.000000

[9 rows x 9 columns]
C:\Program Files\Spyder\pkgs\numpy\core\fromnumeric.py:3472: FutureWarning: In a future version,
DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use
'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```



Practical 6

DATA PREPROCESSING – HANDLING MISSING VALUES

Write a python program to impute missing values with various techniques on given dataset.

a) Remove rows/ attributes

b) Replace with mean or mode

c) Write a python program to perform transformation of data using Discretization (Binning) and N (MinMaxScaler or MaxAbsScaler) on given dataset.

Data pre-processing is a data mining technique which is used to transform the raw data in a useful and efficient format. It describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves the following steps:

1. <u>Data Cleaning</u>

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

(a). Missing Data:
This situation arises when some data is missing in the data. It can be handled in various ways.

- Ignore the tuples
- Fill the missing values

(b). Noisy Data:
Noisy data is a meaningless data that can't be interpreted by machines.It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways :

- Binning Method

2. Data Integrity

Data integrity is the overall accuracy, completeness, and consistency of data.

3. Data Reduction

Since data mining is a technique that is used to handle huge amount of data. While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we uses data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

4. Data Transformation

This step is taken in order to transform the data in appropriate forms suitable for mining process.

Normalization:

In preprocessing, standardization of data is one of the transformation task. Standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively. The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.

Discretization:

Data discretization refers to a method of converting a huge number of data values into smaller ones so that the evaluation and management of data become easy. In other words, data discretization is a method of converting attributes values of continuous data into a finite set of intervals with minimum data loss. Binning refers to a data smoothing technique that helps to group a huge number of continuous values into smaller values.

```python

import pandas as pd
import numpy as np
import statistics as st
from scipy import stats
#import seaborn as sb

df = pd.read_csv(r"employees.csv")
print(df.head())
#Remove rows with missing values
print(df.isnull().sum())
#df=df.dropna()
print(df.isnull().sum())

print(df['Team'].isnull())

#Replace with mean or mode
mode = df['Team'].mode().values[0]
df['Team']=df['Team'].replace(np.nan, mode)
print(df.isnull().sum())

print(df['Team'].isnull())
print(df.head())


#6c
min_value = df['Bonus %'].min()
max_value = df['Bonus %'].max()

print(min_value)
print(max_value)

bins = np.linspace(min_value,max_value,4)
print(bins)

labels = ['small', 'medium', 'big']

df['bins'] = pd.cut(df['Bonus %'], bins=bins, labels=labels, include_lowest=True)
print(df['Bonus %'].head())
```

```python

from sklearn.preprocessing import MinMaxScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
MinMaxScaler()
print("data:\n",scaler.data_max_)
print("Transformed data:\n",scaler.transform(data))
```

OUTPUT

```
In [10]: runfile('C:/Users/Dell/Desktop/PYTHON/q5.py', wdir='C:/Users/Dell/Desktop/PYTHON')
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1
5            5      116             74  ...                     0.201   30        0
6            3       78             50  ...                     0.248   26        1
7           10      115              0  ...                     0.134   29        0
8            2      197             70  ...                     0.158   53        1
9            8      125             96  ...                     0.232   54        1

[10 rows x 9 columns]
       Pregnancies     Glucose  ...         Age     Outcome
count   768.000000  768.000000  ...  768.000000  768.000000
mean      3.845052  120.894531  ...   33.240885    0.348958
std       3.369578   31.972618  ...   11.760232    0.476951
min       0.000000    0.000000  ...   21.000000    0.000000
25%       1.000000   99.000000  ...   24.000000    0.000000
50%       3.000000  117.000000  ...   29.000000    0.000000
75%       6.000000  140.250000  ...   41.000000    1.000000
max      17.000000  199.000000  ...   81.000000    1.000000

[8 rows x 9 columns]
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
0      False
1       True
2       True
3      False
4      False
       ...
763    False
764    False
765    False
766    False
```

```
In [11]: runfile('C:/Users/Dell/Desktop/PYTHON/q6.py', wdir='C:/Users/Dell/Desktop/PYTHON')
  First Name  Gender Start Date  ... Bonus %  Senior Management            Team
0    Douglas    Male   8/6/1993  ...   6.945               True       Marketing
1     Thomas    Male  3/31/1996  ...   4.170               True             NaN
2      Maria  Female  4/23/1993  ...  11.858              False         Finance
3      Jerry    Male   3/4/2005  ...   9.340               True         Finance
4      Larry    Male  1/24/1998  ...   1.389               True  Client Services

[5 rows x 8 columns]
First Name            67
Gender               145
Start Date             0
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management     67
Team                  43
dtype: int64
First Name            67
Gender               145
Start Date             0
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management     67

Team                  43
dtype: int64
0      False
1       True
2      False
3      False
4      False
       ...
995    False
996    False
997    False
998    False
999    False
Name: Team, Length: 1000, dtype: bool
First Name            67
Gender               145
Start Date             0
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management     67
Team                   0
dtype: int64
0      False
1      False
2      False
3      False
4      False
       ...
995    False
996    False
997    False
998    False
999    False
Name: Team, Length: 1000, dtype: bool
  First Name  Gender Start Date  ... Bonus %  Senior Management            Team
0    Douglas    Male   8/6/1993  ...   6.945               True       Marketing
1     Thomas    Male  3/31/1996  ...   4.170               True  Client Services
2      Maria  Female  4/23/1993  ...  11.858              False         Finance
3      Jerry    Male   3/4/2005  ...   9.340               True         Finance
4      Larry    Male  1/24/1998  ...   1.389               True  Client Services
```

```
[5 rows x 8 columns]
1.015
19.944
[ 1.015       7.32466667 13.63433333 19.944     ]
0      6.945
1      4.170
2     11.858
3      9.340
4      1.389
Name: Bonus %, dtype: float64

In [12]:
```

```
In [124]: runfile('C:/Users/Dell/Desktop/PYTHON/q6mam_b.py', wdir='C:/Users/Dell/Desktop/PYTHON')
MinMaxScaler()
data:
 [ 1. 18.]
Transformed data:
 [[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
```

Practical 7

Regression: Linear Regression, Logistic Regression

Linear regression

It is a common method to model the relationship between a dependent variable and one or more independent variables. Linear models are developed using the parameters which are estimated from the data. Linear regression is useful in prediction and forecasting where a predictive model is fit to an observed data set of values to determine the response. Linear regression models are often fitted using the least-squares approach where the goal is to minimize the error.

Slope = Sxy/Sxx, where Sxy and Sxx are sample covariance and sample variance respectively.

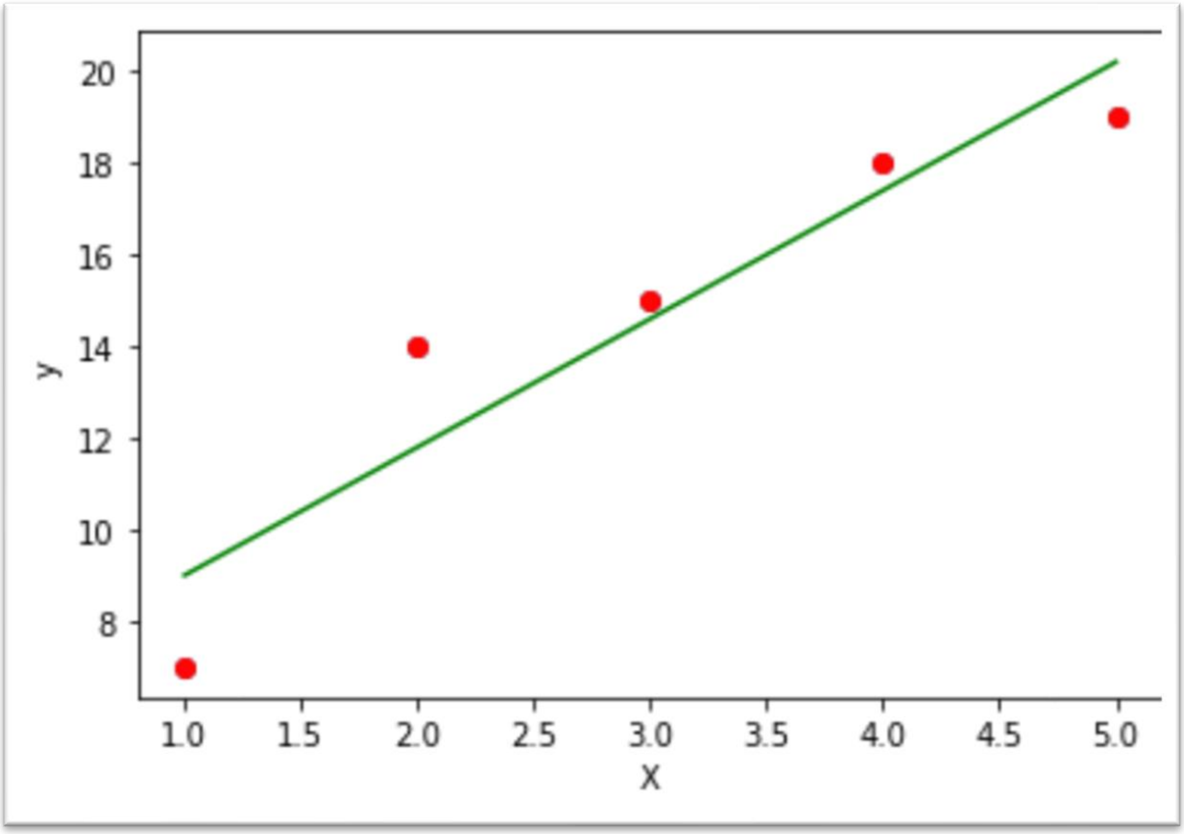
Intercept = ymean – slope* xmean

```python
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm

#initialise
x = np.array([1,2,3,4,5])
y = np.array([7,14,15,18,19])
n = np.size(x)

#calculate means
x_mean = np.mean(x)
y_mean = np.mean(y)
x_mean,y_mean

#calculate the coefficients of the regression line equation
Sxy = np.sum(x*y)- n*x_mean*y_mean
Sxx = np.sum(x*x)-n*x_mean*x_mean

b1 = Sxy/Sxx
b0 = y_mean-b1*x_mean
print('slope b1 is', b1)
print('intercept b0 is', b0)

#Plot the given data set
plt.scatter(x,y)
plt.xlabel('Independent variable X')
plt.ylabel('Dependent variable y')

#plot the regression line
y_pred = b1 * x + b0
plt.scatter(x, y, color = 'red')
plt.plot(x, y_pred, color = 'green')
plt.xlabel('X')
plt.ylabel('y')
```

```python

#check out the root mean squared errors
error = y - y_pred
se = np.sum(error**2)
print('squared error is', se)

mse = se/n
print('mean squared error is', mse)

rmse = np.sqrt(mse)
print('root mean square error is', rmse)

SSt = np.sum((y - y_mean)**2)
R2 = 1- (se/SSt)
print('R square is', R2)
```

OUTPUT

```
In [48]: runfile('C:/Users/Dell/Desktop/PYTHON/linearRegression.py', wdir='C:/Users/Dell/Desktop/PYTHON')
slope b1 is 2.8
intercept b0 is 6.200000000000001
squared error is 10.800000000000004
mean squared error is 2.160000000000001
root mean square error is 1.4696938456699071
R square is 0.8789237668161435

In [49]:
```



Logistic Regression

Logistic Regression is a supervised learning algorithm that is used when the target variable is categorical. Hypothetical function h(x) of linear regression predicts unbounded values. But in the case of Logistic Regression, where the target variable is categorical we have to strict the range of predicted values. Consider a classification problem, where we need to classify whether an email is a spam or not. So, the hypothetical function of linear regression could not be used here to predict as it predicts unbound values, but we have to predict either 0 or 1.

To do, so we apply the sigmoid activation function on the hypothetical function of linear regression. So the resultant hypothetical function for logistic regression is given below :

```
h( x ) = sigmoid( wx + b )


Here, w is the weight vector.
```

x is the feature vector.

b is the bias.

sigmoid(z) = 1 / (1 + e$^{(- z)}$)

```
# Importing libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings( "ignore" )

# to compare our model's accuracy with sklearn model
from sklearn.linear_model import LogisticRegression
# Logistic Regression
class LogitRegression() :
    def __init__( self, learning_rate, iterations ) :
        self.learning_rate = learning_rate
        self.iterations = iterations

    # Function for model training
    def fit( self, X, Y ) :
        # no_of_training_examples, no_of_features
        self.m, self.n = X.shape
        # weight initialization
        self.W = np.zeros( self.n )
        self.b = 0
        self.X = X
        self.Y = Y

        # gradient descent learning

        for i in range( self.iterations ) :
            self.update_weights()
        return self

    # Helper function to update weights in gradient descent

    def update_weights( self ) :
        A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )

        # calculate gradients
        tmp = ( A - self.Y.T )
        tmp = np.reshape( tmp, self.m )
        dW = np.dot( self.X.T, tmp ) / self.m
```

```python
        db = np.sum( tmp ) / self.m

        # update weights
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return self

    # Hypothetical function h( x )

    def predict( self, X ) :
        Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )
        Y = np.where( Z > 0.5, 1, 0 )
        return Y

# Driver code
def main() :

    # Importing dataset
    df = pd.read_csv( "pima.csv" )
    X = df.iloc[:,:-1].values
    Y = df.iloc[:,-1:].values

    # Splitting dataset into train and test set
    X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size = 1/3, random_state = 0 )

    # Model training
    model = LogitRegression( learning_rate = 0.01, iterations = 1000 )

    model.fit( X_train, Y_train )
    model1 = LogisticRegression()
    model1.fit( X_train, Y_train)

    # Prediction on test set
    Y_pred = model.predict( X_test )
    Y_pred1 = model1.predict( X_test )

    # measure performance
    correctly_classified = 0
```

```python
    correctly_classified1 = 0

    # counter
    count = 0
    for count in range( np.size( Y_pred ) ) :

        if Y_test[count] == Y_pred[count] :
            correctly_classified = correctly_classified + 1

        if Y_test[count] == Y_pred1[count] :
            correctly_classified1 = correctly_classified1 + 1

        count = count + 1

    print( "Accuracy on test set by our model    : ", (
    correctly_classified / count ) * 100 )
    print( "Accuracy on test set by sklearn model : ", (
    correctly_classified1 / count ) * 100 )


if __name__ == "__main__" :
    main()
```

OUTPUT

```
In [125]: runfile('C:/Users/Dell/Desktop/PYTHON/logisticsRegression.py', wdir='C:/Users/Dell/Desktop/PYTHON')
Accuracy on test set by our model    :  68.75
Accuracy on test set by sklearn model :  80.078125
```

Practical 8

Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets. Apriori Algorithm is a Machine Learning algorithm which is used to gain insight into the structured relationships between different items involved. The most prominent practical application of the algorithm is to recommend products based on the products already present in the user's cart. Walmart especially has made great use of the algorithm in suggesting products to its users.

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data = pd.read_excel(r"Online_Retail.xlsx")
print(data.head())

data['Description'] = data['Description'].str.strip()
# Dropping the rows without any invoice number
data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)
data['InvoiceNo'] = data['InvoiceNo'].astype('str')
# Dropping all transactions which were done on credit
data = data[~data['InvoiceNo'].str.contains('C')]

basket_France = (data[data['Country'] =="France"]
 .groupby(['InvoiceNo', 'Description'])['Quantity']
 .sum().unstack().reset_index().fillna(0)
 .set_index('InvoiceNo'))

 # Transactions done in the United Kingdom
basket_UK = (data[data['Country'] =="United Kingdom"]
 .groupby(['InvoiceNo', 'Description'])['Quantity']
 .sum().unstack().reset_index().fillna(0)
 .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
 .groupby(['InvoiceNo', 'Description'])['Quantity']
 .sum().unstack().reset_index().fillna(0)
 .set_index('InvoiceNo'))

basket_Sweden = (data[data['Country'] =="Sweden"]
 .groupby(['InvoiceNo', 'Description'])['Quantity']
 .sum().unstack().reset_index().fillna(0)
 .set_index('InvoiceNo'))
def hot_encode(x):
 if(x<= 0):
     return 0
 if(x>= 1):
     return 1
```

```python
# Encoding the datasets
basket_encoded = basket_France.applymap(hot_encode)
basket_France = basket_encoded
basket_encoded = basket_UK.applymap(hot_encode)
basket_UK = basket_encoded
basket_encoded = basket_Por.applymap(hot_encode)
basket_Por = basket_encoded
basket_encoded = basket_Sweden.applymap(hot_encode)
basket_Sweden = basket_encoded
frq_items = apriori(basket_France, min_support = 0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)

rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())
```

OUTPUT

```
In [6]: runfile('C:/Users/Dell/Desktop/PYTHON/q8.py', wdir='C:/Users/Dell/Desktop/PYTHON')
  InvoiceNo StockCode  ... CustomerID         Country
0    536365    85123A  ...    17850.0  United Kingdom
1    536365     71053  ...    17850.0  United Kingdom
2    536365    84406B  ...    17850.0  United Kingdom
3    536365    84029G  ...    17850.0  United Kingdom
4    536365    84029E  ...    17850.0  United Kingdom

[5 rows x 8 columns]
                                          antecedents  ... conviction
45                         (JUMBO BAG WOODLAND ANIMALS)  ...        inf
260  (PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ...  ...        inf
272  (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...  ...        inf
301  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  34.897959
302  (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...  ...  34.489796

[5 rows x 9 columns]
C:\Users\Dell\anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:111:
DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and
their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Practical 9

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm. The Euclidean distance formula is used to find the distance between two points on a plane.

In a few words, the Euclidean distance measures the *shortest path* between two points in a smooth n-dimensional space.

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

iris=pd.read_csv(r"iris_csv.csv")
print(iris.head())
x=iris.iloc[:,:4] #all parameters
y=iris["class"] #class labels

neigh=KNeighborsClassifier(n_neighbors=4)
neigh.fit(iris.iloc[:,:4],iris["class"])
testSet = [[1.4, 3.6, 3.4, 1.2]]
test = pd.DataFrame(testSet)

print(test)
print("predicted:",neigh.predict(test))
print("neighbors",neigh.kneighbors(test))
```

OUTPUT

```
In [3]: runfile('C:/Users/Dell/Desktop/PYTHON/q9.py', wdir='C:/Users/Dell/Desktop/PYTHON')
   sepallength  sepalwidth  petallength  petalwidth        class
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
     0    1    2    3
0  1.4  3.6  3.4  1.2
predicted: ['Iris-setosa']
neighbors (array([[3.7067506 , 3.80657326, 3.81706694, 3.8340579 ]]), array([[57,  8, 42, 93]],
dtype=int64))
```

Practical 10

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by 'K' in K-means.

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.

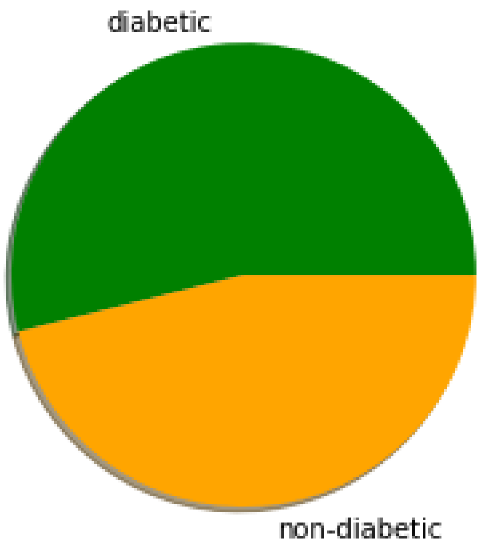
In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

K-means follows Expectation-Maximization approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.cluster.vq import whiten, kmeans, vq

# load the dataset
dataset = pd.read_csv(r"diabetes-train.csv")
# excluding the outcome column
#dataset = dataset[:, 0:8]
#print("Data :\n", dataset, "\n")
# normalize
dataset = whiten(dataset)
# generate code book
centroids, mean_dist = kmeans(dataset, 2)
print("Code-book :\n", centroids, "\n")
clusters, dist = vq(dataset, centroids)
print("Clusters :\n", clusters, "\n")
# count non-diabetic patients
non_diab = list(clusters).count(0)
# count diabetic patients
diab = list(clusters).count(1)
# depict illustration
x_axis = []
x_axis.append(diab)
x_axis.append(non_diab)
colors = ['green', 'orange']
print("No.of.diabetic patients : " + str(x_axis[0]) +
 "\nNo.of.non-diabetic patients : " + str(x_axis[1]))
y = ['diabetic', 'non-diabetic']
plt.pie(x_axis, labels=y, colors=colors, shadow='true')
plt.show()
```

OUTPUT

```
In [4]: runfile('C:/Users/Dell/Desktop/PYTHON/q10.py', wdir='C:/Users/Dell/Desktop/PYTHON')
Code-book :
 [[0.74632408 3.16427132 3.3583524  1.23534247 0.40848287 3.61275216
  1.15573669 2.38395608 0.1618391 ]
 [1.65793783 4.2863705  3.82561217 1.35250442 0.95719689 4.31388164
  1.66800558 3.52827518 1.48382815]]

Clusters :
 [0 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 1
 0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0
 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1
 0 0 1 1 1 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1
 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0
 1 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 1
 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1
 0 1 1 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 1 0 1 0
 0 1 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1
 0 0 0 0 1 0 0 1 0 0 0 0]

No.of.diabetic patients : 178
No.of.non-diabetic patients : 204
```



Practical 11

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

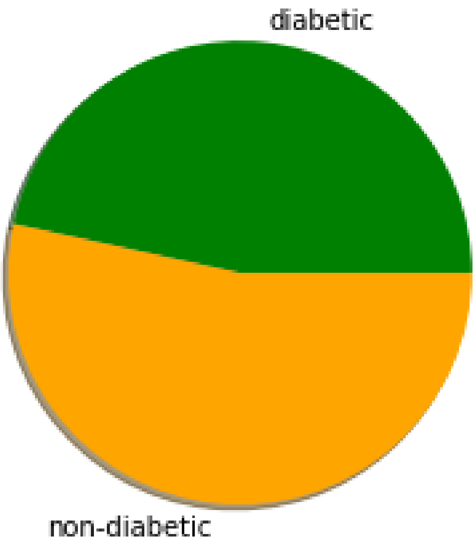
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

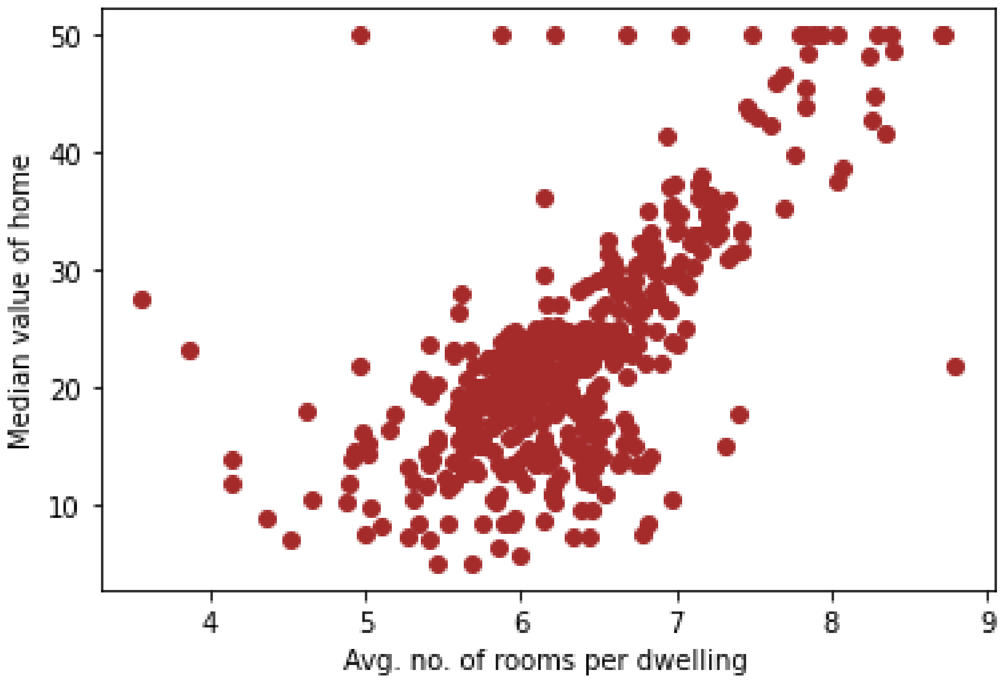
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import export_graphviz
#import pydotplus
boston=pd.read_csv(r"Boston.csv")
print(boston.head())
plt.scatter(x=boston['rm'],y=boston['medv'],color='brown')
plt.xlabel("Avg. no. of rooms per dwelling")
plt.ylabel("Median value of home")
x=pd.DataFrame(boston['rm'])
y=pd.DataFrame(boston['medv'])
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)
from sklearn.tree import DecisionTreeRegressor
regressor=DecisionTreeRegressor(criterion='squared_error', random_state=100,
max_depth=4, min_samples_leaf=1)
regressor.fit(x_train,y_train)
print(regressor)
export_graphviz(regressor, out_file='reg_tree.dot')
y_pred=regressor.predict(x_test)
print(y_pred[4:9])
print(y_test[4:9])
```

OUTPUT

```
In [5]: runfile('C:/Users/Dell/Desktop/PYTHON/q11.py', wdir='C:/Users/Dell/Desktop/PYTHON')
   Unnamed: 0     crim    zn  indus  chas  ...  tax  ptratio   black  lstat  medv
0           1  0.00632  18.0   2.31     0  ...  296     15.3  396.90   4.98  24.0
1           2  0.02731   0.0   7.07     0  ...  242     17.8  396.90   9.14  21.6
2           3  0.02729   0.0   7.07     0  ...  242     17.8  392.83   4.03  34.7
3           4  0.03237   0.0   2.18     0  ...  222     18.7  394.63   2.94  33.4
4           5  0.06905   0.0   2.18     0  ...  222     18.7  396.90   5.33  36.2

[5 rows x 15 columns]
DecisionTreeRegressor(max_depth=4, random_state=100)
[35.34705882 17.13157895 35.34705882 17.13157895 17.13157895]
     medv
227  31.6
383  12.3
299  29.0
19   18.2
426  10.2
```



diabetic
non-diabetic



Median value of home
Avg. no. of rooms per dwelling
50
40
30
20
10
4
5
6
7
8
9