UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION & TECHNOLOGY

Guru Gobind Singh Indraprastha

University Dwarka Sec-16c, New Delhi

DATA ANALYTICS

PRACTICAL FILE

COURSE - MCA(SE)

SUBMITTED BY-

Vidhi Teotia

03116404521

SUBMITTED TO-

PROF. Kavita Sethia

Contents

Practical 1: Introduction to data analytics using python.. 3

- Six steps of data analytics process 3
- Different sources of data for data analysis.. 3

Practical 2: Introduction to python libraries 6

Practical 3: Introduction to python programming... 8

Practical 4: Write a python program to do following operations: 12

Practical 5: Correlation matrix 14

Practical 6: Data pre-processing- handling missing values 17

Practical 7: Linear regression and Logistic regression 19

Practical 8:Apiriori Algorithm……………………………………………………………………………….22

Practical 9:KNN………………………………………………………………………………………………...25

Practical 10: K-means Clustering……………………………………………………………………………27

Practical 11:Decision Tree Classfication……………………………………………………………………30

Practical 1: Introduction to data analytics using python

- **Six steps of data analytics process**
- **Different sources of data for data analysis**

The collection, transformation, and organization of data to draw conclusions make predictions for the future, and make informed data-driven decisions is called Data Analysis.

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. Analyse
5. Share
6. Act or Report

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 favouring 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 origins of information:

- Statistical
- Census

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

Type of data sources

1. Statistical data source
2. Census data source

Additional sources of data

1. Internal sources of data
2. External sources of data

Practical 2: Introduction to python libraries

- **NumPy**
- **Pandas**
- **SciPy**
- **Scikit learn**
- **Matplotlib**
- **Seaborn**

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called Nd array, it provides a lot of supporting functions that make working with Nd array very easy. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

Pandas is a Python library used for working with data sets.

It has functions for analysing, cleaning, exploring, and manipulating data. Pandas allows us to analyse big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values.

SciPy is a scientific computation library that uses NumPy underneath.

SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. SciPy has optimized and added functions that are frequently used in NumPy and Data Science. SciPy is predominantly written in Python, but a few segments are written in C.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.

It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Seaborn is a library that uses Matplotlib underneath to plot graphs.

It will be used to visualize random distributions. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas. Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

Practical 3: Introduction to python programming

- **Datatypes**
- **Operators**
- **Loops**
- **Central tendency measure**
- **Matrix operation**

Datatypes

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Operators

Arithmetic Operators

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Comparison Operators

Comparison of Relational operators compares the values. It either returns True or False according to the condition.

Logical Operators

Logical operators perform Logical AND, Logical OR, and Logical NOT operations. It is used to combine conditional statements.

Bitwise Operators

Bitwise operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

Assignment Operators

Assignment operators are used to assign values to the variables.

Membership Operators

In and not in are the membership operators; used to test whether a value or variable is in a sequence.

Central tendency measure

Mathematically central tendency means measuring the center or distribution of location of values of a data set. It gives an idea of the average value of the data in the data set and also an indication of how widely the values are spread in the data set. That in turn helps in evaluating the chances of a new input fitting into the existing data set and hence probability of success.

There are three main measures of central tendency which can be calculated using the methods in pandas python library.

- Mean - It is the Average value of the data which is a division of sum of the values with the number of values.
- Median - It is the middle value in distribution when the values are arranged in ascending or descending order.
- Mode - It is the most commonly occurring value in a distribution.

Matrix operation

```
import numpy as np
```

```
arr = np.array( [[1,2,3,5],
                 [4,5,6,4],
                 [7,8,9,10]] )
print("Shape of matrix is ",arr.shape )
print("Dimension of the matrix is ",arr.ndim )
```

```
Shape of matrix is  (3, 4)
Dimension of the matrix is  2
```

```
[3] a = np.zeros((3,4))
    b = np.ones((3,4) )
    print( "Matrix full of zeros ",a )
    print( "Matrix full of ones ",b )
```

```
Matrix full of zeros  [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Matrix full of ones  [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

```
[4] narr = arr.reshape( 2,2,3 )
    farr = arr.flatten()
    print("New array after reshaping ",narr )
    print("New array after flattening ",farr )
```

```
New array after reshaping  [[[ 1  2  3]
  [ 5  4  5]]

 [[ 6  4  7]
  [ 8  9 10]]]
New array after flattening  [ 1  2  3  5  4  5  6  4  7  8  9 10]
```

```
print("Horizontal append ", np.append( a, b ))
print("Vertical append ", np.append( a,b,axis=0 ))
```

```
Horizontal append  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Vertical append  [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

```
[6] print("Array indexing at [0,1] is ", arr[0][1] )
    print("Array slicing from [2,3] is ",arr[1,2:4] )
```

```
Array indexing at [0,1] is  2
Array slicing from [2,3] is  [6 4]
```

```
[7] print("Max element of array ",np.max(arr) )
    print("Min element of array ",np.min(arr) )
    print("Mean of array ",np.mean(arr) )
    print("Median of array ",np.median(arr) )
    print("Standard deviation of array ",np.std(arr) )
```

```
Max element of array  10
Min element of array  1
Mean of array  5.333333333333333
Median of array  5.0
Standard deviation of array  2.6562295750848715
```

Linear algebra on matrices

```
import numpy as np
```

```
arr = np.array( [[1,2,3],
                 [4,5,6],
                 [7,8,9]] )
brr = np.array([[7,8,9],
                [4,5,6],
                [1,2,3]] )
```

```
print("Dot product of two array arr and brr is ",np.dot( arr,brr ) )
print("Cross product of two array arr and brr is ", np.cross( arr,brr ))
```

```
Dot product of two array arr and brr is  [[ 18  24  30]
 [ 54  69  84]
 [ 90 114 138]]
Cross product of two array arr and brr is  [[ -6  12  -6]
 [  0   0   0]
 [  6 -12   6]]
```

```
w,v = np.linalg.eig( arr )
print("Eignen value of the array is ",w )
print("Right eigenvectors of the array is ",v )
```

```
Eignen value of the array is  [ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
Right eigenvectors of the array is  [[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

```
var = np.array( [[3,1],
                [1,2]] )
eq = np.array( [9,8] )
print("Linear equation solutin is ", np.linalg.solve( var,eq ))
```

```
Linear equation solutin is  [2. 3.]
```

```
print("Multiplicative inverse of array is ", np.linalg.inv( var ))
```

```
Multiplicative inverse of array is  [[ 0.4 -0.2]
 [-0.2  0.6]]
```

```
print("Rank of matrix is ", np.linalg.matrix_rank(arr))
```

```
Rank of matrix is  2
```

```
print("Determinant of matrix is ", np.linalg.det(var))
```

```
Determinant of matrix is  5.000000000000001
```

Practical 4: Write a python program to do following operations:

- **Dataset : brain_size.csv**
- **Library: pandas**
- **Load data from csv file**
- **Compute basic statistics of given data- shape, number of columns, mean**
- **Splitting data frame on values of categorical variables**
- **Visualizing 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.

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
path="/content/drive/MyDrive/brain_size.csv"
df = pd.read_csv(path)
```

```
print("Shape of data file is ", df.shape)
col = len( df.axes[1] )
row = len( df.axes[0] )
print("No. of coloumn in data file is ",col)
print("No. of rows in data file is ", row )
```

```
Shape of data file is  (39, 3)
No. of coloumn in data file is  3
No. of rows in data file is  39
```

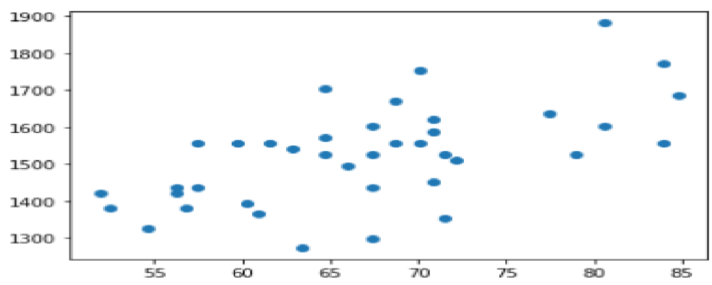
```
[4] df1 = df.groupby('species')
    print(df1.get_group('human'))
```

```
   species  mass   brain
0    human  54.6  1326.1
1    human  52.5  1380.2
2    human  51.9  1422.3
3    human  56.8  1380.2
4    human  60.9  1366.5
5    human  60.3  1394.1
6    human  57.4  1436.6
7    human  56.3  1422.3
8    human  56.3  1436.6
9    human  57.4  1556.2
10   human  59.7  1556.2
11   human  61.6  1556.2
12   human  62.8  1540.7
13   human  64.7  1525.4
14   human  66.0  1495.2
15   human  67.4  1436.6
16   human  70.8  1451.0
```

```
df2 = df[df['mass']>=70]
print(df2)
```

```
        species  mass   brain
16        human  70.8  1451.0
17        human  72.2  1510.2
18        human  79.0  1525.4
19        human  71.5  1525.4
23        human  70.1  1556.2
24        human  70.8  1587.6
25        human  70.8  1619.7
28        human  80.6  1881.8
29        human  83.9  1772.2
32  neanderthal  71.5  1352.9
34  neanderthal  70.1  1754.6
35  neanderthal  77.5  1636.0
36  neanderthal  80.6  1603.6
37  neanderthal  83.9  1556.2
38  neanderthal  84.8  1685.8
```

```
[6] import matplotlib.pyplot as plt
    plt.scatter(df['mass'],df['brain'])
    plt.show()
```



Practical 5: Correlation matrix

Dataset: Pima Indian diabetes dataset

Library: Scipy

- **Load data set describe the given data identify the missing, outlier data items**
- **Find correlation among all attributes**
- **Visualisation correlation matrix**

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.

A correlation matrix consists of rows and columns that show the variables. Each cell in a table contains the correlation coefficient.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] import pandas as pd
    import numpy as np
    import statistics as st
    import matplotlib.pyplot as plt
    from scipy import stats
    import seaborn as sb
    path = "/content/drive/MyDrive/diabetes.csv"
    df = pd.read_csv(path)
```

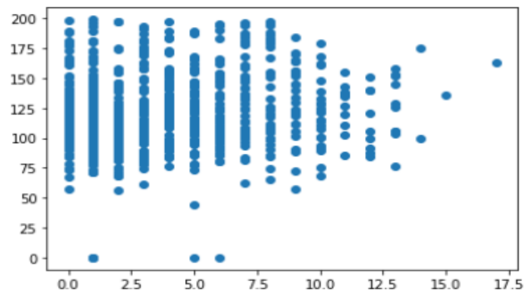
```
[ ] print(df[:10])
    print(df.describe())
    print(df.isnull().sum())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1
5            5      116             74              0        0  25.6
6            3       78             50             32       88  31.0
7           10      115              0              0        0  35.3
8            2      197             70             45      543  30.5
9            8      125             96              0        0   0.0
```

```
[ ]    DiabetesPedigreeFunction  Age  Outcome
     0                    0.627   50        1
     1                    0.351   31        0
     2                    0.672   32        1
     3                    0.167   21        0
     4                    2.288   33        1
     5                    0.201   30        0
     6                    0.248   26        1
     7                    0.134   29        0
     8                    0.158   53        1
     9                    0.232   54        1
            Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
     count   768.000000  768.000000     768.000000     768.000000  768.000000
     mean      3.845052  120.894531      69.105469      20.536458   79.799479
     std       3.369578   31.972618      19.355807      15.952218  115.244002
     min       0.000000    0.000000       0.000000       0.000000    0.000000
     25%       1.000000   99.000000      62.000000       0.000000    0.000000
     50%       3.000000  117.000000      72.000000      23.000000   30.500000
     75%       6.000000  140.250000      80.000000      32.000000  127.250000
     max      17.000000  199.000000     122.000000      99.000000  846.000000

                   BMI  DiabetesPedigreeFunction         Age     Outcome
     count  768.000000                768.000000  768.000000  768.000000
     mean    31.992578                  0.471876   33.240885    0.348958
     std      7.884160                  0.331329   11.760232    0.476951
     min      0.000000                  0.078000   21.000000    0.000000
     25%     27.300000                  0.243750   24.000000    0.000000
     50%     32.000000                  0.372500   29.000000    0.000000
     75%     36.600000                  0.626250   41.000000    1.000000
     max     67.100000                  2.420000   81.000000    1.000000
     Pregnancies                 0
     Glucose                     0
     BloodPressure               0
     SkinThickness               0
     Insulin                     0
     BMI                         0
     DiabetesPedigreeFunction    0
     Age                         0
     Outcome                     0
     dtype: int64
```

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

```
<matplotlib.collections.PathCollection at 0x7f154d60c9d0>
```




```
[ ]  mean = np.mean(df)
     sd = np.std(df)
     print(mean["Pregnancies"])
```

```
3.8450520833333335
```

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

```
[ 13.94720292 216.74991897 127.13507364  68.36194421 425.30632696
  55.62965532   1.46521475  68.49860335   1.77888062]
```

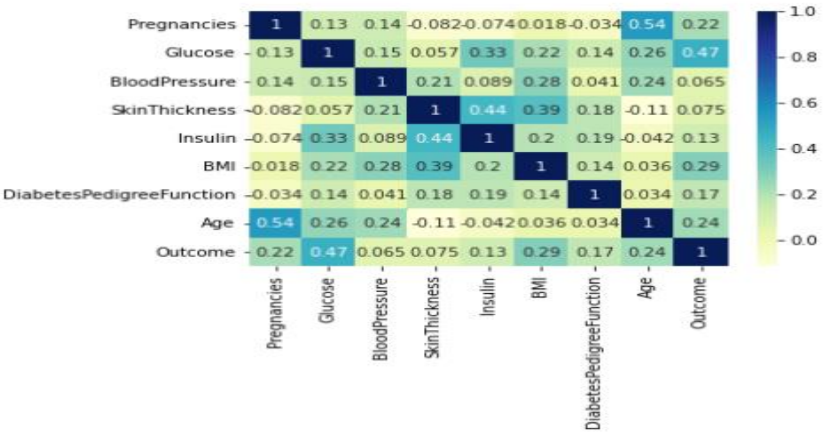
```python
z = np.abs(stats.zscore(df['Glucose']))
idx_outliers = np.where(z>1, True, False)
ans = pd.Series(idx_outliers, index = df['Glucose'].index)
print(ans)
print(np.sum(ans))
```

```
0      False
1       True
2       True
3      False
4      False
       ...
763    False
764    False
765    False
766    False
767    False
Length: 768, dtype: bool
228
```

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

```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282      -0.081672
Glucose                      0.129459  1.000000       0.152590       0.057328
BloodPressure                0.141282  0.152590       1.000000       0.207371
SkinThickness               -0.081672  0.057328       0.207371       1.000000
Insulin                     -0.073535  0.331357       0.088933       0.436783
BMI                          0.017683  0.221071       0.281805       0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
Age                          0.544341  0.263514       0.239528      -0.113970
Outcome                      0.221898  0.466581       0.065068       0.074752

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

```python
dataplot = sb.heatmap(df.corr(), cmap="YlGnBu", annot=True)
plt.show()
```



Practical 6: Data pre-processing- handling missing values

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

- **Remove rows / attributes**
- **Replace with mean or mode**
- **Transformation of data using discretization and normalization on given dataset**

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

Need of Data Preprocessing

- For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
- Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
path = "/content/drive/MyDrive/employees.csv"
df= pd.read_csv(path)
data = pd.read_csv(path)
data2 = pd.read_csv(path)
df.head()
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

```
[ ] df.isnull().sum()
    df.shape
```

```
(1000, 8)
```

```
[ ] df= df.dropna(axis=0)
    df.shape
```

```
(764, 8)
```

```
[ ] data = data.dropna(axis=1)
    data.shape
```

```
(1000, 4)
```

```
[ ] data2 = data2.fillna(data2.mean())
    data2.isnull().sum()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame
  """Entry point for launching an IPython kernel.
First Name            67
Gender               145
Start Date             0
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management      0
Team                  43
dtype: int64
```

```
[ ] mode = data2['Team'].mode().values[0]
    data2['Team'] = data2['Team'].replace(np.nan,mode)
    df.isnull().sum()
```

```
First Name           0
Gender               0
Start Date           0
Last Login Time      0
Salary               0
Bonus %              0
Senior Management    0
Team                 0
dtype: int64
```

```
[ ] minv = data2['Bonus %'].min()
    maxv = data2['Bonus %'].max()
    print(minv)
    print(maxv)
```

```
1.015
19.944
```

```
[ ] bins = np.linspace(minv,maxv,4)
    print(bins)
```

```
[ 1.015       7.32466667 13.63433333 19.944     ]
```

```
[ ] labels = ['small', 'medium', 'big']
    data2['bins']=pd.cut(data2['Bonus %'], bins=bins, labels=labels, include_lowest=True )
    print( data2['Bonus %'].head())
```

```
0     6.945
1     4.170
2    11.858
3     9.340
4     1.389
Name: Bonus %, dtype: float64
```

```
[ ] data2.head()
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	bins
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	small
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	small
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	medium
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	medium
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	small

Practical 7: Linear regression and Logistic regression

Linear regression:

Linear regression 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.

Mathematical formula to calculate slope and intercept are given below

Slope = Sxy/Sxx

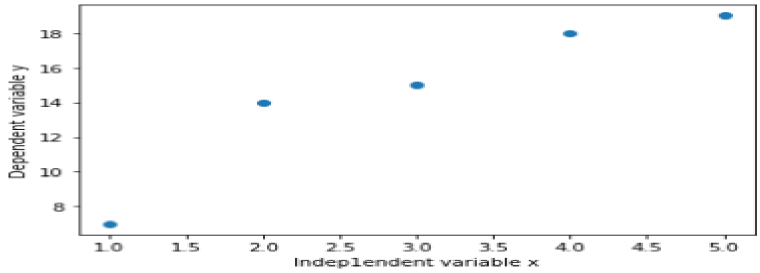
where Sxy and Sxx are sample covariance and sample variance respectively.

Intercept = ymean – slope* xmean

```
[34] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error, r2_score
     import statsmodels.api as st
     from sklearn.linear_model import LogisticRegression
```

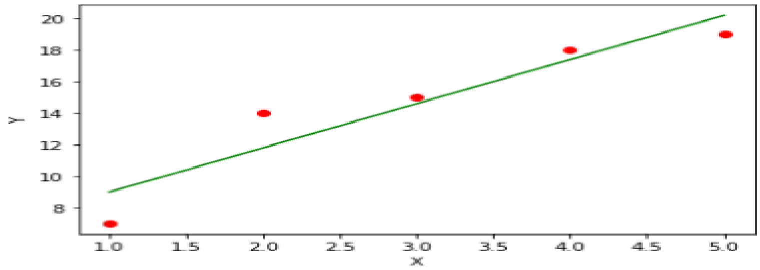
```
[35] x = np.array([1,2,3,4,5])
     y = np.array([7,14,15,18,19])
     n = np.size(x)

     x_mean = np.mean(x)
     y_mean = np.mean(y)

     sxy = np.sum(x*y) - x_mean*n*y_mean
     sxx = np.sum(x*x) - x_mean*n*x_mean

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

     plt.scatter(x,y)
     plt.xlabel('Indep1endent variable x')
     plt.ylabel('Dependent variable y')
```

```
slope b1 is  2.8
intercept b0 is  6.200000000000001
Text(0, 0.5, 'Dependent variable y')
```




```
[36] y_pred = b1*x+b0
     plt.scatter(x,y,color='red' )
     plt.plot( x,y_pred,color='green' )
     plt.xlabel('X')
     plt.ylabel('Y')
```

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




```
error = y-y_pred
se = np.sum(error**2)
print('squarred error is ',se )

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

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

sst = np.sum((y-y_mean)**2)
r2 = 1-(se/sst)
print('r square is ',r2 )
```

```
squarred error is  10.800000000000004
mean squarred error is  2.160000000000001
root mean squarred error is  1.4696938456699071
r square is  0.8789237668161435
```

```
[38] x = x.reshape(-1,1)

     regression_model = LinearRegression()
     regression_model.fit(x, y)

     y_predicted = regression_model.predict(x)
     mse = mean_squared_error(y,y_predicted)
     rmse = np.sqrt(mean_squared_error(y,y_predicted))
     r2 = r2_score( y,y_predicted )

     print('Slope:' ,regression_model.coef_)
     print('Intercept:', regression_model.intercept_)
     print('MSE:',mse)
     print('Root mean squared error: ', rmse)
     print('R2 score: ', r2)
```

```
Slope: [2.8]
Intercept: 6.199999999999999
MSE: 2.160000000000001
Root mean squared error:  1.4696938456699071
R2 score:  0.8789237668161435
```

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.

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))

the simplified cost function we use :

J = - ylog(h(x)) - (1 - y)log(1 - h(x)) here, y is the real target value

h(x) = sigmoid(wx + b)

For y = 0,

J = - log(1 - h(x))

and y = 1,

J = - log(h(x))

This cost function is because when we train, we need to maximize the probability by minimizing the loss function.

Gradient Descent Calculation:

repeat until convergence {   tmpi = wi - alpha * dwi   wi = tmpi   }

where alpha is the learning rate.

The chain rule is used to calculate the gradients like i.e dw.

Chain rule for dw

here, a = sigmoid(z) and z = wx + b.

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings( "ignore" )
from sklearn.linear_model import LogisticRegression
class LogitRegression() :
    def __init__( self, learning_rate, iterations ) :
        self.learning_rate = learning_rate
        self.iterations = iterations
    def fit( self, X, Y ) :
        self.m, self.n = X.shape
        self.W = np.zeros( self.n )
        self.b = 0
        self.X = X
        self.Y = Y

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

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

        tmp = ( A - self.Y.T )
        tmp = np.reshape( tmp, self.m )
        dW = np.dot( self.X.T, tmp ) / self.m
        db = np.sum( tmp ) / self.m
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db
        return self

    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
```

```
def main() :
    df = pd.read_csv( "/content/drive/MyDrive/diabetes.csv" )
    X = df.iloc[:,:-1].values
    Y = df.iloc[:,-1:].values
    X_train, X_test, Y_train, Y_test = train_test_split(
      X, Y, test_size = 1/3, random_state = 0 )
    model = LogitRegression( learning_rate = 0.01, iterations = 1000 )

    model.fit( X_train, Y_train )
    model1 = LogisticRegression()
    model1.fit( X_train, Y_train)
    Y_pred = model.predict( X_test )
    Y_pred1 = model1.predict( X_test )
    correctly_classified = 0
    correctly_classified1 = 0
    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()
```

```
Accuracy on test set by our model       :   33.59375
Accuracy on test set by sklearn model   :   80.46875
```

8. Apriori algorithm

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 it’s users.

import numpy as np

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

data = pd.read_excel(r"C:\Users\HP\OneDrive\Desktop\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
# Encoding the datasets
basket_encoded = basket_France.applymap(hot_encode)
basket_France = basket_encode
basket_encoded = basket_UK.applymap(hot_encode)
basket_UK = basket_encode
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())
```

```
IPython 8.2.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/HP/.spyder-py3/Q8_new.py', wdir='C:/Users/HP/.spyder-py3')
  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]
C:\Users\HP\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(
C:\Users\HP\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(
                                       antecedents  ... conviction
44                         (JUMBO BAG WOODLAND ANIMALS)  ...        inf
259  (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...  ...        inf
270  (PLASTERS IN TIN WOODLAND ANIMALS, RED TOADSTO...  ...        inf
302  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  34.897959
300  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  34.489796

[5 rows x 9 columns]
```

9. KNN

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.

```
import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

iris=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\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))

```
In [2]: runfile('C:/Users/HP/.spyder-py3/Q9.py', wdir='C:/Users/HP/.spyder-py3')
   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))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid
feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid
feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

10. K-means Clustering

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering

algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.

Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

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

```
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"C:\Users\HP\OneDrive\Desktop\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()
```

```
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.2.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/HP/.spyder-py3/Q10.py', wdir='C:/Users/HP/.spyder-py3')
Code-book :
 [[1.6973767  4.29706228 3.84154059 1.31802321 0.93353    4.279573
  1.66229516 3.55694095 1.51478591]
 [0.73548744 3.18226567 3.35634609 1.26668725 0.44120029 3.65792461
  1.17271874 2.38760401 0.16784031]]

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

No.of.diabetic patients : 209
No.of.non-diabetic patients : 173
```

11. Decision Tree Classification

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

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

The decisions or the test are performed on the basis of features of the given dataset.

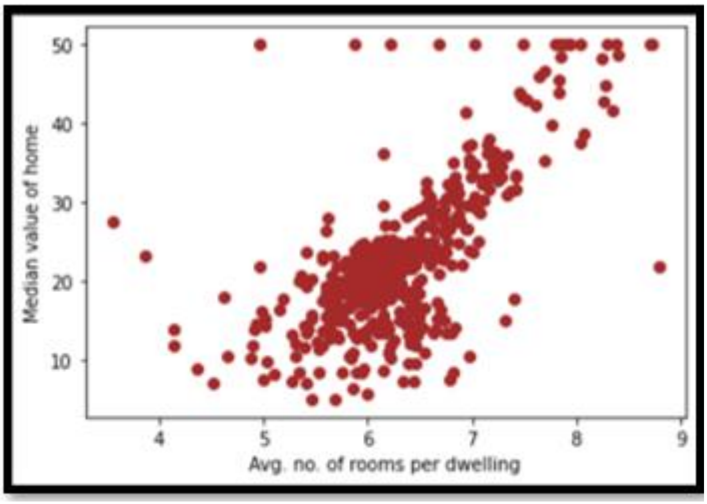
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

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.

```
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"C:\Users\HP\OneDrive\Desktop\boston1.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])
```

```
In [4]: runfile('C:/Users/HP/.spyder-py3/Q11.py', wdir='C:/Users/HP/.spyder-py3')
      crim    zn  indus  chas    nox  ...  tax  ptratio       b  lstat  medv
0  0.00632  18.0   2.31     0  0.538  ...  296     15.3  396.90   4.98  24.0
1  0.02731   0.0   7.07     0  0.469  ...  242     17.8  396.90   9.14  21.6
2  0.02729   0.0   7.07     0  0.469  ...  242     17.8  392.83   4.03  34.7
3  0.03237   0.0   2.18     0  0.458  ...  222     18.7  394.63   2.94  33.4
4  0.06905   0.0   2.18     0  0.458  ...  222     18.7  396.90   5.33  36.2

[5 rows x 14 columns]
DecisionTreeRegressor(max_depth=4, random_state=100)
[19.78138298 19.78138298 17.04186047 19.78138298 19.78138298]
     medv
109  19.4
205  22.6
399   6.3
293  23.9
491  13.6

In [5]:
```


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