Pandas Library

- In real world, we generally require labelled data for effective data analysis and meaningful interpretation.
- Python has an important library named “Pandas” which is helpful for processing labelled data.
- The most useful data structure for data analysis is dataframe. Dataframe in R was inbuilt, in Python it can be created using Pandas library.

Basis of Dataframe:

1. Creating a Series:

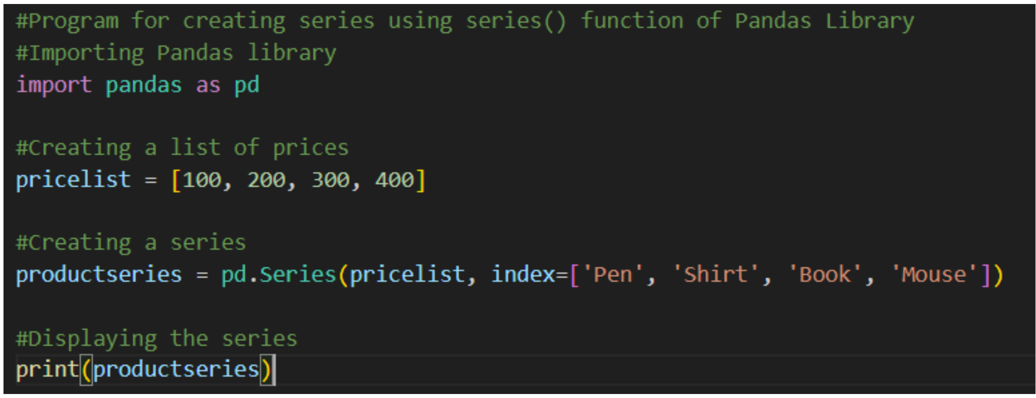
- Series is a one-dimensional labelled data.
- A series can be created using the function **Series()** from Pandas Library.

Syntax:

Series(data, index = list of column names)

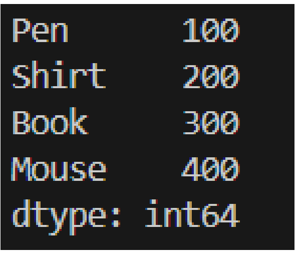
Where,

- data represents any data type(integers, strings, floating point numbers, Python objects, etc.)
- index represents the axis labels.



```python
#Program for creating series using series() function of Pandas Library
#Importing Pandas library
import pandas as pd

#Creating a list of prices
pricelist = [100, 200, 300, 400]

#Creating a series
productseries = pd.Series(pricelist, index=['Pen', 'Shirt', 'Book', 'Mouse'])

#Displaying the series
print(productseries)
```

Output:



```
Pen      100
Shirt    200
Book     300
Mouse    400
dtype: int64
```

2. Creating a Dataframe:

- Dataframe is the most commonly used pandas object and is represented as a two-dimensional labelled data structure with the column of potentially different types.
- It can be thought as a table in RDBMS or a spreadsheet.
- A dataframe is created using **DataFrame()** function.
- **shape-** The dimension of the dataframe can be determined
  Syntax - variable_name.shape
- **Key() –** names of the columns can be determined
  Syntax – variable_name.key()

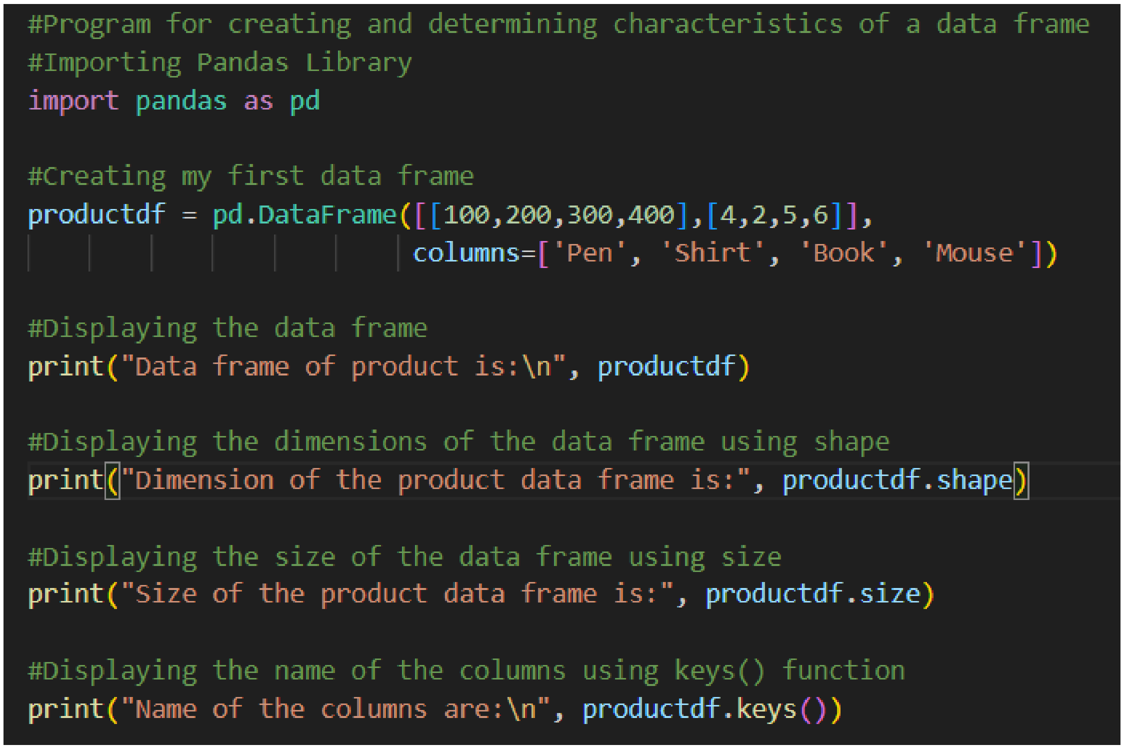
- **Size –** it is used to get the size of the dataframe - rows*columns as result
  Syntax- variable_name.size

Syntax:

DataFrame(data, column = list of column names)

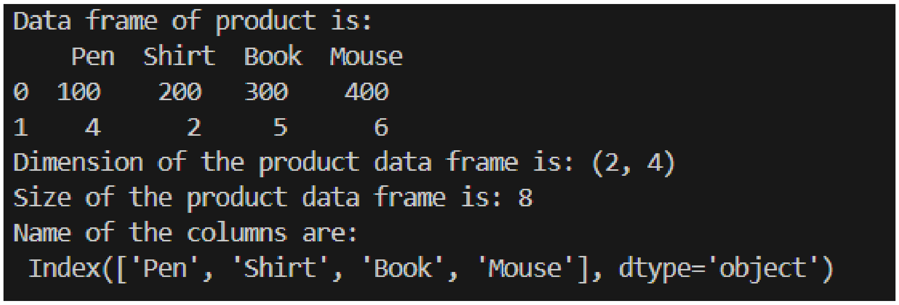
Where,

- data represents a multi-dimensional data of any data type(integers, strings, floating point numbers, etc.)
- columns has a list representing name of the columns.



```python
#Program for creating and determining characteristics of a data frame
#Importing Pandas Library
import pandas as pd

#Creating my first data frame
productdf = pd.DataFrame([[100,200,300,400],[4,2,5,6]],
                         columns=['Pen', 'Shirt', 'Book', 'Mouse'])

#Displaying the data frame
print("Data frame of product is:\n", productdf)

#Displaying the dimensions of the data frame using shape
print("Dimension of the product data frame is:", productdf.shape)

#Displaying the size of the data frame using size
print("Size of the product data frame is:", productdf.size)

#Displaying the name of the columns using keys() function
print("Name of the columns are:\n", productdf.keys())
```

Output:

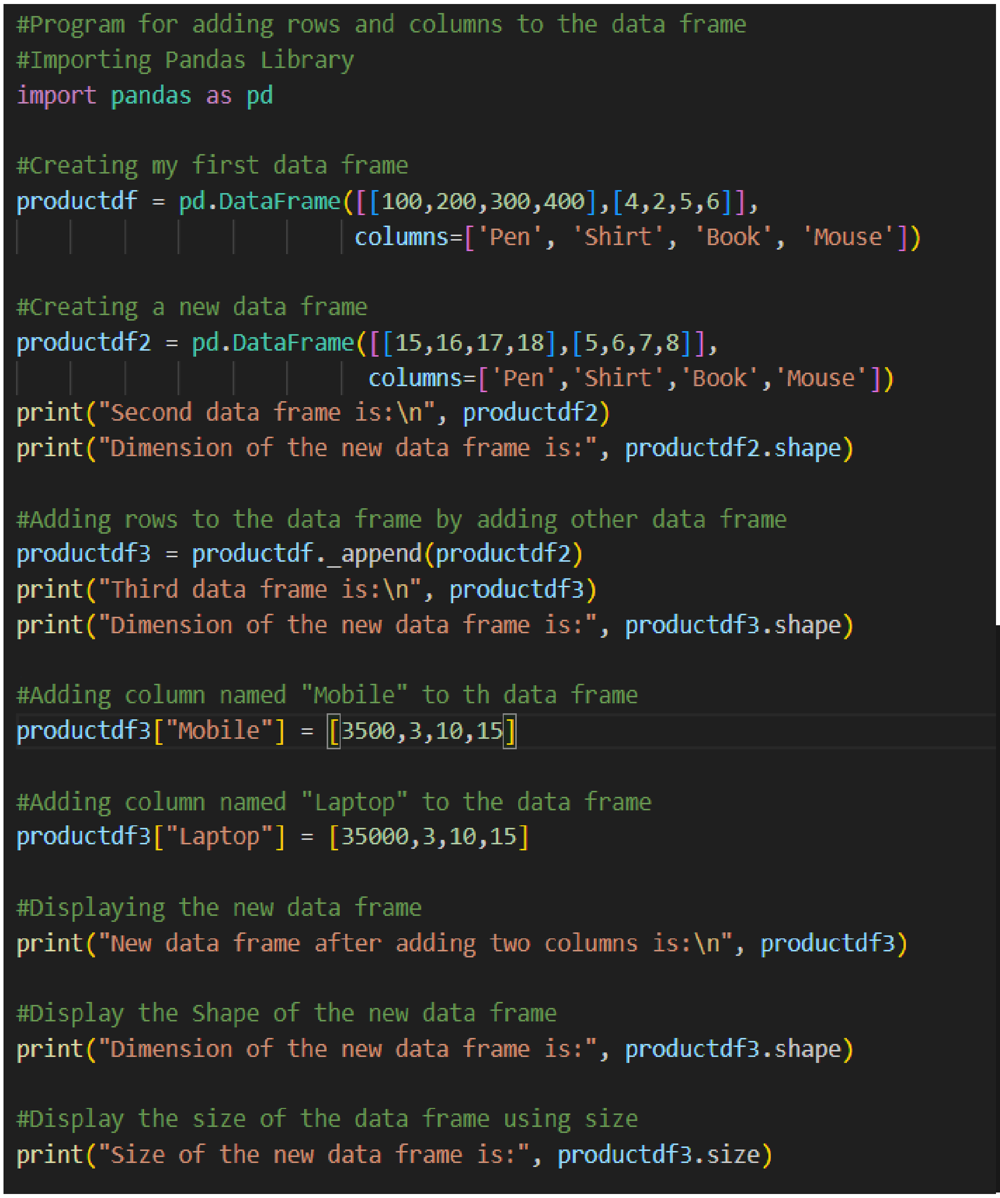


```
Data frame of product is:
    Pen  Shirt  Book  Mouse
0  100    200   300    400
1    4      2     5      6
Dimension of the product data frame is: (2, 4)
Size of the product data frame is: 8
Name of the columns are:
 Index(['Pen', 'Shirt', 'Book', 'Mouse'], dtype='object')
```

Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

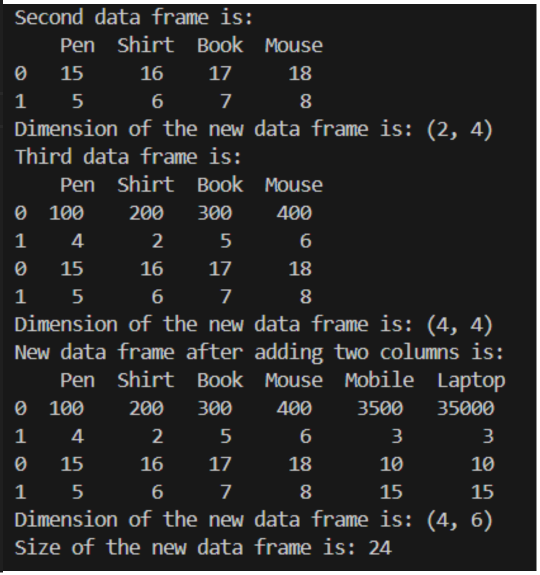
3. Adding Rows And Columns to the Dataframe

- We can add rows to an existing dataframe from a new dataframe using the **_append()** function.
- We can add a new column to the dataframe by writing the name of the column in square brackets along with the name of the dataframe and assigning a list of items to it.



```
#Program for adding rows and columns to the data frame
#Importing Pandas Library
import pandas as pd

#Creating my first data frame
productdf = pd.DataFrame([[100,200,300,400],[4,2,5,6]],
                         columns=['Pen', 'Shirt', 'Book', 'Mouse'])

#Creating a new data frame
productdf2 = pd.DataFrame([[15,16,17,18],[5,6,7,8]],
                          columns=['Pen','Shirt','Book','Mouse'])
print("Second data frame is:\n", productdf2)
print("Dimension of the new data frame is:", productdf2.shape)

#Adding rows to the data frame by adding other data frame
productdf3 = productdf._append(productdf2)
print("Third data frame is:\n", productdf3)
print("Dimension of the new data frame is:", productdf3.shape)

#Adding column named "Mobile" to th data frame
productdf3["Mobile"] = [3500,3,10,15]

#Adding column named "Laptop" to the data frame
productdf3["Laptop"] = [35000,3,10,15]

#Displaying the new data frame
print("New data frame after adding two columns is:\n", productdf3)

#Display the Shape of the new data frame
print("Dimension of the new data frame is:", productdf3.shape)

#Display the size of the data frame using size
print("Size of the new data frame is:", productdf3.size)
```

Output:

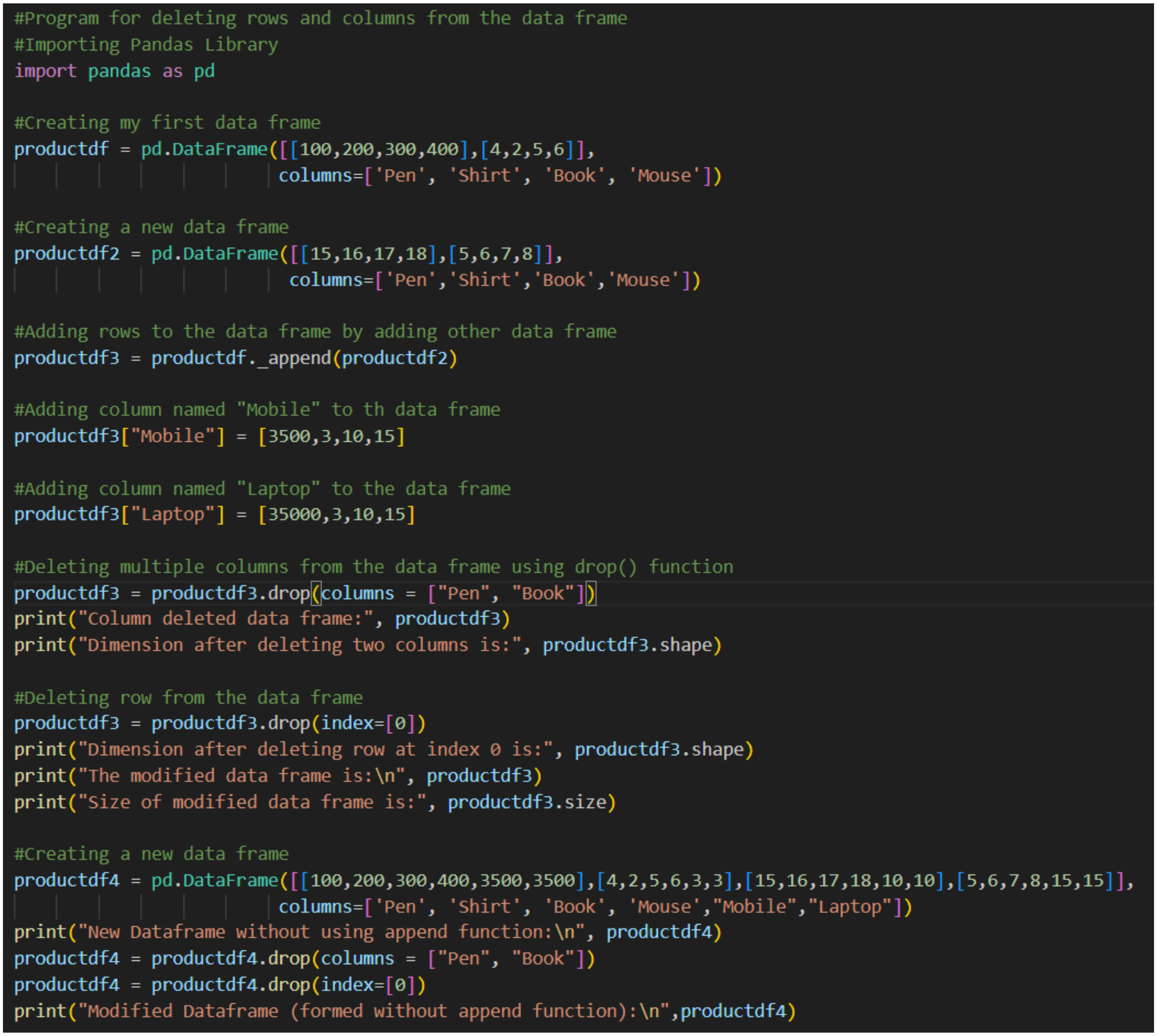


```
Second data frame is:
    Pen  Shirt  Book  Mouse
0   15     16    17     18
1    5      6     7      8
Dimension of the new data frame is: (2, 4)
Third data frame is:
    Pen  Shirt  Book  Mouse
0  100    200   300    400
1    4      2     5      6
0   15     16    17     18
1    5      6     7      8
Dimension of the new data frame is: (4, 4)
New data frame after adding two columns is:
    Pen  Shirt  Book  Mouse  Mobile  Laptop
0  100    200   300    400    3500   35000
1    4      2     5      6       3       3
0   15     16    17     18      10      10
1    5      6     7      8      15      15
Dimension of the new data frame is: (4, 6)
Size of the new data frame is: 24
```

Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

4. Deleting Rows and Columns from the Dataframe

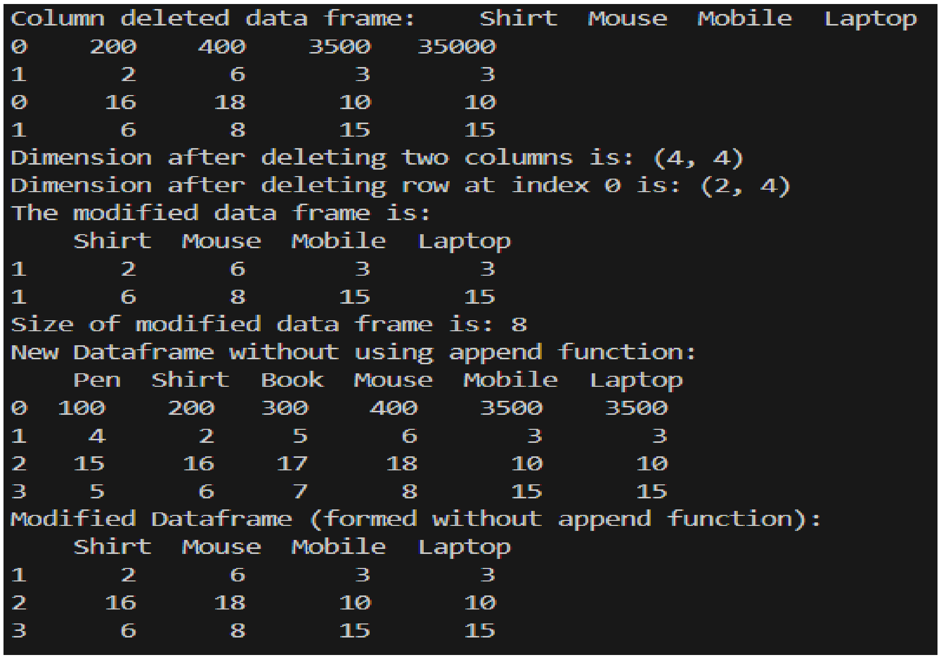
- Columns can be deleted using **drop()** function by passing name of the columns as value of the “columns” argument in the function in the form of list.
  Syntax: variable_name.drop(columns=[“names”,”name”])
- Rows can be deleted by specifying the index of the rows to be deleted as the value of the “index” argument in **drop()** function.
  Syntax: **variable_name.drop(index=0)**



```
#Program for deleting rows and columns from the data frame
#Importing Pandas Library
import pandas as pd

#Creating my first data frame
productdf = pd.DataFrame([[100,200,300,400],[4,2,5,6]],
                         columns=['Pen', 'Shirt', 'Book', 'Mouse'])

#Creating a new data frame
productdf2 = pd.DataFrame([[15,16,17,18],[5,6,7,8]],
                          columns=['Pen','Shirt','Book','Mouse'])

#Adding rows to the data frame by adding other data frame
productdf3 = productdf._append(productdf2)

#Adding column named "Mobile" to th data frame
productdf3["Mobile"] = [3500,3,10,15]

#Adding column named "Laptop" to the data frame
productdf3["Laptop"] = [35000,3,10,15]

#Deleting multiple columns from the data frame using drop() function
productdf3 = productdf3.drop(columns = ["Pen", "Book"])
print("Column deleted data frame:", productdf3)
print("Dimension after deleting two columns is:", productdf3.shape)

#Deleting row from the data frame
productdf3 = productdf3.drop(index=[0])
print("Dimension after deleting row at index 0 is:", productdf3.shape)
print("The modified data frame is:\n", productdf3)
print("Size of modified data frame is:", productdf3.size)

#Creating a new data frame
productdf4 = pd.DataFrame([[100,200,300,400,3500,3500],[4,2,5,6,3,3],[15,16,17,18,10,10],[5,6,7,8,15,15]],
                         columns=['Pen', 'Shirt', 'Book', 'Mouse',"Mobile","Laptop"])
print("New Dataframe without using append function:\n", productdf4)
productdf4 = productdf4.drop(columns = ["Pen", "Book"])
productdf4 = productdf4.drop(index=[0])
print("Modified Dataframe (formed without append function):\n",productdf4)
```

Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

Output:



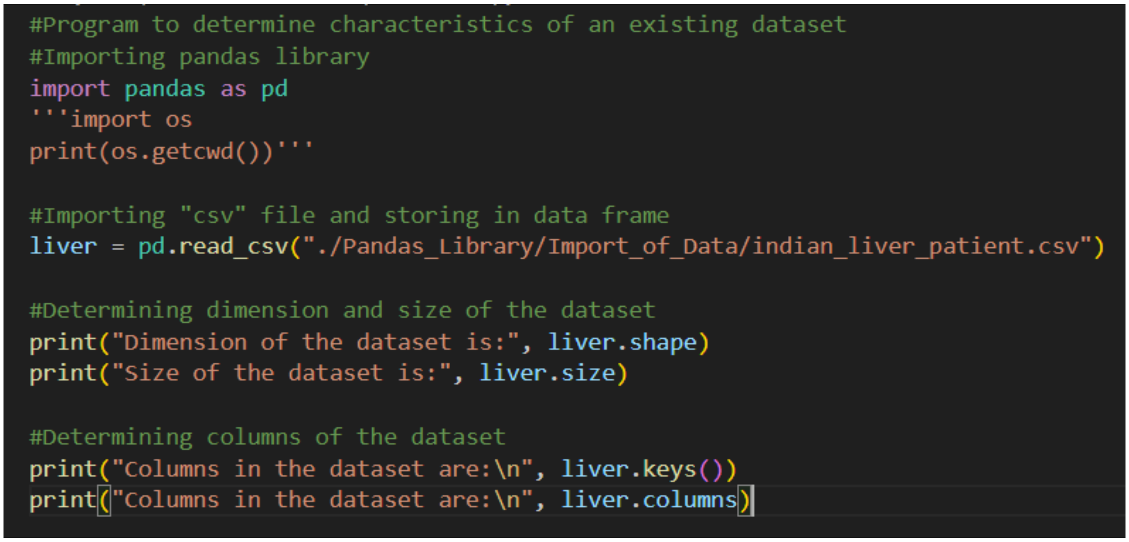
```
Column deleted data frame:    Shirt  Mouse  Mobile  Laptop
0    200    400    3500   35000
1      2      6       3       3
0     16     18      10      10
1      6      8      15      15
Dimension after deleting two columns is: (4, 4)
Dimension after deleting row at index 0 is: (2, 4)
The modified data frame is:
    Shirt  Mouse  Mobile  Laptop
1      2      6       3       3
1      6      8      15      15
Size of modified data frame is: 8
New Dataframe without using append function:
    Pen  Shirt  Book  Mouse  Mobile  Laptop
0  100    200   300    400    3500    3500
1    4      2     5      6       3       3
2   15     16    17     18      10      10
3    5      6     7      8      15      15
Modified Dataframe (formed without append function):
    Shirt  Mouse  Mobile  Laptop
1      2      6       3       3
2     16     18      10      10
3      6      8      15      15
```

Import of Data:

- The Pandas library provides many functions to import data from files of different types of software and stores n a dataframe in Python.
- **Dataset Link:** https://www.kaggle.com/datasets/uciml/indian-liver-patient-records

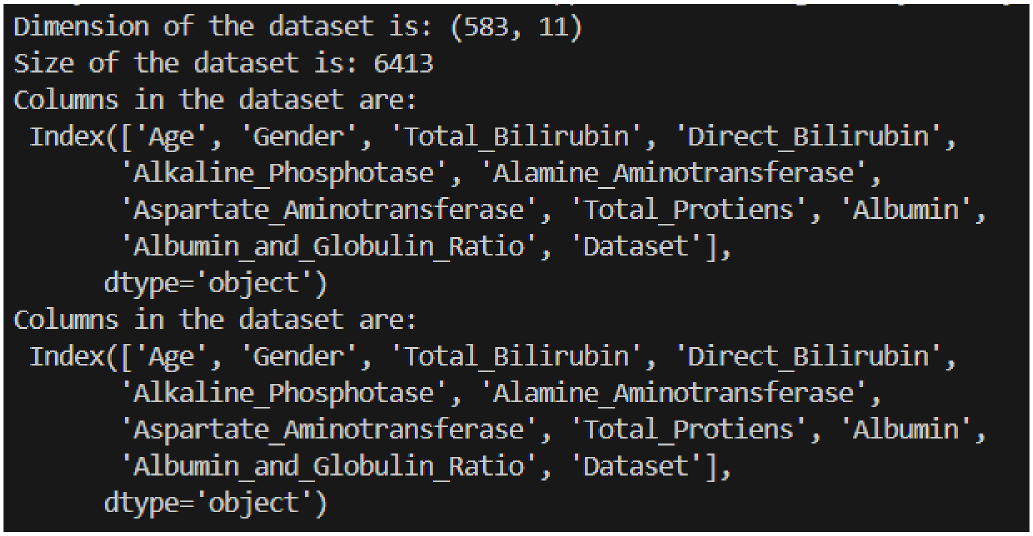
Functions for reading external files:

- **read_csv() -** helps to read a “csv” file
- **read_excel() –** helps to read an “excel” file
- **read_html() –** helps to read an “html” file
- **read_json() –** helps to read an “json” file
- **read_sql() –** helps to read “sql” file



```
#Program to determine characteristics of an existing dataset
#Importing pandas library
import pandas as pd
'''import os
print(os.getcwd())'''

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Determining dimension and size of the dataset
print("Dimension of the dataset is:", liver.shape)
print("Size of the dataset is:", liver.size)

#Determining columns of the dataset
print("Columns in the dataset are:\n", liver.keys())
print("Columns in the dataset are:\n", liver.columns)
```

Output:



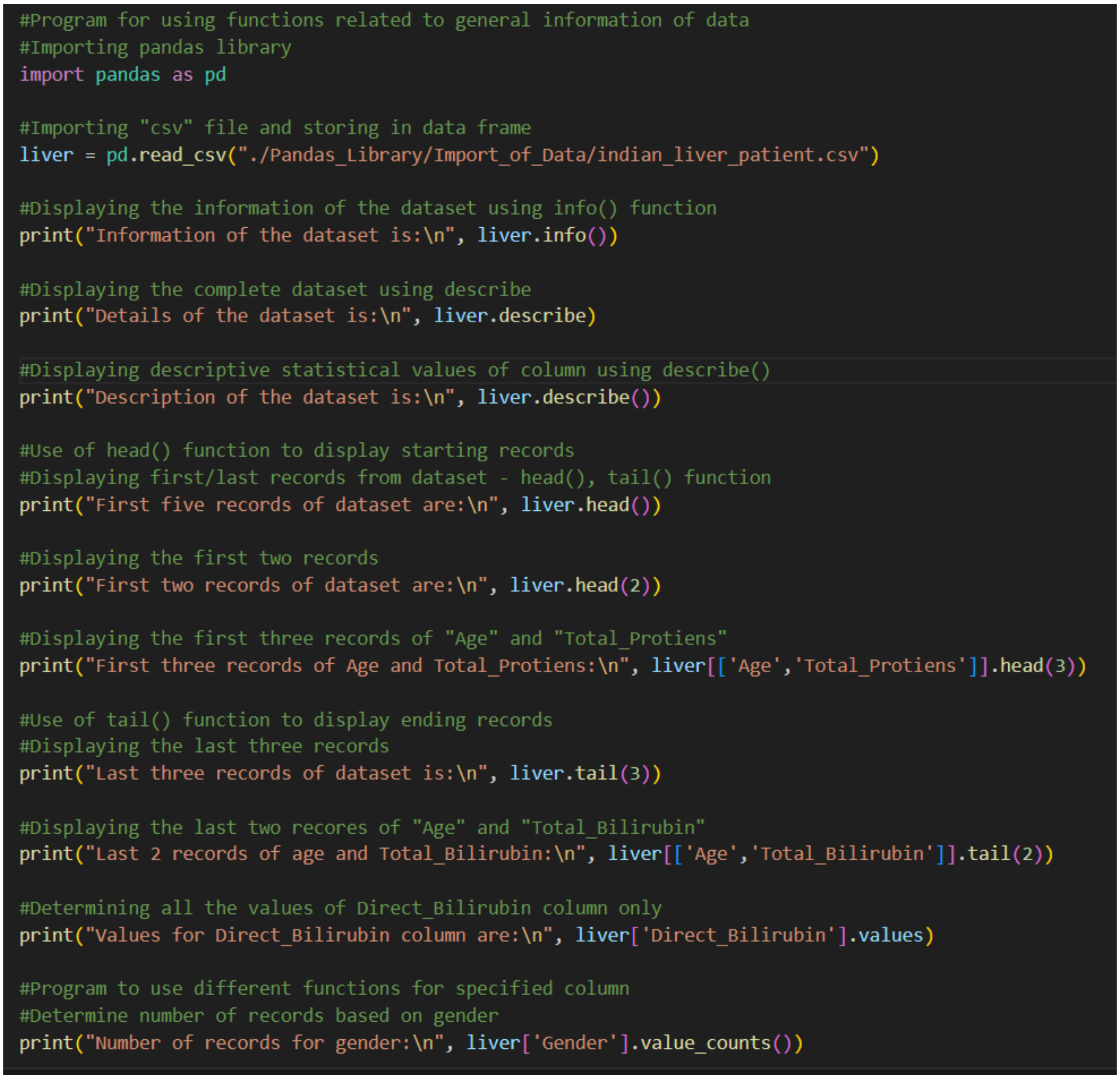
```
Dimension of the dataset is: (583, 11)
Size of the dataset is: 6413
Columns in the dataset are:
 Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
Columns in the dataset are:
 Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

Functions of Dataframe:

1. **Basic Information Fuctions:**
    - **info() –** displays complete information about the dataset
    - **describe –** prints the complete dataset
    - **describe() –** shows the information related to basic statistical values of all the columns of the dataset like count, mean, standard deviation, minimum, quartiles, and maximum value for each column
    - **head() -** display the first five records
    - **tails() –** display the last five records
    - **head(3) –** display the first three records
    - **tail(4) –** display the last four records

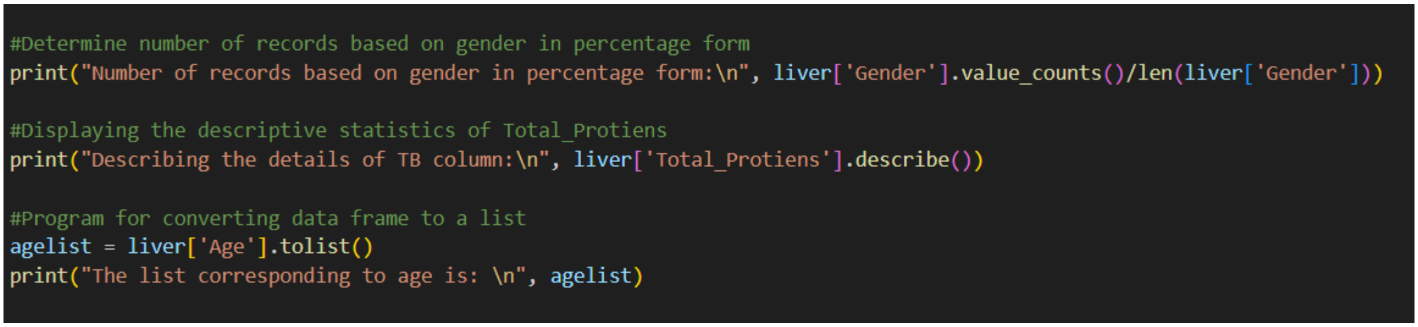
Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

- **variable_name[[‘column_name1’, ‘column_name2’]].head(3) –** give first three records of column_name1 and column_name2 only
- **variable_name[[‘column_name1’, ‘column_name2’]].tail(3) –** give last three records of column_name1 and column_name2 only
- **value_counts() –** count the number of records and display the descriptive statistics of a particular column
- **tolist() –** converts the data of a column into a list



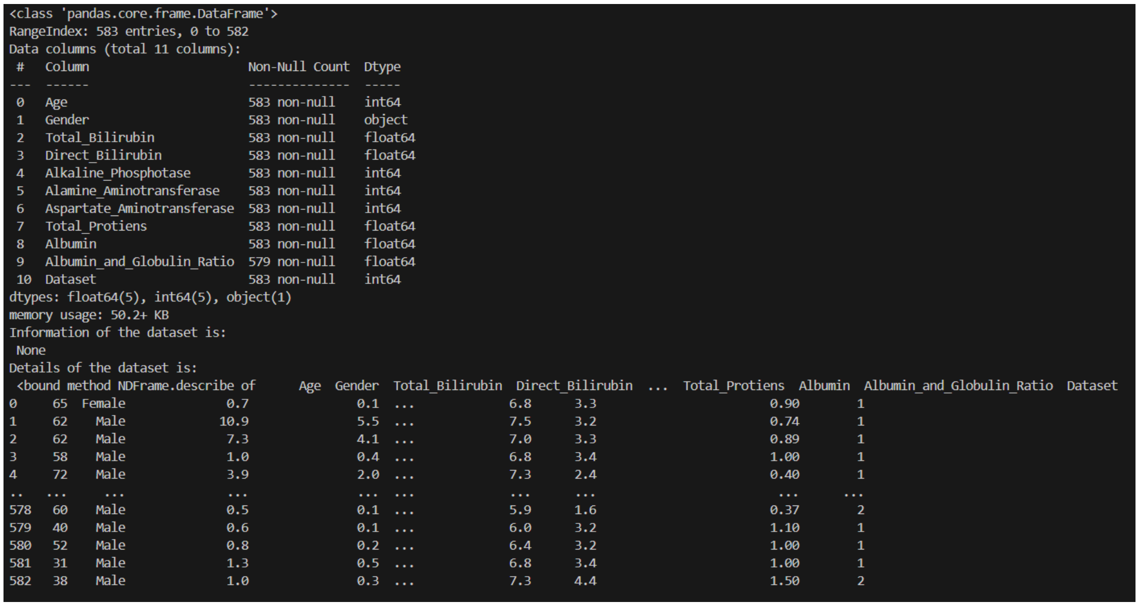
```python
#Program for using functions related to general information of data
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Displaying the information of the dataset using info() function
print("Information of the dataset is:\n", liver.info())

#Displaying the complete dataset using describe
print("Details of the dataset is:\n", liver.describe)

#Displaying descriptive statistical values of column using describe()
print("Description of the dataset is:\n", liver.describe())

#Use of head() function to display starting records
#Displaying first/last records from dataset - head(), tail() function
print("First five records of dataset are:\n", liver.head())

#Displaying the first two records
print("First two records of dataset are:\n", liver.head(2))

#Displaying the first three records of "Age" and "Total_Protiens"
print("First three records of Age and Total_Protiens:\n", liver[['Age','Total_Protiens']].head(3))

#Use of tail() function to display ending records
#Displaying the last three records
print("Last three records of dataset is:\n", liver.tail(3))

#Displaying the last two recores of "Age" and "Total_Bilirubin"
print("Last 2 records of age and Total_Bilirubin:\n", liver[['Age','Total_Bilirubin']].tail(2))

#Determining all the values of Direct_Bilirubin column only
print("Values for Direct_Bilirubin column are:\n", liver['Direct_Bilirubin'].values)

#Program to use different functions for specified column
#Determine number of records based on gender
print("Number of records for gender:\n", liver['Gender'].value_counts())
```

Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

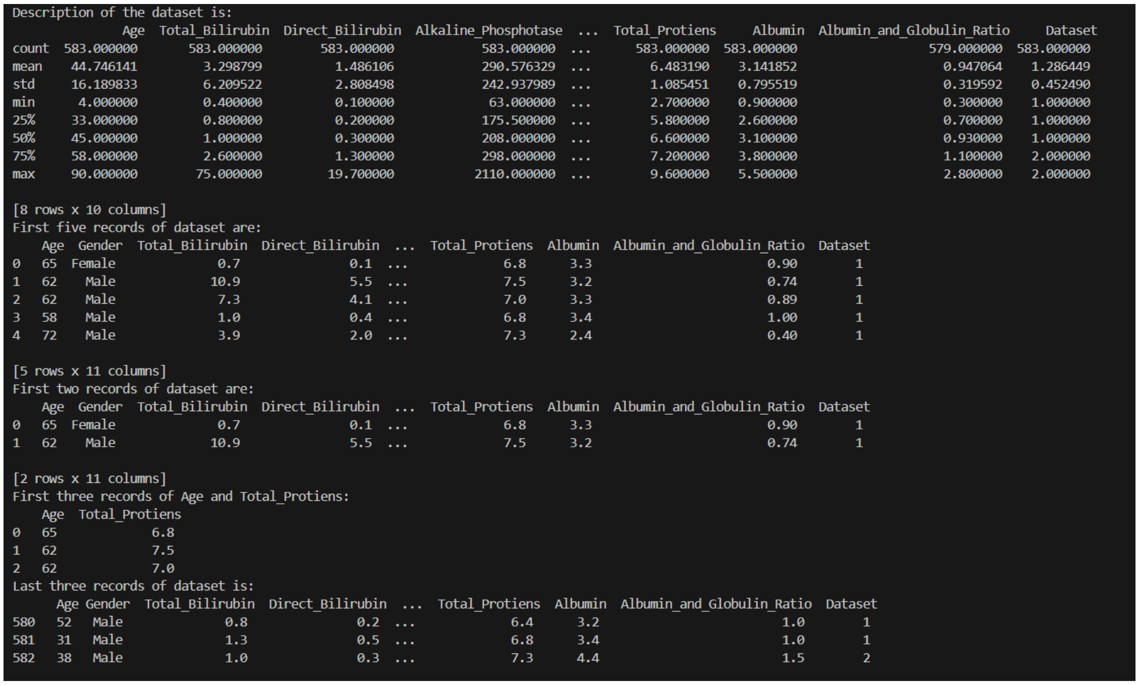


```
#Determine number of records based on gender in percentage form
print("Number of records based on gender in percentage form:\n", liver['Gender'].value_counts()/len(liver['Gender']))

#Displaying the descriptive statistics of Total_Protiens
print("Describing the details of TB column:\n", liver['Total_Protiens'].describe())

#Program for converting data frame to a list
agelist = liver['Age'].tolist()
print("The list corresponding to age is: \n", agelist)
```

Output:

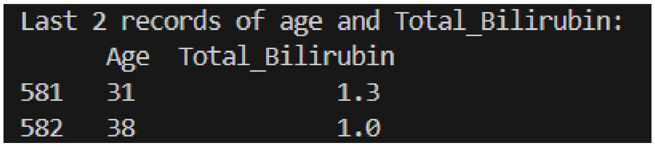


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
Information of the dataset is:
 None
Details of the dataset is:
 <bound method NDFrame.describe of      Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
0     65  Female              0.7               0.1  ...             6.8      3.3                        0.90        1
1     62    Male             10.9               5.5  ...             7.5      3.2                        0.74        1
2     62    Male              7.3               4.1  ...             7.0      3.3                        0.89        1
3     58    Male              1.0               0.4  ...             6.8      3.4                        1.00        1
4     72    Male              3.9               2.0  ...             7.3      2.4                        0.40        1
..   ...     ...              ...               ...  ...             ...      ...                         ...      ...
578   60    Male              0.5               0.1  ...             5.9      1.6                        0.37        2
579   40    Male              0.6               0.1  ...             6.0      3.2                        1.10        1
580   52    Male              0.8               0.2  ...             6.4      3.2                        1.00        1
581   31    Male              1.3               0.5  ...             6.8      3.4                        1.00        1
582   38    Male              1.0               0.3  ...             7.3      4.4                        1.50        2
```

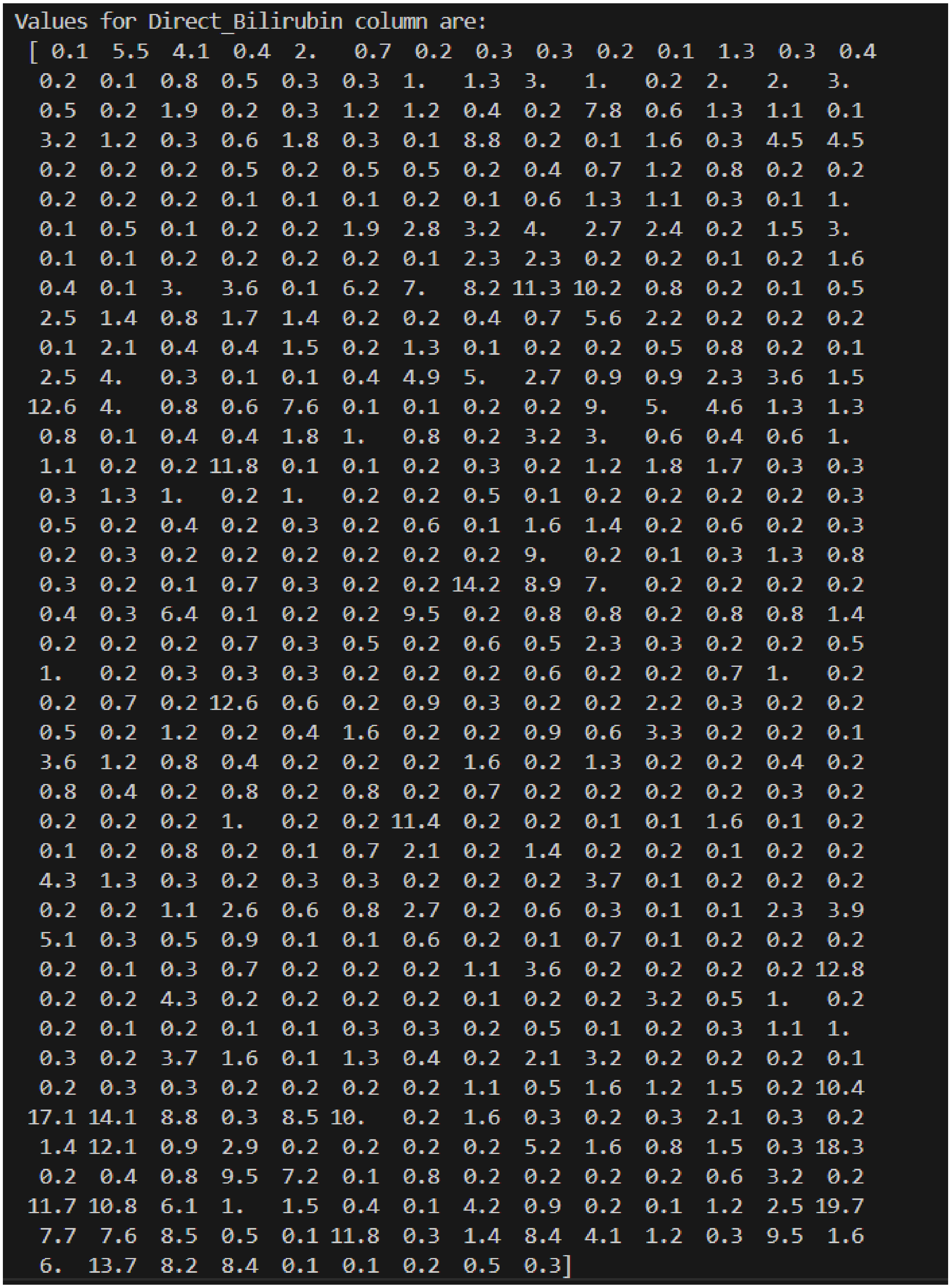


```
Description of the dataset is:
               Age  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  ...  Total_Protiens     Albumin  Albumin_and_Globulin_Ratio     Dataset
count  583.000000       583.000000        583.000000            583.000000  ...      583.000000  583.000000                  579.000000  583.000000
mean    44.746141         3.298799          1.486106            290.576329  ...        6.483190    3.141852                    0.947064    1.286449
std     16.189833         6.209522          2.808498            242.937989  ...        1.085451    0.795519                    0.319592    0.452490
min      4.000000         0.400000          0.100000             63.000000  ...        2.700000    0.900000                    0.300000    1.000000
25%     33.000000         0.800000          0.200000            175.500000  ...        5.800000    2.600000                    0.700000    1.000000
50%     45.000000         1.000000          0.300000            208.000000  ...        6.600000    3.100000                    0.930000    1.000000
75%     58.000000         2.600000          1.300000            298.000000  ...        7.200000    3.800000                    1.100000    2.000000
max     90.000000        75.000000         19.700000           2110.000000  ...        9.600000    5.500000                    2.800000    2.000000

[8 rows x 10 columns]
First five records of dataset are:
    Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
0   65  Female              0.7               0.1  ...             6.8      3.3                        0.90        1
1   62    Male             10.9               5.5  ...             7.5      3.2                        0.74        1
2   62    Male              7.3               4.1  ...             7.0      3.3                        0.89        1
3   58    Male              1.0               0.4  ...             6.8      3.4                        1.00        1
4   72    Male              3.9               2.0  ...             7.3      2.4                        0.40        1

[5 rows x 11 columns]
First two records of dataset are:
    Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
0   65  Female              0.7               0.1  ...             6.8      3.3                        0.90        1
1   62    Male             10.9               5.5  ...             7.5      3.2                        0.74        1

[2 rows x 11 columns]
First three records of Age and Total_Protiens:
    Age  Total_Protiens
0   65             6.8
1   62             7.5
2   62             7.0
Last three records of dataset is:
      Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
580   52   Male              0.8               0.2  ...             6.4      3.2                         1.0        1
581   31   Male              1.3               0.5  ...             6.8      3.4                         1.0        1
582   38   Male              1.0               0.3  ...             7.3      4.4                         1.5        2
```

Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

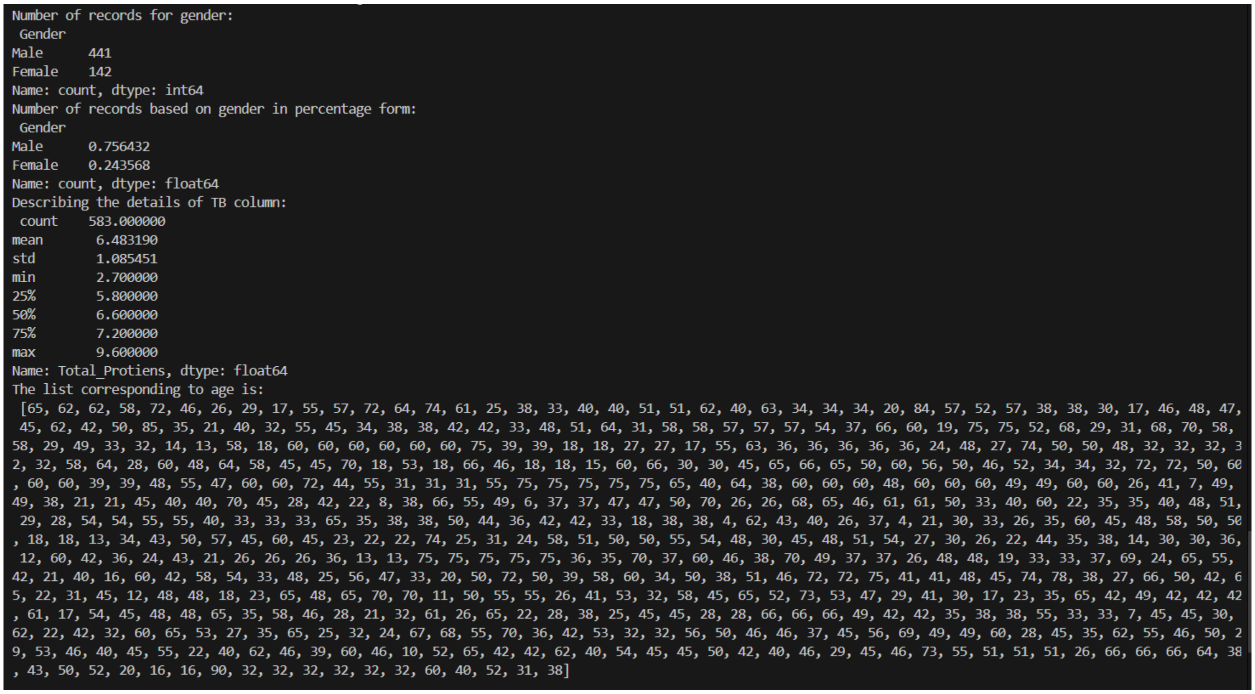


```
Last 2 records of age and Total_Bilirubin:
     Age  Total_Bilirubin
581   31              1.3
582   38              1.0
```



```
Values for Direct_Bilirubin column are:
 [ 0.1  5.5  4.1  0.4  2.   0.7  0.2  0.3  0.3  0.2  0.1  1.3  0.3  0.4
  0.2  0.1  0.8  0.5  0.3  0.3  1.   1.3  3.   1.   0.2  2.   2.   3.
  0.5  0.2  1.9  0.2  0.3  1.2  1.2  0.4  0.2  7.8  0.6  1.3  1.1  0.1
  3.2  1.2  0.3  0.6  1.8  0.3  0.1  8.8  0.2  0.1  1.6  0.3  4.5  4.5
  0.2  0.2  0.2  0.5  0.2  0.5  0.5  0.2  0.4  0.7  1.2  0.8  0.2  0.2
  0.2  0.2  0.2  0.1  0.1  0.1  0.2  0.1  0.6  1.3  1.1  0.3  0.1  1.
  0.1  0.5  0.1  0.2  0.2  1.9  2.8  3.2  4.   2.7  2.4  0.2  1.5  3.
  0.1  0.1  0.2  0.2  0.2  0.2  0.1  2.3  2.3  0.2  0.2  0.1  0.2  1.6
  0.4  0.1  3.   3.6  0.1  6.2  7.   8.2 11.3 10.2  0.8  0.2  0.1  0.5
  2.5  1.4  0.8  1.7  1.4  0.2  0.2  0.4  0.7  5.6  2.2  0.2  0.2  0.2
  0.1  2.1  0.4  0.4  1.5  0.2  1.3  0.1  0.2  0.2  0.5  0.8  0.2  0.1
  2.5  4.   0.3  0.1  0.1  0.4  4.9  5.   2.7  0.9  0.9  2.3  3.6  1.5
 12.6  4.   0.8  0.6  7.6  0.1  0.1  0.2  0.2  9.   5.   4.6  1.3  1.3
  0.8  0.1  0.4  0.4  1.8  1.   0.8  0.2  3.2  3.   0.6  0.4  0.6  1.
  1.1  0.2  0.2 11.8  0.1  0.1  0.2  0.3  0.2  1.2  1.8  1.7  0.3  0.3
  0.3  1.3  1.   0.2  1.   0.2  0.2  0.5  0.1  0.2  0.2  0.2  0.2  0.3
  0.5  0.2  0.4  0.2  0.3  0.2  0.6  0.1  1.6  1.4  0.2  0.6  0.2  0.3
  0.2  0.3  0.2  0.2  0.2  0.2  0.2  0.2  9.   0.2  0.1  0.3  1.3  0.8
  0.3  0.2  0.1  0.7  0.3  0.2  0.2 14.2  8.9  7.   0.2  0.2  0.2  0.2
  0.4  0.3  6.4  0.1  0.2  0.2  9.5  0.2  0.8  0.8  0.2  0.8  0.8  1.4
  0.2  0.2  0.2  0.7  0.3  0.5  0.2  0.6  0.5  2.3  0.3  0.2  0.2  0.5
  1.   0.2  0.3  0.3  0.3  0.2  0.2  0.2  0.6  0.2  0.2  0.7  1.   0.2
  0.2  0.7  0.2 12.6  0.6  0.2  0.9  0.3  0.2  0.2  2.2  0.3  0.2  0.2
  0.5  0.2  1.2  0.2  0.4  1.6  0.2  0.2  0.9  0.6  3.3  0.2  0.2  0.1
  3.6  1.2  0.8  0.4  0.2  0.2  0.2  1.6  0.2  1.3  0.2  0.2  0.4  0.2
  0.8  0.4  0.2  0.8  0.2  0.8  0.2  0.7  0.2  0.2  0.2  0.2  0.3  0.2
  0.2  0.2  0.2  1.   0.2  0.2 11.4  0.2  0.2  0.1  0.1  1.6  0.1  0.2
  0.1  0.2  0.8  0.2  0.1  0.7  2.1  0.2  1.4  0.2  0.2  0.1  0.2  0.2
  4.3  1.3  0.3  0.2  0.3  0.3  0.2  0.2  0.2  3.7  0.1  0.2  0.2  0.2
  0.2  0.2  1.1  2.6  0.6  0.8  2.7  0.2  0.6  0.3  0.1  0.1  2.3  3.9
  5.1  0.3  0.5  0.9  0.1  0.1  0.6  0.2  0.1  0.7  0.1  0.2  0.2  0.2
  0.2  0.1  0.3  0.7  0.2  0.2  0.2  1.1  3.6  0.2  0.2  0.2  0.2 12.8
  0.2  0.2  4.3  0.2  0.2  0.2  0.2  0.1  0.2  0.2  3.2  0.5  1.   0.2
  0.2  0.1  0.2  0.1  0.1  0.3  0.3  0.2  0.5  0.1  0.2  0.3  1.1  1.
  0.3  0.2  3.7  1.6  0.1  1.3  0.4  0.2  2.1  3.2  0.2  0.2  0.2  0.1
  0.2  0.3  0.3  0.2  0.2  0.2  0.2  1.1  0.5  1.6  1.2  1.5  0.2 10.4
 17.1 14.1  8.8  0.3  8.5 10.   0.2  1.6  0.3  0.2  0.3  2.1  0.3  0.2
  1.4 12.1  0.9  2.9  0.2  0.2  0.2  0.2  5.2  1.6  0.8  1.5  0.3 18.3
  0.2  0.4  0.8  9.5  7.2  0.1  0.8  0.2  0.2  0.2  0.2  0.6  3.2  0.2
 11.7 10.8  6.1  1.   1.5  0.4  0.1  4.2  0.9  0.2  0.1  1.2  2.5 19.7
  7.7  7.6  8.5  0.5  0.1 11.8  0.3  1.4  8.4  4.1  1.2  0.3  9.5  1.6
  6.  13.7  8.2  8.4  0.1  0.1  0.2  0.5  0.3]
```

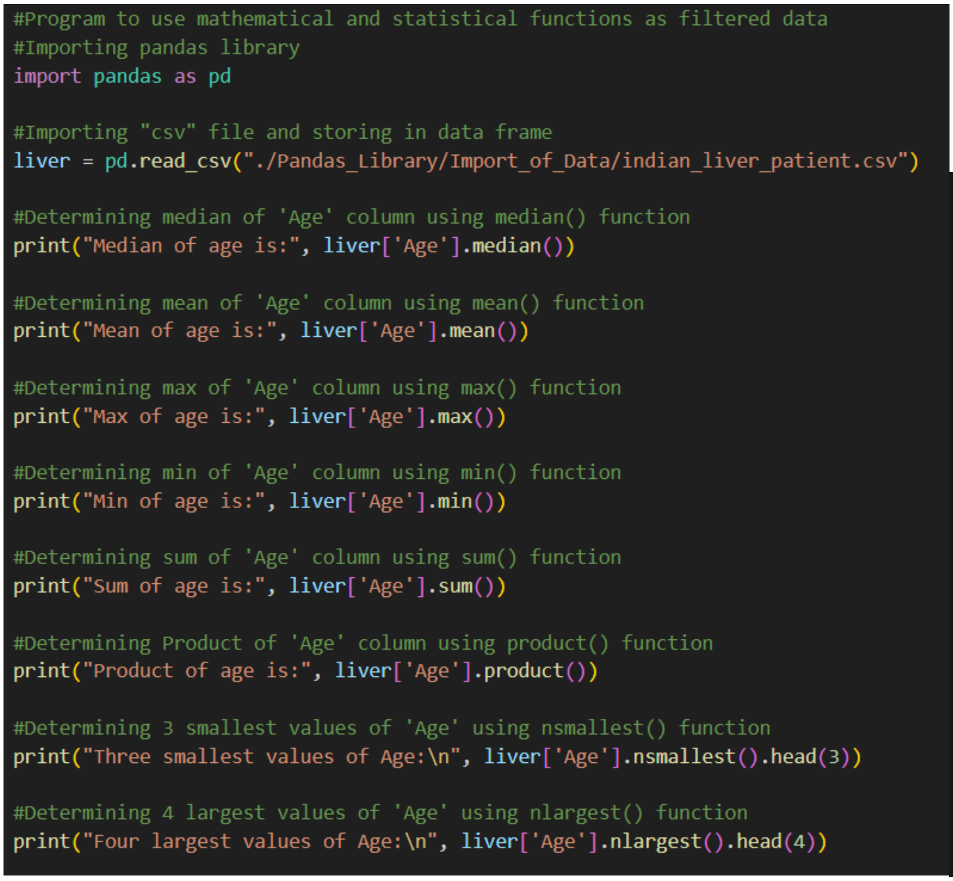
Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main



```
Number of records for gender:
 Gender
Male      441
Female    142
Name: count, dtype: int64
Number of records based on gender in percentage form:
 Gender
Male      0.756432
Female    0.243568
Name: count, dtype: float64
Describing the details of TB column:
 count    583.000000
mean       6.483190
std        1.085451
min        2.700000
25%        5.800000
50%        6.600000
75%        7.200000
max        9.600000
Name: Total_Protiens, dtype: float64
The list corresponding to age is:
 [65, 62, 62, 58, 72, 46, 26, 29, 17, 55, 57, 72, 64, 74, 61, 25, 38, 33, 40, 40, 51, 51, 62, 40, 63, 34, 34, 34, 20, 84, 57, 52, 57, 38, 38, 30, 17, 46, 48, 47,
 45, 62, 42, 50, 85, 35, 21, 40, 32, 55, 45, 34, 38, 38, 42, 42, 33, 48, 51, 64, 31, 58, 58, 57, 57, 57, 54, 37, 66, 60, 19, 75, 75, 52, 68, 29, 31, 68, 70, 58,
58, 29, 49, 33, 32, 14, 13, 58, 18, 60, 60, 60, 60, 60, 60, 75, 39, 39, 18, 18, 27, 27, 17, 55, 63, 36, 36, 36, 36, 36, 24, 48, 27, 74, 50, 50, 48, 32, 32, 32, 3
2, 32, 58, 64, 28, 60, 48, 64, 58, 45, 45, 70, 18, 53, 18, 66, 46, 18, 18, 15, 60, 66, 30, 30, 45, 65, 66, 65, 50, 60, 56, 50, 46, 52, 34, 34, 32, 72, 72, 50, 60
, 60, 60, 39, 39, 48, 55, 47, 60, 60, 72, 44, 55, 31, 31, 31, 55, 75, 75, 75, 75, 75, 65, 40, 64, 38, 60, 60, 60, 48, 60, 60, 60, 49, 49, 60, 60, 26, 41, 7, 49,
49, 38, 21, 21, 45, 40, 40, 70, 45, 28, 42, 22, 8, 38, 66, 55, 49, 6, 37, 37, 47, 47, 50, 70, 26, 26, 68, 65, 46, 61, 61, 50, 33, 40, 60, 22, 35, 35, 40, 48, 51,
 29, 28, 54, 54, 55, 55, 40, 33, 33, 33, 65, 35, 38, 38, 50, 44, 36, 42, 42, 33, 18, 38, 38, 4, 62, 43, 40, 26, 37, 4, 21, 30, 33, 26, 35, 60, 45, 48, 58, 50, 50
, 18, 18, 13, 34, 43, 50, 57, 45, 60, 45, 23, 22, 22, 74, 25, 31, 24, 58, 51, 50, 50, 55, 54, 48, 30, 45, 48, 51, 54, 27, 30, 26, 22, 44, 35, 38, 14, 30, 30, 36,
 12, 60, 42, 36, 24, 43, 21, 26, 26, 26, 36, 13, 13, 75, 75, 75, 75, 75, 36, 35, 70, 37, 60, 46, 38, 70, 49, 37, 37, 26, 48, 48, 19, 33, 33, 37, 69, 24, 65, 55,
42, 21, 40, 16, 60, 42, 58, 54, 33, 48, 25, 56, 47, 33, 20, 50, 72, 50, 39, 58, 60, 34, 50, 38, 51, 46, 72, 72, 75, 41, 41, 48, 45, 74, 78, 38, 27, 66, 50, 42, 6
5, 22, 31, 45, 12, 48, 48, 18, 23, 65, 48, 65, 70, 70, 11, 50, 55, 55, 26, 41, 53, 32, 58, 45, 65, 52, 73, 53, 47, 29, 41, 30, 17, 23, 35, 65, 42, 49, 42, 42, 42
, 61, 17, 54, 45, 48, 48, 65, 35, 58, 46, 28, 21, 32, 61, 26, 65, 22, 28, 38, 25, 45, 45, 28, 28, 66, 66, 66, 49, 42, 42, 35, 38, 38, 55, 33, 33, 7, 45, 45, 30,
62, 22, 42, 32, 60, 65, 53, 27, 35, 65, 25, 32, 24, 67, 68, 55, 70, 36, 42, 53, 32, 32, 56, 50, 46, 46, 37, 45, 56, 69, 49, 49, 60, 28, 45, 35, 62, 55, 46, 50, 2
9, 53, 46, 40, 45, 55, 22, 40, 62, 46, 39, 60, 46, 10, 52, 65, 42, 42, 62, 40, 54, 45, 45, 50, 42, 40, 46, 29, 45, 46, 73, 55, 51, 51, 51, 26, 66, 66, 66, 64, 38
, 43, 50, 52, 20, 16, 16, 90, 32, 32, 32, 32, 32, 32, 60, 40, 52, 31, 38]
```

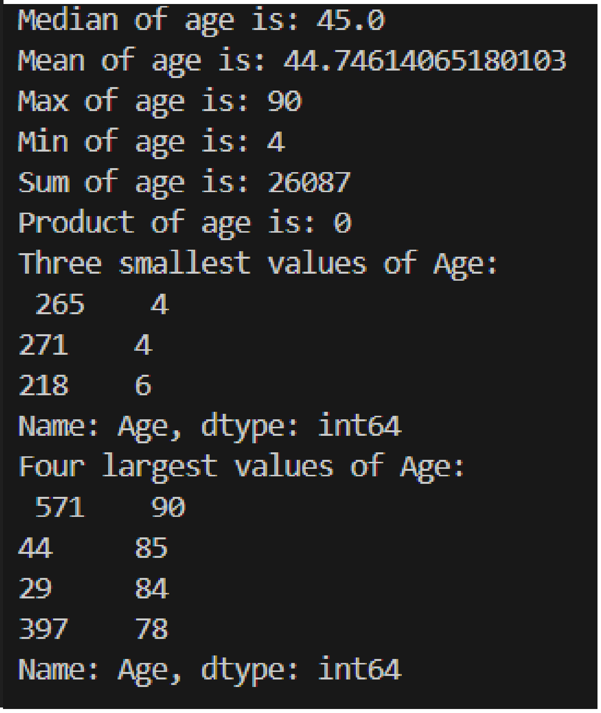
2. Mathematical and Statistical Functins:

- **mean() :** display the mean of the dataset
- **median():** display the median of the dataset
- **max():** display the max of the dataset
- **min():** display the min of the dataset
- **sum():** display the sum of the values
- **product():** display the product of the values
- **nsmallest():** display the nsmallest values in the increasing order
- **nlargest():** display the nlargest values in decreasing order



```
#Program to use mathematical and statistical functions as filtered data
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Determining median of 'Age' column using median() function
print("Median of age is:", liver['Age'].median())

#Determining mean of 'Age' column using mean() function
print("Mean of age is:", liver['Age'].mean())

#Determining max of 'Age' column using max() function
print("Max of age is:", liver['Age'].max())

#Determining min of 'Age' column using min() function
print("Min of age is:", liver['Age'].min())

#Determining sum of 'Age' column using sum() function
print("Sum of age is:", liver['Age'].sum())

#Determining Product of 'Age' column using product() function
print("Product of age is:", liver['Age'].product())

#Determining 3 smallest values of 'Age' using nsmallest() function
print("Three smallest values of Age:\n", liver['Age'].nsmallest().head(3))

#Determining 4 largest values of 'Age' using nlargest() function
print("Four largest values of Age:\n", liver['Age'].nlargest().head(4))
```

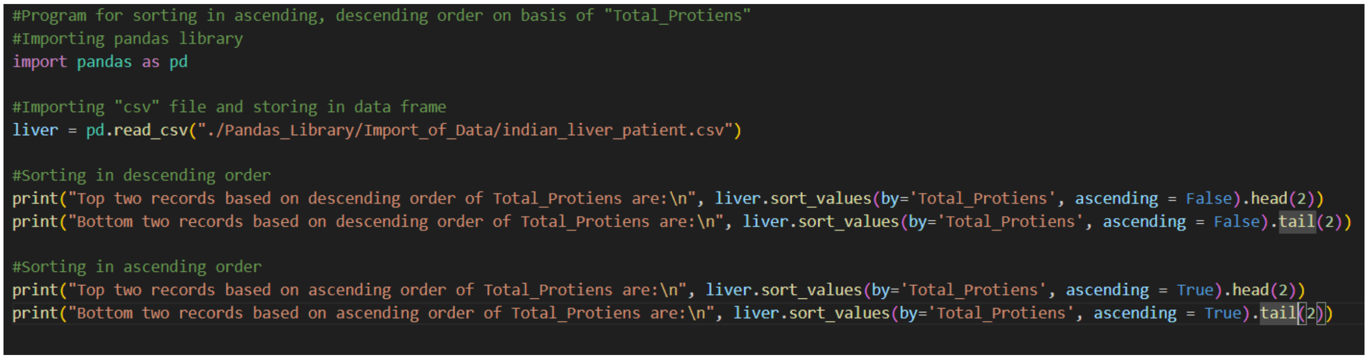
Output:



```
Median of age is: 45.0
Mean of age is: 44.74614065180103
Max of age is: 90
Min of age is: 4
Sum of age is: 26087
Product of age is: 0
Three smallest values of Age:
 265    4
271    4
218    6
Name: Age, dtype: int64
Four largest values of Age:
 571    90
44     85
29     84
397    78
Name: Age, dtype: int64
```

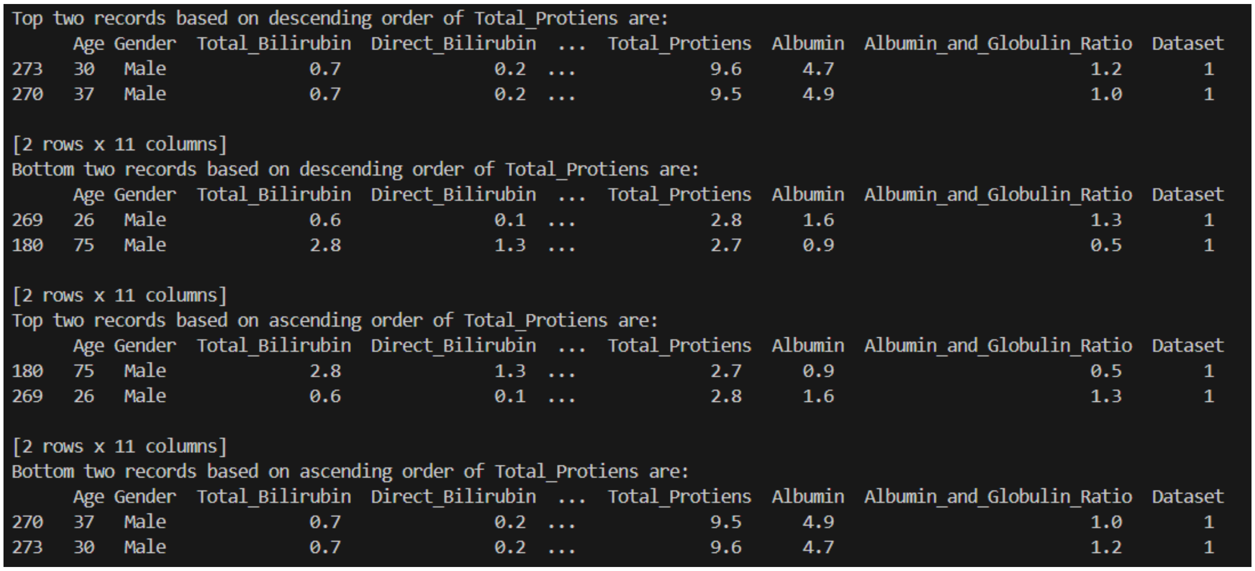
3. Sort Functions

- **sort_values(by= "column_name", ascending = false):** makes the dataset in the descending order
- **sort_values(by= "column_name", ascending = True):** makes the dataset in the ascending order



```
#Program for sorting in ascending, descending order on basis of "Total_Protiens"
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Sorting in descending order
print("Top two records based on descending order of Total_Protiens are:\n", liver.sort_values(by='Total_Protiens', ascending = False).head(2))
print("Bottom two records based on descending order of Total_Protiens are:\n", liver.sort_values(by='Total_Protiens', ascending = False).tail(2))

#Sorting in ascending order
print("Top two records based on ascending order of Total_Protiens are:\n", liver.sort_values(by='Total_Protiens', ascending = True).head(2))
print("Bottom two records based on ascending order of Total_Protiens are:\n", liver.sort_values(by='Total_Protiens', ascending = True).tail(2))
```

Output:

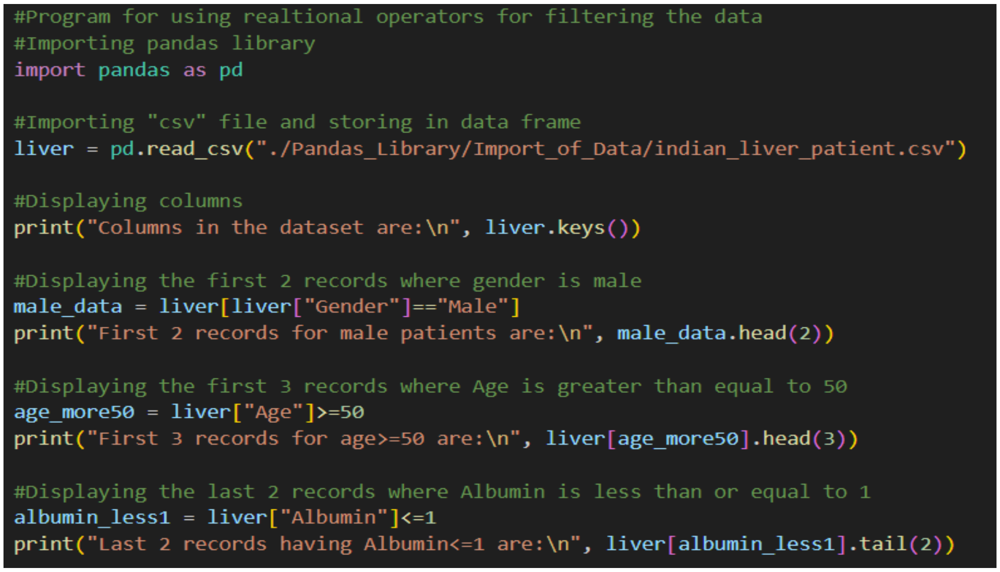


```
Top two records based on descending order of Total_Protiens are:
     Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
273   30   Male              0.7               0.2  ...             9.6      4.7                         1.2        1
270   37   Male              0.7               0.2  ...             9.5      4.9                         1.0        1

[2 rows x 11 columns]
Bottom two records based on descending order of Total_Protiens are:
     Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
269   26   Male              0.6               0.1  ...             2.8      1.6                         1.3        1
180   75   Male              2.8               1.3  ...             2.7      0.9                         0.5        1

[2 rows x 11 columns]
Top two records based on ascending order of Total_Protiens are:
     Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
180   75   Male              2.8               1.3  ...             2.7      0.9                         0.5        1
269   26   Male              0.6               0.1  ...             2.8      1.6                         1.3        1

[2 rows x 11 columns]
Bottom two records based on ascending order of Total_Protiens are:
     Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
270   37   Male              0.7               0.2  ...             9.5      4.9                         1.0        1
273   30   Male              0.7               0.2  ...             9.6      4.7                         1.2        1
```

Data Extraction:

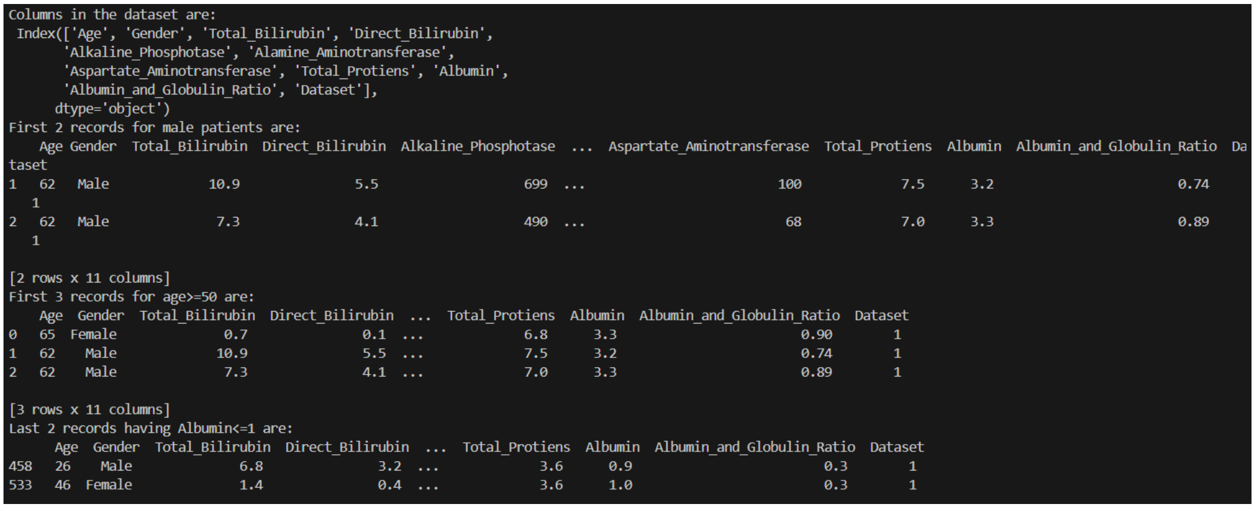
- Data extraction according to the user requirement is an important task and is done at a great level for performing data analysis
- Different relational operators such as <,>,==,<=,>=,!= etc. can be used to create conditions.
- Logical operators such as **and (&)** and **or (|)** help to filter the data on the basis of multiple conditions.
- The use of indexers such as loc and iloc also contribute a lot for extracting data according to the user requirement

1. Using Relational Operators



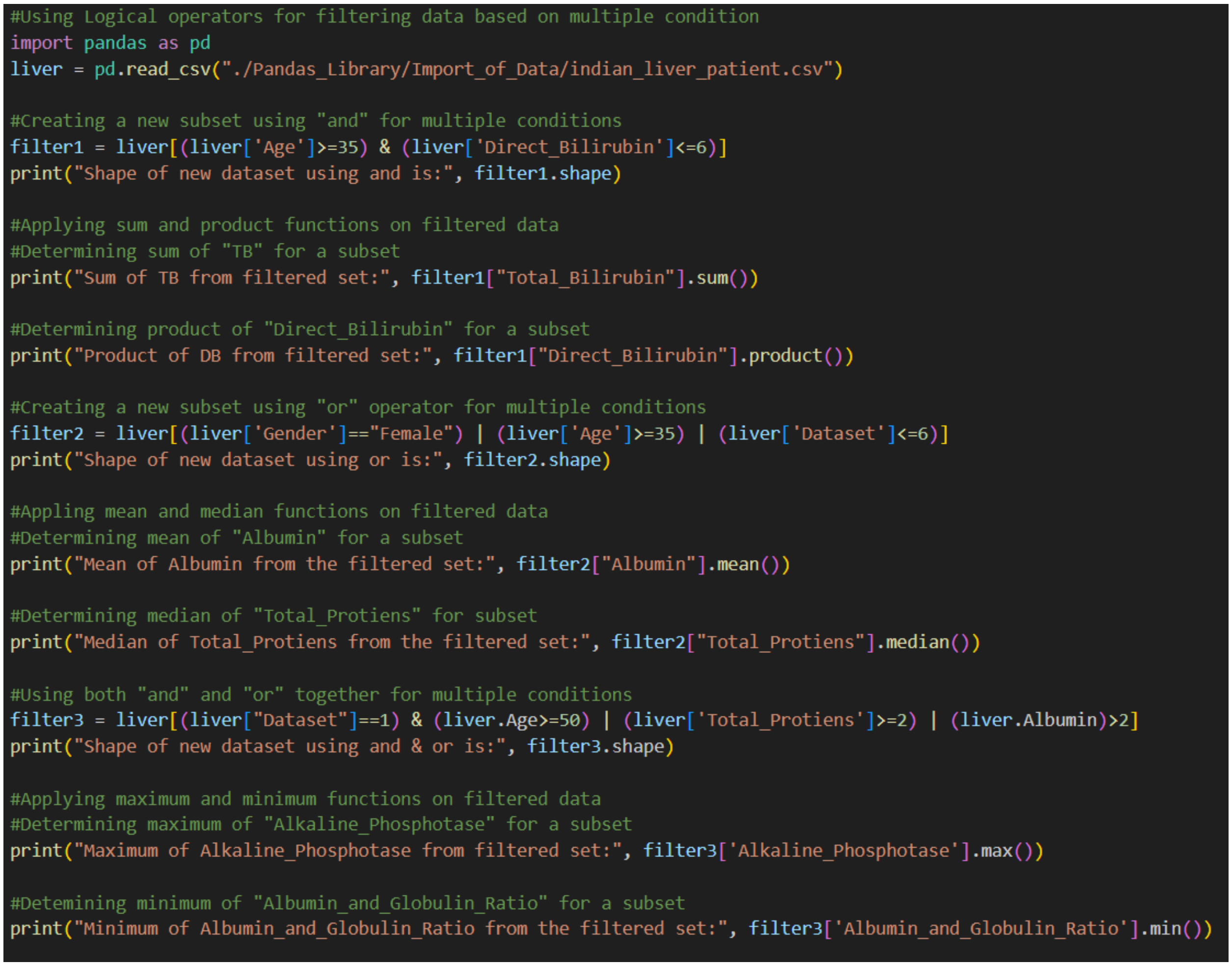
```python
#Program for using realtional operators for filtering the data
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Displaying columns
print("Columns in the dataset are:\n", liver.keys())

#Displaying the first 2 records where gender is male
male_data = liver[liver["Gender"]=="Male"]
print("First 2 records for male patients are:\n", male_data.head(2))

#Displaying the first 3 records where Age is greater than equal to 50
age_more50 = liver["Age"]>=50
print("First 3 records for age>=50 are:\n", liver[age_more50].head(3))

#Displaying the last 2 records where Albumin is less than or equal to 1
albumin_less1 = liver["Albumin"]<=1
print("Last 2 records having Albumin<=1 are:\n", liver[albumin_less1].tail(2))
```

Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

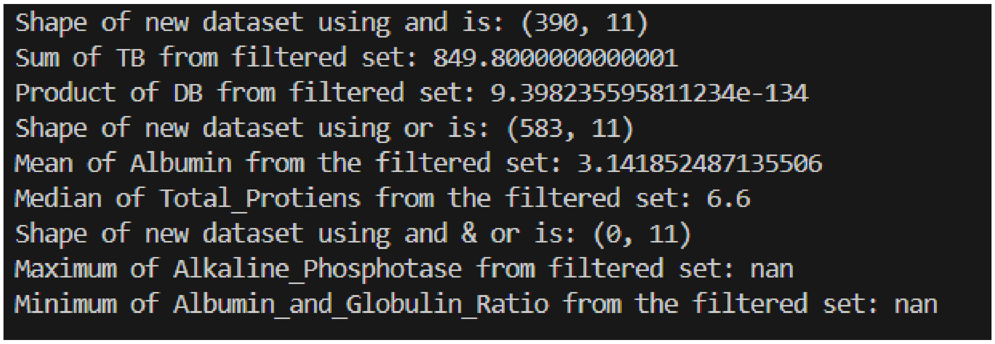


```
Columns in the dataset are:
 Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
First 2 records for male patients are:
   Age Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  ...  Aspartate_Aminotransferase  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Da
taset
1   62   Male             10.9               5.5                   699  ...                         100             7.5      3.2                        0.74
   1
2   62   Male              7.3               4.1                   490  ...                          68             7.0      3.3                        0.89
   1

[2 rows x 11 columns]
First 3 records for age>=50 are:
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
0   65  Female              0.7               0.1  ...             6.8      3.3                        0.90        1
1   62    Male             10.9               5.5  ...             7.5      3.2                        0.74        1
2   62    Male              7.3               4.1  ...             7.0      3.3                        0.89        1

[3 rows x 11 columns]
Last 2 records having Albumin<=1 are:
     Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
458   26    Male              6.8               3.2  ...             3.6      0.9                         0.3        1
533   46  Female              1.4               0.4  ...             3.6      1.0                         0.3        1
```

2. Using Logical Operators

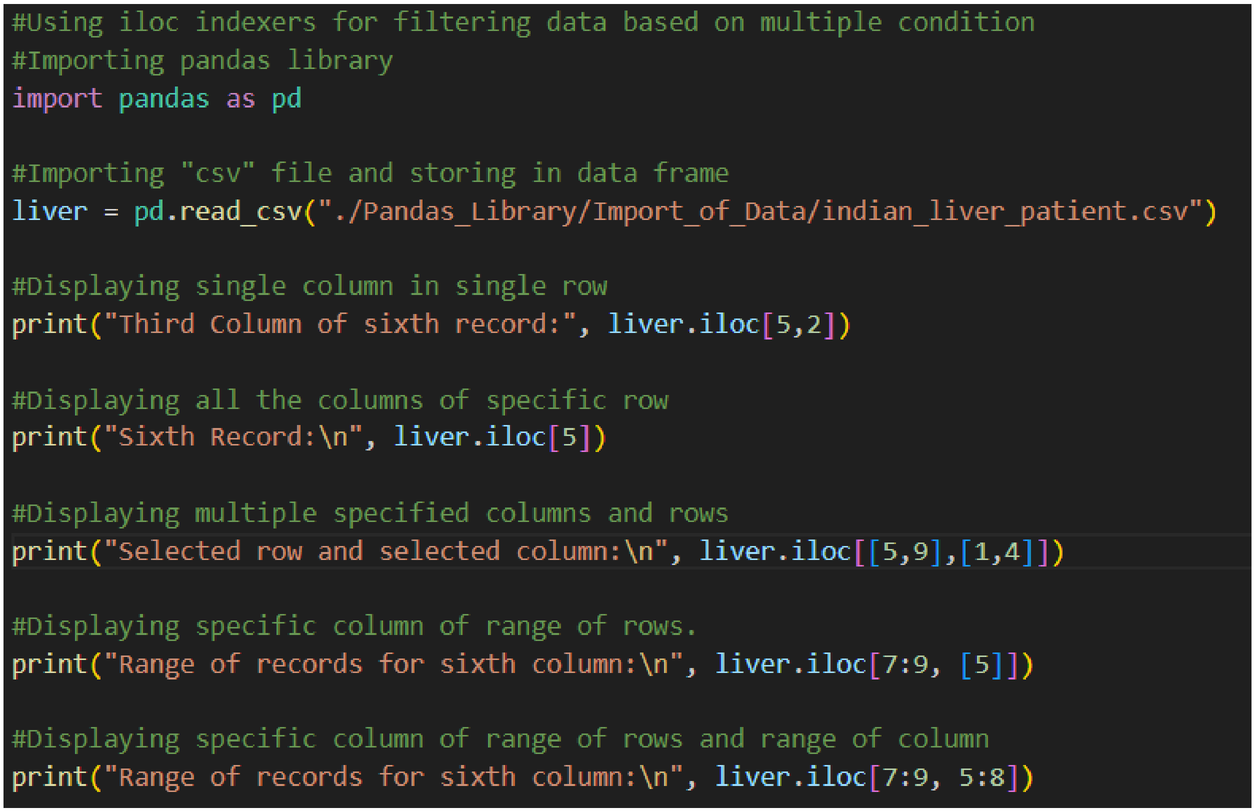


```python
#Using Logical operators for filtering data based on multiple condition
import pandas as pd
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Creating a new subset using "and" for multiple conditions
filter1 = liver[(liver['Age']>=35) & (liver['Direct_Bilirubin']<=6)]
print("Shape of new dataset using and is:", filter1.shape)

#Applying sum and product functions on filtered data
#Determining sum of "TB" for a subset
print("Sum of TB from filtered set:", filter1["Total_Bilirubin"].sum())

#Determining product of "Direct_Bilirubin" for a subset
print("Product of DB from filtered set:", filter1["Direct_Bilirubin"].product())

#Creating a new subset using "or" operator for multiple conditions
filter2 = liver[(liver['Gender']=="Female") | (liver['Age']>=35) | (liver['Dataset']<=6)]
print("Shape of new dataset using or is:", filter2.shape)

#Appling mean and median functions on filtered data
#Determining mean of "Albumin" for a subset
print("Mean of Albumin from the filtered set:", filter2["Albumin"].mean())

#Determining median of "Total_Protiens" for subset
print("Median of Total_Protiens from the filtered set:", filter2["Total_Protiens"].median())

#Using both "and" and "or" together for multiple conditions
filter3 = liver[(liver["Dataset"]==1) & (liver.Age>=50) | (liver['Total_Protiens']>=2) | (liver.Albumin)>2]
print("Shape of new dataset using and & or is:", filter3.shape)

#Applying maximum and minimum functions on filtered data
#Determining maximum of "Alkaline_Phosphotase" for a subset
print("Maximum of Alkaline_Phosphotase from filtered set:", filter3['Alkaline_Phosphotase'].max())

#Detemining minimum of "Albumin_and_Globulin_Ratio" for a subset
print("Minimum of Albumin_and_Globulin_Ratio from the filtered set:", filter3['Albumin_and_Globulin_Ratio'].min())
```



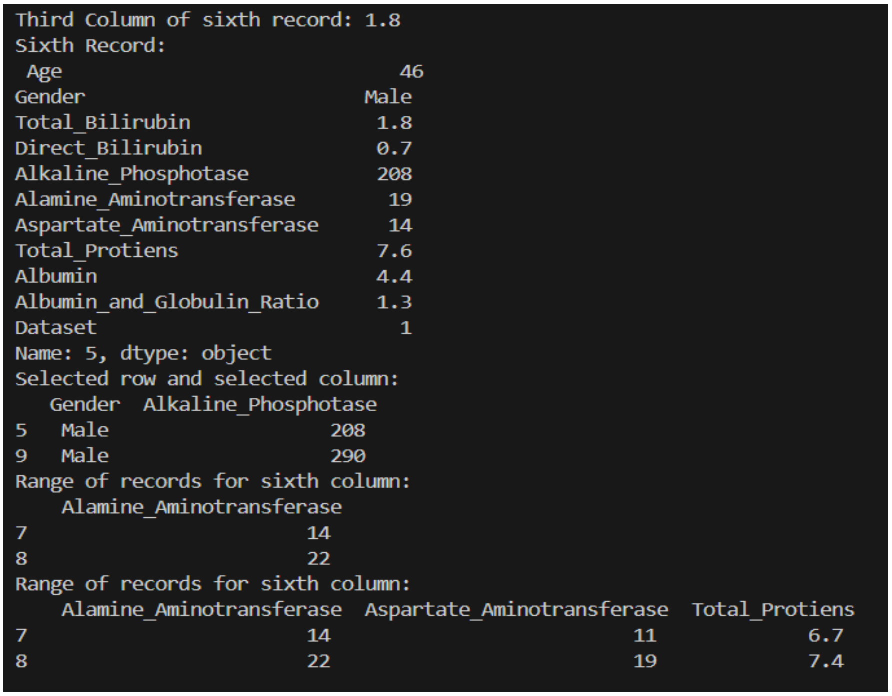
```
Shape of new dataset using and is: (390, 11)
Sum of TB from filtered set: 849.8000000000001
Product of DB from filtered set: 9.398235595811234e-134
Shape of new dataset using or is: (583, 11)
Mean of Albumin from the filtered set: 3.141852487135506
Median of Total_Protiens from the filtered set: 6.6
Shape of new dataset using and & or is: (0, 11)
Maximum of Alkaline_Phosphotase from filtered set: nan
Minimum of Albumin_and_Globulin_Ratio from the filtered set: nan
```

3. **Using iloc Indexers:**
   - These indexers play a major role in the data extraction on the basis of specified row and column.
   - The iloc indexer helps to extract particular row(s) and column(s) at specified numbers in the order that they appear in the dataframe.
   - **Syntax: iloc[row_index,col_index]**
     Or
     iloc[[row_index1,row_index2,..],[col_index1,col_index2,…]]
     Or
     iloc[row_index_range, col_index_range]



```
#Using iloc indexers for filtering data based on multiple condition
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Displaying single column in single row
print("Third Column of sixth record:", liver.iloc[5,2])

#Displaying all the columns of specific row
print("Sixth Record:\n", liver.iloc[5])

#Displaying multiple specified columns and rows
print("Selected row and selected column:\n", liver.iloc[[5,9],[1,4]])

#Displaying specific column of range of rows.
print("Range of records for sixth column:\n", liver.iloc[7:9, [5]])

#Displaying specific column of range of rows and range of column
print("Range of records for sixth column:\n", liver.iloc[7:9, 5:8])
```

Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main



```
Third Column of sixth record: 1.8
Sixth Record:
 Age                             46
Gender                        Male
Total_Bilirubin                1.8
Direct_Bilirubin               0.7
Alkaline_Phosphotase           208
Alamine_Aminotransferase        19
Aspartate_Aminotransferase      14
Total_Protiens                 7.6
Albumin                        4.4
Albumin_and_Globulin_Ratio     1.3
Dataset                          1
Name: 5, dtype: object
Selected row and selected column:
   Gender  Alkaline_Phosphotase
5   Male                   208
9   Male                   290
Range of records for sixth column:
    Alamine_Aminotransferase
7                         14
8                         22
Range of records for sixth column:
    Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens
7                         14                          11             6.7
8                         22                          19             7.4
```

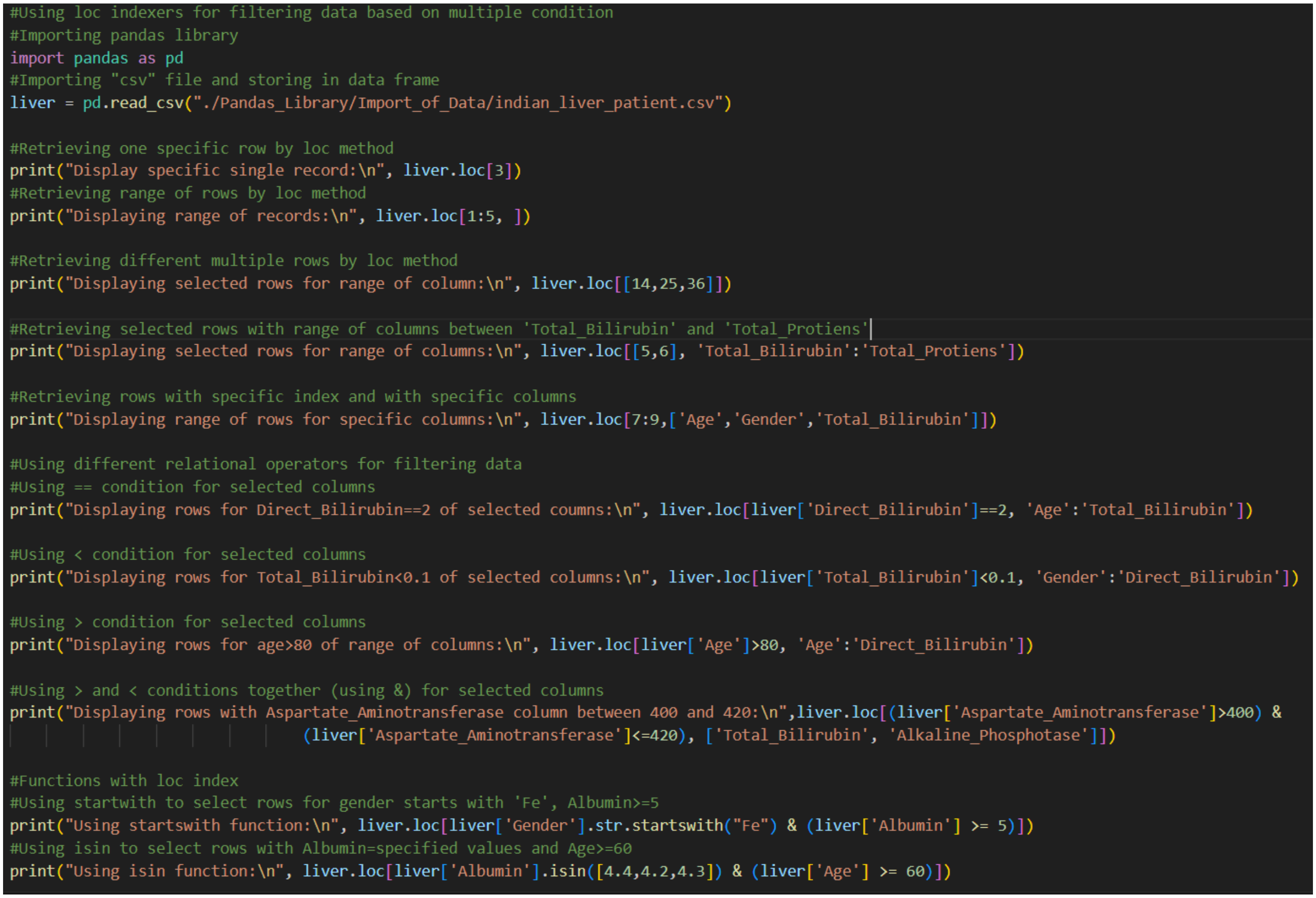
4. **Using loc Indexers:**
   - The loc indexer gives information of the index value specified within the bracked.
   - The index of the dataframe can be either number and/or a string or multi-value.
   - It should be noted that it is possible to change the index.
   - Unlike iloc, the loc indexer can be used for index and label of columns
   - Using the .loc indexer, columns are referred to by names using lists of strings or ‘:’ for silicing

➢ Relational operators with loc indexer:

- The loc indexer helps to apply different relational operators like <, <=, >=, == for extracting data according to user requirement.

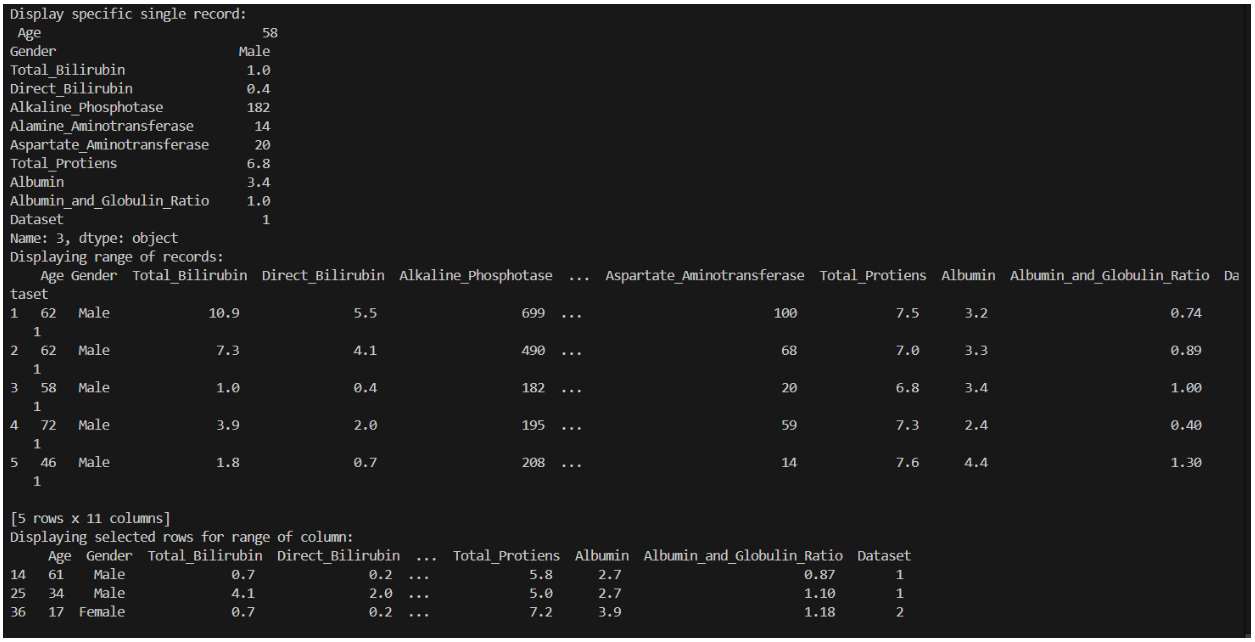
➢ Functions with loc indexer:

- It is also possible to use special functions like **startswith()** and **isin()** to select specific records according to the user’s requirement.
- **startswith():** it is used to check the data present in the column is starting with the particular characters or not
- **isin():** it is used to check the provided data is present in the dataset or in a column or not



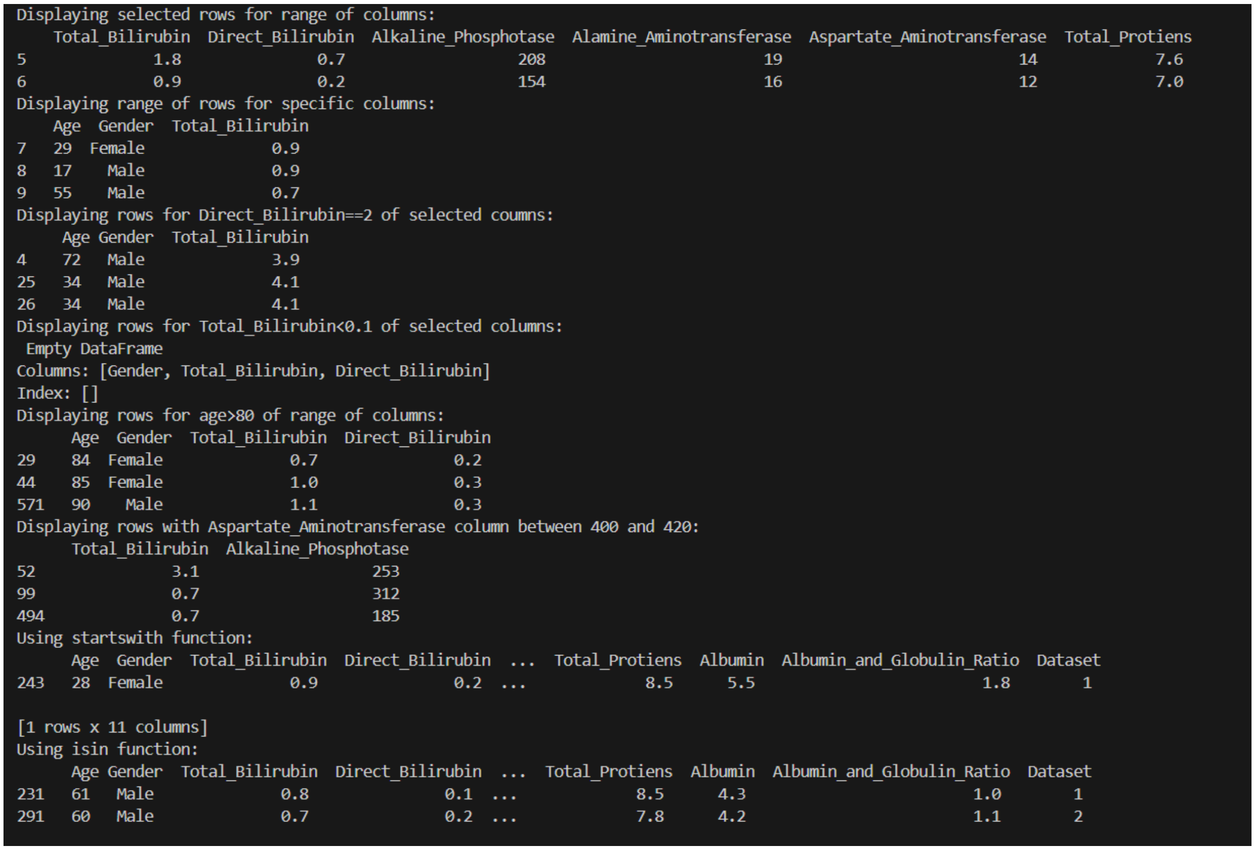
```python
#Using loc indexers for filtering data based on multiple condition
#Importing pandas library
import pandas as pd
#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")

#Retrieving one specific row by loc method
print("Display specific single record:\n", liver.loc[3])
#Retrieving range of rows by loc method
print("Displaying range of records:\n", liver.loc[1:5, ])

#Retrieving different multiple rows by loc method
print("Displaying selected rows for range of column:\n", liver.loc[[14,25,36]])

#Retrieving selected rows with range of columns between 'Total_Bilirubin' and 'Total_Protiens'
print("Displaying selected rows for range of columns:\n", liver.loc[[5,6], 'Total_Bilirubin':'Total_Protiens'])

#Retrieving rows with specific index and with specific columns
print("Displaying range of rows for specific columns:\n", liver.loc[7:9,['Age','Gender','Total_Bilirubin']])

#Using different relational operators for filtering data
#Using == condition for selected columns
print("Displaying rows for Direct_Bilirubin==2 of selected coumns:\n", liver.loc[liver['Direct_Bilirubin']==2, 'Age':'Total_Bilirubin'])

#Using < condition for selected columns
print("Displaying rows for Total_Bilirubin<0.1 of selected columns:\n", liver.loc[liver['Total_Bilirubin']<0.1, 'Gender':'Direct_Bilirubin'])

#Using > condition for selected columns
print("Displaying rows for age>80 of range of columns:\n", liver.loc[liver['Age']>80, 'Age':'Direct_Bilirubin'])

#Using > and < conditions together (using &) for selected columns
print("Displaying rows with Aspartate_Aminotransferase column between 400 and 420:\n",liver.loc[(liver['Aspartate_Aminotransferase']>400) &
                                (liver['Aspartate_Aminotransferase']<=420), ['Total_Bilirubin', 'Alkaline_Phosphotase']])

#Functions with loc index
#Using startwith to select rows for gender starts with 'Fe', Albumin>=5
print("Using startswith function:\n", liver.loc[liver['Gender'].str.startswith("Fe") & (liver['Albumin'] >= 5)])
#Using isin to select rows with Albumin=specified values and Age>=60
print("Using isin function:\n", liver.loc[liver['Albumin'].isin([4.4,4.2,4.3]) & (liver['Age'] >= 60)])
```

Output:



```
Display specific single record:
 Age                              58
Gender                          Male
Total_Bilirubin                  1.0
Direct_Bilirubin                 0.4
Alkaline_Phosphotase             182
Alamine_Aminotransferase          14
Aspartate_Aminotransferase        20
Total_Protiens                   6.8
Albumin                          3.4
Albumin_and_Globulin_Ratio       1.0
Dataset                            1
Name: 3, dtype: object
Displaying range of records:
    Age Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  ...  Aspartate_Aminotransferase  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Da
taset
1   62   Male             10.9               5.5                   699  ...                         100             7.5      3.2                        0.74
   1
2   62   Male              7.3               4.1                   490  ...                          68             7.0      3.3                        0.89
   1
3   58   Male              1.0               0.4                   182  ...                          20             6.8      3.4                        1.00
   1
4   72   Male              3.9               2.0                   195  ...                          59             7.3      2.4                        0.40
   1
5   46   Male              1.8               0.7                   208  ...                          14             7.6      4.4                        1.30
   1

[5 rows x 11 columns]
Displaying selected rows for range of column:
     Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
14   61    Male              0.7               0.2  ...             5.8      2.7                        0.87        1
25   34    Male              4.1               2.0  ...             5.0      2.7                        1.10        1
36   17  Female              0.7               0.2  ...             7.2      3.9                        1.18        2
```

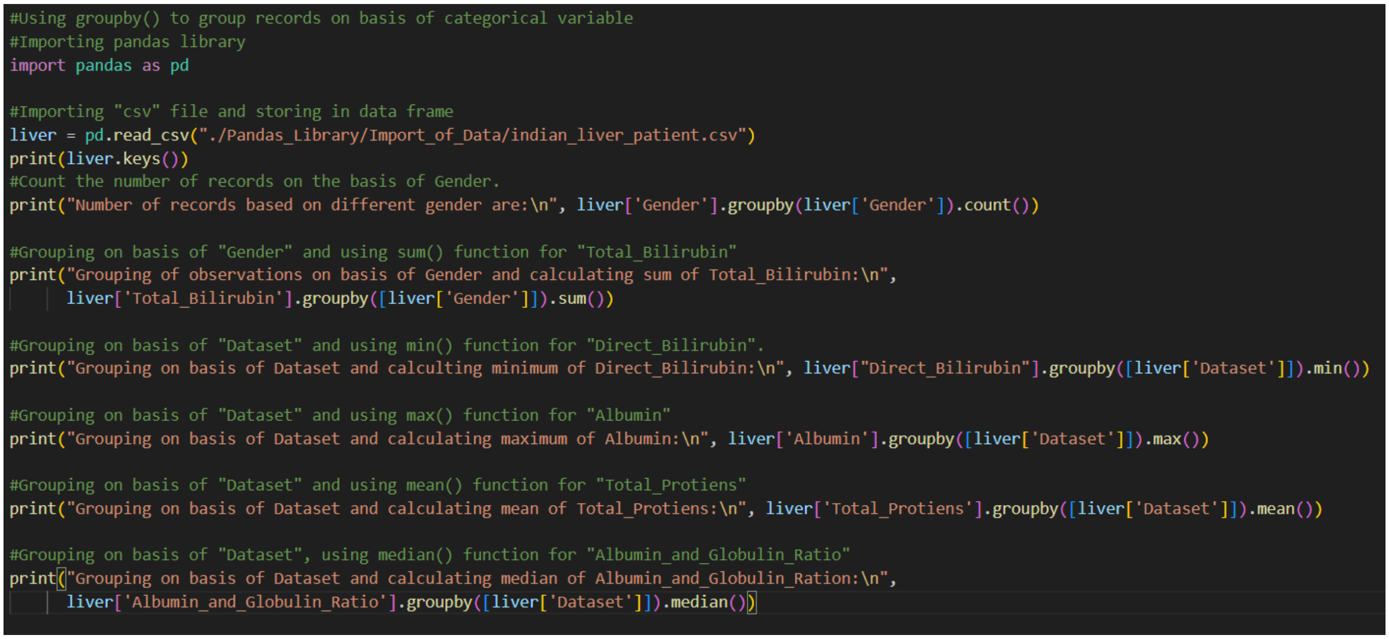
Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main



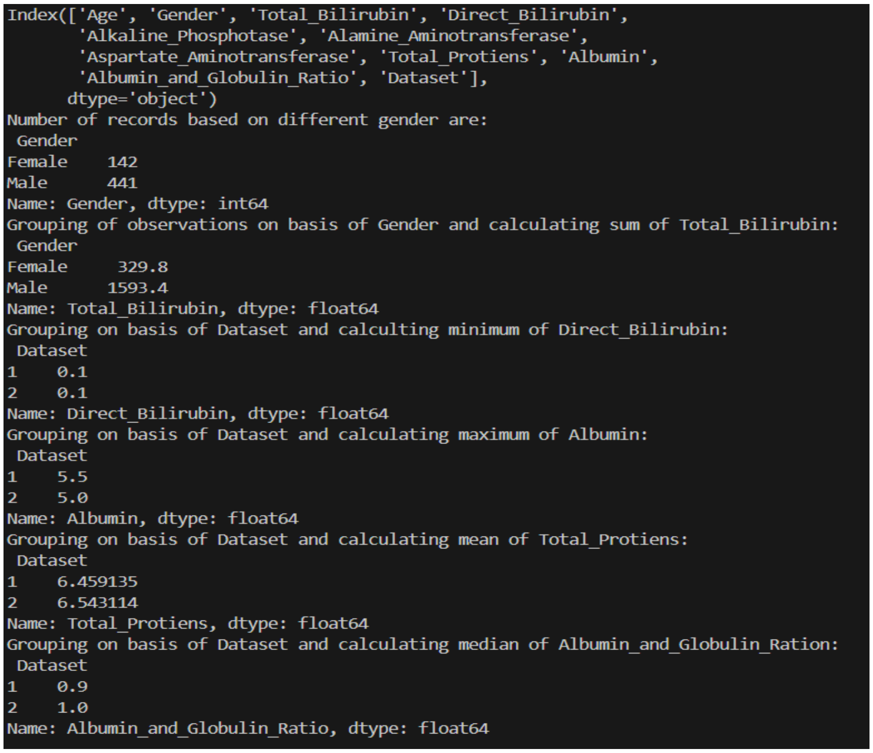
```
Displaying selected rows for range of columns:
    Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens
5              1.8               0.7                   208                        19                          14             7.6
6              0.9               0.2                   154                        16                          12             7.0
Displaying range of rows for specific columns:
    Age  Gender  Total_Bilirubin
7   29  Female              0.9
8   17    Male              0.9
9   55    Male              0.7
Displaying rows for Direct_Bilirubin==2 of selected coumns:
     Age Gender  Total_Bilirubin
4    72   Male              3.9
25   34   Male              4.1
26   34   Male              4.1
Displaying rows for Total_Bilirubin<0.1 of selected columns:
 Empty DataFrame
Columns: [Gender, Total_Bilirubin, Direct_Bilirubin]
Index: []
Displaying rows for age>80 of range of columns:
      Age  Gender  Total_Bilirubin  Direct_Bilirubin
29    84  Female              0.7               0.2
44    85  Female              1.0               0.3
571   90    Male              1.1               0.3
Displaying rows with Aspartate_Aminotransferase column between 400 and 420:
      Total_Bilirubin  Alkaline_Phosphotase
52               3.1                   253
99               0.7                   312
494              0.7                   185
Using startswith function:
      Age  Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
243   28  Female              0.9               0.2  ...             8.5      5.5                         1.8        1

[1 rows x 11 columns]
Using isin function:
      Age Gender  Total_Bilirubin  Direct_Bilirubin  ...  Total_Protiens  Albumin  Albumin_and_Globulin_Ratio  Dataset
231   61   Male              0.8               0.1  ...             8.5      4.3                         1.0        1
291   60   Male              0.7               0.2  ...             7.8      4.2                         1.1        2
```

Group By Fuctionality:

- An important feature of dataframe is the use of “**groupby()”** function which is used to group the observations on the basis of a variable.
- It should be noted that grouping of observations can be done only on the basis of categorical variable and aggregate functions such as **max(), mean(), median(), min(), sum(), count()** are used on any of the continuous/categorical variable in the dataset.



```
#Using groupby() to group records on basis of categorical variable
#Importing pandas library
import pandas as pd

#Importing "csv" file and storing in data frame
liver = pd.read_csv("./Pandas_Library/Import_of_Data/indian_liver_patient.csv")
print(liver.keys())
#Count the number of records on the basis of Gender.
print("Number of records based on different gender are:\n", liver['Gender'].groupby(liver['Gender']).count())

#Grouping on basis of "Gender" and using sum() function for "Total_Bilirubin"
print("Grouping of observations on basis of Gender and calculating sum of Total_Bilirubin:\n",
      liver['Total_Bilirubin'].groupby([liver['Gender']]).sum())

#Grouping on basis of "Dataset" and using min() function for "Direct_Bilirubin".
print("Grouping on basis of Dataset and calculting minimum of Direct_Bilirubin:\n", liver["Direct_Bilirubin"].groupby([liver['Dataset']]).min())

#Grouping on basis of "Dataset" and using max() function for "Albumin"
print("Grouping on basis of Dataset and calculating maximum of Albumin:\n", liver['Albumin'].groupby([liver['Dataset']]).max())

#Grouping on basis of "Dataset" and using mean() function for "Total_Protiens"
print("Grouping on basis of Dataset and calculating mean of Total_Protiens:\n", liver['Total_Protiens'].groupby([liver['Dataset']]).mean())

#Grouping on basis of "Dataset", using median() function for "Albumin_and_Globulin_Ratio"
print("Grouping on basis of Dataset and calculating median of Albumin_and_Globulin_Ration:\n",
      liver['Albumin_and_Globulin_Ratio'].groupby([liver['Dataset']]).median())
```



```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
Number of records based on different gender are:
 Gender
Female    142
Male      441
Name: Gender, dtype: int64
Grouping of observations on basis of Gender and calculating sum of Total_Bilirubin:
 Gender
Female     329.8
Male      1593.4
Name: Total_Bilirubin, dtype: float64
Grouping on basis of Dataset and calculting minimum of Direct_Bilirubin:
 Dataset
1    0.1
2    0.1
Name: Direct_Bilirubin, dtype: float64
Grouping on basis of Dataset and calculating maximum of Albumin:
 Dataset
1    5.5
2    5.0
Name: Albumin, dtype: float64
Grouping on basis of Dataset and calculating mean of Total_Protiens:
 Dataset
1    6.459135
2    6.543114
Name: Total_Protiens, dtype: float64
Grouping on basis of Dataset and calculating median of Albumin_and_Globulin_Ration:
 Dataset
1    0.9
2    1.0
Name: Albumin_and_Globulin_Ratio, dtype: float64
```

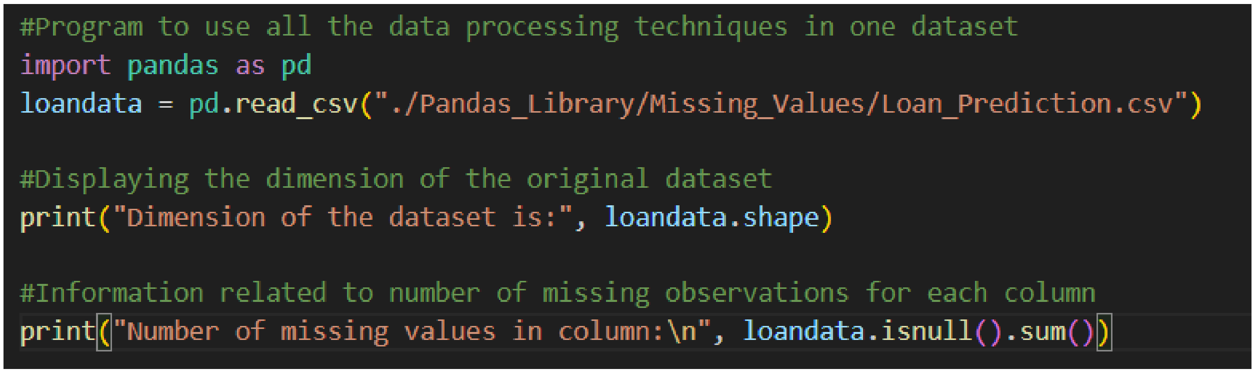
Created by Omesh Sehrawat
Linked   link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

Missing Values:

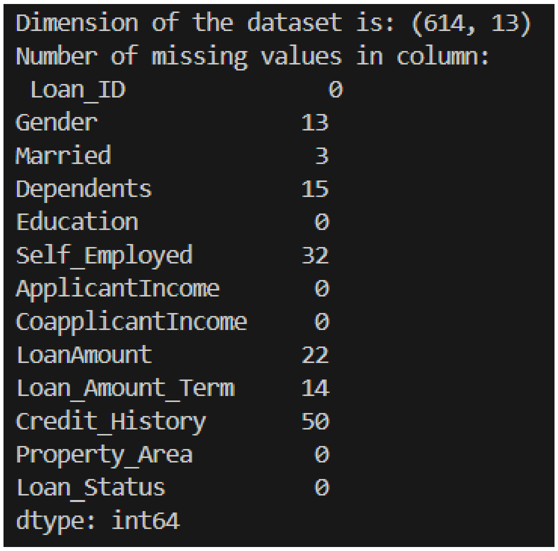
- Missing value is one whose value is unknown in the dataset.
- Missing values are represented in Python by the **NA** symbol.
- **NA** is one of the very few reserved words in Python.
- When an element or value is "not available" or a "missing value" arises in statistical terms, the element is assigned the special value **NA.**
- Second kind of "missing values" which are produced by numerical computation; these are called **NAN (Not a Number)**.
- Impossible values(e.g dividing by zero) are also represented by the symbol **NAN.**
- **Dataset Link:** https://www.kaggle.com/datasets/architsharma01/loan-approval-prediction-dataset

➢ **Determining Missing Values:**

- ✓ The function **isnull()** is used to find the values are null or not.
  It will return the same data with true and false if data not available then true in that column and false when data is available.
- ✓ **isnull().sum()** return the total number of missing values for each column in the dataset .



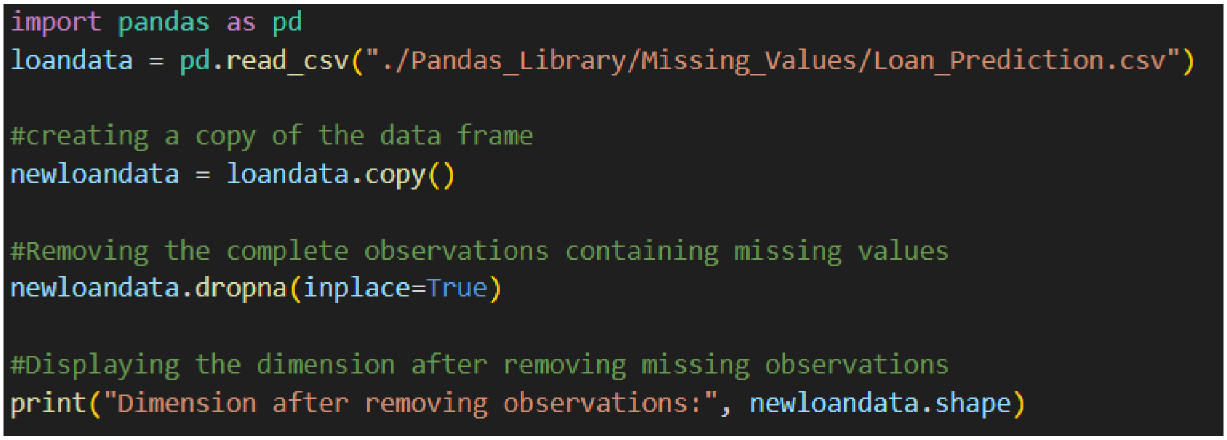
```
#Program to use all the data processing techniques in one dataset
import pandas as pd
loandata = pd.read_csv("./Pandas_Library/Missing_Values/Loan_Prediction.csv")

#Displaying the dimension of the original dataset
print("Dimension of the dataset is:", loandata.shape)

#Information related to number of missing observations for each column
print("Number of missing values in column:\n", loandata.isnull().sum())
```



```
Dimension of the dataset is: (614, 13)
Number of missing values in column:
 Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

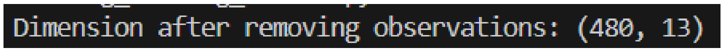
Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main

- **Deleting Observations Containing Missing Values:**
  - The function **dropna(inplace = True)** deletes the observations that contain the missing values from the dataset and hence reduces the number of observations.



```python
import pandas as pd
loandata = pd.read_csv("./Pandas_Library/Missing_Values/Loan_Prediction.csv")

#creating a copy of the data frame
newloandata = loandata.copy()

#Removing the complete observations containing missing values
newloandata.dropna(inplace=True)

#Displaying the dimension after removing missing observations
print("Dimension after removing observations:", newloandata.shape)
```

Output:

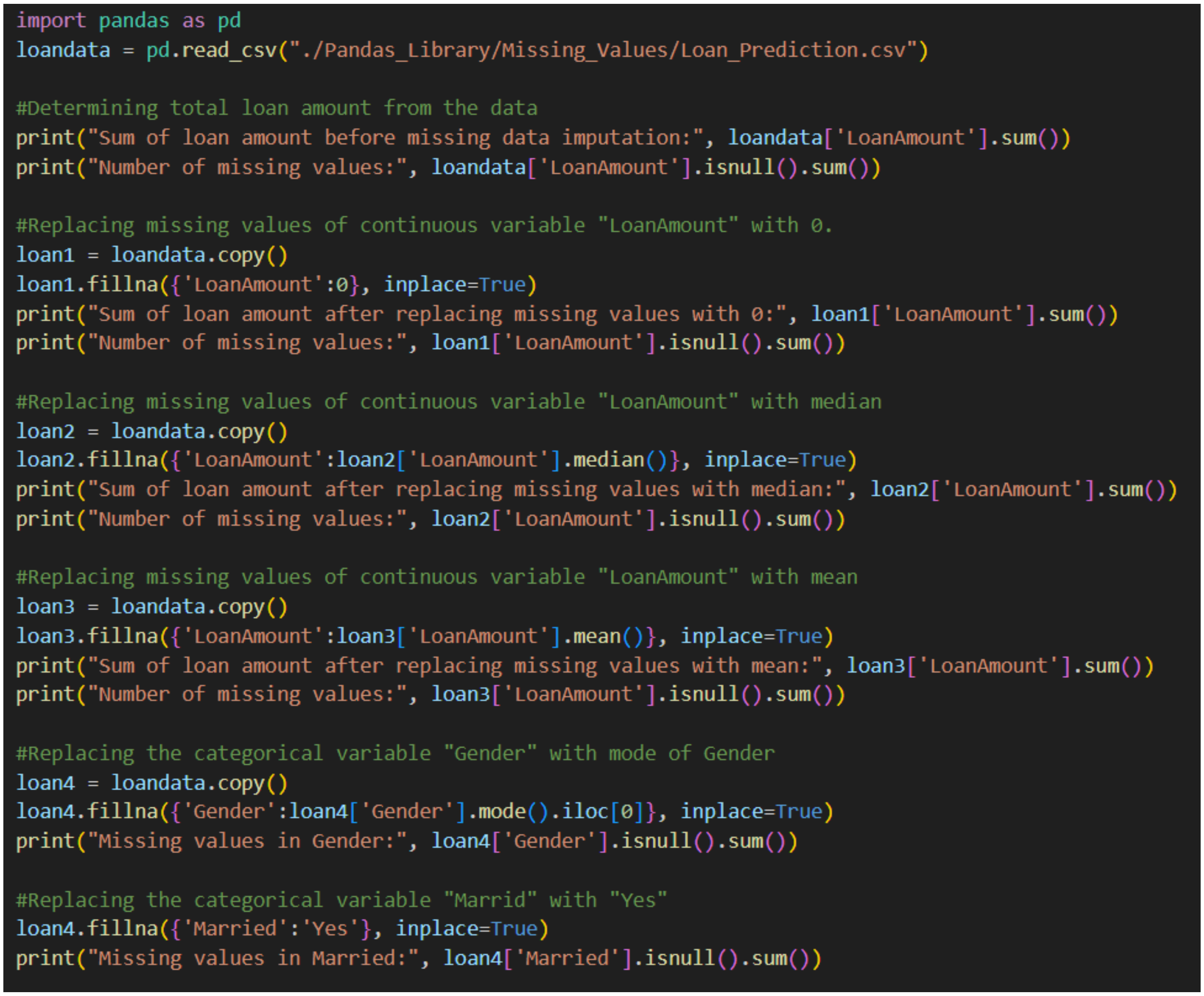


```
Dimension after removing observations: (480, 13)
```

- **Missing Data Imputation:**
  - It is very important to impure the missing data before analysing, because the data analysis functions does not work effectively if missing values exist in the dataset.
  - The function **fillna(value, inplace = True)** fills the missing values (NA) with value written as an argument and thus helps in missing data imputation.
  - The value is generally considered as either **mean(), median(), mode(),** or any specified value.
  - **NOTE:** missing numeric/continuous variables can be replaced with mean, median, or predicted mean

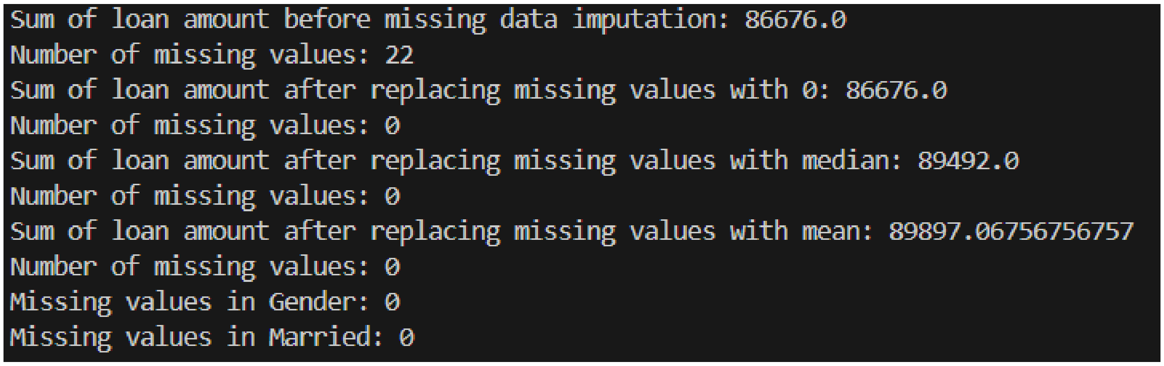
    And

    The categorical variable can be replaced with mode or any predicted categorical value.



```
import pandas as pd
loandata = pd.read_csv("./Pandas_Library/Missing_Values/Loan_Prediction.csv")

#Determining total loan amount from the data
print("Sum of loan amount before missing data imputation:", loandata['LoanAmount'].sum())
print("Number of missing values:", loandata['LoanAmount'].isnull().sum())

#Replacing missing values of continuous variable "LoanAmount" with 0.
loan1 = loandata.copy()
loan1.fillna({'LoanAmount':0}, inplace=True)
print("Sum of loan amount after replacing missing values with 0:", loan1['LoanAmount'].sum())
print("Number of missing values:", loan1['LoanAmount'].isnull().sum())

#Replacing missing values of continuous variable "LoanAmount" with median
loan2 = loandata.copy()
loan2.fillna({'LoanAmount':loan2['LoanAmount'].median()}, inplace=True)
print("Sum of loan amount after replacing missing values with median:", loan2['LoanAmount'].sum())
print("Number of missing values:", loan2['LoanAmount'].isnull().sum())

#Replacing missing values of continuous variable "LoanAmount" with mean
loan3 = loandata.copy()
loan3.fillna({'LoanAmount':loan3['LoanAmount'].mean()}, inplace=True)
print("Sum of loan amount after replacing missing values with mean:", loan3['LoanAmount'].sum())
print("Number of missing values:", loan3['LoanAmount'].isnull().sum())

#Replacing the categorical variable "Gender" with mode of Gender
loan4 = loandata.copy()
loan4.fillna({'Gender':loan4['Gender'].mode().iloc[0]}, inplace=True)
print("Missing values in Gender:", loan4['Gender'].isnull().sum())

#Replacing the categorical variable "Marrid" with "Yes"
loan4.fillna({'Married':'Yes'}, inplace=True)
print("Missing values in Married:", loan4['Married'].isnull().sum())
```

Output:



```
Sum of loan amount before missing data imputation: 86676.0
Number of missing values: 22
Sum of loan amount after replacing missing values with 0: 86676.0
Number of missing values: 0
Sum of loan amount after replacing missing values with median: 89492.0
Number of missing values: 0
Sum of loan amount after replacing missing values with mean: 89897.06756756757
Number of missing values: 0
Missing values in Gender: 0
Missing values in Married: 0
```

Created by Omesh Sehrawat
Linked link - https://www.linkedin.com/in/omesh-sehrawat-836b40186/
GitHub link - https://github.com/omeshsehrawat/Python_Libraries/tree/main