Basics of Data Frame

The Pandas library is specially used for handling data of different dimensions.

Series is a one-dimensional labelled data and "data frame" is a two-dimensional labelled data holding any data type.

A series can be created using the function Series() from pandas library.

Data Frame can be thought as a table in RDBMS or a spread sheet.
A data frame is created using DataFrame() function.

In [1]:
```
#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)
```

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

In [2]:
```
# 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())
```

```
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')
```

Adding Rows and Columns to the Data Frame

We can add rows to an existing data frame from a new data frame by adding a new column to the data frame by writing the name of the column in square brackets along with the name of the data frame and assigning a list of items to it.

In [10]:
```
#Program for adding rows and columns to the data frame.
# 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 second data frame is: ",productdf2.shape)
#Adding rows to the data frame by adding other data frame.
productdf3=pd.concat([productdf,productdf2])
print ("Dimension of the new data frame is: ",productdf3.shape)
#Adding column named "Mobile" to the data frame.
productdf3 ["Mobile"]=[15000, 2, 30, 40]
#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)
#Displaying the shape of the new data frame.
print ("Dimension of the new data frame is:",productdf3.shape)
#Displaying the size of the data frame using size.
print ("Size of the new data frame is:",productdf3.size)
```

```
Second data frame is:
    Pen  Shirt  Book  Mouse
0   15     16    17     18
1    5      6     7      8
Dimension of the second data frame is:  (2, 4)
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   15000   35000
1    4      2     5      6       2       3
0   15     16    17     18      30      10
1    5      6     7      8      40      15
Dimension of the new data frame is: (4, 6)
Size of the new data frame is: 24
```

Deleting Rows and Columns from the Data Frame

It is possible to delete rows and columns from the data frame using the drop() function. Columns which needs to be deleted from the data frame are specified by the names of the columns as value of the “columns” argument in drop() function. Rows can be deleted by specifying the index of the rows to be deleted as the value of the “index” argument in drop() function

In [11]:
```
#Program for deleting rows and columns from the data frame.
#Deleting multiple columns from the data frame using drop() function.
productdf3=productdf3.drop (columns=["Pen", "Book"])
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)
```

```
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       2       3
1      6      8      40      15
Size of modified data frame is: 8
```

Import of Data

Pandas library provide many functions to import data from files of different types of software and stores in a data frame in python.

read_csv() helps to read a “csv” file;

read_excel() helps to read an “excel” file;

read_html() helps to read a “html” file;

read_json() helps to read a “json” file and read_sql() helps to read a “sql” file.

In [12]:
```
#Importing "csv" file and storing in data frame.
liver=pd.read_csv ("Ind-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)
```

```
Dimension of the dataset is: (583, 11)
Size of the dataset is:  6413
Columns in the dataset are:
 Index(['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'Sgpt', 'Sgot', 'TP', 'ALB',
       'AG', 'LiverPatient'],
      dtype='object')
Columns in the dataset are:
 Index(['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'Sgpt', 'Sgot', 'TP', 'ALB',
       'AG', 'LiverPatient'],
      dtype='object')
```

Functions of Data Frame

The Pandas library provides many functions with respect to a data frame.

These functions are related to: basic functions related to information of the data frame like describe(), info() etc.

In [13]:

```
#Program for using functions related to general information of data.
#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())
```

```
<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   TB            583 non-null    float64
 3   DB            583 non-null    float64
 4   Alkphos       583 non-null    int64  
 5   Sgpt          583 non-null    int64  
 6   Sgot          583 non-null    int64  
 7   TP            583 non-null    float64
 8   ALB           583 non-null    float64
 9   AG            579 non-null    float64
 10  LiverPatient  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    TB   DB  Alkphos  Sgpt  Sgot
TP  ALB    AG  LiverPatient
0     65  Female   0.7  0.1      187    16    18  6.8  3.3  0.90            1
1     62    Male  10.9  5.5      699    64   100  7.5  3.2  0.74            1
2     62    Male   7.3  4.1      490    60    68  7.0  3.3  0.89            1
3     58    Male   1.0  0.4      182    14    20  6.8  3.4  1.00            1
4     72    Male   3.9  2.0      195    27    59  7.3  2.4  0.40            1
..   ...     ...   ...  ...      ...   ...   ...  ...  ...   ...          ...
578   60    Male   0.5  0.1      500    20    34  5.9  1.6  0.37            2
579   40    Male   0.6  0.1       98    35    31  6.0  3.2  1.10            1
580   52    Male   0.8  0.2      245    48    49  6.4  3.2  1.00            1
581   31    Male   1.3  0.5      184    29    32  6.8  3.4  1.00            1
582   38    Male   1.0  0.3      216    21    24  7.3  4.4  1.50            2

[583 rows x 11 columns]>
Description of the dataset is:
               Age          TB          DB      Alkphos         Sgpt  \
count  583.000000  583.000000  583.000000   583.000000   583.000000   
mean    44.746141    3.298799    1.486106   290.576329    80.713551   
std     16.189833    6.209522    2.808498   242.937989   182.620356   
min      4.000000    0.400000    0.100000    63.000000    10.000000   
25%     33.000000    0.800000    0.200000   175.500000    23.000000   
50%     45.000000    1.000000    0.300000   208.000000    35.000000   
75%     58.000000    2.600000    1.300000   298.000000    60.500000   
max     90.000000   75.000000   19.700000  2110.000000  2000.000000   

              Sgot          TP         ALB          AG  LiverPatient  
count   583.000000  583.000000  583.000000  579.000000    583.000000  
mean    109.910806    6.483190    3.141852    0.947064      1.286449  
std     288.918529    1.085451    0.795519    0.319592      0.452490  
min      10.000000    2.700000    0.900000    0.300000      1.000000  
25%      25.000000    5.800000    2.600000    0.700000      1.000000  
50%      42.000000    6.600000    3.100000    0.930000      1.000000  
75%      87.000000    7.200000    3.800000    1.100000      2.000000  
max    4929.000000    9.600000    5.500000    2.800000      2.000000  
```

displaying records using head(), tail() etc.,

In [14]:
```
#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 three records.
print ("First two records of dataset are:\n", liver.head(2))
#Displaying the first three records of "TB" and "DB".
print ("First 3 records of TB and DB: \n", liver [['TB', 'DB']].head(3))
#Use of tail() function to display ending records.
#Displaying the last two records.
print ("Last three records of dataset is: \n", liver.tail(3))
#Displaying the last two records of "Age" and "TB".
print ("Last 2 records of age and TB: \n", liver [['Age', 'TB']].tail(2))
#Determining all the values of Alkphos column only.
print ("Values for Alkphos column are:\n", liver ['Alkphos'].values)
```

```
First five records of dataset are:
    Age  Gender    TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
0   65  Female   0.7  0.1      187    16    18  6.8  3.3  0.90             1
1   62    Male  10.9  5.5      699    64   100  7.5  3.2  0.74             1
2   62    Male   7.3  4.1      490    60    68  7.0  3.3  0.89             1
3   58    Male   1.0  0.4      182    14    20  6.8  3.4  1.00             1
4   72    Male   3.9  2.0      195    27    59  7.3  2.4  0.40             1
First two records of dataset are:
    Age  Gender    TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
0   65  Female   0.7  0.1      187    16    18  6.8  3.3  0.90             1
1   62    Male  10.9  5.5      699    64   100  7.5  3.2  0.74             1
First 3 records of TB and DB:
      TB   DB
0   0.7  0.1
1  10.9  5.5
2   7.3  4.1
Last three records of dataset is:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
580   52   Male  0.8  0.2      245    48    49  6.4  3.2  1.0             1
581   31   Male  1.3  0.5      184    29    32  6.8  3.4  1.0             1
582   38   Male  1.0  0.3      216    21    24  7.3  4.4  1.5             2
Last 2 records of age and TB:
      Age   TB
581   31  1.3
582   38  1.0
Values for Alkphos column are:
 [ 187  699  490  182  195  208  154  202  202  290  210  260  310  214
  145  183  342  165  293  293  610  482  542  231  194  289  289  240
  128  188  190  156  187  410  410  482  145  374  263  275  168  160
  630  415  208  275  150  230  176  206  170  161  253  198  272  272
  198  175  367  145  158  158  158  208  259  470  195  215  239  215
  186  188  205  171  145  162  518 1620  146  670  915   75  148  258
  237  269  320  298  538  238  214  308  298  204  168  282  298  215
  265  312  161  243  224  225  170  145  145  158  158  486  188  257
  179  272  661 1580 1630  194  280  298  300  290  188  178  177  201
  802  248 1896  263  512  237  199  238  178 1110  310  282  282  380
  186  159  332  332  189  201  168  392  202  286  180  218  182  178
  290  298  462  196  196  282  750 1050  599  180  180  282  332  292
  962  950  200  298  750  175  175  198  482 1020  562  386  250  218
  170  171  201  298  750  191  614  218  314  257  272  206  209 1124
  664  142  169 1420  218  218  145  142  135  163  285  350  220  189
  190  219  160  401  180  100  116  159  289  125  147  192  265  175
  400  120  173  186  202  290  196  282  157 2110  285  360  300  158
  190  196  165  230  205  316  218  290  272  190  202  498  480  680
  258  180  152  859  901  335  182  285  245  505  228  185  247  348
  195  140  358  110  235  460  380  262  196  180  190  190  209  144
  123  192  188  316  300  575  192  155  239  315  250  174  245  191
  340  202  234  159  190  195  180  280  430  206  155  195  588  174
  165  527  175  574  106  158  195  179  182  198  216  310   63  198
  205  302  171  158  358  174  192  211  157  210  258  152  350  182
  458  375  405  215  206  650  198  198  195  230  115  216  358  158
  145  195  144  621  150  178  256  205  176  146  218  182  215  165
  183  176  418  271  182  130  558  135  326  140  145  206  168  202
  192  185  331  188  172  159  490  152  105  160  160  102  148  162
  149  580  310  140  175  152  208  205  162   92  162  199  198  215
  180  719  554  555  215  509  190  208  260  690  862  592  450 1350
```

```
1350  163  246  178  240  100  166  170  194 1750  182  236  165  201
 194  206  212  157  162  168  198  292  298  152  163  279  181 1550
 142  173  282  279 1100  224  159  186  189  192  140  686  215  309
 110  130  164  270  137   90  190  165  167  185  197  154  226  310
 310  220  196  186  352  282   92  182  103  850  195  276  171  146
 193  180  805  265  185  165  189  198  151  349  365  305  127  238
 218  219  239  194  450  254  205  320  195  215  230  189  168  215
 210  108  224  230  185  137  156  210  268  298  315  214  138  285
 162  298  230  466  227  395   97  406  114  198  173  204  350  153
 188  380  214  768  172  160  196  232  220  290  180  189  275  390
 356  315  388  298  165  143  191  251  200  268  236  215  134  612
 515  560  289  190  500   98  245  184  216]
```

In [15]:

```python
#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())
#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 "TB" column.
print ("Describing the details of TB column: \n", liver ['TB'].describe())
```

```
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       3.298799
std        6.209522
min        0.400000
25%        0.800000
50%        1.000000
75%        2.600000
max       75.000000
Name: TB, dtype: float64
```

statistical functions - mean(), median() etc. mathematical functions - min(), prod(), max(), sum() etc.

In [16]:

```python
#Program to use mathematical and statistical functions on filtered data.
#Determining mean of Age column using mean () function.
print ("Mean of age is:", liver ['Age'].mean())
#Determining median of Age column using median () function.
print ("Median of age is:", liver ['Age'].median())
#Determining maximum of Alkphos column using max() function.
print ("Maximum alkphos is: ", liver['Alkphos'].max())
#Determining minimum of Alkphos column using min() function.
print ("Minimum alkphos is: ", liver['Alkphos'].min())
#Determining sum of 'TB' column using sum() function.
```

```
print ("Sum of TB is:", liver['TB'].sum())
#Determining product of 'TB' column using product() function.
print ("Product of TB is:", liver['TB'].product())
#Determining 3 smallest values of 'DB' using nsmallest() function.
print ("Three smallest values of DB: \n",liver ['DB'].nsmallest().head(3))
#Determining 4 largest values of 'DB' using nlargest () function.
print ("Four largest values of DB: \n", liver ['DB'].nlargest().head (4))
```

```
Mean of age is: 44.74614065180103
Median of age is: 45.0
Maximum alkphos is:  2110
Minimum alkphos is:  63
Sum of TB is: 1923.1999999999998
Product of TB is: 2.163950516974104e+117
Three smallest values of DB:
 0     0.1
10    0.1
15    0.1
Name: DB, dtype: float64
Four largest values of DB:
 559    19.7
531    18.3
504    17.1
259    14.2
Name: DB, dtype: float64
```

sorting of the data frame on the basis of specified column using sort_values() function etc.

In [18]:
```
#Program for sorting in ascending, descending order on basis of "TP".
#Sorting in descending order.
print ("Top two records based on descending order for TP are:\n",
       liver.sort_values (by='TP', ascending=False).head(2))
print ("Bottom two records based on descending order for TP are: \n",
       liver.sort_values (by='TP', ascending=False).tail(2))
#Sorting in ascending order.
print ("Top two records based on ascending order for TP are: \n",
       liver.sort_values (by='TP', ascending=True).head(2))
print ("Bottom two records based on ascending order for TP are:\n",
       liver.sort_values (by='TP', ascending=True).tail(2))
```

```
Top two records based on descending order for TP are:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
273   30    Male  0.7  0.2      262    15    18  9.6  4.7  1.2             1
270   37    Male  0.7  0.2      235    96    54  9.5  4.9  1.0             1
Bottom two records based on descending order for TP are:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
269   26    Male  0.6  0.1      110    15    20  2.8  1.6  1.3             1
180   75    Male  2.8  1.3      250    23    29  2.7  0.9  0.5             1
Top two records based on ascending order for TP are:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
180   75    Male  2.8  1.3      250    23    29  2.7  0.9  0.5             1
269   26    Male  0.6  0.1      110    15    20  2.8  1.6  1.3             1
Bottom two records based on ascending order for TP are:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
270   37    Male  0.7  0.2      235    96    54  9.5  4.9  1.0             1
273   30    Male  0.7  0.2      262    15    18  9.6  4.7  1.2             1
```

Data Extraction

Different relational operators like <, >, ==, <=, >=, != etc. can be used to create conditions. These conditions will help in filtering data from the dataset.

In [19]:
```
#Program for using relational operators for filtering the data.
#Displaying the first 2 records where gender is male.
male_data=liver[liver[ "Gender"]=="Male"]
print ("First 2 records of 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 ALB is less than or equal to 1.
alb_less1=liver['ALB']<=1
print("Last 2 records having ALB<=1 are:\n",liver [alb_less1].tail(2))
```

```
First 2 records of male patients are:
    Age Gender    TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
1   62   Male  10.9  5.5      699    64   100  7.5  3.2  0.74             1
2   62   Male   7.3  4.1      490    60    68  7.0  3.3  0.89             1
First 3 records for age>=50 are:
    Age  Gender    TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
0   65  Female   0.7  0.1      187    16    18  6.8  3.3  0.90             1
1   62    Male  10.9  5.5      699    64   100  7.5  3.2  0.74             1
2   62    Male   7.3  4.1      490    60    68  7.0  3.3  0.89             1
Last 2 records having ALB<=1 are:
      Age  Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
458   26    Male  6.8  3.2      140    37    19  3.6  0.9  0.3             1
533   46  Female  1.4  0.4      298   509   623  3.6  1.0  0.3             1
```

The use of logical operators like and (&), or(|) help to filter the data on the basis of multiple conditions.

In [22]:
```
#Creating a new subset using "and" for multiple conditions.
filter1=liver[(liver ['Age'] >=35) & (liver ['DB']<=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['TB'].sum())
#Determining product of "DB" for a subset.
print("Product of DB from filtered set: ", filter1['DB'].product())
#Creating a new subset using "or" operator for multiple conditions.
filter2=liver[(liver ['Gender']=="Female") | (liver ['Age']>=35) | (liver ['DB']<=6
print ("Shape of new dataset using or is: ", filter2.shape)
#Applying mean and median functions on filtered data.
#Determining mean of "ALB" for a subset.
print ("Mean of ALB from the filtered set: ", filter2 ['ALB'].mean())
#Determining median of "TP" for a subset.
print ("Median of TP from the filtered set: ", filter2 ['TP'].median())
#Using both "and" and "or" together for multiple conditions.
filter3=liver[(liver ['LiverPatient'] ==1) & (liver.Age>=50) | (liver ['TP']>=2)
| (liver.ALB>2)]
```

```
print ("Shape of new dataset using and & or is: ", filter3. shape)
#Applying maximum and minimum functions on filtered data.
#Determining maximum of "Alkphos" for a subset.
print ("Maximum of Alkphos from filtered set: ", filter3 ['Alkphos'].max())
#Determining minimum of "AG" for a subset.
print ("Minimum of AG from the filtered set: ", filter3 ['AG'].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:  (569, 11)
Mean of ALB from the filtered set:  3.1441124780316345
Median of TP from the filtered set:  6.6
Shape of new dataset using and & or is:  (583, 11)
Maximum of Alkphos from filtered set:  2110
Minimum of AG from the filtered set:  0.3
```

The use of indexers like loc and iloc also contribute a lot for extracting data according to the user requirement.

In [23]:
```
#Displaying single column of 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]])
```

```
Third column of sixth record:  1.8
Sixth Record:
 Age                46
Gender          Male
TB               1.8
DB               0.7
Alkphos          208
Sgpt              19
Sgot              14
TP               7.6
ALB              4.4
AG               1.3
LiverPatient       1
Name: 5, dtype: object
Selected row and selected column:
   Gender  Alkphos
5   Male      208
9   Male      290
Range of records for sixth column:
    Sgpt
7    14
8    22
```

In [27]:
```
#Retrieving one specific row by loc method.
print ("Displaying 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 multiple specified records: \n", liver.loc[[14,25,36]])
#Retrieving selected rows with range of columns between 'TB' and 'TP'.
print ("Displaying selected rows for range of columns: \n", liver.loc[[5, 6], 'TB':
#Retrieving rows with specific index and with specific columns.
print ("Displaying range of rows for specific columns: \n", liver.loc[[7,8,9],['Age
```

```
Displaying specific single record:
 Age               58
Gender           Male
TB                1.0
DB                0.4
Alkphos           182
Sgpt               14
Sgot               20
TP                6.8
ALB               3.4
AG                1.0
LiverPatient        1
Name: 3, dtype: object
Displaying range of records:
    Age Gender    TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
1   62    Male  10.9  5.5      699    64   100  7.5  3.2  0.74             1
2   62    Male   7.3  4.1      490    60    68  7.0  3.3  0.89             1
3   58    Male   1.0  0.4      182    14    20  6.8  3.4  1.00             1
4   72    Male   3.9  2.0      195    27    59  7.3  2.4  0.40             1
5   46    Male   1.8  0.7      208    19    14  7.6  4.4  1.30             1
Displaying multiple specified records:
     Age   Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB    AG  LiverPatient
14   61     Male  0.7  0.2      145    53    41  5.8  2.7  0.87             1
25   34     Male  4.1  2.0      289   875   731  5.0  2.7  1.10             1
36   17   Female  0.7  0.2      145    18    36  7.2  3.9  1.18             2
Displaying selected rows for range of columns:
     TB   DB  Alkphos  Sgpt  Sgot   TP
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   TB
7   29  Female  0.9
8   17    Male  0.9
9   55    Male  0.7
```

In [28]:
```
#Using different relational operators for filtering data.
#Using = condition for selected columns.
print ("Displaying rows for DB==2 of selected columns: \n",
       liver.loc [liver ['DB'] ==2, 'Age': 'TB'])
#Using condition for selected columns.
print ("Displaying rows for TB<0.1 of selected columns: \n",
       liver.loc[liver['TB']<0.1, 'Gender': 'DB'])
#Using > condition for selected columns.
print ("Displaying rows for age>80 of range of columns: \n",
       liver.loc[liver ['Age'] >80, 'Age': 'DB'])
#Using > and < conditions together (using &) for selected columns.
print ("Displaying rows with Sgpt column between 400 and 420:\n",
       liver. loc[ (liver['Sgpt']>400) & (liver ['Sgpt']<= 420), ['TB', 'Alkphos']]
```

```
Displaying rows for DB==2 of selected columns:
     Age Gender   TB
4     72   Male  3.9
25    34   Male  4.1
26    34   Male  4.1
Displaying rows for TB<0.1 of selected columns:
 Empty DataFrame
Columns: [Gender, TB, DB]
Index: []
Displaying rows for age>80 of range of columns:
      Age  Gender   TB   DB
29     84  Female  0.7  0.2
44     85  Female  1.0  0.3
571    90    Male  1.1  0.3
Displaying rows with Sgpt column between 400 and 420:
      TB  Alkphos
43  2.6      415
90  5.7      214
91  6.8      308
92  8.6      298
```

In [29]:

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

```
Using startswith Function:
      Age  Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
243   28  Female  0.9  0.2      316    25    23  8.5  5.5  1.8             1
Using isin Function:
      Age Gender   TB   DB  Alkphos  Sgpt  Sgot   TP  ALB   AG  LiverPatient
231   61   Male  0.8  0.1      282    85   231  8.5  4.3  1.0             1
291   60   Male  0.7  0.2      174    32    14  7.8  4.2  1.1             2
```

Group by Functionality

An important feature of data frame 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 like max(), mean(), median(), min(), sum(), count() on any of the continuous/categorical variable in the dataset.

In [31]:

```
#Using groupby() to group records on basis of categorical variable.
#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 "TB".
print ("Grouping of observations on basis of Gender and calculating sum of TB:\n",
       liver ['TB'].groupby([liver ['Gender']]).sum())
#Grouping on basis of "LiverPatient" and using min() function for "DB".
```

```
print ("Grouping on basis of LiverPatient and calculating minimum of DB:\n",
       liver ['DB']. groupby([liver ['LiverPatient']]).min())
#Grouping on basis of "LiverPatient" and using max() function for "ALB".
print ("Grouping on basis of LiverPatient and calculating maximum of ALB: \n",
       liver ['ALB']. groupby([liver ['LiverPatient']]).max())
#Grouping on basis of "LiverPatient" and using mean() function for "TP".
print ("Grouping on basis of LiverPatient and calculating mean of TP:\n",
       liver['TP']. groupby([liver ['LiverPatient']]).mean())
#Grouping on basis of "LiverPatient", using median () function for "AG".
print ("Grouping on basis of LiverPatient and calculating median of AG:\n",
       liver['AG']. groupby([liver ['LiverPatient']]).median () )
```

```
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 TB:
 Gender
Female     329.8
Male      1593.4
Name: TB, dtype: float64
Grouping on basis of LiverPatient and calculating minimum of DB:
 LiverPatient
1    0.1
2    0.1
Name: DB, dtype: float64
Grouping on basis of LiverPatient and calculating maximum of ALB:
 LiverPatient
1    5.5
2    5.0
Name: ALB, dtype: float64
Grouping on basis of LiverPatient and calculating mean of TP:
 LiverPatient
1    6.459135
2    6.543114
Name: TP, dtype: float64
Grouping on basis of LiverPatient and calculating median of AG:
 LiverPatient
1    0.9
2    1.0
Name: AG, dtype: float64
```

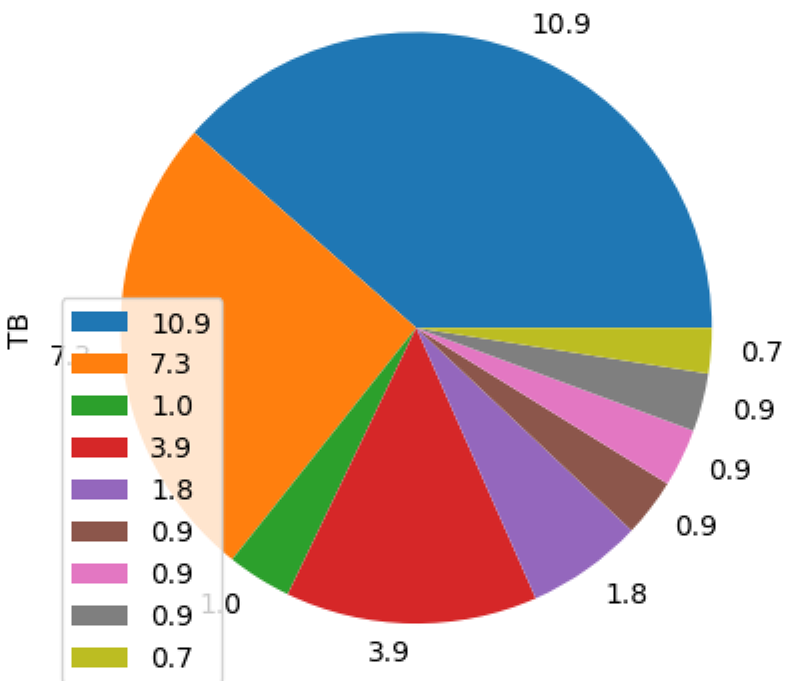
Creating Charts for Data Frame

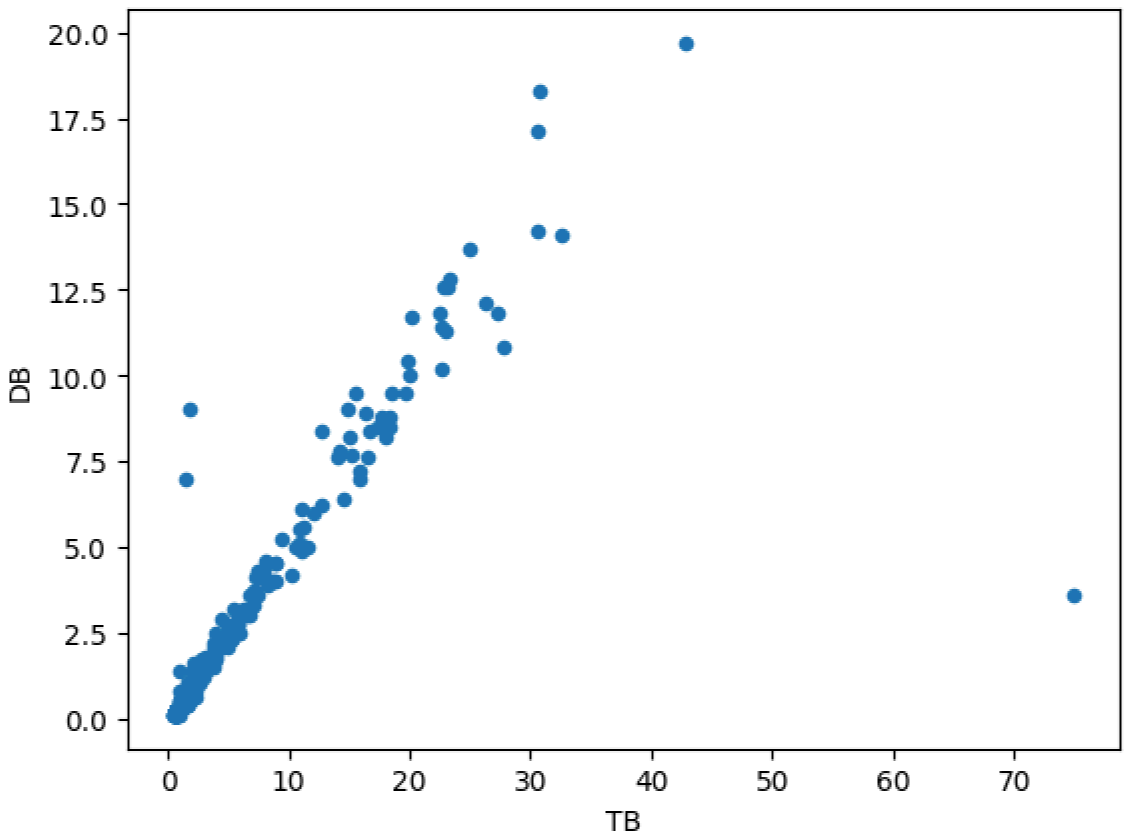
pandas library also supports to create basic charts for a data frame like pie chart using pie() function; scatter plot using scatter() function; histogram using hist() function and boxplot using boxplot() function.

In [35]:
```
#Program for creating charts using Pandas.
#Pie chart for "TB", labels of Liver Patient of 10 records.
data1= liver.iloc[1:10,]
data1.plot.pie(y='TB',labels = data1['TB'])
```

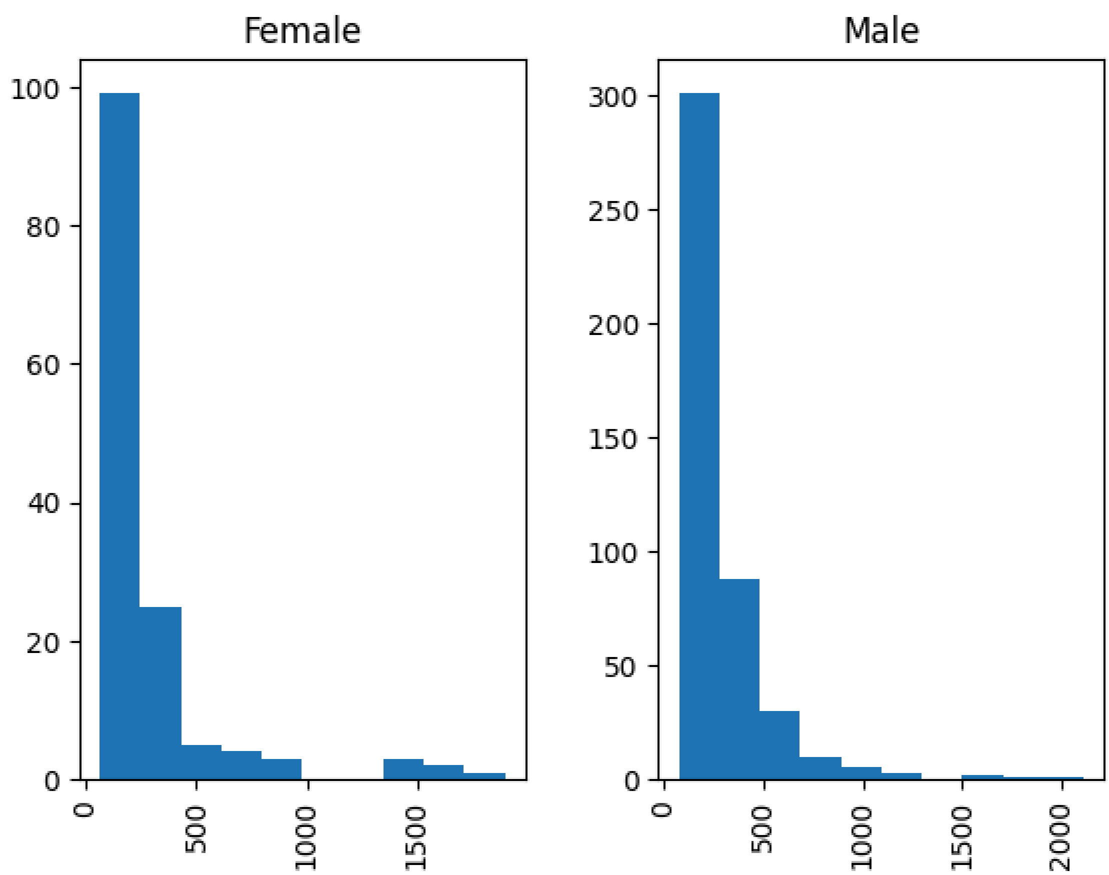
```python
#Scatter plot between "TB" and "DB".
liver.plot.scatter ('TB', 'DB')
#Histogram for "Alkphos" for different values of "Gender".
liver.hist (column='Alkphos', by='Gender')
#Boxplot for "AG" for different values of "LiverPatient".
liver.boxplot (column='AG', by = 'LiverPatient')
```

Out[35]: <Axes: title={'center': 'AG'}, xlabel='LiverPatient'>

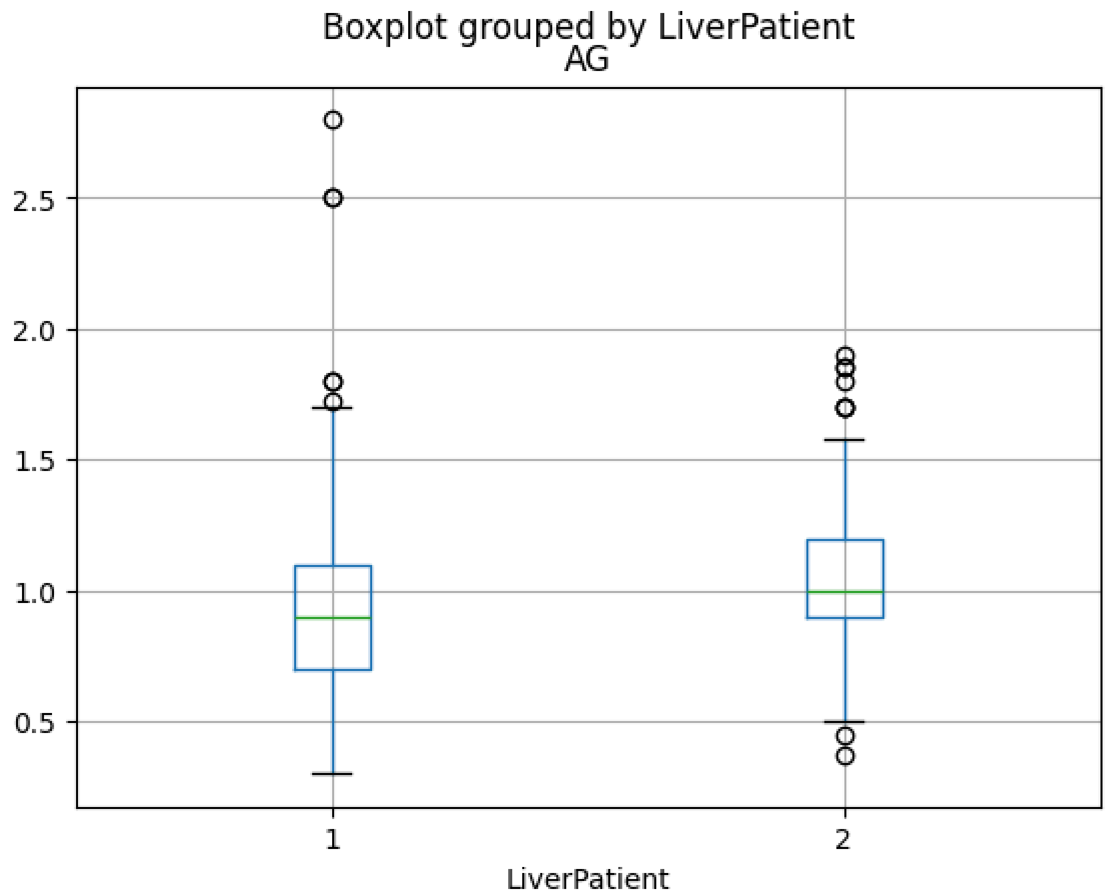




DB
TB



Female
Male



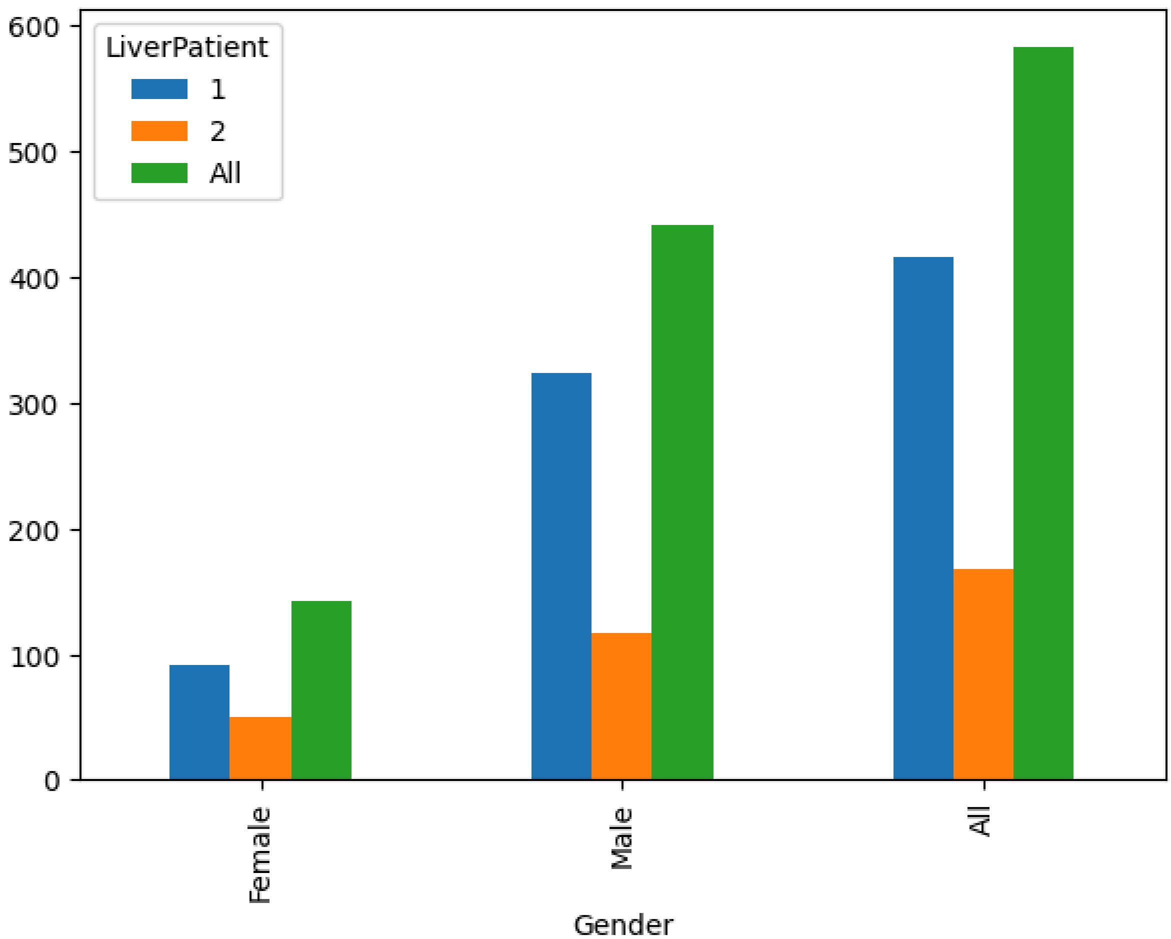
In [38]:

```
#Using crosstab() for determining observations of two categorical variables.
print (pd.crosstab(liver.Gender, liver.LiverPatient, margins=True))
#Displaying the bar chart for categorical variables
pd.crosstab(liver.Gender,liver.LiverPatient, margins=True).plot(kind='bar',figsize=
```

```
LiverPatient    1    2  All
Gender
Female         92   50  142
Male          324  117  441
All           416  167  583
```

Out[38]:

```
<Axes: xlabel='Gender'>
```



Missing Values

A missing value is one whose value is unknown. Missing values are represented in Python by the NA symbol. NAs can arise when there is an empty column in a record in a database, or when excel spreadsheet exist with empty cells. When an element or value is “not available” or a “missing value” arises in statistical terms, the element is assigned the special value NA. There is also a second kind of “missing” values which are produced by numerical computation, these are called NaN (Not a Number) values. Impossible values (e.g., dividing by zero) are also represented by the symbol NaN (not a number).

Determining Missing Values: The function isnull().sum() gives the total number of missing values for each column in the dataset

In [39]:
```
#Program to use all the data processing techniques in one dataset.
loandata=pd.read_csv ("loan.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
```

Deleting Observations containing Missing Values: It is possible to delete the observations from the dataset containing missing values in any column directly. The function dropna(inplace=True) deletes the observations that contain the missing values from the dataset and hence reduces the number of observations.

In [47]:
```
#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)
```

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

Missing Data Imputation: Imputation is a method to fill in the missing values with estimated ones. It is very important to impute the missing data before analysing because the data analysis functions does not work effectively if missing values exist in the dataset. This section focuses on imputation of missing data with different values. 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.

In [46]:
```
#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['LoanAmount'].fillna (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[ 'LoanAmount'].fillna (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[ 'LoanAmount'].fillna (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 ['Gender'].fillna (loan4 ['Gender'].mode () .iloc[0], inplace=True)
print ("Missing values in Gender:", loan4 ['Gender'].isnull().sum())
#Replacing the categorical variable "Married" with "Yes".
loan4 ['Married'].fillna ('Yes', inplace=True)
print ("Missing values in Married: ", loan4 ['Gender'].isnull().sum())
```

```
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
```

```
C:\Users\CDAC\AppData\Local\Temp\ipykernel_14752\1280627076.py:7: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan1['LoanAmount'].fillna (0, inplace=True)
C:\Users\CDAC\AppData\Local\Temp\ipykernel_14752\1280627076.py:14: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan2[ 'LoanAmount'].fillna (loan2 ['LoanAmount'].median (), inplace=True)
C:\Users\CDAC\AppData\Local\Temp\ipykernel_14752\1280627076.py:20: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan3[ 'LoanAmount'].fillna (loan3 [ 'LoanAmount'].mean (), inplace=True)
C:\Users\CDAC\AppData\Local\Temp\ipykernel_14752\1280627076.py:26: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan4 ['Gender'].fillna (loan4 ['Gender'].mode () .iloc[0], inplace=True)
C:\Users\CDAC\AppData\Local\Temp\ipykernel_14752\1280627076.py:29: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
```

```
the operation inplace on the original object.


  loan4 ['Married'].fillna ('Yes', inplace=True)
```

In [ ]:

```
```