Chapter 3
Data Handling using Pandas - II

> "We owe a lot to the Indians, who taught us how to count, without which no worthwhile scientific discovery could have been made."
>
> — Albert Einstein

12149CH03

3.1 INTRODUCTION

As discussed in the previous chapter, Pandas is a well established Python Library used for manipulation, processing and analysis of data. We have already discussed the basic operations on Series and DataFrame like creating them and then accessing data from them. Pandas provides more powerful and useful functions for data analysis.

In this chapter, we will be working with more advanced features of DataFrame like sorting data, answering analytical questions using the data, cleaning data and applying different useful functions on the data. Below is the example data on which we will be applying the advanced features of Pandas.

Case Study

Let us consider the data of marks scored in unit tests held in school. For each unit test, the marks scored by all students of the class is recorded. Maximum marks are 25 in each subject. The subjects are Maths, Science. Social Studies (S.St.), Hindi, and English. For simplicity, we assume there are 4 students in the class and the table below shows their marks in Unit Test 1, Unit Test 2 and Unit Test 3. Table 3.1 shows this data.

Table 3.1 Case Study

Result						
Name/ Subjects	Unit Test	Maths	Science	S.St.	Hindi	Eng
Raman	1	22	21	18	20	21
Raman	2	21	20	17	22	24
Raman	3	14	19	15	24	23
Zuhaire	1	20	17	22	24	19
Zuhaire	2	23	15	21	25	15
Zuhaire	3	22	18	19	23	13
Aashravy	1	23	19	20	15	22
Aashravy	2	24	22	24	17	21
Aashravy	3	12	25	19	21	23
Mishti	1	15	22	25	22	22
Mishti	2	18	21	25	24	23
Mishti	3	17	18	20	25	20

Let us store the data in a DataFrame, as shown in Program 3.1:

Program 3-1 Store the Result data in a DataFrame called marksUT.

```
>>> import pandas as pd
>>> marksUT= {'Name':['Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire', 'Ashravy','Ashravy','Ashravy','Mishti','Mishti','Mishti'],
            'UT':[1,2,3,1,2,3,1,2,3,1,2,3],
            'Maths':[22,21,14,20,23,22,23,24,12,15,18,17],
            'Science':[21,20,19,17,15,18,19,22,25,22,21,18],
            'S.St':[18,17,15,22,21,19,20,24,19,25,25,20],
            'Hindi':[20,22,24,24,25,23,15,17,21,22,24,25],
            'Eng':[21,24,23,19,15,13,22,21,23,22,23,20]
            }
>>> df=pd.DataFrame(marksUT)
>>> print(df)
```

```
        Name   UT   Maths   Science   S.St   Hindi   Eng
0       Raman   1    22       21       18      20      21
1       Raman   2    21       20       17      22      24
2       Raman   3    14       19       15      24      23
3     Zuhaire   1    20       17       22      24      19
4     Zuhaire   2    23       15       21      25      15
5     Zuhaire   3    22       18       19      23      13
6     Ashravy   1    23       19       20      15      22
7     Ashravy   2    24       22       24      17      21
8     Ashravy   3    12       25       19      21      23
9      Mishti   1    15       22       25      22      22
10     Mishti   2    18       21       25      24      23
11     Mishti   3    17       18       20      25      20
```

3.2 DESCRIPTIVE STATISTICS

Descriptive Statistics are used to summarise the given data. In other words, they refer to the methods which are used to get some basic idea about the data.

In this section, we will be discussing descriptive statistical methods that can be applied to a DataFrame. These are max, min, count, sum, mean, median, mode, quartiles, variance. In each case, we will consider the above created DataFrame df.

3.2.1 Calculating Maximum Values

DataFrame.max() is used to calculate the maximum values from the DataFrame, regardless of its data types. The following statement outputs the maximum value of each column of the DataFrame:

```
>>> print(df.max())
Name        Zuhaire      #Maximum value in name column
                         #(alphabetically)
UT                3      #Maximum value in column UT
Maths            24      #Maximum value in column Maths
Science          25      #Maximum value in column Science
S.St             25      #Maximum value in column S.St
Hindi            25      #Maximum value in column Hindi
Eng              24      #Maximum value in column Eng
dtype: object
```

If we want to output maximum value for the columns having only numeric values, then we can set the parameter numeric_only=True in the max() method, as shown below:

```
>>> print(df.max(numeric_only=True))
UT          3
Maths      24
Science    25
S.St       25
Hindi      25
Eng        24
dtype: int64
```

Program 3-2 Write the statements to output the maximum marks obtained in each subject in Unit Test 2.

```
>>> dfUT2 = df[df.UT == 2]
>>> print('\nResult of Unit Test 2:
\n\n',dfUT2)

Result of Unit Test 2:
        Name  UT  Maths  Science  S.St  Hindi  Eng
1      Raman   2     21       20    17     22   24
4    Zuhaire   2     23       15    21     25   15
7    Ashravy   2     24       22    24     17   21
10    Mishti   2     18       21    25     24   23

>>> print('\nMaximum Mark obtained in
Each Subject in Unit Test 2: \n\n',dfUT2.
max(numeric_only=True))

Maximum Mark obtained in Each Subject in Unit
Test 2:

UT          2
Maths      24
Science    22
S.St       25
Hindi      25
Eng        24
dtype: int64
```

> The output of Program 3.2 can also be achieved using the following statements
> ```
> >>> dfUT2=df[df
> ['UT']==2].max
> (numeric_only=True)
> >>> print(dfUT2)
> ```

By default, the max() method finds the maximum value of each column (which means, axis=0). However, to find the maximum value of each row, we have to specify axis = 1 as its argument.

```
#maximum marks for each student in each unit
test among all the subjects
```

```
>>> df.max(axis=1)
```

```
0      22
1      24
2      24
3      24
4      25
5      23
6      23
7      24
8      25
9      25
10     25
11     25
dtype: int64
```

Note: In most of the python function calls, axis = 0 refers to row wise operations and axis = 1 refers to column wise operations. But in the call of max(), axis = 1 gives row wise output and axis = 0 (default case) gives column-wise output. Similar is the case with all statistical operations discussed in this chapter.

3.2.2 Calculating Minimum Values

DataFrame.min() is used to display the minimum values from the DataFrame, regardless of the data types. That is, it shows the minimum value of each column or row. The following line of code output the minimum value of each column of the DataFrame:

```
>>> print(df.min())
Name        Ashravy
UT                1
Maths            12
Science          15
S.St             15
Hindi            15
Eng              13
dtype: object
```

Program 3-3   Write the statements to display the minimum marks obtained by a particular student 'Mishti' in all the unit tests for each subject.

```
>>> dfMishti = df.loc[df.Name == 'Mishti']
```

```
>>> print('\nMarks obtained by Mishti in all
the Unit Tests \n\n',dfMishti)
```

```
Marks obtained by Mishti in all the Unit Tests
       Name   UT   Maths   Science   S.St   Hindi   Eng
9    Mishti    1     15        22      25      22    22
10   Mishti    2     18        21      25      24    23
11   Mishti    3     17        18      20      25    20
```

```
>>> print('\nMinimum Marks obtained by
Mishti in each subject across the unit
tests\n\n', dfMishti[['Maths','Science','S.
St','Hindi','Eng']].min())
```

The output of Program 3.3 can also be achieved using the following statements
```
>>> dfMishti=df[['Maths','Science','S.St','Hindi','Eng']][df.Name == 'Mishti'].min()
>>> print(dfMishti)
```

Minimum Marks obtained by Mishti in each subject across the unit tests:

```
Maths      15
Science    18
S.St       20
Hindi      22
Eng        20
dtype: int64
```

Note: Since we did not want to output the min value of column UT, we mentioned all the other column names for which minimum is to be calculated.

3.2.3 Calculating Sum of Values

DataFrame.sum() will display the sum of the values from the DataFrame regardless of its datatype. The following line of code outputs the sum of each column of the DataFrame:

```
>>> print(df.sum())
Name
RamanRamanRamanZuhaireZuhaireZuhaireAshravyAsh...
UT                                            24
Maths                                        231
Science                                      237
S.St                                         245
Hindi                                        262
Eng                                          246
dtype: object
```

We may not be interested to sum text values. So, to print the sum of a particular column, we need to

specify the column name in the call to function sum. The following statement prints the total marks of subject mathematics:

```
>>> print(df['Maths'].sum())
231
```

To calculate total marks of a particular student, the name of the student needs to be specified.

Program 3-4 Write the python statement to print the total marks secured by raman in each subject.

```
>>> dfRaman=df[df['Name']=='Raman']
>>> print("Marks obtained by Raman in each test
are:\n", dfRaman)
Marks obtained by Raman in each test are:
     Name  UT  Maths  Science  S.St  Hindi  Eng
0   Raman   1     22       21    18     20   21
1   Raman   2     21       20    17     22   24
2   Raman   3     14       19    15     24   23


>>> dfRaman[['Maths','Science','S.
St','Hindi','Eng']].sum()

Maths      57
Science    60
S.St       50
Hindi      66
Eng        68
dtype: int64

#To print total marks scored by Raman in all
subjects in each Unit Test
>>> dfRaman[['Maths','Science','S.
St','Hindi','Eng']].sum(axis=1)
0    102
1    104
2     95
dtype: int64
```

Think and Reflect

Can you write a shortened code to get the output of Program 3.4?

Activity 3.1

Write the python statements to print the sum of the english marks scored by Mishti.

3.2.4 Calculating Number of Values

DataFrame.count() will display the total number of values for each column or row of a DataFrame. To count the rows we need to use the argument axis=1 as shown in the Program 3.5 below.

```
>>> print(df.count())

Name       12
UT         12
Maths      12
Science    12
S.St       12
Hindi      12
Eng        12
dtype: int64
```

Program 3-5 Write a statement to count the number of values in a row.

```
>>> df.count(axis=1)
0      7
1      7
2      7
3      7
4      7
5      7
6      7
7      7
8      7
9      7
10     7
11     7
dtype: int64
```

3.2.5 Calculating Mean

DataFrame.mean() will display the mean (average) of the values of each column of a DataFrame. It is only applicable for numeric values.

```
>>> df.mean()
UT          2.5000
Maths      18.6000
Science    19.8000
S.St       20.0000
Hindi      21.3125
Eng        19.8000
dtype: float64
```

Program 3-6 Write the statements to get an average of marks obtained by Zuhaire in all the Unit Tests.

```
>>> dfZuhaireMarks = dfZuhaire.
loc[:,'Maths':'Eng']
>>> print("Slicing of the DataFrame to get only
the marks\n", dfZuhaireMarks)

Slicing of the DataFrame to get only the marks
    Maths  Science  S.St  Hindi  Eng
3     20       17     22     24   19
4     23       15     21     25   15
5     22       18     19     23   13


>>> print("Average of marks obtained by
Zuhaire in all Unit Tests \n", dfZuhaireMarks.
mean(axis=1))

Average of marks obtained by Zuhaire in all
Unit Tests
3    20.4
4    19.8
5    19.0
dtype: float64
```

In the above output, 20.4 is the average of marks obtained by Zuhaire in Unit Test 1. Similarly, 19.8 and 19.0 are the average of marks in Unit Test 2 and 3 respectively.

3.2.6 Calculating Median

DataFrame.Median() will display the middle value of the data. This function will display the median of the values of each column of a DataFrame. It is only applicable for numeric values.

```
>>> print(df.median())

UT          2.5
Maths      19.0
Science    20.0
S.St       19.5
Hindi      21.5
Eng        21.0
dtype: float64
```

Program 3-7 Write the statements to print the median marks of mathematics in UT1.

```
>>> dfMaths=df['Maths']
```

Think and Reflect

Try to write a short code to get the above output. Remember to print the relevant headings of the output.

```
>>> dfMathsUT1=dfMaths[df.UT==1]
>>> print("Displaying the marks scored in
Mathematics in UT1\n",dfMathsUT1)
```

```
Displaying the marks of UT1, subject
Mathematics
0     22
3     20
6     23
9     15
Name: Maths, dtype: int64
```

```
>>> dfMathMedian=dfMathsUT1.median()
>>> print("Displaying the median of Mathematics
in UT1\n",dfMathMedian)
```

```
Displaying the median of Mathematics in UT1
21.0
```

Here, the number of values are even in number so two middle values are there i.e. 20 and 22. Hence, Median is the average of 20 and 22.

Activity 3.2

Find the median of the values of the rows of the DataFrame.

3.2.7 Calculating Mode

DateFrame.mode() will display the mode. The mode is defined as the value that appears the most number of times in a data. This function will display the mode of each column or row of the DataFrame. To get the mode of Hindi marks, the following statement can be used.

```
>>> df['Hindi']
0     20
1     22
2     24
3     24
4     25
5     23
6     15
7     17
8     21
9     22
10    24
11    25
Name: Hindi, dtype: int64
>>> df['Hindi'].mode()
```

Activity 3.3

Calculate the mode of marks scored in Maths.

```
0    24
dtype: int64
```

Note that three students have got 24 marks in Hindi subject while two students got 25 marks, one student got 23 marks, two students got 22 marks, one student each got 21, 20, 15, 17 marks.

3.2.8 Calculating Quartile

Dataframe.quantile() is used to get the quartiles. It will output the quartile of each column or row of the DataFrame in four parts i.e. the first quartile is 25% (parameter q = .25), the second quartile is 50% (Median), the third quartile is 75% (parameter q = .75). By default, it will display the second quantile (median) of all numeric values.

```
>>> df.quantile() # by default, median is the
output
UT           2.0
Maths        20.5
Science      19.5
S.St         20.0
Hindi        22.5
Eng          21.5
Name: 0.5, dtype: float64

>>> df.quantile(q=.25)
UT           1.00
Maths        16.50
Science      18.00
S.St         18.75
Hindi        20.75
Eng          19.75
Name: 0.25, dtype: float64

>>> df.quantile(q=.75)
UT           3.00
Maths        22.25
Science      21.25
S.St         22.50
Hindi        24.00
Eng          23.00
Name: 0.75, dtype: float64
```

Program 3-8 Write the statement to display the first and third quartiles of all subjects.

```
>>> dfSubject=df[['Maths','Science','S.
St','Hindi','Eng']]
>>> print("Marks of all the subjects:\
n",dfSubject)

Marks of all the subjects:
     Maths  Science  S.St  Hindi  Eng
0       22       21    18     20   21
1       21       20    17     22   24
2       14       19    15     24   23
3       20       17    22     24   19
4       23       15    21     25   15
5       22       18    19     23   13
6       23       19    20     15   22
7       24       22    24     17   21
8       12       25    19     21   23
9       15       22    25     22   22
10      18       21    25     24   23
11      17       18    20     25   20

>>> dfQ=dfSubject.quantile([.25,.75])
>>> print("First and third quartiles of all the
subjects:\n",dfQ)
```

First and third quartiles of all the subjects:

	Maths	Science	S.St	Hindi	Eng
0.25	16.50	18.00	18.75	20.75	19.75
0.75	22.25	21.25	22.50	24.00	23.00

3.2.9 Calculating Variance

DataFrame.var() is used to display the variance. It is the average of squared differences from the mean.

```
>>> df[['Maths','Science','S.
St','Hindi','Eng']].var()


Maths       15.840909
Science      7.113636
S.St         9.901515
```

Activity 3.4

Find the variance and standard deviation of the following scores on an exam: 92, 95, 85, 80, 75, 50.

```
Hindi        9.969697
Eng         11.363636
dtype: float64
```

3.2.10 Calculating Standard Deviation

DataFrame.std() returns the standard deviation of the values. Standard deviation is calculated as the square root of the variance.

```
>>> df[['Maths','Science','S.
St','Hindi','Eng']].std()
```

```
Maths        3.980064
Science      2.667140
S.St         3.146667
Hindi        3.157483
Eng          3.370999
dtype: float64
```

DataFrame.describe() function displays the descriptive statistical values in a single command. These values help us describe a set of data in a DataFrame.

```
>>> df.describe()
```

	UT	Maths	Science	S.St	Hindi	Eng
count	12.000000	12.000000	12.00000	12.000000	12.000000	12.000000
mean	2.000000	19.250000	19.75000	20.416667	21.833333	20.500000
std	0.852803	3.980064	2.66714	3.146667	3.157483	3.370999
min	1.000000	12.000000	15.00000	15.000000	15.000000	13.000000
25%	1.000000	16.500000	18.00000	18.750000	20.750000	19.750000
50%	2.000000	20.500000	19.50000	20.000000	22.500000	21.500000
75%	3.000000	22.250000	21.25000	22.500000	24.000000	23.000000
max	3.000000	24.000000	25.00000	25.000000	25.000000	24.000000

3.3 DATA AGGREGATIONS

Aggregation means to transform the dataset and produce a single numeric value from an array. Aggregation can be applied to one or more columns together. Aggregate functions are max(),min(), sum(), count(), std(), var().

```
>>> df.aggregate('max')
```

```
Name    Zuhaire # displaying the maximum of Name
as well
UT              3
Maths          24
```

```
Science          25
S.St             25
Hindi            25
Eng              24
dtype: object

#To use multiple aggregate functions in a
single statement
>>> df.aggregate(['max','count'])


       Name    UT  Maths Science S.St  Hindi  Eng
max   Zuhaire  3    24      25    25     25    24
count  12      12   12      12    12     12    12

>>> df['Maths'].aggregate(['max','min'])
max    24
min    12
Name: Maths, dtype: int64
```

Note: We can also use the parameter axis with aggregate function. By default, the value of axis is zero, means columns.

```
#Using the above statement with axis=0 gives
the same result
>>> df['Maths'].aggregate(['max','min'],axis=0)
max    24
min    12
Name: Maths, dtype: int64

#Total marks of Maths and Science obtained by
each student.
#Use sum() with axis=1 (Row-wise summation)
>>> df[['Maths','Science']].
aggregate('sum',axis=1)
0      43
1      41
2      33
3      37
4      38
5      40
6      42
7      46
8      37
9      37
10     39
11     35
dtype: int64
```

3.4 SORTING A DATAFRAME

Sorting refers to the arrangement of data elements in a specified order, which can either be ascending or descending. Pandas provide sort_values() function to sort the data values of a DataFrame. The syntax of the function is as follows:

```
DataFrame.sort_values(by, axis=0, ascending=True)
```

Here, a column list (by), axis arguments (0 for rows and 1 for columns) and the order of sorting (ascending = False or True) are passed as arguments. By default, sorting is done on row indexes in ascending order.

Consider a scenario, where the teacher is interested in arranging a list according to the names of the students or according to marks obtained in a particular subject. In such cases, sorting can be used to obtain the desired results. Following is the python code for sorting the data in the DataFrame created at program 3.1.

To sort the entire data on the basis of attribute 'Name', we use the following command:

```
#By default, sorting is done in ascending order.
>>> print(df.sort_values(by=['Name']))
```

```
        Name  UT  Maths  Science  S.St  Hindi  Eng
6    Ashravy   1     23       19    20     15   22
7    Ashravy   2     24       22    24     17   21
8    Ashravy   3     12       25    19     21   23
9     Mishti   1     15       22    25     22   22
10    Mishti   2     18       21    25     24   23
11    Mishti   3     17       18    20     25   20
0      Raman   1     22       21    18     20   21
1      Raman   2     21       20    17     22   24
2      Raman   3     14       19    15     24   23
3    Zuhaire   1     20       17    22     24   19
4    Zuhaire   2     23       15    21     25   15
5    Zuhaire   3     22       18    19     23   13
```

Now, to obtain sorted list of marks scored by all students in Science in Unit Test 2, the following code can be used:

```
# Get the data corresponding to Unit Test 2
>>> dfUT2 = df[df.UT == 2]
# Sort according to ascending order of marks in
Science
```

```
>>> print(dfUT2.sort_values(by=['Science']))
```

	Name	UT	Maths	Science	S.St	Hindi	Eng
4	Zuhaire	2	23	15	21	25	15
1	Raman	2	21	20	17	22	24
10	Mishti	2	18	21	25	24	23
7	Ashravy	2	24	22	24	17	21

Program 3-9 Write the statement which will sort the marks in English in the DataFrame df based on Unit Test 3, in descending order.

```
# Get the data corresponding to Unit Test 3
>>> dfUT3 = df[df.UT == 3]
# Sort according to descending order of marks in
Science
>>> print(dfUT3.sort_values(by=['Eng'],ascending=False))
```

	Name	UT	Maths	Science	S.St	Hindi	Eng
2	Raman	3	14	19	15	24	23
8	Ashravy	3	12	25	19	21	23
11	Mishti	3	17	18	20	25	20
5	Zuhaire	3	22	18	19	23	13

A DataFrame can be sorted based on multiple columns. Following is the code of sorting the DataFrame df based on marks in Science in Unit Test 3 in ascending order. If marks in Science are the same, then sorting will be done on the basis of marks in Hindi.

```
# Get the data corresponding to marks in Unit Test
3
>>> dfUT3 = df[df.UT == 3]
# Sort the data according to Science and then
according to Hindi
>>> print(dfUT3.sort_
values(by=['Science','Hindi']))
```

	Name	UT	Maths	Science	S.St	Hindi	Eng
5	Zuhaire	3	22	18	19	23	13
11	Mishti	3	17	18	20	25	20
2	Raman	3	14	19	15	24	23
8	Ashravy	3	12	25	19	21	23

Here, we can see that the list is sorted on the basis of marks in Science. Two students namely, Zuhaire and Mishti have equal marks (18) in Science. Therefore for them, sorting is done on the basis of marks in Hindi.

3.5 GROUP BY FUNCTIONS

In pandas, DataFrame.GROUP BY() function is used to split the data into groups based on some criteria. Pandas objects like a DataFrame can be split on any of their axes. The GROUP BY function works based on a split-apply-combine strategy which is shown below using a 3-step process:

Step 1: Split the data into groups by creating a GROUP BY object from the original DataFrame.

Step 2: Apply the required function.

Step 3: Combine the results to form a new DataFrame.

To understand this better, let us consider the data shown in the diagram given below. Here, we have a two-column DataFrame (key, data). We need to find the sum of the data column for a particular key, i.e. sum of all the data elements with key A, B and C, respectively. To do so, we first split the entire DataFrame into groups by key column. Then, we apply the sum function on the respective groups. Finally, we combine the results to form a new DataFrame that contains the desired result.
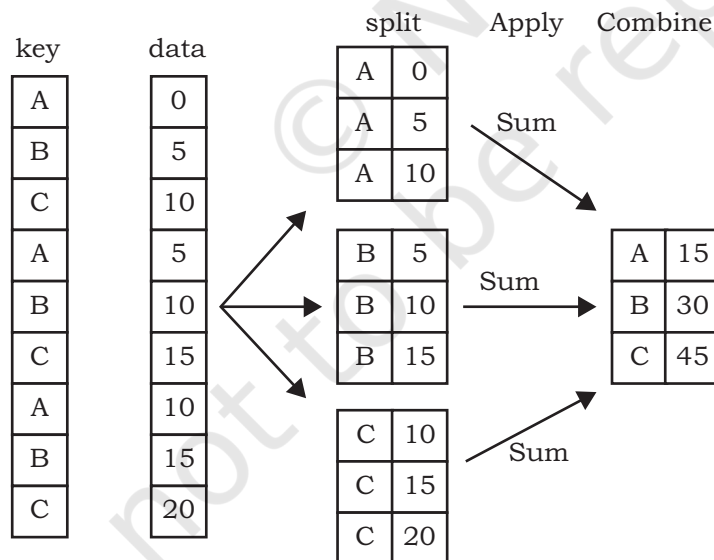

Figure 3.1: A DataFrame with two columns

The following statements show how to apply GROUP BY() function on our DataFrame df created at Program 3.1:

```
#Create a GROUP BY Name of the student from
DataFrame df
>>> g1=df.GROUP BY('Name')
```

```
#Displaying the first entry from each group
>>> g1.first()
         UT  Maths  Science  S.St  Hindi  Eng
Name
Ashravy   1    23       19    20     15   22
Mishti    1    15       22    25     22   22
Raman     1    22       21    18     20   21
Zuhaire   1    20       17    22     24   19


#Displaying the size of each group
>>> g1.size()
Name
Ashravy    3
Mishti     3
Raman      3
Zuhaire    3
dtype: int64


#Displaying group data, i.e., group_name, row
indexes corresponding to the group and their
data type
>>> g1.groups
{'Ashravy': Int64Index([6, 7, 8],
dtype='int64'),
 'Mishti': Int64Index([9, 10, 11],
dtype='int64'),
 'Raman': Int64Index([0, 1, 2], dtype='int64'),
 'Zuhaire': Int64Index([3, 4, 5],
dtype='int64')}


#Printing data of a single group
>>> g1.get_group('Raman')
   UT  Maths  Science  S.St  Hindi  Eng
0   1    22       21    18     20   21
1   2    21       20    17     22   24
2   3    14       19    15     24   23


#Grouping with respect to multiple attributes
#Creating a GROUP BY Name and UT

>>> g2=df.GROUP BY(['Name', 'UT'])

>>> g2.first()
```

```
          Maths   Science   S.St   Hindi   Eng
Name    UT
Ashravy 1      23       19      20      15     22
        2      24       22      24      17     21
        3      12       25      19      21     23
Mishti  1      15       22      25      22     22
        2      18       21      25      24     23
        3      17       18      20      25     20
Raman   1      22       21      18      20     21
        2      21       20      17      22     24
        3      14       19      15      24     23
Zuhaire 1      20       17      22      24     19
        2      23       15      21      25     15
        3      22       18      19      23     13
```

The above statements show how we create groups by splitting a DataFrame using GROUP BY(). Next step is to apply functions over the groups just created. This is done using Aggregation.

Aggregation is a process in which an aggregate function is applied on each group created by GROUP BY(). It returns a single aggregated statistical value corresponding to each group. It can be used to apply multiple functions over an axis. Be default, functions are applied over columns. Aggregation can be performed using agg() or aggregate() function.

```
#Calculating average marks scored by all
students in each subject for each UT
>>> df.GROUP BY(['UT']).aggregate('mean')

     Maths   Science   S.St   Hindi    Eng
UT
1    20.00    19.75   21.25   20.25   21.00
2    21.50    19.50   21.75   22.00   20.75
3    16.25    20.00   18.25   23.25   19.75

#Calculate average marks scored in Maths in
each UT
>>> group1=df.GROUP BY(['UT'])
>>> group1['Maths'].aggregate('mean')
UT
1    20.00
2    21.50
3    16.25
Name: Maths, dtype: float64
```

Program 3-10 Write the python statements to print the mean, variance, standard deviation and quartile of the marks scored in Mathematics by each student across the UTs.

```
>>> df.GROUP BY(by='Name')['Maths'].agg(['mean','var','std','quantile'])
```

```
              mean          var       std      quantile
Name
Ashravy  19.666667   44.333333   6.658328      23.0
Mishti   16.666667    2.333333   1.527525      17.0
Raman    19.000000   19.000000   4.358899      21.0
Zuhaire  21.666667    2.333333   1.527525      22.0
```

Activity 3.5

Write the python statements to print average marks in Science by all the students in each UT.

3.6 ALTERING THE INDEX

We use indexing to access the elements of a DataFrame. It is used for fast retrieval of data. By default, a numeric index starting from 0 is created as a row index, as shown below:

```
>>> df                      #With default Index
         Name   UT   Maths  Science  S.St  Hindi  Eng
0       Raman    1     22       21    18     20    21
1       Raman    2     21       20    17     22    24
2       Raman    3     14       19    15     24    23
3     Zuhaire    1     20       17    22     24    19
4     Zuhaire    2     23       15    21     25    15
5     Zuhaire    3     22       18    19     23    13
6     Ashravy    1     23       19    20     15    22
7     Ashravy    2     24       22    24     17    21
8     Ashravy    3     12       25    19     21    23
9      Mishti    1     15       22    25     22    22
10     Mishti    2     18       21    25     24    23
11     Mishti    3     17       18    20     25    20
```

Here, the integer number in the first column starting from 0 is the index. However, depending on our requirements, we can select some other column to be the index or we can add another index column.

When we slice the data, we get the original index which is not continuous, e.g. when we select marks of all students in Unit Test 1, we get the following result:

```
>>> dfUT1 = df[df.UT == 1]
>>> print(dfUT1)
```

```
        Name  UT  Maths  Science  S.St  Hindi  Eng
0      Raman   1     22       21    18     20   21
3    Zuhaire   1     20       17    22     24   19
6    Ashravy   1     23       19    20     15   22
9     Mishti   1     15       22    25     22   22
```

Notice that the first column is a non-continuous index since it is slicing of original data. We create a new continuous index alongside this using the reset_index() function, as shown below:

```
>>> dfUT1.reset_index(inplace=True)
>>> print(dfUT1)
   index     Name  UT  Maths  Science  S.St  Hindi  Eng
0      0    Raman   1     22       21    18     20   21
1      3  Zuhaire   1     20       17    22     24   19
2      6  Ashravy   1     23       19    20     15   22
3      9   Mishti   1     15       22    25     22   22
```

A new continuous index is created while the original one is also intact. We can drop the original index by using the drop function, as shown below:

```
>>> dfUT1.drop(columns=['index'],inplace=True)
>>> print(dfUT1)

        Name  UT  Maths  Science  S.St  Hindi  Eng
0      Raman   1     22       21    18     20   21
1    Zuhaire   1     20       17    22     24   19
2    Ashravy   1     23       19    20     15   22
3     Mishti   1     15       22    25     22   22
```

We can change the index to some other column of the data.

```
>>> dfUT1.set_index('Name',inplace=True)
>>> print(dfUT1)
         UT  Maths  Science  S.St  Hindi  Eng
Name
Raman     1     22       21    18     20   21
Zuhaire   1     20       17    22     24   19
Ashravy   1     23       19    20     15   22
Mishti    1     15       22    25     22   22
```

We can revert back to previous index by using following statement:

```
>>> dfUT1.reset_index('Name', inplace = True)
>>> print(dfUT1)
```

```
        Name  UT  Maths  Science  S.St  Hindi  Eng
0     Raman   1     22       21    18     20   21
1   Zuhaire   1     20       17    22     24   19
2   Ashravy   1     23       19    20     15   22
3    Mishti   1     15       22    25     22   22
```

3.7 OTHER DATAFRAME OPERATIONS

In this section, we will learn more techniques and functions that can be used to manipulate and analyse data in a DataFrame.

3.7.1 Reshaping Data

The way a dataset is arranged into rows and columns is referred to as the shape of data. Reshaping data refers to the process of changing the shape of the dataset to make it suitable for some analysis problems. The example given in the below section explains the utility of reshaping the data.

For reshaping data, two basic functions are available in Pandas, pivot and pivot_table. This section covers them in detail.

(A) Pivot

The pivot function is used to reshape and create a new DataFrame from the original one. Consider the following example of sales and profit data of four stores: S1, S2, S3 and S4 for the years 2016, 2017 and 2018.

Example 3.1

```
>>> import pandas as pd

>>> data={'Store':['S1','S4','S3','S1','S2','S3
','S1','S2','S3'],  'Year':[2016,2016,2016,2017
,2017,2017,2018,2018,2018],
'Total_sales(Rs)':[12000,330000,420000,
20000,10000,450000,30000, 11000,89000],
'Total profit(
Rs)':[1100,5500,21000,32000,9000,45000,3000,
1900,23000]
}

>>> df=pd.DataFrame(data)
>>> print(df)
```

```
    Store    Year    Total_sales(Rs)    Total_profit(Rs)
0     S1     2016          12000                1100
1     S4     2016         330000                5500
2     S3     2016         420000               21000
```

```
3    S1    2017              20000              32000
4    S2    2017              10000               9000
5    S3    2017             450000              45000
6    S1    2018              30000               3000
7    S2    2018              11000               1900
8    S3    2018              89000              23000
```

Let us try to answer the following queries on the above data.

1) What was the total sale of store S1 in all the years? Python statements to perform this task will be as follows:

```
    # will get the data related to store S1
>>> S1df = df[df.Store=='S1']
#find the total of sales for Store S1
>>> S1df['Total_sales(Rs)'].sum()
62000
```

2) What is the maximum sale value by store S3 in any year?

```
#will get the data related to store S3
>>> S3df = df[df.Store=='S3']
#find the maximum sale for Store S3
>>> S3df['Total_sales(Rs)'].max()
450000
```

3) Which store had the maximum total sale in all the years?

```
>>> S1df = df[df.Store=='S1']
>>> S2df=df[df.Store == 'S2']
>>> S3df = df[df.Store=='S3']
>>> S4df = df[df.Store=='S4']
>>> S1total = S1df['Total_sales(Rs)'].sum()
>>> S2total = S2df['Total_sales(Rs)'].sum()
>>> S3total = S3df['Total_sales(Rs)'].sum()
>>> S4total = S4df['Total_sales(Rs)'].sum()
>>> max(S1total,S2total,S3total,S4total)
959000
```

Notice that we have to slice the data corresponding to a particular store and then answer the query. Now, let us reshape the data using pivot and see the difference.

```
>>>
pivot1=df.pivot(index='Store',columns='Year',va
lues='Total_sales(Rs)')
```

Here, Index specifies the columns that will be acting as an index in the pivot table, columns specifies the new columns for the pivoted data and values specifies columns whose values will be displayed. In this particular case, store names will act as index, year will be the headers for columns and sales value will be displayed as values of the pivot table.

```
>>> print(pivot1)

Year          2016       2017      2018
Store
S1         12000.0    20000.0   30000.0
S2             NaN    10000.0   11000.0
S3        420000.0   450000.0   89000.0
S4        330000.0       NaN       NaN
```

As can be seen above, the value of Total_sales (Rs) for every row in the original table has been transferred to the new table: pivot1, where each row has data of a store and each column has data of a year. Those cells in the new pivot table which do not have a matching entry in the original one are filled with NaN. For instance, we did not have values corresponding to sales of Store S2 in 2016, thus the appropriate cell in pivot1 is filled with NaN.

Now the python statements for the above queries will be as follows:

1) What was the total sale of store S1 in all the years?

```
>>> pivot1.loc['S1'].sum()
```

2) What is the maximum sale value by store S3 in any year?

```
>>> pivot1.loc['S3'].max()
```

3) Which store had the maximum total sale?

```
>>> S1total = pivot1.loc['S1'].sum()
>>> S2total = pivot1.loc['S2'].sum()
>>> S3total = pivot1.loc['S3'].sum()
>>> S4total = pivot1.loc['S4'].sum()
>>> max(S1total,S2total,S3total,S4total)
```

Activity 3.6

Consider the data of unit test marks given at program 3.1, write the python statements to print name wise UT marks in mathematics.

We can notice that reshaping has transformed the structure of the data, which makes it more readable and easy to analyse the data.

(B) Pivoting by Multiple Columns

For pivoting by multiple columns, we need to specify multiple column names in the values parameter of

pivot() function. If we omit the values parameter, it will display the pivoting for all the numeric values.

```
>>> pivot2=df.pivot(index='Store',columns='Year
',values=['Total_sales(Rs)','Total_profit(Rs)'])
```

```
>>> print(pivot2)
```

	Total_sales(Rs)			Total_profit(Rs)		
Year	2016	2017	2018	2016	2017	2018
Store						
S1	12000.0	20000.0	30000.0	1100.0	32000.0	3000.0
S2	NaN	10000.0	11000.0	NaN	9000.0	1900.0
S3	330000.0	NaN	NaN	5500.0	NaN	NaN

Let us consider another example, where suppose we have stock data corresponding to a store as:

```
>>> data={'Item':['Pen','Pen','Pencil','Pencil'
,'Pen','Pen'],
'Color':['Red','Red','Black','Black','Blue','B
lue'],
'Price(Rs)':[10,25,7,5,50,20],
'Units_in_stock':[50,10,47,34,55,14]
}
>>> df=pd.DataFrame(data)
>>> print(df)
```

	Item	Color	Price(Rs)	Units_in_stock
0	Pen	Red	10	50
1	Pen	Red	25	10
2	Pencil	Black	7	47
3	Pencil	Black	5	34
4	Pen	Blue	50	55
5	Pen	Blue	20	14

Now, let us assume, we have to reshape the above table with Item as the index and Color as the column. We will use pivot function as given below:

```
>>> pivot3=df.pivot(index='Item',columns='Color
',values='Units_in_stock')
```

But this statement results in an error: "ValueError: Index contains duplicate entries, cannot reshape". This is because duplicate data can't be reshaped using pivot function. Hence, before calling the pivot() function, we need to ensure that our data do not have rows with duplicate values for the specified columns. If we can't ensure this, we may have to use pivot_table function instead.

(C) Pivot Table

It works like a pivot function, but aggregates the values from rows with duplicate entries for the specified columns. In other words, we can use aggregate functions like min, max, mean etc, wherever we have duplicate entries. The default aggregate function is mean.

Syntax:

```
pandas.pivot_table(data, values=None,
index=None, columns=None, aggfunc='mean')
```

The parameter aggfunc can have values among sum, max, min, len, np.mean, np.median.

We can apply index to multiple columns if we don't have any unique column to act as index.

```
>>> df1 = df.pivot_
table(index=['Item','Color'])
>>> print(df1)
                Price(Rs)  Units_in_stock
Item    Color
Pen     Blue      35.0            34.5
        Red       17.5            30.0
Pencil Black       6.0            40.5
```

Please note that mean has been used as the default aggregate function. Price of the blue pen in the original data is 50 and 20. Mean has been used as aggregate and the price of the blue pen is 35 in df1.

We can use multiple aggregate functions on the data. Below example shows the use of the sum, max and np.mean function.

```
>>> pivot_table1=df.pivot_table(index='
Item',columns='Color',values='Units_in_
stock',aggfunc=[sum,max,np.mean])
```

```
>>> pivot_table1
```

	sum			max			mean		
Color	Black	Blue	Red	Black	Blue	Red	Black	Blue	Red
Item									
Pen	NaN	69.0	60.0	NaN	55.0	50.0	NaN	34.5	30.0
Pencil	81.0	NaN	NaN	47.0	NaN	NaN	40.5	NaN	NaN

Pivoting can also be done on multiple columns. Further, different aggregate functions can be applied on different columns. The following example demonstrates pivoting on two columns - Price(Rs) and Units_in_stock. Also, the application of len() function on the column

Price(Rs) and mean() function of column Units_in_ stock is shown in the example. Note that the aggregate function len returns the number of rows corresponding to that entry.

```
>>> pivot_table1=df.pivot_table(index='Item'
,columns='Color',values=['Price(Rs)','Units_
in_stock'],aggfunc={"Price(Rs)":len,"Units_in_
stock":np.mean})
```

```
>>> pivot_table1
              Price(Rs)          Units_in_stock
Color      Black Blue  Red      Black  Blue   Red
Item
Pen          NaN  2.0  2.0       NaN  34.5  30.0
Pencil       2.0  NaN  NaN      40.5   NaN   NaN
```

Program 3-11 Write the statement to print the maximum price of pen of each color.

```
>>> dfpen=df[df.Item=='Pen']
```

```
>>> pivot_redpen=dfpen.pivot_table(index='Item'
,columns=['Color'],values=['Price(Rs)'],aggfun
c=[max])
```

```
>>> print(pivot_redpen)
```

```
            max
         Price(Rs)
Color    Blue  Red
Item
Pen        50   25
```

3.8 HANDLING MISSING VALUES

As we know that a DataFrame can consist of many rows (objects) where each row can have values for various columns (attributes). If a value corresponding to a column is not present, it is considered to be a missing value. A missing value is denoted by NaN.

In the real world dataset, it is common for an object to have some missing attributes. There may be several reasons for that. In some cases, data was not collected properly resulting in missing data e.g some people did not fill all the fields while taking the survey. Sometimes, some attributes are not relevant to all. For example, if a person is unemployed then salary attribute will be irrelevant and hence may not have been filled up.

Missing values create a lot of problems during data analysis and have to be handled properly. The two most common strategies for handling missing values explained in this section are:

i) drop the object having missing values,
ii) fill or estimate the missing value

Let us refer to the previous case study given at table 3.1. Suppose, the students have now appeared for Unit Test 4 also. But, Raman could not appear for the Science, Maths and English tests, and suppose there is no possibility of a re-test. Therefore, marks obtained by him corresponding to these subjects will be missing. The dataset after Unit Test 4 is as shown at Table 3.2. Note that the attributes 'Science, 'Maths' and 'English' have missing values in Unit Test 4 for Raman.

Table 3.2 Case study data after UT4

Result						
Name/ Subjects	Unit Test	Maths	Science	S.St.	Hindi	Eng
Raman	1	22	21	18	20	21
Raman	2	21	20	17	22	24
Raman	3	14	19	15	24	23
Raman	4			19	18	
Zuhaire	1	20	17	22	24	19
Zuhaire	2	23	15	21	25	15
Zuhaire	3	22	18	19	23	13
Zuhaire	4	19	20	17	19	16
Aashravy	1	23	19	20	15	22
Aashravy	2	24	22	24	17	21
Aashravy	3	12	25	19	21	23
Aashravy	4	15	20	20	20	17
Mishti	1	15	22	25	22	22
Mishti	2	18	21	25	24	23
Mishti	3	17	18	20	25	20
Mishti	4	14	20	19	20	18

To calculate the final result, teachers are asked to submit the percentage of marks obtained by all students. In the case of Raman, the Maths teacher decides to compute the marks obtained in 3 tests and then find the percentage of marks from the total score of 75 marks. In a way, she decides to drop the marks of Unit Test 4. However, the English teacher decides to give the same

marks to Raman in the 4th test as scored in the 3rd test. Science teacher decides to give Raman zero marks in the 4th test and then computes the percentage of marks obtained. Following sections explain the code for checking missing values and the code for replacing those missing values with appropriate values.

3.8.1 Checking Missing Values

Pandas provide a function isnull() to check whether any value is missing or not in the DataFrame. This function checks all attributes and returns True in case that attribute has missing values, otherwise returns False.

The following code stores the data of marks of all the Unit Tests in a DataFrame and checks whether the DataFrame has missing values or not.

```
>>> marksUT = {
'Name':['Raman','Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire'
,'Zuhaire','Ashravy','Ashravy','Ashravy','Ashravy','Mishti','Mishti',
'Mishti','Mishti'],
'UT':[1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4],
'Maths':[22,21,14,np.NaN,20,23,22,19,23,24,12,15,15,18,17,14],
'Science':[21,20,19,np.NaN,17,15,18,20,19,22,25,20,22,21,18,20],
'S.St':[18,17,15,19,22,21,19,17,20,24,19,20,25,25,20,19],
'Hindi':[20,22,24,18,24,25,23,21, 15,17,21,20,22,24,25,20],
'Eng':[21,24,23,np.NaN,19,15,13,16,22,21,23,17,22,23,20,18]        }
>>> df = pd.DataFrame(marksUT)
>>> print(df.isnull())
```

Output of the above code will be

	Name	UT	Maths	Science	S.St	Hindi	Eng
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	True	True	False	False	True
4	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False

One can check for each individual attribute also, e.g. the following statement checks whether attribute 'Science' has a missing value or not. It returns True for each row where there is a missing value for attribute 'Science', and False otherwise.

```
>>> print(df['Science'].isnull())
0      False
1      False
2      False
3       True
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12     False
13     False
14     False
15     False
Name: Science, dtype: bool
```

To check whether a column (attribute) has a missing value in the entire dataset, any() function is used. It returns True in case of missing value else returns False.

```
>>> print(df.isnull().any())
Name        False
UT          False
Maths        True
Science      True
S.St        False
Hindi       False
Eng          True
dtype: bool
```

The function any() can be used for a particular attribute also. The following statements) returns True in case an attribute has a missing value else it returns False.

```
>>> print(df['Science'].isnull().any())
True
```

```
>>> print(df['Hindi'].isnull().any())
False
```

To find the number of NaN values corresponding to each attribute, one can use the sum() function along with isnull() function, as shown below:

```
>>> print(df.isnull().sum())
Name       0
UT         0
Maths      1
Science    1
S.St       0
Hindi      0
Eng        1
dtype: int64
```

To find the total number of NaN in the whole dataset, one can use df.isnull().sum().sum().

```
>>> print(df.isnull().sum().sum())
3
```

Program 3-12 Write a program to find the percentage of marks scored by Raman in hindi.

```
>>> dfRaman = df[df['Name']=='Raman']
>>> print('Marks Scored by Raman \n\n',dfRaman)
```

```
Marks Scored by Raman
     Name  UT  Maths  Science  S.St  Hindi   Eng
0   Raman   1   22.0     21.0    18     20  21.0
1   Raman   2   21.0     20.0    17     22  24.0
2   Raman   3   14.0     19.0    15     24  23.0
3   Raman   4    NaN      NaN    19     18   NaN
```

```
>>> dfHindi = dfRaman['Hindi']
>>> print("Marks Scored by Raman in Hindi
\n\n",dfHindi)
```

Marks Scored by Raman in Hindi

```
0    20
1    22
2    24
3    18
Name: Hindi, dtype: int64
```

```
>>> row = len(dfHindi)  # Number of Unit Tests
held. Here row will be 4
```

```
>>> print("Percentage of Marks Scored by Raman
in Hindi\n\n",(dfHindi.sum()*100)/(25*row),"%")


# denominator in the above formula represents
the aggregate of marks of all tests. Here row
is 4 tests and 25 is maximum marks for one test
```

Percentage of Marks Scored by Raman in Hindi

```
84.0 %
```

Program 3-13 Write a python program to find the percentage of marks obtained by Raman in Maths subject.

```
>>> dfMaths = dfRaman['Maths']
```

```
>>> print("Marks Scored by Raman in Maths
\n\n",dfMaths)
```

```
Marks Scored by Raman in Maths
0    22.0
1    21.0
2    14.0
3    NaN
Name: Maths, dtype: float64
```

```
>>> row = len(dfMaths) # here, row will be 4,
the number of Unit Tests
```

```
>>> print("Percentage of Marks Scored by Raman
in Maths\n\n", dfMaths.sum()*100/(25*row),"%")
```

Percentage of Marks Scored by Raman in Maths

```
57%
```

Here, notice that Raman was absent in Unit Test 4 in Maths Subject. While computing the percentage, marks of the fourth test have been considered as 0.

3.8.2 Dropping Missing Values

Missing values can be handled by either dropping the entire row having missing value or replacing it with appropriate value.

Dropping will remove the entire row (object) having the missing value(s). This strategy reduces the size of the dataset used in data analysis, hence should be used in case of missing values on few objects. The dropna() function can be used to drop an entire row from the DataFrame. For example, calling dropna() function on

the previous example will remove the 4th row having
NaN value.

```
>>> df1 = df.dropna()
>>> print(df1)
        Name   UT   Maths   Science   S.St   Hindi   Eng
0      Raman    1    22.0      21.0     18      20   21.0
1      Raman    2    21.0      20.0     17      22   24.0
2      Raman    3    14.0      19.0     15      24   23.0
4    Zuhaire    1    20.0      17.0     22      24   19.0
5    Zuhaire    2    23.0      15.0     21      25   15.0
6    Zuhaire    3    22.0      18.0     19      23   13.0
7    Zuhaire    4    19.0      20.0     17      21   16.0
8    Ashravy    1    23.0      19.0     20      15   22.0
9    Ashravy    2    24.0      22.0     24      17   21.0
10   Ashravy    3    12.0      25.0     19      21   23.0
11   Ashravy    4    15.0      20.0     20      20   17.0
12    Mishti    1    15.0      22.0     25      22   22.0
13    Mishti    2    18.0      21.0     25      24   23.0
14    Mishti    3    17.0      18.0     20      25   20.0
15    Mishti    4    14.0      20.0     19      20   18.0
```

 Now, let us consider the following code:

```
# marks obtained by Raman in all the unit tests
>>> dfRaman=df[df.Name=='Raman']

# inplace=true makes changes in the #original
DataFrame i.e. dfRaman #here
>>> dfRaman.dropna(inplace=True,how='any')
>>> dfMaths = dfRaman['Maths'] # get the marks
scored in Maths
>>> print("\nMarks Scored by Raman in Maths
\n",dfMaths)
```

Marks Scored by Raman in Maths

```
0     22.0
1     21.0
2     14.0
3      NaN
Name: Maths, dtype: float64

>>> row = len(dfMaths)
>>> print("\nPercentage of Marks Scored by
Raman in Maths\n")
>>> print(dfMaths.sum()*100/(25*row),"%")
```

Percentage of Marks Scored by Raman in Maths

```
76.0 %
```

Note that the number of rows in dfRaman is 3 after using dropna. Hence percentage is computed from marks obtained in 3 Unit Tests.

3.8.3 Estimating Missing Values

Missing values can be filled by using estimations or approximations e.g a value just before (or after) the missing value, average/minimum/maximum of the values of that attribute, etc. In some cases, missing values are replaced by zeros (or ones).

The fillna(num) function can be used to replace missing value(s) by the value specified in num. For example, fillna(0) replaces missing value by 0. Similarly fillna(1) replaces missing value by 1. Following code replaces missing values by 0 and computes the percentage of marks scored by Raman in Science.

```
#Marks Scored by Raman in all the subjects
across the tests
>>> dfRaman = df.loc[df['Name']=='Raman']

>>> (row,col) = dfRaman.shape
>>> dfScience = dfRaman.loc[:,'Science']
>>> print("Marks Scored by Raman in Science
\n\n",dfScience)


Marks Scored by Raman in Science


0     21.0
1     20.0
2     19.0
3      NaN
Name: Science, dtype: float64


>>> dfFillZeroScience = dfScience.fillna(0)
>>> print('\nMarks Scored by Raman in Science
with Missing Values Replaced with Zero\
n',dfFillZeroScience)
```

Marks Scored by Raman in Science with Missing Values Replaced with Zero

```
0     21.0
1     20.0
```

```
2    19.0
3     0.0
Name: Science, dtype: float64
```

```
>>> print("Percentage of Marks Scored by Raman
in Science\n\n",dfFillZeroScience.sum()*100/
(25*row),"%")
```

Percentage of Marks Scored by Raman in Science

```
60.0 %
```

df.fillna(method='pad') replaces the missing value by the value before the missing value while df.fillna(method='bfill') replaces the missing value by the value after the missing value. Following code replaces the missing value in Unit Test 4 of English test by the marks of Unit Test 3 and then computes the percentage of marks obtained by Raman.

```
>>> dfEng = dfRaman.loc[:,'Eng']
>>> print("Marks Scored by Raman in English
\n\n",dfEng)
```

Marks Scored by Raman in English

```
0    21.0
1    24.0
2    23.0
3     NaN
Name: Eng, dtype: float64
```

```
>>>  dfFillPadEng = dfEng.fillna(method='pad')
>>> print('\nMarks Scored by Raman in English
with Missing Values Replaced by Previous Test
Marks\n',dfFillPadEng)
```

Marks Scored by Raman in English with Missing Values Replaced by Previous Test Marks

```
0    21.0
1    24.0
2    23.0
3    23.0
Name: Eng, dtype: float64
```

```
>>> print("Percentage of Marks Scored by Raman
in English\n\n")
>>> print(dfFillPadEng.sum()*100/(25*row),"%")
```

Percentage of Marks Scored by Raman in English

```
91.0 %
```

In this section, we have discussed various ways of handling missing values. Missing value is loss of

information and replacing missing values by some estimation will surely change the dataset. In all cases, data analysis results will not be actual results but will be a good approximation of actual results.

3.9 IMPORT AND EXPORT OF DATA BETWEEN PANDAS AND MYSQL

So far, we have directly entered data and created a DataFrame and learned how to analyse data in a DataFrame. However, in actual scenarios, data need not be typed or copy pasted everytime. Rather, data is available most of the time in a file (text or csv) or in a database. Thus, in real-world scenarios, we will be required to bring data directly from a database and load to a DataFrame. This is called importing data from a database. Likewise, after analysis, we will be required to store data back to a database. This is called exporting data to a database.

Data from DataFrame can be read from and written to MySQL database. To do this, a connection is required with the MySQL database using the pymysql database driver. And for this, the driver should be installed in the python environment using the following command:

```
pip install pymysql
```

sqlalchemy is a library used to interact with the MySQL database by providing the required credentials. This library can be installed using the following command:

```
pip install sqlalchemy
```

Once it is installed, sqlalchemy provides a function create_engine() that enables this connection to be established. The string inside the function is known as connection string. The connection string is composed of multiple parameters like the name of the database with which we want to establish the connection, username, password, host, port number and finally the name of the database. And, this function returns an engine object based on this connection string. The syntax for the same is discussed below:

```
engine=create_engine('driver://
username:password@host:port/name_of_
database',index=false)
```

where,

Driver = mysql+pymysql

username=User name of the mysql (normally it is root)

password= Password of the MySql

port = usually we connect to localhost with port number 3306 (Default port number)

Name of the Database = Your database

In the following subsections, importing and exporting data between Pandas and MySQL applications are demonstrated. For this, we will use the same database CARSHOWROOM and Table INVENTORY created in Chapter 1 of this book.

```
mysql> use CARSHOWROOM ;
Database changed
mysql> select * from INVENTORY;
+-------+---------+-----------+----------+-----------------+----------+
| CarId | CarName | Price     | Model    | YearManufacture | Fueltype |
+-------+---------+-----------+----------+-----------------+----------+
| D001  | Car1    | 582613.00 | LXI      |            2017 | Petrol   |
| D002  | Car1    | 673112.00 | VXI      |            2018 | Petrol   |
| B001  | Car2    | 567031.00 | Sigma1.2 |            2019 | Petrol   |
| B002  | Car2    | 647858.00 | Delta1.2 |            2018 | Petrol   |
| E001  | Car3    | 355205.00 | 5 STR STD|            2017 | CNG      |
| E002  | Car3    | 654914.00 | CARE     |            2018 | CNG      |
| S001  | Car4    | 514000.00 | LXI      |            2017 | Petrol   |
| S002  | Car4    | 614000.00 | VXI      |            2018 | Petrol   |
+-------+---------+-----------+----------+-----------------+----------+
8 rows in set (0.00 sec)
```

3.9.1 Importing Data from MySQL to Pandas

Importing data from MySQL to pandas basically refers to the process of reading a table from MySQL database and loading it to a pandas DataFrame. After establishing the connection, in order to fetch data from the table of the database we have the following three functions:

1) `pandas.read_sql_query(query,sql_conn)`

   It is used to read an sql query (query) into a DataFrame using the connection identifier (sql_conn) returned from the create_engine ().

2) `pandas.read_sql_table(table_name,sql_conn)`

   It is used to read an sql table (table_name) into a DataFrame using the connection identifier (sql_conn).

3) `pandas.read_sql(sql, sql_conn)`

   It is used to read either an sql query or an sql table (sql) into a DataFrame using the connection identifier (sql_conn).

```
>>> import pandas as pd
>>> import pymysql as py
>>> import sqlalchemy
>>> engine=create_engine('mysql+pymysql://
root:smsmb@localhost:3306/CARSHOWROOM')
>>> df = pd.read_sql_query('SELECT * FROM
INVENTORY', engine)
>>> print(df)
```

	CarId	CarName	Price	Model	YearManufacture	Fueltype
0	D001	Car1	582613.00	LXI	2017	Petrol
1	D002	Car1	673112.00	VXI	2018	Petrol
2	B001	Car2	567031.00	Sigma1.2	2019	Petrol
3	B002	Car2	647858.00	Delta1.2	2018	Petrol
4	E001	Car3	355205.00	5STR STD	2017	CNG
5	E002	Car3	654914.00	CARE	2018	CNG
6	S001	Car4	514000.00	LXI	2017	Petrol
7	S002	Car4	614000.00	VXI	2018``	Petrol

3.9.2 Exporting Data from Pandas to MySQL

Exporting data from Pandas to MySQL basically refers to the process of writing a pandas DataFrame to a table of MySQL database. For this purpose, we have the following function:

```
pandas.DataFrame.to_sql(table,sql_conn,if_
exists="fail",index=False/True)
```

- Table specifies the name of the table in which we want to create or append DataFrame values. It is used to write the specified DataFrame to the table the connection identifier (sql_conn) returned from the create_engine ().

- The parameter if_exists specifies "the way data from the DataFrame should be entered in the table. It can have the following three values: "fail", "replace", "append".

  o "fail" is the default value that indicates a ValueError if the table already exists in the database.

  o "replace" specifies that the previous content of the table should be updated by the contents of the DataFrame.

  o "append" specifies that the contents of the DataFrame should be appended to the existing table and when updated the format must be the same (column name sequences).

NOTES

- Index — By default index is True means DataFrame index will be copied to MySQL table. If False, then it will ignore the DataFrame indexing.

```
#Code to write DataFrame df to database

>>> import pandas as pd
>>> import pymysql as py
>>> import sqlalchemy
>>> engine=create_engine('mysql+pymysql://
root:smsmb@localhost:3306/CARSHOWROOM')
>>> data={
'ShowRoomId':[1,2,3,4,5],
'Location':['Delhi','Bangalore','Mumbai','Chand
igarh','Kerala']}

>>> df=pd.DataFrame(data)
>>> df.to_sql('showroom_info',engine,if_
exists="replace",index=False)
```

After running this python script, a mysql table with the name "showroom_info" will be created in the database.

SUMMARY

- Descriptive Statistics are used to quantitatively summarise the given data.
- Pandas provide many statistical functions for analysis of data. Some of the functions are max(), min(), mean(), median(), mode(), std(), var() etc.
- Sorting is used to arrange data in a specified order, i.e. either ascending or descending.
- Indexes or labels of a row or column can be changed in a DataFrame. This process is known as Altering the index. Two functions reset_index and set_index are used for that purpose.
- Missing values are a hindrance in data analysis and must be handled properly.
- There are primarily two main strategies for handling missing data. Either the row (or column) having missing value is removed completely from analysis or missing value is replaced by some

NOTES

appropriate value (which may be zero or one or average etc.)

- Process of changing the structure of the DataFrame is known as Reshaping. Pandas provide two basic functions for this, pivot() and pivot_table().
- pymysql and sqlalchemy are two mandatory libraries for facilitating import and export of data between Pandas and MySQL. Before import and export, a connection needs to be established from python script to MySQL database.
- Importing data from MySQL to Panda refers to the process of fetching data from a MySQL table or database to a pandas DataFrame.
- Exporting data from Pandas to MySQL refers to the process of storing data from a pandas DataFrame to a MySQL table or database.

Exercise

1. Write the statement to install the python connector to connect MySQL i.e. pymysql.

2. Explain the difference between pivot() and pivot_table() function?

3. What is sqlalchemy?

4. Can you sort a DataFrame with respect to multiple columns?

5. What are missing values? What are the strategies to handle them?

6. Define the following terms: Median, Standard Deviation and variance.

7. What do you understand by the term MODE? Name the function which is used to calculate it.

8. Write the purpose of Data aggregation.

9. Explain the concept of GROUP BY with help on an example.

10. Write the steps required to read data from a MySQL database to a DataFrame.

11. Explain the importance of reshaping of data with an example.

12. Why estimation is an important concept in data analysis?

13. Assuming the given table: Product. Write the python code for the following:

Item	Company	Rupees	USD
TV	LG	12000	700
TV	VIDEOCON	10000	650
TV	LG	15000	800
AC	SONY	14000	750

a) To create the data frame for the above table.

b) To add the new rows in the data frame.

c) To display the maximum price of LG TV.

d) To display the Sum of all products.

e) To display the median of the USD of Sony products.

f) To sort the data according to the Rupees and transfer the data to MySQL.

g) To transfer the new dataframe into the MySQL with new values.

14. Write the python statement for the following question on the basis of given dataset:

```
        Name  Degree  Score
0      Aparna     MBA   90.0
1      Pankaj     BCA    NaN
2         Ram  M.Tech   80.0
3      Ramesh     MBA   98.0
4      Naveen     NaN   97.0
5   Krrishnav     BCA   78.0
6      Bhawna     MBA   89.0
```

a) To create the above DataFrame.

b) To print the Degree and maximum marks in each stream.

c) To fill the NaN with 76.

d) To set the index to Name.

e) To display the name and degree wise average marks of each student.

f) To count the number of students in MBA.

g) To print the mode marks BCA.

SOLVED CASE STUDY BASED ON OPEN DATASETS

UCI dataset is a collection of open datasets, available to the public for experimentation and research purposes. 'auto-mpg' is one such open dataset.

It contains data related to fuel consumption by automobiles in a city. Consumption is measured in miles per gallon (mpg), hence the name of the dataset is auto-mpg. The data has 398 rows (also known as items or instances or objects) and nine columns (also known as attributes).

The attributes are: mpg, cylinders, displacement, horsepower, weight, acceleration, model year, origin, car name. Three attributes, cylinders, model year and origin have categorical values, car name is a string with a unique value for every row, while the remaining five attributes have numeric value.

The data has been downloaded from the UCI data repository available at http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/.

Following are the exercises to analyse the data.

1) Load auto-mpg.data into a DataFrame autodf.

2) Give description of the generated DataFrame autodf.

3) Display the first 10 rows of the DataFrame autodf.

4) Find the attributes which have missing values. Handle the missing values using following two ways:

   i. Replace the missing values by a value before that.

   ii. Remove the rows having missing values from the original dataset

5) Print the details of the car which gave the maximum mileage.

6) Find the average displacement of the car given the number of cylinders.

7) What is the average number of cylinders in a car?

8) Determine the no. of cars with weight greater than the average weight.