Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language.

Installing Pandas

To install Python Pandas, go to your command prompt and type "pip install pandas". Once the installation is completed, go to your IDE (For example: PyCharm) or Anaconda Jupyter Notebook and simply import it by typing: **import pandas as pd**.

How to import Pandas

In order to start using Pandas and all of the function available in Pandas, You will need to import it. This can be easily done with this import statement.

```
In [1]:    import pandas as pd
```

NOTE: We shorten pandas to pd in order to save time and also to keep code standardized so that anyone working with your code can easily understand and run it.

Data structures:

Pandas deals with the following three data structures −

1. Series
2. DataFrame
3. Panel

Series:

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.

A pandas Series can be created using the following constructor −

Syntax

```
In [ ]:   pandas.Series(data, index, dtype, copy)
```

Here, data can be many different things:
- a Python dict
- an ndarray
- a scalar value (like 5)

[Example](#) Create an Empty Series

```
In [2]:   import pandas as pd
          s = pd.Series()
          print(s)
```

```
Series([], dtype: float64)
```

Example Create a Series from ndarray

```
In [3]:   import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data)
          print(s)
```

```
0    a
1    b
2    c
3    d
dtype: object
```

Example Create a Series from ndarray with index mentioned

```
In [4]:   import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data,index=[100,101,102,103])
          print(s)
```

```
100    a
101    b
102    c
103    d
dtype: object
```

Example Create a Series from dict

NOTE: A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If index is passed, the values in data corresponding to the labels in the index will be pulled out.

```
In [5]:   import pandas as pd
          import numpy as np
          data = {'a' : 0., 'b' : 1., 'c' : 2.}
          s = pd.Series(data)
          print(s)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

```
In [6]:   import pandas as pd
          import numpy as np
          data = {'a' : 0., 'b' : 1., 'c' : 2.}
          s = pd.Series(data,index=['b','c','d','a'])
          print(s)
```

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

Example Create a Series from Scalar

```
In [7]:   import pandas as pd
```

```
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print(s)
```

```
0    5
1    5
2    5
3    5
dtype: int64
```

NOTE If data is a scalar value, an index must be provided. The value will be repeated to match the length of index

Accessing Data from Series with Position: To access the data you have to mentioned the position within [] bracket.

In [8]:
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print("Series are: \n",s)
#retrieve the first element
print(s[0])
print(s[3])
```

```
Series are:
 a    1
b    2
c    3
d    4
e    5
dtype: int64
1
4
```

Slicing Data from Series with position: For slicing you can use the index from which index to which index you want.

In [9]:
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print("Series are: \n",s)
#retrieve the first three element
print(s[:3])
```

```
Series are:
 a    1
b    2
c    3
d    4
e    5
dtype: int64
a    1
b    2
c    3
dtype: int64
```

In [10]:
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print("Series are: \n",s)
#retrieve the last three element
print(s[-3:])
```

```
Series are:
 a    1
b    2
c    3
d    4
e    5
```

```
dtype: int64
c    3
d    4
e    5
dtype: int64
```

Retrieve Data Using Label: To access the data you have to mentioned the label within [] bracket.

In [11]:
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print("Series are: \n",s)
#retrieve a single element
print(s['a'])
```

```
Series are:
 a    1
 b    2
 c    3
 d    4
 e    5
dtype: int64
1
```

In [12]:
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print("Series are: \n",s)
#retrieve multiple elements
print(s[['a','c','d']])
```

```
Series are:
 a    1
 b    2
 c    3
 d    4
 e    5
dtype: int64
a    1
c    3
d    4
dtype: int64
```

Basic functionality of series:

axes: Returns the list of the labels of the series.

In [13]:
```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print(s)
print("The axes are: ")
print(s.axes)
```

```
100    a
101    b
102    c
103    d
dtype: object
The axes are:
[Int64Index([100, 101, 102, 103], dtype='int64')]
```

empty: Returns the Boolean value saying whether the Object is empty or not. True indicates that the object is empty.

```
In [14]:  import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data,index=[100,101,102,103])
          print(s)
          print("The series is empty or not: ",s.empty)
```

```
100    a
101    b
102    c
103    d
dtype: object
The series is empty or not:  False
```

```
In [15]:  import numpy as np
          import pandas as pd
          s=pd.Series()
          print(s)
          print("The series is empty or not: ",s.empty)
```

```
Series([], dtype: float64)
The series is empty or not:  True
```

<ipython-input-15-500188ae0830>:3: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  s=pd.Series()

ndim: Returns the number of dimensions of the object. By definition, a Series is a 1D data structure, so it returns 1.

```
In [16]:  import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data,index=[100,101,102,103])
          print(s)
          print("The dimension of s: ",s.ndim)
```

```
100    a
101    b
102    c
103    d
dtype: object
The dimension of s:   1
```

size: Returns the size(length) of the series.

```
In [17]:  import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data,index=[100,101,102,103])
          print(s)
          print("The dimension of s: ",s.size)
```

```
100    a
101    b
102    c
103    d
dtype: object
The dimension of s:   4
```

values: Returns the actual data in the series as an array.

```
In [18]:  import pandas as pd
          import numpy as np
          data = np.array(['a','b','c','d'])
          s = pd.Series(data,index=[100,101,102,103])
```

```
print(s)
print("The values of s: ",s.values)
```

```
100    a
101    b
102    c
103    d
dtype: object
The values of s:   ['a' 'b' 'c' 'd']
```

head(): head() returns the first n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

tail(): tail() returns the last n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

In [19]:
```python
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d','e','f','g','h'])
s = pd.Series(data,index=[100,101,102,103,104,105,106,10])
print(s)
print("**************************")
print("The head of s: \n",s.head())
print("**************************")
print("The tail of s: \n",s.tail())
```

```
100    a
101    b
102    c
103    d
104    e
105    f
106    g
10     h
dtype: object
**************************
The head of s:
 100    a
101    b
102    c
103    d
104    e
dtype: object
**************************
The tail of s:
 103    d
104    e
105    f
106    g
10     h
dtype: object
```

Explore via coding

Create a series of age of 15 people acess the fifth element and print the dimension.

In [20]:
```python
import pandas as pd

s=pd.Series([18,19,20,22,24,26,70,69,45,35,76,65,15,35,43],name="age")
print("Series are: \n",s)
print("Fifth element of series are: ",s[4])
print("Dimension is: ",s.ndim)
```

```
Series are:
```

```
 0      18
 1      19
 2      20
 3      22
 4      24
 5      26
 6      70
 7      69
 8      45
 9      35
10      76
11      65
12      15
13      35
14      43
Name: age, dtype: int64
Fifth element of series are:  24
Dimension is:  1
```

DataFrames:

A DataFrame is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, categorical data and more) in columns. It is similar to a spreadsheet, a SQL table.
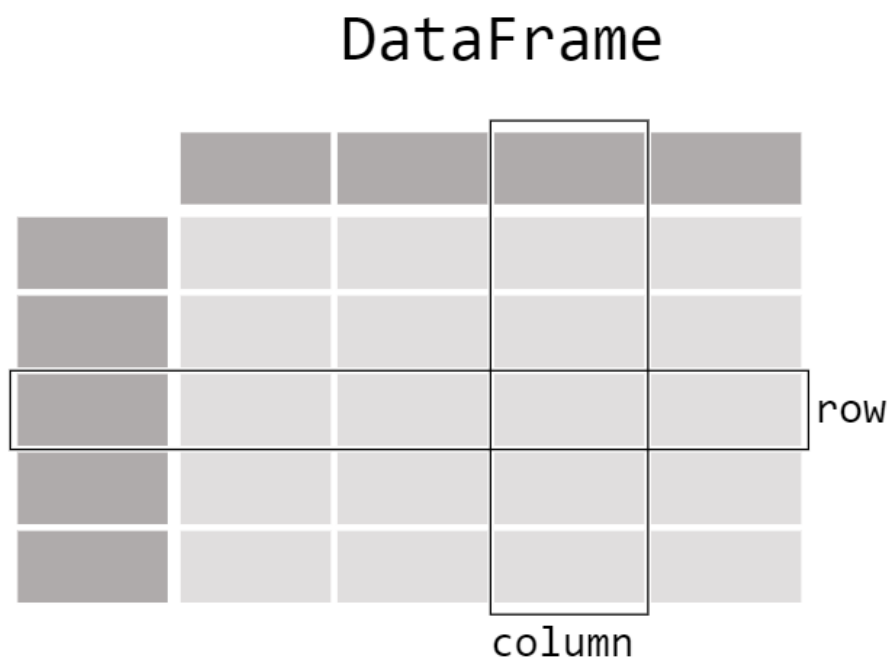
DataFrame

A pandas DataFrame can be created using the following constructor −

Syntax

In [ ]:
```
pandas.DataFrame( data, index, columns, dtype, copy)
```

DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Example Create an Empty DataFrame

In [21]:
```python
import pandas as pd
df = pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

Example Create a DataFrame from Lists

In [22]:
```python
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
```

```
   0
0  1
1  2
2  3
3  4
4  5
```

In [23]:
```python
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
     Name  Age
0    Alex   10
1     Bob   12
2  Clarke   13
```

In [24]:
```python
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print(df)
```

```
     Name   Age
0    Alex  10.0
1     Bob  12.0
2  Clarke  13.0
```

Example Create a DataFrame from Dict of ndarrays / Lists

In [25]:
```python
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age
0    Tom   28
1   Jack   34
2  Steve   29
3  Ricky   42
```

In [26]:
```python
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print(df)
```

```
        Name  Age
rank1    Tom   28
rank2   Jack   34
rank3  Steve   29
rank4  Ricky   42
```

Create a DataFrame from List of Dicts

```
In [27]:  import pandas as pd
          data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
          df = pd.DataFrame(data)
          print(df)
```

```
     a   b     c
0    1   2   NaN
1    5  10  20.0
```

```
In [28]:  import pandas as pd
          data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
          df = pd.DataFrame(data, index=['first', 'second'])
          print(df)
```

```
          a   b     c
first     1   2   NaN
second    5  10  20.0
```

How do i select specific column from a DataFrame:

column selection: We can select the column in data frame by using label. In the data frame you have to pass column name in square bracket.


```
In [29]:  import pandas as pd

          d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
              'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

          df = pd.DataFrame(d)
          print("Data frame is: \n",df)
          print("***********")
          print(df['one'])
```

```
Data frame is:
     one   two
a    1.0     1
b    2.0     2
c    3.0     3
d    NaN     4
***********
a      1.0
b      2.0
c      3.0
d      NaN
Name: one, dtype: float64
```

NOTE: Each column in a DataFrame is a Series. As a single column is selected, the returned object is a pandas Series. We can verify this by checking the type of the output:

```
In [30]:  print(type(df['one']))
```

```
<class 'pandas.core.series.Series'>
```

Multiple column selection: To select multiple columns, use a list of column names within the selection brackets [].

```
In [31]:  import pandas as pd

          d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

          df = pd.DataFrame(d)
          print("Data frame is: \n",df)
          print("***********")
          print(df[['one','two']])
```

```
Data frame is:
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
***********
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
```

How do I filter specific rows from a DataFrame?

To select rows based on a conditional expression, use a condition inside the selection brackets [].
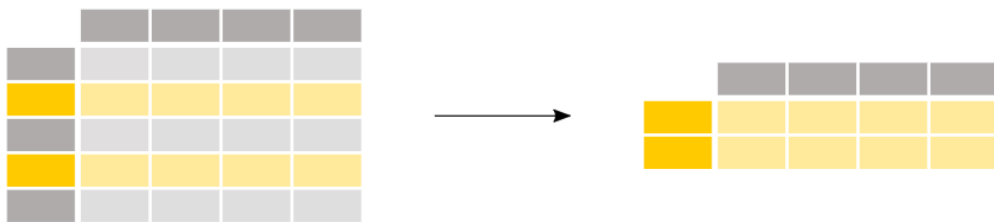

```
In [32]:  import pandas as pd

          d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

          df = pd.DataFrame(d)
          print("Data frame is: \n",df)
          print("***********")
          print(df[df['one']>2])
```

```
Data frame is:
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
***********
```

```
     one   two
c    3.0    3
```

The condition inside the selection brackets df[df['one']] > 2 checks for which rows the one column has a value larger than 2

How to create new columns derived from existing columns?

To create a new column, use the [] brackets with the new column name at the left side of the assignment.
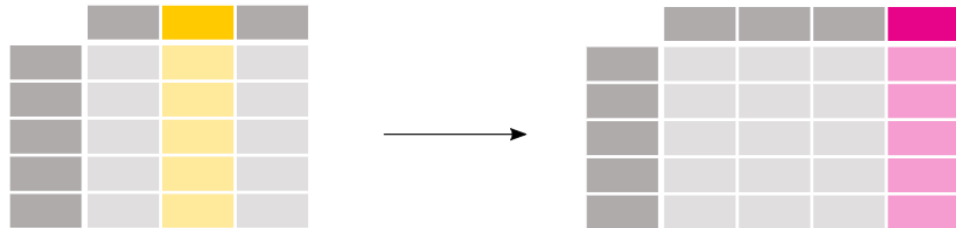

```
In [33]:  import pandas as pd

          d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

          df = pd.DataFrame(d)

          # Adding a new column to an existing DataFrame object with column label

          print ("\nAdding a new column by passing as Series:")
          df['three']=pd.Series([10,20,30],index=['a','b','c'])
          print(df)

          print ("\nAdding a new column using the existing columns in DataFrame:")
          df['four']=df['one']+df['three']

          print(df)
```

```
Adding a new column by passing as Series:
     one   two   three
a    1.0    1    10.0
b    2.0    2    20.0
c    3.0    3    30.0
d    NaN    4     NaN

Adding a new column using the existing columns in DataFrame:
     one   two   three   four
a    1.0    1    10.0    11.0
b    2.0    2    20.0    22.0
c    3.0    3    30.0    33.0
d    NaN    4     NaN     NaN
```

NOTE: The calculation of the values is done element_wise. This means all values in the column['one'] are added with column['two']. You do not need to use a loop to iterate each of the rows!

How to delete any column from DataFrame

If you want to delete any particular column from Dataframe then you can use del with column name.

```python
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
    'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print("Our dataframe is:")
print(df)

# using del function
print ("\nDeleting the first column using DEL function:")
del df['one']
print(df)

# using pop function
print ("\nDeleting another column using POP function:")
df.pop('two')
print(df)
```

```
Our dataframe is:
    one  two   three
a   1.0    1    10.0
b   2.0    2    20.0
c   3.0    3    30.0
d   NaN    4     NaN

Deleting the first column using DEL function:
    two   three
a     1    10.0
b     2    20.0
c     3    30.0
d     4     NaN

Deleting another column using POP function:
    three
a    10.0
b    20.0
c    30.0
d     NaN
```

Indexing and selecting data

.loc: is primarily label based, but may also be used with a boolean array. .loc will raise KeyError when the items are not found. Allowed inputs are:

- A single label, e.g. 5 or 'a' (Note that 5 is interpreted as a label of the index. This use is not an integer position along the index.).
- A list or array of labels ['a', 'b', 'c'].
- A slice object with labels 'a':'f' (Note that contrary to usual Python slices, both the start and the stop are included, when present in the index! See Slicing with labels and Endpoints are inclusive.)
- A boolean array (any NA values will be treated as False).
- A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above).

```python
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
print("***********")
print(df.loc['b'])
```

```
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
***********
one     2.0
two     2.0
Name: b, dtype: float64
```

.iloc: is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array. .iloc will raise IndexError if a requested indexer is out-of-bounds, except slice indexers which allow out-of-bounds indexing. (this conforms with Python/NumPy slice semantics). Allowed inputs are:

- An integer e.g. 5.
- A list or array of integers [4, 3, 0].
- A slice object with ints 1:7.
- A boolean array (any NA values will be treated as False).
- A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above).

```python
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
print("***********")
print("Zeroth location: \n",df.iloc[0])
print("***********")
print("First location: \n",df.iloc[1])
```

```
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
***********
Zeroth location:
 one     1.0
two     1.0
Name: a, dtype: float64
***********
First location:
 one     2.0
two     2.0
Name: b, dtype: float64
```

Slice Rows

Pandas dataframe slicing refers to accessing portion or a subset of Pandas Dataframe while the original dataframe remain unaffected. You can use indexes of dataframe elements to create dataframe slice as per the following syntax:

slice=[StartIndex : StopIndex : Steps]

- The StartIndex represents the index from where the dataframe slicing is supposed to begin. Its default value is 0, i.r., the dataframe begins from index 0 if no StartIndex is specified.
- The StopIndex represents the last index upto which the dataframe slicing will go on. Its default value is (length(dataframe)-1) or the index of the last element in the dataframe element in the dataframe.
- Steps represent the number of steps. It is an optional parameter. Steps, if defined, specifies the number of elements to jump over while counting from StartIndex to StopIndex. By default it is 1.
- The dataframe slices created, include elements failing between the indexes StartIndex and StopIndex, including StartIndex and not including StopIndex.

```
In [37]:   import pandas as pd

           d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

           df = pd.DataFrame(d)
           print(df[1:])            #by default its stop index is last element in d
```

```
     one  two
b    2.0    2
c    3.0    3
d    NaN    4
```

```
In [38]:   import pandas as pd

           d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

           df = pd.DataFrame(d)
           print(df[:4])            #by default its start index is from first element
```

```
     one  two
a    1.0    1
b    2.0    2
c    3.0    3
d    NaN    4
```

```
In [39]:   import pandas as pd

           d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
               'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

           df = pd.DataFrame(d)
           print(df[1:4:2])
```

```
     one  two
b    2.0    2
d    NaN    4
```

```
In [40]:   import pandas as pd
```

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df[::])          #by default its start from first element go till las
```

```
    one  two
a   1.0    1
b   2.0    2
c   3.0    3
d   NaN    4
```

Appending two DataFrame

append() function is used to append rows of other dataframe to the end of the given dataframe, returning a new dataframe object. Columns not in the original dataframes are added as new columns and the new cells are populated with NaN value.
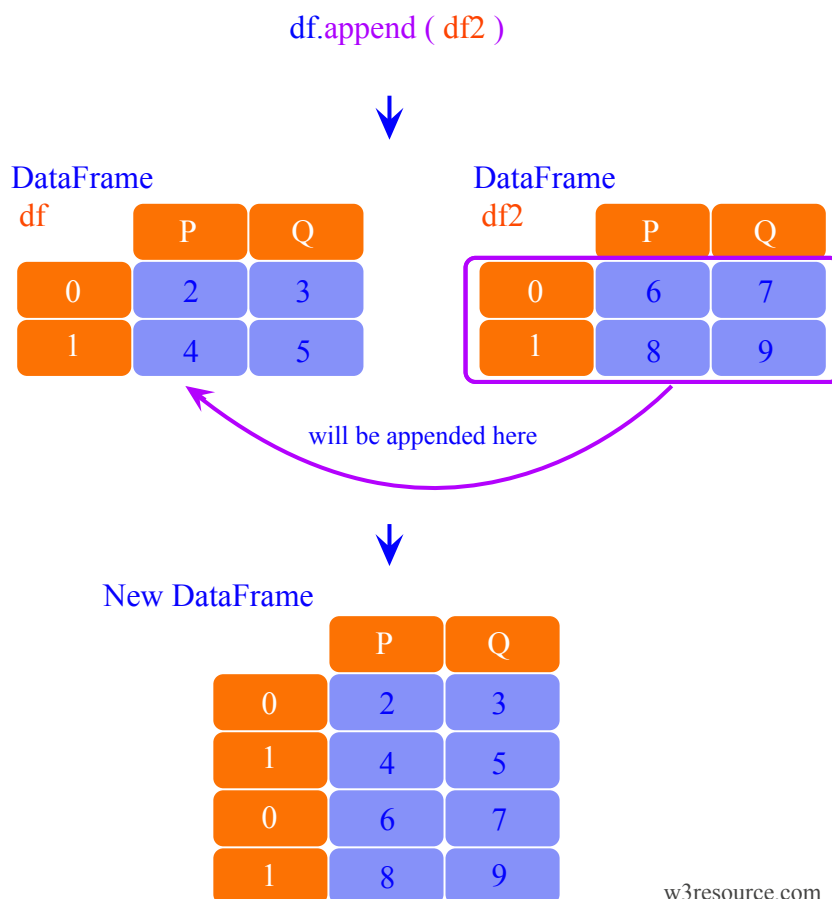

```
In [41]:  import pandas as pd

          df1 = pd.DataFrame([[2, 3], [4, 5]], columns = ['P','Q'])
          df2 = pd.DataFrame([[6, 7], [8, 9]], columns = ['P','Q'])
          print("Dataframe1: ")
          print(df1)
          print("**************")
          print("DataFrame2: ")
          print(df2)
          print("**************")
          df = df1.append(df2)
          print(df)
```

```
Dataframe1:
   P  Q
0  2  3
1  4  5
**************
DataFrame2:
   P  Q
0  6  7
1  8  9
**************
   P  Q
0  2  3
1  4  5
0  6  7
1  8  9
```

Deletion of Rows

In [42]:
```python
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
print("Dataframe: ")
print(df)
print("**************")
print("DataFrame2: ")
print(df2)
print("**************")
df = df.append(df2)
print("DataFrame is: ")
print(df)
print("**************")
df = df.drop(0)
print(df)
```

```
Dataframe:
   a  b
0  1  2
1  3  4
**************
DataFrame2:
   a  b
0  5  6
1  7  8
**************
DataFrame is:
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
**************
   a  b
1  3  4
1  7  8
```

Basic functionality of DataFrame:

T (Transpose): Returns the transpose of the DataFrame. The rows and columns will interchange.

In [43]:
```python
import pandas as pd
import numpy as np
```

```
# Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print("The orifinal dataframe is: ")
print(df)
print("************************************")
print("The transpose of the data series is:")
print(df.T)
```

```
The orifinal dataframe is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
************************************
The transpose of the data series is:
            0      1      2     3      4      5     6
Name      Tom  James  Ricky   Vin  Steve  Smith  Jack
Age        25     26     25    23     30     29    23
Rating   4.23   3.24   3.98  2.56    3.2    4.6   3.8
```

axes: Returns the list of row axis labels and column axis labels.

In [44]:
```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Row axis labels and column axis labels are:")
print(df.axes)
```

```
Row axis labels and column axis labels are:
[RangeIndex(start=0, stop=7, step=1), Index(['Name', 'Age', 'Rating'], dt
ype='object')]
```

dtypes: Returns the data type of each column.

In [45]:
```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("The data type of each column are:")
print(df.dtypes)
```

```
The data type of each column are:
Name        object
```

```
Age          int64
Rating     float64
dtype: object
```

empty: Returns the Boolean value saying whether the Object is empty or not; True indicates that the object is empty.

In [46]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Is the object empty?")
print(df.empty)
```

```
Is the object empty?
False
```

ndim: Returns the number of dimensions of the object. By definition, DataFrame is a 2D object

In [47]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Our object is:")
print(df)
print("The dimension of the object is:")
print(df.ndim)
```

```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The dimension of the object is:
2
```

shape: Returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and b represents the number of columns.

In [48]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
```

```
df = pd.DataFrame(d)
print("Our object is:")
print(df)
print("The shape of the object is:")
print(df.shape)
```

```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The shape of the object is:
(7, 3)
```

size: Returns the number of elements in the DataFrame.

In [49]:
```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Our object is:")
print(df)
print("The total number of elements in our object is:")
print(df.size)
```

```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The total number of elements in our object is:
21
```

values: Returns the actual data in the DataFrame as an NDarray.

In [50]:
```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
    'Age':pd.Series([25,26,25,23,30,29,23]),
    'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Our object is:")
print(df)
print("The actual data in our data frame is:")
print(df.values)
```

```
Our object is:
    Name  Age  Rating
```

```
0     Tom    25    4.23
1   James    26    3.24
2   Ricky    25    3.98
3     Vin    23    2.56
4   Steve    30    3.20
5   Smith    29    4.60
6    Jack    23    3.80
The actual data in our data frame is:
[['Tom' 25 4.23]
 ['James' 26 3.24]
 ['Ricky' 25 3.98]
 ['Vin' 23 2.56]
 ['Steve' 30 3.2]
 ['Smith' 29 4.6]
 ['Jack' 23 3.8]]
```

head(): returns the first n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

tail(): returns the last n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

In [51]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print("Our data frame is:")
print(df)
print("*****************************************")
print("The first two rows of the data frame is:")
print(df.head(2))
print("*****************************************")
print("The last two rows of the data frame is:")
print(df.tail(2))
```

```
Our data frame is:
     Name  Age  Rating
0     Tom   25    4.23
1   James   26    3.24
2   Ricky   25    3.98
3     Vin   23    2.56
4   Steve   30    3.20
5   Smith   29    4.60
6    Jack   23    3.80
*****************************************
The first two rows of the data frame is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
*****************************************
The last two rows of the data frame is:
    Name  Age  Rating
5  Smith   29     4.6
6   Jack   23     3.8
```

Descriptive statistics:

count(): Count number of non-null observations

sum(): Sum of values

mean(): Mean of Values

median(): Median of Values

mode(): Mode of values

std(): Standard Deviation of the Values

min(): Minimum Value

max(): Maximum Value

abs(): Absolute Value

prod(): Product of Values

cumsum(): Cumulative Sum

cumprod(): Cumulative Product **describe()** function computes a summary of statistics pertaining to the DataFrame columns.

In [52]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4
}

#Create a DataFrame
df = pd.DataFrame(d)
print(df)
print("****************************************************************
print("count is: \n",df.count())
print("****************************************************************
print("sum is: \n",df.sum())
print("****************************************************************
print("mean is: \n",df.mean())
print("****************************************************************
print("mode is: \n",df.mode())
print("****************************************************************
print("median is: \n",df.median())
print("****************************************************************
print("Standard deviation is: \n",df.std())
print("****************************************************************
print("minimum is: \n",df.min())
print("****************************************************************
print("maximum is: \n",df.max())
print("****************************************************************
print("Describe is: \n",df.describe())
```

```
      Name  Age  Rating
0      Tom   25    4.23
1    James   26    3.24
2    Ricky   25    3.98
3      Vin   23    2.56
4    Steve   30    3.20
5    Smith   29    4.60
6     Jack   23    3.80
7      Lee   34    3.78
8    David   40    2.98
9   Gasper   30    4.80
10  Betina   51    4.10
11  Andres   46    3.65
**************************************************************************
*************************************
```

```
count is:
 Name     12
Age       12
Rating    12
dtype: int64
**********************************************************************
****************************************
sum is:
 Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Age                                                     382
Rating                                                44.92
dtype: object
**********************************************************************
****************************************
mean is:
 Age       31.833333
Rating     3.743333
dtype: float64
**********************************************************************
****************************************
mode is:
        Name   Age  Rating
0    Andres  23.0    2.56
1    Betina  25.0    2.98
2     David  30.0    3.20
3    Gasper   NaN    3.24
4      Jack   NaN    3.65
5     James   NaN    3.78
6       Lee   NaN    3.80
7     Ricky   NaN    3.98
8     Smith   NaN    4.10
9     Steve   NaN    4.23
10      Tom   NaN    4.60
11      Vin   NaN    4.80
**********************************************************************
****************************************
median is:
 Age       29.50
Rating     3.79
dtype: float64
**********************************************************************
****************************************
Standard deviation is:
 Age       9.232682
Rating    0.661628
dtype: float64
**********************************************************************
****************************************
minimum is:
 Name      Andres
Age           23
Rating      2.56
dtype: object
**********************************************************************
****************************************
maximum is:
 Name      Vin
Age        51
Rating    4.8
dtype: object
**********************************************************************
****************************************
Describe is:
             Age     Rating
count  12.000000  12.000000
mean   31.833333   3.743333
std     9.232682   0.661628
min    23.000000   2.560000
```

```
25%       25.000000     3.230000
50%       29.500000     3.790000
75%       35.500000     4.132500
max       51.000000     4.800000
```

NOTE: describe() function function excludes the character columns and given summary about numeric columns. 'include' is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

object − Summarizes String columns

number − Summarizes Numeric columns

all − Summarizes all columns together (Should not pass it as a list value)

In [53]:
```python
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack
     'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
     'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
}

#Create a DataFrame
df = pd.DataFrame(d)
print(df.describe())
print("*********************************************************************
print(df.describe(include=['object']))
print("*********************************************************************
print(df.describe(include='all'))
```

```
              Age      Rating
count   12.000000   12.000000
mean    31.833333    3.743333
std      9.232682    0.661628
min     23.000000    2.560000
25%     25.000000    3.230000
50%     29.500000    3.790000
75%     35.500000    4.132500
max     51.000000    4.800000
**********************************************************************************
****************************************
          Name
count       12
unique      12
top      Ricky
freq         1
**********************************************************************************
****************************************
          Name        Age      Rating
count       12   12.000000   12.000000
unique      12         NaN         NaN
top      Ricky         NaN         NaN
freq         1         NaN         NaN
mean       NaN   31.833333    3.743333
std        NaN    9.232682    0.661628
min        NaN   23.000000    2.560000
25%        NaN   25.000000    3.230000
50%        NaN   29.500000    3.790000
75%        NaN   35.500000    4.132500
max        NaN   51.000000    4.800000
```

Function application:

Table wise function application/pipe(): Dataframe.pipe() method with appropriate number of argument operation is performed on the whole dataframe.

In [54]:
```python
import numpy as np
import pandas as pd

def adder(ele1,num):
    return ele1+num

data=[[1,2],[3,4],[5,6],[7,8],[9,10]]
df=pd.DataFrame(data,columns=['col1','col2'])
print("Original dataframe is : \n",df)
print("\nAfter apply pipe method: \n",df.pipe(adder,2))
```

```
Original dataframe is :
    col1  col2
0    1    2
1    3    4
2    5    6
3    7    8
4    9   10

After apply pipe method:
    col1  col2
0    3    4
1    5    6
2    7    8
3    9   10
4   11   12
```

Row or column wise function application/apply() : Dataframe.apply() allow the users to pass a function and apply it on every single value of the Pandas series.

In [55]:
```python
import numpy as np
import pandas as pd

data=[[1,2],[3,4],[5,6],[7,8],[9,10]]
df=pd.DataFrame(data,columns=['col1','col2'])
print("Original dataframe is : \n",df)
print("\nAfter apply  method: \n",df.apply(np.mean))
print("\nAfter apply  method: \n",df.apply(np.mean,axis=1))
```

```
Original dataframe is :
    col1  col2
0    1    2
1    3    4
2    5    6
3    7    8
4    9   10

After apply  method:
 col1    5.0
col2    6.0
dtype: float64

After apply  method:
 0    1.5
1    3.5
2    5.5
3    7.5
4    9.5
dtype: float64
```

Element wise function application: Dataframe.applymap() method applies a function that accepts and returns a scalar to every element of a DataFrame.

```
In [56]:  import numpy as np
          import pandas as pd

          data=[[1,2],[3,4],[5,6],[7,8],[9,10]]
          df=pd.DataFrame(data,columns=['col1','col2'])
          print("Original dataframe is : \n",df)
          print("***********************")
          print("After applymap: \n",df.applymap(lambda x:x*100))
          print("***********************")
          print(df.apply(np.mean))
```

```
Original dataframe is :
   col1  col2
0    1     2
1    3     4
2    5     6
3    7     8
4    9    10
***********************
After applymap:
   col1  col2
0   100   200
1   300   400
2   500   600
3   700   800
4   900  1000
***********************
col1    5.0
col2    6.0
dtype: float64
```

Sorting:

There are two kinds of sorting available in Pandas. They are −

By label
By Actual Value

```
In [57]:  import pandas as pd
          import numpy as np

          unsorted_df=pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,
          print(unsorted_df)
```

```
       col1      col2
1 -0.660139  1.441730
4 -0.071603  0.269932
6 -0.336583  0.661928
2 -0.599720 -0.988778
3 -0.642074  1.693148
5  0.745967  0.146820
9  0.567836 -0.175232
8  2.101790 -1.293133
0  0.379764 -1.057593
7  0.009930  1.400048
```

By label:

```
In [58]:  import pandas as pd
```

```python
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_index()
print("Sorted index wise")
print(sorted_df)
print("*************************************************************
sorted_df = unsorted_df.sort_index(ascending=False)
print("Sorted ascending wise")
print(sorted_df)
```

```
Sorted index wise
        col1      col2
0 -0.048471 -0.698869
1  0.625944  0.237917
2 -0.790532  0.529453
3  0.024598  0.263129
4  0.417039  2.403819
5 -2.513857 -1.429845
6  0.569424 -1.110645
7  0.475258 -0.431311
8 -0.644009  0.827621
9  1.565836  0.983716
*************************************************************************
*****
Sorted ascending wise
        col1      col2
9  1.565836  0.983716
8 -0.644009  0.827621
7  0.475258 -0.431311
6  0.569424 -1.110645
5 -2.513857 -1.429845
4  0.417039  2.403819
3  0.024598  0.263129
2 -0.790532  0.529453
1  0.625944  0.237917
0 -0.048471 -0.698869
```

Sort the columns:

In [59]:
```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by=['col1'])
print("Sorted by column1")
print(sorted_df)
print("****************************")
sorted_df=unsorted_df.sort_values(by=['col1'], ascending=False)
print("Sorted ascending wise")
print(sorted_df)
```

```
Sorted by column1
        col2      col1
3 -0.097653 -1.920968
1 -0.759541 -1.707629
7 -1.469773 -1.487811
0  0.879300 -0.587653
9 -0.730604 -0.349882
8  1.683559  0.068307
2  0.088240  0.474390
6  0.321830  0.559457
5  1.209454  1.314141
4  1.011382  2.085866
****************************
```

```
Sorted ascending wise
        col2       col1
4   1.011382   2.085866
5   1.209454   1.314141
6   0.321830   0.559457
2   0.088240   0.474390
8   1.683559   0.068307
9  -0.730604  -0.349882
0   0.879300  -0.587653
7  -1.469773  -1.487811
1  -0.759541  -1.707629
3  -0.097653  -1.920968
```

```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by=['col2'])
print("Sorted by column2")
print(sorted_df)
print("***************************")
sorted_df=unsorted_df.sort_values(by=['col2'], ascending=False)
print("Sorted ascending wise")
print(sorted_df)
```

```
Sorted by column2
        col2       col1
0  -0.425260  -3.586128
6  -0.042807   0.638521
9   0.035894  -0.236678
4   0.159201   0.591428
8   0.251322  -0.403187
3   0.300296  -0.230369
2   0.537355   1.990105
7   0.587931   0.276665
5   1.096417  -0.990225
1   1.297242  -0.630314
***************************
Sorted ascending wise
        col2       col1
1   1.297242  -0.630314
5   1.096417  -0.990225
7   0.587931   0.276665
2   0.537355   1.990105
3   0.300296  -0.230369
8   0.251322  -0.403187
4   0.159201   0.591428
9   0.035894  -0.236678
6  -0.042807   0.638521
0  -0.425260  -3.586128
```

In [61]:
```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by=['col1','col2'])
print("Sorted by column2")
print(sorted_df)
```

```
Sorted by column2
        col2       col1
5  -0.315270  -0.895541
6   0.171963  -0.782162
1  -1.041005  -0.385329
4   1.139732  -0.247565
```

```
8   0.375400 -0.184219
9   0.180701 -0.161054
7   1.647876 -0.133221
2   1.144221  0.332937
0  -1.437100  0.359159
3   0.973520  0.953666
```

sorting algorithm

sort_values() provides a provision to choose the algorithm from mergesort, heapsort and
quicksort. Mergesort is the only stable algorithm

In [62]:
```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by='col1',kind='mergesort')

print(sorted_df)
```
```
        col1      col2
9  -1.060151 -0.442039
7  -0.296782  1.002019
5  -0.192444 -0.952711
8  -0.181269 -0.397951
1   0.013056 -1.617972
3   0.206970  1.799657
2   0.700612  0.923792
6   1.125891  0.352842
4   2.050384  1.861975
0   2.301670 -0.121701
```

In [63]:
```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by='col1',kind='heapsort')

print(sorted_df)
```
```
        col1      col2
4  -1.335856 -0.788297
1  -1.261907  1.613686
8  -1.135699 -0.346609
6  -0.942440  0.117917
5  -0.850910 -0.937891
2  -0.783147  0.706686
9  -0.750098  1.560976
3  -0.222333 -0.871196
0  -0.182771  2.533929
7   0.499805 -0.290617
```

In [64]:
```python
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,

sorted_df=unsorted_df.sort_values(by='col1',kind='quicksort')

print(sorted_df)
```
```
        col1      col2
4  -2.044142 -1.594567
3  -1.753194 -0.596760
```

```
0 -1.105143  1.601309
6 -0.525228  2.713795
2 -0.502357 -0.234311
7 -0.399640  1.158663
9  0.576452  0.337318
1  0.593772  0.258689
8  1.159961  0.705333
5  1.579399 -0.090990
```

NOTE: all string operation u can perform on string column

Find missing values:

Check for missing value: For checking missing values we have two function in Pandas. Let's have a look.

isnull(): isnull() method stores True for ever NaN value and False for a Not null value and returned it.

notnull(): notnull() method stores True for ever NON-NULL value and False for a null value and returned it.

In [65]:
```python
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print(df)
print("********************")
print(df['one'].isnull())
print("********************")
print(df['one'].notnull())
```

```
        one       two      three
a -1.023612  0.522151 -1.969415
b       NaN       NaN        NaN
c -1.863194 -1.077297 -0.747627
d       NaN       NaN        NaN
e  0.572640  1.153686 -0.419022
f -1.648816  1.961191  0.952571
g       NaN       NaN        NaN
h  1.287714 -1.507602  0.289768
********************
a    False
b     True
c    False
d     True
e    False
f    False
g     True
h    False
Name: one, dtype: bool
********************
a     True
b    False
c     True
d    False
e     True
f     True
g    False
h     True
Name: one, dtype: bool
```

Filling missing data

fillna(): DataFrame.fillna() method fills(replaces) NA or NaN values in the DataFrame with the specified values.

In [66]:
```python
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print("Original dataframe is: \n",df)
print("************************************************")
print("After filling missing data with 0: \n",df.fillna(0))
```

```
Original dataframe is:
        one       two     three
a  0.332844  0.162040 -1.015734
b       NaN       NaN       NaN
c  1.328551 -0.061478  1.807803
d       NaN       NaN       NaN
e  0.330107 -1.252296  0.053072
f  0.348133  0.630093 -3.212920
g       NaN       NaN       NaN
h -0.667607  1.193277 -0.621246
************************************************
After filling missing data with 0:
        one       two     three
a  0.332844  0.162040 -1.015734
b  0.000000  0.000000  0.000000
c  1.328551 -0.061478  1.807803
d  0.000000  0.000000  0.000000
e  0.330107 -1.252296  0.053072
f  0.348133  0.630093 -3.212920
g  0.000000  0.000000  0.000000
h -0.667607  1.193277 -0.621246
```

Fill NA forward and backward:

In [67]:
```python
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print("Original dataframe is: \n",df)
print("************************************************")
print(df.fillna(method='pad'))
print("************************************************")
print(df.fillna(method='bfill'))
```

```
Original dataframe is:
        one       two     three
a -2.290439 -0.227909  0.945807
b       NaN       NaN       NaN
c -0.103820 -0.478348 -0.000502
d       NaN       NaN       NaN
e  0.238314  0.352345  0.072301
f  0.100810 -1.036068 -0.000516
g       NaN       NaN       NaN
h -0.852017 -0.824285  0.346426
************************************************
        one       two     three
a -2.290439 -0.227909  0.945807
b -2.290439 -0.227909  0.945807
```

```
c -0.103820 -0.478348 -0.000502
d -0.103820 -0.478348 -0.000502
e  0.238314  0.352345  0.072301
f  0.100810 -1.036068 -0.000516
g  0.100810 -1.036068 -0.000516
h -0.852017 -0.824285  0.346426
**************************************************
        one       two     three
a -2.290439 -0.227909  0.945807
b -0.103820 -0.478348 -0.000502
c -0.103820 -0.478348 -0.000502
d  0.238314  0.352345  0.072301
e  0.238314  0.352345  0.072301
f  0.100810 -1.036068 -0.000516
g -0.852017 -0.824285  0.346426
h -0.852017 -0.824285  0.346426
```

Drop missing values:

dropna(): dropna() method removes the missing values and returns the DataFrame with NA entries dropped from it.

In [68]:
```python
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print(df.dropna())
```

```
        one       two     three
a -0.707849 -0.410537 -1.222403
c  2.110150 -0.226322 -0.104206
e  0.121394 -0.509421  0.985301
f  0.534284  0.943747 -0.917127
h  0.808615  0.617646  0.724728
```

Replace missing values:

replace(): The replace() function is used to replace values given in to_replace with value.

In [69]:
```python
import pandas as pd
import numpy as np

df = pd.DataFrame({'one':[10,20,30,40,50,2000], 'two':[1000,0,30,40,50,6
print(df)
print("*************")
print(df.replace({1000:10,2000:60}))
```

```
    one   two
0    10  1000
1    20     0
2    30    30
3    40    40
4    50    50
5  2000    60
*************
    one  two
0    10   10
1    20    0
2    30   30
3    40   40
```

```
4    50   50
5    60   60
```

Group by:

```python
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
    'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
    'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
    'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017]
    'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)
print(df)
print("**********************")
df.groupby('Team')
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
2   Devils     2  2014     863
3   Devils     3  2015     673
4    Kings     3  2014     741
5    kings     4  2015     812
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
9   Royals     4  2014     701
10  Royals     1  2015     804
11  Riders     2  2017     690
**********************
```

Out[70]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000022ACAF0ABE
0>

Iterating through Groups: With the groupby object in hand, we can iterate through the object similar to itertools.obj.

```python
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
    'Kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
    'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
    'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017]
    'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)
print(df)
print("**********************")
grouped=df.groupby('Team')

for name,group in grouped:
    print("Name: ",name)
    print("Group: \n")
    print(group)
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
2   Devils     2  2014     863
3   Devils     3  2015     673
4    Kings     3  2014     741
5    Kings     4  2015     812
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
```

```
9    Royals    4    2014        701
10   Royals    1    2015        804
11   Riders    2    2017        690
*********************
Name:  Devils
Group:

      Team   Rank   Year   Points
2   Devils     2    2014      863
3   Devils     3    2015      673
Name:  Kings
Group:

      Team   Rank   Year   Points
4    Kings     3    2014      741
5    Kings     4    2015      812
6    Kings     1    2016      756
7    Kings     1    2017      788
Name:  Riders
Group:

       Team   Rank   Year   Points
0    Riders     1    2014      876
1    Riders     2    2015      789
8    Riders     2    2016      694
11   Riders     2    2017      690
Name:  Royals
Group:

      Team   Rank   Year   Points
9    Royals     4    2014      701
10   Royals     1    2015      804
```

Select a Group: Using the get_group() method, we can select a single group.

```python
In [72]:   import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
    'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
    'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
    'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017]
    'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
print(grouped.get_group(2014))
```

```
      Team   Rank   Year   Points
0   Riders     1    2014      876
2   Devils     2    2014      863
4    Kings     3    2014      741
9   Royals     4    2014      701
```

Merging/Joining:

Pandas has full-featured, high performance in-memory join operations idiomatically very
similar to relational databases like SQL.

```python
In [73]:   import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
        'id':[1,2,3,4,5],
```

```
        'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(pd.merge(left,right,on='id'))
```

```
   id  Name_x subject_id_x Name_y subject_id_y
0   1    Alex         sub1  Billy         sub2
1   2     Amy         sub2  Brian         sub4
2   3   Allen         sub4   Bran         sub3
3   4   Alice         sub6  Bryce         sub6
4   5  Ayoung         sub5  Betty         sub5
```

In [74]:
```python
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
        'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(pd.merge(right,left,on='id'))
```

```
   id Name_x subject_id_x  Name_y subject_id_y
0   1  Billy         sub2    Alex         sub1
1   2  Brian         sub4     Amy         sub2
2   3   Bran         sub3   Allen         sub4
3   4  Bryce         sub6   Alice         sub6
4   5  Betty         sub5  Ayoung         sub5
```

In [75]:
```python
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
        'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(pd.merge(left,right,on=['id','subject_id']))
```

```
   id  Name_x subject_id Name_y
0   4   Alice       sub6  Bryce
1   5  Ayoung       sub5  Betty
```

Merge using how arguments:

Merge Method SQL Equivalent Description left LEFT OUTER JOIN Use keys from left object right RIGHT OUTER JOIN Use keys from right object outer FULL OUTER JOIN Use union of keys inner INNER JOIN Use intersection of keys

In [76]:
```python
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(pd.merge(left, right, on='subject_id', how='left'))
print("*************************************************************")
print(pd.merge(left, right, on='subject_id', how='right'))
print("*************************************************************")
print(pd.merge(left, right, on='subject_id', how='inner'))
print("*************************************************************")
```

```
print(pd.merge(left, right, on='subject_id', how='outer'))
print("*****************************************************************")
```

```
   id_x  Name_x subject_id  id_y Name_y
0    1    Alex        sub1   NaN    NaN
1    2     Amy        sub2   1.0  Billy
2    3   Allen        sub4   2.0  Brian
3    4   Alice        sub6   4.0  Bryce
4    5  Ayoung        sub5   5.0  Betty
*****************************************************************
   id_x  Name_x subject_id  id_y Name_y
0   2.0     Amy       sub2     1  Billy
1   3.0   Allen       sub4     2  Brian
2   NaN     NaN       sub3     3   Bran
3   4.0   Alice       sub6     4  Bryce
4   5.0  Ayoung       sub5     5  Betty
*****************************************************************
   id_x  Name_x subject_id  id_y Name_y
0     2     Amy       sub2     1  Billy
1     3   Allen       sub4     2  Brian
2     4   Alice       sub6     4  Bryce
3     5  Ayoung       sub5     5  Betty
*****************************************************************
   id_x  Name_x subject_id  id_y Name_y
0   1.0    Alex       sub1   NaN    NaN
1   2.0     Amy       sub2   1.0  Billy
2   3.0   Allen       sub4   2.0  Brian
3   4.0   Alice       sub6   4.0  Bryce
4   5.0  Ayoung       sub5   5.0  Betty
5   NaN     NaN       sub3   3.0   Bran
*****************************************************************
```

Concatenation:

```
In [77]:  import pandas as pd

          one = pd.DataFrame({
             'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
             'subject_id':['sub1','sub2','sub4','sub6','sub5'],
             'Marks_scored':[98,90,87,69,78]},
             index=[1,2,3,4,5])

          two = pd.DataFrame({
             'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
             'subject_id':['sub2','sub4','sub3','sub6','sub5'],
             'Marks_scored':[89,80,79,97,88]},
             index=[1,2,3,4,5])
          print(pd.concat([one,two],keys=['x','y'],ignore_index=True))
```

```
     Name subject_id  Marks_scored
0    Alex       sub1            98
1     Amy       sub2            90
2   Allen       sub4            87
3   Alice       sub6            69
4  Ayoung       sub5            78
5   Billy       sub2            89
6   Brian       sub4            80
7    Bran       sub3            79
8   Bryce       sub6            97
9   Betty       sub5            88
```

How do I read and write tabular data?

read_csv(): read*csv() function to read data stored as a csv file into a pandas DataFrame.*

pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, …), each of them with the prefix read*
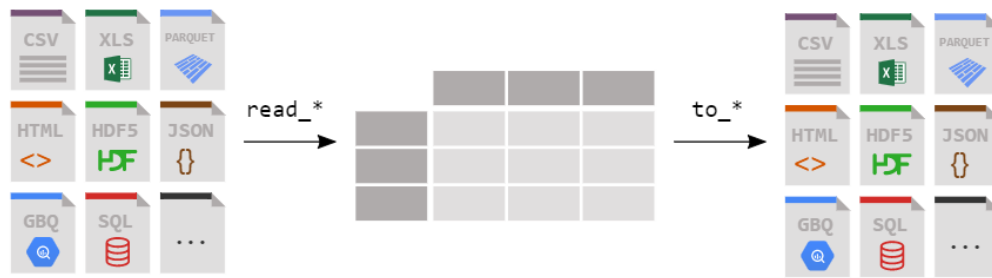

```
In [78]:   #https://www.kaggle.com/uciml/iris          (You can download the iris dat
```

```
In [79]:   import pandas as pd
           iris=pd.read_csv("D:\Python learning track and notes\Dataset\Iris.csv")
```

Explore via coding

Load the iris dataset

```
In [80]:   import pandas as pd
           iris=pd.read_csv("D:\Python learning track and notes\Dataset\Iris.csv")
           iris
```

Out[80]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Create a copy of the data set.

```
In [81]:   ds=iris.copy()   #if any changes make to ds it will not affect the iris d
           ds
```

Out[81]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Select SepalLengthCm column from dataset.

In [82]:
```python
ds['SepalLengthCm']
```

Out[82]:
```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: SepalLengthCm, Length: 150, dtype: float64
```

In [83]:
```python
print(type(ds['SepalLengthCm']))
```

```
<class 'pandas.core.series.Series'>
```

Select SepalLengthCm and PetalLengthCm.

In [84]:
```python
ds[['SepalLengthCm','PetalLengthCm']]
```

Out[84]:

	SepalLengthCm	PetalLengthCm
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
...	...	...
145	6.7	5.2
146	6.3	5.0
147	6.5	5.2
148	6.2	5.4
149	5.9	5.1

150 rows × 2 columns

filter out the column which have SepalLength >5

```
In [85]:   ds['SepalLengthCm']>5
```

```
Out[85]:   0      True
           1      False
           2      False
           3      False
           4      False
                  ...
           145    True
           146    True
           147    True
           148    True
           149    True
           Name: SepalLengthCm, Length: 150, dtype: bool
```

Create new column which is TotalPetalCm=PetalLengthCm+PetalWidthCm.

```
In [86]:   ds['TotalPetalCm']=ds['PetalLengthCm']+ds['PetalWidthCm']
           ds
```

Out[86]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	TotalPeta
0	1	5.1	3.5	1.4	0.2	Iris-setosa	
1	2	4.9	3.0	1.4	0.2	Iris-setosa	
2	3	4.7	3.2	1.3	0.2	Iris-setosa	
3	4	4.6	3.1	1.5	0.2	Iris-setosa	
4	5	5.0	3.6	1.4	0.2	Iris-setosa	
...	...	...	...	...	...	...	
145	146	6.7	3.0	5.2	2.3	Iris-virginica	
146	147	6.3	2.5	5.0	1.9	Iris-virginica	
147	148	6.5	3.0	5.2	2.0	Iris-virginica	
148	149	6.2	3.4	5.4	2.3	Iris-virginica	
149	150	5.9	3.0	5.1	1.8	Iris-virginica	

150 rows × 7 columns

Delete column TotalPetalCm from dataset

```
In [87]:   del ds['TotalPetalCm']
           ds
```

Out[87]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Acess the dataframe

```
In [88]:
ds.iloc[0:10]
```

Out[88]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Delete the first row from the dataframe

```
In [89]:
ds.drop(0, inplace=True)
ds
```

Out[89]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica

| | 149 | 150 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

149 rows × 6 columns

Delete the first 10 rows from dataframe

```
In [90]:  ds.drop(ds.index[1:10],inplace=True)
          ds
```

Out[90]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

140 rows × 6 columns

Reindex the dataframe

```
In [91]:  ds.reset_index(inplace=True, drop=True)
          ds
```

Out[91]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	2	4.9	3.0	1.4	0.2	Iris-setosa
1	12	4.8	3.4	1.6	0.2	Iris-setosa
2	13	4.8	3.0	1.4	0.1	Iris-setosa
3	14	4.3	3.0	1.1	0.1	Iris-setosa
4	15	5.8	4.0	1.2	0.2	Iris-setosa
...	...	...	...	...	...	...
135	146	6.7	3.0	5.2	2.3	Iris-virginica
136	147	6.3	2.5	5.0	1.9	Iris-virginica
137	148	6.5	3.0	5.2	2.0	Iris-virginica
138	149	6.2	3.4	5.4	2.3	Iris-virginica
139	150	5.9	3.0	5.1	1.8	Iris-virginica

140 rows × 6 columns

Use describe function.

```
In [92]:   ds.describe()
```

Out[92]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	140.000000	140.000000	140.000000	140.000000	140.000000
mean	80.435714	5.910000	3.030714	3.922857	1.268571
std	40.676511	0.812696	0.432642	1.711465	0.741677
min	2.000000	4.300000	2.000000	1.000000	0.100000
25%	45.750000	5.200000	2.800000	1.675000	0.400000
50%	80.500000	5.850000	3.000000	4.500000	1.400000
75%	115.250000	6.425000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

find the setosa flower describe

```
In [93]:   ds[ds['Species']=="Iris-virginica"].describe()
```

Out[93]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	50.00000	50.00000	50.000000	50.000000	50.00000
mean	125.50000	6.58800	2.974000	5.552000	2.02600
std	14.57738	0.63588	0.322497	0.551895	0.27465
min	101.00000	4.90000	2.200000	4.500000	1.40000
25%	113.25000	6.22500	2.800000	5.100000	1.80000
50%	125.50000	6.50000	3.000000	5.550000	2.00000
75%	137.75000	6.90000	3.175000	5.875000	2.30000
max	150.00000	7.90000	3.800000	6.900000	2.50000

find the virginica flower describe

```
In [94]:   ds[ds['Species']=="Iris-virginica"].describe()
```

Out[94]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	50.00000	50.00000	50.000000	50.000000	50.00000
mean	125.50000	6.58800	2.974000	5.552000	2.02600
std	14.57738	0.63588	0.322497	0.551895	0.27465
min	101.00000	4.90000	2.200000	4.500000	1.40000
25%	113.25000	6.22500	2.800000	5.100000	1.80000
50%	125.50000	6.50000	3.000000	5.550000	2.00000
75%	137.75000	6.90000	3.175000	5.875000	2.30000
max	150.00000	7.90000	3.800000	6.900000	2.50000

find the virginica flower describe

```
In [95]:   ds[ds['Species']=="Iris-versicolor"].describe()
```

Out[95]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm

	count	50.00000	50.000000	50.000000	50.000000	50.000000
mean	75.50000	5.936000	2.770000	4.260000	1.326000
std	14.57738	0.516171	0.313798	0.469911	0.197753
min	51.00000	4.900000	2.000000	3.000000	1.000000
25%	63.25000	5.600000	2.525000	4.000000	1.200000
50%	75.50000	5.900000	2.800000	4.350000	1.300000
75%	87.75000	6.300000	3.000000	4.600000	1.500000
max	100.00000	7.000000	3.400000	5.100000	1.800000

Sort the dataframe according to SepalLengthCm.

```
In [96]:   ds.sort_values(by='SepalLengthCm')
```

Out[96]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
3	14	4.3	3.0	1.1	0.1	Iris-setosa
32	43	4.4	3.2	1.3	0.2	Iris-setosa
28	39	4.4	3.0	1.3	0.2	Iris-setosa
31	42	4.5	2.3	1.3	0.3	Iris-setosa
37	48	4.6	3.2	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
112	123	7.7	2.8	6.7	2.0	Iris-virginica
108	119	7.7	2.6	6.9	2.3	Iris-virginica
125	136	7.7	3.0	6.1	2.3	Iris-virginica
107	118	7.7	3.8	6.7	2.2	Iris-virginica
121	132	7.9	3.8	6.4	2.0	Iris-virginica

140 rows × 6 columns

Check for the null values in PetalLengthCm.

```
In [97]:   ds['PetalLengthCm'].isnull().sum()
```

Out[97]: 0

Rename the SepalLengthCm column as SP(Cm)

```
In [98]:   ds.rename(columns={'SepalLengthCm':'SP(Cm)'},inplace=True)
           ds
```

Out[98]:

	Id	SP(Cm)	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	2	4.9	3.0	1.4	0.2	Iris-setosa
1	12	4.8	3.4	1.6	0.2	Iris-setosa
2	13	4.8	3.0	1.4	0.1	Iris-setosa
3	14	4.3	3.0	1.1	0.1	Iris-setosa
4	15	5.8	4.0	1.2	0.2	Iris-setosa
...	...	...	...	...	...	...

	Id					
135	146	6.7	3.0	5.2	2.3	Iris-virginica
136	147	6.3	2.5	5.0	1.9	Iris-virginica
137	148	6.5	3.0	5.2	2.0	Iris-virginica
138	149	6.2	3.4	5.4	2.3	Iris-virginica
139	150	5.9	3.0	5.1	1.8	Iris-virginica

140 rows × 6 columns

Rename the PetalLengthCm column as PL(Cm), SepalWidthCm as SW(Cm) and PetalWidthCm as PW(Cm).

```
In [99]:  ds.rename(columns={'PetalLengthCm':'PL(Cm)','SepalWidthCm':'SW(Cm)','Peta
          ds
```

Out[99]:

	Id	SP(Cm)	SW(Cm)	PL(Cm)	PW(Cm)	Species
0	2	4.9	3.0	1.4	0.2	Iris-setosa
1	12	4.8	3.4	1.6	0.2	Iris-setosa
2	13	4.8	3.0	1.4	0.1	Iris-setosa
3	14	4.3	3.0	1.1	0.1	Iris-setosa
4	15	5.8	4.0	1.2	0.2	Iris-setosa
...	...	...	...	...	...	...
135	146	6.7	3.0	5.2	2.3	Iris-virginica
136	147	6.3	2.5	5.0	1.9	Iris-virginica
137	148	6.5	3.0	5.2	2.0	Iris-virginica
138	149	6.2	3.4	5.4	2.3	Iris-virginica
139	150	5.9	3.0	5.1	1.8	Iris-virginica

140 rows × 6 columns

Find and print count of each kind of flower. Print the count as integer value.

```
In [100…  ds['Species'].value_counts()
```

```
Out[100…  Iris-virginica     50
          Iris-versicolor    50
          Iris-setosa        40
          Name: Species, dtype: int64
```

```
In [101…  df=ds[ds['Species']=="Iris-setosa"]
          print("Count of setosa flower: ",df['Id'].count())

          df=ds[ds['Species']=="Iris-virginica"]
          print("Count of virginica flower: ",df['Id'].count())

          df=ds[ds['Species']=="Iris-versicolor"]
          print("Count of versicolor flower: ",df['Id'].count())
```

```
          Count of setosa flower:  40
          Count of virginica flower:  50
          Count of versicolor flower:  50
```

Find the data of flower "iris-virginiva" type where petal-length>1.5

```
In [102... df=ds[ds['Species']=="Iris-virginica"]
         df[df['PL(Cm)']>1.5]
```

Out[102...

	Id	SP(Cm)	SW(Cm)	PL(Cm)	PW(Cm)	Species
90	101	6.3	3.3	6.0	2.5	Iris-virginica
91	102	5.8	2.7	5.1	1.9	Iris-virginica
92	103	7.1	3.0	5.9	2.1	Iris-virginica
93	104	6.3	2.9	5.6	1.8	Iris-virginica
94	105	6.5	3.0	5.8	2.2	Iris-virginica
95	106	7.6	3.0	6.6	2.1	Iris-virginica
96	107	4.9	2.5	4.5	1.7	Iris-virginica
97	108	7.3	2.9	6.3	1.8	Iris-virginica
98	109	6.7	2.5	5.8	1.8	Iris-virginica
99	110	7.2	3.6	6.1	2.5	Iris-virginica
100	111	6.5	3.2	5.1	2.0	Iris-virginica
101	112	6.4	2.7	5.3	1.9	Iris-virginica
102	113	6.8	3.0	5.5	2.1	Iris-virginica
103	114	5.7	2.5	5.0	2.0	Iris-virginica
104	115	5.8	2.8	5.1	2.4	Iris-virginica
105	116	6.4	3.2	5.3	2.3	Iris-virginica
106	117	6.5	3.0	5.5	1.8	Iris-virginica
107	118	7.7	3.8	6.7	2.2	Iris-virginica
108	119	7.7	2.6	6.9	2.3	Iris-virginica
109	120	6.0	2.2	5.0	1.5	Iris-virginica
110	121	6.9	3.2	5.7	2.3	Iris-virginica
111	122	5.6	2.8	4.9	2.0	Iris-virginica
112	123	7.7	2.8	6.7	2.0	Iris-virginica
113	124	6.3	2.7	4.9	1.8	Iris-virginica
114	125	6.7	3.3	5.7	2.1	Iris-virginica
115	126	7.2	3.2	6.0	1.8	Iris-virginica
116	127	6.2	2.8	4.8	1.8	Iris-virginica
117	128	6.1	3.0	4.9	1.8	Iris-virginica
118	129	6.4	2.8	5.6	2.1	Iris-virginica
119	130	7.2	3.0	5.8	1.6	Iris-virginica
120	131	7.4	2.8	6.1	1.9	Iris-virginica
121	132	7.9	3.8	6.4	2.0	Iris-virginica
122	133	6.4	2.8	5.6	2.2	Iris-virginica
123	134	6.3	2.8	5.1	1.5	Iris-virginica
124	135	6.1	2.6	5.6	1.4	Iris-virginica
125	136	7.7	3.0	6.1	2.3	Iris-virginica

126	137	6.3	3.4	5.6	2.4	Iris-virginica
127	138	6.4	3.1	5.5	1.8	Iris-virginica
128	139	6.0	3.0	4.8	1.8	Iris-virginica
129	140	6.9	3.1	5.4	2.1	Iris-virginica
130	141	6.7	3.1	5.6	2.4	Iris-virginica
131	142	6.9	3.1	5.1	2.3	Iris-virginica
132	143	5.8	2.7	5.1	1.9	Iris-virginica
133	144	6.8	3.2	5.9	2.3	Iris-virginica
134	145	6.7	3.3	5.7	2.5	Iris-virginica
135	146	6.7	3.0	5.2	2.3	Iris-virginica
136	147	6.3	2.5	5.0	1.9	Iris-virginica
137	148	6.5	3.0	5.2	2.0	Iris-virginica
138	149	6.2	3.4	5.4	2.3	Iris-virginica
139	150	5.9	3.0	5.1	1.8	Iris-virginica

Find and print the minimum and maximum of the feature for each kind of flower.

In [103...

```python
df=ds[ds['Species']=="Iris-setosa"]
print("Setosa")
print("The minimum value of: \n",df.min())
print("The maximum value of: \n",df.max())
print("*********************************")
df=ds[ds['Species']=="Iris-virginica"]
df=ds[ds['Species']==""]
print("Verginica")
print("The minimum value of: \n",df.min())
print("The maximum value of: \n",df.max())
print("*********************************")
df=ds[ds['Species']=="Iris-versicolor"]
print("Versicolor")
print("The minimum value of: \n",df.min())
print("The maximum value of: \n",df.max())
```

```
Setosa
The minimum value of:
 Id                     2
SP(Cm)              4.3
SW(Cm)              2.3
PL(Cm)                1
PW(Cm)              0.1
Species      Iris-setosa
dtype: object
The maximum value of:
 Id                    50
SP(Cm)              5.8
SW(Cm)              4.4
PL(Cm)              1.9
PW(Cm)              0.6
Species      Iris-setosa
dtype: object
*********************************
Verginica
The minimum value of:
 Id           NaN
SP(Cm)       NaN
SW(Cm)       NaN
```

```
PL(Cm)      NaN
PW(Cm)      NaN
Species     NaN
dtype: float64
The maximum value of:
 Id         NaN
SP(Cm)      NaN
SW(Cm)      NaN
PL(Cm)      NaN
PW(Cm)      NaN
Species     NaN
dtype: float64
**********************************
Versicolor
The minimum value of:
 Id                        51
SP(Cm)                    4.9
SW(Cm)                      2
PL(Cm)                      3
PW(Cm)                      1
Species     Iris-versicolor
dtype: object
The maximum value of:
 Id                       100
SP(Cm)                      7
SW(Cm)                    3.4
PL(Cm)                    5.1
PW(Cm)                    1.8
Species     Iris-versicolor
dtype: object
```

Use group by on species.

```python
import pandas as pd
iris=pd.read_csv("D:\Python learning track and notes\Dataset\Iris.csv")
grouped_df=iris.groupby('Species')
for name,group in grouped_df:
    print("Name is: ",name)
    print("Group is:")
    print(group)
```

```
Name is:  Iris-setosa
Group is:
     Id SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Spe
cies
0    1           5.1           3.5            1.4           0.2  Iris-se
tosa
1    2           4.9           3.0            1.4           0.2  Iris-se
tosa
2    3           4.7           3.2            1.3           0.2  Iris-se
tosa
3    4           4.6           3.1            1.5           0.2  Iris-se
tosa
4    5           5.0           3.6            1.4           0.2  Iris-se
tosa
5    6           5.4           3.9            1.7           0.4  Iris-se
tosa
6    7           4.6           3.4            1.4           0.3  Iris-se
tosa
7    8           5.0           3.4            1.5           0.2  Iris-se
tosa
8    9           4.4           2.9            1.4           0.2  Iris-se
tosa
9   10           4.9           3.1            1.5           0.1  Iris-se
tosa
10  11           5.4           3.7            1.5           0.2  Iris-se
tosa
11  12           4.8           3.4            1.6           0.2  Iris-se
```

					tosa
12 13	4.8	3.0	1.4	0.1	Iris-se
					tosa
13 14	4.3	3.0	1.1	0.1	Iris-se
					tosa
14 15	5.8	4.0	1.2	0.2	Iris-se
					tosa
15 16	5.7	4.4	1.5	0.4	Iris-se
					tosa
16 17	5.4	3.9	1.3	0.4	Iris-se
					tosa
17 18	5.1	3.5	1.4	0.3	Iris-se
					tosa
18 19	5.7	3.8	1.7	0.3	Iris-se
					tosa
19 20	5.1	3.8	1.5	0.3	Iris-se
					tosa
20 21	5.4	3.4	1.7	0.2	Iris-se
					tosa
21 22	5.1	3.7	1.5	0.4	Iris-se
					tosa
22 23	4.6	3.6	1.0	0.2	Iris-se
					tosa
23 24	5.1	3.3	1.7	0.5	Iris-se
					tosa
24 25	4.8	3.4	1.9	0.2	Iris-se
					tosa
25 26	5.0	3.0	1.6	0.2	Iris-se
					tosa
26 27	5.0	3.4	1.6	0.4	Iris-se
					tosa
27 28	5.2	3.5	1.5	0.2	Iris-se
					tosa
28 29	5.2	3.4	1.4	0.2	Iris-se
					tosa
29 30	4.7	3.2	1.6	0.2	Iris-se
					tosa
30 31	4.8	3.1	1.6	0.2	Iris-se
					tosa
31 32	5.4	3.4	1.5	0.4	Iris-se
					tosa
32 33	5.2	4.1	1.5	0.1	Iris-se
					tosa
33 34	5.5	4.2	1.4	0.2	Iris-se
					tosa
34 35	4.9	3.1	1.5	0.1	Iris-se
					tosa
35 36	5.0	3.2	1.2	0.2	Iris-se
					tosa
36 37	5.5	3.5	1.3	0.2	Iris-se
					tosa
37 38	4.9	3.1	1.5	0.1	Iris-se
					tosa
38 39	4.4	3.0	1.3	0.2	Iris-se
					tosa
39 40	5.1	3.4	1.5	0.2	Iris-se
					tosa
40 41	5.0	3.5	1.3	0.3	Iris-se
					tosa
41 42	4.5	2.3	1.3	0.3	Iris-se
					tosa
42 43	4.4	3.2	1.3	0.2	Iris-se
					tosa
43 44	5.0	3.5	1.6	0.6	Iris-se
					tosa
44 45	5.1	3.8	1.9	0.4	Iris-se
					tosa
45 46	4.8	3.0	1.4	0.3	Iris-se

```
tosa
46  47              5.1            3.8              1.6              0.2  Iris-se
tosa
47  48              4.6            3.2              1.4              0.2  Iris-se
tosa
48  49              5.3            3.7              1.5              0.2  Iris-se
tosa
49  50              5.0            3.3              1.4              0.2  Iris-se
tosa
Name is:  Iris-versicolor
Group is:
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
50  51              7.0            3.2              4.7              1.4
51  52              6.4            3.2              4.5              1.5
52  53              6.9            3.1              4.9              1.5
53  54              5.5            2.3              4.0              1.3
54  55              6.5            2.8              4.6              1.5
55  56              5.7            2.8              4.5              1.3
56  57              6.3            3.3              4.7              1.6
57  58              4.9            2.4              3.3              1.0
58  59              6.6            2.9              4.6              1.3
59  60              5.2            2.7              3.9              1.4
60  61              5.0            2.0              3.5              1.0
61  62              5.9            3.0              4.2              1.5
62  63              6.0            2.2              4.0              1.0
63  64              6.1            2.9              4.7              1.4
64  65              5.6            2.9              3.6              1.3
65  66              6.7            3.1              4.4              1.4
66  67              5.6            3.0              4.5              1.5
67  68              5.8            2.7              4.1              1.0
68  69              6.2            2.2              4.5              1.5
69  70              5.6            2.5              3.9              1.1
70  71              5.9            3.2              4.8              1.8
71  72              6.1            2.8              4.0              1.3
72  73              6.3            2.5              4.9              1.5
73  74              6.1            2.8              4.7              1.2
74  75              6.4            2.9              4.3              1.3
75  76              6.6            3.0              4.4              1.4
76  77              6.8            2.8              4.8              1.4
77  78              6.7            3.0              5.0              1.7
78  79              6.0            2.9              4.5              1.5
79  80              5.7            2.6              3.5              1.0
80  81              5.5            2.4              3.8              1.1
81  82              5.5            2.4              3.7              1.0
82  83              5.8            2.7              3.9              1.2
83  84              6.0            2.7              5.1              1.6
84  85              5.4            3.0              4.5              1.5
85  86              6.0            3.4              4.5              1.6
86  87              6.7            3.1              4.7              1.5
87  88              6.3            2.3              4.4              1.3
88  89              5.6            3.0              4.1              1.3
89  90              5.5            2.5              4.0              1.3
90  91              5.5            2.6              4.4              1.2
91  92              6.1            3.0              4.6              1.4
92  93              5.8            2.6              4.0              1.2
93  94              5.0            2.3              3.3              1.0
94  95              5.6            2.7              4.2              1.3
95  96              5.7            3.0              4.2              1.2
96  97              5.7            2.9              4.2              1.3
97  98              6.2            2.9              4.3              1.3
98  99              5.1            2.5              3.0              1.1
99  100             5.7            2.8              4.1              1.3

            Species
50  Iris-versicolor
51  Iris-versicolor
52  Iris-versicolor
53  Iris-versicolor
```

```
54   Iris-versicolor
55   Iris-versicolor
56   Iris-versicolor
57   Iris-versicolor
58   Iris-versicolor
59   Iris-versicolor
60   Iris-versicolor
61   Iris-versicolor
62   Iris-versicolor
63   Iris-versicolor
64   Iris-versicolor
65   Iris-versicolor
66   Iris-versicolor
67   Iris-versicolor
68   Iris-versicolor
69   Iris-versicolor
70   Iris-versicolor
71   Iris-versicolor
72   Iris-versicolor
73   Iris-versicolor
74   Iris-versicolor
75   Iris-versicolor
76   Iris-versicolor
77   Iris-versicolor
78   Iris-versicolor
79   Iris-versicolor
80   Iris-versicolor
81   Iris-versicolor
82   Iris-versicolor
83   Iris-versicolor
84   Iris-versicolor
85   Iris-versicolor
86   Iris-versicolor
87   Iris-versicolor
88   Iris-versicolor
89   Iris-versicolor
90   Iris-versicolor
91   Iris-versicolor
92   Iris-versicolor
93   Iris-versicolor
94   Iris-versicolor
95   Iris-versicolor
96   Iris-versicolor
97   Iris-versicolor
98   Iris-versicolor
99   Iris-versicolor
Name is:  Iris-virginica
Group is:
     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
100  101            6.3           3.3            6.0           2.5
101  102            5.8           2.7            5.1           1.9
102  103            7.1           3.0            5.9           2.1
103  104            6.3           2.9            5.6           1.8
104  105            6.5           3.0            5.8           2.2
105  106            7.6           3.0            6.6           2.1
106  107            4.9           2.5            4.5           1.7
107  108            7.3           2.9            6.3           1.8
108  109            6.7           2.5            5.8           1.8
109  110            7.2           3.6            6.1           2.5
110  111            6.5           3.2            5.1           2.0
111  112            6.4           2.7            5.3           1.9
112  113            6.8           3.0            5.5           2.1
113  114            5.7           2.5            5.0           2.0
114  115            5.8           2.8            5.1           2.4
115  116            6.4           3.2            5.3           2.3
116  117            6.5           3.0            5.5           1.8
117  118            7.7           3.8            6.7           2.2
118  119            7.7           2.6            6.9           2.3
```

```
119  120              6.0        2.2        5.0        1.5
120  121              6.9        3.2        5.7        2.3
121  122              5.6        2.8        4.9        2.0
122  123              7.7        2.8        6.7        2.0
123  124              6.3        2.7        4.9        1.8
124  125              6.7        3.3        5.7        2.1
125  126              7.2        3.2        6.0        1.8
126  127              6.2        2.8        4.8        1.8
127  128              6.1        3.0        4.9        1.8
128  129              6.4        2.8        5.6        2.1
129  130              7.2        3.0        5.8        1.6
130  131              7.4        2.8        6.1        1.9
131  132              7.9        3.8        6.4        2.0
132  133              6.4        2.8        5.6        2.2
133  134              6.3        2.8        5.1        1.5
134  135              6.1        2.6        5.6        1.4
135  136              7.7        3.0        6.1        2.3
136  137              6.3        3.4        5.6        2.4
137  138              6.4        3.1        5.5        1.8
138  139              6.0        3.0        4.8        1.8
139  140              6.9        3.1        5.4        2.1
140  141              6.7        3.1        5.6        2.4
141  142              6.9        3.1        5.1        2.3
142  143              5.8        2.7        5.1        1.9
143  144              6.8        3.2        5.9        2.3
144  145              6.7        3.3        5.7        2.5
145  146              6.7        3.0        5.2        2.3
146  147              6.3        2.5        5.0        1.9
147  148              6.5        3.0        5.2        2.0
148  149              6.2        3.4        5.4        2.3
149  150              5.9        3.0        5.1        1.8

                 Species
100  Iris-virginica
101  Iris-virginica
102  Iris-virginica
103  Iris-virginica
104  Iris-virginica
105  Iris-virginica
106  Iris-virginica
107  Iris-virginica
108  Iris-virginica
109  Iris-virginica
110  Iris-virginica
111  Iris-virginica
112  Iris-virginica
113  Iris-virginica
114  Iris-virginica
115  Iris-virginica
116  Iris-virginica
117  Iris-virginica
118  Iris-virginica
119  Iris-virginica
120  Iris-virginica
121  Iris-virginica
122  Iris-virginica
123  Iris-virginica
124  Iris-virginica
125  Iris-virginica
126  Iris-virginica
127  Iris-virginica
128  Iris-virginica
129  Iris-virginica
130  Iris-virginica
131  Iris-virginica
132  Iris-virginica
133  Iris-virginica
134  Iris-virginica
```

```
135   Iris-virginica
136   Iris-virginica
137   Iris-virginica
138   Iris-virginica
139   Iris-virginica
140   Iris-virginica
141   Iris-virginica
142   Iris-virginica
143   Iris-virginica
144   Iris-virginica
145   Iris-virginica
146   Iris-virginica
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica
```

In [105… `grouped_df.get_group('Iris-versicolor')`

Out[105…

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
53	54	5.5	2.3	4.0	1.3	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
55	56	5.7	2.8	4.5	1.3	Iris-versicolor
56	57	6.3	3.3	4.7	1.6	Iris-versicolor
57	58	4.9	2.4	3.3	1.0	Iris-versicolor
58	59	6.6	2.9	4.6	1.3	Iris-versicolor
59	60	5.2	2.7	3.9	1.4	Iris-versicolor
60	61	5.0	2.0	3.5	1.0	Iris-versicolor
61	62	5.9	3.0	4.2	1.5	Iris-versicolor
62	63	6.0	2.2	4.0	1.0	Iris-versicolor
63	64	6.1	2.9	4.7	1.4	Iris-versicolor
64	65	5.6	2.9	3.6	1.3	Iris-versicolor
65	66	6.7	3.1	4.4	1.4	Iris-versicolor
66	67	5.6	3.0	4.5	1.5	Iris-versicolor
67	68	5.8	2.7	4.1	1.0	Iris-versicolor
68	69	6.2	2.2	4.5	1.5	Iris-versicolor
69	70	5.6	2.5	3.9	1.1	Iris-versicolor
70	71	5.9	3.2	4.8	1.8	Iris-versicolor
71	72	6.1	2.8	4.0	1.3	Iris-versicolor
72	73	6.3	2.5	4.9	1.5	Iris-versicolor
73	74	6.1	2.8	4.7	1.2	Iris-versicolor
74	75	6.4	2.9	4.3	1.3	Iris-versicolor
75	76	6.6	3.0	4.4	1.4	Iris-versicolor
76	77	6.8	2.8	4.8	1.4	Iris-versicolor
77	78	6.7	3.0	5.0	1.7	Iris-versicolor

78	79	6.0	2.9	4.5	1.5	Iris-versicolor
79	80	5.7	2.6	3.5	1.0	Iris-versicolor
80	81	5.5	2.4	3.8	1.1	Iris-versicolor
81	82	5.5	2.4	3.7	1.0	Iris-versicolor
82	83	5.8	2.7	3.9	1.2	Iris-versicolor
83	84	6.0	2.7	5.1	1.6	Iris-versicolor
84	85	5.4	3.0	4.5	1.5	Iris-versicolor
85	86	6.0	3.4	4.5	1.6	Iris-versicolor
86	87	6.7	3.1	4.7	1.5	Iris-versicolor
87	88	6.3	2.3	4.4	1.3	Iris-versicolor
88	89	5.6	3.0	4.1	1.3	Iris-versicolor
89	90	5.5	2.5	4.0	1.3	Iris-versicolor
90	91	5.5	2.6	4.4	1.2	Iris-versicolor
91	92	6.1	3.0	4.6	1.4	Iris-versicolor
92	93	5.8	2.6	4.0	1.2	Iris-versicolor
93	94	5.0	2.3	3.3	1.0	Iris-versicolor
94	95	5.6	2.7	4.2	1.3	Iris-versicolor
95	96	5.7	3.0	4.2	1.2	Iris-versicolor
96	97	5.7	2.9	4.2	1.3	Iris-versicolor
97	98	6.2	2.9	4.3	1.3	Iris-versicolor
98	99	5.1	2.5	3.0	1.1	Iris-versicolor
99	100	5.7	2.8	4.1	1.3	Iris-versicolor