By Shreeyansh Das, Source: gfg, Pandas Documentation

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- *Size mutability*: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

Why Pandas is used for Data Science - This is because pandas are used in conjunction with other libraries that are used for data science. It is built on the top of the NumPy library which means that a lot of structures of NumPy are used or replicated in Pandas. The data produced by Pandas are often used as input for plotting functions of Matplotlib, statistical analysis in SciPy, machine learning algorithms in Scikit-learn. Pandas program can be run from any text editor but it is recommended to use Jupyter Notebook for this as Jupyter given the ability to execute code in a particular cell rather than executing the entire file. Jupyter also provides an easy way to visualize pandas data frames and plots.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

---

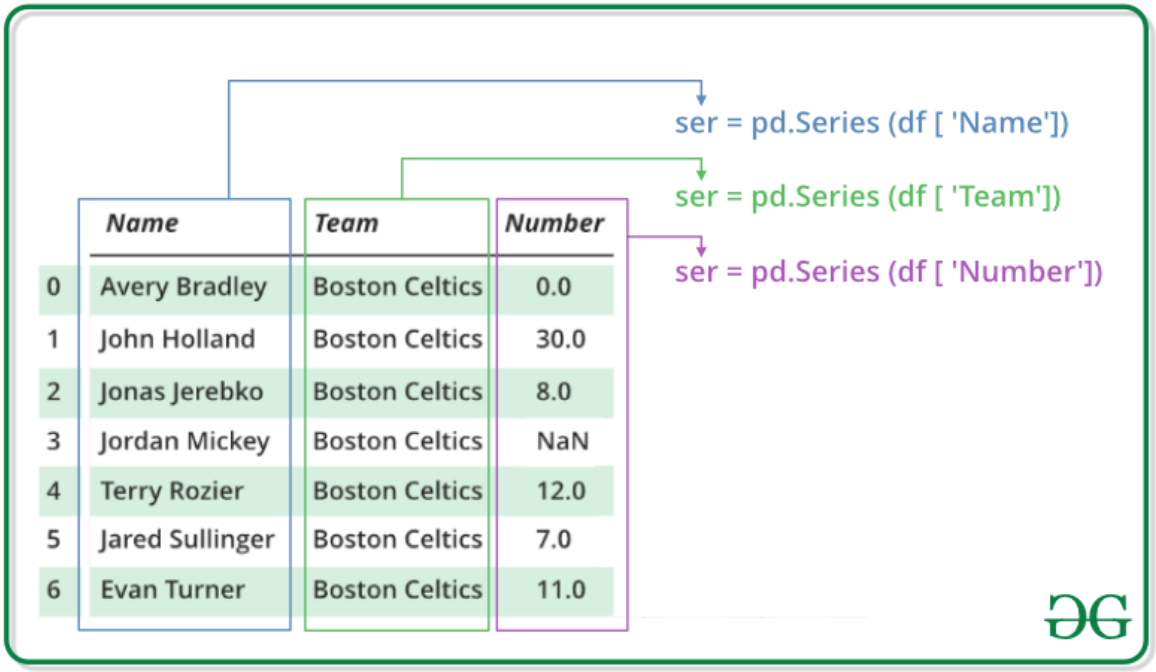
1. Introduction

Pandas generally provide two data structures for manipulating data, They are:

- Series
- DataFrame

1.1 Series

Pandas Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. **Pandas Series is nothing but a column in an excel sheet**. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.



1.1.1 Creating a Pandas Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas Series can be created from the lists, dictionary, and from a scalar value etc. Series can be created in different ways, here are some ways by which we create a series:

Creating a series from array : In order to create a series from array, we have to import a numpy module and have to use `array()` function.

In [2]:
```
arr = np.array(['Volt','Excalibur','Gauss','Mag','Xaku','Revenant','Rhino','Garuda',
```

In [3]:
```
S = pd.Series(arr)
S
```

Out[3]:
```
0         Volt
1    Excalibur
2        Gauss
3          Mag
4         Xaku
```

```
5     Revenant
6        Rhino
7       Garuda
8         Loki
9        Nidus
dtype: object
```

Creating series from list : In order to create a series from list, we have to first create a list after that we can create a series from list.

In [4]:
```
lst = list(arr)
```

In [5]:
```
L = pd.Series(lst)
L
```

Out[5]:
```
0         Volt
1    Excalibur
2        Gauss
3          Mag
4         Xaku
5     Revenant
6        Rhino
7       Garuda
8         Loki
9        Nidus
dtype: object
```

Creating Series From Dictionary : In order to create a series from dictionary, we have to first create a dictionary after that we can make a series using dictionary. Dictionary key are used to construct a index.

In [6]:
```
dct = {1:'Nidus', 2:'Excalibur Umbra', 3:'Volt Prime', 4:'Gauss Prime', 5: 'Atlas Pr
pd.Series(dct)
```

Out[6]:
```
1              Nidus
2    Excalibur Umbra
3         Volt Prime
4        Gauss Prime
5        Atlas Prime
dtype: object
```

Creating a series from Scalar value : In order to create a series from scalar value, an index must be provided. The scalar value will be repeated to match the length of index.

In [7]:
```
pd.Series(10, index = np.asarray(np.arange(0,6,1)))
```

Out[7]:
```
0    10
1    10
2    10
3    10
4    10
5    10
dtype: int64
```

1.1.2 Accessing Elements

There are two ways through which we can access element of series, they are :

- Accessing Element from Series with Position
- Accessing Element Using Label (index)

Accessing Elements via Position : In order to access the series element refers to the index number. Use the index operator `[ ]` to access an element in a series. The index must be an integer. In order to access multiple elements from a series, we use Slice operation.

In [8]:
```
# Accessing 1st five elements
S[:5]
```

Out[8]:
```
0         Volt
1    Excalibur
2        Gauss
3          Mag
4         Xaku
dtype: object
```

In [9]:
```
# Accessing elements from position 4 to position 7
S[4:8]
```

Out[9]:
```
4        Xaku
5    Revenant
6       Rhino
7      Garuda
dtype: object
```

Accessing Elements via Labels : In order to access an element from series, we have to set values by index label. A Series is like a fixed-size dictionary in that you can get and set values by index label.

In [10]:
```
data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])
ser = pd.Series(data, index = np.asarray(np.arange(10,23,1)))
```

In [11]:
```
ser[15]
```

Out[11]:
```
'f'
```

1.1.3. Indexing and Selecting Data

Indexing in pandas means simply selecting particular data from a Series. Indexing could mean selecting all the data, some of the data from particular columns. Indexing can also be known as Subset Selection. It can be done via 3 methods

- Index Operator - `[ ]`
- Using `.loc`
- Using `.iloc`

Index Operator : Indexing operator is used to refer to the square brackets following an object. In this indexing operator to refer to `df[ ]`

In [12]:
```
S[4:9]                                          #from index 4 to index 8
```

Out[12]:
```
4        Xaku
5    Revenant
6       Rhino
7      Garuda
8        Loki
dtype: object
```

`df.loc[ ]` **Method** : This function selects data by refering the explicit index . The `df.loc[ ]` indexer selects data in a different way than just the indexing operator. It can select subsets of data. *It doesnt excludes the last index.*

In [13]:
```
S.loc[4:9]                                    #from index 4 to index 9
```

Out[13]:
```
4        Xaku
5    Revenant
6       Rhino
7      Garuda
8        Loki
9       Nidus
dtype: object
```

`df.iloc[ ]` **Method** : This function allows us to retrieve data by position. In order to do that, we'll need to specify the positions of the data that we want. The `df.iloc` indexer is very similar to `df.loc` but only uses integer locations to make its selections. *This is similar to index operator since it excludes the last index*

In [14]:
```
S.iloc[4:9]                                   #from index 4 to index 8
```

Out[14]:
```
4        Xaku
5    Revenant
6       Rhino
7      Garuda
8        Loki
dtype: object
```

The main distinction between `loc` and `iloc` is:

- `loc` is label-based, which means that you have to specify rows and columns based on their row and column labels.
- `iloc` is integer position-based, so you have to specify rows and columns by their integer position values (0-based integer position).

	loc	iloc
A value	A single label or integer e.g. `loc[A]` or `loc[1]`	A single integer e.g. `iloc[1]`
A list	A list of labels e.g. `loc[[A, B]]`	A list of integers e.g. `iloc[[1,2,3]]`
Slicing	e.g. `loc[A:B]`, A and B are included	e.g. `iloc[n:m]`, n is included, m is excluded
Conditions	A bool Series or list	A bool list
Callable function	`loc[lambda x: x[2]]`	`iloc[lambda x: x[2]]`

1.1.4 Binary Operations on Series

We can perform binary operation on series like addition, subtraction and many other operation. In order to perform binary operation on series we have to use some function like

`.add()`, `.sub()`, `.mul()`, etc.

- `add()` - Method is used to add series or list like objects with same length to the caller series
- `sub()` - Method is used to subtract series or list like objects with same length from the caller series
- `mul()` - Method is used to multiply series or list like objects with same length with the caller series
- `div()` - Method is used to divide series or list like objects with same length by the caller series
- `sum()` - Returns the sum of the values for the requested axis
- `prod()` - Returns the product of the values for the requested axis
- `mean()` - Returns the mean of the values for the requested axis
- `pow()` - Method is used to put each element of passed series as exponential power of caller series and returned the results
- `abs()` - Method is used to get the absolute numeric value of each element in Series/DataFrame
- `cov()` - Method is used to find covariance of two series

For non matching indices, pass the `fill_value` argument.

In [15]:
```
d1 = pd.Series([5, 2, 3,7], index=['a', 'b', 'c', 'd'])
d2 = pd.Series([1, 6, 4, 9], index=['a', 'b', 'd', 'e'])
```

In [16]:
```
d1.add(d2, fill_value = 0)
```

Out[16]:
```
a     6.0
b     8.0
c     3.0
d    11.0
e     9.0
dtype: float64
```

In [17]:
```
d1.sub(d2, fill_value = 0)
```

Out[17]:
```
a    4.0
b   -4.0
c    3.0
d    3.0
e   -9.0
dtype: float64
```

1.1.4 Conversion Operation on Series

In conversion operation we perform various operation like changing datatype of series, changing a series to list etc. In order to perform conversion operation we have various function which help in conversion like `.astype()`, `.tolist()` etc. Changing the Series into numpy array is acheived by using a method `Series.to_numpy()` or `Series.as_matrix()`.

In [18]:
```
iris = sns.load_dataset('iris')
```

In [19]:
```
iris.head()
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [20]:
```
iris['sepal_width'].astype(int)
```

Out[20]:
```
0      3
1      3
2      3
3      3
4      3
      ..
145    3
146    2
147    3
148    3
149    3
Name: sepal_width, Length: 150, dtype: int32
```

In [21]:
```
iris['petal_length'].tolist()[:10]     #showing only first 10 conversions
```

Out[21]:
```
[1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5]
```

In [22]:
```
iris['petal_length'].to_numpy()[:10]
```

Out[22]:
```
array([1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5])
```

1.1.5 Miscellaneous Pandas Series Operations

In [23]:
```
iris['sepal_length'].count()      #Returns number of non-NA/null observations in the
```

Out[23]:
```
150
```

In [24]:
```
iris.size                         #Returns the number of elements in the underlying
```

Out[24]:
```
750
```

In [25]:
```
S.name = "Warframes"              #Method allows to give a name to a Series object,
```

In [26]:
```
iris['sepal_length'].is_unique    #Method returns boolean if values in the object ar
```

Out[26]:
```
False
```

In [27]:
```
iris['species'].unique()        #to see the unique values in a particular column
```

Out[27]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

In [28]:
```
iris['species'].nunique()  #to see the no. of unique values in a particular column
```

Out[28]:
```
3
```

In [29]:
```
iris['sepal_length'].idxmin()  #To see the idx with minimum value in a series, for m
```

Out[29]:
```
13
```

- **Comapre Series** - `le()` , `ge()` , `lt()` , `gt()` , `eq()` , `ne()`

In [30]:
```
a = pd.Series([1, 1, 1, np.nan], index = ['a', 'b', 'c', 'd'])
b = pd.Series([2, np.nan, 0.5, np.nan], index = ['a', 'b', 'd', 'e'])

a.le(b, fill_value = 0) #Used to compare every element of Caller series (a here) les
```

Out[30]:
```
a     True
b    False
c    False
d     True
e    False
dtype: bool
```

In [31]:
```
a = pd.Series([1, 1, 1, np.nan], index = ['a', 'b', 'c', 'd'])
b = pd.Series([1, np.nan, 1, np.nan], index = ['a', 'b', 'd', 'e'])

a.eq(b, fill_value = 0) #returns True for every element in Caller Series which is Eq
```

Out[31]:
```
a     True
b    False
c    False
d    False
e    False
dtype: bool
```

In [32]:
```
a.eq(b, fill_value = 0) #Return Equal to of series and other, element-wise (binary o
```

Out[32]:
```
a     True
b    False
c    False
d    False
e    False
dtype: bool
```

- **Check for Empty Dataframe**

In [33]:
```
iris = sns.load_dataset('iris')
iris.empty
```

Out[33]:
```
False
```

- **Check for NaN's**

`df.hasnans` - Return if true have any nans; enables various perf speedups.

In [34]:
```
b.hasnans
```

Out[34]:
```
True
```

- **Drop NaN's**

`Series.dropna(axis=0, inplace=False, how=None)` - Return a new Series with missing values removed. Parameters

1. axis{*0 or 'index'*}, default 0 - There is only one axis to drop values from.
2. inplacebool, default False - If True, do operation inplace and return None.
3. howstr, optional - Not in use. Kept for compatibility.

In [35]:
```
ser = pd.Series([1., 2., 3., 4., np.nan])
ser.dropna()
```

Out[35]:
```
0    1.0
1    2.0
2    3.0
3    4.0
dtype: float64
```

- **Fill NaN's**

`Series.fillna(value, method, axis, inplace)` - Fill NA/NaN values using the specified method.

1. valuescalar, dict, Series, or DataFrame - Value to use to fill holes (e.g. 0), alternately a dict/Series/DataFrame of values specifying which value to use for each index (for a Series) or column (for a DataFrame). Values not in the dict/Series/DataFrame will not be filled. This value cannot be a list.
2. method{'*backfill*', '*bfill*', '*pad*', '*ffill*', *None*}, default None - Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use next valid observation to fill gap.
3. axis{0 or 'index'} - Axis along which to fill missing values.

In [36]:
```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list("ABCD"))
df
```

Out[36]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5
3	NaN	3.0	NaN	4

In [37]:
```
df.fillna(0) #fill all NaN with 0
```

Out[37]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	0.0	3.0	0.0	4

In [38]:
```
df.fillna(value = {"A": 0, "B": 1, "C": 2, "D": 3}) #Replace all NaN elements with s
```

Out[38]:

	A	B	C	D
0	0.0	2.0	2.0	0
1	3.0	4.0	2.0	1
2	0.0	1.0	2.0	5
3	0.0	3.0	2.0	4

- **Combine 2 series**

`s1.combine(s2, func, fill_value)` - Combine the Series with a Series or scalar according to func. Combine the Series and other using func to perform elementwise selection for combined Series. fill_value is assumed when value is missing at some index from one of the two objects being combined.

In [39]:
```
s1 = pd.Series({'falcon': 330.0, 'eagle': 160.0})
s2 = pd.Series({'falcon': 345.0, 'eagle': 200.0, 'duck': 30.0})
s1.combine(s2, max)
```

Out[39]:
```
duck        NaN
eagle     200.0
falcon    345.0
dtype: float64
```

- **Removing Series Elements**

`Series.drop(labels, axis, index, columns)` - Remove elements of a Series based on specifying the index labels. When using a multi-index, labels on different levels can be removed by specifying the level.

`labels` - Index labels to drop.

`axis`, default 0 - Redundant for application on Series.

`index` - Redundant for application on Series, but 'index' can be used instead of 'labels'.

`columns` - No change is made to the Series; use 'index' or 'labels' instead.

In [40]:
```
s = pd.Series(data=np.arange(5), index=['A', 'B', 'C', 'D', 'E'])
```

```
s.drop(labels=['B', 'C'])
```

Out[40]:
```
A    0
D    3
E    4
dtype: int32
```

- **Find Series Duplicates**

Indicate duplicate Series values.

`Series.duplicated(keep = 'first')` - Duplicated values are indicated as True values in the resulting Series. Either all duplicates, all except the first or all except the last occurrence of duplicates can be indicated.

Parameters - keep{'first', 'last', False}, default 'first'. Method to handle dropping duplicates:

1. 'first' : Mark duplicates as True except for the first occurrence.
2. 'last' : Mark duplicates as True except for the last occurrence.
3. False : Mark all duplicates as True.

In [41]:
```
animals = pd.Series(['lama', 'cow', 'lama', 'beetle', 'lama'])
animals.duplicated()
```

Out[41]:
```
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

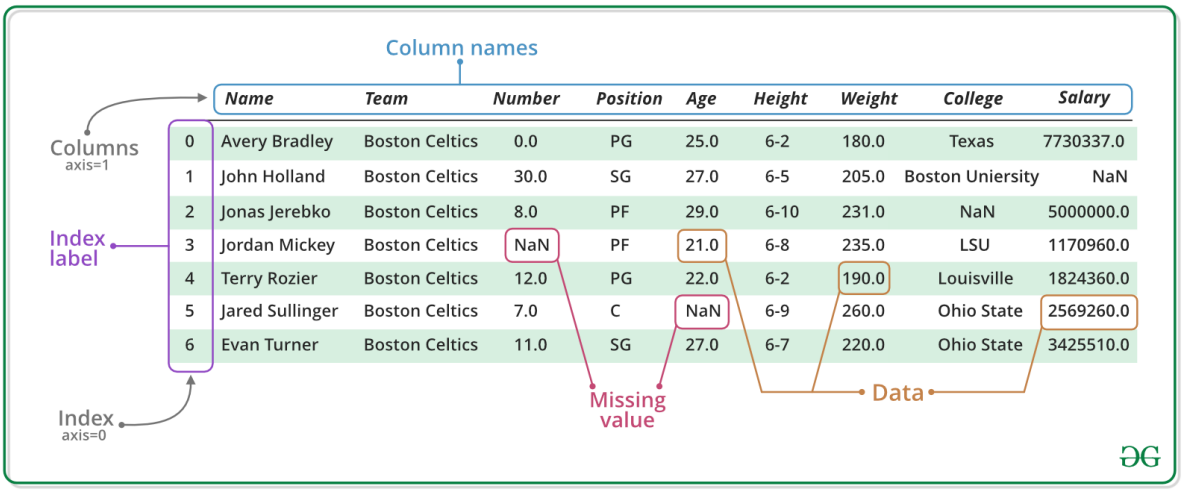
In [42]:
```
animals.duplicated(keep = False)
```

Out[42]:
```
0     True
1    False
2     True
3    False
4     True
dtype: bool
```

1.2 DataFrame

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame

consists of three principal components - the data, rows, and columns.



1.2.1 Creating Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. However, Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

- **Empty Dataframe**

In [43]:
```
pd.DataFrame()
```

Out[43]: —

- **Dataframe from List**

In [44]:
```
np.asarray(lst).T #List
```

Out[44]:
```
array(['Volt', 'Excalibur', 'Gauss', 'Mag', 'Xaku', 'Revenant', 'Rhino',
       'Garuda', 'Loki', 'Nidus'], dtype='<U9')
```

In [45]:
```
pd.DataFrame(lst) #Dataframe from list
```

Out[45]:

	0
0	Volt
1	Excalibur
2	Gauss
3	Mag
4	Xaku
5	Revenant
6	Rhino
7	Garuda
8	Loki
9	Nidus

- **Dataframe from Dictionary** To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

In [46]:
```
data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
```

In [47]:
```
pd.DataFrame(data)
```

Out[47]:

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

1.2.2. Rows and Columns Selection

- **Rows Selection** Pandas provide a unique method to retrieve rows from a Data frame. `DataFrame.loc[]` method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an `iloc[]` function.

In [48]:
```
iris.loc[3]             # 3rd index record
```

Out[48]:
```
sepal_length       4.6
sepal_width        3.1
petal_length       1.5
petal_width        0.2
species         setosa
Name: 3, dtype: object
```

OR

In [49]:
```
iris.loc[:4]             # First 5 records
```

Out[49]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [50]:
```
iris.loc[:4,['sepal_length','species']]  # First 5 records with specific columns
```

Out[50]:

	sepal_length	species
0	5.1	setosa

	sepal_length	species
1	4.9	setosa
2	4.7	setosa
3	4.6	setosa
4	5.0	setosa

In [51]:
```
iris.loc[[3,5,7]]  #Specific Records - note the double brackets
```

Out[51]:

	sepal_length	sepal_width	petal_length	petal_width	species
3	4.6	3.1	1.5	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
7	5.0	3.4	1.5	0.2	setosa

In [52]:
```
iris.sample(frac = 0.5).head() #Randomly select fraction of rows.
```

Out[52]:

	sepal_length	sepal_width	petal_length	petal_width	species
129	7.2	3.0	5.8	1.6	virginica
6	4.6	3.4	1.4	0.3	setosa
130	7.4	2.8	6.1	1.9	virginica
39	5.1	3.4	1.5	0.2	setosa
30	4.8	3.1	1.6	0.2	setosa

In [53]:
```
iris.sample(n = 34).head(3)  #Randomly select 'n' rows
```

Out[53]:

	sepal_length	sepal_width	petal_length	petal_width	species
96	5.7	2.9	4.2	1.3	versicolor
118	7.7	2.6	6.9	2.3	virginica
81	5.5	2.4	3.7	1.0	versicolor

- **Column Selection** In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

In [54]:
```
iris['sepal_width'][:6] # First 6 entries in column named 'sepal_width'
```

Out[54]:
```
0    3.5
1    3.0
2    3.2
3    3.1
4    3.6
5    3.9
Name: sepal_width, dtype: float64
```

In [55]:
```
iris[iris.columns[1:3]].head() #Select 2nd to 3rd column.
```

Out[55]:

	sepal_width	petal_length
0	3.5	1.4
1	3.0	1.4
2	3.2	1.3
3	3.1	1.5
4	3.6	1.4

In [56]:
```
iris.loc[3:9,'sepal_width':'petal_width']  #Continuous Selection of columns
```

Out[56]:

	sepal_width	petal_length	petal_width
3	3.1	1.5	0.2
4	3.6	1.4	0.2
5	3.9	1.7	0.4
6	3.4	1.4	0.3
7	3.4	1.5	0.2
8	2.9	1.4	0.2
9	3.1	1.5	0.1

- **Explicit Selection**

In [57]:
```
iris.iloc[0:5, 1:3]
```

Out[57]:

	sepal_width	petal_length
0	3.5	1.4
1	3.0	1.4
2	3.2	1.3
3	3.1	1.5
4	3.6	1.4

1.2.3 Handling Missing Data

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA(Not Available) values in pandas.

- **Checking for missing values using `isnull()` and `notnull()`** : In order to check missing values in Pandas DataFrame, we use a function `isnull()` and `notnull()`. Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

In [58]:
```
df
```

Out[58]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5
3	NaN	3.0	NaN	4

In [59]:

```
df.isnull()
```

Out[59]:

	A	B	C	D
0	True	False	True	False
1	False	False	True	False
2	True	True	True	False
3	True	False	True	False

- **Fill missing values using `fillna()`, `replace()`, `interpolate()`** : These functions replace NaN values with some value of their own. All these function help in filling a null values in datasets of a DataFrame. `Interpolate()` function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.

In [60]:

```
df.fillna(0.0)
```

Out[60]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	0.0	3.0	0.0	4

In [61]:

```
df.interpolate()
```

Out[61]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	3.0	3.5	NaN	5
3	3.0	3.0	NaN	4

As we can see the output, values in the first row could not get filled as the direction of filling of values is forward and there is no previous value which could have been used in interpolation.

In [62]:

```
df.fillna(method = 'pad') #Filling null values with the previous ones
```

Out[62]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	3.0	4.0	NaN	5
3	3.0	3.0	NaN	4

In [63]:
```
df.fillna(method ='bfill') #Filling null value with the next ones
```

Out[63]:

	A	B	C	D
0	3.0	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	3.0	NaN	5
3	NaN	3.0	NaN	4

- **Dropping missing values using `dropna()`** : In order to drop a null values from a dataframe, we used `dropna()` function this fuction drop Rows/Columns of datasets with Null values in different ways.

In [64]:
```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65],
        'Fifth Score':[12, 78, 45, 90]}
```

In [65]:
```
DT = pd.DataFrame(dict)
DT
```

Out[65]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score
0	100.0	30.0	52	NaN	12
1	90.0	NaN	40	NaN	78
2	NaN	45.0	80	NaN	45
3	95.0	56.0	98	65.0	90

In [66]:
```
dict_2 = {'First Score':[100, np.nan, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, np.nan, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
DT_2 = pd.DataFrame(dict_2)
DT_2
```

Out[66]:

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	NaN
1	NaN	NaN	NaN	NaN

	First Score	Second Score	Third Score	Fourth Score
2	NaN	45.0	80.0	NaN
3	95.0	56.0	98.0	65.0

In [67]:
```
DT_2.dropna(axis = 0, how = 'all') # drop rows whose all data is missing or contain
```

Out[67]:

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	NaN
2	NaN	45.0	80.0	NaN
3	95.0	56.0	98.0	65.0

In [68]:
```
DT.dropna()                          #Dropping rows with atleast 1 NaN
```

Out[68]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score
3	95.0	56.0	98	65.0	90

In [69]:
```
DT.dropna(axis = 1)                  #Dropping Columns with with atleast 1 NaN
```

Out[69]:

	Third Score	Fifth Score
0	52	12
1	40	78
2	80	45
3	98	90

1.2.4 Iterating over Dataframe

Pandas DataFrame consists of rows and columns so, in order to iterate over dataframe, we have to iterate a dataframe like a dictionary.

- **Iterating Over Rows** : In order to iterate over rows, we can use three function `iteritems()` , `iterrows()` , `itertuples()` . These three function will help in iteration over rows.

In [70]:
```
for i,j in iris.iterrows():
    print(i,j)
```

```
0 sepal_length       5.1
sepal_width        3.5
petal_length       1.4
petal_width        0.2
species         setosa
Name: 0, dtype: object
1 sepal_length       4.9
sepal_width        3.0
petal_length       1.4
petal_width        0.2
species         setosa
```

```
Name: 1, dtype: object
2 sepal_length       4.7
sepal_width        3.2
petal_length       1.3
petal_width        0.2
species         setosa
Name: 2, dtype: object
3 sepal_length       4.6
sepal_width        3.1
petal_length       1.5
petal_width        0.2
species         setosa
Name: 3, dtype: object
4 sepal_length       5.0
sepal_width        3.6
petal_length       1.4
petal_width        0.2
species         setosa
Name: 4, dtype: object
5 sepal_length       5.4
sepal_width        3.9
petal_length       1.7
petal_width        0.4
species         setosa
Name: 5, dtype: object
6 sepal_length       4.6
sepal_width        3.4
petal_length       1.4
petal_width        0.3
species         setosa
Name: 6, dtype: object
7 sepal_length       5.0
sepal_width        3.4
petal_length       1.5
petal_width        0.2
species         setosa
Name: 7, dtype: object
8 sepal_length       4.4
sepal_width        2.9
petal_length       1.4
petal_width        0.2
species         setosa
Name: 8, dtype: object
9 sepal_length       4.9
sepal_width        3.1
petal_length       1.5
petal_width        0.1
species         setosa
Name: 9, dtype: object
10 sepal_length       5.4
sepal_width        3.7
petal_length       1.5
petal_width        0.2
species         setosa
Name: 10, dtype: object
11 sepal_length       4.8
sepal_width        3.4
petal_length       1.6
petal_width        0.2
species         setosa
Name: 11, dtype: object
12 sepal_length       4.8
sepal_width        3.0
petal_length       1.4
petal_width        0.1
species         setosa
Name: 12, dtype: object
13 sepal_length       4.3
sepal_width        3.0
```

```
petal_length       1.1
petal_width        0.1
species         setosa
Name: 13, dtype: object
14 sepal_length       5.8
sepal_width        4.0
petal_length       1.2
petal_width        0.2
species         setosa
Name: 14, dtype: object
15 sepal_length       5.7
sepal_width        4.4
petal_length       1.5
petal_width        0.4
species         setosa
Name: 15, dtype: object
16 sepal_length       5.4
sepal_width        3.9
petal_length       1.3
petal_width        0.4
species         setosa
Name: 16, dtype: object
17 sepal_length       5.1
sepal_width        3.5
petal_length       1.4
petal_width        0.3
species         setosa
Name: 17, dtype: object
18 sepal_length       5.7
sepal_width        3.8
petal_length       1.7
petal_width        0.3
species         setosa
Name: 18, dtype: object
19 sepal_length       5.1
sepal_width        3.8
petal_length       1.5
petal_width        0.3
species         setosa
Name: 19, dtype: object
20 sepal_length       5.4
sepal_width        3.4
petal_length       1.7
petal_width        0.2
species         setosa
Name: 20, dtype: object
21 sepal_length       5.1
sepal_width        3.7
petal_length       1.5
petal_width        0.4
species         setosa
Name: 21, dtype: object
22 sepal_length       4.6
sepal_width        3.6
petal_length       1.0
petal_width        0.2
species         setosa
Name: 22, dtype: object
23 sepal_length       5.1
sepal_width        3.3
petal_length       1.7
petal_width        0.5
species         setosa
Name: 23, dtype: object
24 sepal_length       4.8
sepal_width        3.4
petal_length       1.9
petal_width        0.2
species         setosa
```

```
Name: 24, dtype: object
25 sepal_length       5.0
sepal_width        3.0
petal_length       1.6
petal_width        0.2
species         setosa
Name: 25, dtype: object
26 sepal_length       5.0
sepal_width        3.4
petal_length       1.6
petal_width        0.4
species         setosa
Name: 26, dtype: object
27 sepal_length       5.2
sepal_width        3.5
petal_length       1.5
petal_width        0.2
species         setosa
Name: 27, dtype: object
28 sepal_length       5.2
sepal_width        3.4
petal_length       1.4
petal_width        0.2
species         setosa
Name: 28, dtype: object
29 sepal_length       4.7
sepal_width        3.2
petal_length       1.6
petal_width        0.2
species         setosa
Name: 29, dtype: object
30 sepal_length       4.8
sepal_width        3.1
petal_length       1.6
petal_width        0.2
species         setosa
Name: 30, dtype: object
31 sepal_length       5.4
sepal_width        3.4
petal_length       1.5
petal_width        0.4
species         setosa
Name: 31, dtype: object
32 sepal_length       5.2
sepal_width        4.1
petal_length       1.5
petal_width        0.1
species         setosa
Name: 32, dtype: object
33 sepal_length       5.5
sepal_width        4.2
petal_length       1.4
petal_width        0.2
species         setosa
Name: 33, dtype: object
34 sepal_length       4.9
sepal_width        3.1
petal_length       1.5
petal_width        0.2
species         setosa
Name: 34, dtype: object
35 sepal_length       5.0
sepal_width        3.2
petal_length       1.2
petal_width        0.2
species         setosa
Name: 35, dtype: object
36 sepal_length       5.5
sepal_width        3.5
```

```
petal_length       1.3
petal_width        0.2
species         setosa
Name: 36, dtype: object
37 sepal_length       4.9
sepal_width        3.6
petal_length       1.4
petal_width        0.1
species         setosa
Name: 37, dtype: object
38 sepal_length       4.4
sepal_width        3.0
petal_length       1.3
petal_width        0.2
species         setosa
Name: 38, dtype: object
39 sepal_length       5.1
sepal_width        3.4
petal_length       1.5
petal_width        0.2
species         setosa
Name: 39, dtype: object
40 sepal_length       5.0
sepal_width        3.5
petal_length       1.3
petal_width        0.3
species         setosa
Name: 40, dtype: object
41 sepal_length       4.5
sepal_width        2.3
petal_length       1.3
petal_width        0.3
species         setosa
Name: 41, dtype: object
42 sepal_length       4.4
sepal_width        3.2
petal_length       1.3
petal_width        0.2
species         setosa
Name: 42, dtype: object
43 sepal_length       5.0
sepal_width        3.5
petal_length       1.6
petal_width        0.6
species         setosa
Name: 43, dtype: object
44 sepal_length       5.1
sepal_width        3.8
petal_length       1.9
petal_width        0.4
species         setosa
Name: 44, dtype: object
45 sepal_length       4.8
sepal_width        3.0
petal_length       1.4
petal_width        0.3
species         setosa
Name: 45, dtype: object
46 sepal_length       5.1
sepal_width        3.8
petal_length       1.6
petal_width        0.2
species         setosa
Name: 46, dtype: object
47 sepal_length       4.6
sepal_width        3.2
petal_length       1.4
petal_width        0.2
species         setosa
```

```
Name: 47, dtype: object
48 sepal_length       5.3
sepal_width        3.7
petal_length       1.5
petal_width        0.2
species         setosa
Name: 48, dtype: object
49 sepal_length       5.0
sepal_width        3.3
petal_length       1.4
petal_width        0.2
species         setosa
Name: 49, dtype: object
50 sepal_length           7.0
sepal_width            3.2
petal_length           4.7
petal_width            1.4
species         versicolor
Name: 50, dtype: object
51 sepal_length           6.4
sepal_width            3.2
petal_length           4.5
petal_width            1.5
species         versicolor
Name: 51, dtype: object
52 sepal_length           6.9
sepal_width            3.1
petal_length           4.9
petal_width            1.5
species         versicolor
Name: 52, dtype: object
53 sepal_length           5.5
sepal_width            2.3
petal_length           4.0
petal_width            1.3
species         versicolor
Name: 53, dtype: object
54 sepal_length           6.5
sepal_width            2.8
petal_length           4.6
petal_width            1.5
species         versicolor
Name: 54, dtype: object
55 sepal_length           5.7
sepal_width            2.8
petal_length           4.5
petal_width            1.3
species         versicolor
Name: 55, dtype: object
56 sepal_length           6.3
sepal_width            3.3
petal_length           4.7
petal_width            1.6
species         versicolor
Name: 56, dtype: object
57 sepal_length           4.9
sepal_width            2.4
petal_length           3.3
petal_width            1.0
species         versicolor
Name: 57, dtype: object
58 sepal_length           6.6
sepal_width            2.9
petal_length           4.6
petal_width            1.3
species         versicolor
Name: 58, dtype: object
59 sepal_length           5.2
sepal_width            2.7
```

```
petal_length           3.9
petal_width            1.4
species         versicolor
Name: 59, dtype: object
60 sepal_length           5.0
sepal_width            2.0
petal_length           3.5
petal_width            1.0
species         versicolor
Name: 60, dtype: object
61 sepal_length           5.9
sepal_width            3.0
petal_length           4.2
petal_width            1.5
species         versicolor
Name: 61, dtype: object
62 sepal_length           6.0
sepal_width            2.2
petal_length           4.0
petal_width            1.0
species         versicolor
Name: 62, dtype: object
63 sepal_length           6.1
sepal_width            2.9
petal_length           4.7
petal_width            1.4
species         versicolor
Name: 63, dtype: object
64 sepal_length           5.6
sepal_width            2.9
petal_length           3.6
petal_width            1.3
species         versicolor
Name: 64, dtype: object
65 sepal_length           6.7
sepal_width            3.1
petal_length           4.4
petal_width            1.4
species         versicolor
Name: 65, dtype: object
66 sepal_length           5.6
sepal_width            3.0
petal_length           4.5
petal_width            1.5
species         versicolor
Name: 66, dtype: object
67 sepal_length           5.8
sepal_width            2.7
petal_length           4.1
petal_width            1.0
species         versicolor
Name: 67, dtype: object
68 sepal_length           6.2
sepal_width            2.2
petal_length           4.5
petal_width            1.5
species         versicolor
Name: 68, dtype: object
69 sepal_length           5.6
sepal_width            2.5
petal_length           3.9
petal_width            1.1
species         versicolor
Name: 69, dtype: object
70 sepal_length           5.9
sepal_width            3.2
petal_length           4.8
petal_width            1.8
species         versicolor
```

```
Name: 70, dtype: object
71 sepal_length           6.1
sepal_width            2.8
petal_length           4.0
petal_width            1.3
species         versicolor
Name: 71, dtype: object
72 sepal_length           6.3
sepal_width            2.5
petal_length           4.9
petal_width            1.5
species         versicolor
Name: 72, dtype: object
73 sepal_length           6.1
sepal_width            2.8
petal_length           4.7
petal_width            1.2
species         versicolor
Name: 73, dtype: object
74 sepal_length           6.4
sepal_width            2.9
petal_length           4.3
petal_width            1.3
species         versicolor
Name: 74, dtype: object
75 sepal_length           6.6
sepal_width            3.0
petal_length           4.4
petal_width            1.4
species         versicolor
Name: 75, dtype: object
76 sepal_length           6.8
sepal_width            2.8
petal_length           4.8
petal_width            1.4
species         versicolor
Name: 76, dtype: object
77 sepal_length           6.7
sepal_width            3.0
petal_length           5.0
petal_width            1.7
species         versicolor
Name: 77, dtype: object
78 sepal_length           6.0
sepal_width            2.9
petal_length           4.5
petal_width            1.5
species         versicolor
Name: 78, dtype: object
79 sepal_length           5.7
sepal_width            2.6
petal_length           3.5
petal_width            1.0
species         versicolor
Name: 79, dtype: object
80 sepal_length           5.5
sepal_width            2.4
petal_length           3.8
petal_width            1.1
species         versicolor
Name: 80, dtype: object
81 sepal_length           5.5
sepal_width            2.4
petal_length           3.7
petal_width            1.0
species         versicolor
Name: 81, dtype: object
82 sepal_length           5.8
sepal_width            2.7
```

```
petal_length           3.9
petal_width            1.2
species         versicolor
Name: 82, dtype: object
83 sepal_length           6.0
sepal_width            2.7
petal_length           5.1
petal_width            1.6
species         versicolor
Name: 83, dtype: object
84 sepal_length           5.4
sepal_width            3.0
petal_length           4.5
petal_width            1.5
species         versicolor
Name: 84, dtype: object
85 sepal_length           6.0
sepal_width            3.4
petal_length           4.5
petal_width            1.6
species         versicolor
Name: 85, dtype: object
86 sepal_length           6.7
sepal_width            3.1
petal_length           4.7
petal_width            1.5
species         versicolor
Name: 86, dtype: object
87 sepal_length           6.3
sepal_width            2.3
petal_length           4.4
petal_width            1.3
species         versicolor
Name: 87, dtype: object
88 sepal_length           5.6
sepal_width            3.0
petal_length           4.1
petal_width            1.3
species         versicolor
Name: 88, dtype: object
89 sepal_length           5.5
sepal_width            2.5
petal_length           4.0
petal_width            1.3
species         versicolor
Name: 89, dtype: object
90 sepal_length           5.5
sepal_width            2.6
petal_length           4.4
petal_width            1.2
species         versicolor
Name: 90, dtype: object
91 sepal_length           6.1
sepal_width            3.0
petal_length           4.6
petal_width            1.4
species         versicolor
Name: 91, dtype: object
92 sepal_length           5.8
sepal_width            2.6
petal_length           4.0
petal_width            1.2
species         versicolor
Name: 92, dtype: object
93 sepal_length           5.0
sepal_width            2.3
petal_length           3.3
petal_width            1.0
species         versicolor
```

```
Name: 93, dtype: object
94 sepal_length          5.6
sepal_width            2.7
petal_length           4.2
petal_width            1.3
species         versicolor
Name: 94, dtype: object
95 sepal_length          5.7
sepal_width            3.0
petal_length           4.2
petal_width            1.2
species         versicolor
Name: 95, dtype: object
96 sepal_length          5.7
sepal_width            2.9
petal_length           4.2
petal_width            1.3
species         versicolor
Name: 96, dtype: object
97 sepal_length          6.2
sepal_width            2.9
petal_length           4.3
petal_width            1.3
species         versicolor
Name: 97, dtype: object
98 sepal_length          5.1
sepal_width            2.5
petal_length           3.0
petal_width            1.1
species         versicolor
Name: 98, dtype: object
99 sepal_length          5.7
sepal_width            2.8
petal_length           4.1
petal_width            1.3
species         versicolor
Name: 99, dtype: object
100 sepal_length         6.3
sepal_width           3.3
petal_length          6.0
petal_width           2.5
species         virginica
Name: 100, dtype: object
101 sepal_length         5.8
sepal_width           2.7
petal_length          5.1
petal_width           1.9
species         virginica
Name: 101, dtype: object
102 sepal_length         7.1
sepal_width           3.0
petal_length          5.9
petal_width           2.1
species         virginica
Name: 102, dtype: object
103 sepal_length         6.3
sepal_width           2.9
petal_length          5.6
petal_width           1.8
species         virginica
Name: 103, dtype: object
104 sepal_length         6.5
sepal_width           3.0
petal_length          5.8
petal_width           2.2
species         virginica
Name: 104, dtype: object
105 sepal_length         7.6
sepal_width           3.0
```

```
petal_length          6.6
petal_width           2.1
species         virginica
Name: 105, dtype: object
106 sepal_length          4.9
sepal_width           2.5
petal_length          4.5
petal_width           1.7
species         virginica
Name: 106, dtype: object
107 sepal_length          7.3
sepal_width           2.9
petal_length          6.3
petal_width           1.8
species         virginica
Name: 107, dtype: object
108 sepal_length          6.7
sepal_width           2.5
petal_length          5.8
petal_width           1.8
species         virginica
Name: 108, dtype: object
109 sepal_length          7.2
sepal_width           3.6
petal_length          6.1
petal_width           2.5
species         virginica
Name: 109, dtype: object
110 sepal_length          6.5
sepal_width           3.2
petal_length          5.1
petal_width           2.0
species         virginica
Name: 110, dtype: object
111 sepal_length          6.4
sepal_width           2.7
petal_length          5.3
petal_width           1.9
species         virginica
Name: 111, dtype: object
112 sepal_length          6.8
sepal_width           3.0
petal_length          5.5
petal_width           2.1
species         virginica
Name: 112, dtype: object
113 sepal_length          5.7
sepal_width           2.5
petal_length          5.0
petal_width           2.0
species         virginica
Name: 113, dtype: object
114 sepal_length          5.8
sepal_width           2.8
petal_length          5.1
petal_width           2.4
species         virginica
Name: 114, dtype: object
115 sepal_length          6.4
sepal_width           3.2
petal_length          5.3
petal_width           2.3
species         virginica
Name: 115, dtype: object
116 sepal_length          6.5
sepal_width           3.0
petal_length          5.5
petal_width           1.8
species         virginica
```

```
Name: 116, dtype: object
117 sepal_length          7.7
sepal_width           3.8
petal_length          6.7
petal_width           2.2
species         virginica
Name: 117, dtype: object
118 sepal_length          7.7
sepal_width           2.6
petal_length          6.9
petal_width           2.3
species         virginica
Name: 118, dtype: object
119 sepal_length          6.0
sepal_width           2.2
petal_length          5.0
petal_width           1.5
species         virginica
Name: 119, dtype: object
120 sepal_length          6.9
sepal_width           3.2
petal_length          5.7
petal_width           2.3
species         virginica
Name: 120, dtype: object
121 sepal_length          5.6
sepal_width           2.8
petal_length          4.9
petal_width           2.0
species         virginica
Name: 121, dtype: object
122 sepal_length          7.7
sepal_width           2.8
petal_length          6.7
petal_width           2.0
species         virginica
Name: 122, dtype: object
123 sepal_length          6.3
sepal_width           2.7
petal_length          4.9
petal_width           1.8
species         virginica
Name: 123, dtype: object
124 sepal_length          6.7
sepal_width           3.3
petal_length          5.7
petal_width           2.1
species         virginica
Name: 124, dtype: object
125 sepal_length          7.2
sepal_width           3.2
petal_length          6.0
petal_width           1.8
species         virginica
Name: 125, dtype: object
126 sepal_length          6.2
sepal_width           2.8
petal_length          4.8
petal_width           1.8
species         virginica
Name: 126, dtype: object
127 sepal_length          6.1
sepal_width           3.0
petal_length          4.9
petal_width           1.8
species         virginica
Name: 127, dtype: object
128 sepal_length          6.4
sepal_width           2.8
```

```
petal_length          5.6
petal_width           2.1
species         virginica
Name: 128, dtype: object
129 sepal_length          7.2
sepal_width           3.0
petal_length          5.8
petal_width           1.6
species         virginica
Name: 129, dtype: object
130 sepal_length          7.4
sepal_width           2.8
petal_length          6.1
petal_width           1.9
species         virginica
Name: 130, dtype: object
131 sepal_length          7.9
sepal_width           3.8
petal_length          6.4
petal_width           2.0
species         virginica
Name: 131, dtype: object
132 sepal_length          6.4
sepal_width           2.8
petal_length          5.6
petal_width           2.2
species         virginica
Name: 132, dtype: object
133 sepal_length          6.3
sepal_width           2.8
petal_length          5.1
petal_width           1.5
species         virginica
Name: 133, dtype: object
134 sepal_length          6.1
sepal_width           2.6
petal_length          5.6
petal_width           1.4
species         virginica
Name: 134, dtype: object
135 sepal_length          7.7
sepal_width           3.0
petal_length          6.1
petal_width           2.3
species         virginica
Name: 135, dtype: object
136 sepal_length          6.3
sepal_width           3.4
petal_length          5.6
petal_width           2.4
species         virginica
Name: 136, dtype: object
137 sepal_length          6.4
sepal_width           3.1
petal_length          5.5
petal_width           1.8
species         virginica
Name: 137, dtype: object
138 sepal_length          6.0
sepal_width           3.0
petal_length          4.8
petal_width           1.8
species         virginica
Name: 138, dtype: object
139 sepal_length          6.9
sepal_width           3.1
petal_length          5.4
petal_width           2.1
species         virginica
```

```
Name: 139, dtype: object
140 sepal_length          6.7
sepal_width           3.1
petal_length          5.6
petal_width           2.4
species         virginica
Name: 140, dtype: object
141 sepal_length          6.9
sepal_width           3.1
petal_length          5.1
petal_width           2.3
species         virginica
Name: 141, dtype: object
142 sepal_length          5.8
sepal_width           2.7
petal_length          5.1
petal_width           1.9
species         virginica
Name: 142, dtype: object
143 sepal_length          6.8
sepal_width           3.2
petal_length          5.9
petal_width           2.3
species         virginica
Name: 143, dtype: object
144 sepal_length          6.7
sepal_width           3.3
petal_length          5.7
petal_width           2.5
species         virginica
Name: 144, dtype: object
145 sepal_length          6.7
sepal_width           3.0
petal_length          5.2
petal_width           2.3
species         virginica
Name: 145, dtype: object
146 sepal_length          6.3
sepal_width           2.5
petal_length          5.0
petal_width           1.9
species         virginica
Name: 146, dtype: object
147 sepal_length          6.5
sepal_width           3.0
petal_length          5.2
petal_width           2.0
species         virginica
Name: 147, dtype: object
148 sepal_length          6.2
sepal_width           3.4
petal_length          5.4
petal_width           2.3
species         virginica
Name: 148, dtype: object
149 sepal_length          5.9
sepal_width           3.0
petal_length          5.1
petal_width           1.8
species         virginica
Name: 149, dtype: object
```

- **Iterating over Columns** :In order to iterate over columns, we need to create a list of dataframe columns and then iterate through that list to pull out the dataframe columns.

In [71]:

```
list(iris)
```

Out[71]: ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

In [72]:
```
for i in list(iris):
    print(iris[i][2])    #print details of 3rd record
```

```
4.7
3.2
1.3
0.2
setosa
```

which is same as

In [73]:
```
iris.loc[2]
```

Out[73]:
```
sepal_length       4.7
sepal_width        3.2
petal_length       1.3
petal_width        0.2
species         setosa
Name: 2, dtype: object
```

1.2.5 Miscellaneous Dataframe Operations

- **First 5 Entries**

In [74]:
```
iris.head()
```

Out[74]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- **Last 5 entries**

In [75]:
```
iris.tail()
```

Out[75]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

- **Descriptive statistics.**

In [76]:
```
iris.describe()
```

Out[76]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [77]:
```
iris.index  #The index (row labels) of the DataFrame.
```

Out[77]:
```
RangeIndex(start=0, stop=150, step=1)
```

In [78]:
```
iris.columns #The column labels of the DataFrame.
```

Out[78]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

OR

In [79]:
```
list(iris)
```

Out[79]:
```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
```

Use `sorted(df)` to get column names in alphabetical order.

- **No. of Columns**

In [80]:
```
len(iris.columns)
```

Out[80]:
```
5
```

- **No. of Rows**

In [81]:
```
len(iris)  #no. of rows in dataframe
```

Out[81]:
```
150
```

- **Concise summary of a DataFrame.**

In [82]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
```

```
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

- **Datatypes of DataFrame**

In [83]:
```
iris.dtypes
```

Out[83]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

- **Numpy Representation of DataFrame**

In [84]:
```
iris.values[:5]
```

Out[84]:
```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
       [4.9, 3.0, 1.4, 0.2, 'setosa'],
       [4.7, 3.2, 1.3, 0.2, 'setosa'],
       [4.6, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.6, 1.4, 0.2, 'setosa']], dtype=object)
```

In [85]:
```
type(iris.values[:5])
```

Out[85]:
```
numpy.ndarray
```

In [86]:
```
iris.ndim #Return an int representing the number of axes / array dimensions.
```

Out[86]:
```
2
```

In [87]:
```
iris.size  #Return an int representing the number of elements in this object.
```

Out[87]:
```
750
```

In [88]:
```
iris.memory_usage()   #Return the memory usage of each column in bytes.
```

Out[88]:
```
Index            128
sepal_length    1200
sepal_width     1200
petal_length    1200
petal_width     1200
species         1200
dtype: int64
```

- **Convert to NumPy Array**

DataFrame can be converted to NumPy ndarray with the help of `Dataframe.to_numpy()` method.

In [89]:
```
iris.to_numpy()[:5]
```

Out[89]:
```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
       [4.9, 3.0, 1.4, 0.2, 'setosa'],
       [4.7, 3.2, 1.3, 0.2, 'setosa'],
       [4.6, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.6, 1.4, 0.2, 'setosa']], dtype=object)
```

- **Accessing Specific Value**

Pandas `at[]` is used to return data in a dataframe at the passed location. The passed location is in the format [position, Column Name]. This method works in a similar way to Pandas `loc[ ]` but `at[ ]` is used to return an only single value and hence works faster than it.

Note

1. Unlike, `DataFrame.loc[ ]`, this method only returns single value. Hence `DataFrame.at[3:6, label]` will return an error.
2. Since this method only works for single values, it is faster than `DataFrame.loc[]` method.

In [90]:
```
iris.at[3,'species']
```

Out[90]:
```
'setosa'
```