Introduction to Pandas

Importing the `Pandas package`.

In [1]:
```python
import pandas as pd #this will import pandas into your workspace
```

`Data Structures` in pandas

There are two basic data structures in pandas: Series and DataFrame

Series

It is similar to a NumPy 1-dimensional array. In addition to the values that are specified by the programmer, pandas attaches a label to each of the values. If the labels are not provided by the programmer, then pandas assigns labels (0 for first element, 1 for second element and so on). A benefit of assigning labels to data values is that it becomes easier to perform manipulations on the dataset as the whole dataset becomes more of a dictionary where each value is associated with a label.

An example of first python `series`.

In [2]:
```python
series1 = pd.Series([10,20,30,40]) #we have used a list to create a series.
print(series1)
```
```
0    10
1    20
2    30
3    40
dtype: int64
```

Printing the `values` that are there in the series

In [3]:
```python
series1.values
```
Out[3]:
```
array([10, 20, 30, 40], dtype=int64)
```

In [4]:
```python
series1.values[0]
```
Out[4]:
```
10
```

Specifying `custom index values` rather than the default ones provided, you can do so using the following command.

In [5]:
```python
series2 = pd.Series([10,20,30,40,50], index=['one','two','three','four','five'])
series2
```
Out[5]:
```
one      10
two      20
three    30
four     40
five     50
dtype: int64
```

In [6]:
```python
# Lets print the element which is there in the position 2
```

In [7]:
```python
series2[4]
```
Out[7]:
```
50
```

Lets retrive the element using `index number`.

In [8]:
```python
series2 = pd.Series([10,20,30,40,50], index=['one','two','three','four','five'])
print(series2)
```
```
one      10
two      20
three    30
four     40
five     50
dtype: int64
```

In [9]:
```python
series2['three']
```
Out[9]:
```
30
```

Lets access multiple elements.

In [10]:
```python
series2[['one', 'three', 'five']]
```
Out[10]:
```
one      10
three    30
five     50
dtype: int64
```

Lets `add "4"` to each element of the series (math operations)

In [11]:
```python
series2 + 4
```
Out[11]:
```
one      14
two      24
three    34
four     44
five     54
dtype: int64
```

Lets `subset` the entire series whose value is greater than 30

In [12]:
```python
series2[series2>30]
```
Out[12]:
```
four    40
five    50
dtype: int64
```

Data Frame

DataFrame is a tabular data structure in which data is laid out in rows and column format (similar to a CSV or tables in SQL file), but it can also be used for higher dimensional data sets. The DataFrame object can contain homogenous and heterogenous values, and can be thought of as a logical extension of Series data structures. In contrast to Series, where there is one index, a DataFrame object has one index for column and one index for rows. This allows flexibility in accessing and manipulating data.

Lets `create` a Data Frame with multiple columns called Price, Ticker and Company.

In [13]:
```python
data = pd.DataFrame({'price':[95, 25, 85, 41, 78],
                     'ticker':['AXP', 'CSCO', 'DIS', 'MSFT', 'WMT'],
                     'company':['American Express', 'Cisco', 'Walt Disney','Microsoft', 'Walmart']})
data
```
Out[13]:

	price	ticker	company
0	95	AXP	American Express
1	25	CSCO	Cisco
2	85	DIS	Walt Disney
3	41	MSFT	Microsoft
4	78	WMT	Walmart

In [14]:
```python
data.shape
```
Out[14]:
```
(5, 3)
```

Note: If a column is passed with no values, it will simply have NaN values

How to access a `specific column` from the data frame?

In [15]:
```python
data['company']
```
Out[15]:
```
0    American Express
1               Cisco
2         Walt Disney
3           Microsoft
4             Walmart
Name: company, dtype: object
```

How to access a `specific row` from the data frame?

In [16]:
```python
data.loc[2] #Will print all the elements of second row
```
Out[16]:
```
price               85
ticker             DIS
company    Walt Disney
Name: 2, dtype: object
```

Note: loc is the shortform form for index.

How to `add` a new column in the data frame?

In [17]:
```python
data['Year'] = 2014
data
```
Out[17]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014

How to create a column and populate it with `missing values(NaN)`?

In [18]:
```python
data['delta_col'] = 'NaN'
data
```
Out[18]:

	price	ticker	company	Year	delta_col
0	95	AXP	American Express	2014	NaN
1	25	CSCO	Cisco	2014	NaN
2	85	DIS	Walt Disney	2014	NaN
3	41	MSFT	Microsoft	2014	NaN
4	78	WMT	Walmart	2014	NaN

How to `delete` a column

In [19]:
```python
# del data['name_of_the_col_to_delete']

del data['delta_col']
print(data)
```
```
   price ticker           company  Year
0     95    AXP  American Express  2014
1     25   CSCO             Cisco  2014
2     85    DIS       Walt Disney  2014
3     41   MSFT         Microsoft  2014
4     78    WMT           Walmart  2014
```

How to `drop` a specific Row?

In [20]:
```python
data.drop(1)
```
Out[20]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014

How to do a `transpose` of a dataframe?

In [21]:
```python
dft = data.T #Transpose operation will interchange the rows and columns
dft
```
Out[21]:

	0	1	2	3	4
price	95	25	85	41	78
ticker	AXP	CSCO	DIS	MSFT	WMT
company	American Express	Cisco	Walt Disney	Microsoft	Walmart
Year	2014	2014	2014	2014	2014

Indexing

    loc gets rows (and/or columns) with particular labels.

    iloc gets rows (and/or columns) at integer locations.

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[row_label, column_label]`

`iloc[row_position, column_position]`

In [22]:
```python
s = pd.Series(list("abcdef"), index=[49, 48, 47, 0, 1, 2])
s
```
Out[22]:
```
49    a
48    b
47    c
0     d
1     e
2     f
dtype: object
```

In [23]:
```python
s.loc[0]    # value at index label 0
```
Out[23]:
```
'd'
```

In [24]:
```python
s.iloc[0]   # value at index location 0
```
Out[24]:
```
'a'
```

In [25]:
```python
s.loc[0:1]  # rows at index labels between 0 and 1 (inclusive)
```
Out[25]:
```
0    d
1    e
dtype: object
```

In [26]:
```python
s.iloc[0:1] # rows at index location between 0 and 1 (exclusive)
```
Out[26]:
```
49    a
dtype: object
```

In [27]:
```python
s.iloc[-5:]
```
Out[27]:
```
48    b
47    c
0     d
1     e
2     f
dtype: object
```

In [28]:
```python
data
```
Out[28]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014

In [29]:
```python
df1= data.loc[:,'price']
df1
```
Out[29]:
```
0    95
1    25
2    85
3    41
4    78
Name: price, dtype: int64
```

In [30]:
```python
# select first 2 rows
data.iloc[:2]
# or
data.iloc[:2,]
```
Out[30]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014

In [31]:
```python
# select 3rd to 5th rows
data.iloc[2:5]
# or
data.iloc[2:5,]
```
Out[31]:

	price	ticker	company	Year
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014

Indexing with iloc `Index Based Location`

Select column by using column number in pandas with `.iloc`.

In [32]:
```python
# select first 2 columns
data.iloc[:,:2]
```
Out[32]:

	price	ticker
0	95	AXP
1	25	CSCO
2	85	DIS
3	41	MSFT
4	78	WMT

In [33]:
```python
# select 1st and 4thcolumn
data.iloc[:,[0,3]]
```
Out[33]:

	price	Year
0	95	2014
1	25	2014
2	85	2014
3	41	2014
4	78	2014

indexing with `loc` : `Label Based Indexing`

In [34]:
```python
# select row by row name
data.loc[1]
```
Out[34]:
```
price         25
ticker      CSCO
company    Cisco
Year        2014
Name: 1, dtype: object
```

In [35]:
```python
data
```
Out[35]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014

In [37]:
```python
data.loc[len(data.index)] = [95,'ML','Machine Learning', 2020]
```

In [38]:
```python
data
```
Out[38]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014
5	95	ML	Machine Learning	2020

How to `reindex` the data?

In [39]:
```python
data
```
Out[39]:

	price	ticker	company	Year
0	95	AXP	American Express	2014
1	25	CSCO	Cisco	2014
2	85	DIS	Walt Disney	2014
3	41	MSFT	Microsoft	2014
4	78	WMT	Walmart	2014
5	95	ML	Machine Learning	2020

In [40]:
```python
new_data = data.reindex(index=[0,2], columns=['company', 'price'])
new_data
```
Out[40]:

	company	price
0	American Express	95
2	Walt Disney	85

In [41]:
```python
#Note: reindex with only row arguments i.e we want row 88, 89 etc from above df2
reindexdf2 = data.reindex([88,3,89,90,91,92,93,94,95,96,97,98], fill_value=0) #Interpolation of new df with the
reindexdf2
```
Out[41]:

	price	ticker	company	Year
88	0	0	0	0
3	41	MSFT	Microsoft	2014
89	0	0	0	0
90	0	0	0	0
91	0	0	0	0
92	0	0	0	0
93	0	0	0	0
94	0	0	0	0
95	0	0	0	0
96	0	0	0	0
97	0	0	0	0
98	0	0	0	0

The reason we have zeros is due to the fact that row 88, 89, 86,97 and 97 are not present in our original data frame df2

`Drop` the `duplicate row` of a dataframe

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

#Create a DataFrame
d = {
    'Name':['Alisa','Bobby','jodha','jack','raghu','Cathrine',
            'Alisa','Bobby','kumar','Alisa','Alex','Cathrine'],
    'Age':[26,24,23,22,23,24,26,24,22,23,24,24],

    'Score':[85,63,55,74,31,77,85,63,42,62,89,77]}

df = pd.DataFrame(d,columns=['Name','Age','Score'])
df
```
Out[42]:

	Name	Age	Score
0	Alisa	26	85
1	Bobby	24	63
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
6	Alisa	26	85
7	Bobby	24	63
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89
11	Cathrine	24	77

Drop the `duplicate rows`:

In [43]:
```python
df.drop_duplicates()
```
Out[43]:

	Name	Age	Score
0	Alisa	26	85
1	Bobby	24	63
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89

`Drop` the duplicate by `retaining last occurrence`:

In [44]:
```python
df.drop_duplicates(keep='last')
```
Out[44]:

	Name	Age	Score
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
6	Alisa	26	85
7	Bobby	24	63
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89
11	Cathrine	24	77

`Drop` the duplicate by `column`:

In [45]:
```python
df.drop_duplicates(['Name'], keep='last')
```
Out[45]:

	Name	Age	Score
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
7	Bobby	24	63
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89
11	Cathrine	24	77

Simply `drop a row` or observation:

In [46]:
```python
df.drop([1,2])
```
Out[46]:

	Name	Age	Score
0	Alisa	26	85
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
6	Alisa	26	85
7	Bobby	24	63
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89
11	Cathrine	24	77

Drop a row or observation by condition:

In [47]:
```python
df[df.Name != 'Alisa']
```
Out[47]:

	Name	Age	Score
1	Bobby	24	63
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
7	Bobby	24	63
8	kumar	22	42
10	Alex	24	89
11	Cathrine	24	77

The above code takes up all the names except Alisa, thereby dropping the row with name 'Alisa'.

Drop a row or observation by `index`:

In [48]:
```python
df.drop(df.index[2])
```
Out[48]:

	Name	Age	Score
0	Alisa	26	85
1	Bobby	24	63
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
6	Alisa	26	85
7	Bobby	24	63
8	kumar	22	42
9	Alisa	23	62
10	Alex	24	89
11	Cathrine	24	77

Drop the row by `position`:

In [49]:
```python
# Drop bottom 3 rows
df[:-3]
```
Out[49]:

	Name	Age	Score
0	Alisa	26	85
1	Bobby	24	63
2	jodha	23	55
3	jack	22	74
4	raghu	23	31
5	Cathrine	24	77
6	Alisa	26	85
7	Bobby	24	63
8	kumar	22	42

Drop a `column by name`:

In [50]:
```python
# drop a column based on name

df.drop('Age',axis=1)
```
Out[50]:

	Name	Score
0	Alisa	85
1	Bobby	63
2	jodha	55
3	jack	74
4	raghu	31
5	Cathrine	77
6	Alisa	85
7	Bobby	63
8	kumar	42
9	Alisa	62
10	Alex	89
11	Cathrine	77

Drop a column based on `column index`:

In [51]:
```python
# drop a column based on column index

df.drop(df.columns[2],axis=1)
```
Out[51]:

	Name	Age
0	Alisa	26
1	Bobby	24
2	jodha	23
3	jack	22
4	raghu	23
5	Cathrine	24
6	Alisa	26
7	Bobby	24
8	kumar	22
9	Alisa	23
10	Alex	24
11	Cathrine	24

`Delete` a column based on `column name`:

In [52]:
```python
# delete a column

del df['Age']
df
```
Out[52]:

	Name	Score
0	Alisa	85
1	Bobby	63
2	jodha	55
3	jack	74
4	raghu	31
5	Cathrine	77
6	Alisa	85
7	Bobby	63
8	kumar	42
9	Alisa	62
10	Alex	89
11	Cathrine	77

`sort` a dataframe in python

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

#Create a Dictionary of series
d = {'Name':pd.Series(['Alisa','Bobby','Cathrine','Madonna','Rocky','Sebastian','Jaqluine',
   'Rahul','David','Andrew','Ajay','Teresa']),
   'Age':pd.Series([26,27,25,24,31,27,25,33,42,32,51,47]),
   'Score':pd.Series([89,87,67,55,47,72,76,79,44,92,99,69])}

#Create a DataFrame
df2 = pd.DataFrame(d)
print (df2)
```

```
    Name  Age  Score
0       Alisa   26     89
1       Bobby   27     87
2    Cathrine   25     67
3     Madonna   24     55
4       Rocky   31     47
5   Sebastian   27     72
6    Jaqluine   25     76
7       Rahul   33     79
8       David   42     44
9      Andrew   32     92
10       Ajay   51     99
11     Teresa   47     69
```

`Sort` the python pandas Dataframe by single column – `Ascending order`:

In [54]:
```
# sort the pandas dataframe by ascending value of single column

df2.sort_values(by='Score')
```

Out[54]:

	Name	Age	Score
8	David	42	44
4	Rocky	31	47
3	Madonna	24	55
2	Cathrine	25	67
11	Teresa	47	69
5	Sebastian	27	72
6	Jaqluine	25	76
7	Rahul	33	79
1	Bobby	27	87
0	Alisa	26	89
9	Andrew	32	92
10	Ajay	51	99

`Sort` a Dataframe in python pandas by single Column – `descending order`.

In [55]:
```
df2.sort_values(by='Score',ascending=0)
```

Out[55]:

	Name	Age	Score
10	Ajay	51	99
9	Andrew	32	92
0	Alisa	26	89
1	Bobby	27	87
7	Rahul	33	79
6	Jaqluine	25	76
5	Sebastian	27	72
11	Teresa	47	69
2	Cathrine	25	67
3	Madonna	24	55
4	Rocky	31	47
8	David	42	44

`Sort` the pandas Dataframe by Multiple Columns

In [56]:
```
# sort the pandas dataframe by multiple columns

df2.sort_values(by=['Age', 'Score'],ascending=[True,False])
```

Out[56]:

	Name	Age	Score
3	Madonna	24	55
6	Jaqluine	25	76
2	Cathrine	25	67
0	Alisa	26	89
1	Bobby	27	87
5	Sebastian	27	72
4	Rocky	31	47
9	Andrew	32	92
7	Rahul	33	79
8	David	42	44
11	Teresa	47	69
10	Ajay	51	99

In [57]:
```
df2.sort_values(by=['Age', 'Score'],ascending=[True,True])
```

Out[57]:

	Name	Age	Score
3	Madonna	24	55
2	Cathrine	25	67
6	Jaqluine	25	76
0	Alisa	26	89
5	Sebastian	27	72
1	Bobby	27	87
4	Rocky	31	47
9	Andrew	32	92
7	Rahul	33	79
8	David	42	44
11	Teresa	47	69
10	Ajay	51	99

Sort the dataframe in python pandas by index in ascending order:

In [58]:
```
df3=df2.sort_index()
df3
```

Out[58]:

	Name	Age	Score
0	Alisa	26	89
1	Bobby	27	87
2	Cathrine	25	67
3	Madonna	24	55
4	Rocky	31	47
5	Sebastian	27	72
6	Jaqluine	25	76
7	Rahul	33	79
8	David	42	44
9	Andrew	32	92
10	Ajay	51	99
11	Teresa	47	69

* HAPPY LEARNINIG **

In [ ]: