Pandas Tutorial Practice

Pandas Introduction

What is Pandas?

** Pandas is a python library used for working with data sets.

** It has functions for analyzing, cleaning, exploring and manipulating data.

** The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by wes Mckinney in 2008

** It is a high performance Data analysis Tool.

** It is used to working with large data set in efficient way.

** It supports or It can load files with different formats.

** More flexible to use.

** It represents in tabular way (rows and columns).

** It is more efficient to work with missing data.

** Pandas used for Indexing _ slicing - subsetting the large data.

** We can merge or join two different datasets easily.

** Can reshape datasets.

Pandas will use three Data structures:

1. Series - The data is represented in one dimensional array. i.e., list,tuple
2. Dataframe - Two dimensional - list/dictionary/series/another Dataframe
3. Panel - Multi dimensional - syntax: data, Column major axis, Rows minor axis

** Pandas is used for data analysis.

** Data is common for all companies such as Twitter, Facebook, Instagram, Social media or Organization.

** Pandas is the library of Python.

** Pandas has derived it's name from panel DataSystem.

** Pandas author is wis mackenny.

** In Two types, We can analyze the data:

i. Series
ii. DataFrame

** Pandas library is a open source library.

** BSD library Ber Software Distribution.It is a license to provide software to utilize the users.

** NumPy & Pandas are used to run in Jupyter Notebook or IDLE.

** The name "Pandas" has a reference to both "panel data", and "Python data analysis" and was created by wes mckinney in 2008.

** Pandas is a python library used for working with data sets.

** It has functions for analyzing, cleaning, exploring, and manipulating data.

** Read & write data structures and different formats csv, JSON, Zip, txt, XML,etc.

pandas is a Python Data Analysis Library.

Pandas is an open source, BSD-licensed library providing high_ performance, easy - to - use data structures and data Analysis tools for the Python Programming language.

We are proud to announce that pandas has become a sponsored projectof the (NUMFOCUS organization). This will help ensure the success of development of pandas as a world - class open - source project.

Pandas Data Structure:

1. Data Structure is a way to store data, retrive data, to manipulate data.
2. Data Structure is a specific way of string and organising the data in a computer to perform specific task or purpose.

    i. Series

    ii. Dataframe

    iii. Panel

--> Series is a one dimensional array.

--> One - Dimensional labeled arrays pd.Series(data).

--> Dataframe is two dimensional Array.

--> Two - Dimensional data structure with columns, much like a table.

--> A panel is a 3D container of data.

The Pandas provides three data structures for processing the data:

1. Series
2. DataFrame
3. Panel

Importance of Pandas in Python

1. Pandas allows us to analyze big data and make conclusions based on statistical theories.
2. Pandas can clean messy data sets, and make them readable and relevant.
3. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
4. Size Mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects.
5. Data set merging and joining flexible reshaping and pivoting of data sets provides time-series functionality.

Data Analysis:

Data analysis is a process of inspecting, cleaning,transforming, and modelling data with the goal of discovering useful information, informing conclusions, and supporting decision - making.

Python Libraries for Data Analysis

1. NumPy
2. SciPy
3. Statsmodel
4. Scikit learn

5. Matplotlib
6. Pandas
7. Seaborn

Why use Pandas?

1. Pandas allows us to analyze big data and make conclusions based on statistical theories.
2. Pandas can clean messy datasets and make them readable and relevant.
3. Relevant data is very important in data science.

What can Pandas do?

1. Pandas gives you answers about the data.Like:
2. Is there a correlation between two or more columns.
3. what is average value?
4. Max value? Min value? Pandas are also able to delete rows that are not relevant or contains wrong values, like empty or NULL values. This is called cleaning the data.

How to import Pandas?

There are two ways to import pandas.

Import pandas: This will import the entire pandas module.

```
or
```

From Pandas import * : This will import all class, objects, Variables etc. from pandas package.

```
here * means all.
```

Difference between Series and Dataframe?

Series:

1. One-Dimensional Array
2. Homogeneous
3. Same Datatypes or elements
4. value mutuble
5. size-immutable

DataFrame:

1. Two-Dimensional Array
2. Heterogeneous
3. different Datatypes or elements
4. value Mutable
5. size-Mutable

Series Data Structure:

Series:

1. Series is an important Data Structure of Pandas.
2. It represents one dimensional array.
3. Series type object has two main components:

   i. Data

   ii. Index
4. It is defined as a one- dimensional array that is capable of storing various data types.

What is Series?

1. A Pandas Series is like a column in a table.
2. It is a one-dimensional array holding data of any type.

Syntax:

import pandas as pd

pd.Series(data,index)

pd.Dataframe(data) ---> Most Efficient data structure

pd.Panel(data,major axis, minor axis, dtype)

What is Lables?

** If nothing else is Specified, the values are labled with their index number.

** first value has index(), second value has index 1 etc.

** This lable can be used to access a specified value.

Return the index values of the Series:

In [2]:
```
import pandas as pd
a11 = [1,2,4,8,5,9]
b11 = pd.Series(a11)
print(b11)
```

```
0    1
1    2
2    4
3    8
4    5
5    9
dtype: int64
```

How to create lables?

In [5]:
```
import pandas as pd
c11 = [45,89,63,24,51]
d11 = pd.SEries(c11,index=["a","b","c","d"])
```

```
a    45
b    89
c    63
d    24
e    51
dtype: int64
```

How to access values in Lables?

When you have created lables,you can access an item by referring to the label.

In [6]:
```
print(d11["b"])
```

```
89
```

In [8]:
```
print(d11["a"])
```

```
45
```

What is key/value objects as Series?

1. We can also use a key/value object, like a dictionary,when creating a Series.

Note: The keys of the dictionary become the lables.

Create a simple pandas series from a dictionary

In [10]:
```
import pandas as pd
Marks = {"English":80,"Maths":95,"Telugu":78,"Social":86,"Science":69}
S11 = pd.Series(Marks)
print(S11)
```

```
English    80
Maths      95
Telugu     78
Social     86
Science    69
dtype: int64
```

How to include Specific Items in Series?

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

Create a Series using only data from "day1" and "day2":

In [16]:
```
import pandas as pd
Marks1 = {"English":80,"Maths":95,"Telugu":78,"Social":86,"Science":69}
S12 = pd.Series(Marks,index=("English","Telugu","Science"))
print(S12)
```

```
English    80
Telugu     78
Science    69
dtype: int64
```

Difference between NumPy array and Series Object?

1. Vector operation can be performed only when 2-D array shapes are same.Otherwise we can not perform the vector operation.
2. Indexes are always numerics stats from 0 in Ndarray.

   In Series we can generate indexes as our requirement.

Code:

In [2]:
```
import pandas as pd
l = [13,26,93,52,67]
s = pd.Series(l)
print(s)
```

```
0    13
1    26
2    93
3    52
4    67
dtype: int64
```

In [4]:
```
import pandas as pd
l1 = [13,26,93,52,67]
s1 = pd.Series(l,index = ["i","ii","iii","iv","v"])
print(s1)
```

```
i      13
ii     26
iii    93
iv     52
v      67
dtype: int64
```

How to create Series in pandas & different ways to create?

1. Empty Series
2. Series Using Array
3. Series Using Lists
4. Series Using Dictionary

In [21]:
```
import numpy as np
import pandas as pd
s = pd.Series()
print(s)
```

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

C:\Users\mouni\AppData\Local\Temp\ipykernel_22456\2009084833.py:3: FutureWarning: The default dtype
for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicit
ly to silence this warning.
  s = pd.Series()
```

In [6]:
```
import numpy as np
import pandas as pd
a = np.array([2,4,6,8])
s = pd.Series(a)
print(s)
```

```
0    2
1    4
2    6
3    8
dtype: int32
```

In [19]:
```
import numpy as np
import pandas as pd
a1 = np.array([2,4,6,8])
s1 = pd.Series(a1,index = ["a","b","c","d"])
print(s1)
```

```
a    2
b    4
c    6
d    8
dtype: int32
```

In [24]:
```
import numpy as np
import pandas as pd
l1 =[23,45,67,89]
s2 = pd.Series(l1)
print(s2)
```

```
0    23
1    45
2    67
3    89
dtype: int64
```

In [25]:
```
import numpy as np
import pandas as pd
l3 =[23,45,67,89]
s3= pd.Series(l3,index = ["a","b","c","d"])
print(s3)
```

```
a    23
b    45
c    67
d    89
dtype: int64
```

In [30]:
```
import numpy as np
import pandas as pd
d1 = {"a":10,"b":20,"c":30,"d":40}
s4= pd.Series(d1)
print(s4)
```

```
a    10
b    20
c    30
d    40
dtype: int64
```

In [32]:
```
import numpy as np
import pandas as pd
d2 = {"a":10,"b":20,"c":30,"d":40}
s5= pd.Series(d2,index = ["i","ii","iii","iv"])
print(s5)
```

```
i     NaN
ii    NaN
iii   NaN
iv    NaN
dtype: float64
```

In [39]:
```
import numpy as np
import pandas as pd
d3 = [("a",10),("b",20),("c",30)]
print(d3)
s6 = pd.Series(d3)
print(s6)
```

```
[('a', 10), ('b', 20), ('c', 30)]
0    (a, 10)
1    (b, 20)
2    (c, 30)
dtype: object
```

Creation of Series Object:

1. List
2. Tuple
3. String
4. Dictionary

Syntax:

Series Object = Pandas.Series()

In [1]:
```
import pandas as pd
a = pd.Series()
print(a)
```

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

C:\Users\mouni\AppData\Local\Temp\ipykernel_32248\2114686964.py:2: FutureWarning: The default dtype
for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicit
ly to silence this warning.
  a = pd.Series()
```

Syntax:

Series Object = pd.Series(data,index=ind)

data --> list,tuple,string,dictionary array,etc.,

In [3]:
```
import pandas as pd
b = pd.Series([1,3,5,7,9])
print(b)
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```

In [4]:
```
import pandas as pd
b = pd.Series([1,3,5,7,9],index = [1,2,3,4,5])
print(b)
```

```
1    1
2    3
3    5
4    7
5    9
dtype: int64
```

In [13]:
```
import pandas as pd
b1 = pd.Series(["A","P","Y","m",1])
print(b1)
```

```
0    A
1    P
2    Y
3    m
4    1
dtype: object
```

In [ ]:
```

```

In [5]:
```
import pandas as pd
c = pd.Series(range(5))
print(c)
```

```
0    0
1    1
2    2
3    3
4    4
dtype: int64
```

In [7]:
```
import pandas as pd
d = pd.Series(range(5), index = ["i","ii","iii","iv","v"])
print(d)
```

```
i      0
ii     1
iii    2
iv     3
v      4
dtype: int64
```

In [8]:
```
import pandas as pd
e = pd.Series((5,6,8,7,9,2))
print(e)
```

```
0    5
1    6
2    8
3    7
4    9
5    2
dtype: int64
```

In [14]:
```
import pandas as pd
f = pd.Series((5,6,8,7,9,2),index = ["a","b","c","d","e","f"])
print(f)
```

```
a    5
b    6
c    8
d    7
e    9
f    2
dtype: int64
```

In [15]:
```
import pandas as pd
g = pd.Series("I Like Python")
print(g)
```

```
0    I Like Python
dtype: object
```

In [17]:
```
import pandas as pd
h = pd.Series(["I", "Like", "Python"])
print(h)
```

```
0         I
1      Like
2    Python
dtype: object
```

In [18]:
```
import pandas as pd
i = pd.Series(["I", "Like", "Python"],index = [1,2,3])
print(i)
```

```
1         I
2      Like
3    Python
dtype: object
```

In [19]:
```
import pandas as pd
j = pd.Series({'Year':1,'Month':12,"Week":52,"Days":365})
print(j)
```

```
Year       1
Month     12
Week      52
Days     365
dtype: int64
```

In [1]:
```
import pandas as pd
k = pd.Series({'a':2,'Monday':1,"hour":60,"Day":"Sunday"})
print("Series object values are:")
print(k)
```

```
Series object values are:
a               2
Monday          1
hour           60
Day        Sunday
dtype: object
```

In [2]:
```
import pandas as pd
dic = {'a':2,'Monday':1,"hour":60,"Day":"Sunday"}
k = pd.Series(dic)
print("Series object values are:")
print(k)
```

```
Series object values are:
a               2
Monday          1
hour           60
Day        Sunday
dtype: object
```

In [32]:
```
import pandas as pd
di = {"name":['Pthon',"SQL","R","EXCEL"],"count":[2,4,6,8],'s.no':[1,2,3,4]}
di1 = pd.Series(di)
print(di1)
```

```
name     [Pthon, SQL, R, EXCEL]
count               [2, 4, 6, 8]
s.no                [1, 2, 3, 4]
dtype: object
```

In [28]:
```
import pandas as pd
x1 = [5,7,9,14,19,16]
x2 = pd.Series(x1,index=["y1","y2","y3","y4","y5","y6"],dtype = "float")
print(x2)
print(type(x2))
print(x2[4])
print(x2[1])
print(x2[0])
```

```
y1     5.0
y2     7.0
y3     9.0
y4    14.0
y5    19.0
y6    16.0
dtype: float64
<class 'pandas.core.series.Series'>
19.0
7.0
5.0
```

In [30]:
```
import pandas as pd
x3 = [5,7,9,14,19,16]
x4 = pd.Series(x1,index=["y1","y2","y3","y4","y5","y6"],dtype = "float",name="Python")
print(x4)
print(type(x4))
print(x4[4])
print(x4[1])
print(x4[0])
```

```
y1     5.0
y2     7.0
y3     9.0
y4    14.0
y5    19.0
y6    16.0
Name: Python, dtype: float64
<class 'pandas.core.series.Series'>
19.0
7.0
5.0
```

In [34]:
```
import pandas as pd
x5 = pd.Series(12,index = [1,4,8,12,16,18])
print(x5)
print(type(x5))
```

```
1     12
4     12
8     12
12    12
16    12
18    12
dtype: int64
<class 'pandas.core.series.Series'>
```

In [36]:
```
import pandas as pd
x6=pd.Series(15,index = ["a","b","c","d","e","f"])
x7=pd.Series(9,index = ["a","b","c","d","e","f"])
print(x6+x7)
```

```
a    24
b    24
c    24
d    24
e    24
f    24
dtype: int64
```

In [38]:
```
import pandas as pd
x8=pd.Series(15,index = ["a","b","c","d","e","f"])
x9=pd.Series(9,index = ["a","b","c","d"])
print(x8+x9)
```

```
a    24.0
b    24.0
c    24.0
d    24.0
e     NaN
f     NaN
dtype: float64
```

In [22]:
```
import pandas as pd
import numpy as np
arr = np.array([1,6,12,24,36])
l = pd.Series(arr)
print(l)
print(arr)
```

```
0     1
1     6
2    12
3    24
4    36
dtype: int32
[ 1  6 12 24 36]
```

Possible Error:

In [24]:
```
import pandas as pd
import numpy as np
arr1 = np.array([[1,6,12,24,36],[13,26,39,52,65]])
m = pd.Series(arr1)
print(m)
print(arr1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_32248\1089031007.py in <module>
      2 import numpy as np
      3 arr1 = np.array([[1,6,12,24,36],[13,26,39,52,65]])
----> 4 m = pd.Series(arr1)
      5 print(m)
      6 print(arr1)

C:\anaconda\lib\site-packages\pandas\core\series.py in __init__(self, data, index, dtype, name, cop
y, fastpath)
    449                     data = data.copy()
    450             else:
--> 451                 data = sanitize_array(data, index, dtype, copy)
    452
    453                 manager = get_option("mode.data_manager")

C:\anaconda\lib\site-packages\pandas\core\construction.py in sanitize_array(data, index, dtype, cop
y, raise_cast_failure, allow_2d)
    599                 subarr = maybe_infer_to_datetimelike(subarr)
    600
--> 601     subarr = _sanitize_ndim(subarr, data, dtype, index, allow_2d=allow_2d)
    602
    603     if isinstance(subarr, np.ndarray):

C:\anaconda\lib\site-packages\pandas\core\construction.py in _sanitize_ndim(result, data, dtype, ind
ex, allow_2d)
    650             if allow_2d:
    651                 return result
--> 652             raise ValueError("Data must be 1-dimensional")
    653         if is_object_dtype(dtype) and isinstance(dtype, ExtensionDtype):
    654             # i.e. PandasDtype("O")

ValueError: Data must be 1-dimensional
```

In [26]:
```
import pandas as pd
import numpy as np
arr2 = np.arange(2,13,2.6)
n = pd.Series(arr2)
print(n)
print(arr2)
```

```
0     2.0
1     4.6
2     7.2
3     9.8
4    12.4
dtype: float64
[ 2.   4.6  7.2  9.8 12.4]
```

In [28]:
```
import pandas as pd
import numpy as np
arr3 = np.linspace(1,15,9)
o = pd.Series(arr2)
print(o)
print(arr3)
```

```
0     2.0
1     4.6
2     7.2
3     9.8
4    12.4
dtype: float64
[ 1.    2.75  4.5   6.25  8.    9.75 11.5  13.25 15.  ]
```

In [31]:
```
import pandas as pd
import numpy as np
arr4 = np.linspace(1,9,5)
p = pd.Series(arr4)
print(p)
print(arr4)
```

```
0    1.0
1    3.0
2    5.0
3    7.0
4    9.0
dtype: float64
[1. 3. 5. 7. 9.]
```

In [32]:
```
import pandas as pd
import numpy as np
q = pd.Series(np.tile([1,2,3],3))
print(q)
```

```
0    1
1    2
2    3
3    1
4    2
5    3
6    1
7    2
8    3
dtype: int32
```

In [33]:
```
import pandas as pd
r = pd.Series(3,index = range(0,1))
print(r)
```

```
0    3
dtype: int64
```

In [35]:
```
import pandas as pd
s = pd.Series(3,index = range(0,5))
print(s)
```

```
0    3
1    3
2    3
3    3
4    3
dtype: int64
```

In [37]:
```
import pandas as pd
t = pd.Series(9,index = range(1,8,2))
print(t)
```

```
1    9
3    9
5    9
7    9
dtype: int64
```

In [38]:
```
import pandas as pd
u =pd.Series("Completed",index = ['python','SQL'])
print(u)
```

```
python    Completed
SQL       Completed
dtype: object
```

In [39]:
```
import pandas as pd
v =pd.Series(("Completed","Yet to start"),index = ['python','ML'])
print(v)
```

```
python       Completed
ML        Yet to start
dtype: object
```

In [40]:
```
import pandas as pd
w = pd.Series(6000,index =["November","December","January","February"])
print(w)
```

```
November    6000
December    6000
January     6000
February    6000
dtype: int64
```

In [42]:
```
import pandas as pd
x = pd.Series(150,index = range(2000,2019))
print(x)
```

```
2000    150
2001    150
2002    150
2003    150
2004    150
2005    150
2006    150
2007    150
2008    150
2009    150
2010    150
2011    150
2012    150
2013    150
2014    150
2015    150
2016    150
2017    150
2018    150
dtype: int64
```

In [43]:
```
import pandas as pd
y = pd.Series(150,index = range(2000,2019,3))
print(y)
```

```
2000    150
2003    150
2006    150
2009    150
2012    150
2015    150
2018    150
dtype: int64
```

Iterrows() and Iteritems() functions:

Python Pandas:

--> Until now, we have learnt about the two most popular data structures of python pandas data structure called Series and DataFrame.

--> Series data Structure is a one dimensional data structure whereas DataFrame is a 2D data structure.

--> Now we are moving towards more about DataFrame concepts.

--> Iterating over DataFrame

--> When we need to go through each element of a DataFrame,We can do it using two functions:

1. iterrows() - This method iterates over dataframe row wise.

2. iteritems() - This method iterates over dataframe column wise.

What is DataFrame?

1. Data set in pandas are usually multi-dimensional tables, calles Dataframes.
2. Series is like a column, a Dataframe is the whole table.

Create a DataFrame from two Series:

In [2]:
```
import pandas as pd
DATA = {"Names":["Ajith","Bhavana","Charitha","Davan"],"Roll_Num":[101,12,103,104]}
DATAFRAME = pd.DataFrame(DATA)
print(DATAFRAME)
```

```
      Names  Roll_Num
0     Ajith       101
1   Bhavana        12
2  Charitha       103
3     Davan       104
```

Different ways to creating DataFrame in Pandas:

1. load the data from excel file
2. load the data from csv file
3. create DF using dictionary
4. create DF using list of tuples

```
read-excel
read-csv
```

import pandas as pd df = pd.read-excel()

How to named Indexes?

With the index argument, you can name your own indexes.

Add a list of names to give each row a name:

In [9]:
```
import pandas as pd
DATA1 = {"Names":["Ajith","Bhavana","Charitha","Davan"],"Roll_Num":[101,12,103,104]}
DATAFRAME1 = pd.DataFrame(DATA1,index=[1,2,3,4])
print(DATAFRAME1)
```

```
      Names  Roll_Num
1     Ajith       101
2   Bhavana        12
3  Charitha       103
4     Davan       104
```

How to locate Named Indexes?

Use the named index in the loc attribute to return the specified row(s).

Return 2:

In [12]:
```
import pandas as pd
DATA2 = {"Names":["Ajith","Bhavana","Charitha","Davan"],"Roll_Num":[101,12,103,104]}
DATAFRAME2 = pd.DataFrame(DATA2,index=["row1","row2","row3","row4"])
print(DATAFRAME2)
```

```
         Names  Roll_Num
row1     Ajith       101
row2   Bhavana        12
row3  Charitha       103
row4     Davan       104
```

In [22]:
```
print(DATAFRAME2.loc["row1"])
```

```
Names       Ajith
Roll_Num      101
Name: row1, dtype: object
```

In [17]:
```
print(DATAFRAME2.loc[["row1","row2"]])
```

```
        Names  Roll_Num
row1    Ajith       101
row2  Bhavana        12
```

Attributes of Series in Pandas:

1. index -->Series.index - Return all index values
2. array -->Series.array - Return an array of values
3. values -->Series.values - Return values of series
4. name -->Series.name - Return name of series
5. shape -->Series.shape - Return the shape
6. ndim -->Series.ndim - Return the dimension of series
7. size -->Series.size - Returns the size of series
8. nbytes -->Series.nbytes - Returns the memory occupied by values
9. memory_usage() -->Series.memory_usage() - Returns memory occupied by both index and values.
10. empty -->SEries.empty - Returns True - if Series is empty

    False - if Series is not empty

Code:

In [40]:
```
import pandas as pd
S1 = pd.Series([25,35,45,65,75])
print(S1)
```

```
0    25
1    35
2    45
3    65
4    75
dtype: int64
```

In [41]:
```
import pandas as pd
S2 = pd.Series([25,35,45,65,75],name = "Numbers")
print(S2)
```

```
0    25
1    35
2    45
3    65
4    75
Name: Numbers, dtype: int64
```

In [42]:
```
import pandas as pd
S3 = pd.Series([22,33,44,55,66,77,88,99],index = ['a','b','c','d','e','f','g','h'])
print(S3)
```

```
a    22
b    33
c    44
d    55
e    66
f    77
g    88
h    99
dtype: int64
```

In [58]:
```
import pandas as pd
S4 = pd.Series([22,33,44,55,66,77,88,99],index = ['a','b','c','d','e','f','g','h'])
print(S4)
print(S4.index)
print(S4.array)
print(S4.values)
print(S4.dtype)
print(S4.shape)
print(S4.ndim)
print(S4.nbytes)
print(S4.size)
print(S4.memory_usage())
print(S4.memory_usage(index = False))
print(S4.name)
print(S4.empty)
```

```
a    22
b    33
c    44
d    55
e    66
f    77
g    88
h    99
dtype: int64
Index(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'], dtype='object')
<PandasArray>
[22, 33, 44, 55, 66, 77, 88, 99]
Length: 8, dtype: int64
[22 33 44 55 66 77 88 99]
int64
(8,)
1
64
8
128
64
None
False
```

Series Object Attributes Python Pandas:

In [3]:
```
import pandas as pd
a1 = [1,12,52,365]
b1 = ["Year","Months","Week","Days"]
c1 = pd.Series(data = a1, index = b1)
print(c1)
```

```
Year        1
Months     12
Week       52
Days      365
dtype: int64
```

In [ ]:
```
```

In [10]:
```
print(c1.index)
print(c1.values)
print(c1.dtype)
print(c1.size)
print(c1.dtype)
print(c1.shape)
print(c1.ndim)
print(c1.size)
print(c1.nbytes)
print(c1.empty)
```

```
Index(['Year', 'Months', 'Week', 'Days'], dtype='object')
[  1  12  52 365]
int64
4
int64
(4,)
1
4
32
False
```

Series object Indexing|Slicing|Manipulating Python Pandas:

In [12]:
```
import pandas as pd
a2 = pd.Series([1,3,4,6,8,9,7])
print(a2)
```

```
0    1
1    3
2    4
3    6
4    8
5    9
6    7
dtype: int64
```

In [13]:
```
print(a2[0])
```

```
1
```

In [14]:
```
print(a2[3])
```

```
6
```

In [15]:
```
print(a2[5])
```

```
9
```

Possible errors:

In [16]:
```
print(a2[7])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\anaconda\lib\site-packages\pandas\core\indexes\range.py in get_loc(self, key, method, tolerance)
    384                 try:
--> 385                     return self._range.index(new_key)
    386                 except ValueError as err:

ValueError: 7 is not in range

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8448\1935906415.py in <module>
----> 1 print(a2[7])

C:\anaconda\lib\site-packages\pandas\core\series.py in __getitem__(self, key)
    956
    957         elif key_is_scalar:
--> 958             return self._get_value(key)
    959
    960         if is_hashable(key):

C:\anaconda\lib\site-packages\pandas\core\series.py in _get_value(self, label, takeable)
   1067
   1068         # Similar to Index.get_value, but we do not fall back to positional
-> 1069         loc = self.index.get_loc(label)
   1070         return self.index._get_values_for_loc(self, loc, label)
   1071

C:\anaconda\lib\site-packages\pandas\core\indexes\range.py in get_loc(self, key, method, tolerance)
    385                     return self._range.index(new_key)
    386                 except ValueError as err:
--> 387                     raise KeyError(key) from err
    388             self._check_indexing_error(key)
    389             raise KeyError(key)

KeyError: 7
```

In [17]:
```
print(a2[0:4])
```

```
0    1
1    3
2    4
3    6
dtype: int64
```

In [18]:
```
print(a2[:])
```

```
0    1
1    3
2    4
3    6
4    8
5    9
6    7
dtype: int64
```

In [19]:
```
print(a2[:6])
```

```
0    1
1    3
2    4
3    6
4    8
5    9
dtype: int64
```

In [20]:
```
print(a2[1:])
```

```
1    3
2    4
3    6
4    8
5    9
6    7
dtype: int64
```

In [21]:
```
print(a2[3:6])
```

```
3    6
4    8
5    9
dtype: int64
```

In [ ]:
```

```

In [23]:
```
print(a2[1]==12)
```

```
False
```

In [27]:
```
a2[1]=12
print(a2)
```

```
0     1
1    12
2     4
3     6
4     8
5     9
6     7
dtype: int64
```

In [28]:
```
a2[5]=34
print(a2)
```

```
0     1
1    12
2     4
3     6
4     8
5    34
6     7
dtype: int64
```

In [29]:
```
a2[4]=42
print(a2)
```

```
0     1
1    12
2     4
3     6
4    42
5    34
6     7
dtype: int64
```

In [32]:
```
a2.index = ["a","b","c","d","e","f","g"]
print(a2)
```

```
a     1
b    12
c     4
d     6
e    42
f    34
g     7
dtype: int64
```

In [33]:
```
a2.index = ["i","ii","iii","iv","v","vi","vii"]
print(a2)
```

```
i       1
ii     12
iii     4
iv      6
v      42
vi     34
vii     7
dtype: int64
```

In [34]:
```
a2.head()
```

Out[34]:
```
i       1
ii     12
iii     4
iv      6
v      42
dtype: int64
```

In [35]:
```
a2.tail()
```

Out[35]:
```
iii     4
iv      6
v      42
vi     34
vii     7
dtype: int64
```

In [36]:
```
a2.head(2)
```

Out[36]:
```
i      1
ii    12
dtype: int64
```

In [37]:
```
a2.tail(2)
```

Out[37]:
```
vi     34
vii     7
dtype: int64
```

Vector Operations on Series Object

Vector operations means that if you want to apply a function or expression, then it is individually applied on each item of the object.

In [38]:
```
import pandas as pd
a3 = pd.Series([1,4,8,3,5,9])
print(a3)
```

```
0    1
1    4
2    8
3    3
4    5
5    9
dtype: int64
```

In [44]:
```
print(a3+9)
```

```
0    10
1    13
2    17
3    12
4    14
5    18
dtype: int64
```

In [43]:
```
print(a3*4)
```

```
0     4
1    16
2    32
3    12
4    20
5    36
dtype: int64
```

In [41]:
```
print(a3-2)
```

```
0   -1
1    2
2    6
3    1
4    3
5    7
dtype: int64
```

In [42]:
```
print(a3**3)
```

```
0      1
1     64
2    512
3     27
4    125
5    729
dtype: int64
```

Arithmetic Operations on Series object:

In [50]:
```
import pandas as pd
a4 = pd.Series([1,3,7,9,12,19])
print(a4)
```

```
0     1
1     3
2     7
3     9
4    12
5    19
dtype: int64
```

In [49]:
```
import pandas as pd
a5 = pd.Series([2,4,6,8,15])
print(a5)
```

```
0     2
1     4
2     6
3     8
4    15
dtype: int64
```

In [51]:
```
import pandas as pd
a6 = pd.Series([2,3,6,9])
print(a6)
```

```
0    2
1    3
2    6
3    9
dtype: int64
```

In [52]:
```
a4+a5
```

Out[52]:
```
0     3.0
1     7.0
2    13.0
3    17.0
4    27.0
5     NaN
dtype: float64
```

In [53]:
```
a5+a6
```

Out[53]:
```
0     4.0
1     7.0
2    12.0
3    17.0
4     NaN
dtype: float64
```

In [54]:
```
a4+a6
```

Out[54]:
```
0     3.0
1     6.0
2    13.0
3    18.0
4     NaN
5     NaN
dtype: float64
```

In [55]:
```
a4*a3
```

Out[55]:
```
0      1
1     12
2     56
3     27
4     60
5    171
dtype: int64
```

In [56]:
```
a5-a4
```

Out[56]:
```
0    1.0
1    1.0
2   -1.0
3   -1.0
4    3.0
5    NaN
dtype: float64
```

In [57]:
```
a4/a6
```

Out[57]:
```
0    0.500000
1    1.000000
2    1.166667
3    1.000000
4         NaN
5         NaN
dtype: float64
```

In [58]:
```
a5%a6
```

Out[58]:
```
0    0.0
1    1.0
2    0.0
3    8.0
4    NaN
dtype: float64
```

Mathematical functions on series in pandas:

In [67]:
```
S5 = pd.Series([56,78,95,53,42])
S6 = pd.Series([13,62,94,32,84])
add = S5.add(S6)
print(add)
subtract = S5.subtract(S6)
print(subtract)
multiply = S5.multiply(S6)
print(multiply)
division = S5.div(S6)
print(division)
modulo = S5.mod(S6)
print(modulo)
power = S5.pow(S6)
print(power)
lessthan = S5.le(S6)
print(lessthan)
greaterthan = S5.gt(S6)
print(greaterthan)
equalto = S5.eq(S6)
print(equalto)
```

```
0     69
1    140
2    189
3     85
4    126
dtype: int64
0    43
1    16
2     1
3    21
4   -42
dtype: int64
0     728
1    4836
2    8930
3    1696
4    3528
dtype: int64
0    4.307692
1    1.258065
2    1.010638
3    1.656250
4    0.500000
dtype: float64
0     4
1    16
2     1
3    21
4    42
dtype: int64
0   -8900111769997934592
1    4611686018427387904
2   -4504589942734944063
3    -464456653302420095
4                      0
dtype: int64
0    False
1    False
2    False
3    False
4     True
dtype: bool
0     True
1     True
2     True
3     True
4    False
dtype: bool
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

In [72]:
```
print(S5+S6)
print(S5-S6)
print(S5*S6)
print(S5/S6)
print(S5%S6)
print(S5**S6)
print(S5<S6)
print(S5>S6)
print(S5==S6)
```

```
0     69
1    140
2    189
3     85
4    126
dtype: int64
0    43
1    16
2     1
3    21
4   -42
dtype: int64
0     728
1    4836
2    8930
3    1696
4    3528
dtype: int64
0    4.307692
1    1.258065
2    1.010638
3    1.656250
4    0.500000
dtype: float64
0     4
1    16
2     1
3    21
4    42
dtype: int64
0   -8900111769997934592
1    4611686018427387904
2   -4504589942734944063
3    -464456653302420095
4                       0
dtype: int64
0    False
1    False
2    False
3    False
4     True
dtype: bool
0     True
1     True
2     True
3     True
4    False
dtype: bool
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

In [1]:
```
import pandas as pd
a6 = pd.Series(data=[12,15,18,19,21])
print(a6)
print(a6>80)
print(a6[a6>60])
```

```
0    12
1    15
2    18
3    19
4    21
dtype: int64
0    False
1    False
2    False
3    False
4    False
dtype: bool
Series([], dtype: int64)
```

In [3]:
```
import pandas as pd
a7 = pd.Series(data=[12,15,18,19,21,78,59,84,52,34,85,73,61,83,96])
print(a7)
print(a7>50)
print(a7[a7>60])
```

```
0     12
1     15
2     18
3     19
4     21
5     78
6     59
7     84
8     52
9     34
10    85
11    73
12    61
13    83
14    96
dtype: int64
0     False
1     False
2     False
3     False
4     False
5      True
6      True
7      True
8      True
9     False
10     True
11     True
12     True
13     True
14     True
dtype: bool
5     78
7     84
10    85
11    73
12    61
13    83
14    96
dtype: int64
```

sort():

In [4]:
```
a7.sort_values()
```

Out[4]:
```
0     12
1     15
2     18
3     19
4     21
9     34
8     52
6     59
12    61
11    73
5     78
13    83
7     84
10    85
14    96
dtype: int64
```

In [5]:
```
a7.sort_values(ascending=False)
```

Out[5]:
```
14    96
10    85
7     84
13    83
5     78
11    73
12    61
6     59
8     52
9     34
4     21
3     19
2     18
1     15
0     12
dtype: int64
```

sort_index():

In [6]:
```
a7.sort_index()
```

Out[6]:
```
0     12
1     15
2     18
3     19
4     21
5     78
6     59
7     84
8     52
9     34
10    85
11    73
12    61
13    83
14    96
dtype: int64
```

In [7]:
```
a7.sort_index(ascending=False)
```

Out[7]:
```
14    96
13    83
12    61
11    73
10    85
9     34
8     52
7     84
6     59
5     78
4     21
3     19
2     18
1     15
0     12
dtype: int64
```

Creating DataFrame Objects:Python Pandas

Data Structure:

1. Series --> 1-d array

2. DataFrame --> 2-D array

Creating Dataframe object using 2-D Dictionary:

In [11]:
```
import pandas as pd
d3 = {'Employees':["Tommy","John","Josh","Akash","Mahitha"],'Salary':[100000,50000,60000,30000,90000]
print(d3)
print()
df3 = pd.DataFrame(d3)
print(df3)
```

```
{'Employees': ['Tommy', 'John', 'Josh', 'Akash', 'Mahitha'], 'Salary': [100000, 50000, 60000, 30000,
90000], 'Performance': ['Good', 'Average', 'Poor', 'Excellent', 'Good']}

  Employees  Salary Performance
0     Tommy  100000        Good
1      John   50000     Average
2      Josh   60000        Poor
3     Akash   30000   Excellent
4   Mahitha   90000        Good
```

In [13]:
```
df4 = pd.DataFrame(d3, index=["i","ii","iii","iv","v"])
print(df4)
```

```
    Employees  Salary Performance
i       Tommy  100000        Good
ii       John   50000     Average
iii      Josh   60000        Poor
iv      Akash   30000   Excellent
v     Mahitha   90000        Good
```

In [14]:
```
import pandas as pd
d5 = {'Country':["India","Bangladesh","Pakistan"],'Code':[123,146,645]}
df5 = pd.DataFrame(d5)
print(df5)
```

```
      Country  Code
0       India   123
1  Bangladesh   146
2   Pakistan   645
```

Creation of DataFrame object from a 2D dictionary having values on dictionnary object

In [15]:
```
import pandas as pd
d6 = {'Student':{'name':'Mohit','age':29,'Sex':'Male'},'Employee':{'name':'John','age':35,'Sex':'Male
df6 = pd.DataFrame(d6)
print(df6)
```

```
     Student Employee
name   Mohit     John
age       29       35
Sex     Male     Male
```

In [16]:
```
import pandas as pd
d7 = {'Student':{'name':'Mohit','age':29,'Sex':'Male'},'Employee':{'name':'John','age':35}}
df7 = pd.DataFrame(d7)
print(df7)
```

```
     Student Employee
name   Mohit     John
age       29       35
Sex     Male      NaN
```

In [17]:
```
import pandas as pd
d8 = {'Student':{'name':'Mohit','age':29,'Sex':'Male'},'Employee':{'name':'John','age':35,'Salary':10
df8 = pd.DataFrame(d8)
print(df8)
```

```
       Student Employee
name     Mohit     John
age         29       35
Sex       Male      NaN
Salary     NaN  1000000
```

Creating DataFrame object from List of Dictionary

In [1]:
```
import pandas as pd
d9 = {'Student name':'Mohit','age':29,'Sex':'Male'}
d10 = {'Student name':'Mohitha','age':25,'Sex':'Female'}
d11 = {'Student name':'Kavitha','age':25,'Sex':'Female'}
list1 = [d9,d10,d11]
df9 = pd.DataFrame(list1)
print(df9)
```

```
  Student name  age     Sex
0        Mohit   29    Male
1      Mohitha   25  Female
2      Kavitha   25  Female
```

In [2]:
```
import pandas as pd
d12 = {'Student name':'Mohit','age':29,'Sex':'Male'}
d13 = {'Student name':'Mohitha','age':25,'Sex':'Female'}
d14 = {'Student name':'Kavitha','age':25,'Sex':'Female'}
list2 = [d12,d13,d14]
df10 = pd.DataFrame(list2,index = [1,2,3])
print(df10)
```

```
  Student name  age     Sex
1        Mohit   29    Male
2      Mohitha   25  Female
3      Kavitha   25  Female
```

In [4]:
```
import pandas as pd
d15 = {'Student name':'Mohit','age':29,'Sex':'Male'}
d16 = {'Student name':'Mohitha','age':25,'Sex':'Female'}
d17 = {'Student name':'Kavitha','age':23,'Sex':'Female'}
list3 = [d15,d16,d17]
df11 = pd.DataFrame(list3,index = ["row1","row2","row3"],columns = ["col1","col2","col3"])
print(df11)
```

```
      col1  col2  col3
row1   NaN   NaN   NaN
row2   NaN   NaN   NaN
row3   NaN   NaN   NaN
```

In [5]:
```
import pandas as pd
list4 = [[2,4,6,8],[10,12,14,18],[1,3,5,7]]
df12 = pd.DataFrame(list4)
print(df12)
```

```
    0   1   2   3
0   2   4   6   8
1  10  12  14  18
2   1   3   5   7
```

In [6]:
```
import pandas as pd
list5 = [[2,4,6,8],[10,12,14,18],[1,3,5,7]]
df13 = pd.DataFrame(list5,index = ["row1","row2","row3"])
print(df13)
```

```
       0   1   2   3
row1   2   4   6   8
row2  10  12  14  18
row3   1   3   5   7
```

In [8]:
```
import pandas as pd
list6 = [[2,4,6,8],[10,12,14,18],[1,3,5,7]]
df14 = pd.DataFrame(list6,index = ["row1","row2","row3"],columns = ["col1","col2","col3","col4"])
print(df14)
```

```
      col1  col2  col3  col4
row1     2     4     6     8
row2    10    12    14    18
row3     1     3     5     7
```

Creating DataFrame Object from 2-D ndarray

In [10]:
```
import numpy as np
import pandas as pd
arr_1 = np.array([[12,23,35,64,89],[83,95,63,28,19]])
df15 = pd.DataFrame(arr_1)
print(df15)
```

```
    0   1   2   3   4
0  12  23  35  64  89
1  83  95  63  28  19
```

In [13]:
```
import numpy as np
import pandas as pd
arr_2 = np.array([[12,23,35,64,89],[83,95,63,28,19]])
df16 = pd.DataFrame(arr_2,columns = ['One','Two','Three','Four','Five'],index = ['First','Second'])
print(df16)
```

```
        One  Two  Three  Four  Five
First    12   23     35    64    89
Second   83   95     63    28    19
```

In [14]:
```
import numpy as np
import pandas as pd
arr_3 = np.array([[20,30,40],[50,60],[70,80,90,100]])
df17 = pd.DataFrame(arr_3)
print(df17)
```

```
                  0
0       [20, 30, 40]
1           [50, 60]
2  [70, 80, 90, 100]

C:\Users\mouni\AppData\Local\Temp\ipykernel_30448\2465195056.py:3: VisibleDeprecationWarning: Creati
ng an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays
with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=ob
ject' when creating the ndarray.
  arr_3 = np.array([[20,30,40],[50,60],[70,80,90,100]])
```

Creating DataFrame Object from a 2-D Dictionary with values on Series Object

Possible Error:

In [16]:
```
import pandas as pd
a2 = pd.Series([10,20,30])
b2 = pd.Series([100,200,300])
dict1 = {"Name":'Akhila',"Age":28,"Location":'Kolkata'}
df18 = pd.DataFrame(dict1)
print(df18)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_30448\3928661225.py in <module>
      3 b2 = pd.Series([100,200,300])
      4 dict1 = {"Name":'Akhila',"Age":28,"Location":'Kolkata'}
----> 5 df18 = pd.DataFrame(dict1)
      6 print(df18)

C:\anaconda\lib\site-packages\pandas\core\frame.py in __init__(self, data, index, columns, dtype, co
py)
    634         elif isinstance(data, dict):
    635             # GH#38939 de facto copy defaults to False only in non-dict cases
--> 636             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    637         elif isinstance(data, ma.MaskedArray):
    638             import numpy.ma.mrecords as mrecords

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in dict_to_mgr(data, index, colu
mns, dtype, typ, copy)
    500         # TODO: can we get rid of the dt64tz special case above?
    501
--> 502     return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, consolidate=copy)
    503
    504

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in arrays_to_mgr(arrays, column
s, index, dtype, verify_integrity, typ, consolidate)
    118         # figure out the index, if necessary
    119         if index is None:
--> 120             index = _extract_index(arrays)
    121         else:
    122             index = ensure_index(index)

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in _extract_index(data)
    662
    663         if not indexes and not raw_lengths:
--> 664             raise ValueError("If using all scalar values, you must pass an index")
    665
    666         elif have_series:

ValueError: If using all scalar values, you must pass an index
```

In [17]:
```
import pandas as pd
a2 = pd.Series([10,20,30])
b2 = pd.Series([100,200,300])
dict1 = {"Name":'Akhila',"Age":28,"Location":'Kolkata'}
df18 = pd.DataFrame(dict1,index=[1,2,3])
print(df18)
```

```
     Name  Age Location
1  Akhila   28  Kolkata
2  Akhila   28  Kolkata
3  Akhila   28  Kolkata
```

In [21]:
```
import pandas as pd
a3 = pd.Series([10,20,30])
b3 = pd.Series([100,200,300])
dict2 = {"Roll no":a3,"wage":b3}
df19 = pd.DataFrame(dict2,index=[1,2,3])
print(df19)
```

```
   Roll no   wage
1     20.0  200.0
2     30.0  300.0
3      NaN    NaN
```

In [22]:
```
import pandas as pd
a4= pd.Series([10,20,30])
b4 = pd.Series([100,200,300])
dict3 = {"Roll no":a3,"wage":b3}
df20 = pd.DataFrame(dict3)
print(df20)
```

```
   Roll no  wage
0       10   100
1       20   200
2       30   300
```

Creating DataFrame Object from another DataFrame Object

In [23]:
```
import pandas as pd
df21 = pd.DataFrame([[29,48,57,62],[83,51,95,74]])
df22 = pd.DataFrame(df21)
print(df22)
```

```
    0   1   2   3
0  29  48  57  62
1  83  51  95  74
```

In [25]:
```
import pandas as pd
df23 = pd.DataFrame([[29,48,57,62],[83,51,95,74]])
df24 = pd.DataFrame(df23,index=["1","2"],columns=["a","b","c"])
print(df24)
```

```
    a   b   c
1 NaN NaN NaN
2 NaN NaN NaN
```

In [ ]:
```

```

In [27]:
```
import pandas as pd
d25 = {'Student name':'Mohit','age':29,'Sex':'Male'}
d26 = {'Student name':'Mohitha','age':25,'Sex':'Female'}
d27 = {'Student name':'Kavitha','age':25,'Sex':'Female'}
list7 = [d25,d26,d27]
df25 = pd.DataFrame(list7)
print(df25)
```

```
  Student name  age     Sex
0        Mohit   29    Male
1      Mohitha   25  Female
2      Kavitha   25  Female
```

In [33]:
```
print(df25.index)
print(df25.columns)
print(df25.axes)
print(df25.dtypes)
print(df25.size)
print(df25.shape)
print(df25.ndim)
print(df25.empty)
print(len(df25))
print(df25.count())
print(df25.T)
print(df25.values)
```

```
RangeIndex(start=0, stop=3, step=1)
Index(['Student name', 'age', 'Sex'], dtype='object')
[RangeIndex(start=0, stop=3, step=1), Index(['Student name', 'age', 'Sex'], dtype='object')]
Student name    object
age              int64
Sex             object
dtype: object
9
(3, 3)
2
False
3
Student name    3
age             3
Sex             3
dtype: int64
                  0        1        2
Student name  Mohit  Mohitha  Kavitha
age              29       25       25
Sex            Male   Female   Female
[['Mohit' 29 'Male']
 ['Mohitha' 25 'Female']
 ['Kavitha' 25 'Female']]
```

In [39]:
```
import pandas as pd
Data1 = [2,4,6,8,10,12]
Dataframe1 = pd.DataFrame(Data1)
print(Dataframe1)
print(type(Dataframe1))
```

```
    0
0   2
1   4
2   6
3   8
4  10
5  12
<class 'pandas.core.frame.DataFrame'>
```

Possible Error:

In [41]:
```
import pandas as pd
Data2 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9]}
Dataframe2 = pd.DataFrame(Data2)
print(Dataframe2)
print(type(Dataframe2))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10920\2923933313.py in <module>
      1 import pandas as pd
      2 Data2 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9]}
----> 3 Dataframe2 = pd.DataFrame(Data2)
      4 print(Dataframe2)
      5 print(type(Dataframe2))

C:\anaconda\lib\site-packages\pandas\core\frame.py in __init__(self, data, index, columns, dtype, co
py)
    634         elif isinstance(data, dict):
    635             # GH#38939 de facto copy defaults to False only in non-dict cases
--> 636             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    637         elif isinstance(data, ma.MaskedArray):
    638             import numpy.ma.mrecords as mrecords

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in dict_to_mgr(data, index, colu
mns, dtype, typ, copy)
    500         # TODO: can we get rid of the dt64tz special case above?
    501
--> 502     return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, consolidate=copy)
    503
    504

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in arrays_to_mgr(arrays, column
s, index, dtype, verify_integrity, typ, consolidate)
    118         # figure out the index, if necessary
    119         if index is None:
--> 120             index = _extract_index(arrays)
    121         else:
    122             index = ensure_index(index)

C:\anaconda\lib\site-packages\pandas\core\internals\construction.py in _extract_index(data)
    672             lengths = list(set(raw_lengths))
    673             if len(lengths) > 1:
--> 674                 raise ValueError("All arrays must be of the same length")
    675
    676             if have_dicts:

ValueError: All arrays must be of the same length
```

In [44]:
```
import pandas as pd
Data3 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11]}
Dataframe3 = pd.DataFrame(Data3)
print(Dataframe3)
print(type(Dataframe3))
```

```
    A   B
0   2   1
1   4   3
2   6   5
3   8   7
4  10   9
5  12  11
<class 'pandas.core.frame.DataFrame'>
```

In [45]:
```
import pandas as pd
Data4 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11]}
Dataframe4 = pd.DataFrame(Data4,columns=["B"])
print(Dataframe4)
print(type(Dataframe4))
```

```
    B
0   1
1   3
2   5
3   7
4   9
5  11
<class 'pandas.core.frame.DataFrame'>
```

In [47]:
```
import pandas as pd
Data5 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11]}
Dataframe5 = pd.DataFrame(Data5,columns=["B","A","a"])
print(Dataframe5)
print(type(Dataframe5))
```

```
    B   A   a
0   1   2 NaN
1   3   4 NaN
2   5   6 NaN
3   7   8 NaN
4   9  10 NaN
5  11  12 NaN
<class 'pandas.core.frame.DataFrame'>
```

In [49]:
```
import pandas as pd
Data6 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11],9:[1,2,3,4,6,9]}
Dataframe6 = pd.DataFrame(Data6,columns=["B",9])
print(Dataframe6)
print(type(Dataframe6))
```

```
    B  9
0   1  1
1   3  2
2   5  3
3   7  4
4   9  6
5  11  9
<class 'pandas.core.frame.DataFrame'>
```

In [53]:
```
import pandas as pd
Data7 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11],9:[1,2,3,4,6,9]}
Dataframe7 = pd.DataFrame(Data6,columns=["B",9],index=[1,2,3,4,5,6])
print(Dataframe7)
print(type(Dataframe7))
print(Dataframe7["B"][4])
```

```
    B  9
1   1  1
2   3  2
3   5  3
4   7  4
5   9  6
6  11  9
<class 'pandas.core.frame.DataFrame'>
7
```

In [54]:
```
import pandas as pd
Data8 = {"A":[2,4,6,8,10,12],"B":[1,3,5,7,9,11],9:[1,2,3,4,6,9]}
Dataframe8 = pd.DataFrame(Data6,columns=["B",9],index=[1,2,3,4,5,6])
print(Dataframe8)
print(type(Dataframe8))
print(Dataframe8["B"],[4])
```

```
    B  9
1   1  1
2   3  2
3   5  3
4   7  4
5   9  6
6  11  9
<class 'pandas.core.frame.DataFrame'>
1     1
2     3
3     5
4     7
5     9
6    11
Name: B, dtype: int64 [4]
```

In [58]:
```
import pandas as pd
list1 = [[2,89,74,63,29],[91,28,67,84,65]]
Dataframe9 = pd.DataFrame(list1)
print(Dataframe9)
print(type(Dataframe9))
```

```
    0   1   2   3   4
0   2  89  74  63  29
1  91  28  67  84  65
<class 'pandas.core.frame.DataFrame'>
```

In [62]:
```
import pandas as pd
di1 = {"M":pd.Series([56,78,92,14]),"N":pd.Series([19,82,64,53])}
Dataframe10 = pd.DataFrame(di1)
print(Dataframe10)
print(type(Dataframe10))
```

```
    M   N
0  56  19
1  78  82
2  92  64
3  14  53
<class 'pandas.core.frame.DataFrame'>
```

loc and iloc function for DataFrame object:

In [1]:
```
import pandas as pd
dict4 = {"Roll_No":[1,2,3,4,5],"Name":["Ankitha","Bhavana","Charishma","Dhavan","Easther"],"Salary":[
df26 = pd.DataFrame(dict4)
print(df26)
```

```
   Roll_No       Name  Salary
0        1    Ankitha   20000
1        2    Bhavana   68000
2        3  Charishma   50000
3        4     Dhavan   45000
4        5    Easther   30000
```

In [3]:
```
import pandas as pd
dict5 = {'Student name':'Mohit','age':29,'Sex':'Male'}
dict6 = {'Student name':'Mohitha','age':25,'Sex':'Female'}
dict7 = {'Student name':'Kavitha','age':25,'Sex':'Female'}
list9 = [dict5,dict6,dict7]
df27 = pd.DataFrame(list9,index = [1,2,3])
print(df27)
```

```
  Student name  age     Sex
1        Mohit   29    Male
2      Mohitha   25  Female
3      Kavitha   25  Female
```

1.Selecting/Accessing DataFrame Columns:

In [7]:
```
df27['Student name']
```

Out[7]:
```
1      Mohit
2    Mohitha
3    Kavitha
Name: Student name, dtype: object
```

In [8]:
```
df27["Sex"]
```

Out[8]:
```
1      Male
2    Female
3    Female
Name: Sex, dtype: object
```

In [9]:
```
df27["age"]
```

Out[9]:
```
1    29
2    25
3    25
Name: age, dtype: int64
```

In [10]:
```
df27[["Student name","age"]]
```

Out[10]:

	Student name	age
1	Mohit	29
2	Mohitha	25
3	Kavitha	25

In [11]:
```
df27[["age","Sex"]]
```

Out[11]:

	age	Sex
1	29	Male
2	25	Female
3	25	Female

In [12]:
```
df27[["Student name","age","Sex"]]
```

Out[12]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

2.Selecting Rows/Columns:

Syntax:

dataframeobject.loc[start row:end row,start column:end column]

In [13]:
```
df27.loc[1,:]
```

Out[13]:
```
Student name    Mohit
age                29
Sex              Male
Name: 1, dtype: object
```

In [14]:
```
df27.loc[3,:]
```

Out[14]:
```
Student name    Kavitha
age                  25
Sex              Female
Name: 3, dtype: object
```

In [15]:
```
df27.loc[1:2:3]
```

Out[15]:

	Student name	age	Sex
1	Mohit	29	Male

In [16]:
```
df27.loc[2:3:]
```

Out[16]:

	Student name	age	Sex
2	Mohitha	25	Female
3	Kavitha	25	Female

In [17]:
```
df27.loc[1::2]
```

Out[17]:

	Student name	age	Sex
1	Mohit	29	Male
3	Kavitha	25	Female

In [19]:
```
df27.loc[1:2:]
```

Out[19]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female

In [20]:
```
df27.loc[2:3]
```

Out[20]:

	Student name	age	Sex
2	Mohitha	25	Female
3	Kavitha	25	Female

In [5]:
```
df27.loc[:,"Student name":"age"]
```

Out[5]:

	Student name	age
1	Mohit	29
2	Mohitha	25
3	Kavitha	25

In [6]:
```
df27.loc[1:3,'age':'Sex']
```

Out[6]:

	age	Sex
1	29	Male
2	25	Female
3	25	Female

Possible Errors:

In [4]:
```
df27.loc[1:2,1:3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8672\1866851460.py in <module>
----> 1 df27.loc[1:2,1:3]

C:\anaconda\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    959             if self._is_scalar_access(key):
    960                 return self.obj._get_value(*key, takeable=self._takeable)
--> 961             return self._getitem_tuple(key)
    962         else:
    963             # we by definition only have the 0th axis

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self, tup)
   1147             return self._multi_take(tup)
   1148
-> 1149         return self._getitem_tuple_same_dim(tup)
   1150
   1151     def _get_label(self, label, axis: int):

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_tuple_same_dim(self, tup)
    825                 continue
    826
--> 827             retval = getattr(retval, self.name)._getitem_axis(key, axis=i)
    828             # We should never have retval.ndim < self.ndim, as that should
    829             #  be handled by the _getitem_lowerdim call above.

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1181         if isinstance(key, slice):
   1182             self._validate_key(key, axis)
-> 1183             return self._get_slice_axis(key, axis=axis)
   1184         elif com.is_bool_indexer(key):
   1185             return self._getbool_axis(key, axis=axis)

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _get_slice_axis(self, slice_obj, axis)
   1215
   1216         labels = obj._get_axis(axis)
-> 1217         indexer = labels.slice_indexer(slice_obj.start, slice_obj.stop, slice_obj.step)
   1218
   1219         if isinstance(indexer, slice):

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in slice_indexer(self, start, end, step, k
ind)
   6286         self._deprecated_arg(kind, "kind", "slice_indexer")
   6287
-> 6288         start_slice, end_slice = self.slice_locs(start, end, step=step)
   6289
   6290         # return a slice

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in slice_locs(self, start, end, step, kin
d)
   6496         start_slice = None
   6497         if start is not None:
-> 6498             start_slice = self.get_slice_bound(start, "left")
   6499         if start_slice is None:
   6500             start_slice = 0

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in get_slice_bound(self, label, side, kin
d)
   6405         # For datetime indices label may be a string that has to be converted
   6406         # to datetime boundary according to its resolution.
-> 6407         label = self._maybe_cast_slice_bound(label, side)
   6408
   6409         # we need to look up the label

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in _maybe_cast_slice_bound(self, label, si
de, kind)
   6352         # reject them, if index does not contain label
   6353         if (is_float(label) or is_integer(label)) and label not in self:
-> 6354             raise self._invalid_indexer("slice", label)
   6355
   6356         return label

TypeError: cannot do slice indexing on Index with these indexers [1] of type int
```

In [21]:
```
df27.loc[:,0:1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14252\2037933868.py in <module>
----> 1 df27.loc[:,0:1]

C:\anaconda\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    959             if self._is_scalar_access(key):
    960                 return self.obj._get_value(*key, takeable=self._takeable)
--> 961             return self._getitem_tuple(key)
    962         else:
    963             # we by definition only have the 0th axis

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self, tup)
   1147             return self._multi_take(tup)
   1148
-> 1149         return self._getitem_tuple_same_dim(tup)
   1150
   1151     def _get_label(self, label, axis: int):

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_tuple_same_dim(self, tup)
    825                 continue
    826
--> 827             retval = getattr(retval, self.name)._getitem_axis(key, axis=i)
    828             # We should never have retval.ndim < self.ndim, as that should
    829             #  be handled by the _getitem_lowerdim call above.

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1181         if isinstance(key, slice):
   1182             self._validate_key(key, axis)
-> 1183             return self._get_slice_axis(key, axis=axis)
   1184         elif com.is_bool_indexer(key):
   1185             return self._getbool_axis(key, axis=axis)

C:\anaconda\lib\site-packages\pandas\core\indexing.py in _get_slice_axis(self, slice_obj, axis)
   1215
   1216         labels = obj._get_axis(axis)
-> 1217         indexer = labels.slice_indexer(slice_obj.start, slice_obj.stop, slice_obj.step)
   1218
   1219         if isinstance(indexer, slice):

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in slice_indexer(self, start, end, step, k
ind)
   6286         self._deprecated_arg(kind, "kind", "slice_indexer")
   6287
-> 6288         start_slice, end_slice = self.slice_locs(start, end, step=step)
   6289
   6290         # return a slice

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in slice_locs(self, start, end, step, kin
d)
   6496         start_slice = None
   6497         if start is not None:
-> 6498             start_slice = self.get_slice_bound(start, "left")
   6499         if start_slice is None:
   6500             start_slice = 0

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in get_slice_bound(self, label, side, kin
d)
   6405         # For datetime indices label may be a string that has to be converted
   6406         # to datetime boundary according to its resolution.
-> 6407         label = self._maybe_cast_slice_bound(label, side)
   6408
   6409         # we need to look up the label

C:\anaconda\lib\site-packages\pandas\core\indexes\base.py in _maybe_cast_slice_bound(self, label, si
de, kind)
   6352         # reject them, if index does not contain label
   6353         if (is_float(label) or is_integer(label)) and label not in self:
-> 6354             raise self._invalid_indexer("slice", label)
   6355
   6356         return label

TypeError: cannot do slice indexing on Index with these indexers [0] of type int
```

iloc-(Index)-->Syntax:

dataframeobject.iloc[start row index:end row index, start column index:end column index]

In [24]:
```
df27.iloc[0:2,1:2]
```

Out[24]:

	age
1	29
2	25

In [25]:
```
df27.iloc[1:2,]
```

Out[25]:

	Student name	age	Sex
2	Mohitha	25	Female

In [26]:
```
df27.iloc[:2,:3]
```

Out[26]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female

In [30]:
```
df27.iloc[:,1:2]
```

Out[30]:

	age
1	29
2	25
3	25

In [33]:
```
df27.iloc[1:,1:3]
```

Out[33]:

	age	Sex
2	25	Female
3	25	Female

In [34]:
```
df27.iloc[:,]
```

Out[34]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

In [35]:
```
df27.iloc[:]
```

Out[35]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

In [36]:
```
df27.iloc[::]
```

Out[36]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

In [37]:
```
df27.iloc[::,]
```

Out[37]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

In [38]:
```
df27.iloc[::,1:3]
```

Out[38]:

	age	Sex
1	29	Male
2	25	Female
3	25	Female

In [39]:
```
df27.iloc[:,0:3:]
```

Out[39]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female
3	Kavitha	25	Female

In [45]:
```
df27.iloc[0:3:2,]
```

Out[45]:

	Student name	age	Sex
1	Mohit	29	Male
3	Kavitha	25	Female

In [46]:
```
df27.iloc[0:2,0:3]
```

Out[46]:

	Student name	age	Sex
1	Mohit	29	Male
2	Mohitha	25	Female

Adding,Deleting,Updating,Renaming Row Columns cells in DataFrame Object:

In [11]:
```
import pandas as pd
diction1 = {'Student_name':'Mohit','age':29,'Sex':'Male'}
diction2 = {'Student_name':'Mohitha','age':25,'Sex':'Female'}
diction3 = {'Student_name':'Kavitha','age':29,'Sex':'Female'}
diction4 = {'Student_name':'govind','age':23,'Sex':'Male'}
list12 = [diction1,diction2,diction3,diction4]
df28 = pd.DataFrame(list12,index = [1,2,3,4])
print(df28)
```

```
  Student_name  age     Sex
1        Mohit   29    Male
2      Mohitha   25  Female
3      Kavitha   29  Female
4       govind   23    Male
```

1. Accessing Individual Element:

In [8]:
```
df28.age[1]
```

Out[8]: 29

In [9]:
```
df28.Sex[3]
```

Out[9]: 'Female'

In [12]:
```
df28.Student_name[2]
```

Out[12]: 'Mohitha'

In [16]:
```
df28.at[3,"Sex"]
```

Out[16]: 'Female'

In [17]:
```
df28.at[4,"age"]
```

Out[17]: 23

In [18]:
```
df28.at[1,"Student_name"]
```

Out[18]: 'Mohit'

In [22]:
```
df28.iat[3,1]
```

Out[22]: 23

In [23]:
```
df28.iat[2,2]
```

Out[23]: 'Female'

In [24]:
```
df28.iat[0,1]
```

Out[24]: 29

In [25]:
```
df28.iat[0,0]
```

Out[25]: 'Mohit'

In [27]:
```
df28.iat[0,2]
```

Out[27]: 'Male'

In [28]:
```
df28.iat[2,0]
```

Out[28]: 'Kavitha'

2. Adding/Modifying a column:

In [32]:
```
df28['Stream'] = 45
df28
```

Out[32]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	45
2	Mohitha	25	Female	45
3	Kavitha	29	Female	45
4	govind	23	Male	45

In [36]:
```
df28['Stream']=['EEE','ECE','Civil','Mechanical']
df28
```

Out[36]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	EEE
2	Mohitha	25	Female	ECE
3	Kavitha	29	Female	Civil
4	govind	23	Male	Mechanical

In [40]:
```
df28.loc[:,'Stream']=[10,20,30,40]
df28
```

Out[40]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	10
2	Mohitha	25	Female	20
3	Kavitha	29	Female	30
4	govind	23	Male	40

In [41]:
```
df28.loc[1:3,"age"]
```

Out[41]:
```
1    29
2    25
3    29
Name: age, dtype: int64
```

In [42]:
```
df28.loc[:4,"Student_name"]
```

Out[42]:
```
1      Mohit
2    Mohitha
3    Kavitha
4     govind
Name: Student_name, dtype: object
```

In [43]:
```
df28.loc[2:4,"Student_name"]
```

Out[43]:
```
2    Mohitha
3    Kavitha
4     govind
Name: Student_name, dtype: object
```

In [44]:
```
df28.loc[1:5,"Student_name"]
```

Out[44]:
```
1      Mohit
2    Mohitha
3    Kavitha
4     govind
Name: Student_name, dtype: object
```

In [45]:
```
df28.loc[1:5,"Student_name":'Stream']
```

Out[45]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	10
2	Mohitha	25	Female	20
3	Kavitha	29	Female	30
4	govind	23	Male	40

In [47]:
```
df28.loc[2:5,"Student_name":'Sex']
```

Out[47]:

	Student_name	age	Sex
2	Mohitha	25	Female
3	Kavitha	29	Female
4	govind	23	Male

In [51]:
```
df28.loc[2:5,"Student_name":'Sex':2]
```

Out[51]:

	Student_name	Sex
2	Mohitha	Female
3	Kavitha	Female
4	govind	Male

In [49]:
```
df28.loc[1:5:2,"Student_name":'Sex':2]
```

Out[49]:

	Student_name	Sex
1	Mohit	Male
3	Kavitha	Female

Adding/Modifying a Row:

In [55]:
```
df28.loc[5,:]=50
df28
```

Out[55]:

	Student_name	age	Sex	Stream
1	Mohit	29.0	Male	10.0
2	Mohitha	25.0	Female	20.0
3	Kavitha	29.0	Female	30.0
4	govind	23.0	Male	40.0
5	50	50.0	50	50.0

In [56]:
```
df28.loc[3,:]=30.6
df28
```

Out[56]:

	Student_name	age	Sex	Stream
1	Mohit	29.0	Male	10.0
2	Mohitha	25.0	Female	20.0
3	30.6	30.6	30.6	30.6
4	govind	23.0	Male	40.0
5	50	50.0	50	50.0

In [60]:
```
df28.loc[1:6,:]=70
df28
```

Out[60]:

	Student_name	age	Sex	Stream
1	70	70	70	70
2	70	70	70	70
3	70	70	70	70
4	70	70	70	70
5	70	70	70	70

In [61]:
```
df28.loc[2:5,1:4]=90
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\949955293.py:1: FutureWarning: Slicing a positiona
l slice with .loc is not supported, and will raise TypeError in a future version.  Use .loc with lab
els or .iloc with positions instead.
  df28.loc[2:5,1:4]=90
```

Out[61]:

	Student_name	age	Sex	Stream
1	70	70	70	70
2	70	90	90	90
3	70	90	90	90
4	70	90	90	90
5	70	90	90	90

In [62]:
```
df28.loc[2:5,:4]=90
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\2146471230.py:1: FutureWarning: Slicing a position
al slice with .loc is not supported, and will raise TypeError in a future version.  Use .loc with la
bels or .iloc with positions instead.
  df28.loc[2:5,:4]=90
```

Out[62]:

	Student_name	age	Sex	Stream
1	70	70	70	70
2	90	90	90	90
3	90	90	90	90
4	90	90	90	90
5	90	90	90	90

In [65]:
```
df28.loc[:,:]=[90,70,50,30]
df28
```

Out[65]:

	Student_name	age	Sex	Stream
1	90	70	50	30
2	90	70	50	30
3	90	70	50	30
4	90	70	50	30
5	90	70	50	30

Modifying a cell:

In [95]:
```
import pandas as pd
diction1 = {'Student_name':'Mohit','age':29,'Sex':'Male'}
diction2 = {'Student_name':'Mohitha','age':25,'Sex':'Female'}
diction3 = {'Student_name':'Kavitha','age':29,'Sex':'Female'}
diction4 = {'Student_name':'govind','age':23,'Sex':'Male'}
list12 = [diction1,diction2,diction3,diction4]
df28 = pd.DataFrame(list12,index = [1,2,3,4])
print(df28)
df28['Stream']=['EEE','ECE','Civil','Mechanical']
print(df28)
```

```
  Student_name  age     Sex
1        Mohit   29    Male
2      Mohitha   25  Female
3      Kavitha   29  Female
4       govind   23    Male
  Student_name  age     Sex      Stream
1        Mohit   29    Male         EEE
2      Mohitha   25  Female         ECE
3      Kavitha   29  Female       Civil
4       govind   23    Male  Mechanical
```

In [96]:
```
df28
```

Out[96]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	EEE
2	Mohitha	25	Female	ECE
3	Kavitha	29	Female	Civil
4	govind	23	Male	Mechanical

In [97]:
```
df28.age[3]
```

Out[97]:
```
29
```

In [98]:
```
df28.age[3]=209
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\4168710713.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df28.age[3]=209
```

Out[98]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	EEE
2	Mohitha	25	Female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

In [99]:
```
df28.Sex[2]='female'
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\916655782.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df28.Sex[2]='female'
```

Out[99]:

	Student_name	age	Sex	Stream
1	Mohit	29	Male	EEE
2	Mohitha	25	female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

In [100]:
```
df28.Student_name[1]='Manisha'
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\390146418.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df28.Student_name[1]='Manisha'
```

Out[100]:

	Student_name	age	Sex	Stream
1	Manisha	29	Male	EEE
2	Mohitha	25	female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

In [101]:
```
df28.Sex[1]='Female'
df28
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\3428991382.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df28.Sex[1]='Female'
```

Out[101]:

	Student_name	age	Sex	Stream
1	Manisha	29	Female	EEE
2	Mohitha	25	female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

In [102]:
```
df28
```

Out[102]:

	Student_name	age	Sex	Stream
1	Manisha	29	Female	EEE
2	Mohitha	25	female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

Deleting Row/ Column (del):

In [103]:
```
df28
```

Out[103]:

	Student_name	age	Sex	Stream
1	Manisha	29	Female	EEE
2	Mohitha	25	female	ECE
3	Kavitha	209	Female	Civil
4	govind	23	Male	Mechanical

In [104]:
```
del df28['Sex']
df28
```

Out[104]:

	Student_name	age	Stream
1	Manisha	29	EEE
2	Mohitha	25	ECE
3	Kavitha	209	Civil
4	govind	23	Mechanical

In [105]:
```
df28
```

Out[105]:

	Student_name	age	Stream
1	Manisha	29	EEE
2	Mohitha	25	ECE
3	Kavitha	209	Civil
4	govind	23	Mechanical

In [110]:
```
del df28(['age','Stream'],axis=1)
df28.drop([1,3])
```

```
  File "C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\2023024580.py", line 1
    del df28(['age','Stream'],axis=1)
        ^
SyntaxError: cannot delete function call
```

In [112]:
```
import pandas as pd
Exams = {'Year1':{'Mid1':18,'Mid2':19,'Final':82},\
        'Year2':{'Mid1':15,'Mid2':17,'Final':87},\
         'Year3':{'Mid1':12,'Mid2':13,'Final':79}}
df29 = pd.DataFrame(Exams)
print(df29)
```

```
       Year1  Year2  Year3
Mid1      18     15     12
Mid2      19     17     13
Final     82     87     79
```

In [114]:
```
for(a,b) in df29.iterrows():
    print('Row Index:',a,'\n value = \n',b)
```

```
Row Index: Mid1
 value =
 Year1    18
Year2    15
Year3    12
Name: Mid1, dtype: int64
Row Index: Mid2
 value =
 Year1    19
Year2    17
Year3    13
Name: Mid2, dtype: int64
Row Index: Final
 value =
 Year1    82
Year2    87
Year3    79
Name: Final, dtype: int64
```

In [126]:
```
import pandas as pd
Exams1 = {'Year1':{'Mid1':18,'Mid2':19,'Final':82},\
        'Year2':{'Mid1':15,'Mid2':17,'Final':87},\
         'Year3':{'Mid1':12,'Mid2':13,'Final':79}}
df30 = pd.DataFrame(Exams1)
print(df30)
for(a1,b1) in df30.iterrows():
    print('Row Index:'a1,'\n Contains\n')
    for i in b1:
        print(i)
```

```
  File "C:\Users\mouni\AppData\Local\Temp\ipykernel_22944\1145278622.py", line 8
    print('Row Index:'a1,'\n Contains\n')
                       ^
SyntaxError: invalid syntax
```

In [120]:
```
import pandas as pd
dictionary = {'Name':["Rama","Sita","Hanuman","Lakshmana"],'Count':[1,2,3,4]}
df31 = pd.DataFrame(dictionary)
print("Name\tCount")
print()
for i,j in df31.iterrows():
    print(j)
    print()
```

```
Name    Count

Name     Rama
Count       1
Name: 0, dtype: object

Name     Sita
Count       2
Name: 1, dtype: object

Name     Hanuman
Count          3
Name: 2, dtype: object

Name     Lakshmana
Count            4
Name: 3, dtype: object
```

In [129]:
```
import pandas as pd
dictionary2 = {'Name':["Rama","Sita","Hanuman","Lakshmana"],'Count':[1,2,3,4]}
df32 = pd.DataFrame(dictionary2)
print(df32)
```

```
        Name  Count
0       Rama      1
1       Sita      2
2    Hanuman      3
3  Lakshmana      4
```

In [132]:
```
import pandas as pd
dictionary3 = {'Name':["Rama","Sita","Hanuman","Lakshmana"],'Count':[1,2,3,4]}
df33 = pd.DataFrame(dictionary3)
print(df33)
for i,j in df33.iterrows():
    print(j)
```

```
        Name  Count
0       Rama      1
1       Sita      2
2    Hanuman      3
3  Lakshmana      4
Name     Rama
Count       1
Name: 0, dtype: object
Name     Sita
Count       2
Name: 1, dtype: object
Name     Hanuman
Count          3
Name: 2, dtype: object
Name     Lakshmana
Count            4
Name: 3, dtype: object
```

In [133]:
```
import pandas as pd
dictionary4 = {'Name':["Rama","Sita","Hanuman","Lakshmana"],'Count':[1,2,3,4]}
df34 = pd.DataFrame(dictionary4)
for i,j in df34.iterrows():
    print(j['Count'])
```

```
1
2
3
4
```

In [ ]:
```

```

Binary Operations in DataFrame

In a binary operation, two dataframes are aligned on the basis of their row and column idexes and for the matching row, column index, the given operation is performed and for non matching NaN is stored.

In [2]:
```
import pandas as pd
dictionary5 = {'A':[1,2,4],'B':[5,9,8],'C':[6,3,7]}
df35 = pd.DataFrame(dictionary5)
print(df35)
```

```
   A  B  C
0  1  5  6
1  2  9  3
2  4  8  7
```

In [3]:
```
import pandas as pd
dictionary6 = {'A':[100,200,400],'B':[500,700,900],'C':[300,600,700]}
df36 = pd.DataFrame(dictionary6)
print(df36)
```

```
     A    B    C
0  100  500  300
1  200  700  600
2  400  900  700
```

In [4]:
```
import pandas as pd
dictionary7 = {'A':[100,200],'B':[500,700],'C':[300,600]}
df37 = pd.DataFrame(dictionary7)
print(df37)
```

```
     A    B    C
0  100  500  300
1  200  700  600
```

In [5]:
```
import pandas as pd
dictionary8 = {'A':[100,200,900],'B':[300,500,700]}
df38 = pd.DataFrame(dictionary8)
print(df38)
```

```
     A    B
0  100  300
1  200  500
2  900  700
```

In [6]:
```
df35+df36
```

Out[6]:

	A	B	C
0	101	505	306
1	202	709	603
2	404	908	707

In [7]:
```
df35.add(df36)
```

Out[7]:

	A	B	C
0	101	505	306
1	202	709	603
2	404	908	707

In [8]:
```
df35.radd(df35)
```

Out[8]:

	A	B	C
0	2	10	12
1	4	18	6
2	8	16	14

In [9]:
```
df36-df37
```

Out[9]:

	A	B	C
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	NaN	NaN	NaN

In [10]:
```
df35-df36
```

Out[10]:

	A	B	C
0	-99	-495	-294
1	-198	-691	-597
2	-396	-892	-693

In [11]:
```
df36-df35
```

Out[11]:

	A	B	C
0	99	495	294
1	198	691	597
2	396	892	693

In [12]:
```
df35-df38
```

Out[12]:

	A	B	C
0	-99	-295	NaN
1	-198	-491	NaN
2	-896	-692	NaN

In [13]:
```
df35-df37
```

Out[13]:

	A	B	C
0	-99.0	-495.0	-294.0
1	-198.0	-691.0	-597.0
2	NaN	NaN	NaN

In [14]:
```
df35.sub(38)
```

Out[14]:

	A	B	C
0	-37	-33	-32
1	-36	-29	-35
2	-34	-30	-31

In [15]:
```
df35.rsub(df38)
```

Out[15]:

	A	B	C
0	99	295	NaN
1	198	491	NaN
2	896	692	NaN

In [16]:
```
df35.mul(df36)
```

Out[16]:

	A	B	C
0	100	2500	1800
1	400	6300	1800
2	1600	7200	4900

In [17]:
```
df37.mul(df35)
```

Out[17]:

	A	B	C
0	100.0	2500.0	1800.0
1	400.0	6300.0	1800.0
2	NaN	NaN	NaN

In [18]:
```
df38.mul(df36)
```

Out[18]:

	A	B	C
0	10000	150000	NaN
1	40000	350000	NaN
2	360000	630000	NaN

In [19]:
```
df35/df36
```

Out[19]:

	A	B	C
0	0.01	0.010000	0.020
1	0.01	0.012857	0.005
2	0.01	0.008889	0.010

In [21]:
```
df35.div(df36)
```

Out[21]:

	A	B	C
0	0.01	0.010000	0.020
1	0.01	0.012857	0.005
2	0.01	0.008889	0.010

In [22]:
```
df37.div(df38)
```

Out[22]:

	A	B	C
0	1.0	1.666667	NaN
1	1.0	1.400000	NaN
2	NaN	NaN	NaN

In [23]:
```
df38.rdiv(df35)
```

Out[23]:

	A	B	C
0	0.010000	0.016667	NaN
1	0.010000	0.018000	NaN
2	0.004444	0.011429	NaN

In [24]:
```
df35.rdiv(df37)
```

Out[24]:

	A	B	C
0	100.0	100.000000	50.0
1	100.0	77.777778	200.0
2	NaN	NaN	NaN

Arithmetic Operations:

In [70]:
```
import pandas as pd
DF1 = pd.DataFrame({"a":[37,89,72,54],"b":[50,94,71,28]})
print(DF1)
```

```
    a   b
0  37  50
1  89  94
2  72  71
3  54  28
```

In [72]:
```
DF1["c"] = DF1["a"]+DF1["b"]
print(DF1)
```

```
    a   b    c
0  37  50   87
1  89  94  183
2  72  71  143
3  54  28   82
```

In [73]:
```
DF1["c"] = DF1["a"]-DF1["b"]
print(DF1)
```

```
    a   b   c
0  37  50 -13
1  89  94  -5
2  72  71   1
3  54  28  26
```

In [74]:
```
DF1["c"] = DF1["a"]*DF1["b"]
print(DF1)
```

```
    a   b     c
0  37  50  1850
1  89  94  8366
2  72  71  5112
3  54  28  1512
```

In [75]:
```
DF1["c"] = DF1["a"]/DF1["b"]
print(DF1)
```

```
    a   b         c
0  37  50  0.740000
1  89  94  0.946809
2  72  71  1.014085
3  54  28  1.928571
```

In [13]:
```
import pandas as pd
DF2 = pd.DataFrame({"A":[20,40,60,80,100],"B":[10,30,50,70,90]})
print(DF2)
```

```
     A   B
0   20  10
1   40  30
2   60  50
3   80  70
4  100  90
```

In [14]:
```
DF2["Name"]=DF2["A"]<=60
print(DF2)
```

```
     A   B   Name
0   20  10   True
1   40  30   True
2   60  50   True
3   80  70  False
4  100  90  False
```

In [21]:
```
DF2["Name"]=DF2["A"]>=60
print(DF2)
```

```
     A   B   Name
0   20  10  False
1   40  30  False
2   60  50   True
3   80  70   True
4  100  90   True
```

In [23]:
```
DF2["Name"]=DF2["A"]>=60
DF2["Name_1"]=DF2["B"]>=30
print(DF2)
```

```
     A   B   Name  Name_1
0   20  10  False   False
1   40  30  False    True
2   60  50   True    True
3   80  70   True    True
4  100  90   True    True
```

Delete and Insert Functions in Python Pandas:

Insert:

In [2]:
```
import pandas as pd
DF3 = pd.DataFrame({"A":[2,4,6,8,10],"B":[1,3,5,7,9]})
print(DF3)
```

```
    A  B
0   2  1
1   4  3
2   6  5
3   8  7
4  10  9
```

In [3]:
```
DF3.insert(1,"Name",DF3[["B"]])
print(DF3)
```

```
    A  Name  B
0   2     1  1
1   4     3  3
2   6     5  5
3   8     7  7
4  10     9  9
```

In [4]:
```
DF3.insert(2,"Salary",DF3[["A"]])
print(DF3)
```

```
    A  Name  Salary  B
0   2     1       2  1
1   4     3       4  3
2   6     5       6  5
3   8     7       8  7
4  10     9      10  9
```

In [7]:
```
DF3.insert(1,"Age",[1,2,3,4,5])
print(DF3)
```

```
    A  Age  Name  Salary  B
0   2    1     1       2  1
1   4    2     3       4  3
2   6    3     5       6  5
3   8    4     7       8  7
4  10    5     9      10  9
```

In [9]:
```
DF3["Name_1"]=DF3["B"][:4]
print(DF3)
```

```
   A  Age  Name  Salary  B  Name_1
0  2    1     1       2  1     1.0
1  4    2     3       4  3     3.0
2  6    3     5       6  5     5.0
3  8    4     7       8  7     7.0
4 10    5     9      10  9     NaN
```

Delete:

In [14]:
```
import pandas as pd
DF4 = pd.DataFrame({"abc":[12,14,16,18],"def":[23,26,28,21],"ghi":[54,86,52,91]})
print(DF4)
```

```
   abc  def  ghi
0   12   23   54
1   14   26   86
2   16   28   52
3   18   21   91
```

In [17]:
```
DF5 = DF4.pop("abc")
print(DF5)
```

```
0    12
1    14
2    16
3    18
Name: abc, dtype: int64
```

In [19]:
```
DF3
```

Out[19]:

	A	Age	Name	Salary	B	Name_1
0	2	1	1	2	1	1.0
1	4	2	3	4	3	3.0
2	6	3	5	6	5	5.0
3	8	4	7	8	7	7.0
4	10	5	9	10	9	NaN

In [22]:
```
del DF4["def"]
print(DF4)
```

```
   ghi
0   54
1   86
2   52
3   91
```

In [ ]:
```

```

Merging & Concat in Python pandas:

Merge:

In [23]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
print(DF6)
```

```
    a   b
0  19  91
1  37  84
2  84  63
3  62  18
```

In [29]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[83,72,16,70],"b":[81,39,54,95]})
pd.merge(DF6,DF7)
```

Out[29]:

a	b

In [30]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[83,72,16,70],"c":[81,39,54,95]})
pd.merge(DF6,DF7,on="a")
```

Out[30]:

a	b	c

In [31]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,62],"c":[81,39,54,95]})
pd.merge(DF6,DF7,on="a")
```

Out[31]:

	a	b	c
0	19	91	81
1	37	84	39
2	84	63	54
3	62	18	95

In [32]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,62],"c":[81,39,54,95]})
pd.merge(DF7,DF6,on="a")
```

Out[32]:

	a	c	b
0	19	81	91
1	37	39	84
2	84	54	63
3	62	95	18

In [35]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,62],"c":[81,39,54,95]})
pd.merge(DF6,DF7,on="a")
```

Out[35]:

	a	b	c
0	19	91	81
1	37	84	39
2	84	63	54
3	62	18	95

In [36]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,56],"c":[81,39,54,95]})
pd.merge(DF6,DF7,on="a")
```

Out[36]:

	a	b	c
0	19	91	81
1	37	84	39
2	84	63	54

In [37]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,56],"c":[81,39,54,95]})
pd.merge(DF6,DF7)
```

Out[37]:

	a	b	c
0	19	91	81
1	37	84	39
2	84	63	54

In [38]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,56],"c":[81,39,54,95]})
pd.merge(DF6,DF7,how='inner')
```

Out[38]:

	a	b	c
0	19	91	81
1	37	84	39
2	84	63	54

In [39]:
```
pd.merge(DF6,DF7,how="left")
```

Out[39]:

	a	b	c
0	19	91	81.0
1	37	84	39.0
2	84	63	54.0
3	62	18	NaN

In [40]:
```
pd.merge(DF6,DF7,how="right")
```

Out[40]:

	a	b	c
0	19	91.0	81
1	37	84.0	39
2	84	63.0	54
3	56	NaN	95

In [41]:
```
pd.merge(DF6,DF7,how="outer")
```

Out[41]:

	a	b	c
0	19	91.0	81.0
1	37	84.0	39.0
2	84	63.0	54.0
3	62	18.0	NaN
4	56	NaN	95.0

In [42]:
```
pd.merge(DF6,DF7,how="outer",indicator=True)
```

Out[42]:

	a	b	c	_merge
0	19	91.0	81.0	both
1	37	84.0	39.0	both
2	84	63.0	54.0	both
3	62	18.0	NaN	left_only
4	56	NaN	95.0	right_only

In [43]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,56],"b":[81,39,54,95]})
pd.merge(DF6,DF7)
```

Out[43]:

	a	b

In [44]:
```
import pandas as pd
DF6 = pd.DataFrame({"a":[19,37,84,62],"b":[91,84,63,18]})
DF7 = pd.DataFrame({"a":[19,37,84,56],"b":[81,39,54,95]})
pd.merge(DF6,DF7,left_index=True,right_index=True)
```

Out[44]:

	a_x	b_x	a_y	b_y
0	19	91	19	81
1	37	84	37	39
2	84	63	84	54
3	62	18	56	95

In [45]:
```
pd.merge(DF6,DF7,left_index=True,right_index=True,suffixes=("Num","Count"))
```

Out[45]:

	aNum	bNum	aCount	bCount
0	19	91	19	81
1	37	84	37	39
2	84	63	84	54
3	62	18	56	95

In [48]:
```
pd.merge(DF6,DF7,left_index=True,right_index=True,suffixes=("A","B"))
```

Out[48]:

	aA	bA	aB	bB
0	19	91	19	81
1	37	84	37	39
2	84	63	84	54
3	62	18	56	95

Concat:

In [49]:
```
S1 = pd.Series([134,168,194,159])
S2 = pd.Series([527,962,198,463])
print(S1)
print(S2)
```

```
0    134
1    168
2    194
3    159
dtype: int64
0    527
1    962
2    198
3    463
dtype: int64
```

In [50]:
```
pd.concat([S1,S2])
```

Out[50]:
```
0    134
1    168
2    194
3    159
0    527
1    962
2    198
3    463
dtype: int64
```

In [51]:
```
import pandas as pd
DF8 = pd.DataFrame({"A":[45,87,93,36],"B":[89,57,84,32]})
DF9 = pd.DataFrame({"A":[45,87,93,39],"B":[12,15,18,17]})
pd.concat([DF8,DF9])
```

Out[51]:

	A	B
0	45	89
1	87	57
2	93	84
3	36	32
0	45	12
1	87	15
2	93	18
3	39	17

In [56]:
```
import pandas as pd
DF8 = pd.DataFrame({"A":[45,87,93,36],"B":[89,57,84,32]})
DF9 = pd.DataFrame({"A":[45,87,93,36],"B1":[102,115,168,197]})
pd.merge(DF8,DF9)
```

Out[56]:

	A	B	B1
0	45	89	102
1	87	57	115
2	93	84	168
3	36	32	197

In [58]:
```
import pandas as pd
DF8 = pd.DataFrame({"A":[45,87,93,36],"B":[89,57,84,32]})
DF9 = pd.DataFrame({"A":[45,87,93,36],"B":[102,115,168,197]})
pd.concat([DF8,DF9],axis=1)
```

Out[58]:

	A	B	A	B
0	45	89	45	102
1	87	57	87	115
2	93	84	93	168
3	36	32	36	197

In [59]:
```
pd.concat([DF8,DF9],axis=0)
```

Out[59]:

	A	B
0	45	89
1	87	57
2	93	84
3	36	32
0	45	102
1	87	115
2	93	168
3	36	197

In [61]:
```
import pandas as pd
DF8 = pd.DataFrame({"A":[45,87,93,36],"B":[89,57,84,32]})
DF9 = pd.DataFrame({"A":[45,87],"C":[102,115]})
pd.concat([DF8,DF9],axis=1)
```

Out[61]:

	A	B	A	C
0	45	89	45.0	102.0
1	87	57	87.0	115.0
2	93	84	NaN	NaN
3	36	32	NaN	NaN

In [62]:
```
pd.concat([DF8,DF9],axis=1,join="outer")
```

Out[62]:

	A	B	A	C
0	45	89	45.0	102.0
1	87	57	87.0	115.0
2	93	84	NaN	NaN
3	36	32	NaN	NaN

In [63]:
```
pd.concat([DF8,DF9],axis=1,join="inner")
```

Out[63]:

	A	B	A	C
0	45	89	45	102
1	87	57	87	115

In [65]:
```
import pandas as pd
DF8 = pd.DataFrame({"A":[45,87,93,36],"B":[89,57,84,32]})
DF9 = pd.DataFrame({"A":[45,87,93,36],"C":[102,115,189,657]})
pd.concat([DF8,DF9],axis=1,keys=["DF8","DF9"])
```

Out[65]:

	DF8		DF9	
	A	B	A	C
0	45	89	45	102
1	87	57	87	115
2	93	84	93	189
3	36	32	36	657

In [66]:
```
pd.concat([DF8,DF9],axis=0,keys=["DF8","DF9"])
```

Out[66]:

		A	B	C
DF8	0	45	89.0	NaN
	1	87	57.0	NaN
	2	93	84.0	NaN
	3	36	32.0	NaN
DF9	0	45	NaN	102.0
	1	87	NaN	115.0
	2	93	NaN	189.0
	3	36	NaN	657.0

In [68]:
```
import pandas as pd
DF8 = pd.DataFrame({"a":[12,17,93,52]})
DF9 = pd.DataFrame({"a":[83,62,51,98],"c":[91,42,68,93]})
pd.concat([DF8,DF9])
```

Out[68]:

	a	c
0	12	NaN
1	17	NaN
2	93	NaN
3	52	NaN
0	83	91.0
1	62	42.0
2	51	68.0
3	98	93.0

In [69]:
```
import pandas as pd
DF8 = pd.DataFrame({"b":[12,17,93,52]})
DF9 = pd.DataFrame({"a":[83,62,51,98],"c":[91,42,68,93]})
pd.concat([DF8,DF9])
```

Out[69]:

	b	a	c
0	12.0	NaN	NaN
1	17.0	NaN	NaN
2	93.0	NaN	NaN
3	52.0	NaN	NaN
0	NaN	83.0	91.0
1	NaN	62.0	42.0
2	NaN	51.0	68.0
3	NaN	98.0	93.0

Group by in Pandas Pandas:

Group by:

In [78]:
```
import pandas as pd
DF11 = pd.DataFrame({"Name":["A","B","G","C","B","D","E","A","F","G","A"],"C1":[1,4,7,98,45,32,65,98,
print(DF11)
```

```
   Name  C1    C2
0     A   1   345
1     B   4   823
2     G   7  9123
3     C  98   457
4     B  45   683
5     D  32   953
6     E  65   238
7     A  98   687
8     F  56   346
9     G  32   901
10    A  17   289
```

In [79]:
```
DF11_new = DF11.groupby("Name")
DF11
```

Out[79]:

	Name	C1	C2
0	A	1	345
1	B	4	823
2	G	7	9123
3	C	98	457
4	B	45	683
5	D	32	953
6	E	65	238
7	A	98	687
8	F	56	346
9	G	32	901
10	A	17	289

In [80]:
```
for i,j in DF11_new:
    print(i)
    print(j)
```

```
A
   Name  C1   C2
0     A   1  345
7     A  98  687
10    A  17  289
B
  Name  C1   C2
1    B   4  823
4    B  45  683
C
  Name  C1   C2
3    C  98  457
D
  Name  C1   C2
5    D  32  953
E
  Name  C1   C2
6    E  65  238
F
  Name  C1   C2
8    F  56  346
G
  Name  C1    C2
2    G   7  9123
9    G  32   901
```

In [81]:
```
DF11_new.get_group("A")
```

Out[81]:

	Name	C1	C2
0	A	1	345
7	A	98	687
10	A	17	289

In [82]:
```
DF11_new.get_group("F")
```

Out[82]:

	Name	C1	C2
8	F	56	346

In [83]:
```
DF11_new.min()
```

Out[83]:

	C1	C2
Name		
A	1	289
B	4	683
C	98	457
D	32	953
E	65	238
F	56	346
G	7	901

In [84]:
```
DF11_new.max()
```

Out[84]:

Name	C1	C2
A	98	687
B	45	823
C	98	457
D	32	953
E	65	238
F	56	346
G	32	9123

In [85]:
```
DF11_new.mean()
```

Out[85]:

Name	C1	C2
A	38.666667	440.333333
B	24.500000	753.000000
C	98.000000	457.000000
D	32.000000	953.000000
E	65.000000	238.000000
F	56.000000	346.000000
G	19.500000	5012.000000

In [87]:
```
List1 = list(DF11_new)
print(List1)
```

```
[('A',    Name  C1   C2
0     A   1  345
7     A  98  687
10    A  17  289), ('B',   Name  C1   C2
1    B   4  823
4    B  45  683), ('C',   Name  C1   C2
3    C  98  457), ('D',   Name  C1   C2
5    D  32  953), ('E',   Name  C1   C2
6    E  65  238), ('F',   Name  C1   C2
8    F  56  346), ('G',   Name  C1    C2
2    G   7  9123
9    G  32   901)]
```

Join & Append in Python Pandas:

Join:

In [89]:
```
import pandas as pd
DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
DF13 = pd.DataFrame({"C":[79,53,85,26],"D":[51,23,50,84]})
DF12.join(DF13)
```

Out[89]:

	A	B	C	D
0	13	24	79	51
1	16	67	53	23
2	18	42	85	50
3	15	96	26	84

In [90]:
```
import pandas as pd
DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
DF13 = pd.DataFrame({"C":[79,53],"D":[51,23]})
DF12.join(DF13)
```

Out[90]:

	A	B	C	D
0	13	24	79.0	51.0
1	16	67	53.0	23.0
2	18	42	NaN	NaN
3	15	96	NaN	NaN

In [92]:
```
import pandas as pd
DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]},index=["i","ii","iii","iv"])
DF13 = pd.DataFrame({"C":[79,53],"D":[51,23]})
DF12.join(DF13)
```

Out[92]:

	A	B	C	D
i	13	24	NaN	NaN
ii	16	67	NaN	NaN
iii	18	42	NaN	NaN
iv	15	96	NaN	NaN

In [93]:
```
import pandas as pd
DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]},index=["i","ii","iii","iv"])
DF13 = pd.DataFrame({"C":[79,53],"D":[51,23]},index=["i","ii"])
DF12.join(DF13)
```

Out[93]:

	A	B	C	D
i	13	24	79.0	51.0
ii	16	67	53.0	23.0
iii	18	42	NaN	NaN
iv	15	96	NaN	NaN

In [94]:
```
DF13.join(DF12)
```

Out[94]:

	C	D	A	B
i	79	51	13	24
ii	53	23	16	67

In [95]:
```
DF13.join(DF12,how ="left")
```

Out[95]:

	C	D	A	B
i	79	51	13	24
ii	53	23	16	67

In [96]:
```
DF13.join(DF12,how ="right")
```

Out[96]:

	C	D	A	B
i	79.0	51.0	13	24
ii	53.0	23.0	16	67
iii	NaN	NaN	18	42
iv	NaN	NaN	15	96

In [97]:
```
DF13.join(DF12,how="inner")
```

Out[97]:

	C	D	A	B
i	79	51	13	24
ii	53	23	16	67

Possible Errors:

In [98]:
```
import pandas as pd
DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
DF13 = pd.DataFrame({"C":[79,53,85,26],"B":[51,23,50,84]})
DF13.join(DF12)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26500\2528714263.py in <module>
      2 DF12 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
      3 DF13 = pd.DataFrame({"C":[79,53,85,26],"B":[51,23,50,84]})
----> 4 DF13.join(DF12)

C:\anaconda\lib\site-packages\pandas\core\frame.py in join(self, other, on, how, lsuffix, rsuffix, s
ort)
   9261         5  K1  A5   B1
   9262         """
-> 9263         return self._join_compat(
   9264             other, on=on, how=how, lsuffix=lsuffix, rsuffix=rsuffix, sort=sort
   9265         )

C:\anaconda\lib\site-packages\pandas\core\frame.py in _join_compat(self, other, on, how, lsuffix, rs
uffix, sort)
   9292                     sort=sort,
   9293                 )
-> 9294             return merge(
   9295                 self,
   9296                 other,

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in merge(left, right, how, on, left_on, r
ight_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
    120         validate=validate,
    121     )
--> 122     return op.get_result()
    123
    124

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in get_result(self)
    716         join_index, left_indexer, right_indexer = self._get_join_info()
    717
--> 718         llabels, rlabels = _items_overlap_with_suffix(
    719             self.left._info_axis, self.right._info_axis, self.suffixes
    720         )

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in _items_overlap_with_suffix(left, righ
t, suffixes)
   2315
   2316     if not lsuffix and not rsuffix:
-> 2317         raise ValueError(f"columns overlap but no suffix specified: {to_rename}")
   2318
   2319     def renamer(x, suffix):

ValueError: columns overlap but no suffix specified: Index(['B'], dtype='object')
```

In [1]:
```
import pandas as pd
DF14 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
DF15 = pd.DataFrame({"C":[79,53],"B":[50,84]})
DF15.join(DF14,how="inner")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_34700\1803462179.py in <module>
      2 DF14 = pd.DataFrame({"A":[13,16,18,15],"B":[24,67,42,96]})
      3 DF15 = pd.DataFrame({"C":[79,53],"B":[50,84]})
----> 4 DF15.join(DF14,how="inner")

C:\anaconda\lib\site-packages\pandas\core\frame.py in join(self, other, on, how, lsuffix, rsuffix, s
ort)
   9261         5  K1  A5   B1
   9262         """
-> 9263         return self._join_compat(
   9264             other, on=on, how=how, lsuffix=lsuffix, rsuffix=rsuffix, sort=sort
   9265         )

C:\anaconda\lib\site-packages\pandas\core\frame.py in _join_compat(self, other, on, how, lsuffix, rs
uffix, sort)
   9292                     sort=sort,
   9293                 )
-> 9294             return merge(
   9295                 self,
   9296                 other,

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in merge(left, right, how, on, left_on, r
ight_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
    120         validate=validate,
    121     )
--> 122     return op.get_result()
    123
    124

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in get_result(self)
    716         join_index, left_indexer, right_indexer = self._get_join_info()
    717
--> 718         llabels, rlabels = _items_overlap_with_suffix(
    719             self.left._info_axis, self.right._info_axis, self.suffixes
    720         )

C:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in _items_overlap_with_suffix(left, righ
t, suffixes)
   2315
   2316     if not lsuffix and not rsuffix:
-> 2317         raise ValueError(f"columns overlap but no suffix specified: {to_rename}")
   2318
   2319     def renamer(x, suffix):

ValueError: columns overlap but no suffix specified: Index(['B'], dtype='object')
```

In [2]:
```
DF15.join(DF14,how="inner",lsuffix="1")
```

Out[2]:

	C	B1	A	B
0	79	50	13	24
1	53	84	16	67

In [3]:
```
DF15.join(DF14,how="outer",lsuffix="1")
```

Out[3]:

	C	B1	A	B
0	79.0	50.0	13	24
1	53.0	84.0	16	67
2	NaN	NaN	18	42
3	NaN	NaN	15	96

In [4]:
```
DF15.join(DF14,how="outer",lsuffix="2",rsuffix="3")
```

Out[4]:

	C	B2	A	B3
0	79.0	50.0	13	24
1	53.0	84.0	16	67
2	NaN	NaN	18	42
3	NaN	NaN	15	96

In [5]:
```
DF14.join(DF15,how="outer",lsuffix="2",rsuffix="1")
```

Out[5]:

	A	B2	C	B1
0	13	24	79.0	50.0
1	16	67	53.0	84.0
2	18	42	NaN	NaN
3	15	96	NaN	NaN

In [7]:
```
DF15.join(DF14,how="outer",lsuffix="2",rsuffix="1")
```

Out[7]:

	C	B2	A	B1
0	79.0	51.0	13	24
1	53.0	23.0	16	67
2	NaN	NaN	18	42
3	NaN	NaN	15	96

Append:

In [10]:
```
DF16 = pd.DataFrame({"a":[2,6,9,5,1],"b":[23,84,91,54,61]},index = ["i","ii","iii","iv","v"])
DF17 = pd.DataFrame({"c":[29,35],"d":[89,43]},index = ["i","ii"])
DF16.append(DF17)
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_34700\2771012051.py:3: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  DF16.append(DF17)
```

Out[10]:

	a	b	c	d
i	2.0	23.0	NaN	NaN
ii	6.0	84.0	NaN	NaN
iii	9.0	91.0	NaN	NaN
iv	5.0	54.0	NaN	NaN
v	1.0	61.0	NaN	NaN
i	NaN	NaN	29.0	89.0
ii	NaN	NaN	35.0	43.0

In [12]:
```
DF18 = pd.DataFrame({"a":[2,6,9,5,1],"b":[23,84,91,54,61]},index = ["i","ii","iii","iv","v"])
DF19 = pd.DataFrame({"c":[29,35],"b":[89,43]},index = ["i","ii"])
DF19.append(DF18)
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_34700\1623146669.py:3: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  DF19.append(DF18)
```

Out[12]:

	c	b	a
i	29.0	89	NaN
ii	35.0	43	NaN
i	NaN	23	2.0
ii	NaN	84	6.0
iii	NaN	91	9.0
iv	NaN	54	5.0
v	NaN	61	1.0

In [13]:
```
DF20 = pd.DataFrame({"a":[2,6,9,5,1],"b":[23,84,91,54,61]})
DF21 = pd.DataFrame({"c":[29,35],"b":[89,43]})
DF21.append(DF21)
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_34700\3767530846.py:3: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  DF21.append(DF21)
```

Out[13]:

	c	b
0	29	89
1	35	43
0	29	89
1	35	43

In [14]:
```
DF22 = pd.DataFrame({"a":[2,6,9,5,1],"b":[23,84,91,54,61]},index = ["i","ii","iii","iv","v"])
DF23 = pd.DataFrame({"c":[29,35],"b":[89,43]},index = ["i","ii"])
DF23.append(DF22)
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_34700\1061274591.py:3: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  DF23.append(DF22)
```

Out[14]:

	c	b	a
i	29.0	89	NaN
ii	35.0	43	NaN
i	NaN	23	2.0
ii	NaN	84	6.0
iii	NaN	91	9.0
iv	NaN	54	5.0
v	NaN	61	1.0

In [15]:
```
DF22.append(DF23,ignore_index = True)
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_34700\3960243151.py:1: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  DF22.append(DF23,ignore_index = True)
```

Out[15]:

	a	b	c
0	2.0	23	NaN
1	6.0	84	NaN
2	9.0	91	NaN
3	5.0	54	NaN
4	1.0	61	NaN
5	NaN	89	29.0
6	NaN	43	35.0

Pivot Table & Melt Function in Python pandas:

Melt:

In [16]:
```
import pandas as pd
DF24 = pd.DataFrame({"week":[3,5,4,1,6,2],"Month":[10,9,3,6,11,12],"Year":[2000,1990,2004,2005,2001,2
DF24
```

Out[16]:

	week	Month	Year
0	3	10	2000
1	5	9	1990
2	4	3	2004
3	1	6	2005
4	6	11	2001
5	2	12	2008

In [18]:
```
pd.melt(DF24)
```

Out[18]:

	variable	value
0	week	3
1	week	5
2	week	4
3	week	1
4	week	6
5	week	2
6	Month	10
7	Month	9
8	Month	3
9	Month	6
10	Month	11
11	Month	12
12	Year	2000
13	Year	1990
14	Year	2004
15	Year	2005
16	Year	2001
17	Year	2008

In [20]:
```
pd.melt(DF24,id_vars=["Month"])
```

Out[20]:

	Month	variable	value
0	10	week	3
1	9	week	5
2	3	week	4
3	6	week	1
4	11	week	6
5	12	week	2
6	10	Year	2000
7	9	Year	1990
8	3	Year	2004
9	6	Year	2005
10	11	Year	2001
11	12	Year	2008

In [21]:
```
pd.melt(DF24,id_vars=["Year"])
```

Out[21]:

	Year	variable	value
0	2000	week	3
1	1990	week	5
2	2004	week	4
3	2005	week	1
4	2001	week	6
5	2008	week	2
6	2000	Month	10
7	1990	Month	9
8	2004	Month	3
9	2005	Month	6
10	2001	Month	11
11	2008	Month	12

In [26]:
```
pd.melt(DF24,id_vars=["week"],var_name="WEEK")
```

Out[26]:

	week	WEEK	value
0	3	Month	10
1	5	Month	9
2	4	Month	3
3	1	Month	6
4	6	Month	11
5	2	Month	12
6	3	Year	2000
7	5	Year	1990
8	4	Year	2004
9	1	Year	2005
10	6	Year	2001
11	2	Year	2008

In [36]:
```
pd.melt(DF24,id_vars=["week"],var_name="MONTH",value_name="YEAR")
```

Out[36]:

	week	MONTH	YEAR
0	3	Month	10
1	5	Month	9
2	4	Month	3
3	1	Month	6
4	6	Month	11
5	2	Month	12
6	3	Year	2000
7	5	Year	1990
8	4	Year	2004
9	1	Year	2005
10	6	Year	2001
11	2	Year	2008

Pivot:

In [2]:
```
import pandas as pd
DF25=pd.DataFrame({"s.no":["a","b","c","d","e","f"],"Rank":[23,56,85,98,24,59],"Grade_Points":[9,8,6,
print(DF25)
```

```
  s.no  Rank  Grade_Points  Percentage
0    a    23             9          80
1    b    56             8          90
2    c    85             6          75
3    d    98             5          97
4    e    24            10          86
5    f    59             7          69
```

Possible Error:

In [3]:
```
DF25.pivot()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22388\4054149670.py in <module>
----> 1 DF25.pivot()

C:\anaconda\lib\site-packages\pandas\core\frame.py in pivot(self, index, columns, values)
   7883         from pandas.core.reshape.pivot import pivot
   7884
-> 7885         return pivot(self, index=index, columns=columns, values=values)
   7886
   7887     _shared_docs[

C:\anaconda\lib\site-packages\pandas\core\reshape\pivot.py in pivot(data, index, columns, values)
    478 ) -> DataFrame:
    479     if columns is None:
--> 480         raise TypeError("pivot() missing 1 required argument: 'columns'")
    481
    482     columns_listlike = com.convert_to_list_like(columns)

TypeError: pivot() missing 1 required argument: 'columns'
```

In [4]:
```
DF25.pivot(index="s.no",columns="Rank")
```

Out[4]:

	Grade_Points						Percentage					
Rank	**23**	**24**	**56**	**59**	**85**	**98**	**23**	**24**	**56**	**59**	**85**	**98**
s.no												
a	9.0	NaN	NaN	NaN	NaN	NaN	80.0	NaN	NaN	NaN	NaN	NaN
b	NaN	NaN	8.0	NaN	NaN	NaN	NaN	NaN	90.0	NaN	NaN	NaN
c	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN	NaN	NaN	75.0	NaN
d	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	97.0
e	NaN	10.0	NaN	NaN	NaN	NaN	NaN	86.0	NaN	NaN	NaN	NaN
f	NaN	NaN	NaN	7.0	NaN	NaN	NaN	NaN	NaN	69.0	NaN	NaN

In [5]:
```
DF25.pivot(index="Rank",columns="s.no")
```

Out[5]:

	Grade_Points						Percentage					
s.no	**a**	**b**	**c**	**d**	**e**	**f**	**a**	**b**	**c**	**d**	**e**	**f**
Rank												
23	9.0	NaN	NaN	NaN	NaN	NaN	80.0	NaN	NaN	NaN	NaN	NaN
24	NaN	NaN	NaN	NaN	10.0	NaN	NaN	NaN	NaN	NaN	86.0	NaN
56	NaN	8.0	NaN	NaN	NaN	NaN	NaN	90.0	NaN	NaN	NaN	NaN
59	NaN	NaN	NaN	NaN	NaN	7.0	NaN	NaN	NaN	NaN	NaN	69.0
85	NaN	NaN	6.0	NaN	NaN	NaN	NaN	NaN	75.0	NaN	NaN	NaN
98	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	97.0	NaN	NaN

In [6]:
```
import pandas as pd
DF26=pd.DataFrame({"s.no":["a","b","c","a","d","c"],"Rank":[23,56,85,98,24,59],"Grade_Points":[9,8,6,
print(DF26)
```

```
  s.no  Rank  Grade_Points  Percentage
0    a    23             9          80
1    b    56             8          90
2    c    85             6          75
3    a    98             5          97
4    d    24            10          86
5    c    59             7          69
```

In [7]:
```
DF26.pivot(index="Rank",columns="s.no",values="Grade_Points")
```

Out[7]:

s.no	**a**	**b**	**c**	**d**
Rank				
23	9.0	NaN	NaN	NaN
24	NaN	NaN	NaN	10.0
56	NaN	8.0	NaN	NaN
59	NaN	NaN	7.0	NaN
85	NaN	NaN	6.0	NaN
98	5.0	NaN	NaN	NaN

In [8]:
```
DF26.pivot(index="Rank",columns="s.no",values="Percentage")
```

Out[8]:

s.no	**a**	**b**	**c**	**d**
Rank				
23	80.0	NaN	NaN	NaN
24	NaN	NaN	NaN	86.0
56	NaN	90.0	NaN	NaN
59	NaN	NaN	69.0	NaN
85	NaN	NaN	75.0	NaN
98	97.0	NaN	NaN	NaN

In [9]:
```
DF26.pivot_table(index="s.no",columns="Rank",aggfunc="sum")
```

Out[9]:

	Grade_Points						Percentage					
Rank	**23**	**24**	**56**	**59**	**85**	**98**	**23**	**24**	**56**	**59**	**85**	**98**
s.no												
a	9.0	NaN	NaN	NaN	NaN	5.0	80.0	NaN	NaN	NaN	NaN	97.0
b	NaN	NaN	8.0	NaN	NaN	NaN	NaN	NaN	90.0	NaN	NaN	NaN
c	NaN	NaN	NaN	7.0	6.0	NaN	NaN	NaN	NaN	69.0	75.0	NaN
d	NaN	10.0	NaN	NaN	NaN	NaN	NaN	86.0	NaN	NaN	NaN	NaN

In [10]:
```
DF26.pivot_table(index="s.no",columns="Rank",aggfunc="mean",margins="True")
```

Out[10]:

	Grade_Points							Percentage						
Rank	**23**	**24**	**56**	**59**	**85**	**98**	**All**	**23**	**24**	**56**	**59**	**85**	**98**	**All**
s.no														
a	9.0	NaN	NaN	NaN	NaN	5.0	7.0	80.0	NaN	NaN	NaN	NaN	97.0	88.500000
b	NaN	NaN	8.0	NaN	NaN	NaN	8.0	NaN	NaN	90.0	NaN	NaN	NaN	90.000000
c	NaN	NaN	NaN	7.0	6.0	NaN	6.5	NaN	NaN	NaN	69.0	75.0	NaN	72.000000
d	NaN	10.0	NaN	NaN	NaN	NaN	10.0	NaN	86.0	NaN	NaN	NaN	NaN	86.000000
All	9.0	10.0	8.0	7.0	6.0	5.0	7.5	80.0	86.0	90.0	69.0	75.0	97.0	82.833333

In [11]:
```
DF26.pivot_table(index="s.no",columns="Rank",aggfunc="sum",margins="True")
```

Out[11]:

	Grade_Points							Percentage						
Rank	**23**	**24**	**56**	**59**	**85**	**98**	**All**	**23**	**24**	**56**	**59**	**85**	**98**	**All**
s.no														
a	9.0	NaN	NaN	NaN	NaN	5.0	14	80.0	NaN	NaN	NaN	NaN	97.0	177
b	NaN	NaN	8.0	NaN	NaN	NaN	8	NaN	NaN	90.0	NaN	NaN	NaN	90
c	NaN	NaN	NaN	7.0	6.0	NaN	13	NaN	NaN	NaN	69.0	75.0	NaN	144
d	NaN	10.0	NaN	NaN	NaN	NaN	10	NaN	86.0	NaN	NaN	NaN	NaN	86
All	9.0	10.0	8.0	7.0	6.0	5.0	45	80.0	86.0	90.0	69.0	75.0	97.0	497