PANDAS : BUILD ON TOP OF NUMPY :

PAGE 233

In [1]:
```
# The particular 'PANDAS' has been build 'on Top of NUMPY'
# 'PANDAS' also Specially designed to handle 'Mathematical Working Environment'
# 'NUMPY' Specially designed to handle the 'Numbers (or) Integers' called as 'Numeric Python'
# The 'PANDAS' designed on Top of 'NUMPY' to handle Data very perfectly.
# In 'PANDAS' We can see - Any Data going to be Converted to 'Data Frames' which is called as an a -
# 'Structured Format of Data'.
# Now something more we are going to see in 'PANDAS' are (1)Series, (2)Data Frames, (3)Groups,
# (4)Group By, (5)Merging, (6)Concating. we are going apply on Data.

# (1)Series - The 1st imp thing - Any Data we take, it will get with an a 'Index'
# (2)Data Frame(df) - Always will be "Index + Label" , Whenever we call 'Data' - Mandatorily We -
#    will get 'Index + Label'. Default 'Index' will be Generated, The first Row is Cosidered as'Label'
# If we don't have Labels, We can create New labels. That is the Advantage.
```

NOTE : PANDAS LIBRARY :

PAGE 235

In [2]:
```
# Pandas has been built on top of 'Numpy'
# We required Pandas, because ,
# (1) Data Operation Function, We can work with,
# (2) Data Handling Function,
# (3) Data Structure Handle - Major use cases,
# (4) Data Standardisation Functions.
```

FEATURES OF PANDAS :

In [3]:
```
# (1) PANDAS maintain 'Powerful Data Structures'
# (2) It is Fast and Effeciant - Data Wrangling
# (3) It is Easy Data Agregassion and Transformation
# (4) It maintain a Data Structures with one dimensional labeled array -> Supports multiple data -
#     type, called as an a 'Series'
# (5) Two Dimensional Labeled array - Which Supports 'Multiple Data types', is Called as 'Data Frames'
```

1. SERIES : in PANDAS :

PAGE 236

In [9]:
```
import pandas as pd
import numpy as np
```

In [6]:
```
# Now, "Series will have with an a Index Number"
# Here, pd.Series --> 'S' Capital,
# Now, We have seen the Index Numbers - 0,1,2,3,4,5

first_series = pd.Series(list('abcdef'))
print(first_series)
```

```
0    a
1    b
2    c
3    d
4    e
5    f
dtype: object
```

In [10]:
```
# Here, We daclared - List data,list data, Numpy array, Dictionary. '4' different types of data taken:

labels = ['a','b','c']
my_data = [10,20,30]

arr = np.array(my_data)
d = {'a':10,'b':20,'c':30}
```

In [11]:
```
pd.Series(data = my_data)
```

Out[11]:
```
0    10
1    20
2    30
dtype: int64
```

In [12]:
```
pd.Series(data = my_data, index = labels)
```

Out[12]:
```
a    10
b    20
c    30
dtype: int64
```

In [13]:
```
pd.Series(arr,labels)
```

Out[13]:
```
a    10
b    20
c    30
dtype: int32
```

In [14]:
```
pd.Series(d)
```

Out[14]:
```
a    10
b    20
c    30
dtype: int64
```

In [16]:
```
ser1 = pd.Series([1,2,3,4],['USA','GERMANY','USSR','JAPAN'])
ser1s
```

Out[16]:
```
USA        1
GERMANY    2
USSR       3
JAPAN      4
dtype: int64
```

In [17]:
```
ser2 = pd.Series([1,2,5,4],['USA','GERMANY','ITALY','JAPAN'])
ser2
```

Out[17]:
```
USA        1
GERMANY    2
ITALY      5
JAPAN      4
dtype: int64
```

In [18]:
```
# Now, Combining both the Serieses :
```

In [20]:
```
# Here, Germany 4.0, Italy and Ussr are NaN(means 'Non'), Because, We don't have them in -
# above both the Serieses :
# 'Index of Values' : Summations are taking here : That's why this is called as an a -
# - "SUMMATION OF SERIES"

ser1 + ser2
```

Out[20]:
```
GERMANY    4.0
ITALY      NaN
JAPAN      8.0
USA        2.0
USSR       NaN
dtype: float64
```

2. DATA FRAMES :

PAGE 240

NOTE : DATA FRAMES :

In [22]:
```
#(A) DATA FRAME is an a Main Object in pandas. It is used to Represent the Data with Rows and Columns
#(B) DATA FRAME is an a 'Data Structure' Represents the Data in 'Tabuler' (or) -
#   'Excel Spread Sheet' like 'Data'.
#(C) Whenever we Create Data Frame, We need to Declare 'Index'
```

In [23]:
```
# Now i want to prepare some small Data Frame :
```

In [24]:
```
from numpy.random import randn
```

In [84]:
```
# DataFrame -- 'D' and 'F' Capital.
# This is called as 'Data Frame'.
# Create an a Data Frame, by using Random Numbers of size of 5 by 4 (5,4)

df = pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])
df
```

Out[84]:

	W	X	Y	Z
A	-3.511005	0.665134	0.416017	0.019686
B	0.562341	0.558105	-0.431101	0.785004
C	1.557771	-1.267716	-0.528726	1.058957
D	1.329812	0.573643	0.944960	-1.162622
E	1.372157	-0.708813	-1.250706	-0.922913

COLLECTION OF (or) COMBINATION OF SERIES can called as 'df' :

page 241

In [29]:
```
# Individually i'm calling :

type(df ['W'])
```

Out[29]:
```
pandas.core.series.Series
```

In [30]:
```
type(df)
```

Out[30]:
```
pandas.core.frame.DataFrame
```

In [46]:
```
# Here 'df' means, The Entire Data will Come :

df
```

Out[46]:

	W	X	Y	Z
A	0.795903	-1.571015	-0.443312	-0.736873
B	0.563868	1.407798	-0.769244	-0.709620
C	-0.746885	-0.110760	-0.424851	0.105887
D	-1.695963	-0.648636	0.783839	-1.340966
E	-1.013000	0.810013	0.237945	1.964760

In [35]:
```
df['W']
```

Out[35]:
```
A    0.795903
B    0.563868
C   -0.746885
D   -1.695963
E   -1.013000
Name: W, dtype: float64
```

In [36]:
```
df['Z']
```

Out[36]:
```
A   -0.736873
B   -0.709620
C    0.105887
D   -1.340966
E    1.964760
Name: Z, dtype: float64
```

In [38]:
```
# It will throw because, Whenever calling an a '2' Dimensional representation of Data -
# We need to Declare Mandatorily Double Brases - [['W','X']].

df['W','X']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: ('W', 'X')

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

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15684\251720071.py in <module>
      2 # We need to Declare Mandatorily [['W','X']]
      3
----> 4 df['W','X']

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: ('W', 'X')
```

In [42]:
```
# Here, We gave Double Brases to the '2' Dimensional Representation of 'W' and 'X' :

df[['W','X']]
```

Out[42]:

	W	X
A	0.795903	-1.571015
B	0.563868	1.407798
C	-0.746885	-0.110760
D	-1.695963	-0.648636
E	-1.013000	0.810013

In [43]:
```
# df.W - In this way also, We can call it Directly :

df.W
```

Out[43]:
```
A    0.795903
B    0.563868
C   -0.746885
D   -1.695963
E   -1.013000
Name: W, dtype: float64
```

3. ADD A NEW COLUMN : DATA FRAMES :

PAGE 244

In [69]:
```
# Here, New COLUMN ADDED to a DATA FRAME :
# The New Data of 'W' + 'Y' added :
# This SUM Method is very very important :

df ['New'] = df['W'] + df['Y']
df
```

Out[69]:

	W	X	Y	Z	New
A	-1.179505	-0.155733	-0.060330	2.129718	-1.239835
B	0.787849	-0.626276	-0.155569	-2.367262	0.632280
C	1.560154	-1.426602	0.805180	1.136293	2.365334
D	1.389385	0.648558	0.466362	0.423214	1.855746
E	2.022955	0.834661	-1.069212	0.673099	0.953744

4. DROP (): DROP THE COLUMN FOR PARTICULAR TRANSACTION ONLY :

PAGE 245

In [61]:
```
# axis = 1 means, Working on COLUMNS :
# axis = 0 means, Working on ROWS :
```

In [70]:
```
df
```

Out[70]:

	W	X	Y	Z	New
A	-1.179505	-0.155733	-0.060330	2.129718	-1.239835
B	0.787849	-0.626276	-0.155569	-2.367262	0.632280
C	1.560154	-1.426602	0.805180	1.136293	2.365334
D	1.389385	0.648558	0.466362	0.423214	1.855746
E	2.022955	0.834661	-1.069212	0.673099	0.953744

In [62]:
```
# Here we 'dropped' the COLUMN - 'New'

df.drop('New',axis = 1)
```

Out[62]:

	W	X	Y	Z
A	-1.179505	-0.155733	-0.060330	2.129718
B	0.787849	-0.626276	-0.155569	-2.367262
C	1.560154	-1.426602	0.805180	1.136293
D	1.389385	0.648558	0.466362	0.423214
E	2.022955	0.834661	-1.069212	0.673099

5. DROP (or) DELETE COLUMN PERMANENTLY :

PAGE 246

In [71]:
```
# 'inplace = True' - There is a One Particular Parameter Called 'inplace = True' to Delete Column
#  permanently.

df.drop('New', axis = 1, inplace = True)
df
```

Out[71]:

	W	X	Y	Z
A	-1.179505	-0.155733	-0.060330	2.129718
B	0.787849	-0.626276	-0.155569	-2.367262
C	1.560154	-1.426602	0.805180	1.136293
D	1.389385	0.648558	0.466362	0.423214
E	2.022955	0.834661	-1.069212	0.673099

6. loc and iloc :

page 246

NOTE : loc and iloc :

In [78]:
```
# loc : Represents the Data from 'Row' to 'Index' pattern.

# iloc : Represents the Data from 'Row' to 'Column' pattern. But here we need to declare - 'Index No'.
```

(A) 'loc' : Transposing the Data from 'Row' to 'Index' :

In [77]:
```
# 'loc' : Transposing The Data from 'Row' to 'Index'
# means, 'Rows' become 'Columns'
```

In [72]:
```
# Now let me call 'df' first:

df
```

Out[72]:

	W	X	Y	Z
A	-1.179505	-0.155733	-0.060330	2.129718
B	0.787849	-0.626276	-0.155569	-2.367262
C	1.560154	-1.426602	0.805180	1.136293
D	1.389385	0.648558	0.466362	0.423214
E	2.022955	0.834661	-1.069212	0.673099

In [74]:
```
# Now, This particular 'C' Data Changed from 'Label' to 'Index'.

df.loc['C']
```

Out[74]:
```
W    1.560154
X   -1.426602
Y    0.805180
Z    1.136293
Name: C, dtype: float64
```

(B) 'iloc' : Index Number :

page 248

In [76]:
```
# iloc : Represents the Data from 'Row' to 'Column' pattern. But here we need to declare - 'Index No'.
# Here, Index Number '2' Means, 'C' - it is in the 2nd position in the 'Index':

df.iloc[2]
```

Out[76]:
```
W    1.560154
X   -1.426602
Y    0.805180
Z    1.136293
Name: C, dtype: float64
```

In [79]:
```
df
```

Out[79]:

	W	X	Y	Z
A	-1.179505	-0.155733	-0.060330	2.129718
B	0.787849	-0.626276	-0.155569	-2.367262
C	1.560154	-1.426602	0.805180	1.136293
D	1.389385	0.648558	0.466362	0.423214
E	2.022955	0.834661	-1.069212	0.673099

7. DROP (): DROP THE 'ROW' FOR PARTICULAR TRANSACTION ONLY : (OR) PERMANENTLY USING 'inplace = True'

page 249

In [81]:
```
df.drop('E', axis = 0)
```

Out[81]:

	W	X	Y	Z
A	-1.179505	-0.155733	-0.060330	2.129718
B	0.787849	-0.626276	-0.155569	-2.367262
C	1.560154	-1.426602	0.805180	1.136293
D	1.389385	0.648558	0.466362	0.423214

In [85]:
```
df.drop('E', axis = 0, inplace = True)
df
```

Out[85]:

	W	X	Y	Z
A	-3.511005	0.665134	0.416017	0.019686
B	0.562341	0.558105	-0.431101	0.785004
C	1.557771	-1.267716	-0.528726	1.058957
D	1.329812	0.573643	0.944960	-1.162622

In [94]:
```
# Here, We applied [df['W'] > 0], for that particular Column('W') only. and it is printing which is
# greater than > '0' of that particular Column 'W'

df[df['W'] > 0]
```

Out[94]:

	W	X	Y	Z
B	0.562341	0.558105	-0.431101	0.785004
C	1.557771	-1.267716	-0.528726	1.058957
D	1.329812	0.573643	0.944960	-1.162622

In [95]:
```
# it is printing which is less than < '0' of that particular Column 'W'

df[df['W'] < 0]
```

Out[95]:

	W	X	Y	Z
A	-3.511005	0.665134	0.416017	0.019686

In [99]:
```
df[df['W'] > 0 ] ['W']
```

Out[99]:
```
B    0.562341
C    1.557771
D    1.329812
Name: W, dtype: float64
```

In [106]:
```
# Here i want 'Y' value also along with 'W'

df[df['W'] > 0][['W','Y']]
```

Out[106]:

	W	Y
B	0.562341	-0.431101
C	1.557771	-0.528726
D	1.329812	0.944960

8. MULTI INDEX ( ) :

PAGE 251

'GROUPY' OR 'MULTI INDEX METHOD' :

In [116]:
```
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]

hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

In [117]:
```
# Now we need to create one particular Data Frame :
```

In [122]:
```
# Here, We have a 'Two Index'
# Multi Index in a Single Data

df = pd.DataFrame(randn(6,2),hier_index,['A','B'])
df
```

Out[122]:

		A	B
G1	**1**	-1.392848	-1.363109
	2	1.976930	0.372129
	3	-0.890870	0.079029
G2	**1**	2.014288	0.000531
	2	-1.180491	-0.377249
	3	-0.654705	-1.523372

Now, i want to get only Group 1(G1) Data

In [123]:
```
df.loc['G1']
```

Out[123]:

	A	B
1	-1.392848	-1.363109
2	1.976930	0.372129
3	-0.890870	0.079029

9. CREATE LABELS FOR INDEX :

PAGE 252

In [129]:
```
# Previously we don't have any Index Names, Now We Created 'Index Names(Groups, Nums)'

df.index.names = ['Groups','Nums']
df
```

Out[129]:

		A	B
Groups	**Nums**		
G1	**1**	-1.392848	-1.363109
	2	1.976930	0.372129
	3	-0.890870	0.079029
G2	**1**	2.014288	0.000531
	2	-1.180491	-0.377249
	3	-0.654705	-1.523372

In [131]:
```
# Here 'df' means the Entire Data.
# 'loc' of an a 'Group 2(G2)'
# In 'Group 2(G2)' - I Want '2nd Num' -> 'B Value'.


df.loc['G2'].loc[2]['B']
```

Out[131]:
```
-0.37724915558367816
```

10. MISSING DATA in DATA FRAMES :

PAGE 254

In [132]:
```
# Here How to Identify "Missing Data" :
# DATA FRAMES : Is Called as an a 'Structured Pattern of an a DATA' :
```

In [133]:
```
import numpy as np
import pandas as pd
```

In [138]:
```
# 'NaN' - 'Not Applicable Number'. very important thing we need to know in 'nan'.
# Generally in Other Languages (or) Other Pattern of Working, We Call it as an a 'Null Values'.
# Here, In Python - We Call it as an a 'Not Applicable Number(NaN)'
# {Key : Value}


d = {'A':[1,2,np.nan], 'B':[5,np.nan,np.nan], 'C':[1,2,3]}
d
```

Out[138]:
```
{'A': [1, 2, nan], 'B': [5, nan, nan], 'C': [1, 2, 3]}
```

11. Converting to Structured Data :

page 255

In [140]:
```
# The 'Data' We are Converting it to an a 'Structured Data' :

df = pd.DataFrame(d)
df
```

Out[140]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

12. REMOVE 'NUll VALUE' ::: 'dropna( )' :

page 255

In [142]:
```
# Here, Defaultly Effect On 'ROWS'.
# Here, '0' Value record onluy we don't have NaN's. But '1' record and '2' record have 'NaN Values'
# That's Why they were removed defaultly :


df.dropna()
```

Out[142]:

	A	B	C
0	1.0	5.0	1

13. REMOVE NULL(NaN) : COLUMN BASED :

PAGE 256

In [146]:
```
# 'dropna' - means, 'drop nulls' or 'Remove NaN'

df.dropna(axis = 1)
```

Out[146]:

	C
0	1
1	2
2	3

In [147]:
```
df.dropna(axis = 0)
```

Out[147]:

	A	B	C
0	1.0	5.0	1

14. FILL THE DATA IN NULL(NaN):

page 256

In [145]:
```
# Whenever We identify the Nulls(NaN's), There i want to fill the Data in Nulls(NaN's):
```

In [155]:
```
df['A'].fillna(value = 25)
```

Out[155]:
```
0     1.0
1     2.0
2    25.0
Name: A, dtype: float64
```

In [156]:
```
df
```

Out[156]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [162]:
```
df['B'].fillna(value = 50)
```

Out[162]:
```
0     5.0
1    50.0
2    50.0
Name: B, dtype: float64
```

In [163]:
```
df
```

Out[163]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

15. GROUP BY : MATHEMATICAL EXPRESSION :

PAGE 257

In [172]:
```
# Group By : eg: Whenever we call India Population roughly 120cr.
# Now, i want to split the Population into an a 'Statebase Population', Here i want to identify.
# So, i want to take something like - 'GROUP BY' into the Consideration.
```

In [173]:
```
data = {'COMPANY':['GOOG','GOOG','FB','FB','MSFT','MSFT'],
        'PERSON':['SAM','CHARLE','AMY','VANESSA','CARL','SARAH'],'SALES':[200,100,230,520,650,850]}

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

Out[173]:

	COMPANY	PERSON	SALES
0	GOOG	SAM	200
1	GOOG	CHARLE	100
2	FB	AMY	230
3	FB	VANESSA	520
4	MSFT	CARL	650
5	MSFT	SARAH	850

In [176]:
```
# bycomp - means, left hand variable (name)

bycomp = df.groupby('COMPANY')
```

In [180]:
```
# We '.mean()' - The 'Average' we are taking, eg : FB 230+520 = 750(Avg) = 375
# (Half Value to its Main Value)


bycomp.mean( )
```

Out[180]:

COMPANY	SALES
FB	375.0
GOOG	150.0
MSFT	750.0

In [184]:
```
# Here, We are doing 'SUMMATION' :

bycomp.sum()
```

Out[184]:

COMPANY	SALES
FB	750
GOOG	300
MSFT	1500

In [186]:
```
# STANDARD DIVIATION :

bycomp.std()
```

Out[186]:

COMPANY	SALES
FB	205.060967
GOOG	70.710678
MSFT	141.421356

In [188]:
```
# Here, i want to get a particular 'COMPANY' related stuff of 'FB' :

bycomp.sum().loc['FB']
```

Out[188]:
```
SALES    750
Name: FB, dtype: int64
```

In [189]:
```
df.groupby('COMPANY').sum().loc['FB']
```

Out[189]:
```
SALES    750
Name: FB, dtype: int64
```

In [190]:
```
# And How many Records We Want to find it Out :
# 'df' means Total Records :

df.groupby('COMPANY').count()
```

Out[190]:

COMPANY	PERSON	SALES
FB	2	2
GOOG	2	2
MSFT	2	2

In [191]:
```
# And What is an a 'MINIMUM METHOD' also We can find in the 'COMPANY' :


df.groupby('COMPANY').min()
```

Out[191]:

COMPANY	PERSON	SALES
FB	AMY	230
GOOG	CHARLE	100
MSFT	CARL	650

16. DESCRIBE METHOD() :

PAGE 260

In [193]:
```
# Now, it is going to be changed into an a particular working environment :

df.groupby('COMPANY').describe()
```

Out[193]:

	SALES							
	count	mean	std	min	25%	50%	75%	max
COMPANY								
FB	2.0	375.0	205.060967	230.0	302.5	375.0	447.5	520.0
GOOG	2.0	150.0	70.710678	100.0	125.0	150.0	175.0	200.0
MSFT	2.0	750.0	141.421356	650.0	700.0	750.0	800.0	850.0

17. DECLARING DATA MANUALLY :

PAGE 261

In [197]:
```
df1 = pd.DataFrame({"A":["A0","A1","A2","A3"],
                    "B":["B0","B1","B2","B3"],
                    "C":["C0","C1","C2","C3"],
                    "D":["D0","D1","D2","D3"]})
df2 = pd.DataFrame({"A":["A4","A5","A6","A7"],
                    "B":["B4","B5","B6","B7"],
                    "C":["C4","C5","C6","C7"],
                    "D":["D4","D5","D6","D7"]})
df3 = pd.DataFrame({"A":["A8","A9","A10","A11"],
                    "B":["B8","B9","B10","B11"],
                    "C":["C8","C9","C10","C11"],
                    "D":["D8","D9","D10","D11"]})
```

In [206]:
```
df1
```

Out[206]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [199]:
```
df2
```

Out[199]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [209]:
```
df1,df2,df3
```

Out[209]:
```
(    A   B   C   D
 0  A0  B0  C0  D0
 1  A1  B1  C1  D1
 2  A2  B2  C2  D2
 3  A3  B3  C3  D3,
     A   B   C   D
 0  A4  B4  C4  D4
 1  A5  B5  C5  D5
 2  A6  B6  C6  D6
 3  A7  B7  C7  D7,
      A    B    C    D
 0   A8   B8   C8   D8
 1   A9   B9   C9   D9
 2  A10  B10  C10  D10
 3  A11  B11  C11  D11)
```

18. Now 'CONCATING' THE DATA :

PAGE 262

In [210]:
```
# "Concat function concatenates dataframes along rows or columns".
# We can think of it as stacking up multiple dataframes.


# axis = 1 means, 'COLUMNS'. Here, The Data will exists 'Side by Side' - like - ABCD ABCD ABCD.

# axis = 0 means, 'ROWS'.
```

In [203]:
```
pd.concat([df1,df2,df3], axis = 1)
```

Out[203]:

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	A4	B4	C4	D4	A8	B8	C8	D8
1	A1	B1	C1	D1	A5	B5	C5	D5	A9	B9	C9	D9
2	A2	B2	C2	D2	A6	B6	C6	D6	A10	B10	C10	D10
3	A3	B3	C3	D3	A7	B7	C7	D7	A11	B11	C11	D11

In [204]:
```
pd.concat([df1,df2,df3], axis = 0)
```

Out[204]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7
0	A8	B8	C8	D8
1	A9	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

19. MERGING THE DATA :

PAGE 263 BLUE NOTEBOOK

In [221]:
```
# Merge combines dataframes based on values in shared columns.
# Merge function offers more flexibility compared to concat function because
# it allows combinations based on a condition.

# Merge : Merge Operation always depends upon -> In Both DatasSets -> It Should have an a 'key'.
# If we have an a 'key' also, something it need to be worked with 'inner' merge only, taken into-
# -consideration.
```

In [212]:
```
left = pd.DataFrame({'key':['ko','k1','k2','k3'],
                     "A":["A0","A1","A2","A3"],
                     "B":["B0","B1","B2","B3"]})
right = pd.DataFrame({'key':['ko','k1','k2','k3'],
                     "C":["C0","C1","C2","C3"],
                     "D":["D0","D1","D2","D3"]})
```

In [213]:
```
left
```

Out[213]:

	key	A	B
0	ko	A0	B0
1	k1	A1	B1
2	k2	A2	B2
3	k3	A3	B3

In [214]:
```
right
```

Out[214]:

	key	C	D
0	ko	C0	D0
1	k1	C1	D1
2	k2	C2	D2
3	k3	C3	D3

20. MERGE LEFT OF RIGHT : 'INNER OPERATION' :

PAGE 264

In [217]:
```
# Here, The 'inner merge' we are going to Call,
# 'on' What basis, we need to have something - 'key' should be 'mapped'.
# Without 'key mapping' - We can't do it.


pd.merge(left,right, how ='inner', on = 'key')
```

Out[217]:

	key	A	B	C	D
0	ko	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	C2	D2
3	k3	A3	B3	C3	D3

21. MERGE RIGHT OF LEFT : 'INNER OPERATION' :

PAGE 265

In [218]:
```
pd.merge(right,left, how = 'inner', on = 'key')
```

Out[218]:

	key	C	D	A	B
0	ko	C0	D0	A0	B0
1	k1	C1	D1	A1	B1
2	k2	C2	D2	A2	B2
3	k3	C3	D3	A3	B3

22. OUTER OPERATION :

PAGE 265

In [220]:
```
pd.merge(right,left, how = 'outer', on = 'key')
```

Out[220]:

	key	C	D	A	B
0	ko	C0	D0	A0	B0
1	k1	C1	D1	A1	B1
2	k2	C2	D2	A2	B2
3	k3	C3	D3	A3	B3

23. READ THE DATA IN THE 'PANDAS' :

PAGE 266

In [231]:
```
# Here, We read the 'Data' and We kept in the name - 'df3'

# A CSV (comma-separated values) file is a text file that has a specific format which allows
# data to be saved in a table structured format.

# What is import CSV file in Python?

# Source code: Lib/csv.py.
# The so-called CSV (Comma Separated Values) format is the most common import and export format
# for spreadsheets and databases.
# CSV format was used for many years prior to attempts to describe the format
# in a standardized way in RFC.


df3 = pd.read_csv("DataS/enterprise survey.csv")
df3
```

Out[231]:

	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable_name	Variable_cat
0	2021	Level 1	99999	All industries	Dollars (millions)	H01	Total income	Fir perfor
1	2021	Level 1	99999	All industries	Dollars (millions)	H04	Sales, government funding, grants and subsidies	Fir perfor
2	2021	Level 1	99999	All industries	Dollars (millions)	H05	Interest, dividends and donations	Fir perfor
3	2021	Level 1	99999	All industries	Dollars (millions)	H07	Non-operating income	Fir perfor
4	2021	Level 1	99999	All industries	Dollars (millions)	H08	Total expenditure	Fir perfor
...	...	...	...	...	...	...	...	
41710	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H37	Quick ratio	Financia
41711	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H38	Margin on sales of goods for resale	Financia
41712	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H39	Return on equity	Financia
41713	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H40	Return on total assets	Financia
41714	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H41	Liabilities structure	Financia

41715 rows × 10 columns

24. WRITE THE DATA IN THE 'PANDAS' : VVIMP

PAGE 266

In [232]:
```
# Here, We created new file 'LIC2.csv' and Saved the data of 'df3'.


df3.to_csv("LIC2.csv")
```

25. REMOVE INDEX : VVIMP

PAGE 266

In [230]:
```
# Here, Again We created new file 'LIC5.csv' and Saved the data of 'df3',
# and 'Removed' the 'INDEX Numbers' by using -> 'index = False'.


df3.to_csv("LIC5.csv", index = False)
```

26. Excel Sheet Reading :

page 267

In [234]:
```
# pd.read_excel("File_name", sheet_name = 'sheet1')

# 'sheet_name' is compulsary
```

27. READING 'HTML' DATA :

PAGE 268

In [235]:
```
# Now, i'm taking 'Failed Bank List in India' from chrome and i'm, going to work on this by using
# 'HTML' . Taking(copied) link of Certain Bank list.

data = pd.read_html("https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list/")
```

In [236]:
```
type(data)
```

Out[236]:
```
list
```

In [237]:
```
# Now 'Data' of an a '0' position dot. head method() :

data[0].head()
```

Out[237]:

	Bank NameBank	CityCity	StateSt	CertCert	Acquiring InstitutionAI	Closing DateClosing	FundFund
0	First Republic Bank	San Francisco	CA	59017	JPMorgan Chase Bank, N.A.	May 1, 2023	10543
1	Signature Bank	New York	NY	57053	Flagstar Bank, N.A.	March 12, 2023	10540
2	Silicon Valley Bank	Santa Clara	CA	24735	First–Citizens Bank & Trust Company	March 10, 2023	10539
3	Almena State Bank	Almena	KS	15426	Equity Bank	October 23, 2020	10538
4	First City Bank of Florida	Fort Walton Beach	FL	16748	United Fidelity Bank, fsb	October 16, 2020	10537

In [238]:
```
data
```

Out[238]:
```
[                        Bank NameBank           CityCity StateSt  CertCert  \
 0                  First Republic Bank      San Francisco      CA     59017
 1                       Signature Bank           New York      NY     57053
 2                  Silicon Valley Bank        Santa Clara      CA     24735
 3                    Almena State Bank             Almena      KS     15426
 4            First City Bank of Florida  Fort Walton Beach      FL     16748
 ..                                  ...                ...     ...       ...
 561                 Superior Bank, FSB           Hinsdale      IL     32646
 562                Malta National Bank              Malta      OH      6629
 563    First Alliance Bank & Trust Co.         Manchester      NH     34264
 564  National State Bank of Metropolis         Metropolis      IL      3815
 565                   Bank of Honolulu           Honolulu      HI     21029

                  Acquiring InstitutionAI Closing DateClosing  FundFund
 0              JPMorgan Chase Bank, N.A.         May 1, 2023     10543
 1                   Flagstar Bank, N.A.      March 12, 2023     10540
 2    First-Citizens Bank & Trust Company      March 10, 2023     10539
 3                             Equity Bank    October 23, 2020     10538
 4              United Fidelity Bank, fsb    October 16, 2020     10537
 ..                                   ...                ...       ...
 561                Superior Federal, FSB       July 27, 2001      6004
 562                    North Valley Bank         May 3, 2001      4648
 563  Southern New Hampshire Bank & Trust    February 2, 2001      4647
 564              Banterra Bank of Marion   December 14, 2000      4646
 565                   Bank of the Orient    October 13, 2000      4645

 [566 rows x 7 columns]]
```

In [ ]:
```

```