TE Computer Pandas Important Function

Pandas

- Pandas is a Python library. Pandas is used to analyze data.
- Called as Python Data Analysis ---> pandas

In [1]:

```
import pandas as pd
```

```
Pandas has two Data Structures :-
    - 1. DataFrame :- (2D --> Row and Col)
    - 2. Series :- (1D --> A single col of DataFrame)
```

```
DataFrame :  two-dimensional data structure with columns, much like a table.
Series : one-dimensional labeled arrays pd.Series(data)
```

```
DataFrame :-
- Use Cases :-
    - CSV/Excel/JSON file reading
    - EDA
    - Data Cleaning
    - Encoding Technique
    - Joining Tables(left, right, inner, outer)
    - Feature selection
```

```
We can Create DataFrame using following
    - list
    - numpy array
    - dict
    - series
    - CSV/JSON/Excel file
```

Empty DataFrame and series

In [2]:

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

Out[2]:

```
pandas.core.frame.DataFrame
```

In [3]:

```
df.empty  # Check an empty DataFrame
```

Out[3]:

```
True
```

In [4]:

```
sr=pd.Series()
type(sr)
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_8864\4217705239.py:1: Future
Warning: The default dtype for empty Series will be 'object' instead of 'f
loat64' in a future version. Specify a dtype explicitly to silence this wa
rning.
  sr=pd.Series()
```

Out[4]:

```
pandas.core.series.Series
```

In [5]:

```
sr.empty # Check an empty Series
```

Out[5]:

```
True
```

How to read a csv/Excel file

In [6]:

```
df_csv = pd.read_csv("Heart_Data.csv")  # Reading CSV file
df_csv
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	

303 rows × 14 columns

In [8]:

```
df_excel = pd.read_excel("Employee_Records.xlsx")
df_excel
```

Out[8]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60	13.68	168251	Denver
1	940761	Brenda	47.02	60	9.01	51063	Stonewall
2	428945	Joe	54.15	68	0.98	50155	Michigantown
3	408351	Diane	39.67	51	18.30	180294	Hydetown
4	193819	Benjamin	40.31	58	4.01	117642	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22.82	89	1.05	129774	Biloxi
96	704709	Harold	32.61	77	5.93	156194	Carol Stream
97	461593	Nicole	52.66	60	28.53	95673	Detroit
98	392491	Theresa	29.60	57	6.99	51015	Mc Grath
99	495141	Tammy	38.38	55	2.26	93650	Alma

100 rows × 7 columns

Important functions of Pandas

1. df.shape :

- Return a tuple representing the dimensionality of the DataFrame.

In [9]:

```
df_excel.shape
```

Out[9]:

```
(100, 7)
```

2. df.columns :

- The column labels of the DataFrame.

In [10]:

```
df_excel.columns
```

Out[10]:

```
Index(['Emp ID', 'First Name', 'Age in Yrs', 'Weight in Kgs', 'Age in Comp
any',
       'Salary', 'City'],
      dtype='object')
```

3. df.index :

- The index (row labels) of the DataFrame.

In [11]:

```
df_excel.index
```

Out[11]:

```
RangeIndex(start=0, stop=100, step=1)
```

4. df.dtypes :

- This returns a Series with the data type of each column.

In [12]:

```
df_excel.dtypes
```

Out[12]:

```
Emp ID              int64
First Name         object
Age in Yrs        float64
Weight in Kgs       int64
Age in Company    float64
Salary              int64
City               object
dtype: object
```

5. df.info() :

- This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

In [13]:

```
df_excel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Emp ID          100 non-null    int64  
 1   First Name      100 non-null    object 
 2   Age in Yrs      100 non-null    float64
 3   Weight in Kgs   100 non-null    int64  
 4   Age in Company  100 non-null    float64
 5   Salary          100 non-null    int64  
 6   City            100 non-null    object 
dtypes: float64(2), int64(3), object(2)
memory usage: 5.6+ KB
```

6. df.describe() :

- Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

In [14]:

```
df_excel.describe()
```

Out[14]:

	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary
count	100.00000	100.000000	100.000000	100.000000	100.000000
mean	547652.10000	39.238700	58.080000	8.978400	119738.090000
std	257664.16679	12.066252	12.294106	8.657358	46185.278194
min	134841.00000	21.100000	40.000000	0.020000	42005.000000
25%	328643.75000	28.177500	50.000000	2.152500	83979.750000
50%	497414.00000	37.595000	56.000000	6.435000	118049.500000
75%	766040.00000	49.900000	61.250000	13.762500	162509.250000
max	979607.00000	59.470000	90.000000	34.520000	197537.000000

7. df.head() :

- This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

In [15]:

```
df_excel.head()
```

Out[15]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60	13.68	168251	Denver
1	940761	Brenda	47.02	60	9.01	51063	Stonewall
2	428945	Joe	54.15	68	0.98	50155	Michigantown
3	408351	Diane	39.67	51	18.30	180294	Hydetown
4	193819	Benjamin	40.31	58	4.01	117642	Fremont

8. df.tail() :

- Return the last n rows.

In [16]:

```
df_excel.tail()
```

Out[16]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
95	639892	Jose	22.82	89	1.05	129774	Biloxi
96	704709	Harold	32.61	77	5.93	156194	Carol Stream
97	461593	Nicole	52.66	60	28.53	95673	Detroit
98	392491	Theresa	29.60	57	6.99	51015	Mc Grath
99	495141	Tammy	38.38	55	2.26	93650	Alma

9. df.sort_values(colomn_name) :

- Sort by the values along either axis.

In [17]:

```
df_excel.sort_values(by = "Age in Yrs" )
```

Out[17]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
13	301576	Wayne	21.10	87	0.02	92758	Maida
49	879753	Pamela	21.30	47	0.13	149262	Banner
59	750173	Antonio	21.93	82	0.24	181646	Mc Calla
6	539712	Nancy	22.14	50	0.87	98189	Atlanta
11	153989	Jack	22.21	61	0.56	82965	Las Vegas
...	...	...	...	...	...	...	...
74	528673	Paul	58.43	60	22.10	145235	Blue River
7	380086	Carol	59.12	40	34.52	60918	Blanchester
57	515103	Anne	59.27	48	14.01	114426	Cookeville
24	560455	Carolyn	59.42	53	16.08	42005	Saint Cloud
14	441771	Cheryl	59.47	47	26.69	92220	Quecreek

100 rows × 7 columns

10. df.reset_index() :

- Reset the index, or a level of it.

In [18]:

```
df_excel.reset_index()
```

Out[18]:

	index	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	0	677509	Lois	36.36	60	13.68	168251	Denver
1	1	940761	Brenda	47.02	60	9.01	51063	Stonewall
2	2	428945	Joe	54.15	68	0.98	50155	Michigantown
3	3	408351	Diane	39.67	51	18.30	180294	Hydetown
4	4	193819	Benjamin	40.31	58	4.01	117642	Fremont
...	...	...	...	...	...	...	...	...
95	95	639892	Jose	22.82	89	1.05	129774	Biloxi
96	96	704709	Harold	32.61	77	5.93	156194	Carol Stream
97	97	461593	Nicole	52.66	60	28.53	95673	Detroit
98	98	392491	Theresa	29.60	57	6.99	51015	Mc Grath
99	99	495141	Tammy	38.38	55	2.26	93650	Alma

100 rows × 8 columns

11. df.isna()

- To check null values in dataframe

In [19]:

```
df_excel.isna()
```

Out[19]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
95	False	False	False	False	False	False	False
96	False	False	False	False	False	False	False
97	False	False	False	False	False	False	False
98	False	False	False	False	False	False	False
99	False	False	False	False	False	False	False

100 rows × 7 columns

In [20]:

```
df_excel.isna().sum() # sum of null values
```

Out[20]:

```
Emp ID            0
First Name        0
Age in Yrs        0
Weight in Kgs     0
Age in Company    0
Salary            0
City              0
dtype: int64
```

12.df.count()

- Count non-NA cells for each column or row.

In [21]:

```
df_excel.count()
```

Out[21]:

```
Emp ID            100
First Name        100
Age in Yrs        100
Weight in Kgs     100
Age in Company    100
Salary            100
City              100
dtype: int64
```

13. df.any()

```
Returns False unless there is at least one element within a series or along a Dataframe
axis that is True or equivalent (e.g. non-zero or non-empty).
```

In [24]:

```
df_excel.any()
```

Out[24]:

```
Emp ID            True
First Name        True
Age in Yrs        True
Weight in Kgs     True
Age in Company    True
Salary            True
City              True
dtype: bool
```

In [25]:

```
df_excel.any().any()
```

Out[25]:

```
True
```

14. df.axes() :

- Return a list representing the axes of the DataFrame.

In [39]:

```
df_excel.axes
```

Out[39]:

```
[RangeIndex(start=0, stop=100, step=1),
 Index(['Emp ID', 'First Name', 'Age in Yrs', 'Weight in Kgs', 'Age in Com
pany',
        'Salary', 'City'],
       dtype='object')]
```

15. df.empty :

- Indicator whether Series/DataFrame is empty.

In [44]:

```
df_excel.empty
```

Out[44]:

```
False
```

16. df.dtypes

- Return the dtypes in the DataFrame.

In [43]:

```
df_excel.dtypes
```

Out[43]:

```
Emp ID              int64
First Name         object
Age in Yrs        float64
Weight in Kgs       int64
Age in Company    float64
Salary              int64
City               object
dtype: object
```

17. df.astype() :

- Cast a pandas object to a specified dtype dtype.

In [4]:

```
df_excel=df_excel.astype({"Age in Yrs": 'int32'})
df_excel
```

Out[4]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36	60	13.68	168251	Denver
1	940761	Brenda	47	60	9.01	51063	Stonewall
2	428945	Joe	54	68	0.98	50155	Michigantown
3	408351	Diane	39	51	18.30	180294	Hydetown
4	193819	Benjamin	40	58	4.01	117642	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22	89	1.05	129774	Biloxi
96	704709	Harold	32	77	5.93	156194	Carol Stream
97	461593	Nicole	52	60	28.53	95673	Detroit
98	392491	Theresa	29	57	6.99	51015	Mc Grath
99	495141	Tammy	38	55	2.26	93650	Alma

100 rows × 7 columns

In [6]:

```
df_excel.dtypes # You can see the diff in column "Age in Yrs"
```

Out[6]:

```
Emp ID              int64
First Name         object
Age in Yrs          int32
Weight in Kgs       int64
Age in Company    float64
Salary              int64
City               object
dtype: object
```

18. DataFrame.loc[row, columns]

- Access a group of rows and columns by label(s) or a boolean array.

In [9]:

```
df_excel.loc[2, "Emp ID"]
```

Out[9]:

```
428945
```

In [13]:

```
df_excel.loc[[2,3], ["Emp ID","First Name"]]
```

Out[13]:

	Emp ID	First Name
2	428945	Joe
3	408351	Diane

19. DataFrame.iloc[row_index, col_index]

- Purely integer-location based indexing for selection by position.

In [12]:

```
df_excel.iloc[2, 0]
```

Out[12]:

```
428945
```

In [14]:

```
df_excel.iloc[[2,3], [0,1]]
```

Out[14]:

	Emp ID	First Name
2	428945	Joe
3	408351	Diane

20. df.insert(loc, column, value, allow_duplicates=_NoDefault.no_default)

- Insert column into DataFrame at specified location.

In [15]:

```
df = pd.DataFrame(
{"Name" : ["Sandeep", "Pradeep"]

})
```

In [16]:

```
df
```

Out[16]:

	Name
0	Sandeep
1	Pradeep

In [17]:

```
df.insert(1, "Surname", ["Shukla", "Shukla"])
```

In [18]:

```
df
```

Out[18]:

	Name	Surname
0	Sandeep	Shukla
1	Pradeep	Shukla

21. df.drop(labels=None, *, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

- Drop specified labels from rows or columns.

In [21]:

```
df_excel
```

Out[21]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36	60	13.68	168251	Denver
1	940761	Brenda	47	60	9.01	51063	Stonewall
2	428945	Joe	54	68	0.98	50155	Michigantown
3	408351	Diane	39	51	18.30	180294	Hydetown
4	193819	Benjamin	40	58	4.01	117642	Fremont
...	...	...	...	...	...	...	...
95	639892	Jose	22	89	1.05	129774	Biloxi
96	704709	Harold	32	77	5.93	156194	Carol Stream
97	461593	Nicole	52	60	28.53	95673	Detroit
98	392491	Theresa	29	57	6.99	51015	Mc Grath
99	495141	Tammy	38	55	2.26	93650	Alma

100 rows × 7 columns

In [38]:

```
df_excel=df_excel.drop(["First Name", "City"], axis=1)
```

In [39]:

```
df_excel
```

Out[39]:

	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary
0	677509	36.36	60	13.68	168251
1	940761	47.02	60	9.01	51063
2	428945	54.15	68	0.98	50155
3	408351	39.67	51	18.30	180294
4	193819	40.31	58	4.01	117642
...	...	...	...	...	...
95	639892	22.82	89	1.05	129774
96	704709	32.61	77	5.93	156194
97	461593	52.66	60	28.53	95673
98	392491	29.60	57	6.99	51015
99	495141	38.38	55	2.26	93650

100 rows × 5 columns

22. df.sum()

- Return the sum of the values over the requested axis.

In [40]:

```
df_excel.sum(axis = 0)
```

Out[40]:

```
Emp ID            54765210.00
Age in Yrs            3923.87
Weight in Kgs         5808.00
Age in Company         897.84
Salary             11973809.00
dtype: float64
```

In [41]:

```
df_excel.sum(axis = 1)
```

Out[41]:

```
0      845870.04
1      991940.03
2      479223.13
3      588753.97
4      311563.32
         ...
95     769778.87
96     861018.54
97     557407.19
98     443599.59
99     588886.64
Length: 100, dtype: float64
```

23. df.dropna(*, axis=0, how=_NoDefault.no_default, thresh=_NoDefault.no_default, subset=None, inplace=False)

- Remove missing values (if any)

In [42]:

```
df_excel.dropna()
```

Out[42]:

	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary
0	677509	36.36	60	13.68	168251
1	940761	47.02	60	9.01	51063
2	428945	54.15	68	0.98	50155
3	408351	39.67	51	18.30	180294
4	193819	40.31	58	4.01	117642
...	...	...	...	...	...
95	639892	22.82	89	1.05	129774
96	704709	32.61	77	5.93	156194
97	461593	52.66	60	28.53	95673
98	392491	29.60	57	6.99	51015
99	495141	38.38	55	2.26	93650

100 rows × 5 columns

24. df.count(axis=0, level=None, numeric_only=False)

- Count non-NA cells for each column or row.

In [43]:

```
df_excel.count()
```

Out[43]:

```
Emp ID            100
Age in Yrs        100
Weight in Kgs     100
Age in Company    100
Salary            100
dtype: int64
```

25. df.value_counts(subset=None, normalize=False, sort=True, ascending=False, dropna=True)

- Return a Series containing counts of unique rows in the DataFrame.

In [44]:

```
df_excel.value_counts()
```

Out[44]:

```
Emp ID  Age in Yrs  Weight in Kgs  Age in Company  Salary
134841  33.89       57             4.03            129836    1
639892  22.82       89             1.05            129774    1
761821  32.77       87             2.49            176675    1
758872  32.75       70             0.10            102384    1
750173  21.93       82             0.24            181646    1
                                                           ..
386158  45.45       55             1.68            166892    1
380086  59.12       40             34.52           60918     1
363065  23.73       60             1.78            61924     1
335732  53.68       41             29.96           60508     1
979607  22.64       56             1.26            93967     1
Length: 100, dtype: int64
```

26. df.to_dict(orient='dict', into=<class 'dict'>)

- Convert the DataFrame to a dictionary.

27.df.nunique(axis=0, dropna=True)

- Count number of distinct elements in specified axis.

28.df.rename(mapper=None, *, index=None, columns=None, axis=None, copy=None, inplace=False, level=None, errors='ignore')

- Alter axes labels.

In [69]:

```
df = pd.DataFrame(
{"Name" : ["Sandeep", "Pradeep"]
 })
df
```

Out[69]:

	Name
0	Sandeep
1	Pradeep

In [70]:

```
df.rename(columns ={"Name":"First Name"}, index={0:"A", 1:"B"})
```

Out[70]:

	First Name
A	Sandeep
B	Pradeep

29. df.replace(to_replace=None, value=_NoDefault.no_default, *, inplace=False, limit=None, regex=False, method=_NoDefault.no_default)

- Replace values given in to_replace with value.

In [71]:

```
df = df.replace({"Sandeep":"Jaydeep"})
df
```

Out[71]:

	Name
0	Jaydeep
1	Pradeep

30. df.apply(func, axis=0, raw=False, result_type=None, args=(), **kwargs)

- Apply a function along an axis of the DataFrame.

In [73]:

```
df["Name"] = df["Name"].apply(lambda x: x.upper())
df
```

Out[73]:

	Name
0	JAYDEEP
1	PRADEEP

31. df..groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=_NoDefault.no_default, squeeze=_NoDefault.no_default, observed=False, dropna=True)

- Group DataFrame using a mapper or by a Series of columns.

In [74]:

```
df_excel
```

Out[74]:

	Emp ID	Age in Yrs	Weight in Kgs	Age in Company	Salary
0	677509	36.36	60	13.68	168251
1	940761	47.02	60	9.01	51063
2	428945	54.15	68	0.98	50155
3	408351	39.67	51	18.30	180294
4	193819	40.31	58	4.01	117642
...	...	...	...	...	...
95	639892	22.82	89	1.05	129774
96	704709	32.61	77	5.93	156194
97	461593	52.66	60	28.53	95673
98	392491	29.60	57	6.99	51015
99	495141	38.38	55	2.26	93650

100 rows × 5 columns

In [77]:

```
df_excel.groupby('Weight in Kgs')
```

Out[77]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000027D415FEAD0
>
```

In [78]:

```
df_excel.groupby('Weight in Kgs').first()
```

Out[78]:

Weight in Kgs	Emp ID	Age in Yrs	Age in Company	Salary
40	380086	59.12	34.52	60918
41	474599	42.39	18.84	48944
42	477616	58.18	23.27	121587
43	253573	26.25	3.39	190139
44	766610	55.97	7.21	119321
45	923947	55.14	13.20	155442
47	441771	59.47	26.69	92220
48	515103	59.27	14.01	114426
49	329752	36.24	2.29	67251
50	539712	22.14	0.87	98189
51	408351	39.67	18.30	180294
52	893212	36.14	14.17	112715
53	560455	59.42	16.08	42005
54	333476	49.69	25.62	109394
55	386158	45.45	1.68	166892
56	890290	58.08	12.43	141518
57	134841	33.89	4.03	129836
58	193819	40.31	4.01	117642
59	214352	24.66	2.52	197537
60	677509	36.36	13.68	168251
61	153989	22.21	0.56	82965
62	456747	26.24	4.93	170895
64	621833	23.92	1.83	169245
65	802554	24.01	2.91	106628
68	428945	54.15	0.98	50155
70	818384	34.75	7.67	173226
73	622406	49.85	19.15	73862
74	917937	25.93	1.22	163560
75	958326	28.76	0.43	97904
77	683826	48.09	8.50	129625
80	231469	42.50	8.29	118457
81	278556	24.02	0.94	122226
82	750173	21.93	0.24	181646
87	301576	21.10	0.02	92758
89	639892	22.82	1.05	129774
90	726264	43.63	10.14	162159

32. df.append(other, ignore_index=False, verify_integrity=False, sort=False)

- Append rows of other to the end of caller, returning a new object.

In [82]:

```
df1 = pd.DataFrame(
    {
        "Names" : ["Ajay","Suraj","Ganesh","Sanket"],
        "Age" : [23,54,34,25],
        "Location" : ["Pune","Mumbai","Delhi","Chennai"]
    }
)

df2 = pd.DataFrame(
    {
        "Names" : ["Akshay","Pushkar","Soham","Rohit"],
        "Age" : [22,24,33,32],
        "Location" : ["Pune","Mumbai","Delhi","Chennai"]
    }
)
df1.append(df2)
```

```
C:\Users\Sandeep Shukla\AppData\Local\Temp\ipykernel_3904\4068845963.py:1
6: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df1.append(df2)
```

Out[82]:

	Names	Age	Location
0	Ajay	23	Pune
1	Suraj	54	Mumbai
2	Ganesh	34	Delhi
3	Sanket	25	Chennai
0	Akshay	22	Pune
1	Pushkar	24	Mumbai
2	Soham	33	Delhi
3	Rohit	32	Chennai

33. df.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False, validate=None)

- Join columns of another DataFrame.

In [85]:

```
df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'], 'A': ['A0', 'A1', 'A2', '
other = pd.DataFrame({'key': ['K0', 'K1', 'K2'], 'B': ['B0', 'B1', 'B2']})
df.join(other, lsuffix='_caller', rsuffix='_other')
```

Out[85]:

	key_caller	A	key_other	B
0	K0	A0	K0	B0
1	K1	A1	K1	B1
2	K2	A2	K2	B2
3	K3	A3	NaN	NaN
4	K4	A4	NaN	NaN
5	K5	A5	NaN	NaN

34. df.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

- Merge DataFrame or named Series objects with a database-style join.

In [84]:

```
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'], 'value': [1, 2, 3, 5]})
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'], 'value': [5, 6, 7, 8]})
df1.merge(df2, left_on='lkey', right_on='rkey')
```

Out[84]:

	lkey	value_x	rkey	value_y
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

35. pandas.concat(objs, *, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=False, copy=True)

- Concatenate pandas objects along a particular axis.

In [83]:

```
pd.concat([df1,df2])
```

Out[83]:

	Names	Age	Location
0	Ajay	23	Pune
1	Suraj	54	Mumbai
2	Ganesh	34	Delhi
3	Sanket	25	Chennai
0	Akshay	22	Pune
1	Pushkar	24	Mumbai
2	Soham	33	Delhi
3	Rohit	32	Chennai

In [ ]:

```

```