Pandas Data Frames

Pandas Series

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

Example

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

Example

Return the first value of the Series:

```
print(myvar[0])
```

Create Labels

With the `index` argument, you can name your own labels.

Example

Create you own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

Accessing Elements

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

Example

Return the value of "y":

```
print(myvar["y"])
```

Key/Value Objects as Series

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

Example

Create a simple Pandas Series from a dictionary:

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

Accessing Dictionary Values

- 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.

Example

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

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

Data Frames

- Data sets in Pandas are usually **multi-dimensional tables**, called Data Frames.
- Series is like a column, a Data Frame is the whole table.

Example

Create a DataFrame from two Series:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

What is a Data Frame?

- A Pandas Data Frame is a **2 dimensional data structure**, like a 2 dimensional array, or a table with rows and columns.

Example

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Output

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

- As you can see from the result above, the DataFrame is like a table with rows and columns.
- Pandas use the **loc** attribute to return one or more specified row(s)

Example

Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Output

```
calories    420
duration     50
Name: 0, dtype: int64
```

Example2

Return row 0 and 1:

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

Output

```
   calories  duration
0       420        50
1       380        40
```

Named Indexes

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

Example

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

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

Output:

	calories	duration
day1	420	50
day2	380	40
day3	390	45

Locate Named Indexes

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

Example

Return "day2":

```
#refer to the named index:
print(df.loc["day2"])
```

Output

```
calories    380
duration     40
Name: 0, dtype: int64
```

Load Files Into a DataFrame

- If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well known format that can be read by everyone including Pandas.

Example

Load the CSV into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

- By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows

Introduction

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- A pandas DataFrame can be created using the following constructor
- `pandas.DataFrame( data, index, columns, dtype, copy)`

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.
5	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

Create a DataFrame

- A pandas DataFrame can be created using various inputs like –
  - Lists
  - dict
  - Series
  - Numpy ndarrays
  - Another DataFrame
- Create an Empty DataFrame
- A basic DataFrame, which can be created is an Empty Dataframe.
- Example

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

Its **output** is as follows –

```
Empty DataFrame
Columns: []
Index: []
```

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists

Example 1

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

	0
0	1
1	2
2	3
3	4
4	5

Example 2

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
```

Its **output** is as follows −

	Name	Age
0	Alex	10
1	Bob	12
2	Clarke	13

Its **output** is as follows −

	Name	Age
0	Alex	10.0
1	Bob	12.0
2	Clarke	13.0

Create a Data Frame from Dictionary of n-d arrays / Lists

- All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.
- If no index is passed, then by default, index will be range(n), where **n** is the array length.
- Example:

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
      Age      Name
0     28        Tom
1     34       Jack
2     29      Steve
3     42      Ricky
```

- Let us now create an indexed DataFrame using arrays.

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4’])
print df
```

output

	Age	Name
rank1	28	Tom
rank2	34	Jack
rank3	29	Steve
rank4	42	Ricky

- List of Dictionaries can be passed as input data to create a DataFrame.
- The dictionary keys are by default taken as column names.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

	a	b	c
0	1	2	NaN
1	5	10	20.0

- The following example shows how to create a DataFrame by passing a list of dictionaries and the row indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second’])
print df
```

	a	b	c
first	1	2	NaN
second	5	10	20.0

- The following example shows how to create a DataFrame with a list of dictionaries, row indices, and column indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b’])
 #With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1’])
print df1
 print df2
```

output

```
#df1 output
          a  b
first     1  2
second    5  10

#df2 output
          a  b1
first     1  NaN
second    5  NaN
```

- Create a DataFrame from Dict of Series
- Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c’]),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df
```

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Data Frames: Columns

- selecting a column from the DataFrame.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df ['one']
```

output

```
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
```

Data Frames: Columns

- Adding a new column to an existing data frame.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd’])}
df = pd.DataFrame(d)
# Adding a new column to an existing DataFrame object with column label by passing new series
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c’])
print df
print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three’]
print df
```

output

```
Adding a new column by passing as Series:
     one   two   three
a    1.0    1    10.0
b    2.0    2    20.0
c    3.0    3    30.0
d    NaN    4    NaN

Adding a new column using the existing columns in DataFrame:
      one   two   three    four
a     1.0    1    10.0     11.0
b     2.0    2    20.0     22.0
c     3.0    3    30.0     33.0
d     NaN    4     NaN     NaN
```

Data Frames: Columns

- Columns can be deleted or popped;

```
# Using the previous DataFrame, we will delete a column using del function
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30], index=['a','b','c'])}
df = pd.DataFrame(d)
print ("Our dataframe is:")
print df
# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print df
# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print df
```

output

```
Our dataframe is:
      one   three  two
a     1.0    10.0   1
b     2.0    20.0   2
c     3.0    30.0   3
d     NaN     NaN   4

Deleting the first column using DEL function:
      three    two
a     10.0     1
b     20.0     2
c     30.0     3
d     NaN      4

Deleting another column using POP function:
   three
a  10.0
b  20.0
c  30.0
d  NaN
```

DataFrames : Row Selection

- Rows can be selected by passing **row label** to a **loc** function.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df.loc['b']
```

output

```
one 2.0
two 2.0
Name: b, dtype: float64
```

- Rows can be selected by passing **integer location** to an iloc function.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd’])}
df = pd.DataFrame(d)
print df.iloc[2]
```

```
one   3.0
two   3.0
Name: c, dtype: float64
```

- Multiple rows can be selected using ‘ **:** ’ operator.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print df[2:4]
```

	one	two
c	3.0	3
d	NaN	4

DataFrames : Addition, and Deletion

Addition of Rows

- Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b’])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b’])
df = df.append(df2)
print df
```

output

	a	b
0	1	2
1	3	4
0	5	6
1	7	8

Deletion of Rows

- Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.
- If you observe, in the above example, the labels are duplicate. Let us drop a label and will see how many rows will get dropped.

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b’])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b’])
df = df.append(df2)
# Drop rows with label 0
df = df.drop(0)
print df
```

	a	b
1	3	4
1	7	8

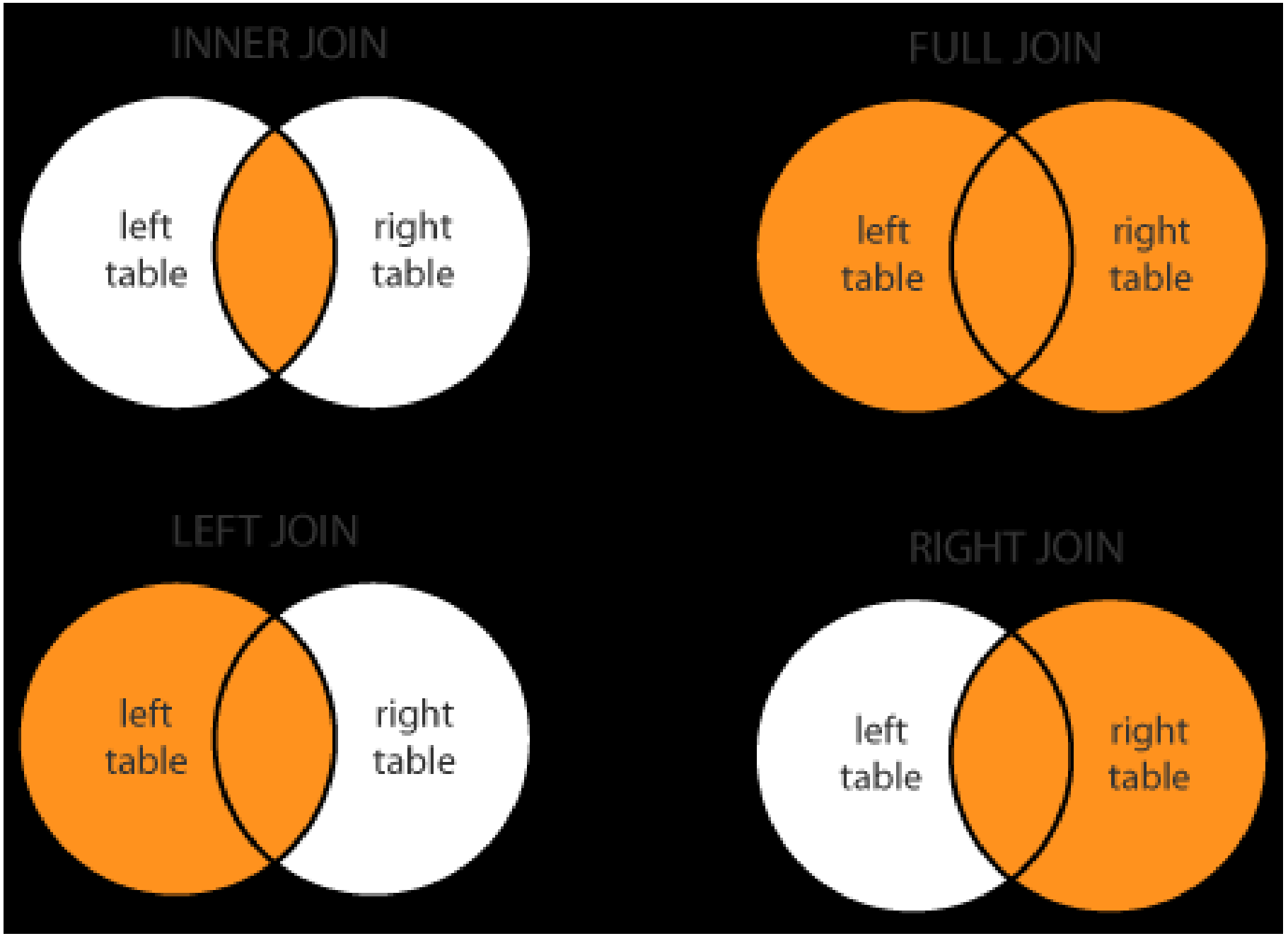
Merging/Joining

- Pandas provides a single function, **merge**, as the entry point for all standard database join operations between DataFrame objects

```
pd.merge(left_df, right_df, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True)
```

- **left** − A DataFrame object.
- **right** − Another DataFrame object.
- **on** − Columns (names) to join on. Must be found in both the left and right DataFrame objects.
- **left_on** − Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.
- **right_on** − Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.
- **left_index** − If **True,** use the index (row labels) from the left DataFrame as its join key(s). In case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame.
- **right_index** − If **True,** use the index (row labels) from the Right DataFrame as its join key(s). In case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the left DataFrame.
- **how** − One of 'left', 'right', 'outer', 'inner'. Defaults to inner. Each method has been described below.
- **sort** − Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve the performance substantially in many cases.

Types of Joins



Merging/Joining

Let us now create two different DataFrames and perform the merging operations on it.

```
# import the pandas library
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame(
   {'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print left
print right
```

```
    Name  id   subject_id
0   Alex   1         sub1
1    Amy   2         sub2
2  Allen   3         sub4
3  Alice   4         sub6
4  Ayoung  5         sub5

    Name  id   subject_id
0  Billy   1         sub2
1  Brian   2         sub4
2  Bran    3         sub3
3  Bryce   4         sub6
4  Betty   5         sub5
```

Merging/Joining

- Merge Two DataFrames on a Key

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
        'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on='id')
```

	Name_x	id	subject_id_x	Name_y	subject_id_y
0	Alex	1	sub1	Billy	sub2
1	Amy	2	sub2	Brian	sub4
2	Allen	3	sub4	Bran	sub3
3	Alice	4	sub6	Bryce	sub6
4	Ayoung	5	sub5	Betty	sub5

Merging/Joining

- Merge Two DataFrames on Multiple Keys

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
        'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on=['id','subject_id'])
```

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

	Name_x	id	subject_id	Name_y
0	Alice	4	sub6	Bryce
1	Ayoung	5	sub5	Betty

Merging/Joining

- The **how** argument to merge specifies how to determine which keys are to be included in the resulting table. If a key combination does not appear in either the left or the right tables, the values in the joined table will be NA.

Merge Method	SQL Equivalent	Description
left	LEFT OUTER JOIN	Use keys from left object
right	RIGHT OUTER JOIN	Use keys from right object
outer	FULL OUTER JOIN	Use union of keys
inner	INNER JOIN	Use intersection of keys

Merging/Joining : Left Join

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, on='subject_id', how='left')
```

Left

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

Right

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

	Name_x	id_x	subject_id	Name_y	id_y
0	Alex	1	sub1	NaN	NaN
1	Amy	2	sub2	Billy	1.0
2	Allen	3	sub4	Brian	2.0
3	Alice	4	sub6	Bryce	4.0
4	Ayoung	5	sub5	Betty	5.0

Merging/Joining : Right Join

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, on='subject_id', how='right')
```

Left

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

Right

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

	Name_x	id_x	subject_id	Name_y	id_y
0	Amy	2.0	sub2	Billy	1
1	Allen	3.0	sub4	Brian	2
2	Alice	4.0	sub6	Bryce	4
3	Ayoung	5.0	sub5	Betty	5
4	NaN	NaN	sub3	Bran	3

Merging/Joining : Outer Join

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, how='outer', on='subject_id')
```

Left

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

Right

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

	Name_x	id_x	subject_id	Name_y	id_y
0	Alex	1.0	sub1	NaN	NaN
1	Amy	2.0	sub2	Billy	1.0
2	Allen	3.0	sub4	Brian	2.0
3	Alice	4.0	sub6	Bryce	4.0
4	Ayoung	5.0	sub5	Betty	5.0
5	NaN	NaN	sub3	Bran	3.0

Merging/Joining : Inner Join

- Joining will be performed on index. Join operation honors the object on which it is called. So, **a.join(b)** is not equal to **b.join(a)**.

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, on='subject_id', how='inner')
```

Left

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

Right

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

	Name_x	id_x	subject_id	Name_y	id_y
0	Amy	2	sub2	Billy	1
1	Allen	3	sub4	Brian	2
2	Alice	4	sub6	Bryce	4
3	Ayoung	5	sub5	Betty	5