PANDAS

In [1]:
```
import numpy as np

import pandas as pd
```

```
Series: a one-dimensional labeled array holding data of any type
such as integers, strings, Python objects etc.

DataFrame: a two-dimensional data structure that holds data like a two-dimension array
or a table with rows and columns.
```

In [2]:
```
a=pd.Series([1,8,90,5.3,"hello"])
a
```

Out[2]:
```
0        1
1        8
2       90
3      5.3
4    hello
dtype: object
```

In [3]:
```
de=pd.DataFrame(
{
    "A":1.6,
    "B":pd.Timestamp("2004-08-15"),
    "C":pd.Series(1,index=list(range(4)),dtype="float32"),
    "D":np.array([3]*4,dtype="int32"),
    "E":pd.Categorical(["A","B","C","D"]),
    "F":"hi",
})

de
```

Out[3]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi
3	1.6	2004-08-15	1.0	3	D	hi

In [4]:
```
de.dtypes
```

Out[4]:
```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

Viewing data

In [5]:
```
de.head()
```

Out[5]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi
3	1.6	2004-08-15	1.0	3	D	hi

In [6]:
```
de.head(2)   #first 2 row
```

Out[6]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi

In [7]:
```
de.tail()  #form last
```

Out[7]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi
3	1.6	2004-08-15	1.0	3	D	hi

In [8]:
```
de.index
```

Out[8]:
```
Int64Index([0, 1, 2, 3], dtype='int64')
```

In [9]:
```
de.columns
```

Out[9]:
```
Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
```

In [10]:
```
# without index and columns
de.to_numpy()
```

Out[10]:
```
array([[1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'A', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'B', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'C', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'D', 'hi']],
      dtype=object)
```

```
NumPy arrays have one dtype for the entire array while pandas DataFrames have one dtype
per column. When you call de.to_numpy(), pandas will find the NumPy dtype that can hold
all of the dtypes in the DataFrame. If the common data type is object, de.to_numpy()
will require copying data.
```

In [11]:
```
# quick statistic summary of your data
de.describe()
```

Out[11]:

	A	C	D
count	4.0	4.0	4.0
mean	1.6	1.0	3.0
std	0.0	0.0	0.0
min	1.6	1.0	3.0
25%	1.6	1.0	3.0
50%	1.6	1.0	3.0
75%	1.6	1.0	3.0
max	1.6	1.0	3.0

In [12]:
```
#transposing
de.T
```

Out[12]:

	0	1	2	3
A	1.6	1.6	1.6	1.6
B	2004-08-15 00:00:00	2004-08-15 00:00:00	2004-08-15 00:00:00	2004-08-15 00:00:00
C	1.0	1.0	1.0	1.0
D	3	3	3	3
E	A	B	C	D
F	hi	hi	hi	hi

Sorting

In [13]:
```
# sort_index() and sort_values()
```

In [14]:
```
de.sort_index(axis=1,ascending=False)
#column index (axis=1)
```

Out[14]:

	F	E	D	C	B	A
0	hi	A	3	1.0	2004-08-15	1.6
1	hi	B	3	1.0	2004-08-15	1.6
2	hi	C	3	1.0	2004-08-15	1.6
3	hi	D	3	1.0	2004-08-15	1.6

In [15]:
```
de.sort_index(axis=0,ascending=False)
#row based sorting
```

Out[15]:

	A	B	C	D	E	F
3	1.6	2004-08-15	1.0	3	D	hi
2	1.6	2004-08-15	1.0	3	C	hi
1	1.6	2004-08-15	1.0	3	B	hi
0	1.6	2004-08-15	1.0	3	A	hi

In [16]:
```
de.sort_values(by="E",ascending=True)
```

Out[16]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi
3	1.6	2004-08-15	1.0	3	D	hi

```
Column => axis=1
row=> axis=0
column&row=>label
```

Selection

```
.at .iat .loc .iloc
```

```
DataFrame.at
Access a single value for a row/column pair by label.

DataFrame.iat
Access a single value for a row/column pair by integer position.

DataFrame.loc
Access a group of rows and columns by label(s).

DataFrame.iloc
Access a group of rows and columns by integer position(s).
```

In [17]:
```
de["A"]
```

Out[17]:
```
0    1.6
1    1.6
2    1.6
3    1.6
Name: A, dtype: float64
```

In [18]:
```
de[1:3]
```

Out[18]:

	A	B	C	D	E	F
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi

```
Selection by label   .at() & .loc()
```

In [19]:
```
de.loc[1]
```

Out[19]:
```
A                     1.6
B    2004-08-15 00:00:00
C                     1.0
D                       3
E                       B
F                      hi
Name: 1, dtype: object
```

In [20]:
```
#all rows (:) with a select column
de.loc[:,["A"]]
```

Out[20]:

	A
0	1.6
1	1.6
2	1.6
3	1.6

In [21]:
```
de.loc[:,:] #all rows and columns
```

Out[21]:

	A	B	C	D	E	F
0	1.6	2004-08-15	1.0	3	A	hi
1	1.6	2004-08-15	1.0	3	B	hi
2	1.6	2004-08-15	1.0	3	C	hi
3	1.6	2004-08-15	1.0	3	D	hi

In [22]:
```
print(de.loc[3,"C"])#Selecting a single row and column label returns a scala
print(de.at[3,"C"])#For getting fast access to a scalar (equivalent to the prior method)
```

```
1.0
1.0
```

Selection by position

```
.iloc() & iat()
```

In [23]:
```
de.iloc[3]
```

Out[23]:
```
A                      1.6
B    2004-08-15 00:00:00
C                      1.0
D                        3
E                        D
F                       hi
Name: 3, dtype: object
```

In [24]:
```
de.iloc[2:5, 0:2]
```

Out[24]:

	A	B
2	1.6	2004-08-15
3	1.6	2004-08-15

In [25]:
```
de.iloc[[1, 2, 0], [0, 2]]
```

Out[25]:

	A	C
1	1.6	1.0
2	1.6	1.0
0	1.6	1.0

In [26]:
```
print(de.iloc[:,0:2])#all row
print(de.iloc[0:2,:])#all column
```

```
     A          B
0  1.6 2004-08-15
1  1.6 2004-08-15
2  1.6 2004-08-15
3  1.6 2004-08-15
     A          B    C  D  E   F
0  1.6 2004-08-15  1.0  3  A  hi
1  1.6 2004-08-15  1.0  3  B  hi
```

In [27]:
```
print(de.iloc[1,1])#value explicitly:
print(de.iat[1,1])#fast access to scalar
```

```
2004-08-15 00:00:00
2004-08-15 00:00:00
```