
(http://www.pieriandata.com)

DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic!

In [183]:

```
import pandas as pd
import numpy as np
```

In [184]:

```
from numpy.random import randn
np.random.seed(101)
```

In [185]:

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

In [186]:

```
df
```

Out[186]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Selection and Indexing

Let's learn the various methods to grab data from a DataFrame

In [187]:

```
df['W']
```

Out[187]:

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

In [188]:

```
# Pass a list of column names
df[['W','Z']]
```

Out[188]:

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

In [189]:

```
# SQL Syntax (NOT RECOMMENDED!)
df.W
```

Out[189]:

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

DataFrame Columns are just Series

In [190]:

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

Out[190]:

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

Creating a new column:

In [191]:

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

In [192]:

```
df
```

Out[192]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

Removing Columns

In [193]:

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

Out[193]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [194]:

```
# Not inplace unless specified!
df
```

Out[194]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

In [195]:

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

In [196]:

```
df
```

Out[196]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Can also drop rows this way:

In [197]:

```
df.drop('E',axis=0)
```

Out[197]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

Selecting Rows

In [198]:

```
df.loc['A']
```

Out[198]:

```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

Or select based off of position instead of label

In [199]:

```
df.iloc[2]
```

Out[199]:

```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

Selecting subset of rows and columns

In [200]:

```
df.loc['B','Y']
```

Out[200]:

```
-0.84807698340363147
```

In [201]:

```
df.loc[['A','B'],['W','Y']]
```

Out[201]:

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

Conditional Selection

An important feature of pandas is conditional selection using bracket notation, very similar to numpy:

In [202]:

```
df
```

Out[202]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [203]:

```
df>0
```

Out[203]:

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

In [204]:

```
df[df>0]
```

Out[204]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [205]:

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

Out[205]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [206]:

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

Out[206]:

```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

In [207]:

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

Out[207]:

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

For two conditions you can use | and & with parenthesis:

In [208]:

```
df[(df['W']>0) & (df['Y'] > 1)]
```

Out[208]:

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

More Index Details

Let's discuss some more features of indexing, including resetting the index or setting it something else. We'll also talk about index hierarchy!

In [209]:

```
df
```

Out[209]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [210]:

```
# Reset to default 0,1...n index
df.reset_index()
```

Out[210]:

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

In [211]:

```
newind = 'CA NY WY OR CO'.split()
```

In [212]:

```
df['States'] = newind
```

In [213]:

```
df
```

Out[213]:

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

In [214]:

```
df.set_index('States')
```

Out[214]:

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

In [215]:

```
df
```

Out[215]:

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

In [216]:

```
df.set_index('States',inplace=True)
```

In [218]:

```
df
```

Out[218]:

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

Multi-Index and Index Hierarchy

Let us go over how to work with Multi-Index, first we'll create a quick example of what a Multi-Indexed DataFrame would look like:

In [253]:

```
# Index Levels
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 [254]:

```
hier_index
```

Out[254]:

```
MultiIndex(levels=[['G1', 'G2'], [1, 2, 3]],
           labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])
```

In [257]:

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

Out[257]:

		A	B
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

Now let's show how to index this! For index hierarchy we use df.loc[], if this was on the columns axis, you would just use normal bracket notation df[]. Calling one level of the index returns the sub-dataframe:

In [260]:

```
df.loc['G1']
```

Out[260]:

	A	B
1	0.153661	0.167638
2	-0.765930	0.962299
3	0.902826	-0.537909

In [263]:

```
df.loc['G1'].loc[1]
```

Out[263]:

```
A    0.153661
B    0.167638
Name: 1, dtype: float64
```

In [265]:

```
df.index.names
```

Out[265]:

```
FrozenList([None, None])
```

In [266]:

```
df.index.names = ['Group','Num']
```

In [267]:

```
df
```

Out[267]:

Group	Num	A	B
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

In [270]:

```
df.xs('G1')
```

Out[270]:

Num	A	B
1	0.153661	0.167638
2	-0.765930	0.962299
3	0.902826	-0.537909

In [271]:

```
df.xs(['G1',1])
```

Out[271]:

```
A    0.153661
B    0.167638
Name: (G1, 1), dtype: float64
```

In [273]:

```
df.xs(1,level='Num')
```

Out[273]:

Group	A	B
G1	0.153661	0.167638
G2	-1.549671	0.435253

Great Job!