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 [1]:

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

In [2]:

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

In [3]:

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

In [4]:

```
df
```

Out[4]:

	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 [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

	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 [7]:

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

Out[7]:

```
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 [8]:

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

Out[8]:

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

Creating a new column:

In [9]:

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

In [10]:

```
df
```

Out[10]:

	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 [11]:

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

Out[11]:

	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 [12]:

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

Out[12]:

	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 [13]:

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

In [14]:

```
df
```

Out[14]:

	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 [15]:

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

Out[15]:

	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 [16]:

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

Out[16]:

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

Or select based on the position instead of label

In [17]:

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

Out[17]:

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

Selecting subset of rows and columns

In [18]:

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

Out[18]:

```
-0.8480769834036315
```

In [19]:

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

Out[19]:

	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 [20]:

```
df
```

Out[20]:

	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 [21]:

```
df>0
```

Out[21]:

	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 [22]:

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

Out[22]:

	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 [23]:

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

Out[23]:

	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 [24]:

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

Out[24]:

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

In [25]:

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

Out[25]:

	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 [26]:

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

Out[26]:

	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 [27]:

```
df
```

Out[27]:

	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 [28]:

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

Out[28]:

	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 [29]:

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

In [30]:

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

In [31]:

```
df
```

Out[31]:

	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 [32]:

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

Out[32]:

	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 [33]:

```python
df
```

Out[33]:

	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 [34]:

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

In [35]:

```python
df
```

Out[35]:

	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 [36]:

```
# 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 [37]:

```
hier_index
```

Out[37]:

```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

In [38]:

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

Out[38]:

		A	B
G1	**1**	0.302665	1.693723
	2	-1.706086	-1.159119
	3	-0.134841	0.390528
G2	**1**	0.166905	0.184502
	2	0.807706	0.072960
	3	0.638787	0.329646

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 [39]:

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

Out[39]:

	A	B
1	0.302665	1.693723
2	-1.706086	-1.159119
3	-0.134841	0.390528

In [40]:

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

Out[40]:

```
A    0.302665
B    1.693723
Name: 1, dtype: float64
```

In [41]:

```
df.index.names
```

Out[41]:

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

In [42]:

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

In [43]:

```
df
```

Out[43]:

```
                  A          B
Group Num
   G1   1   0.302665   1.693723
        2  -1.706086  -1.159119
        3  -0.134841   0.390528
   G2   1   0.166905   0.184502
        2   0.807706   0.072960
        3   0.638787   0.329646
```

In [44]:

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

Out[44]:

```
            A          B
Num
  1   0.302665   1.693723
  2  -1.706086  -1.159119
  3  -0.134841   0.390528
```

In [45]:

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

Out[45]:

```
A    0.302665
B    1.693723
Name: (G1, 1), dtype: float64
```

In [46]:

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

Out[46]:

	A	B
Group		
G1	0.302665	1.693723
G2	0.166905	0.184502

Great Job!

In [ ]:

```

```