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'],columns=['col1','col2'
```

In [6]:

```
df
```

Out[6]:

	col1	col2	col3	col4
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 [ ]:

```

```

In [185]:

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

In [7]:

```
1 df
```

Out[7]:

	col1	col2	col3	col4
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 [ ]:

```
1
```

Selection and Indexing

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

In [9]:

```
1 df['col4']
```

Out[9]:

```
A    0.503826
B    0.605965
C   -0.589001
D    0.955057
E    0.683509
Name: col4, dtype: float64
```

In [11]:

```
1 df[['col1','col3','col4']]
```

Out[11]:

	col1	col3	col4
A	2.706850	0.907969	0.503826
B	0.651118	-0.848077	0.605965
C	-2.018168	0.528813	-0.589001
D	0.188695	-0.933237	0.955057
E	0.190794	2.605967	0.683509

In [ ]:

```
1 df.agg
```

In [12]:

```
# Pass a list of column names
df[['col1','col2']]
```

Out[12]:

	col1	col2
A	2.706850	0.628133
B	0.651118	-0.319318
C	-2.018168	0.740122
D	0.188695	-0.758872
E	0.190794	1.978757

In [14]:

```
# SQL Syntax (NOT RECOMMENDED!)
df.col1 #to access the column
```

Out[14]:

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: col1, dtype: float64
```

In [15]:

```
df['col1']
```

Out[15]:

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: col1, dtype: float64
```

DataFrame Columns are just Series

In [16]:

```
type(df['col1'])
```

Out[16]:

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

Creating a new column:

In [17]:

```
1 df
```

Out[17]:

	col1	col2	col3	col4
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 [18]:

```
1 df['new'] = df['col1']+df['col2']+df['col3']+df['col4']
```

In [19]:

```
1 df
```

Out[19]:

	col1	col2	col3	col4	new
A	2.706850	0.628133	0.907969	0.503826	4.746778
B	0.651118	-0.319318	-0.848077	0.605965	0.089688
C	-2.018168	0.740122	0.528813	-0.589001	-1.338233
D	0.188695	-0.758872	-0.933237	0.955057	-0.548357
E	0.190794	1.978757	2.605967	0.683509	5.459028

In [ ]:

```
1
```

In [191]:

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

In [21]:

```
1 df
```

Out[21]:

	col1	col2	col3	col4	new
A	2.706850	0.628133	0.907969	0.503826	4.746778
B	0.651118	-0.319318	-0.848077	0.605965	0.089688
C	-2.018168	0.740122	0.528813	-0.589001	-1.338233
D	0.188695	-0.758872	-0.933237	0.955057	-0.548357
E	0.190794	1.978757	2.605967	0.683509	5.459028

In [25]:

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

In [26]:

```
1 df
```

Out[26]:

	col1	col2	col3	col4
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

** Removing Columns**

In [193]:

```
1 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 [28]:

```
df
```

Out[28]:

	col1	col2	col3	col4
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 [30]:

```
df.drop('A',axis=0) #removing row
```

Out[30]:

	col1	col2	col3	col4
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 [31]:

```
df.loc['D']
```

Out[31]:

```
col1    0.188695
col2   -0.758872
col3   -0.933237
col4    0.955057
Name: D, dtype: float64
```

** 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 [ ]:

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

In [ ]:

```
1
```

In [33]:

```
1 df
```

Out[33]:

	col1	col2	col3	col4
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 [34]:

```
1 df.loc[['A','B'],['col1','col2']]
```

Out[34]:

	col1	col2
A	2.706850	0.628133
B	0.651118	-0.319318

In [ ]:

```
1
```

In [201]:

```
1 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 [35]:

```
1 df
```

Out[35]:

	col1	col2	col3	col4
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 [36]:

```
1 df>0
```

Out[36]:

	col1	col2	col3	col4
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 [38]:

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

Out[38]:

	col1	col2	col3	col4
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 [39]:

```
df[df['col1']>0]
```

Out[39]:

	col1	col2	col3	col4
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 [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 [43]:

```
df
```

Out[43]:

	col1	col2	col3	col4
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 [ ]:

```
1
```

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

In [41]:

```
1 df[(df['col1']>0) & (df['col2']>1)]
```

Out[41]:

	col1	col2	col3	col4
E	0.190794	1.978757	2.605967	0.683509

In [ ]:

```
1
```

In [ ]:

```
1
```

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 [44]:

```
1 df
```

Out[44]:

	col1	col2	col3	col4
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 [45]:

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

Out[45]:

	index	col1	col2	col3	col4
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 [46]:

```
capital = ['Delhi','New York','Tokyo','Paris','Moscow']
```

In [47]:

```
df['States'] = capital
```

In [48]:

```
df
```

Out[48]:

	col1	col2	col3	col4	States
A	2.706850	0.628133	0.907969	0.503826	Delhi
B	0.651118	-0.319318	-0.848077	0.605965	New York
C	-2.018168	0.740122	0.528813	-0.589001	Tokyo
D	0.188695	-0.758872	-0.933237	0.955057	Paris
E	0.190794	1.978757	2.605967	0.683509	Moscow

In [49]:

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

Out[49]:

States	col1	col2	col3	col4
Delhi	2.706850	0.628133	0.907969	0.503826
New York	0.651118	-0.319318	-0.848077	0.605965
Tokyo	-2.018168	0.740122	0.528813	-0.589001
Paris	0.188695	-0.758872	-0.933237	0.955057
Moscow	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]:

States	W	X	Y	Z
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 [51]:

```
# 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 [53]:

```
hier_index
```

Out[53]:

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

In [54]:

```
df1 = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['col1','col2'
```

In [55]:

```
df1
```

Out[55]:

		col1	col2
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 [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

In [56]:

```
1 df1
```

Out[56]:

		col1	col2
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 [59]:

```
1 df1.loc['G1']
```

Out[59]:

	col1	col2
1	0.302665	1.693723
2	-1.706086	-1.159119
3	-0.134841	0.390528

In [62]:

```
1 df1.loc['G1'].loc[3]
```

Out[62]:

```
col1   -0.134841
col2    0.390528
Name: 3, dtype: float64
```

In [263]:

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

Out[263]:

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

In [265]:

```
1 df.index.names
```

Out[265]:

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

In [63]:

```
1 df1
```

Out[63]:

		col1	col2
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 [65]:

```
1 df1.index.names = ['Group','Index']
```

In [66]:

```
1 df1
```

Out[66]:

		col1	col2
Group	**Index**		
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 [266]:

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

In [67]:

```
1 df1
```

Out[67]:

		col1	col2
Group	**Index**		
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 [70]:

```
1 df1.xs('G1')
```

Out[70]:

	col1	col2
Index		
1	0.302665	1.693723
2	-1.706086	-1.159119
3	-0.134841	0.390528

In [71]:

```
1 df1.xs('G2')
```

Out[71]:

	col1	col2
Index		
1	0.166905	0.184502
2	0.807706	0.072960
3	0.638787	0.329646

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!