In [1]:
```
#Now we'll learn about pandas built-in methods of summarizing data founr in DataF
import numpy as np
from pandas import Series,DataFrame
import pandas as pd
```

In [3]:
```
#Let's create a dataframe to work with
arr = np.array([[1,2,np.nan],[np.nan,3,4]])
dframe1 = DataFrame(arr,index=['A','B'],columns = ['One','Two','Three'])

#Show
dframe1
```

Out[3]:

	One	Two	Three
A	1	2	NaN
B	NaN	3	4

In [4]:
```
#Let's see the sum() method in action
dframe1.sum()
```

Out[4]:
```
One      1
Two      5
Three    4
dtype: float64
```

In [5]:
```
#Notice how it ignores NaN values
```

In [6]:
```
#We can also over rows instead of columns
dframe1.sum(axis=1)
```

Out[6]:
```
A    3
B    7
dtype: float64
```

In [7]:
```
#Can also grab min and max values of dataframe
dframe1.min()
```

Out[7]:
```
One      1
Two      2
Three    4
dtype: float64
```

In [8]:
```
#As well as there index
dframe1.idxmin()
```

Out[8]:
```
One      A
Two      A
Three    B
dtype: object
```

In [9]:
```
#Same deal with max, just replace min for max
```

In [10]:
```
#Show
dframe1
```

Out[10]:

	One	Two	Three
A	1	2	NaN
B	NaN	3	4

In [11]:
```
#Can also do an accumulation sum
dframe1.cumsum()
```

Out[11]:

	One	Two	Three
A	1	2	NaN
B	NaN	5	4

In [12]:
```
#A very useful feature is describe, which provides summary statistics
dframe1.describe()
```

Out[12]:

	One	Two	Three
count	1	2.000000	1
mean	1	2.500000	4
std	NaN	0.707107	NaN
min	1	2.000000	4
25%	1	2.250000	4
50%	1	2.500000	4
75%	1	2.750000	4
max	1	3.000000	4

In [13]:
```
# We can also get information on correlation and covariance

#For more info on correlation and covariance, check out the videos below!
```

In [14]:
```
from IPython.display import YouTubeVideo
# For more information about Covariaance and Correlation
# Check out these great videos!
# Video credit: Brandon Foltz.

#CoVariance
YouTubeVideo('xGbpuFNR1ME')
```

Out[14]:

In [15]:
```
#Correlation
YouTubeVideo('4EXNedimDMs')
```

Out[15]:

In [16]:
```
#Now lets check correlation and covariance on some stock prices!

#Pandas can get info off the web
import pandas.io.data as pdweb

#Set datetime for date input
import datetime

#Get the closing prices

prices = pdweb.get_data_yahoo(['CVX','XOM','BP'],
                              start=datetime.datetime(2010, 1, 1),
                              end=datetime.datetime(2013, 1, 1))['Adj Close']

#Show preview
prices.head()
```

Out[16]:

	BP	CVX	XOM
Date			
2010-01-04	46.97	66.17	60.26
2010-01-05	47.30	66.63	60.50
2010-01-06	47.55	66.64	61.02
2010-01-07	47.53	66.39	60.83
2010-01-08	47.64	66.51	60.59

In [17]:
```
#Now lets get the volume trades

volume = pdweb.get_data_yahoo(['CVX','XOM','BP'],
                              start=datetime.datetime(2010, 1, 1),
                              end=datetime.datetime(2013, 1, 1))['Volume']


#Show preview
volume.head()
```

Out[17]:

	BP	CVX	XOM
Date			
2010-01-04	3956100	10173800	27809100
2010-01-05	4109600	10593700	30174700
2010-01-06	6227900	11014600	35044700
2010-01-07	4431300	9626900	27192100
2010-01-08	3786100	5624300	24891800

In [18]:
```
#Lets get the return
rets = prices.pct_change()
```

In [23]:
```
#Get the correlation of the stocks
corr = rets.corr
```

In [24]:
```
#Lets see the prices over time to get a very rough idea of the correlation betwee
prices.plot()
```

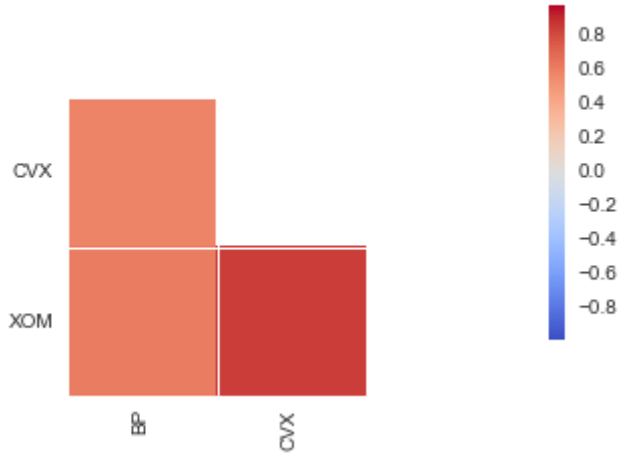
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1aa126a0>

In [20]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#As expected pretty strong correlations with eachother
sns.heatmap(rets.corr())

#We'll learn much more about seaborn later!
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x18b63a58>



In [2]:
```
# We can also check for unique values and their counts

#For example
ser1 = Series(['w','w','x', 'y', 'z' ,'w' ,'w' ,'x' ,'x' ,'y' ,'a' ,'z' ])

#Show
ser1
```

Out[2]:
```
0     w
1     w
2     x
3     y
4     z
5     w
6     w
7     x
8     x
9     y
10    a
11    z
dtype: object
```

In [3]:
```
#Grab the unique values
ser1.unique()
```

Out[3]:
```
array(['w', 'x', 'y', 'z', 'a'], dtype=object)
```

In [4]:
```
#Now get the count of the unique values
ser1.value_counts()
```

Out[4]:
```
w    4
x    3
z    2
y    2
a    1
dtype: int64
```

In [ ]:
```
#Next we'll learn how to best deal with missing data!
```