Pandas Introduction

Pandas BKHW with Stock Data and Correlation Examples

Author list: Ikhlaq Sidhu

References / Sources: Includes examples from Wes McKinney

License Agreement: Feel free to do whatever you want with this code

What Does Pandas Do?

Pandas lets us construct tables, called Data Frames

With NumPy, we can store and manipulate a matrix

m =

```
[[-0.09443539 -0.09443531  0.29860729 -0.09761513 -0.09440866]
 [-0.09443526 -0.09443531  0.25596021 -0.10824217 -0.094422  ]
 [-0.09443524 -0.09443531  0.37198598 -0.12371693 -0.09442577]
 [-0.09443568 -0.09443531  0.30667577 -0.10257815 -0.09441752]
 [-0.09443562 -0.09443531  0.41545527 -0.06368836 -0.09441873]
 [-0.09443647 -0.09443531  0.34410876  0.00738793 -0.09440932]
 [-0.0944355  -0.09443531  0.33180906 -0.12472302 -0.09442687]
 [-0.09443587 -0.09443531  0.3643611  -0.16894118 -0.09443041]
 [-0.09443721 -0.09443531  0.43028699  0.0095     -0.09441093]
 [-0.09443846 -0.09443531  0.34737789 -0.07818481 -0.09439922]
```

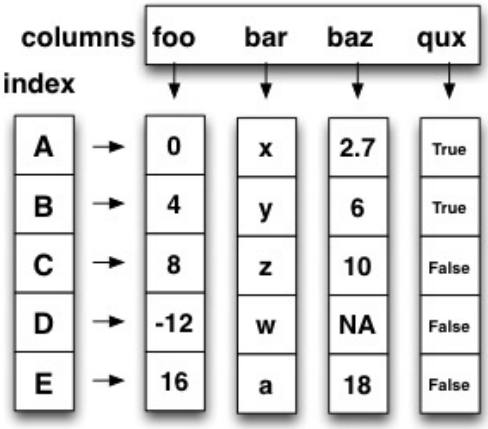
With Pandas, we can store and manipulate a full table

df =

	Birth Month	Origin	Age	Gender
Carly	January	UK	27	f
Rachel	September	Spain	28	f
Nicky	September	Jamaica	28	f
Wendy	November	Italy	22	f
Judith	February	France	19	f

What is a Pandas Table Object?

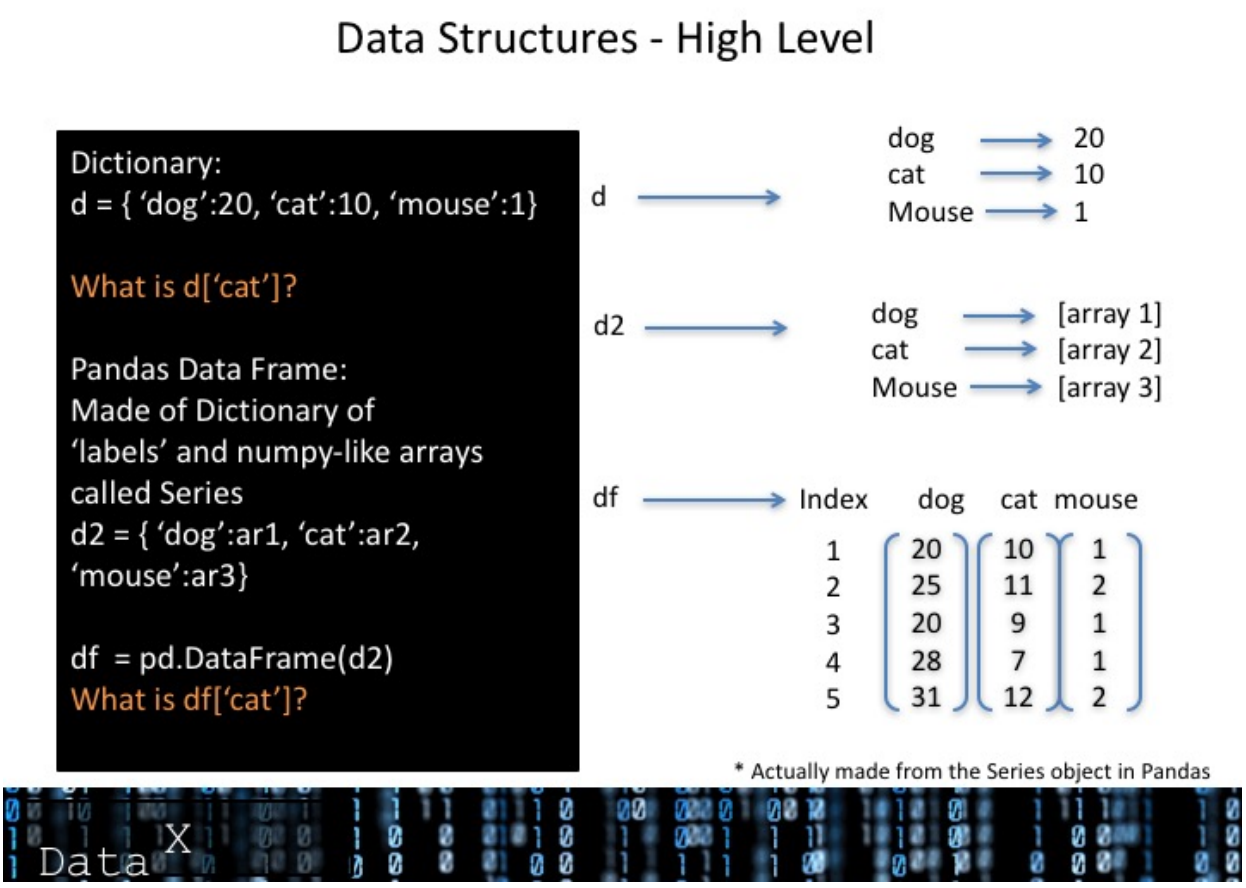
Pandas has an object called a *Data Frame* which is like a table



- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns

Wes Mckinney

This table is a dictionary of sequences (like np arrays)



Creating a Data Frame

Key Points: Pandas has Series (like Arrays), DataFrames (like Tables), and Panels (3D version)

In [1]:
```
import pandas as pd
import numpy as np
#pd.show_versions()
pd.__version__
```

Out[1]: '0.20.1'

Our first goal: Learn that it is easy to create a data frame

We use pandas.DataFrame and put in just about anything data type as an argument

class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

data ca be a numpy ndarray (structured or homogeneous), dict, or DataFrame. Dict can contain Series, arrays, constants, or list-like objects

In [2]:
```
# Try it with an array
a1 = np.array(np.random.randn(3))
a2 = np.array(np.random.randn(3))
a3 = np.array(np.random.randn(3))

print (a1, type(a1))
print (a2)
print (a3)
```

```
[-0.53865623 -1.59430988  0.55825226] <class 'numpy.ndarray'>
[ 0.74500661  1.76264309 -1.19715705]
[-0.75096349 -0.09997209  0.1688658 ]
```

In [3]:
```
df0 = pd.DataFrame(a1)
print ("This is a dataframe")
df0
```

```
This is a dataframe
```

Out[3]:

	0
0	-0.538656
1	-1.594310
2	0.558252

In [4]:
```
# What if I give it a list of np arrays?
df0 = pd.DataFrame([a1, a2, a3])
print (type(df0))
df0
# notice that there is no column label and the index is set automatically
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[4]:

	0	1	2
0	-0.538656	-1.594310	0.558252
1	0.745007	1.762643	-1.197157
2	-0.750963	-0.099972	0.168866

In [5]:
```
# Let use a 2-D arrays?
ax = np.array(np.random.randn(9)).reshape(3,3)
print (ax)
df0 = pd.DataFrame(ax)
df0
```

```
[[ 1.60321811  0.45283317  0.06095341]
 [ 0.69255117 -0.14957509  0.74226605]
 [-0.45836558 -0.20986929 -0.50253965]]
```

Out[5]:

	0	1	2
0	1.603218	0.452833	0.060953
1	0.692551	-0.149575	0.742266
2	-0.458366	-0.209869	-0.502540

In [6]:
```
# And now, lets use a dictionary as input

dict1 = {'A':a1, 'B':a2}
df1 = pd.DataFrame(dict1)
df1
# note headings
```

Out[6]:

	A	B
0	-0.538656	0.745007
1	-1.594310	1.762643
2	0.558252	-1.197157

In [7]:
```
# Lets add another column
df1['C']=a3
df1
```

Out[7]:

	A	B	C
0	-0.538656	0.745007	-0.750963
1	-1.594310	1.762643	-0.099972
2	0.558252	-1.197157	0.168866

In [8]:
```
# What is we assign a list instead of numpy array?
df1['L'] = ["List", "of", "words"]
print ("The column L is a ",type (df1['L']))
df1
```

```
The column L is a  <class 'pandas.core.series.Series'>
```

Out[8]:

	A	B	C	L
0	-0.538656	0.745007	-0.750963	List
1	-1.594310	1.762643	-0.099972	of
2	0.558252	-1.197157	0.168866	words

In [9]:
```
# Introducing a pandas.Series
# Its like an np.array but it has its own index
s = pd.Series([1,np.nan,3])
s2 = pd.Series([2, 3, 4], index = ['a','b','c'])
print (s)
print (s2)
```

```
0    1.0
1    NaN
2    3.0
dtype: float64
a    2
b    3
c    4
dtype: int64
```

In [10]:
```
# We will add the Series s to the table as column S
df1['S'] = s
df1
```

Out[10]:

	A	B	C	L	S
0	-0.538656	0.745007	-0.750963	List	1.0
1	-1.594310	1.762643	-0.099972	of	NaN
2	0.558252	-1.197157	0.168866	words	3.0

In [11]:
```
# This time, we will Series s2, which has a different index
df1['S2']=s2
print (df1)
```

```
          A         B         C      L    S  S2
0 -0.538656  0.745007 -0.750963   List  1.0 NaN
1 -1.594310  1.762643 -0.099972     of  NaN NaN
2  0.558252 -1.197157  0.168866  words  3.0 NaN
```

In [12]:
```
# But if we create a new dataframe, we can add the data but with the new index
df2 = pd.DataFrame(s2)
df2['A']= a1
df2['B']=a2
df2['C']=a3
print (df2)
```

```
   0         A         B         C
a  2 -0.538656  0.745007 -0.750963
b  3 -1.594310  1.762643 -0.099972
c  4  0.558252 -1.197157  0.168866
```

In [13]:
```
# You can extract rows by position or label
print (df2[1:3])
print (df2['a':'b'])
```

```
   0         A         B         C
b  3 -1.594310  1.762643 -0.099972
c  4  0.558252 -1.197157  0.168866
   0         A         B         C
a  2 -0.538656  0.745007 -0.750963
b  3 -1.594310  1.762643 -0.099972
```

In [14]:
```
# recall
df1
```

Out[14]:

	A	B	C	L	S	S2
0	-0.538656	0.745007	-0.750963	List	1.0	NaN
1	-1.594310	1.762643	-0.099972	of	NaN	NaN
2	0.558252	-1.197157	0.168866	words	3.0	NaN

In [15]:
```
# Renaming a column
df1 = df1.rename(columns = {'L':'D'})
df1
```

Out[15]:

	A	B	C	D	S	S2
0	-0.538656	0.745007	-0.750963	List	1.0	NaN
1	-1.594310	1.762643	-0.099972	of	NaN	NaN
2	0.558252	-1.197157	0.168866	words	3.0	NaN

In [16]:
```
# And delete column
del df1['S2']
df1
```

Out[16]:

	A	B	C	D	S
0	-0.538656	0.745007	-0.750963	List	1.0
1	-1.594310	1.762643	-0.099972	of	NaN
2	0.558252	-1.197157	0.168866	words	3.0

In [17]:
```
# Example: view only a column
print (df1['A'])
```

```
0   -0.538656
1   -1.594310
2    0.558252
Name: A, dtype: float64
```

In [18]:
```
# A list of colums
print (df1[['A','D']])
print (type(df1['A']))

# Notice the data structure of the column is a Series, not an array
```

```
          A      D
0 -0.538656   List
1 -1.594310     of
2  0.558252  words
<class 'pandas.core.series.Series'>
```

In the 10 min Pandas Guide, you will see many ways to view, slice a dataframe

- view/slice by rows, eg df[1:3], etc.
- view by index location, see df.iloc (iloc)
- view by ranges of labels, ie index label 2 to 5, or dates feb 3 to feb 25, see df.loc (loc)
- view a single row by the index df.xs (xs)
- filtering rows that have certain conditions
- add column
- add row
- How to change the index

and more...

In [19]:
```
print (df1[0:2])  # ok
# df1[1]  # not ok

#What will this do?
# print df1[0:2]
# print (df1[0:2][0:1])
```

```
          A         B         C     D    S
0 -0.538656  0.745007 -0.750963  List  1.0
1 -1.594310  1.762643 -0.099972    of  NaN
```

Finance example

Now, lets get some data in a CSV file is like this.

See https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free (https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free)

In [20]:
```
# We can use this 'pd.read_csv' method with the yahoo url, 1 year back with goog
dfg = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=goog')
dfa = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=aapl')
```

In [21]:
```
# some viewing options
print ('aaple:')
print (dfa[0:3])
print ('google:')
print (dfg.head(10))
# dfg[0:10]
# dfg
print (dfg.tail(3))
```

```
aaple:
        Date    Open    High     Low   Close    Volume
0  21-Jun-17  145.52  146.07  144.61  145.87  21202929
1  20-Jun-17  146.87  146.87  144.94  145.01  24900073
2  19-Jun-17  143.66  146.74  143.66  146.34  32541404
google:
        Date    Open    High     Low   Close   Volume
0  21-Jun-17  953.64  960.10  950.76  959.45  1200693
1  20-Jun-17  957.52  961.62  950.01  950.63  1125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1533336
3  16-Jun-17  940.00  942.04  931.60  939.78  3094711
4  15-Jun-17  933.97  943.34  924.44  942.31  2133050
5  14-Jun-17  959.92  961.15  942.25  950.76  1489715
6  13-Jun-17  951.91  959.98  944.09  953.40  2013337
7  12-Jun-17  939.56  949.36  915.23  942.90  3763529
8   9-Jun-17  984.50  984.50  935.63  949.83  3309389
9   8-Jun-17  982.35  984.57  977.20  983.41  1481916
          Date    Open    High     Low   Close   Volume
248  27-Jun-16  671.00  672.30  663.28  668.26  2641085
249  24-Jun-16  675.17  689.40  673.45  675.22  4449022
250  23-Jun-16  697.45  701.95  687.00  701.87  2171415
```

In [22]:
```
# if you want the index by itelf, use .index
# dfg.index
dfg.index[2:20]
```

Out[22]:
```
RangeIndex(start=2, stop=20, step=1)
```

In [23]:
```
# Here is a list of the columns, so you know the names of the columns
dfg.columns
#dfg.columns[2]
#dfg.columns[2:4]

#try this: (element of row 3 from every column)

#for i in dfg.columns:
#    print (i, df0_goog[i][3])
```

Out[23]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume'], dtype='object')
```

In [24]:
```
# If you want the values in an np array
npg = dfg.values
print (npg, type(npg))
```

```
[['21-Jun-17' 953.64 960.1 950.76 959.45 1200693]
 ['20-Jun-17' 957.52 961.62 950.01 950.63 1125990]
 ['19-Jun-17' 949.96 959.99 949.05 957.37 1533336]
 ...,
 ['27-Jun-16' 671.0 672.3 663.28 668.26 2641085]
 ['24-Jun-16' 675.17 689.4 673.45 675.22 4449022]
 ['23-Jun-16' 697.45 701.95 687.0 701.87 2171415]] <class 'numpy.ndarray'>
```

Getting or Viewing Data within a DataFrame

Note: While standard Python / Numpy expressions for selecting and setting are intuitive and come in handy for interactive work, for production code, we recommend the optimized pandas data access methods, .at, .iat, .loc, .iloc and .ix. (from 10 min guide to Pandas)

In [26]:
```
# Lets print the 1 column, which show close prices of Google
# This is a new data frame (like a new table)
dfg['Close'][0:5]
```

Out[26]:
```
0    959.45
1    950.63
2    957.37
3    939.78
4    942.31
Name: Close, dtype: float64
```

In [27]:
```
# Lets print the 2 column, and only 3 down, which show close prices of Google
# Instead of one column name, we have a python list ['Date',Close']
dfg[['Date','Close']][0:3]
```

Out[27]:

	Date	Close
0	21-Jun-17	959.45
1	20-Jun-17	950.63
2	19-Jun-17	957.37

In [28]:
```
# A slice: by rows (row numbers)
print (dfg[1:5])
```

```
        Date    Open    High     Low   Close   Volume
1  20-Jun-17  957.52  961.62  950.01  950.63  1125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1533336
3  16-Jun-17  940.00  942.04  931.60  939.78  3094711
4  15-Jun-17  933.97  943.34  924.44  942.31  2133050
```

In [29]:
```
# Getting a cross section with .loc - BY VALUES of the index and headers
# Introduce loc: this will get you a cross section of the table by label ran
# df.loc[a:b, x:y], by rows and column location

print (dfg.loc[1:4,'Open':'Close'])

# recall:
# dfg[a:b] by rows
# dfg[[col]] or df[[list of col]] by columns
# df.loc[a:b, x:y], by rows and column location
# df.iloc[3:5,0:2], by slicing by specific position - BY POSITION in the table

print (dfg.iloc[1:4,4:6])
```

```
     Open    High     Low   Close
1  957.52  961.62  950.01  950.63
2  949.96  959.99  949.05  957.37
3  940.00  942.04  931.60  939.78
4  933.97  943.34  924.44  942.31
    Close   Volume
1  950.63  1125990
2  957.37  1533336
3  939.78  3094711
```

In [30]:
```
#n Data only from row with index value '3'
print (dfg.iloc[3])
```

```
Date      16-Jun-17
Open            940
High         942.04
Low           931.6
Close        939.78
Volume      3094711
Name: 3, dtype: object
```

In [31]:
```
# iloc will accept 'lists' of position numbers
dfg.iloc[[1,2,4],[0,2]]
```

Out[31]:

	Date	High
1	20-Jun-17	961.62
2	19-Jun-17	959.99
4	15-Jun-17	943.34

In [32]:
```
# iloc will accept a range with ':', just like numpy
dfg.iloc[1:3,:]
```

Out[32]:

	Date	Open	High	Low	Close	Volume
1	20-Jun-17	957.52	961.62	950.01	950.63	1125990
2	19-Jun-17	949.96	959.99	949.05	957.37	1533336

In [35]:
```
# Can also return specific value
print (dfg.iloc[2,1])
# same as above but faster for one single scaler value
print (dfg.iat[2,1])
```

```
949.96
949.96
```

Basic Statistics

In [36]:
```
# A quick way to get statistics
dfg.describe()
# dfg.describe()['A'][1]
# dfg.describe()[2:3]
```

Out[36]:

	Open	High	Low	Close	Volume
count	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	811.345139	816.271036	806.340159	811.818645	1.540603e+06
std	67.900802	68.045718	67.343274	67.890060	6.777015e+05
min	671.000000	672.300000	663.280000	668.260000	5.874210e+05
25%	772.450000	776.260000	767.150000	771.535000	1.108766e+06
50%	795.470000	801.190000	791.190000	795.370000	1.362115e+06
75%	831.635000	836.015000	827.870000	831.580000	1.704087e+06
max	984.500000	988.250000	977.200000	983.680000	4.745183e+06

In [37]:
```
print (dfg[0:5])
dfg[0:5].sort_index(axis=1, ascending=False) # by index or column, try axis = 0
```

```
        Date    Open    High     Low   Close   Volume
0  21-Jun-17  953.64  960.10  950.76  959.45  1200693
1  20-Jun-17  957.52  961.62  950.01  950.63  1125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1533336
3  16-Jun-17  940.00  942.04  931.60  939.78  3094711
4  15-Jun-17  933.97  943.34  924.44  942.31  2133050
```

Out[37]:

	Volume	Open	Low	High	Date	Close
0	1200693	953.64	950.76	960.10	21-Jun-17	959.45
1	1125990	957.52	950.01	961.62	20-Jun-17	950.63
2	1533336	949.96	949.05	959.99	19-Jun-17	957.37
3	3094711	940.00	931.60	942.04	16-Jun-17	939.78
4	2133050	933.97	924.44	943.34	15-Jun-17	942.31

In [38]:
```
# sort by value
dfg[0:5].sort_values(by='Open')
```

Out[38]:

	Date	Open	High	Low	Close	Volume
4	15-Jun-17	933.97	943.34	924.44	942.31	2133050
3	16-Jun-17	940.00	942.04	931.60	939.78	3094711
2	19-Jun-17	949.96	959.99	949.05	957.37	1533336
0	21-Jun-17	953.64	960.10	950.76	959.45	1200693
1	20-Jun-17	957.52	961.62	950.01	950.63	1125990

In [39]:
```
# Transpose in Pandas
print (dfg.describe()[2:3])
print(dfg.describe()[2:3].T)

# try other methods like sum, mean, etc. in the same way.
```

```
          Open       High        Low     Close         Volume
std  67.900802  68.045718  67.343274  67.89006  677701.486063
                    std
Open        67.900802
High        68.045718
Low         67.343274
Close       67.890060
Volume  677701.486063
```

In [ ]:
```

```

Masks and Boolean Indexing

In [40]:
```
dfg[0:10]
```

Out[40]:

	Date	Open	High	Low	Close	Volume
0	21-Jun-17	953.64	960.10	950.76	959.45	1200693
1	20-Jun-17	957.52	961.62	950.01	950.63	1125990
2	19-Jun-17	949.96	959.99	949.05	957.37	1533336
3	16-Jun-17	940.00	942.04	931.60	939.78	3094711
4	15-Jun-17	933.97	943.34	924.44	942.31	2133050
5	14-Jun-17	959.92	961.15	942.25	950.76	1489715
6	13-Jun-17	951.91	959.98	944.09	953.40	2013337
7	12-Jun-17	939.56	949.36	915.23	942.90	3763529
8	9-Jun-17	984.50	984.50	935.63	949.83	3309389
9	8-Jun-17	982.35	984.57	977.20	983.41	1481916

In [41]:
```
# mask 1
mg1 = dfg['Open'][0:10]>941
print (mg1)
# dfg.Open[0:10]>941    # same thing

dfg[dfg['Open']>941][0:10]
# shows only rows with opening price greater than 941
```

```
0     True
1     True
2     True
3    False
4    False
5     True
6     True
7    False
8     True
9     True
Name: Open, dtype: bool
```

Out[41]:

	Date	Open	High	Low	Close	Volume
0	21-Jun-17	953.64	960.10	950.76	959.45	1200693
1	20-Jun-17	957.52	961.62	950.01	950.63	1125990
2	19-Jun-17	949.96	959.99	949.05	957.37	1533336
5	14-Jun-17	959.92	961.15	942.25	950.76	1489715
6	13-Jun-17	951.91	959.98	944.09	953.40	2013337
8	9-Jun-17	984.50	984.50	935.63	949.83	3309389
9	8-Jun-17	982.35	984.57	977.20	983.41	1481916
10	7-Jun-17	979.65	984.15	975.77	981.08	1453874
11	6-Jun-17	983.16	988.25	975.14	976.57	1814624
12	5-Jun-17	976.55	986.91	975.10	983.68	1252106

In [42]:
```
# mask 2
mg2 = dfg[0:10]>941
print (mg2)
print (dfg[dfg>941].head(10))
# replaces every value in the entire table with NaN if the value of below 941
```

```
   Date   Open  High    Low  Close  Volume
0  True   True  True   True   True    True
1  True   True  True   True   True    True
2  True   True  True   True   True    True
3  True  False  True  False  False    True
4  True  False  True  False   True    True
5  True   True  True   True   True    True
6  True   True  True   True   True    True
7  True  False  True  False   True    True
8  True   True  True  False   True    True
9  True   True  True   True   True    True
        Date    Open    High     Low   Close   Volume
0  21-Jun-17  953.64  960.10  950.76  959.45  1200693
1  20-Jun-17  957.52  961.62  950.01  950.63  1125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1533336
3  16-Jun-17     NaN  942.04     NaN     NaN  3094711
4  15-Jun-17     NaN  943.34     NaN  942.31  2133050
5  14-Jun-17  959.92  961.15  942.25  950.76  1489715
6  13-Jun-17  951.91  959.98  944.09  953.40  2013337
7  12-Jun-17     NaN  949.36     NaN  942.90  3763529
8   9-Jun-17  984.50  984.50     NaN  949.83  3309389
9   8-Jun-17  982.35  984.57  977.20  983.41  1481916
```

In [43]:
```
# another way to filter is with isin()
# syntax only
# df2[df2['E'].isin(['two','four'])]
```

In [44]:
```
# like Numpy, sometimes you need an actual copy, not a view or slice of the same
dfg2 = dfg.copy()
dfg2[0:5]
```

Out[44]:

	Date	Open	High	Low	Close	Volume
0	21-Jun-17	953.64	960.10	950.76	959.45	1200693
1	20-Jun-17	957.52	961.62	950.01	950.63	1125990
2	19-Jun-17	949.96	959.99	949.05	957.37	1533336
3	16-Jun-17	940.00	942.04	931.60	939.78	3094711
4	15-Jun-17	933.97	943.34	924.44	942.31	2133050

In [ ]:
```
```

Setting Values

In [49]:
```
# Recall
print(dfg.head(10))
```

```
        Date    Open    High     Low   Close   Volume
0  21-Jun-17  953.64  960.10  950.76  959.45  1200693
1  20-Jun-17  957.52  961.62  950.01  950.63  1125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1533336
3  16-Jun-17  940.00  942.04  931.60  939.78  3094711
4  15-Jun-17  933.97  943.34  924.44  942.31  2133050
5  14-Jun-17  959.92  961.15  942.25  950.76  1489715
6  13-Jun-17  951.91  959.98  944.09  953.40  2013337
7  12-Jun-17  939.56  949.36  915.23  942.90  3763529
8   9-Jun-17  984.50  984.50  935.63  949.83  3309389
9   8-Jun-17  982.35  984.57  977.20  983.41  1481916
```

In [52]:
```
# All the ways to view (by location, by index, iat, etc) can also be used to set
dfg['Volume'] = dfg['Volume']/1000.0
print(dfg.head(10))
```

```
        Date    Open    High     Low   Close    Volume
0  21-Jun-17  953.64  960.10  950.76  959.45  1.200693
1  20-Jun-17  957.52  961.62  950.01  950.63  1.125990
2  19-Jun-17  949.96  959.99  949.05  957.37  1.533336
3  16-Jun-17  940.00  942.04  931.60  939.78  3.094711
4  15-Jun-17  933.97  943.34  924.44  942.31  2.133050
5  14-Jun-17  959.92  961.15  942.25  950.76  1.489715
6  13-Jun-17  951.91  959.98  944.09  953.40  2.013337
7  12-Jun-17  939.56  949.36  915.23  942.90  3.763529
8   9-Jun-17  984.50  984.50  935.63  949.83  3.309389
9   8-Jun-17  982.35  984.57  977.20  983.41  1.481916
```

In [53]:
```
dfg['Volume'] = 0
print(dfg.head(10))
```

```
        Date    Open    High     Low   Close  Volume
0  21-Jun-17  953.64  960.10  950.76  959.45       0
1  20-Jun-17  957.52  961.62  950.01  950.63       0
2  19-Jun-17  949.96  959.99  949.05  957.37       0
3  16-Jun-17  940.00  942.04  931.60  939.78       0
4  15-Jun-17  933.97  943.34  924.44  942.31       0
5  14-Jun-17  959.92  961.15  942.25  950.76       0
6  13-Jun-17  951.91  959.98  944.09  953.40       0
7  12-Jun-17  939.56  949.36  915.23  942.90       0
8   9-Jun-17  984.50  984.50  935.63  949.83       0
9   8-Jun-17  982.35  984.57  977.20  983.41       0
```

In [57]:
```
dfg.iat[0,1] = 0
dfg.head(3)
```

Out[57]:

	Date	Open	High	Low	Close	Volume
0	21-Jun-17	0.00	960.10	950.76	959.45	0
1	20-Jun-17	957.52	961.62	950.01	950.63	0
2	19-Jun-17	949.96	959.99	949.05	957.37	0

In [ ]:
```

```

In [ ]:
```
# Comments on dropping and filling NaN values
# A view where we drop any rows with value NnN
# dfg.dropna(how='any')  # this would be used to drop rows with Nan
# df1.fillna(value=5)    # this would be used to fill NaN values with 5
```

In [ ]:
```

```

More Statistics and Operations

In [60]:
```
dfg.mean()   # mean by column, also try var()
```

Out[60]:
```
Open      807.545777
High      816.271036
Low       806.340159
Close     811.818645
Volume      0.000000
dtype: float64
```

In [62]:
```
dfg[0:5].mean(1)
# dfg.mean(axis = 1)
```

Out[62]:
```
0    574.062
1    763.956
2    763.274
3    750.684
4    748.812
dtype: float64
```

In [65]:
```
# Use the apply method to perform calculations on every element
dfg2[0:10].apply(np.cumsum)
```

Out[65]:

	Date	Open	High	Low	Close	Volume
0	21-Jun-17	953.64	960.10	950.76	959.45	1200693
1	21-Jun-1720-Jun-17	1911.16	1921.72	1900.77	1910.08	2326683
2	21-Jun-1720-Jun-1719-Jun-17	2861.12	2881.71	2849.82	2867.45	3860019
3	21-Jun-1720-Jun-1719-Jun-1716-Jun-17	3801.12	3823.75	3781.42	3807.23	6954730
4	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-17	4735.09	4767.09	4705.86	4749.54	9087780
5	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-171...	5695.01	5728.24	5648.11	5700.30	10577495
6	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-171...	6646.92	6688.22	6592.20	6653.70	12590832
7	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-171...	7586.48	7637.58	7507.43	7596.60	16354361
8	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-171...	8570.98	8622.08	8443.06	8546.43	19663750
9	21-Jun-1720-Jun-1719-Jun-1716-Jun-1715-Jun-171...	9553.33	9606.65	9420.26	9529.84	21145666

In [155]:
```
# Reload
dfg = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=goog')
dfa = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=aapl')
dfm = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=msft')
dfd = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=dis')
dfn = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=nke')
dfb = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=ba')
```

In [156]:
```
print (dfb.head())
```

```
        Date    Open    High     Low   Close   Volume
0  21-Jun-17  199.58  199.71  197.56  199.17  3151125
1  20-Jun-17  198.75  201.24  198.31  198.33  3077408
2  19-Jun-17  197.88  199.47  197.00  199.08  2726416
3  16-Jun-17  196.01  197.95  195.81  196.44  6939081
4  15-Jun-17  191.43  195.55  191.37  195.45  2894081
```

In [157]:
```
# Rename columns
dfg = dfg.rename(columns = {'Close':'GOOG'})
#print (dfg.head())

dfa = dfa.rename(columns = {'Close':'AAPL'})
#print (dfa.head())

dfm = dfm.rename(columns = {'Close':'MSFT'})
#print (dfm.head())

dfd = dfd.rename(columns = {'Close':'DIS'})
#print (dfd.head())

dfn = dfn.rename(columns = {'Close':'NKE'})
#print (dfn.head())

dfb = dfb.rename(columns = {'Close':'BA'})
print (dfb.head())
```

```
        Date    Open    High     Low      BA   Volume
0  21-Jun-17  199.58  199.71  197.56  199.17  3151125
1  20-Jun-17  198.75  201.24  198.31  198.33  3077408
2  19-Jun-17  197.88  199.47  197.00  199.08  2726416
3  16-Jun-17  196.01  197.95  195.81  196.44  6939081
4  15-Jun-17  191.43  195.55  191.37  195.45  2894081
```

In [159]:
```
# Lets merge some tables
df = dfg[['Date','GOOG']].merge(dfa[['Date','AAPL']])
df = df.merge(dfm[['Date','MSFT']])
df = df.merge(dfd[['Date','DIS']])
df = df.merge(dfn[['Date','NKE']])
df = df.merge(dfb[['Date','BA']])
print (df[0:10])
```

```
        Date    GOOG    AAPL   MSFT     DIS    NKE      BA
0  21-Jun-17  959.45  145.87  70.27  104.80  52.59  199.17
1  20-Jun-17  950.63  145.01  69.91  103.94  51.56  198.33
2  19-Jun-17  957.37  146.34  70.87  105.37  52.02  199.08
3  16-Jun-17  939.78  142.27  70.00  105.51  51.10  196.44
4  15-Jun-17  942.31  144.29  69.90  105.98  52.90  195.45
5  14-Jun-17  950.76  145.16  70.27  106.14  54.66  192.38
6  13-Jun-17  953.40  146.59  70.65  106.56  54.31  191.09
7  12-Jun-17  942.90  145.42  69.78  107.04  54.03  190.00
8   9-Jun-17  949.83  148.98  70.32  105.62  53.46  190.03
9   8-Jun-17  983.41  154.99  71.95  104.32  53.20  189.93
```

In [175]:
```
# show a correlation matrix (pearson)
crl = df.corr()
print (crl)
print()
print (crl.sort_values(by='GOOG'))

df[30:120].corr()
```

```
          GOOG      AAPL      MSFT       DIS       NKE        BA
GOOG  1.000000  0.913180  0.923314  0.577681 -0.087191  0.848205
AAPL  0.913180  1.000000  0.930165  0.765451  0.016812  0.930749
MSFT  0.923314  0.930165  1.000000  0.759337 -0.127814  0.941887
DIS   0.577681  0.765451  0.759337  1.000000  0.187045  0.872439
NKE  -0.087191  0.016812 -0.127814  0.187045  1.000000 -0.025626
BA    0.848205  0.930749  0.941887  0.872439 -0.025626  1.000000

          GOOG      AAPL      MSFT       DIS       NKE        BA
NKE  -0.087191  0.016812 -0.127814  0.187045  1.000000 -0.025626
DIS   0.577681  0.765451  0.759337  1.000000  0.187045  0.872439
BA    0.848205  0.930749  0.941887  0.872439 -0.025626  1.000000
AAPL  0.913180  1.000000  0.930165  0.765451  0.016812  0.930749
MSFT  0.923314  0.930165  1.000000  0.759337 -0.127814  0.941887
GOOG  1.000000  0.913180  0.923314  0.577681 -0.087191  0.848205
```

Out[175]:

	GOOG	AAPL	MSFT	DIS	NKE	BA
GOOG	1.000000	0.721274	0.905010	0.581947	0.308513	0.744658
AAPL	0.721274	1.000000	0.802563	0.841859	0.650994	0.937443
MSFT	0.905010	0.802563	1.000000	0.755536	0.285074	0.788427
DIS	0.581947	0.841859	0.755536	1.000000	0.524095	0.803436
NKE	0.308513	0.650994	0.285074	0.524095	1.000000	0.713459
BA	0.744658	0.937443	0.788427	0.803436	0.713459	1.000000

Homework Section

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1) Read in a CSV file called 'data3.csv' into a dataframe called df.

2) Display the first few lines

In [32]:
```
df = pd.read_csv('data3.csv')
df.head(3)
```

Out[32]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol	Md
0	Argentina	9.0	Total area of the country	4100.0	1962.0	278040.0	E	NaN
1	Argentina	9.0	Total area of the country	4100.0	1967.0	278040.0	E	NaN
2	Argentina	9.0	Total area of the country	4100.0	1972.0	278040.0	E	NaN

3) If you inspect the whole file in an editor, you will see that the bottom 8 lines in the frame need to be dropped. Remove those 8 lines from df. You can also use df.tail() to see the last lines.

In [33]:
```
df = df[:-8]
df.tail(3)
```

Out[33]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol	Md
387	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1992.0	1020.0	E	NaN
388	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1996.0	1005.0	E	NaN
389	United States of America	231.0	National Rainfall Index (NRI)	4472.0	2002.0	938.7	E	NaN

4) Drop columns: drop the columns Area Id, Variable Id, Symbol, and Md. Hint Google 'delete a column in pandas

In [38]:
```
df = df.drop(df[['Area Id', 'Variable Id', 'Symbol', 'Md']], axis=1)
df.head(3)
```

Out[38]:

	Area	Variable Name	Year	Value
0	Argentina	Total area of the country	1962.0	278040.0
1	Argentina	Total area of the country	1967.0	278040.0
2	Argentina	Total area of the country	1972.0	278040.0

5) # Convert the year column to pandas datetime?

Convert the 'Year' column string values to pandas datetime objects, where only the year is specified. To do this, simply run the code below (please review it carefully and make sure you understand what happens):

```
df['Year'] =
pd.to_datetime(pd.Series(df['Year']).astype(int),format='%Y').dt.year
Run df.tail() to see if you get what you expect
```

In [40]:
```
df['Year'] = pd.to_datetime(pd.Series(df['Year']).astype(int),format='%Y').dt.ye
df.tail(3)
```

Out[40]:

	Area	Variable Name	Year	Value
387	United States of America	National Rainfall Index (NRI)	1992	1020.0
388	United States of America	National Rainfall Index (NRI)	1996	1005.0
389	United States of America	National Rainfall Index (NRI)	2002	938.7

6) Print rows where Area is Iceland

In [44]:
```
print(df[df.Area=='Iceland'])
```

```
        Area                  Variable Name  Year         Value
166  Iceland      Total area of the country  1962  1.030000e+04
167  Iceland      Total area of the country  1967  1.030000e+04
168  Iceland      Total area of the country  1972  1.030000e+04
169  Iceland      Total area of the country  1977  1.030000e+04
170  Iceland      Total area of the country  1982  1.030000e+04
171  Iceland      Total area of the country  1987  1.030000e+04
172  Iceland      Total area of the country  1992  1.030000e+04
173  Iceland      Total area of the country  1997  1.030000e+04
174  Iceland      Total area of the country  2002  1.030000e+04
175  Iceland      Total area of the country  2007  1.030000e+04
176  Iceland      Total area of the country  2012  1.030000e+04
177  Iceland      Total area of the country  2014  1.030000e+04
178  Iceland               Total population  1962  1.826000e+02
179  Iceland               Total population  1967  1.974000e+02
180  Iceland               Total population  1972  2.099000e+02
181  Iceland               Total population  1977  2.221000e+02
182  Iceland               Total population  1982  2.331000e+02
183  Iceland               Total population  1987  2.469000e+02
184  Iceland               Total population  1992  2.599000e+02
185  Iceland               Total population  1997  2.728000e+02
186  Iceland               Total population  2002  2.869000e+02
187  Iceland               Total population  2007  3.054000e+02
188  Iceland               Total population  2012  3.234000e+02
189  Iceland               Total population  2015  3.294000e+02
190  Iceland            Population density  1962  1.773000e+00
191  Iceland            Population density  1967  1.917000e+00
192  Iceland            Population density  1972  2.038000e+00
193  Iceland            Population density  1977  2.156000e+00
194  Iceland            Population density  1982  2.263000e+00
195  Iceland            Population density  1987  2.397000e+00
196  Iceland            Population density  1992  2.523000e+00
197  Iceland            Population density  1997  2.649000e+00
198  Iceland            Population density  2002  2.785000e+00
199  Iceland            Population density  2007  2.965000e+00
200  Iceland            Population density  2012  3.140000e+00
201  Iceland            Population density  2015  3.198000e+00
202  Iceland   Gross Domestic Product (GDP)  1962  2.849165e+08
203  Iceland   Gross Domestic Product (GDP)  1967  6.212260e+08
204  Iceland   Gross Domestic Product (GDP)  1972  8.465069e+08
205  Iceland   Gross Domestic Product (GDP)  1977  2.226539e+09
206  Iceland   Gross Domestic Product (GDP)  1982  3.232804e+09
207  Iceland   Gross Domestic Product (GDP)  1987  5.565384e+09
208  Iceland   Gross Domestic Product (GDP)  1992  7.138788e+09
209  Iceland   Gross Domestic Product (GDP)  1997  7.596126e+09
210  Iceland   Gross Domestic Product (GDP)  2002  9.161798e+09
211  Iceland   Gross Domestic Product (GDP)  2007  2.129384e+10
212  Iceland   Gross Domestic Product (GDP)  2012  1.419452e+10
213  Iceland   Gross Domestic Product (GDP)  2015  1.659849e+10
214  Iceland  National Rainfall Index (NRI)  1967  8.160000e+02
215  Iceland  National Rainfall Index (NRI)  1971  9.632000e+02
216  Iceland  National Rainfall Index (NRI)  1975  1.010000e+03
217  Iceland  National Rainfall Index (NRI)  1981  9.326000e+02
218  Iceland  National Rainfall Index (NRI)  1986  9.685000e+02
```

```
219  Iceland  National Rainfall Index (NRI)  1991  1.095000e+03
220  Iceland  National Rainfall Index (NRI)  1997  9.932000e+02
221  Iceland  National Rainfall Index (NRI)  1998  9.234000e+02
```

7) Print the years when the National Rainfall Index (NRI) was greater than 950 or less than 900 in Iceland, by using this code. Put it ina dataframe called df_temp so you don't change the original 'df'

In [58]:
```
df_temp = df[(df.Area=='Iceland') & ((df.Value<900)|(df.Value>950))]
print(df_temp.Year)
```

```
166    1962
167    1967
168    1972
169    1977
170    1982
171    1987
172    1992
173    1997
174    2002
175    2007
176    2012
177    2014
178    1962
179    1967
180    1972
181    1977
182    1982
183    1987
184    1992
185    1997
186    2002
187    2007
188    2012
189    2015
190    1962
191    1967
192    1972
193    1977
194    1982
195    1987
196    1992
197    1997
198    2002
199    2007
200    2012
201    2015
202    1962
203    1967
204    1972
205    1977
206    1982
207    1987
208    1992
209    1997
210    2002
211    2007
212    2012
213    2015
214    1967
215    1971
216    1975
218    1986
219    1991
```

```
220    1997
Name: Year, dtype: int64
```

8) Get al the rows of the original df area is United States of America

a) put the USA into a df_usa data frame

b) Add the columns to ['GDP','NRI','PD','Area','Population']

c) delending on the 'Variable Name' put the value into respective column 'GDP','NRI','PD','Area', or 'Population'

Look into this syntax in case its helpful: df['color'] = np.where(df['Set']=='Z', 'green', 'red')

d) Find the maximum value and minimum value of the 'NRI' column in the US (using pandas methods). What years do the min and max values occur?

In [160]:
```
import warnings
warnings.filterwarnings('ignore')

df_usa = df[df.Area == 'United States of America']
df_usa
df_usa['NRI'] = df[['Value']].where(df['Variable Name']=='National Rainfall Inde
df_usa['GDP'] = df[['Value']].where(df['Variable Name']=='Gross Domestic Product
df_usa['PD'] = df[['Value']].where(df['Variable Name']=='Population density')
df_usa['Area'] = df[['Value']].where(df['Variable Name']=='Total area of the cou
df_usa['Population'] = df[['Value']].where(df['Variable Name']=='Total populatic
df_usa
```

Out[160]:

	Area	Variable Name	Year	Value	NRI	GDP	PD	Population
334	962909.0	Total area of the country	1962	9.629090e+05	NaN	NaN	NaN	NaN
335	962909.0	Total area of the country	1967	9.629090e+05	NaN	NaN	NaN	NaN
336	962909.0	Total area of the country	1972	9.629090e+05	NaN	NaN	NaN	NaN
337	962909.0	Total area of the country	1977	9.629090e+05	NaN	NaN	NaN	NaN
338	962909.0	Total area of the country	1982	9.629090e+05	NaN	NaN	NaN	NaN
339	962909.0	Total area of the country	1987	9.629090e+05	NaN	NaN	NaN	NaN
340	962909.0	Total area of the country	1992	9.629090e+05	NaN	NaN	NaN	NaN
341	962909.0	Total area of the country	1997	9.629090e+05	NaN	NaN	NaN	NaN
342	963203.0	Total area of the country	2002	9.632030e+05	NaN	NaN	NaN	NaN
343	963203.0	Total area of the country	2007	9.632030e+05	NaN	NaN	NaN	NaN
344	983151.0	Total area of the country	2012	9.831510e+05	NaN	NaN	NaN	NaN
345	983151.0	Total area of the country	2014	9.831510e+05	NaN	NaN	NaN	NaN
346	NaN	Total population	1962	1.918610e+05	NaN	NaN	NaN	191861.0
347	NaN	Total population	1967	2.037130e+05	NaN	NaN	NaN	203713.0

	Area	Variable Name	Year	Value	NRI	GDP	PD	Population
348	NaN	Total population	1972	2.132200e+05	NaN	NaN	NaN	213220.0
349	NaN	Total population	1977	2.230910e+05	NaN	NaN	NaN	223091.0
350	NaN	Total population	1982	2.339540e+05	NaN	NaN	NaN	233954.0
351	NaN	Total population	1987	2.454250e+05	NaN	NaN	NaN	245425.0
352	NaN	Total population	1992	2.579080e+05	NaN	NaN	NaN	257908.0
353	NaN	Total population	1997	2.728830e+05	NaN	NaN	NaN	272883.0
354	NaN	Total population	2002	2.884710e+05	NaN	NaN	NaN	288471.0
355	NaN	Total population	2007	3.016560e+05	NaN	NaN	NaN	301656.0
356	NaN	Total population	2012	3.147990e+05	NaN	NaN	NaN	314799.0
357	NaN	Total population	2015	3.217740e+05	NaN	NaN	NaN	321774.0
358	NaN	Population density	1962	1.993000e+01	NaN	NaN	19.93	NaN
359	NaN	Population density	1967	2.116000e+01	NaN	NaN	21.16	NaN
360	NaN	Population density	1972	2.214000e+01	NaN	NaN	22.14	NaN
361	NaN	Population density	1977	2.317000e+01	NaN	NaN	23.17	NaN
362	NaN	Population density	1982	2.430000e+01	NaN	NaN	24.30	NaN
363	NaN	Population density	1987	2.549000e+01	NaN	NaN	25.49	NaN
364	NaN	Population density	1992	2.678000e+01	NaN	NaN	26.78	NaN
365	NaN	Population density	1997	2.834000e+01	NaN	NaN	28.34	NaN

	Area	Variable Name	Year	Value	NRI	GDP	PD	Population
366	NaN	Population density	2002	2.995000e+01	NaN	NaN	29.95	NaN
367	NaN	Population density	2007	3.132000e+01	NaN	NaN	31.32	NaN
368	NaN	Population density	2012	3.202000e+01	NaN	NaN	32.02	NaN
369	NaN	Population density	2015	3.273000e+01	NaN	NaN	32.73	NaN
370	NaN	Gross Domestic Product (GDP)	1962	6.050000e+11	NaN	6.050000e+11	NaN	NaN
371	NaN	Gross Domestic Product (GDP)	1967	8.620000e+11	NaN	8.620000e+11	NaN	NaN
372	NaN	Gross Domestic Product (GDP)	1972	1.280000e+12	NaN	1.280000e+12	NaN	NaN
373	NaN	Gross Domestic Product (GDP)	1977	2.090000e+12	NaN	2.090000e+12	NaN	NaN
374	NaN	Gross Domestic Product (GDP)	1982	3.340000e+12	NaN	3.340000e+12	NaN	NaN
375	NaN	Gross Domestic Product (GDP)	1987	4.870000e+12	NaN	4.870000e+12	NaN	NaN
376	NaN	Gross Domestic Product (GDP)	1992	6.540000e+12	NaN	6.540000e+12	NaN	NaN
377	NaN	Gross Domestic Product (GDP)	1997	8.610000e+12	NaN	8.610000e+12	NaN	NaN

	Area	Variable Name	Year	Value	NRI	GDP	PD	Population
378	NaN	Gross Domestic Product (GDP)	2002	1.100000e+13	NaN	1.100000e+13	NaN	NaN
379	NaN	Gross Domestic Product (GDP)	2007	1.450000e+13	NaN	1.450000e+13	NaN	NaN
380	NaN	Gross Domestic Product (GDP)	2012	1.620000e+13	NaN	1.620000e+13	NaN	NaN
381	NaN	Gross Domestic Product (GDP)	2015	1.790000e+13	NaN	1.790000e+13	NaN	NaN
382	NaN	National Rainfall Index (NRI)	1965	9.285000e+02	928.5	NaN	NaN	NaN
383	NaN	National Rainfall Index (NRI)	1969	9.522000e+02	952.2	NaN	NaN	NaN
384	NaN	National Rainfall Index (NRI)	1974	1.008000e+03	1008.0	NaN	NaN	NaN
385	NaN	National Rainfall Index (NRI)	1981	9.492000e+02	949.2	NaN	NaN	NaN
386	NaN	National Rainfall Index (NRI)	1984	9.746000e+02	974.6	NaN	NaN	NaN
387	NaN	National Rainfall Index (NRI)	1992	1.020000e+03	1020.0	NaN	NaN	NaN
388	NaN	National Rainfall Index (NRI)	1996	1.005000e+03	1005.0	NaN	NaN	NaN
389	NaN	National Rainfall Index (NRI)	2002	9.387000e+02	938.7	NaN	NaN	NaN

In [161]:
```
print('year maximum NRI is')
print(df_usa.Year.ix[df_usa[['NRI']].idxmax()])
print()
print('year minimum NRI is')
print(df_usa.Year.ix[df_usa[['NRI']].idxmin()])
```

```
year maximum NRI is
387    1992
Name: Year, dtype: int64

year minimum NRI is
382    1965
Name: Year, dtype: int64
```

In [162]:
```
print('group by year')
df_usa1 = df_usa.fillna(0)
df_usa1 = df_usa1.groupby(['Year']).sum().replace(0,np.NaN)
df_usa1
```

```
group by year
```

Out[162]:

	Area	Value	NRI	GDP	PD	Population
Year						
1962	962909.0	6.050012e+11	NaN	6.050000e+11	19.93	191861.0
1965	NaN	9.285000e+02	928.5	NaN	NaN	NaN
1967	962909.0	8.620012e+11	NaN	8.620000e+11	21.16	203713.0
1969	NaN	9.522000e+02	952.2	NaN	NaN	NaN
1972	962909.0	1.280001e+12	NaN	1.280000e+12	22.14	213220.0
1974	NaN	1.008000e+03	1008.0	NaN	NaN	NaN
1977	962909.0	2.090001e+12	NaN	2.090000e+12	23.17	223091.0
1981	NaN	9.492000e+02	949.2	NaN	NaN	NaN
1982	962909.0	3.340001e+12	NaN	3.340000e+12	24.30	233954.0
1984	NaN	9.746000e+02	974.6	NaN	NaN	NaN
1987	962909.0	4.870001e+12	NaN	4.870000e+12	25.49	245425.0
1992	962909.0	6.540001e+12	1020.0	6.540000e+12	26.78	257908.0
1996	NaN	1.005000e+03	1005.0	NaN	NaN	NaN
1997	962909.0	8.610001e+12	NaN	8.610000e+12	28.34	272883.0
2002	963203.0	1.100000e+13	938.7	1.100000e+13	29.95	288471.0
2007	963203.0	1.450000e+13	NaN	1.450000e+13	31.32	301656.0
2012	983151.0	1.620000e+13	NaN	1.620000e+13	32.02	314799.0
2014	983151.0	9.831510e+05	NaN	NaN	NaN	NaN
2015	NaN	1.790000e+13	NaN	1.790000e+13	32.73	321774.0

9) Show general statistics for columns 'GDP','NRI','PD', and 'Population'

In [183]:
```
genstat = df_usa[['GDP', 'NRI', 'PD', 'Population']].describe()
print(genstat)
```

```
                GDP          NRI         PD     Population
count  1.200000e+01     8.000000  12.000000      12.000000
mean   7.316417e+12   972.025000  26.444167  255729.583333
std    6.256868e+12    35.068861   4.425996   44281.029610
min    6.050000e+11   928.500000  19.930000  191861.000000
25%    1.887500e+12   946.575000  22.912500  220623.250000
50%    5.705000e+12   963.400000  26.135000  251666.500000
75%    1.187500e+13  1005.750000  30.292500  291767.250000
max    1.790000e+13  1020.000000  32.730000  321774.000000
```

10 a) Show a 3 x 3 correlation matrix for Nike, Apple, and Disney stock prices over the past year

In [172]:
```
dfa = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=aapl')
dfn = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=nke')
dfd = pd.read_csv('https://www.google.com/finance/historical?output=csv&q=dis')
```

In [188]:
```
dfa = dfa.rename(columns = {'Close':'AAPL'})
dfd = dfd.rename(columns = {'Close':'DIS'})
dfn = dfn.rename(columns = {'Close':'NKE'})
df_full = dfa[['Date','AAPL']].merge(dfd[['Date','DIS']])
df_full = df_full.merge(dfn[['Date','NKE']])
corrl = df_full.corr()
print(corrl)
```

```
          AAPL       DIS       NKE
AAPL  1.000000  0.482569  0.554249
DIS   0.482569  1.000000  0.488676
NKE   0.554249  0.488676  1.000000
```

10b) Show the same correlation matrix but over different time periods,

i) the last 20 days ii) the last 80 days

In [190]:
```
print('corr the last 20 days')
print(df_full[:20].corr())
print()
print('corr the last 80 days')
print(df_full[:80].corr())
```

```
corr the last 20 days
          AAPL       DIS       NKE
AAPL  1.000000  0.237467 -0.547436
DIS   0.237467  1.000000  0.240004
NKE  -0.547436  0.240004  1.000000

corr the last 80 days
          AAPL       DIS       NKE
AAPL  1.000000 -0.507881 -0.078793
DIS  -0.507881  1.000000  0.272226
NKE  -0.078793  0.272226  1.000000
```

11) Change the code so that it accepts a list of any stock symbols, ie ['NKE', 'APPL', 'DIS', ...] and creates a correlation matrix for the time period of the past 100 days

In [232]:
```
# Insert list of companies here. Should be >=2 companies
l = ['AAPL', 'NKE', 'DIS', 'GOOG']
df1  = pd.read_csv('https://www.google.com/finance/historical?output=csv&q='+l[0
df2  = pd.read_csv('https://www.google.com/finance/historical?output=csv&q='+l[1
df1 = df1.rename(columns = {'Close':l[0]})
df2 = df2.rename(columns = {'Close':l[1]})

dff = df1[['Date',l[0]]].merge(df2[['Date',l[1]]])

for n in l[2:]:
    dfn = pd.read_csv('https://www.google.com/finance/historical?output=csv&q='+
    dfn = dfn.rename(columns = {'Close':n})
    dff = dff.merge(dfn[['Date', n]])

corr = dff[:100].corr()
print(corr)
```

```
          AAPL       NKE       DIS      GOOG
AAPL  1.000000 -0.068436 -0.529913  0.028501
NKE  -0.068436  1.000000  0.106217 -0.201377
DIS  -0.529913  0.106217  1.000000 -0.091609
GOOG  0.028501 -0.201377 -0.091609  1.000000
```