Advanced Data After Dark Python and Pandas

In [1]:
```
%matplotlib inline
```

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import pandas
```

In [16]:
```
#Object creation
s = pandas.Series([1,3,5,np.nan,6,8])
s
```

Out[16]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

In [17]:
```
#datetime objects week beginning 23rd May 2016
dates = pandas.date_range('20160523',periods=7)
dates
```

Out[17]:
```
DatetimeIndex(['2016-05-23', '2016-05-24', '2016-05-25', '2016-05-26',
               '2016-05-27', '2016-05-28', '2016-05-29'],
              dtype='datetime64[ns]', freq='D')
```

In [26]:
```
#create a DataFrame using the dates array as the index
#fill it with some random values using numpy, and add columns labels.
dataframe1 = pandas.DataFrame(np.random.randn(7,4), index=dates, columns={'Treatm
ent1','Treatment2','Treatment3','Control'})
dataframe1
```

Out[26]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-23	-1.603459	-0.559487	-1.974327	1.338059
2016-05-24	1.434379	-1.639297	0.859592	0.196948
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-27	-1.148925	-0.071029	0.564620	0.312048
2016-05-28	1.861991	0.635728	1.053611	0.683112
2016-05-29	1.401221	1.251225	0.796542	-1.139129

In [27]:

```
#create dataframe based on dictionary
dataframe2 = pandas.DataFrame({ 'A' : 1.,
                            'B' : pandas.Timestamp('20160501'),
                            'C' : pandas.Series(1,index=list(range(4)),dtype='flo
at32'),
                            'D' : np.array([3] * 4,dtype='int32'),
                            'E' : pandas.Categorical(["testing","training","testi
ng","training"]),
                            'F' : 'Valid' })
dataframe2
```

Out[27]:

	A	B	C	D	E	F
0	1.0	2016-05-01	1.0	3	testing	Valid
1	1.0	2016-05-01	1.0	3	training	Valid
2	1.0	2016-05-01	1.0	3	testing	Valid
3	1.0	2016-05-01	1.0	3	training	Valid

In [29]:

```
dataframe1.index
```

Out[29]:

```
DatetimeIndex(['2016-05-23', '2016-05-24', '2016-05-25', '2016-05-26',
               '2016-05-27', '2016-05-28', '2016-05-29'],
              dtype='datetime64[ns]', freq='D')
```

In [30]:

```
dataframe1.columns
```

Out[30]:

```
Index([u'Control', u'Treatment2', u'Treatment3', u'Treatment1'], dtype='object')
```

In [31]:

```
dataframe1.values
```

Out[31]:

```
array([[-1.6034594 , -0.55948726, -1.97432717,  1.33805925],
       [ 1.4343786 , -1.63929704,  0.85959167,  0.19694766],
       [ 0.80653708,  0.31811149,  0.2486886 ,  0.27855941],
       [ 0.47219173,  0.37247168,  0.31405396,  1.49255787],
       [-1.14892484, -0.07102856,  0.56461981,  0.31204801],
       [ 1.86199147,  0.63572778,  1.05361112,  0.6831123 ],
       [ 1.40122117,  1.25122535,  0.7965424 , -1.1391292 ]])
```

In [35]:

```
#Selection
 #To select only first few rows
dataframe1.head()
```

Out[35]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-23	-1.603459	-0.559487	-1.974327	1.338059
2016-05-24	1.434379	-1.639297	0.859592	0.196948
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-27	-1.148925	-0.071029	0.564620	0.312048

In [38]:

```
# To select last 3 rows
dataframe1.tail(3)
```

Out[38]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-27	-1.148925	-0.071029	0.564620	0.312048
2016-05-28	1.861991	0.635728	1.053611	0.683112
2016-05-29	1.401221	1.251225	0.796542	-1.139129

In [39]:

```
# Selecting a single column (series)

dataframe1['Control']
```

Out[39]:

```
2016-05-23   -1.603459
2016-05-24    1.434379
2016-05-25    0.806537
2016-05-26    0.472192
2016-05-27   -1.148925
2016-05-28    1.861991
2016-05-29    1.401221
Freq: D, Name: Control, dtype: float64
```

In [41]:

```
#equivalent
dataframe1.Control
```

Out[41]:

```
2016-05-23   -1.603459
2016-05-24    1.434379
2016-05-25    0.806537
2016-05-26    0.472192
2016-05-27   -1.148925
2016-05-28    1.861991
2016-05-29    1.401221
Freq: D, Name: Control, dtype: float64
```

In [43]:

```
# Subset rows (Slicing)

dataframe1[1:2]
```

Out[43]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-24	1.434379	-1.639297	0.859592	0.196948

In [44]:

```
#Data range slice
dataframe1['20160524':'20160528']
```

Out[44]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-24	1.434379	-1.639297	0.859592	0.196948
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-27	-1.148925	-0.071029	0.564620	0.312048
2016-05-28	1.861991	0.635728	1.053611	0.683112

In [52]:

```
#Selection of more than 1 column
dataframe1.loc[:,['Control','Treatment1']]
```

Out[52]:

	Control	Treatment1
2016-05-23	-1.603459	1.338059
2016-05-24	1.434379	0.196948
2016-05-25	0.806537	0.278559
2016-05-26	0.472192	1.492558
2016-05-27	-1.148925	0.312048
2016-05-28	1.861991	0.683112
2016-05-29	1.401221	-1.139129

In [54]:

```
dataframe1.loc['20160524':'20160528',['Control','Treatment1']]
```

Out[54]:

	Control	Treatment1
2016-05-24	1.434379	0.196948
2016-05-25	0.806537	0.278559
2016-05-26	0.472192	1.492558
2016-05-27	-1.148925	0.312048
2016-05-28	1.861991	0.683112

In [55]:

```
### Boolean Indexing

# Select all rows that meet some criteria.


dataframe1[dataframe1['Control'] > 0]
```

Out[55]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-24	1.434379	-1.639297	0.859592	0.196948
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-28	1.861991	0.635728	1.053611	0.683112
2016-05-29	1.401221	1.251225	0.796542	-1.139129

In [61]:

```
#Remove all negative values
nonnegative_only = dataframe1[dataframe1 > 0]
nonnegative_only
```

Out[61]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-23	NaN	NaN	NaN	1.338059
2016-05-24	1.434379	NaN	0.859592	0.196948
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-27	NaN	NaN	0.564620	0.312048
2016-05-28	1.861991	0.635728	1.053611	0.683112
2016-05-29	1.401221	1.251225	0.796542	NaN

In [62]:

```
#If we remove NAs
nonnegative_only.dropna()
```

Out[62]:

	Control	Treatment2	Treatment3	Treatment1
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-28	1.861991	0.635728	1.053611	0.683112

In [63]:

```
#Nonnumeric data
dataframe2[dataframe2['E'].isin(['testing'])]
```

Out[63]:

	A	B	C	D	E	F
0	1.0	2016-05-01	1.0	3	testing	Valid
2	1.0	2016-05-01	1.0	3	testing	Valid

In [65]:

```
#Combining DataFrames
dframe_one = pandas.DataFrame(np.random.randn(5, 4))
dframe_one
```

Out[65]:

	0	1	2	3
0	-1.291165	0.888212	0.456905	0.608305
1	1.253618	0.886168	-1.119199	-0.972536
2	1.398617	0.894724	0.204543	-0.491903
3	-0.073117	-0.531563	0.400757	0.818488
4	0.330302	0.972307	0.834731	-0.956549

In [66]:

```
dframe_two = pandas.DataFrame(np.random.randn(5, 4))
dframe_two
```

Out[66]:

	0	1	2	3
0	0.093893	-0.712054	-0.923578	0.585721
1	-1.693582	-0.038767	0.865429	0.924333
2	-1.066737	0.198798	-2.252600	0.645166
3	1.281326	-0.082939	0.446806	-1.987437
4	0.253514	-0.879641	0.854847	-0.694206

In [68]:

```
pandas.concat([dframe_one, dframe_two])
```

Out[68]:

	0	1	2	3
0	-1.291165	0.888212	0.456905	0.608305
1	1.253618	0.886168	-1.119199	-0.972536
2	1.398617	0.894724	0.204543	-0.491903
3	-0.073117	-0.531563	0.400757	0.818488
4	0.330302	0.972307	0.834731	-0.956549
0	0.093893	-0.712054	-0.923578	0.585721
1	-1.693582	-0.038767	0.865429	0.924333
2	-1.066737	0.198798	-2.252600	0.645166
3	1.281326	-0.082939	0.446806	-1.987437
4	0.253514	-0.879641	0.854847	-0.694206

In [69]:

```
# when data frames are not identifical in sturcture but share a common key

left = pandas.DataFrame({'key': ['CT', 'EXP'], 'lval': [1, 2]})
left
right = pandas.DataFrame({'key': ['CT', 'CT', 'EXP'], 'rval': [3, 4, 5]})
right
pandas.merge(left, right, on='key')
```

Out[69]:

	key	lval	rval
0	CT	1	3
1	CT	1	4
2	EXP	2	5

In [71]:

```
### Create a df for grouping

CT_EXP = pandas.DataFrame({'A' : ['CT', 'EXP', 'CT', 'EXP','CT', 'EXP', 'CT', 'CT
'],
                           'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'o
ne', 'three'],
                           'C' : np.random.randn(8),
                           'D' : np.random.randn(8)})

CT_EXP
```

Out[71]:

	A	B	C	D
0	CT	one	-0.239759	0.615669
1	EXP	one	0.214751	1.244051
2	CT	two	0.117444	-0.965550
3	EXP	three	1.668322	0.060816
4	CT	two	0.657154	1.612214
5	EXP	two	-0.175435	0.717797
6	CT	one	-0.740445	1.688249
7	CT	three	1.059128	-1.043676

In [72]:

```
#group by A and then sum
CT_EXP.groupby('A').sum()
```

Out[72]:

	C	D
A		
CT	0.853521	1.906905
EXP	1.707638	2.022664

In [73]:

```
#retain B
grouped = CT_EXP.groupby(['A','B']).sum()
grouped
```

Out[73]:

		C	D
A	**B**		
CT	**one**	-0.980205	2.303918
	three	1.059128	-1.043676
	two	0.774597	0.646664
EXP	**one**	0.214751	1.244051
	three	1.668322	0.060816
	two	-0.175435	0.717797

In [75]:

```
#compression
stacked = grouped.stack()
stacked
```

Out[75]:

```
A    B
CT   one    C   -0.980205
            D    2.303918
     three  C    1.059128
            D   -1.043676
     two    C    0.774597
            D    0.646664
EXP  one    C    0.214751
            D    1.244051
     three  C    1.668322
            D    0.060816
     two    C   -0.175435
            D    0.717797
dtype: float64
```

In [77]:

```
#Transformations
#Sort the rows (axis = 0) by their index/header values:
dataframe1.sort_index(axis=0, ascending=False)
```

Out[77]:

	Control	**Treatment2**	**Treatment3**	**Treatment1**
2016-05-29	1.401221	1.251225	0.796542	-1.139129
2016-05-28	1.861991	0.635728	1.053611	0.683112
2016-05-27	-1.148925	-0.071029	0.564620	0.312048
2016-05-26	0.472192	0.372472	0.314054	1.492558
2016-05-25	0.806537	0.318111	0.248689	0.278559
2016-05-24	1.434379	-1.639297	0.859592	0.196948
2016-05-23	-1.603459	-0.559487	-1.974327	1.338059

In [78]:

```
dataframe1.sort_index(axis=1)
```

Out[78]:

	Control	**Treatment1**	**Treatment2**	**Treatment3**
2016-05-23	-1.603459	1.338059	-0.559487	-1.974327
2016-05-24	1.434379	0.196948	-1.639297	0.859592
2016-05-25	0.806537	0.278559	0.318111	0.248689
2016-05-26	0.472192	1.492558	0.372472	0.314054
2016-05-27	-1.148925	0.312048	-0.071029	0.564620
2016-05-28	1.861991	0.683112	0.635728	1.053611
2016-05-29	1.401221	-1.139129	1.251225	0.796542

In [80]:

```
# Resampling operations during frequency conversion (converting seconds to minute
s)
#Create array - time interval - convert to minutes and change time zone

One50s = pandas.date_range('05/21/2016', periods=150, freq='S')
time_series = pandas.Series(np.random.randint(0, 500, len(One50s)), index=One50s)
time_series.resample('1Min', how='sum')
ts_utc = time_series.tz_localize('UTC')
ts_utc.head()
ts_utc.tz_convert('US/Eastern').head()
```

```
/Users/mcweeney/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:6: Fu
tureWarning: how in .resample() is deprecated
the new syntax is .resample(...).sum()
```

Out[80]:

```
2016-05-20 20:00:00-04:00    387
2016-05-20 20:00:01-04:00    192
2016-05-20 20:00:02-04:00    404
2016-05-20 20:00:03-04:00     93
2016-05-20 20:00:04-04:00    381
Freq: S, dtype: int64
```

In [81]:

```
ts_utc.to_csv('test.csv')
```

In [82]:

```
new_frame = pandas.read_csv('test.csv')
new_frame.head()
#Check format here - be careful how you read and write files
```

Out[82]:

	2016-05-21 00:00:00+00:00	387
0	2016-05-21 00:00:01+00:00	192
1	2016-05-21 00:00:02+00:00	404
2	2016-05-21 00:00:03+00:00	93
3	2016-05-21 00:00:04+00:00	381
4	2016-05-21 00:00:05+00:00	138

In [83]:

```
new_frame.to_excel('test.xlsx', sheet_name='Sheet1')
pandas.read_excel('test.xlsx', 'Sheet1', index_col=None, na_values=['NA']).head()
```

Out[83]:

	2016-05-21 00:00:00+00:00	387
0	2016-05-21 00:00:01+00:00	192
1	2016-05-21 00:00:02+00:00	404
2	2016-05-21 00:00:03+00:00	93
3	2016-05-21 00:00:04+00:00	381
4	2016-05-21 00:00:05+00:00	138

In [86]:

```
#Querying data from web - example 1
import requests
url = 'https://health.data.ny.gov/resource/dk4z-k3xb.json'
xstr = 'Rate significantly higher than Statewide Rate'
data = requests.get(url).json()
records = [r for r in data if xstr in r['comparison_results']]
print(len(records))
```

```
40
```

In [87]:

```
#Querying data from web - example 2
import re
import requests

formurl = 'http://www.accessdata.fda.gov/scripts/cder/ob/docs/tempai.cfm'
post_params = {'Generic_Name': 'Methadone', 'table1': 'OB_Disc'}
resp = requests.post(formurl, data = post_params)
m = re.search('(?<=Displaying records) *[\d,]+ *to *[\d,]+ *of *([\d,]+)', resp.t
ext)
print(m.groups()[0])
```

```
9
```

In [ ]:

```

```