LAB-10

Hierarchical Indexing

Welcome to Jupyter!

In [1]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
# Creation of a Series Object pop2000 representing the population in ye
index = ['California','New York','Texas']
populations = [33871648, 18976457, 20851820]
pop2000 = pd.Series(populations, index=index)
pop2000
```

Out[3]:
```
California    33871648
New York      18976457
Texas         20851820
dtype: int64
```

In [4]:
```
# Creation of a Series Object pop2010 representing the population in ye
index = ['California','New York','Texas']
populations = [37253956, 19378102, 25145561]
pop2010 = pd.Series(populations, index=index)
pop2010
```

Out[4]:
```
California    37253956
New York      19378102
Texas         25145561
dtype: int64
```

In [5]:
```
index=[('California',2000),('California',2010),('New York',2000),('New
Populations=[33871648,37253956,18975487,77755584,36512487,25498724]
pop = pd.Series(Populations,index=index)
pop
```

Out[5]:
```
(California, 2000)    33871648
(California, 2010)    37253956
(New York, 2000)      18975487
(New York, 2010)      77755584
(Texas, 2000)         36512487
(Texas, 2010)         25498724
dtype: int64
```

In [6]:
```
pop[('California',2010):('Texas',2000)]
```

Out[6]:
```
(California, 2010)    37253956
(New York, 2000)      18975487
(New York, 2010)      77755584
(Texas, 2000)         36512487
dtype: int64
```

In [7]:
```
pop[[i for i in pop.index if i[1] == 2010]]
```

Out[7]:
```
(California, 2010)    37253956
(New York, 2010)      77755584
(Texas, 2010)         25498724
```

In [9]:
```
index = pd.MultiIndex.from_tuples(index)
index
```

Out[9]:
```
MultiIndex([('California', 2000),
            ('California', 2010),
            (  'New York', 2000),
            (  'New York', 2010),
            (     'Texas', 2000),
            (     'Texas', 2010)],
           )
```

In [11]:
```
pop = pop.reindex(index)
pop
```

Out[11]:
```
California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
Texas       2000    36512487
            2010    25498724
dtype: int64
```

In [12]:
```
pop[:,2010]
```

Out[12]:
```
California    37253956
New York      77755584
Texas         25498724
dtype: int64
```

In [13]:
```
pop['California']
```

Out[13]:
```
2000    33871648
2010    37253956
dtype: int64
```

In [14]:
```
#print pop
print("Mulpy indexed Series Object 'pop'")
print('\n')
print(pop)
print("\n")
#convert series object 'pop' to the data frame object
pop_df=pop.unstack()
#print pop_df
print("Conventionally indexed DataFrame Object 'pop_df'")
print("\n")
print(pop_df)
```

```
Mulpy indexed Series Object 'pop'


California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
Texas       2000    36512487
            2010    25498724
dtype: int64
```

```
Conventionally indexed DataFrame Object 'pop_df'


                2000      2010
California  33871648  37253956
New York    18975487  77755584
```

In [15]:
```
pop_df.stack()
```

Out[15]:
```
California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
Texas       2000    36512487
            2010    25498724
dtype: int64
```

In [19]:
```
df=pd.DataFrame(np.random.rand(4,2),
                index=[['a','a','b','b'],[1,2,1,2]],
                columns=['data1','data2'])
df
```

Out[19]:

		data1	data2
a	**1**	0.791097	0.119694
	2	0.477289	0.340659
b	**1**	0.072534	0.802740
	2	0.710597	0.680109

In [20]:
```
data = {('California', 2000): 33871648,
 ('California', 2010): 37253956,
 ('Texas', 2000): 20851820,
 ('Texas', 2010): 25145561,
 ('New York', 2000): 18976457,
 ('New York', 2010): 19378102}
pd.Series(data)
```

Out[20]:
```
California  2000    33871648
            2010    37253956
Texas       2000    20851820
            2010    25145561
New York    2000    18976457
            2010    19378102
dtype: int64
```

In [21]:
```
pop.index.names=['state','year']
pop
```

Out[21]:
```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
```

```
Texas       2000    36512487
            2010    25498724
```

In [22]:
```
pop
```

Out[22]:
```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
Texas       2000    36512487
            2010    25498724
dtype: int64
```

In [24]:
```
pop['California',2000]
```

Out[24]:
```
33871648
```

In [25]:
```
pop['California']
```

Out[25]:
```
year
2000    33871648
2010    37253956
dtype: int64
```

In [30]:
```
pop.loc['California':'New York']
```

Out[30]:
```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18975487
            2010    77755584
dtype: int64
```

In [31]:
```
pop[:,2000]
```

Out[31]:
```
state
California    33871648
New York      18975487
Texas         36512487
dtype: int64
```

In [32]:
```
pop[pop>22000000]
```

Out[32]:
```
state       year
California  2000    33871648
            2010    37253956
New York    2010    77755584
Texas       2000    36512487
            2010    25498724
dtype: int64
```

In [33]:
```
pop[['California','Texas']]
```

Out[33]:
```
state       year
California  2000    33871648
```

```
            2010    37253956
Texas       2000    36512487
            2010    25498724
```

Handling Missing Data

In [1]:
```
import numpy as np
import pandas as pd
vals1 = np.array([1, None, 3, 4])
# Print array
print(vals1)
print("\n")
# Print data type of the array
print(vals1.dtype)
```

```
[1 None 3 4]


object
```

In [2]:
```
vals1.sum()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-2-30a3fc8c6726> in <module>
----> 1 vals1.sum()

/srv/conda/envs/notebook/lib/python3.6/site-packages/numpy/core/_method
s.py in _sum(a, axis, dtype, out, keepdims, initial, where)
     45 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     46          initial=_NoValue, where=True):
---> 47     return umr_sum(a, axis, dtype, out, keepdims, initial, wher
e)
     48
     49 def _prod(a, axis=None, dtype=None, out=None, keepdims=False,

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

In [6]:
```
vals2=np.array([1,np.nan,3,4])
#print array
print(vals2)
print("\n")
#print data type of the array
print(vals2.dtype)
```

```
[ 1. nan  3.  4.]


float64
```

In [7]:
```
1+np.nan
```

Out[7]: nan

In [8]:
```
0*np.nan
```

Out[8]: nan

In [9]:
```
np.nan+np.nan
```

Out[9]:
```
nan
```

In [10]:
```
vals2.sum(),vals2.min(),vals2.max()
```

Out[10]:
```
(nan, nan, nan)
```

In [11]:
```
np.nansum(vals2),np.nanmin(vals2),np.nanmax(vals2)
```

Out[11]:
```
(8.0, 1.0, 4.0)
```

In [12]:
```
pd.Series([1,np.nan,2,None])
```

Out[12]:
```
0    1.0
1    NaN
2    2.0
3    NaN
dtype: float64
```

In [13]:
```
x = pd.Series(np.arange(2), dtype=int)
x
```

Out[13]:
```
0    0
1    1
dtype: int64
```

In [14]:
```
x[0]=None
```

In [17]:
```
data=pd.Series([1,np.nan,'hello',None])
data
```

Out[17]:
```
0        1
1      NaN
2    hello
3     None
dtype: object
```

In [18]:
```
data.isnull()
```

Out[18]:
```
0    False
1     True
2    False
3     True
dtype: bool
```

In [19]:
```
#print the rows with missing letters
data[data.isnull()]
```

Out[19]:
```
1     NaN
3    None
dtype: object
```

In [20]:
```
data.notnull()
```

Out[20]:
```
0     True
1    False
2     True
3    False
dtype: bool
```

In [21]:
```
#print the rows with non missing values
data[data.notnull()]
```

Out[21]:
```
0        1
2    hello
dtype: object
```

In [22]:
```
#print the rows with non missing values
data.dropna()
```

Out[22]:
```
0        1
2    hello
dtype: object
```

In [23]:
```
#fill the missing values with zero
data.fillna(0)
```

Out[23]:
```
0        1
1        0
2    hello
3        0
dtype: object
```

In [24]:
```
#fill the missing values with 333
data.fillna(333)
```

Out[24]:
```
0        1
1      333
2    hello
3      333
dtype: object
```

In [26]:
```
data = pd.Series([1,np.nan,3,None])
#fill the missing values with mean of remaining values
data.fillna(np.nanmean(data))
```

Out[26]:
```
0    1.0
1    2.0
2    3.0
3    2.0
dtype: float64
```

In [27]:
```
#fill the missing values with standard devaition of remaning values
data.fillna(np.nanstd(data))
```

Out[27]:
```
0    1.0
1    1.0
```

```
2    3.0
3    1.0
```

In [32]:
```
df=pd.DataFrame([[1,      np.nan,2],
                 [2,      3,     5],
                 [np.nan, 4,     6]])
df
```

Out[32]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

In [34]:
```
df.dropna()
```

Out[34]:

	0	1	2
1	2.0	3.0	5

In [35]:
```
df.dropna(axis='columns')
```

Out[35]:

	2
0	2
1	5
2	6

In [37]:
```
df[3]=np.nan
df
```

Out[37]:

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

In [38]:
```
df.dropna(axis='columns',how='all')
```

Out[38]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

In [40]:
```
df.dropna(axis='rows',thresh=3)
```

Out[40]:

	0	1	2	3
1	2.0	3.0	5	NaN

In [42]:
```
data=pd.Series([1,np.nan,2,None,3],index=list('abcde'))
data
```

Out[42]:
```
a    1.0
b    NaN
c    2.0
d    NaN
e    3.0
dtype: float64
```

In [43]:
```
data.fillna(0)
```

Out[43]:
```
a    1.0
b    0.0
c    2.0
d    0.0
e    3.0
dtype: float64
```

In [44]:
```
#forward-fill
data.fillna(method='ffill')
```

Out[44]:
```
a    1.0
b    1.0
c    2.0
d    2.0
e    3.0
dtype: float64
```

In [45]:
```
#back-fill
data.fillna(method='bfill')
```

Out[45]:
```
a    1.0
b    2.0
c    2.0
d    3.0
e    3.0
dtype: float64
```

In [46]:
```
df
```

Out[46]:

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

In [47]:
```
df.fillna(method='ffill',axis=1)
```

Out[47]:

	0	1	2	3
0	1.0	1.0	2.0	2.0
1	2.0	3.0	5.0	5.0
2	NaN	4.0	6.0	6.0