*This document includes sample of code syntax which include libraries of* ***Numpy, Pandas, Matplotlib, Seaborn***

*@ np.where*

* + Series(np.where(pd.isnull(ser1), ser2, ser1), index=ser1.index)

*@ np.combine*

* + ser1.combine\_first(ser2) <Result will be same as above written code>

*@ DataFrame read dictionary data*

* + dframe\_odds = DataFrame({'X': [1., np.nan, 3., np.nan],

'Y': [np.nan, 5., np.nan, 7.],

'Z': [np.nan, 9., np.nan, 11.]})

*@ Use Pandas to create “index with name”*

* + dframe1 = DataFrame(np.arange(8).reshape((2, 4)),

index=**pd.Index**(['LA', 'SF'], name='city'),

columns=pd.Index(['A', 'B', 'C','D'], name='letter'))

*@ DataFrame.stack and unstack*

* + dframe.stack( )
  + dframe.unstack( 0 ) < To specify the row or column)
  + dframe.unstack( ‘letter’ )

***# Now stack will filter out NAN by default***

* + dframe.unstack().stack()

***# IF we don’t want this we can set it to False (NaN show in key data)***

* + dframe.unstack().stack(dropna=**False**)

@ Duplicate Data in DataFrame

* + dframe = DataFrame({'key1': ['A'] **\* 2** + ['B'] **\* 3**,

'key2': [2, 2, 2, 3, 3]})

***#We can use duplicated to find duplicates***

* + dframe.duplicated()

*#* drop\_duplicates() <*Drop duplicates like this:>*

* + dframe.drop\_duplicates()

*#You can filter which duplicates to drop by a single column*

* + dframe.drop\_duplicates(['key1'])

*#By default the first value was taken for the duplicates, we can also take the last value instead*

* + dframe.drop\_duplicates(['key1'],take\_last=**True**)

@ DataFrame.map

* + dframe = DataFrame({'city':['Alma','Brian Head','Fox Park'],'altitude':[3158,3000,2762]})

*#Now let's say we wanted to add a column for the States, we can do that with a mapping.*

* + state\_map={'Alma':'Colorado','Brian Head':'Utah','Fox Park':'Wyoming'}

*# Now we can map that data to our current dframe*

* + dframe['state'] = dframe['city'].map(state\_map)

@ series.replace

# Using replace we can select --> .replace(value to be replaced, new\_value)

* + ser1.replace(1,np.nan)

#Can also input lists

* + ser1.replace([1,4],[100,400])

**#Can also input *dictionary***

* + ser1.replace({4:np.nan})

>> Rename Index (DataFrame)

# Making a DataFrame

* + dframe= DataFrame(np.arange(12).reshape((3, 4)),

index=['NY', 'LA', 'SF'],

columns=['A', 'B', 'C', 'D'])

@ dframe.index.map

* + dframe.index = dframe.index.map(str.lower)
  + dframe.index.map(str.lower)

@ dframe.rename

* dframe.rename (index=str.title, columns=str.lower)

# We can also use rename to insert dictionaries providing new values for indexes or columns!

* dframe.rename(index={'ny': 'NEW YORK'}, columns={'A': 'ALPHA'})

# If you would like to actually edit the data set in place, set inplace=True

* dframe.rename(index={'ny': 'NEW YORK'}, inplace=True)

>> Binning

years = [1990,1991,1992,2008,2012,2015,1987,1969,2013,2008,1999]

# We can seperate these years by decade

* decade\_bins = [1960,1970,1980,1990,2000,2010,2020]

#Now we'll use cut to get somethign called a Category object

* decade\_cat = pd.cut(years,decade\_bins)

# We can check the categories using .categories

* decade\_cat.categories

# Then we can check the value counts in each category

* pd.value\_counts(decade\_cat)

# We can also pass data values to the cut.

#For instance, if we just wanted to make two bins, evenly spaced based on max and min year, with a 1 year precision

* pd.cut(years, 2, precision=1)

>> Outliers

@ np.random.seed(12345)

@ DataFrame(np.random.randn(1000,4))

* dframe.head()
* dframe.describe()
* col = dframe[0]
* col[np.abs(col)>3]

# So we now know in column[0], rows 523 and 900 have values with abs > 3

#How about all the columns?

# We can use the "any" method

* dframe[(np.abs(dframe)>3).any(1)]

# WE could also possibly cap the data at 3

* dframe[np.abs(dframe)>3] = np.sign(dframe) \*3
* dframe.describe()

>> Blender

# WE can randomly reorder (permutate) a Series, or the rows in a DataFrame

#Let's take a look

* dframe = DataFrame(np.arange(4 \* 4).reshape((4, 4)))

#Create an array with a random perumation of 0,1,2,3

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 0 | 1 | 2 | 3 |
| **1** | 4 | 5 | 6 | 7 |
| **2** | 8 | 9 | 10 | 11 |
| **3** | 12 | 13 | 14 | 15 |

@ blender = np.random.permutation(4)

# Now permutate the dframe based on the blender

@ dframe.take(blender)

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 0 | 1 | 2 | 3 |
| **2** | 8 | 9 | 10 | 11 |
| **3** | 12 | 13 | 14 | 15 |
| **1** | 4 | 5 | 6 | 7 |
|  |  |  |  |  |

# Now what if we want permuations WITH replacement

# Let imagine a box with 3 marbles in it: labeled 1, 2, and 3

box = np.array([1,2,3])

# Now lets create a random permuation WITH replacement using randint

@ shaker = np.random.randint(0, len(box), size=10)

# Let's check teh box "shaker"

* + Shaker
    - array([2, 0, 1, 2, 1, 0, 0, 2, 0, 2])

#Now lets grab form the box

hand\_grabs = box.take(shaker)

#show

* + hand\_grabs
    - array([3, 1, 2, 3, 2, 1, 1, 3, 1, 3])

>> GroupBy

import numpy as np

import pandas as pd

from pandas import DataFrame, Series

#Let's make a dframe

dframe = DataFrame({'k1':['X','X','Y','Y','Z'],

'k2':['alpha','beta','alpha','beta','alpha'],

'dataset1':np.random.randn(5),

'dataset2':np.random.randn(5)})

#Show

dframe

#Now let's see how to use groupby

#Lets grab the dataset1 column and group it by the k1 key

group1 = dframe['dataset1'].groupby(dframe['k1'])

#Show the groupby object

group1

#Now we can perform operations on this particular group

group1.mean()

# We can use group keys that are series as well

#For example:

#We'll make some arrays for use as keys

cities = np.array(['NY','LA','LA','NY','NY'])

month = np.array(['JAN','FEB','JAN','FEB','JAN'])

#Now using the data from dataset1, group the means by city and month

dframe['dataset1'].groupby([cities,month]).mean()

# let's see the original dframe again.

dframe

# WE can also pass column names as group keys

dframe.groupby('k1').mean()

# Or multiple column names

dframe.groupby(['k1','k2']).mean()

# Another useful groupby method is getting the group sizes

dframe.groupby(['k1']).size()

# We can also iterate over groups

#For example:

for name,group in dframe.groupby('k1'):

print "This is the %s group" %name

print group

print '\n'

# We can also iterate with multiple keys

for (k1,k2) , group in dframe.groupby(['k1','k2']):

print "Key1 = %s Key2 = %s" %(k1,k2)

print group

print '\n'

# A possibly useful tactic is creating a dictionary of the data pieces

group\_dict = dict(list(dframe.groupby('k1')))

#Show the group with X

group\_dict['X']

# We could have also chosen to do this with axis = 1

# Let's creat a dictionary for dtypes of objects!

group\_dict\_axis1 = dict(list(dframe.groupby(dframe.dtypes,axis=1)))

#show

group\_dict\_axis1

# Next we'll learn how to use groupby with columns

# For example if we only wanted to group the dataset2 column with both sets of keys

dataset2\_group = dframe.groupby(['k1','k2'])[['dataset2']]

dataset2\_group.mean()

#Next we'll have a quick lesson on grouping with dictionaries and series!