*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)