**Proteomics Informatics (Spring 2014): Week 11**

**Pandas – III**

**Data Aggregations and Transformations**

1. > import pandas as pd

> from pandas import DataFrame

> from pandas import Series

> import numpy as np

1. **Intro to GroupBy operation**

> df = DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'], 'key2' : ['one', 'two', 'one', 'two', 'one'], 'data1' : np.random.randn(5), 'data2' : np.random.randn(5)})

> df

## Compute the mean of ‘data1’, after grouping according to key1

> grouped1 = df.groupby(df['key1'])

> grouped1 ## nothing done yet… just grouping of data points

 ## But it has all the information to apply any operation

## **You can iterate over the grouped object, like a list. It returns 2-tuples (key(s), data)**

> for key, data in grouped1: ## This is a simple loop, but you can do

 print key ## anything with the data

 print data, "\n"

> grouped1.<tab> ## functions that you can directly apply on grouped objects

> grouped1.mean()

> grouped1.describe()

## Group by multiple keys

> grouped2 = df.groupby([df['key1'], df['key2']])

**## Chk out how to access data from a hierarchically indexed DF**

> grouped2 ## Chk. out the hierarchical index

> grouped2.mean() ## **Built-in function**

> grouped2.size() ## Another useful built-in function. Likewise there are several other built-in functions… **but you could even write your own function to apply (See ‘aggregate’ method below)**

## You can also group along another axis (**column**)

## Also there are other ways to specify keys… dict/mapping,

## If you have to do some non-built-in computation, write a function and use aggregate method

## **All these aggregations take in a column (array/list of values) and apply the function**

## Ex. max(array) – min(array)

> def get\_range(arr):

 return arr.max() – arr.min()

 > grouped1[['data1', 'data2']].aggregate(get\_range)

 ## **Check** with grouped1.max(), grouped1.min()

## You can even provide multiple functions to apply on each group…

> grouped1[['data1', 'data2']].aggregate([get\_range, np.mean, np.std])

## Chk out hierarchical columns if multiple functions are applied to each column

## **Aggregation is limited… convert an array/list of values (one-dimensional) into a scalar…**

## **Transform function… grouped.transform(<some function>)… Will replace each element of a group with some transformed value… Ex. Replace value by a normalized value within the group… Or replace everything with the mean of the group… Or subtract group mean from each value within the group**

> def de\_mean(arr):

 return arr - arr.mean()

> grouped1.transform(de\_mean)

**## But again… applies to an array, and generates a new array to replace the old values… (Transformation). Limitation…**

**## Most general groupBy operation is the “apply” method…**

**## This will split into groups, apply some operation to group to generate new data… return back concatenated groups… with new values… “Transform” and “Aggregate” are special cases…**

**## Ex. For each group, return top-5 rows (by some column value)**

**## Like, top-5 peptides by e-value scores, for each sample…**

**Now this doesn’t return a scalar, or replace original values by some transformed values… Rather it generates totally new data with different dimensions… and puts them in appropriate groups…**

 > data = pd.read\_csv("/Users/hgrover/Desktop/pepIDs.csv")

 # data.ndim

# data.shape

# data.columns

# data.index

# data.describe()

# data.<Some Col>.describe()

# data.samplename.describe()

# data.samplename.value\_counts()

 > grouped3 = data.groupby(["samplename", "charge"])

> def top\_eVal(df, column='eVal', n=5):

 return df.sort\_index(by=column, ascending=True)[-n:]

 > grouped3\_topN = grouped3.apply(top\_eVal)

 > grouped3\_topN.ix["A"]; grouped3\_topN.ix["A", 2]