Lecture 11: Analyzing data II – Grouping and analyzing

Topic – Summary functions

Reread in the csv file we covered last lecture.

```
>>> import pandas as pd
>>> dt = pd.read_csv('021821_medium_threshold2.csv', index_col = 0)
```

Dataframes come with summary statistics functions, which summarize the value for a row or column. For example, if you wanted to calculate the average gene expression in each cell type, you could do the following (use drop to get rid of the non-numerical columns).

```
>>> dt.drop(columns = ['gene_name', 'Wormbase_ID']).mean()
ADA       70.876487
ADE       71.643830
ADF       72.214044
ADL       71.873994
AFD       71.160180
            ...
VB01      72.527223
VB02      72.368139
VC        71.968446
VC_4_5    72.100255
VD_DD     70.920625
Length: 128, dtype: float64
```

Other summary functions include median(), min(), max(), count(), std(), kurtosis(). There is also a describe method that includes a number of these functions:

```
>>> dt.drop(columns = ['gene_name', 'Wormbase_ID']).describe()
                ADA           ADE            ADF            ADL  ...          VB02
VC        VC_4_5         VD_DD
count  13669.000000  13669.000000   13669.000000   13669.000000  ...  13669.000000
13669.000000  13669.000000  13669.000000
mean      70.876487     71.643830      72.214044      71.873994  ...     72.368139
71.968446     72.100255     70.920625
std      835.806300   1310.171641    2403.896102    2922.675367  ...    823.131334
1357.371236   1464.616460    964.106266
min        0.000000      0.000000       0.000000       0.000000  ...      0.000000
0.000000      0.000000      0.000000
25%        0.000000      0.000000       0.000000       0.000000  ...      0.000000
0.000000      0.000000      0.000000
50%        0.000000      0.000000       1.626997       1.072438  ...      1.102683
0.000000      0.000000      0.000000
75%       14.118176     14.698212      16.613110      16.421330  ...     24.470680
17.023660     14.461254     24.199716
max      46308.923938   98000.306128   202697.711797   336350.854760   ...   56469.718483
92023.392079  89161.640000  80754.488434
```

You can also run these on rows, using the axis argument.

```
dt.drop(columns = ['gene_name', 'Wormbase_ID']).mean(axis = 1)

1        8926.858079
2        1060.437786
3         856.869660
4        1519.028523
5        4458.778125
           ...
13665       0.674315
13666       4.338739
```

```
13667       1.186117
13668       2.618098
13669       1.182652
Length: 13669, dtype: float64
```

Topic – Creating and modifying a dataframe

It is also possible to create a dataframe from other objects. Typically, this easiest using a dictionary, where the key is the name of the column and the value is a list of data. For example:

```
pd.DataFrame({'Test':[1,2,3,4], 'OtherData':[4,3,2,1]})
   Test  OtherData
0     1          4
1     2          3
2     3          2
3     4          1
```

Notice that an index is automatically created for you.

If you want to specify the index, you can use the index argument and supply it with a list that is the same size as the other data like so:

```
>>> pd.DataFrame({'Test':[1,2,3,4], 'OtherData':[4,3,2,1]}, index = ['A','B','C','D'])
   Test  OtherData
A     1          4
B     2          3
C     3          2
D     4          1
```

If you would like to modify the dataframe, you can modify a single element pretty straightforwardly. Recall how to use loc to access an individual cell and then set it equal to whatever you want:

```
>>> dt = pd.DataFrame({'Test':[1,2,3,4], 'OtherData':[4,3,2,1]}, index =
['A','B','C','D'])
>>> dt.loc['A','Test'] = 3
>>> dt
   Test  OtherData
A     3          4
B     2          3
C     3          2
D     4          1
```

You can also add an entire column by accessing the new columns name and setting it equal to a list:

```
>>> dt['NewData'] = [3,4,2,5]
>>> dt
   Test  OtherData  NewData
A     3          4        3
B     2          3        4
C     3          2        2
D     4          1        5
```

Or if you want to create a new column based upon existing data you can use apply to apply a function to every element of a column.

```
>>> import math
>>> dt['NewDataSquareRoot'] = dt['NewData'].apply(math.sqrt)
>>> dt
   Test  OtherData  NewData  NewDataSquareRoot
A     3          4        3           1.732051
```

```
B     2          3         4           2.000000
C     3          2         2           1.414214
D     4          1         5           2.236068
```

Or even get more complicated:

```
dt['CombinedData'] = dt['NewData'].apply(math.sqrt) + dt.OtherData
>>> dt
   Test  OtherData  NewData  NewDataSquareRoot  CombinedData
A     3          4         3           1.732051      5.732051
B     2          3         4           2.000000      5.000000
C     3          2         2           1.414214      3.414214
D     4          1         5           2.236068      3.236068
```

Topic – Grouping data

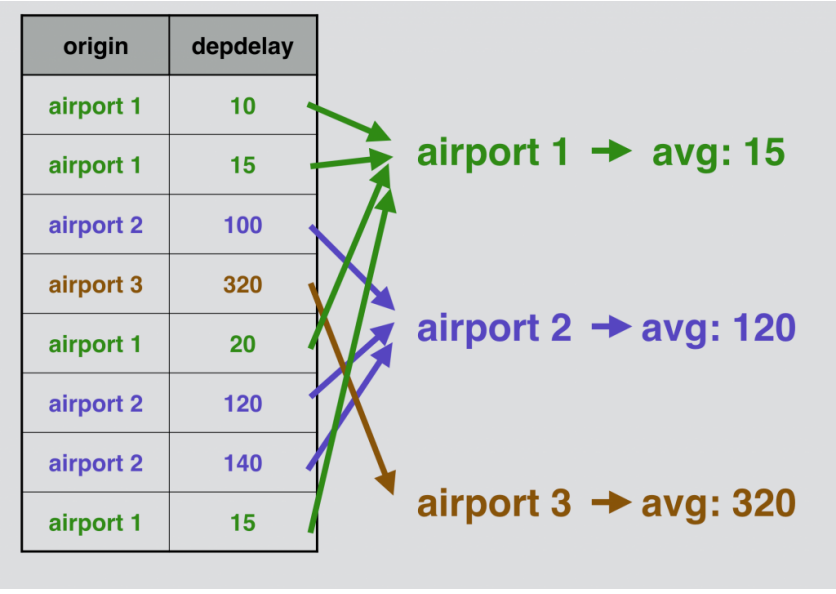
Data in a dataframe is organized in a tabular format, which is often most easily thought of as a collection of records. Each ROW consists of data for one record. The type of data that is stored is specified by the columns. For example, you might have a large data file containing information for every flight that flew in the US. Each record is UNIQUE. In other words, no two rows should be exactly the same as each flight is unique.

However, each column is not necessarily unique. For example, for the flight example above, you might store the airport that the flight left from. Since many flights leave from a given airport, there are many rows that share the same origin airport information. An example is shown below.

Now, sometimes you want to GROUP records together that share the same values. For example, let's say I wanted to find the airport that had the longest delays. To do that I need to average all of the delays together for EVERY flight that leaves from that airport.

Step 1: identify all the rows that have airport 1 together and group them together. In your mind you can think of it as putting them all in the same bucket. The number of buckets is determined by the number of unique values in a column (i.e. the number of airports in the US). Each bucket also contains multiple records (i.e. the number of flights that left from a particular airport.

Your next goal is to take all of the records in a bucket and condense them down to a single number. For example, you might want to find the average time delay for each airport. In this case you would take the AVERAGE for all the records in the bucket. Or you might want to find the maximum delay. Then you would take the MAX of the records.



Your next goal is to take all of the records in a bucket and condense them down to a single number. For example, you might want to find the average time delay for each airport. In this case you would take the AVERAGE for all the records in the bucket. Or you might want to find the maximum delay. Then you would take the MAX of the records. Let's see an example of this using the WorldDemographics.csv file:

```
>>> import pandas as pd
>>> dt = pd.read_csv('WorldDemographics.csv', sep = ',', index_col = 0)
>>> dt
      PopulationID  country_code    Level  continent_code  Age   #Alive
0      Afghanistan             4  Country            5501    0  1134501
1      Afghanistan             4  Country            5501    1  1134501
2      Afghanistan             4  Country            5501    2  1134501
3      Afghanistan             4  Country            5501    3  1124250
4      Afghanistan             4  Country            5501    4  1113998
...            ...           ...      ...             ...  ...      ...
20296     Zimbabwe           716  Country             910   96      163
20297     Zimbabwe           716  Country             910   97       53
20298     Zimbabwe           716  Country             910   98       41
20299     Zimbabwe           716  Country             910   99       30
20300     Zimbabwe           716  Country             910  100       19

[20301 rows x 6 columns]
```

Notice that there are many records or rows that share the sample Population ID. What if we wanted to find the total population of a country? We could do that using a groupby followed by a summary command.

```
>>> country_data = dt.groupby(['PopulationID'])
>>> type(country_data)
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

We use the groupby method of the DataFrame class, passing it the name of the column we want to group by (we could also send it a list of column names if we wanted to group on two columns).

Now we need to specify what function to use to combine all the records:

```
>>> country_data.sum()
                    country_code  continent_code   Age    #Alive
PopulationID
Afghanistan                  404          555601  5050  38928204
Albania                      808           93425  5050   2877640
Algeria                     1212           92112  5050  43850203
Angola                      2424           92011  5050  32866164
Antigua and Barbuda         2828           92415  5050     97878
...                          ...             ...   ...       ...
Viet Nam                   71104           92920  5050  97342743
Western Sahara             73932           92112  5050    597282
Yemen                      89587           93122  5050  29825815
Zambia                     90294           91910  5050  18383891
Zimbabwe                   72316           91910  5050  14862850

[201 rows x 4 columns]
>>> country_data.mean()
                    country_code  continent_code   Age         #Alive
PopulationID
Afghanistan                  4.0          5501.0  50.0  385427.762376
Albania                      8.0           925.0  50.0   28491.485149
Algeria                     12.0           912.0  50.0  434160.425743
Angola                      24.0           911.0  50.0  325407.564356
Antigua and Barbuda         28.0           915.0  50.0     969.089109
```

```
...                          ...           ...   ...           ...
Viet Nam                   704.0           920.0  50.0  963789.534653
Western Sahara             732.0           912.0  50.0    5913.683168
Yemen                      887.0           922.0  50.0  295305.099010
Zambia                     894.0           910.0  50.0  182018.722772
Zimbabwe                   716.0           910.0  50.0  147156.930693

[201 rows x 4 columns]
>>> country_data.min()
                    country_code    Level  continent_code  Age  #Alive
PopulationID
Afghanistan                    4  Country            5501    0      39
Albania                        8  Country             925    0      51
Algeria                       12  Country             912    0     963
Angola                        24  Country             911    0      38
Antigua and Barbuda           28  Country             915    0       4
...                          ...      ...             ...  ...     ...
Viet Nam                     704  Country             920    0   19645
Western Sahara               732  Country             912    0       1
Yemen                        887  Country             922    0      78
Zambia                       894  Country             910    0       5
Zimbabwe                     716  Country             910    0      19
```

Notice this works on all of the data columns that have data that can handle the function (i.e. you can't sum a string but you can take the minimum of it). If you want to only act on specific columns, use the agg method and specify the column/function combo using a dictionary:

```
>>> country_data.agg({'Age':min, '#Alive':sum})
                    Age    #Alive
PopulationID
Afghanistan           0  38928204
Albania               0   2877640
Algeria               0  43850203
Angola                0  32866164
Antigua and Barbuda   0     97878
...                 ...       ...
Viet Nam              0  97342743
Western Sahara        0    597282
Yemen                 0  29825815
Zambia                0  18383891
Zimbabwe              0  14862850

[201 rows x 2 columns]
```

The numpy module contains a large number of functions that are useful for analyzing arrays of data. We will cover those later. For now just remember that you will probably want to use numpy functions in the future for more complex summary operations.

```
>>> country_dt = country_data.agg({'Age':min, '#Alive':sum})
>>> country_dt
                    Age    #Alive
PopulationID
Afghanistan           0  38928204
Albania               0   2877640
Algeria               0  43850203
Angola                0  32866164
Antigua and Barbuda   0     97878
...                 ...       ...
Viet Nam              0  97342743
Western Sahara        0    597282
Yemen                 0  29825815
```

```
Zambia                  0  18383891
Zimbabwe                0  14862850

[201 rows x 2 columns]
>>> country_dt.loc['Angola','#Alive']
32866164
```