**Pandas**

**General Notes**

* import pandas as pd
* Way to work with tabular (table-like) data
* Similar to numpy, but numpy can only handle one data type per row, while pandas can have multiple data types
* **Layout**
  + Unique row names
  + Unique column names # data in columns can be different types unlike with numpy
  + similar to R dataframes, in fact a pandas object is a **dataframe**

**Building a Pandas Dataframe**

* Start with Dictionary
  + keys are column labels
  + values are data column by column
  + *dataFrame* = pd.DataFrame(*dict*) # where *dict* is your dictionary formatted as above
    - pandas will assign automatic row labels as integers beginning with 0
    - pandas will assign your keys as column labels
  + changing the row labels
    - *dataFrame*.index = *list* # where *list* provides indexes to use as row labels (check length)
* Start with CSV File
  + *dataFrame* = pd.read\_csv(“path/to/*file.csv*”, index\_col=0) # file path in “”
    - specify your file path
    - if row indexes are the first column, set index\_col=0
      * otherwise, pandas will generate numbered indexes and create first row as ‘unnamed’
        + that is if your first column has no header
    - formatting this csv file prior to import
      * best way is to set up exactly like a pandas dataframe
      * your first column should be row keys (cell A1, top left cell, should be blank)
      * starting with A2, the first row has column names
      * data are contained within the table

**Selecting Elements in a Dataframe**

* Select an entire column
  + *dataFrame*[“*column\_name*”]
    - returns a pandas.core.series.Series object # 1D labelled array
    - has extra info stored in it
    - does NOT return a dataframe
  + *dataFrame*[[“*column\_name*”]]
    - returns a pandas.core.series.DataFrame object
    - can provide multiple columns in the supplied list
      * *dataFrame*[[“*column1\_name”, “column2\_name*”]]
* Select rows
  + *dataFrame*[*start:end*]
    - use numbers for *start/end* recalling 0 indexing (row 0 is first data row, not headers)
    - just like slicing a list
    - returns the *start* row and every row until the row before the *end* index
  + loc
    - *dataFrame*.loc[“*row\_label*”] # use your row label in “”
      * returns a pandas.core.series.Series object
    - *dataFrame*.loc[[“*row\_label*”]]
      * returns a pandas.core.series.DataFrame object
      * can supply multiple row labels in your supplied list
      * can supply a second list of columns to include
        + *dataFrame*.loc[[*rows\_list*], [*columns\_list*]]

if you want all rows but only a few columns, supply : as the *rows\_list*

*dataFrame*.loc[:, [*columns\_list*]]

* + iloc
    - works the same as loc, except you supply indexes
    - *dataFrame*.iloc[[1]] # 2nd row as a dataframe
    - *dataFrame*.iloc[[1,2,3]] # rows 2, 3, and 4 as a dataframe
    - *dataFrame*.iloc[[1,2,3], [0,1]] # 1st two cols of rows 2, 3, and 4 as a dataframe
    - *dataFrame*.iloc[:, [0,1]] # 1st two cols of all rows as a dataframe
* Using Conditionals
  + *dataFrame*[‘*column\_name*’] *conditional* # *dataFrame[‘column1’] > 8*
    - returns a Series object filled with Booleans assigned based on the conditional
    - if you store this in a variable, you can pass the variable as the index
      * this returns a dataframe (not series) with only True values included
    - can also put this code directly into the [] to use as the index
      * *dataFrame*[*dataFrame[‘column’] > 24*]
  + Can use numpy’s logical\_and, logical\_or, logical\_not as long as numpy is imported
    - np.logical\_and(*dataFrame[‘column’] > 8, dataFrame[‘column’] < 12*)
      * returns a Series object contain Booleans that check each value
    - can use this statement as the index of a dataframe to return those values rather than True/False
      * *dataFrame*[np.logical\_or(*dataFrame[‘column’] > 10, dataFrame[‘column’ <= 20*)]

**Loops**

* Access column labels
  + for *value* in *dataFrame*:

*statements* referencing *value*

* + this will only access the column labels, and does not appear to leave a space for “cell A1”
* Access rows
  + for *label, row* in *dataFrame*.iterrows():

*statements*

* + the *label* statements will access the label for each row
  + the *row* statements access a Series object that includes column name and value
  + to access just one column
    - for *label, row* in *dataFrame*.iterrows():

print(*label* + “: “ + *row*[“*column\_name*”])

* + - this result will access the data
  + Add a new column that contains data that is a calculation
    - Less efficient, works on small dataframes # because creates a new series object each loop
      * for *label, row* in *dataFrame*.iterrows():

*dataFrame*.loc[label, “*new\_column”*] = *calculation* involving row[“*column\_name(s)*”]

* + - More efficient, works well on larger dataframes
      * *dataFrame*[“*new\_column*”] = *dataFrame*[“*column*”].apply(*function*)
        + your *function* input is just the name of the function # NOT *function()*
        + however, methods of objects need to be formatted propery

i.e. *function* could be str.upper or str.lower

* + - * + Example

*dataFrame*[“country\_length”] = *dataFrame*[“country”].apply(len)

* **Iterating Through Large Files**
  + Use comprehensions (see Python Library)
    - can also combine with pandas syntax to collect only certain chars from a column (like a time from a datetime column
    - Example
  + Chunking
    - for chunk in pd.read\_csv(‘*filename.csv’*, chunksize=*num*):
      * chunksize is a number of lines or rows to take at a time
      * need to update a value or variable through each iteration
    - Example
      * # Initialize an empty dictionary: counts\_dict
      * counts\_dict = {}
      * # Iterate over the file chunk by chunk
      * for chunk in pd.read\_csv('tweets.csv', chunksize=10):
      * # Iterate over the column ‘lang’ in tweets.csv DataFrame
      * for entry in chunk['lang']:
      * if entry in counts\_dict.keys():
      * counts\_dict[entry] += 1
      * else:
      * counts\_dict[entry] = 1
      * # Print the populated dictionary
      * print(counts\_dict)