Tutorial MATH 1MP3 – March 23, 2017

The following tutorial goes over the basics of the data analysis and data frames in Pandas. Similarly to last week, the goal is so that you've seen all this stuff before, so that when you go into the final project or studying for the final you can recall different methods of plotting, and have a resource to flip back to.

At the end of the tutorial there are two questions (similar in size to previous tutorials) to test these concepts.

Import the required libraries

Before you do anything you should import numpy

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

What are DataFrames?

- Primary "data structure" used in data analysis (big data, aritifical intelligence, machine learning, etc.). ("Data structures" is just a computer science word for variable types, like strings or lists).
- Stores data in a rectangular grid
- Columns for specific variables and rows for specific instances
- Three components
  - data
    - Numpy Array, Pandas Series, lists, dictionaries, etc.
  - Index
    - Rows of the data frame, like columns can be named
  - columns
    - Can name them! Each column should be a new variable type

Creating DataFrames with Pandas

```
grades = pd.DataFrame(data=[["A","B"],["B+","B-"]],
                  index=["Test 1", "Test 2"],
                  columns=["Jill", "Jack"])
print(grades)
```

In the above example we made lists into data frames. Notice how we had the three components: data, index, and columns all specified by their own lists. In the next example we'll see how we can make a DataFrame out of dictionaries.

```
predprey_dict = {"Wolves": ['Rabbits', 'Deer'], "Lions": ['Zebras',
'Hippo'], "Shark": ['Fish', 'Whales']}

predprey_df = pd.DataFrame(predprey_dict)

print(predprey_df)
```

The dictionary's *values* ended up becoming the data in the data frame and the *keys* ended up becoming the columns.

Indexing DataFrames

Say we had the DataFrame from the predator prey example above, and we wanted to index the element "Rabbits". There are many ways we could go about that.

```
print(predprey_df.Wolves)
```
will tell us everything wolves eat, we can then get the "first" thing in the last by saying

```
print(predprey_df.Wolves[0])
```

Similarly we could use other functions, iloc, for example. (Stands for **I**nteger **Loc**ating, it's function is to use normal integer indexing to get into data frames). Try to figure out what all the following do.

```
print(predprey_df.iloc[0,2])
print(predprey_df.loc[0]['Wolves'])
print(predprey_df.at[0,'Wolves'])
print(predprey_df.iat[0,0])
print(predprey_df.get_value(0, 'Wolves'))
```

Some of these are more suiting then others. For instance, since we built this DataFrame from a dictionary, on *my* computer Wolves are in column 3 (so position 2), but since orders of Dictionaries are all wacky that might be different for *you*.

Knowing the difference between loc and iloc is super important. Let's have another look.

```
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
index= [2, 'A', 4], columns=[48, 49, 50])
print(df)
print(df.loc[2])
print(df.iloc[2])
print(df.ix[2])
```

Loc looks at values with index *label* 2. (which is the first (0th) index..). iloc looks at values with index *position* 2. What ix does is slightly more subtle. In this example ix returned the values with position 2.

```
df2 = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
index= [2, 3, 4], columns=[48, 49, 50])
print(df2)
print(df2.ix[2])
```

Now df2 is identical to df, we just have the second index now labeled 3 instead of 'A'. What ix did in this example is returned the values with label 2. If all your labels are pure integers, ix will return the values with label that matches its argument (like in the df2 example). If your labels aren't all pure integers, ix returns the the corresponding position (like in the df example).

DataFrames also do that “smart indexing” with booleans that Numpy does. Try the following

```
print(df>4)
print(df[df>4])
print(df[49]>4)
print(df[df[49]>4])
```

Try looking into the .between command for data frames. (Hint help(df.between)).

Adding, Deleting, and Renaming within DataFrames
Adding a column to a data frame is simple enough, and similar to how it's done with dictionaries

```
print(grades)
grades['Sally'] = ['A+', 'A+']
print(grades)
```

If we wanted to, we could also add on with loc.

```
print(df)
df.loc[:,4] = [7, 8, 9]
print(df)
```

The reason we might want to do this, is because it translates easily into adding rows as well.

```
print(df)
df.loc["C"] = [1,2,3,4]
print(df)
```

We can also delete using the drop command and axis argument (think back to axis from hw3)

```
print(df.drop(4, axis=1))
print(df)
print(df.drop("A",axis=0))
print(df)
```

Now try doing the drop again with the optional argument `inplace=True` and see what happens.

Iterating over DataFrames

Looping in DataFrames isn't so bad once you've seen it once.

```
for index, row in df.iterrows():
     print(row, index)
```

Or, for instance, for looping over all Sally's grades

```
for index, row in grades.iterrows():
     print(row['Sally'])
```

Writing DataFrames to file

Writing pandas DataFrames to a file is conveniently nice and easy. Just try

```
df.to_csv('nameOfDataFrame.csv', sep=',')
```

the sep argument tells the computer what to use as a seperator. In this case I'm saying put a comma between each data value.

Exercise

Consider the following Python dictionary data and Python list labels:

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat',
'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no',
'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

(This is just some meaningless data I made up with the theme of animals and trips to a vet.)

1. Create a DataFrame df from this dictionary data which has the index labels.
2. Use the .info and .describe commands to see a summary of the data
3. return the first three rows of df
4. Select just the animal and age columns
5. Select data in rows [3,4,8] *and* columns ['animal','age']
6. Select only the rows where the number of visits is greater than 3.
7. Select the rows where the animal is a cat *and* the age is less than 3.
8. Append a new row 'k' to df with your choice of values for each column. Then delete that row to return the original DataFrame.
9. Calculate the sum of all visits (the total number of visits).
10. Try running `df.groupby('animal')['age'].mean()`, what does it do?