NIAPythonDay3_July2017

July 19, 2017

NIA Python Bootcamp DAY 3 - Wednesday July 19, 2017

1 Day 1 review

1. Python ecosystem of tools
2. Jupyter Notebook is code, output and documentation all in one document
3. Type code into cells, and to run them you press Shift-Enter
4. Different data types for different data
5. Tab completion reduces typing, shows you pop-up menu of all the things you can do with that piece of data
6. Operators take one or more input values and turn them into other values *based on the input values type*
7. Converting data from one type to another using the function syntax, e.g., int()

2 Day 2 Review

1. Exploring data types using the TAB key
2. Python syntax for taking slices of iterables
3. NumPy arrays: basic math operations in 1-D and 2-D (e.g., row-wise and column-wise eman)
4. Subselecting based on a boolean criterion
5. Example: Images as 3-D matrices

3 Day 3:

3. PANDAS DataFrames
4. Simple and complex sorting

3.1 PANDAS DataFrame

- pandas = Python Data Analysis Library
- Emulate R's data.frame structure.
- Basically a NumPy matrix with
  - Row and column names
  - Can have columns of different types
  - Handles missing data better

3.2 Load the PANDAS package into memory using import()

```
In [1]: import pandas as pd
```

3.3 Use PANDAS read_* functions to import data

- There are many functions to import data
- Type pd.read_ then TAB to see all the import functions

```
In [ ]: pd.read_
```

3.4 Read data from file or URL

```
In [2]: titanic_data_url  = "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.
```

```
In [3]: titanic = pd.read_excel( titanic_data_url )
```

3.5 Return type is a DataFrame

```
In [4]: type(titanic)
```

```
Out[4]: pandas.core.frame.DataFrame
```

3.6 What did we just load?

```
In [ ]: titanic
```

3.6.1 Change the number of rows Pandas will display using the set_option() function

Use the word None if you want to display all of them.

```
In [ ]: pd.set_option( 'display.max_rows', None )
```

3.6.2 See the first N rows using .head(N)

Defaults to first 5

```
In [ ]: titanic.head()
```

3.6.3 See the last N rows using .tail(N)

Defaults to last 5.

```
In [ ]: titanic.tail()
```

3.6.4 See random N rows using .sample(N)

```
In [ ]: titanic.sample(3)
```

3.7 len() return number of observations (rows)

```
In [8]: len(titanic)
```

```
Out[8]: 1309
```

3.8 .shape attribute gives the shape

```
In [9]: titanic.shape
```

```
Out[9]: (1309, 14)
```

3.9 .describe(): Get basic statistics across all columns

- Detects which columns are quantitative gives descriptive stats for those

```
In [10]: titanic.describe()
```

```
Out[10]:              pclass     survived          age        sibsp        parch  \
         count  1309.000000  1309.000000  1046.000000  1309.000000  1309.000000
         mean      2.294882     0.381971    29.881135     0.498854     0.385027
         std       0.837836     0.486055    14.413500     1.041658     0.865560
         min       1.000000     0.000000     0.166700     0.000000     0.000000
         25%       2.000000     0.000000    21.000000     0.000000     0.000000
         50%       3.000000     0.000000    28.000000     0.000000     0.000000
         75%       3.000000     1.000000    39.000000     1.000000     0.000000
         max       3.000000     1.000000    80.000000     8.000000     9.000000

                      fare        body
         count  1308.000000  121.000000
         mean     33.295479  160.809917
         std      51.758668   97.696922
         min       0.000000    1.000000
         25%       7.895800   72.000000
         50%      14.454200  155.000000
         75%      31.275000  256.000000
         max     512.329200  328.000000
```

3.10 .count() give number of non-empty cells

```
In [11]: titanic.count()
```

```
Out[11]: pclass       1309
         survived     1309
         name         1309
         sex          1309
         age          1046
         sibsp        1309
         parch        1309
```

```
ticket       1309
fare         1308
cabin         295
embarked     1307
boat          486
body          121
home.dest     745
dtype: int64
```

3.11 DataFrame row and column headers

- Like a NumPy array, but with column and row headers.
- Enables slicing by headers, and not just indices like with NumPy arrays
- The collection of row headers is stored in the .index attribute.
- The collection of column headers is stored in the .columns attribute.

```
In [12]: titanic.columns

Out[12]: Index(['pclass', 'survived', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket',
                'fare', 'cabin', 'embarked', 'boat', 'body', 'home.dest'],
               dtype='object')

In [13]: titanic.index

Out[13]: RangeIndex(start=0, stop=1309, step=1)
```

3.12 Get a single column

Two ways to do it:

1. Use the "object-oriented" style of API, i.e., the "dot."
2. Use the dict style, i.e., key-value style (put the column name into brackets, get the column)
3. The returned data type is a PANDAS Series object, which keeps the index from the DataFrame attached

```
In [ ]: titanic.age

In [ ]: titanic['home.dest']
```

3.13 .value_counts()

```
In [14]: titanic.sex.value_counts()

Out[14]: male      843
         female    466
         Name: sex, dtype: int64
```

3.14 Use .pivot_table() to have a breakdown of the data

3.14.1 For categorical data, use aggfunc='count'

```
In [15]: titanic.pivot_table( values='survived', index='sex', columns='pclass', aggfunc='count
```

```
Out[15]: pclass      1      2      3     All
         sex
         female  144.0  106.0  216.0   466.0
         male    179.0  171.0  493.0   843.0
         All     323.0  277.0  709.0  1309.0
```

3.14.2 For non-categorical data, can use another statistical measure for aggregation, like mean

```
In [16]: titanic.pivot_table( values='age', index='sex', columns='pclass', aggfunc='mean', marg
```

```
Out[16]: pclass          1          2          3        All
         sex
         female  37.037594  27.499191  22.185307  28.687071
         male    41.029250  30.815401  25.962273  30.585233
         All     39.159918  29.506705  24.816367  29.881135
```

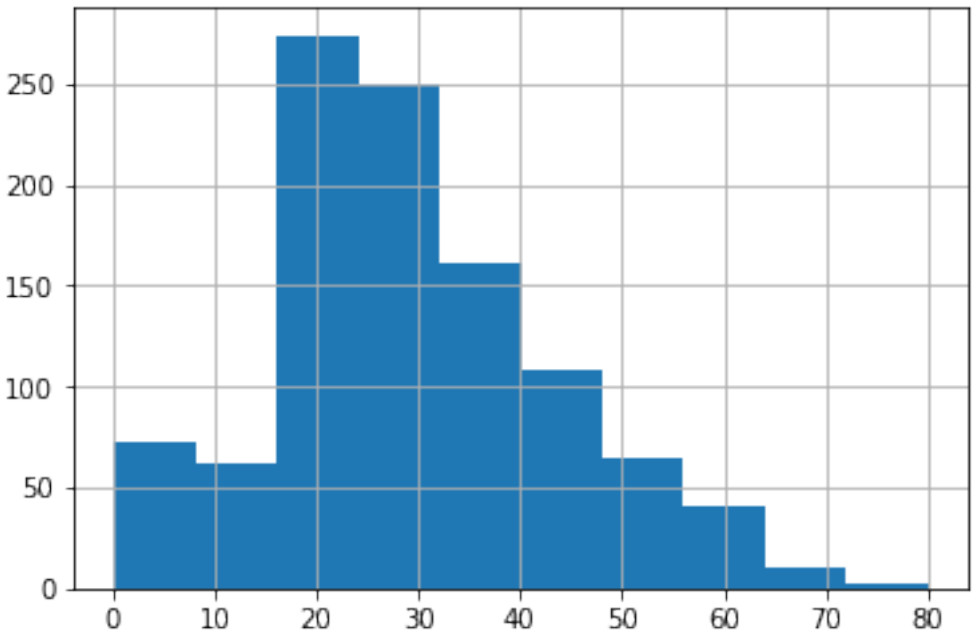
3.15 Quick figures

```
In [17]: %matplotlib inline
```

3.15.1 Univarate histograms

```
In [18]: titanic.age.hist()
```

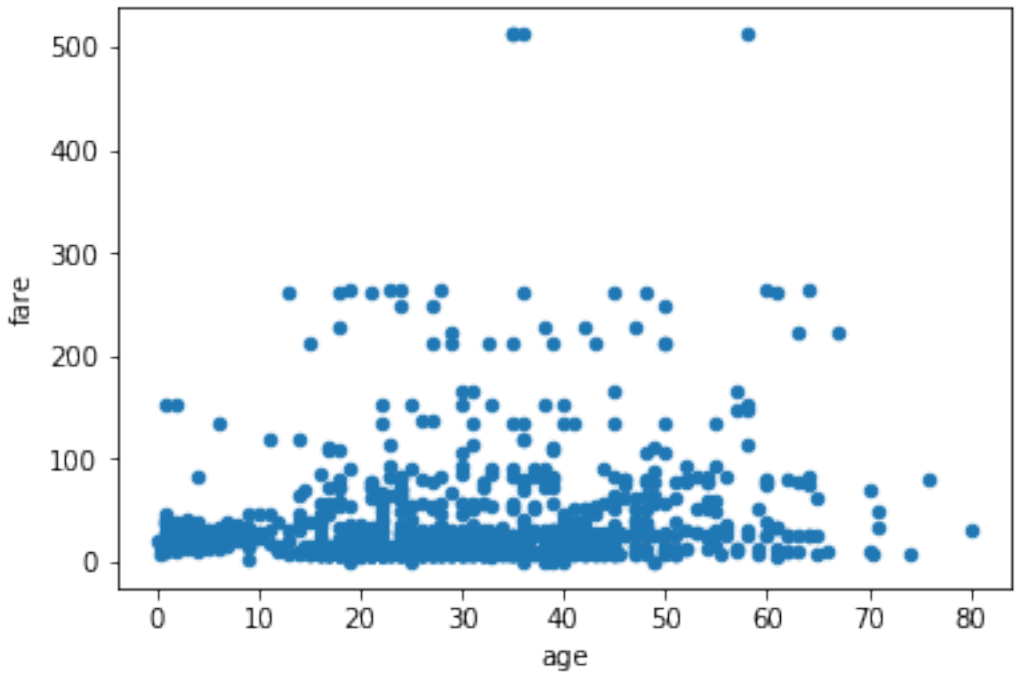
```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1122da198>
```



3.15.2 Bivariate scatter plot using the .plot attribute

```
In [19]: titanic.plot.scatter( 'age', 'fare' )
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1123c14e0>
```



3.16 Missing data

- Oftentimes, missing data is represented as np.nan, which stands for "Not A Number"
- No missing data representation for an integer

3.16.1 Using the .dtypes attribute to check data types for each column

Tips

- If an int64 (just a fancy int), then probably no missing values in that column, check the .count() to confirm
- If a float64 (just a fancy float), then probably missing values in the form of NaN are possible
- If an object, this almost always means it's a string in there, missing values are ""

```
In [20]: titanic.dtypes
```

```
Out[20]: pclass         int64
         survived       int64
         name          object
         sex           object
         age          float64
         sibsp          int64
         parch          int64
         ticket        object
         fare         float64
         cabin         object
         embarked      object
         boat          object
         body         float64
         home.dest     object
         dtype: object
```

3.17 Stats on a DataFrame are NaN sensitive (unlike NumPy)

- In other words, doesn't count missing values as 0

```
In [21]: titanic.count()
```

```
Out[21]: pclass       1309
         survived     1309
         name         1309
         sex          1309
         age          1046
         sibsp        1309
         parch        1309
         ticket       1309
         fare         1308
         cabin         295
         embarked     1307
         boat          486
         body          121
         home.dest     745
         dtype: int64
```

```
In [22]: titanic.age.describe()
```

```
Out[22]: count    1046.000000
         mean       29.881135
         std        14.413500
         min         0.166700
         25%        21.000000
         50%        28.000000
         75%        39.000000
         max        80.000000
         Name: age, dtype: float64
```

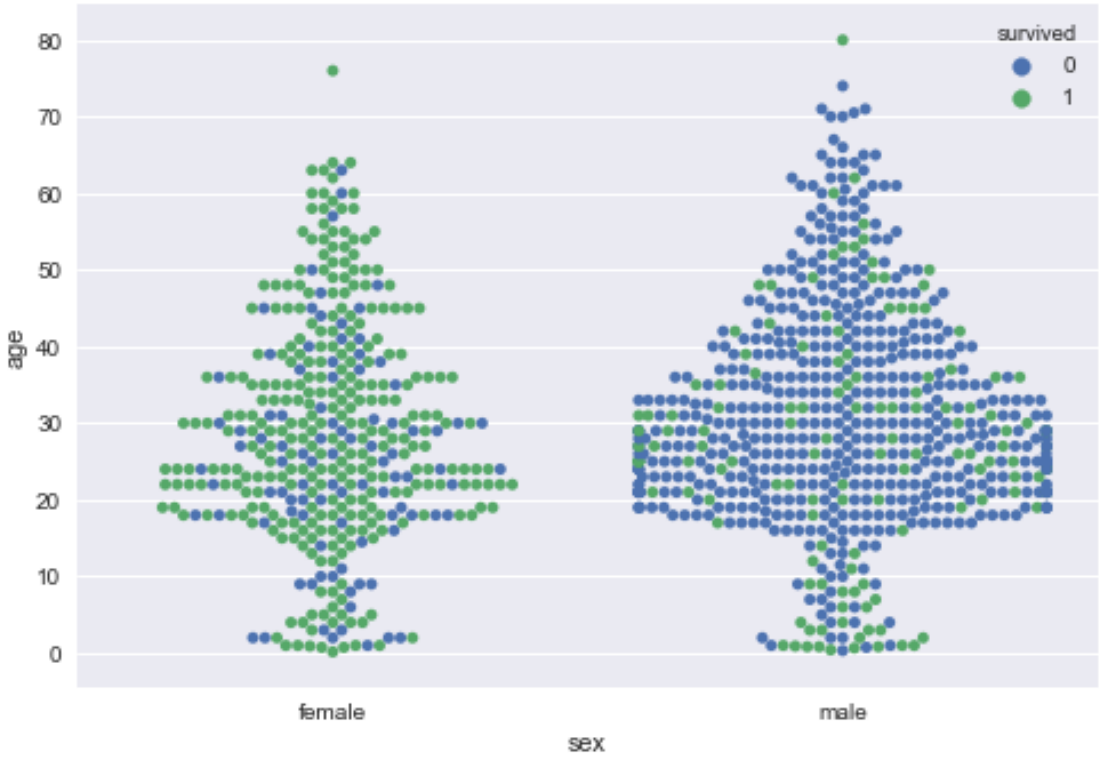
3.18 Using the Seaborn Package for visualization

- Browse this page to see all the types of nice figures you can make

```
In [23]: import seaborn as sns
```

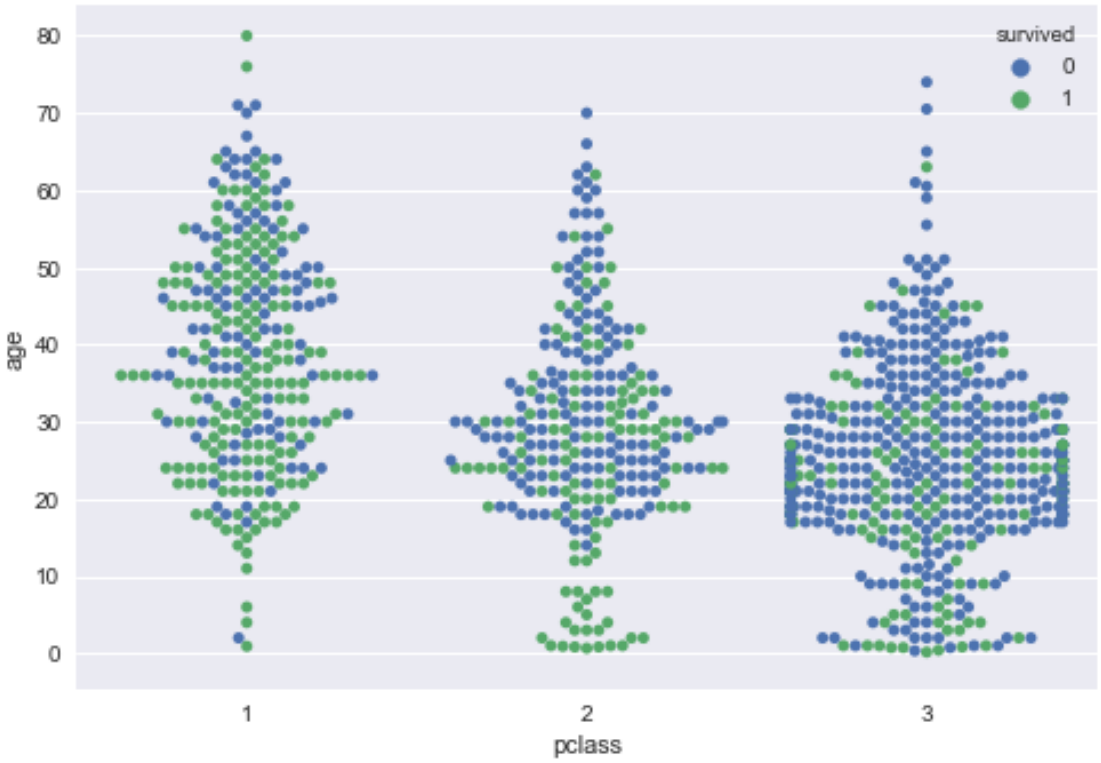
```
In [24]: sns.swarmplot( x='sex', y='age', hue='survived', data=titanic )
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1143baac8>
```



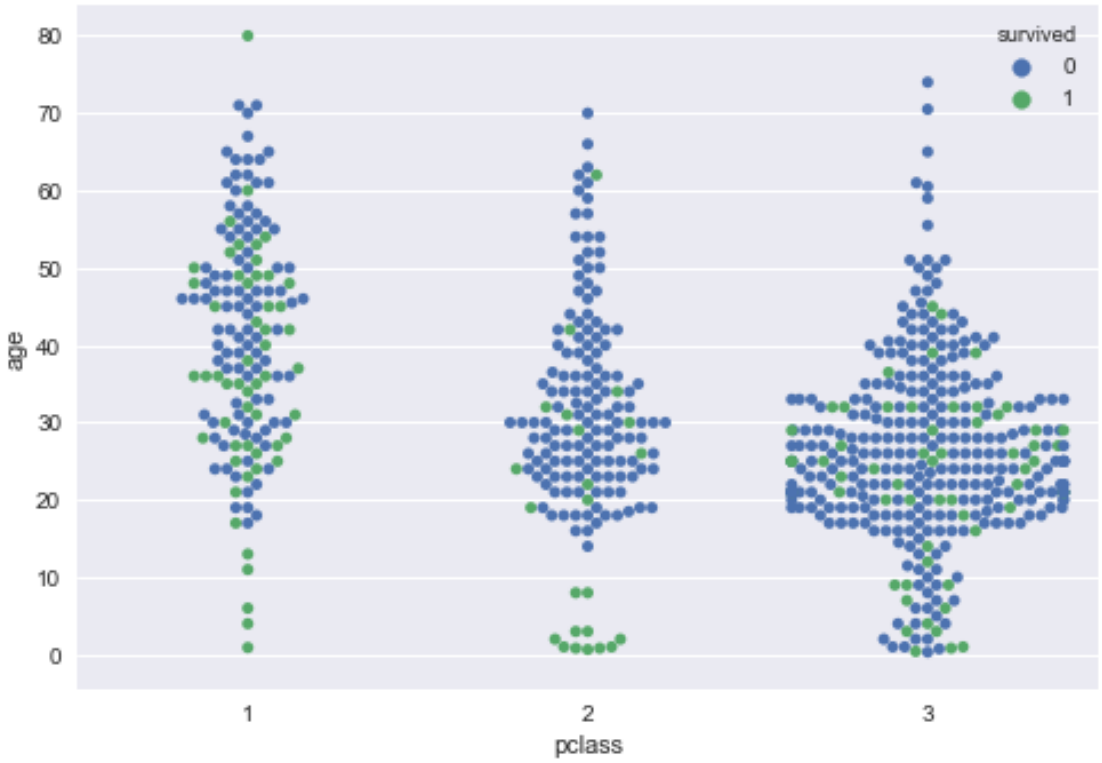

```
In [25]: sns.swarmplot( x='pclass', y='age', hue='survived', data=titanic)
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x114449cc0>
```



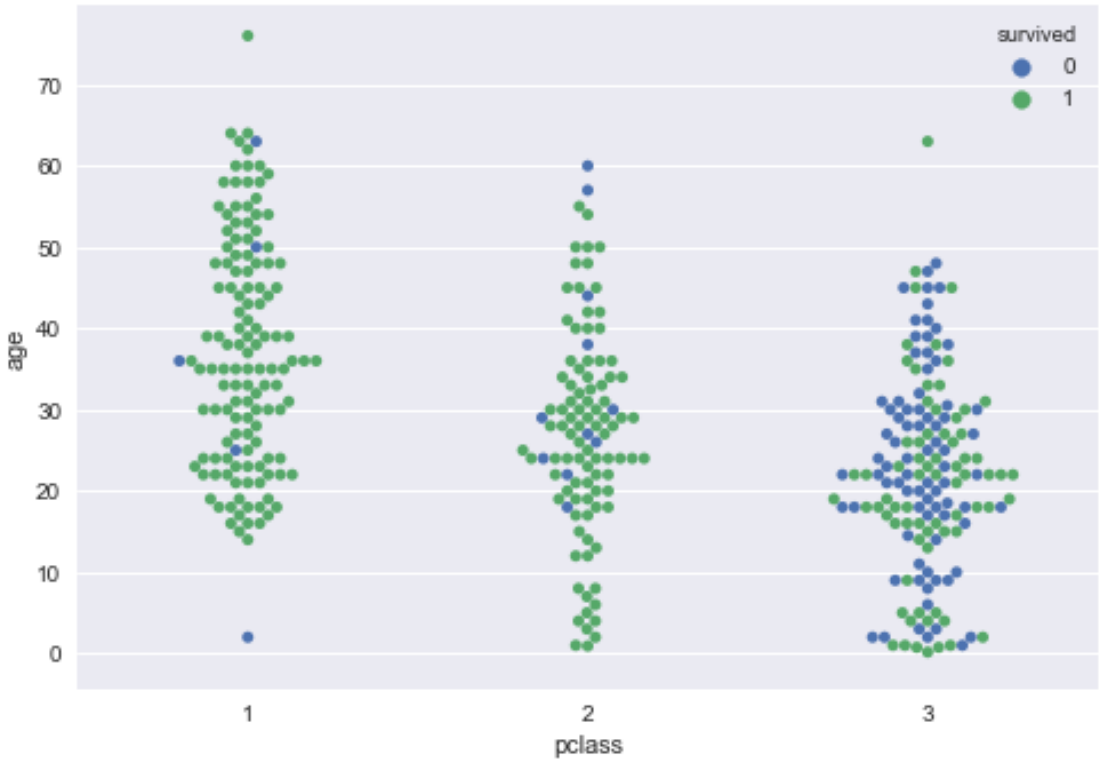
3.19 Subselecting based on one of the variables

```
In [26]: male = titanic[ titanic.sex == 'male']

In [27]: female = titanic[ titanic.sex == 'female']

In [28]: sns.swarmplot( x='pclass', y='age', hue='survived', data=male)

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x114465a20>
```




```
In [29]: sns.swarmplot( x='pclass', y='age', hue='survived', data=female )
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x11471e358>
```



3.20 Slicing by rows and columns using .loc[]

```
In [30]: subset = titanic.loc[ titanic.age < 25, ['pclass','age'] ]

In [31]: len(subset)

Out[31]: 409
```

3.21 Using .sort_values() for simple or complex sorting

```
In [ ]: titanic.sort_values?

In [ ]: titanic.sort_values( by='age').head(10)
```