NIAPythonDay3

March 27, 2019

NIA Python Bootcamp UNIT 3 - Wednesday July 19, 2017

1 UNIT 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 UNIT 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 UNIT 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 [5]: titanic.shape
```

```
Out[5]: (1309, 14)
```

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 [6]: pd.set_option( 'display.max_rows', 50 )
```

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

Defaults to first 5

```
In [7]: titanic.head(2)
```

```
Out[7]:    pclass  survived                             name     sex      age  sibsp  \
        0       1         1    Allen, Miss. Elisabeth Walton  female  29.0000      0
        1       1         1   Allison, Master. Hudson Trevor    male   0.9167      1

           parch  ticket      fare    cabin embarked boat  body  \
        0      0   24160  211.3375       B5        S    2   NaN
        1      2  113781  151.5500  C22 C26        S   11   NaN

                                 home.dest
        0                     St Louis, MO
        1  Montreal, PQ / Chesterville, ON
```

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

Defaults to last 5.

```
In [8]: titanic.tail(1)
```

```
Out[8]:       pclass  survived                 name   sex   age  sibsp  parch  ticket  \
        1308       3         0  Zimmerman, Mr. Leo  male  29.0      0      0  315082

              fare cabin embarked boat  body home.dest
        1308  7.875   NaN        S  NaN   NaN       NaN
```

3.6.4 See random N rows using .sample(N)

```
In [9]: titanic.sample(3)
```

```
Out[9]:       pclass  survived                                   name     sex   age  \
        811        3         0  Ford, Mrs. Edward (Margaret Ann Watson)  female  48.0
        348        2         0                     Bracken, Mr. James H    male  27.0
        1075       3         0                    Odahl, Mr. Nils Martin    male  23.0

              sibsp  parch      ticket    fare cabin embarked boat  body  \
        811       1      3  W./C. 6608  34.375   NaN        S  NaN   NaN
        348       0      0      220367  13.000   NaN        S  NaN   NaN
        1075      0      0        7267   9.225   NaN        S  NaN   NaN

                                         home.dest
        811   Rotherfield, Sussex, England Essex Co, MA
        348                 Lake Arthur, Chavez County, NM
        1075                                       NaN
```

3.7 len() return number of observations (rows)

```
In [10]: len(titanic)
```

```
Out[10]: 1309
```

3.8 .shape attribute gives the shape

```
In [11]: titanic.shape
```

```
Out[11]: (1309, 14)
```

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

- Detects which columns are quantitative gives descriptive stats for those

```
In [12]: titanic.describe()
```

```
Out[12]:             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 [13]: titanic.count()
```

```
Out[13]: 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 [14]: titanic.columns
```

```
Out[14]: Index(['pclass', 'survived', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket',
                'fare', 'cabin', 'embarked', 'boat', 'body', 'home.dest'],
               dtype='object')
```

```
In [15]: titanic.index
```

```
Out[15]: 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 [16]: titanic.columns
```

```
Out[16]: Index(['pclass', 'survived', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket',
                'fare', 'cabin', 'embarked', 'boat', 'body', 'home.dest'],
               dtype='object')
```

```
In [17]: titanic['home.dest']
```

```
Out[17]: 0                         St Louis, MO
         1      Montreal, PQ / Chesterville, ON
         2      Montreal, PQ / Chesterville, ON
         3      Montreal, PQ / Chesterville, ON
         4      Montreal, PQ / Chesterville, ON
         5                         New York, NY
         6                           Hudson, NY
         7                          Belfast, NI
         8                  Bayside, Queens, NY
         9                  Montevideo, Uruguay
         10                        New York, NY
         11                        New York, NY
         12                       Paris, France
         13                                 NaN
         14                       Hessle, Yorks
         15                        New York, NY
         16                        Montreal, PQ
         17                        Montreal, PQ
         18                                 NaN
         19                        Winnipeg, MN
         20                        New York, NY
         21                        New York, NY
         22                        New York, NY
         23                                 NaN
```

```
24                                         NaN
                                ...
1284                                       NaN
1285                                       NaN
1286                                       NaN
1287                                       NaN
1288                                       NaN
1289                                       NaN
1290                                       NaN
1291                                       NaN
1292                                       NaN
1293                                       NaN
1294                                       NaN
1295                                       NaN
1296                                       NaN
1297                                       NaN
1298                                       NaN
1299                                       NaN
1300                                       NaN
1301                                       NaN
1302                                       NaN
1303                                       NaN
1304                                       NaN
1305                                       NaN
1306                                       NaN
1307                                       NaN
1308                                       NaN
Name: home.dest, Length: 1309, dtype: object
```

3.13 using .values

```
In [18]: titanic['home.dest'].values
```

```
Out[18]: array(['St Louis, MO', 'Montreal, PQ / Chesterville, ON',
                'Montreal, PQ / Chesterville, ON', ..., nan, nan, nan],
               dtype=object)
```

3.14 .value_counts()

```
In [19]: titanic['sex']
```

```
Out[19]: 0       female
         1         male
         2       female
         3         male
         4       female
         5         male
         6       female
```

```
7         male
8       female
9         male
10        male
11      female
12      female
13      female
14        male
15        male
16        male
17      female
18      female
19        male
20        male
21      female
22        male
23      female
24      female
         ...
1284      male
1285      male
1286    female
1287      male
1288      male
1289      male
1290    female
1291      male
1292      male
1293      male
1294      male
1295      male
1296      male
1297      male
1298      male
1299      male
1300    female
1301      male
1302      male
1303      male
1304    female
1305    female
1306      male
1307      male
1308      male
Name: sex, Length: 1309, dtype: object
```

In [20]:
```
titanic.sex.value_counts()
```

Out[20]:
```
male      843
```

```
female    466
Name: sex, dtype: int64
```

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

3.15.1 For categorical data, use aggfunc='count'

```
In [ ]: titanic.pivot_table?
```

```
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.pivot_table( values='survived', index='pclass',
                              columns='sex', aggfunc='count',
                              margins=True)
```

```
Out[22]: sex     female  male   All
         pclass
         1          144   179   323
         2          106   171   277
         3          216   493   709
         All        466   843  1309
```

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

```
In [23]: titanic.pivot_table( values='age', index='sex',
                              columns='pclass',
                              aggfunc='mean', margins=True)
```

```
Out[23]: 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
```

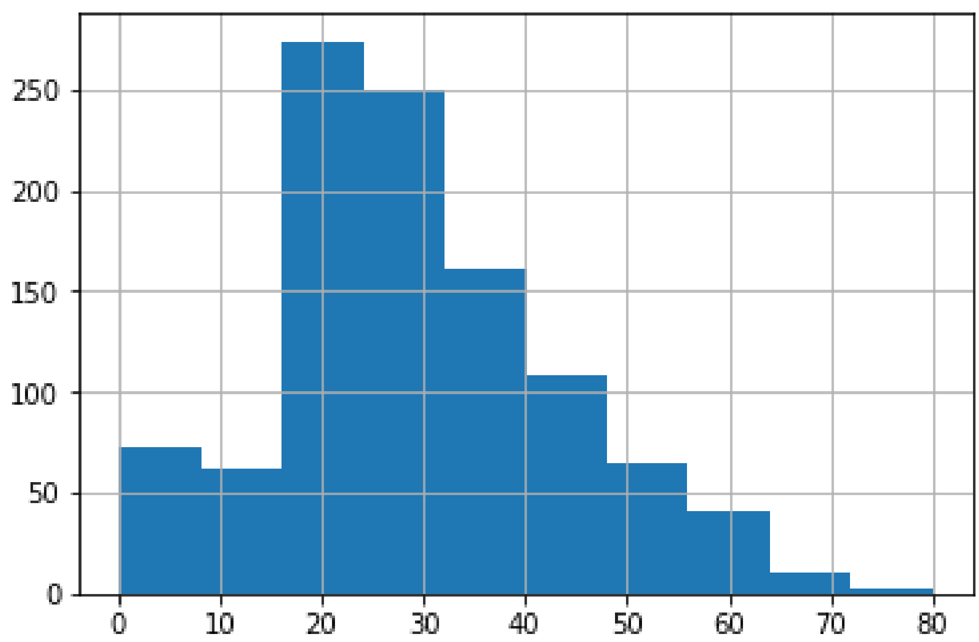
3.16 Quick figures

- Execute this Jupyter command `%matplotlib inline` before executing code that makes figures to get Jupyter to render them as output.

```
In [27]: %matplotlib inline
```

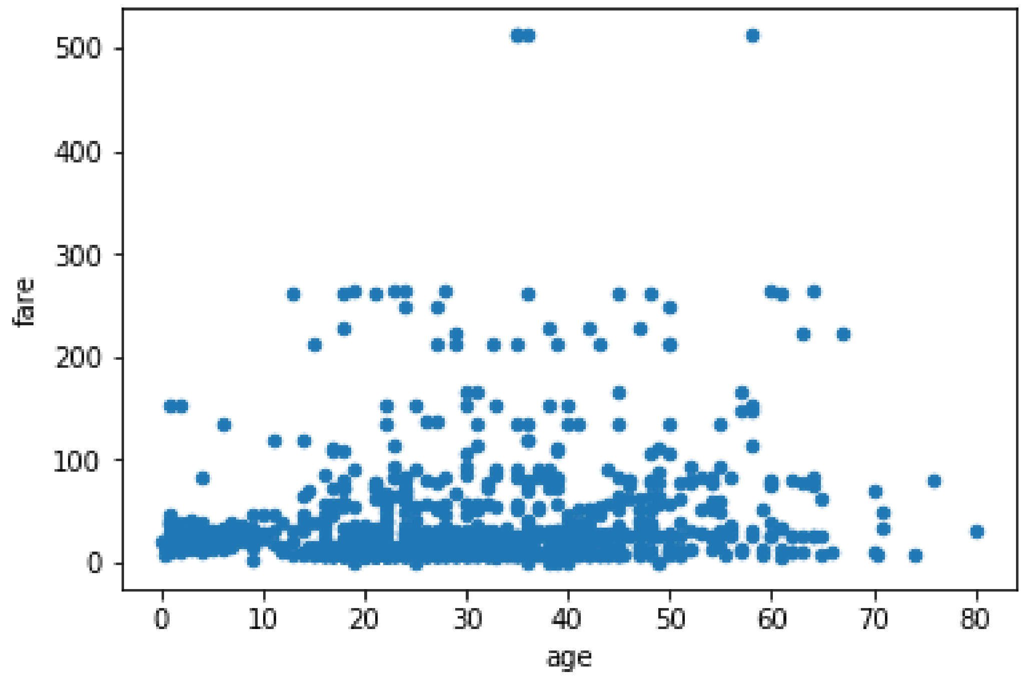
3.16.1 Univarate histograms

```
In [ ]: titanic.age.hist?

In [24]: thing = titanic.age

In [25]: type( thing)

Out[25]: pandas.core.series.Series

In [ ]: thing.hist?

In [28]: titanic.age.hist()

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x11a320588>
```



3.16.2 Bivariate scatter plot using the .plot attribute

```
In [29]: titanic.plot.scatter( 'age', 'fare' )

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x11a3a17b8>
```



3.17 Missing data in PANDAS

- Represented as np.nan, which stands for "Not A Number"
- NaN has type float
- No missing data representation for an integer!
    - Either convert all to floats to use NaN (recommended!), or
    - Convert values into strings and store empties as "" (less recommended)
    - Establish a "flag" value, e.g., -999 and filter out those before using (not recommended!)

```
In [30]: import numpy as np

In [31]: np.nan

Out[31]: nan

In [32]: type( np.nan )

Out[32]: float
```

3.18 Column data types

- A single column of data within a PANDAS DataFrame is called a Series.
- All values within a Series must be of the same type.
- Use the .dtypes attribute to check data types for each column

In [33]: `titanic.head(3)`

```
Out[33]:    pclass  survived                             name     sex      age  sibsp  \
        0       1         1   Allen, Miss. Elisabeth Walton  female  29.0000      0
        1       1         1  Allison, Master. Hudson Trevor    male   0.9167      1
        2       1         0    Allison, Miss. Helen Loraine  female   2.0000      1

           parch  ticket      fare    cabin embarked boat  body  \
        0      0   24160  211.3375       B5        S    2   NaN
        1      2  113781  151.5500  C22 C26        S   11   NaN
        2      2  113781  151.5500  C22 C26        S  NaN   NaN

                                home.dest
        0                    St Louis, MO
        1  Montreal, PQ / Chesterville, ON
        2  Montreal, PQ / Chesterville, ON
```

In [34]: `titanic.count()`

```
Out[34]: 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 [35]: `titanic.dtypes`

```
Out[35]: 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
```

3.19 Column data types may hint at missing values

When using pd.read_csv() and pd.read_excel() to load a file form disk, PANDAS will try to pick a data type for a column that makes sense.

- If a float64 (just a fancy float), then missing values in the form of NaN are possible
  - Use .count() to count non-empty (non-NaN) values
- If an int64 (just a fancy int), then probably no missing values in that column
- If an object, this almost always means it's a string in there
  - Can represent missing values as "", but .count() only works for float data types!

```
In [36]: some_emptys = pd.Series( ["","asdf","","","","27",""] )
         print( some_emptys.dtype )
         some_emptys.count()

object


Out[36]: 7
```

3.19.1 Coerce to numeric values using pd.to_numeric()

```
In [37]: some_emptys = pd.to_numeric( some_emptys, errors='coerce')

In [38]: some_emptys

Out[38]: 0      NaN
         1      NaN
         2      NaN
         3      NaN
         4      NaN
         5     27.0
         6      NaN
         dtype: float64

In [39]: print( some_emptys.dtype )
         some_emptys.count()

float64


Out[39]: 1
```

3.20 Statistics on a DataFrame ignore NaNs (as one might expect)

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

In [40]:
```
titanic.count()
```

Out[40]:
```
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 [41]:
```
titanic.age.describe()
```

Out[41]:
```
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
```

3.21 Using the Seaborn Package for visualization

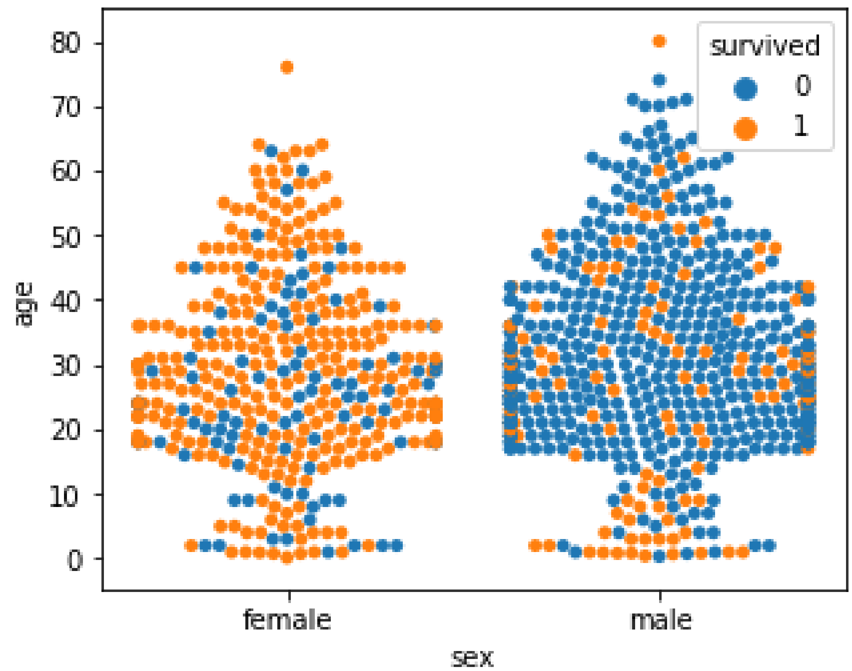
- Browse this page to see all the types of nice figures you can make

In [42]:
```
import seaborn as sns
```

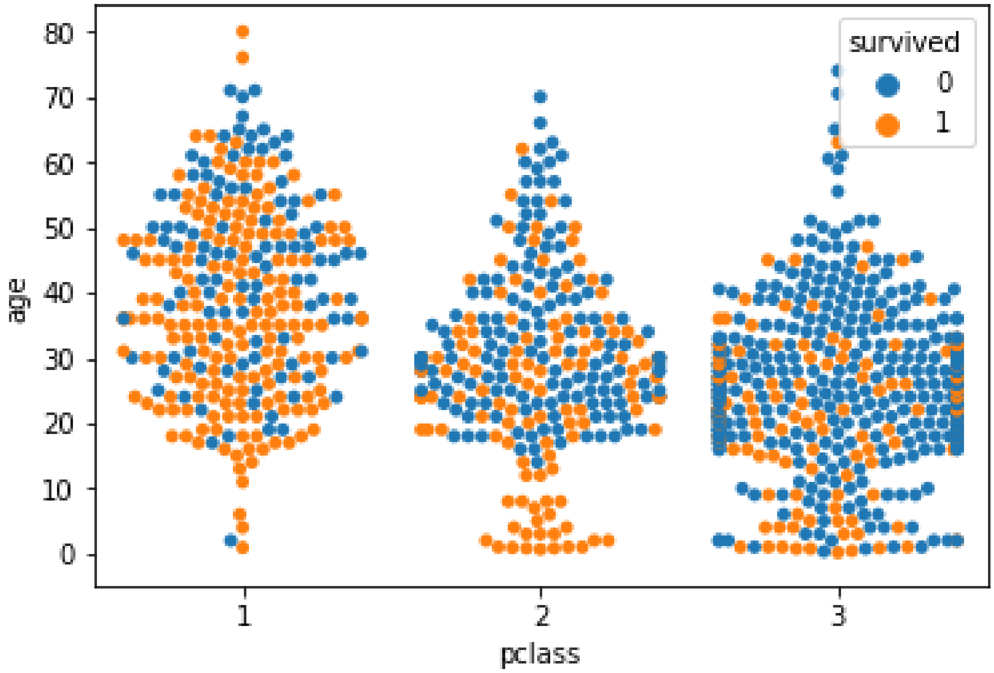
In [43]:
```
import matplotlib.pyplot as plt
```

In [44]:
```
fig, ax = plt.subplots( figsize=(5,4) )

sns.swarmplot( x='sex', y='age', hue='survived',
              data=titanic, ax=ax )
#fig.savefig( 'testytest.pdf')
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11a49b860>
```




```
In [45]: type( fig )

Out[45]: matplotlib.figure.Figure

In [46]: type( ax)

Out[46]: matplotlib.axes._subplots.AxesSubplot

In [ ]: ax.

In [ ]: sns.swarmplot?

In [47]: sns.swarmplot( x='pclass', y='age', hue='survived',
                       data=titanic )

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1262ae240>
```



3.22 Subselecting based on one of the variables

```
In [48]: titanic.shape

Out[48]: (1309, 14)

In [49]: titanic.sex.value_counts()

Out[49]: male      843
         female    466
         Name: sex, dtype: int64

In [50]: titanic.sex.head()

Out[50]: 0    female
         1      male
         2    female
         3      male
         4    female
         Name: sex, dtype: object

In [51]: titanic.sex == 'male'

Out[51]: 0       False
         1        True
```

```
2       False
3        True
4       False
5        True
6       False
7        True
8       False
9        True
10       True
11      False
12      False
13      False
14       True
15       True
16       True
17      False
18      False
19       True
20       True
21      False
22       True
23      False
24      False
        ...
1284     True
1285     True
1286    False
1287     True
1288     True
1289     True
1290    False
1291     True
1292     True
1293     True
1294     True
1295     True
1296     True
1297     True
1298     True
1299     True
1300    False
1301     True
1302     True
1303     True
1304    False
1305    False
1306     True
1307     True
```

```
1308     True
Name: sex, Length: 1309, dtype: bool
```

In [52]:
```
bool_array = titanic.sex == 'male'
```

In [53]:
```
len(bool_array)
```

Out[53]:
```
1309
```

In [54]:
```
(titanic.sex == 'male').head()
```

Out[54]:
```
0    False
1     True
2    False
3     True
4    False
Name: sex, dtype: bool
```

In [55]:
```
male = titanic[ titanic.sex == 'male' ]
```

In [56]:
```
# Boolean selector array have to be the sahe shape as the array itself!!
bool_array = [True]*1000
```

In [57]:
```
#titanic[ bool_array ]
```

In [58]:
```
male.shape
```

Out[58]:
```
(843, 14)
```

In [59]:
```
gender_tf = titanic.sex == 'male'
```

In [60]:
```
gender_tf.shape
```

Out[60]:
```
(1309,)
```

In [61]:
```
male.shape
```

Out[61]:
```
(843, 14)
```

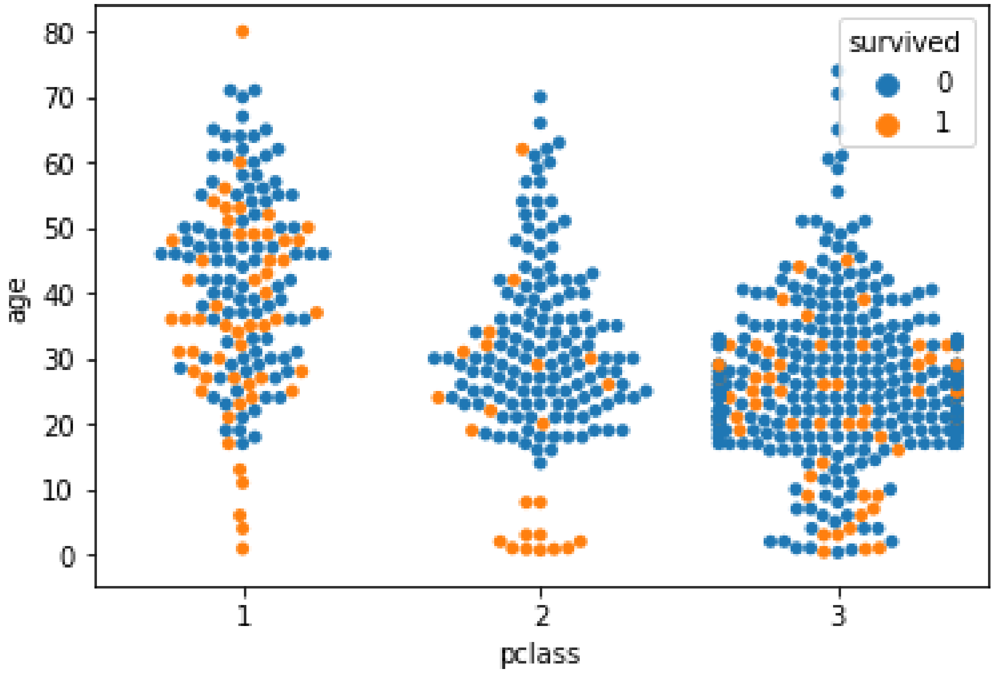
In [62]:
```
female = titanic[ titanic.sex == 'female']
```

In [63]:
```
female.shape
```

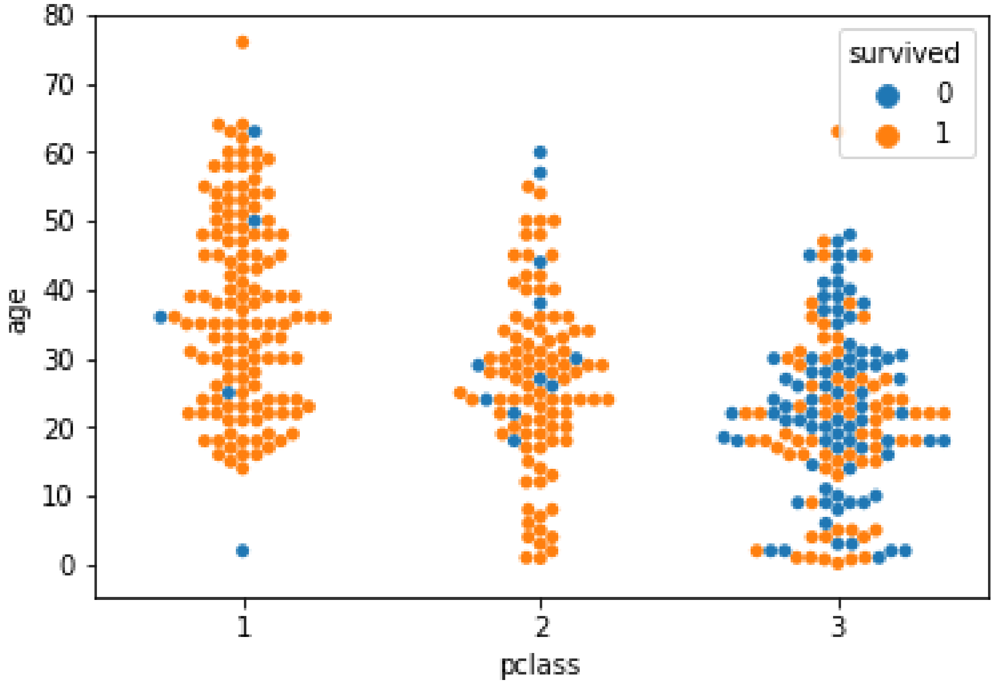
Out[63]:
```
(466, 14)
```

In [64]:
```
sns.swarmplot( x='pclass', y='age', hue='survived',
              data=male)
```

Out[64]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12636ba20>
```




```
In [65]: sns.swarmplot( x='pclass', y='age', hue='survived',
                        data=female )

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x126370160>
```



3.23 Slicing by rows and columns using .loc[]

```
In [66]: subset = titanic[ titanic.age < 25 ]

In [67]: subset.shape

Out[67]: (409, 14)

In [68]: subset = titanic.loc[ titanic.age < 25 ]

In [69]: subset.shape

Out[69]: (409, 14)
```

4 Complex sort

```
In [70]: age_bool = titanic.age < 10

In [71]: age_bool.value_counts()

Out[71]: False    1227
         True       82
         Name: age, dtype: int64

In [72]: class_bool = titanic.pclass == 1

In [73]: class_bool.value_counts()

Out[73]: False    986
         True     323
         Name: pclass, dtype: int64

In [74]: age_class_bool = age_bool & class_bool

In [75]: age_class_bool.value_counts()

Out[75]: False    1305
         True        4
         dtype: int64

In [76]: titanic.loc[ age_class_bool, 'age' ]

Out[76]: 1      0.9167
         2      2.0000
         94     4.0000
         273    6.0000
         Name: age, dtype: float64

In [77]: len(subset)

Out[77]: 409
```

4.1 Using .sort_values() for simple or complex sorting

```
In [ ]: titanic.sort_values?
```

```
In [78]: titanic.shape
```

```
Out[78]: (1309, 14)
```

```
In [82]: titanic.sort_values( by=['pclass','age']).head()
```

```
Out[82]:      pclass  survived                                  name     sex      age  \
         1         1         1       Allison, Master. Hudson Trevor    male   0.9167
         2         1         0         Allison, Miss. Helen Loraine  female   2.0000
         94        1         1            Dodge, Master. Washington    male   4.0000
         273       1         1        Spedden, Master. Robert Douglas    male   6.0000
         54        1         1  Carter, Master. William Thornton II    male  11.0000

              sibsp  parch  ticket      fare    cabin embarked boat  body  \
         1        1      2  113781  151.5500  C22 C26        S   11   NaN
         2        1      2  113781  151.5500  C22 C26        S  NaN   NaN
         94       0      2   33638   81.8583      A34        S    5   NaN
         273      0      2   16966  134.5000      E34        C    3   NaN
         54       1      2  113760  120.0000  B96 B98        S    4   NaN

                                home.dest
         1    Montreal, PQ / Chesterville, ON
         2    Montreal, PQ / Chesterville, ON
         94                San Francisco, CA
         273                 Tuxedo Park, NY
         54                    Bryn Mawr, PA
```

```
In [83]: titanic.sort_values( by=['pclass','age'],
                              ascending=False).head()
```

```
Out[83]:       pclass  survived                         name     sex   age  sibsp  parch  \
         1235       3         0          Svensson, Mr. Johan    male  74.0      0      0
         727        3         0         Connors, Mr. Patrick    male  70.5      0      0
         782        3         0             Duane, Mr. Frank    male  65.0      0      0
         1261       3         1       Turkula, Mrs. (Hedwig)  female  63.0      0      0
         1068       3         0  Nysveen, Mr. Johan Hansen    male  61.0      0      0

               ticket    fare cabin embarked boat   body home.dest
         1235  347060  7.7750   NaN        S  NaN    NaN       NaN
         727   370369  7.7500   NaN        Q  NaN  171.0       NaN
         782   336439  7.7500   NaN        Q  NaN    NaN       NaN
         1261    4134  9.5875   NaN        S   15    NaN       NaN
         1068  345364  6.2375   NaN        S  NaN    NaN       NaN
```

```
In [84]: titanic['home.dest'].sample(10)
```

```
Out[84]: 568                           NaN
         1147                          NaN
         786     Tofta, Sweden Joliet, IL
         1104                          NaN
         604       Norway Los Angeles, CA
         94             San Francisco, CA
         784               West Haven, CT
         125                           NaN
         864                           NaN
         214                Lexington, MA
         Name: home.dest, dtype: object
```