Homework 2

February 8, 2018

1 Homework 2: Exploring & Visualizing Data

Make sure you have seaborn and missingno installed. Run `pip3 install seaborn` and `pip3 install missingno` in your container/shell if you don't.

1.1 Setup

In this homework, we will more rigorously explore data visualization and data manipulation with a couple datasets. Please fill in the cells with `## YOUR CODE HERE` following the appropriate directions.

```
In [1]: # removes the need to call plt.show() every time
        %matplotlib inline
```

Seaborn is a powerful data visualization library built on top of matplotlib. We will be using seaborn for this homework (since it is a better tool and you should know it well). Plus seaborn comes default with *much* better aesthetics (invoked with the `set()` function call).

```
In [2]: import missingno as msno
        import seaborn as sns
        sns.set()
```

Import `numpy` and `pandas` (remember to abbreviate them accordingly!)

```
In [3]: ## YOUR CODE HERE
        import numpy as np
        import pandas as pd
```

1.2 Getting to know a new dataset

First load the `titanic` dataset directly from seaborn. The `load_dataset` function will return a pandas dataframe.

```
In [4]: titanic = sns.load_dataset('titanic')
```

Now use some pandas functions to get a quick overview/statistics on the dataset. Take a quick glance at the overview you create.

In [5]:
```
## YOUR CODE HERE
titanic.info()
titanic.age.max()
titanic.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
survived       891 non-null int64
pclass         891 non-null int64
sex            891 non-null object
age            714 non-null float64
sibsp          891 non-null int64
parch          891 non-null int64
fare           891 non-null float64
embarked       889 non-null object
class          891 non-null category
who            891 non-null object
adult_male     891 non-null bool
deck           203 non-null category
embark_town    889 non-null object
alive          891 non-null object
alone          891 non-null bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.6+ KB
```

Out[5]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [6]:
```
## YOUR CODE HERE
titanic.loc[ titanic['survived'] == 1]
print("survival rate: " + str(342 / 891) )
```

```
survival rate: 0.3838383838383838
```

With your created overview, you should be able to answer these questions:

- What was the age of the oldest person on board? 80 years old
- What was the survival rate of people on board? 38%
- What was the average fare of people on board? $32.20

By the way, for getting overviews, pandas also has a groupby function that is quite nice to use. example:

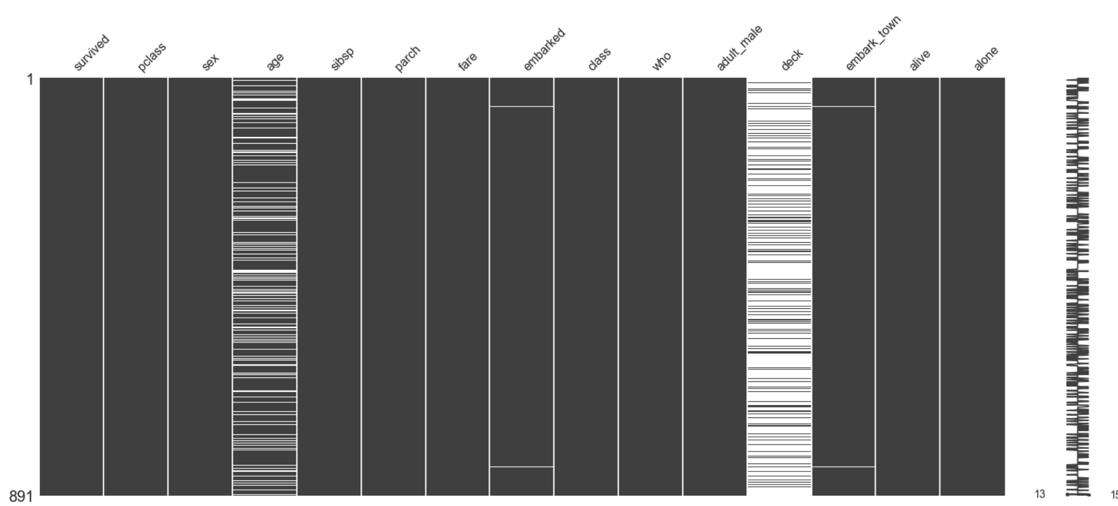
```
In [7]: titanic.groupby(['sex','embark_town'])['survived'].mean()
```

```
Out[7]: sex     embark_town
        female  Cherbourg      0.876712
                Queenstown     0.750000
                Southampton    0.689655
        male    Cherbourg      0.305263
                Queenstown     0.073171
                Southampton    0.174603
        Name: survived, dtype: float64
```

Now we have an overview of our dataset. The next thing we should do is clean it - check for missing values and deal with them appropriately.

missingno allows us to really easily see where missing values are in our dataset. It's a simple command:

```
In [8]: msno.matrix(titanic)
```



The white lines show us the missing data. One quick observation is the deck has a lot of missing data. Let's just go ahead and drop that column from the dataset since it's not relevant.

```
In [9]: titanic.drop('deck', axis=1, inplace=True)
```

Now let's rerun the matrix and see. All that white is gone! Nice.

We still have a bunch of missing values for the age field. We can't just drop the age column since it is a pretty important datapoint. One way to deal with this is simply to just remove the records with missing information with dropna(), but this would end up removing out a significant amount of our data.

What do we do now? We can now explore a technique called `missing value imputation`. What this means is basically we find a reasonable way to *replace* the unknown data with workable values.

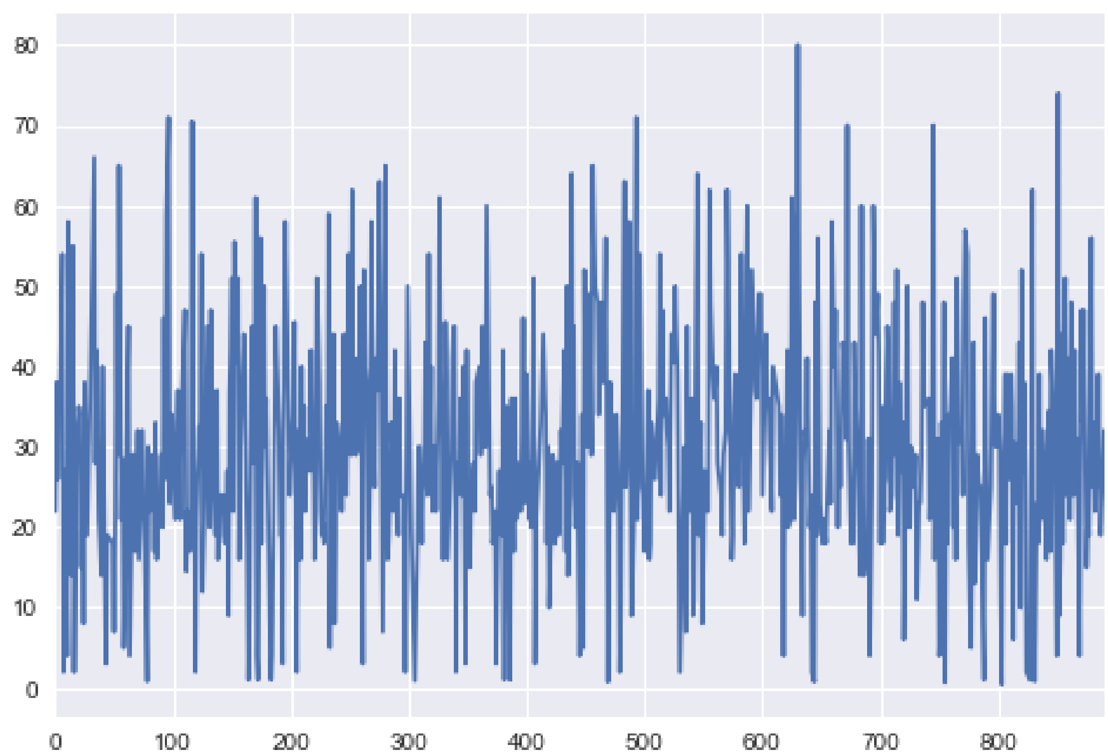
There's a lot of theory regarding how to do this properly, (for the curious look here). We can simply put in the average age value for the missing ages. But this really isn't so great, and would skew our stats.

If we assume that the data is missing *at random* (which actually is rarely the case and very hard to prove), we can just fit a model to predict the missing value based on the other available factors. One popular way to do this is to use KNN (where you look at the nearest datapoints to a certain point to conclude the missing value), but we can also use deep neural networks to achieve this task.

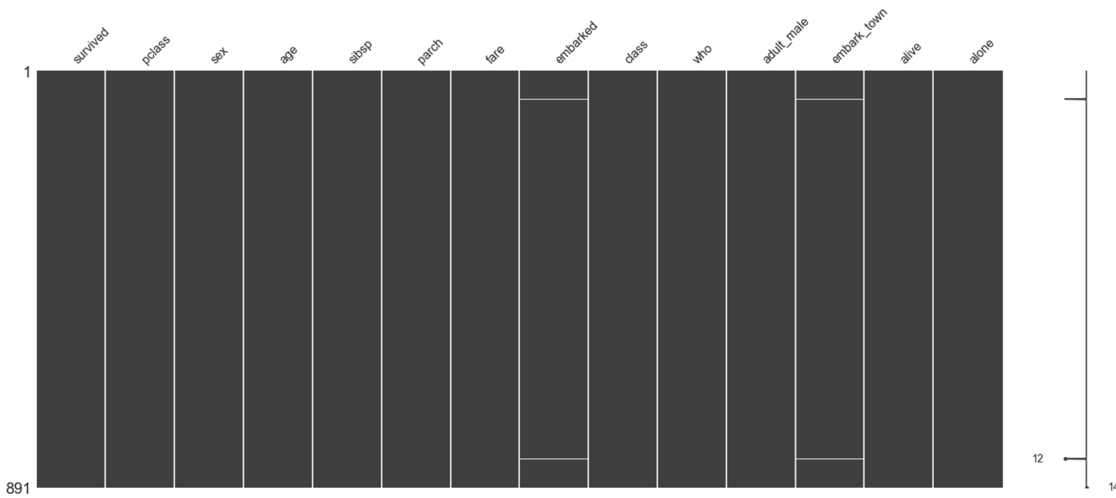
You must now make you own decision on how to deal with the missing data. You may choose any of the methods discussed above. Easiest would be to fill in with average value (but this will skew our visualizations) (if you use pandas correctly, you can do this in one line - try looking at pandas documentation!). After writing your code, verify the result by rerunning the matrix.

In [10]:
```
## YOUR CODE HERE
titanic.age.interpolate().plot()
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x2575a108da0>



In [11]:
```
titanicFilled = titanic.copy()
titanicFilled.age = titanicFilled.age.interpolate()
msno.matrix(titanicFilled)
```



1.3 Intro to Seaborn

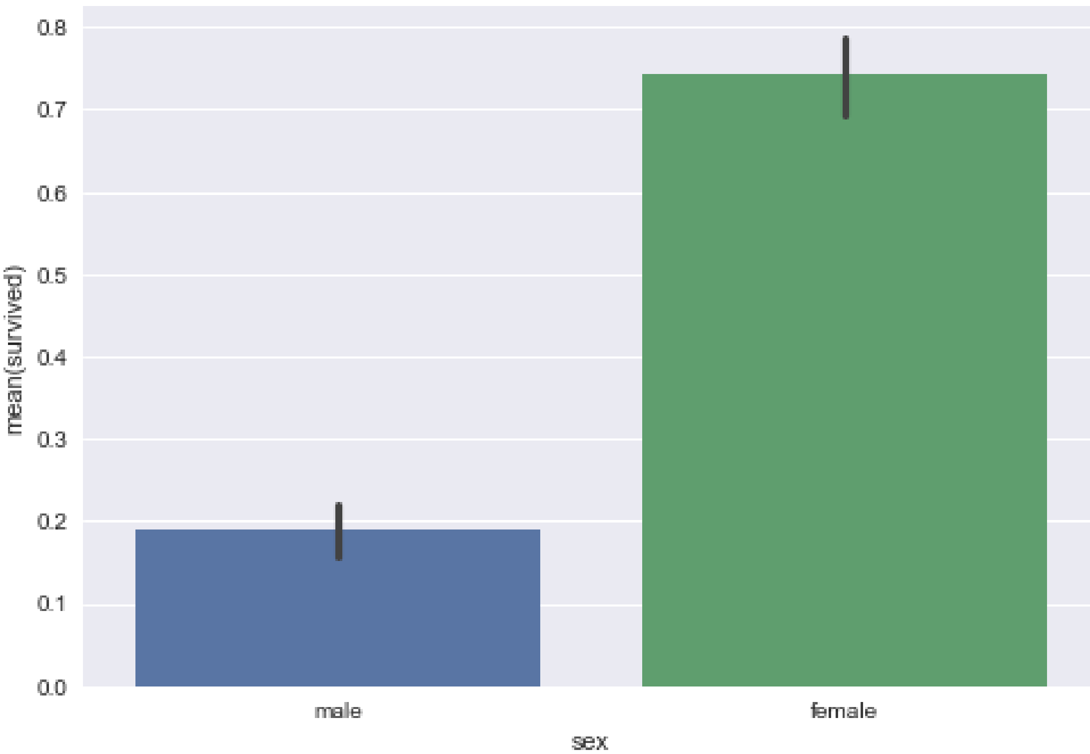
There are 2 types of data in any dataset: categorial and numerical data. We will first explore categorical data.

One really easy way to show categorical data is through bar plots. Let's explore how to make some in seaborn. We want to investigate the difference in rates at which males vs females survived the accident. Using the documentation here and example here, create a `barplot` to depict this. It should be a really simple one-liner.

We will show you how to do this so you can get an idea of how to use the API.

```
In [12]: sns.barplot(x='sex', y='survived', data=titanic)
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2575b55d208>
```

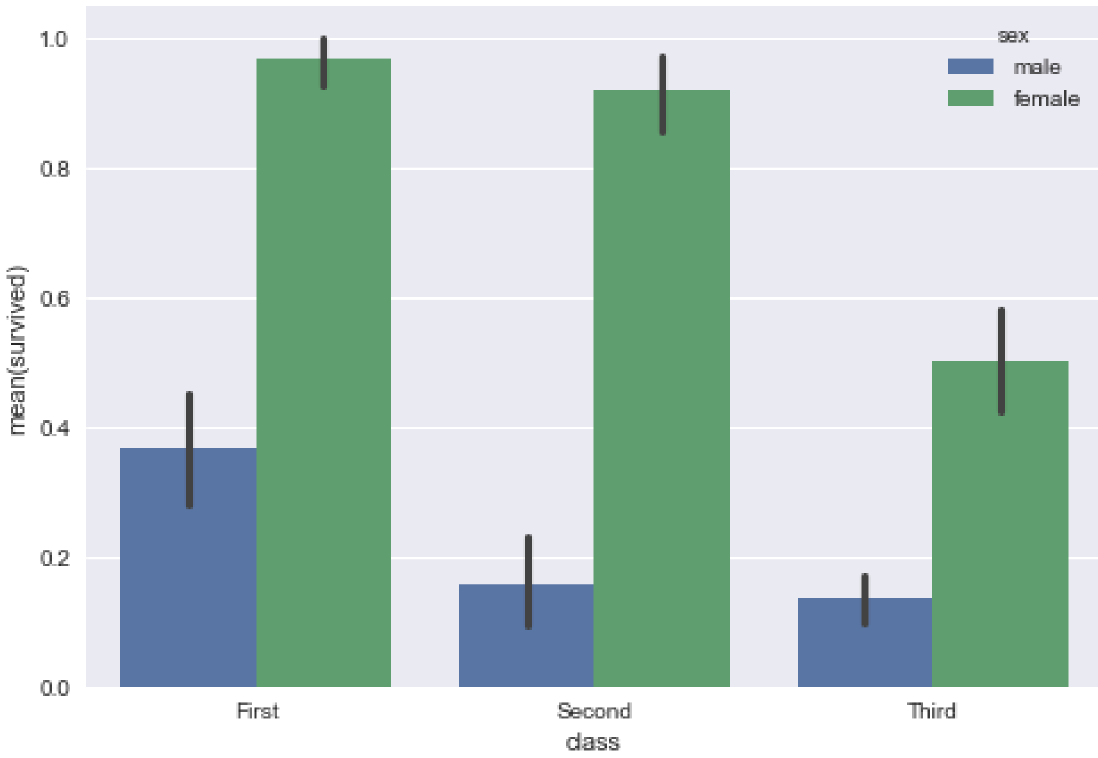


Notice how it was so easy to create the plot! You simply passed in the entire dataset, and just specified the `x` and `y` fields that you wanted exposed for the barplot. Behind the scenes seaborn ignored `NaN` values for you and automatically calculated the survival rate to plot. Also, that black tick is a 95% confidence interval that seaborn plots.

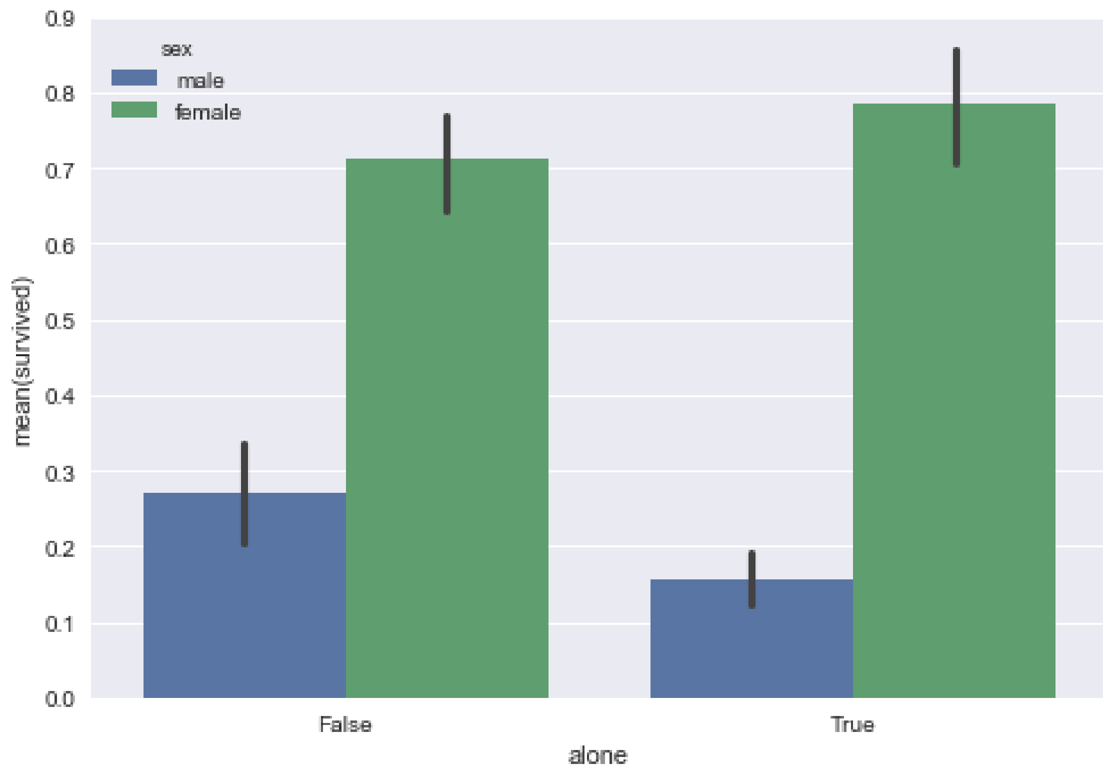
So we see that females were much more likely to make it out alive. What other factors do you think could have an impact on surival rate? Plot a couple more barplots below. Make sure to use *categorical* values, not something numerical like age or fare.

In [13]:
```
## YOUR CODE HERE
sns.barplot(x='class', y = 'survived', hue = 'sex', data=titanic)
```

Out[13]: `<matplotlib.axes._subplots.AxesSubplot at 0x2575b654518>`




```
In [14]: ## YOUR CODE HERE
         sns.barplot(x='alone', y = 'survived', hue='sex', data=titanic)

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2575a12a668>
```

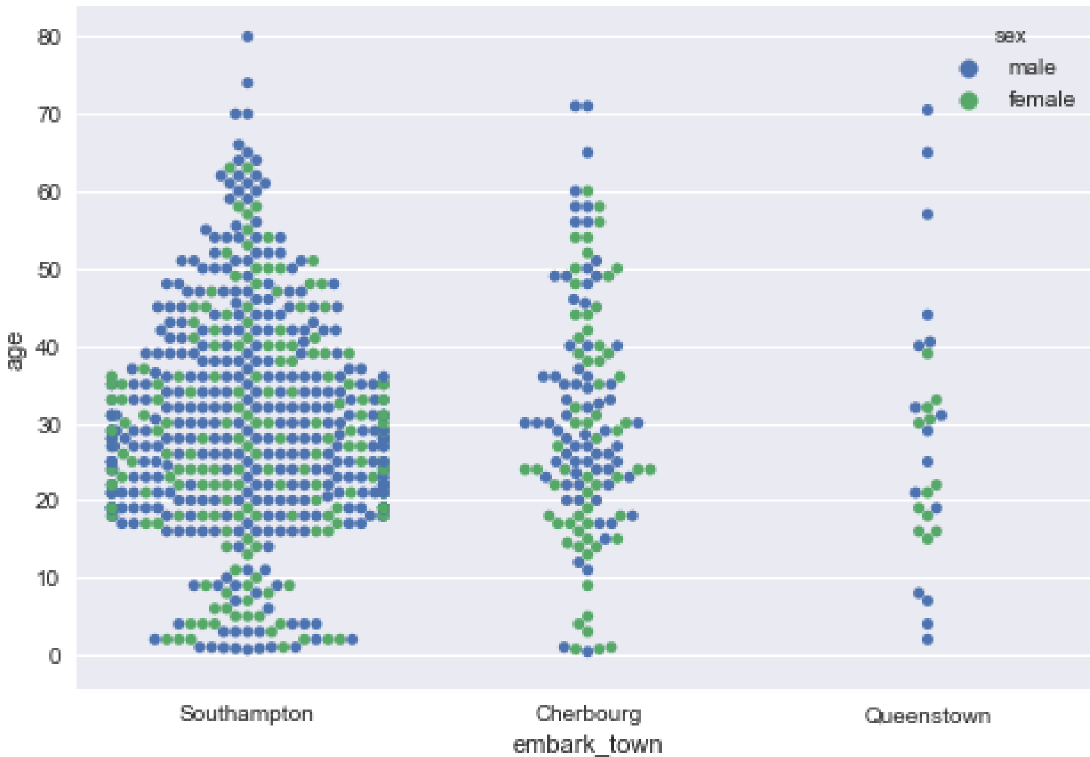


What if we wanted to add a further sex breakdown for the categories chosen above? Go back and add a `hue='sex'` parameter for the couple plots you just created, and seaborn will split each bar into a male/female comparison.

Now we want to compare the embarking town vs the age of the individuals. We don't simply want to use a barplot, since that will just give the average age; rather, we would like more insight into the relative and numeric *distribution* of ages.

A good tool to help us here is `swarmplot`. Use this function to view `embark_town` vs `age`, again using `sex` as the `hue`.

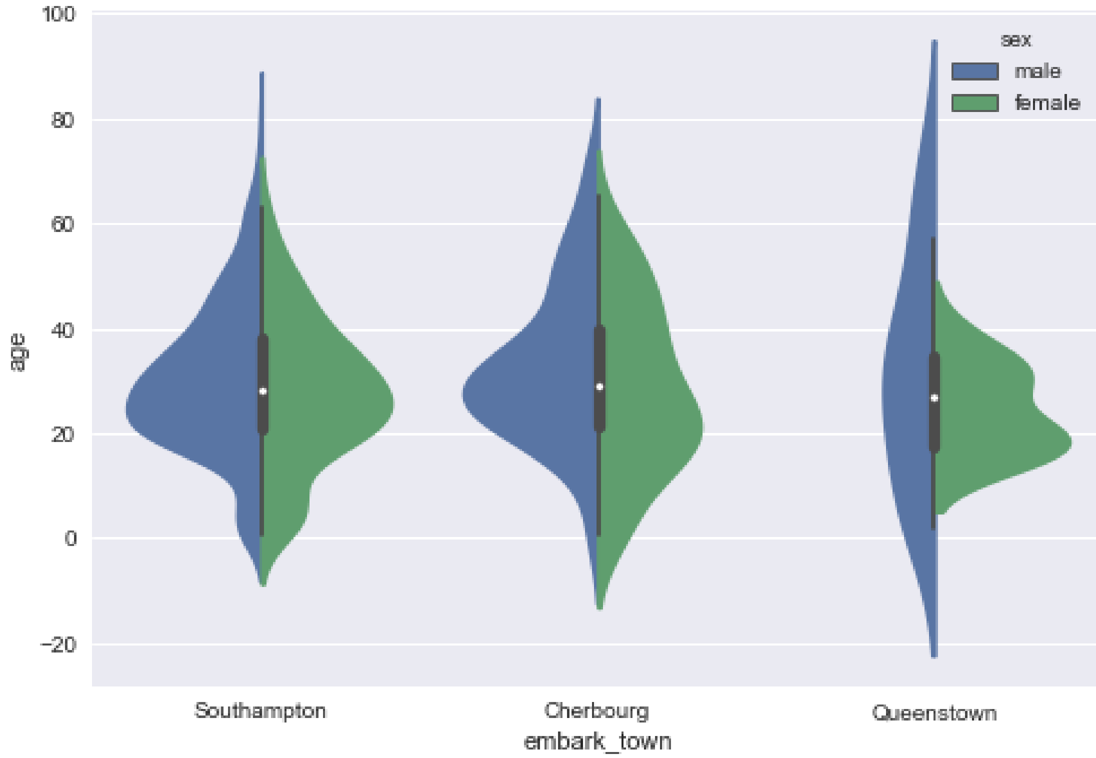
```
In [15]: ## YOUR CODE HERE
         sns.swarmplot(x='embark_town', y='age', hue = 'sex', data=titanic)

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2575b5d81d0>
```



Cool! This gives us much more information. What if we didn't care about the number of individuals in each category at all, but rather just wanted to see the *distribution* in each category? `violinplot` plots a density distribution. Plot that. Keep the `hue`.

In [16]:
```
## YOUR CODE HERE
sns.violinplot(x='embark_town', y='age', hue='sex', split='true', data=titanic)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2575b51eba8>
```



Go back and clean up the violinplot by adding `split='True'` parameter.

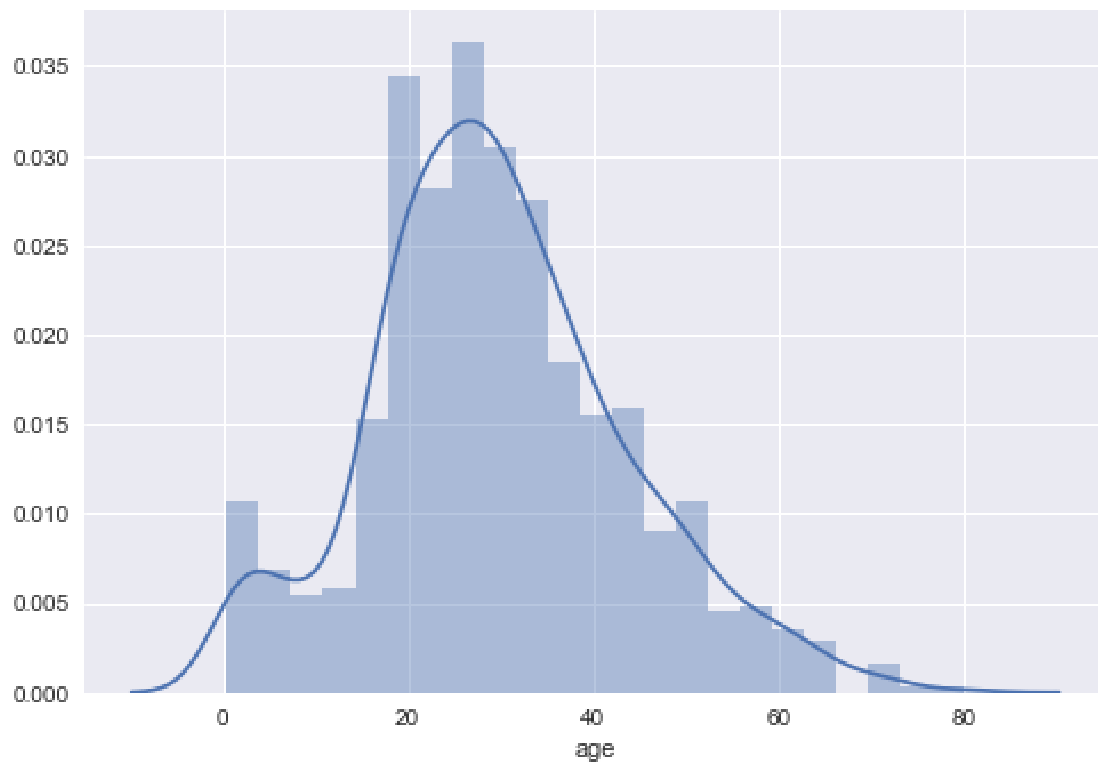
Now take a few seconds to look at the graphs you've created of this data. What are some observations? Jot a couple down here.

- It was far more likely that a female survives vs a male. It is also more likeley that someone in first class survives and if they are not single.
- The majority of people leaving from Southampton, Cherbourg, and Queenstown are between 20-40. Females leaving are slightly younger.

As I mentioned, data is categorical or numeric. We already started getting into numerical data with the swarmplot and violinplot. We will now explore a couple more examples.

Let's look at the distribution of ages. Use `displot` to make a histogram of just the ages.

```
In [17]: ## YOUR CODE HERE
         sns.distplot(titanicFilled.age)

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x2575a18c940>
```

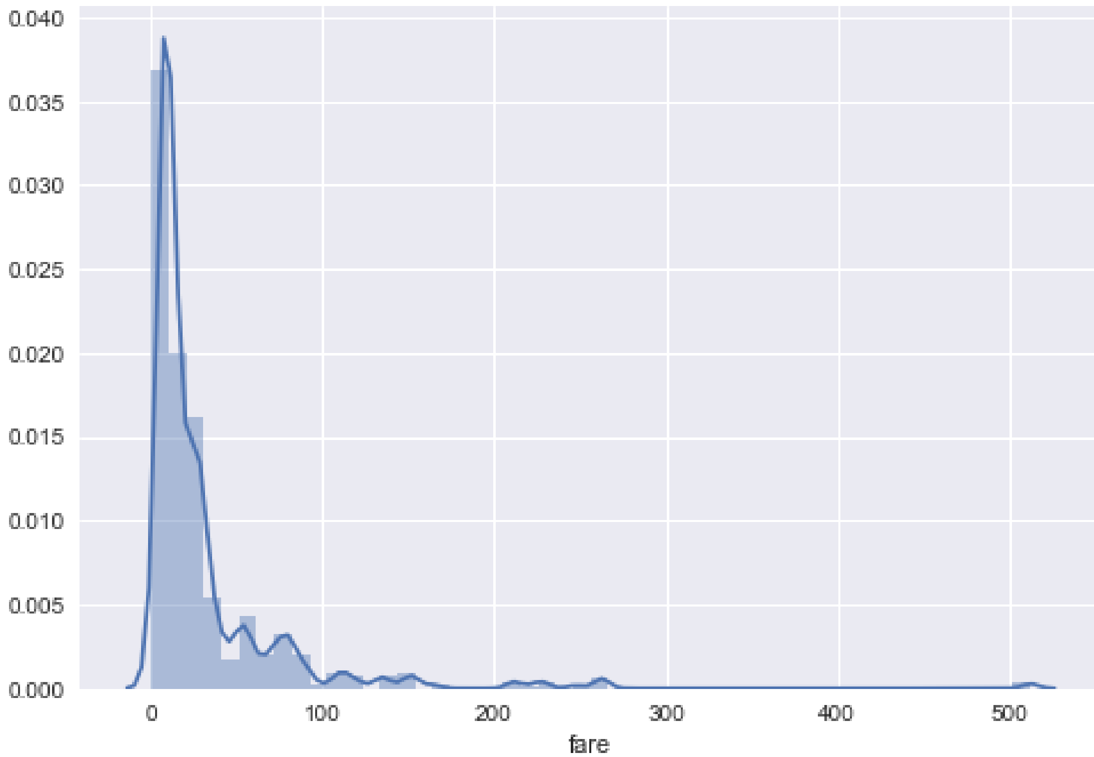


If you did your missing value imputation by average value, your results will look very skewed. This is why we don't normally just fill in an average. As a quick fix for now, though, you can filter out the age values that equal the mean before passing it in to `displot`. Do this.

A histogram can nicely represent numerical data by breaking up numerical ranges into chunks so that it is easier to visualize. As you might notice from above, seaborn also automatically plots a gaussian kernel density estimate.

Do the same thing for fares - do you notice something odd about that histogram? What does that skew mean?

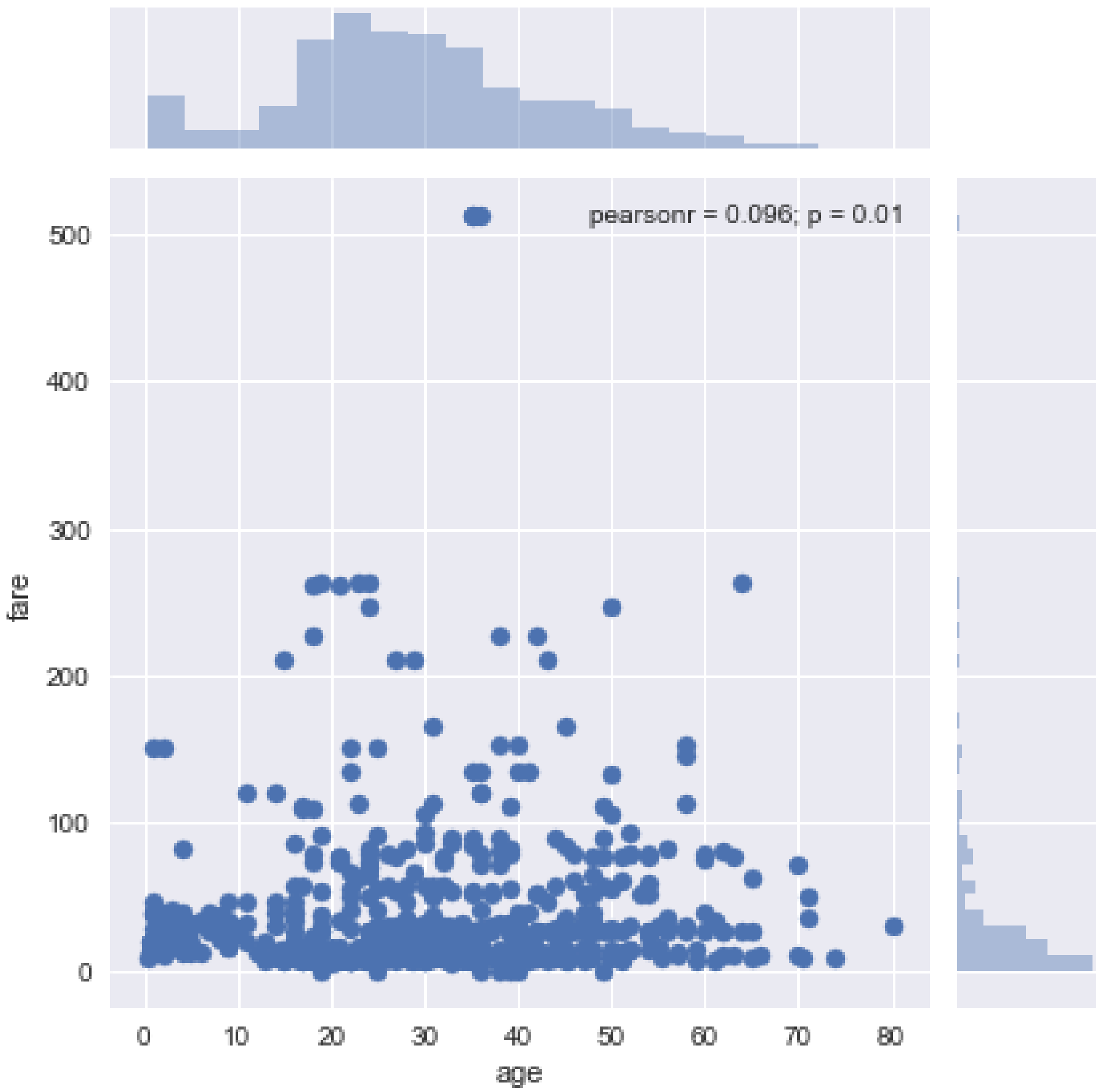
```
In [18]: ## YOUR CODE HERE
         sns.distplot(titanic.fare)

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2575b7ed198>
```



The majority of fares are very cheap. Below $50.

Now, using the `jointplot` function, make a scatterplot of the `age` and `fare` variables to see if there is any relationship between the two.

```
In [19]: ## YOUR CODE HERE
         sns.jointplot(x='age', y='fare', data=titanic)

Out[19]: <seaborn.axisgrid.JointGrid at 0x2575b9c9c18>
```

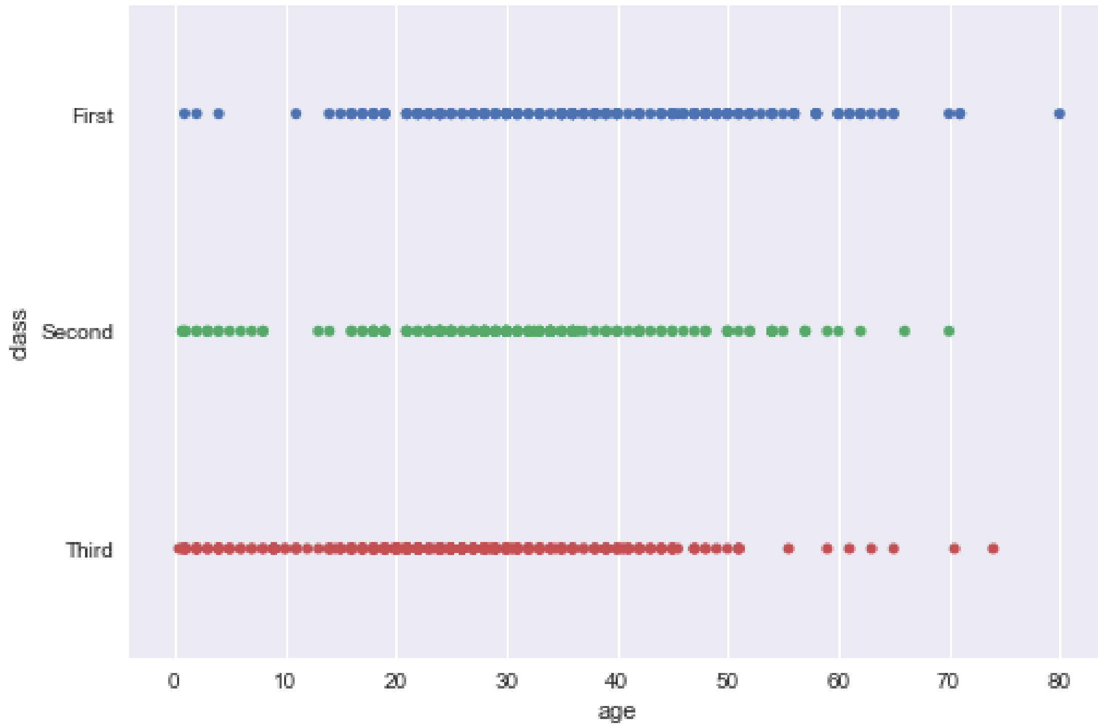


Scatterplots allow one to easily see trends/coorelations in data. As you can see here, there seems to be very little correlation. Also observe that seaborn automatically plots histograms.

Now, use a seaborn function we haven't used yet to plot something. The API has a list of all the methods.

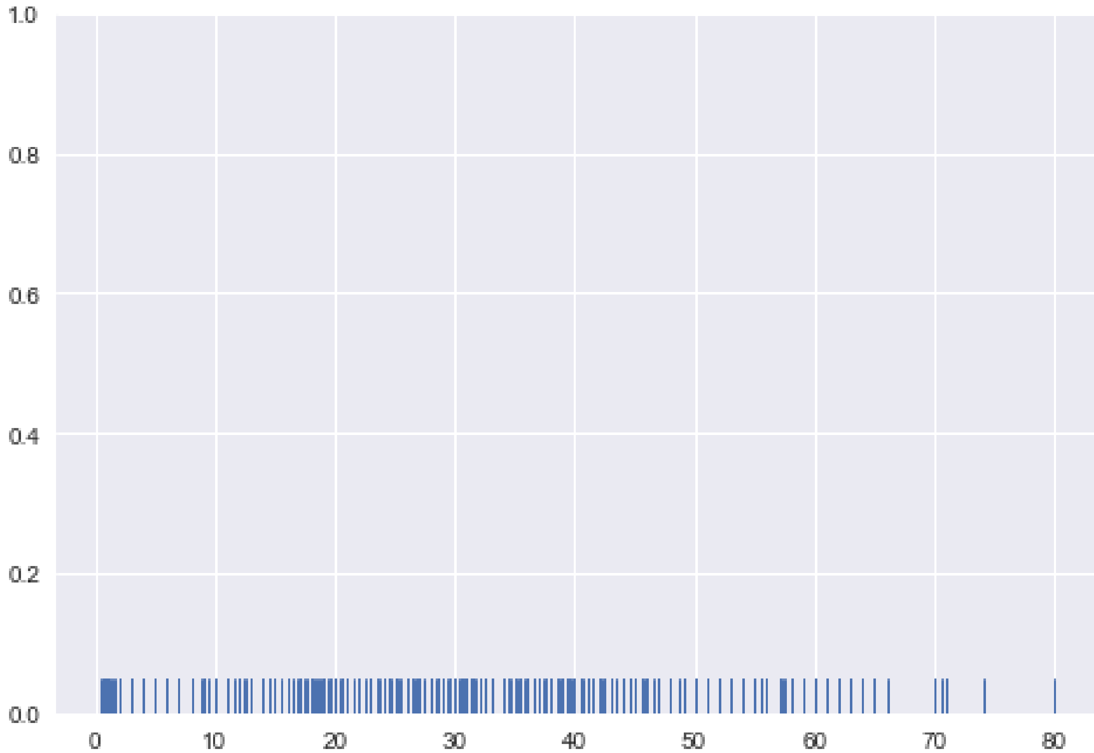
```
In [20]: ## YOUR CODE HERE
         sns.stripplot(y='class', x='age', data=titanic)
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2575bcbf6d8>
```




```
In [21]: sns.rugplot(titanicFilled.age)
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x2575bde69e8>
```




```
In [ ]:
```