HW 1 - Data Visualization + Data Preparation

February 22, 2021

1 Homework 1: Exploring & Visualizing Data

1.1 Part 1: Data Exploration and Preparation

1.2 Setup

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

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.

[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).

Please import seaborn as sns

[2]:
```
import seaborn as sns
sns.set() # This sets aesthetic parameters in one step
```

First load the `titanic` dataset directly from seaborn. The `load_dataset` function will return a pandas dataframe. Documentation: https://seaborn.pydata.org/generated/seaborn.load_dataset.html

[3]:
```
titanic = sns.load_dataset("titanic")
```

1.3 Preparing a new dataset (Pandas)

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

[4]:
```
import numpy as np
import pandas as pd
```

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

```
[5]: titanic.describe()
```

```
[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
```

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

```
[6]:    survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
     0         0       3    male  22.0      1      0   7.2500        S  Third
     1         1       1  female  38.0      1      0  71.2833        C  First
     2         1       3  female  26.0      0      0   7.9250        S  Third
     3         1       1  female  35.0      1      0  53.1000        S  First
     4         0       3    male  35.0      0      0   8.0500        S  Third

          who  adult_male deck  embark_town alive  alone
     0    man        True  NaN  Southampton    no  False
     1  woman       False    C    Cherbourg   yes  False
     2  woman       False  NaN  Southampton   yes   True
     3  woman       False    C  Southampton   yes  False
     4    man        True  NaN  Southampton    no   True
```

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

- What was the age of the oldest person on board? 80
- What was the survival rate of people on board? 38.3838%
- 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:

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

```
[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.

One quick observation is that there are a couple of repetitive columns. One example is the 'embarked_town' and 'embarked' columns which are conveying the same information. As we are preparing this dataset to be used in a model we only want to keep one of those columns. **Find the other repetitive pair of columns and drop the non-numerical one.**

Hint: is there a function that could helps us preview the first few items in the dataset?

Note: adult_male differs from sex since there could be children males

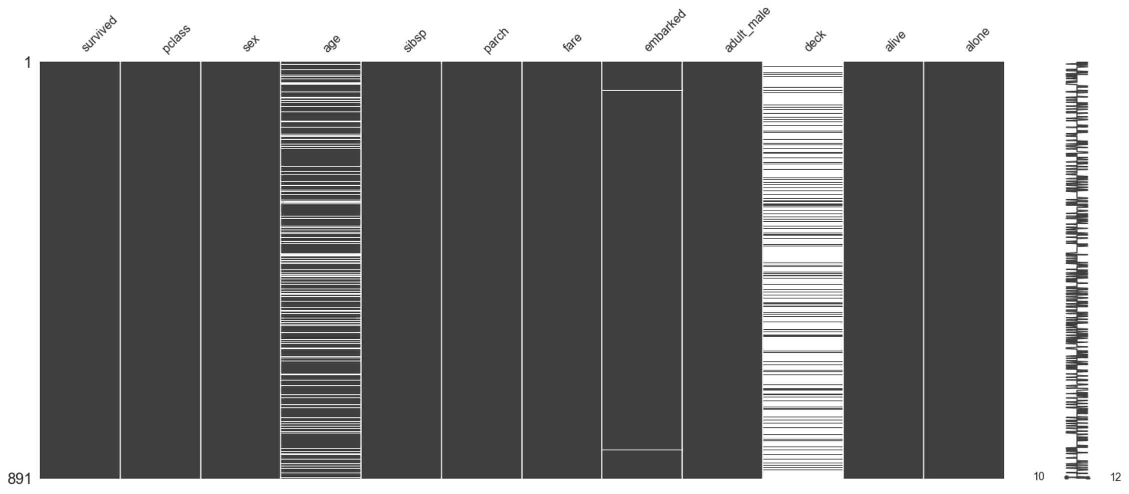
[8]:
```python
titanic.drop('embark_town', axis=1, inplace=True)
titanic.drop('who', axis=1, inplace=True)
titanic.drop('class', axis=1, inplace=True)
```

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. Import missingno as msno and use its matrix function on titanic. https://www.residentmar.io/2016/03/28/missingno.html

[9]:
```python
import missingno as msno

msno.matrix(titanic)
```

[9]:
```
<AxesSubplot:>
```



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 the deck column from the dataset** since it's not that relevant. Make sure to set inplace=True. Documentation: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html.

[10]:
```python
titanic.drop('deck', axis=1, inplace=True)
```

If we were to feed our dataset into a model, the model cannot understand features such as 'True', 'Yes', 'male'. You have to turn the data into binary format, 1 for positive and 0 negative. **Fill in toBinary and then use it to binarize the 'alive', 'sex', 'alone', and 'adult_male' columns.**

[11]:
```
def toBinary(data, positive):
    data.where(data == positive, 0, inplace=True)
    data.where(data != positive, 1, inplace=True)
    pass

toBinary(titanic['alive'], 'yes')
toBinary(titanic['sex'], 'male')
toBinary(titanic['alone'], True)
toBinary(titanic['adult_male'], True)
```

[12]:
```
titanic.head()
```

[12]:
```
   survived  pclass sex   age  sibsp  parch     fare embarked  adult_male  \
0         0       3   1  22.0      1      0   7.2500        S         1.0
1         1       1   0  38.0      1      0  71.2833        C         0.0
2         1       3   0  26.0      0      0   7.9250        S         0.0
3         1       1   0  35.0      1      0  53.1000        S         0.0
4         0       3   1  35.0      0      0   8.0500        S         1.0

  alive  alone
0     0    0.0
1     1    0.0
2     1    1.0
3     1    0.0
4     0    1.0
```

We have to deal with the categorical columns that have more than two categories. For this situation we will use One-Hot encoding http://queirozf.com/entries/one-hot-encoding-a-feature-on-a-pandas-dataframe-an-example. Pandas has a good method 'get_dummies' that makes this task much easier. Read the documentation and fill in the code to **One-Hot encode the 'pclass' and 'embarked' columns. .concat the output from get_dummies and drop the original columns.**

[13]:
```
titanic = pd.concat([titanic, pd.get_dummies(titanic['embarked'])], axis=1)
titanic.drop('embarked', axis=1, inplace=True)

titanic = pd.concat([titanic, pd.get_dummies(titanic['pclass'])], axis=1)
titanic.drop('pclass', axis=1, inplace=True)
```

[14]:
```
titanic.head()
```

[14]:
```
   survived sex   age  sibsp  parch     fare  adult_male alive  alone  C  Q  \
0         0   1  22.0      1      0   7.2500         1.0     0    0.0  0  0
```

```
1          1   0  38.0      1      0  71.2833        0.0     1    0.0  1  0
2          1   0  26.0      0      0   7.9250        0.0     1    1.0  0  0
3          1   0  35.0      1      0  53.1000        0.0     1    0.0  0  0
4          0   1  35.0      0      0   8.0500        1.0     0    1.0  0  0

   S  1  2  3
0  1  0  0  1
1  0  1  0  0
2  1  0  0  1
3  1  1  0  0
4  1  0  0  1
```

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 will 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 - you should not see any white lines.

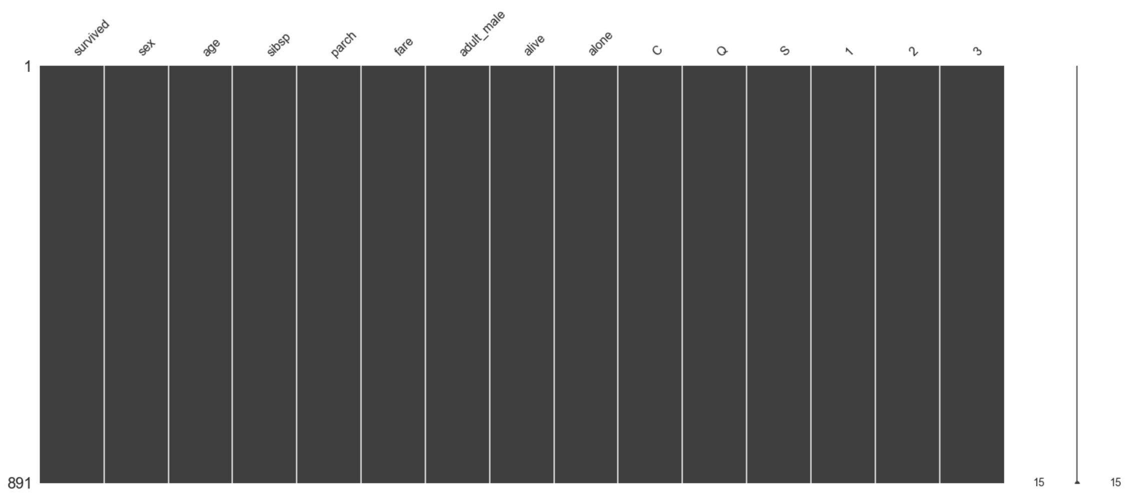
[15]:
```python
## Standard KNN Imputer
from sklearn.impute import KNNImputer
imp = KNNImputer(n_neighbors=5, weights='uniform')
ages = imp.fit_transform(titanic)[:, 2]
ages = np.around(ages)
titanic['age'] = ages

titanic.dropna(inplace=True)
```

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

[16]:
```python
msno.matrix(titanic)
```

[16]:
```
<AxesSubplot:>
```



[17]:
```
titanic.head()
```

[17]:
```
   survived  sex   age  sibsp  parch     fare  adult_male  alive  alone  C  Q  \
0         0    1  22.0      1      0   7.2500         1.0      0    0.0  0  0
1         1    0  38.0      1      0  71.2833         0.0      1    0.0  1  0
2         1    0  26.0      0      0   7.9250         0.0      1    1.0  0  0
3         1    0  35.0      1      0  53.1000         0.0      1    0.0  0  0
4         0    1  35.0      0      0   8.0500         1.0      0    1.0  0  0

   S  1  2  3
0  1  0  0  1
1  0  1  0  0
2  1  0  0  1
3  1  1  0  0
4  1  0  0  1
```

As we learned in lecture, we have to Normalize our numerical data points. Now you must **define the function 'Standardize' that standardizes the column passed in**, and then pass in the 'age' and 'fare' columns.

Hint: There is a `sklearn` function that can help with normalization

[18]:
```
titanic.describe()
```

[18]:
```
         survived         age       sibsp       parch        fare  adult_male  \
count  891.000000  891.000000  891.000000  891.000000  891.000000  891.000000
mean     0.383838   29.965208    0.523008    0.381594   32.204208    0.602694
std      0.486592   13.734559    1.102743    0.806057   49.693429    0.489615
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000   21.000000    0.000000    0.000000    7.910400    0.000000
50%      0.000000   29.000000    0.000000    0.000000   14.454200    1.000000
```

```
75%       1.000000   38.000000    1.000000    0.000000   31.000000    1.000000
max       1.000000   80.000000    8.000000    6.000000  512.329200    1.000000

            alone           C           Q           S           1           2  \
count  891.000000  891.000000  891.000000  891.000000  891.000000  891.000000
mean     0.602694    0.188552    0.086420    0.722783    0.242424    0.206510
std      0.489615    0.391372    0.281141    0.447876    0.428790    0.405028
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      1.000000    0.000000    0.000000    1.000000    0.000000    0.000000
75%      1.000000    0.000000    0.000000    1.000000    0.000000    0.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

                3
count  891.000000
mean     0.551066
std      0.497665
min      0.000000
25%      0.000000
50%      1.000000
75%      1.000000
max      1.000000
```

[19]:
```python
from sklearn import preprocessing
def Standardize(data):
    return preprocessing.normalize([data], norm='max')[0]

titanic['age'] =  Standardize(titanic['age'])
titanic['fare'] = Standardize(titanic['fare'])
```

[20]:
```python
titanic.head()
```

[20]:
```
   survived sex     age  sibsp  parch      fare  adult_male alive  alone  C  \
0         0   1  0.2750      1      0  0.014151         1.0     0    0.0  0
1         1   0  0.4750      1      0  0.139136         0.0     1    0.0  1
2         1   0  0.3250      0      0  0.015469         0.0     1    1.0  0
3         1   0  0.4375      1      0  0.103644         0.0     1    0.0  0
4         0   1  0.4375      0      0  0.015713         1.0     0    1.0  0

   Q  S  1  2  3
0  0  1  0  0  1
1  0  0  1  0  0
2  0  1  0  0  1
3  0  1  1  0  0
4  0  1  0  0  1
```

Seaborn can handle categorical data and NaN directly, **reload the titanic dataset to its original**

version

[21]:
```
titanic = sns.load_dataset("titanic")
```

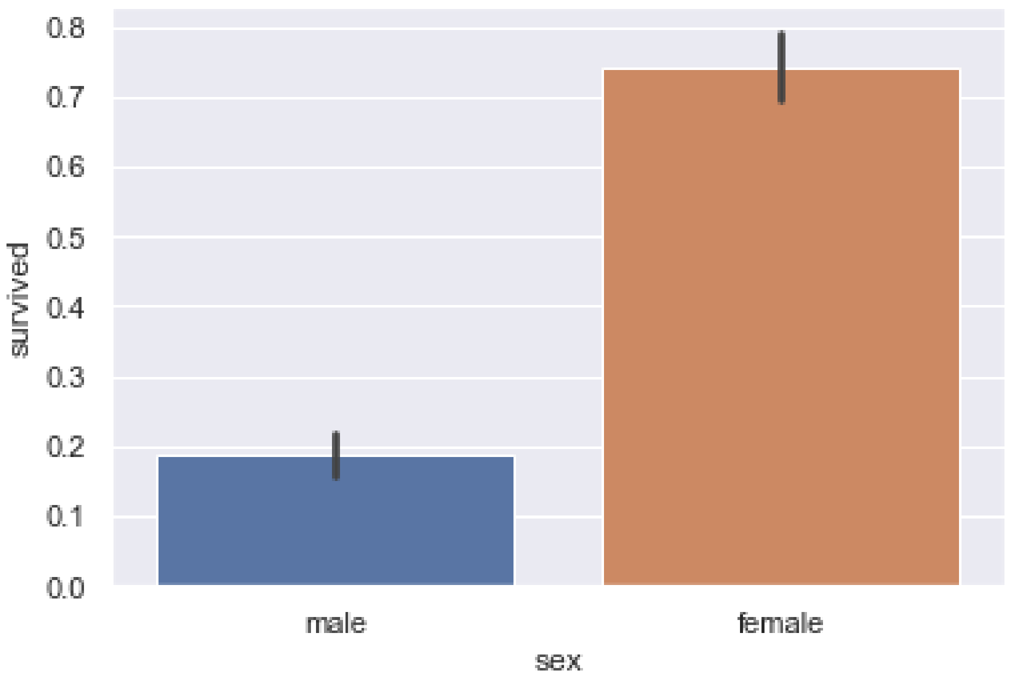
There are 2 types of data in any dataset: categorial and numerical data. We will first explore categorical data.

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.

[22]:
```
sns.barplot(x='sex', y='survived', data=titanic)
```

[22]:
```
<AxesSubplot:xlabel='sex', ylabel='survived'>
```

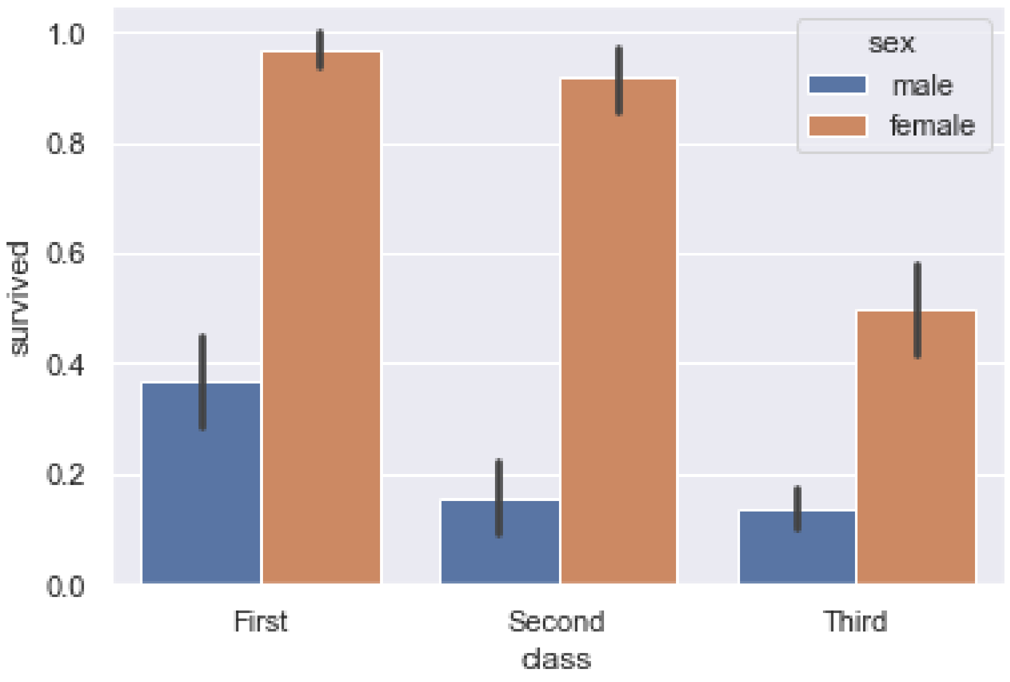


Notice how it was so easy to create the plot! You simply passed in the entire dataset, and just specified the `x` and `y` fields that you wanted exposed for the barplot. Behind the scenes seaborn ignored `NaN` values for you and automatically calculated the survival rate to plot. Also, that black tick 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 survival rate? ** Plot a couple more barplots below. ** Make sure to use *categorical* values, not something numerical like age or fare.

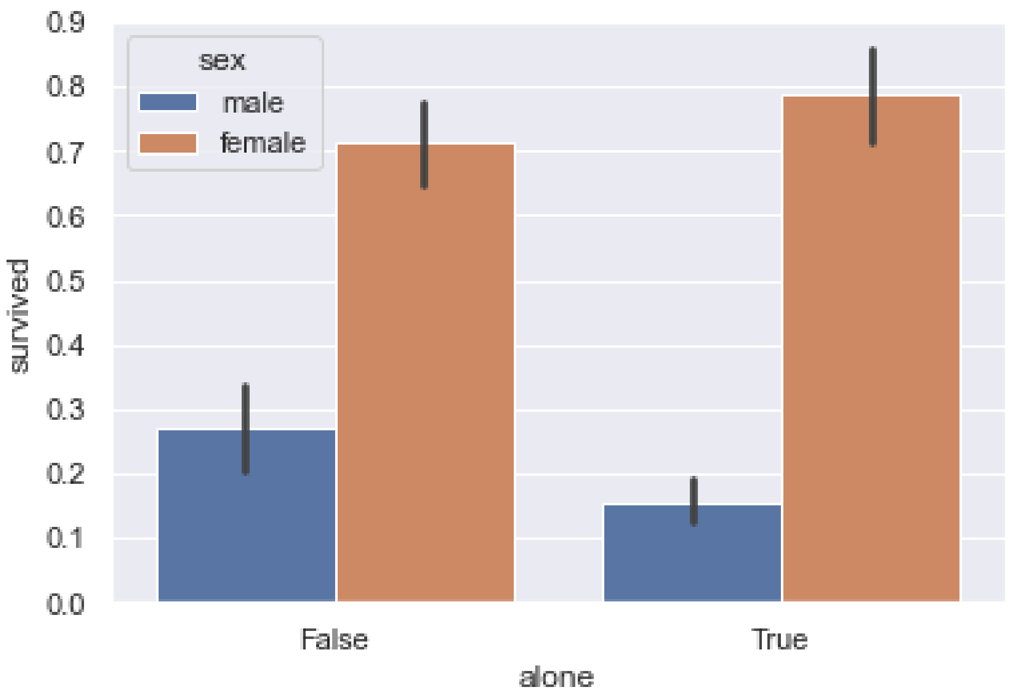
[23]:
```
sns.barplot(x='class', y='survived', data=titanic, hue='sex')
```

[23]:
```
<AxesSubplot:xlabel='class', ylabel='survived'>
```



[24]:
```
sns.barplot(x='alone', y='survived', data=titanic, hue='sex')
```

[24]:
```
<AxesSubplot:xlabel='alone', ylabel='survived'>
```



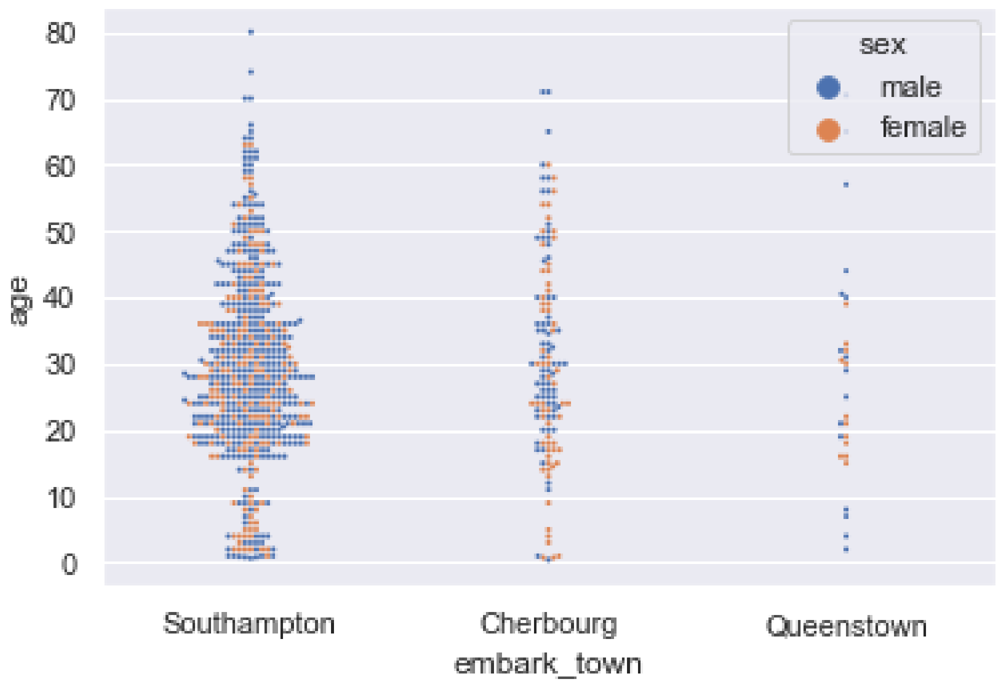
What if we wanted to add a further sex breakdown for the categories chosen above? Go back and add a `hue='sex'` parameter for the couple plots you just created, and seaborn will split each bar into a male/female comparison.

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`.

```
[25]: sns.swarmplot(x='embark_town', y='age', data=titanic, hue='sex', size=2)
```

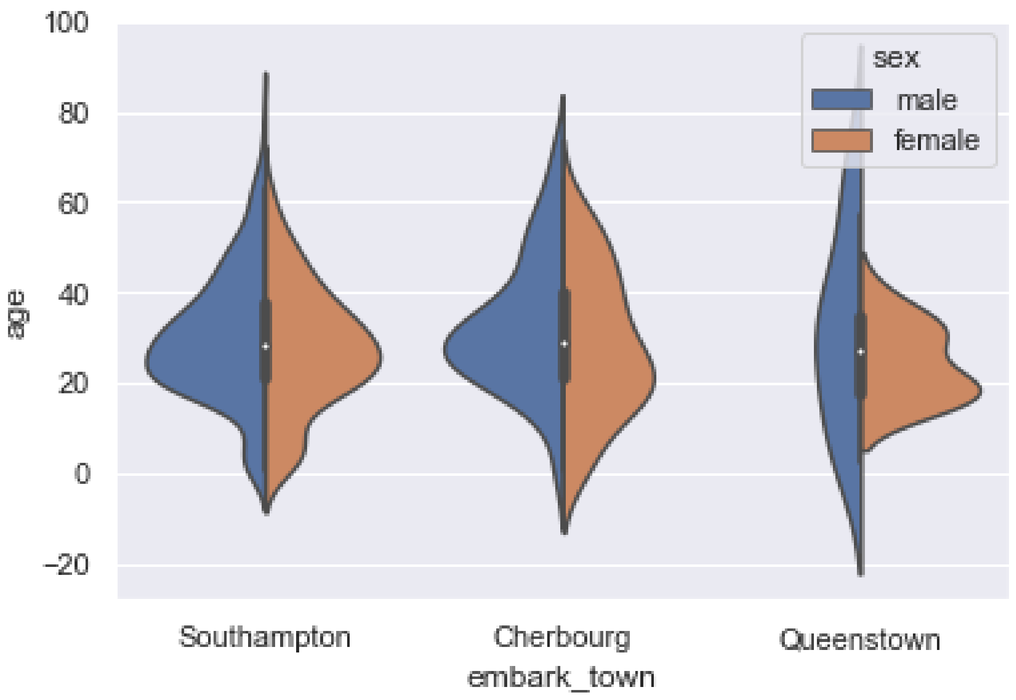
```
[25]: <AxesSubplot:xlabel='embark_town', ylabel='age'>
```



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`.

```
[26]: sns.violinplot(x='embark_town', y='age', data=titanic, hue='sex', split=True)
```

```
[26]: <AxesSubplot:xlabel='embark_town', ylabel='age'>
```



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.

1.4 #### Your observations Here

- Females tended to have a more centered distrubution in age while men were more varied
- Most of the passengers embarked from Southampton
- Females had a much higher survival rate than men

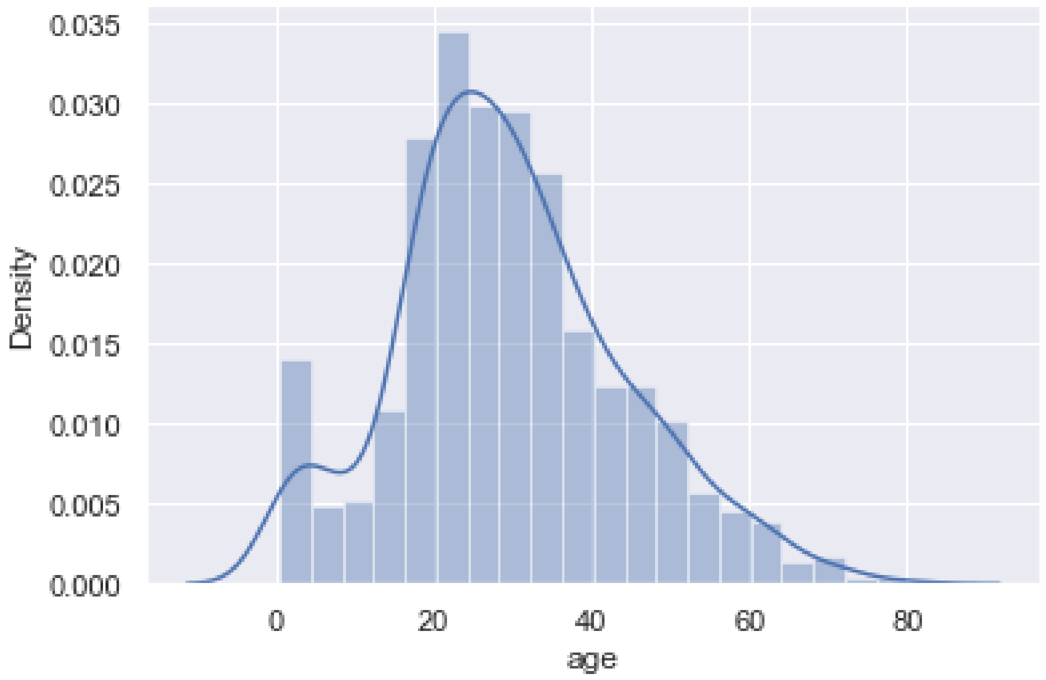
As I mentioned, data is categorical or numeric. We already started getting into numerical data with the swarmplot and violinplot. We will now explore a couple more examples.

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

```
[27]: sns.distplot(titanic['age'])
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
[27]: <AxesSubplot:xlabel='age', ylabel='Density'>
```



If you did your missing value imputation by average value (If we had not reloaded the data set), 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`.

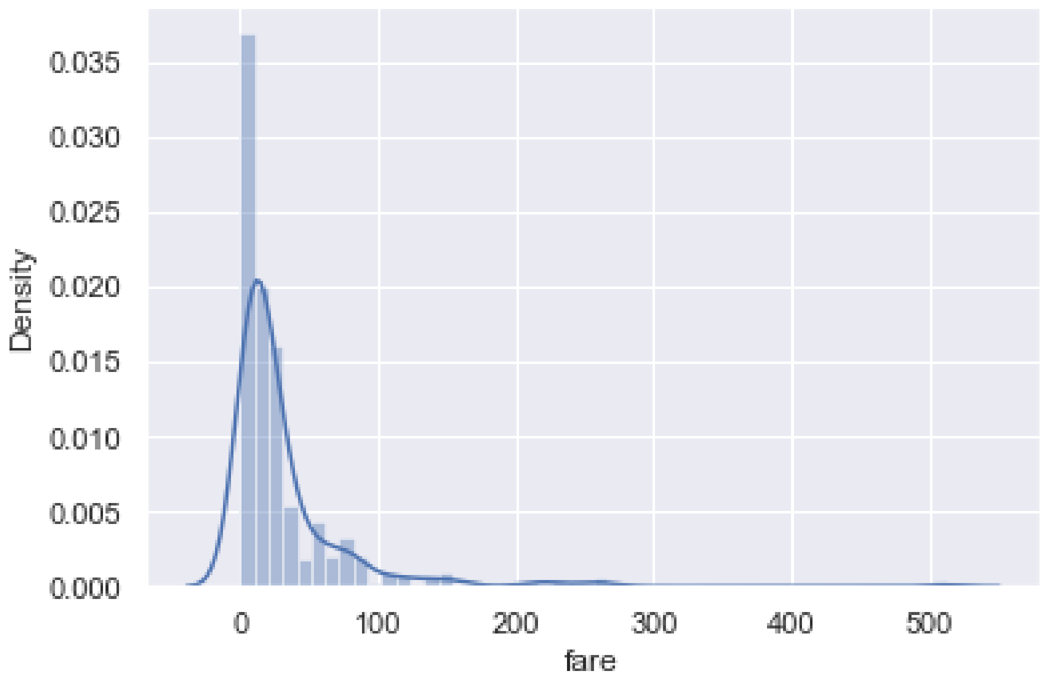
A histogram can nicely represent numerical data by breaking up numerical ranges into chunks so that it is easier to visualize. As you might notice from above, seaborn also automatically plots a gaussian kernel density estimate.

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

```
[28]: sns.distplot(titanic['fare'])
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

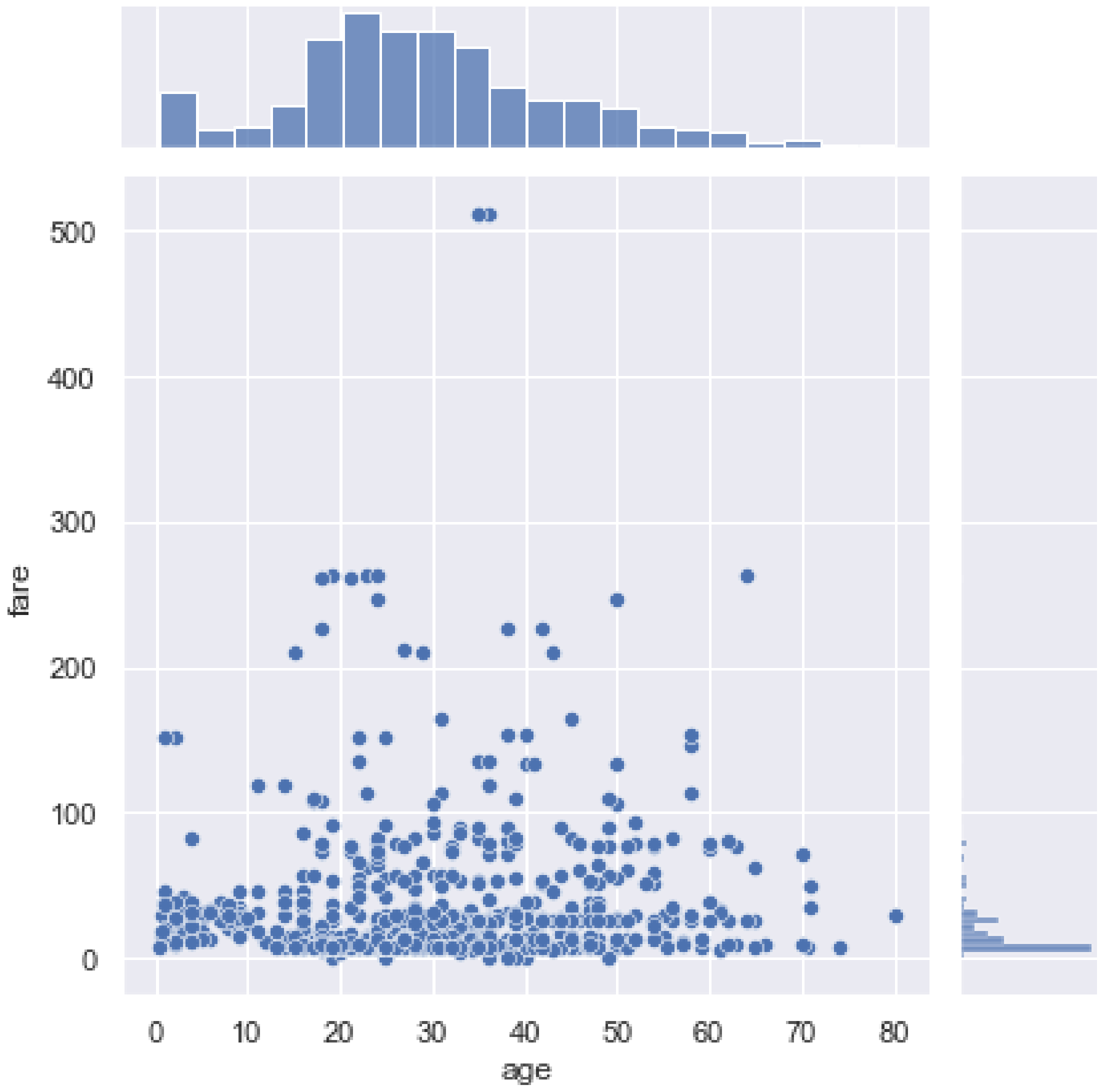
```
[28]: <AxesSubplot:xlabel='fare', ylabel='Density'>
```



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

```
[29]: sns.jointplot(data=titanic, x="age", y="fare")
```

```
[29]: <seaborn.axisgrid.JointGrid at 0x1200c6c40>
```

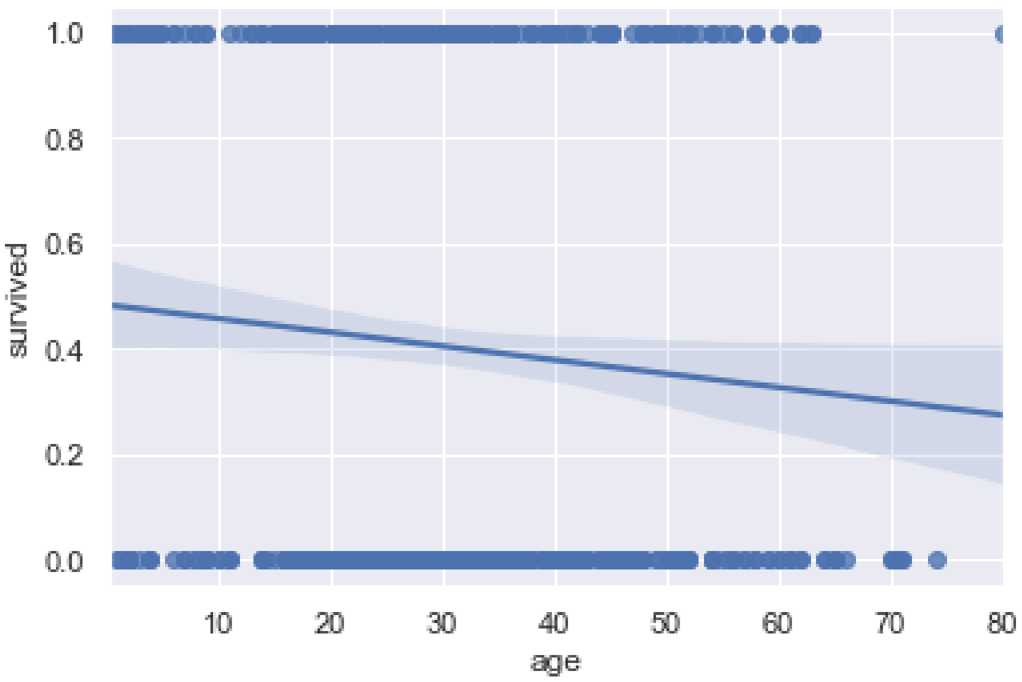


Scatterplots allow one to easily see trends/coorelations in data. As you can see here, there seems to be very little correlation. Also observe that seaborn automatically plots histograms.

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

[30]:
```
sns.regplot(data=titanic, x="age", y="survived")
```

[30]:
```
<AxesSubplot:xlabel='age', ylabel='survived'>
```


1.0
0.8
0.6
0.4
0.2
0.0
survived
10
20
30
40
50
60
70
80
age