# Homework 1: Exploring & Visualizing Data

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

## 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
import matplotlib.pyplot as plt
%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 pandas as pd
import numpy as np
from collections import defaultdict
```

## Getting to know a new dataset

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

In [16]:

```
titanic = sns.load_dataset('titanic')
```

Use a couple pandas functions to get a quick overview and some statistics on the dataset (remember the commands we used from lecture). Take a quick glance at the overview you create.

In [17]:

```
## YOUR CODE HERE
titanic.head()
```

Out[17]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | ac |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22 | 1 | 0 | 7.2500 | S | Third | man | Tr |
| **1** | 1 | 1 | female | 38 | 1 | 0 | 71.2833 | C | First | woman | Fa |
| **2** | 1 | 3 | female | 26 | 0 | 0 | 7.9250 | S | Third | woman | Fa |
| **3** | 1 | 1 | female | 35 | 1 | 0 | 53.1000 | S | First | woman | Fa |
| **4** | 0 | 3 | male | 35 | 0 | 0 | 8.0500 | S | Third | man | Tr |

In [18]:

```
## YOUR CODE HERE
titanic.describe()
```

Out[18]:

| | survived | pclass | age | sibsp | parch | fare | ac |
|---|---|---|---|---|---|---|---|
| **count** | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 | 89 |
| **mean** | 0.383838 | 2.308642 | 29.699118 | 0.523008 | 0.381594 | 32.204208 | 0. |
| **std** | 0.486592 | 0.836071 | 14.526497 | 1.102743 | 0.806057 | 49.693429 | 0. |
| **min** | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 | Fa |
| **25%** | 0.000000 | 2.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 | 0 |
| **50%** | 0.000000 | 3.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 | 1 |
| **75%** | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 | 1 |
| **max** | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 | Tr |

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 (£/$ not sure)

Pro tip: What about if we wanted to know not just the overall survival rate, but the survival rate broken down by sex and embark_town?

In [19]:

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

Out[19]:

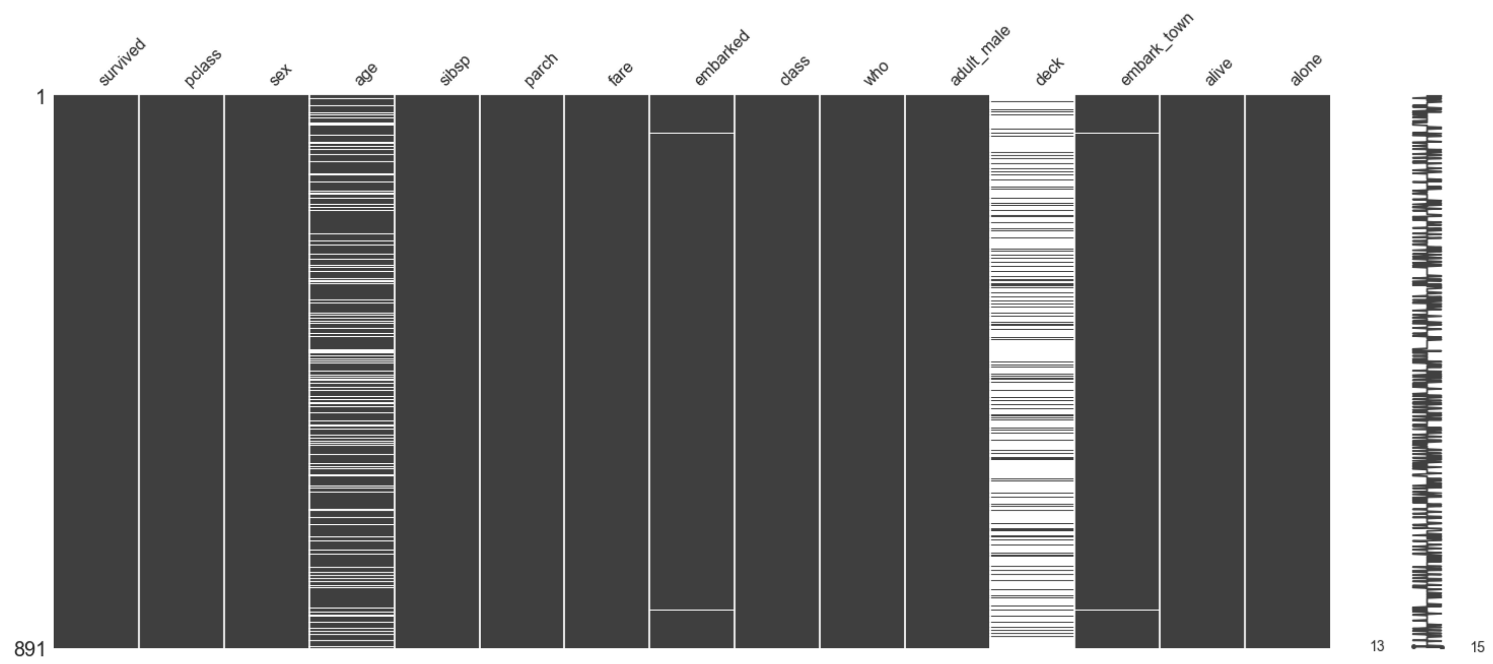
```
sex     embark_town
female  Cherbourg      0.876712
        Queenstown     0.750000
        Southampton    0.689655
male    Cherbourg      0.305263
        Queenstown     0.073171
        Southampton    0.174603
Name: survived, dtype: float64
```

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 [20]:

```
msno.matrix(titanic)
```



The white lines show us the missing data. One quick observation is that `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 [21]:

```
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 `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 (http://www.stat.columbia.edu/~gelman/arm/missing.pdf)). We can simply put in the average age value for the missing ages. But this really isn't so great as it would skew our stats towards the mean without taking into account various trends within the data.

If we assume that the data is missing *at random* (which actually is rarely the case), we can fit a model to predict the missing value based on the other factors in the dataset. One popular way to do this is a linear regression (coming forth in lecture #3!) `regplot` (https://seaborn.pydata.org/generated/seaborn.regplot.html) is a seaborn function to easily plot a linear regression.

Considering all these factors, assumptions and trade-offs, you must now make you own decision on how to deal with the missing data. You may choose any of the methods discussed above, or choose to not do anything at all (in which case you would only drop missing values when plotting). After writing your code below, verify the result by rerunning the matrix.

In [22]:

```
## YOUR CODE HERE
"""def convert_sex(titanic):
    if titanic["sex"] == "male":
        return 1
    else:
        return 0
titanic["bool_sex"] = titanic.apply(convert_sex,axis=1)
sns.regplot(y="age",x="bool_sex",data=titanic)

# not sure how to impliment multiple regression so sticking to average
def update_age(age):
    return 29.7
titanic["age"] = titanic.apply(update_age, axis=1)"""
titanic = titanic.dropna()
titanic.describe()
```

Out[22]:

| | survived | pclass | age | sibsp | parch | fare | ac |
|---|---|---|---|---|---|---|---|
| **count** | 712.000000 | 712.000000 | 712.000000 | 712.000000 | 712.000000 | 712.000000 | 71 |
| **mean** | 0.404494 | 2.240169 | 29.642093 | 0.514045 | 0.432584 | 34.567251 | 0. |
| **std** | 0.491139 | 0.836854 | 14.492933 | 0.930692 | 0.854181 | 52.938648 | 0. |
| **min** | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 | Fa |
| **25%** | 0.000000 | 1.000000 | 20.000000 | 0.000000 | 0.000000 | 8.050000 | 0 |
| **50%** | 0.000000 | 2.000000 | 28.000000 | 0.000000 | 0.000000 | 15.645850 | 1 |
| **75%** | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 1.000000 | 33.000000 | 1 |
| **max** | 1.000000 | 3.000000 | 80.000000 | 5.000000 | 6.000000 | 512.329200 | Tr |

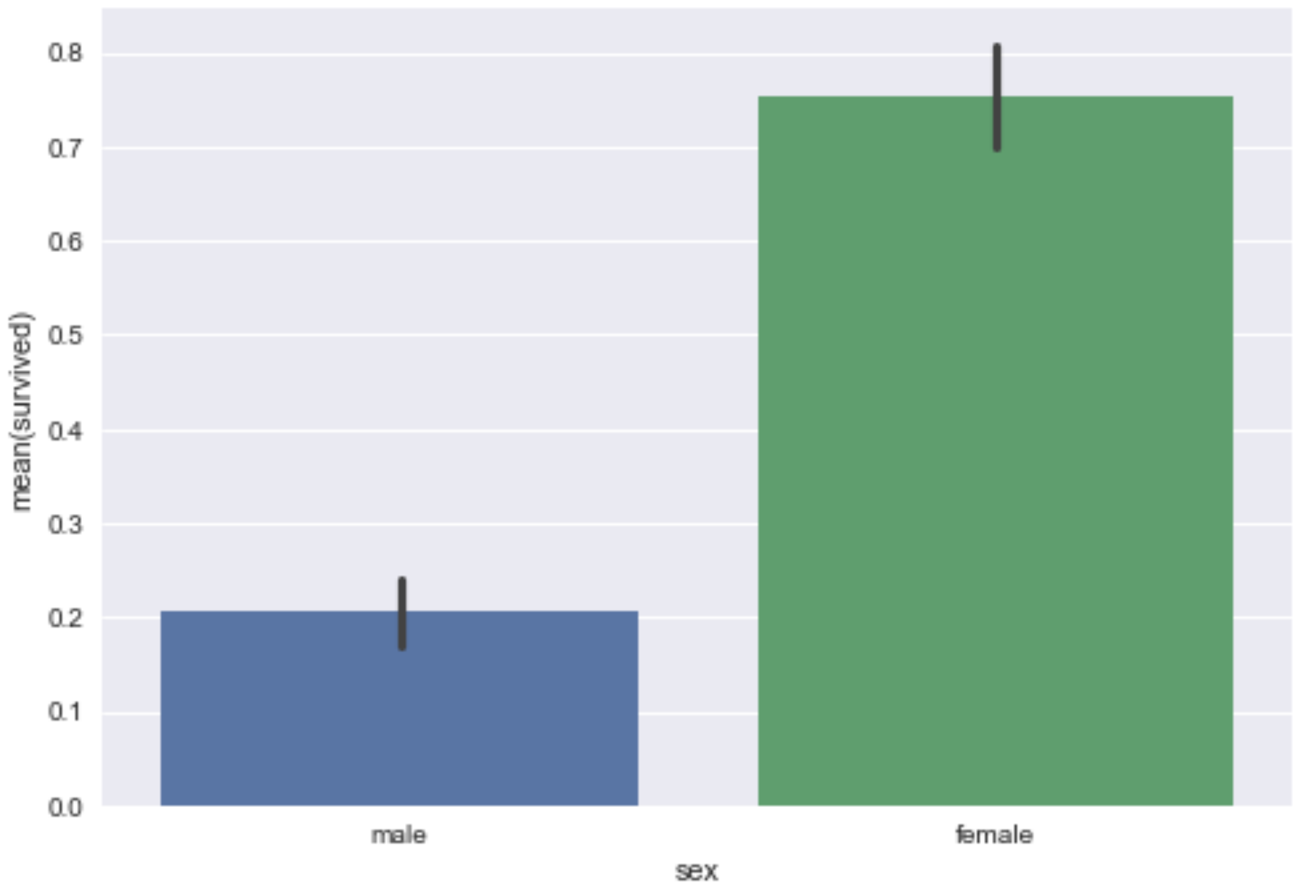
## Intro to Seaborn

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 (https://seaborn.pydata.org/generated/seaborn.barplot.html) and example here (http://seaborn.pydata.org/examples/color_palettes.html), create a `barplot` to depict this. It should be a really simple one-liner.

In [23]:

```
## YOUR CODE HERE
ax = sns.barplot(x="sex", y="survived", data=titanic)
```

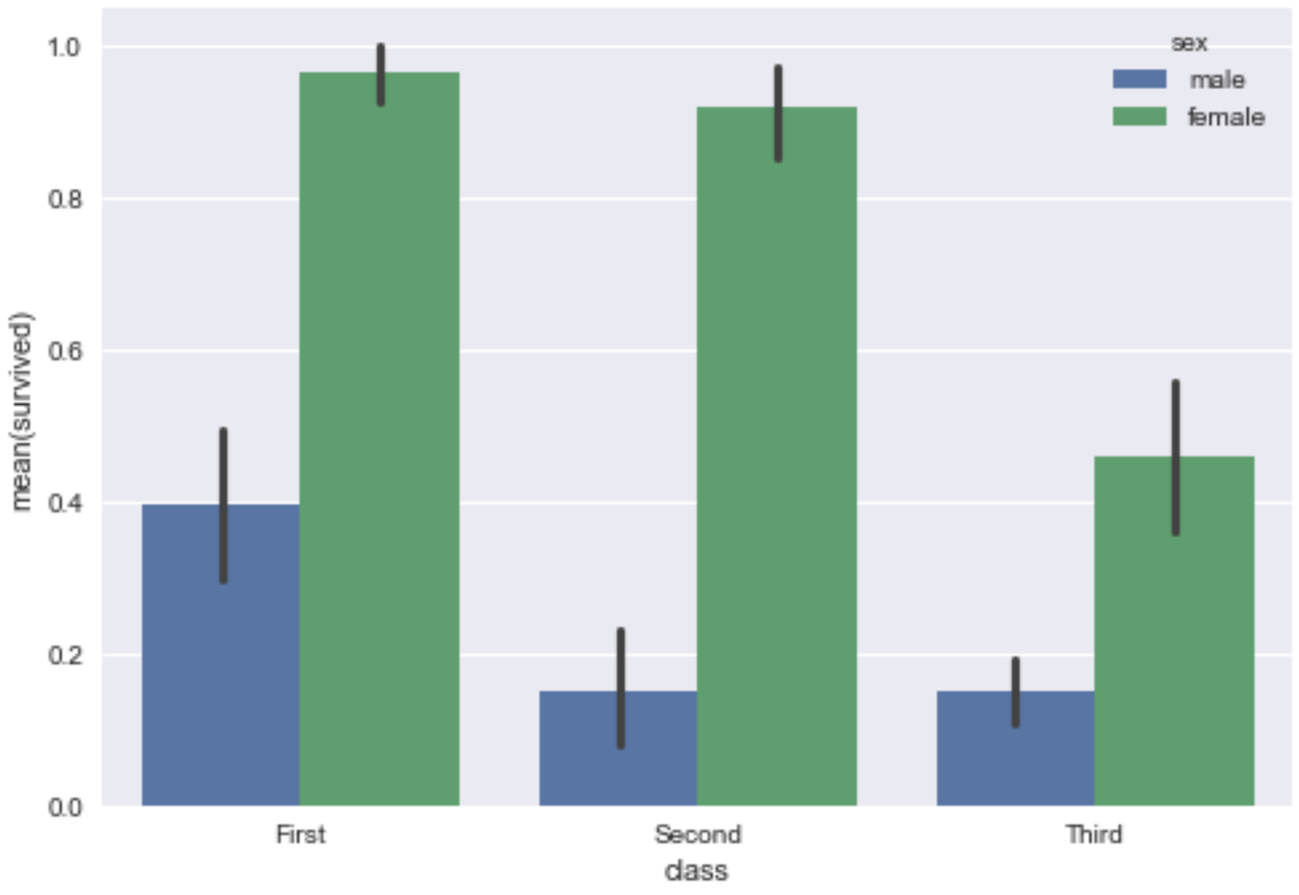


Notice how it was so easy to create the plot! You simply passed in the entire dataset, and just specified the `x` and `y` fields that you wanted exposed for the barplot. Behind the scenes seaborn ignored `NaN` values for you and automatically calculated the survival rate to plot. Also, that black tick 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 like `sex` used above, not something numerical like age or fare.

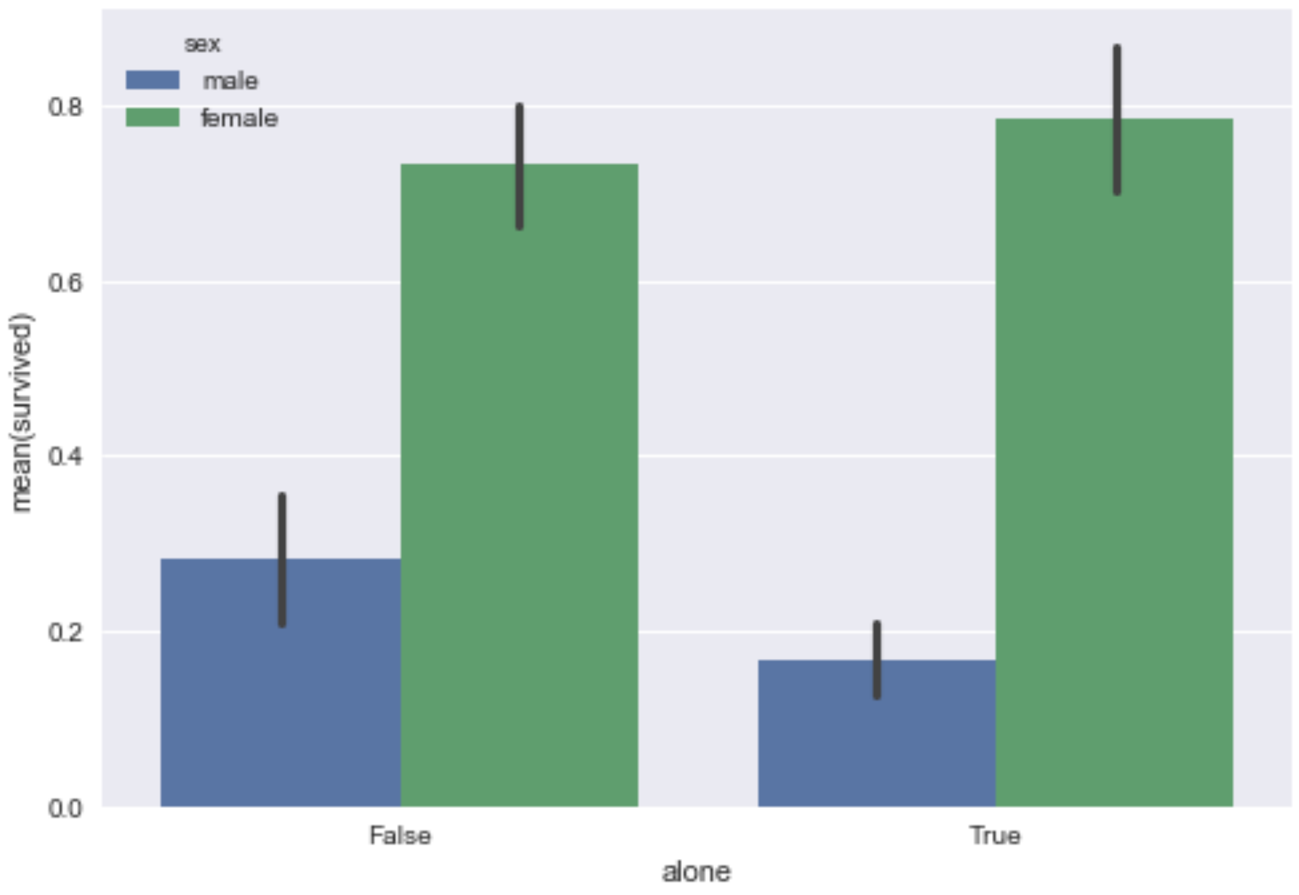
In [24]:

```
## YOUR CODE HERE
ax = sns.barplot(x="class", y="survived", hue="sex", data=titanic)
```



In [25]:

```
## YOUR CODE HERE
ax = sns.barplot(x="alone", y="survived",hue="sex", data=titanic)
```



What if we wanted to add a further sex breakdown for the categories chosen above? Go back and add a `hue` parameter for `sex` 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.

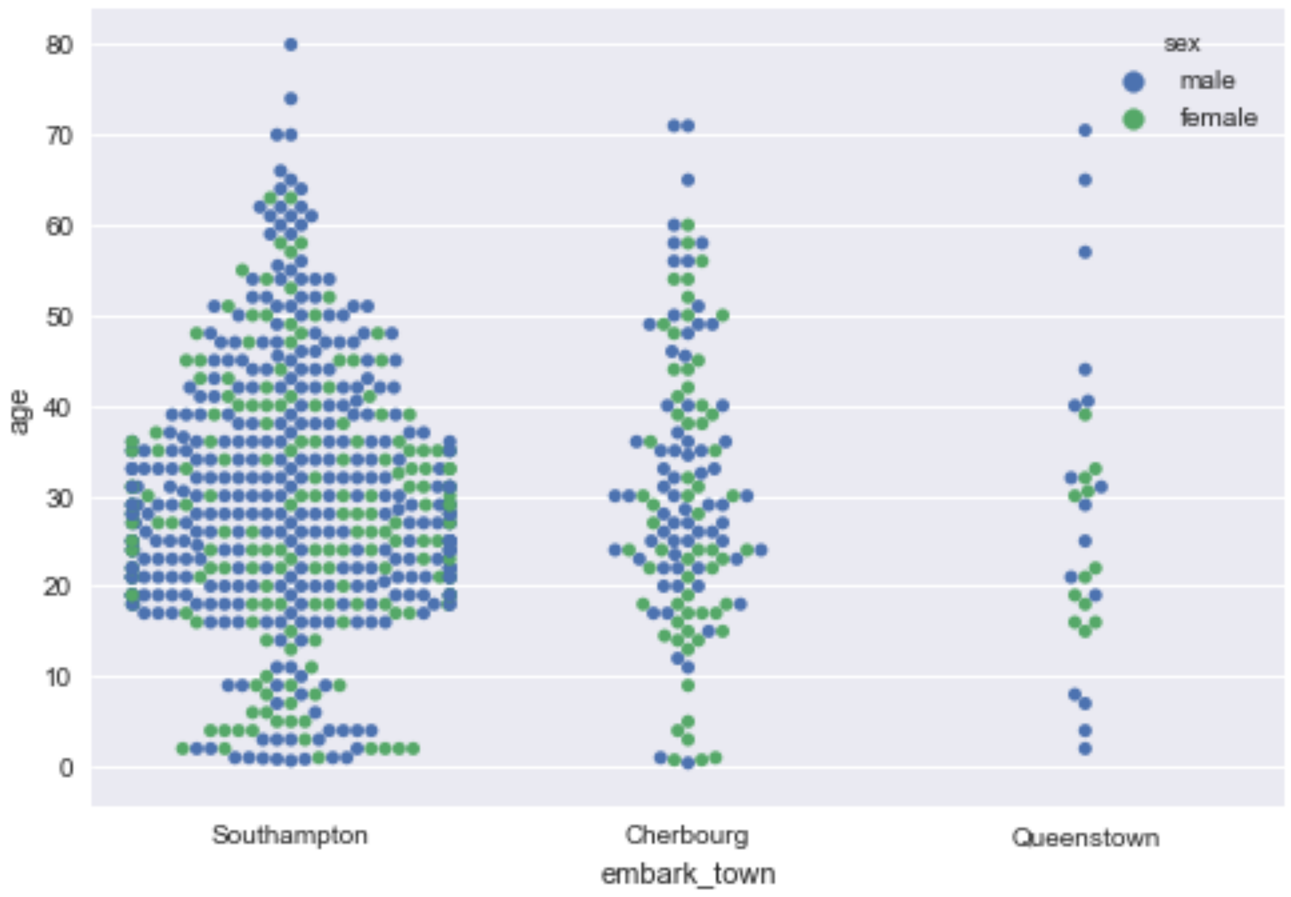
A good tool to help us here is `swarmplot` (https://seaborn.pydata.org/generated/seaborn.swarmplot.html). Use this function to view `embark_town` vs `age`, again using `sex` as the `hue`.

In [26]:

```
## YOUR CODE HERE
sns.swarmplot(x="embark_town", y="age", hue="sex", data=titanic)
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26295ab90>
```



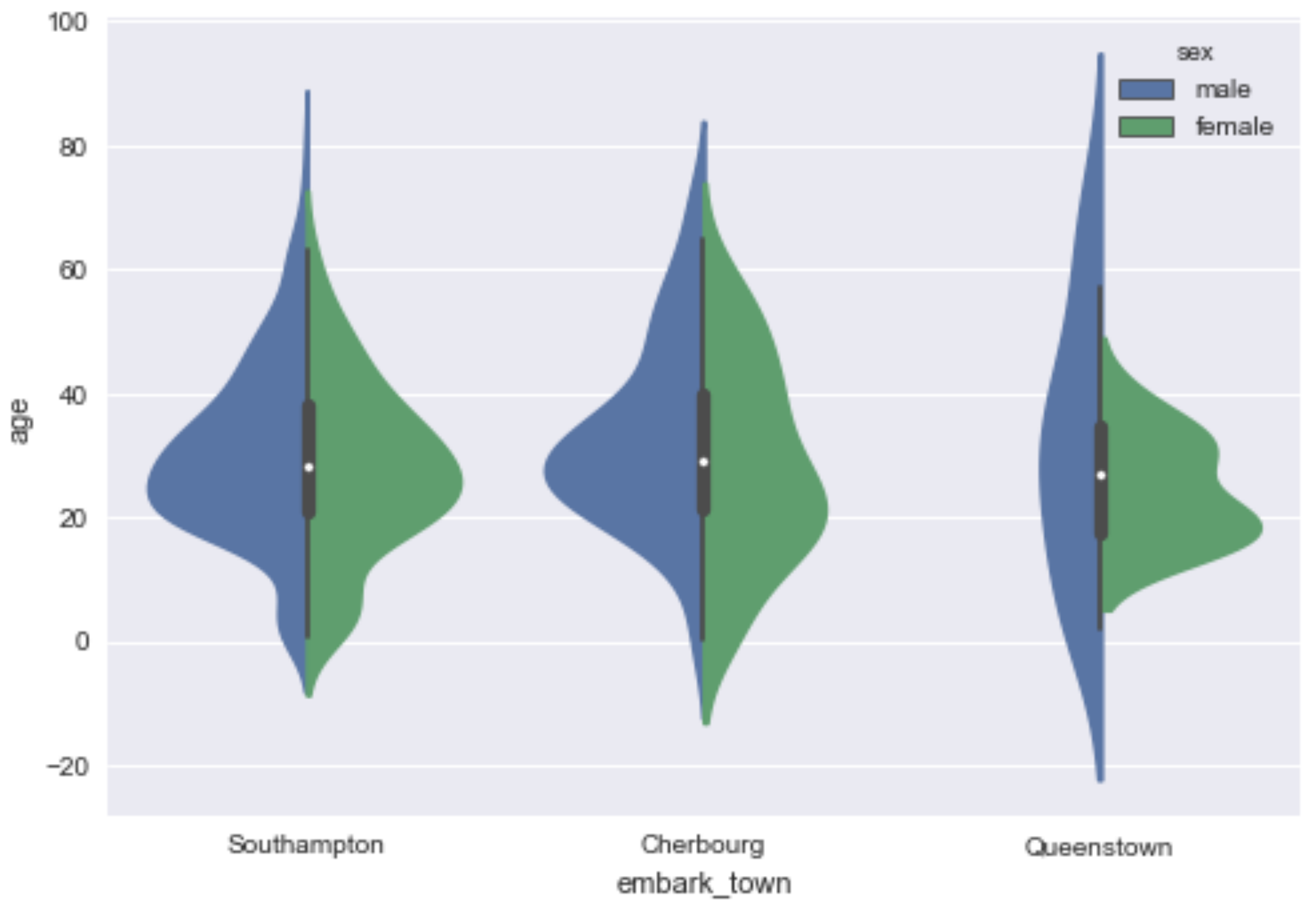
Cool! This gives us much more information. What if we didn't care about the number of individuals in each category at all, but rather just wanted to see the *distribution* in each category? `violinplot` (https://seaborn.pydata.org/generated/seaborn.violinplot.html) plots a density distribution. Plot that.

In [27]:

```
## YOUR CODE HERE
sns.violinplot(x="embark_town", y="age", hue="sex", split="True",data=titanic)
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x151a08290>
```



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.

- YOUR OBSERVATION HERE
- YOUR OBSERVATION HERE

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.

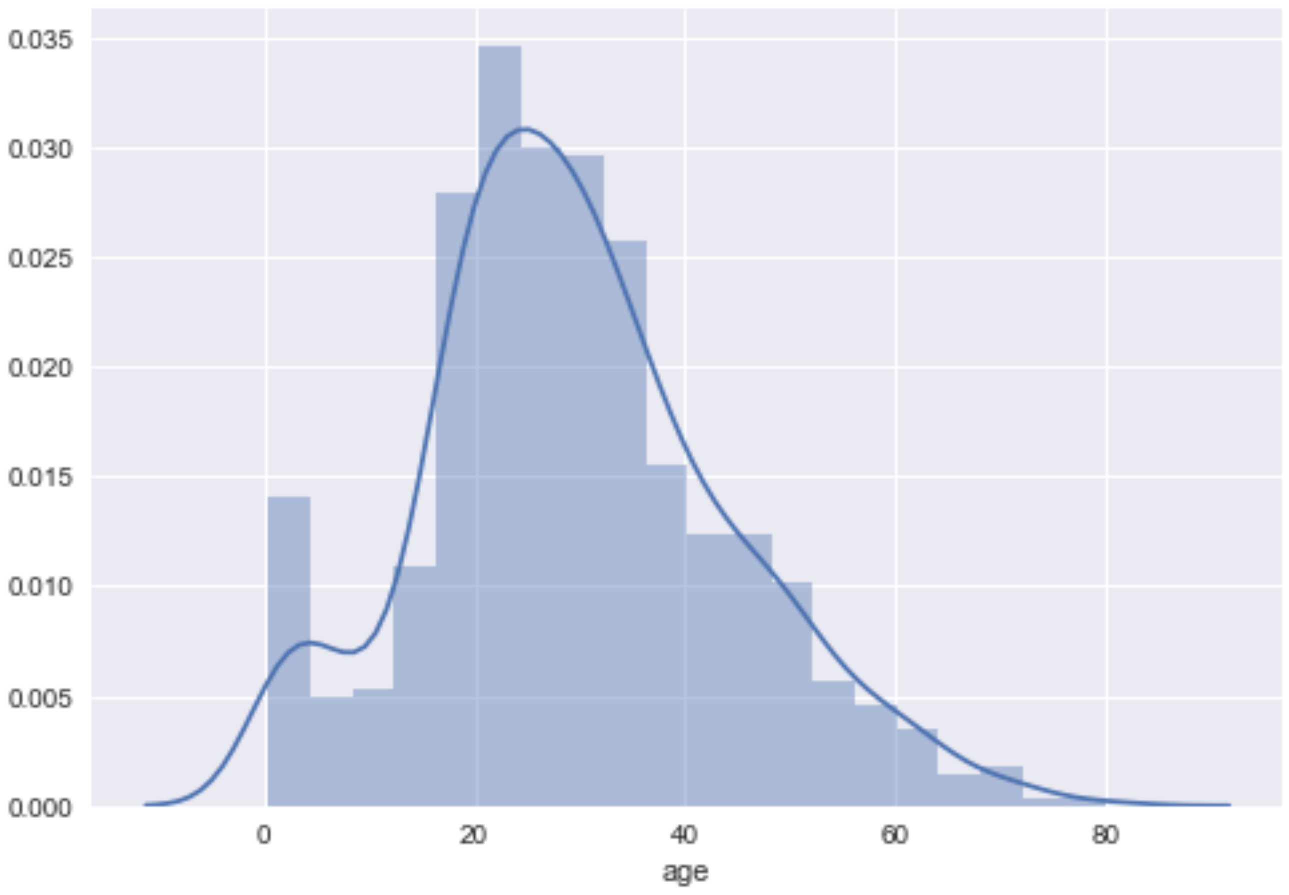
Let's look at the distribution of ages. Use `displot` (https://seaborn.pydata.org/generated/seaborn.distplot.html) to make a histogram of just the ages.

In [28]:

```
## YOUR CODE HERE
sns.distplot(a=titanic["age"])
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x151a28590>
```



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, seaborn also automatically plots a gaussian kernel density estimate.

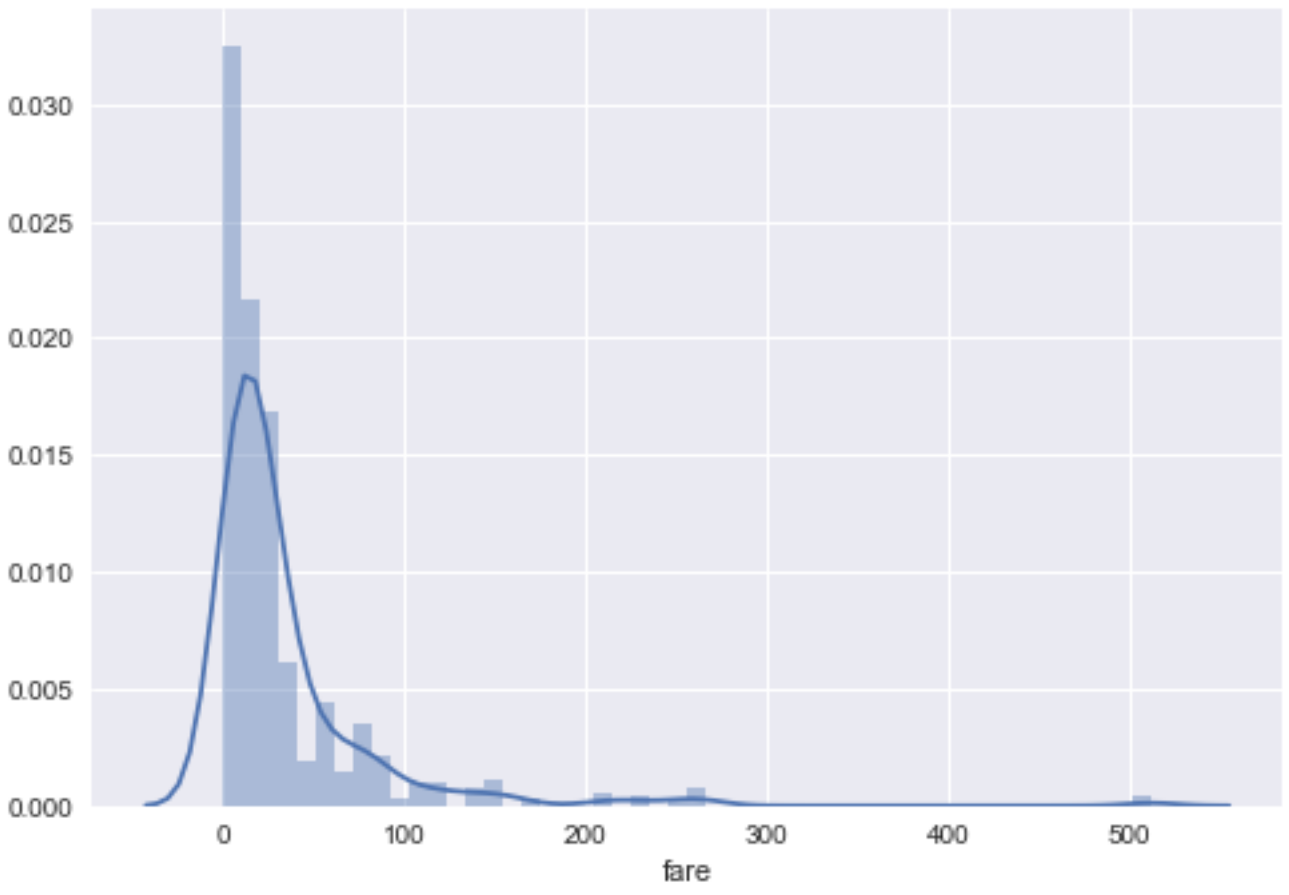
Do the same thing for fares - do you notice something odd about that histogram? What does that skew mean?

In [29]:

```
## YOUR CODE HERE
sns.distplot(a=titanic["fare"])
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x116741990>
```



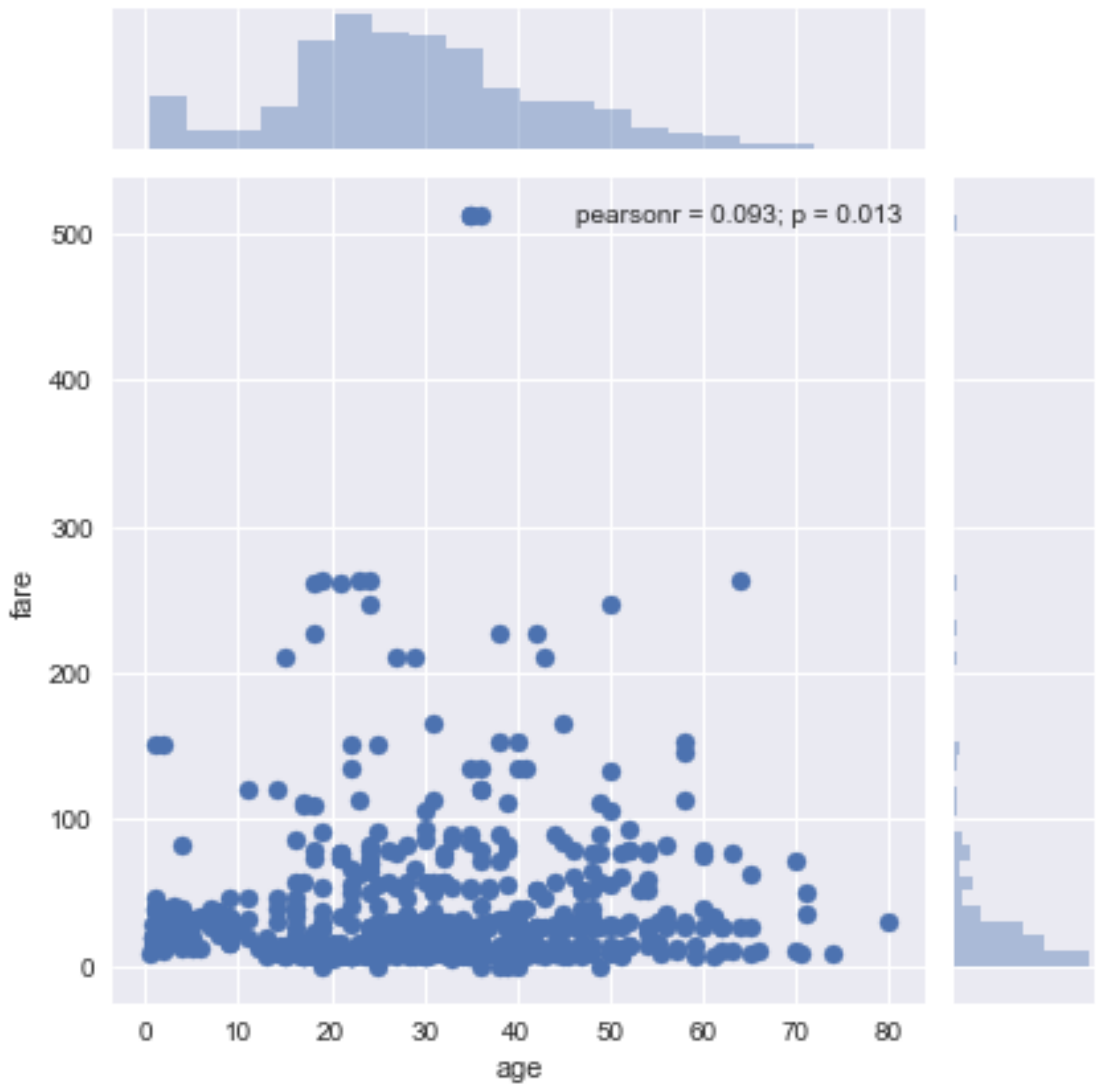
Now, using the `jointplot` (https://seaborn.pydata.org/generated/seaborn.jointplot.html#seaborn.jointplot) function, make a scatterplot of the `age` and `fare` variables to see if there is any relationship between the two.

In [30]:

```
## YOUR CODE HERE
sns.jointplot(x="age", y="fare", data=titanic)
```

Out[30]:

```
<seaborn.axisgrid.JointGrid at 0x1a1010550>
```



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.

## Diving into a familiar dataset

Now you hopefully have a pretty good understanding of both seaborn and matplotlib. You will now apply your learned skills to a familiar dataset, the 2016 election contributions. Navigate here (http://classic.fec.gov/disclosurep/PDownload.do) and download ALL.zip.

There will be no hand-holding in this section. You know how to import a dataset, pull out and clean the values you need, and then plot it. You will follow this whole pipeline yourself.

Please plot 2 graphs:

- the first graph should show the *cumulative* contributions for the candidate of your choide
- the second graph should be a histogram of the contributions (not cumulative), with a bin for each month

You may use whatever outside libraries you wish. The `tsplot` (http://seaborn.pydata.org/generated/seaborn.tsplot.html) and `distplot` (http://seaborn.pydata.org/generated/seaborn.distplot.html) from seaborn might be useful. The `hist` (https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html) from matplotlib and `cumsum` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.cumsum.html) from numpy may also be useful.

In [4]:

```
## YOUR CODE HERE
import datetime as dt
donations = pd.read_csv('donations.csv', index_col=False)
```

```
/usr/local/lib/python2.7/site-packages/IPython/core/interactiveshe
ll.py:2717: DtypeWarning: Columns (6,11,12,13) have mixed types. S
pecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [100]:

```
# drop columns with data not required
donations = donations[['cand_nm', 'contb_receipt_dt', 'contb_receipt_amt']]
# drop rows with NaN values
donations = donations.dropna()
```

In [101]:

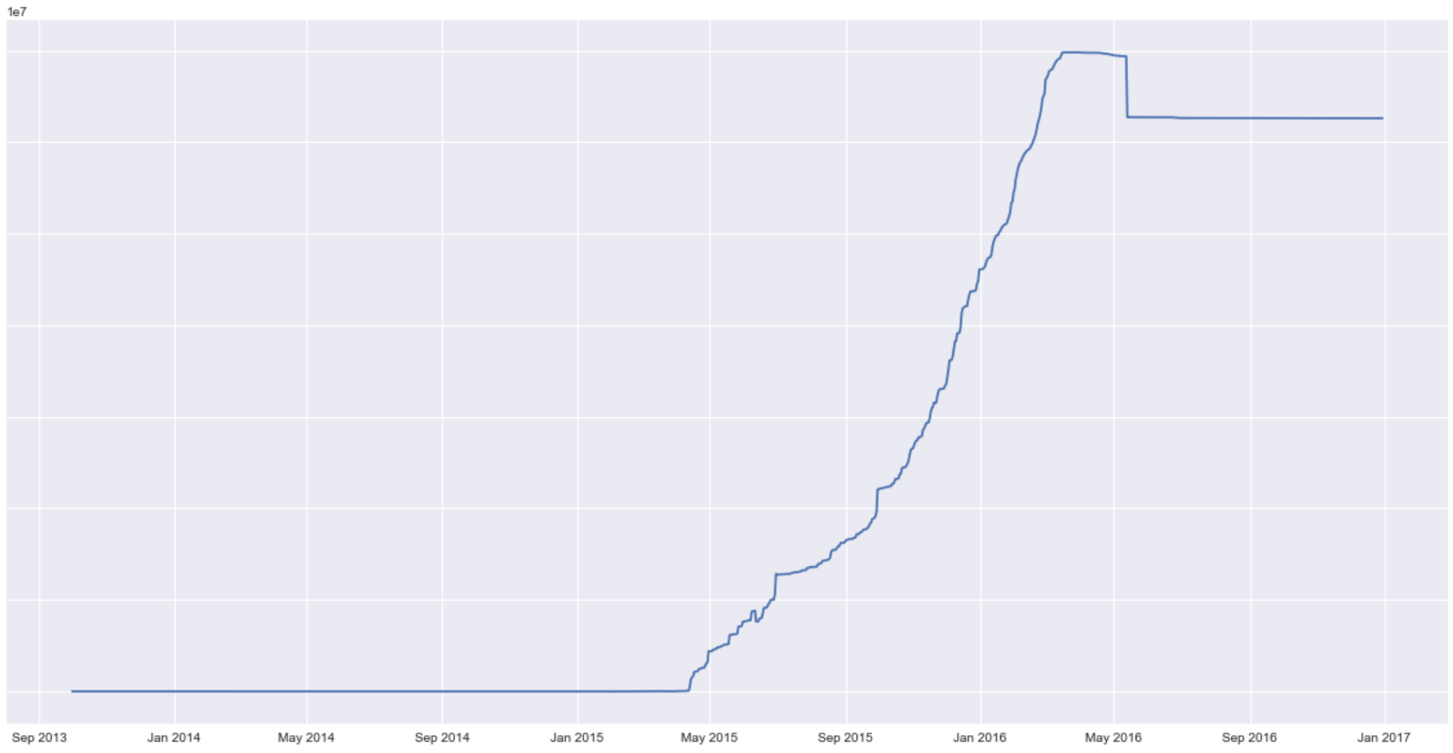
```
# candidate of choice is Marco Rubio
rubio = donations[donations.cand_nm == "Rubio, Marco"]
```

In [104]:

```
# plot the graph of cumulative donations over time
rubio_x,rubio_y = to_donation_xy_points(rubio)
plt.figure(figsize=(20,10))
plt.plot(rubio_x,np.cumsum(rubio_y))
```

Out[104]:

```
[<matplotlib.lines.Line2D at 0x166997910>]
```



In [103]:

```
def to_donation_xy_points(candidate):
    donations_dict = defaultdict(lambda:0)
    for index, row in candidate.iterrows():
        date = dt.datetime.strptime(row.contb_receipt_dt, '%d-%b-%y')
        donations_dict[date] += row.contb_receipt_amt
    sorted_by_date = sorted(donations_dict.items())
    return zip(*sorted_by_date)
```

In [75]:

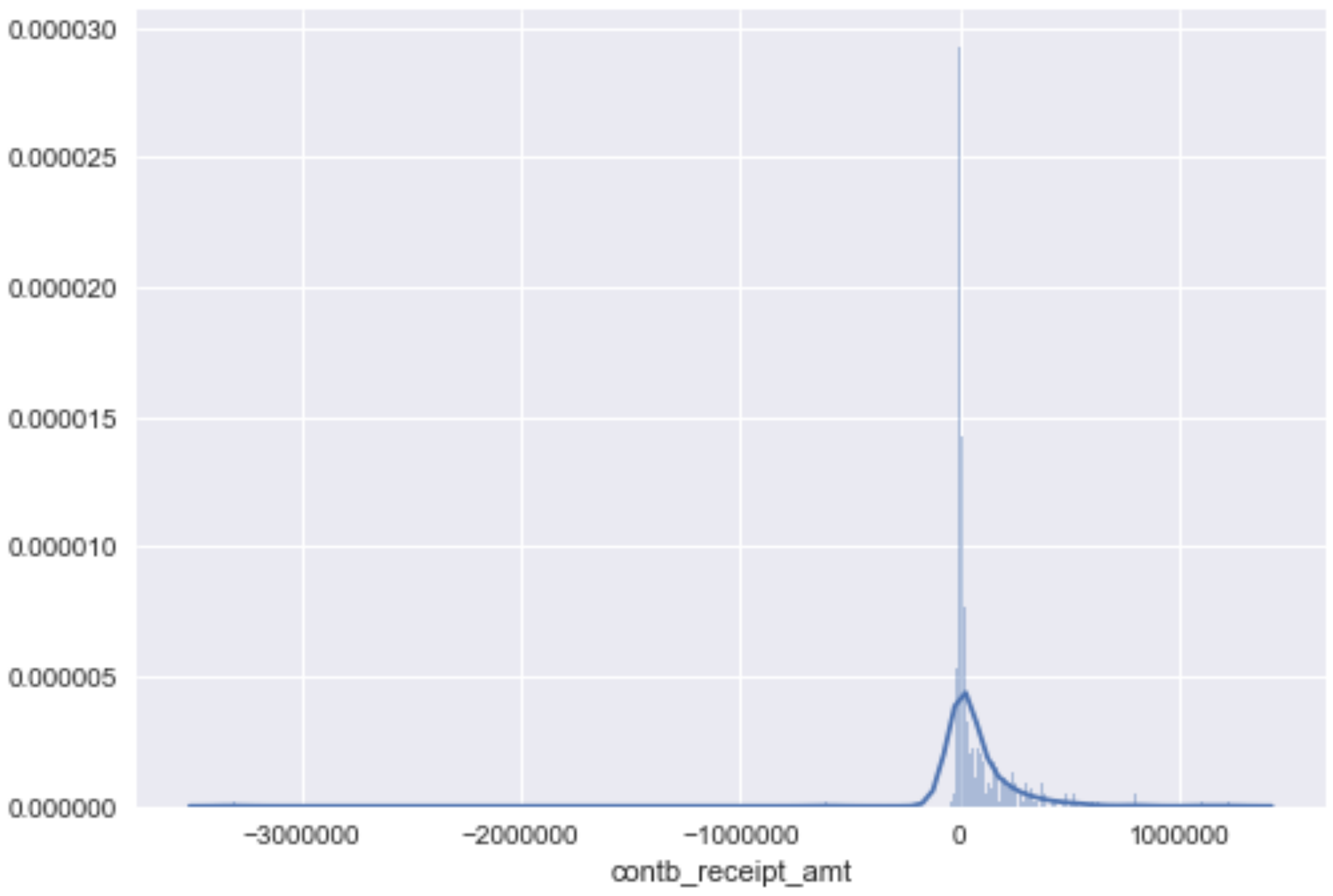
```
# histogram of the contributions (not cumulative), with a bin for each month
#hist_data = pd.Series(monthly_total,index=dates.unique())
#plt.hist(hist_data,bins=len(dates.unique()))
```

In [149]:

```
rubio.contb_receipt_dt = pd.to_datetime(rubio.contb_receipt_dt)
rubio = rubio.sort_values("contb_receipt_dt")
#hist_data = rubio.groupby((pd.Grouper(key='contb_receipt_dt', freq='M'))).sum
()
hist_data = rubio.groupby("contb_receipt_dt").sum()
#hist_data = hist_data.dropna()
sns.distplot(a=hist_data["contb_receipt_amt"],bins=len(hist_data))
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x164d7c610>
```



In [ ]:

```
```

In [ ]:

```
```