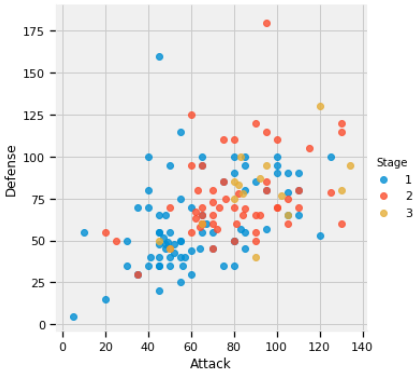
Using matplotlib's ylim() and xlim():

You were asked to create the following plot:



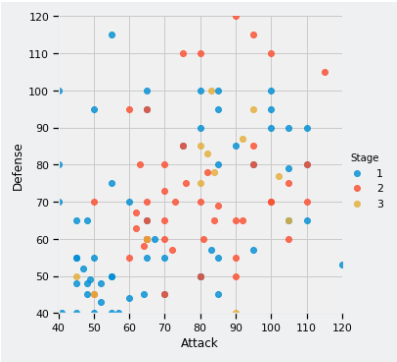

```
plot = sns.lmplot(x="Attack", y="Defense", data=poke, fit_reg = False, hue='Stage')
```

Then, you were asked to invoke Matplotlib's customization functions. Use the ylim() and xlim() functions.

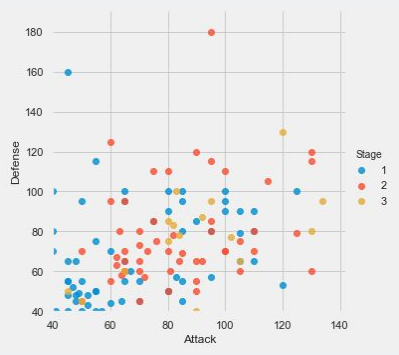
After the previous code, you can add limits. Think of this as "zooming" in. Call signatures:

```
left, right = xlim()   # return the current xlim
xlim((left, right))    # set the xlim to left, right
xlim(left, right)      # set the xlim to left, right
```

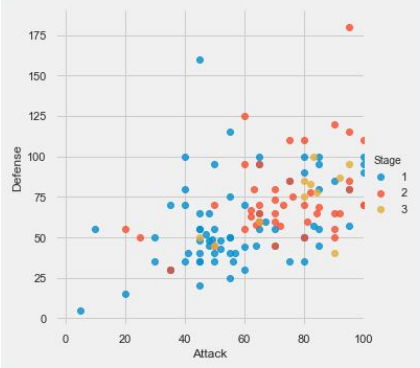
```
plt.ylim(40, 120)
plt.xlim(40, 120);
```



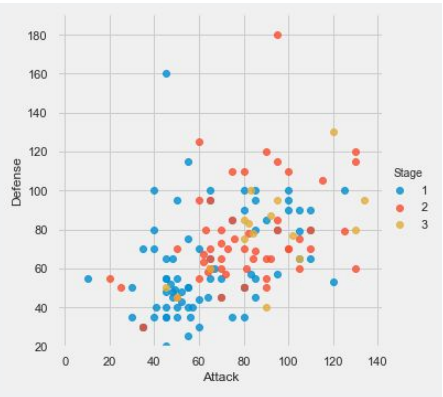

```
# Use None if you don't want to specify a min or max
plt.ylim(40, None)
plt.xlim(40, None);
```




```
# You can also use keywords "left" or "right" to specify only one or the other for xlim
plt.xlim(right = 100);
```




```
# Similarly, "top" or "bottom" for ylim
plt.ylim(bottom = 20);
```



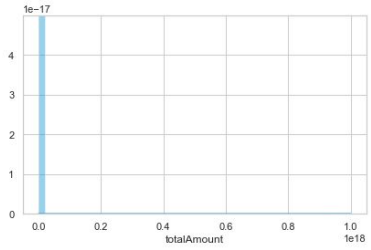
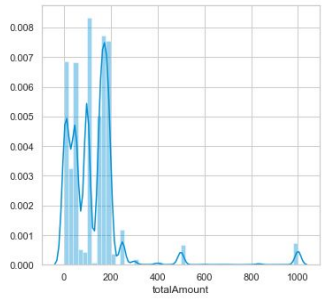
Using matplotlib's **ylim()** and **xlim()** or **.clip()** to zoom into an informative window:

In Part 4 Repetition, we asked **[Viz] Plot the 'totalAmount' column for transactions data. Using xlim, ylim, or the .clip() method, zoom into the window that provides the most informative visualization.**
Think of .clip() like ylim()/xlim(), but for a data frame rather than a plot:

```
df.clip(lower, upper)
```

You can then work with a smaller subset or "clip" of the dataframe (less computation).

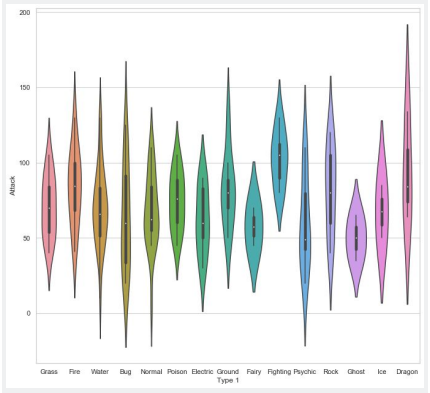
Your plots without clipping looked like this:	This is what they should look like, with solution code:

	`lower = 0`
	`upper = 1000`
	`capped_amounts = transactions_raw['totalAmount'].clip(lower = lower, upper = upper)`
	`sns.distplot(capped_amounts);`
	 
	By limiting outliers on the right, we gain more insight into where majority of the distribution lies.

Using seaborn's **palette =** :

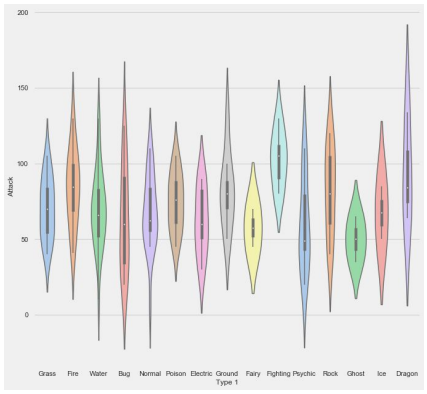
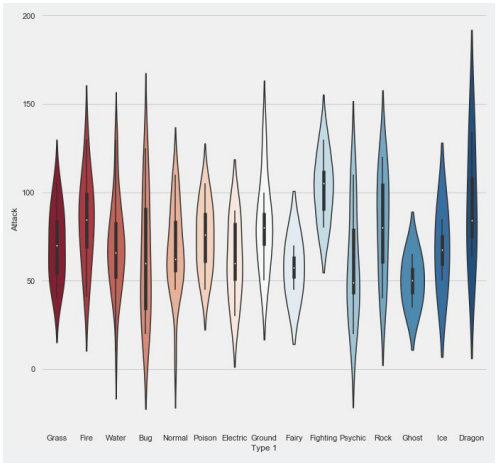
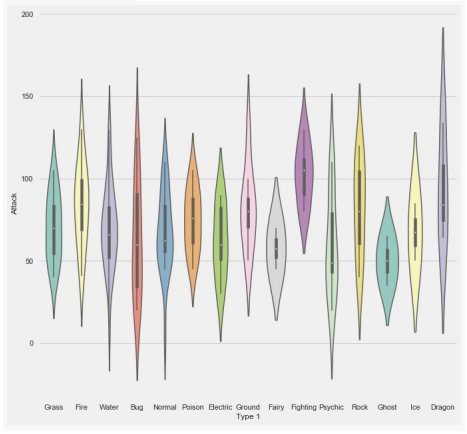
You were asked to create the following plot:

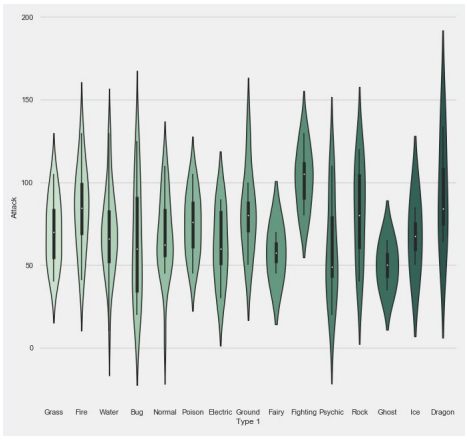
```
sns.violinplot(x='Type 1', y='Attack', data=poke)
```



Then, you were asked to use the palette = argument to recolor the above chart.
There are a few ways you can do this:

Use an available seaborn palette name: deep, muted, bright, pastel, dark, colorblind	`sns.violinplot(x='Type 1', y='Attack', data=poke, palette = "pastel");`

	
Use discrete values from one of the built-in matplotlib colormaps. When using discrete values, specify `n_colors=` appropriately.	`sns.violinplot(x='Type 1', y='Attack', data=poke, palette = "RdBu", n_colors=15);` 
Similarly, use a categorical matplotlib palette. You can change saturation with argument `desat=`	`sns.violinplot(x='Type 1', y='Attack', data=poke, palette = "Set3", n_colors=15, desat=1.5)` 
Make a customized cubehelix color palette:	`# "Why is everything chrome" tik tok` `sns.violinplot(x='Type 1', y='Attack', data=poke, palette = "ch:2.5,-.2,dark=.3")`



Displaying summary statistics for 2 dataframes:

The very straightforward way, with 2 cells:

For both data frames, find mean, median, etc - think summary statistics.

```
In [63]: 1 transactions_raw.describe()
```

Out[63]:

	amount	totalAmount	__v	metadata2
count	3.086500e+04	3.086500e+04	30865.0	0.0
mean	6.479832e+13	6.479832e+13	0.0	NaN
std	8.049610e+15	8.049610e+15	0.0	NaN
min	-1.000000e+00	-1.000000e+00	0.0	NaN
25%	5.000000e+01	5.000000e+01	0.0	NaN
50%	1.000000e+02	1.000000e+02	0.0	NaN
75%	1.750000e+02	1.760000e+02	0.0	NaN
max	1.000000e+18	1.000000e+18	0.0	NaN

```
In [64]: 1 users_raw.describe()
```

Out[64]:

	earningsThisYear	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	__v
count	22470.000000	22470.000000	2.247000e+04	22470.000000	22470.000000	22470.0
mean	180.149266	174.536983	4.450378e+09	0.088963	0.000846	0.0
std	560.380192	368.473235	6.671116e+11	0.829041	0.030560	0.0
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.0
25%	0.000000	36.000000	0.000000e+00	0.000000	0.000000	0.0
50%	155.000000	200.000000	1.530000e+02	0.000000	0.000000	0.0
75%	191.000000	200.000000	1.900000e+02	0.000000	0.000000	0.0
max	40189.000000	14210.000000	1.000000e+14	58.000000	2.000000	0.0

Pro Tip: if you use `display(display1, display2)`, you will get 2 outputs for one cell. I.e. `display(transactions_raw.describe(),users_raw.describe())`

```
#SOLUTION
display(transactions_raw.describe(),users_raw.describe())
```

	amount	totalAmount	__v	metadata2
count	3.086500e+04	3.086500e+04	30865.0	0.0
mean	6.479832e+13	6.479832e+13	0.0	NaN
std	8.049610e+15	8.049610e+15	0.0	NaN
min	-1.000000e+00	-1.000000e+00	0.0	NaN
25%	5.000000e+01	5.000000e+01	0.0	NaN
50%	1.000000e+02	1.000000e+02	0.0	NaN
75%	1.750000e+02	1.760000e+02	0.0	NaN
max	1.000000e+18	1.000000e+18	0.0	NaN

	earningsThisYear	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	__v
count	22470.000000	22470.000000	2.247000e+04	22470.000000	22470.000000	22470.0
mean	180.149266	174.536983	4.450378e+09	0.088963	0.000846	0.0
std	560.380192	368.473235	6.671116e+11	0.829041	0.030560	0.0
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.0
25%	0.000000	36.000000	0.000000e+00	0.000000	0.000000	0.0
50%	155.000000	200.000000	1.530000e+02	0.000000	0.000000	0.0
75%	191.000000	200.000000	1.900000e+02	0.000000	0.000000	0.0
max	40189.000000	14210.000000	1.000000e+14	58.000000	2.000000	0.0

Using pandas `.shape`:

You were asked to **Find the # of transactions where the totalAmount is greater than 600.**
Your code:

```
arr = transactions_raw["totalAmount"] > 600
count = 0
for i in arr:
    if i:
        count+=1
count
```

```
501
```

This does work, however it is not the most computationally efficient method, nor the cleanest code.

A better approach:

```
# Filter the dataset for only transaction amounts >600
temp = transactions_raw[transactions_raw['totalAmount']>600]
#Take the .shape of this subsetted data:
temp.shape
#Shape returns a tuple with the dimensions of a dataframe, allowing us to index in.
# shape[0] gives us the # rows, shape[1] the number of columns.
temp.shape[0]
```

Remember to **always look for a built in function** before writing new code - no need to reinvent the wheel!

Displaying when using pandas `groupby()` and `sort_values()`:

You were asked to **find the # of users/location, sorted in descending order.**

This is the output you provided

```
users_raw.groupby("location").count()["firstName"].sort_values(ascending = False)
```

```
location
San Jose                                4135
Omaha                                   2211
Chennai                                 1927
Chandler                                1576
Dublin                                  1322
Bangalore                               1322
Shanghai                                1272
Dundalk                                 1200
Chicago                                  699
Scottsdale                               542
Singapore                                531
San Francisco                            450
Timonium                                 448
Kuala Lumpur                             443
Hunt Valley                              437
Austin                                   399
New York                                 390
Berlin                                   354
Manila                                   282
London                                   248
Tel Aviv                                 240
Conshohocken                             194
Dreilinden                               191
Sao Paulo                                184
Guatemala                                179
Newton - 275                             134
Sydney                                   125
Wilmington                               117
Luxembourg                               116
Paris                                     94
                                        ...
Vancouver                                  7
3PP - Manila - ePerformax                  6
Melbourne - Braintree                      6
TSA_Location                               5
Gurgaon                                    5
Vancouver - Virtual Way                    4
Istanbul - 2                               4
Phoenix Data Center                        4
New Delhi                                  3
Bangkok                                    3
Seoul                                      3
Silver Spring - Tio                        3
Beijing                                    3
3PP - Manila - Sutherland                  2
Brussels - St Michel                       2
3PP - Beijing - Symbio                     2
3PP - Chennai - Syntel                     2
Denver                                     2
3PP - Pennsylvania - RMS                   2
3PP - San Jose - Convergys                 2
Geneva                                     2
3PP - Moscow - Usethics                    1
Boston                                     1
3PP - San Jose - Concentrix                1
3PP - Magdeburg                            1
3PP - Hyderabad - IBM                      1
3PP - Denver - EOS                         1
3PP - Cleveland - RightSource              1
3PP - Berlin - Arvato                      1
Remote                                     1
Name: firstName, Length: 79, dtype: int64
```

For a more visually appealing/easier to read display, you can do the following:

```
users_per_loc = pd.DataFrame(users_raw.groupby("location").count().iloc[:, 0]).rename({"_id":'Users'}, axis=1)
users_per_loc_desc = users_per_loc.sort_values("Users", ascending=False)
users_per_loc_desc.head(10)
```

location	Users
San Jose	4135
Omaha	2211
Chennai	1927
Chandler	1576
Dublin	1322
Bangalore	1322
Shanghai	1272
Dundalk	1200
Chicago	699
Scottsdale	542

This is generally better practice, particularly for notebooks you will send to clients/managers.

Let's break down the code.

```
#Group by the count, by location
users_raw.groupby("location").count()
```

location	_id	earningsThisYear	hidePhoto	level	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	status	firstName	lastName
3PP - Bangalore - Genpact	74	74	74	74	74	74	74	74	74	74	74
3PP - Beijing - Symbio	2	2	2	2	2	2	2	2	2	2	2
3PP - Berlin - Arvato	1	1	1	1	1	1	1	1	1	1	1
3PP - Chennai - Syntel	2	2	2	2	2	2	2	2	2	2	2
3PP - Cleveland - RightSource	1	1	1	1	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
Vancouver - Howe Street	19	19	19	19	19	19	19	19	19	19	19
Vancouver - Virtual Way	4	4	4	4	4	4	4	4	4	4	4
Warsaw	14	14	14	14	14	14	14	14	14	14	14
Washington DC	28	28	28	28	28	28	28	28	28	28	28
Wilmington	117	117	117	117	117	117	117	117	117	117	117

79 rows × 16 columns

```
#Subset only the first column, _id, as we are only interested in the counts/location
users_raw.groupby("location").count().iloc[:,0]
```

```
location
3PP - Bangalore - Genpact          74
3PP - Beijing - Symbio              2
3PP - Berlin - Arvato               1
3PP - Chennai - Syntel              2
3PP - Cleveland - RightSource       1
                                  ...
Vancouver - Howe Street            19
Vancouver - Virtual Way             4
Warsaw                             14
Washington DC                      28
Wilmington                        117
Name: _id, Length: 79, dtype: int64
```

```
#Convert from series.Series object into a dataframe (we subsetted only 1 column, which outputs a series)
pd.DataFrame(users_raw.groupby("location").count().iloc[:,0])
```

location	_id
3PP - Bangalore - Genpact	74
3PP - Beijing - Symbio	2
3PP - Berlin - Arvato	1
3PP - Chennai - Syntel	2
3PP - Cleveland - RightSource	1
...	...
Vancouver - Howe Street	19
Vancouver - Virtual Way	4
Warsaw	14
Washington DC	28
Wilmington	117

79 rows × 1 columns

```
#Convert from groupby object into a dataframe
users_per_loc =
pd.DataFrame(users_raw.groupby("location").count().iloc[:,0]).rename({"_id":'Users'}, axis=1)
```

location	Users
3PP - Bangalore - Genpact	74
3PP - Beijing - Symbio	2
3PP - Berlin - Arvato	1
3PP - Chennai - Syntel	2
3PP - Cleveland - RightSource	1
...	...
Vancouver - Howe Street	19
Vancouver - Virtual Way	4
Warsaw	14
Washington DC	28
Wilmington	117

79 rows × 1 columns

```
#Sort descending
users_per_loc_desc = users_per_loc.sort_values("Users", ascending=False)
#Show just the top 10, just to limit less useful output
users_per_loc_desc.head(10)
```

location	Users
San Jose	4135
Omaha	2211
Chennai	1927
Chandler	1576
Dublin	1322
Bangalore	1322
Shanghai	1272
Dundalk	1200
Chicago	699
Scottsdale	542

Now, the one everyone struggled with the most - I received blank cells or a few lines of attempt, but no successful visualizations!

You were asked to: **[Viz] Plot the # of users/location, sorted in descending order for the largest 30 locations.**

This question is based on the previous, so let's break down the code.

```
users_per_loc_desc #Our previous df
#We reset_index. We had MultiIndex before, and in order to plot, we need the
'Users' label to be at the same index level as 'location'.
temp = users_per_loc_desc.reset_index().head(30)
```

Before resetting the index:

location	Users
San Jose	4135
Omaha	2211
Chennai	1927
Chandler	1576
Dublin	1322
Bangalore	1322
Shanghai	1272
Dundalk	1200
Chicago	699
Scottsdale	542

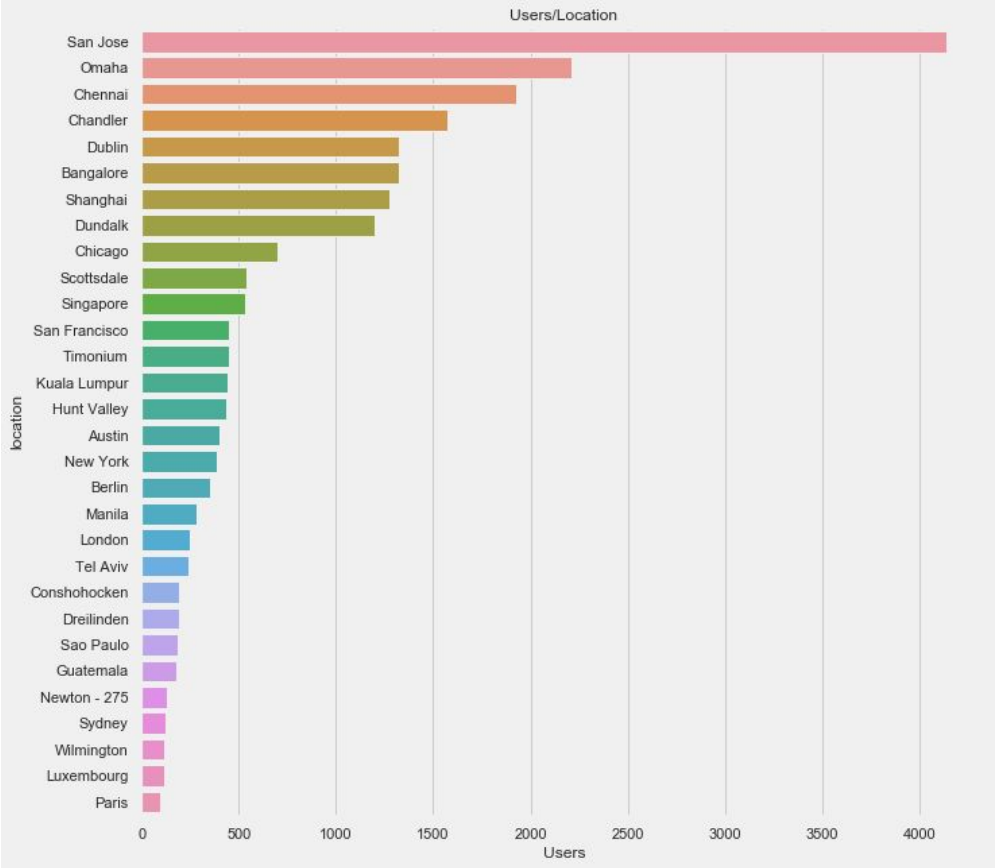
After:

	location	Users
0	San Jose	4135
1	Omaha	2211
2	Chennai	1927
3	Chandler	1576
4	Dublin	1322
5	Bangalore	1322
6	Shanghai	1272
7	Dundalk	1200
8	Chicago	699
9	Scottsdale	542
10	Singapore	531

We reset the index so that we can use the `location` column in the code below.

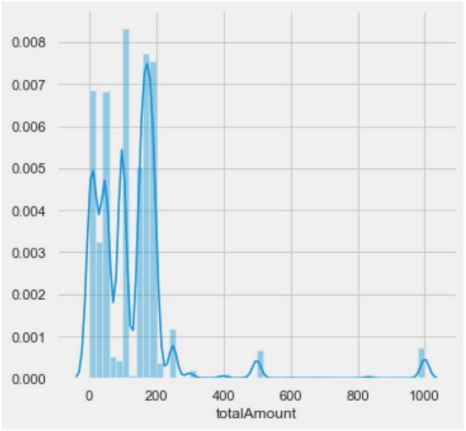
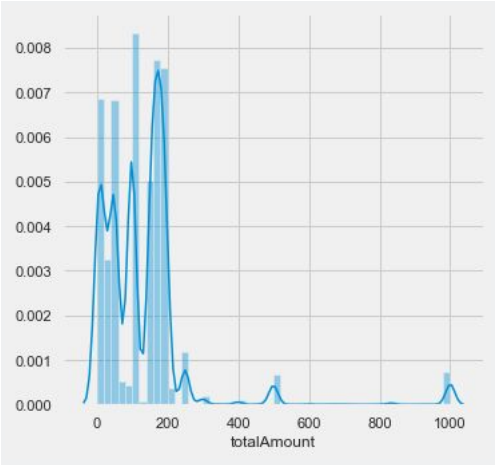
Now, it is just a matter of throwing our data into seaborn barplot:

```
plt.title("Users/Location")
sns.barplot(y=temp['location'], x=temp['Users']);
```



Other notes and tips:

ITEM 1: You may have noticed for plots, I add a `;` after the last line when plotting. This is to limit unnecessary output, or to make it non-verbose. Generally, just put `;` on the last line of plotting code.

Without ;	**With `;`**
`<matplotlib.axes._subplots.AxesSubplot at 0x1a257d5e10>` 	

ITEM 2: On the last question, **Find the last (most recent) transaction from 'god'**, everyone provided this output and code

```
1 transactions_raw[transactions_raw['from'].str.contains('god')].iloc[23132]
```

```
_id                                  5d8d1aa0fc3ea75696ab88d3
to                                                    shleong
type                                                  airdrop
subtype                                               welcome
status                                               complete
amount                                                    174
from                                                      god
totalAmount                                               174
transactionHash                                           NaN
timeCreated          Thu Sep 26 2019 13:08:00 GMT-0700 (PDT)
timeUpdated          Thu Sep 26 2019 13:08:00 GMT-0700 (PDT)
__v                                                         0
newsfeedId                           5d8d1aa0fc3ea75696ab88d3
note                                                      NaN
metadata                                                  NaN
metadata2                                                 NaN
Name: 30864, dtype: object
```

While this is the right answer, there are a few issues with the code:
`transactions_raw[transactions_raw['from'].str.contains('god')].iloc[23132]`
What you have done here is 1) Manually find the most recent time 2) hardcoding, where you put in a fixed parameter (the only way to change the number is to manually go in and change the code).

1) Always go a coding approach as opposed to a manual search approach. You should sort_descending as follows
    a) `god_sorted = pd.to_datetime(god["timeCreated"]).sort_values(ascending=False)`
2) Then, index in to grab the first value:
    a) `god_sorted.reset_index().iloc[0]`

```
index                                    30864
timeCreated    2019-09-26 13:08:00-07:00
Name: 0, dtype: object
```

Same answer, more reproducible approach - if the dataset changed, you would still be outputting the row corresponding to the most recent time, meaning your code could be easily re-used.

ITEM 3: Groupby objects vs dataframes
Groupby objects are special objects in Pandas that are a result of applying the `.groupby()` function.

However, if you've used the datascience module from Data 8, you'll know that the analogous .group() function returns a table like this: (something from data8 textbook)

Groupby objects in pandas just return a groupby object:

```
users_raw.groupby("location")
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x1a257ddd90>
```

```
type(users_raw.groupby("location"))
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

To actually get useful results from groupby, we need to apply some kind of aggregator to this groupby object to get a DataFrame. Some common aggregators are .sum(), .count(), and .max()/.min().

```
users_raw.groupby("location").count()
```

location	_id	earningsThisYear	hidePhoto	level	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	status	firstName	lastName	c
3PP - Bangalore - Genpact	74	74	74	74	74	74	74	74	74	74	74	
3PP - Beijing - Symbio	2	2	2	2	2	2	2	2	2	2	2	
3PP - Berlin - Arvato	1	1	1	1	1	1	1	1	1	1	1	
3PP - Chennai - Syntel	2	2	2	2	2	2	2	2	2	2	2	
3PP - Cleveland - RightSource	1	1	1	1	1	1	1	1	1	1	1	

```
type(users_raw.groupby("location").count())
```

```
pandas.core.frame.DataFrame
```

Note: The above aggregators (aside from count) mention **only apply to numerical variables** → you can't logically take the sum, min, or max of non-numerical variables like strings.

You can also group by **multiple columns** and similarly apply relevant aggregators to your groupby object:

```
users_raw.groupby(["location","earningsThisYear"]).count()
```

location	earningsThisYear	_id	hidePhoto	level	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	status	firstName	lastName	qi
3PP - Bangalore - Genpact	151	2	2	2	2	2	2	2	2	2	2	
	153	1	1	1	1	1	1	1	1	1	1	
	154	2	2	2	2	2	2	2	2	2	2	
	155	2	2	2	2	2	2	2	2	2	2	
	160	2	2	2	2	2	2	2	2	2	2	
...	...	...	...	...	...	...	...	...	...	...	...	...
Wilmington	341	1	1	1	1	1	1	1	1	1	1	
	350	1	1	1	1	1	1	1	1	1	1	
	405	1	1	1	1	1	1	1	1	1	1	
	496	1	1	1	1	1	1	1	1	1	1	
	500	1	1	1	1	1	1	1	1	1	1	

4508 rows × 15 columns

This leads to a MultiLevelIndex in your DataFrame, which really just means you have multiple indices. If for some reason you want to access your indices as columns, you can simply use `.reset_index()`.

```
users_raw.groupby(["location","earningsThisYear"]).count().reset_index()
```

	location	earningsThisYear	_id	hidePhoto	level	pointToNextLevel	currentBalance	totalIdeasSubmitted	totalExpRedeemed	status	firstName	lastNan
0	3PP - Bangalore - Genpact	151	2	2	2	2	2	2	2	2	2	
1	3PP - Bangalore - Genpact	153	1	1	1	1	1	1	1	1	1	
2	3PP - Bangalore - Genpact	154	2	2	2	2	2	2	2	2	2	
3	3PP - Bangalore - Genpact	155	2	2	2	2	2	2	2	2	2	
4	3PP - Bangalore - Genpact	160	2	2	2	2	2	2	2	2	2	
...	...	...	...	...	...	...	...	...	...	...	...	
4503	Wilmington	341	1	1	1	1	1	1	1	1	1	
4504	Wilmington	350	1	1	1	1	1	1	1	1	1	
4505	Wilmington	405	1	1	1	1	1	1	1	1	1	
4506	Wilmington	496	1	1	1	1	1	1	1	1	1	
4507	Wilmington	500	1	1	1	1	1	1	1	1	1	

4508 rows × 17 columns

As always, since the resulting objects are DataFrames, you can access and manipulate columns (Series) as usual.

For output types, we also see `series.Series` We saw this earlier to **find the # of users/location, sorted in descending order.** Basically, when in doubt, take your output, and put it in `pd.DataFrame(output)`.