Practice Exercises 4: Functions and Visualizations

Welcome to Practice Exercises 4! This week, we'll learn about functions, table methods such as `apply`, and how to generate visualizations!

Recommended Reading:

- Applying a Function to a Column (https://www.inferentialthinking.com/chapters/08/1/applying-a-function-to-a-column.html)
- Visualizations (https://www.inferentialthinking.com/chapters/07/visualization.html)

Remember, practice exercises are *not* required and will not be turned in for credit...but they are helpful for developing your YData skills!

Credit: These practice exercises have been adapted from Berkeley's Data8 course.

Let's begin by running the cell below.

In [1]:
```
import numpy as np
from datascience import *

# These lines set up graphing capabilities.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

1. Functions and CEO Incomes

In this question, we'll look at the 2015 compensation of CEOs at the 100 largest companies in California. The data was compiled from a Los Angeles Times analysis (http://spreadsheets.latimes.com/california-ceo-compensation/), and ultimately came from filings (https://www.sec.gov/answers/proxyhtf.htm) mandated by the SEC from all publicly-traded companies. Two companies have two CEOs, so there are 102 CEOs in the dataset.

We've copied the raw data from the LA Times page into a file called `raw_compensation.csv`. (The page notes that all dollar amounts are in millions of dollars.)

In [2]:
```
raw_compensation = Table.read_table('raw_compensation.csv')
raw_compensation
```

Out[2]:

Rank	Name	Company (Headquarters)	Total Pay	% Change	Cash Pay	Equity Pay	Other Pay	Ratio of CEO pay to average industry worker pay
1	Mark V. Hurd*	Oracle (Redwood City)	$53.25	(No previous year)	$0.95	$52.27	$0.02	362
2	Safra A. Catz*	Oracle (Redwood City)	$53.24	(No previous year)	$0.95	$52.27	$0.02	362
3	Robert A. Iger	Walt Disney (Burbank)	$44.91	-3%	$24.89	$17.28	$2.74	477
4	Marissa A. Mayer	Yahoo! (Sunnyvale)	$35.98	-15%	$1.00	$34.43	$0.55	342
5	Marc Benioff	salesforce.com (San Francisco)	$33.36	-16%	$4.65	$27.26	$1.45	338
6	John H. Hammergren	McKesson (San Francisco)	$24.84	-4%	$12.10	$12.37	$0.37	222
7	John S. Watson	Chevron (San Ramon)	$22.04	-15%	$4.31	$14.68	$3.05	183
8	Jeffrey Weiner	LinkedIn (Mountain View)	$19.86	27%	$2.47	$17.26	$0.13	182
9	John T. Chambers**	Cisco Systems (San Jose)	$19.62	19%	$5.10	$14.51	$0.01	170
10	John G. Stumpf	Wells Fargo (San Francisco)	$19.32	-10%	$6.80	$12.50	$0.02	256

... (92 rows omitted)

Question 1. We want to compute the average of the CEOs' pay. Try running the cell below.

```
In [3]: np.average(raw_compensation.column("Total Pay"))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-f97fab5a8083> in <module>
----> 1 np.average(raw_compensation.column("Total Pay"))

/usr/local/lib/python3.7/site-packages/numpy/lib/function_base.py in average(a, axis, weights, returned)
    354 
    355     if weights is None:
--> 356         avg = a.mean(axis)
    357         scl = avg.dtype.type(a.size/avg.size)
    358     else:

/usr/local/lib/python3.7/site-packages/numpy/core/_methods.py in _mean(a, axis, dtype, out, keepdims)
     73             is_float16_result = True
     74 
---> 75     ret = umr_sum(arr, axis, dtype, out, keepdims)
     76     if isinstance(ret, mu.ndarray):
     77         ret = um.true_divide(

TypeError: cannot perform reduce with flexible type
```

You should see an error. Let's examine why this error occurred by looking at the values in the "Total Pay" column. Use the `type` function and set `total_pay_type` to the type of the first value in the "Total Pay" column.

```
In [ ]: total_pay_type = ...
        total_pay_type
```

```
In [4]: #Answer
        total_pay_type = type(raw_compensation.column("Total Pay").item(0))
        total_pay_type

Out[4]: str
```

Question 2. You should have found that the values in "Total Pay" column are strings. It doesn't make sense to take the average of string values, so we need to convert them to numbers if we want to do this. Extract the first value in the "Total Pay" column. It's Mark Hurd's pay in 2015, in *millions* of dollars. Call it `mark_hurd_pay_string`.

```
In [ ]: mark_hurd_pay_string = ...
        mark_hurd_pay_string
```

```
In [5]: #Answer
        mark_hurd_pay_string = raw_compensation.column("Total Pay").item(0)
        mark_hurd_pay_string
```

```
Out[5]: '$53.25 '
```

Question 3. Convert `mark_hurd_pay_string` to a number of *dollars*. The string method `strip` will be useful for removing the dollar sign; it removes a specified character from the start or end of a string. For example, the value of `"100%".strip("%")` is the string `"100"`. You'll also need the function `float`, which converts a string that looks like a number to an actual number. Last, remember that the answer should be in dollars, not millions of dollars.

```
In [ ]: mark_hurd_pay = ...
        mark_hurd_pay
```

```
In [6]: #Answer
        mark_hurd_pay = float(mark_hurd_pay_string.strip("$"))
        mark_hurd_pay
```

```
Out[6]: 53.25
```

To compute the average pay, we need to do this for every CEO. But that looks like it would involve copying this code 102 times.

This is where functions come in. First, we'll define a new function, giving a name to the expression that converts "total pay" strings to numeric values. Later in these exercises we'll see the payoff: we can call that function on every pay string in the dataset at once.

Question 4. Copy the expression you used to compute `mark_hurd_pay` as the `return` expression of the function below, but replace the specific `mark_hurd_pay_string` with the generic `pay_string` name specified in the first line of the `def` statement.

Hint: When dealing with functions, you should generally not be referencing any variable outside of the function. Usually, you want to be working with the arguments that are passed into it, such as `pay_string` for this function. If you're using `mark_hurd_pay_string` within your function, you're referencing an outside variable!

```
In [ ]: def convert_pay_string_to_number(pay_string):
            """Converts a pay string like '$100' (in millions) to a number of dollars."""
            return ...
```

```
In [7]: #Answer
        def convert_pay_string_to_number(pay_string):
            """Converts a pay string like '$100' (in millions) to a number of dollars."""
            return float(pay_string.strip("$"))
```

Running that cell doesn't convert any particular pay string. Instead, it creates a function called `convert_pay_string_to_number` that can convert any string with the right format to a number representing millions of dollars.

We can call our function just like we call the built-in functions we've seen. It takes one argument, a string, and it returns a number.

```
In [8]: convert_pay_string_to_number('$42')

Out[8]: 42.0
```

```
In [9]: convert_pay_string_to_number(mark_hurd_pay_string)

Out[9]: 53.25
```

In [10]:
```
# We can also compute Safra Catz's pay in the same way:
convert_pay_string_to_number(raw_compensation.where("Name", are.containing("Safra")).column("Total Pay").item(0))
```

Out[10]:
```
53.24
```

So, what have we gained by defining the `convert_pay_string_to_number` function? Well, without it, we'd have to copy that `10**6 * float(pay_string.strip("$"))` code line each time we wanted to convert a pay string. Now we just call a function whose name says exactly what it's doing.

Soon, we'll see how to apply this function to every pay string in a single expression. First, let's take a brief detour and introduce `interact`.

Using `interact`

We've included a nifty function called `interact` that allows you to call a function with different arguments.

To use it, call `interact` with the function you want to interact with as the first argument, then specify a default value for each argument of the original function like so:

In [11]:
```
_ = interact(convert_pay_string_to_number, pay_string='$42')
```

You can now change the value in the textbox to automatically call `convert_pay_string_to_number` with the argument you enter in the `pay_string` textbox. For example, entering in `'$49'` in the textbox will display the result of running `convert_pay_string_to_number('$49')`. Neat!

Note that we'll never ask you to write the `interact` function calls yourself as part of a question. However, we'll include it here and there where it's helpful and you'll probably find it useful to use yourself.

Now, let's continue on and write more functions.

2. Defining functions

Let's write a very simple function that converts a proportion to a percentage by multiplying it by 100. For example, the value of `to_percentage(.5)` should be the number 50. (No percent sign)

A function definition has a few parts.

def

It always starts with `def` (short for **def**ine):

```
def
```

Name

Next comes the name of the function. Let's call our function `to_percentage`.

```
def to_percentage
```

Signature

Next comes something called the *signature* of the function. This tells Python how many arguments your function should have, and what names you'll use to refer to those arguments in the function's code. `to_percentage` should take one argument, and we'll call that argument `proportion` since it should be a proportion.

```
def to_percentage(proportion)
```

We put a colon after the signature to tell Python it's over.

```
def to_percentage(proportion):
```

Documentation

Functions can do complicated things, so you should write an explanation of what your function does. For small functions, this is less important, but it's a good habit to learn from the start. Conventionally, Python functions are documented by writing a triple-quoted string:

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
```

Body

Now we start writing code that runs when the function is called. This is called the *body* of the function. We can write anything we could write anywhere else. First let's give a name to the number we multiply a proportion by to get a percentage.

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
    factor = 100
```

return

The special instruction `return` in a function's body tells Python to make the value of the function call equal to whatever comes right after `return` . We want the value of `to_percentage(.5)` to be the proportion .5 times the factor 100, so we write:

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
    factor = 100
    return proportion * factor
```

Note that `return` inside a function gives the function a value, while `print` , which we have used before, is a function which has no `return` value and just prints a certain value out to the console. The two are **very** different.

Question 1. Define `to_percentage` in the cell below. Call your function to convert the proportion .2 to a percentage. Name that percentage `twenty_percent` .

In [ ]:
```
def ...
    """ ... """
    ... = ...
    return ...

twenty_percent = ...
twenty_percent
```

In [12]:
```
#Answer
def to_percentage(proportion):
    """ Convert a proportion to a percentage """
    factor = 100
    return proportion*factor

twenty_percent = to_percentage(.2)
twenty_percent
```

Out[12]:
```
20.0
```

Like the built-in functions, you can use named values as arguments to your function.

Question 2. Use `to_percentage` again to convert the proportion named `a_proportion` (defined below) to a percentage called `a_percentage`.

Note: You don't need to define `to_percentage` again! Just like other named things, functions stick around after you define them.

```
In [ ]: a_proportion = 2**(.5) / 2
        a_percentage = ...
        a_percentage
```

```
In [13]: #Answer
         a_proportion = 2**(.5) / 2
         a_percentage = to_percentage(a_proportion)
         a_percentage

Out[13]: 70.71067811865476
```

Here's something important about functions: the names assigned within a function body are only accessible within the function body. Once the function has returned, those names are gone. So even though you defined `factor = 100` inside the body of the `to_percentage` function up above and then called `to_percentage`, you cannot refer to `factor` anywhere except inside the body of `to_percentage`:

```
In [14]: # You should see an error when you run this.  (If you don't, you might
         # have defined factor somewhere above.)
         factor
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-14-a219be0dab32> in <module>
      1 # You should see an error when you run this.  (If you don't,
you might
      2 # have defined factor somewhere above.)
----> 3 factor

NameError: name 'factor' is not defined
```

As we've seen with the built-in functions, functions can also take strings (or arrays, or tables) as arguments, and they can return those things, too.

Question 3. Define a function called `disemvowel`. It should take a single string as its argument. (You can call that argument whatever you want.) It should return a copy of that string, but with all the characters that are vowels removed. (In English, the vowels are the characters "a", "e", "i", "o", and "u".)

Hint: To remove all the "a"s from a string, you can use `that_string.replace("a", "")`. The `.replace` method for strings returns another string, so you can call `replace` multiple times, one after the other.

```
In [ ]: def disemvowel(a_string):
            ...
            ...

        # An example call to your function.  (It's often helpful to run
        # an example call from time to time while you're writing a function,
        # to see how it currently works.)
        disemvowel("Can you read this without vowels?")
```

```
In [15]: #Answer
         def disemvowel(a_string):
             """Remove the vowels from the string"""
             return a_string.replace("a","").replace("e","").replace("i","").re
         place("o","").replace("u","")

         # An example call to your function.  (It's often helpful to run
         # an example call from time to time while you're writing a function,
         # to see how it currently works.)
         disemvowel("Can you read this without vowels?")

Out[15]: 'Cn y rd ths wtht vwls?'
```

```
In [16]: # Alternatively, you can use interact to call your function
         _ = interact(disemvowel, a_string='Hello world')
```

Calls on calls on calls

Just as you write a series of lines to build up a complex computation, it's useful to define a series of small functions that build on each other. Since you can write any code inside a function's body, you can call other functions you've written.

If a function is a like a recipe, defining a function in terms of other functions is like having a recipe for cake telling you to follow another recipe to make the frosting, and another to make the sprinkles. This makes the cake recipe shorter and clearer, and it avoids having a bunch of duplicated frosting recipes. It's a foundation of productive programming.

For example, suppose you want to count the number of characters *that aren't vowels* in a piece of text. One way to do that is this to remove all the vowels and count the size of the remaining string.

Question 4. Write a function called `num_non_vowels` . It should take a string as its argument and return a number. The number should be the number of characters in the argument string that aren't vowels.

Hint: The function `len` takes a string as its argument and returns the number of characters in it.

In [ ]:
```
def num_non_vowels(a_string):
    """The number of characters in a string, minus the vowels."""
    ...

# Try calling your function yourself to make sure the output is what
# you expect. You can also use the interact function if you'd like.
```

In [17]:
```
#Answer
def num_non_vowels(a_string):
    """The number of characters in a string, minus the vowels."""
    return len(disemvowel(a_string))

# Try calling your function yourself to make sure the output is what
# you expect. You can also use the interact function if you'd like.
num_non_vowels("hello there")
```

Out[17]:
```
7
```

Functions can also encapsulate code that *do things* rather than just compute values. For example, if you call `print` inside a function, and then call that function, something will get printed.

The `movies_by_year` dataset in the textbook has information about movie sales in recent years. Suppose you'd like to display the year with the 5th-highest total gross movie sales, printed in a human-readable way. You might do this:

In [18]:
```
movies_by_year = Table.read_table("movies_by_year.csv")
rank = 5
fifth_from_top_movie_year = movies_by_year.sort("Total Gross", descend
ing=True).column("Year").item(rank-1)
print("Year number", rank, "for total gross movie sales was:", fifth_f
rom_top_movie_year)
```

```
Year number 5 for total gross movie sales was: 2010
```

After writing this, you realize you also wanted to print out the 2nd and 3rd-highest years. Instead of copying your code, you decide to put it in a function. Since the rank varies, you make that an argument to your function.

Question 5. Write a function called `print_kth_top_movie_year`. It should take a single argument, the rank of the year (like 2, 3, or 5 in the above examples). It should print out a message like the one above. It shouldn't have a `return` statement.

In [ ]:
```
def print_kth_top_movie_year(k):
    # Our solution used 2 lines.
    ...
    ...

# Example calls to your function:
print_kth_top_movie_year(2)
print_kth_top_movie_year(3)
```

In [19]:
```
#Answer
def print_kth_top_movie_year(k):
    # Our solution used 2 lines.
    out = movies_by_year.sort("Total Gross", descending=True).column("
Year").item(k-1)
    print("Year number", k, "for total gross movie sales was:", out)

# Example calls to your function:
print_kth_top_movie_year(2)
print_kth_top_movie_year(3)
```

```
Year number 2 for total gross movie sales was: 2013
Year number 3 for total gross movie sales was: 2012
```

In [20]:
```
# interact also allows you to pass in an array for a function argument
. It will
# then present a dropdown menu of options.
_ = interact(print_kth_top_movie_year, k=np.arange(1, 10))
```

Print is not the same as Return

The `print_kth_top_movie_year(k)` function prints the total gross movie sales for the year that was provided! However, since we did not return any value in this function, we can not use it after we call it. Let's look at an example of a function that prints a value but does not return it.

```
In [21]: def print_number_five():
             print(5)
```

```
In [22]: print_number_five()
```

```
5
```

However, if we try to use the output of `print_number_five()` , we see that we get an error when we try to add the number 5 to it!

```
In [23]: print_number_five_output = print_number_five()
         print_number_five_output + 5
```

```
5

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-23-a1be8ba5b336> in <module>
      1 print_number_five_output = print_number_five()
----> 2 print_number_five_output + 5

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

It may seem that `print_number_five()` is returning a value, 5. In reality, it just displays the number 5 to you without giving you the actual value! If your function prints out a value without returning it and you try to use it, you will run into errors so be careful!

3. `apply` ing functions

Defining a function is a lot like giving a name to a value with `=` . In fact, a function is a value just like the number 1 or the text "the"!

For example, we can make a new name for the built-in function `max` if we want:

```
In [24]: our_name_for_max = max
         our_name_for_max(2, 6)
Out[24]: 6
```

The old name for `max` is still around:

```
In [25]: max(2, 6)
Out[25]: 6
```

Try just writing `max` or `our_name_for_max` (or the name of any other function) in a cell, and run that cell. Python will print out a (very brief) description of the function.

```
In [26]: max
Out[26]: <function max>
```

Why is this useful? Since functions are just values, it's possible to pass them as arguments to other functions. Here's a simple but not-so-practical example: we can make an array of functions.

```
In [27]: make_array(max, np.average, are.equal_to)
Out[27]: array([<built-in function max>, <function average at 0x112aeaf80>,
                <function are.equal_to at 0x11b5fc560>], dtype=object)
```

Question 1. Make an array containing any 3 other functions you've seen. Call it `some_functions` .

```
In [ ]: some_functions = ...
        some_functions
```

In [28]:
```
#Answer
some_functions = make_array(min, abs, np.sum)
some_functions
```

Out[28]:
```
array([<built-in function min>, <built-in function abs>,
       <function sum at 0x1129ed4d0>], dtype=object)
```

Working with functions as values can lead to some funny-looking code. For example, see if you can figure out why this works:

In [29]:
```
make_array(max, np.average, are.equal_to).item(0)(4, -2, 7)
```

Out[29]:
```
7
```

Here's a simpler example that's actually useful: the table method `apply`.

`apply` calls a function many times, once on *each* element in a column of a table. It produces an array of the results. Here we use `apply` to convert every CEO's pay to a number, using the function you defined:

```
In [30]: raw_compensation.apply(convert_pay_string_to_number, "Total Pay")
```

```
Out[30]: array([5.325e+01, 5.324e+01, 4.491e+01, 3.598e+01, 3.336e+01, 2.484e
+01,
       2.204e+01, 1.986e+01, 1.962e+01, 1.932e+01, 1.876e+01, 1.861e
+01,
       1.836e+01, 1.809e+01, 1.710e+01, 1.663e+01, 1.633e+01, 1.614e
+01,
       1.610e+01, 1.602e+01, 1.510e+01, 1.498e+01, 1.463e+01, 1.451e
+01,
       1.444e+01, 1.436e+01, 1.431e+01, 1.409e+01, 1.400e+01, 1.367e
+01,
       1.234e+01, 1.220e+01, 1.218e+01, 1.213e+01, 1.205e+01, 1.184e
+01,
       1.171e+01, 1.163e+01, 1.116e+01, 1.111e+01, 1.111e+01, 1.073e
+01,
       1.050e+01, 1.043e+01, 1.037e+01, 1.028e+01, 1.027e+01, 1.018e
+01,
       1.016e+01, 9.970e+00, 9.960e+00, 9.860e+00, 9.740e+00, 9.420e
+00,
       9.390e+00, 9.220e+00, 9.060e+00, 9.030e+00, 8.860e+00, 8.760e
+00,
       8.570e+00, 8.380e+00, 8.360e+00, 8.350e+00, 8.230e+00, 7.860e
+00,
       7.700e+00, 7.580e+00, 7.510e+00, 7.230e+00, 7.210e+00, 7.120e
+00,
       6.880e+00, 6.770e+00, 6.640e+00, 6.560e+00, 6.140e+00, 5.920e
+00,
       5.900e+00, 5.890e+00, 5.730e+00, 5.420e+00, 5.040e+00, 4.920e
+00,
       4.920e+00, 4.470e+00, 4.250e+00, 4.080e+00, 3.930e+00, 3.720e
+00,
       2.880e+00, 2.830e+00, 2.820e+00, 2.450e+00, 1.790e+00, 1.680e
+00,
       1.530e+00, 9.400e-01, 8.100e-01, 7.000e-02, 4.000e-02, 0.000e
+00])
```

Here's an illustration of what that did:

Note that we didn't write something like `convert_pay_string_to_number()` or `convert_pay_string_to_number("Total Pay")`. The job of `apply` is to call the function we give it, so instead of calling `convert_pay_string_to_number` ourselves, we just write its name as an argument to `apply`.

Question 2. Using `apply`, make a table that's a copy of `raw_compensation` with one more column called "Total Pay ($)". It should be the result of applying `convert_pay_string_to_number` to the "Total Pay" column, as we did above, and creating a new table which is the old one, but with the additional "Total Pay ($)" column. Call the new table `compensation`.

In [ ]:
```
compensation = raw_compensation.with_column(
    "Total Pay ($)",
    ...
compensation
```

In [31]:
```
#Answer
compensation = raw_compensation.with_column(
    "Total Pay ($)", raw_compensation.apply(convert_pay_string_to_numb
er, "Total Pay"))
compensation
```

```
Out[31]:
```

Rank	Name	Company (Headquarters)	Total Pay	% Change	Cash Pay	Equity Pay	Other Pay	Ratio of CEO pay to average industry worker pay	Total Pay ($)
1	Mark V. Hurd*	Oracle (Redwood City)	$53.25	(No previous year)	$0.95	$52.27	$0.02	362	53.25
2	Safra A. Catz*	Oracle (Redwood City)	$53.24	(No previous year)	$0.95	$52.27	$0.02	362	53.24
3	Robert A. Iger	Walt Disney (Burbank)	$44.91	-3%	$24.89	$17.28	$2.74	477	44.91
4	Marissa A. Mayer	Yahoo! (Sunnyvale)	$35.98	-15%	$1.00	$34.43	$0.55	342	35.98
5	Marc Benioff	salesforce.com (San Francisco)	$33.36	-16%	$4.65	$27.26	$1.45	338	33.36
6	John H. Hammergren	McKesson (San Francisco)	$24.84	-4%	$12.10	$12.37	$0.37	222	24.84
7	John S. Watson	Chevron (San Ramon)	$22.04	-15%	$4.31	$14.68	$3.05	183	22.04
8	Jeffrey Weiner	LinkedIn (Mountain View)	$19.86	27%	$2.47	$17.26	$0.13	182	19.86
9	John T. Chambers**	Cisco Systems (San Jose)	$19.62	19%	$5.10	$14.51	$0.01	170	19.62
10	John G. Stumpf	Wells Fargo (San Francisco)	$19.32	-10%	$6.80	$12.50	$0.02	256	19.32

... (92 rows omitted)

Now that we have the pay in numbers, we can compute things about them.

Question 3. Compute the average total pay of the CEOs in the dataset.

```
In [ ]: average_total_pay = ...
        average_total_pay
```

```
In [32]: #Answer
         average_total_pay = np.average(compensation.column("Total Pay ($)"))
         average_total_pay
```

```
Out[32]: 11.445294117647055
```

Question 4. Companies pay executives in a variety of ways: directly in cash; by granting stock or other "equity" in the company; or with ancillary benefits (like private jets). Compute the proportion of each CEO's pay that was cash. (Your answer should be an array of numbers, one for each CEO in the dataset.)

```
In [ ]: cash_proportion = ...
        cash_proportion
```

```
In [33]: #Answer - quick check (students can use the next cell...they do not ne
         ed to remove the 0 in the denominator)
         compensation2 = compensation.where("Total Pay ($)", are.above(0))
         cash_proportion = compensation2.apply(convert_pay_string_to_number, "C
         ash Pay")/compensation2.column("Total Pay ($)")
         cash_proportion
```

```
Out[33]: array([0.01784038, 0.01784373, 0.55421955, 0.02779322, 0.13938849,
                0.48711755, 0.19555354, 0.12437059, 0.25993884, 0.35196687,
                0.3075693 , 0.22138635, 0.13126362, 0.1708126 , 0.23099415,
                0.06734817, 0.13043478, 0.28004957, 0.33229814, 0.15355805,
                0.29337748, 0.21829105, 0.31100478, 0.25086147, 0.2299169 ,
                0.16991643, 0.31795947, 0.26188786, 0.28357143, 0.15654718,
                0.38168558, 0.28934426, 0.20361248, 0.47650453, 0.45643154,
                0.36402027, 0.2177626 , 0.24763543, 0.42562724, 0.2610261 ,
                0.18361836, 0.1444548 , 0.33333333, 0.10834132, 0.20925747,
                0.97276265, 0.22979552, 0.22789784, 0.37893701, 0.25175527,
                0.73895582, 0.37018256, 0.2412731 , 0.2133758 , 0.20553781,
                0.23318872, 0.33664459, 0.3875969 , 0.56094808, 0.11757991,
                0.35239207, 0.24463007, 0.25      , 0.23712575, 0.43377886,
                0.31424936, 0.46363636, 0.32585752, 0.24766977, 0.98755187,
                0.27184466, 0.96207865, 0.31831395, 0.81979321, 0.23795181,
                0.17530488, 0.21172638, 0.37162162, 0.27288136, 0.26994907,
                0.55148342, 0.3597786 , 0.        , 0.47154472, 0.47154472,
                0.29753915, 0.16235294, 0.48529412, 0.46819338, 0.32526882,
                0.98958333, 0.61130742, 0.67021277, 0.75510204, 0.50837989,
                0.98809524, 0.98039216, 0.9893617 , 0.87654321, 0.        ,
                1.        ])
```

In [34]:
```
#Answer
cash_proportion = compensation.apply(convert_pay_string_to_number, "Ca
sh Pay")/compensation.column("Total Pay ($)")
cash_proportion
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: Runt
imeWarning: invalid value encountered in true_divide
```

Out[34]:
```
array([0.01784038, 0.01784373, 0.55421955, 0.02779322, 0.13938849,
       0.48711755, 0.19555354, 0.12437059, 0.25993884, 0.35196687,
       0.3075693 , 0.22138635, 0.13126362, 0.1708126 , 0.23099415,
       0.06734817, 0.13043478, 0.28004957, 0.33229814, 0.15355805,
       0.29337748, 0.21829105, 0.31100478, 0.25086147, 0.2299169 ,
       0.16991643, 0.31795947, 0.26188786, 0.28357143, 0.15654718,
       0.38168558, 0.28934426, 0.20361248, 0.47650453, 0.45643154,
       0.36402027, 0.2177626 , 0.24763543, 0.42562724, 0.2610261 ,
       0.18361836, 0.1444548 , 0.33333333, 0.10834132, 0.20925747,
       0.97276265, 0.22979552, 0.22789784, 0.37893701, 0.25175527,
       0.73895582, 0.37018256, 0.2412731 , 0.2133758 , 0.20553781,
       0.23318872, 0.33664459, 0.3875969 , 0.56094808, 0.11757991,
       0.35239207, 0.24463007, 0.25      , 0.23712575, 0.43377886,
       0.31424936, 0.46363636, 0.32585752, 0.24766977, 0.98755187,
       0.27184466, 0.96207865, 0.31831395, 0.81979321, 0.23795181,
       0.17530488, 0.21172638, 0.37162162, 0.27288136, 0.26994907,
       0.55148342, 0.3597786 , 0.        , 0.47154472, 0.47154472,
       0.29753915, 0.16235294, 0.48529412, 0.46819338, 0.32526882,
       0.98958333, 0.61130742, 0.67021277, 0.75510204, 0.50837989,
       0.98809524, 0.98039216, 0.9893617 , 0.87654321, 0.        ,
       1.        ,        nan])
```

Check out the "% Change" column in `compensation`. It shows the percentage increase in the CEO's pay from the previous year. For CEOs with no previous year on record, it instead says "(No previous year)". The values in this column are *strings*, not numbers, so like the "Total Pay" column, it's not usable without a bit of extra work.

Given your current pay and the percentage increase from the previous year, you can compute your previous year's pay. For example, if your pay is $\$100$ this year, and that's an increase of 50% from the previous year, then your previous year's pay was $\frac{\$100}{1+\frac{50}{100}}$, or around \$66.66.

Question 5. Create a new table called `with_previous_compensation`. It should be a copy of `compensation`, but with the "(No previous year)" CEOs filtered out, and with an extra column called "2014 Total Pay ($)". That column should have each CEO's pay in 2014.

Hint 1: You can print out your results after each step to make sure you're on the right track.

Hint 2: We've provided a structure that you can use to get to the answer. However, if it's confusing, feel free to delete the current structure and approach the problem your own way!

In [ ]:
```
# Definition to turn percent to number
def percent_string_to_num(percent_string):
    return ...

# Compensation table where there is a previous year
having_previous_year = ...

# Get the percent changes as numbers instead of strings
percent_changes = ...

# Calculate the previous years pay
previous_pay = ...

# Put the previous pay column into the compensation table
with_previous_compensation = ...

with_previous_compensation
```

In [35]:
```
#Answer
# Definition to turn percent to number
def percent_string_to_num(percent_string):
    return float(percent_string.strip("%"))

# Compensation table where there is a previous year
having_previous_year = compensation.where("% Change", are.not_equal_to
("(No previous year)"))

# Get the percent changes as numbers instead of strings
percent_changes = having_previous_year.apply(percent_string_to_num, "%
Change")

# Calculate the previous years pay
previous_pay = having_previous_year.column('Total Pay ($)')/(1+percent
_changes/100)

# Put the previous pay column into the compensation table
with_previous_compensation = having_previous_year.with_column("Previou
s Pay", previous_pay)

with_previous_compensation
```

Out[35]:

Rank	Name	Company (Headquarters)	Total Pay	% Change	Cash Pay	Equity Pay	Other Pay	Ratio of CEO pay to average industry worker pay	Total Pay ($)	Pr
3	Robert A. Iger	Walt Disney (Burbank)	$44.91	-3%	$24.89	$17.28	$2.74	477	44.91	
4	Marissa A. Mayer	Yahoo! (Sunnyvale)	$35.98	-15%	$1.00	$34.43	$0.55	342	35.98	4
5	Marc Benioff	salesforce.com (San Francisco)	$33.36	-16%	$4.65	$27.26	$1.45	338	33.36	3
6	John H. Hammergren	McKesson (San Francisco)	$24.84	-4%	$12.10	$12.37	$0.37	222	24.84	
7	John S. Watson	Chevron (San Ramon)	$22.04	-15%	$4.31	$14.68	$3.05	183	22.04	2
8	Jeffrey Weiner	LinkedIn (Mountain View)	$19.86	27%	$2.47	$17.26	$0.13	182	19.86	1
9	John T. Chambers**	Cisco Systems (San Jose)	$19.62	19%	$5.10	$14.51	$0.01	170	19.62	1
10	John G. Stumpf	Wells Fargo (San Francisco)	$19.32	-10%	$6.80	$12.50	$0.02	256	19.32	2
11	John C. Martin**	Gilead Sciences (Foster City)	$18.76	-1%	$5.77	$12.98	$0.01	117	18.76	1
13	Shantanu Narayen	Adobe Systems (San Jose)	$18.36	3%	$2.41	$15.85	$0.09	125	18.36	1

... (71 rows omitted)

Question 6. What was the average pay of these CEOs in 2014?

In [ ]:
```
average_pay_2014 = ...
average_pay_2014
```

In [36]:
```
#Answer
average_pay_2014 = np.average(with_previous_compensation.column("Previ
ous Pay"))
average_pay_2014
```

Out[36]:
```
11.649176115603435
```

4. Histograms

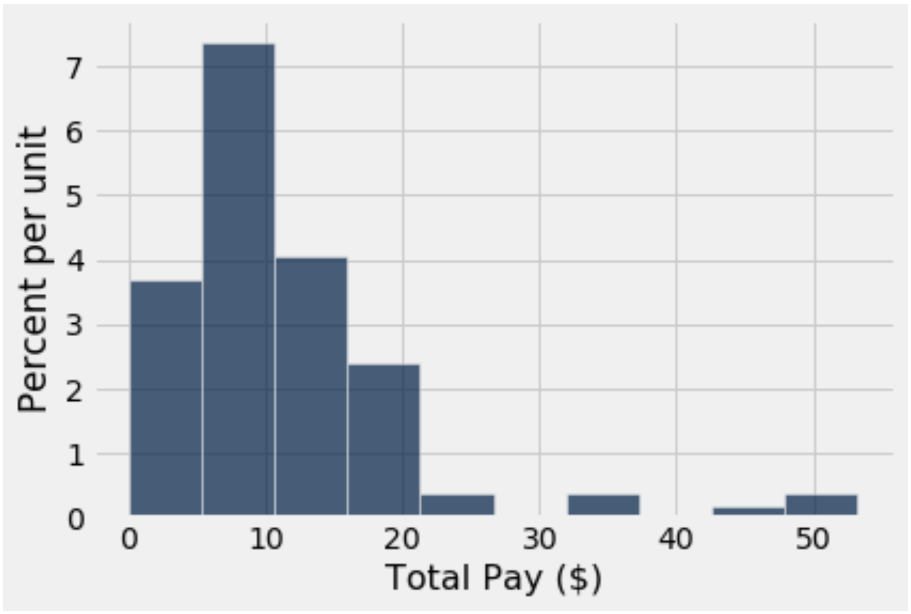
Earlier, we computed the average pay among the CEOs in our 102-CEO dataset. The average doesn't tell us everything about the amounts CEOs are paid, though. Maybe just a few CEOs make the bulk of the money, even among these 102.

We can use a *histogram* method to display more information about a set of numbers. The table method `hist` takes a single argument, the name of a column of numbers. It produces a histogram of the numbers in that column.

Question 1. Make a histogram of the pay of the CEOs in `compensation`.

In [ ]:
```
...
```

In [37]:
```
#Answer
compensation.hist("Total Pay ($)")
```



Question 2. Looking at the histogram, how many CEOs made more than $30 million? Answer the question with code. *Hint:* Use the table method `where` and the property `num_rows`.

In [ ]:
```
num_ceos_more_than_30_million_2 = ...
num_ceos_more_than_30_million_2
```

In [38]:
```
#Answer
num_ceos_more_than_30_million_2 = compensation.where("Total Pay ($)", 
are.above(30)).num_rows
num_ceos_more_than_30_million_2
```

Out[38]:
```
5
```