lab06

November 8, 2018

1 Lab 6

Welcome to Lab 6! In this lab we will learn about sampling strategies. More information about Sampling in the textbook can be found here!

The data used in this lab will contain salary data and statistics for basketball players from the 2014-2015 NBA season. This data was collected from basketball-reference and spotrac.

In [70]:
```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')

# Don't change this cell; just run it.
from client.api.notebook import Notebook
ok = Notebook('lab06.ok')
_ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: Statistics_and_sampling
OK, version v1.13.11
=====================================================================

Successfully logged in as wec149@ucsd.edu
```

1.1 1. Dungeons and Dragons and Sampling

In the game Dungeons & Dragons, each player plays the role of a fantasy character.

A player performs actions by rolling a 20-sided die, adding a "modifier" number to the roll, and comparing the total to a threshold for success. The modifier depends on her character's competence in performing the action.

For example, suppose Alice's character, a barbarian warrior named Roga, is trying to knock down a heavy door. She rolls a 20-sided die, adds a modifier of 11 to the result (because her character is good at knocking down doors), and succeeds if the total is greater than 15.

** Question 1.1 ** Write code that simulates that procedure. Compute three values: the result of Alice's roll (`roll_result`), the result of her roll plus Roga's modifier (`modified_result`), and a boolean value indicating whether the action succeeded (`action_succeeded`). **Do not fill in any of the results manually**; the entire simulation should happen in code.

Hint: A roll of a 20-sided die is a number chosen uniformly from the array `make_array(1, 2, 3, 4, ..., 20)`. So a roll of a 20-sided die *plus 11* is a number chosen uniformly from that array, plus 11.

```
In [71]: possible_rolls = make_array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
         roll_result = np.random.choice(possible_rolls)
         modified_result = roll_result + 11
         action_succeeded = modified_result > 15

         # The next line just prints out your results in a nice way
         # once you're done.  You can delete it if you want.
         print("On a modified roll of {:d}, Alice's action {}.".format(modified_result, "succeed
```

```
On a modified roll of 16, Alice's action succeeded.
```

```
In [72]: _ = ok.grade('q1_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% passed
```

** Question 1.2 ** Run your cell 7 times to manually estimate the chance that Alice succeeds at this action. (Don't use math or an extended simulation.). Your answer should be a fraction.

```
In [73]: for i in np.arange(7):
             roll_result = np.random.choice(possible_rolls)
             modified_result = roll_result + 11
             action_succeeded = modified_result > 15
             print(modified_result)
             print(action_succeeded)

         rough_success_chance = 5/7
         rough_success_chance
```

```
13
False
17
True
12
False
12
False
20
True
29
True
25
True
```

```
Out[73]: 0.7142857142857143
```

```
In [74]: _ = ok.grade('q1_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Suppose we don't know that Roga has a modifier of 11 for this action. Instead, we observe the modified roll (that is, the die roll plus the modifier of 11) from each of 7 of her attempts to knock down doors. We would like to estimate her modifier from these 7 numbers.

** Question 1.3 ** Write a Python function called `simulate_observations`. It should take no arguments, and it should return an array of 7 numbers. Each of the numbers should be the modified roll from one simulation. **Then**, call your function once to compute an array of 7 simulated modified rolls. Name that array `observations`.

```
In [75]: modifier = 11
         num_observations = 7

         def simulate_observations():
             """Produces an array of 7 simulated modified die rolls"""
             modified_roll = make_array()
             for i in np.arange(num_observations):
                 roll_result = np.random.choice(possible_rolls)
                 modified_result = roll_result + modifier
                 modified_roll = np.append(modified_roll, modified_result)
```

```
            return modified_roll

        observations = simulate_observations()
        observations
```

```
Out[75]: array([14., 12., 14., 24., 24., 20., 17.])
```

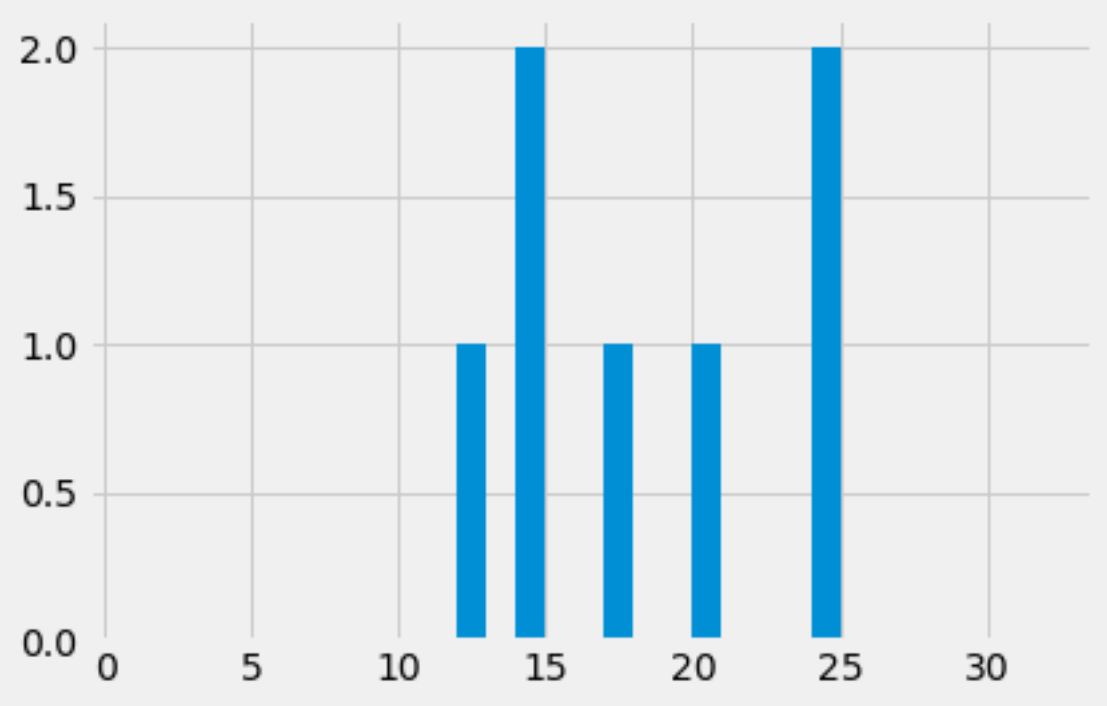
```
In [76]: _ = ok.grade('q1_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

** Question 1.4 ** Draw a histogram to display the *probability distribution* of the modified rolls we might see.

```
In [77]: # We suggest using these bins.
         roll_bins = np.arange(1, modifier+2+20, 1)

         plots.hist(observations, bins=roll_bins)
```

```
Out[77]: (array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 2., 0., 0., 1.,
                 0., 0., 1., 0., 0., 0., 2., 0., 0., 0., 0., 0., 0., 0.]),
          array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
                 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
          <a list of 31 Patch objects>)
```



Now let's imagine we don't know the modifier and try to estimate it from `observations`.

One straightforward way to do that is to find the *smallest* total roll. The smallest roll on a 20-sided die is 1, so if we get a roll of 1 then the modifier must be 0.

** Question 1.5 ** Using that method, estimate `modifier` from `observations`. Name your estimate `min_estimate`.

```
In [78]: min_estimate = min(observations) - 1
         min_estimate

Out[78]: 11.0

In [79]: _ = ok.grade('q1_5')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% passed
```

Another way to estimate the modifier involves the mean of `observations`.

** Question 1.6 ** Figure out a good estimate based on that quantity. **Then**, write a function named `mean_based_estimator` that computes your estimate. It should take an array of modified

rolls (like the array `observations`) as its argument and return an estimate of `modifier` based on those numbers.

```
In [80]: def mean_based_estimator(nums):
             """Estimate the roll modifier based on observed modified rolls in the array nums."
             roll_mean = np.mean(nums)
             estimate = roll_mean - 11
             return estimate

         # Here is an example call to your function.  It computes an estimate
         # of the modifier from our 7 observations.
         mean_based_estimate = mean_based_estimator(observations)
         mean_based_estimate

Out[80]: 6.857142857142858

In [81]: _ = ok.grade('q1_6')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% passed
```

1.2 2. Sampling

Run the cell below to load the player and salary data.

```
In [82]: player_data = Table().read_table("player_data.csv")
         salary_data = Table().read_table("salary_data.csv")
         full_data = salary_data.join("PlayerName", player_data, "Name")
         # The show method immediately displays the contents of a table.
         # This way, we can display the top of two tables using a single cell.
         player_data.show(3)
         salary_data.show(3)
         full_data.show(3)

<IPython.core.display.HTML object>


<IPython.core.display.HTML object>


<IPython.core.display.HTML object>
```

Rather than getting data on every player, imagine that we had gotten data on only a smaller subset of the players. For 492 players, it's not so unreasonable to expect to see all the data, but usually we aren't so lucky. Instead, we often make *statistical inferences* about a large underlying population using a smaller sample.

A statistical inference is a statement about some statistic of the underlying population, such as "the average salary of NBA players in 2014 was $3". You may have heard the word "inference" used in other contexts. It's important to keep in mind that statistical inferences can be wrong.

A general strategy for inference using samples is to estimate statistics of the population by computing the same statistics on a sample. This strategy sometimes works well and sometimes doesn't. The degree to which it gives us useful answers depends on several factors, and we'll touch lightly on a few of those today.

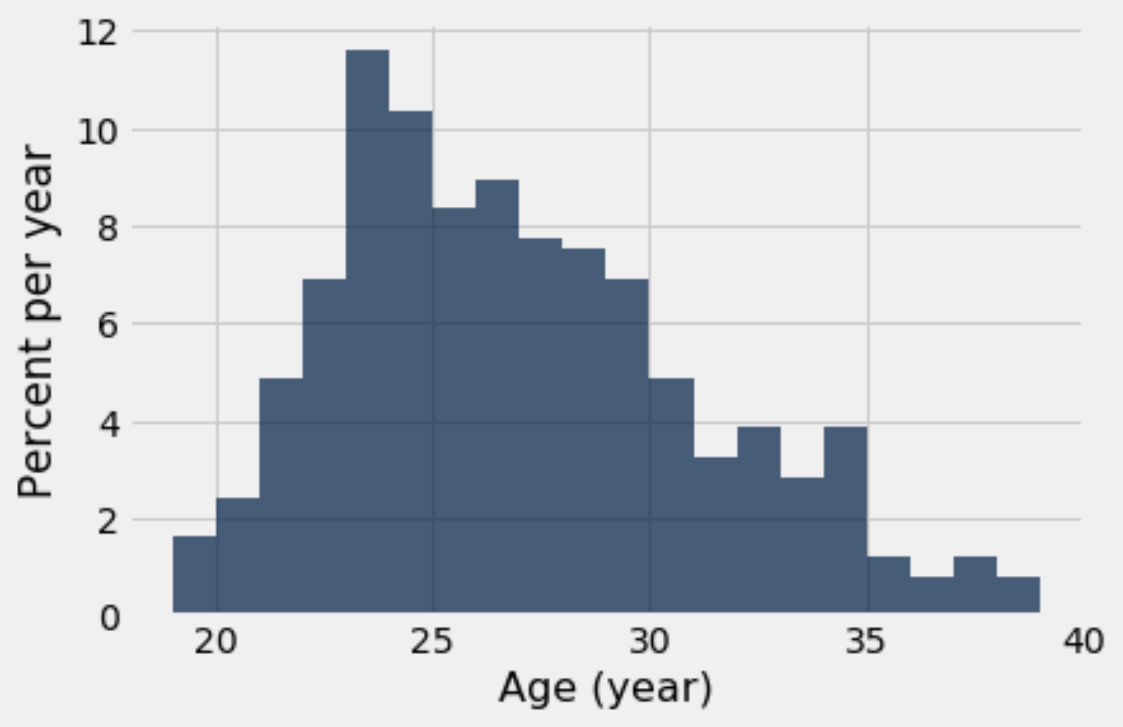
One very important factor in the utility of samples is how they were gathered. We have prepared some example sample datasets to simulate inference from different kinds of samples for the NBA player dataset. Later we'll ask you to create your own samples to see how they behave.

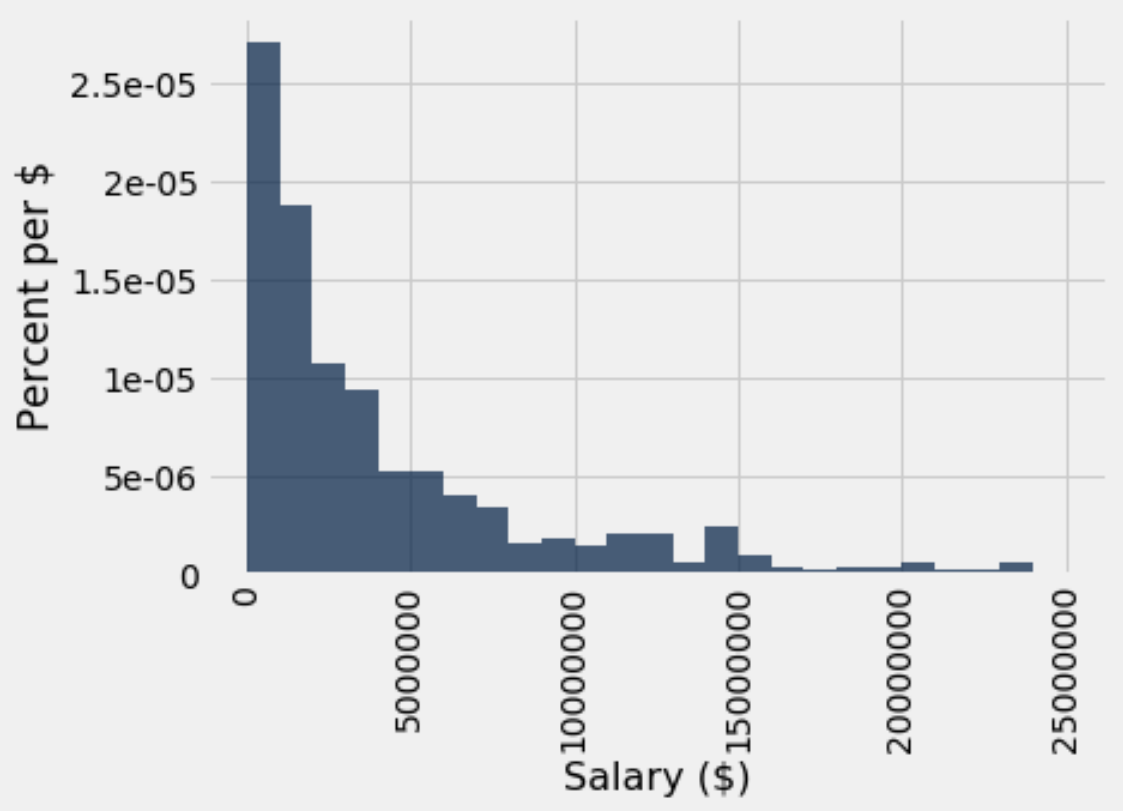
To save typing and increase the clarity of your code, we will package the loading and analysis code into two functions. This will be useful in the rest of the lab as we will repeatedly need to create histograms and collect summary statistics from that data.

Question 2.1. Complete the `histograms` function, which takes a table with columns `Age` and `Salary` and draws a histogram for each one. Use the min and max functions to pick the bin boundaries so that all data appears for any table passed to your function. Use the same bin widths as before (1 year for `Age` and $1,000,000 for `Salary`).

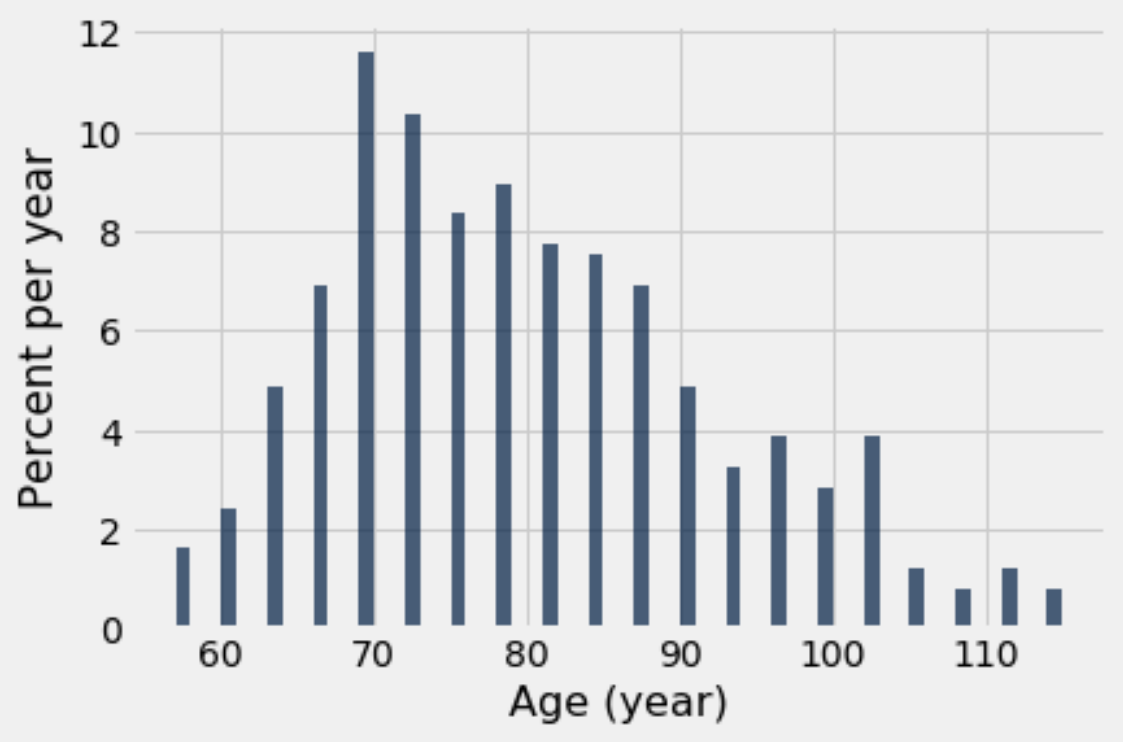
Hint: Make sure that your bins **include** the maximum value. Remember that bins include the left value but exclude the right value.

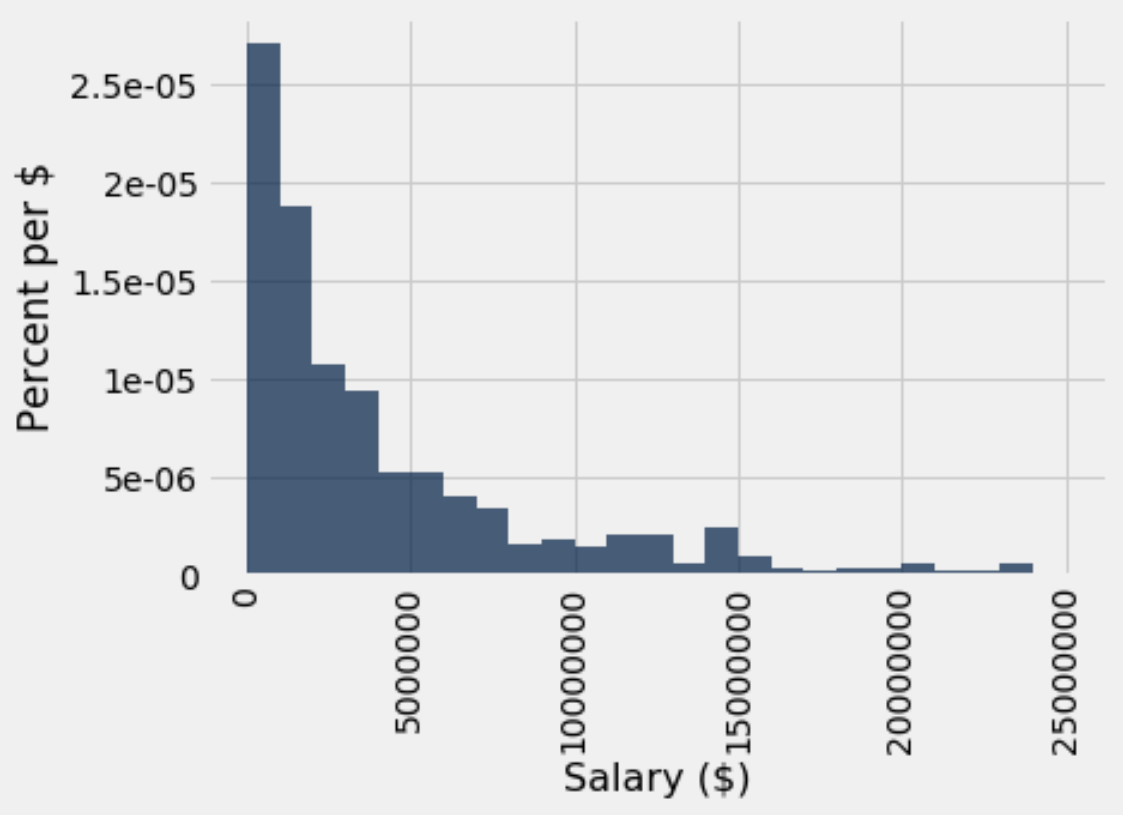
```
In [83]: def histograms(t):
             ages = t.column('Age')
             salaries = t.column('Salary')
             age_bins = np.arange(min(t.column('Age')),max(t.column('Age')) + 2,1)
             salary_bins = np.arange(min(t.column('Salary')),max(t.column('Salary')) + 2000000,
             t.hist('Age', bins=age_bins, unit='year')
             t.hist('Salary', bins=salary_bins, unit='$')
             return age_bins, salary_bins # Keep this statement so that your work can be checked

         histograms(full_data)
         print('Two histograms should be displayed below')

Two histograms should be displayed below
```

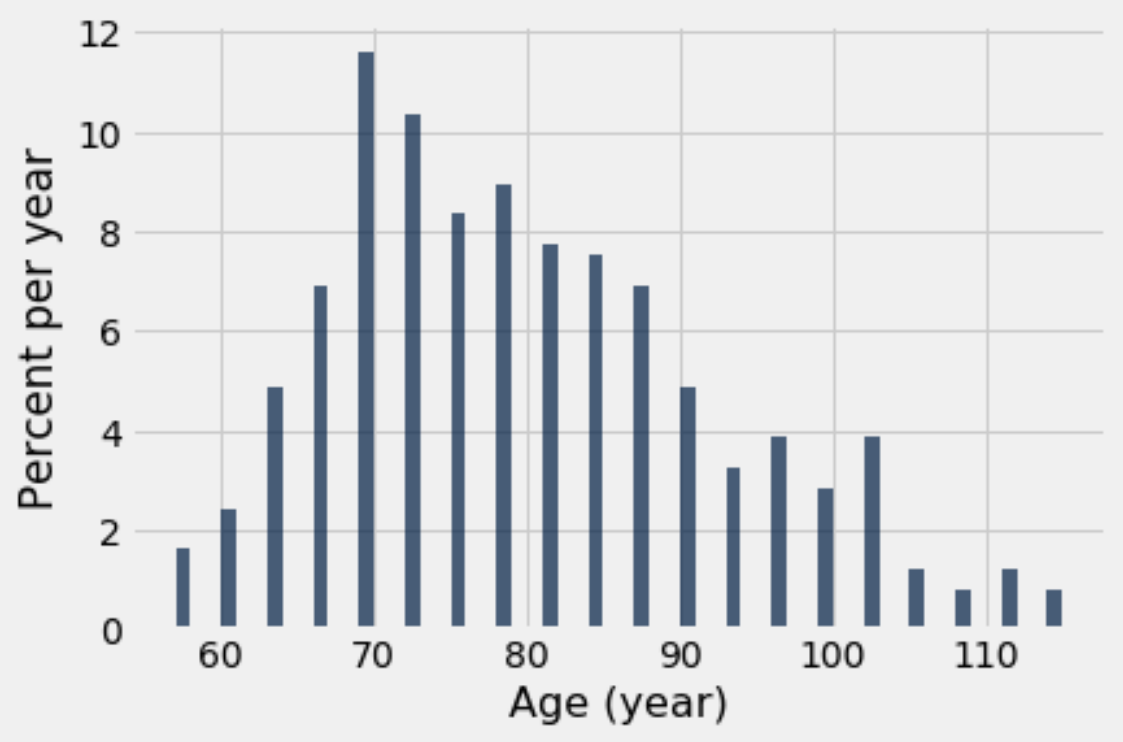

Percent per year
12
10
8
6
4
2
0
20
25
30
35
40
Age (year)

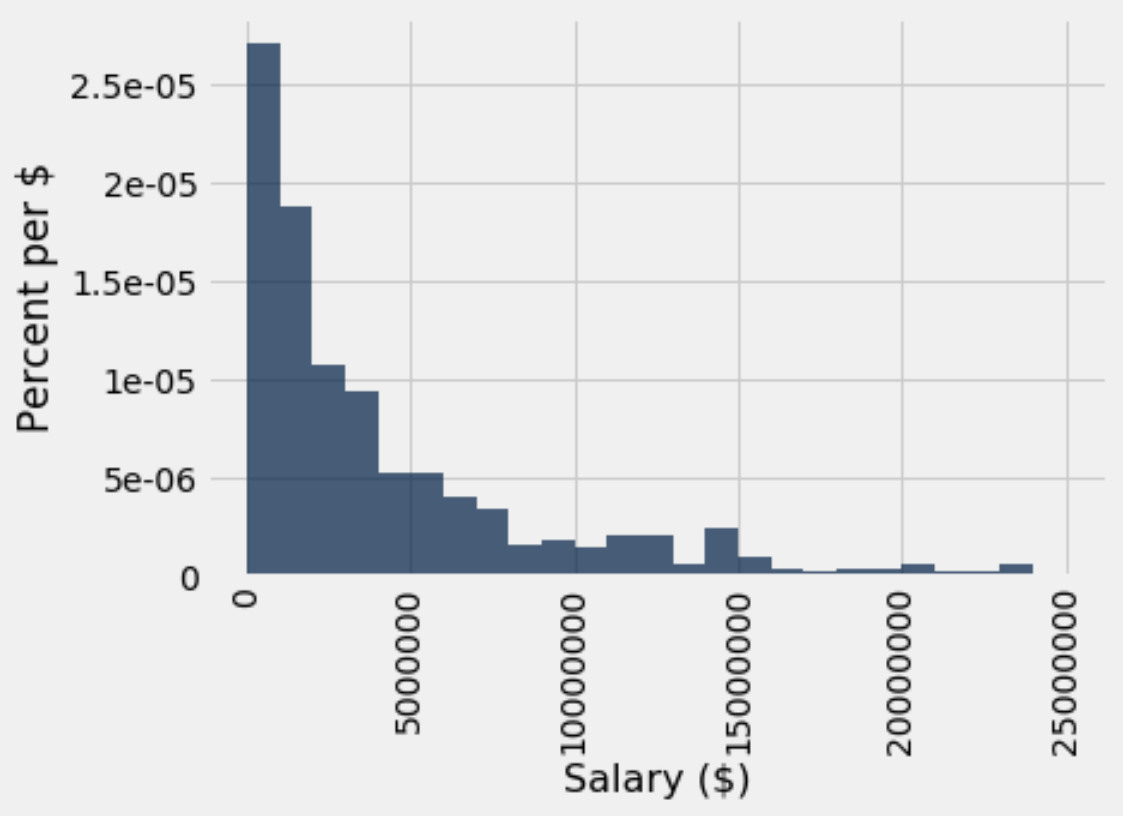

Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

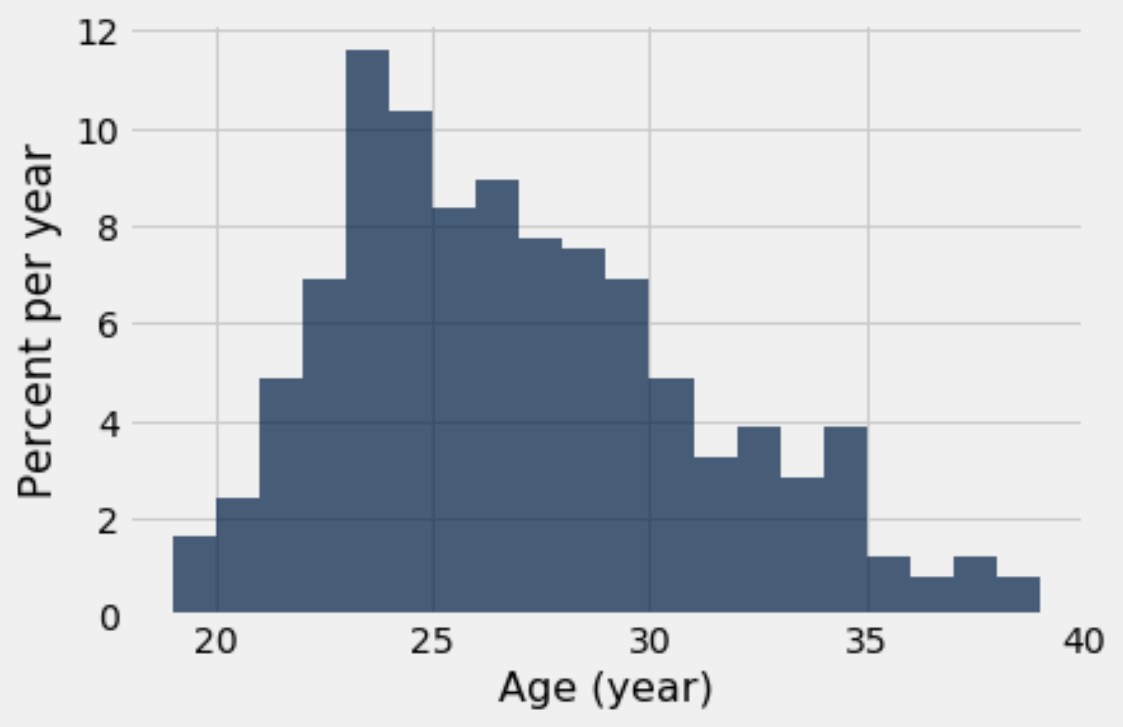
```
In [84]: _ = ok.grade('q2_1') # Warning: Charts will be displayed while running this test

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

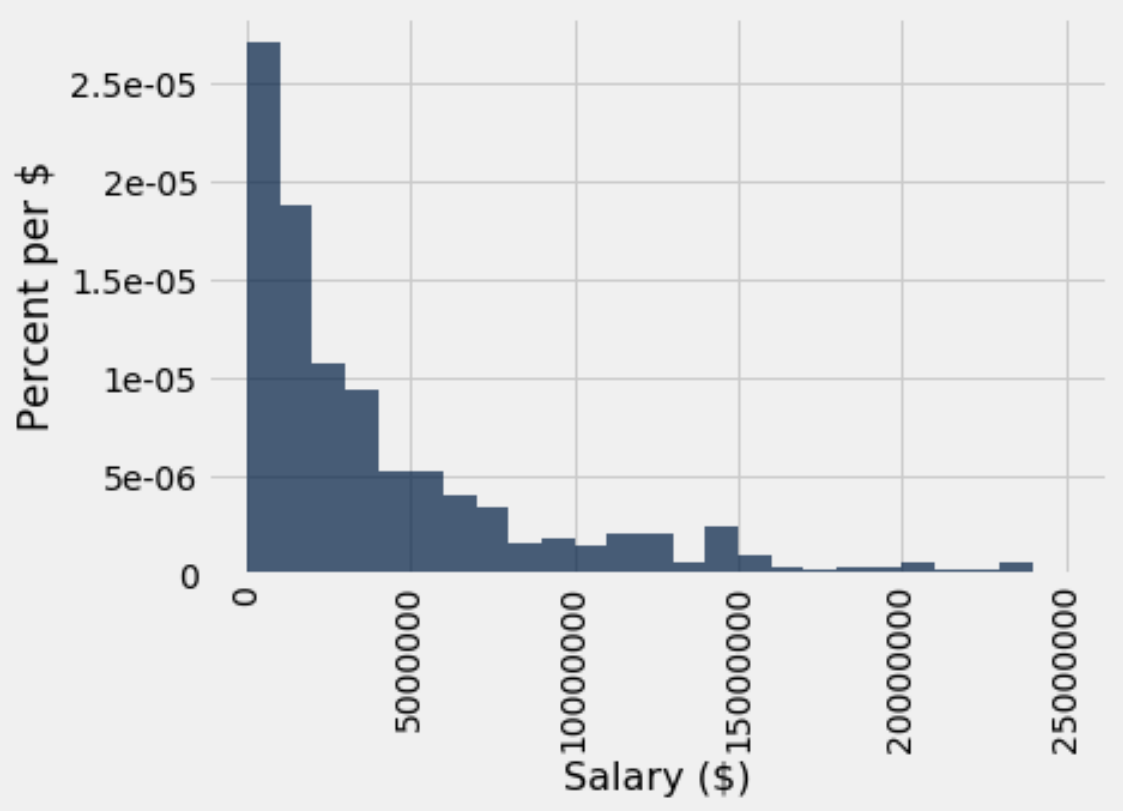


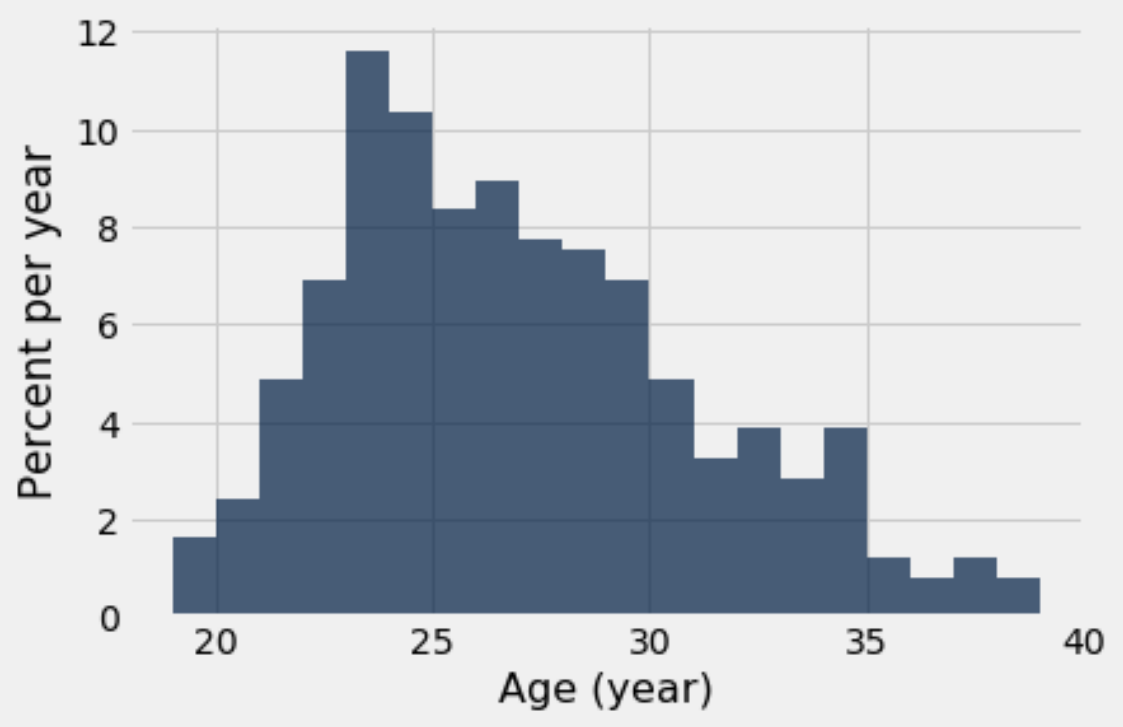

2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

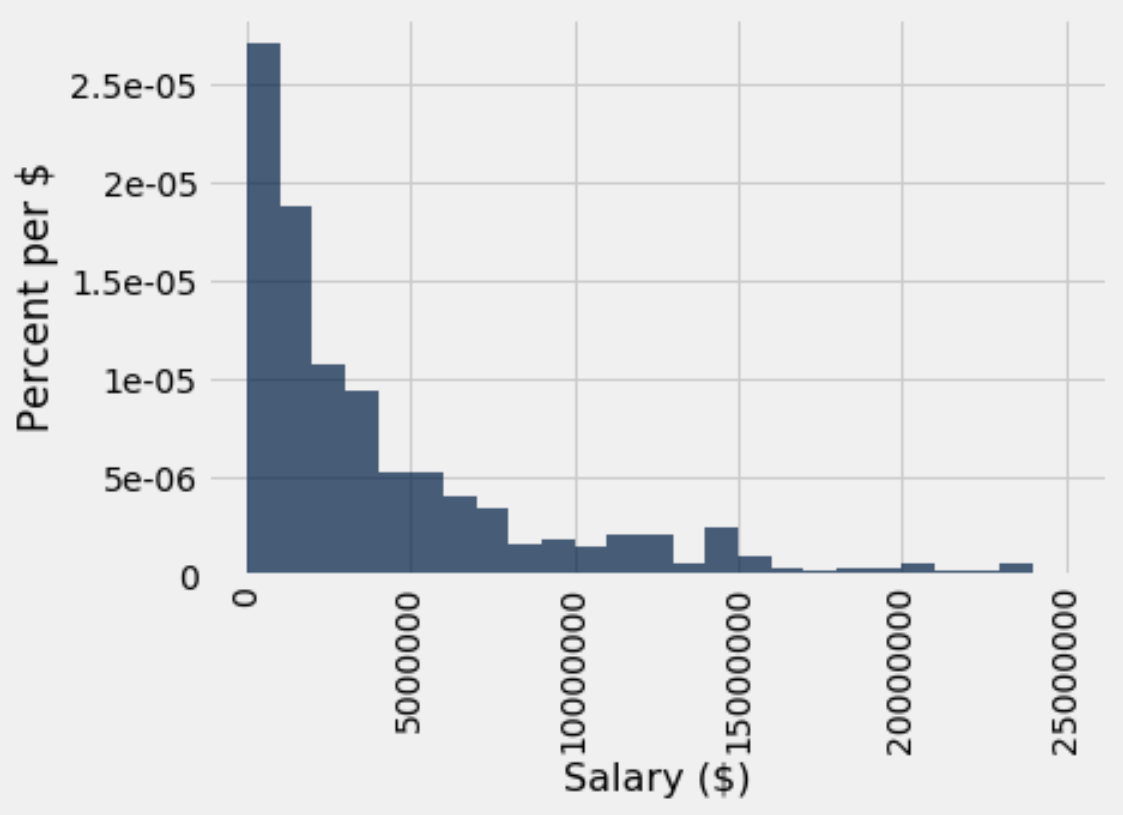

12
10
8
6
4
2
0
Percent per year
60
70
80
90
100
110
Age (year)


2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


Percent per year
12
10
8
6
4
2
0
20
25
30
35
40
Age (year)


Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


12
10
8
6
4
2
0
Percent per year
20
25
30
35
40
Age (year)

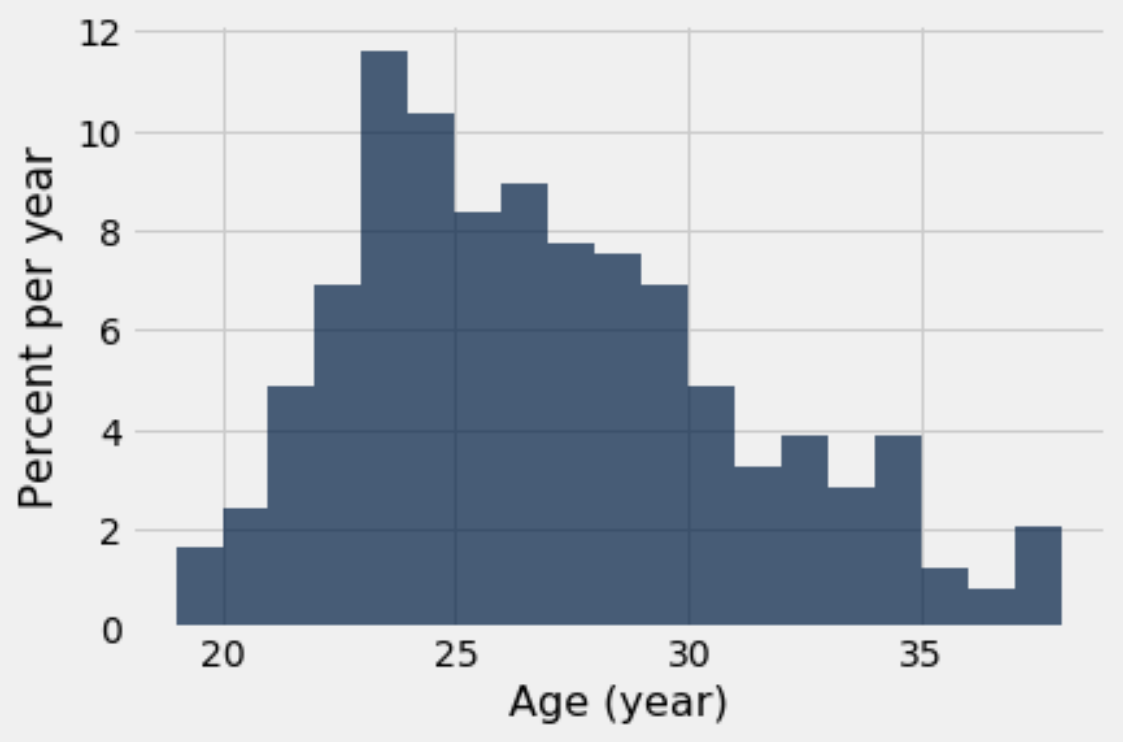


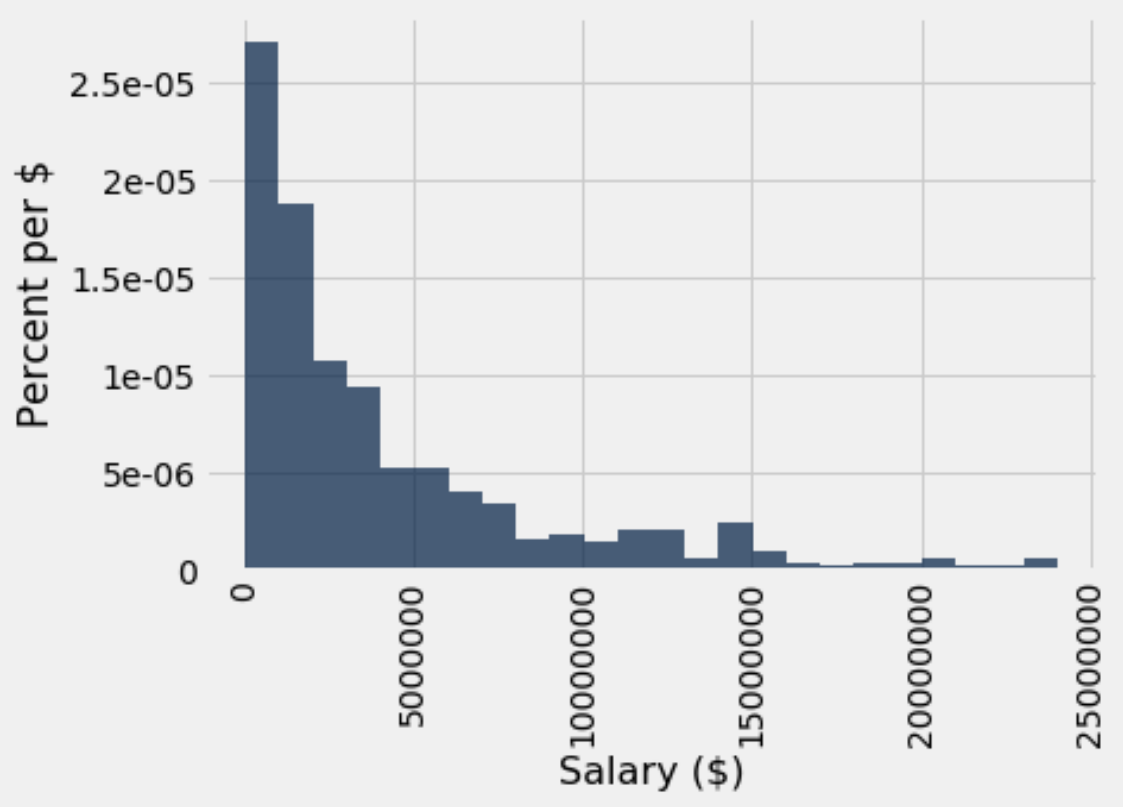
Question 2.2. Create a function called `compute_statistics` that takes a Table containing ages and salaries and: - Draws a histogram of ages - Draws a histogram of salaries - Return a two-element array containing the average age and average salary

You can call your `histograms` function to draw the histograms!

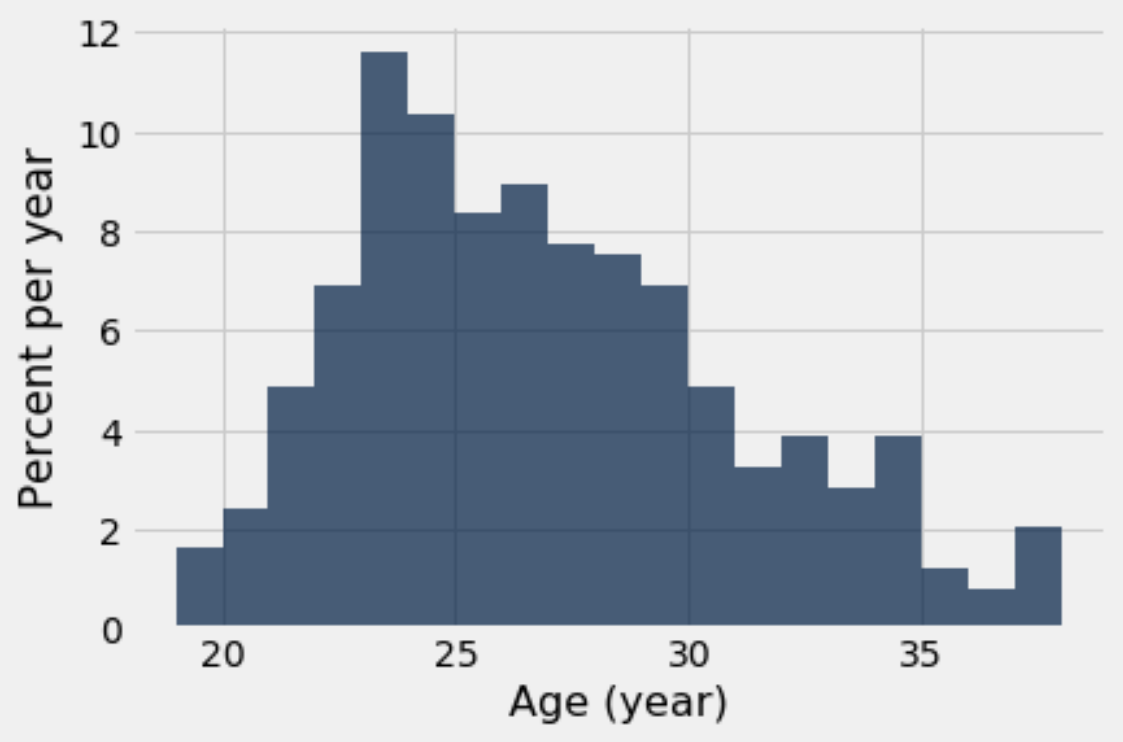
```
In [85]: def compute_statistics(age_and_salary_data):

             age_of_table = age_and_salary_data.column('Age')
             salaries_of_table = age_and_salary_data.column('Salary')

             age_bins = np.arange(min(age_of_table),max(age_of_table) + 1,1)
             salary_bins = np.arange(min(salaries_of_table),max(salaries_of_table) + 1000000, 10

             age = age_and_salary_data.hist('Age', bins=age_bins, unit='year')
             salary = age_and_salary_data.hist('Salary', bins=salary_bins, unit='$')

             answer = make_array(np.mean(age_of_table), np.mean(salaries_of_table))

             return answer


         full_stats = compute_statistics(full_data)
         full_stats
```

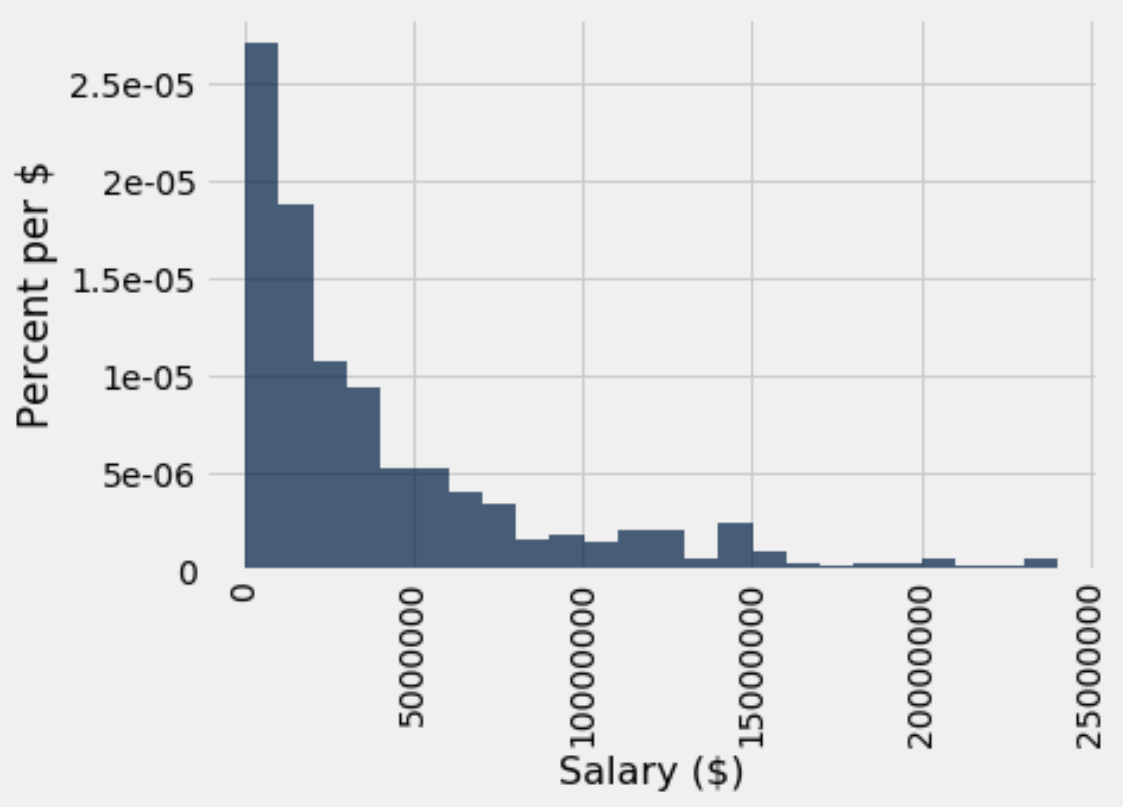
Out[85]: array([2.65365854e+01, 4.26977577e+06])

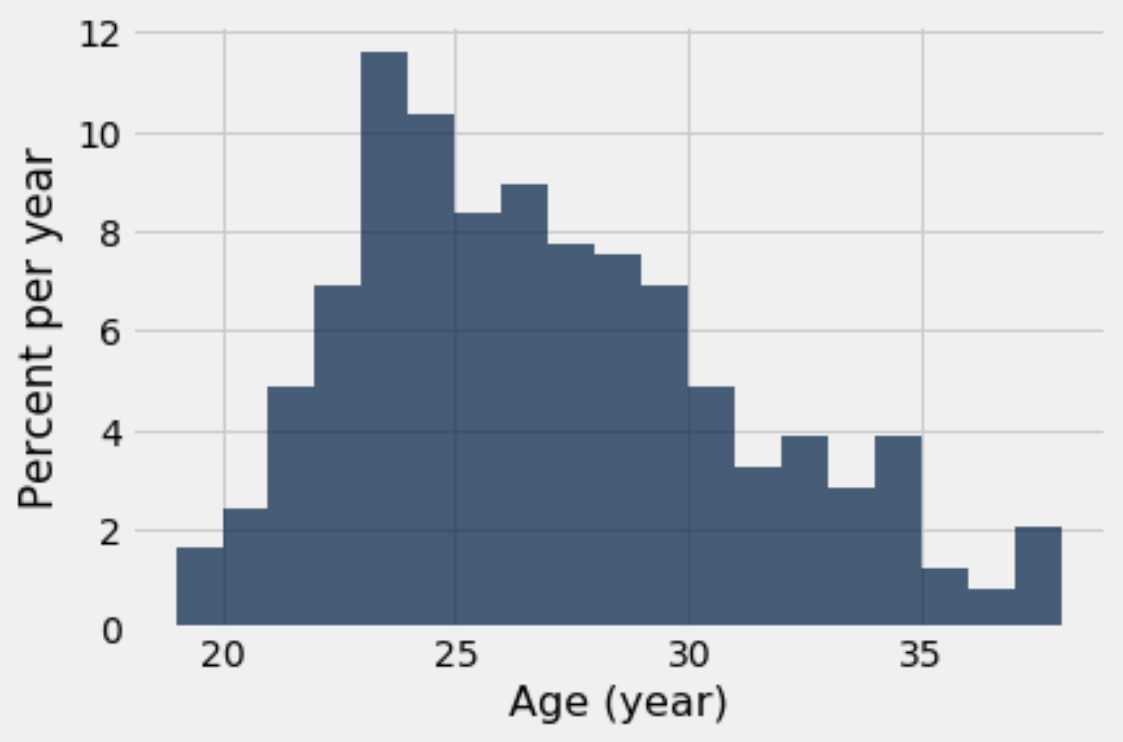


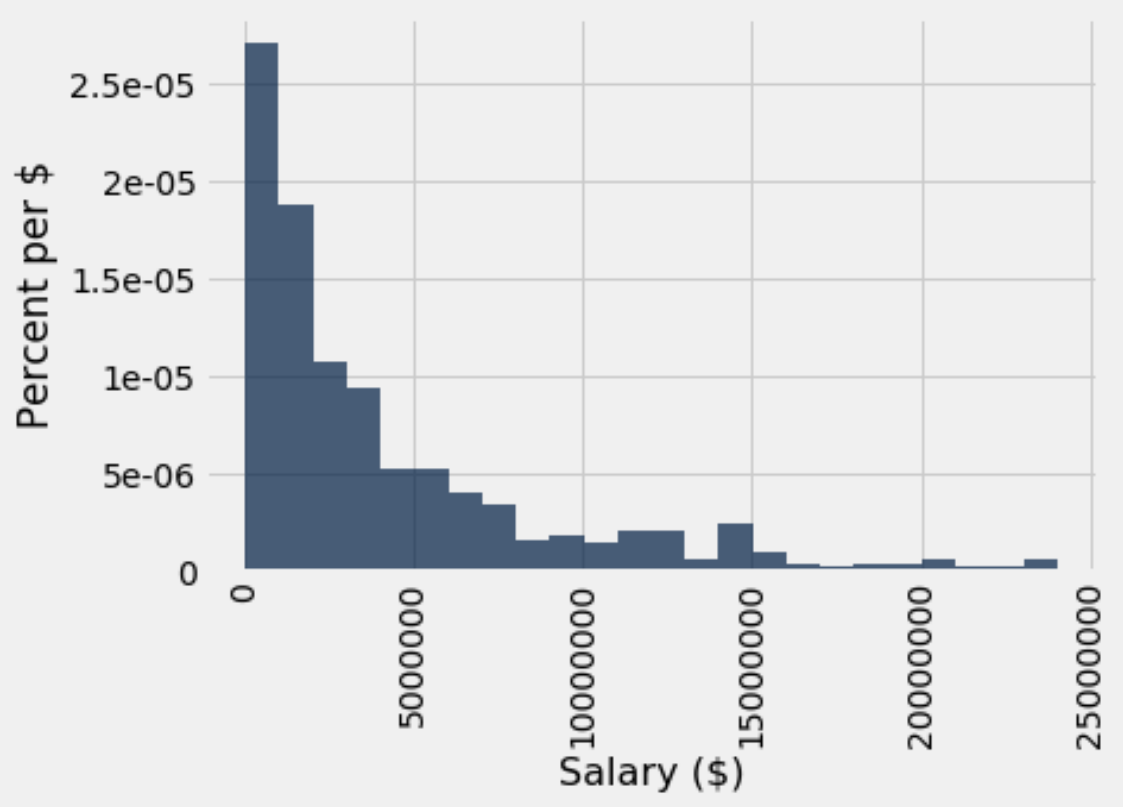


```
In [86]: _ = ok.grade('q2_2') # Warning: Charts will be displayed while running this test

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```




2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)



1.2.1 Convenience sampling

One sampling methodology, which is **generally a bad idea**, is to choose players who are somehow convenient to sample. For example, you might choose players from one team that's near your house, since it's easier to survey them. This is called, somewhat pejoratively, *convenience sampling*.

Suppose you survey only *relatively new* players with ages less than 22. (The more experienced players didn't bother to answer your surveys about their salaries.)

Question 2.3 Assign `convenience_sample_data` to a subset of `full_data` that contains only the rows for players under the age of 22.

```
In [87]: convenience_sample = full_data.where('Age', are.below(22))
         convenience_sample
```

```
Out[87]: PlayerName      | Salary  | Age  | Team | Games | Rebounds | Assists | Steals | Blocks
         Aaron Gordon    | 3992040 | 19   | ORL  | 47    | 169      | 33      | 21     | 22
         Alex Len        | 3649920 | 21   | PHO  | 69    | 454      | 32      | 34     | 105
         Andre Drummond  | 2568360 | 21   | DET  | 82    | 1104     | 55      | 73     | 153
         Andrew Wiggins  | 5510640 | 19   | MIN  | 82    | 374      | 170     | 86     | 50
         Anthony Bennett | 5563920 | 21   | MIN  | 57    | 216      | 48      | 27     | 16
```

```
Anthony Davis   | 5607240 | 21   | NOP  | 68   | 696      | 149     | 100   | 200
Archie Goodwin  | 1112280 | 20   | PHO  | 41   | 74       | 44      | 18    | 9
Ben McLemore    | 3026280 | 21   | SAC  | 82   | 241      | 140     | 77    | 19
Bradley Beal    | 4505280 | 21   | WAS  | 63   | 241      | 194     | 76    | 18
Bruno Caboclo   | 1458360 | 19   | TOR  | 8    | 2        | 0       | 0     | 1
... (34 rows omitted)
```

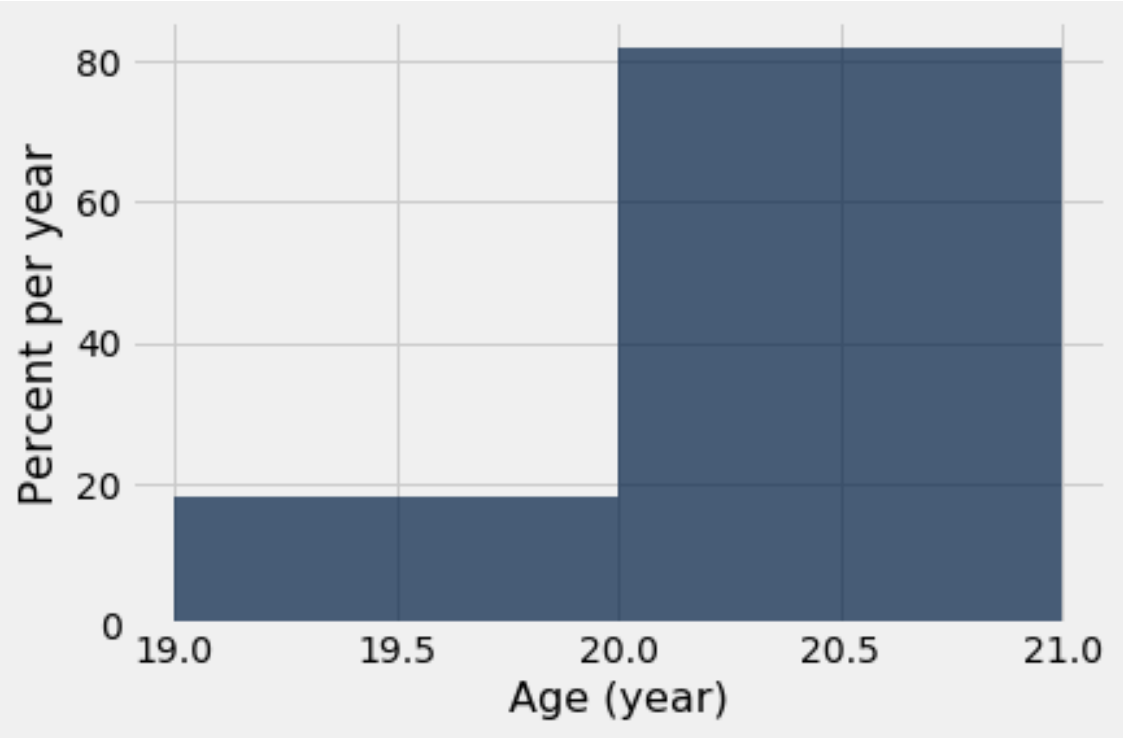
```
In [88]: _ = ok.grade('q2_3')
```

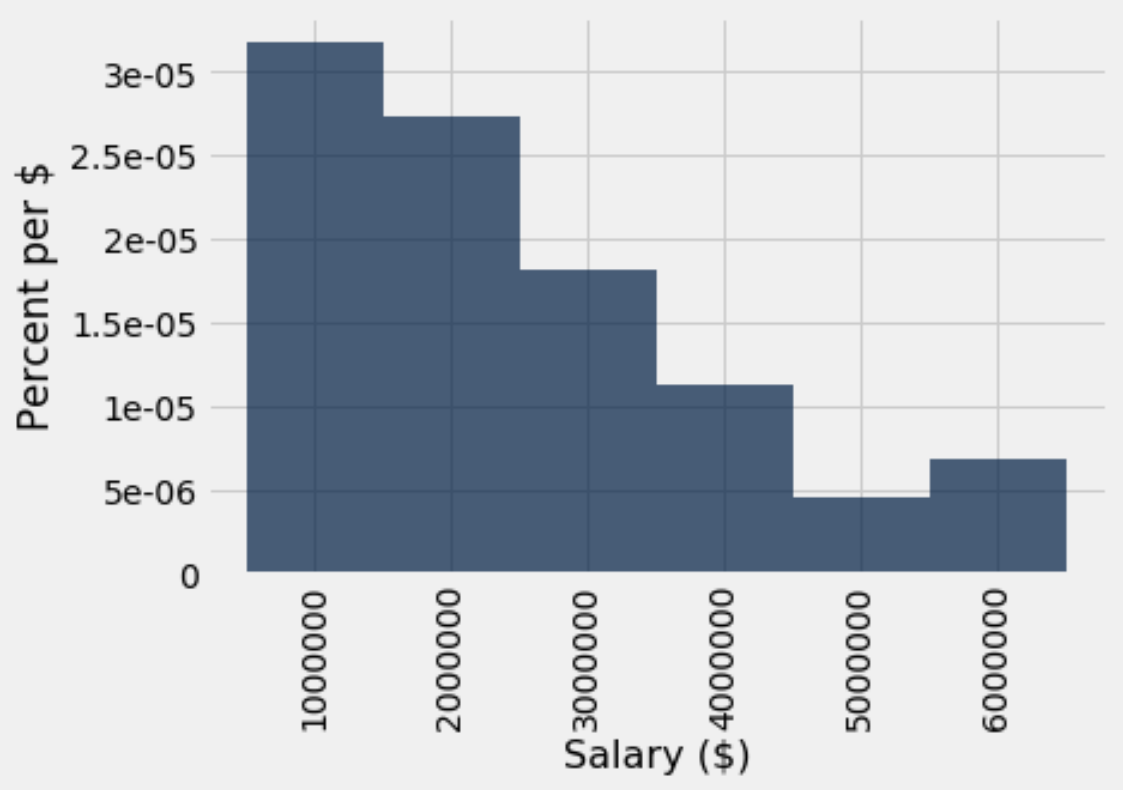
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 2.4 Assign `convenience_stats` to an array of the average age and average salary of your convenience sample, using the `compute_statistics` function. Since they're computed on a sample, these are called *sample averages*.

```
In [89]: convenience_stats = compute_statistics(convenience_sample)
         convenience_stats
```

```
Out[89]: array([2.03636364e+01, 2.38353382e+06])
```



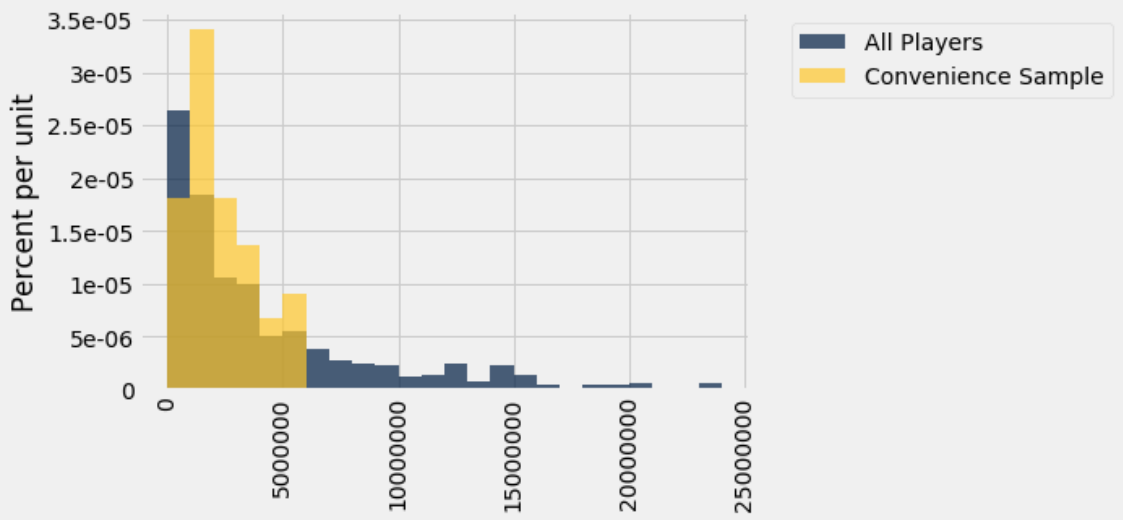



```
In [90]: _ = ok.grade('q2_4')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooo] 100.0% passed
```

Next, we'll compare the convenience sample salaries with the full data salaries in a single histogram. To do that, we'll need to use the `bin_column` option of the `hist` method, which indicates that all columns are counts of the bins in a particular column. The following cell should not require any changes; just run it.

```
In [91]: def compare_salaries(first, second, first_title, second_title):
             """Compare the salaries in two tables."""
```

```
    max_salary = max(np.append(first.column('Salary'), second.column('Salary')))
    bins = np.arange(0, max_salary+1e6+1, 1e6)
    first_binned = first.bin('Salary', bins=bins).relabeled(1, first_title)
    second_binned = second.bin('Salary', bins=bins).relabeled(1, second_title)
    first_binned.join('bin', second_binned).hist(bin_column='bin')

compare_salaries(full_data, convenience_sample, 'All Players', 'Convenience Sample')
```



Question 2.5 Does the convenience sample give us an accurate picture of the age and salary of the full population of NBA players in 2014-2015? Would you expect it to, in general? Assign either 1, 2, 3, or 4 to the variable `sampling_q5` below. 1. Yes. The sample is large enough, so it is an accurate representation of the population. 2. No. The sample is too small, so it won't give us an accurate representation of the population. 3. No. But this was just an unlucky sample, normally this would give us an accurate representation of the population. 4. No. This type of sample doesn't give us an accurate representation of the population.

```
In [92]: sampling_q5 = 4

In [93]: _ = ok.grade('q2_5')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooook] 100.0% passed
```

1.2.2 Simple random sampling

A more principled approach is to sample uniformly at random from the players. If we ensure that each player is selected at most once, this is a *simple random sample without replacement*, sometimes abbreviated to "simple random sample" or "SRSWOR". Imagine writing down each player's name on a card, putting the cards in a hat, and shuffling the hat. Then, pull out cards one by one and set them aside, stopping when the specified *sample size* is reached.

We've produced two samples of the `salary_data` table in this way: `small_srswor_salary.csv` and `large_srswor_salary.csv` contain, respectively, a sample of size 44 (the same as the convenience sample) and a larger sample of size 100.

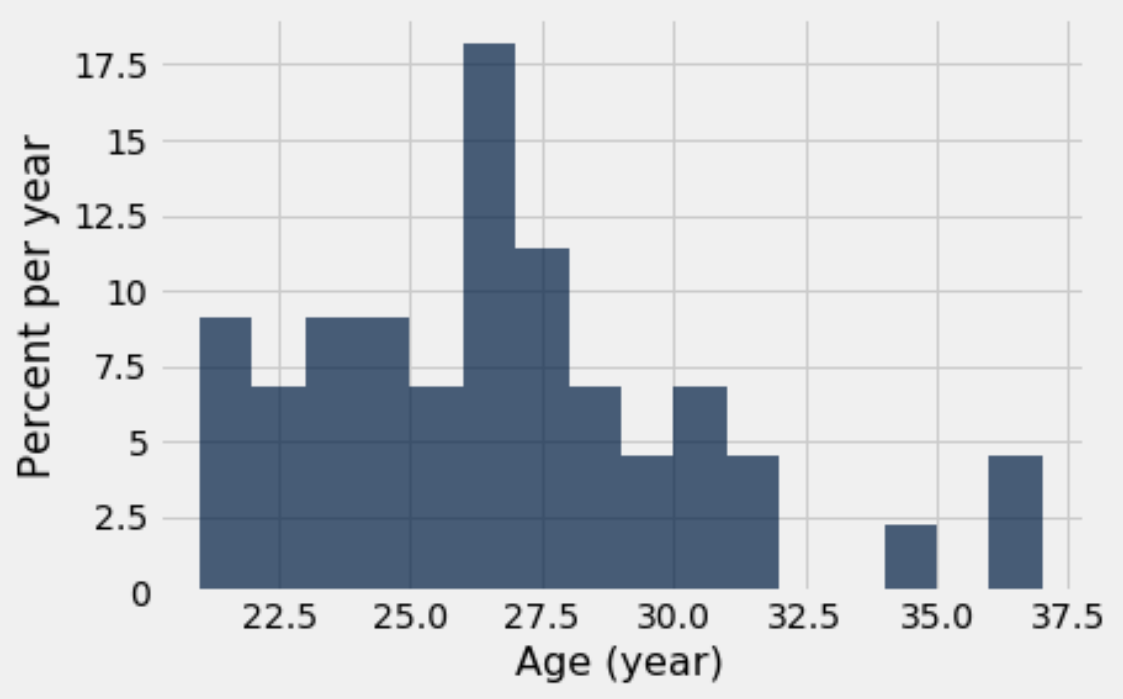
The `load_data` function below loads a salary table and joins it with `player_data`.

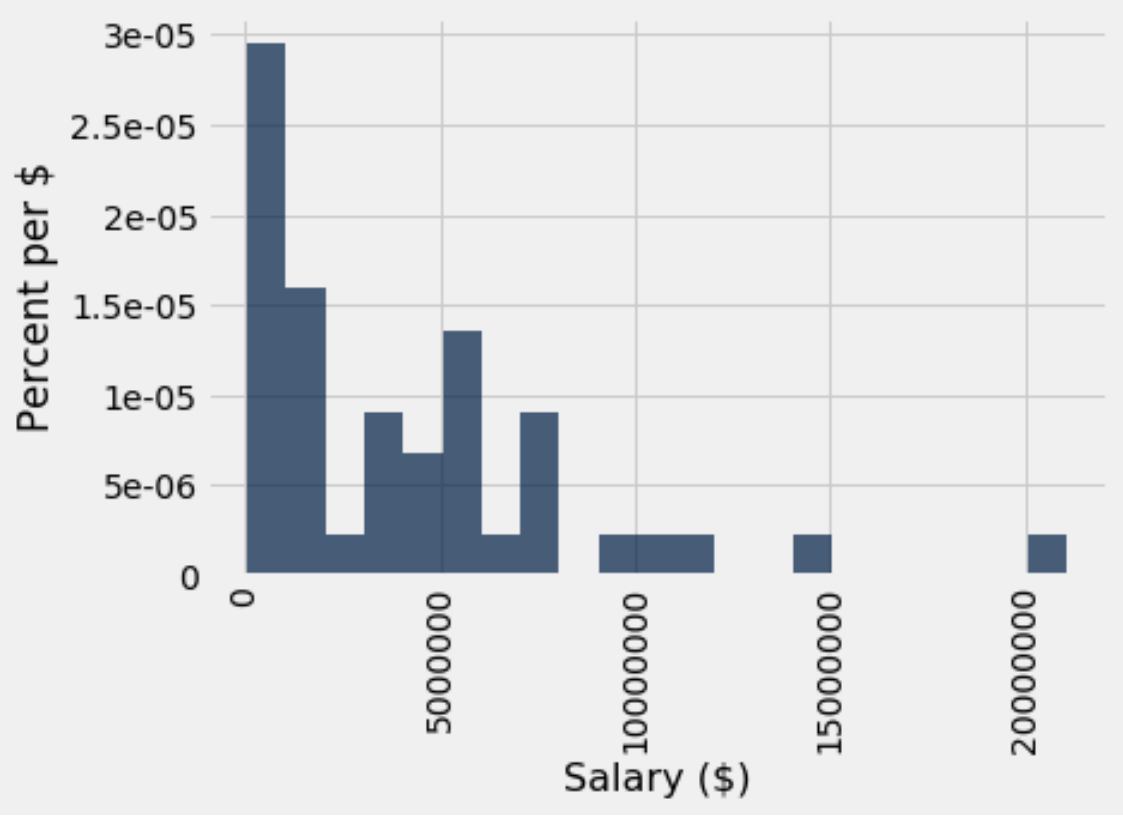
```
In [94]: def load_data(salary_file):
             return player_data.join('Name', Table.read_table(salary_file), 'PlayerName')
```

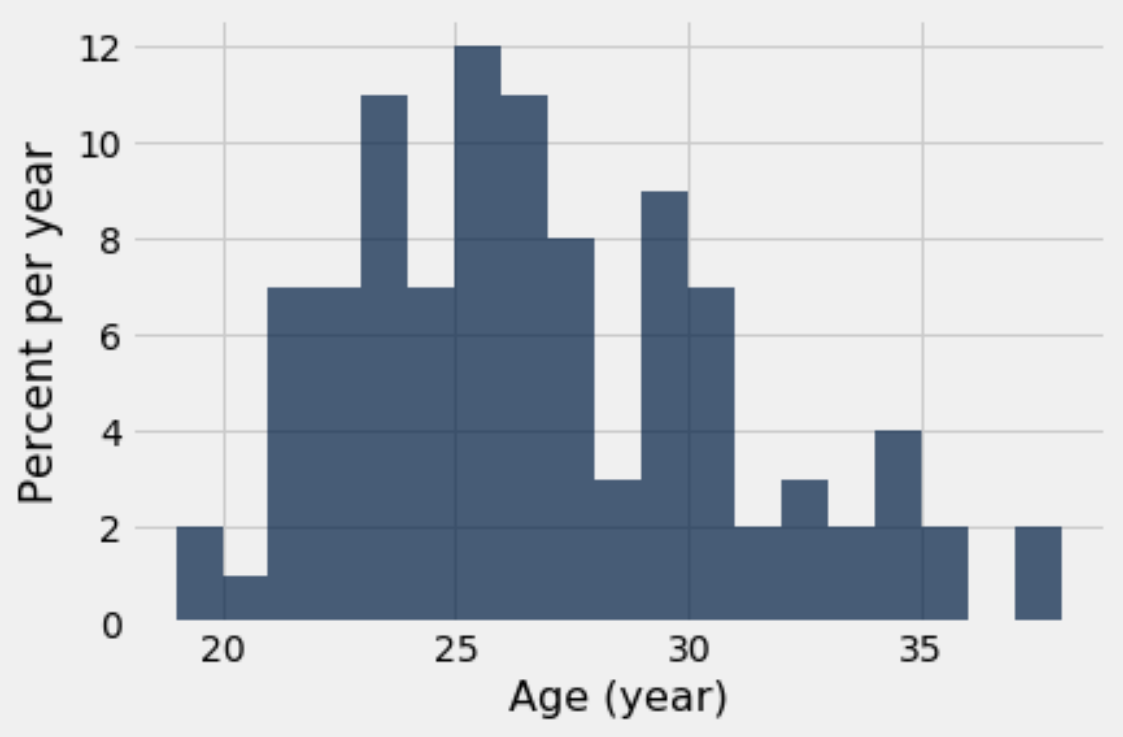
Question 2.6 Run the same analyses on the small and large samples that you previously ran on the full dataset and on the convenience sample. Compare the accuracy of the estimates of the population statistics that we get from the small simple random sample, the large simple random sample, and the convenience sample.

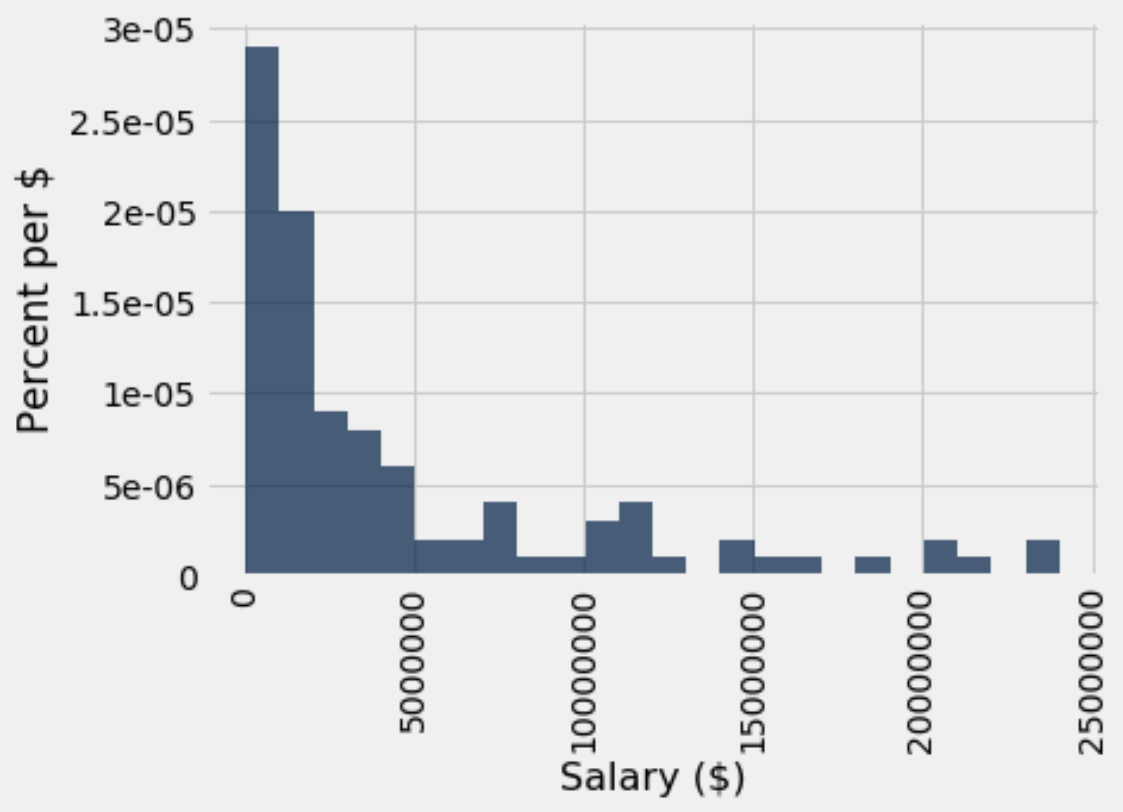
```
In [95]: # Original:
         small_srswor_data = load_data('small_srswor_salary.csv')
         small_stats = compute_statistics(small_srswor_data)
         large_srswor_data = load_data('large_srswor_salary.csv')
         large_stats = compute_statistics(large_srswor_data)
         convenience_stats = compute_statistics(convenience_sample)
         print('Full data stats:                ', full_stats)
         print('Small simple random sample stats:', small_stats)
         print('Large simple random sample stats:', large_stats)
         print('Convenience sample stats:       ', convenience_stats)
```

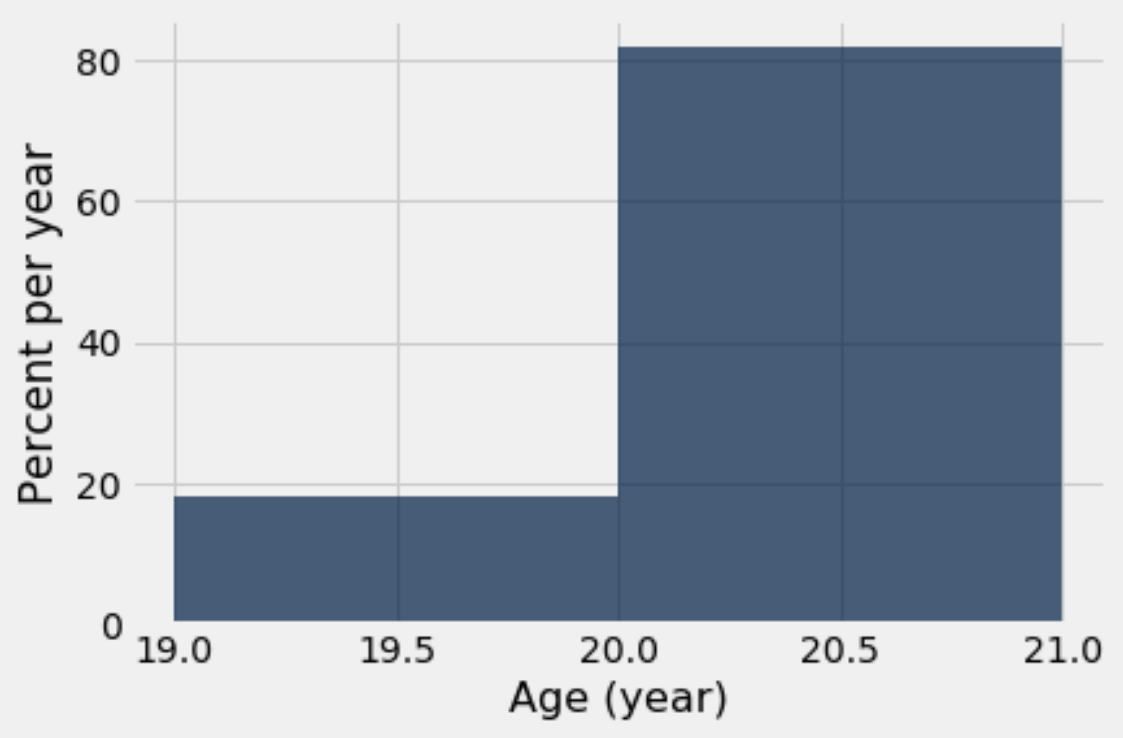
```
Full data stats:                 [2.65365854e+01 4.26977577e+06]
Small simple random sample stats: [2.63181818e+01 4.28391089e+06]
Large simple random sample stats: [2.6420000e+01 4.8213225e+06]
Convenience sample stats:        [2.03636364e+01 2.38353382e+06]
```

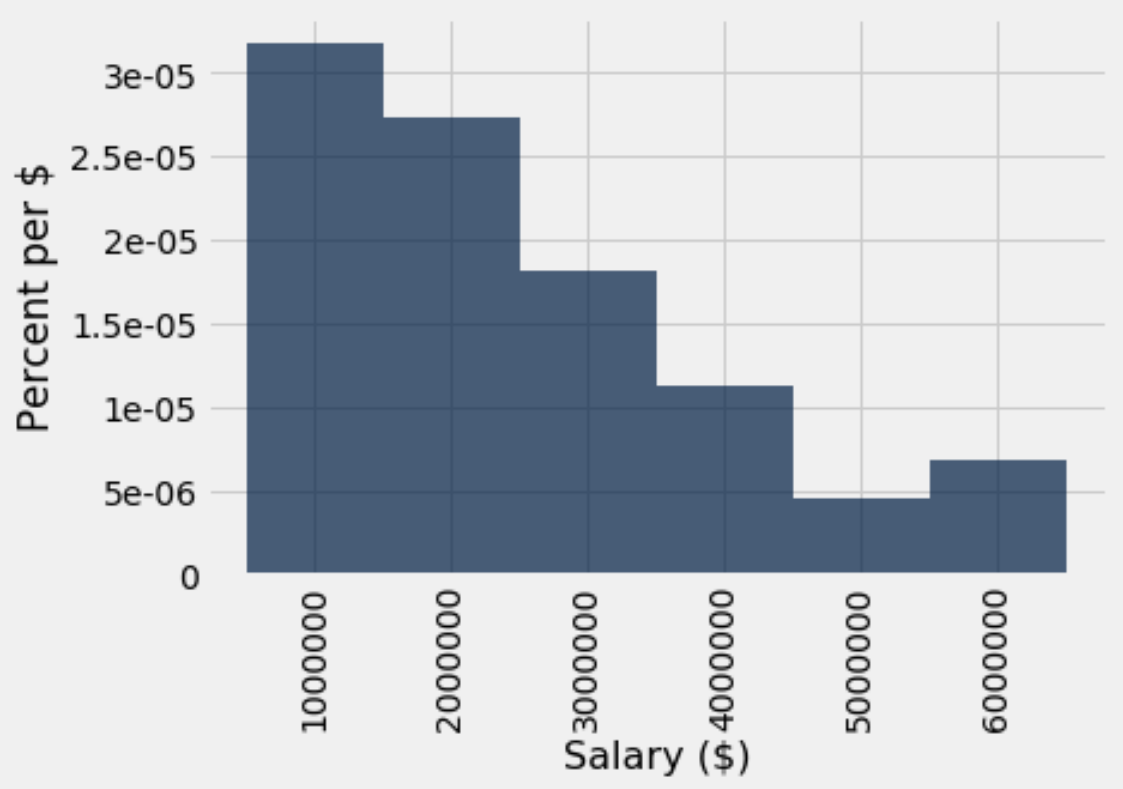

17.5
15
12.5
10
7.5
5
2.5
0
Percent per year
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Age (year)


3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
50000000
100000000
150000000
200000000
Salary ($)


12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)


3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


Percent per year
80
60
40
20
0
19.0
19.5
20.0
20.5
21.0
Age (year)


Percent per $
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
1000000
2000000
3000000
4000000
5000000
6000000
Salary ($)

```
In [96]: _ = ok.grade('q2_6')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooo] 100.0% passed
```

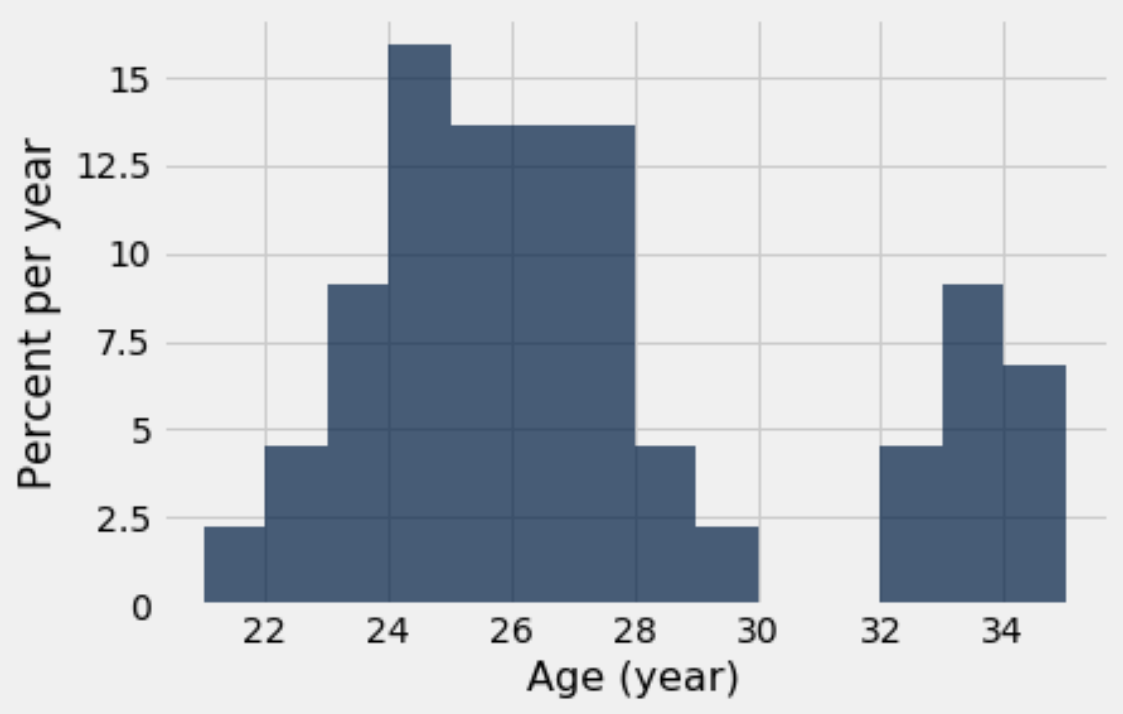
1.2.3 Producing simple random samples

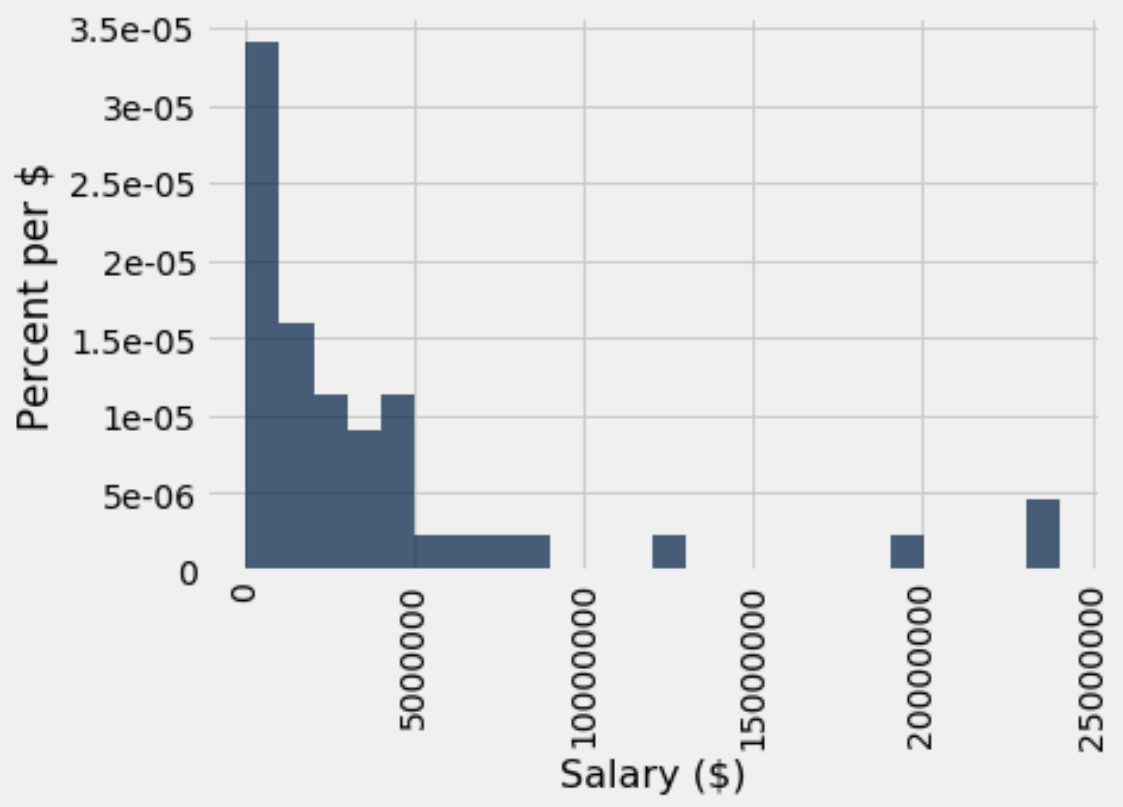
Often it's useful to take random samples even when we have a larger dataset available. The randomized response technique was one example we saw in lecture. Another is to help us understand how inaccurate other samples are.

Tables provide the method `sample()` for producing random samples. Note that its default is to sample with replacement. To see how to call `sample()`, search the documentation on `data8.org/datascience`, or enter `full_data.sample?` into a code cell and press Enter.

Question 2.7 Produce a simple random sample of size 44 from `full_data`. (You don't need to bother with a join this time —- just use `full_data.sample(...)` directly. That will have the same result as sampling from `salary_data` and joining with `player_data`.) Run your analysis on it again.

```
In [97]: my_small_srswor_data = full_data.sample(44)
         my_small_stats = compute_statistics(my_small_srswor_data)
         my_small_stats

Out[97]: array([2.67500000e+01, 4.02681923e+06])
```


Percent per year
15
12.5
10
7.5
5
2.5
0
22
24
26
28
30
32
34
Age (year)


Percent per $
3.5e-05
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

Are your results similar to those in the small sample we provided you? Do things change alot across separate samples? Run your code several times to get new samples. Assign either 1, 2, 3, or 4 to the variable `sampling_q7` below. 1. The results are very different from the small sample, and don't change at all across separate samples. 2. The results are very different from the small sample, and change a bit across separate samples. 3. The results are slightly different from the small sample, and change a bit across separate samples. 4. The results are not at all different from the small sample, and don't change at all across separate samples.

```
In [98]: sampling_q7 = 3

In [99]: _ = ok.grade('q2_7')

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 2.8 As in the previous question, analyze several simple random samples of size 100 from `full_data`.

```
In [100]: my_large_srswor_data = ...
          my_large_stats = ...
          my_large_stats

Out[100]: Ellipsis
```

Do the average and histogram statistics seem to change more or less across samples of this size than across samples of size 44? And are the sample averages and histograms closer to their true values for age or for salary? Assign either 1, 2, 3, 4, or 5 to the variable `sampling_q8` below.

Is this what you expected to see? 1. The statistics change *less* across samples of this size than across smaller samples. The statistics are closer to their true values for *age* than they are for *salary*. 2. The statistics change *less* across samples of this size than across smaller samples. The statistics are closer to their true values for *salary* than they are for *age*. 3. The statistics change *more* across samples of this size than across smaller samples. The statistics are closer to their true values for *age* than they are for *salary*. 4. The statistics change *more* across samples of this size than across smaller samples. The statistics are closer to their true values for *salary* than they are for *age*. 5. The statistics change an *equal amount* across samples of this size as across smaller samples. The statistics for age and salary are *equally close* to their true values.

```
In [101]: sampling_q8 = 1

In [102]: _ = ok.grade('q2_8')
```

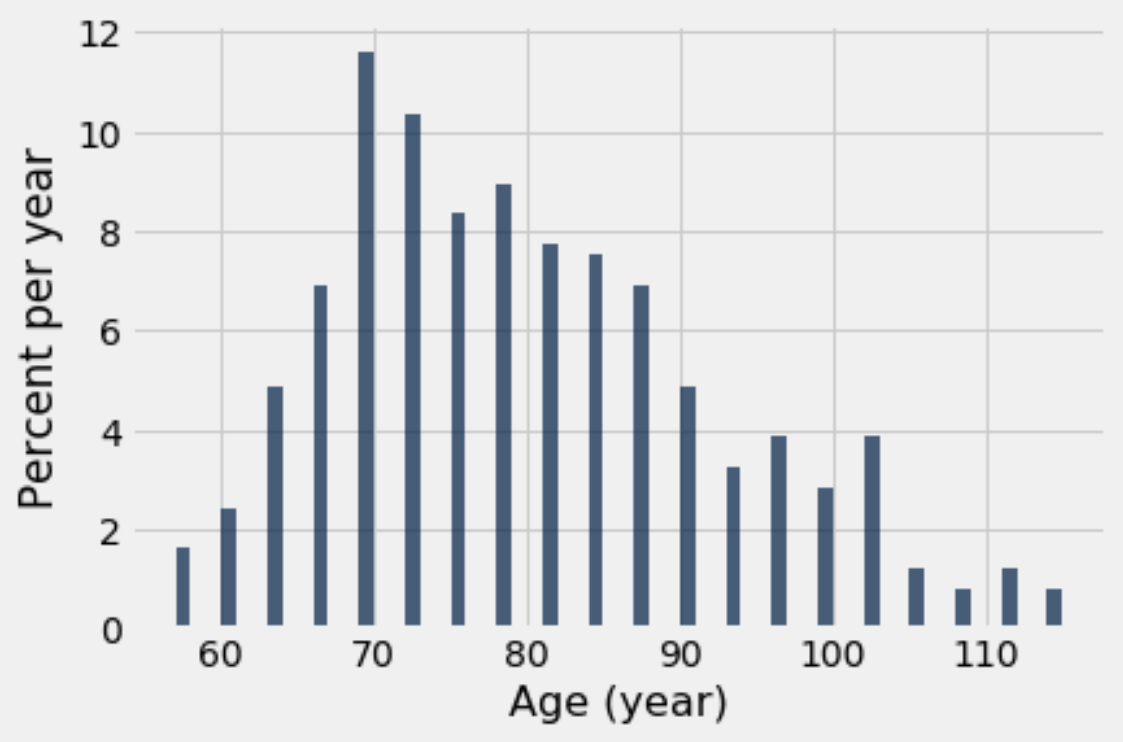
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 5
    Failed: 0
[ooooooooook] 100.0% passed
```

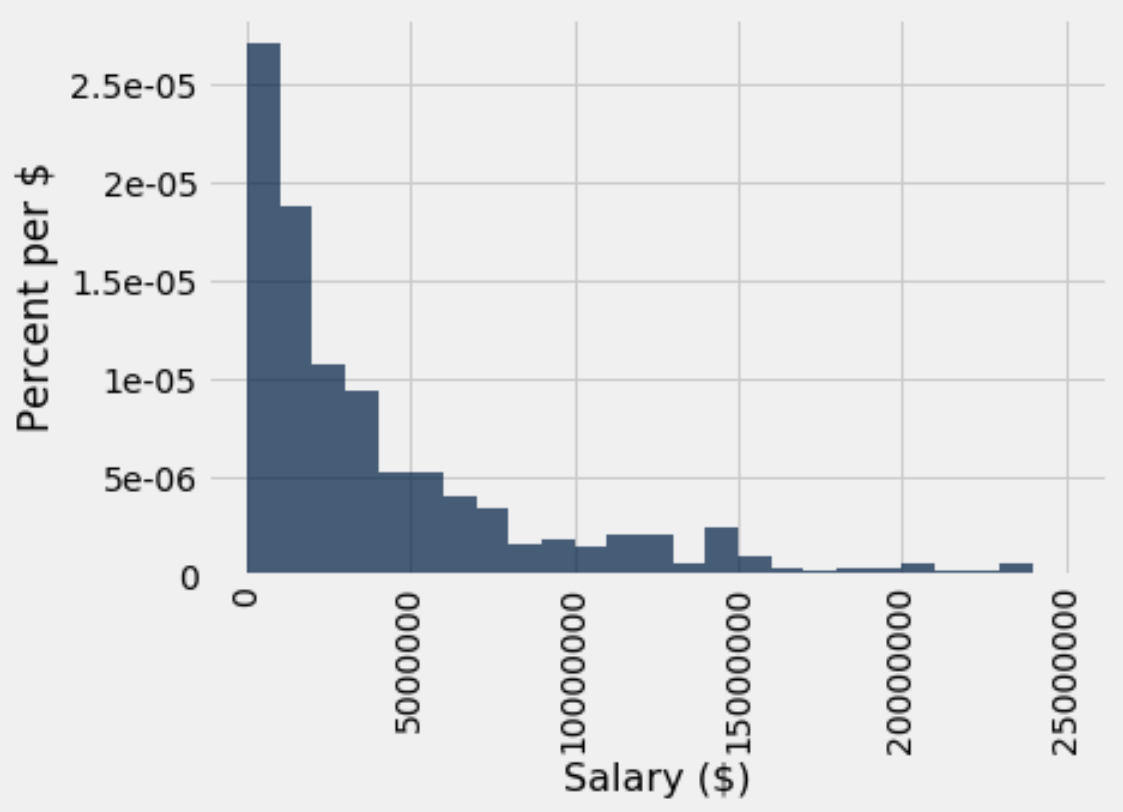
```
In [103]: # For your convenience, you can run this cell to run all the tests at once!
          import os
          _ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q')]
```

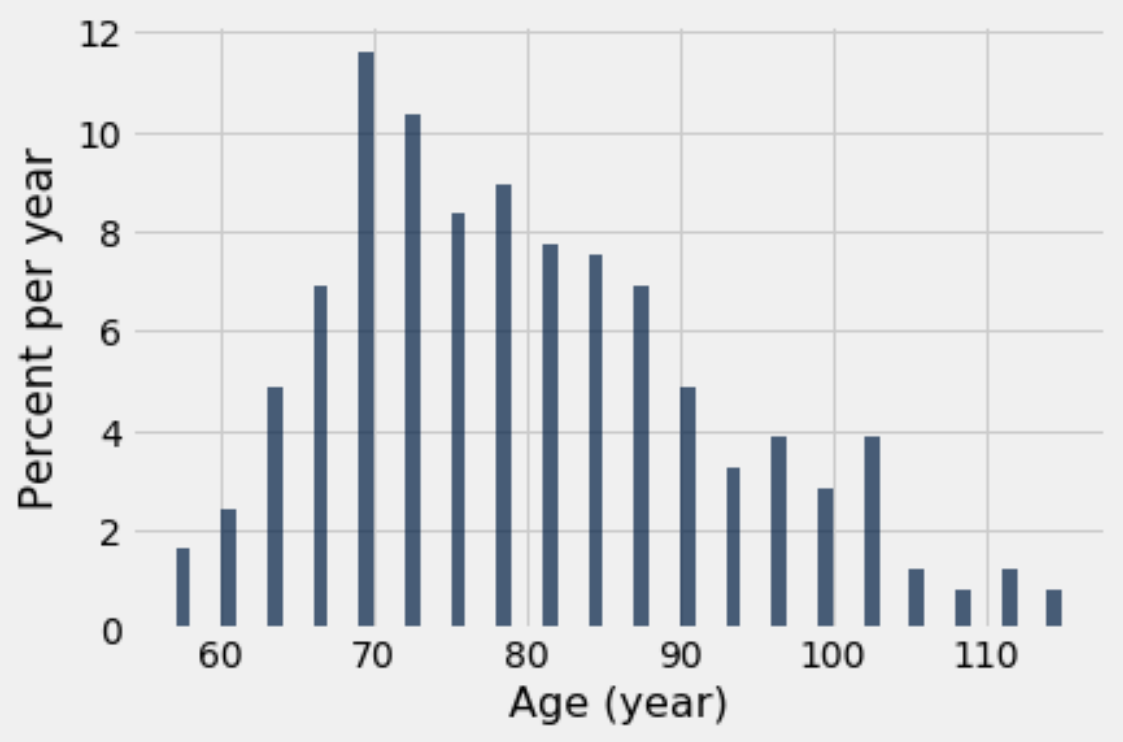
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
```

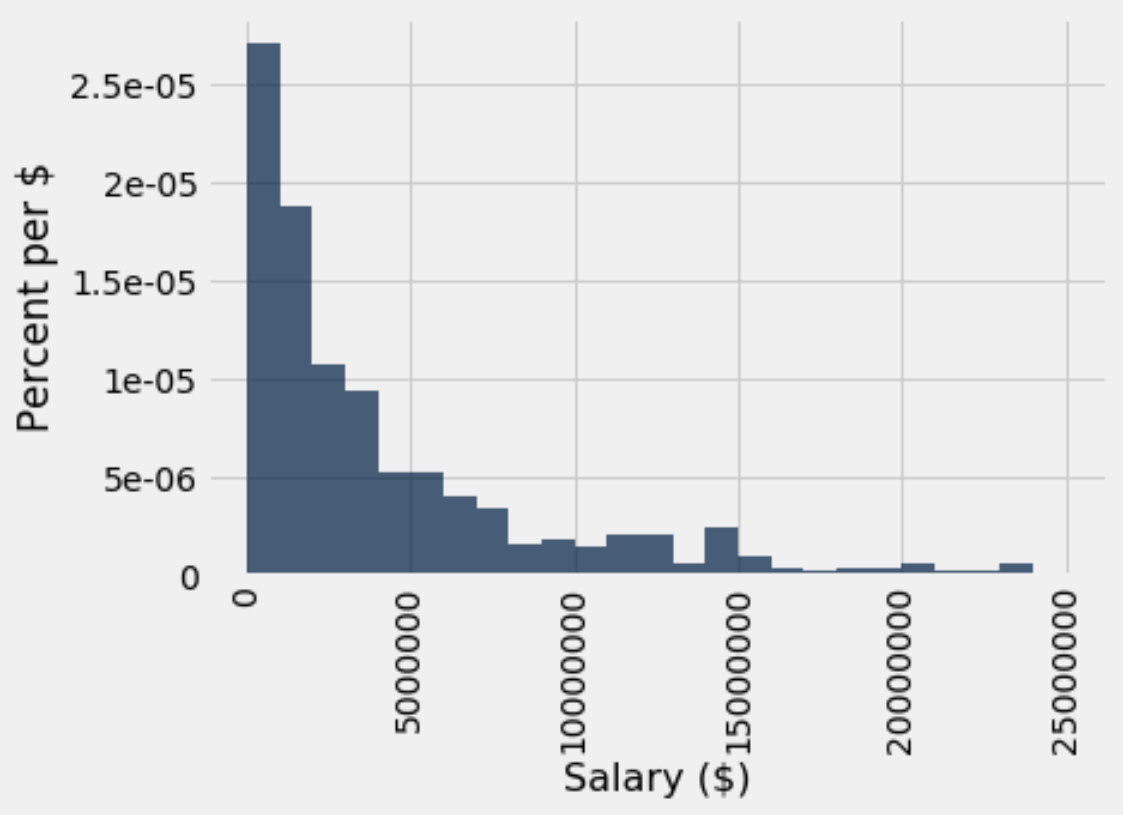
```
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

------------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

------------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

------------------------------------------------------------------------
Test summary
    Passed: 5
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

------------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

------------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

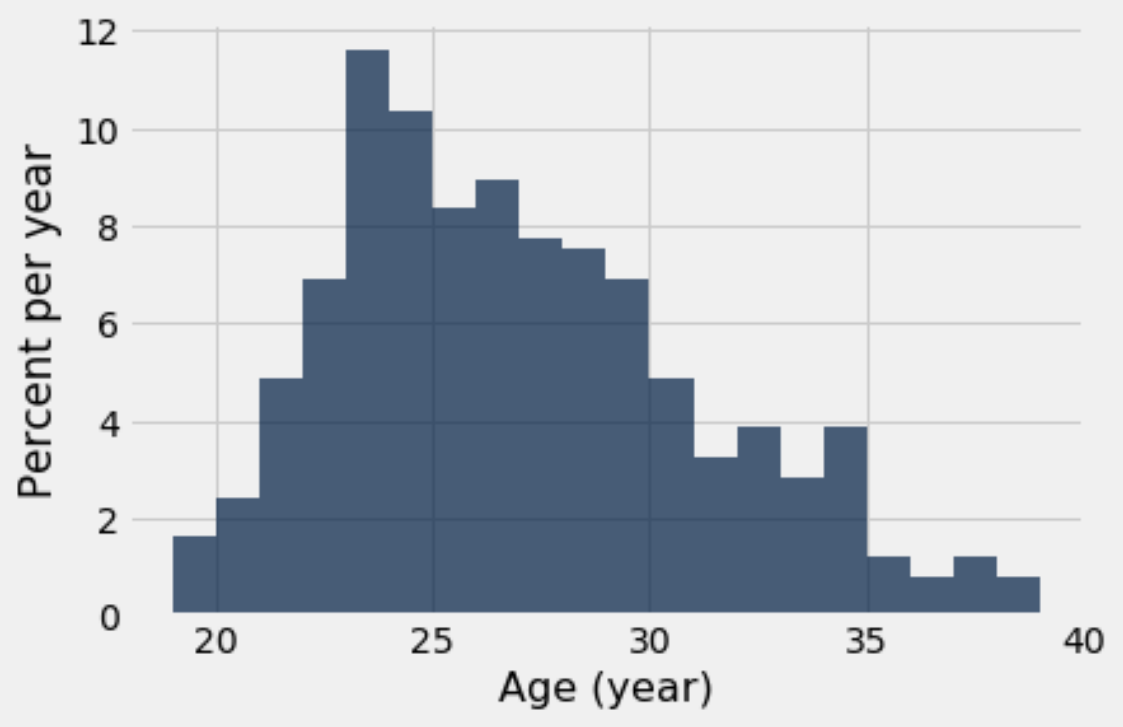
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed
```

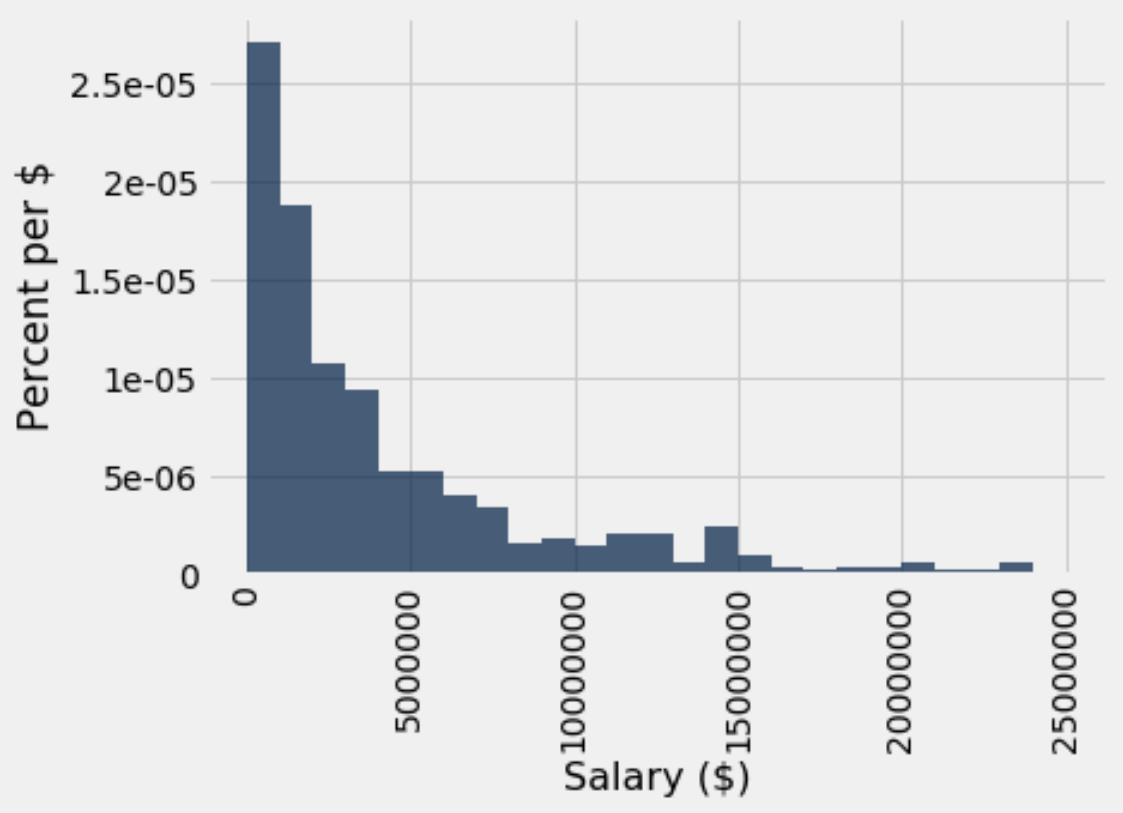

Percent per year
12
10
8
6
4
2
0
60
70
80
90
100
110
Age (year)

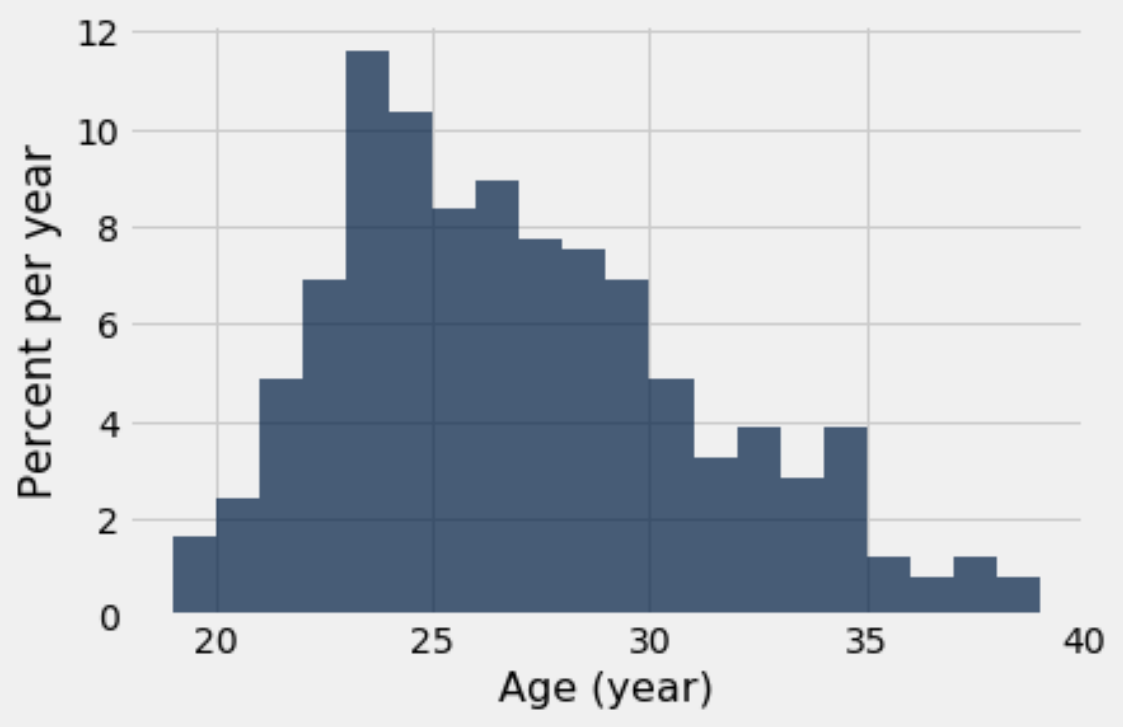

Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

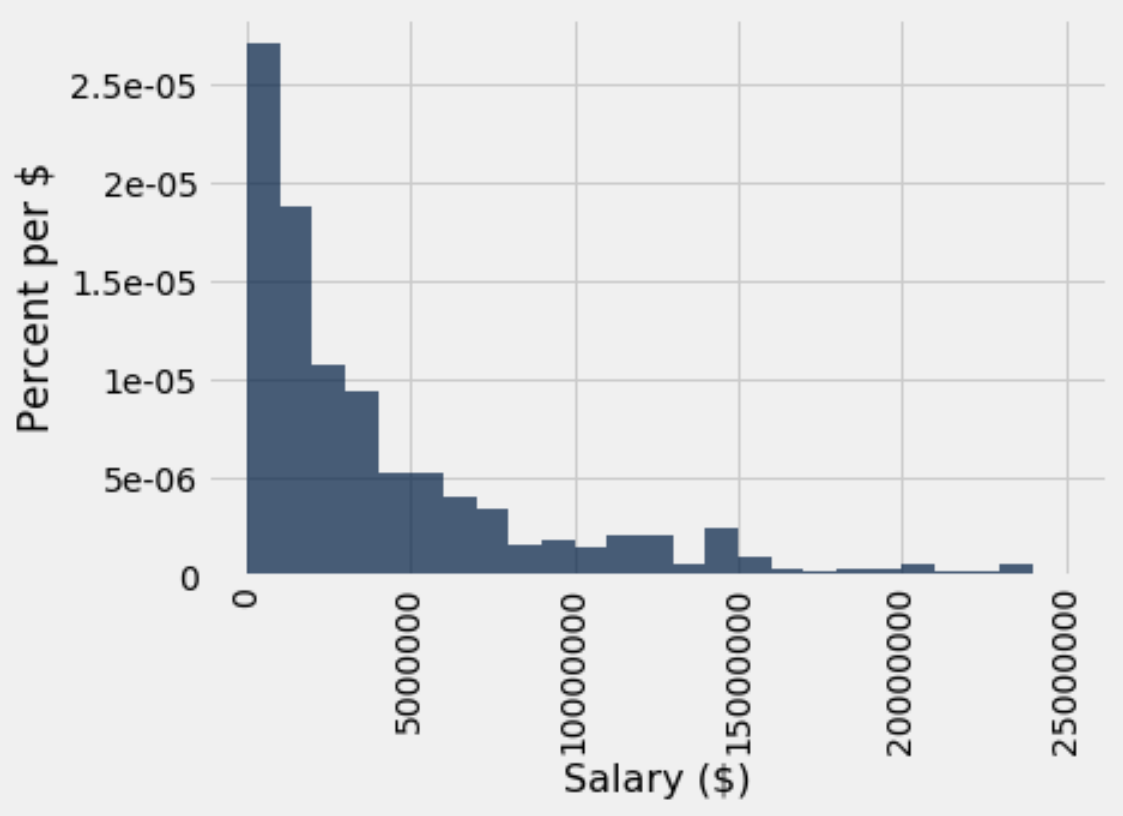

12
10
8
6
4
2
0
Percent per year
60
70
80
90
100
110
Age (year)

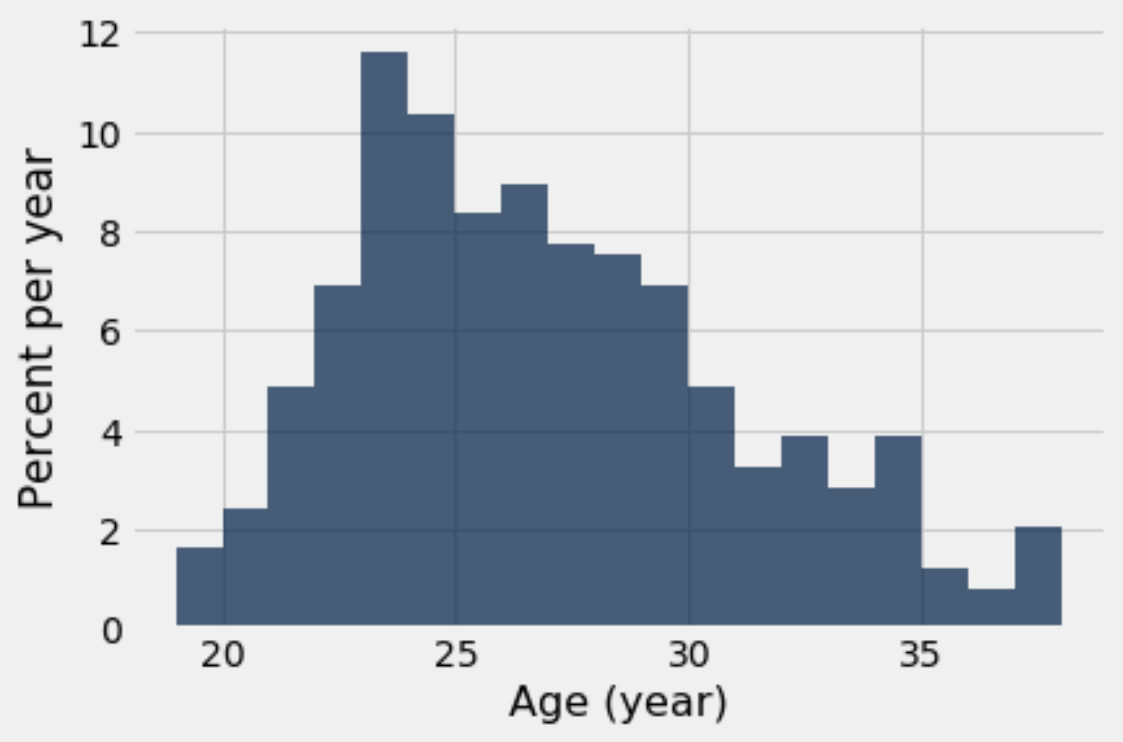

2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

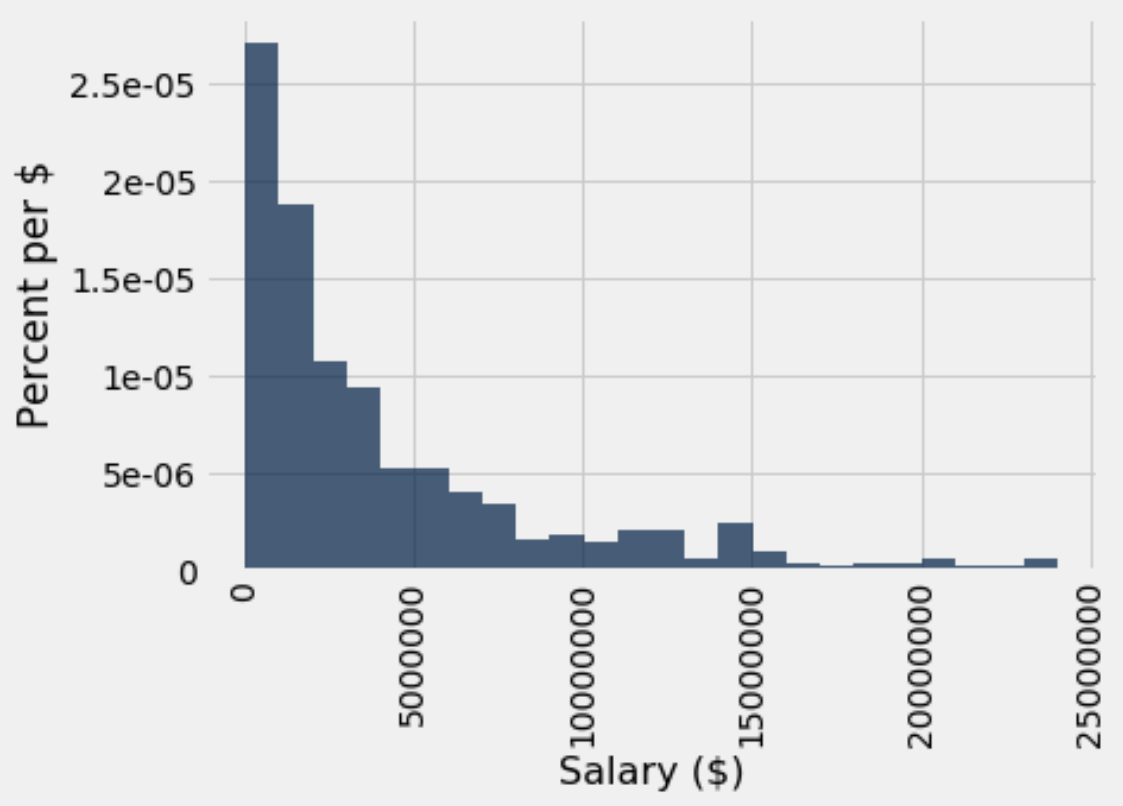

12
10
8
6
4
2
0
Percent per year
20
25
30
35
40
Age (year)

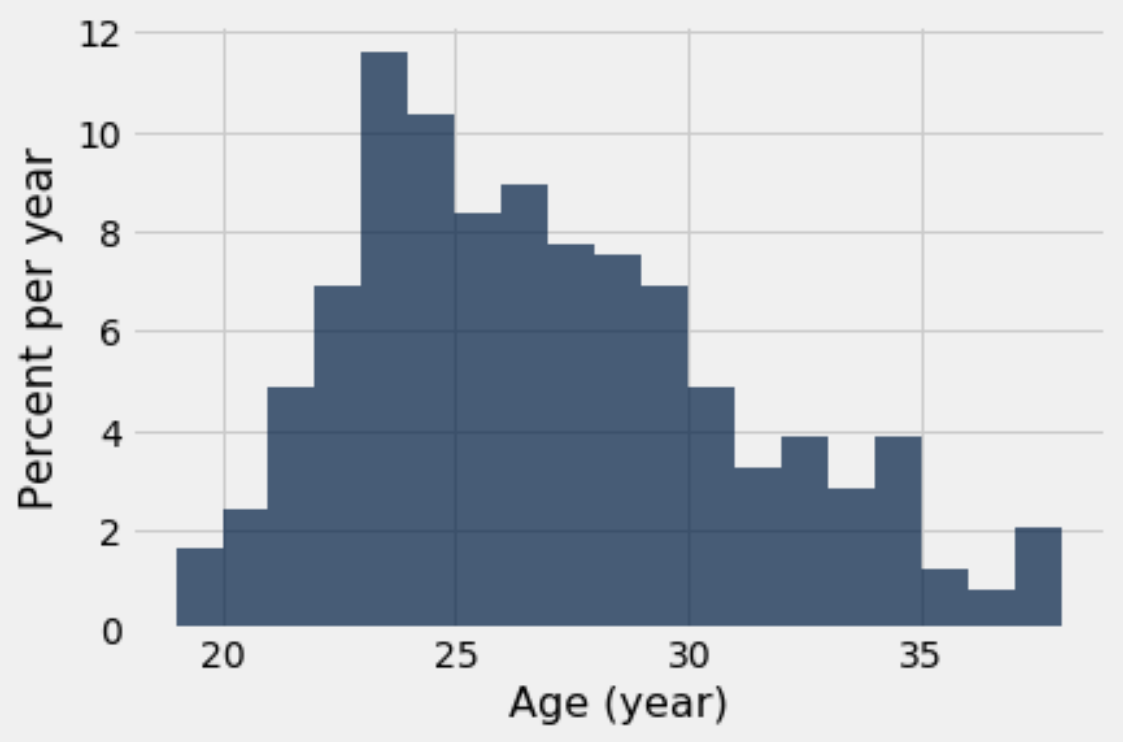

2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)

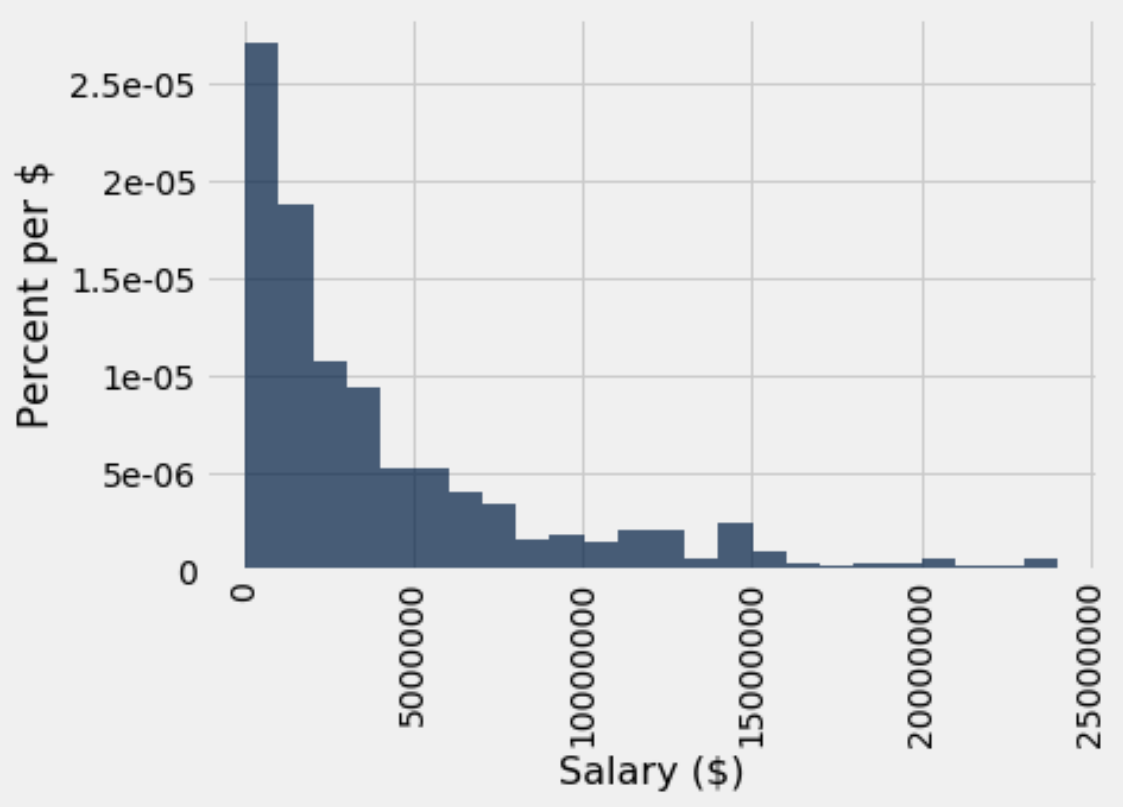

12
10
8
6
4
2
0
Percent per year
20
25
30
35
40
Age (year)


2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)


2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
25000000
Salary ($)


12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)




```
In [104]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Saving notebook... Saved 'lab06.ipynb'.
Submit... 100% complete
Submission successful for user: wec149@ucsd.edu
URL: https://okpy.org/ucsd/dsc10/fa18/lab06/submissions/VPEPPo
```