lab07

April 17, 2022

Name: Allan Gongora

Section: 0131

1 Lab 7: Sampling

Welcome to Lab 7!

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.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Requirement already satisfied: gofer-grader in /opt/conda/lib/python3.7/site-
packages (1.1.0)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# 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 gofer.ok import check
```

Recommended Reading: * Sampling and Empirical Distributions

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
    A) Sentence reponses to questions that ask for an explanation
    B) Numeric responses to multiple choice questions
    C) Programming code
2) Moreover, throughout this lab and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. Dungeons & 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 his/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.

[3]:
```
from random import randint, choice
```

```
[4]: possible_rolls = np.arange(1, 21)
     roll_result = choice(possible_rolls)
     modified_result = roll_result + 11
     action_succeeded = (randint(1, 20) + 11) > 15
     # the top 3 are garbage so i did it all in one line but i had to do it your way␣
      ↪since
     # the code and tests expect that variable.

     # 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,␣
      ↪"succeeded" if action_succeeded else "failed"))
```

```
On a modified roll of 31, Alice's action succeeded.
```

```
[5]: check('tests/q1_1.py')
```

```
[5]: <gofer.ok.OKTestsResult at 0x7f24199a6c90>
```

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.

```
[6]: rough_success_chance = sum([(randint(1, 20) + 11) > 15 for _ in range(7)]) / 7␣
      ↪# not sure how to not simulate it
     rough_success_chance
```

```
[6]: 0.8571428571428571
```

```
[7]: check('tests/q1_2.py')
```

```
[7]: <gofer.ok.OKTestsResult at 0x7f2403ab0dd0>
```

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`.

```
[8]: modifier = 11
     num_observations = 7

     def simulate_observations():
         """Produces an array of 7 simulated modified die rolls"""
         return np.array([randint(1, 20) + modifier for _ in␣
      ↪range(num_observations)])
```

```
observations = simulate_observations()
observations
```

```
[8]: array([23, 14, 27, 19, 15, 27, 24])
```

```
[9]: check('tests/q1_3.py')
```

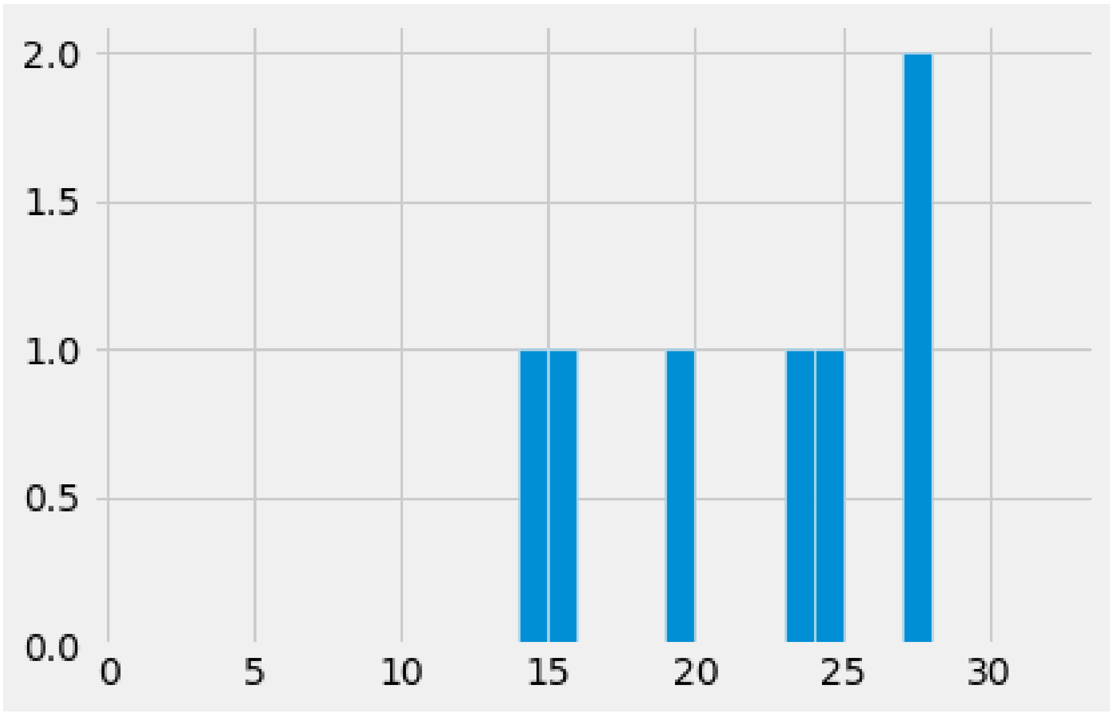
```
[9]: <gofer.ok.OKTestsResult at 0x7f241995b710>
```

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

Question 1.4 does not have an autograder test, so it is not graded and not in the overall lab grade.

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

      fig, ax = plots.subplots()

      ax.hist(observations, roll_bins)

      np.arange(1+modifier, 20+modifier+1)
```

```
[10]: array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
             29, 30, 31])
```



Your histogram should have values 12 to 31, each with a probability of 5%.

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

One straightforward (but clearly suboptimal) way to do that is to find the *smallest* total roll, since the smallest roll on a 20-sided die is 1.

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

```
[11]: ((observations.min() - 1) + (observations.max() - 20)) / 2
```

```
[11]: 10.0
```

```
[12]: min_estimate = observations.min() - 1
      min_estimate
```

```
[12]: 13
```

```
[13]: check('tests/q1_5.py')
```

```
[13]: <gofer.ok.OKTestsResult at 0x7f2401893950>
```

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.

```
[14]: def mean_based_estimator(nums):
          """Estimate the roll modifier based on observed modified rolls in the array␣
      ↪nums."""
          return ((nums.min() - 1) + (nums.max() - 20)) / 2

      # 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
```

```
[14]: 10.0
```

```
[15]: check('tests/q1_6.py')
```

```
[15]: <gofer.ok.OKTestsResult at 0x7f2401810690>
```

1.2 2. Sampling Basketball Data

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

[16]:
```
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, unlike, say, logical 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`).

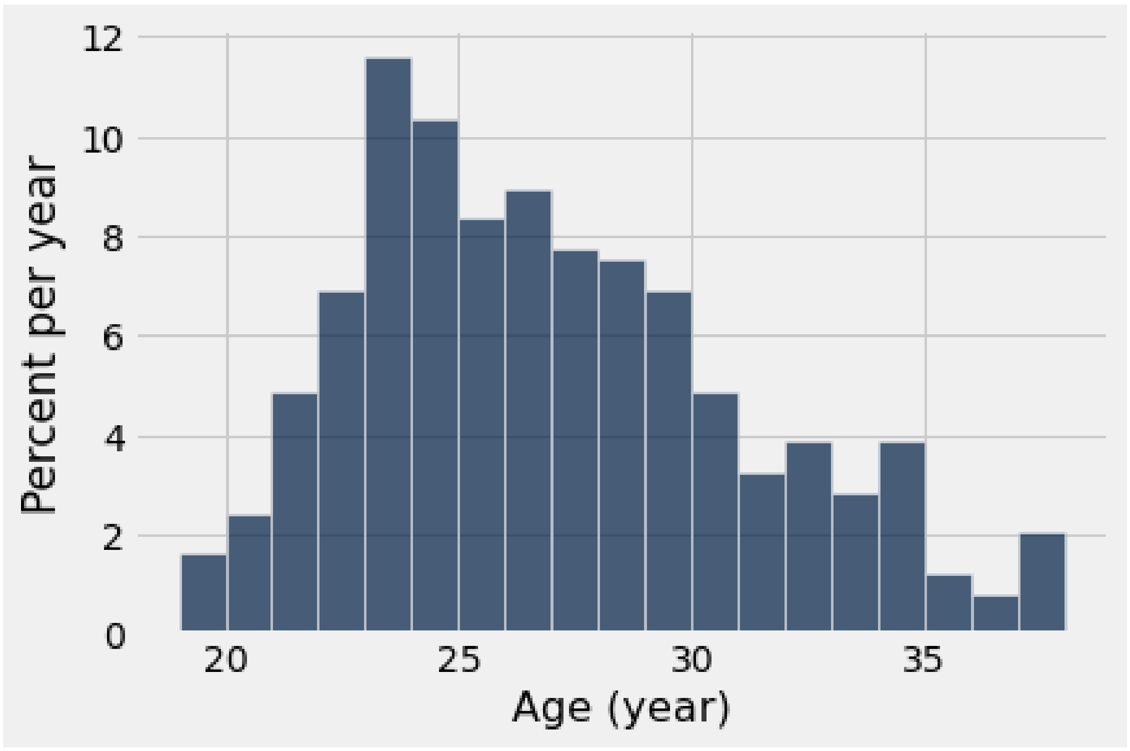
Hint: When creating the bins for the histograms, think critically about what the stop argument should be for `np.arange`. Histograms are inclusive on the left-hand side of the interval, but not the right. So, if we have a maximum age of 80, we need a 80-81 bin in order to capture this in the histogram.

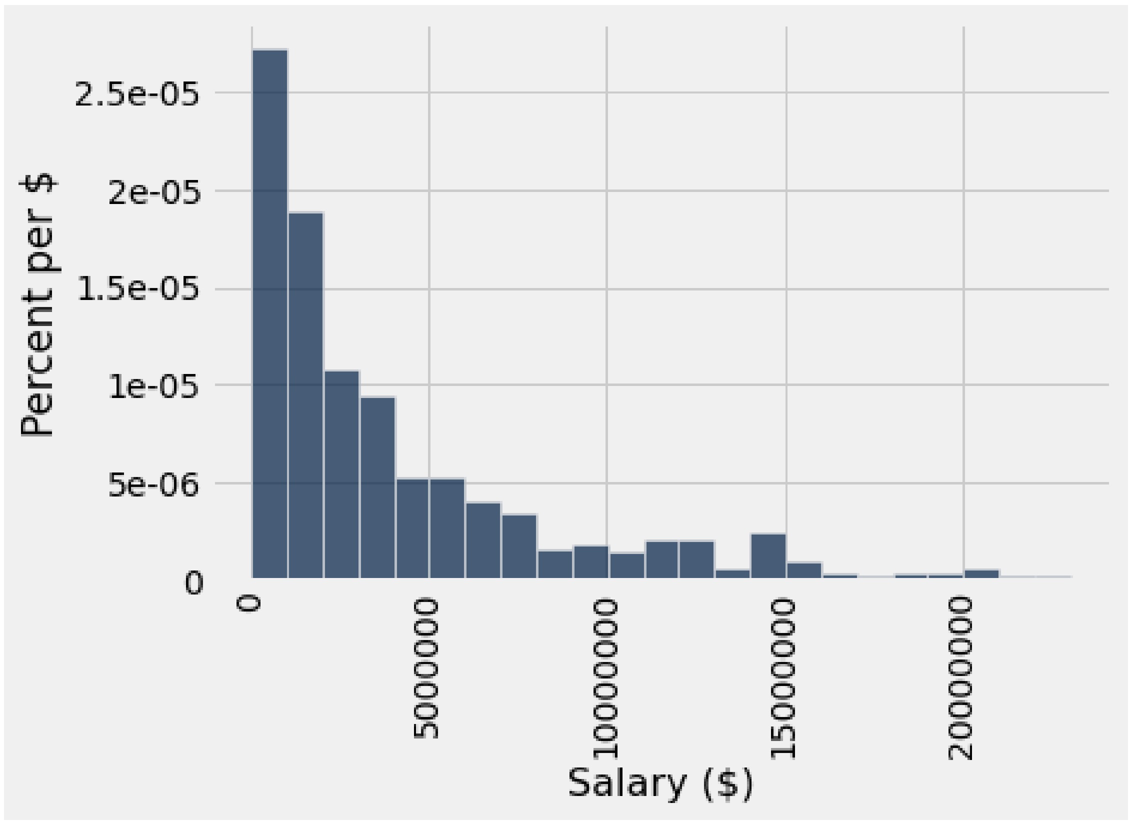
Hint 2: You may have a kernel issue with running it. If you do, then you didn't make your bin widths 1,000,000 for the salaries. Be sure to fix that and you should be able to work the entire lab.

[17]:
```python
def histograms(t):
    ages = t.column('Age')
    salaries = t.column('Salary')
    age_bins = range(ages.min(), ages.max() + 1)
    salary_bins = range(salaries.min(), salaries.max() + 1, 1000000)
    t.hist('Age', bins=age_bins, unit='year')
    t.hist('Salary', bins=salary_bins, unit='$')
    return age_bins # Keep this statement so that your work can be checked

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

```
/opt/conda/lib/python3.7/site-packages/datascience/tables.py:5206: UserWarning:
FixedFormatter should only be used together with FixedLocator
  axis.set_xticklabels(ticks, rotation='vertical')
```

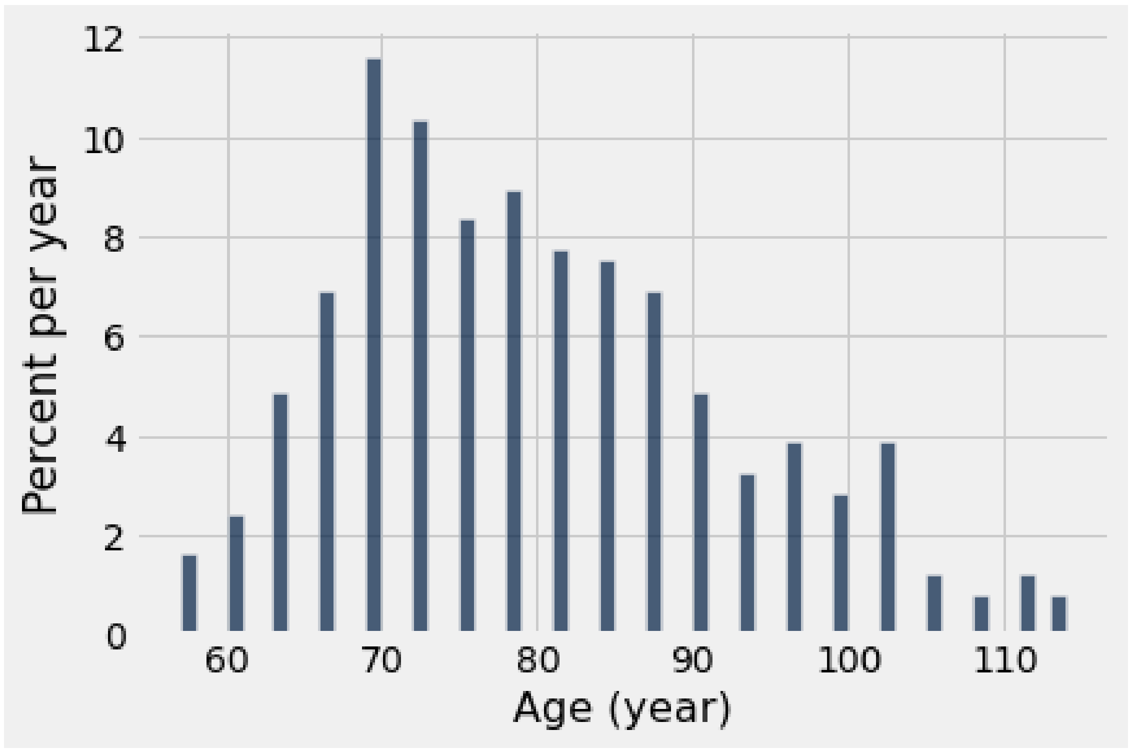
```
Two histograms should be displayed below
```



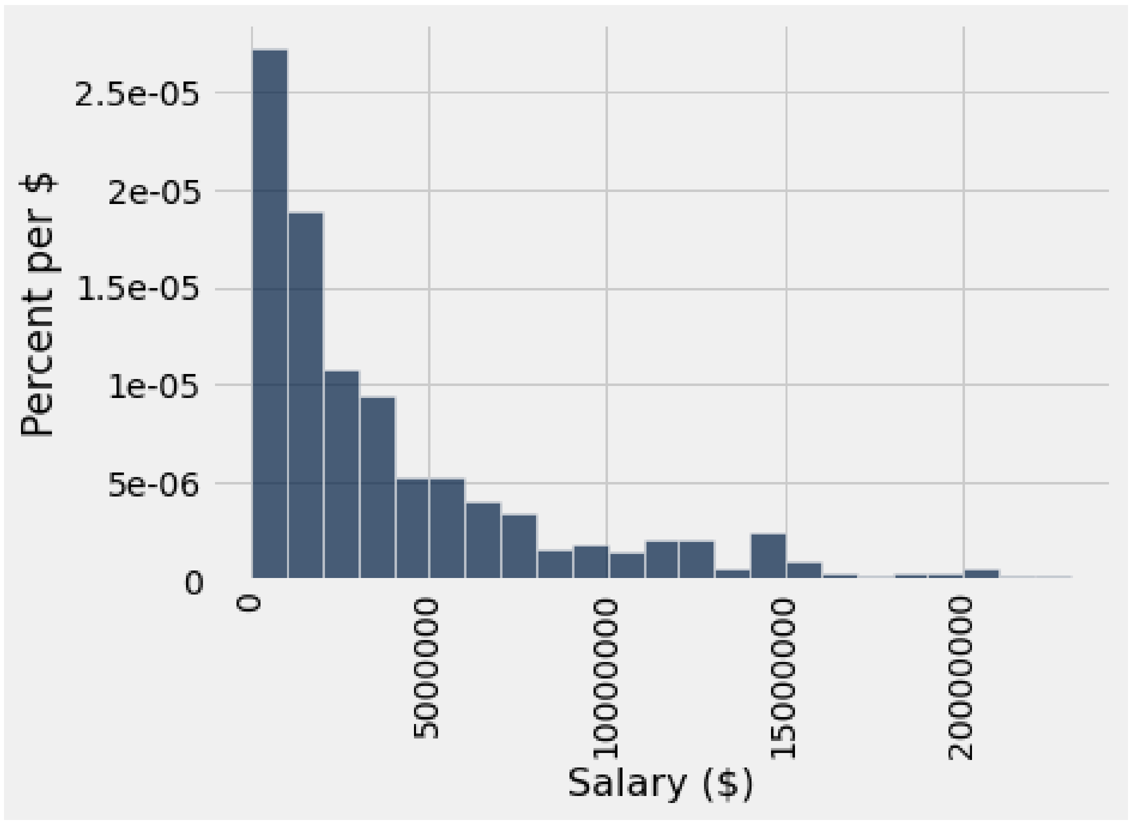


```
[18]: check('tests/q2_1.py') # Warning: Charts will be displayed while running this␣
      ↪test
```

```
[18]: <gofer.ok.OKTestsResult at 0x7f240188fd10>
```



Percent per year
Age (year)

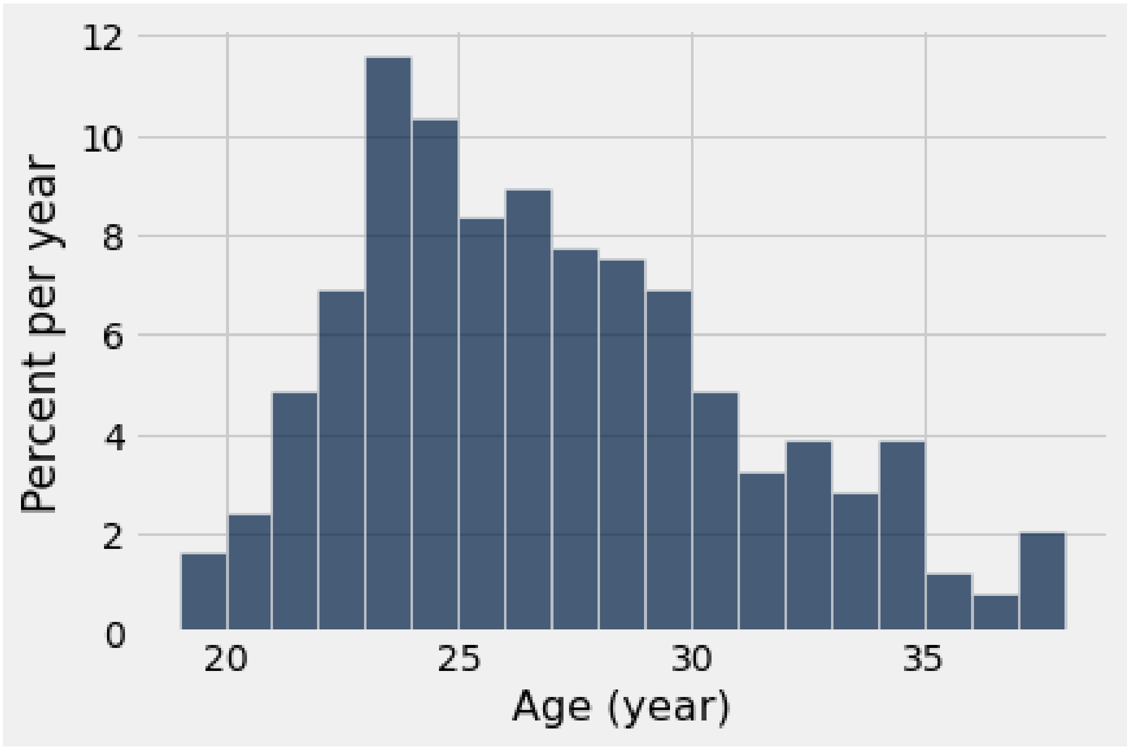


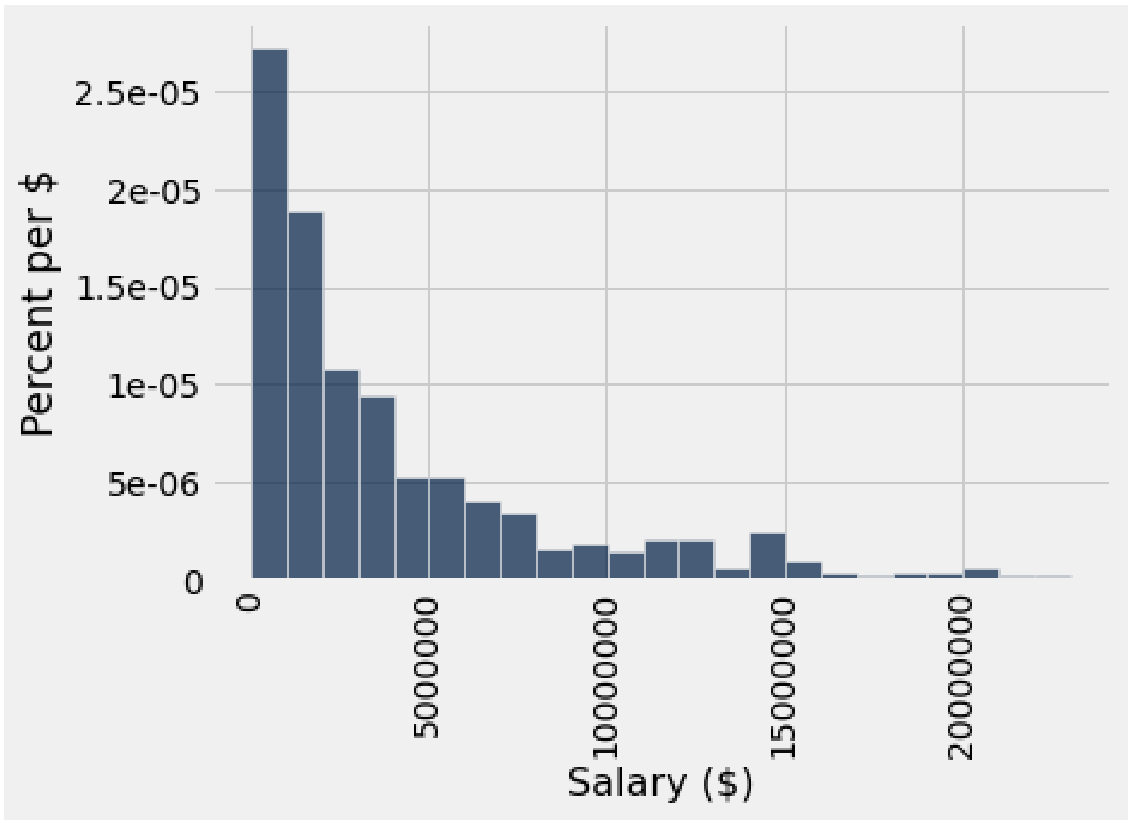
Percent per $
Salary ($)

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 - Returns a two-element array containing the average age and average salary

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

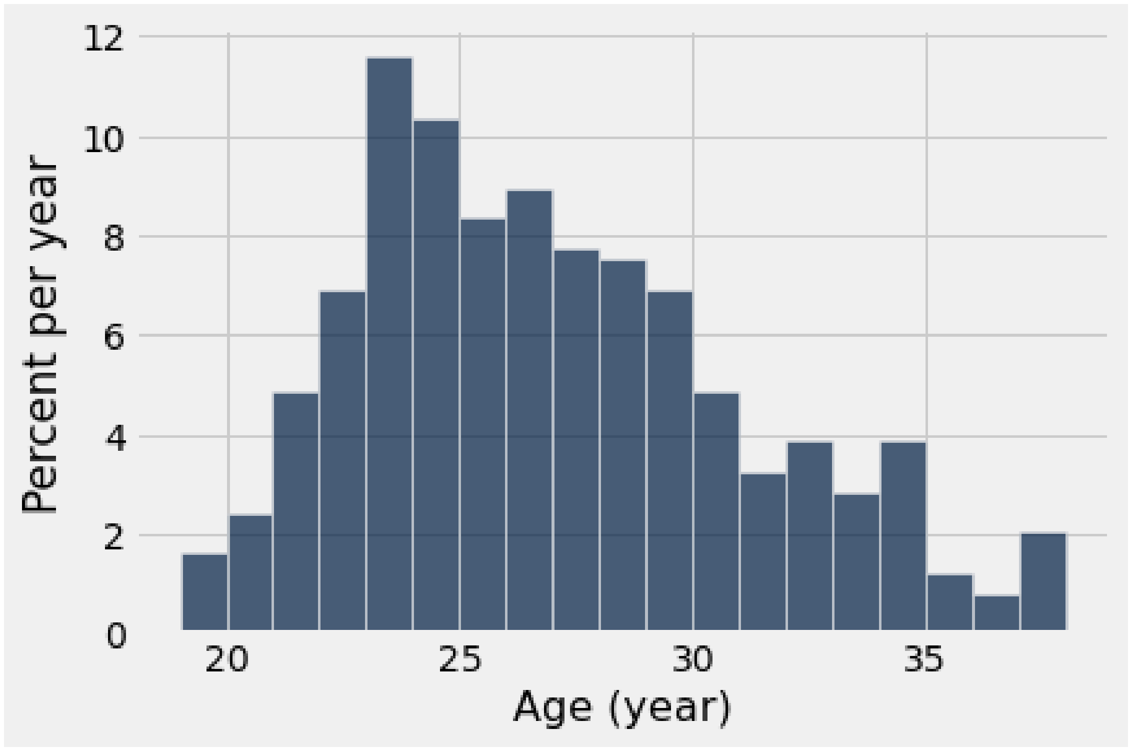
[19]:
```
def compute_statistics(age_and_salary_data):
    histograms(age_and_salary_data)
    age = age_and_salary_data["Age"]
    salary = age_and_salary_data["Salary"]
    return np.array([age.mean(), salary.mean()])


full_stats = compute_statistics(full_data)
```

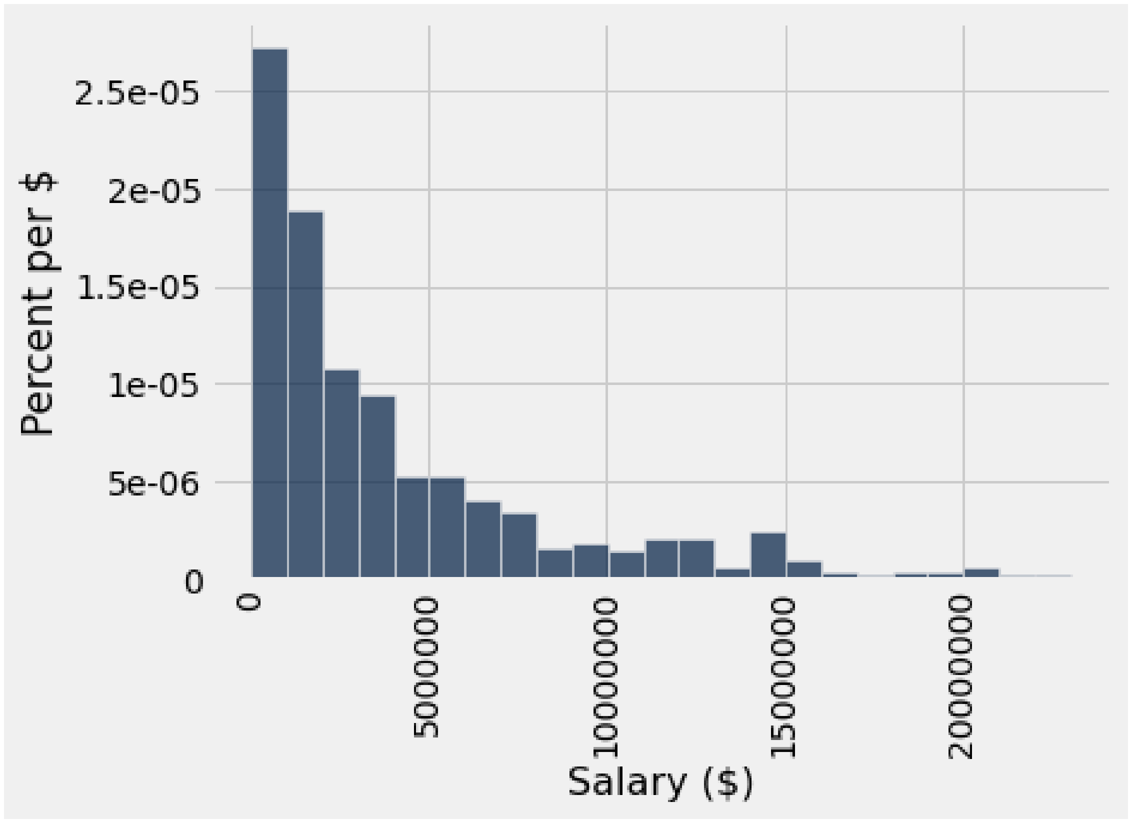


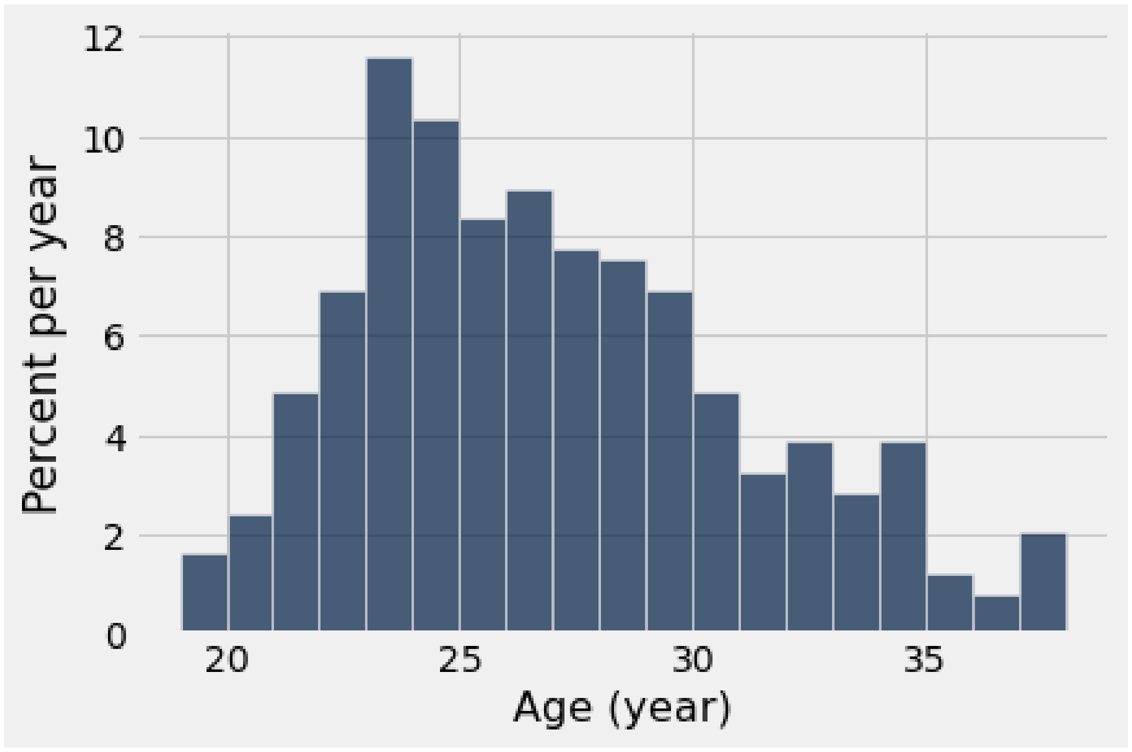


```
[20]: check('tests/q2_2.py') # Warning: Charts will be displayed while running this␣
        ↪test
```

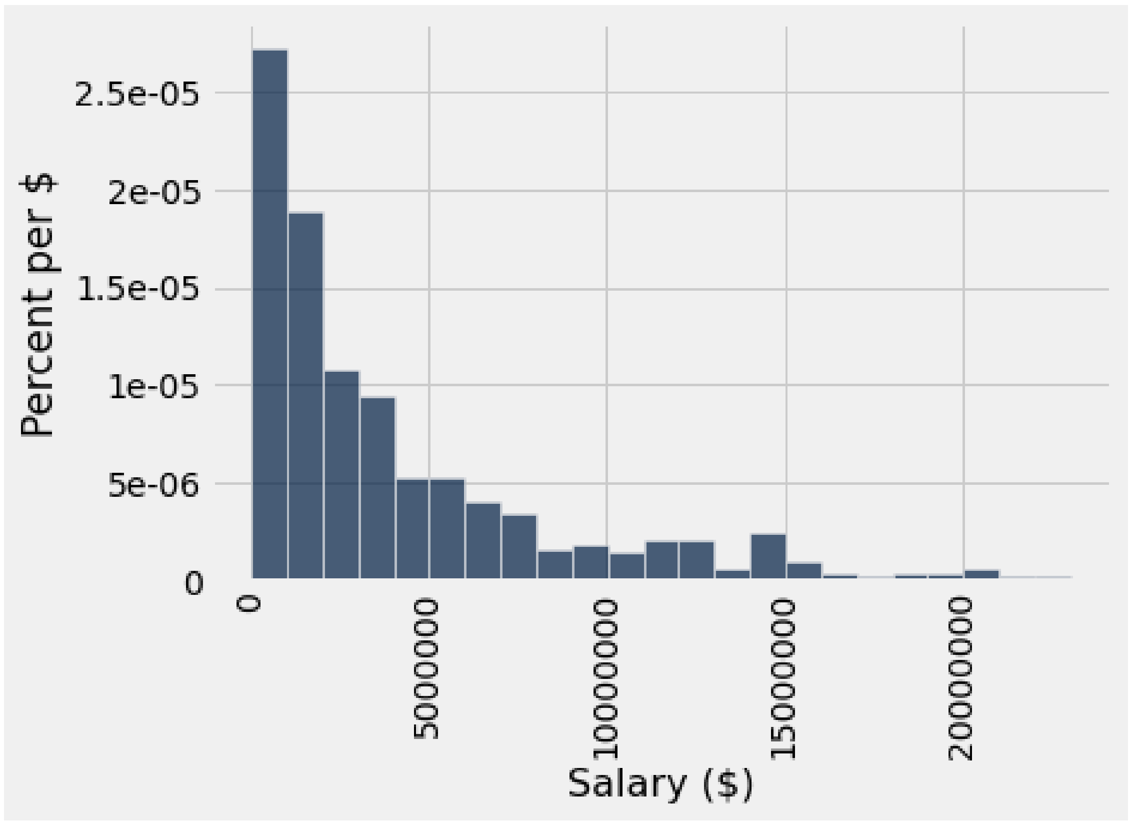
```
[20]: <gofer.ok.OKTestsResult at 0x7f24015e0f10>
```


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


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



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



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

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.

```
[21]: convenience_sample = full_data.where("Age", are.below(22))
      convenience_sample
```

```
[21]: PlayerName       | Salary  | Age  | Team | Games | Rebounds | Assists | Steals |
      Blocks | Turnovers | Points
      Aaron Gordon     | 3992040 | 19   | ORL  | 47    | 169      | 33      | 21     |
      22     | 38        | 243
      Alex Len         | 3649920 | 21   | PHO  | 69    | 454      | 32      | 34     |
      105    | 74        | 432
      Andre Drummond   | 2568360 | 21   | DET  | 82    | 1104     | 55      | 73     |
      153    | 120       | 1130
      Andrew Wiggins   | 5510640 | 19   | MIN  | 82    | 374      | 170     | 86     |
      50     | 177       | 1387
      Anthony Bennett  | 5563920 | 21   | MIN  | 57    | 216      | 48      | 27     |
      16     | 36        | 298
      Anthony Davis    | 5607240 | 21   | NOP  | 68    | 696      | 149     | 100    |
      200    | 95        | 1656
      Archie Goodwin   | 1112280 | 20   | PHO  | 41    | 74       | 44      | 18     |
      9      | 48        | 231
      Ben McLemore     | 3026280 | 21   | SAC  | 82    | 241      | 140     | 77     |
      19     | 138       | 996
      Bradley Beal     | 4505280 | 21   | WAS  | 63    | 241      | 194     | 76     |
      18     | 123       | 962
      Bruno Caboclo    | 1458360 | 19   | TOR  | 8     | 2        | 0       | 0      |
      1      | 4         | 10
      … (34 rows omitted)
```

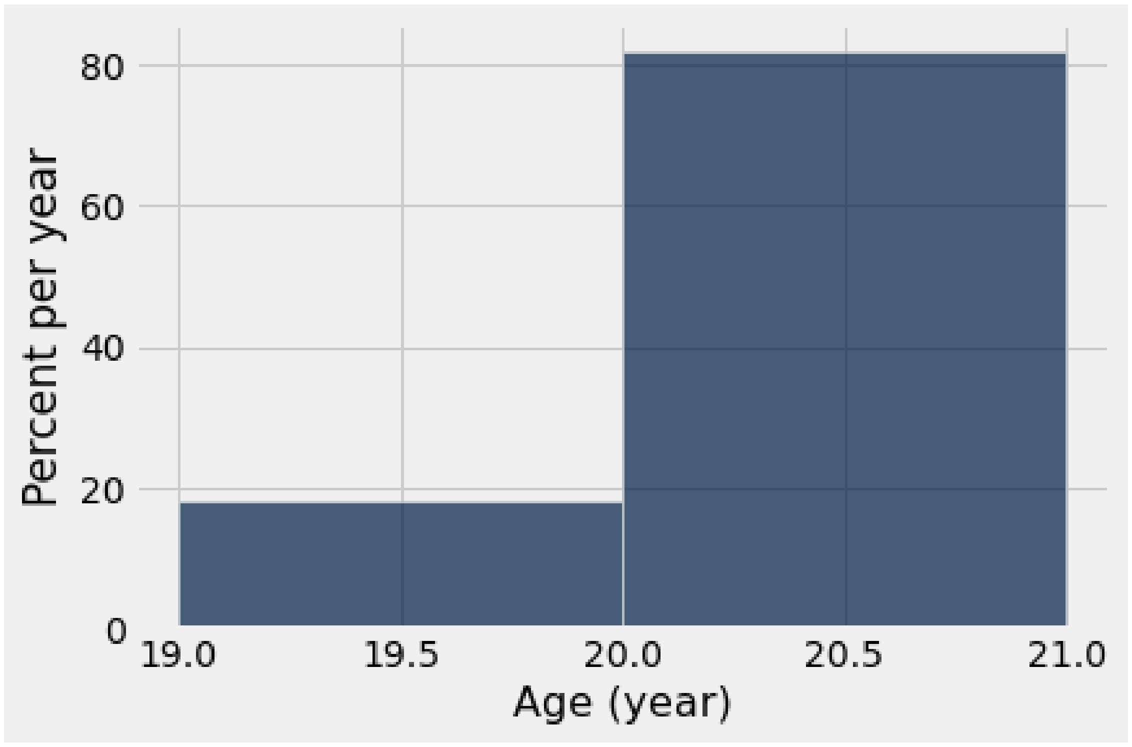
```
[22]: check('tests/q2_3.py')
```

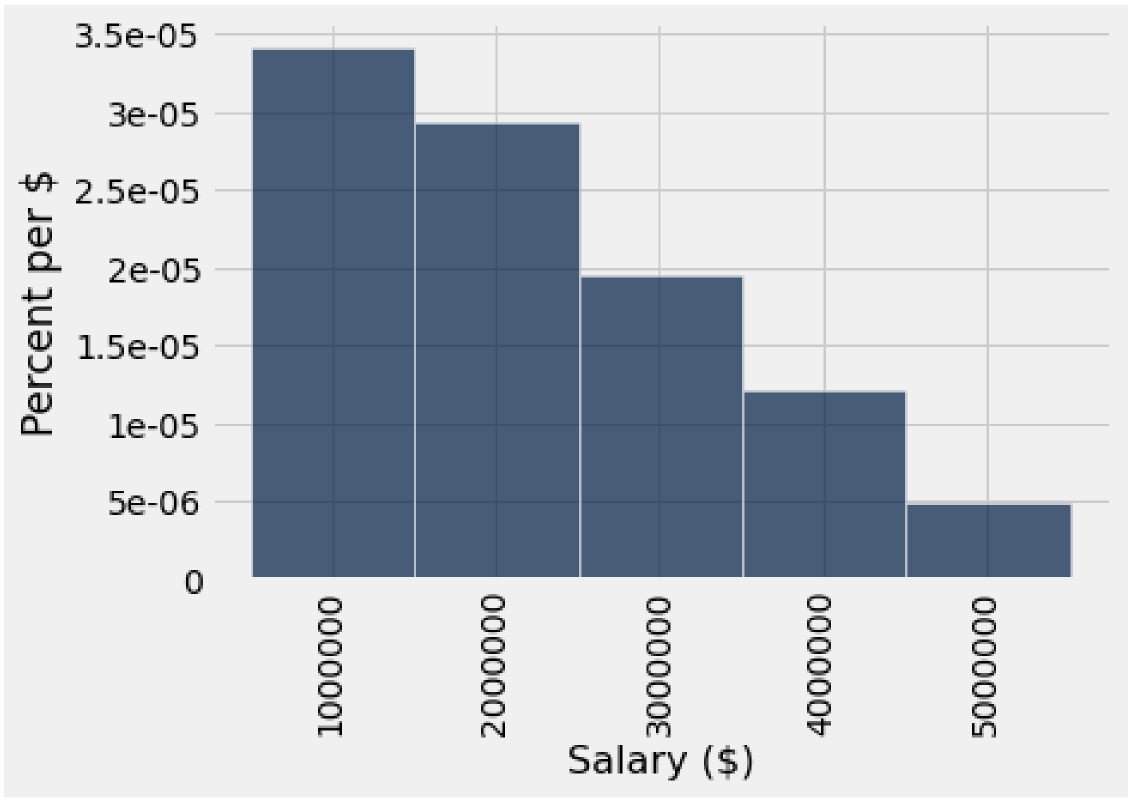
```
[22]: <gofer.ok.OKTestsResult at 0x7f24012c3a10>
```

Question 2.4 Assign `convenience_stats` to a list 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*.

```
[23]: convenience_stats = compute_statistics(convenience_sample)
      convenience_stats
```

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





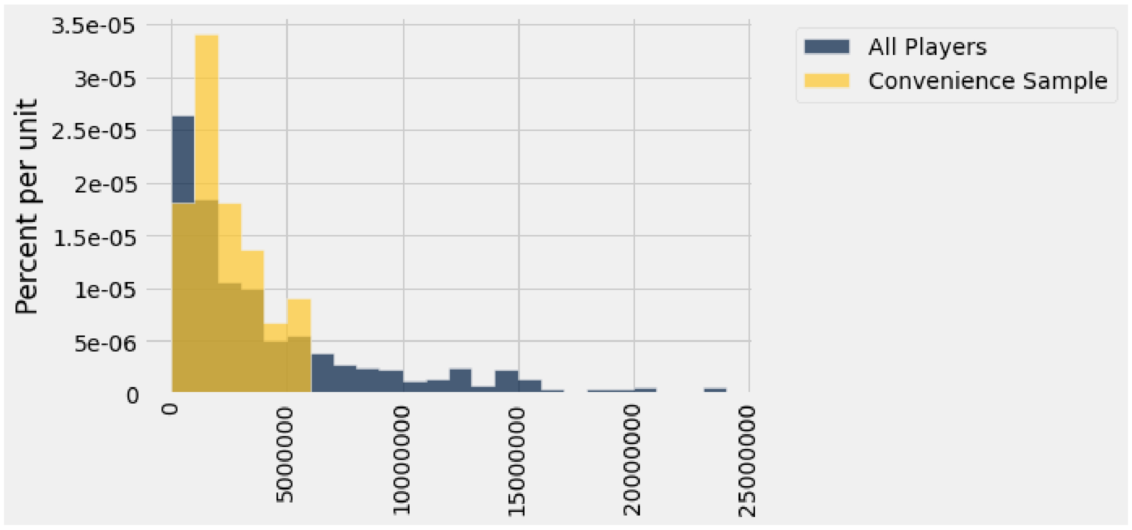
[24]:
```
check('tests/q2_4.py')
```

[24]:
```
<gofer.ok.OKTestsResult at 0x7f240111ec90>
```

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.

[25]:
```
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')
```

```
/opt/conda/lib/python3.7/site-packages/datascience/tables.py:5206: UserWarning:
FixedFormatter should only be used together with FixedLocator
  axis.set_xticklabels(ticks, rotation='vertical')
```



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 an urn, and shuffling the urn. 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.

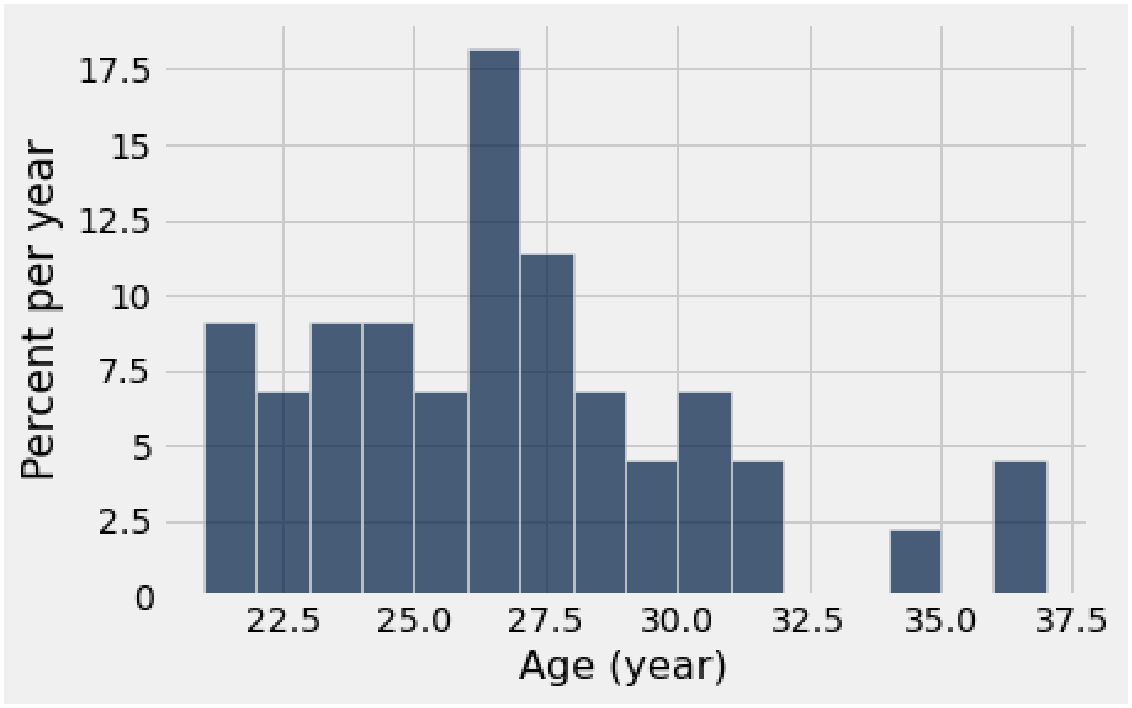
The `load_data` function below loads a salary table and joins it with `player_data`.

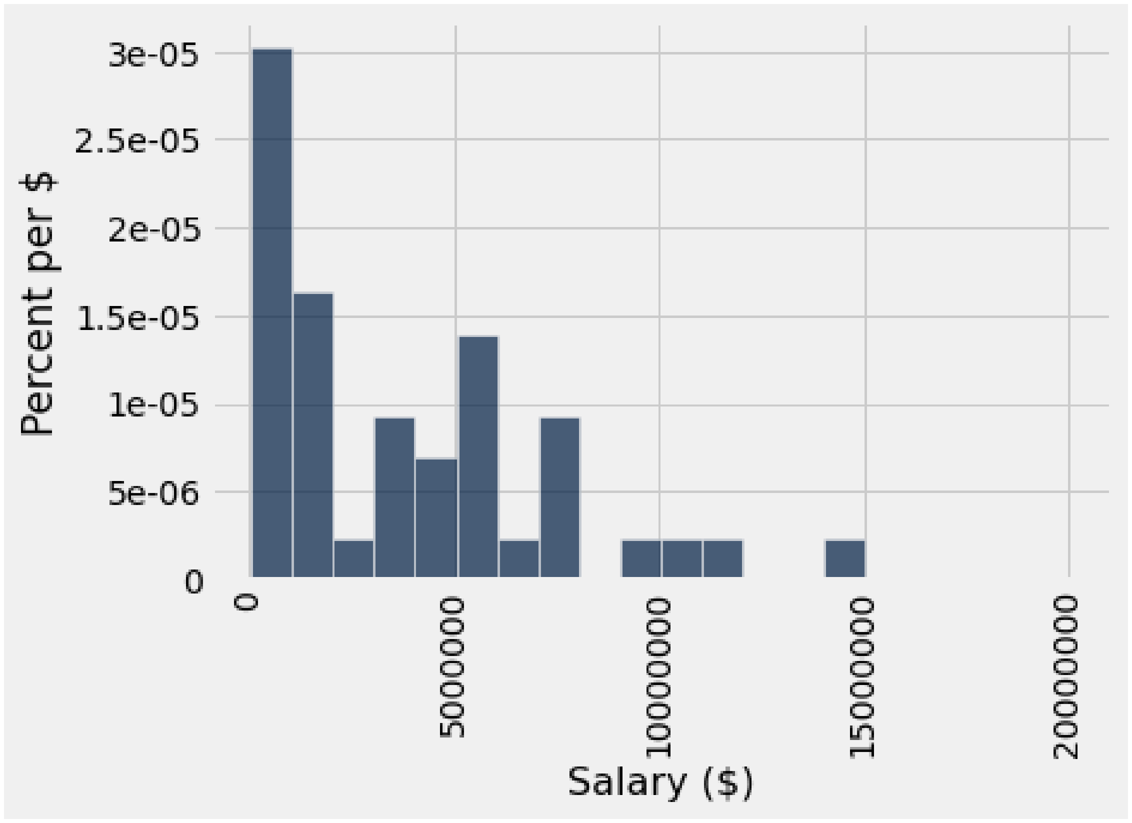
[26]:
```
def load_data(salary_file):
    return player_data.join('Name', Table.read_table(salary_file), 'PlayerName')
```

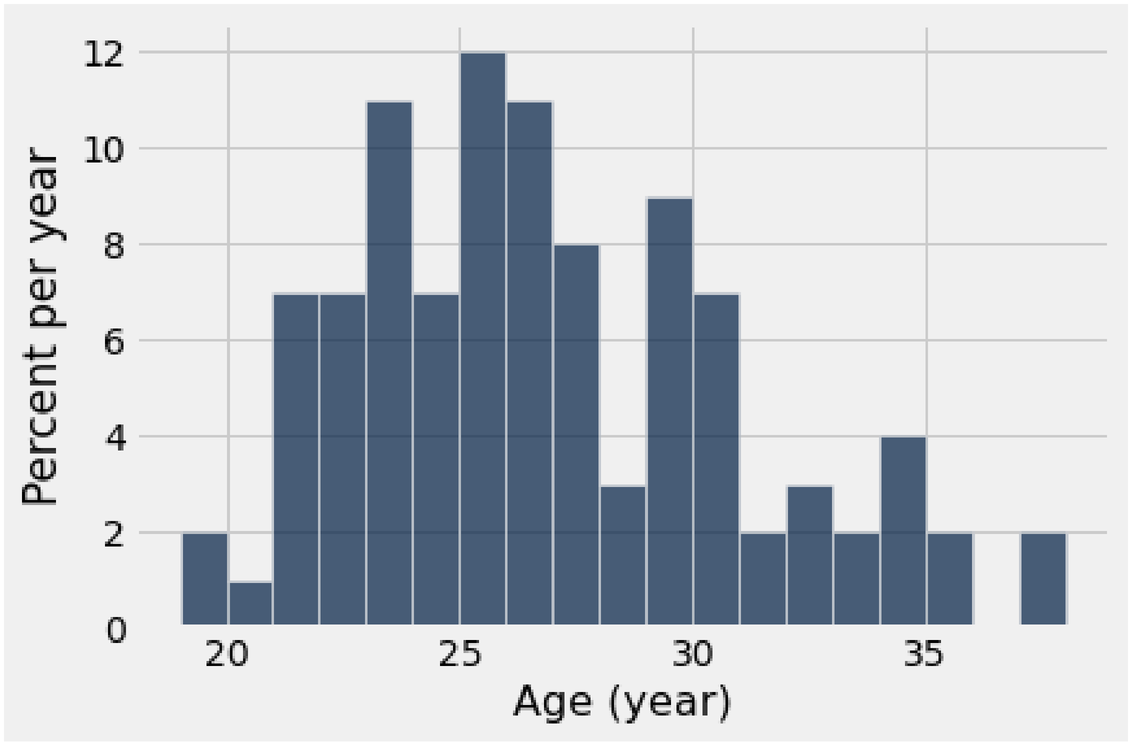
Question 2.5 Run the same analyses on the small and large samples that you previously ran on the full dataset and on the convenience sample. Compare the accuracy of the estimates of the population statistics that we get from the convenience sample, the small simple random sample, and the large simple random sample. (Just notice this for yourself – the autograder will check your sample statistics but will not validate whatever you do to compare.)

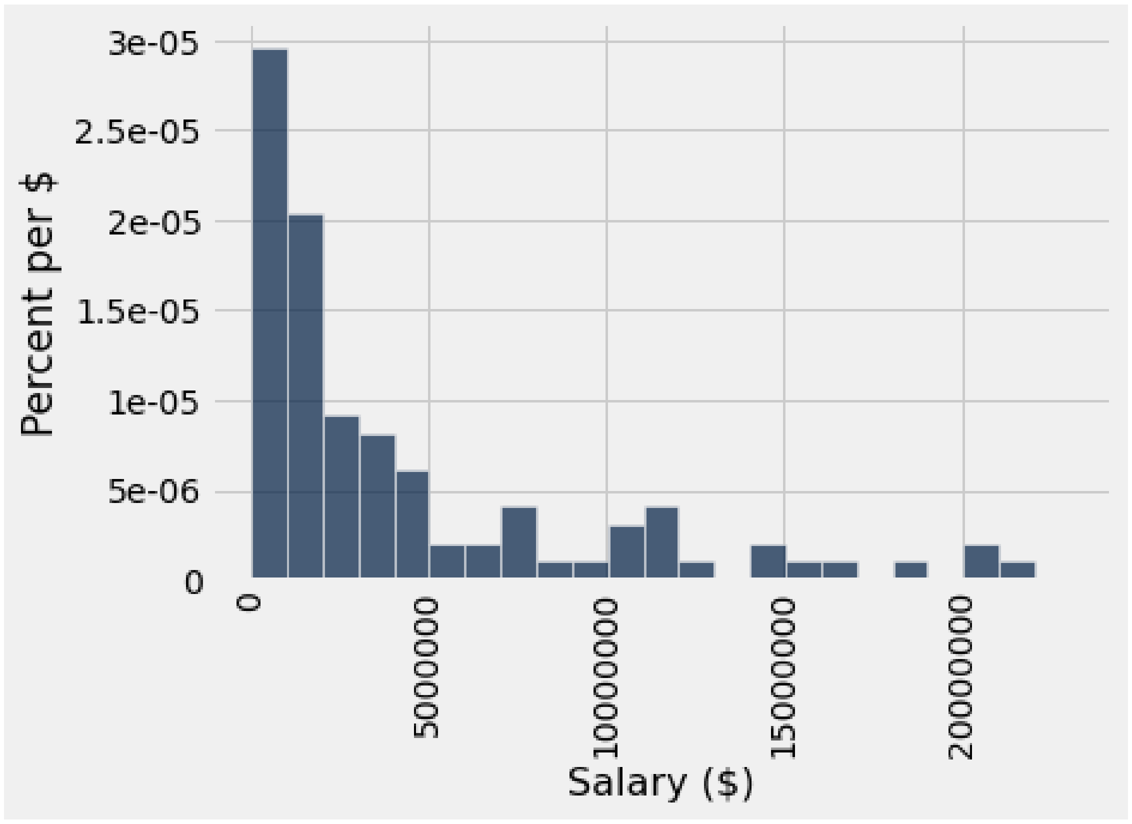
[27]:
```
# Original:
small_srswor_data = load_data("small_srswor_salary.csv")
small_stats = compute_statistics(small_srswor_data)
large_srswor_data = load_data("large_srswor_salary.csv")
large_stats = compute_statistics(large_srswor_data)
print('Full data stats:                  ', full_stats)
print('Small simple random sample stats:', small_stats)
print('Large simple random sample stats:', large_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]
```




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


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



```
[28]: check('tests/q2_5.py')
```

```
[28]: <gofer.ok.OKTestsResult at 0x7f2400eb9e50>
```

1.2.3 Producing Simple Random Samples

Often it's useful to take random samples even when we have a larger dataset available. Another is to help us understand how inaccurate other samples are.

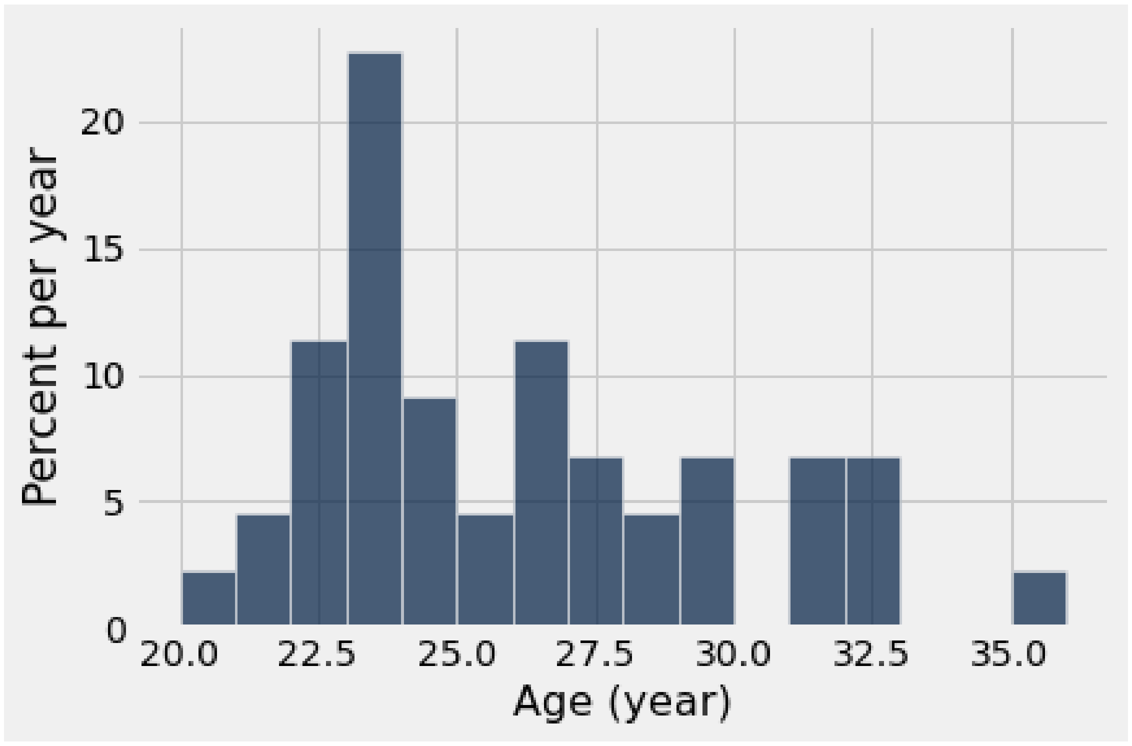
Tables provide the method `sample()` for producing random samples. Note that its default is to sample with replacement. To see how to call `sample()`, search the documentation on the datascience documentation on the course website, or enter `full_data.sample?` into a code cell and press Shift + Enter.

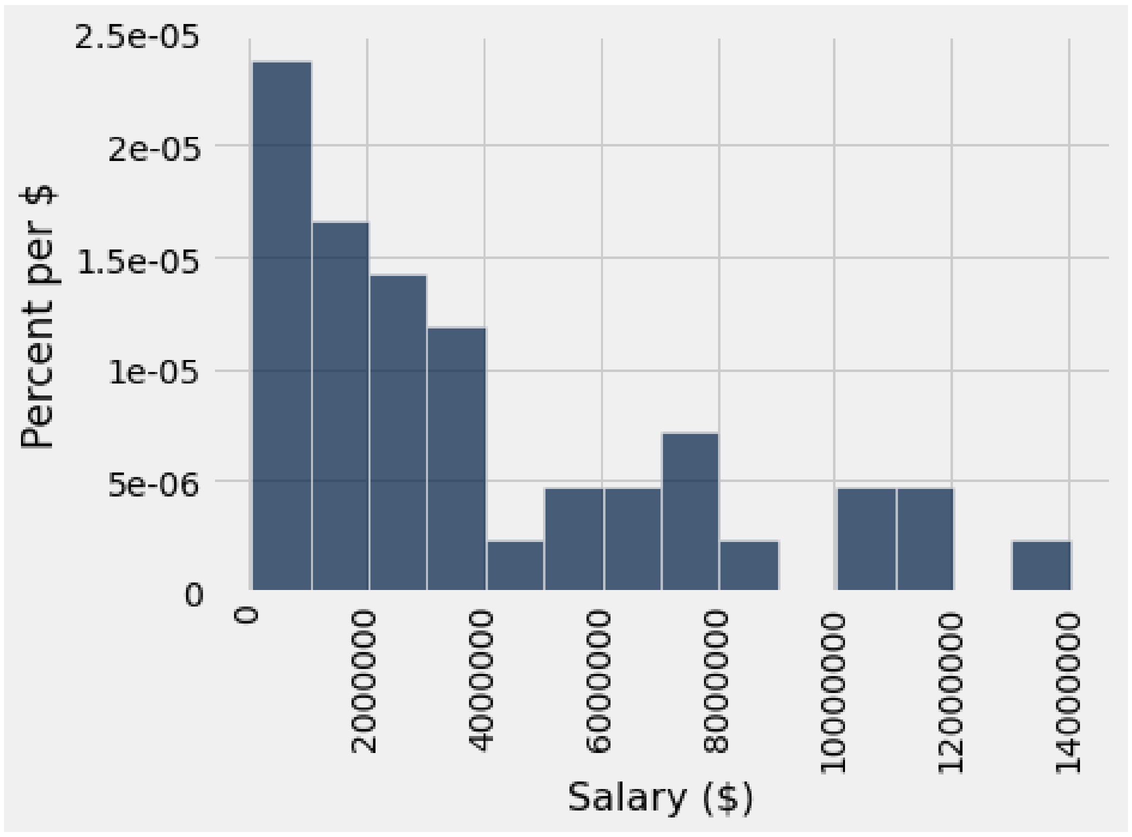
Question 2.6 Produce a simple random sample of size 44 from `full_data`. (You don't need to bother with a join this time – just use `full_data.sample(...)` directly. That will have the same result as sampling from `salary_data` and joining with `player_data`.) Run your analysis on it again and think about these following questions. - Are your results roughly similar to those in the small sample we provided you? Run your code several times to get new samples.
- How much does the average age change across samples? - What about average salary?

Question 2.6 does not have an autograder test, so it is not graded and not in the overall lab grade.

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

```
[29]: array([2.56136364e+01, 4.43859105e+06])
```





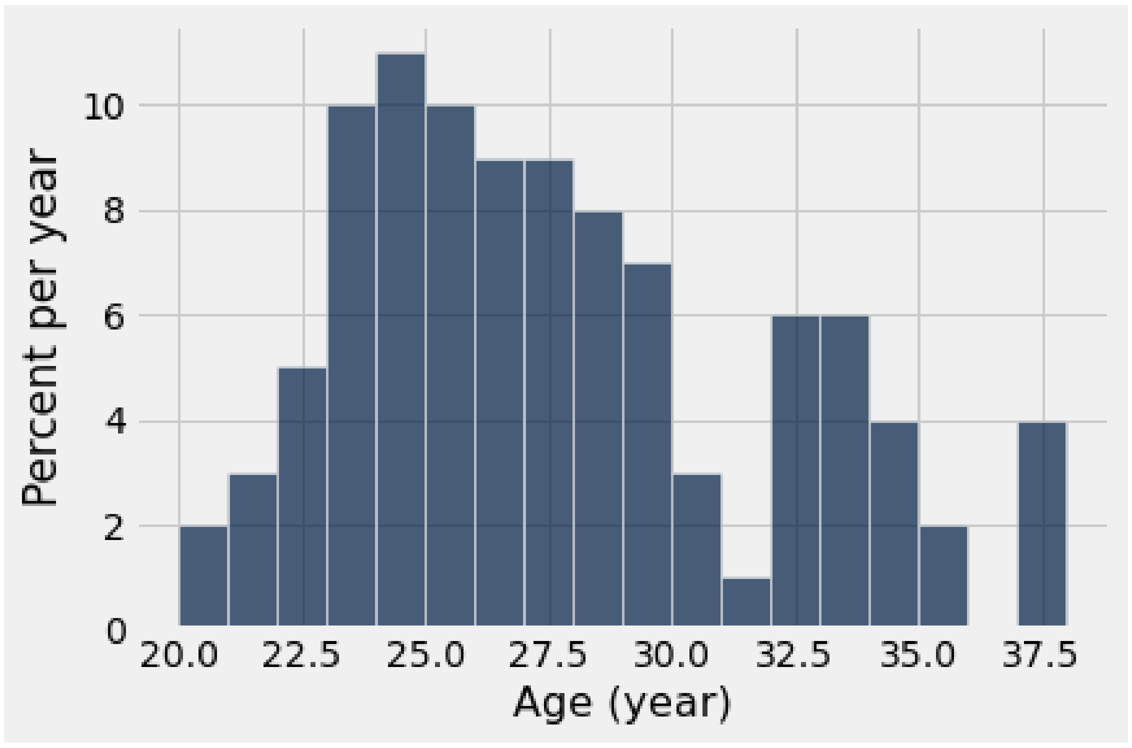
Notice that the results are similar, but not the same, to the sample we were given. The average age tends to stay around the same value as there is a limited range of ages for NBA players, but the salary changes by a sizeable factor due to larger variability in salary.

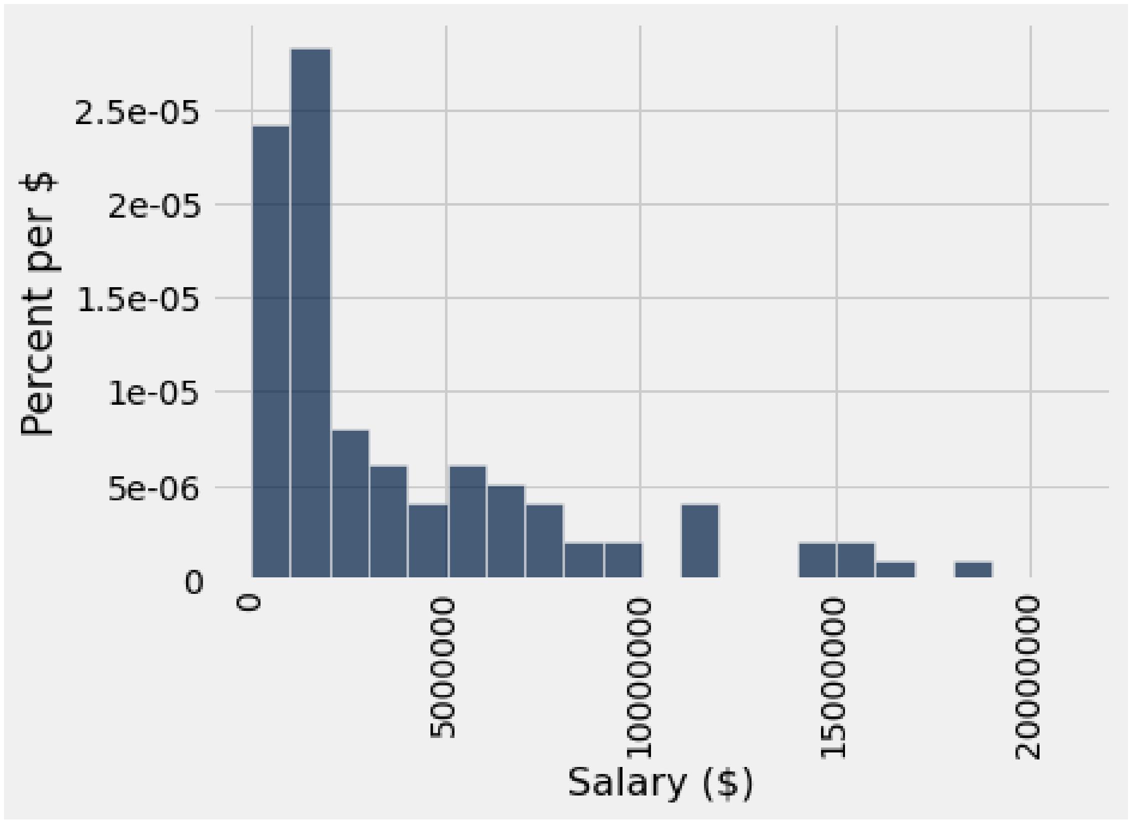
Question 2.7 As in the previous question, analyze several simple random samples of size 100 from `full_data`.
- Do the histogram statistics seem to change more or less across samples of 100 than across samples of size 44?
- Are the sample averages and histograms closer to their true values for age or for salary? What did you expect to see?

Question 2.7 does not have an autograder test, so it is not graded and not in the overall lab grade.

```
[30]: my_large_srswor_data = full_data.sample(100)
      compute_statistics(my_large_srswor_data)
```

```
[30]: array([2.72800000e+01, 4.10393233e+06])
```


Percent per year
0
2
4
6
8
10
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Age (year)


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

The average and histogram statistics seem to change less across samples of this size. They are closer to their true values, which is what we'd expect to see because we are sampling a larger subset of the population.

1.3 3. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

Double check that you have completed all of the free response questions as the autograder does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this lab. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

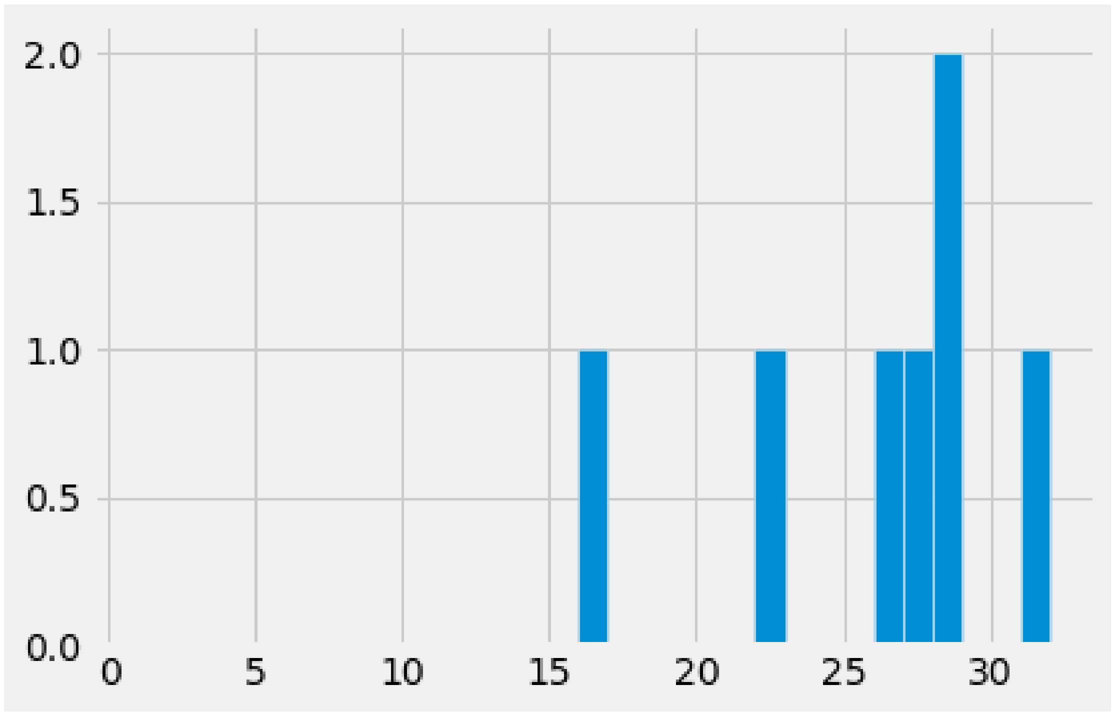
The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

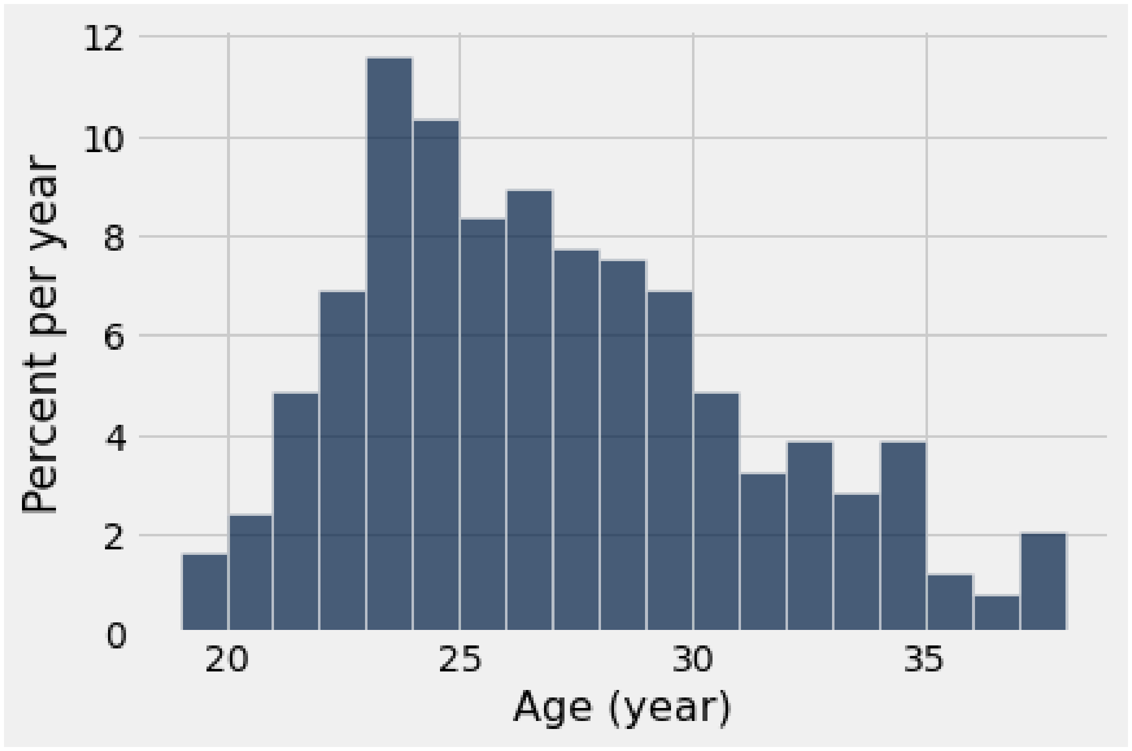
Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

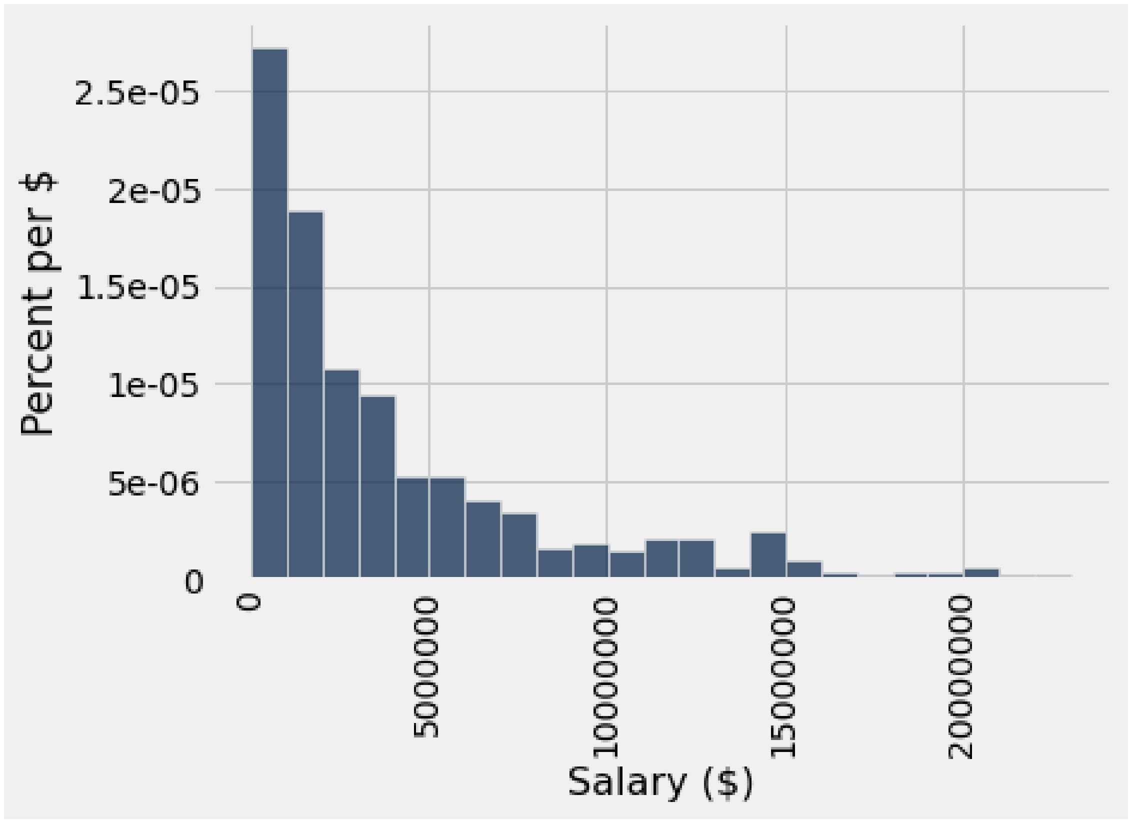
[31]:
```python
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('lab07.ipynb', sorted(glob.glob('tests/q*.py'))))
```

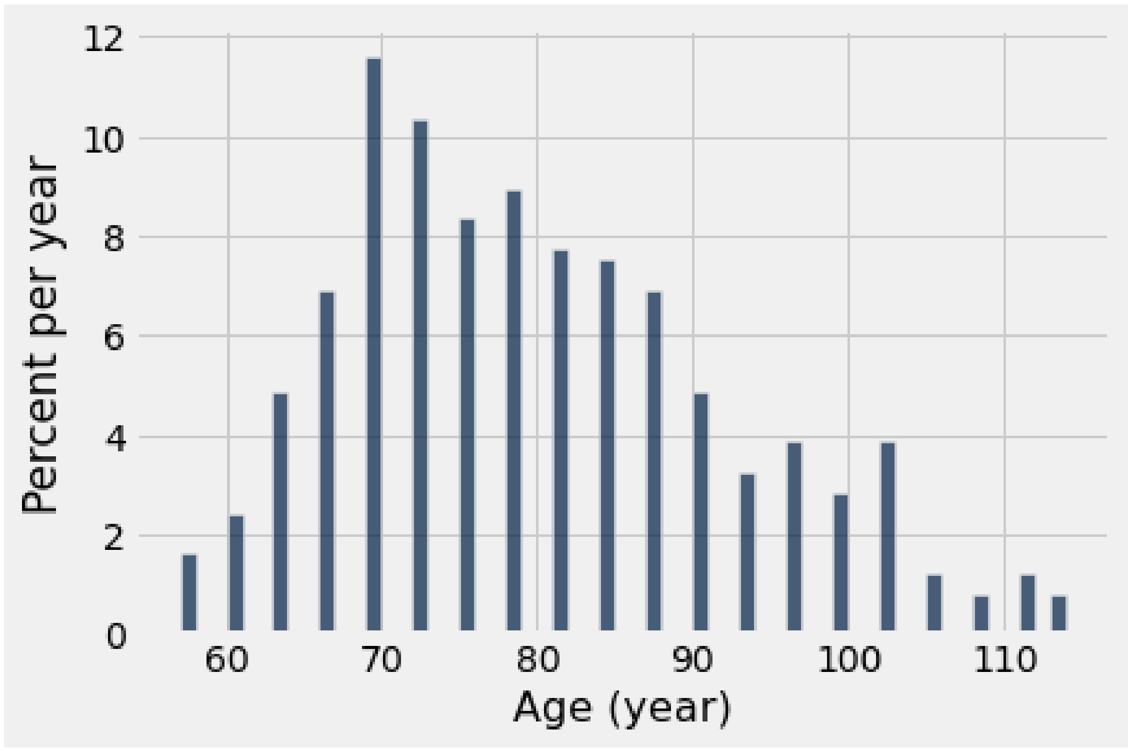
```
On a modified roll of 19, Alice's action failed.
Two histograms should be displayed below
Full data stats:                  [2.65365854e+01 4.26977577e+06]
Small simple random sample stats: [2.63181818e+01 4.28391089e+06]
Large simple random sample stats: [2.6420000e+01 4.8213225e+06]
['tests/q1_1.py', 'tests/q1_2.py', 'tests/q1_3.py', 'tests/q1_5.py',
'tests/q1_6.py', 'tests/q2_1.py', 'tests/q2_2.py', 'tests/q2_3.py',
'tests/q2_4.py', 'tests/q2_5.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7f2403ae2d90>

Question 2:

<gofer.ok.OKTestsResult at 0x7f2400ff9b10>

Question 3:

<gofer.ok.OKTestsResult at 0x7f24014ebad0>

Question 4:
```

```
<gofer.ok.OKTestsResult at 0x7f240111e350>

Question 5:

<gofer.ok.OKTestsResult at 0x7f24010da1d0>

Question 6:

<gofer.ok.OKTestsResult at 0x7f2401111a10>

Question 7:

<gofer.ok.OKTestsResult at 0x7f2400b82f10>

Question 8:

<gofer.ok.OKTestsResult at 0x7f24011ade50>

Question 9:

<gofer.ok.OKTestsResult at 0x7f2400b89a50>

Question 10:

<gofer.ok.OKTestsResult at 0x7f240030db10>

0.9
```

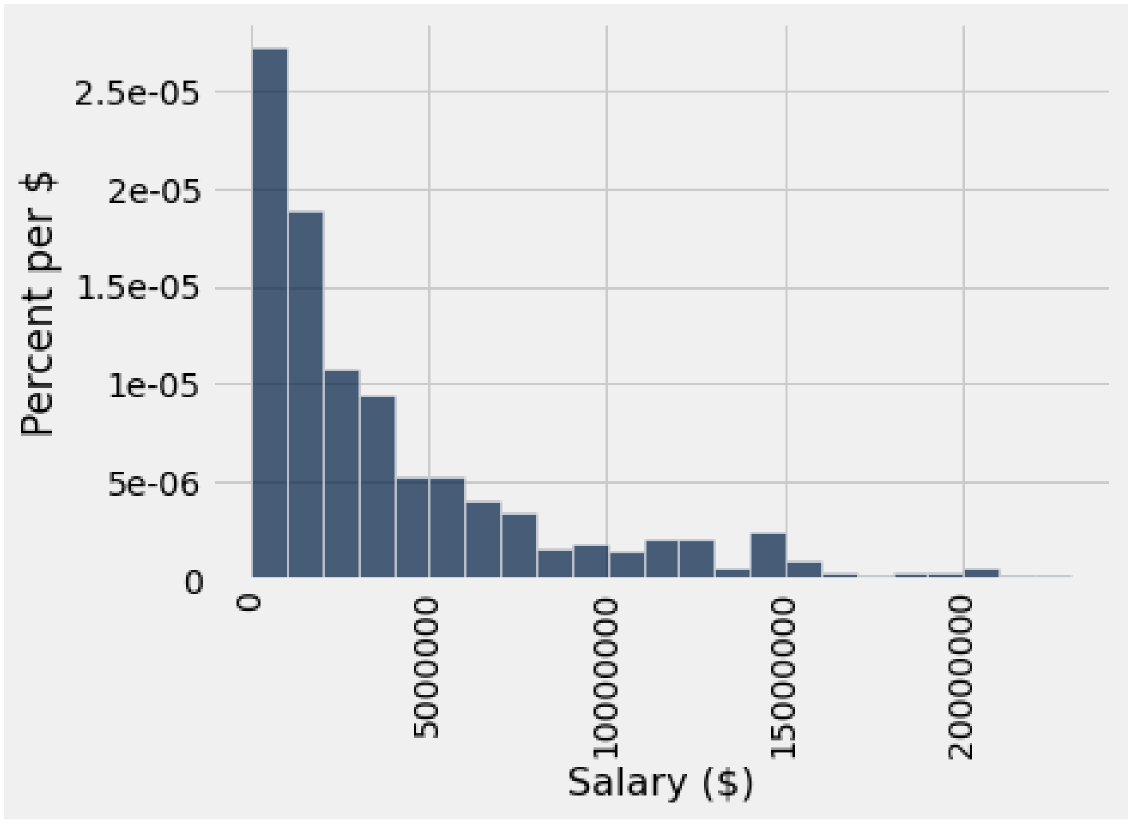



Percent per year
0
2
4
6
8
10
12
20
25
30
35
Age (year)

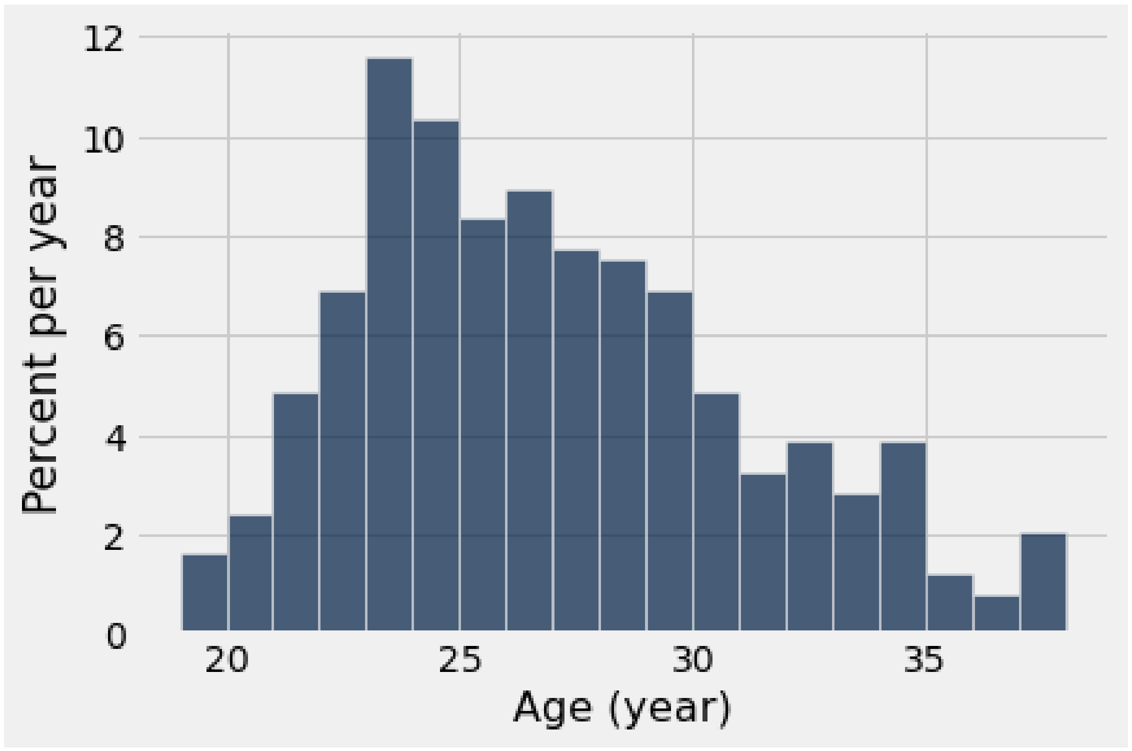

Percent per $
0
5e-06
1e-05
1.5e-05
2e-05
2.5e-05
0
5000000
10000000
15000000
20000000
Salary ($)

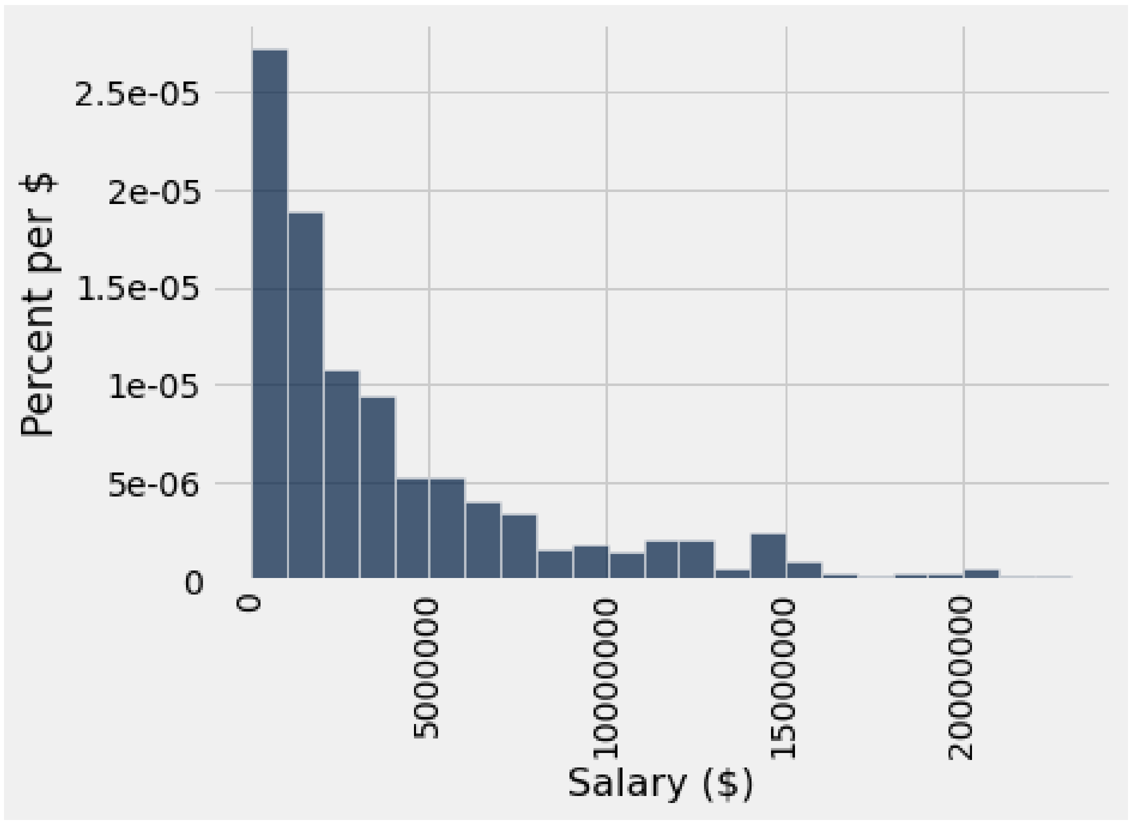


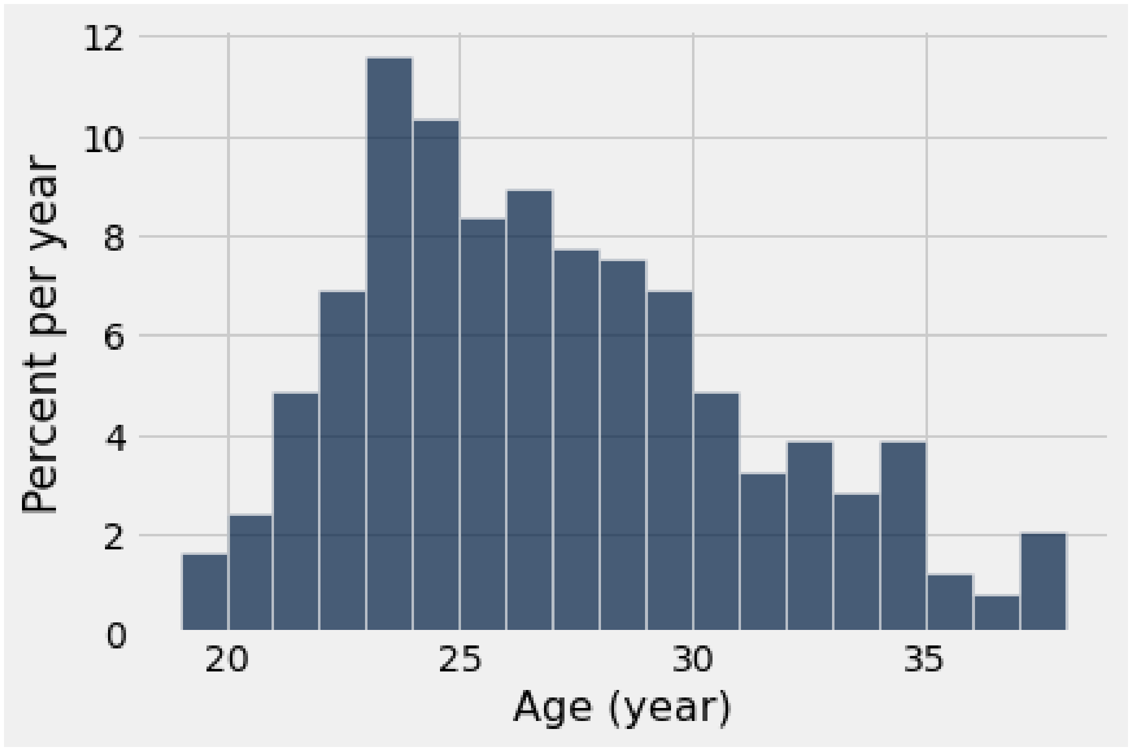
Percent per year
12
10
8
6
4
2
0
60
70
80
90
100
110
Age (year)

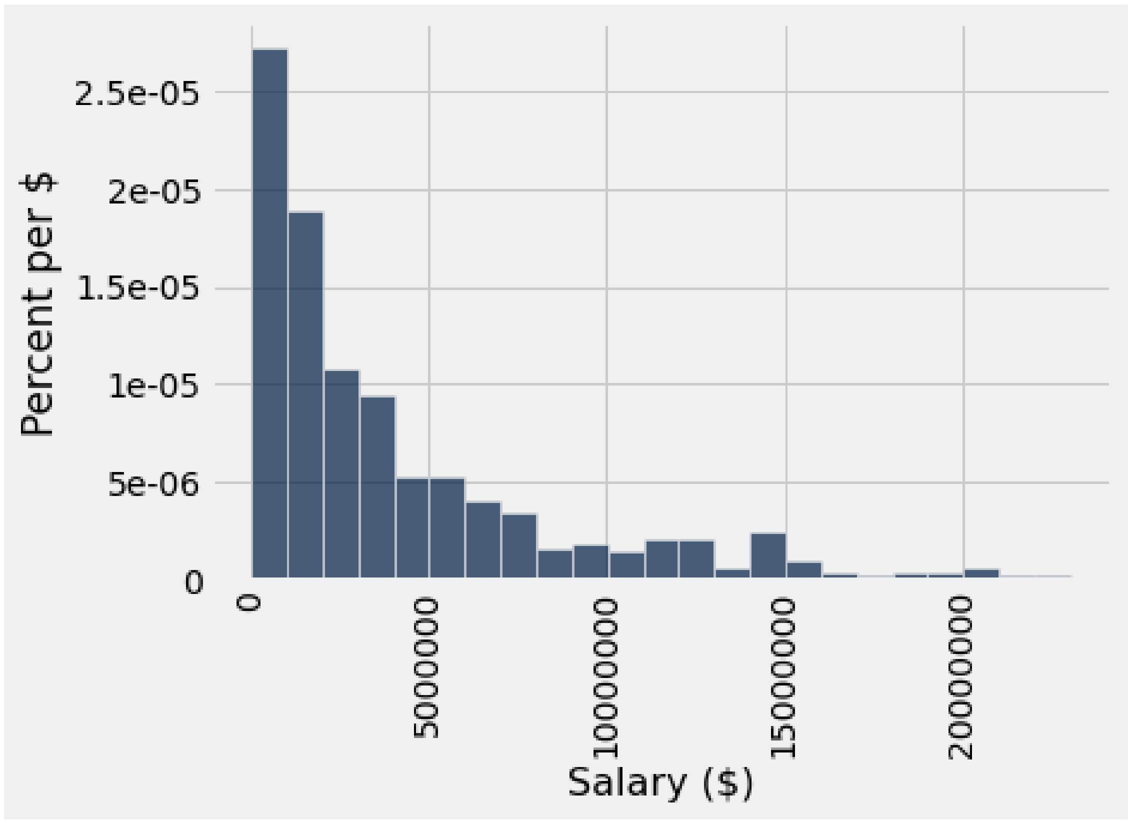


Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

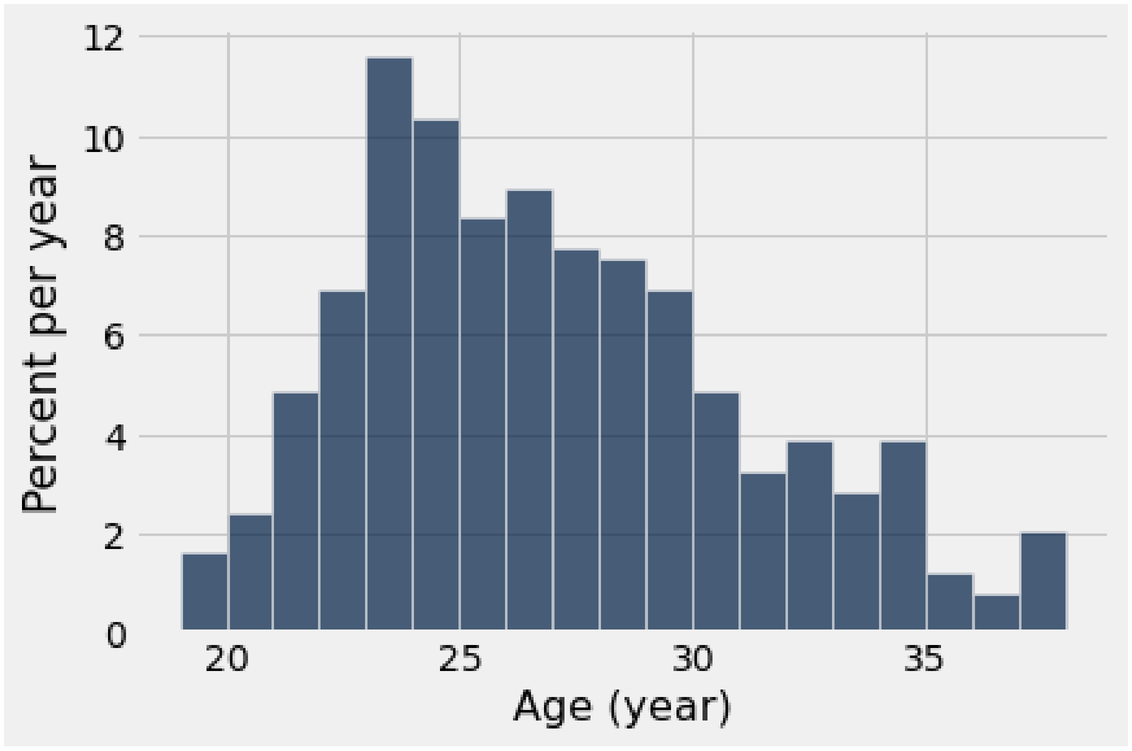

Percent per year
0
2
4
6
8
10
12
20
25
30
35
Age (year)


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

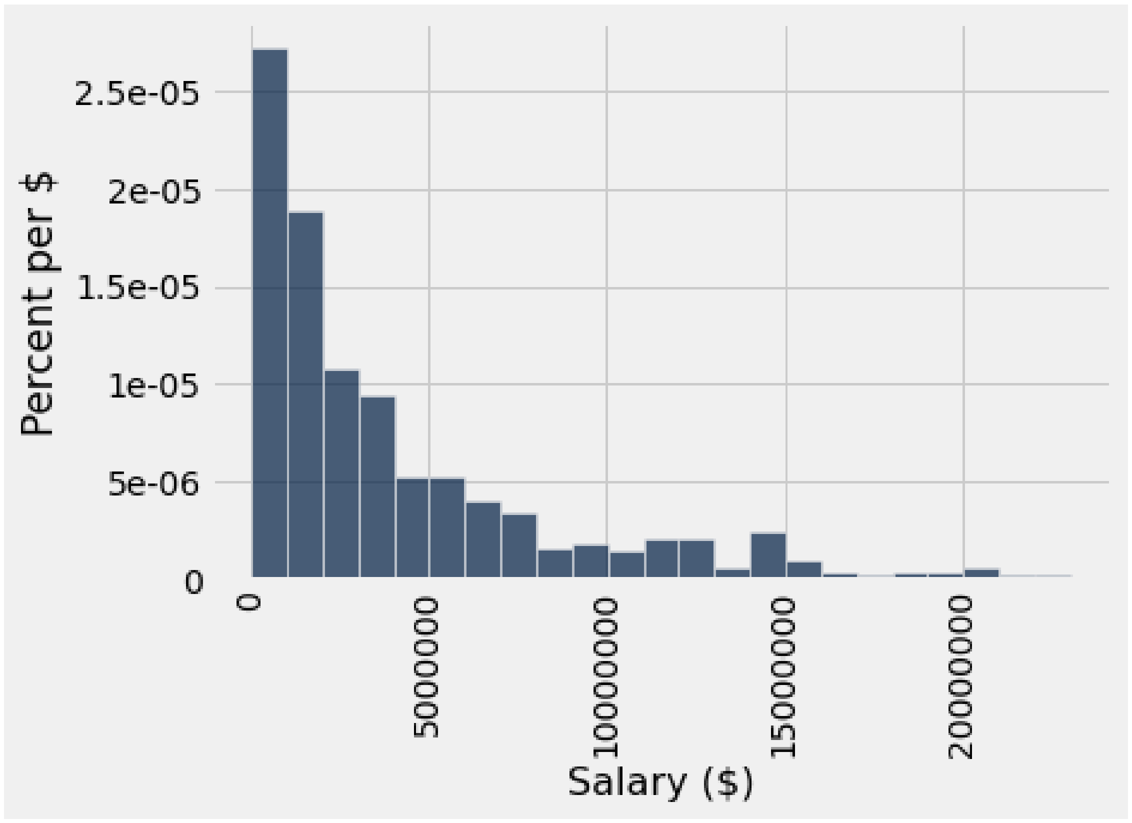

Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)

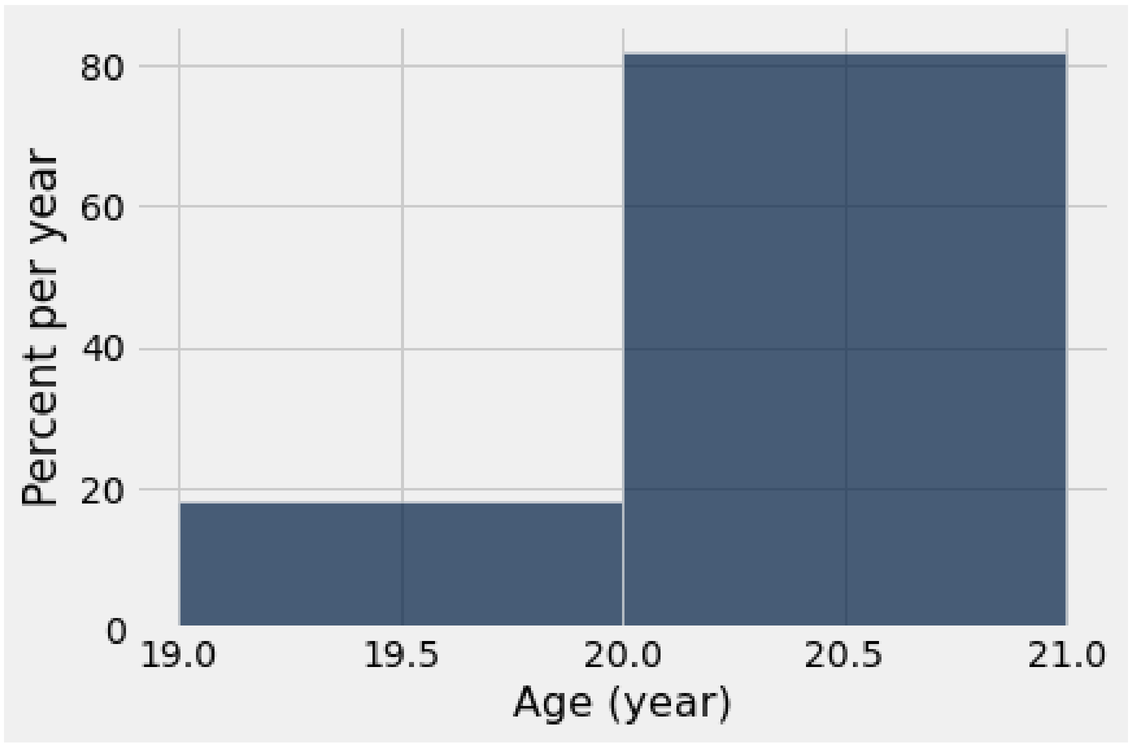


Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

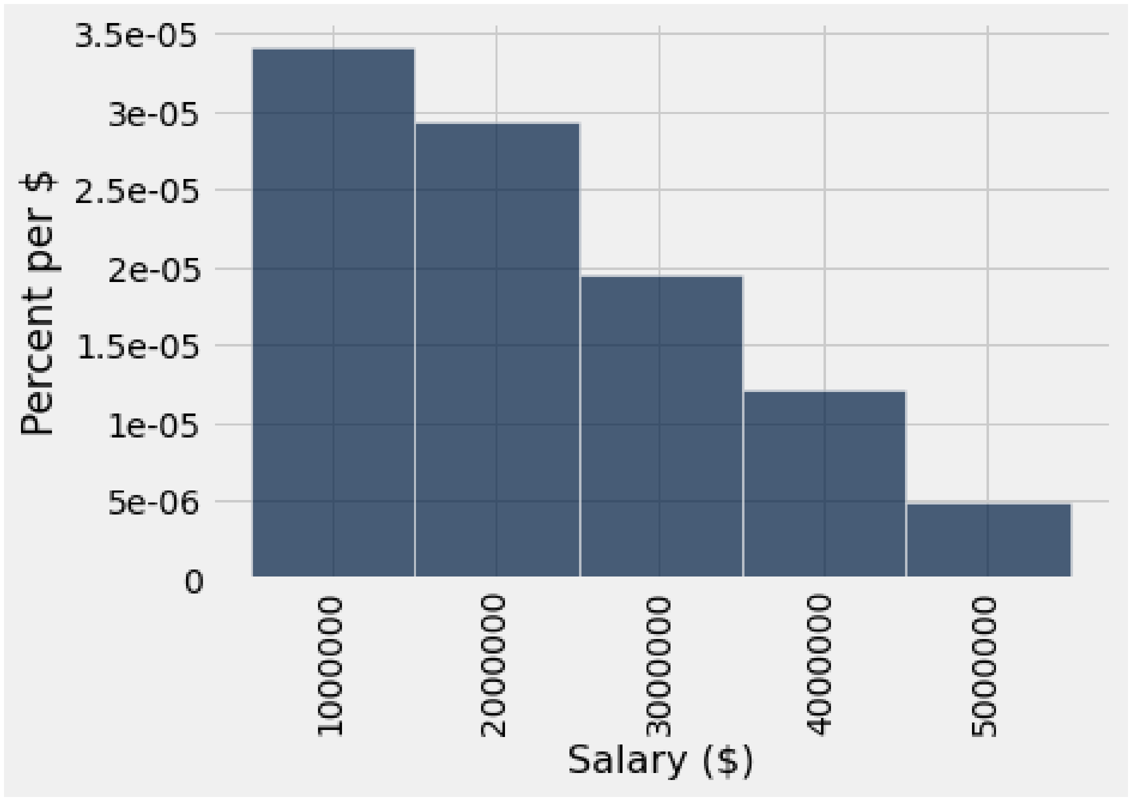


Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)

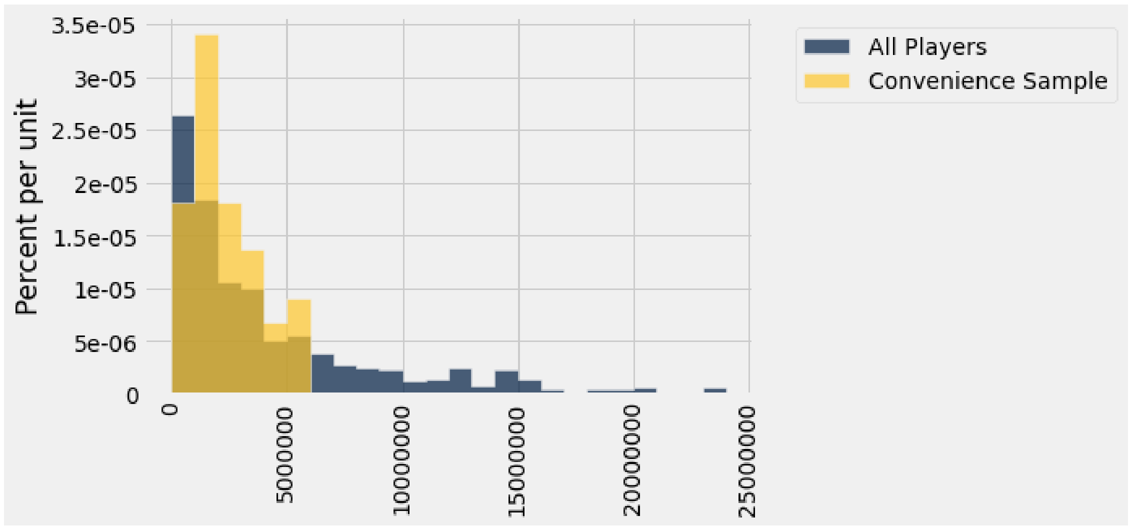

2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
Salary ($)



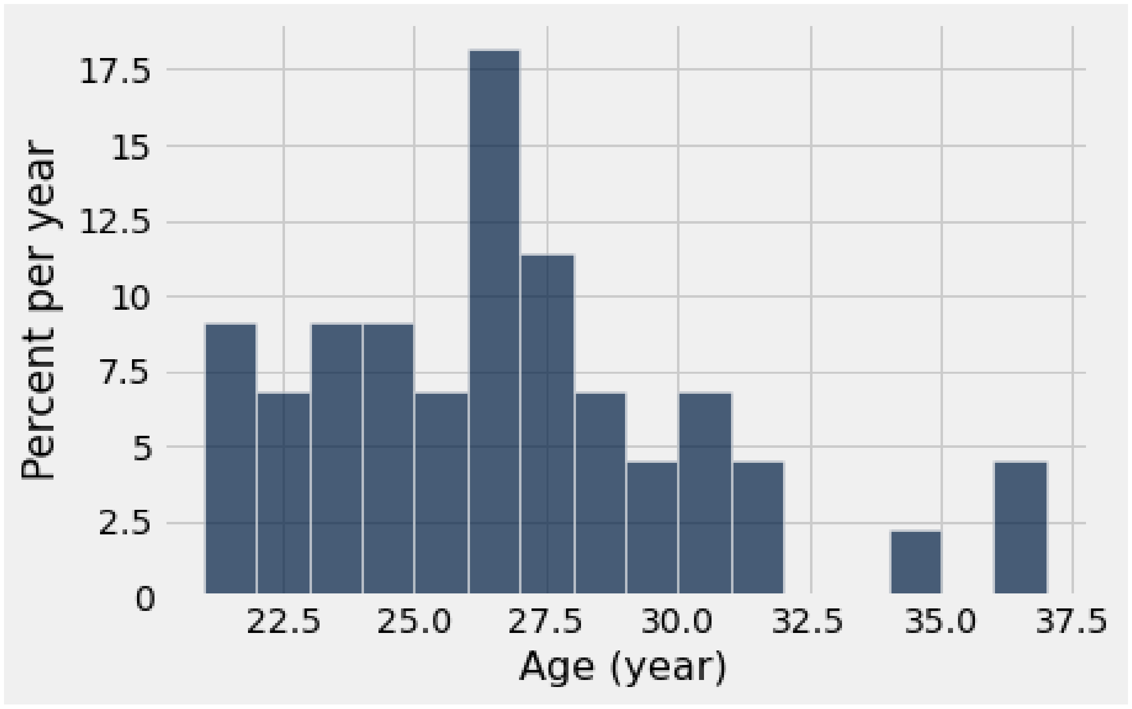
Percent per year
0
20
40
60
80
19.0
19.5
20.0
20.5
21.0
Age (year)



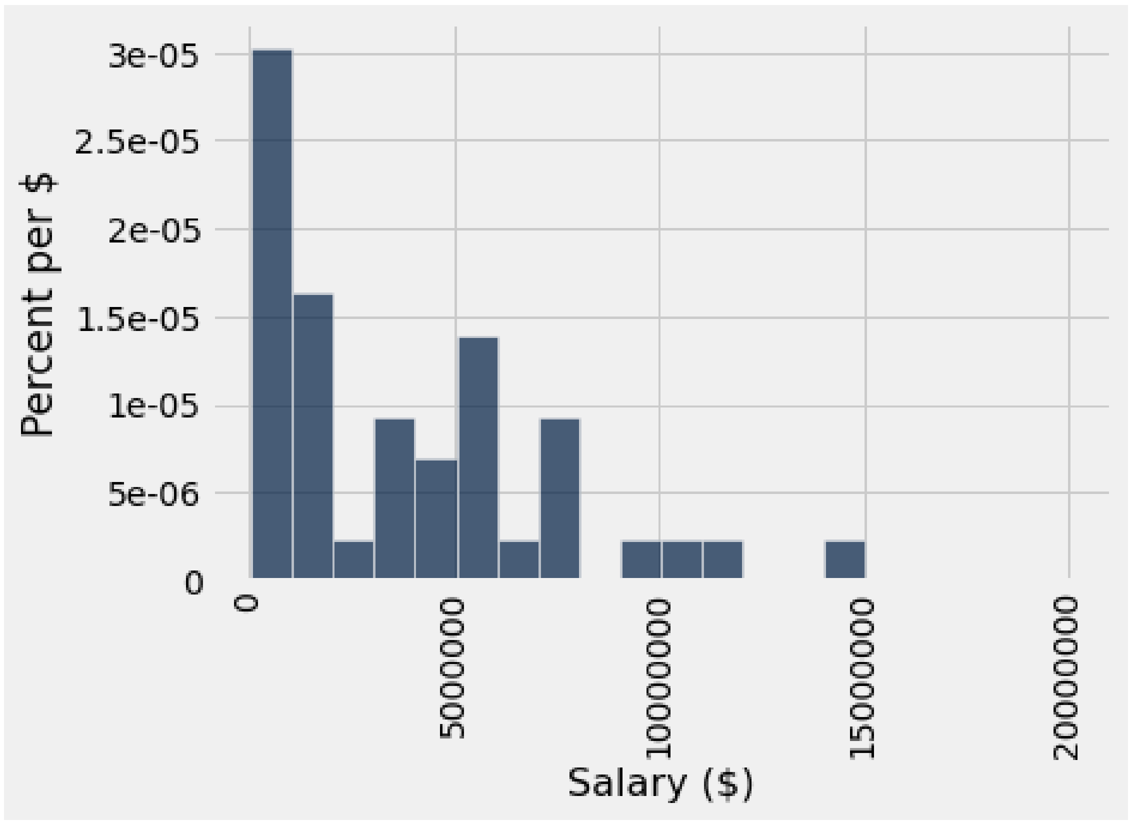
Percent per $
0
5e-06
1e-05
1.5e-05
2e-05
2.5e-05
3e-05
3.5e-05
1000000
2000000
3000000
4000000
5000000
Salary ($)

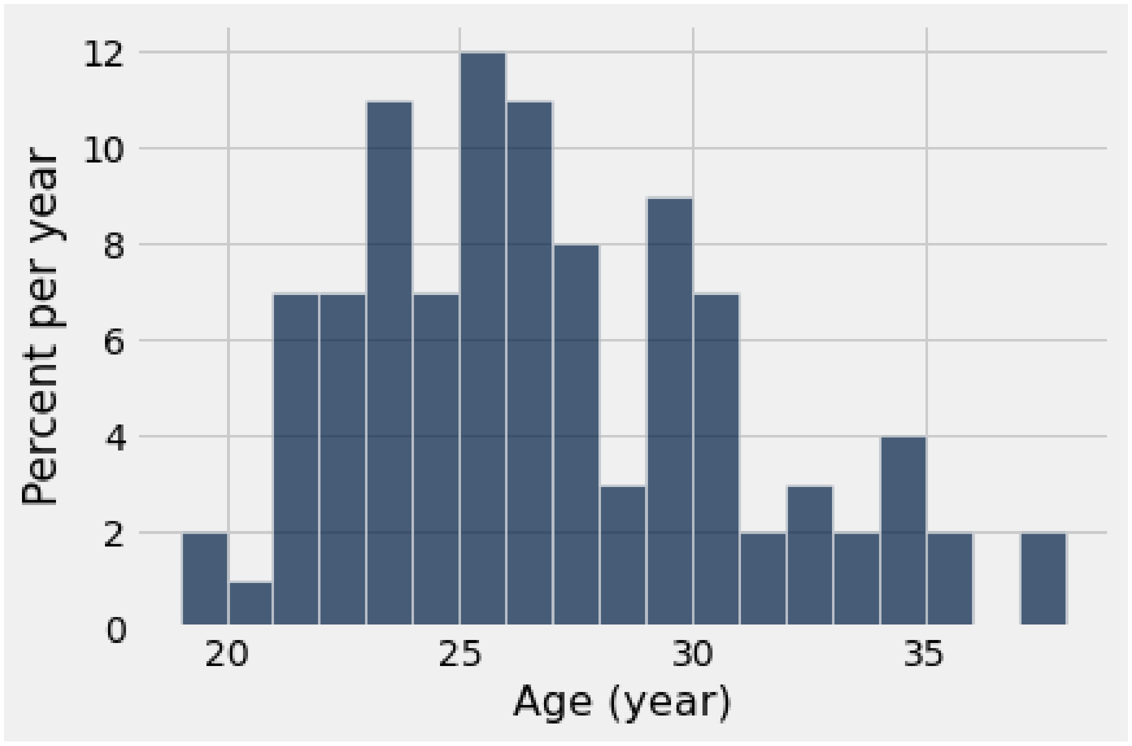


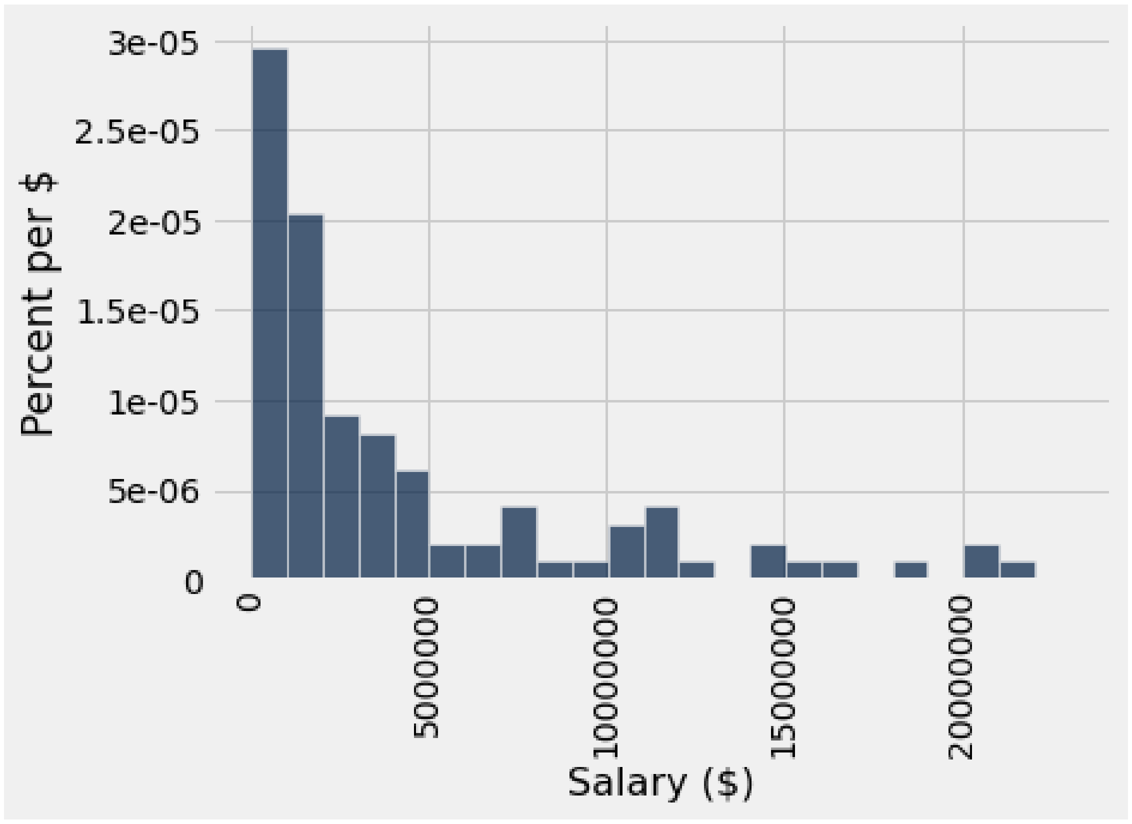
Percent per unit
3.5e-05
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000
All Players
Convenience Sample



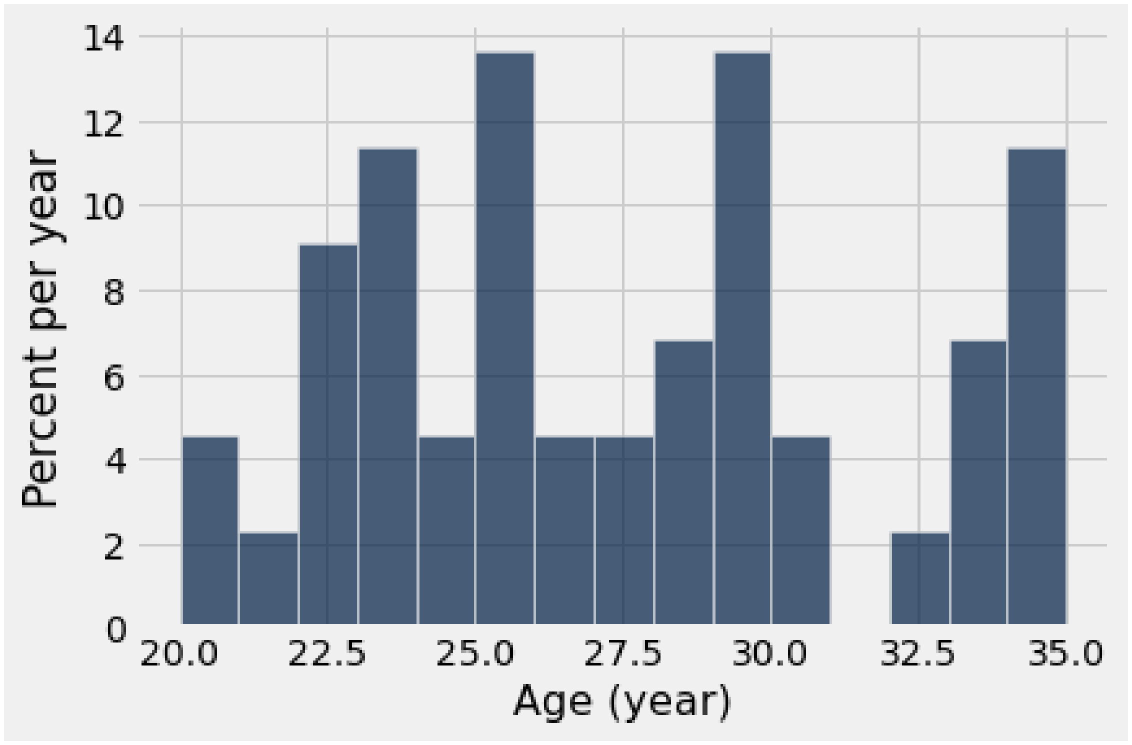
Percent per year
17.5
15
12.5
10
7.5
5
2.5
0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Age (year)


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

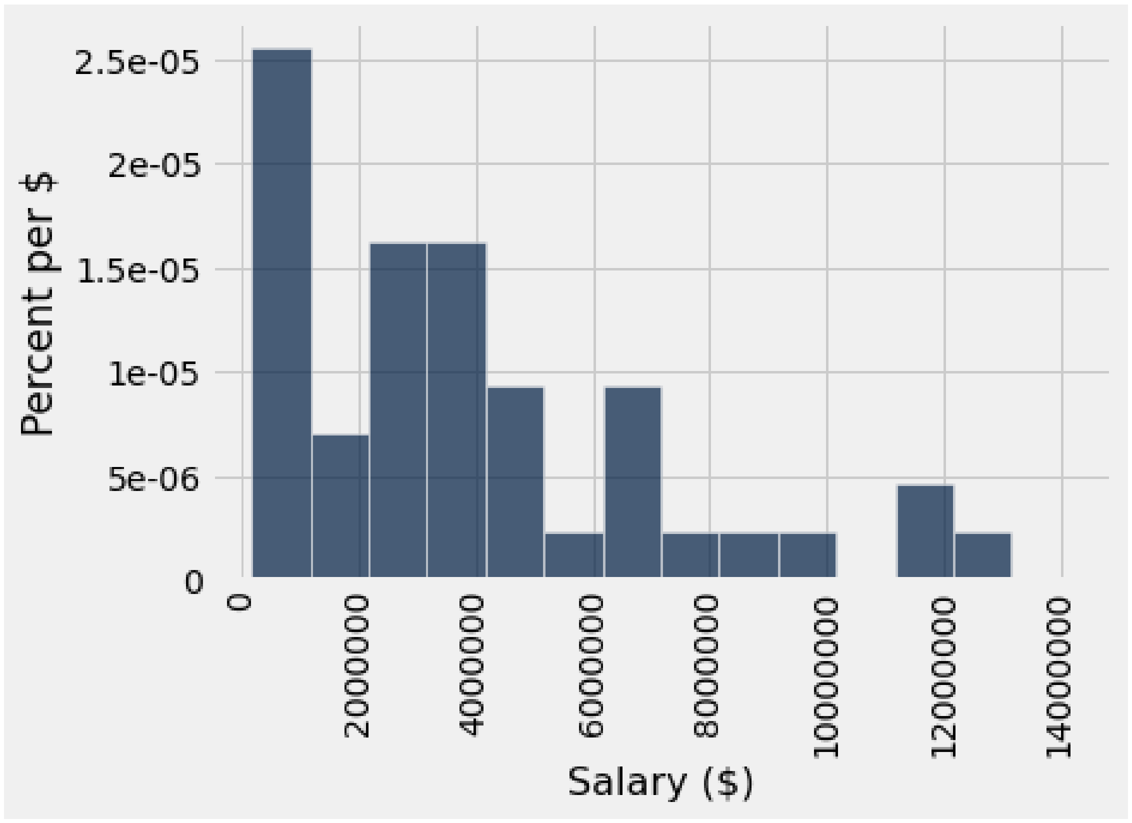

Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)

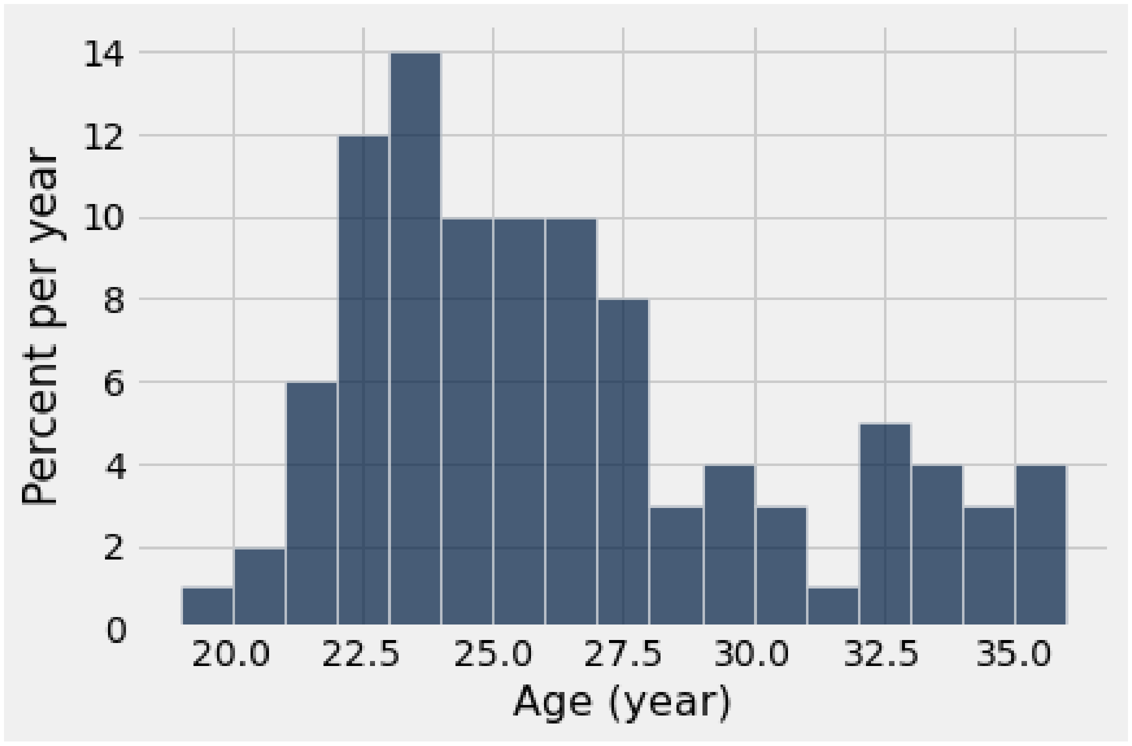


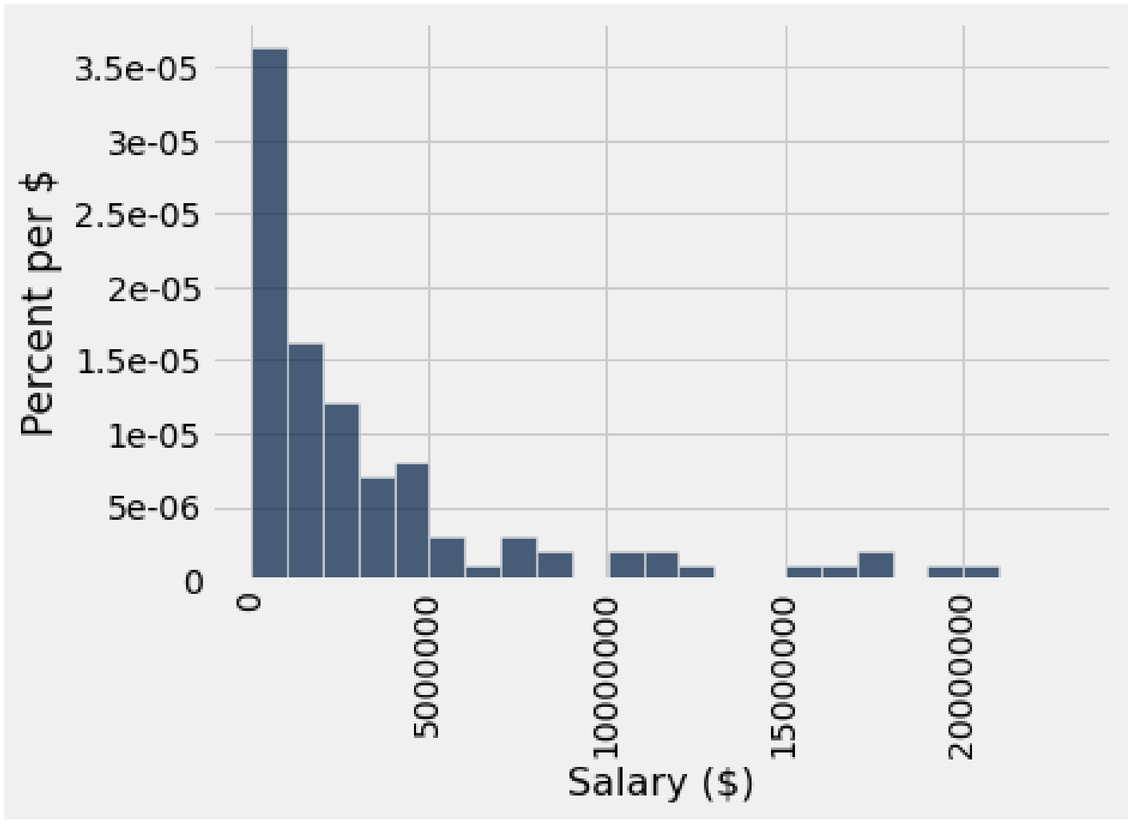
Percent per $
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

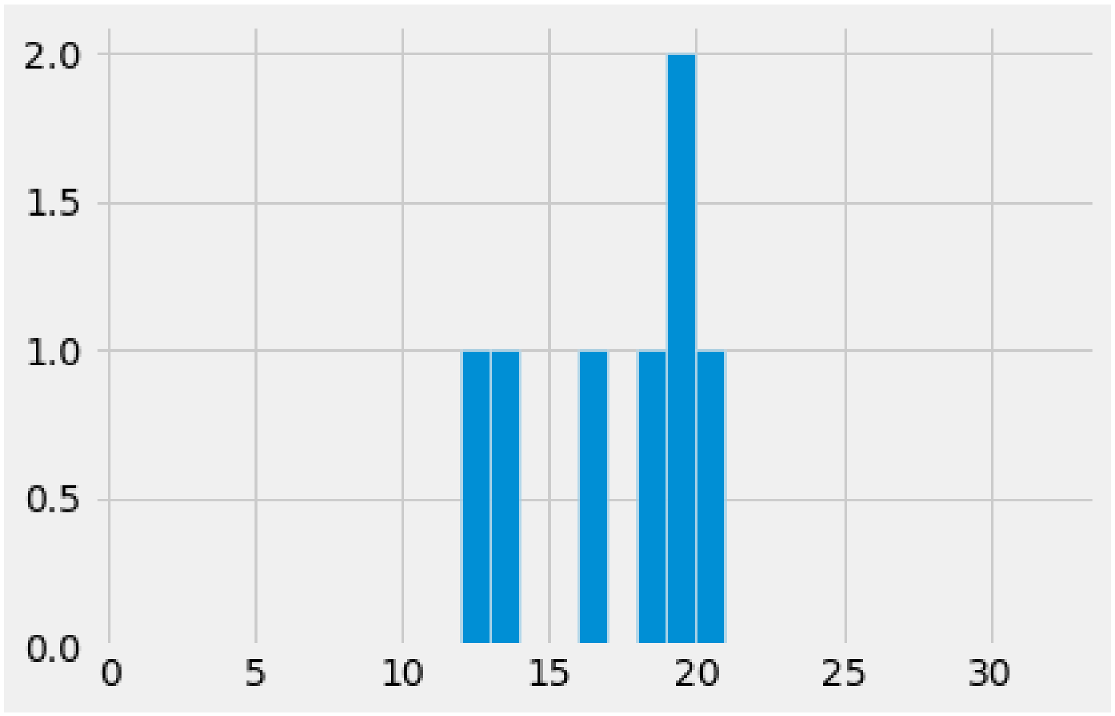


Percent per year
14
12
10
8
6
4
2
0
20.0
22.5
25.0
27.5
30.0
32.5
35.0
Age (year)

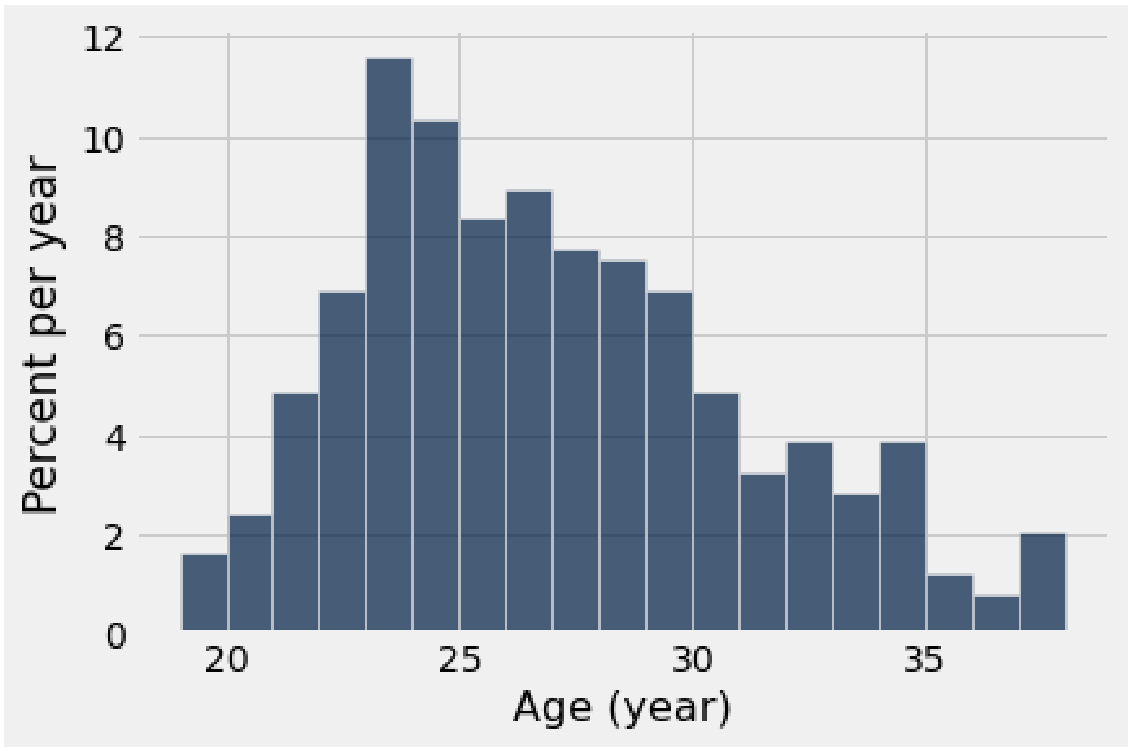

Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
Salary ($)


Percent per year
14
12
10
8
6
4
2
0
20.0
22.5
25.0
27.5
30.0
32.5
35.0
Age (year)

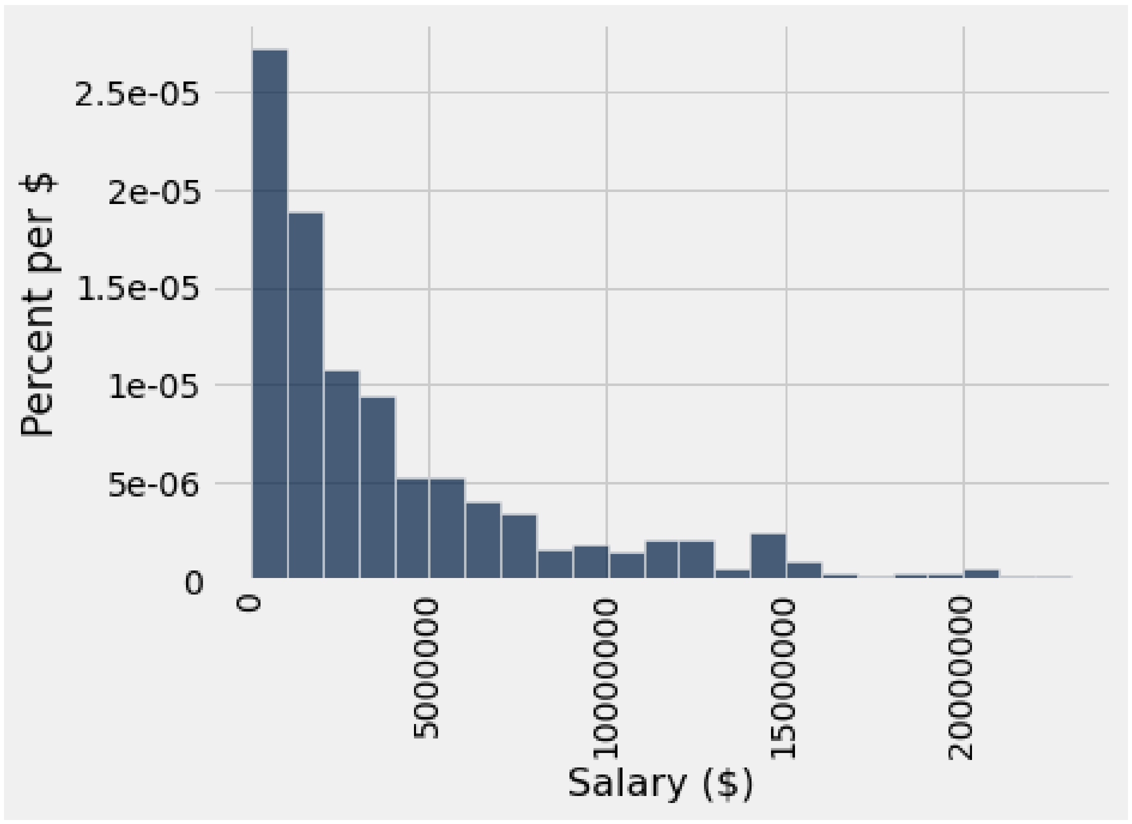

Percent per $
3.5e-05
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

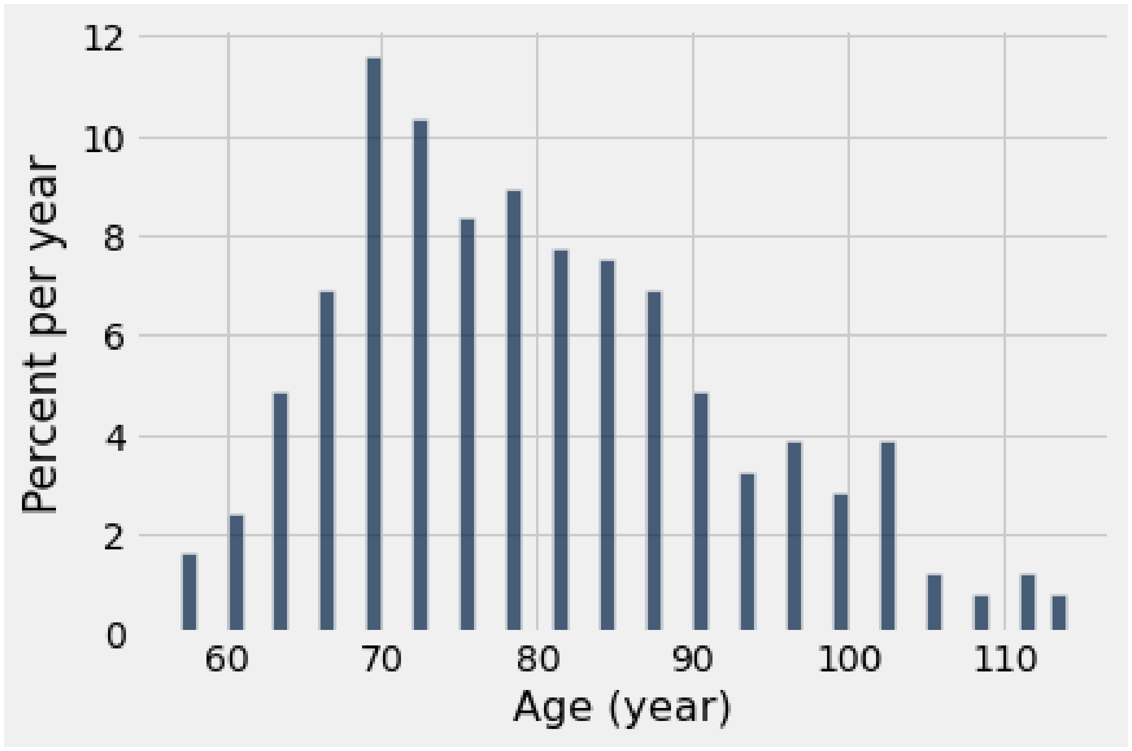


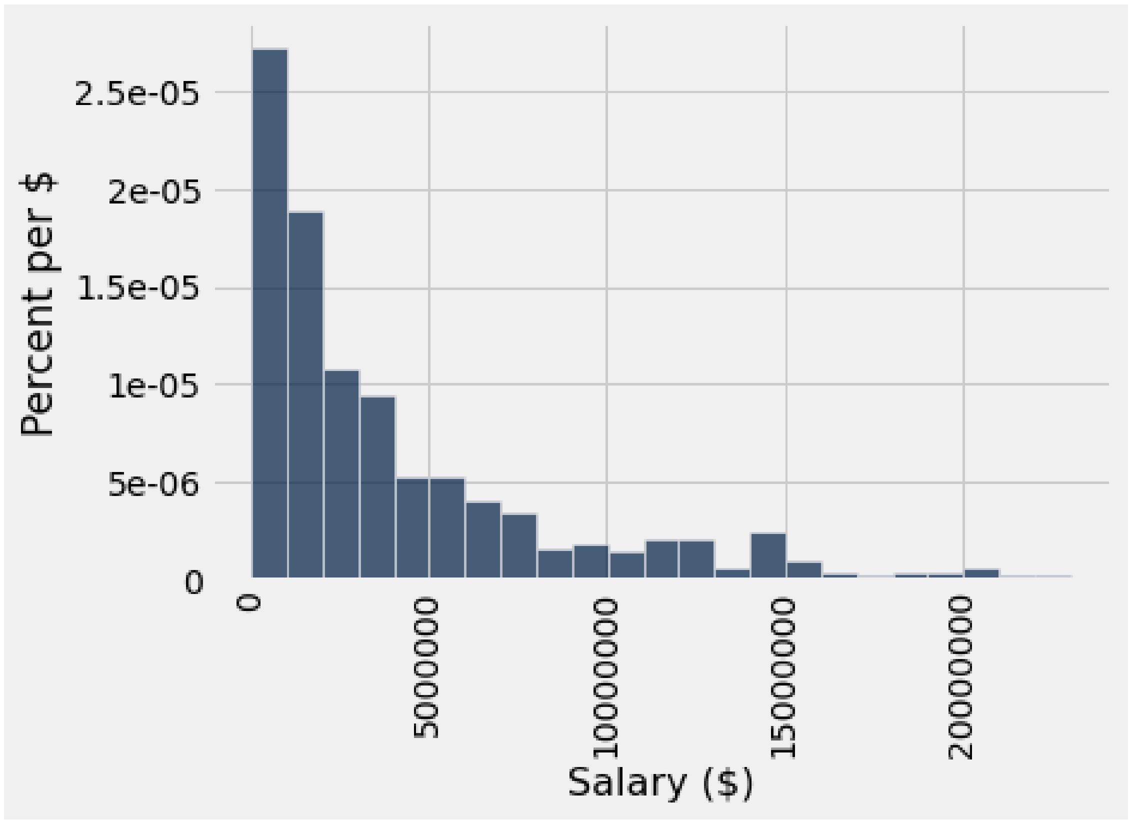
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30

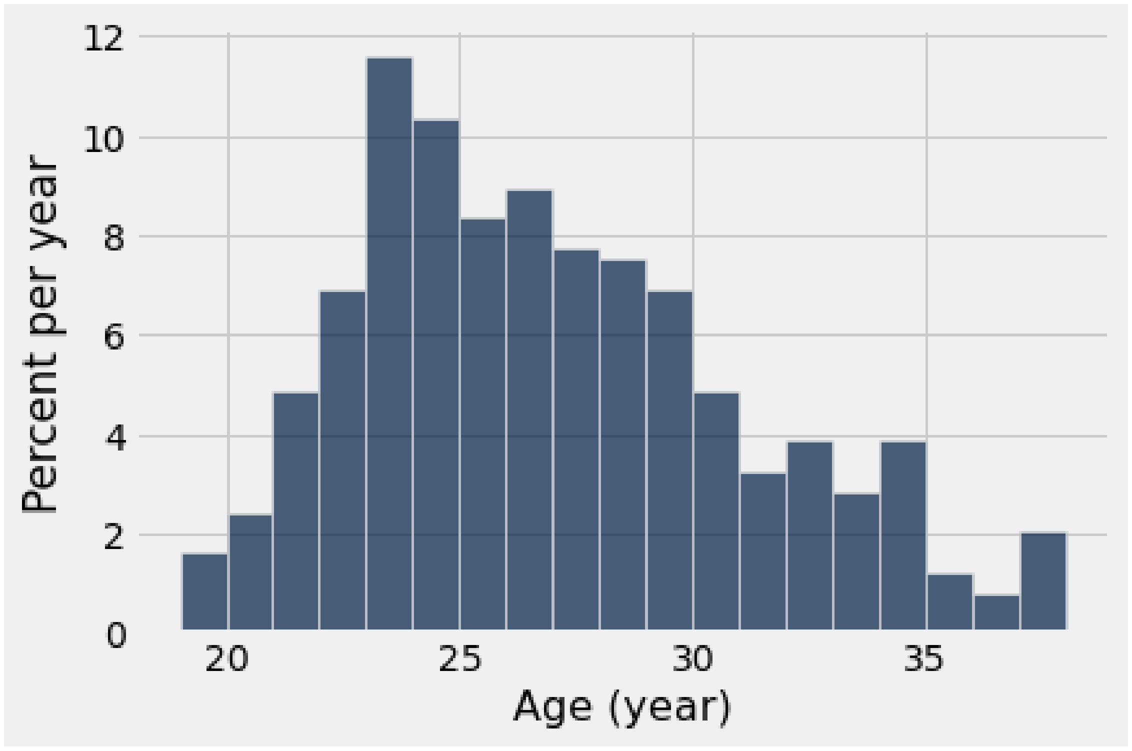


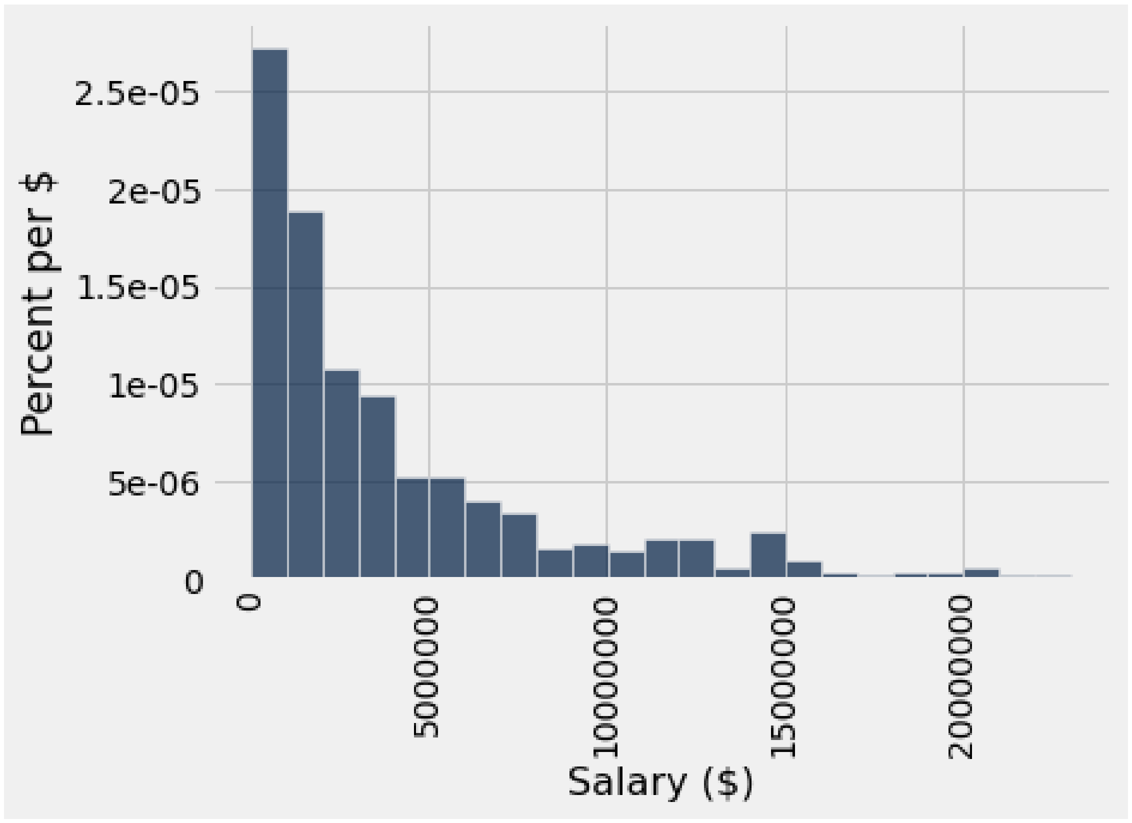
12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)


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

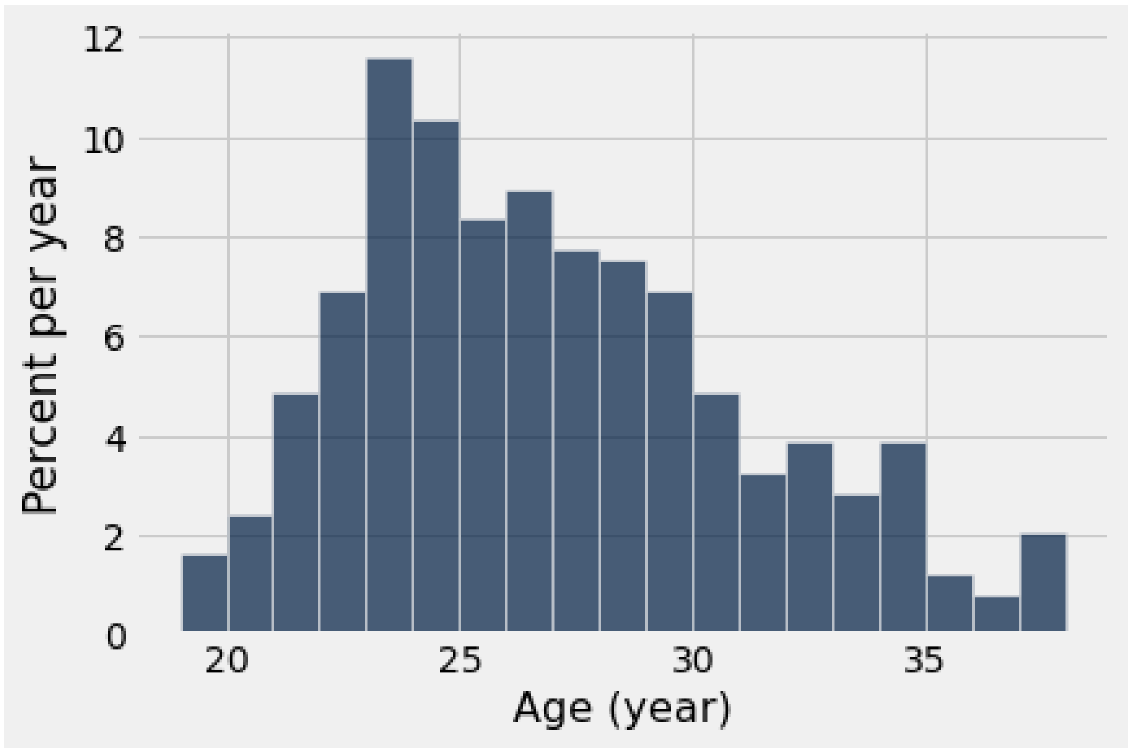

Percent per year
12
10
8
6
4
2
0
60
70
80
90
100
110
Age (year)

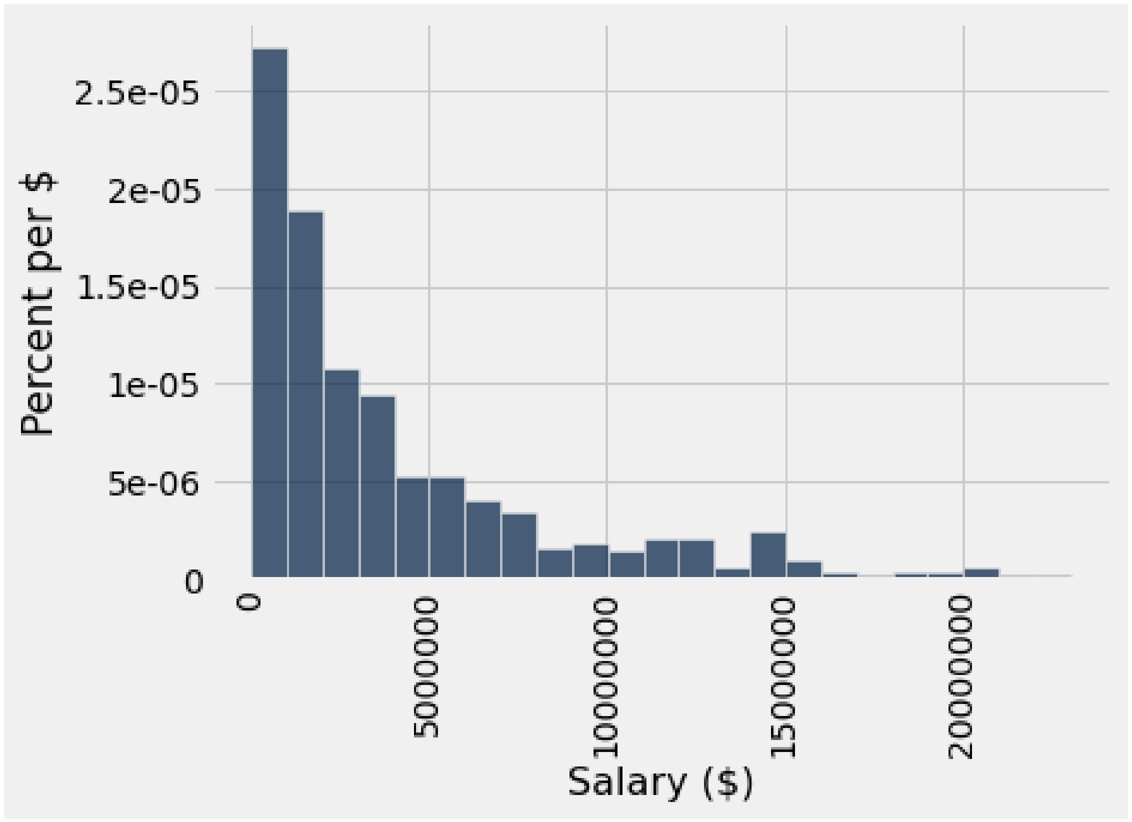

2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
5000000
10000000
15000000
20000000
Salary ($)

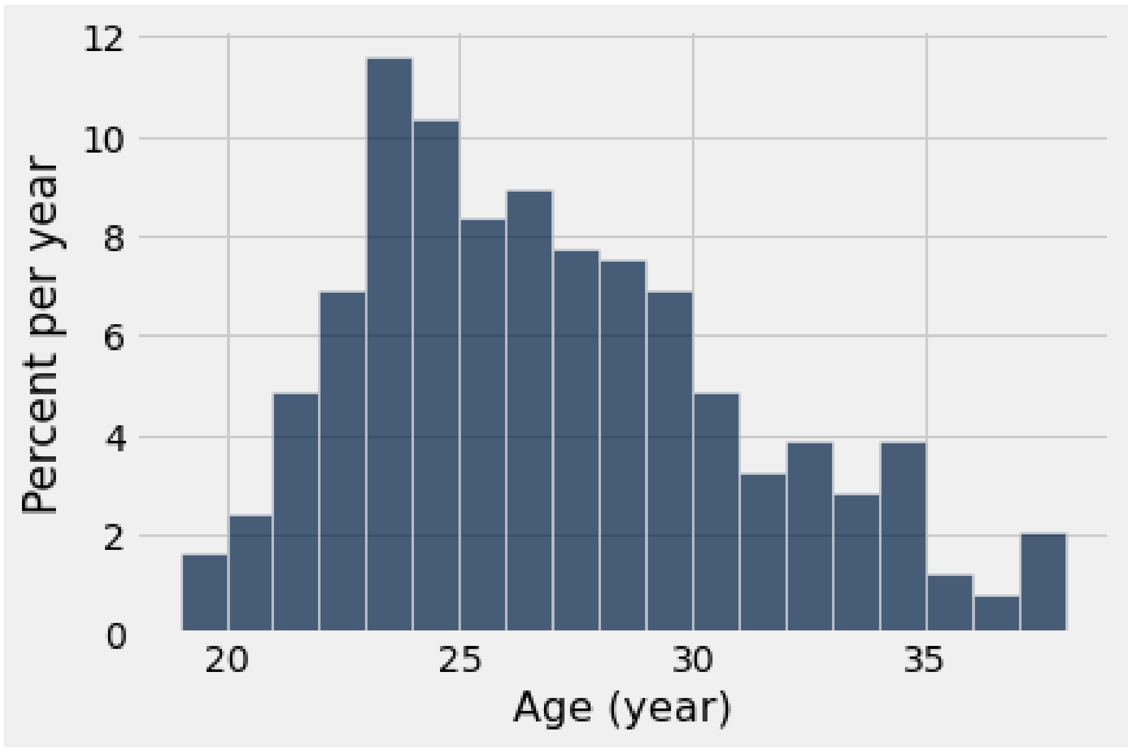

12
10
8
6
4
2
0
Percent per year
20
25
30
35
Age (year)

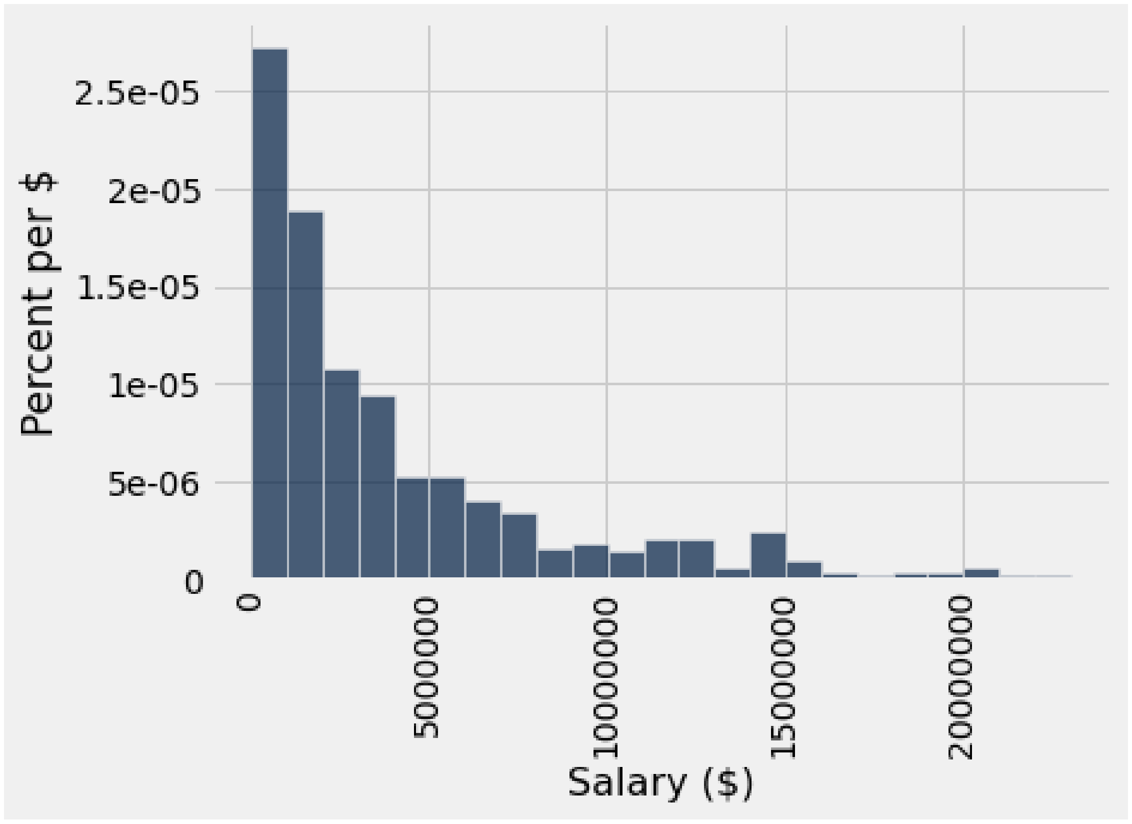


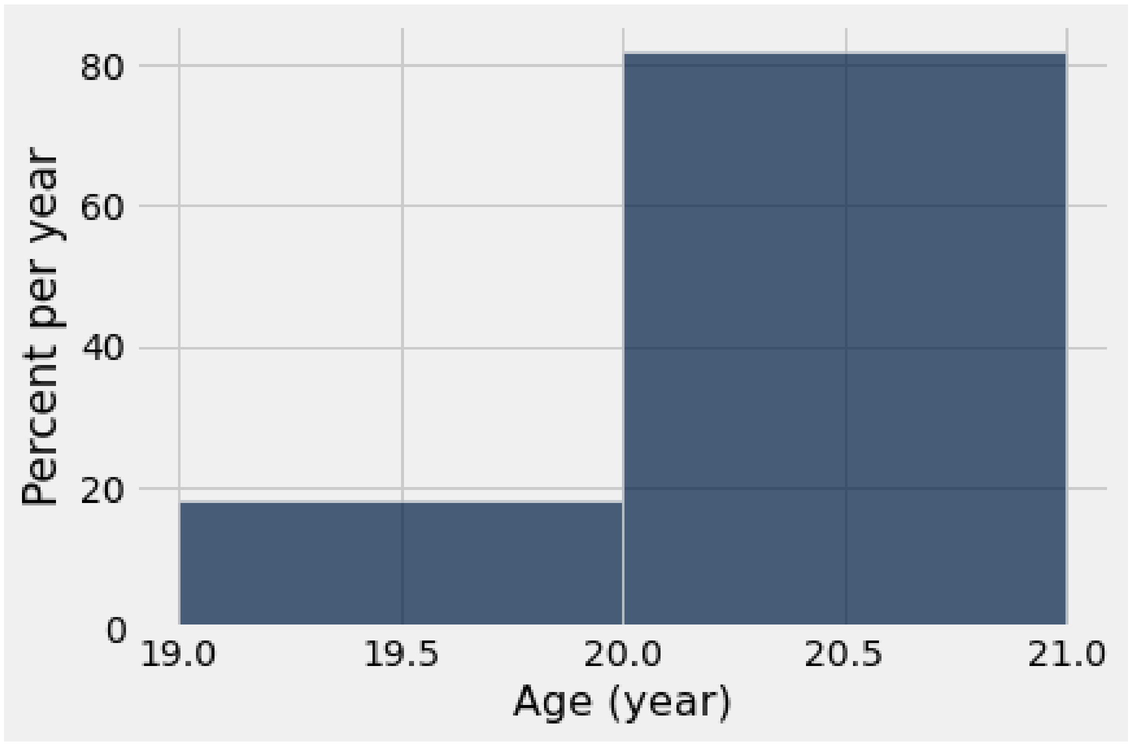
Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

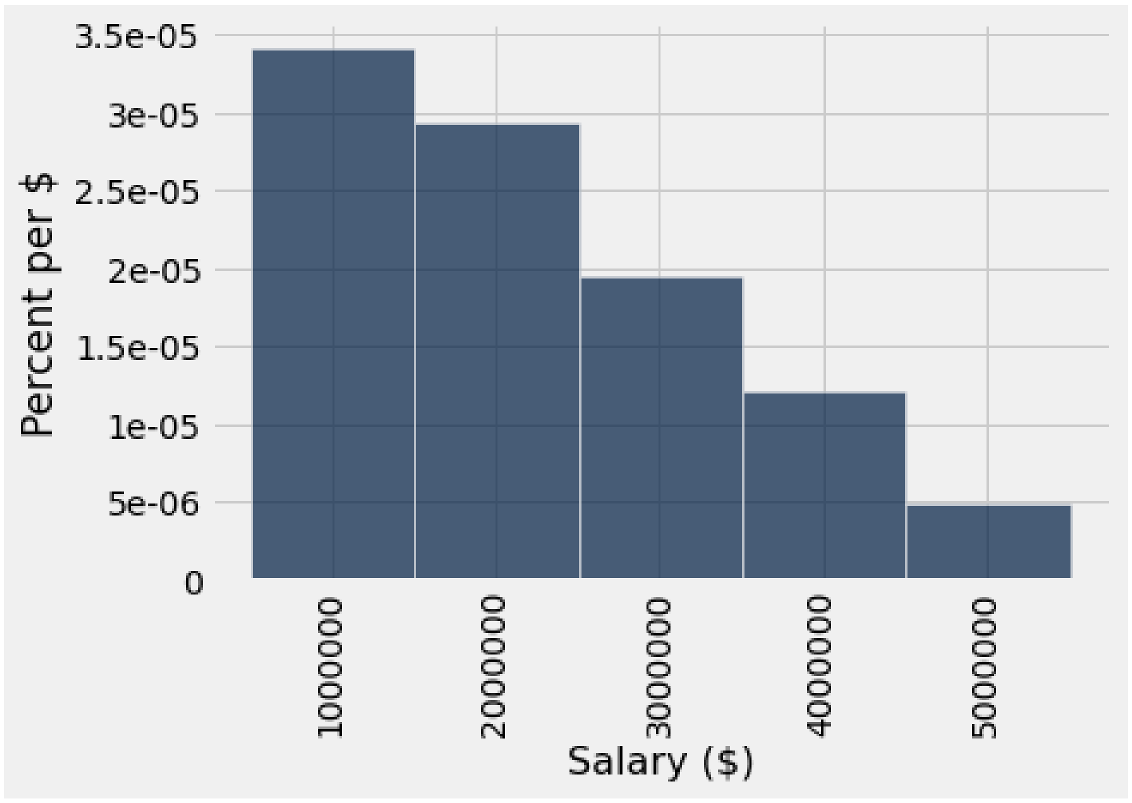

Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)

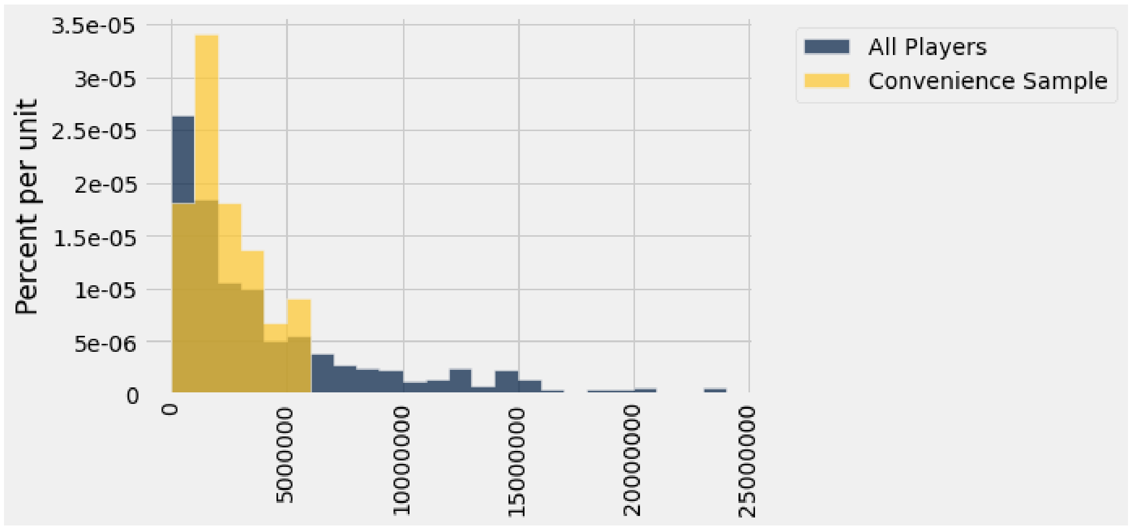

Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

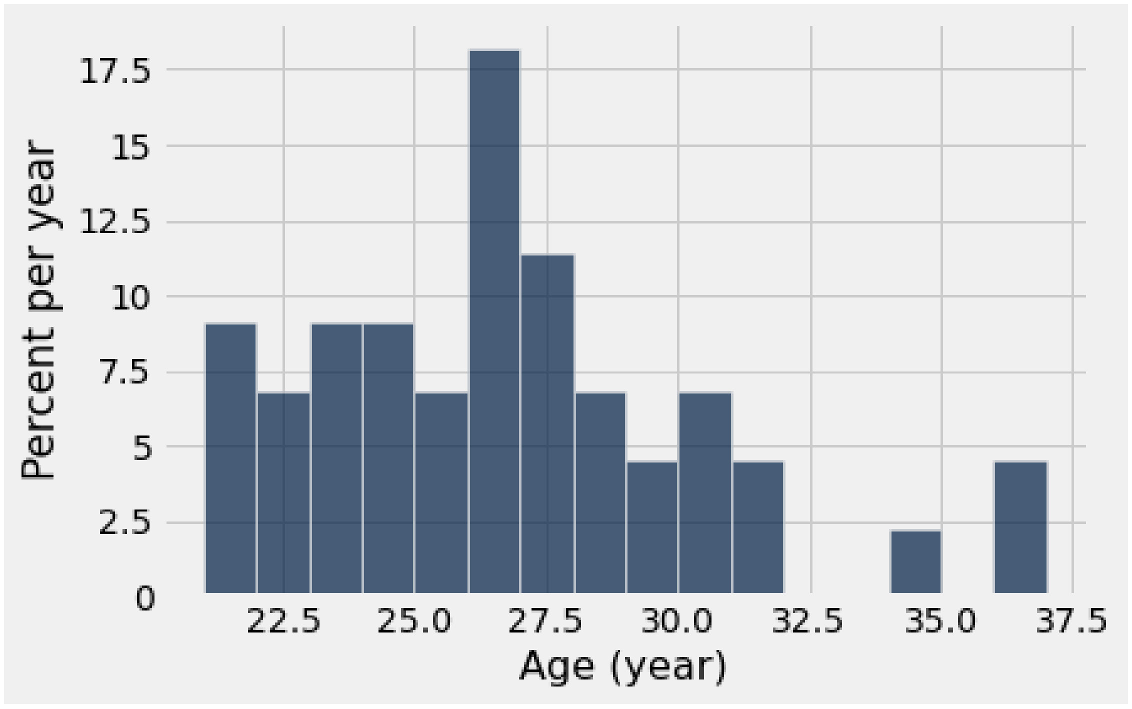

Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)

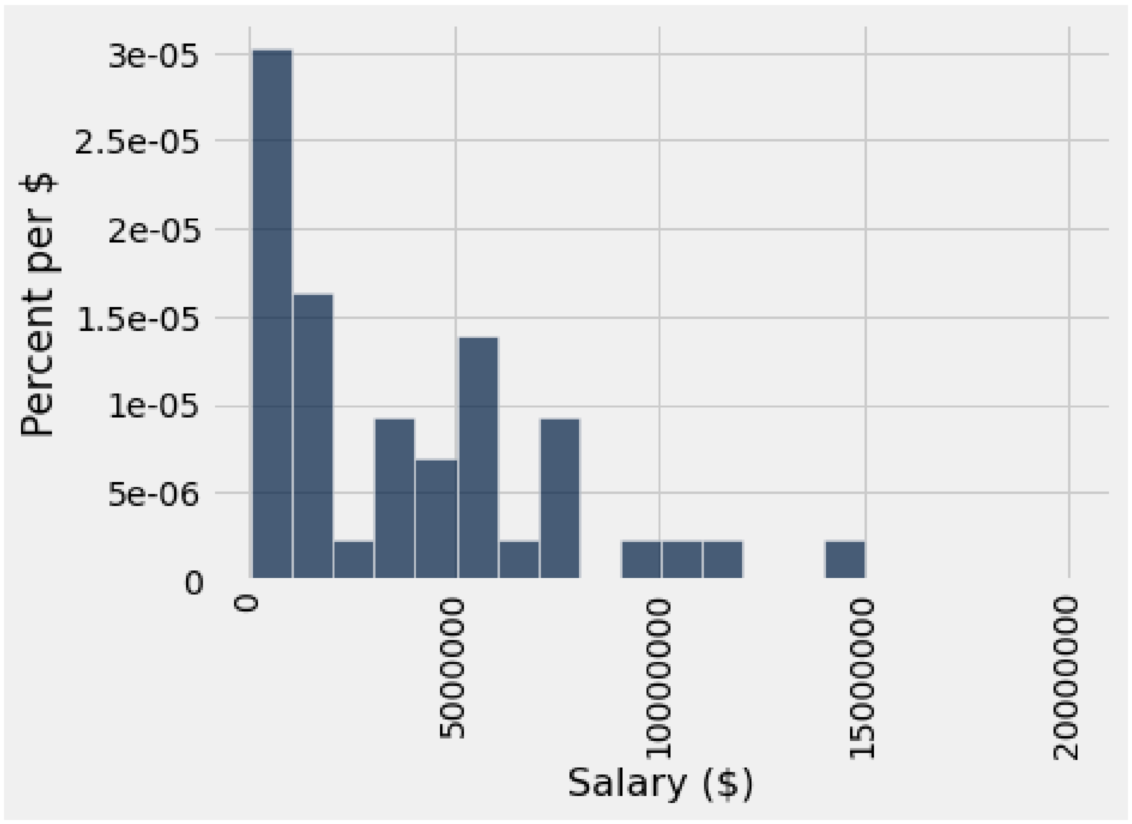

Percent per $
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)

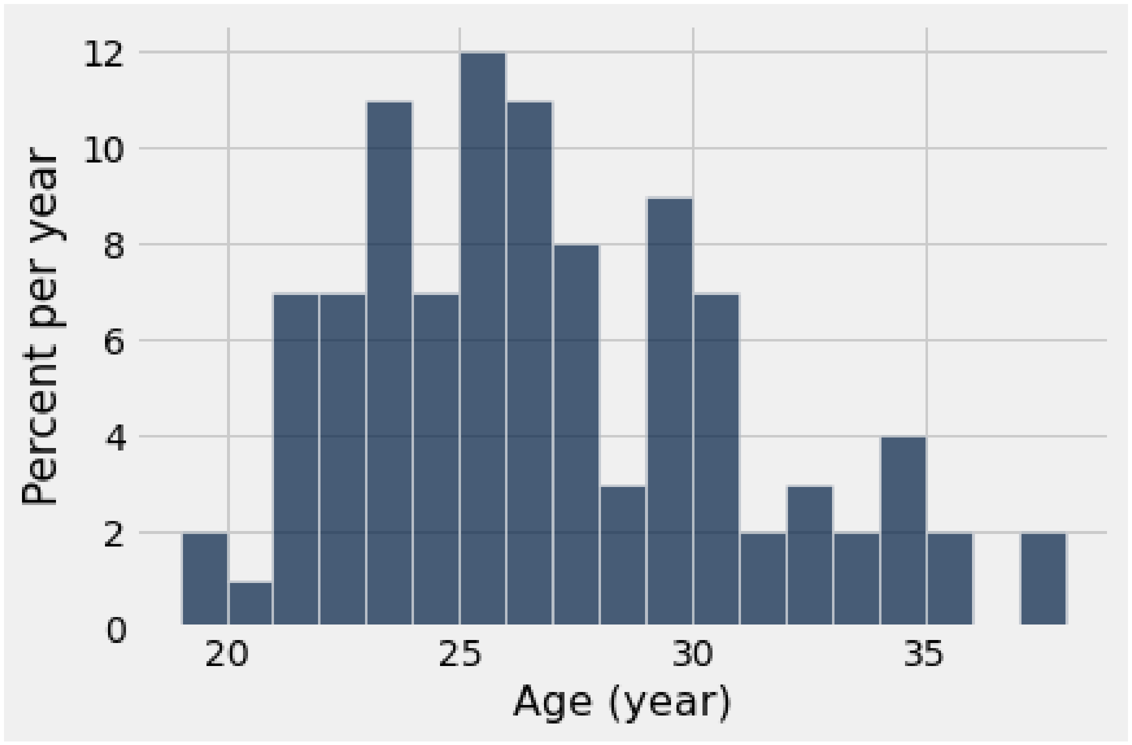

Percent per year
0
20
40
60
80
19.0
19.5
20.0
20.5
21.0
Age (year)

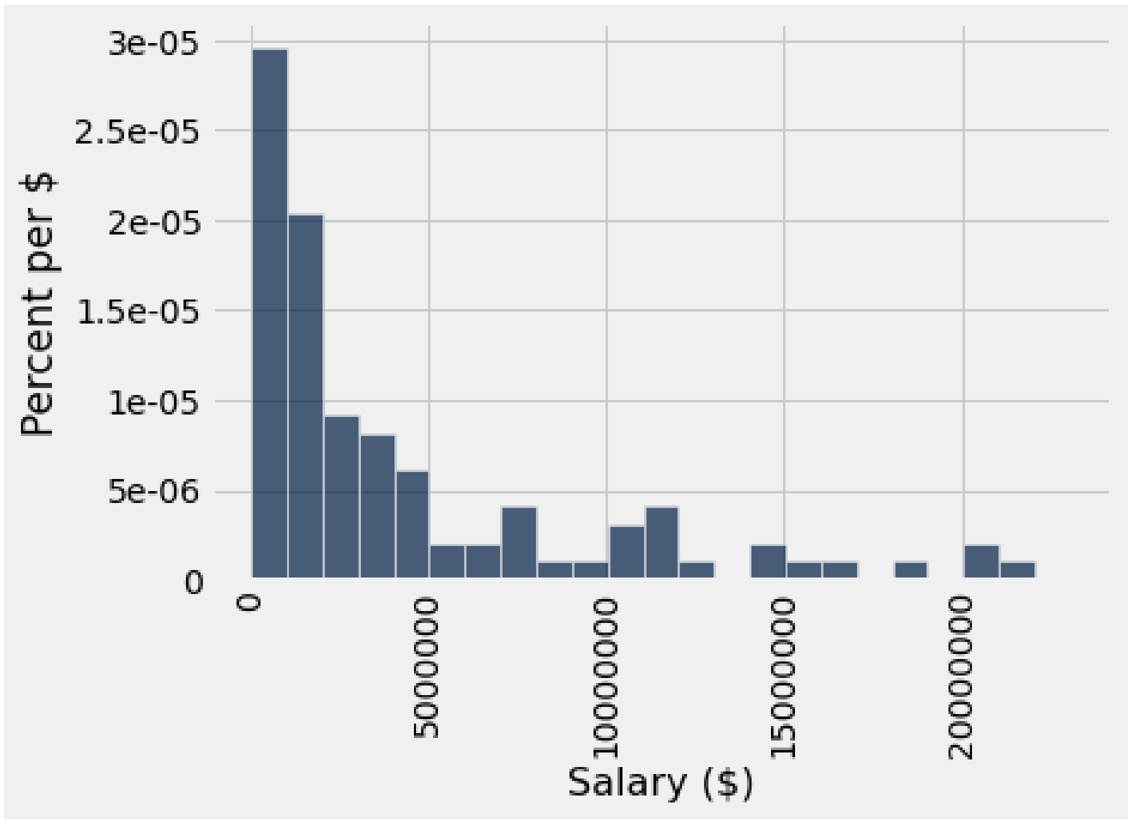

Percent per $
0
5e-06
1e-05
1.5e-05
2e-05
2.5e-05
3e-05
3.5e-05
1000000
2000000
3000000
4000000
5000000
Salary ($)


All Players
Convenience Sample
Percent per unit
3.5e-05
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
25000000

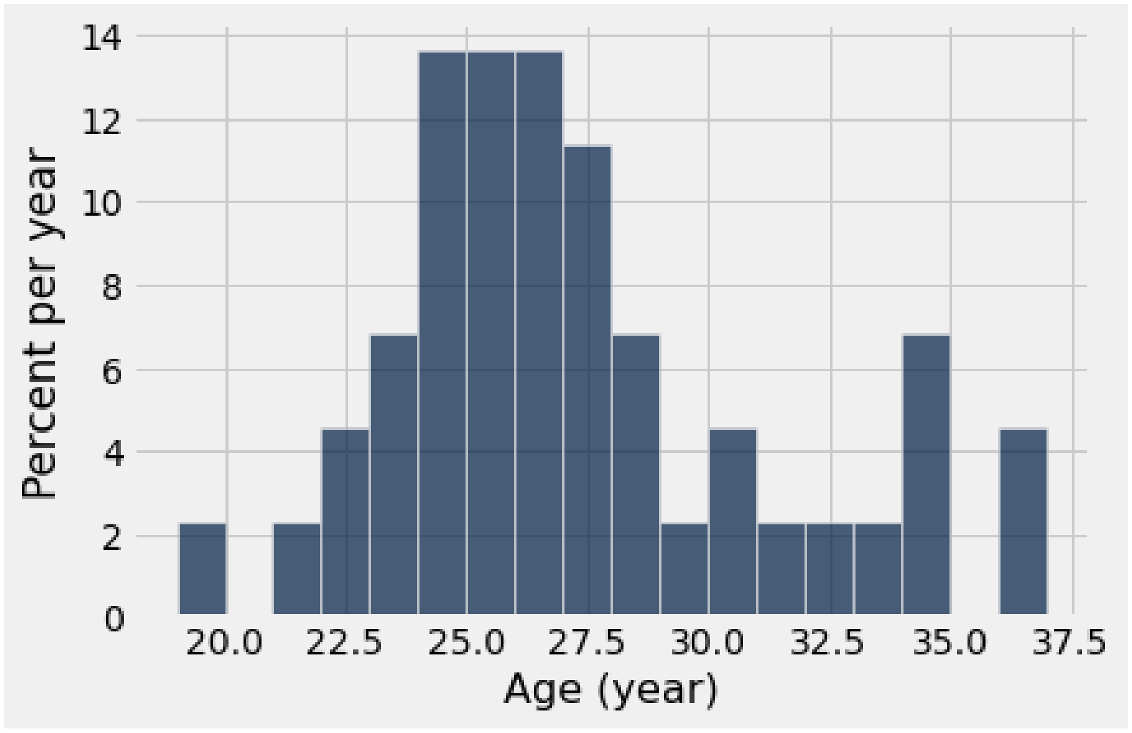

Percent per year
17.5
15
12.5
10
7.5
5
2.5
0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Age (year)


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

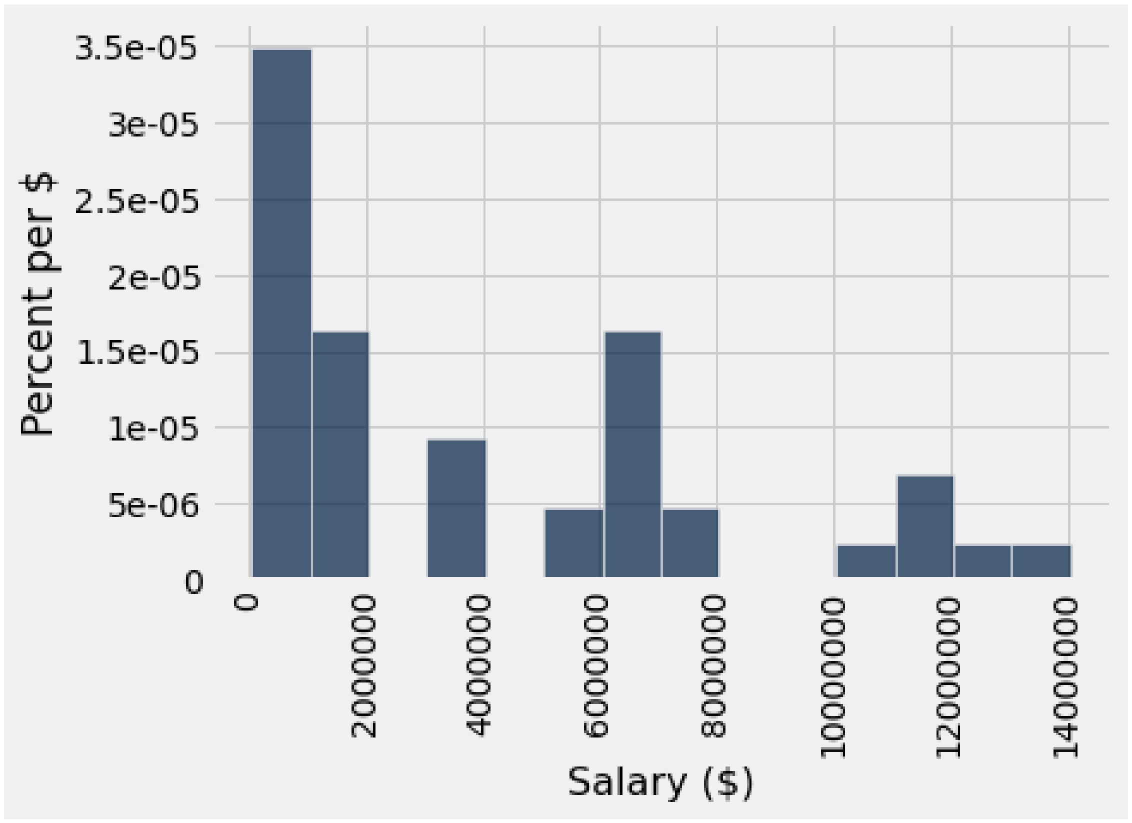

Percent per year
12
10
8
6
4
2
0
20
25
30
35
Age (year)



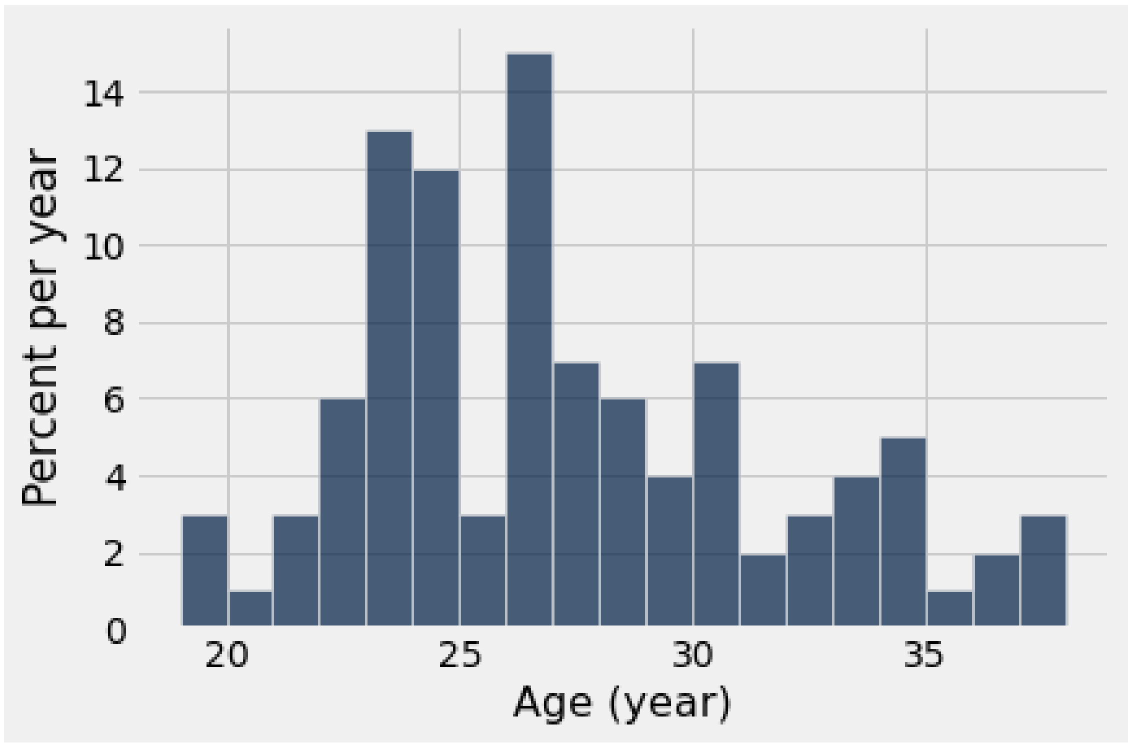
Percent per $
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
0
5000000
10000000
15000000
20000000
Salary ($)



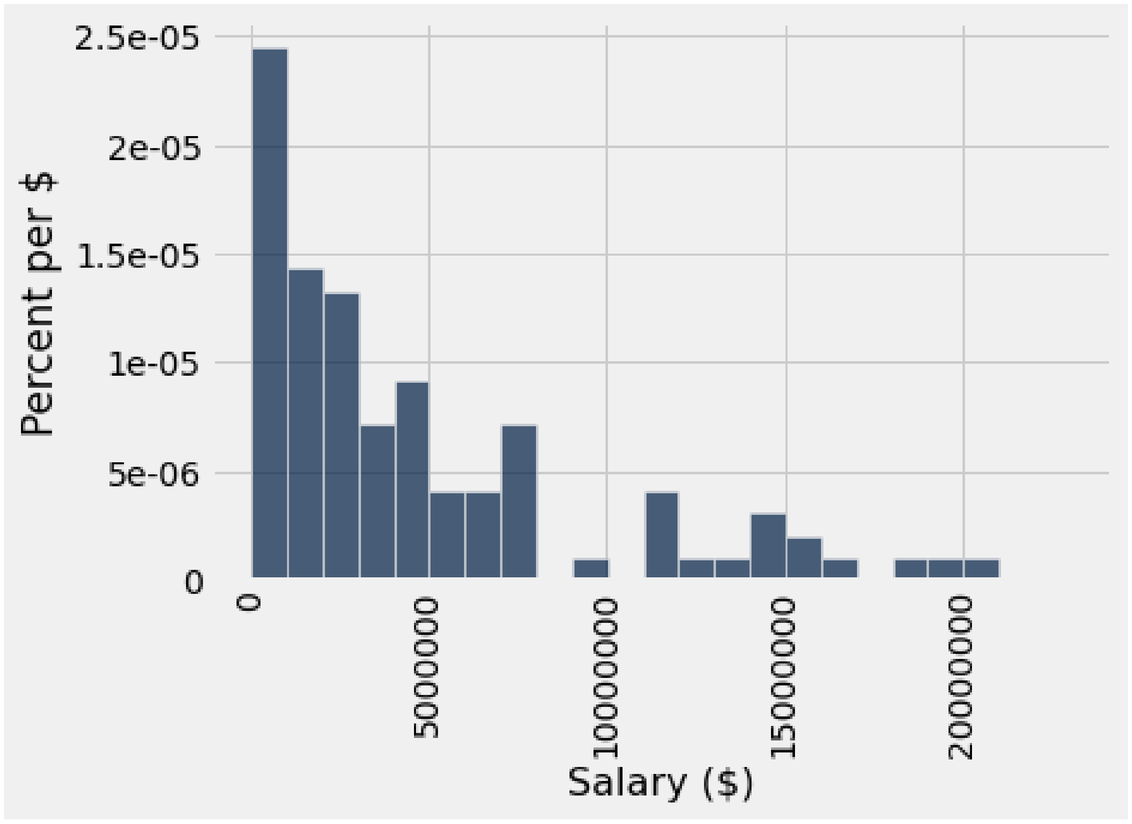
Percent per year
14
12
10
8
6
4
2
0
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
Age (year)



3.5e-05
3e-05
2.5e-05
2e-05
1.5e-05
1e-05
5e-06
0
Percent per $
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
Salary ($)



Percent per year
0
2
4
6
8
10
12
14
20
25
30
35
Age (year)



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

Name: Allan Gongora

Section: 0131