lab05

September 26, 2024

[36]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab05.ipynb")
```

1 Lab 5: Simulations

Welcome to Lab 5!

We will go over iteration and simulations, as well as introduce the concept of randomness.

The data used in this lab will contain salary data and other statistics for basketball players from the 2014-2015 NBA season. This data was collected from the following sports analytic sites: Basketball Reference and Spotrac.

First, set up the notebook by running the cell below.

[37]:
```
# 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 plt
plt.style.use('fivethirtyeight')
```

2 Lab Warm Up!

We will work together as a class in the following coding cells to prepare you for all sections of this lab.

Make sure to come to lab on time so you don't miss points for this warm-up!

[3]:
```
# Just run this cell
my_array = make_array(13,45,28,300,10,7,3,21,42)
my_array
```

[3]:
```
array([ 13,  45,  28, 300,  10,   7,   3,  21,  42])
```

[4]:
```python
# Getting one or more value(s) at random from array
at_random = np.random.choice(my_array)
at_random
```

[4]:
```
7
```

[5]:
```python
# Conditionals: if..elif..else
if at_random < 10:
    print('One digit')
elif at_random < 100:
    print('Two digits')
else:
    print('Big Number')
```

```
One digit
```

[34]:
```python
# Counting even numbers
# You can use % to find the remainder when you divide
# If a%2 == 0, then a is an even numner
my_array % 2 == 0
np.count_nonzero(my_array % 2 == 0)
```

[34]:
```
4
```

[35]:
```python
# Conditionals inside a function
def printNumDigits(num):
    if num < 0:
        print('One digit')
    elif num < 100:
        print('Two digits')
    else:
        print('Big Number')
printNumDigits(4)
```

```
Two digits
```

[36]:
```python
# For loops
# Examples from section 2:
# Example 1: Iterate over a container such as an array
rainbow = make_array("red", "orange", "yellow", "green", "blue", "indigo",␣
 ↪"violet")
for color in rainbow:
    print(color)

# Example 2: Using np.arange to iterate a number of times
```

```
#ADD MORE AFTER????????????????????????????????????????????????????????????
#for i in np.arange(5):
```

```
red
orange
yellow
green
blue
indigo
violet
```

[37]:
```
#Just run this cell. This data will also be used in section 3
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.
full_data.show(3)
```

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

[38]:
```
# Convenience sample: Suppose you only survey newer players (under 22 years old)
under_22 = np.average(full_data.where('Age', are.below(22)).column('Salary'))
all_players = np.average(full_data.column('Salary'))
under_22, all_players
# Does the convenience smaple give an acurate picture of the salary for all␣
 ↪players?
# Would you expect it to?
```

[38]:
```
(2383533.8181818184, 4269775.7662601629)
```

[40]:
```
# Random sample from a table
# np.random.choice is for arrays
# if we want to sample randomly rows from a table
# table.sample()
#
#full_data.sample(10) #default is with replacement
# simple random sample without replacement(srswor)

full_data.sample(10, with_replacement = False)
np.average(full_data.sample(30, with_replacement = False).column("Salary"))
```

[40]:
```
4441746.2666666666
```

2.1 1. Nachos and Conditionals

In Python, the boolean data type contains only two unique values: `True` and `False`. Expressions containing comparison operators such as `<` (less than), `>` (greater than), and `==` (equal to) evaluate to Boolean values. A list of common comparison operators can be found below!

Run the cell below to see an example of a comparison operator in action.

```
[41]: 3 > (1 + 1)
```

```
[41]: True
```

We can even assign the result of a comparison operation to a variable.

```
[42]: result = 10 / 2 == 5
      result
```

```
[42]: True
```

Arrays are compatible with comparison operators. The output is an array of boolean values.

```
[ ]: make_array(1, 5, 7, 8, 3, -1) > 3
```

One day, when you come home after a long week, you see a hot bowl of nachos waiting on the dining table! Let's say that whenever you take a nacho from the bowl, it will either have only **cheese**, only **salsa**, **both** cheese and salsa, or **neither** cheese nor salsa (a sad tortilla chip indeed).

Let's try and simulate taking nachos from the bowl at random using the function, `np.random.choice(...)`.

2.1.1 `np.random.choice`

`np.random.choice` picks one item at random from the given array. It is equally likely to pick any of the items. Run the cell below several times, and observe how the results change.

```
[26]: nachos = make_array('cheese', 'salsa', 'both', 'neither')
      np.random.choice(nachos)
```

```
[26]: 'neither'
```

To repeat this process multiple times, pass in an int `n` as the second argument to return `n` different random choices. By default, `np.random.choice` samples **with replacement** and returns an *array* of items. Sampling **with replacement** means if we sample `n` times, each time, every element has an equal chance of being selected.

Run the next cell to see an example of sampling with replacement 10 times from the `nachos` array.

```
[27]: np.random.choice(nachos, 10)
```

```
[27]: array(['salsa', 'both', 'salsa', 'cheese', 'cheese', 'neither', 'salsa',
             'both', 'cheese', 'both'],
      dtype='<U7')
```

To count the number of times a certain type of nacho is randomly chosen, we can use `np.count_nonzero`

2.1.2 `np.count_nonzero`

`np.count_nonzero` counts the number of non-zero values that appear in an array. When an array of boolean values are passed through the function, it will count the number of `True` values (remember that in Python, `True` is coded as 1 and `False` is coded as 0.)

Run the next cell to see an example that uses `np.count_nonzero`.

[28]:
```
np.count_nonzero(make_array(True, False, False, True, True))
```

[28]:
```
3
```

Question 1.1 Assume we took ten nachos at random, and stored the results in an array called `ten_nachos` as done below. Find the number of nachos with only cheese using code (do not hardcode the answer).

[29]:
```
ten_nachos = make_array('neither', 'cheese', 'both', 'both', 'cheese', 'salsa',␣
 ↪'both', 'neither', 'cheese', 'both')
# HINT: Our solution involves a comparison operator (e.g. `==`, `<`, ...) and␣
 ↪the `np.count_nonzero` method.
number_cheese = np.count_nonzero(ten_nachos == 'cheese')
number_cheese
```

[29]:
```
3
```

[30]:
```
grader.check("q11")
```

[30]:
```
q11 results: All test cases passed!
```

Conditional Statements

A conditional statement is a multi-line statement that allows Python to choose among different alternatives based on the truth value of an expression.

Here is a basic example.

```
def sign(x):
    if x > 0:
        return 'Positive'
    else:
        return 'Negative'
```

If the input `x` is greater than `0`, we return the string `'Positive'`. Otherwise, we return `'Negative'`.

If we want to test multiple conditions at once, we use the following general format.

```
if <if expression>:
    <if body>
elif <elif expression 0>:
    <elif body 0>
elif <elif expression 1>:
    <elif body 1>
...
else:
    <else body>
```

Only the body for the first conditional expression that is true will be evaluated. Each `if` and `elif` expression is evaluated and considered in order, starting at the top. `elif` can only be used if an `if` clause precedes it. As soon as a true value is found, the corresponding body is executed, and the rest of the conditional statement is skipped. If none of the `if` or `elif` expressions are true, then the `else body` is executed.

For more examples and explanation, refer to the section on conditional statements here.

Question 1.2 Complete the following conditional statement so that the string `'More please'` is assigned to the variable `say_please` if the number of nachos with cheese in `ten_nachos` is less than `5`. Use the if statement to do this (do not directly reassign the variable `say_please`).

[31]:
```
#HINT: You should be using the variable `number_cheese` from Question 1.
say_please = '?'

if number_cheese < 5:
    say_please = 'More please'
say_please
```

[31]:
```
'More please'
```

[10]:
```
grader.check("q12")
```

[10]:
```
q12 results: All test cases passed!
```

Question 1.3 Write a function called `nacho_reaction` that returns a reaction (as a string) based on the type of nacho passed in as an argument. Use the table below to match the nacho type to the appropriate reaction.

[32]:
```
# HINT 1: Double check the spelling of your reactions.
# HINT 2: Make sure that you are comparing and returning strings
def nacho_reaction(nacho):
    if nacho == "cheese":
        return 'Cheesy!'
    elif nacho == "salsa":
        return 'Spicy!'
    elif nacho == "both":
        return 'Wow!'
    elif nacho == 'neither':
        return 'Meh.'

spicy_nacho = nacho_reaction('salsa')
spicy_nacho
```

[32]:
```
'Spicy!'
```

[18]:
```
grader.check("q13")
```

[18]:
```
q13 results: All test cases passed!
```

Question 1.4 Create a table `ten_nachos_reactions` that consists of the nachos in `ten_nachos` as well as the reactions for each of those nachos. The columns should be called `Nachos` and `Reactions`.

[33]:
```
ten_nachos_tbl = Table().with_column('Nachos', ten_nachos)
# STEP 1: Create an array of reactions using the apply method with the function␣
 ↪nacho_reaction that was created in the previous question
reactions_array = ten_nachos_tbl.apply(nacho_reaction, 'Nachos')
# STEP 2: Add the array to the table as a column
ten_nachos_reactions = ten_nachos_tbl.with_column('Reactions', reactions_array)
ten_nachos_reactions
```

[33]:
```
Nachos  | Reactions
neither | Meh.
cheese  | Cheesy!
both    | Wow!
both    | Wow!
cheese  | Cheesy!
salsa   | Spicy!
both    | Wow!
neither | Meh.
cheese  | Cheesy!
both    | Wow!
```

[20]:
```
grader.check("q14")
```

[20]:
```
q14 results: All test cases passed!
```

Question 1.5 Using code, find the number of 'Wow!' reactions for the nachos in `ten_nachos_reactions`.

[38]:
```
number_wow_reactions = np.count_nonzero(ten_nachos_reactions.
 ↪column('Reactions') == 'Wow!')
number_wow_reactions
```

[38]:
```
4
```

[39]:
```
grader.check("q15")
```

[39]:
```
q15 results: All test cases passed!
```

2.2 2. Simulations and For Loops

Using a `for` statement, we can perform a task multiple times. This is known as iteration. The general structure of a for loop is:

`for <placeholder> in <array>:` followed by indented lines of code that are repeated for each element of the `array` being iterated over. You can read more about for loops here.

NOTE: We often use `i` as the `placeholder` in our class examples, but you could name it anything! Some examples can be found below.

One use of iteration is to loop through a set of values. For instance, we can print out all of the colors of the rainbow.

[40]:
```
rainbow = make_array("red", "orange", "yellow", "green", "blue", "indigo",␣
 ↪"violet")

for color in rainbow:
    print(color)
```

```
red
orange
yellow
green
blue
indigo
violet
```

We can see that the indented part of the `for` loop, known as the body, is executed once for each item in `rainbow`. The name `color` is assigned to the next value in `rainbow` at the start of each iteration. Note that the name `color` is arbitrary; we could easily have named it something else. The important thing is we stay consistent throughout the `for` loop.

[ ]:
```
for another_name in rainbow:
    print(another_name)
```

In general, however, we would like the variable name to be somewhat informative.

Question 2.1 In the following cell, we've loaded the text of *Pride and Prejudice* by Jane Austen, split it into individual words, and stored these words in an array `p_and_p_words`. Using a `for` loop, assign `longer_than_five` to the number of words in the novel that are more than 5 letters long.

[47]:
```
austen_string = open('Austen_PrideAndPrejudice.txt', encoding='utf-8').read()
p_and_p_words = np.array(austen_string.split())
# HINT 1: For the if statement: use the function len to find the number of␣
 ↪letters in a word and compare it to 5.
```

```
# HINT 2: How can you use longer_than_five to keep track of the number of words␣
↪that are longer than 5 letters?
longer_than_five = 0
for word in p_and_p_words:
    if len(word) > 5:
        longer_than_five += 1
longer_than_five
```

[47]:
```
35453
```

[48]:
```
grader.check("q21")
```

[48]:
```
q21 results: All test cases passed!
```

Another way we can use `for` loops is to repeat lines of code many times. Recall the structure of a `for` loop:

`for <placeholder> in <array>:` followed by indented lines of code that are repeated for each element of the array being iterated over.

Sometimes, we don't care about what the value of the placeholder is. We instead take advantage of the fact that the `for` loop will repeat as many times as the length of our array. In the following cell, we iterate through an array of length 5 and print out "Hello, world!" in each iteration.

[49]:
```
for i in np.arange(5):
    print("Hello, world!")
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

Question 2.2 Using a simulation with 10,000 trials, assign `num_different` to the number of times, in 10,000 trials, that two words picked uniformly at random (with replacement) from Pride and Prejudice have different lengths.

[52]:
```
# HINT 1: What function did we use in section 1 to sample at random with␣
↪replacement from an array?
# Can you use it again inside the loop?
# HINT 2: Remember that != checks for non-equality between 2 items. Can you use␣
↪it in combination
# with the len function to compare the length of the 2 words inside the loop?
trials = 10000
num_different = 0 #counter

for i in np.arange(trials):
    word1 = np.random.choice(p_and_p_words)
    word2 = np.random.choice(p_and_p_words)
    if len(word1) != len(word2):
        num_different += 1
num_different
```

[52]:
```
8657
```

[53]:
```
grader.check("q22")
```

[53]:
```
q22 results: All test cases passed!
```

2.3 3. Sampling Basketball Data

We will now introduce the topic of sampling, which we'll be discussing in more depth in this week's lectures. We'll guide you through this code, but if you wish to read more about different kinds of samples before attempting this question, you can check out section 10 of the textbook.

Run the cell below to load player and salary data that we will use for our sampling.

[54]:
```
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 (as in the tables loaded above), 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.

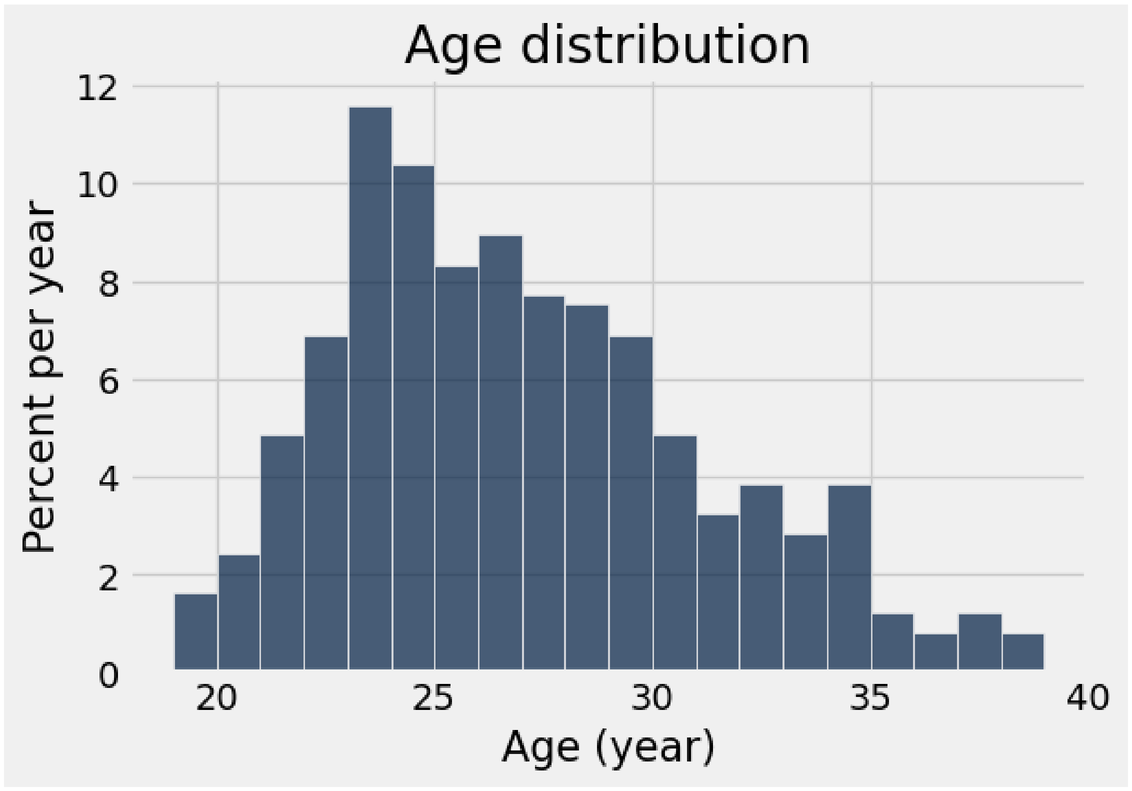
If we want to make estimates about a certain numerical property of the population, we may have to come up with these estimates based only on a smaller sample. The numerical property of the population is known as a parameter, and the estimate is known as a statistic (e.g. the mean or median). Whether these estimates are useful or not often depends on how the sample was gathered. We have prepared some example sample datasets to see how they compare to the full NBA dataset. Later we'll ask you to create your own samples to see how they behave.

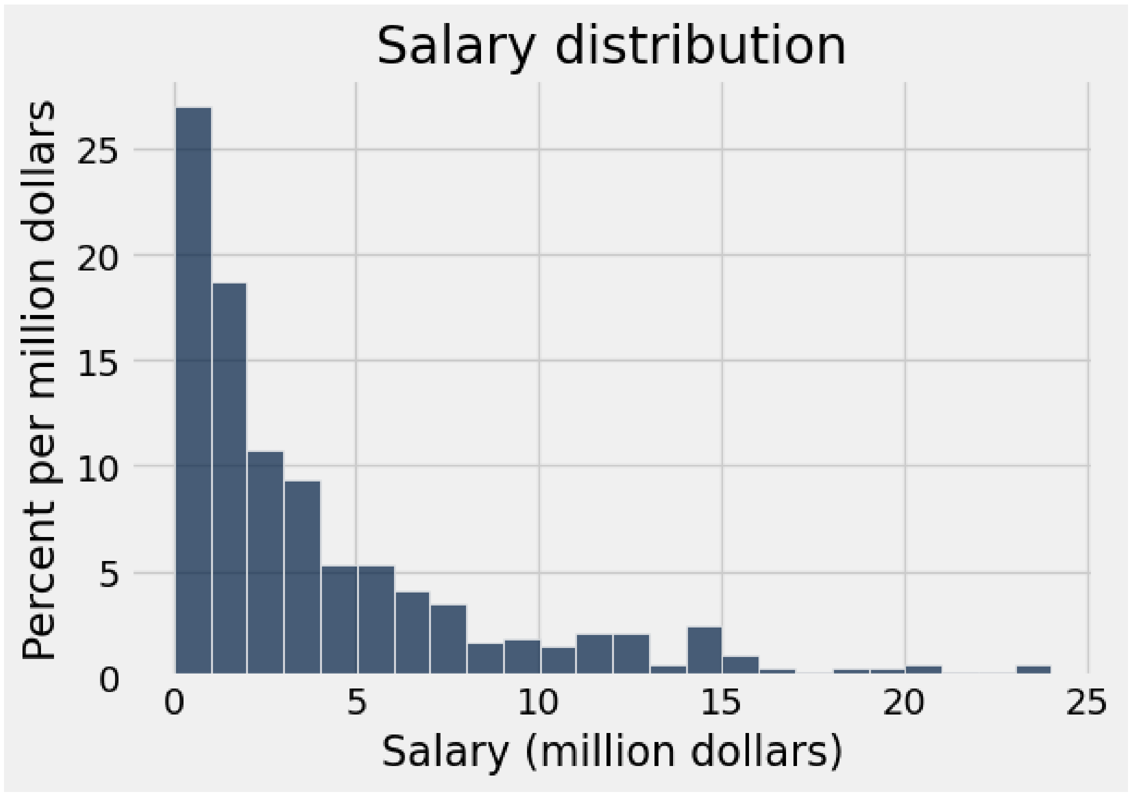
To save typing and increase the clarity of your code, we will package the analysis code into a few functions. This will be useful in the rest of the lab as we will repeatedly need to create histograms and collect summary statistics from that data.

We've defined the `histograms` function below, which takes a table with columns `Age` and `Salary` and draws a histogram for each one. It uses bin widths of 1 year for `Age` and $1,000,000 for `Salary`.

[55]:
```python
def histograms(t):
    ages = t.column('Age')
    salaries = t.column('Salary')/1000000
    t1 = t.drop('Salary').with_column('Salary', salaries)
    age_bins = np.arange(min(ages), max(ages) + 2, 1)
    salary_bins = np.arange(min(salaries), max(salaries) + 1, 1)
    t1.hist('Age', bins=age_bins, unit='year')
    plt.title('Age distribution')
    t1.hist('Salary', bins=salary_bins, unit='million dollars')
    plt.title('Salary distribution')

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

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





Question 3.1. Create a function called `compute_statistics` that takes a table containing an "Age" column and a "Salary" column and: - Draws a histogram of ages - Draws a histogram of salaries - Returns a two-element array containing the average age and average salary (in that order)

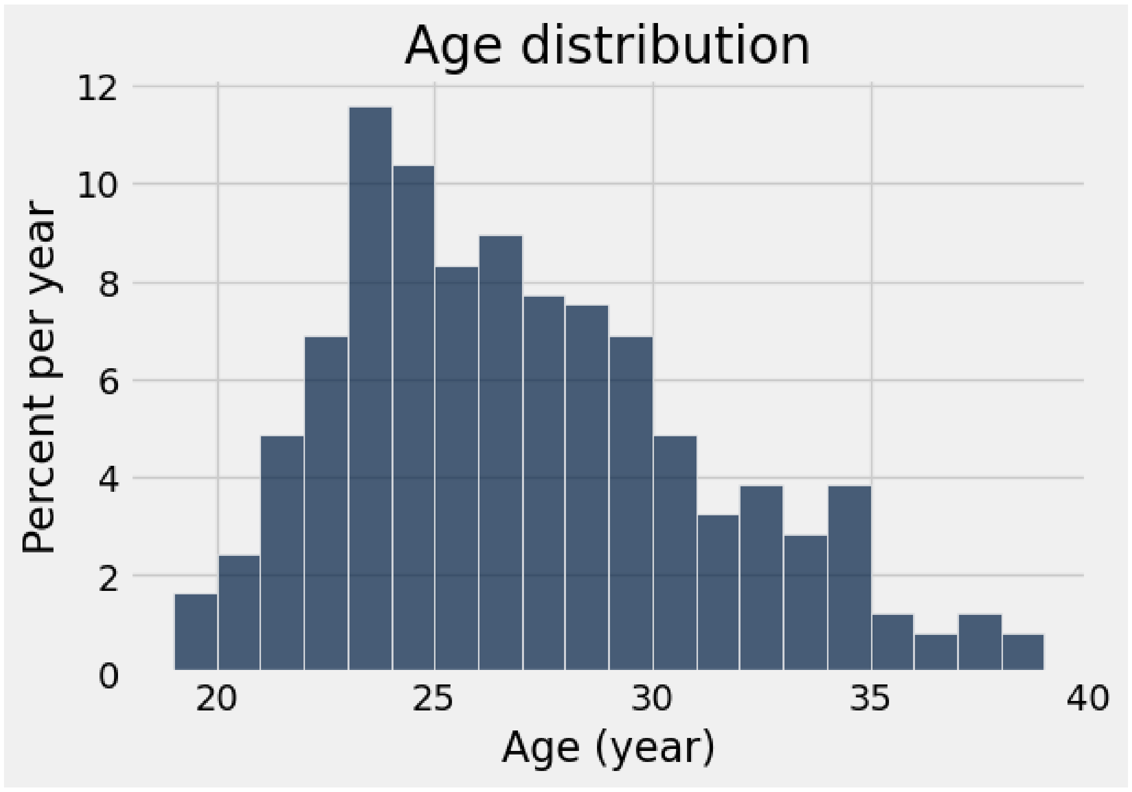
You can call the `histograms` function to draw the histograms!

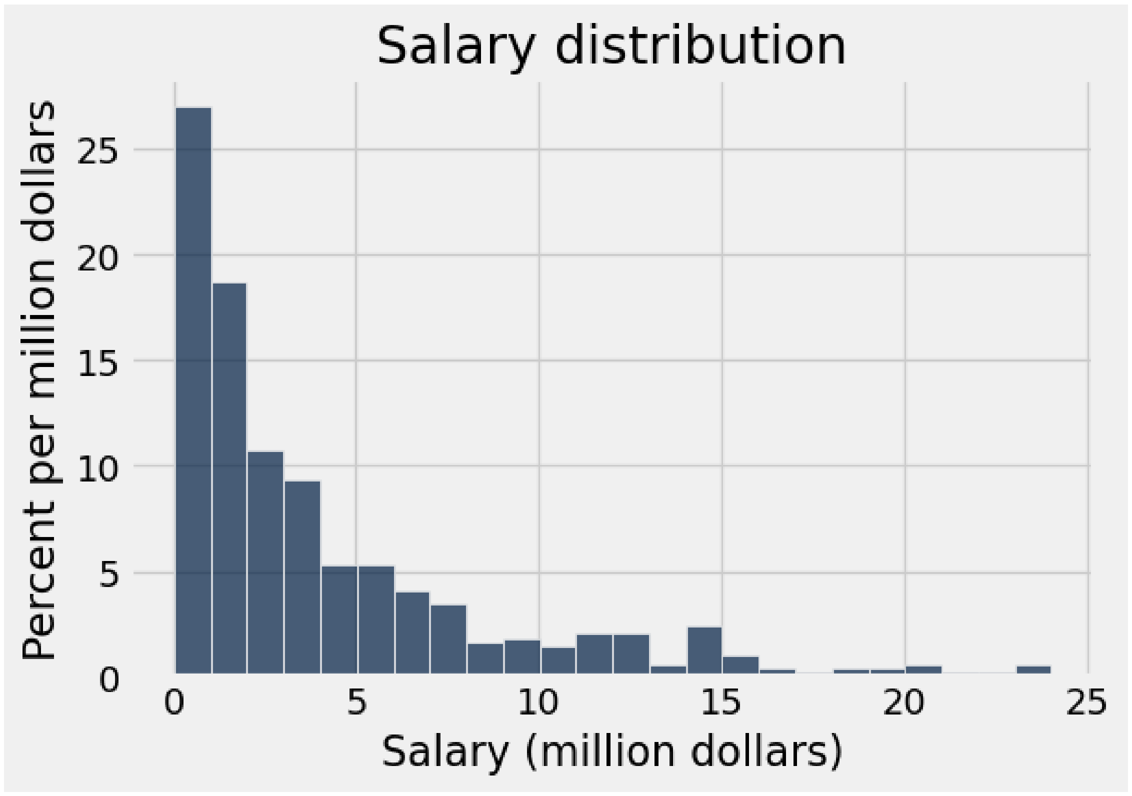
Note: More charts will be displayed when running the test cell. Please feel free to ignore the charts.

[73]:
```
def compute_statistics(age_and_salary_data):
    # STEP 1: Use the histograms function defined above question 3.1 to draw␣
 ↪the histograms for ages and salaries
     histograms(age_and_salary_data)
    # STEP 2: get the ages and salaries as arrays
     ages = age_and_salary_data.column('Age')
     salary = age_and_salary_data.column('Salary')
    # STEP 3: return the average age and average salary as an array
     average_age = np.mean(ages)
     average_salary = np.mean(salary)

     return np.array([average_age, average_salary])


full_stats = compute_statistics(full_data)
```

```
full_stats
```

```
[73]: array([  2.65365854e+01,   4.26977577e+06])
```





```
[74]: grader.check("q31")
```

```
[74]: q31 results: All test cases passed!
```

2.3.1 Simple random sampling

A more justifiable approach is to sample uniformly at random from the players. In a **simple random sample (SRS) without replacement**, we ensure that each player is selected at most once. Imagine writing down each player's name on a card, putting the cards in an box, and shuffling the box. Then, pull out cards one by one and set them aside, stopping when the specified sample size is reached.

2.3.2 Producing simple random samples

Sometimes, it's useful to take random samples even when we have the data for the whole population. It helps us understand sampling accuracy.

2.3.3 `sample`

The table method **`sample`** produces a random sample from the table. By default, it draws at random **with replacement** from the rows of a table. Sampling with replacement means for any row selected randomly, there is a chance it can be selected again if we sample multiple times.

`Sample` takes in the sample size as its argument and returns a **table** with only the rows that were selected.

Run the cell below to see an example call to `sample()` with a sample size of 5, with replacement.

[75]:
```
# Just run this cell

salary_data.sample(5)
```

[75]:
```
PlayerName    | Salary
Jabari Parker | 4930560
Kevin Love    | 15719063
Austin Rivers | 2439840
T.J. Warren   | 1953120
Jabari Parker | 4930560
```

The optional argument `with_replacement=False` can be passed through `sample()` to specify that the sample should be drawn without replacement.

Run the cell below to see an example call to `sample()` with a sample size of 5, without replacement.

[76]:
```
# Just run this cell

salary_data.sample(5, with_replacement=False)
```

[76]:
```
PlayerName    | Salary
Erick Green   | 507336
Trevor Booker | 5000000
Paul Pierce   | 5305000
Mitch McGary  | 1400040
Kyrie Irving  | 7070730
```

[77]:
```
# Run this cell to recap and look at the original table full_data (you will␣
↪start with this one in the next question)
full_data.show(3)
```

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

Question 3.2 Produce a simple random sample **without** replacement of size **44** from `full_data`. Then, run your analysis on it again by using the `compute_statistics` function. Run the cell a few times to see how the histograms and statistics change across different samples.

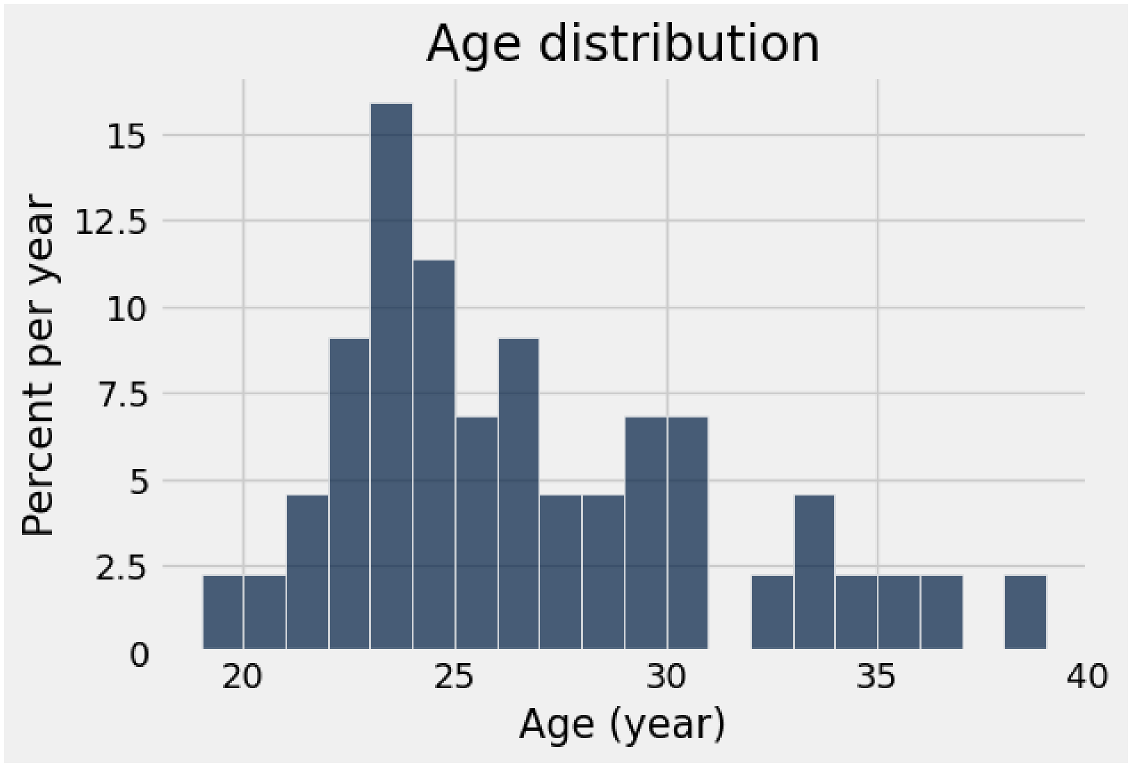
- How much does the average age change across samples?
- What about average salary?

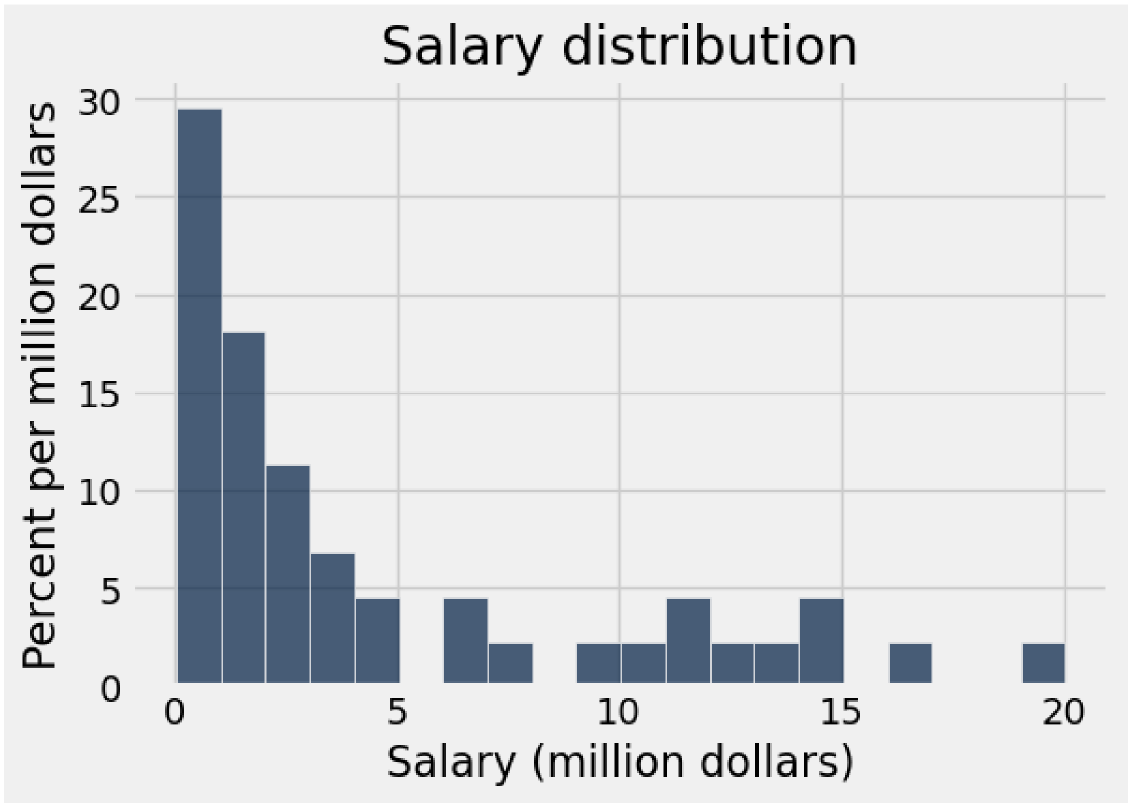
(FYI: srs = simple random sample, wor = without replacement)

Type your answer here, replacing this text.

[79]:
```
# SRSWOR? Did I do this correct?
# STEP 1: produce a simple random sample without replacement of size 44 and␣
  ↪save it in a variable (this variable is a table)
my_small_srswor_data = full_data.sample(44, with_replacement=False)
# STEP 2: Use the compute_statistics function by passing in the table you␣
  ↪created
my_small_stats = compute_statistics(my_small_srswor_data)
my_small_stats
# STEP 3: Run this cell a couple of times and answer the 2 questions above in␣
  ↪your own words
```

```
[79]: array([  2.62954545e+01,   4.69327091e+06])
```





2.4 Submission

Luke & Leia want to congratulate you on finishing the required part of lab 5! You are ready to submit!

Be sure to…

- run the tests and verify that they all pass,
- choose **Download as PDF** from the **File** menu
- submit the .pdf file on **canvas**.

If you would like more practice, continue with the section below :)!

2.5 4. More Random Sampling Practice (OPTIONAL: this section will not be graded)

More practice for random sampling using `np.random.choice`.

2.5.1 Simulations and For Loops (cont.)

Question 4.1 We can use `np.random.choice` to simulate multiple trials.

After finishing the Data Science class project, Stephanie decides to spend the rest of her night rolling a standard six-sided die. She wants to know what her total score would be if she rolled the die 1000 times. Write code that simulates her total score after 1000 rolls.

Hint: First decide the possible values you can take in the experiment (point values in this case). Then use `np.random.choice` to simulate Stephanie's rolls. Finally, sum up the rolls to get Stephanie's total score.

```
[ ]: possible_point_values = ...
     num_tosses = 1000
     simulated_tosses = ...
     total_score = ...
     total_score
```

```
[ ]: grader.check("q41")
```

2.5.2 Simple random sampling (cont.)

Question 4.2 As in the previous question, analyze several simple random samples of size 100 from `full_data` by using the `compute_statistics` function.
- Do the histogram shapes 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/shape for age or for salary? What did you expect to see?

Type your answer here, replacing this text.

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