YData: Introduction to Data Science

Class 04: Loops and conditional statements

Overview

Brief review of list, dictionaries and statistics

For loops

Conditional statements

If there is time

- Array computation (NumPy)
- Numerical array computations

Announcement: Homework 1

Homework 1 has been posted!

It is due on Gradescope on Sunday January 29th at 11pm

- **Be sure to mark each question on Gradescope!**

How is it going so far?

Very quick review of lists and dictionaries

Lists are ways to store multiple items

- my_list = [1, 2, 3 , 4, 5, 'six'] # create a list
- my_list2 = my_list[0:3] # get the first 3 elements
- my_list2.sort() # sort the list (what does this return?)
- sum(my_list2) # sum the elements in the list

Dictionaries allow you to look up ***values*** based on a ***key***

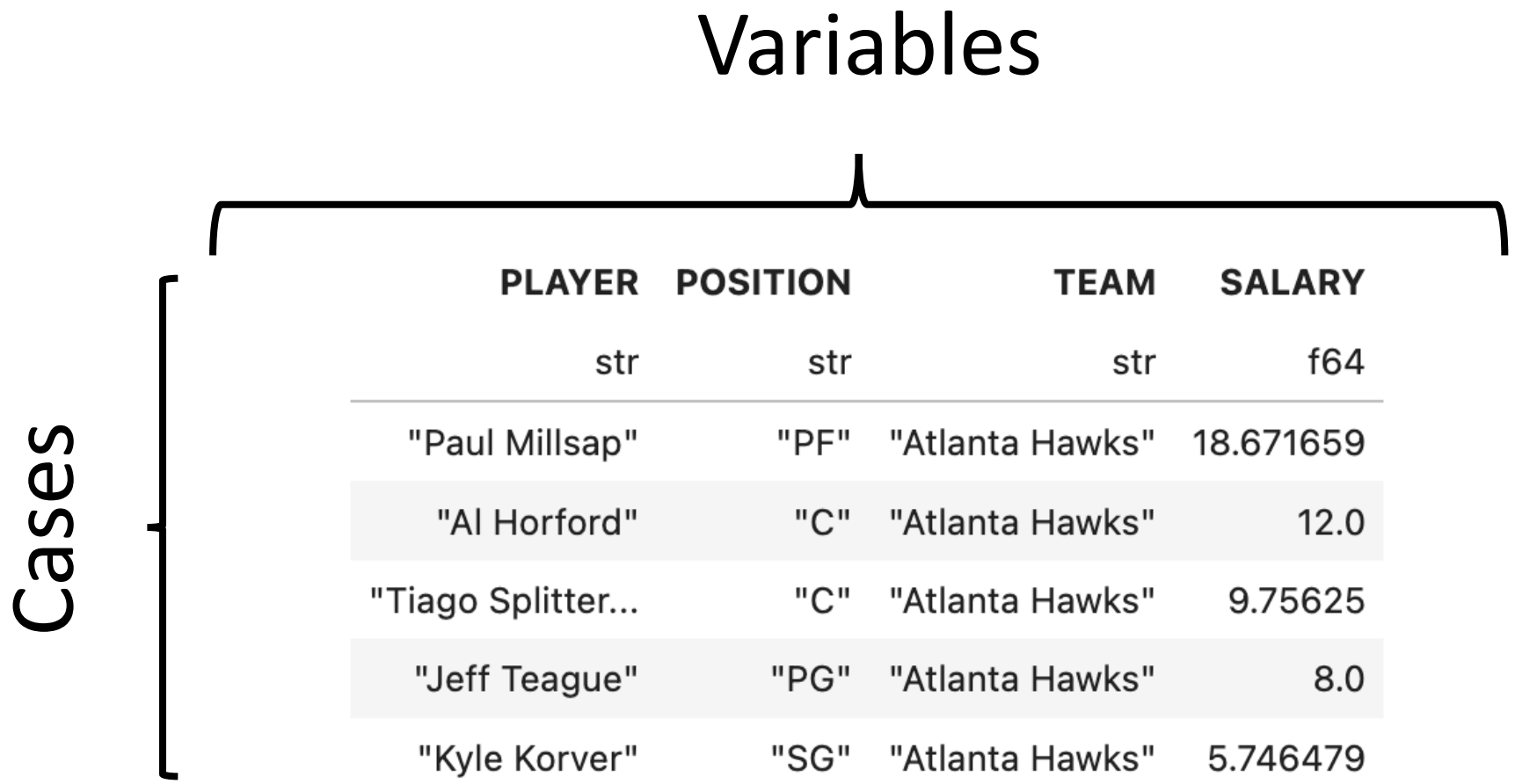
- my_dict = { 'key1': 5, ‘key2’: 20} # create a dictionary
- my_dict['key2'] # get a value based on a key

Review of statistics via NBA salaries

Let's explore salaries of NBA players (from the 2015-2016 season)

PLAYER	POSITION	TEAM	SALARY
Paul Millsap	PF	Atlanta Hawks	18.6717
Al Horford	C	Atlanta Hawks	12
Tiago Splitter	C	Atlanta Hawks	9.75625
Jeff Teague	PG	Atlanta Hawks	8
Kyle Korver	SG	Atlanta Hawks	5.74648
Thabo Sefolosha	SF	Atlanta Hawks	4
Mike Scott	PF	Atlanta Hawks	3.33333
Kent Bazemore	SF	Atlanta Hawks	2
Dennis Schroder	PG	Atlanta Hawks	1.7634
Tim Hardaway Jr.	SG	Atlanta Hawks	1.30452

Example Dataset – NBA player statistics



PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter...	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

Categorical and Quantitative Variables

Cases

Categorical Variable

Quantitative Variable

PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter...	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

statistics

A ***statistic*** is a number computed from a sample of data

Quantitative Variable

PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter...	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

5.07

Proportions

For a *single* ***categorical variable***, the main ***statistic*** of interest is the *proportion* in each category

- E.g., the proportion of basketball players that are Centers (C)

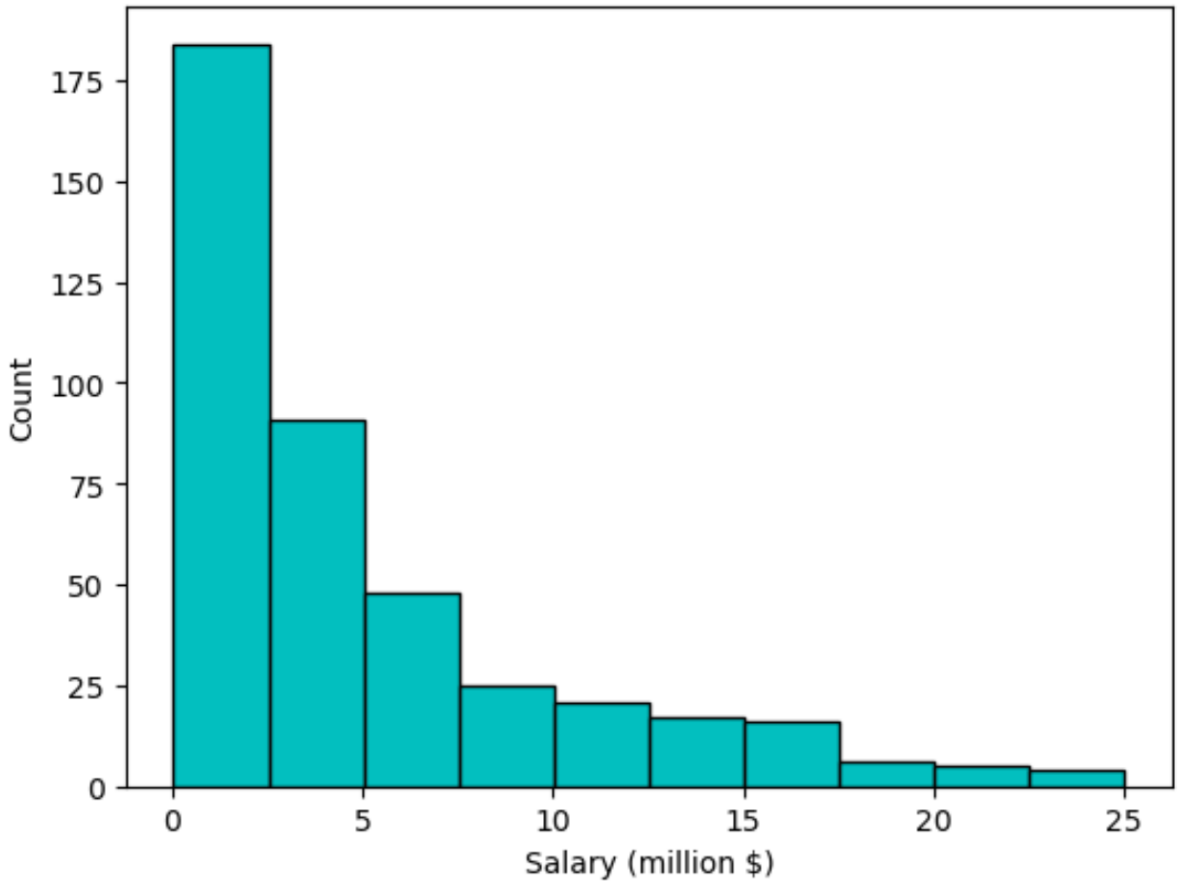
$$\text{Proportion centers} = \frac{\text{number of centers}}{\text{total number}}$$

Categorical Variable

PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter...	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

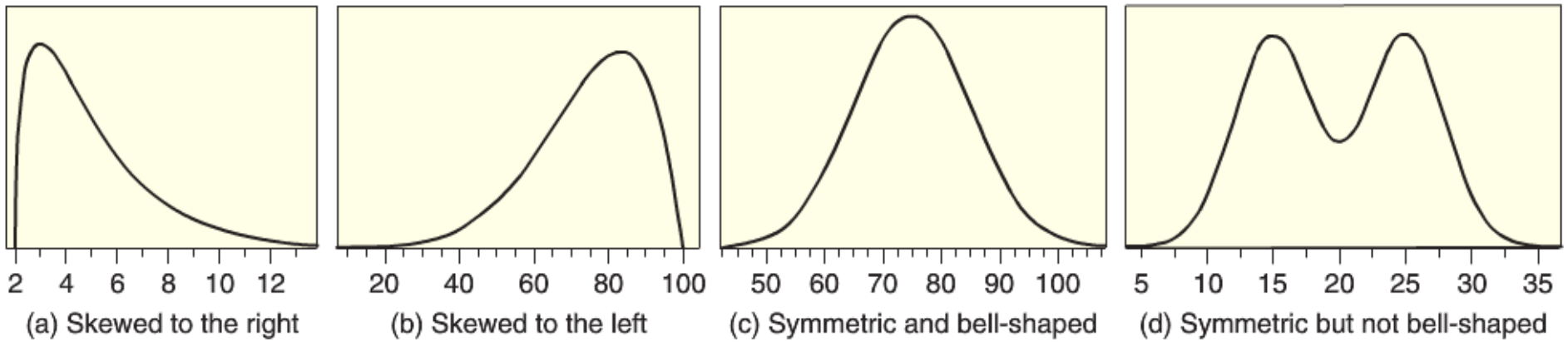
Histograms – of NBA salaries

Life Expectancy	Frequency Count
(0 – 2.5]	184
(2.5 – 5]	91
(5 – 7.5]	48
(7.5 – 10]	25
(10 – 12.5]	21
(12.5 – 15]	17
(15 – 17.5]	16
(17.5 – 20]	6
(20 – 22.5]	5
(22.5 – 25]	4

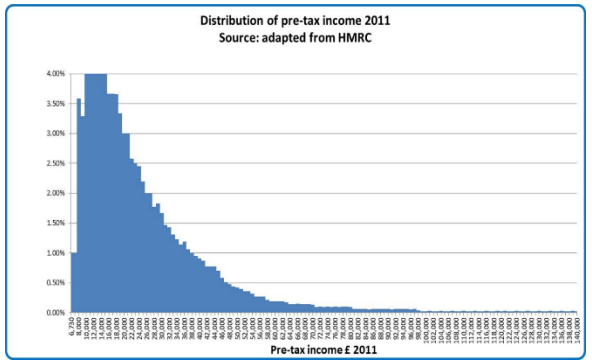


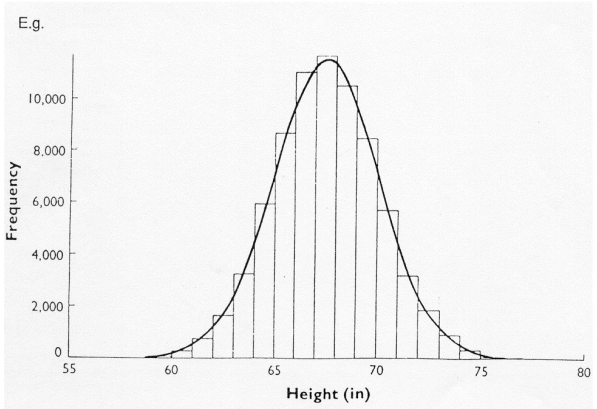
`Matplotlib: plt.hist(data)`

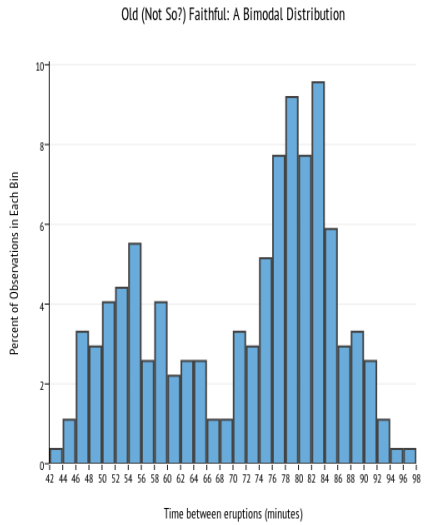
Common shapes of data distributions



Income distribution

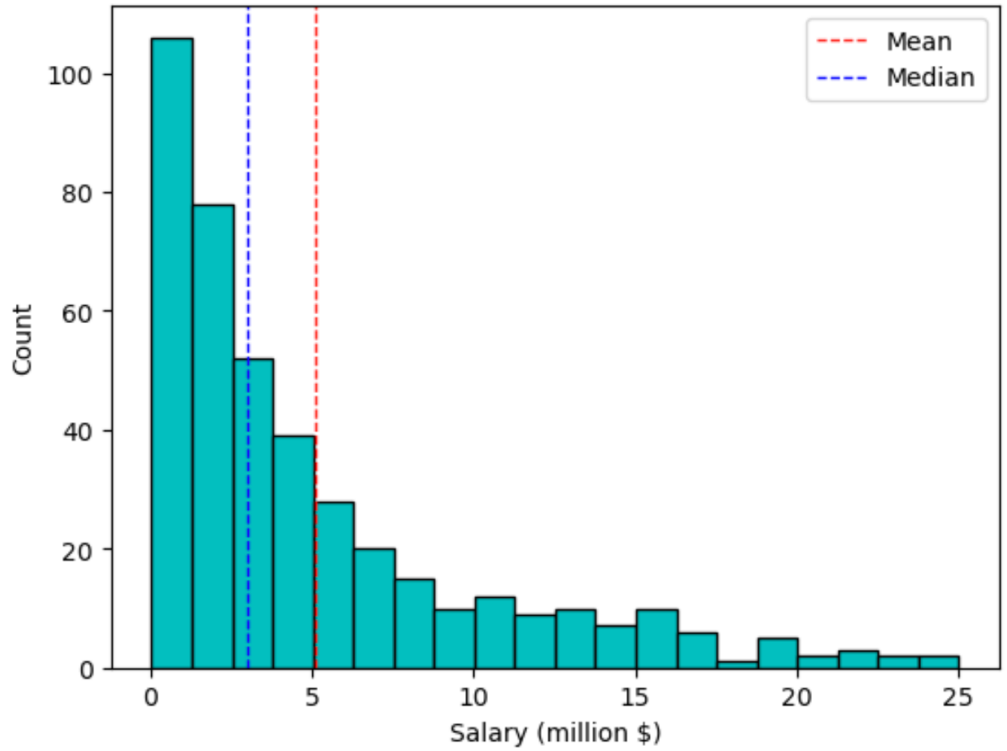






Quantitative data: statistics for central tendency

Two statistics for measuring the “central value” of a sample of quantitative data are the ***mean*** and the ***median***



The mean

$$\text{Mean} = \frac{\text{Sum of all data values}}{\text{Number of data values}}$$

$$\text{Mean} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n} = \sum_{i=1}^{n} \frac{x_i}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```
import statistics
statistics.mean(data_list)
```

The median

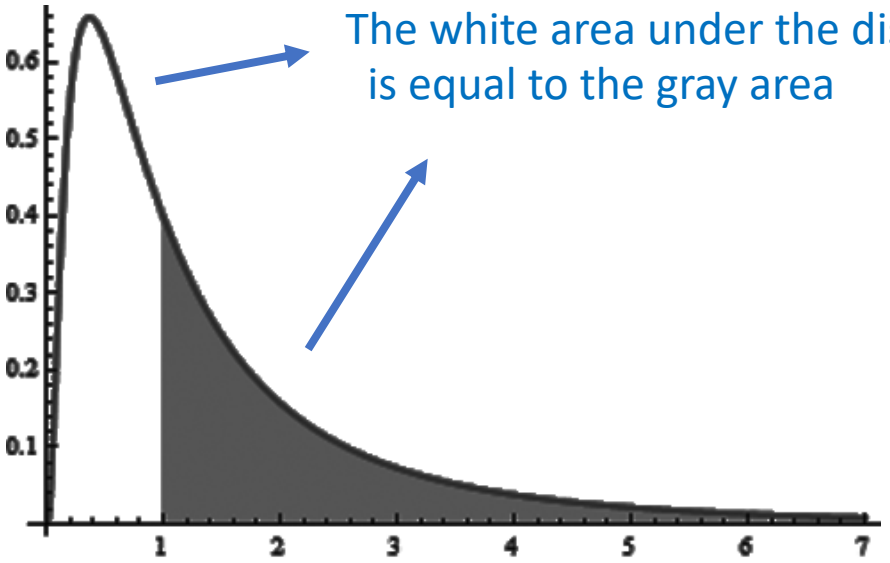
The **median** is a value that splits the data in half

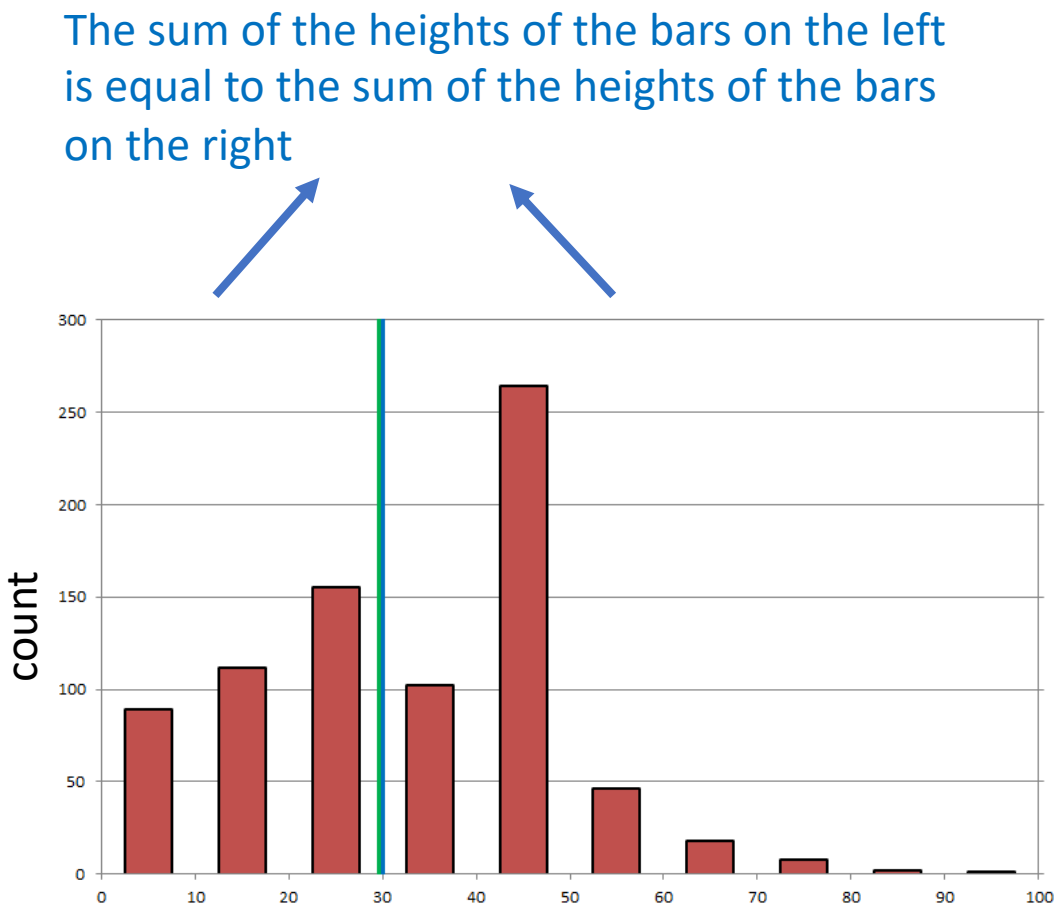
- i.e., half the values in the data are smaller than the median and half are larger

To calculate the median for a data sample of size *n*, sort the data and then:

- If n is odd: The middle value of the sorted data
- If n is even: The average of the middle two values of the sorted data

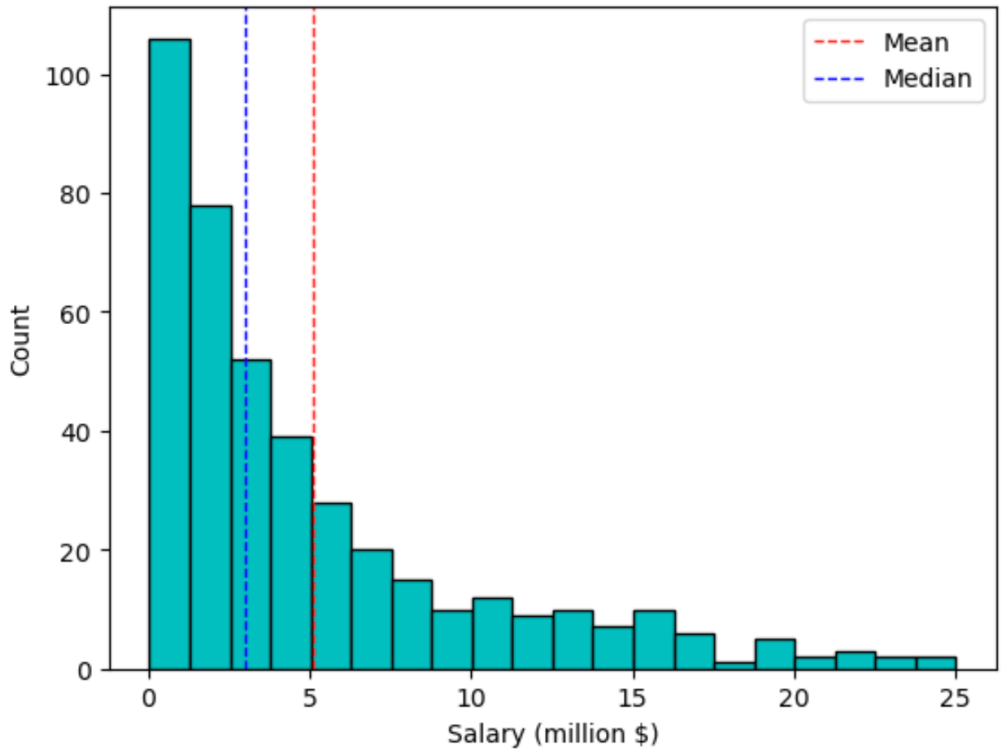
The median






```
import statistics
statistics.median(data_list)
```

Measure of central tendency: mean and median



The median is *resistant* to outliers

- i.e., not affected much by outliers

The mean is not resistant to outliers

What is the mean and median of the data: 1, 2, 3, 4, 990?

- Mean = 200
- Median = 3

Let's explore this in Jupyter!

Back to Python…

Loops

For loops

For loops repeat a process many times, iterating over a sequence of items

- Often we are iterating over an array of sequential numbers

```
animals =  ["cat", "dog", "bat”]
for creature in animals:
    print(creature)

for i in range(4):
    print(i**2)
```

Ranges

A range gives us a sequence of consecutive numbers

An sequence of increasing integers from 0 up to *end* - 1

- range(end)

An sequence of increasing integers from *start* up to *end* - 1

- range(start, end)

A sequence with step between consecutive values

- range(start, end, step)

The range always includes start but excludes end

Let's explore this in Jupyter!

Conditional statements

Comparisons

We can use mathematical operators to compare numbers and strings

- Results return Boolean values True and False

Comparison	Operator	True example	False Example
Less than	<	2 < 3	2 < 2
Greater than	>	3 > 2	3 > 3
Less than or equal	<=	2 <= 2	3 <= 2
Greater or equal	>=	3 >= 3	2 >= 3
Equal	==	3 == 3	3 == 2
Not equal	!=	3 != 2	2 != 2

We can also make comparisons across elements in an array

Let's explore this in Jupyter!

Conditional statements

Conditional statements control the sequence of computations that are performed in a program

We use the keyword if to begin a conditional statement to only execute lines of code if a particular condition is met.

We can use elif to test additional conditions

We can use an else statement to run code if none of the if or elif conditions have been met.

```
num = 5
if num == 1:
    print("Monday")
elif num == 2:
    print("Tuesday")
elif num == 3:
    print("Wednesday")
elif num == 4:
    print("Thursday")
elif num == 5:
    print("Friday")
elif num == 6:
    print("Saturday")
elif num == 7:
    print("Sunday")
else:
    print("Invalid input")
```

Let's explore this in Jupyter!

Array computations

Arrays

Often we are processing data that is all of the same type

- For example, we might want to do processing on a data set of numbers
  - e.g., if we were just analyzing salary data

When we have data that is all of the same type, there are more efficient ways to process data than using a list

- i.e., methods that are faster and take up less memory

In Python, the *NumPy package* offers ways to store and process data that is all of the same type using a data structure called a ***ndarray***

There are also functions that operate on ndarrays that can do computations very efficiently.

ndarrays

We can import the NumPy package using: import numpy as np

We can then create an array by passing a list to the *np.array()* function

- my_array = np.array([1, 2, 3])
  - Note the textbook uses a function called make_array() that we will not use!

We can get elements of an array using similar syntax as using a list

- my_array[1] # what does this return

ndarrays have properties that tell us the type and size

- my_array.dtype # get the type of elements stored in the array
- my_array.shape # get the dimension of the array

Let's explore this in Jupyter!

NumPy functions on numerical arrays

The NumPy package has a number of functions that operate very efficiently on numerical ndarrays

- np.sum()
- np.max(), np.min()
- np.mean(), np.median()
- np.diff() # takes the difference between elements
- np.cumsum() # cumulative sum

There are also "broadcast" functions that operate on all elements in an array

- my_array = np.array([12, 4, 6, 3, 4, 3, 7, 4])
- my_array * 2
- my_array + 7

Let's explore this in Jupyter!