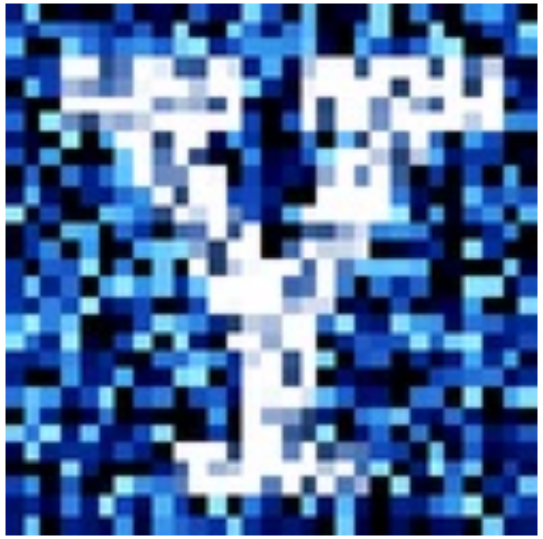
YData: Introduction to Data Science



Class 04: Array computation

Overview

Brief review of list and categorical data

Continuation of statistics and visualizations

- Quantitative data: mean and median, histograms
- If there is time: plots for 2 quantitative variables

NumPy arrays

- Creating arrays
- Array computations
- If there is time: Boolean masking

Very quick review

TO DO LIST
1. make lists
2. look at lists
3. PANIC!

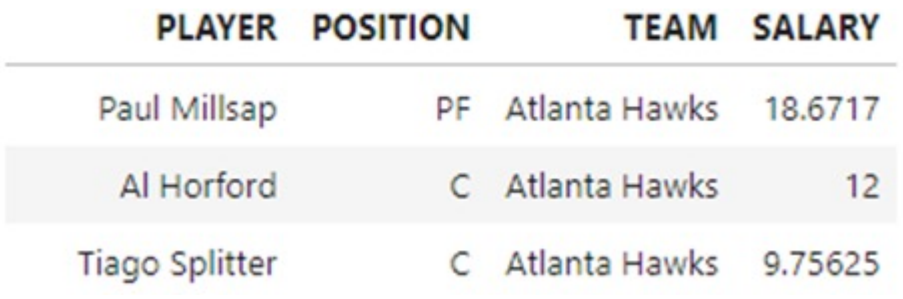
Lists are ways to store multiple items

- 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
- sum(my_list2)   # sum the elements in the list

Categorical data

- Proportion = $\frac{\text{number in category}}{\text{total number}}$

position_list.count("C")/len(position_list)

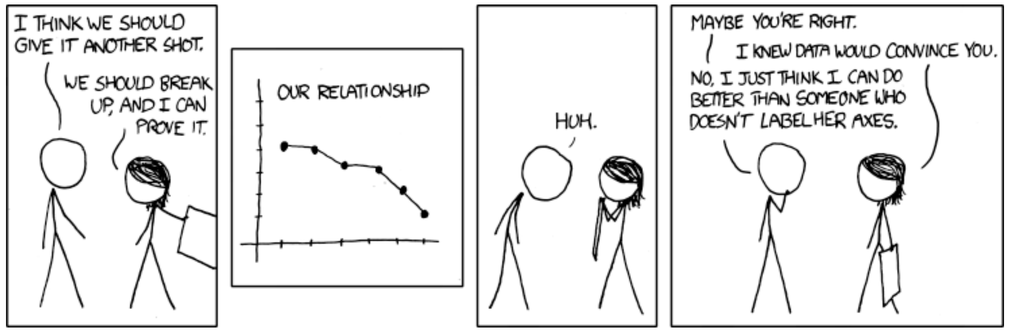


PLAYER	POSITION	TEAM	SALARY
Paul Millsap	PF	Atlanta Hawks	18.6717
Al Horford	C	Atlanta Hawks	12
Tiago Splitter	C	Atlanta Hawks	9.75625

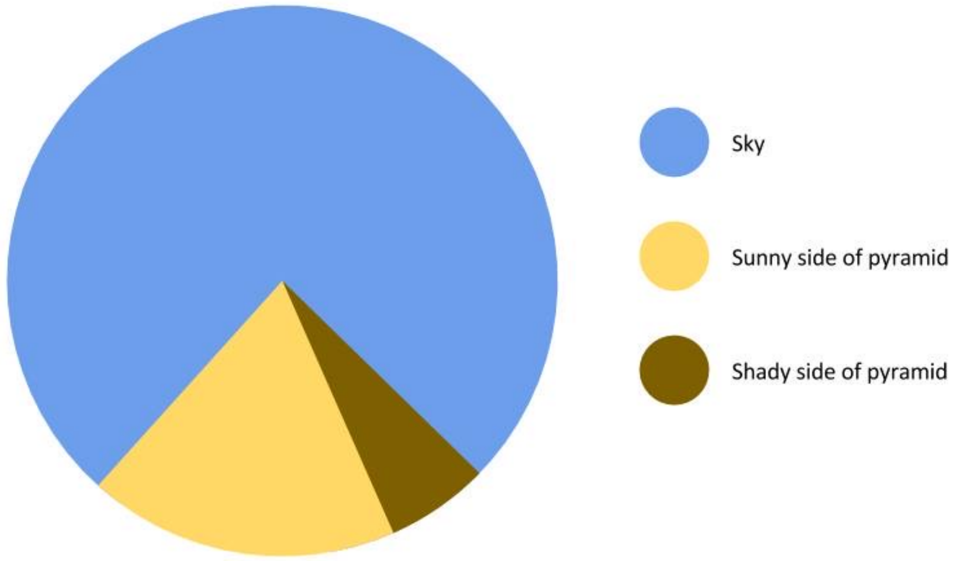
Very quick review

Visualize categorical data

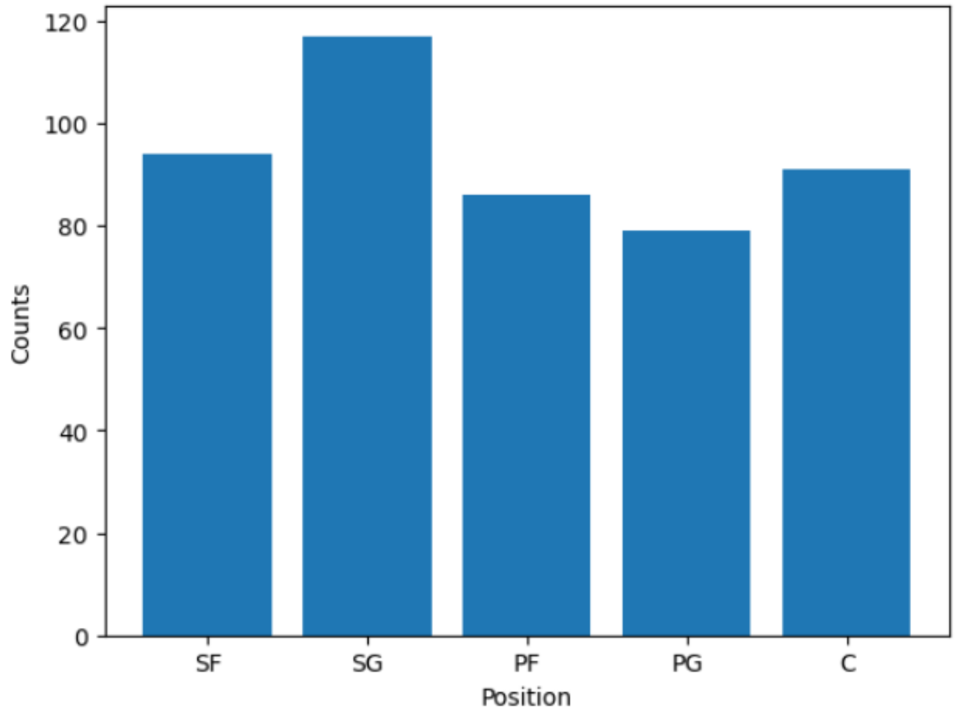
```
import matplotlib.pyplot as plt
plt.bar(labels, data)
plt.pie(data)
```


I THINK WE SHOULD GIVE IT ANOTHER SHOT.
WE SHOULD BREAK UP, AND I CAN PROVE IT.
OUR RELATIONSHIP
HUH.
MAYBE YOU'RE RIGHT.
I KNEW DATA WOULD CONVINCE YOU.
NO, I JUST THINK I CAN DO BETTER THAN SOMEONE WHO DOESN'T LABEL HER AXES.

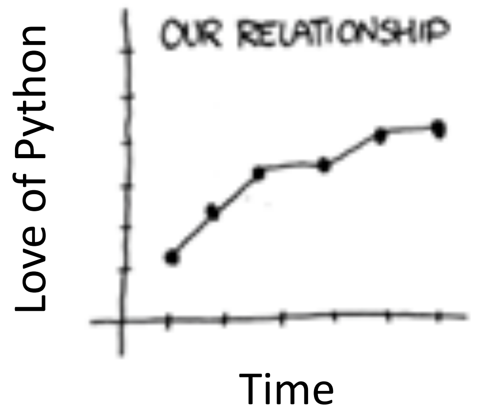
If you don't want exes, label you axes!



Labeling axes!



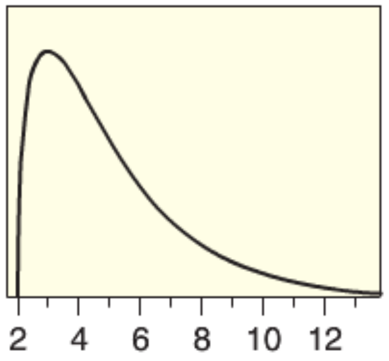
```
plt.bar(position_names, position_counts)

plt.ylabel("Counts")

plt.xlabel("Position");
```



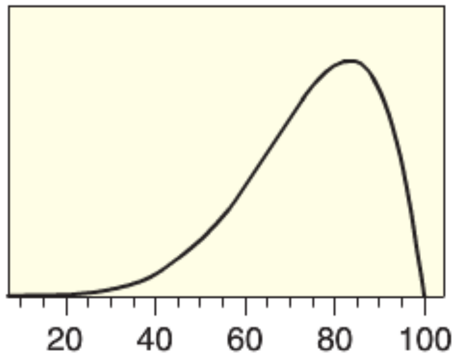
Let's explore this in Jupyter!

Quantitative data

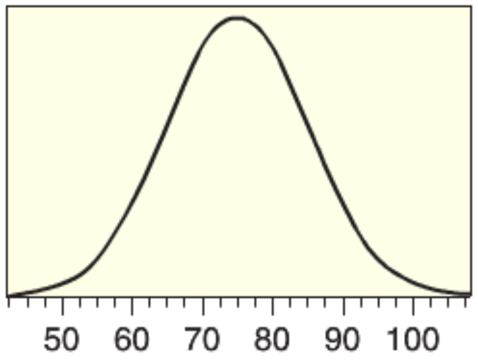
Common shapes of data distributions



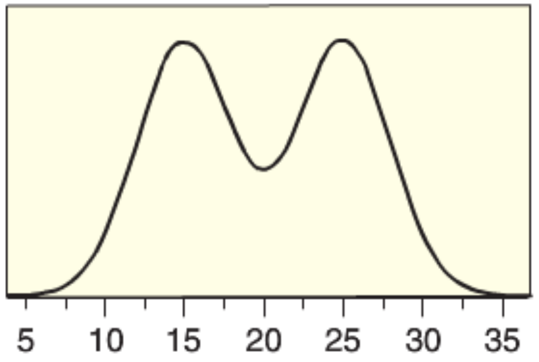
(a) Skewed to the right



(b) Skewed to the left

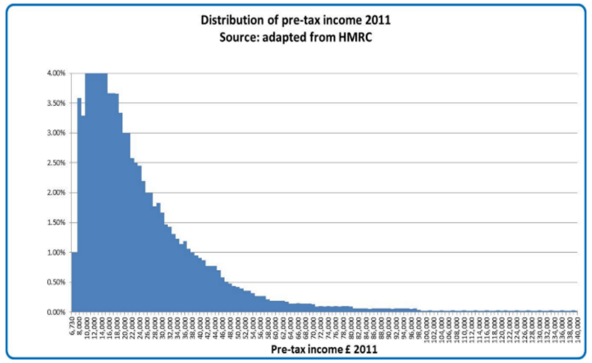


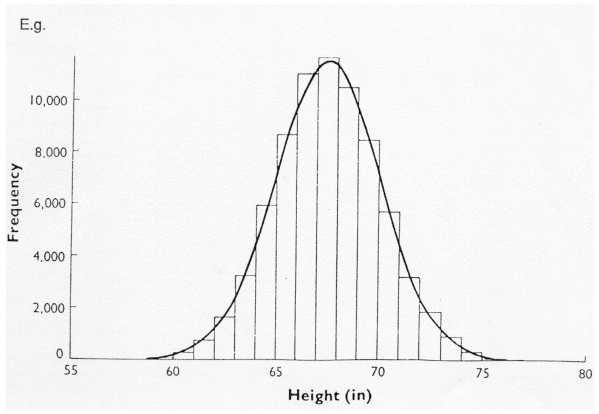
(c) Symmetric and bell-shaped

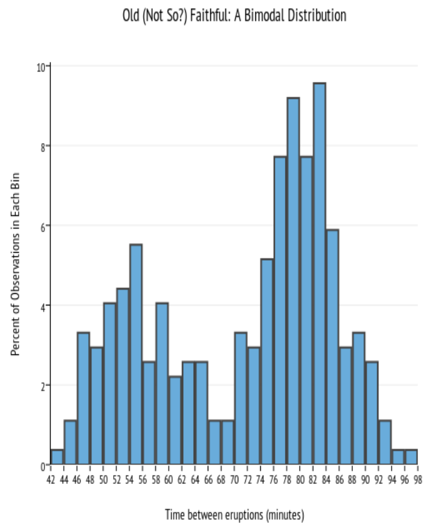


(d) Symmetric but not bell-shaped

Income distribution







Visualizing quantitative data: histograms

Salaries of basketball players (in millions of dollars):

- 2.53, 18.6, 9.4, 21.7, …

To create a histogram we create a set of intervals

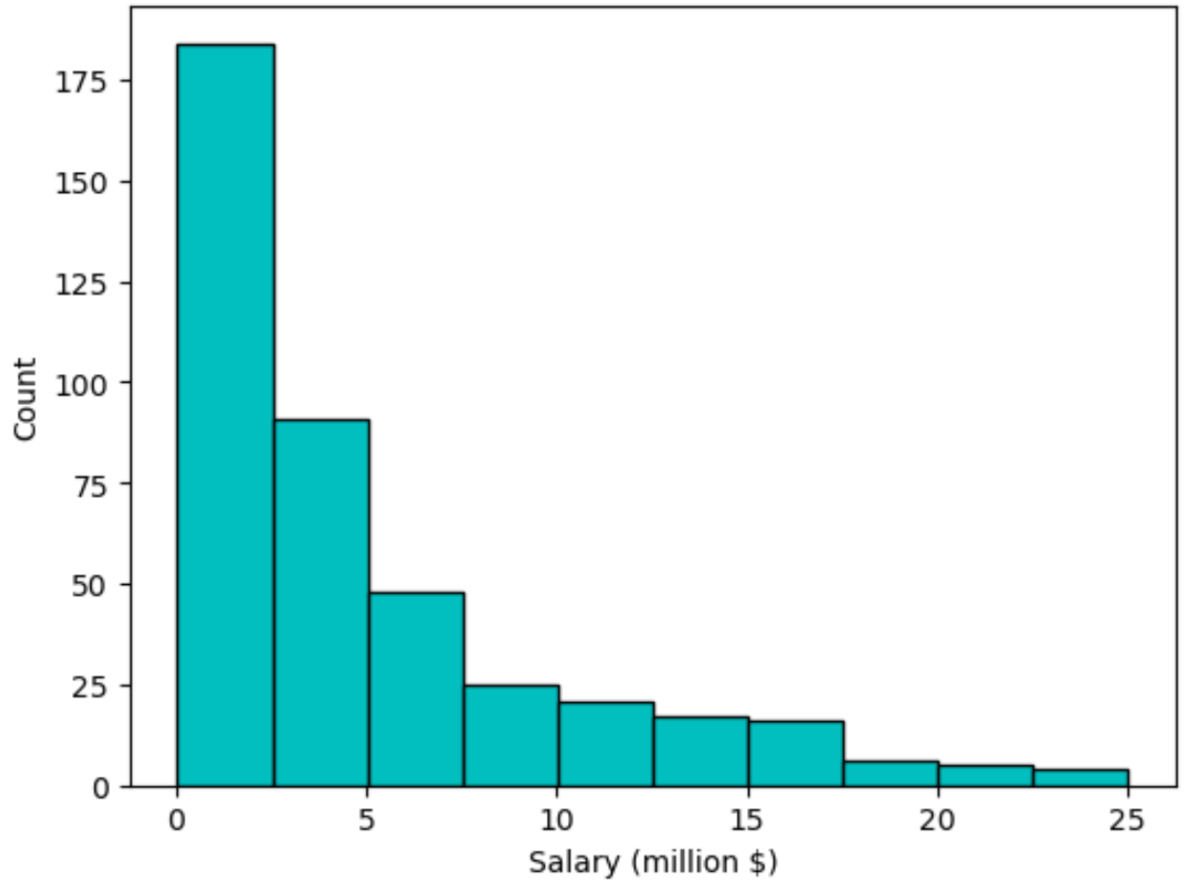
- 0-2.5, 2.5-5, 5-7.5, … 20-22.5, 22.5-25.0

We count the number of points that fall in each interval

We create a bar chart where the height of the bars is the counts in each bin

Histograms – countries life expectancy in 2007

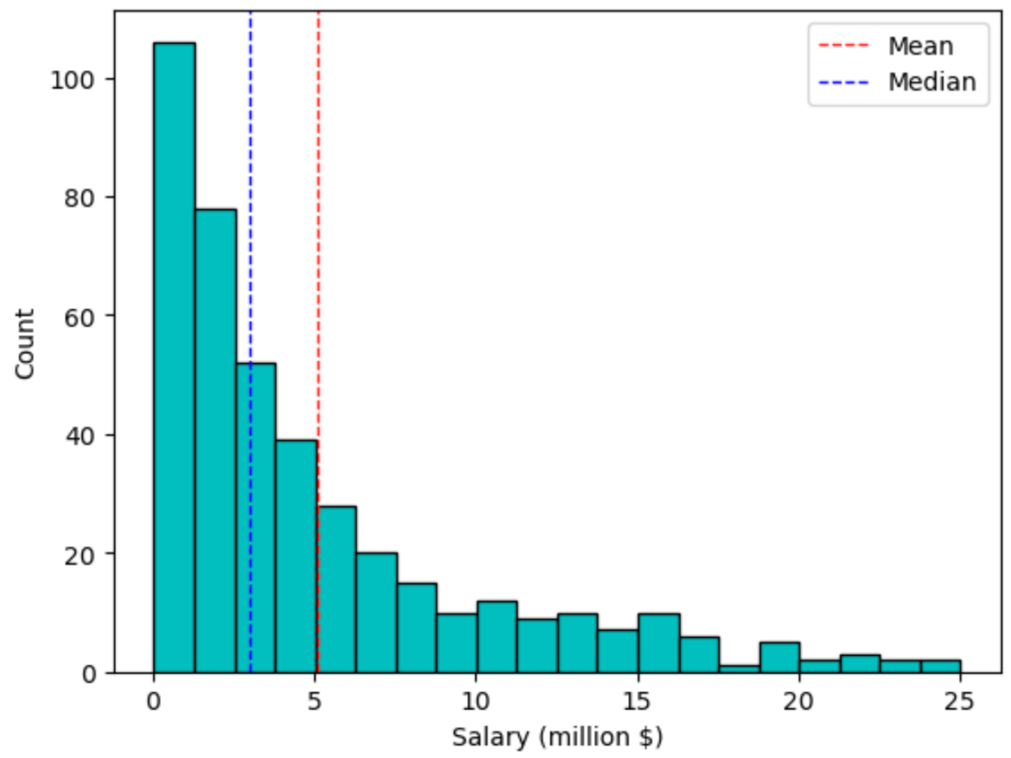
Life Expectancy	Frequency Count
(0 – 2.5]	184
(2.5 – 5]	91
(5 – 7.5]	48
(7.5 – 10]	25
(10 – 12.5]	21
(12.5 – 15]	17
(15 – 17.5]	16
(17.5 – 20]	6
(20 – 22.5]	5
(22.5 – 25]	4



Matplotlib: `plt.hist(data)`

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

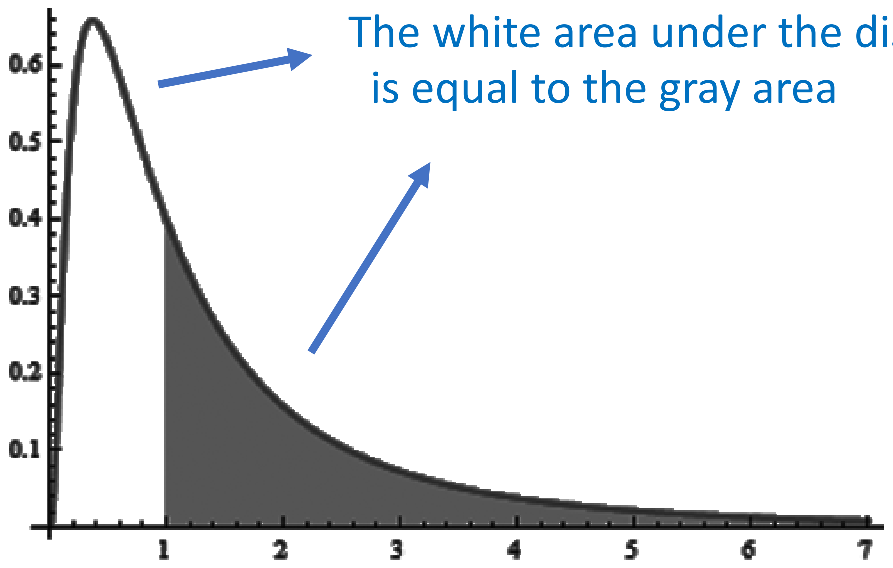
The **median** is a value that splits the data in half

- 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:

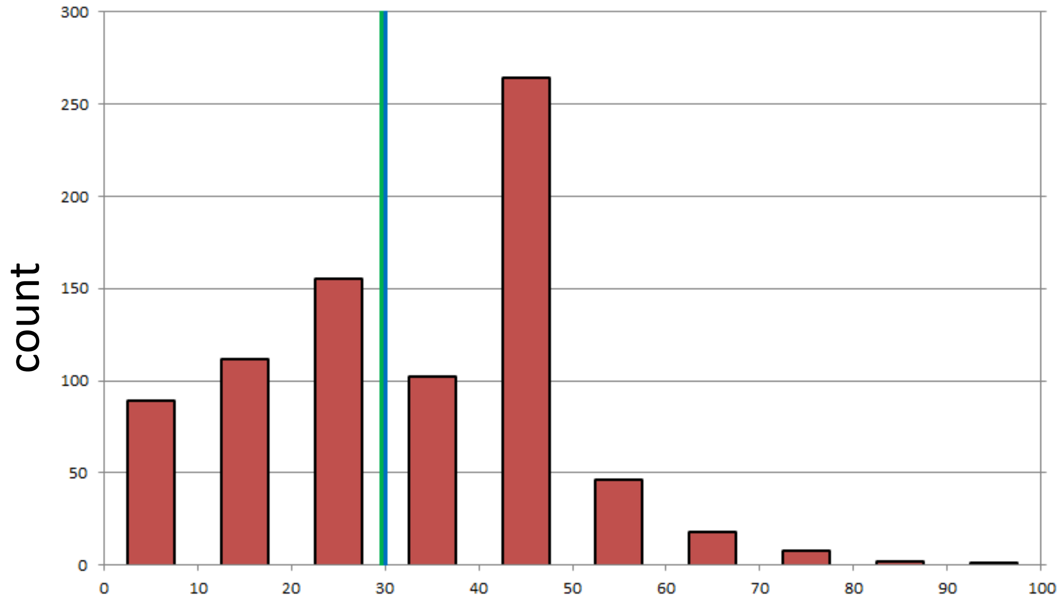
- If n is odd: The middle value of the sorted data
- If n is even: The average of the middle two values of the sorted data

The median

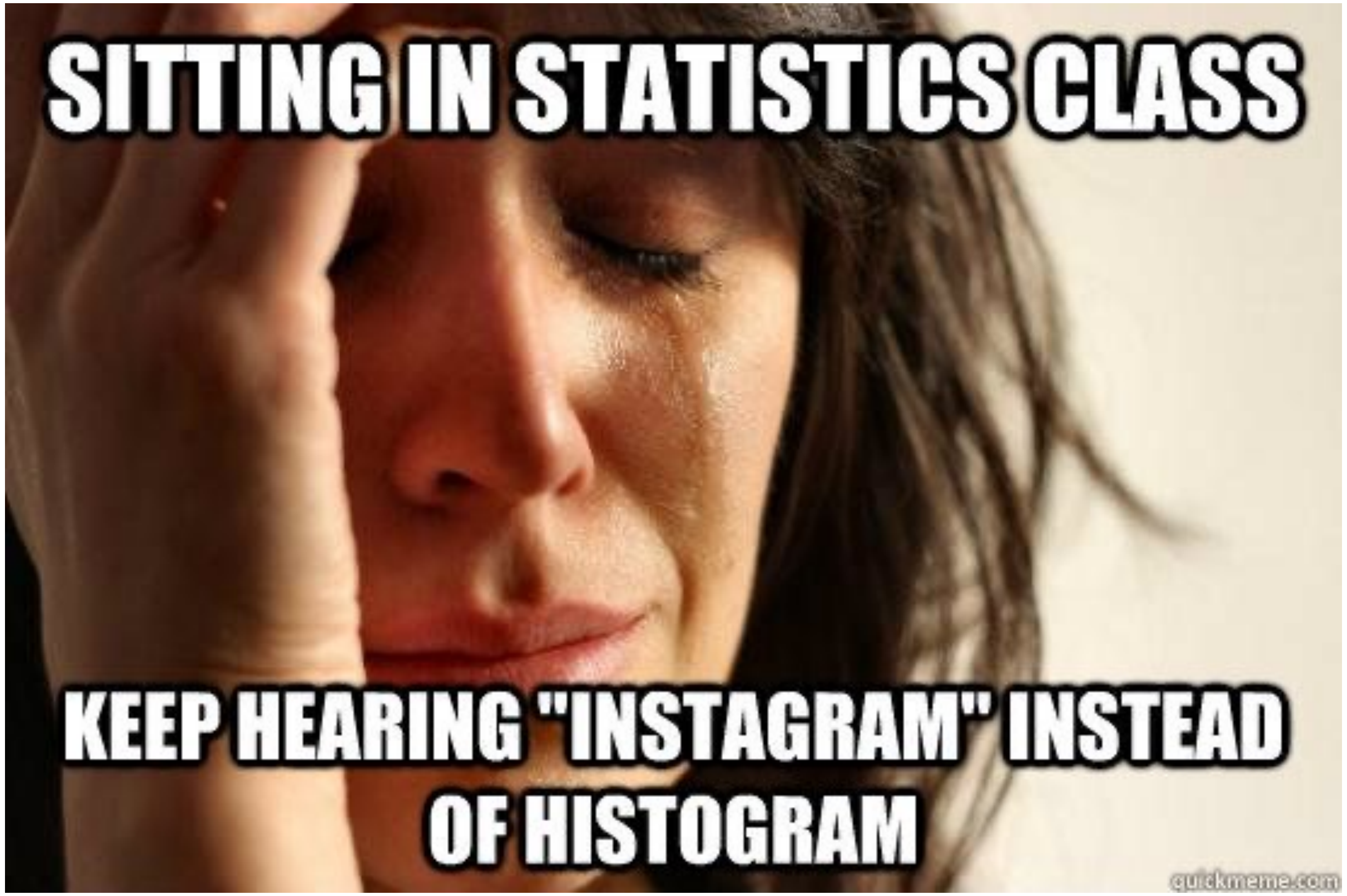


The white area under the distribution is equal to the gray area

The sum of the heights of the bars on the left is equal to the sum of the heights of the bars on the right



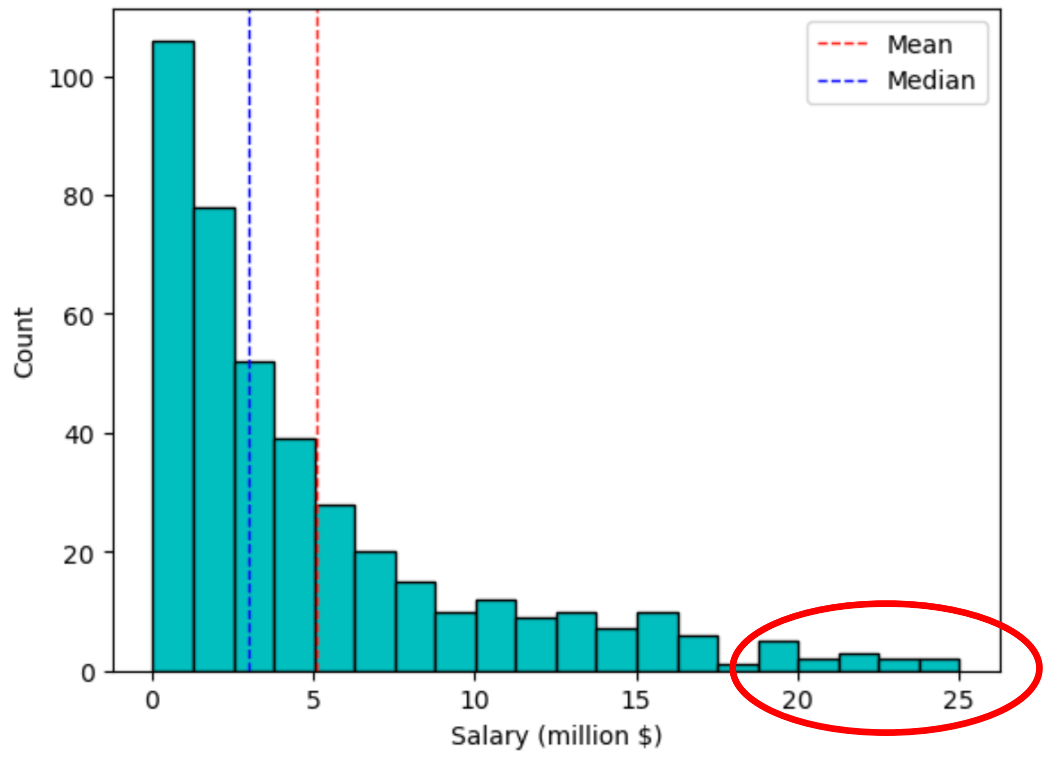
```
import statistics
statistics.median(data_list)
```


SITTING IN STATISTICS CLASS
KEEP HEARING "INSTAGRAM" INSTEAD OF HISTOGRAM
quickmeme.com

Let's explore this in Jupyter!

Outliers

An **outlier** is an observed value that is notably distinct from the other values in a dataset by being much smaller or larger than the rest of the data.

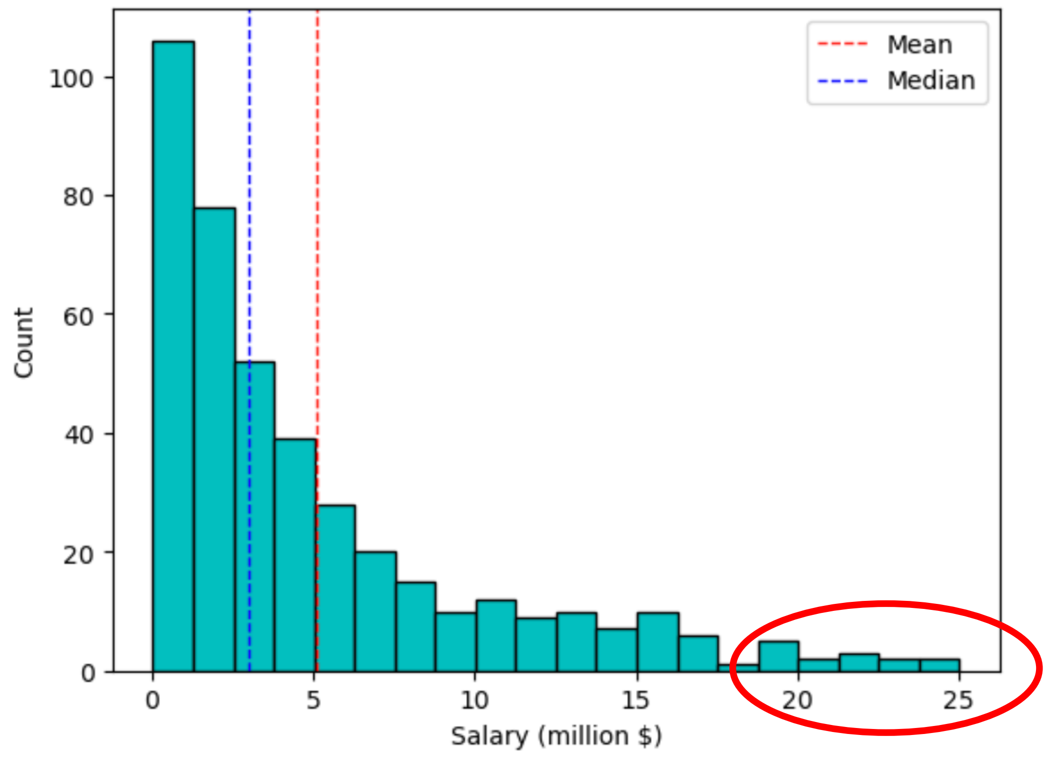


Outliers can potentially have a large influence on the statistics you calculate

One should examine outliers to understand what is causing them

- If there are due to an error, remove them
- Otherwise, need to think about how to treat them
  - Could be interesting phenomenon
  - Could restrict data to a particular range of values
  - Etc.

Outliers' impact on mean and median



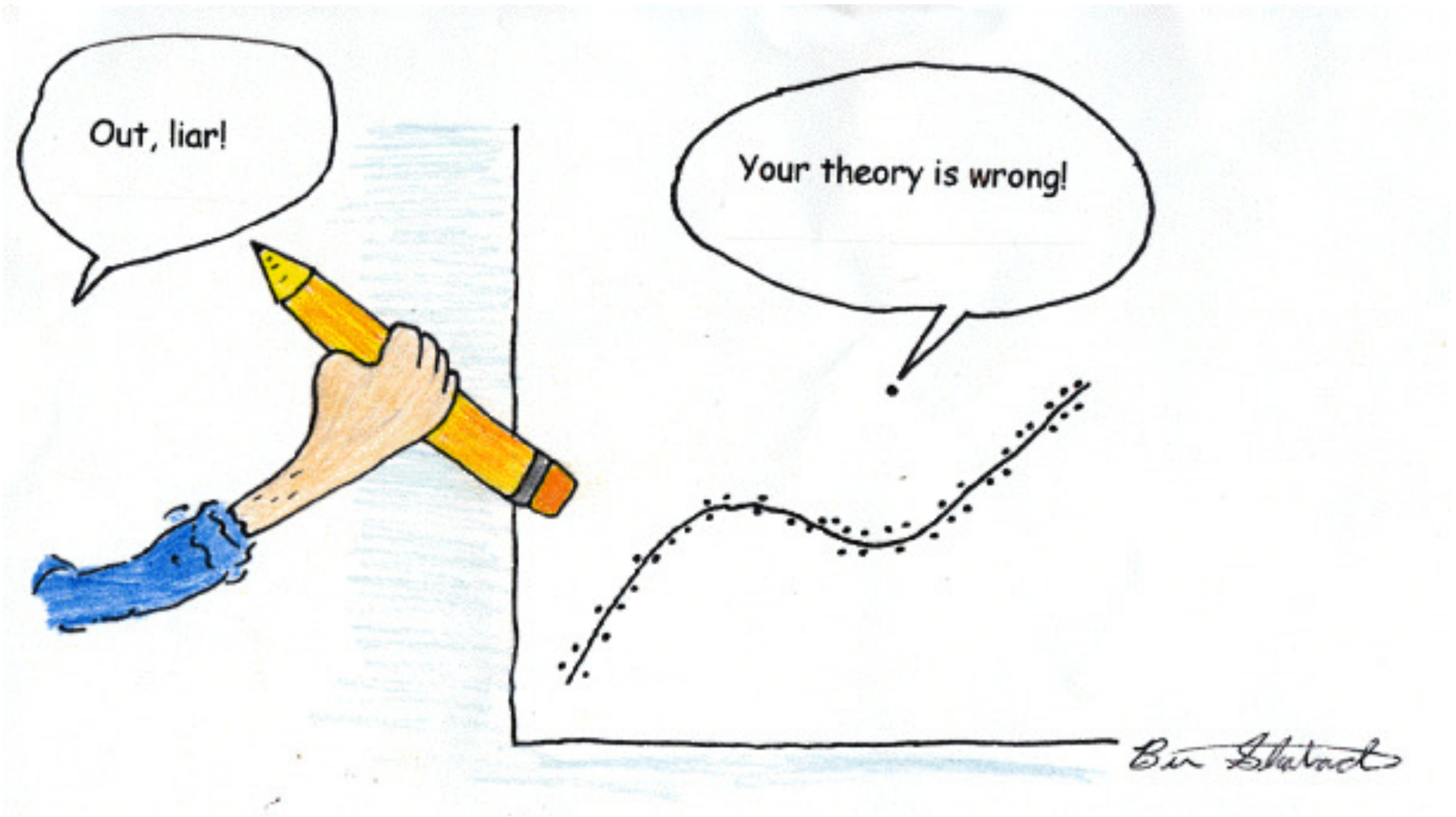
The median is *resistant* to outliers

- 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!

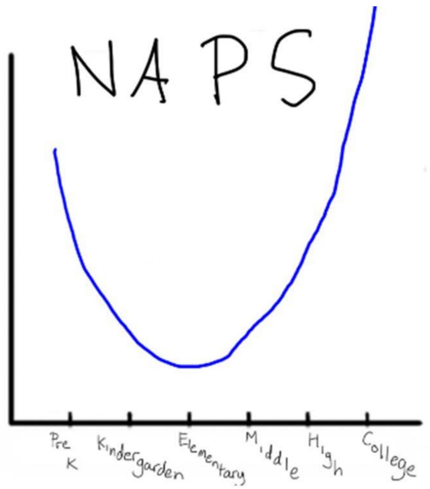
ANY QUESTIONS?

Visualizing sequences: line graph

Line graph (line chart, or curve chart) displays information as a series of data points called "markers" connected by straight line segments.

We can create line graphs in matplotlib using:

- plt.plot(y)   # create a line graph with successive integers as the x-values
- plt.plot(x, y)   # plots y as a function of x
- plt.plot(x, y, '-.') # creates lines with circle markers

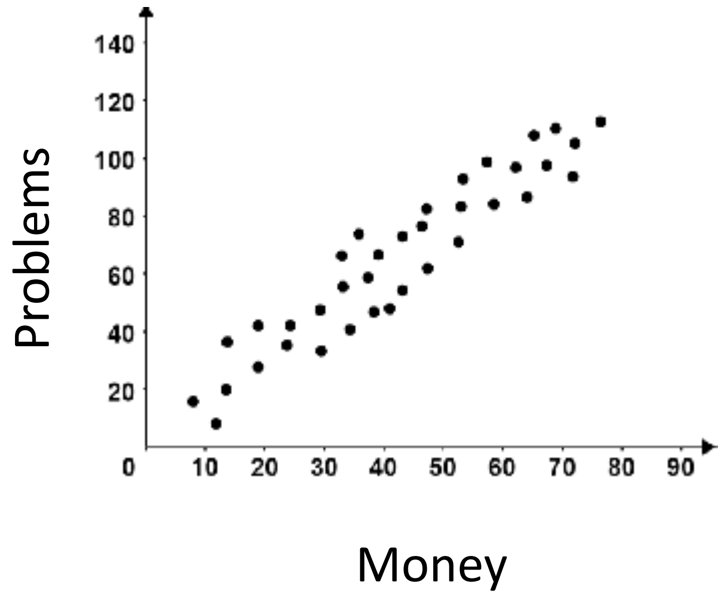


Visualizing two quantitative variables: scatterplots

A **scatterplot** graphs the relationship between two variables

- Each axis represents the value of one variables
- Each point the plot shows the value for the two variables for a single data case

If there is an explanatory and response variable, then the explanatory variable is put on the x-axis and the response variable is put on the y-axis.



plt.plot(x, y, '.')

```
plt.ylabel("y label")

plt.xlabel("x label")

plt.title("my title")


plt.plot(x, y, label = "blah”)

plt.legend()
```

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 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])

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
- my_array.astype('str')   # convert the numbers to strings
- sequential_nums = np.arrange(1, 10)   # creates numbers 1 to 9

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_array2 = np.array([10, 9, 2, 8, 9, 3, 8, 5])
- my_array - my_array2

Let's explore this in Jupyter!

Boolean arrays

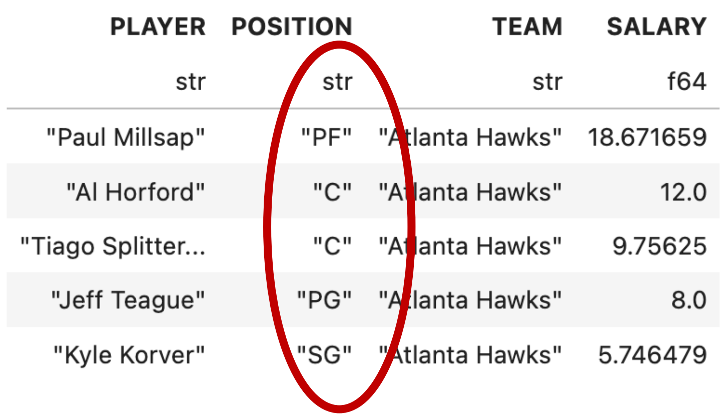
It is often to compare all values in an ndarray to a particular value

- my_array = np.array([12, 4, 6, 3, 4, 3, 7, 4])
- my_array < 5   # any guesses what this will return
  - array([False, True, False, True, True, True, False, True])

This can be useful for calculating proportions

- True == 1 and False == 0
- Taking the sum of a Boolean array gives the total number of True values
- The number of True 's divided by the length is the proportion
  - Or we can use the np.mean() function

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

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

Let's explore this in Jupyter!

Boolean masking

We can also use Boolean arrays to return values in another array

- This is called "Boolean masking" or "Boolean indexing"

```
my_array = np.array([12, 4, 6, 3])
boolean_mask = np.array([False, True, False, True, True])

smaller_array = my_array[boolean_mask]
```

This can be useful for calculating statistics on data that meet particular criteria:

- `np.mean(my_array[my_array < 5])    # what does this do?`

Boolean masking

Suppose you wanted to get the average salary of NBA players who were centers and you had these two ndarrays:

- Position: The position of all NBA players
- Salary: Their salaires

Could you do it?

Let's explore this in Jupyter!