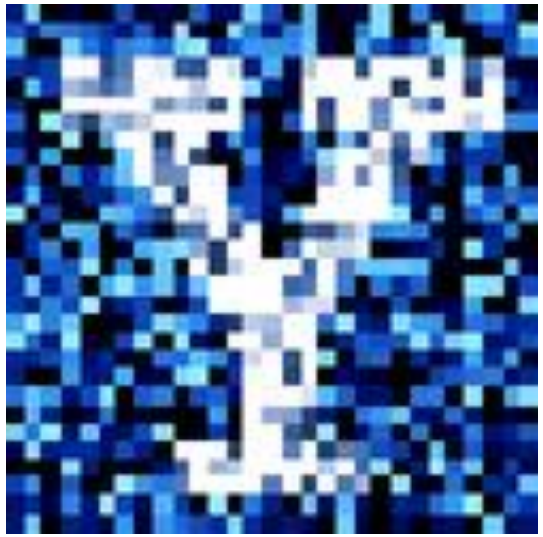
YData: Introduction to Data Science



Class 16: For loops and writing functions

Overview

Very quick review of mapping

For loops

Conditional statements

Writing functions

Reminder: class project

The class project is a 6-10 page Jupyter notebook report where you analyze data you find interesting.

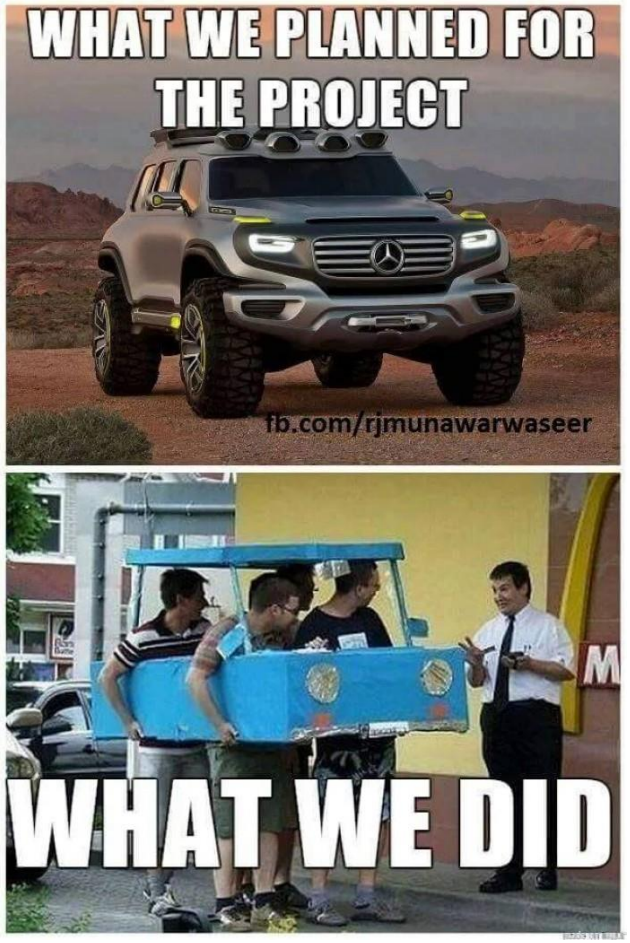
Think about what questions you want to examine, find data, and load it into Python

- A few sources for data sets are listed on Canvas

You can download a project template Jupyter notebook using:

```
import YData
YData.download_class_file('project_template.ipynb', 'homework')
```

A polished draft of the project is due on November 10th



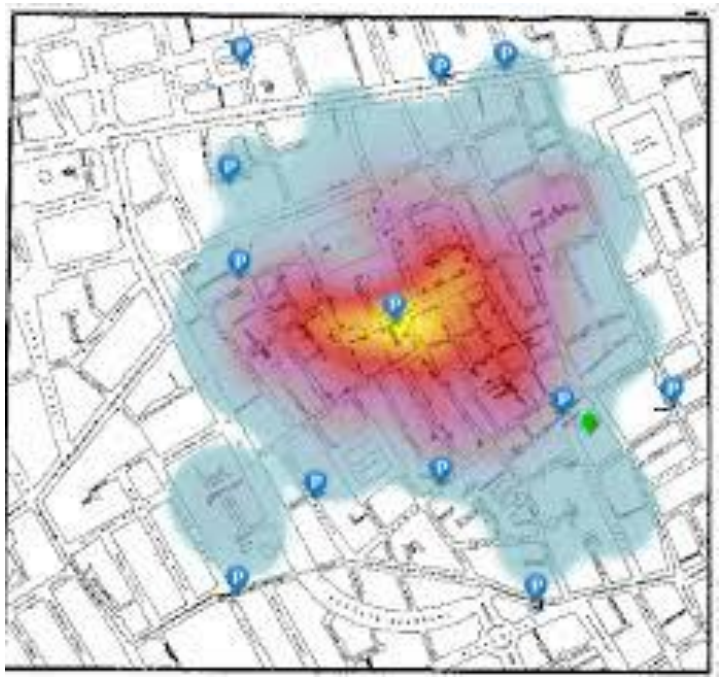
Quick review of mapping

Quick review of maps

Visualizing data on a map can be a powerful way to see spatial trends

We can create maps in Python using geopandas DataFrames

- Like regular DataFrames with an additional geometry column that has Shaply objects



John Snow's ghost map (1854)

	key_comb_drvr	geometry
0	M11551	POINT (117.525391 34.008926)
1	M17307	POINT (86.51248 30.474344)
2	M19584	POINT (89.537415 37.157627)

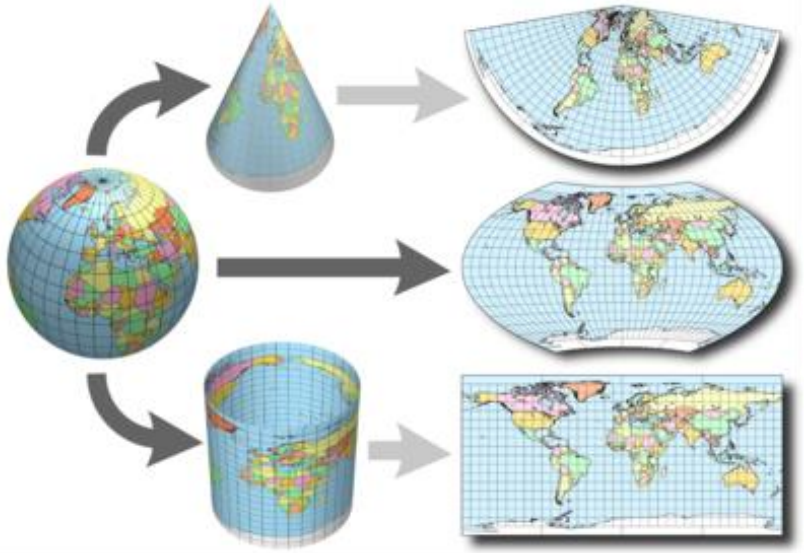
Review: CRSs and map projections

A coordinate reference system (CRS) is a framework used to precisely measure locations on the surface of the Earth as coordinates

- Needed for aligning different layers on maps

There are many map projections to display Earth's 3D structure on a 2D map surface.

- **Mercator projection** keeps angles intact
- **Eckert IV projection** keeps the size of land areas intact



WHAT YOUR FAVORITE MAP PROJECTION SAYS ABOUT YOU

MERCATOR

YOU'RE NOT REALLY INTO MAPS.

VAN DER GRINTEN

YOU'RE NOT A COMPLICATED PERSON. YOU LOVE THE MERCATOR PROJECTION; YOU JUST WISH IT WEREN'T SQUARE. THE EARTH'S NOT A SQUARE, IT'S A CIRCLE. YOU LIKE CIRCLES. TODAY IS GONNA BE A GOOD DAY!

ROBINSON

YOU HAVE A COMFORTABLE PAIR OF RUNNING SHOES THAT YOU WEAR EVERYWHERE. YOU LIKE COFFEE AND ENJOY THE BEATLES. YOU THINK THE ROBINSON IS THE BEST-LOOKING PROJECTION, HANDS DOWN.

DYMAXION

YOU LIKE ISAAC ASIMOV, XML, AND SHOES WITH TOES. YOU THINK THE SEGWAY GOT A BAD RAP. YOU OWN 3D GOGGLES, WHICH YOU USE TO VIEW ROTATING MODELS OF BETTER 3D GOGGLES. YOU TYPE IN DVORAK.

WINKEL-TRIPEL

NATIONAL GEOGRAPHIC ADOPTED THE WINKEL-TRIPEL IN 1998, BUT YOU'VE BEEN A WT FAN SINCE *LONG* BEFORE 'NATGEO' SHOWED UP. YOU'RE WORRIED IT'S GETTING PLAYED OUT, AND ARE THINKING OF SWITCHING TO THE KAVRAYSKIY. YOU ONCE LEFT A PARTY IN DISGUST WHEN A GUEST SHOWED UP WEARING SHOES WITH TOES. YOUR FAVORITE MUSICAL GENRE IS "POST-".

GOODE HOMOLOSINE

THEY SAY MAPPING THE EARTH ON A 2D SURFACE IS LIKE FLATTENING AN ORANGE PEEL, WHICH SEEMS EASY ENOUGH TO YOU. YOU LIKE EASY SOLUTIONS. YOU THINK WE WOULDN'T HAVE SO MANY PROBLEMS IF WE'D JUST ELECT *NORMAL* PEOPLE TO CONGRESS INSTEAD OF POLITICIANS. YOU THINK AIRLINES SHOULD JUST BUY FOOD FROM THE RESTAURANTS NEAR THE GATES AND SERVE *THAT* ON BOARD. YOU CHANGE YOUR CAR'S OIL, BUT SECRETLY WONDER IF YOU REALLY *NEED* TO.

HOBO-DYER

YOU WANT TO AVOID CULTURAL IMPERIALISM, BUT YOU'VE HEARD BAD THINGS ABOUT GALL-PETERS. YOU'RE CONFLICT-AVERSE AND BUY ORGANIC. YOU USE A RECENTLY-INVENTED SET OF GENDER-NEUTRAL PRONOUNS AND THINK THAT WHAT THE WORLD NEEDS IS A REVOLUTION IN CONSCIOUSNESS.

A GLOBE!

YES, YOU'RE VERY CLEVER.

PEIRCE QUINCUNCIAL

YOU THINK THAT WHEN WE LOOK AT A MAP, WHAT WE REALLY SEE IS OURSELVES. AFTER YOU FIRST SAW *INCEPTION*, YOU SAT SILENT IN THE THEATER FOR SIX HOURS. IT FREAKS YOU OUT TO REALIZE THAT EVERYONE AROUND YOU HAS A SKELETON INSIDE THEM. YOU *HAVE* REALLY LOOKED AT YOUR HANDS.

PLATE CARRÉE

(EQUIRECTANGULAR)

YOU THINK THIS ONE IS FINE. YOU LIKE HOW X AND Y MAP TO LATITUDE AND LONGITUDE. THE OTHER PROJECTIONS OVERCOMPLICATE THINGS. YOU WANT ME TO STOP ASKING ABOUT MAPS SO YOU CAN ENJOY DINNER.

WATERMAN BUTTERFLY

REALLY? YOU KNOW THE WATERMAN? HAVE YOU SEEN THE 1909 CAHILL MAP IT'S BASED—... YOU HAVE A FRAMED REPRODUCTION AT HOME?! WHOA. ...LISTEN, FORGET THESE QUESTIONS. ARE YOU DOING ANYTHING TONIGHT?

GALL-PETERS

I *HATE* YOU.

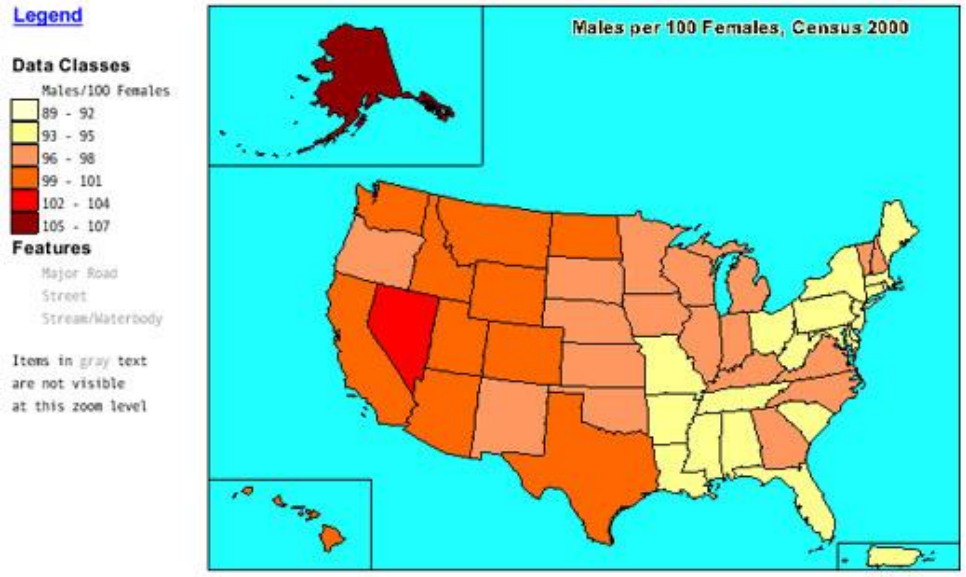
Review: Choropleth and Isopleth maps

Choropleth maps: shades/colors in predefined areas based on properties of a variable

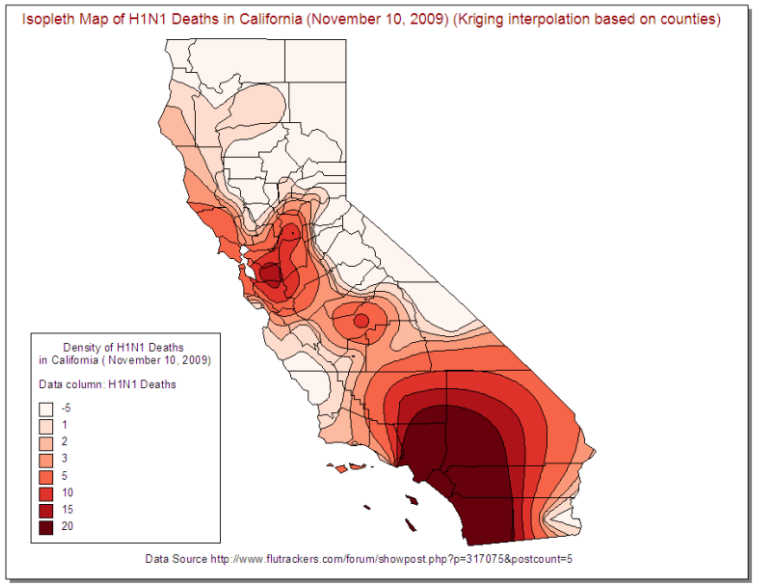
- We can then use the gpd.plot(column =) method to create choropleth maps

Isopleth maps: creates regions based on constant values

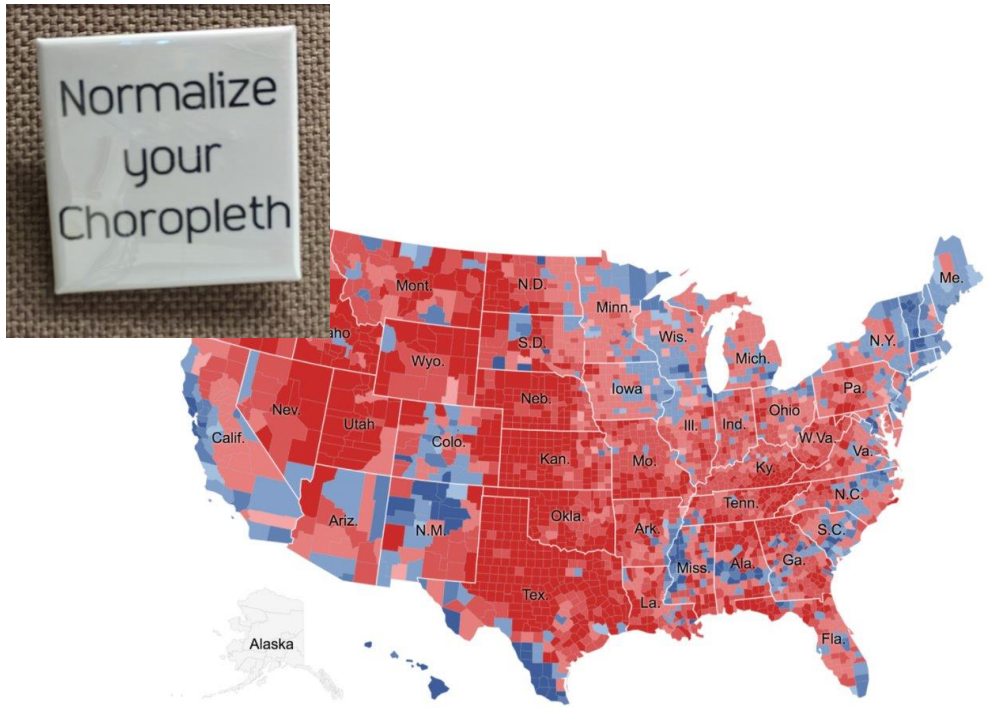
Choropleth map



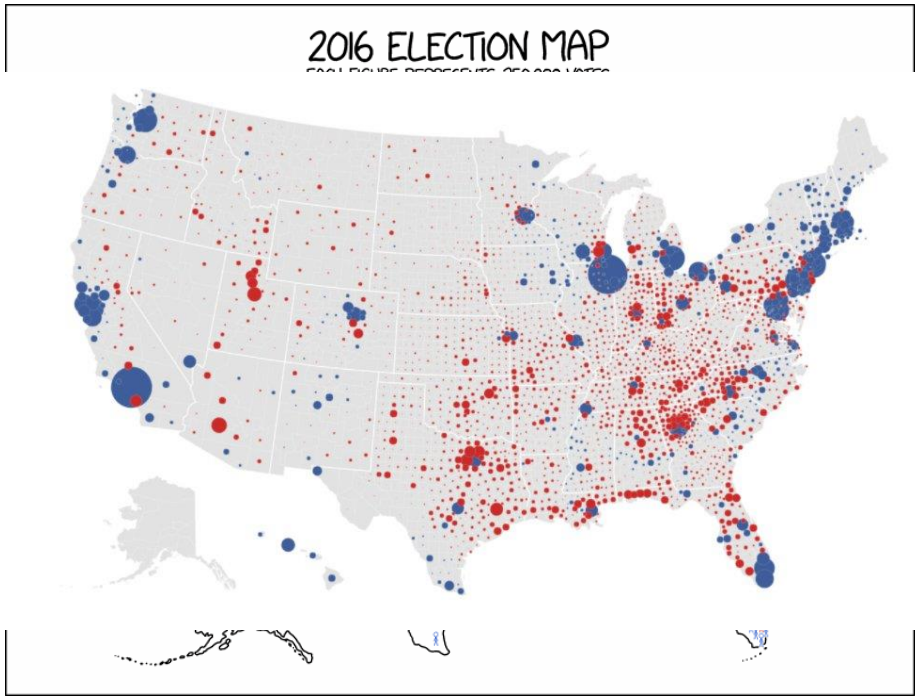
Isopleth map



Cloropleth maps can be misleading



Looks like most of the country voted republican



Let’s try a quick warm-up exercise in Jupyter!

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 np.arange(4):
    print(i**2)
```

Review: 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)
```

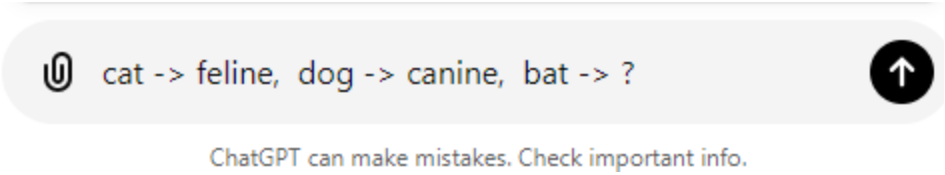
The range always includes start but excludes end

Let's explore this in Jupyter!

Enumerate and zip

We can use the enumerate() function to both items in a list, and sequential integers:

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



We can use the zip() function to get items for two lists:

```
        animal_order =  ["feline", "canine", "chiropteran"]
        for curr_order, curr_animal in zip(animal_order, animals):
                print(curr_order, curr_animal)
```

Let's explore this in Jupyter!

Conditional statements

Review: 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

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!

Defining functions

Writing functions

We have already used many functions that are built into Python or are imported from different modules/packages.

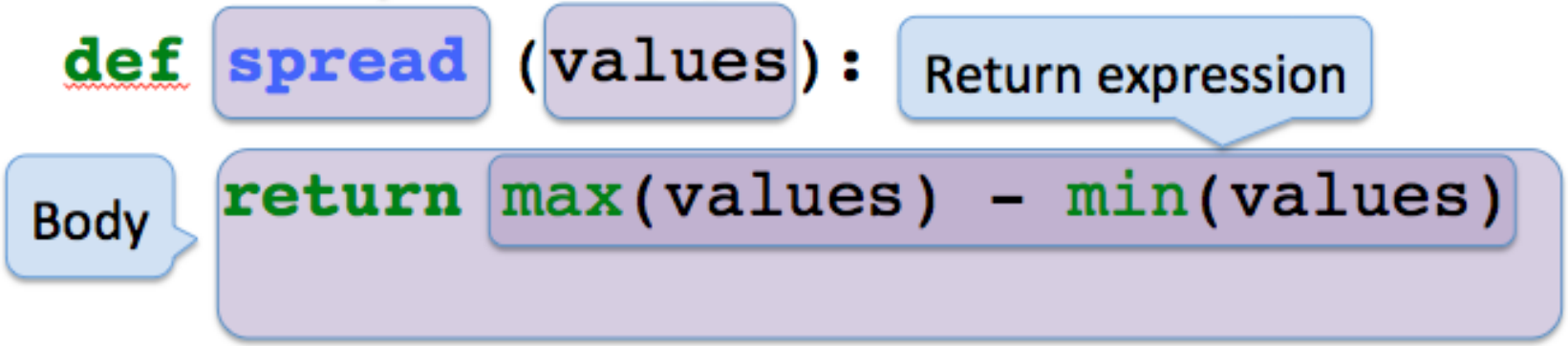
Examples…???

- sum()
- statistics.mean()
- np.diff()
- etc.

Let’s now write our own functions!

Def statements

User-defined functions give names to blocks of code



Let's explore this in Jupyter!

Practice: simulating flipping coins

Simulating flipping a coin

Let's practice writing functions by writing a function that can simulate flipping coins, where each coin has **π** probability of being heads

- Where **π** is a number between 0 and 1; e.g., **π** = 0.5 is a fair coin

We can do this using the following procedure:

1. Generate a random number between 0 and 1

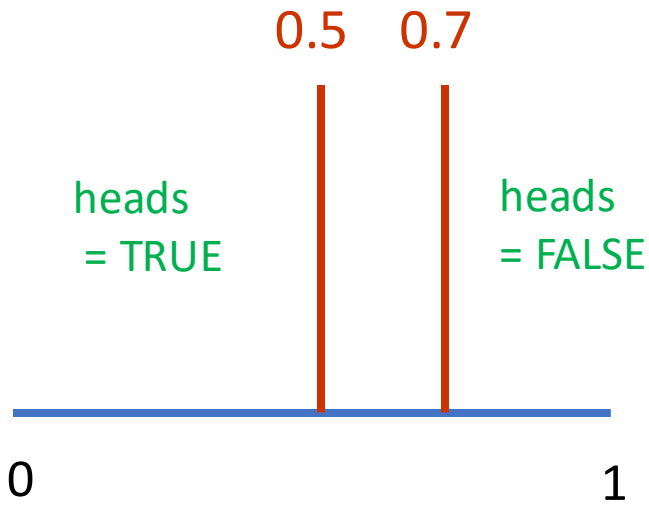
```
rand_num = np.random.rand(1)
```

2. Simulate a fair coin (.5) by mark values less than .5 as heads (True)

```
heads = rand_num <=  .5
```

3. We can simulate a biased coin that will come up with heads 70% of the time (**π** = 0.7) using:

```
rand_num = np.random.rand(1)
heads = rand_num <=  .7
```



Simulating n random coin flips

We can simulate the number of heads we would get flipping a coin ***n*** times using:

1. Generate ***n*** random numbers uniformly distributed between 0 and 1

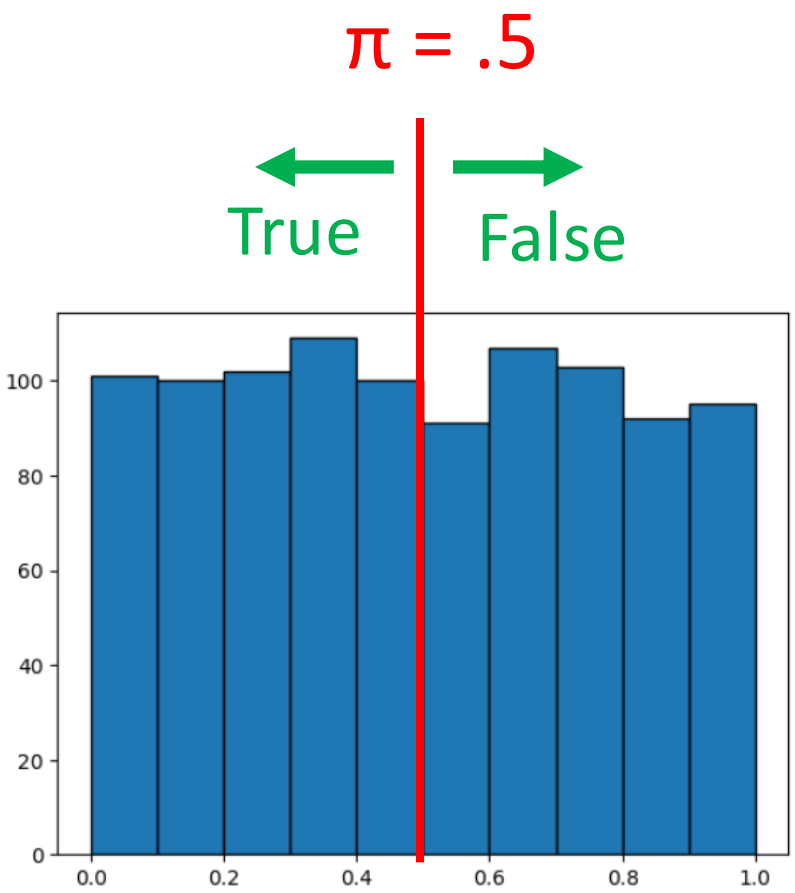
```
rand_nums = np.random.rand(n)
```

2. Mark points less than π as being True, and greater π than as being False

```
rand_binary = rand_nums <=  prob_value
```

3. Sum the number of heads (True's) we get

```
num_heads = np.sum(rand_binary)
```



Let's explore this in Jupyter!