Data 8 Final Review #1

Topics we'll cover:

- Visualizations
- Arrays and Table Manipulations
- Programming constructs (functions, for loops, conditional statements)
- Chance, Simulation, Sampling and Distributions
- Hypothesis Testing and A/B Testing

Howard, Rohan, Claire, Ryan, Hari

Final Logistics

- Thursday, 5-8pm (unless you have an alternate exam)
  - Check your seat assignment!
- Reference guides, that we give to you. Posted on Piazza!

Announcements

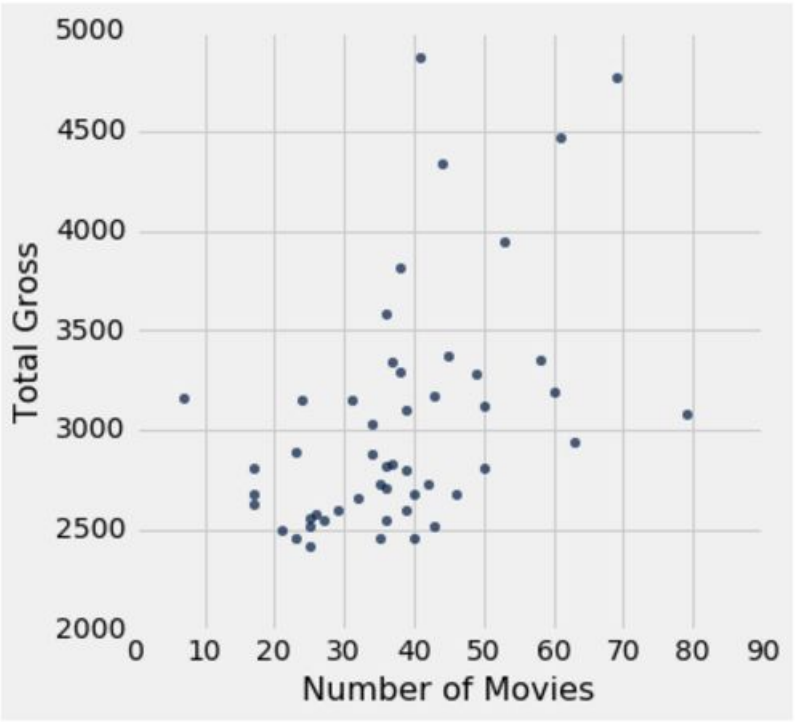
- Check your grades
- FIll out the surveys
  - We’re at 25% right now :(
  - Extra point :)))

Data Visualizations/Causality

Scatter Plots

- Display the relation between two numerical variables
- Creates one point for each row in the table
- **tbl.scatter('Column 1 (x)', 'Column 2(y)')**

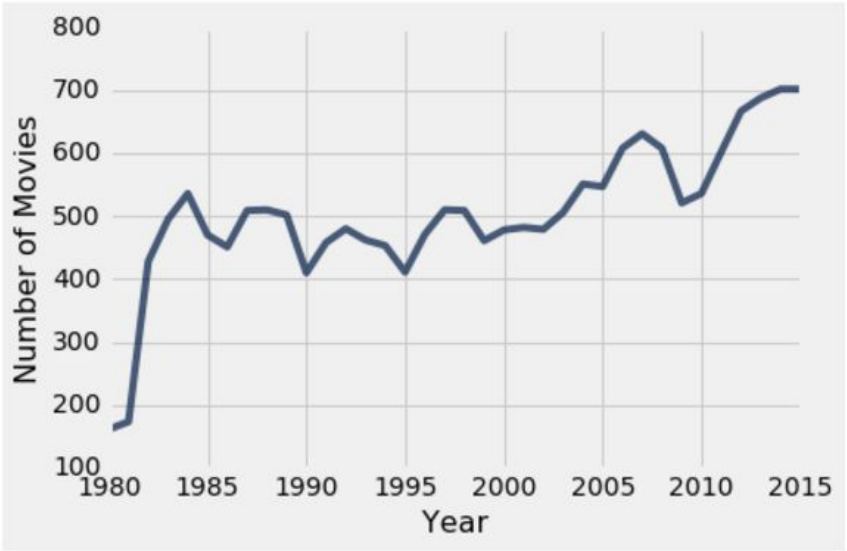
```
actors.scatter('Number of Movies', 'Total Gross')
```



Line Graphs

- Used to study **chronological trends and patterns**
  - If you ever have a **time variable,** consider line graphs!
- **tbl.plot('Column 1 (x)', 'Column 2 (y)')**

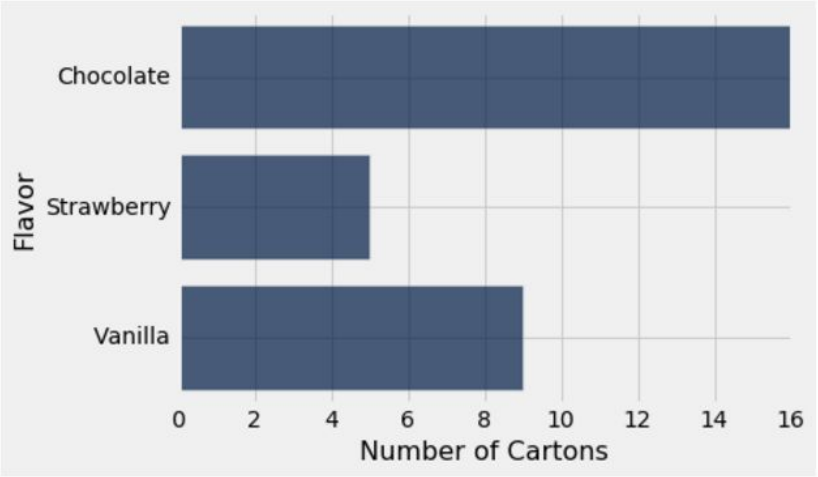
```
movies_by_year.plot('Year', 'Number of Movies')
```



Bar Charts

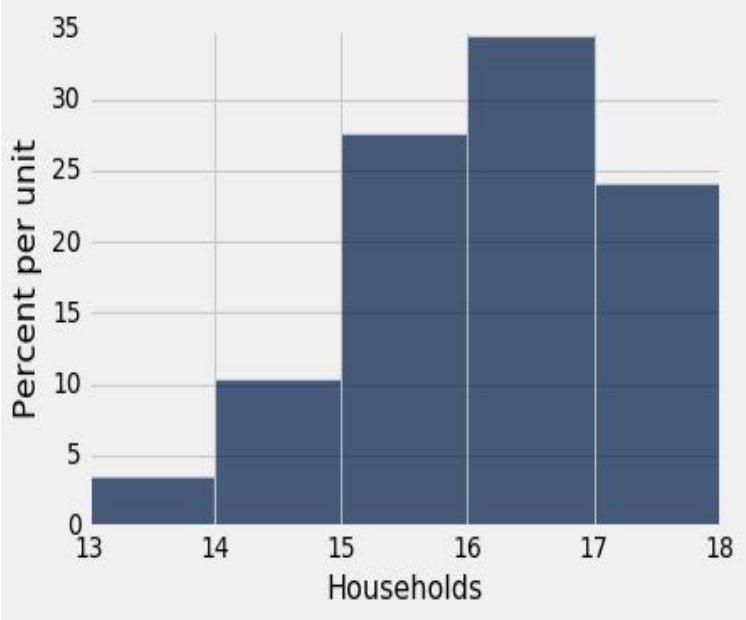
- Used to visualize categorical data
- **Length** of bar is proportional to frequency of category
- Displays equally spaced and equally wide bar for each category
- **tbl.barh(‘Categorical Variable’)**

```
icecream.barh('Flavor')
```



Histograms

- Used to visualize numerical data
- **Area** of a bar represents the percent of data points that fall into the corresponding bin
- Total area of the bars = 100%
- **tbl.hist(label, bins=np.arange(...))**



Association

- An association is defined as *any relationship* between two variables, regardless of whether or not one caused the other
- An association does not necessarily indicate causality
  - This is due to *confounding factors*
  - Defined as an underlying difference between the control and treatment group that might “confound” you when you are attempting to reach a conclusion

Observational vs Randomized Controlled

Observational Studies

- Compare some aspect of treatment group and control group
- Assignment of participants to groups is out of our control
  - e.g. smoking
- Can demonstrate an *association* between the treatment and the result

Randomized Controlled Experiments

- Compare some aspect of treatment group and control group
- Assignment to groups is **randomized** to minimize differences between the two groups (*confounding factors*)
- Can demonstrate *causality* between the treatment and the result being studied

Arrays and Tables

Array Properties

- Element-wise arithmetic operations
  - 2 * make_array(1, 2, 3)

    Result: array(2, 4, 6)
- All elements should have same type
- Element-wise array operations for arrays of the same size
  - make_array(2, 3, 5) + make_array(7, 8, 9)

    Result: array(9, 11, 14)

Table Operations

- How do we find the number of rows in a table?
  - table.num_rows

- How do we get an item out of the table?
  - table.column(‘Column Name’).item(Number)

Where

```
tbl.where(column, predicate)
```

- A table of the rows for which the column satisfies some predicate.
- “Filtering” out rows by a condition
- Returns a table

Group

```
table.group("ColName", function)
```

- *Group table by categories*
  - **1st argument:** column we want to group by
    - split column into unique values
  - **2nd argument:** (optional) aggregate function
    - If no 2nd argument, then just count # of entries for each category

Group -- Function Argument

- **No argument:** only “count” column
  - Number of rows for each category
- **Argument:** “ColName FunctionName” column
  - Applies function to values in “ColName” for each category
  - sum, mean, min, max, count, list, abs… or make your own functions and pass them in!

Group Example: sp18 table

Student ID	Class	GPA
1234	CS 61A	3.3
5678	CS 61A	3.0
5678	DATA 8	4.0

- sp18.drop('Class').group('StudentID', np.mean)

Student ID	GPA mean
1234	3.3
5678	3.5

Group with multiple columns:

- *Group table by multiple levels of categories*
  - *i.e. color and shape. WOW!*
- **First Argument:** array of column names
- table.group(make_array(‘Color’, ‘Shape’), func)

Pivot

- *Redesign table structure based on chosen categories*
- Arguments: (Pivot takes either 2 or 4 arguments)
  - **1st argument**: category unique on the columns
  - **2nd argument**: category unique on the rows
    - Note: if we stop here with 2 arguments, the default values are counts
  - 3rd argument: values to aggregate
  - 4th argument: aggregate function
    - E.g. max, np.mean, etc.

Pivot Example

Student ID	Class	GPA
1234	CS 61A	3.3
5678	CS 61A	3.0
5678	DATA 8	4.0

- sp18.pivot('Student ID', 'Class')

Class	1234	5678
CS 61A	1	1
DATA 8	0	1

Pivot Example

Student ID	Class	GPA
1234	CS 61A	3.3
5678	CS 61A	3.0
5678	DATA 8	4.0

- sp18.pivot('Student ID', 'Class', 'GPA', mean)

Class	1234	5678
CS 61A	3.3	3.0
DATA 8	0	4.0

Join

- *Combine 2 tables*
- Arguments: (Join takes 3 arguments)
  - **1st argument**: column to join on from the first table
  - **2nd argument**: second table
  - **3rd argument**: column to join on from the second table

table1.join('Username', table2, 'First Name')

Join Example

sp16

Student ID	Class	GPA
1234	CS 61A	3.3
5678	CS 61A	3.0
5678	DATA 8	4.0

prof

Course	Professor
CS 61A	Hilfinger
DATA 8	DeNero

sp16.join('Class', prof, 'Course')

Join Example

```
In [9]: sp16.join('Class', prof, 'Course')
```

Out[9]:

Class	Student ID	GPA	Professor
CS 61A	1234	3.3	Hilfinger
CS 61A	5678	3	Hilfinger
DATA 8	5678	4	DeNero

Programming Constructs

Functions

Encapsulates code so that you can use it over and over without rewriting it.

Structure of a function definition:

```
def <function name>(<zero, or, more, arguments>):
    <body>
    return <some value> # optional
```

Recall:

- **indentation matters!**
- One level of indentation is one level of organization
  - the body of a loop is indented once from the **`for`** statement.
  - the body of a function is indented once from the **`def`** statement.

Functions

This is very similar to how functions work in math

```
f(x) = x * x
```

```
f(2)
```

is equivalent to...

```
2 * 2
```

Functions

To think about how a function will work, substitute arguments with their values and substitute the value the function returns for the **function call**. So:

```
def multiply_by_three(x):
    return 3*x

multiply_by_three(4)
```

is equivalent to...

```
3*4
```

Hungry Adder

I want to write a function called **`hungry_adder`** that does the following:

1. Takes two numbers as arguments.
2. Prints the string “Thanks for feeding me those tasty numbers!”.
3. Adds the numbers together and returns the value.

Hungry Adder

I want to write a function called **hungry_adder** that does the following:

```
def hungry_adder(x, y):

    return x + y

    print(“Thanks for feeding me those tasty numbers!”)
```

Why doesn’t this function do what we want?

Hungry Adder

I want to write a function called **hungry_adder** that does the following:

```
def hungry_adder(x, y):
    return x + y
    print("Thanks for feeding me those tasty numbers!")
```

The **return** statement kicks you out of the body of the function.

This statement is never reached!

Hungry Adder

I want to write a function called **hungry_adder** that does the following:

```
def hungry_adder(x, y):
    print("Thanks for feeding me those tasty numbers!")
    return x + y
```

Must print message before returning!

If Statement

- *If ___ is true, then do ____*
- Condition: Statement evaluating to True or False

```
if <condition>:
    <if body>
elif <condition>:
    <elif body>
else:
    <else body>
```

Positive

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

sign(3)
```

Positive

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

sign(3)
```

‘Positive’

Positive

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

sign(-3)
```

Positive

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

sign(-3)
```

‘Negative’

Positive

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

sign(0)
```

What will this return?

Positive

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

sign(0)
```

Nothing!

Positive

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

    else:

        Return ‘Zero’

sign(0)
```

Positive

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

    else:

        return ‘Zero’

sign(0)
```

‘Zero’

Probability, Simulation, Sampling, and Distributions

Probabilities

Sampling with Replacement vs Sampling without Replacement

With Replacement:

- Each time we pick something, we put it back. So, if we had a population of this review session and we picked one person, we can still pick them again.

Without Replacement:

- Each time we pick something, we do not put it back. If we take someone out, then we can't pick them again!

Probabilities Continued

- What is the probability that you roll a five and a six in two rolls of a six-sided die:
  - Two ways: Add probabilities of each way!!!
  - Option 1 - 5 then 6: = P(roll 5)*P(roll6) = ⅙*⅙ = 1/36
  - Option 2 - 6 then 5: ⅙*⅙ = P(roll 6)P(roll 5) = ⅙*⅙ = 1/36
  - Adding them, the probability is 1/36 + 1/36 = 2/36
- Always think about if it's easier to calculate the chance of the opposite occurring! (HINT: AT LEAST!)

Random Selection

- *np.random.choice(array_name, sample_size)*
  - **1st argument:** array_name (array)
    - array we want to sample from
  - **2nd argument:** sample size (integer)
    - specified number of times
  - Replacement? (boolean)
    - **3nd argument:** true or false

Iterations

- It is often the case in programming that we want to repeat a process many times.
- Sometimes we want to run the process with slightly different inputs.
- This process can be automated by **looping over a sequence of elements.**

```
for i in np.arange(3):

    print(np.random.choice(['Heads', 'Tails']))
```

Heads
Heads
Tails

sample_proportions

- Model = make_arrray(0.5,0.5)
- sample_proportions(100, Model)
  - Flip a coin 100 times: we get 45 heads and 55 tails
  - This function shows us the proportions of heads and tails
  - [0.45,0.55]

sample_proportions(10, Model)

[H,H,T,H,T,T,T,T,T,T]

→ [0.3,0.7]

For loops

We automate this process in python with the `for` loop. General structure:

```
for <dummy variable> in <some sequence>:
    <body>
```

Note:

- **indentation matters!**
- One level of indentation is one level of organization
  - the body of a loop is indented once from the **`for`** statement.
  - the body of a function is indented once from the **`def`** statement.

Simulation

- Simulation is a way to run a physical experiment many times using code
- We often use simulation to approximate something we don't have the ability to figure out
  - A specific probability
  - A probability distribution
- Four steps:
  - 1: Figure out what to simulate
  - 2: Simulate one value
  - 3: Number of repetitions
  - 4. Coding the simulation

Coding a simulation

```
coin = make_array('Heads', 'Tails')
heads = make_array()

repetitions = 1000
reps_sequence = np.arange(repetitions)

for i in reps_sequence:
    flips = np.random.choice(coin, 100)
    num_heads = np.count_nonzero(flips == 'Heads')
    heads = np.append(heads, num_heads)
```

Samples

- Deterministic
- Systematic

Samples

- **Deterministic**
  - Specify the elements you want, no chance involved
  - Assuming all elements are in a **table**, you could:
    i. Use **`take`** and specify indices as an array

```
movies.take(make_array(3, 18, 100))
```

    ii. Use **`where`**

```
movies.where('Title', are.containing('Harry Potter'))
```

- Systematic

Samples

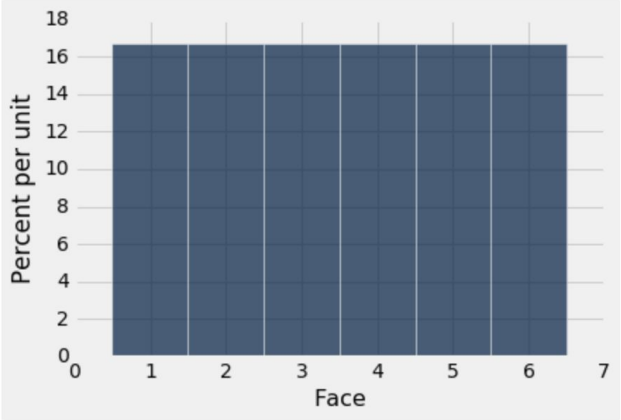
- Deterministic
- **Systematic**
  - Assume all elements of the population are listed as a sequence
  - Pick a **starting point**, at random
  - Select elements at **evenly spaced positions** after that

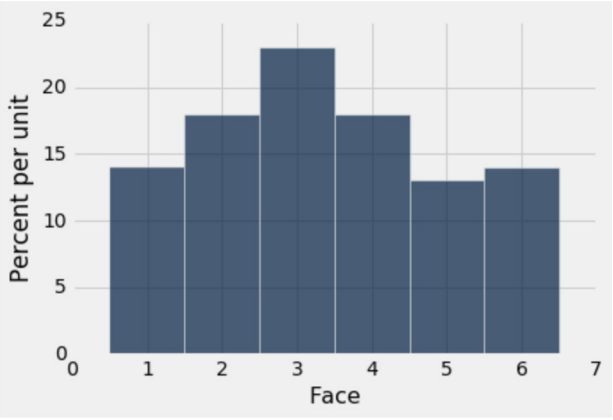
```
"""Choose a random start among rows 0 through 9;

then take every 10th row."""

start = np.random.choice(np.arange(10))

movies.take(np.arange(start, top.num_rows, 10))
```

Key Concepts

- Probability Distribution: Distribution of theoretical probabilities
- Empirical Distribution: Distributions of observed data

Consider a 6-sided, fair die. Identify what kind of distributions are shown below:

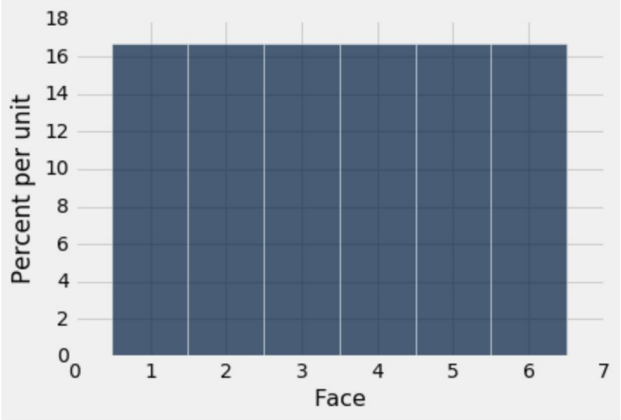




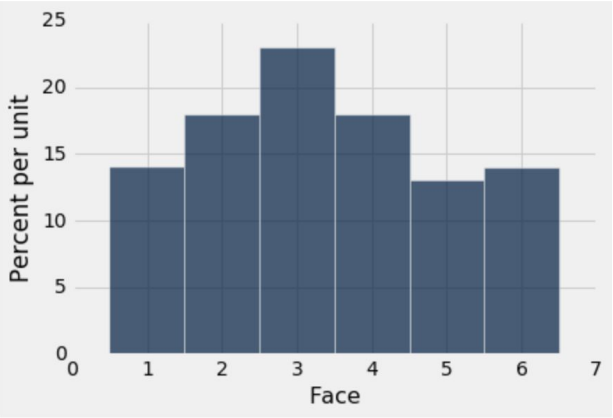
Key Concepts

- Probability Distribution: Distribution of theoretical probabilities
- Empirical Distribution: Distributions of observed data

Consider a 6-sided, fair die. Identify what kind of distributions are shown below:



Probability distribution



Empirical distribution

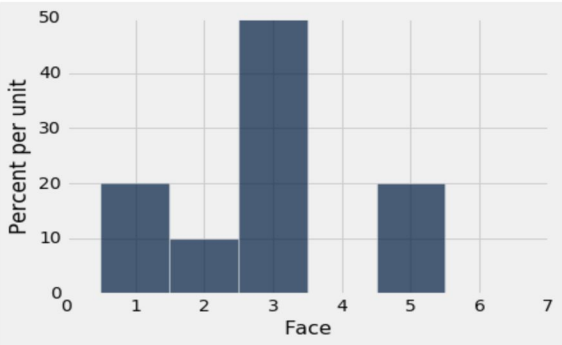
Law of Averages

If a chance experiment is repeated independently and under identical conditions, then, in the long run, the proportion of times that an event occurs gets closer and closer to the theoretical probability of the event.

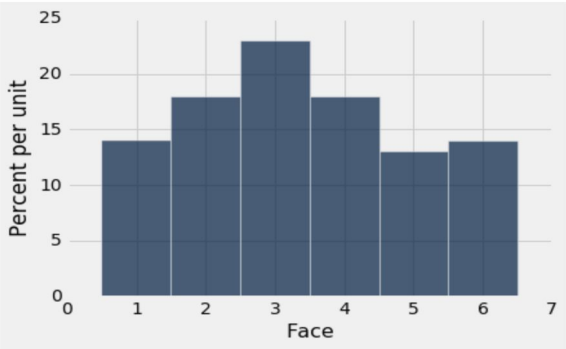
Convergence of the Empirical Histogram of the Sample

- We can use the empirical distribution to estimate the probability distribution
  - In this class, we use simple random sampling
    - Each item in the population has an equal chance of being picked
    - Sampling is done without replacement
- For a large random sample, the empirical histogram of the sample resembles the histogram of the population, with high probability

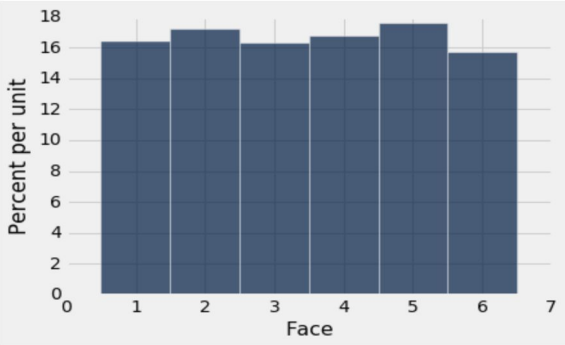
Convergence of the Empirical Histogram of the Sample



10 rolls



100 rolls



1000 rolls

Empirical Distribution of a Statistic

- Parameter: numerical quantities associated with a population
  - E.g. population mean, population median, etc.
  - This value is fixed. Doesn't change for a specific population
- Statistic: any number computed using data in a sample
  - If the sample is different, the statistic is different
- How different could the statistic be?
  - Simulate the statistic under different samples
  - Approximate the probability distribution of the statistic by using an empirical distribution
    - This is from the previous slide
  - We need to simulate a large amount of times, and our sample size must be large as well

Hypothesis Testing

The Goal

We have a model that we can simulate under (chance model). We postulate that the model could be something else (an alternative). We then observe data, and our goal is to see if the model is consistent with our chance model or the alternative.

How can we check what sort of values we should expect to observe if the chance model was current?

- Simulation!

In Detail

- Step 1: Pick your hypotheses
  - Null: Hypothesis **under which you can simulate**
  - Alternative: Predicts some reason (other than chance) as to why the data is not what the null hypothesis predicted. **Can't use to simulate.**
- Step 2: Pick a test statistic
  - A statistic computed on the observed data and samples/simulations that **helps choose between the two hypothesis. Large values point towards the alternative.**
- Step 3: Simulate the test statistic under the null hypothesis
  - What values of the test statistic should we expect if the null is true?
- Step 4: Conclusion
  - Is our observed test statistic consistent with our null hypothesis, or is it in favor of the alternative?

Conclusion of the test

- P-Value: The **chance**, based on the model of the **null hypothesis,** that the test statistic is equal to the observed value from the data, or further in **the direction of the alternative**
  - This is why need the test statistic to help differentiate the two hypotheses
  - Small value: The data is very unlikely under the model of the null, and the data supports the alternative instead
  - Use the simulation from before, as that acts as our probability distribution for the null
- P-Value Cutoff:
  - The arbitrary value at which we decide to say that the model supports the alternative instead if the p-value is less
  - Common ones: 1%, 5%

A/B Testing

- **Setup:** We want to examine two groups with respect to some quantitative variable.
- **Example:** Do apples have a longer shelf life in days than oranges?
  - Group A: Apples
  - Group B: Oranges
  - Quantitative variable of interest: Shelf life in days
- You are given a random sample from the population of all apples and oranges.
- As you examine the distributions, you observed that they are different.
- **Could the difference be due to chance alone?**

Setting up your A/B test

- **Step 1:** Write down your Null and Alternative Hypotheses:
  - Null: The distributions of [quantitative variable] for [first group] and [second group] are the same; they are from the same population. Any difference is due to chance.
  - Alternative: The distributions of [quantitative variable] for [first group] and [second group] are different, and come from different underlying populations.
- **Step 2:** Pick your test statistic. (aka choose how you want to compare the two distributions)
  - We have a couple of options but a natural one is often to look at the difference or distance between the averages.
  - **What are some other test statistics you could use here?**
  - **Think about when you want to use difference and when you want to use distance. (Absolute value!)**

Carrying out your A/B test

- **Step 3:** Simulate under the Null
  - **What the Null tell us:** If I pick out out one shelf life from my sample at random, I should not be able to tell whether that belongs to an apple or orange!
  - **What we can do with that:**
    - Shuffle all the shelf lives in the table, and attach them to our original ordering of groups.
    - Calculate the test statistic for the new shuffled groups
    - Repeat multiple times to obtain an empirical distribution for your test statistic.
- **Step 4:** Calculate the p-value associated with your observed TS and make a conclusion
  - Here we use the empirical distribution we built in step 3 and the observed value of our test statistic!

Good Luck!