Why This Isn’t a Stats Class
Why computing matters in data analytics
Recap: Of Last Time

**Step 1.** Build a logical data model

Describe dataset attributes, data types, and integrity constraints.

*Use software or manual effort to fix violations of the data model*

**Step 2.** Build a statistical data model

Describe dataset attributes in terms of their distributions

*Choose the appropriate robust/non-robust tests/statistics to work with those distributions.*
Recap: Of The Class So Far

Deep Dive into 4 Basic Stats Concepts

• Simple Random Sampling
• Stratified Sampling
• Hypothesis Testing
• Randomized Controlled Trials

Where’s the Computer Science? These are all simple formulas…
Example

Suppose you win an election by plurality, how likely is the winner Masako Holley?

Masako Holley (58%, K=50)

Genevieve Gallagos (42%, K=50)

Pearle Goodman (19%, K=50)

Margins of error only capture variations in percentage not the winner of the election!
We know how to program!

Basic statistics is the starting point for computational techniques.

Idea: Simulate different scenarios and see in how many of them Masako Holley is the winner.
Step 1. Describe Simulation Distributions

Using margin of error formulas

\[ N\left(\hat{\mu}, \frac{1}{2\sqrt{K}}\right) \]

Masako Holley (58%, K=50) \[ N(0.58, 0.07) \]

Genevieve Gallagos (42%, K=50) \[ N(0.42, 0.07) \]

Pearle Goodman (19%, K=50) \[ N(0.19, 0.07) \]
Step 2. Use Code to “Sample” From Distributions

```
numpy.random.randn
```

```
numpy.random.randn(d0, d1, ..., dn)
Return a sample (or samples) from the “standard normal” distribution.
```

```
masako_holley = np.random.randn(1)*0.07 + .58
genevieve_gallagos = np.random.randn(1)*0.07 + .42
pearle_goodman = np.random.randn(1)*0.07 + .19
```

Masako Holley (58%, K=50) \( \rightarrow \mathcal{N}(0.58, 0.07) \)

Genevieve Gallagos (42%, K=50) \( \rightarrow \mathcal{N}(0.42, 0.07) \)

Pearle Goodman (19%, K=50) \( \rightarrow \mathcal{N}(0.19, 0.07) \)
Step 3. Run Repeated Indp. Trials

```python
import numpy as np

def one_run():
    masako_holley = np.random.randn(1)*0.07 + .58
    genevieve_gallagos = np.random.randn(1)*0.07 + .42
    pearle_goodman = np.random.randn(1)*0.07 + .19

    if masako_holley > genevieve_gallagos and masako_holley > pearle_goodman:
        return 'MH'
    elif genevieve_gallagos > masako_holley and genevieve_gallagos > pearle_goodman:
        return 'GG'
    else:
        return 'PG'

simulation_output = []
for i in range(100):
    simulation_output.append(one_run())

print('Win Probability MH: ', simulation_output.count('MH')/100)
print('Win Probability GG: ', simulation_output.count('GG')/100)
print('Win Probability PG: ', simulation_output.count('PG')/100)

Win Probability MH:  0.92
Win Probability GG:  0.08
Win Probability PG:  0.0
```
Another Example

Green State Buffaloes

Springfield Armadillos

New City Vikings

Mountain High Grizzlies
Another Example

<table>
<thead>
<tr>
<th></th>
<th>Green State Buffaloes</th>
<th>New City Vikings</th>
<th>Springfield Armadillos</th>
<th>Mountain High Grizzlies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green State Buffaloes</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New City Vikings</td>
<td>36%</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Springfield Armadillos</td>
<td>24%</td>
<td>61%</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Mountain High Grizzlies</td>
<td>18%</td>
<td>21%</td>
<td>13%</td>
<td>-</td>
</tr>
</tbody>
</table>
def one_run(win_matrix):
    if np.random.rand(1) > win_matrix[('GS', 'S')]:
        final1 = 'GS'
    else:
        final1 = 'S'

    if np.random.rand(1) > win_matrix[('NC', 'MH')]:
        final2 = 'NC'
    else:
        final2 = 'MH'

    if np.random.rand(1) > win_matrix[(final1, final2)]:
        return final1
    else:
        return final2

win_matrix = flip(win_matrix)

simulation_output = []
for i in range(100):
    simulation_output.append(one_run(win_matrix))

print('Win Probability GS: ', simulation_output.count('GS')/100)
print('Win Probability S: ', simulation_output.count('S')/100)
print('Win Probability NC: ', simulation_output.count('NC')/100)
print('Win Probability MH: ', simulation_output.count('MH')/100)
Important to Know Your Distributions!

Simulating Sampling Error
np.random.randn

Simulating “Power Laws”
np.random.zipf

Simulating “Arrival” Times
np.random.exp  “Memory-less”

np.random.rand  “Uniform”
Problem?

Invalid or impossible samples

```
masako_holley = np.random.randn(1)*0.07 + .58
genevieve_gallagos = np.random.randn(1)*0.07 + .42
pearle_goodman = np.random.randn(1)*0.07 + .19
```

Can go negative!

Analytic formulas are often approximations of the physical world.
Rejection Sampling

Discard invalid or impossible samples

```python
import numpy as np

def one_run():
    masako_holley = np.random.randn(1)*0.07 + .58
    genevieve_gallagos = np.random.randn(1)*0.07 + .42
    pearle_goodman = np.random.randn(1)*0.07 + .19

    if masako_holley < 0 or pearle_goodman < 0 or genevieve_gallagos < 0:
        return 'Fail'
    if masako_holley > genevieve_gallagos and masako_holley > pearle_goodman:
        return 'MH'
    elif genevieve_gallagos > masako_holley and genevieve_gallagos > pearle_goodman:
        return 'GG'
    else:
        return 'PG'

simulation_output = []
for i in range(1000):
    simulation_output.append(one_run())

print('Failed Samples: ', simulation_output.count('Fail'))
```

Failed Samples: 3
Rejection Sampling

Yield \[= \frac{\# \text{Accepted Samples}}{\# \text{Total Samples}}\]

Overhead \[= \frac{1}{\text{Yield}}\]
Recap: Use Stats for CI

Population

Sample

“Sample Mean”

“Population Mean”

Error in an estimate is normally distributed

\[ \varepsilon \sim N(0, \frac{\sigma^2_{pop}}{K}) \]

Variance of the population

Size of the sample
Simulation Method for CIs

"Bootstrapping"

Sample

Statistic()

Sub-Sample1

Statistic()

Sub-Sample2

Statistic()

...

Sub-SampleN

Statistic()
Simulation Method

"Bootstrapping"
Simulation Method

“Bootstrapping”

No analytic formula for the output just a set of simulated observations
Recap: Hypothesis Testing

Sample 1

\( \text{stat1} \)

Sample 2

\( \text{stat2} \)

\[ z = \frac{\Delta}{\sqrt{\frac{\sigma_1^2}{K_1} + \frac{\sigma_2^2}{K_2}}} \]
Simulation Method for Hyp Testing

“Permutation” Test

Sample 1

Sample 2

Combine samples into a hypothetical population
Simulation Method for Hyp Testing

Generate new random samples from this population

- stat1
- stat2

Ignores the original partitioning!!!
Empirical Distribution of Statistics

No analytic formula for the output just a set of simulated observations
Compare to original partitioning

Sample 1

stat1

Sample 2

stat2
Simulation Method for Hyp Testing

Generate new random samples from this population

How does the original split compare to a random one?
Recap: Of The Class So Far

Deep Dive into 4 Basic Stats Concepts

- Simple Random Sampling
- Stratified Sampling
- Hypothesis Testing
- Randomized Controlled Trials

Where’s the Computer Science? These are all simple formulas…