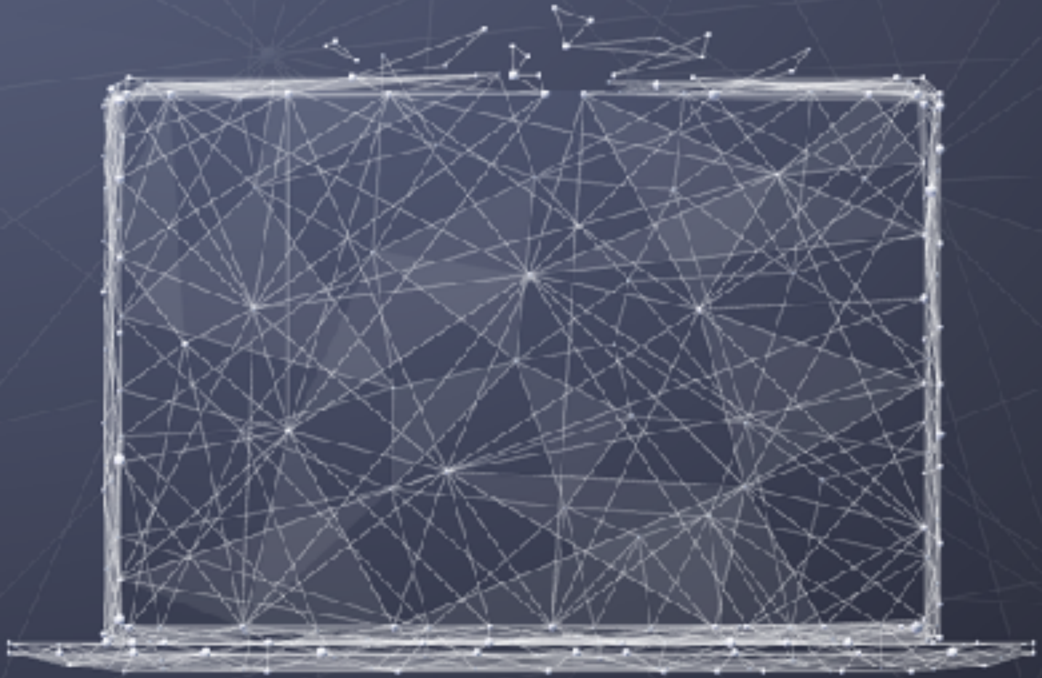
# Data Science Foundations of Decision Making

## Randomization and resampling



PURDUE UNIVERSITY | College of Science

## Computationally intensive statistical tests

- Two broad categories of computational approaches:
- In randomization we systematically shuffle observed data many times (without replacement)
    - Typically used to quantify the null distribution. That is, they seek to break whatever structure might be present in a dataset, and quantify the kinds of patterns one expects to see “purely by chance”
- In bootstrapping resampling we draw samples with replacement from the observed data
    - Typically used in cases where you’d like to extract a statistic from your data, and the sampling distribution of that statistic is not available in closed form

PURDUE UNIVERSITY | College of Science

Source: Lawrence Hunter, Colorado School of Mines

## Example

- Let’s say we measured streaming hours levels from two types of users
- A randomization approach would ask if it is likely that we would obtain a difference in streaming as large as the one we observed assuming the user types had the same streaming rates
- A bootstrap resampling approach would give us variances of population parameters of interest (e.g. mean streaming rate)

## Why use computational tests?

- Useful when we know little about the distribution from which a sample was drawn
- Useful when we know that assumptions required for other tests have been violated
- Simple/straightforward

## Randomization testing

- Randomization testing (aka permutation testing) has three steps:
  - Consider an observed sample as one of many equally possible outcomes that could have arisen by chance
  - Enumerate the possible outcomes that could be observed by randomly rearranging the data in the sample
  - On the basis of the resulting distribution of outcomes, decide whether the observed outcome is improbable enough to warrant rejection of $H_0$

PURDUE UNIVERSITY | College of Science

## Sampled randomization testing

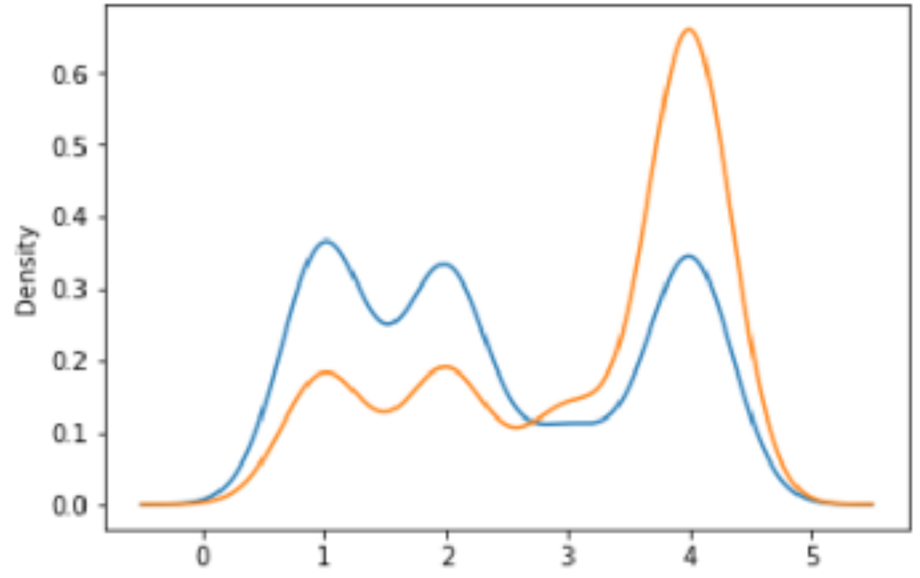
- In some cases, we cannot feasibly write down all possible outcomes
- To get around this problem:
  - We take random samples without replacement and compute our test statistic value
  - Do this enough times to generate distribution for the test statistic and then use that distribution to test the hypothesis

# Example: sampled randomization

- For example, let's say that we are considering two groups in the mammography data (1) women less than or equal to the median age, and (2) women greater than the median age

- We may want to test whether the observed difference in the 'shape' variable are significantly different between the two groups

```
d1 = data[data.Age <= ageThresh]
d2 = data[data.Age > ageThresh]
d1.Shape.plot(kind='density')
d2.Shape.plot(kind='density')
```



- If we are not testing means, and the data does not follow a parametric distribution, it can be difficult to apply conventional parametric hypothesis tests

## Example: sampled randomization

- Consider the difference in 25th percentile of the Shape attribute

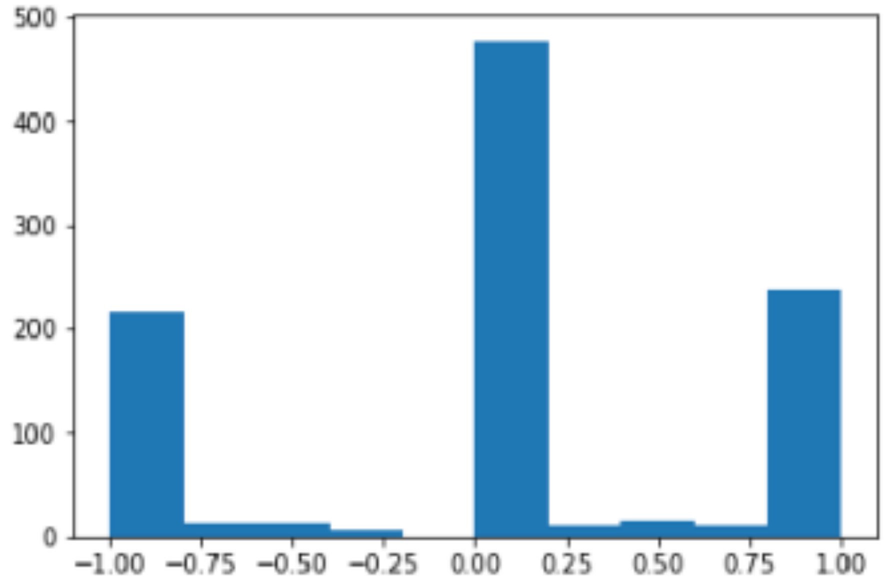
```
d1S25 = d1.Shape.quantile(0.25)
d2S25 = d2.Shape.quantile(0.25)
print('Quantiles:' +str(d1S25) +',' +str(d2S25) +'; diff=' +str(diffS25))
Quantiles:1.0,2.0; diff=-1.0
```

- The null hypothesis is that there is no difference (i.e., d1S25-d2S25=0)
- How do we generate the distribution of this statistic under the null hypothesis?

# Example: sampled randomization

- Randomize assignment to the two groups (rather than based on Age)
- Measure difference in 25th percentile of Shape variable

```
numTrials = 1000
dShape = data.Shape.values
diffQs = []
for i in range(numTrials):
    # randomly split data into two groups
    np.random.shuffle(dShape)
    partitions = np.split(dShape,2)
    tD1 = pd.DataFrame(partitions[0])
    tD2 = pd.DataFrame(partitions[1])
    g1Q = tD1[0].quantile(0.25)
    g2Q = tD2[0].quantile(0.25)
    dQ = g1Q-g2Q
    diffQs.append(dQ)
```

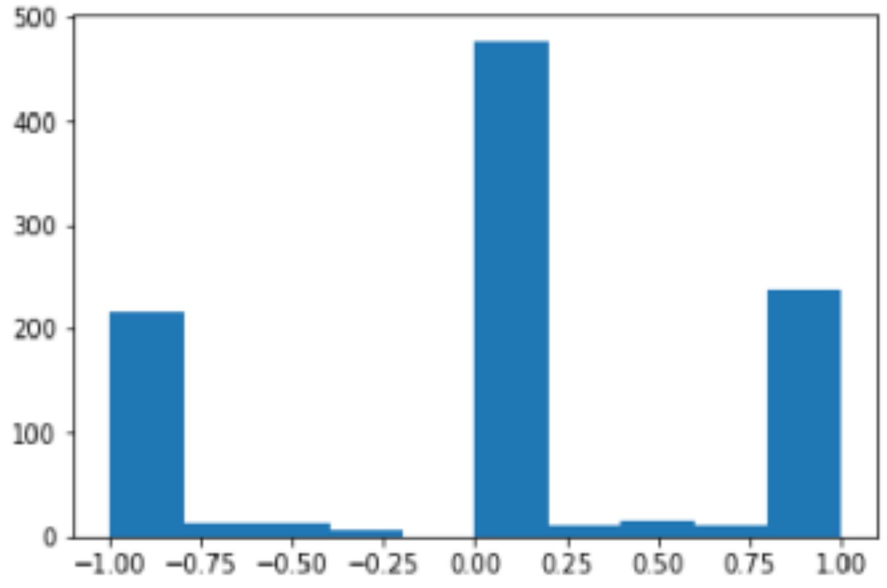


# Example: sampled randomization

- Determine if observed statistic (-1) is unlikely given the computed null distribution

```
# determine likelihood of observing a value <= -1
allQs = np.array(diffQs)
sigQs = allQs[allQs <= -1]
len(sigQs)/numTrials
0.217
```

- In this case, we would fail to reject the null, since given random groups of the data, there is 21.7% chance of observing the -1 as the difference in the 25th percentile of the Shape attribute



## Randomization testing

- Easy to choose other statistics (e.g. a ratio or anything else)
  - No need to derive analytical distribution of the statistic
- Assumes samples are independent and identically distributed (because we assumed that data point labels are exchangeable under the null hypothesis)
- Can be computationally expensive
- May not yield exact p-value

## Bootstrap resampling

- Used to assign measures of accuracy (e.g. confidence intervals, variance, bias) to sample estimates of parameters
- I.e., the bootstrap is useful for estimating the sampling distribution of a statistic without using a parametric distribution
- Assumes samples are independent and identically distributed

## Bootstrap procedure

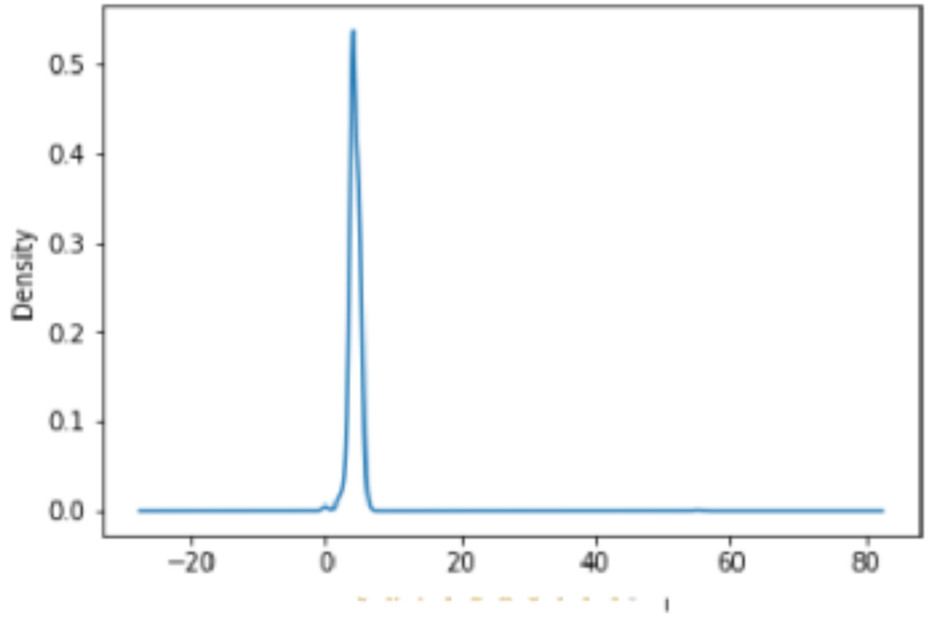
- We treat the sample as the “population”
- Then we resample with replacement from the sample and recalculate our test/descriptive statistic
- We use the resampled distribution to draw conclusions

# Example: bootstrap resampling

- Consider the BIRADS attribute in the mammography data, let's estimate a 95% CI on the mean

```
data.BIRADS.plot(kind='density')
data.BIRADS.mean()
4.347599164926931
```

- To calculate one bootstrapped mean, take a san equal in size to the original one with replacemer
- Then sample with replacement 1,000 times to generate a bootstrapped sampling distribution o the sample mean

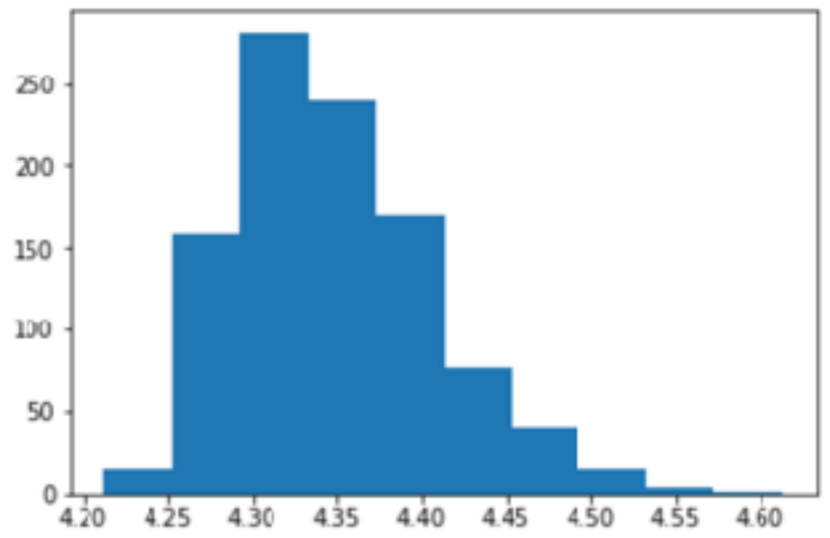


# Example: bootstrap resampling

- Using estimated sampling distribution find the 2.5th and 97.5th percentile

```
numTrials = 1000
dB = data.BIRADS
numExamples = len(dB)
dBmeans = []
for i in range(numTrials):
    # sample with replacement
    tmpB = data.BIRADS.sample(numExamples, replace=True)
    dBmeans.append(tmpB.mean())

dBmeans.sort()
print('CI=' +str(dBmeans[25]) +',' +str(dBmeans[975]))
CI=4.26123301985371,4.484880083420229
```



## Why haven't we always used computational tests?

- Computationally intensive statistical tests such as the bootstrap and permutation were popularized in the 1970s
- They are powerful (about as powerful as equivalent parametric methods, generally speaking)
- Can easily be used to adjust for multiple comparisons
- But randomization and resampling methods are computationally intensive so may be difficult to use when data is large and/or a large number of tests are being conducted