Applied Statistical Methods

Permutation Tests and Bootstrap Methods

Xuemao Zhang
East Stroudsburg University

March 13, 2023

Outline

- Permutation Tests
- Bootstrap Methods

Permutation Tests - Introductory Example

- Twelve men are recruited from among those attending a fitness clinic and asked to participate in an experiment to establish whether eating fish (but not meat) results in lower plasma cholesterol concentrations than eating meat (but not fish). The subjects are randomly allocated to the meat-eating regimens ($n_1 = 5$) and fish-eating ($n_2 = 7$). At the end of one year their plasma cholesterol concentrations are measured.

Meat eaters	6.51	7.56	7.61	7.84	11.50		
Fish eaters	5.42	5.86	6.16	6.55	6.80	7.00	7.11

```
import numpy as np
y1=np.array([6.51,7.56,7.61,7.84, 11.50])
y2=np.array([5.42,5.86,6.16,6.55,6.80,7.00,7.11])
np.mean(y1)-np.mean(y2)
```

```
## 1.7897142857142834
```

Permutation Tests - Introductory Example

- Is the difference due to the two treatments?
  - Creating multiple subplots using `plt.subplots`: https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html

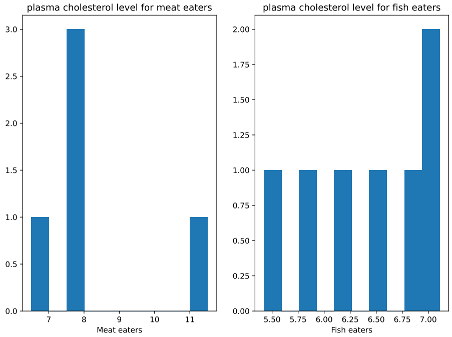
```
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.hist(y1)
ax1.set_xlabel('Meat eaters')
ax1.set_title('plasma cholesterol level for meat eaters')
ax2.hist(y2)
ax1.set_xlabel('Fish eaters')
ax2.set_title('plasma cholesterol level for fish eaters')
plt.show()
```

Permutation Tests - Introductory Example

- Histograms

```
## (array([1., 0., 3., 0., 0., 0., 0., 0., 0., 1.]), array([ 6.51 ,  7.009,
##        10.502, 11.001, 11.5  ]), <BarContainer object of 10 artists>)
```

```
## (array([1., 0., 1., 0., 1., 0., 1., 0., 1., 2.]), array([5.42 , 5.589, 5
##        6.941, 7.11 ]), <BarContainer object of 10 artists>)
```



Permutation Tests - Introductory Example

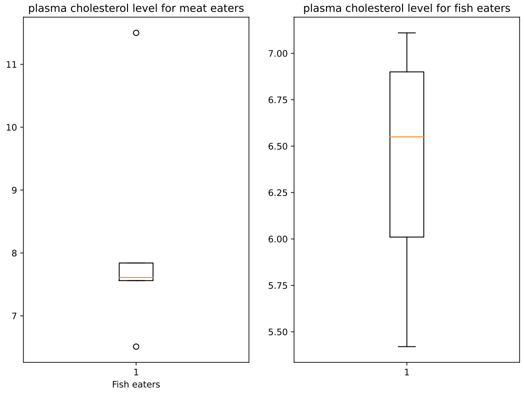
- Box plots

```
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.boxplot(y1)
ax1.set_xlabel('Meat eaters')
ax1.set_title('plasma cholesterol level for meat eaters')
ax2.boxplot(y2)
ax1.set_xlabel('Fish eaters')
ax2.set_title('plasma cholesterol level for fish eaters')
plt.show()
```

Permutation Tests - Introductory Example

- Box plots

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x000001E2A67E8A30>, <m
## {'whiskers': [<matplotlib.lines.Line2D object at 0x000001E2A67E9F60>, <m
```



Permutation Tests - Introductory Example

- R code

```r
y1=c(6.51,7.56,7.61,7.84, 11.50)
y2=c(5.42,5.86,6.16,6.55,6.80,7.00,7.11)
mean(y1)-mean(y2)
par(mfrow=c(1,2))
hist(y1, xlab="Meat eaters",
     main="plasma cholesterol level for meat eaters")
hist(y2, xlab="Fish eaters",
     main="plasma cholesterol level for fish eaters")
par(mfrow=c(1,2));
boxplot(y1, ylim = c(5, 12), xlab="Meat eaters",
        main="plasma cholesterol level for meat eaters");
boxplot(y2, ylim = c(5, 12), xlab="Fish eaters",
        main="plasma cholesterol level for fish eaters");
```

Permutation Tests

- The parametric (t-test in the example) test is based on the population model

$$Y_i \sim N(\mu_i, \sigma_i^2), i = 1, 2.$$

- If both populations are normal, then the test statistic

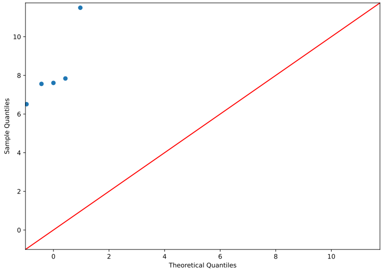
$$t_0 = \frac{\bar{y}_1 - \bar{y}_2 - 0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

  follows a t-distribution (with df determined by the Welch–Satterthwaite equation) under $H_0 : \mu_1 = \mu_2$.

Permutation Tests

- Normality of the data

```
import statsmodels.api as sm
from scipy import stats
sm.qqplot(y1, line ='45')
plt.show()
```



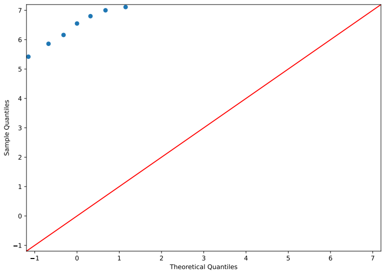
```
stats.shapiro(y1)
```

```
## ShapiroResult(statistic=0.782620906829834, pvalue=0.058042097836732864)
```

Permutation Tests

- Normality of the data

```
sm.qqplot(y2, line ='45')
plt.show()
```



```
stats.shapiro(y2)
```

```
## ShapiroResult(statistic=0.9413102865219116, pvalue=0.65054851770401)
```

Permutation Tests

- R code for normality of the data

```
qqnorm(y1)
qqline(y1)
shapiro.test(y1)
qqnorm(y2)
qqline(y2)
shapiro.test(y2)
```

Permutation Tests

- The *p*-value is defined as the probability under the assumption of no effect or no difference (null hypothesis), of obtaining a result equal to or more extreme than what was actually observed.
  - The *p*-value stands for probability and measures how likely it is that any observed difference between groups is due to chance.
  - The *p*-value is an index measuring the strength of evidence against the null hypothesis.
- *p*-values close to 0 indicate that the observed difference is unlikely to be due to chance, whereas a *p*-value close to 1 suggests there is no difference between groups other than that due to random variation.
- For example, it is common in medical journals to see adjectives such as “highly significant” or “very significant” after quoting the *p*-value depending on how close to zero the value is.

Does P-value indicate significance?

What influences *p*-value?

- Effect size
  - It is a usual research objective to detect a difference between two drugs, procedures or programs.
- Size of sample(s)
  - The larger the sample the more likely a difference to be detected.
- Spread of the data
  - The smaller the spread of observations, the lower the *p*-value.

Does P-value indicate significance?

- The threshold value, 0.05 or 0.01, is arbitrary.
- Small *p*-Value indicates large effects?
  - No! *p*-value does not tell anything about size of an effect.
  - So in general, we obtain confidence intervals as well when we conduct hypotheis tests.
  - The width of the CIs provides a measure of the reliability or precision of the estimate.
- Statistical significance implies (clinical) importance?
  - No! A large study can detect a small, clinically unimportant finding.

Permutation Tests

The Randomization Model (Fisher, 1936):

- A sample of experimental units is divided randomly into two or more groups. These are then exposed to different conditions or treatments.
- The null hypothesis is merely that the conditions or treatments have no differential effects on the groups with respect to a selected statistic such as the mean.
- There is no reference to a population. That is, no parametric model is assumed.

Permutation Tests

Permutation Principle

- If there is no difference between the two treatments, then all the datasets obtained by randomly assigning five of these cholesterol levels to the "Meat eater" and the other seven to the "Fish eater" would have equal chance of being observed in the study. That is, the two labels "Meat eater" and "Fish eater" are interchangeable. This is known as the **Permutation Principle**.
- In this case, there are $\binom{12}{5} = 792$ possible two-sample datasets.
- The distribution of the 792 differences of sample means is called the Permutation Distribution. If the treatments have same effects, the means of the cholesterol levels should be similar.

Permutation Tests

Permutation Principle

- The probability attached to the null hypothesis is then

$$\frac{\text{Number of same or more extreme outcomes as observed}}{\text{Total number of possible outcomes}}$$

- In this example, the outcomes are differences in sample means and the *p*-value is

$$\frac{\text{Number of differences } \geq \text{observed difference } 1.789714}{792 \text{ (Total number possible permutations)}}$$

Permutation Tests

- Generate all sample of size 5 from the combined data

```
import math
print(math.comb(12, 5))
```

```
## 792
```

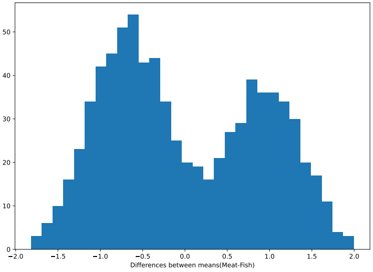
```
data=np.concatenate((y1,y2)) #combined data
from itertools import combinations
all_combinations=combinations(data, 5)

y1_combinations = list()
for i in list(all_combinations):
    y1_combinations.append(i)
#len(y1_combinations) #All possible samples of size 5
y1means=[np.mean(x) for x in y1_combinations]
y2means=[(sum(data)-sum(x))/7 for x in y1_combinations]
diff=np.array(y1means)-np.array(y2means)
```

Permutation Tests

```
import matplotlib.pyplot as plt
plt.hist(diff, bins=30)
plt.xlabel('Differences between means(Meat-Fish)')
plt.show()
```

Permutation Tests

```
## (array([ 3.,  6., 10., 16., 23., 34., 42., 45., 51., 54., 43., 44., 34.,
##         25., 20., 19., 16., 21., 27., 29., 39., 36., 36., 34., 30., 20.,
##         17., 11.,  4.,  3.]), array([-1.81714286, -1.69005714, -1.5629714
##         -1.18171429, -1.05462857, -0.92754286, -0.80045714, -0.67337143,
##         -0.54628571, -0.4192    , -0.29211429, -0.16502857, -0.03794286,
##          0.08914286,  0.21622857,  0.34331429,  0.4704    ,  0.59748571,
##          0.72457143,  0.85165714,  0.97874286,  1.10582857,  1.23291429,
##          1.36      ,  1.48708571,  1.61417143,  1.74125714,  1.86834286,
##          1.99542857]), <BarContainer object of 30 artists>)
```



Permutation Tests

- R code

```
library(combinat)
data=rbind(as.matrix(y1,5,1),as.matrix(y2,7,1))
length(data)
y1.p = combn(data, 5)
#Generate all combinations of data taken 5 at a time.
#we take 5 elements to form the 1st group
ncol(y1.p)
y1means=colMeans(y1.p[,1:792])
y2means=(sum(data)-5*y1means)/7
diff=y1means-y2means

hist(diff, breaks = 30, xlab="Difference between means
     (Meat-Fish)",main="")
box()
```

Permutation Tests

```
diffobs=np.mean(y1)-np.mean(y2)
signs=np.sign(diff-diffobs)
list(signs).count(-1)
```

```
## 786
```

```
list(signs).count(1)
```

```
## 6
```

- R code

```
diffobs=mean(y1)-mean(y2)
table(sign(diff-diffobs))
```

Permutation Tests

- The bimodal and asymmetric shape of the permutation distribution bears no resemblance to the symmetrical t distribution.
- Six of these permutations produce group means greater than or equal to the observed difference of 1.789714.
  - This corresponds to a right-sided p-value of $6/792 = 0.007576$
- Under the randomization model, the experimenters can therefore reject the null hypothesis with some confidence, and infer that diet did have an effect on plasma cholesterol concentration in the sample of men.

Permutation Tests

- Permutation test now is standard tool for testing a hypothesis of no effect in Biomedical and Bioinformatics since the distribution of the test statistic is unknown in general.
- Exact permutation tests may take too long: if $n_1 = n_2 = 15$, the total number of permutations is more than 1.5×10^8.
- Various methods to approximate the p-value of a permutation test have been developed.

Permutation Tests

- `scipy.stats.permutation_test`: Performs a permutation test of a given statistic on provided data.
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.permutation_test.html

```
import numpy as np
from scipy.stats import permutation_test

def statistic(x, y):
    return np.mean(x) - np.mean(y)

res=permutation_test((y1, y2), statistic, n_resamples=9999,
alternative='greater')
print(res.statistic)
```

```
## 1.7897142857142834
```

```
print(res.pvalue)
```

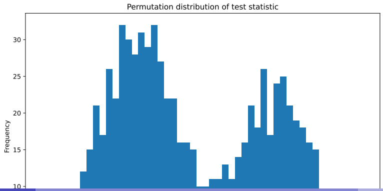
```
## 0.007575757575757576
```

Permutation Tests

```python
import matplotlib.pyplot as plt
plt.hist(res.null_distribution, bins=50)
plt.title("Permutation distribution of test statistic")
plt.xlabel("Value of Statistic")
plt.ylabel("Frequency")
plt.show()
```

Permutation Tests

```
## (array([ 1.,  2.,  5.,  3.,  8.,  8., 12., 15., 21., 17., 26., 22., 32.,
##        30., 28., 31., 29., 32., 27., 22., 22., 16., 16., 15., 10., 10.,
##        11., 11., 13., 11., 14., 16., 21., 18., 26., 17., 24., 25., 21.,
##        19., 18., 16., 15.,  9.,  9.,  7.,  4.,  4.,  1.,  2.]), array([-
##        -1.43588571, -1.35963429, -1.28338286, -1.20713143, -1.13088   ,
##        -1.05462857, -0.97837714, -0.90212571, -0.82587429, -0.74962286,
##        -0.67337143, -0.59712   , -0.52086857, -0.44461714, -0.36836571,
##        -0.29211429, -0.21586286, -0.13961143, -0.06336   ,  0.01289143,
##         0.08914286,  0.16539429,  0.24164571,  0.31789714,  0.39414857,
##         0.4704    ,  0.54665143,  0.62290286,  0.69915429,  0.77540571,
##         0.85165714,  0.92790857,  1.00416   ,  1.08041143,  1.15666286,
##         1.23291429,  1.30916571,  1.38541714,  1.46166857,  1.53792   ,
##         1.61417143,  1.69042286,  1.76667429,  1.84292571,  1.91917714,
##         1.99542857]), <BarContainer object of 50 artists>)
```



Permutation Tests

- R code

```
# There is a `coin` package in R that does permutation testing.
library(coin)
data1=rbind(cbind(y1,1),cbind(y2,2))
data2=as.data.frame(data1)
colnames(data2)<-c("resp","group")
data2$group=as.factor(data2$group)
oneway_test(resp~group,alternative ="greater",
            distribution ="exact",data=data2)
```

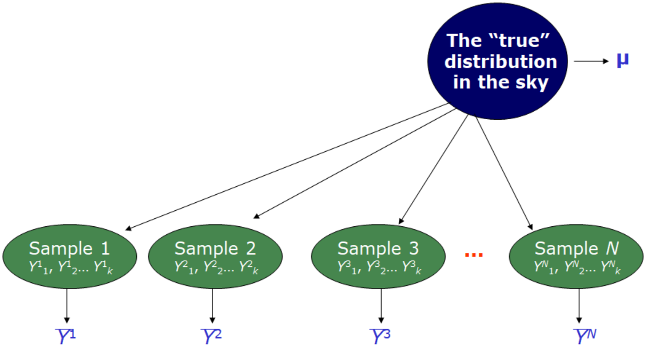
Bootstrap Methods

- Bootstrap is another resampling method.
- The bootstrap is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator and is particularly good for getting their:
  - standard errors
  - and confidence limits
- The use of the term bootstrap derives from the phrase *to pull oneself up by one's bootstraps*, widely thought to be based on one of the eighteenth century "Surprising Adventures of Baron Munchausen" by Rudolph Erich Raspe:

```
The Baron had fallen to the bottom of a deep lake. Just when it
looked like all was lost, he thought to pick himself up by his
own bootstraps.
```

Bootstrap Methods

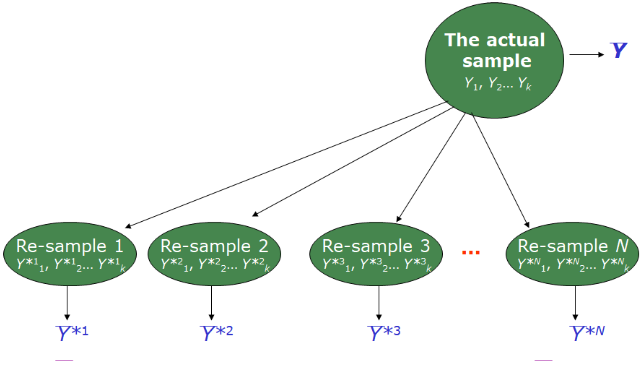
- Any actual sample of data was drawn from the unknown "true" distribution
- The distribution of our estimator ($\bar{Y}$) depends on both the true distribution and the size (k) of our sample.



The "true" distribution in the sky
μ
Sample 1
$Y^1_1, Y^1_2 \dots Y^1_k$
Sample 2
$Y^2_1, Y^2_2 \dots Y^2_k$
Sample 3
$Y^3_1, Y^3_2 \dots Y^3_k$
...
Sample N
$Y^N_1, Y^N_2 \dots Y^N_k$
$\bar{Y}^1$
$\bar{Y}^2$
$\bar{Y}^3$
$\bar{Y}^N$

Bootstrap Methods

- Treat the data as a population of size N (a proxy for the true distribution).
- Sample with replacement N times.
- Each resample simulates the process of taking a sample from the "true" distribution. Compute the statistic of interest on each "re-sample".
- $\{\overline{Y^*}\}$ constitutes an estimate of the distribution of $\overline{Y}$.



The actual sample
$Y_1, Y_2 \ldots Y_k$
Y
Re-sample 1
$Y^{*1}_1, Y^{*1}_2 \ldots Y^{*1}_k$
Re-sample 2
$Y^{*2}_1, Y^{*2}_2 \ldots Y^{*2}_k$
Re-sample 3
$Y^{*3}_1, Y^{*3}_2 \ldots Y^{*3}_k$
...
Re-sample N
$Y^{*N}_1, Y^{*N}_2 \ldots Y^{*N}_k$
Y^{*1}
Y^{*2}
Y^{*3}
Y^{*N}

Bootstrap Methods

We want to find the confidence interval for a mean from a sample of only 4 observations: 6, -3, 5, 3. If the data are from a normal population, we use the general t-interval to get a confidence interval of the population mean.

- The first thing that bootstrapping does is estimate the population distribution of Y from the four observations in the sample
- In other words, the random variable Y^* is defined:

y^*	$p^*(y^*)$
6	0.25
-3	0.25
5	0.25
3	0.25

- The mean of Y^* is then simply the mean of the sample:

$$E^*(Y^*) = \sum y^* p^*(y^*) = 2.75 = \overline{Y}$$

Bootstrap Methods

- We now treat the sample as if it were the population, and resample from it
- In this example we take all possible samples with replacement, meaning that we take $n^n = 4^4 = 256$ different samples
- Since we found all possible samples, the mean of these samples is simply the original mean
- The standard error of the sample means from these samples is:

$$SE^*(\overline{Y}^*) = \frac{\sum_{b=1}^{n^n}(\overline{Y_b}^* - \overline{Y})^2}{n^n} = 1.7455$$

We now make an adjustment for the sample size

$$\widehat{SE(\overline{Y})} = \sqrt{\frac{n}{n-1}} SE^*(\overline{Y}^*) = 2.015.$$

Bootstrap Methods

- Some of the original data points will appear more than once; others won't appear at all.
- For a sample of size 100, if we sample with replacement 100 times. In fact, there is a chance of

$$(1 - 1/100)^{100} \approx 0.366$$

  that any one of the original data points won't appear at all.
    - any data point is included with Prob ≈ 0.634.
- We treat the original sample as the "true population".
- Each resample simulates the process of taking a sample from the "true" distribution.

Bootstrap Methods

- Sampling 100 from $N(\mu = 50, \sigma = 10)$ and Bootstrap method to estimate μ
  - https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html

```
import numpy as np
from scipy.stats import norm
norm_data=norm.rvs(loc=50, scale=10, size=100, random_state=20)
b_avg = []
sd_est= []
for i in range (100):
  ystar=np.random.choice(norm_data,size=100, replace=True)
  b_avg.append(np.mean(ystar))
  sd_est.append(np.std(ystar))

np.std(b_avg)
```

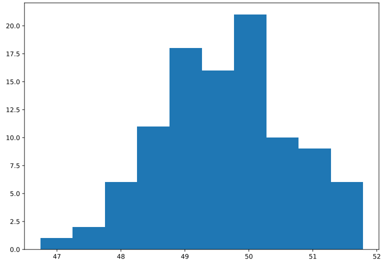
```
## 1.007680037839671
```

Bootstrap Methods

```
import matplotlib.pyplot as plt
plt.hist(b_avg)
```

```
## (array([ 1.,  2.,  6., 11., 18., 16., 21., 10.,  9.,  6.]), array([46.74
##        49.26313073, 49.7669745 , 50.27081827, 50.77466205, 51.27850582,
##        51.78234959]), <BarContainer object of 10 artists>)
```

```
plt.show()
```



Bootstrap Methods

- R code

```r
norm.data <- rnorm(100, mean=50, sd=10)
set.seed(20)
b_avg = c()
sd_est=c()
for(i in 1:100)
{
ystar=sample(norm.data,length(norm.data),replace=T)
b_avg[i] = mean(ystar)
sd_est[i]=sd(ystar)
}
sd(b_avg)
hist(b_avg)
```

Bootstrap Methods

- By the sampling theory

$$\overline{Y} \sim N(\mu_{\overline{Y}} = 50, \sigma_{\overline{Y}} = \frac{10}{\sqrt{n}} = 1)$$

Bootstrap Methods - Confidence Interval

- The first way is by the approximate normality assumption. For example, a 95% confidence interval of μ is

```
[np.mean(b_avg)-1.96*np.std(b_avg), np.mean(b_avg)+1.96*np.std(b_avg)]
```

```
## [47.63778405339637, 51.58788980172788]
```

- The second is to use the percentiles of the bootstrap histogram. A 95% confidence interval of μ is

```
import numpy as np
np.quantile(b_avg,[0.025, 0.975])
```

```
## array([47.73441183, 51.47136033])
```

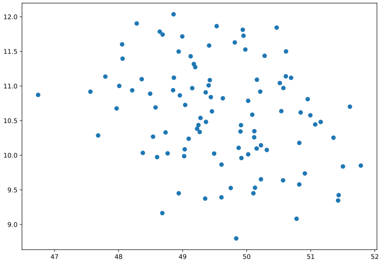
- R code for quantiles

```
quantile(b_avg, c(0.025, 0.975))
```

Bootstrap Methods - Confidence Interval

- Note that we can calculate both the mean and standard deviation of each pseudo-dataset.
- This enables us to estimate the correlation between the mean and s.d.
- Normal distribution is not skewed: mean and s.d. are uncorrelated.
- Our bootstrapping experiment confirms this.

```
plt.scatter(x=b_avg, y=sd_est)
plt.show()
```



Bootstrap Methods

- The `bootstrap()` function in `scipy` automates the bootstrap approach.
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html

```
from scipy.stats import bootstrap
norm_data = (norm_data,)  # samples must be in a sequence
bootstrap_ci=bootstrap(data=norm_data, statistic=np.mean, n_resamples=5000,
confidence_level=0.95,random_state=1, method='percentile')
ci_l, ci_u = bootstrap_ci.confidence_interval
print(ci_l)
```

```
## 47.335281547446385
```

```
print(ci_u)
```

```
## 51.48456672299607
```

```
print((ci_l+ci_u)/2)
```

```
## 49.40992413522123
```

```
print(bootstrap_ci.standard_error)
```

```
## 1.0733010990456229
```

Bootstrap Methods

- We've seen that bootstrapping replicates a result we know to be true from theory.
- Often in the real world we either don't know the 'true' distributional properties of a random variable or the distribution is not typical.
- This is when bootstrapping really comes in handy.

License