Topic 8 – Permutation tests and bootstrap

ENVX1002 Introduction to Statistical Methods

Januar Harianto

The University of Sydney

Apr 2024

THE UNIVERSITY OF SYDNEY

CRICOS 00026A TEQSA PRV12057

Recap

Non-normal data

Where data does not meet the assumptions of parametric tests, we have two options:

1. **Transform** the data, and continue with parametric tests; or
2. Use **non-parametric** “equivalents” of parametric tests at the cost of power and loss of information.

A third option exists.

- Use *computer intensive*, randomisation-based methods to test hypotheses, called **resampling techniques**.
- These methods include **randomisation (or permutation) tests** and **bootstrap**.
- Retains estimates of **effect size** and **confidence intervals**.

Resampling techniques

> Two roads diverged in a yellow wood,
> And sorry I could not travel both
> And be one traveler, long I stood
> And looked down one as far as I could
> To where it bent in the undergrowth;

– Robert Frost, The Road Not Taken, 1916

Model-based inferential techniques

- Traditionally, inferential statistics is based on **mathematical approximations** and *assumptions* about how data is obtained.
- Based on knowledge that “randomness” somehow obeys certain patterns in nature which can be *reliably* described by **probability distributions**.
- Uses **probability theory** to draw approximate conclusions about these patterns when we observe data.

Resampling techniques

- Based on the idea that we can use the **data itself** to estimate the distribution of the test statistic or parameter of interest.
- These methods are **model-free** and **distribution-free**.
- Requires comparatively higher computational power, **but nowadays it is not a problem** – any modern personal computer can handle it.

Randomisation or Bootstrap?

They are not the same:

Randomisation

Generate a distribution of the **test statistic** under the null hypothesis by randomly sub-sampling the data, *without replacement*. Can be used to estimate a **p-value**.

> Basically, shuffle the data and see what happens.

Bootstrap

Generate a distribution of the **parameter** of interest (e.g. mean) by resampling the data with replacement[1]. Can be used to estimate **confidence intervals**.

> Basically, create alternative versions of the data and see what happens.

Why would these techniques work?

At the core of the resampling approach is the idea that the **observed data** is a **random sample** from a **larger population**.

If the sampled data is truly representative of the population…

Then, if we *infinitely resample from the sample itself*, we should be able to *somewhat* approximate the distribution of the test statistic under the null hypothesis, or parameter of interest (will show example later).

Randomisation tests

To generate a distribution of the test statistic under the null hypothesis.

Example: comparing two groups

- Suppose we have two samples (groups) and we want to test if the **mean scores**[1] are different.
- Under the **null hypothesis** that there is *no difference* between the groups, the two sets of scores will have the same distribution.
- Thus, we can **pool** the scores and reassign them to the two groups, since any score is equally likely to belong in either group, i.e. the scores are **exchangeable**.

Steps

1. **Pool** the scores from both groups into a single dataset.
2. **Randomly reassign** the scores to two groups.
3. Calculate the **test statistic** of interest, in this case the *t*-test statistic.
4. Repeat steps 2 and 3 many times to generate a distribution of the test statistic under the null hypothesis.
5. Compare the **observed** test statistic to the **randomised** distribution to calculate a **p-value**.

Data

The `sleep` dataset in R contains the average **extra** hours of sleep, compared to control, for 10 patients who were given two different drugs.

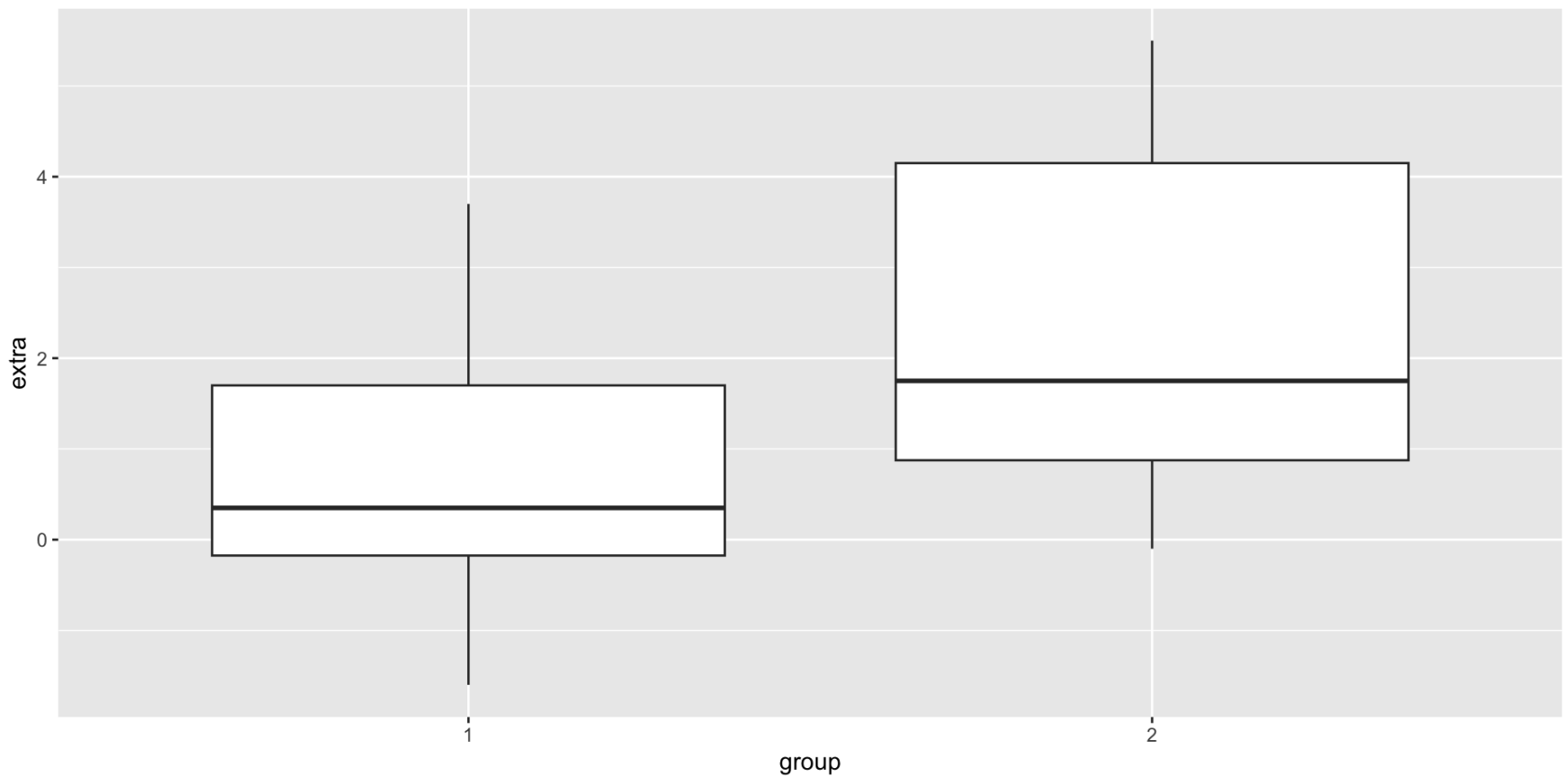
```
library(tidyverse)
glimpse(sleep)
```

```
Rows: 20
Columns: 3
$ extra <dbl> 0.7, -1.6, -0.2, -1.2, -0.1, 3.4, 3.7, 0.8, 0.0, 2.0, 1.9, 0.8, …
$ group <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
$ ID    <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Are assumptions of normality met?

We picked a dataset where the assumptions are met, so that we can compare the results with the parametric test.

▶ Code



Calculating the *t*-test statistic

Recall that the test statistic for the two-sample *t*-test is:

$$t = \frac{observed\ value - expected\ value}{standard\ error}$$

We could calculate it manually, but let's just use the `t.test()` function in R since the function calculates the test statistic for us. For example, the observed test statistic for the `sleep` data is:

```
1 t.test(extra ~ group, data = sleep)$statistic
```

```
        t 
-1.860813 
```

Step 1: Pool the scores

The first step is to pool the data. The pooled data is:

```
pooled_data <- sleep$extra
pooled_data
```

```
 [1]  0.7 -1.6 -0.2 -1.2 -0.1  3.4  3.7  0.8  0.0  2.0  1.9  0.8  1.1  0.1 -0.1
[16]  4.4  5.5  1.6  4.6  3.4
```

Where the first 10 scores are from the first group, and the next 10 scores are from the second group.

Step 2: Randomly reassign the scores

▶ Code

```
 [1]  0.7 -1.6 -0.2 -1.2 -0.1  3.4  3.7  0.8  0.0  2.0  1.9  0.8  1.1  0.1 -0.1
[16]  4.4  5.5  1.6  4.6  3.4
```

Next, we randomly shuffle the pooled data and re-assign the first 10 scores to group 1, and the next 10 scores to group 2.

```
set.seed(1022)
shuffled_data <- sample(pooled_data)
shuffled_data
```

```
 [1] -1.2  2.0  3.7  0.1  0.8  1.6  0.8  1.9  0.0  3.4 -1.6 -0.2  1.1 -0.1  0.7
[16]  4.4  5.5  3.4 -0.1  4.6
```

```
group1 <- shuffled_data[1:10]
group2 <- shuffled_data[11:20]
```

Step 3: Calculate the test statistic

We're not using the results from the `t.test()` function, but just extracting the test statistic.

```
t.test(group1, group2)$statistic
```

```
          t
-0.4995608
```

Step 4: Repeat many times

Putting it all together, we can write a function to obtain the test statistic:

▶ Code

If we use the function with the same seed the result should be identical to before:

```
# Test the function
set.seed(1022)
random_t(pooled_data)
```

```
          t
-0.4995608
```

Repeat the function 100,000 times:

```
set.seed(1034)
random_t_values <- replicate(10000, random_t(pooled_data))
```

Step 5: Compare the observed test statistic

Finally, we can compare the observed test statistic to the randomised distribution. This can be done by calculating the proportion of randomised test statistics that are more extreme than the observed test statistic.

```
fit <- t.test(extra ~ group, data = sleep) # test on observed data
p_value <- mean(abs(random_t_values) >= abs(fit$statistic))
round(p_value, 2)
```

```
[1] 0.08
```

How does this compare to the parametric t-test?

If we round the p-values of both tests to two decimal places, we get:

```
round(fit$p.value, 2)
```

```
[1] 0.08
```

As we can see, the p-values are very similar. This is because the assumptions of the parametric test are met, so the results will be close even though the methods are different!

What's the difference between the two techniques?

t-test

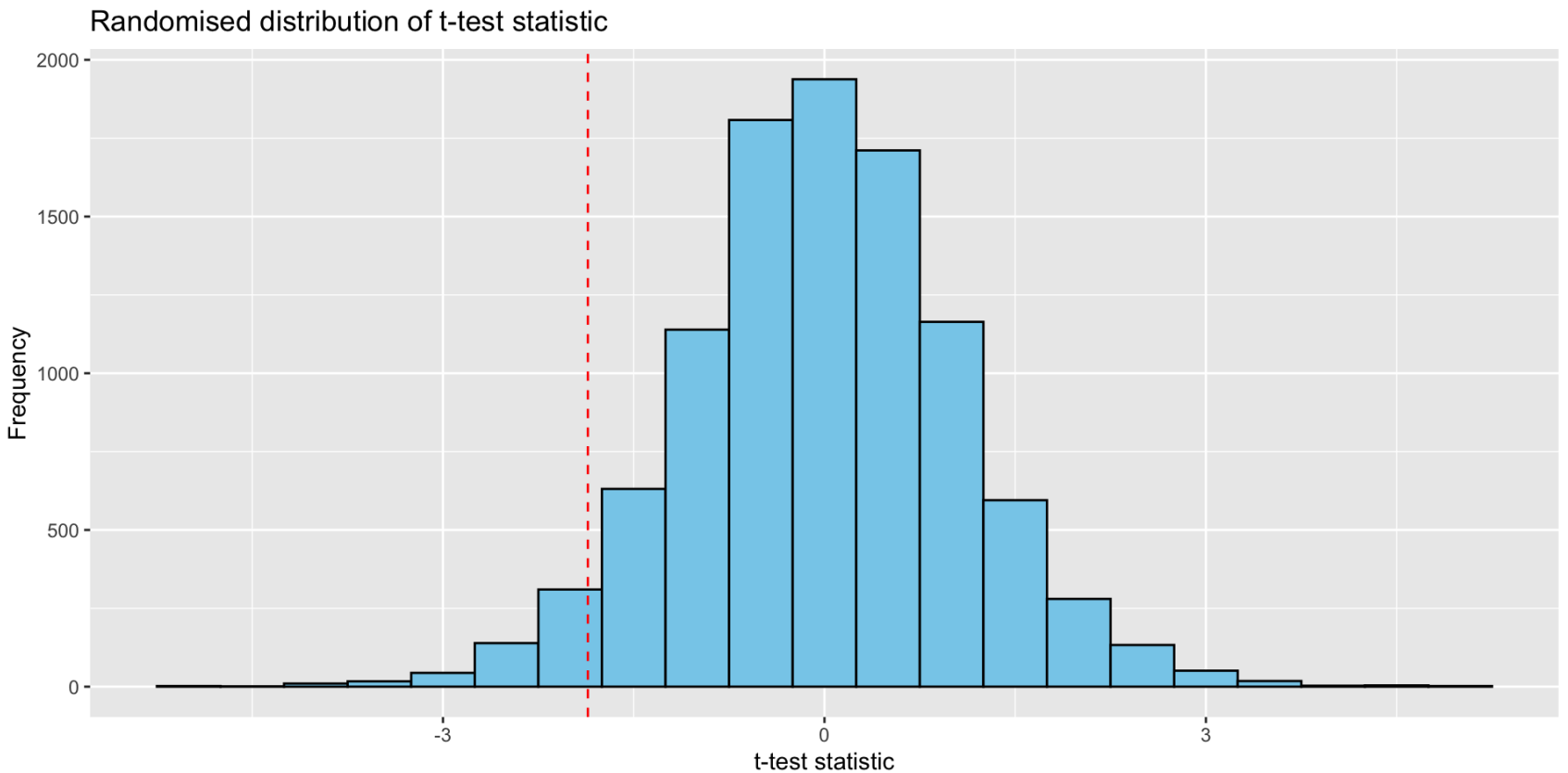
- The **parametric** test compares the **observed** test statistic to a **theoretical** distribution that has fixed parameters.
- Assumes that the data is **normally distributed**.
- P-value is calculated from the **theoretical** distribution.

Randomisation test

- The **randomisation** test compares the **observed** test statistic to a **randomised** distribution that is generated from the data itself.
- **No assumptions** about the data distribution, but if the assumption were met, the results would be similar.
- P-value is calculated from the *simulated* distribution.

Visualising the randomised distribution

▶ Code



We can see that the observed test statistic is well within the distribution of the randomised test statistics (which is normally distributed).

Bootstrap

To generate a distribution of the parameter of interest.

Example: estimating the mean

- Suppose we have two samples (groups) and we want to estimate the **difference in means**, and the **95% confidence interval** of the difference.
- We can use the usual **mathematical equation** to calculate 95% CI, but if the data *does not meet the assumption of normality*, then the CI will be a bad estimate.
- Instead, we can use the **bootstrap** to estimate the 95% CI, which is based on the **simulated distribution of the mean difference**.

Steps

1. **Resample** the data with replacement.
2. Calculate the **parameter of interest** (e.g. mean) for each resample.
3. Repeat steps 1 and 2 many times (N) to generate a distribution of the parameter of interest.
4. Calculate the **95% confidence interval** from the simulated distribution:
    - The mean of the distribution is the **point estimate**.
    - The $0.025 \times N$th smallest mean is the **lower bound** of the 95% CI.
    - The $0.975 \times N$th smallest mean is the **upper bound** of the 95% CI.

Data

The `BOD` dataset in R contains the **biochemical oxygen demand** (mg/L) measurements of 6 samples over time.

```
1 glimpse(BOD)
```

```
Rows: 6
Columns: 2
$ Time   <dbl> 1, 2, 3, 4, 5, 7
$ demand <dbl> 8.3, 10.3, 19.0, 16.0, 15.6, 19.8
```

Step 1: Resample the data

From the pooled original data:

```
BOD$demand
```

```
[1]  8.3 10.3 19.0 16.0 15.6 19.8
```

We `sample()` with replacement:

```
set.seed(1113)
resampled_data <- sample(BOD$demand, replace = TRUE)
resampled_data
```

```
[1] 15.6  8.3  8.3  8.3 16.0  8.3
```

Noting that some scores will be repeated, and some will be missing.

Step 2: Calculate the parameter of interest

The parameter of interest is the mean value.

```
1 mean(resampled_data)
```

```
[1] 10.8
```

Step 3: Repeat many times

Since it's a simple process, we can write a function to calculate the mean:

▶ Code

Then repeat the function 10,000 times:

```
set.seed(1116)
bootstrap_means <- replicate(10000, bootstrap_mean(BOD$demand))
```

Step 4: Calculate the 95% CI

The 95% CI is calculated from the simulated distribution:

```
meanval <- mean(bootstrap_means)
CI <- quantile(bootstrap_means, c(0.025, 0.975))
```

Putting it together, the mean is 14.85131 with a 95% CI of [11.25, 18.13].

How does this compare to the parametric test?

If we use the `t.test()` function to calculate the 95% CI:

```
t.test(BOD$demand)
```

```
    One Sample t-test

data:  BOD$demand
t = 7.8465, df = 5, p-value = 0.0005397
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  9.973793 19.692874
sample estimates:
mean of x
 14.83333
```

How different are the results?

Method	Mean	95% CI	CI size
Bootstrap	14.85	[11.25, 18.13]	6.88
Parametric	14.83	[9.97, 19.69]	9.72

- The **point estimates** of the mean are almost identical.
- The **95% CI** of the mean are similar, but the **bootstrap** CI is non-symmetric - it represents the **true** distribution of the mean.
- The **size** of the CI is smaller for the **bootstrap** method, indicating that the **parametric** method is **overestimating** the precision of the estimate.

Modern statistics using `tidymodels`

The `tidymodels` framework

- It is clear that bootstrapping and randomisation tests are powerful tools for modern statistics, but they can be cumbersome to implement manually.
- Interestingly, modern data science prefers resampling techniques over traditional methods even when the assumptions are met, because they are more robust and provide more information.
- The `tidymodels` framework in R contains the `infer` package which provides a simple interface to perform most parametric tests using resampling techniques by default!

TLDR

The general trend in modern statistics is to use resampling techniques over traditional methods, even when the assumptions are met – and this is currently led by the `tidymodels` framework in R.

Using `infer`

```
1 library(tidymodels)
```

Let's use the `sleep` dataset to demonstrate how to use the `infer` package to perform a randomisation test (also makes it easier to compare against manual method).

The `infer` package requires the user to use an expressive grammar to specify the analysis.

Steps

1. `specify()` the response variable of interest, then
2. `hypothesise()` the null hypothesis, then
3. `generate()` the null distribution, and finally
4. `calculate()` the p-value.

Two-sample t-test using `infer`

First we need to calculate the observed test statistic so that we can compare it to the simulated distribution.

```
observed <- sleep %>%
  specify(extra ~ group) %>%
  calculate(stat = "diff in means", order = c(1, 2))
observed
```

```
Response: extra (numeric)
Explanatory: group (factor)
# A tibble: 1 × 1
   stat
  <dbl>
1 -1.58
```

Then we generate the null distribution and calculate the p-value:

```
set.seed(1034)
pval_infer <-
  sleep %>%
  specify(extra ~ group) %>%
  hypothesise(null = "independence") %>%
  generate(reps = 10000, type = "permute") %>%
  calculate(stat = "diff in means", order = c(1, 2)) %>%
```

What are the differences?

Method	P-value
Manual	0.082
`infer`	0.084
`t.test()`	0.079

As we can see, the results are very similar because the assumptions of the parametric test were already met.

> **Note**
>
> To calculate confidence intervals, use the `get_ci()` function as documented here.

What about non-normal data?

Example: beetles

The `beetle` dataset was used in last week's lecture to demonstrate the non-parametric Wilcoxon rank-sum test.

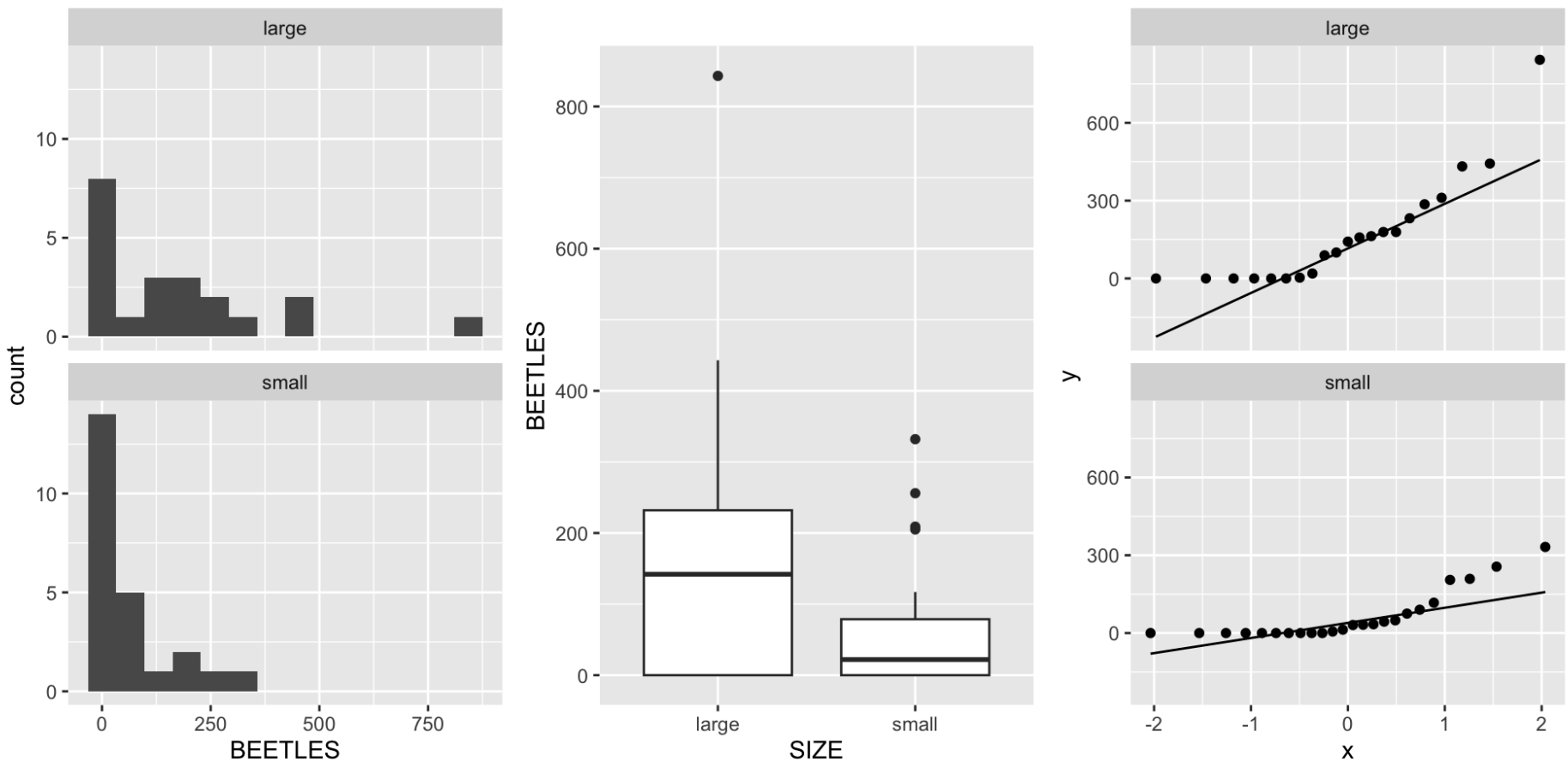
```
beetle <- readr::read_csv("data/beetle.csv")
glimpse(beetle)
```

```
Rows: 45
Columns: 2
$ SIZE    <chr> "small", "small", "small", "small", "small", "small", "small",…
$ BEETLES <dbl> 256, 209, 0, 0, 0, 44, 49, 117, 6, 0, 0, 75, 34, 13, 0, 90, 0,…
```

Assumption

Recall that the data does not meet the assumptions of normality:

▶ Code



T-test via resampling using `infer`

First, calculate the test statistic:

```
observed <- beetle %>%
  specify(BEETLES ~ SIZE) %>%
  calculate(stat = "diff in means", order = c("small", "large"))
```

Then generate the null distribution and calculate the p-value:

```
set.seed(1034)
pval_infer <-
  beetle %>%
  specify(BEETLES ~ SIZE) %>%
  hypothesise(null = "independence") %>%
  generate(reps = 10000, type = "permute") %>%
  calculate(stat = "diff in means", order = c("small", "large")) %>%
  get_p_value(obs_stat = observed,
              direction = "two-sided")

pval_infer
```

```
# A tibble: 1 × 1
  p_value
    <dbl>
1   0.025
```

Comparisons

Let's compare the p-values from

1. a t-test (if we ignore violations of assumptions),
2. the wilcoxon rank-sum test, and
3. the `infer` randomisation test.

Comparisons

T-test

```
p_ttest <- t.test(BEETLES ~ SIZE, data = beetle)$p.value %>%
  round(3)
```

Wilcoxon rank-sum test

```
p_wilcox <- wilcox.test(BEETLES ~ SIZE, data = beetle)$p.value %>%
  round(3)
```

Randomisation test

```
p_infer <- pull(pval_infer, p_value) %>%
  round(3)
```

Results

Method	P-value
T-test	0.037
Wilcoxon	0.075
`infer`	0.025

- As we can see, the p-values are quite different because the assumptions of the parametric test were violated.
- The randomisation test is more robust and provides a more accurate estimate of the p-value than the Wilcoxon rank-sum test.

How to report results of randomisation test

The results of the randomisation test can be reported as follows:

> Beetle consumption was significantly different between small and large beetles (t = 2.19, R = 10000, p = 0.025).

Summary

- Resampling techniques are **model-free** and **distribution-free** and requires only that the data is a random sample that is representative of the population.
- If the **assumptions of parametric tests are met**, the results of resampling techniques will be **similar** to traditional methods.
- No information is lost in resampling techniques, and they are more robust than traditional methods.

Thanks!

This presentation is based on the SOLES Quarto reveal.js template and is licensed under a Creative Commons Attribution 4.0 International License.