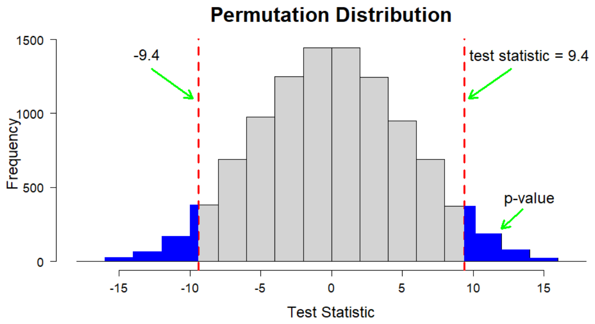
# Introduction to Permutation Tests



## Outline

## Permutation Tests

**A new approach to statistical inference:**

- ***Permutation-based hypothesis testing***, which is also called ”hypothesis testing with randomisation”.
- We will simply refer to the tests as ”***permutation tests***”.

## Learning Objectives

**In this video, we will**

- Elaborate the idea of permutation tests.
- Implement a permutation test in ***R***.

## Introduction to Permutation Tests

## Why are we interested in permutation tests?

1. Permutation tests allow us to use any test statistic for which the sampling distribution has not been derived yet.
   - In the two-sample set-up, examples include the difference of medians, ratio of means, and ratio of variances.
2. Permutation tests require less distributional assumptions.
   - We do not need to assume a particular family of distributions for the population data.
   - In particular, permutation tests work well in the cases when the sample size is small, or the population data don't follow a normal distribution.

## How to implement permutation tests?

- Randomised experiment: 20 students were randomly split into two groups: A and B.
- $\boldsymbol{H_0}$ : The popuation mean scores of the two groups are the same.
- If $H_0$ is ture, the difference between the two samples' mean scores should be due to natural variability inherent to the sample dataset.
- We will set up a sham randomised experiment by reallocating the group order.
- We will refer to the sham experiment's result as the "***permutation sample***".

***Original Sample***

| Group | Scores | Group | Scores |
|---|---|---|---|
| A | 64 | B | 60 |
| A | 66 | B | 69 |
| A | 50 | B | 84 |
| ... | ... | ... | ... |
| A | 59 | B | 61 |
| A | 65 | B | 66 |
| A | 43 | B | 54 |

## “Shuffle the Cards”

***Original Sample***

| Group | Scores | Group | Scores |
|---|---|---|---|
| A | 64 | B | 60 |
| A | 66 | B | 69 |
| A | 50 | B | 84 |
| ... | ... | ... | ... |
| A | 59 | B | 61 |
| A | 65 | B | 66 |
| A | 43 | B | 54 |

***Permutation Sample***

| Group | Scores | Group | Scores |
|---|---|---|---|
| A | 64 | A | 60 |
| B | 66 | B | 69 |
| A | 50 | B | 84 |
| ... | ... | ... | ... |
| B | 59 | B | 61 |
| A | 65 | A | 66 |
| A | 43 | B | 54 |

## “Shuffle the Cards”

***Original Sample***

| Group | Scores | Group | Scores |
|---|---|---|---|
| A | 64 | B | 60 |
| A | 66 | B | 69 |
| A | 50 | B | 84 |
| ... | ... | ... | ... |
| A | 59 | B | 61 |
| A | 65 | B | 66 |
| A | 43 | B | 54 |

***Sorted Permutation Sample***

| Group | Scores | Group | Scores |
|---|---|---|---|
| A | 64 | B | 66 |
| A | 60 | B | 69 |
| A | 50 | B | 84 |
| ... | ... | ... | ... |
| A | 66 | B | 61 |
| A | 65 | B | 59 |
| A | 43 | B | 54 |

## Implement A Permutation Test in R

## Some Preparations in R

- We first prepare the variables.

```
treatment <- df1$treatment
outcome <- df1$outcome
```

```
> treatment
 [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "b" "b" "b" "b" "b" "b" "b" "b"
[19] "b" "b"
> outcome
 [1] 64 66 50 36 43 60 66 59 65 43 60 69 84 81 48 59 64 61 66 54
```

- We can access one variable of the data frame, by specifying the name of the data frame, followed by a dollar sign and the variable name.

## “sample()” Function

- The base R function “***sample()***” can help us reallocate the order of a variable or a vector.

```r
set.seed(123)
sample(1:5, size = 1)
```

```
[1] 3
```

```r
sample(1:5, size = 5, replace=FALSE)
```

```
[1] 3 2 4 5 1
```

- It will randomly draw 5 numbers from 1 to 5, without replacement.
- This is equivalent to returning a permutation sequence of 1 to 5.

## Generating a Permutation Sample

- We use the following code to generate a new group order for the permutation sample.

```
treatment_p <- sample(treatment, size= length(treatment), replace=
   FALSE)
treatment_p
```

```
 [1] "a" "b" "b" "a" "b" "a" "a" "a" "a" "b" "b" "b" "a" "a" "a" "b" "b" "b"
[19] "a" "b"
```

- We define the test statistic as the difference of the two samples' means.
- We use the following code to calculate the test statistic of the permutation sample.

```
mean(outcome[treatment_p == "b"]) - mean(outcome[treatment_p == "a"])
```

```
[1] -6
```

```
outcome[treatment_p == "a"]
```

```
[1] 64 36 60 66 59 65 84 81 48 66
```

```
outcome[treatment_p == "b"]
```

```
[1] 66 50 43 43 60 69 59 64 61 54
```

## “for loop”

- The syntax of ***for loop*** is as follows,

```
for (variable in sequence)
{
statement
}
```

- Let us use the following example to explain how it works.

```
sum <- 0
for (i in 1:2) {
  sum <- sum + i
}
sum
```

```
[1] 3
```

- The example is iteratively calculating $1 + 2$.

## Use “for loop” to Generate Multiple Permutation Samples

- Some preparations:

```
size <- 10000
perm_dist <- rep(0, size)
```

- To generate $10,000$ permutation samples:

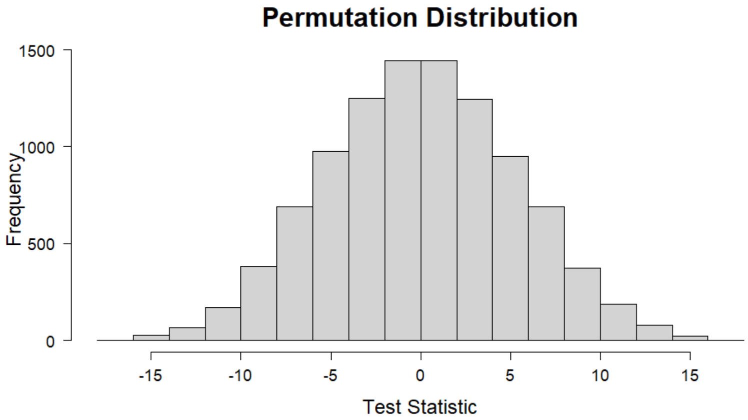
```
for(i in 1:size){
    # Take a permutation sample without replacement
    treatment_p <- sample(treatment, size= length(treatment), replace
       =FALSE)
    # For each permutation sample, calculate the test statistic
    perm_dist[i] <- mean(outcome[treatment_p == "b"]) -
                          mean(outcome[treatment_p == "a"])
}
```

- The test statistic of each permutation sample is recorded at the ***i***-th position of “perm dist”.

## Generate the Histogram of all Test Statistics

- We use the "hist()" function to generate the histogram of the test statistics for the 10,000 permutation samples.

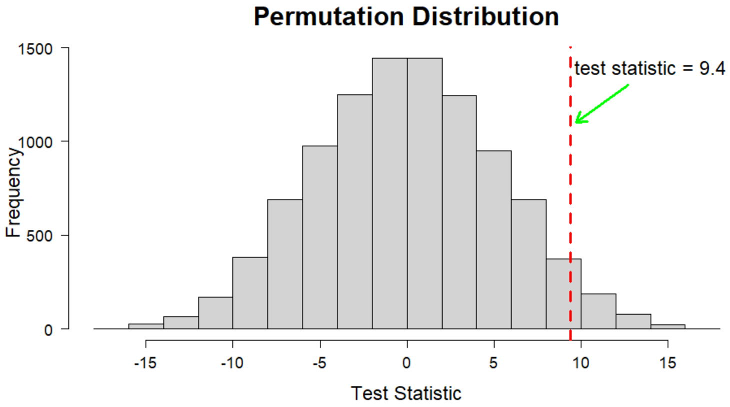
```
hist(perm_dist)
```



## Generate the Histogram of all Test Statistics

Cont'd

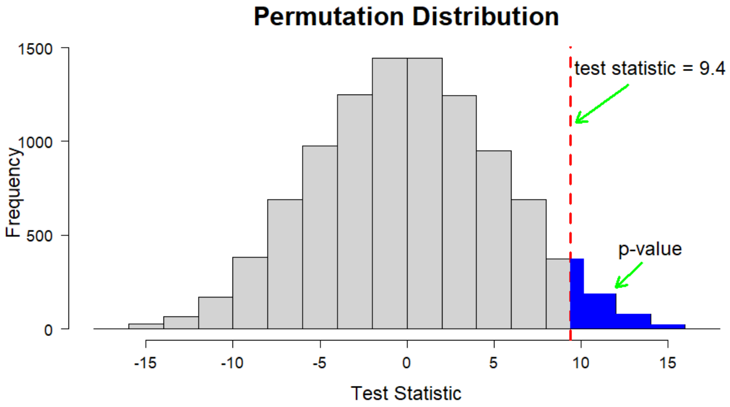
- Next, we mark the original sample statistic, 9.4.
- ***p-value*** is defined as the proportion of the permutation test statistics that are at least as extreme as the observed 9.4.



## Generate the Histogram of all Test Statistics

Cont'd

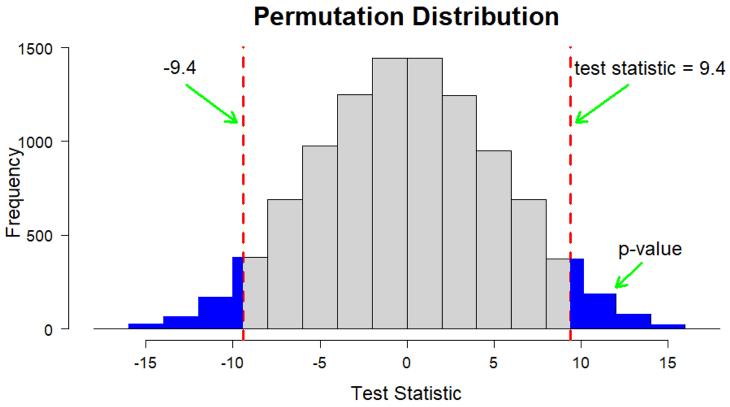
- Next, we mark the original sample statistic, 9.4.
- ***p-value*** is defined as the proportion of the permutation test statistics that are at least as extreme as the observed 9.4.



## Generate the Histogram of all Test Statistics

Cont'd

- Next, we mark the original sample statistic, 9.4.
- ***p-value*** is defined as the proportion of the permutation test statistics that are at least as extreme as the observed 9.4.



## Calculate p-value

- We can calculate the p-value as follows.

```
pvalue <- mean(abs(perm_dist) >= abs(original))
pvalue
```

```
[1] 0.0762
```

```
head(abs(perm_dist) >= abs(original))
```

```
[1] FALSE  TRUE  TRUE FALSE  TRUE FALSE
```

- Roughly 7.6% of the simulated permutation samples have more extreme values than the observed difference.
- As $0.076 > 0.05$, we do not reject the null hypothesis.

## Define a Function

- The syntax of defining a function is as follows:

```
function name <- function(variable1, variable2)
{
Actions
}
```

- We wrap the actions of one round of simulation inside the following function.

```
permutation.test <- function(treatment, outcome){
    # Generate a permutation sample
    treatment_p <- sample(treatment, size= length(treatment), replace
        =FALSE)
    # Calculate the test statistic for the permutation sample
   mean(outcome[treatment_p == "b"]) -
                              mean(outcome[treatment_p == "a"])
  }
```

- The function does two things: generate a permutation sample and calculate the test statistic.

## Use the “replicate()” Function to Run Multiple Simulations

- To run the function once is equivalent to one simulation.
- Next, we use the “replicate()” function to run the function multiple times.

```
test <- replicate(10000, permutation.test(treatment, outcome))
```

- The Replicate function needs two inputs:
  - The front number indicates the number of simulations to be run.
  - The latter part quotes the function that we designed for one simulation.
- The following codes generate the permutation distribution and the p-value, just like the earlier ones.

```
hist(test)
mean(abs(test) >= abs(original))
```

## Check Assumption

### Assumption of permutation tests

1. ***Exchangeability***: The observations are exchangeable.

- It means the labels of the dataset can be reordered, which does not affect the underlying joint distribution.
- The assumption is naturally satisfied in the following two cases:
  - (a) a randomised experiment;
  - (b) a random sample without replacement.
- As our example belongs to the former case, the assumption is verified.
- In general, the exchangeability assumption is much weaker than those of the classical hypothesis tests.

## Compare the p-values

- We tried the following code in the last video.

```
t.test(outcome~treatment, alternative = "two.sided", paired=F,
var.equal=T, data = df1)
```

```
Two Sample t-test
data:  outcome by treatment
t = -1.8767, df = 18, p-value = 0.07687
alternative hypothesis: true difference in means between group a and group b
is not equal to 0
95 percent confidence interval:
 -19.923268   1.123268
sample estimates:
mean in group a     mean in group b
           55.2                64.6
```

- In our case, the p-value of the permutation two sample test is very close to that of the classical two sample t-test.

## Summary

## Summary

**We have:**

- Explained the idea and the steps of conducting a permutation test.
- Learnt how to implement a permutation test in R.

**In the next video,**

We will use another case study to show how we deal with the more complex test statistic, and implement it in R.

## References

Mine Çetinkaya-Rundel and Johanna Hardin (2021)
Introduction to Modern Statistics