MONASH University

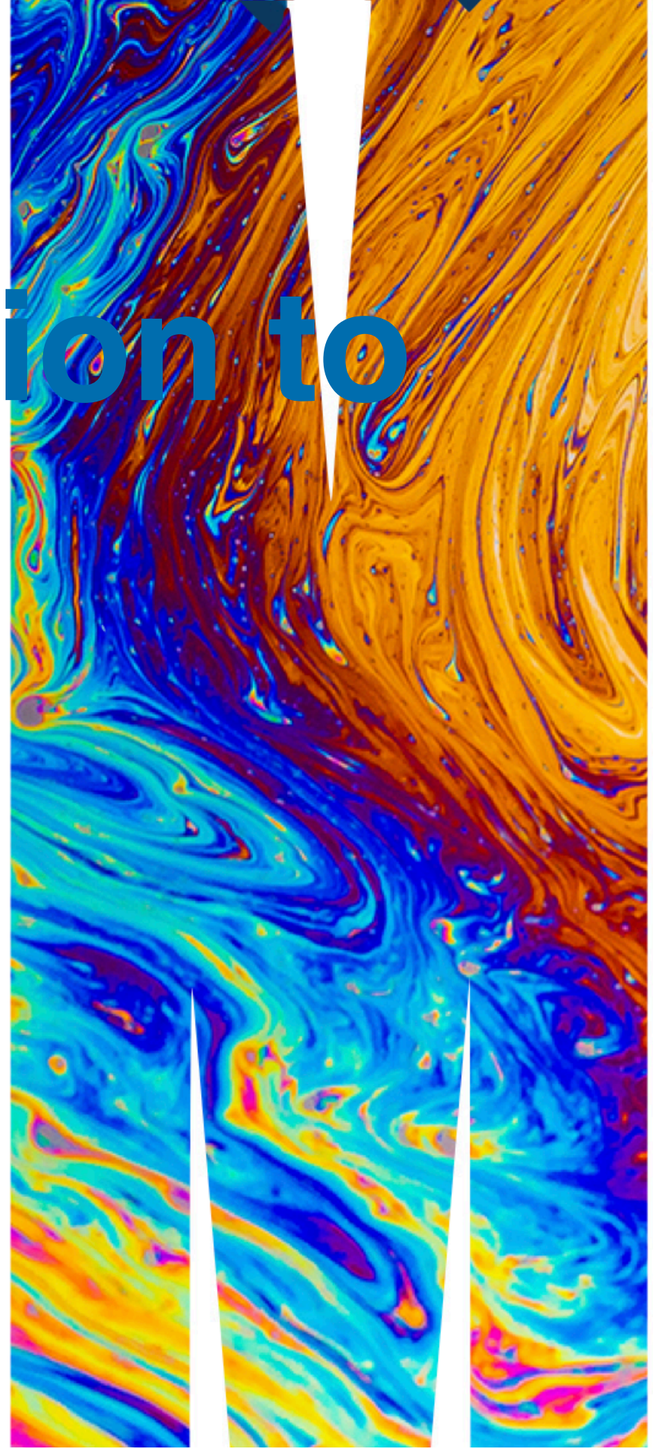
# ETC3250/5250 Introduction to Machine Learning

*Week 3: Re-sampling and regularisation*

Professor Di Cook

*etc3250.clayton-x@monash.edu*

*Department of Econometrics and Business Statistics*



# Overview

We will cover:

- Common re-sampling methods: **bootstrap**, **cross-validation**, **permutation**, **simulation**.
- **Cross-validation** for checking generalisability of model fit, parameter tuning, variable selection.
- **Bootstrapping** for understanding variance of parameter estimates.
- **Permutation** to understand significance of associations between variables, and variable importance.
- **Simulation** can be used to assess what might happen with samples from known distributions.
- What can go wrong in high-d, and how to adjust using **regularisation** methods.

# Model development and choice

# How do you get new data?

# Common re-sampling methods

- **Cross-validation**: Splitting the data into multiple samples.
- **Bootstrap**: Sampling with replacement
- **Permutation**: Re-order the values of one or more variables

- Cross-validation: This is used to gain some understanding of the variance (as in *bias-variance trade-off* ) of models, and how parameter or algorithm choices affect the performance of the model on future samples.

- Bootstrap: Compute confidence intervals for model parameters, or the model fit statistics. can be used similarly to cross-validation samples but avoids the complication of smaller sample size that may affect interpretation of cross-validation samples.
- Permutation: Used to assess significance of relationships, especially to assess the importance of individual variables or combinations of variables for a fitted model.

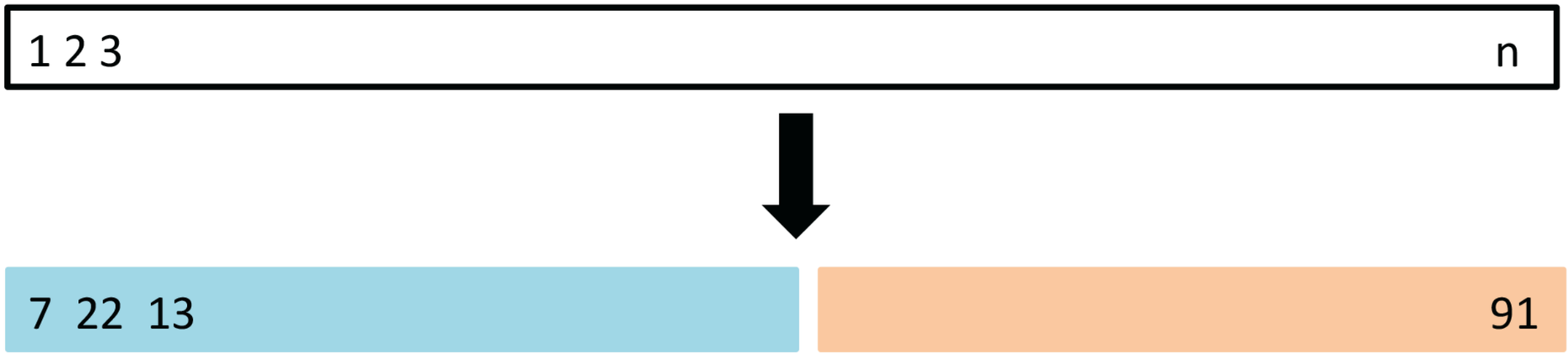
# Cross-validation

- Training/test split: make one split of your data, keeping one purely for assessing future performance.

After making that split, we would use these methods on the training sample:

- Leave-one-out: make $n$ splits, fitting multiple models and using left-out observation for assessing variability.
- $k$-fold: break data into $k$ subsets, fitting multiple models with one group left out each time.

# Training/test split (1/3)




A set of $n$ observations are randomly split into a training set (blue, containing observations 7, 22, 13, …) and a test set (yellow, all other observations not in training set).

- Need to stratify the sampling to ensure training and test groups are appropriately balanced.
- Only one split of data made, may have a lucky or unlucky split, accurately estimating test error relies on the one sample.

(Chapter5/5.1.pdf)

# Training/test split (2/3)

With tidymodels, the function `initial_split()` creates the indexes of observations to be allocated into training or test samples. To generate these samples use `training()` and `test()` functions.

```
d_bal <- tibble(y=c(rep("A", 6), rep("B", 6)),
                x=c(runif(12)))
d_bal$y
```

```
 [1] "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B"
```

```
set.seed(130)
d_bal_split <- initial_split(d_bal, prop = 0.7
training(d_bal_split)$y
```

```
[1] "A" "A" "B" "A" "B" "A" "B" "A"
```

```
testing(d_bal_split)$y
```

```
[1] "A" "B" "B" "B"
```

How do you ensure that you get 0.70 in each class?

## Stratify the sampling

```
d_bal$y
```

```
 [1] "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B"
```

```
set.seed(1225)
d_bal_split <- initial_split(d_bal,
                             prop = 0.70,
                             strata=y)
training(d_bal_split)$y
```

```
[1] "A" "A" "A" "A" "B" "B" "B" "B"
```

```
testing(d_bal_split)$y
```

```
[1] "A" "A" "B" "B"
```

Now the test set has 2 A's and 2 B'2. This is best practice!

# Training/test split (3/3)

Not stratifying can cause major problems with unbalanced samples.

```
d_unb <- tibble(y=c(rep("A", 2), rep("B", 10)),
                x=c(runif(12)))
d_unb$y
```

```
 [1] "A" "A" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B"
```

```
set.seed(132)
d_unb_split <- initial_split(d_unb, prop = 0.7
training(d_unb_split)$y
```

```
[1] "B" "B" "A" "B" "B" "A" "B" "B"
```

```
testing(d_unb_split)$y
```

```
[1] "B" "B" "B" "B"
```

The test set is missing one entire class!

> Always stratify splitting by sub-groups, especially response variable classes, and possibly other variables too.

```
d_unb_strata <- initial_split(d_unb,
                              prop = 0.70,
                              strata=y)
training(d_unb_strata)$y
```

```
[1] "A" "B" "B" "B" "B" "B" "B" "B"
```

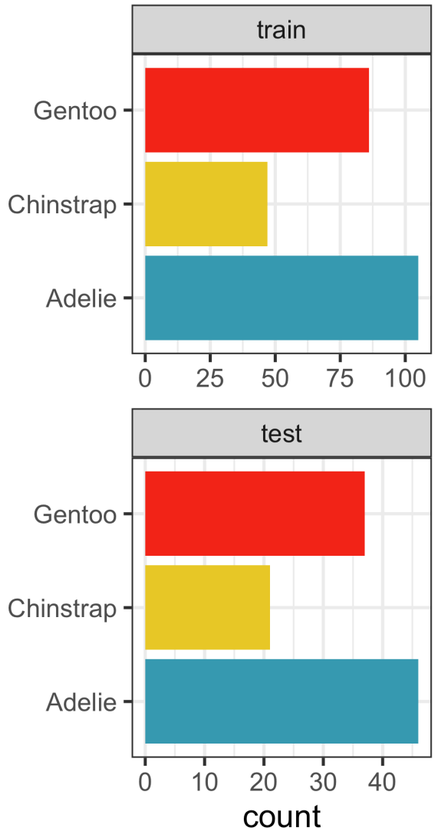
```
testing(d_unb_strata)$y
```

```
[1] "A" "B" "B" "B"
```

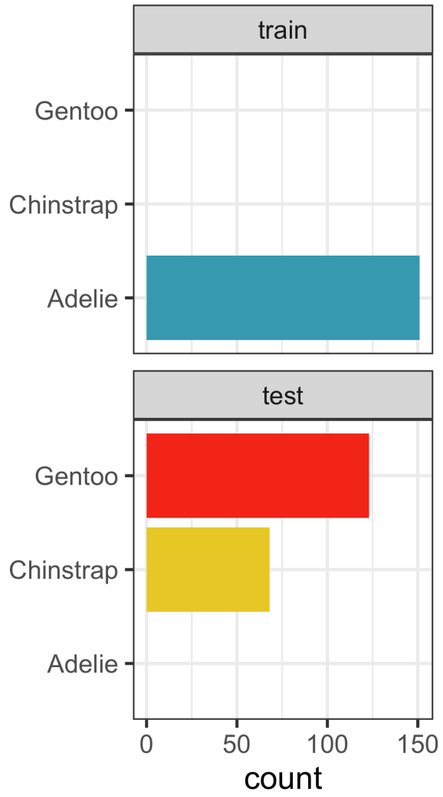
Now there is an A in the test set!

# Checking the training/test split: response

GOOD




BAD

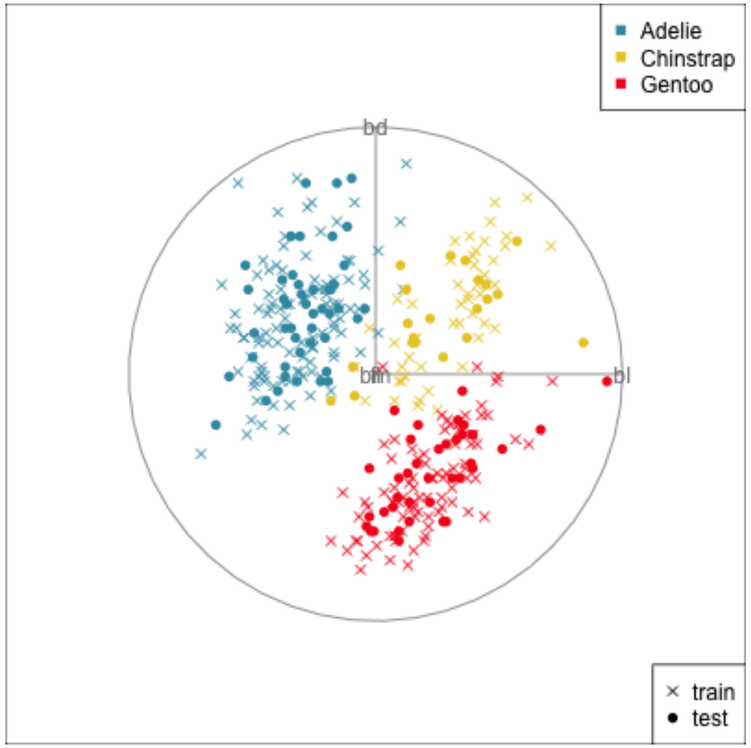



Check the class proportions of the response by computing counts and proportions in each class, and tabulating or plotting the result.

It's good if there are similar numbers of each class in both sets.

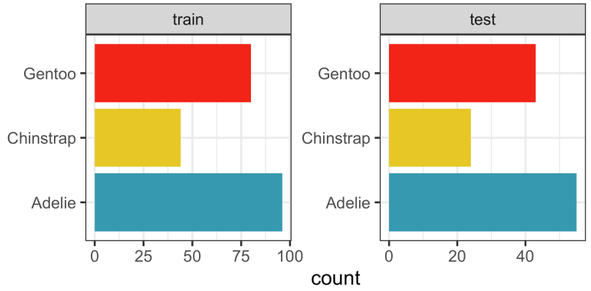
# Checking the training/test split: predictors

## GOOD



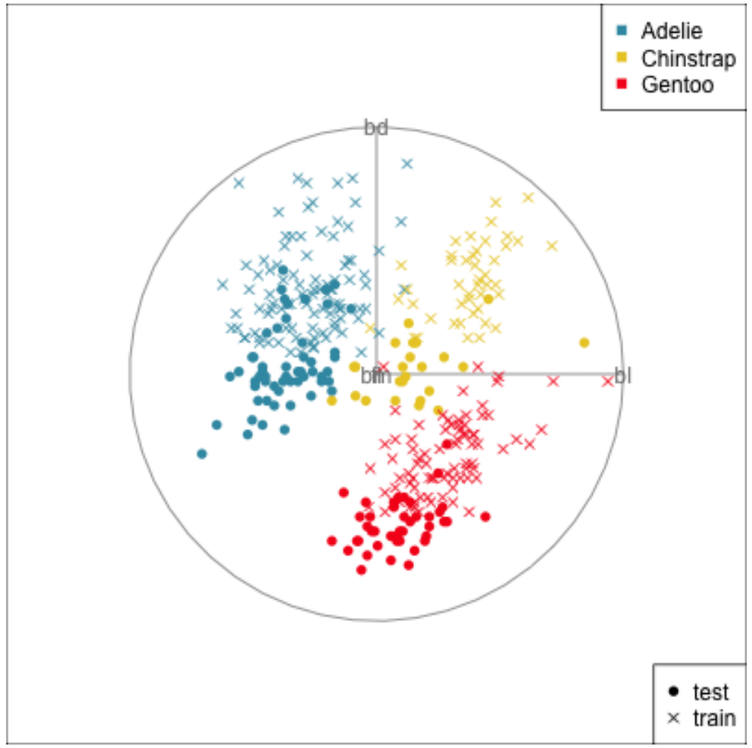

Make a training/test variable and plot the predictors. Need to have similar distributions.

## Looks good




On the response training and test sets have similar proportions of each class so looks good BUT it's not

## But BAD




Test set has smaller penguins on at least two of the variables.

# Cross-validation

# k-fold cross validation (1/4)

1. Divide the data set into $k$ different parts.
2. Remove one part, fit the model on the remaining $k-1$ parts, and compute the **statistic of interest** on the omitted part.
3. Repeat $k$ times taking out a different part each time

# k-fold cross validation (2/4)

1. Divide the data set into $k$ different parts.
2. Remove one part, fit the model on the remaining $k-1$ parts, and compute the **statistic of interest** on the omitted part.
3. Repeat $k$ times taking out a different part each time

Here are the row numbers for $k = 5$ folds:

```
> p_folds <- vfold_cv(p_sub, 5, strata=species)
> c(1:nrow(p_sub))[-p_folds$splits[[1]]$in_id]
 [1]  5  6  8 12 16 23 28 31 43 44 45 53 57 58 70 73 74 77
> c(1:nrow(p_sub))[-p_folds$splits[[2]]$in_id]
 [1]  2  9 10 11 13 17 22 25 39 48 50 51 55 61 65 69 75 78
> c(1:nrow(p_sub))[-p_folds$splits[[3]]$in_id]
 [1]  1  3 14 18 20 26 33 41 42 49 56 67 72 81 83 84
> c(1:nrow(p_sub))[-p_folds$splits[[4]]$in_id]
 [1]  4 19 29 32 34 35 36 40 46 52 63 64 66 76 79 80
> c(1:nrow(p_sub))[-p_folds$splits[[5]]$in_id]
 [1]  7 15 21 24 27 30 37 38 47 54 59 60 62 68 71 82
```

# k-fold cross validation (3/4)

1. Divide the data set into $k$ different parts.
2. Remove one part, fit the model on the remaining $k-1$ parts, and compute the **statistic of interest** on the omitted part.
3. Repeat $k$ times taking out a different part each time

Fit the model to the $k-1$ set, and compute the statistic on the $k$-fold, that was not used in the model fit.

Here we use the **accuracy** as the statistic of interest.

Value for fold 1 is:

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy multiclass     0.889
```

# k-fold cross validation (4/4)

1. Divide the data set into $k$ different parts.
2. Remove one part, fit the model on the remaining $k-1$ parts, and compute the **statistic of interest** on the omitted part.
3. Repeat $k$ times taking out a different part each time

Here is the **accuracy** computed for each of the $k = 5$ folds. Remember, this means that the model was fitted to the rest of the data, and accuracy was calculate on the observations in this fold.

```
[1] 0.89 0.89 1.00 0.88 1.00
```

Recommended reading: Alison Hill's Take a Sad Script & Make it Better: Tidymodels Edition

# LOOCV

> Leave-one-out (LOOCV) is a special case of $k$-fold cross-validation, where $k = n$.
> There are $n$ CV sets, each with ONE observation left out.

Benefits:

- Useful when sample size is very small.
- Some statistics can be calculated algebraically, without having to do computation for each fold.

# Where is cross-validation used?

- Model evaluation and selection, by estimating the generalisability on future data.
- Parameter tuning: finding optimal choice of parameters or control variables, like number of trees or branches, or polynomial terms to generate the best model fit.
- Variable selection: which variables are more or less important for the best model fit. Possibly some variables can be dropped from the model.

# Bootstrap

# Bootstrap (1/5)

A bootstrap sample is a sample that is the same size as the original data set that is made using replacement. This results in analysis samples that have multiple replicates of some of the original rows of the data. The assessment set is defined as the rows of the original data that were not included in the bootstrap sample, referred to as the **out-of-bag** (OOB) sample.

```
set.seed(115)
df <- tibble(id = 1:26,
             cl = c(rep("A", 12), rep("B", 14)))
df_b <- bootstraps(df, times = 100, strata = cl)
t(df_b$splits[[1]]$data[df_b$splits[[1]]$in_id,])
```

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
id " 1" " 2" " 2" " 2" " 2" " 5" " 6" " 7" " 9" "11"  "12"
cl "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"   "A"
   [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
id "12"  "14"  "14"  "18"  "18"  "18"  "18"  "18"  "19"
cl "A"   "B"   "B"   "B"   "B"   "B"   "B"   "B"   "B"
   [,21] [,22] [,23] [,24] [,25] [,26]
id "21"  "21"  "21"  "22"  "25"  "25"
cl "B"   "B"   "B"   "B"   "B"   "B"
```

Which observations are out-of-bag in bootstrap sample 1?

# Bootstrap (2/5)

- Bootstrap is preferable to cross-validation when the sample size is small, or if the structure in the data being modelled is complex.
- It is commonly used for estimating the variance of parameter estimates, especially when the data is non-normal.

# Bootstrap (3/5)

In dimension reduction it can be used to assess if the coefficients of a PC (the eigenvectors) are significantly different from ZERO. The 95% bootstrap confidence intervals can be computed by:

1. Generating B bootstrap samples of the data
2. Compute PCA, record the loadings
3. Re-orient the loadings, by choosing one variable with large coefficient to be the direction base
4. If B=1000, 25th and 975th sorted values yields the lower and upper bounds for confidence interval for each PC.

# Bootstrap (4/5)

Assessing the loadings for PC 2 of PCA on the womens track data. Remember the summary:

```
Standard deviations (1, .., p=7):
[1] 2.41 0.81 0.55 0.35 0.23 0.20 0.15

Rotation (n x k) = (7 x 7):
          PC1   PC2    PC3    PC4    PC5     PC6    PC7
m100     0.37  0.49 -0.286  0.319  0.231  0.6198  0.052
m200     0.37  0.54 -0.230 -0.083  0.041 -0.7108 -0.109
m400     0.38  0.25  0.515 -0.347 -0.572  0.1909  0.208
m800     0.38 -0.16  0.585 -0.042  0.620 -0.0191 -0.315
m1500    0.39 -0.36  0.013  0.430  0.030 -0.2312  0.693
m3000    0.39 -0.35 -0.153  0.363 -0.463  0.0093 -0.598
marathon 0.37 -0.37 -0.484 -0.672  0.131  0.1423  0.070
```

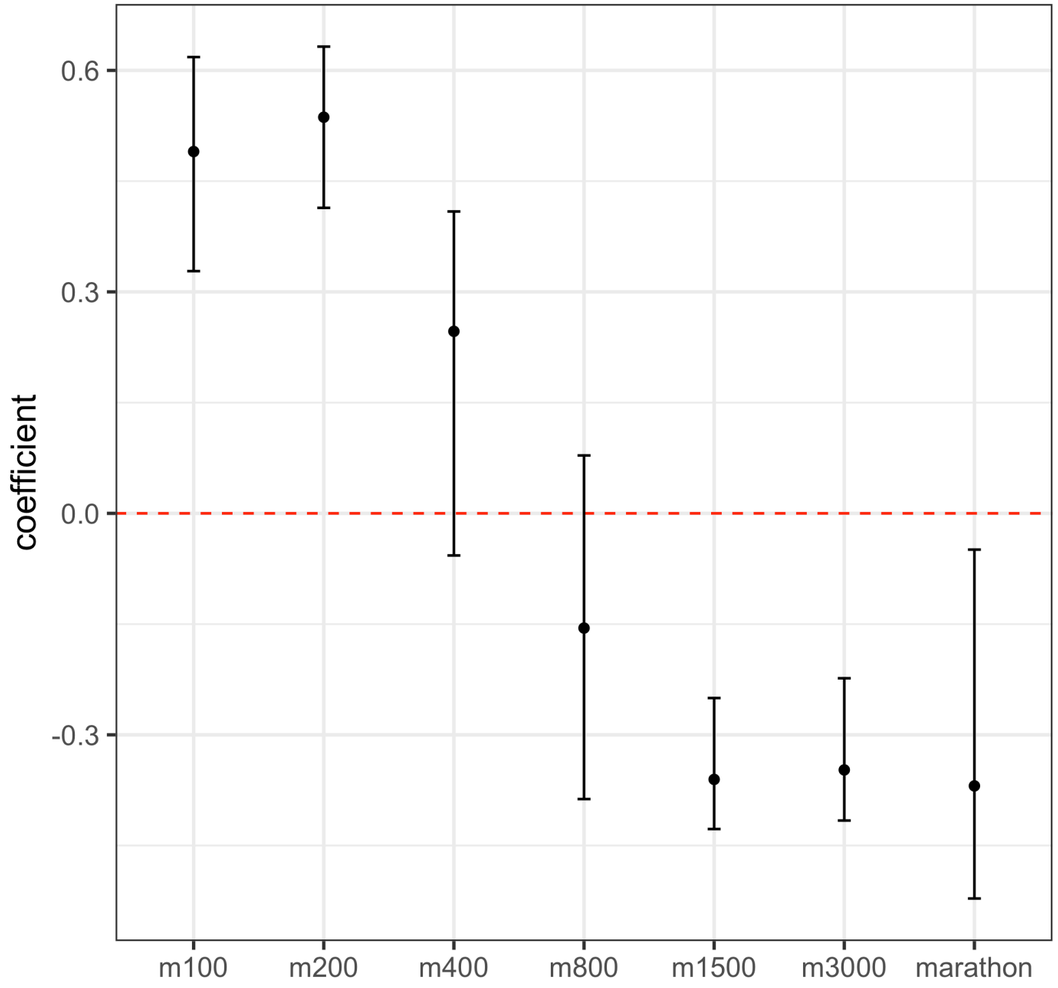
Should we consider `m800`, `m400` contributing to PC2 or not?

# Bootstrap (5/5)

We said that *PC2 is a contrast between short distance events and long distance events, particularly 100m, 200m vs 1500m, 3000m, marathon*. How reliably can we state this?

▶ Code

Confidence intervals for `m400` and `m800` cross ZERO, hence zero is a plausible value for the population coefficient corresponding to this estimate.



# Permutation

# Permutation (1/3)

Permutation breaks relationships, and is often used for conducting statistical hypothesis tests, without requiring too many assumptions.

Is there a difference in the medians of the groups?

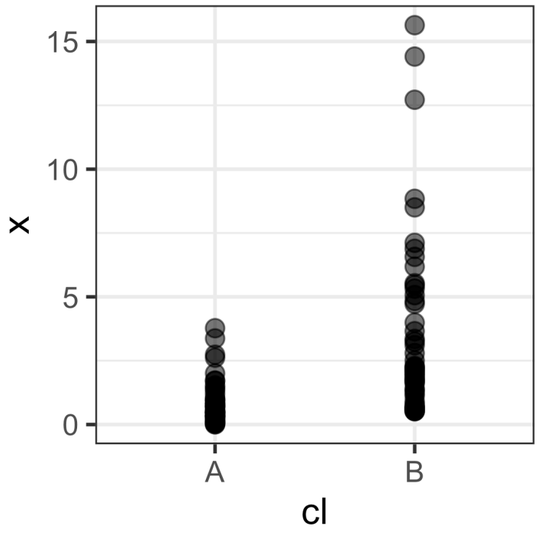
DATA

```
# A tibble: 10 × 2
       x cl
   <dbl> <chr>
 1 0.281 A
 2 0.330 A
 3 0.708 A
 4 0.463 A
 5 3.37  A
 6 0.528 B
 7 0.852 B
 8 5.58  B
 9 0.685 B
10 3.28  B
```

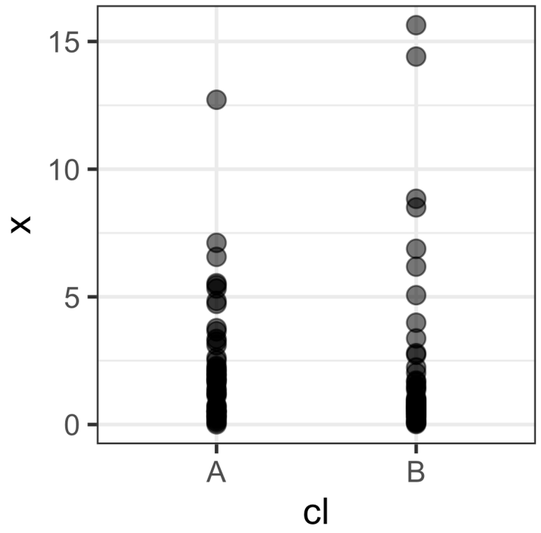
PERMUTE `cl`

```
# A tibble: 10 × 2
       x cl
   <dbl> <chr>
 1 0.281 A
 2 0.330 B
 3 0.708 A
 4 0.463 B
 5 3.37  B
 6 0.528 A
 7 0.852 B
 8 5.58  B
 9 0.685 A
10 3.28  A
```

DATA



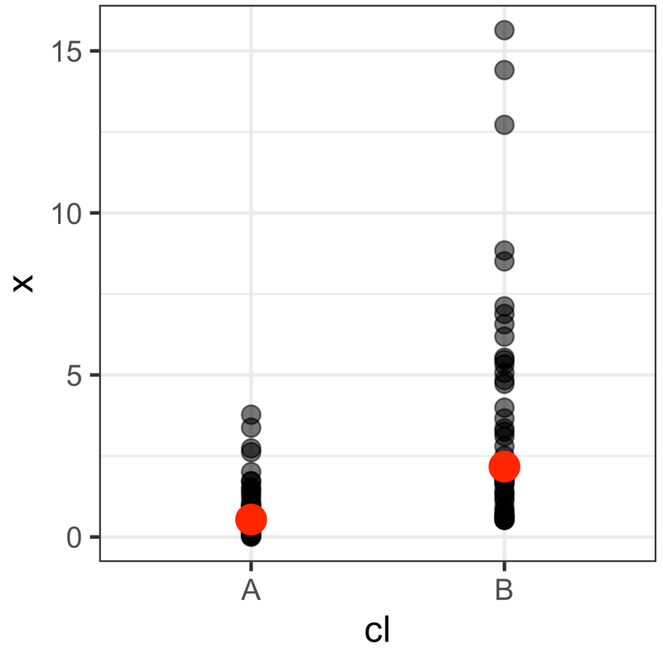

PERMUTED



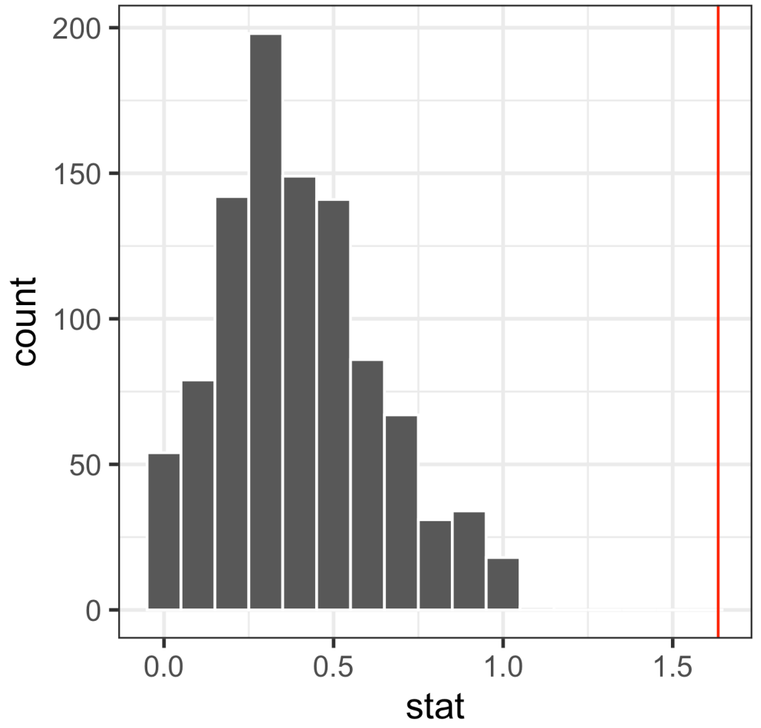
# Permutation (2/3)

Is there a difference in the medians of the groups?

DATA




Generate $k$ permutation samples, compute the medians for each, and compare the difference with original.



# Permutation (3/3)

- Caution: permuting small numbers, especially classes may return very similar samples to the original data.
- Stay tuned for random forest models, where permutation is used to help assess the importance of all the variables.

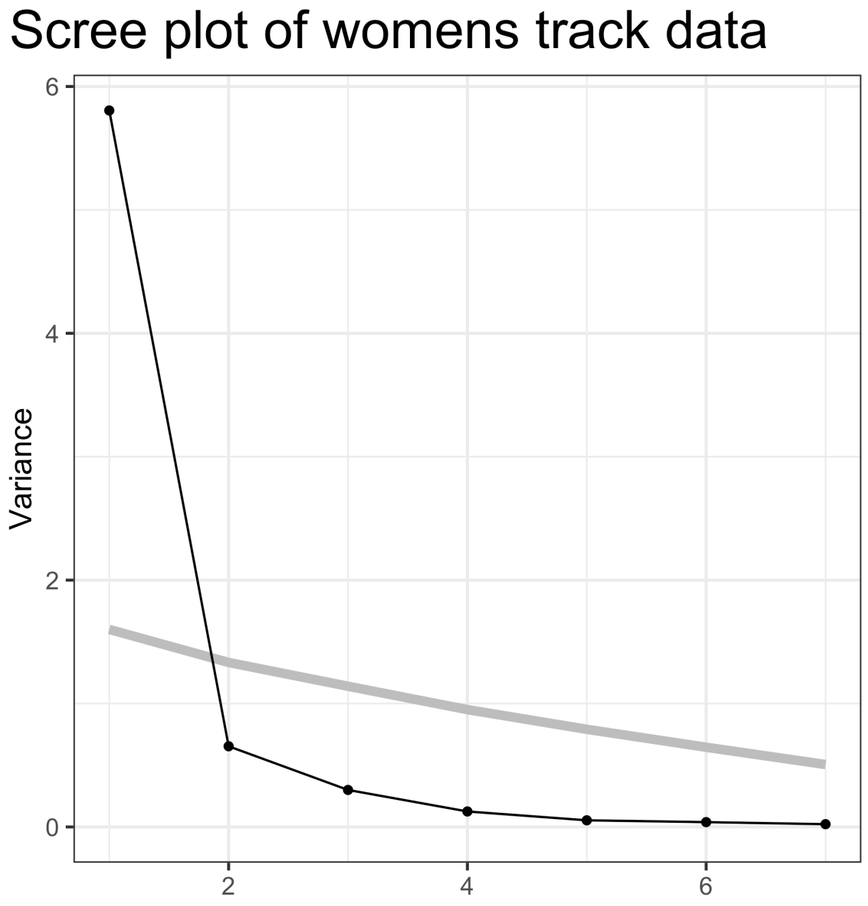
# Simulation

# Simulation (1/2)

Simulation from known statistical distributions allows us to check data and calculations against what is known is controlled conditions.

For example, how likely is it to see the extreme a value if my data is a sample from a normal distribution?

# Simulation (2/2)




Grey line is a guide line, computed by doing PCA on 100 samples from a standard $p$-dimensional normal distribution.

That is a comparison of the correlation matrix of the track data with a correlation matrix that is the identity matrix, where there is no association between variables.

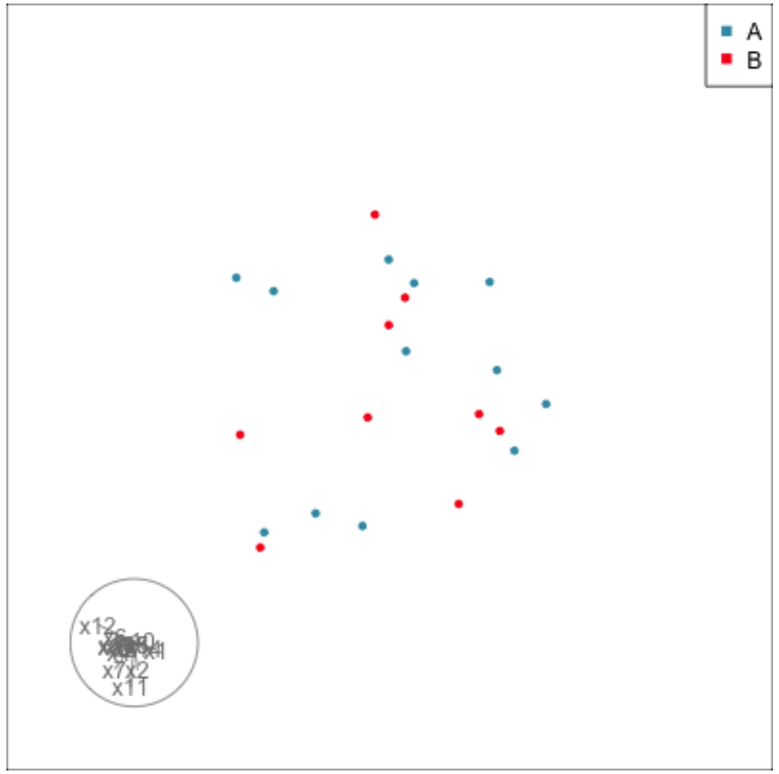
The largest variance we expect is under 2. The observed variance for PC 1 is much higher. Much larger than expected, very important for capturing the variability in the data!

Why is there a difference in variance, when there is no difference in variance?

# What can go wrong in high-dimensions

# Space is huge!

```
set.seed(357)
my_sparse_data <- tibble(cl = c(rep("A", 12),
                                rep("B", 9)),
                         x1 = rnorm(21),
                         x2 = rnorm(21),
                         x3 = rnorm(21),
                         x4 = rnorm(21),
                         x5 = rnorm(21),
                         x6 = rnorm(21),
                         x7 = rnorm(21),
                         x8 = rnorm(21),
                         x9 = rnorm(21),
                         x10 = rnorm(21),
                         x11 = rnorm(21),
                         x12 = rnorm(21),
                         x13 = rnorm(21),
                         x14 = rnorm(21),
                         x15 = rnorm(21)) |>
```




Do we agree that there is no REAL difference between A and B?

Difference is due to having insufficient data with too many variables.

# Regularisation

The fitting criteria has an added penalty term with the effect being that some parameter estimates are forced to ZERO. This effectively reduces the dimensionality by removing noise, and variability in the sample that is consistent with what would be expected if it was purely noise.

Stay tuned for examples in various methods!

# Next: Logistic regression and discriminant analysis