INTRO TO RESAMPLING AND SELECTION METHODS

And looking ahead…

TODAY

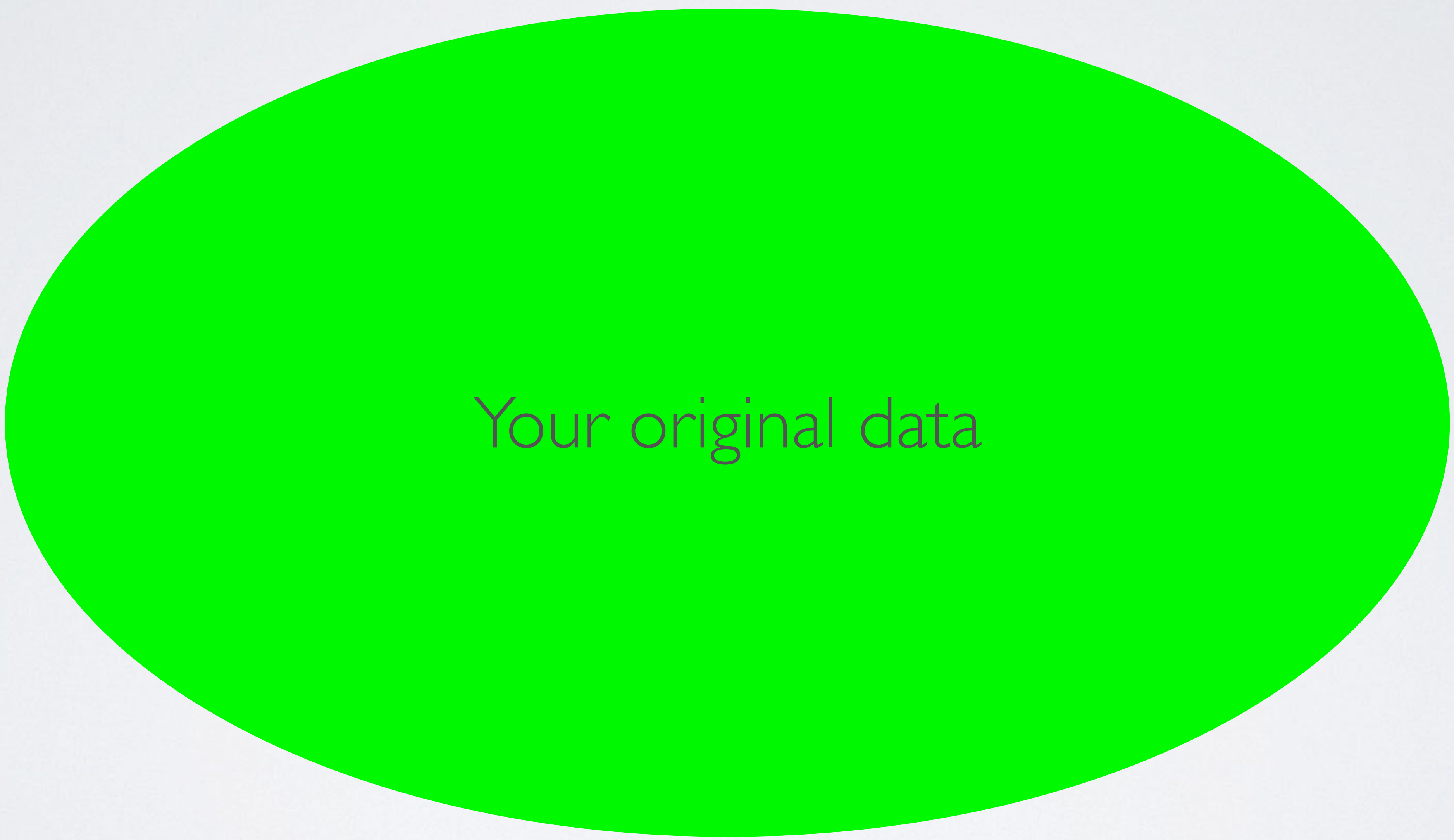
- Cross-validation of models
- Bootstrapping
- Variable selection methods (brief)
- Q&A on case study

PACKAGES NEEDED

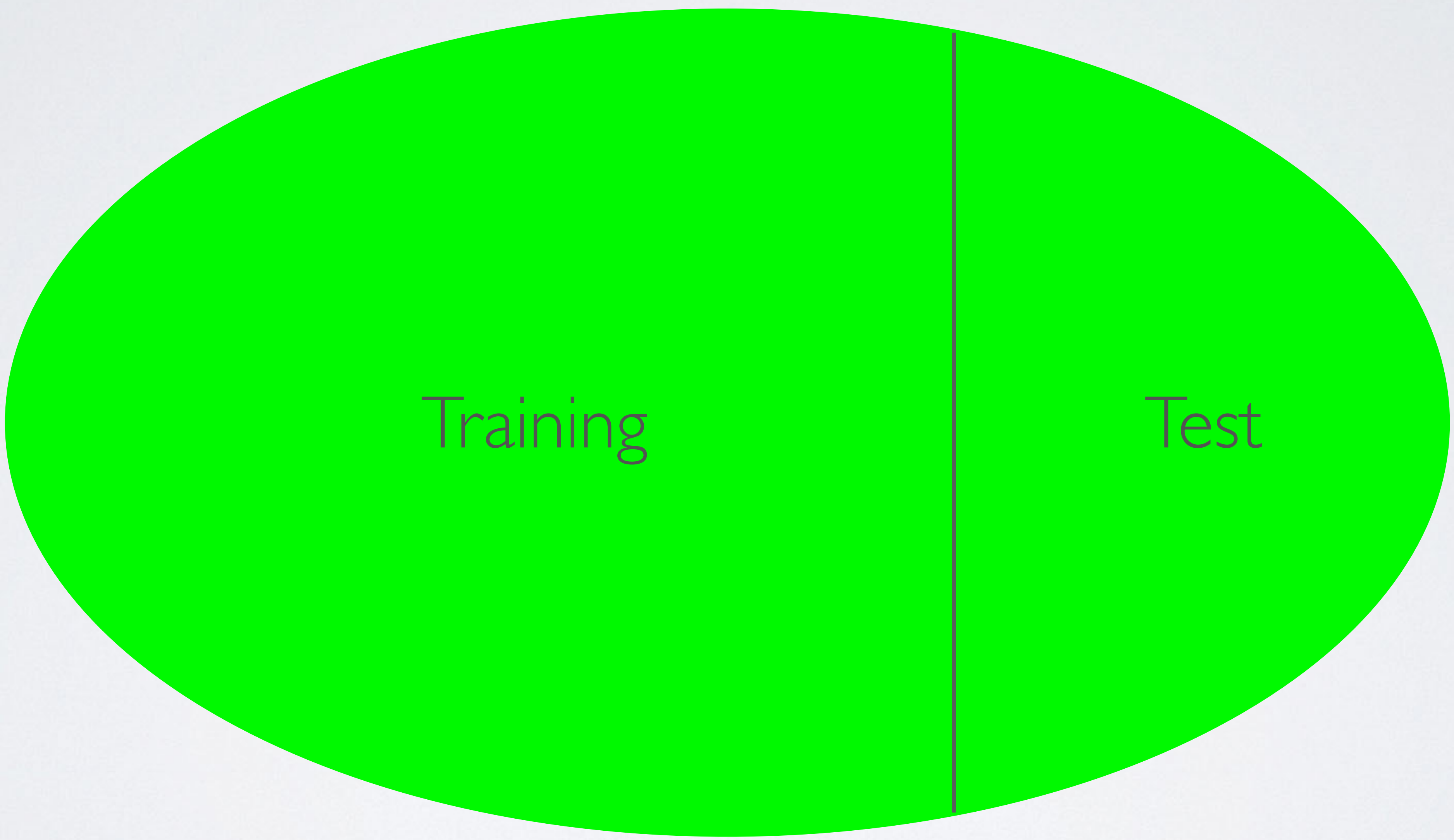
- `Tidyverse`
- `readxl`
- `boot`
- `leaps`

CROSS-VALIDATION

BASIC IDEA



BASIC IDEA



POSSIBLE APPROACHES

- Validation set
- Leave one out
- k-fold

VALIDATION SET

- Split original data in half (training and test)
- Decide on models using training
- Run on test and check

```r
# Data ----

gss <- read_excel(here("raw_data", "GSS2008.xls"))
# Quick and dirty way to get lower case names
# Could make this one line
var.names <- tolower(colnames(gss))
colnames(gss) <- var.names

gss_income <- gss %>%
  mutate(
    female = sex == 2,
    inc_1000 = income / 1000
  ) %>%
  filter(
    educ != 0,
    age <= 65
  ) %>%
  filter(
    !is.na(income),
    !is.na(hrs)
  )
```

SPLITTING DATA

```r
set.seed(2)
train <- sample(956, 478)
```

RUN MODELS

```
val_model_1 <- lm(log(income) ~ female + educ + age + hrs,
                  data = gss_income, subset = train)
summary(val_model_1)

val_model_2 <- lm(log(income) ~ female + educ + age + I(age^2) + hrs,
                  data = gss_income, subset = train)
summary(val_model_2)

val_model_3 <- lm(log(income) ~ female + educ + log(age) + hrs,
                  data = gss_income, subset = train)
summary(val_model_3)
```

FIND MSE

```
mean((log(gss_income$income) -
     predict(val_model_1, gss_income))[-train]^2)

mean((log(gss_income$income) -
     predict(val_model_2, gss_income))[-train]^2)

mean((log(gss_income$income) -
     predict(val_model_3, gss_income))[-train]^2)
```

ASIDE ON FUNCTIONS

- "Rule": Never copy/paste more than once!
- Write function
- Easier to read
- Easier to edit

HOW?

```
function_name <- function(input1, input2) {
      regular R code
}
```

```r
# Time for a function
mse <- function(model_name) {
  mean((log(gss_income$income) -
          predict(model_name, gss_income))[-train]^2)
}
```

No need for explicit return statement

PRETTIER, BUT STILL CLUNKY

```
mse(val_model_1)
mse(val_model_2)
mse(val_model_3)
```

```
> mse(val_model_1)
[1] 0.8088536
> mse(val_model_2)
[1] 0.7738513
> mse(val_model_3)
[1] 0.7950036
```

FANCY LOOPING

Rather than looping R offers:

`apply`
`lapply`
`sapply`
`vapply`

If you have more than one input:

`mapply`
`Map`

Faster speed: Parallelisation

THE SAPPLY WAY

```r
models <- list(val_model_1, val_model_2, val_model_3)
sapply(models, mse)
```

```
> models <- list(val_model_1, val_model_2, val_model_3)
> sapply(models, mse)
[1] 0.8088536 0.7738513 0.7950036
```

LOOCV

- Problem with validation set approach:
  - High variance in estimate of MSE (try different seeds to see)
  - Overestimate of error rate
- Alternative:
  - Only leave one obs out
  - Run n times

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

PROS AND CONS

- Pros:
  - Lower bias
  - Always same results
- Cons:
  - Computational intensive

No need to code this; built into `glm`

```r
loocv_model_1 <- glm(log(income) ~ female + educ + age + hrs,
                     data = gss_income)
summary(loocv_model_1)

loocv_model_2 <- glm(log(income) ~ female + educ + age + I(age^2) + hrs,
                     data = gss_income)
summary(loocv_model_2)

loocv_model_3 <- glm(log(income) ~ female + educ + log(age) + hrs,
                     data = gss_income)
summary(loocv_model_3)
```

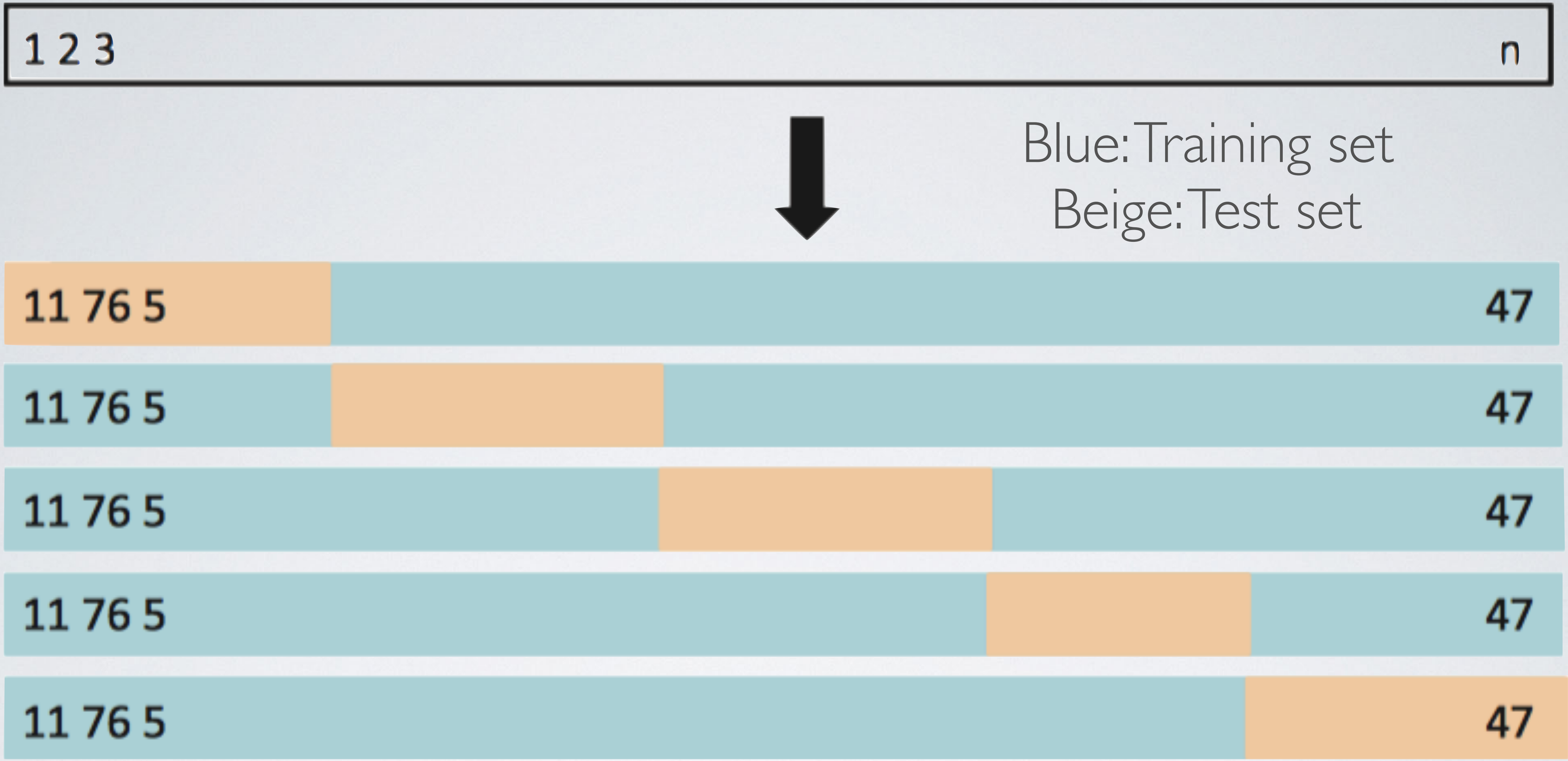
ANOTHER SAPPLY

```
models <- list(loocv_model_1, loocv_model_2, loocv_model_3)
sapply(models, function(x) cv.glm(gss_income, x)$delta[1])
```

```
> models <- list(loocv_model_1, loocv_model_2, loocv_model_3)
> sapply(models, function(x) cv.glm(gss_income, x)$delta[1])
[1] 0.7582453 0.7226206 0.7445689
```

K-FOLD CV

- If LOOCV too computational intensive:
- Combine CV and LOOCV!
- Make k test sets and use rest as training



$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{n} MSE_i$$

DONE IN ONE LINE

```r
# k-fold CV ----
sapply(models, function(x) cv.glm(gss_income, x, K = 10)$delta[1])
```

```
> sapply(models, function(x) cv.glm(gss_income, x, K = 10)$delta[1])
[1] 0.7609286 0.7209373 0.7459189
```

BOOTSTRAPPING

BASIC IDEA

- Treat your data as representing the true data
- Repeated resample (w/ replacement) and estimate
- Calculate statistics based on estimates

WHEN AND WHERE?

- Anytime and everywhere!!
- Most sense: No closed form solution for statistic
- Example: Elasticity of children with respect to education

FIRST FUNCTION

```r
# function for calculating elasticity
calc_elasticity <- function(model_name, var_name, level) {
  model_name$coefficients[[var_name]] / level
  # can also write coef(model_name)[[var_name]] / level
}

calc_elasticity(elas_mod_2, "log(educ)", 12)
```

```r
gss_kids <- gss_kids %>%
  mutate(
    log_educ = log(educ)
  )
```

SECOND FUNCTION

```r
# combine into one function and check
run_elasticity <- function(df, var_name, level) {
  # Run regression
  m <- lm(childs ~ df[[var_name]], data = df)

  # Calculate and return elasticity
  coef(m)[["df[[var_name]]"]] / level
}

run_elasticity(gss_kids, "log_educ", 12)
```

FUNCTION FOR BOOTSTRAP

```
# bootstrap function for elasticity
boot_elasticity <- function(df, i, var_name, level) {
  # Select obs
  df <- df[i, ]

  # Run regression
  m <- lm(childs ~ df[[var_name]], data = df)

  # Calculate and return elasticity
  coef(m)[["df[[var_name]]"]] / level
}
```

RUN BOOTSTRAP

```
set.seed(2)
boot(gss_kids, boot_elasticity, R = 1000,
     var_name = "log_educ", level = 12)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = gss_kids, statistic = boot_elasticity, R = 1000,
    var_name = "log_educ", level = 12)


Bootstrap Statistics :
      original        bias    std. error
t1* -0.1487021 -0.002518578  0.03034937
```

"OBSERVED" CI

```r
set.seed(2)
boot_results <- boot(gss_kids, boot_elasticity, R = 1000,
     var_name = "log_educ", level = 12)

boot.ci(boot_results, conf = c(0.99, 0.95, 0.9), type = "norm")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_results, conf = c(0.99, 0.95, 0.9), type = "norm")

Intervals :
Level      Normal
99%   (-0.2225, -0.0680 )
95%   (-0.2041, -0.0864 )
90%   (-0.1946, -0.0959 )
Calculations and Intervals on Original Scale
```

VARIABLE SELECTION

METHODS

- Stepwise (forward / backward)
- Shrinkage
- Dimension reduction ($p > n$)

STEPWISE

- Best if:
  - Not testing theory
  - Large number of variables
- No guarantee of same model on new data

LASSO/RIDGE

- Shrink coefficient towards zero
- Why? Reduce coefficient variance
- Lasso: Allow some coefficients to be zero

LOOKING AHEAD

COMMENTS ON CLASS?

- What worked?
- What didn't?
- Suggestions for next time

Q&A ON CASE STUDY