Summary Statistics

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- Use the Central Limit Theorem to construct confidence intervals for means
- Manipulate data using `tidyverse` functions

Motivation

- It’s one of the great paradoxes of statistics:
    - To better understand data, summarise it.
- The mean/average occupies a special place, because of its nice properties.
- But looking at multiple summaries gives us a fuller picture.

Mean

- Recall the definition: if we have n observations $X_1, \dots, X_n$, their **mean** (or average) is their sum divided by n:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

- The mean is a measure of *central tendency*, i.e. where the bulk of the observations tend to fall.
- In R, we can compute the mean using the function `mean`.

Examples i

- We'll use the dataset `mtcars`, which comes with `R` by default.
- Datasets are usually stored in `data.frame`s.
    - We can inspect `data.frame`s using `str` or `head`/`tail`.

```
str(mtcars)
```

```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3
24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
```

Examples ii

```
## $ hp : num 110 110 93 110 175 105 245 62 95
123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21
3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

Examples iii

- There are many ways to compute the average miles per gallon (`mpg`) for this dataset. I will demonstrate two ways.
    - Extract column
    - Use `summarise`

```r
# Extract column with $ and use mean function
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

Examples iv

```r
# Use summarise
library(tidyverse)
summarise(mtcars, mean(mpg))
```

```
##   mean(mpg)
## 1  20.09062
```

- In the second approach, we use the function `summarise`. The first argument is the `data.frame`; the second argument is the summary statistic we want to compute. One advantage is that we can compute many summaries in one function call.

Examples v

```r
# Both mean and standard deviation
summarise(mtcars, mean(mpg), sd(mpg))
```

```
##   mean(mpg)  sd(mpg)
## 1  20.09062 6.026948
```

```r
# Average mpg and average qsec
summarise(mtcars, mean(mpg), mean(qsec))
```

```
##   mean(mpg) mean(qsec)
## 1  20.09062   17.84875
```

Central Limit Theorem i

- The sample mean gives us some idea about the *population* mean.
  - "What if we could measure the height of all Canadians, instead of a sample?"
- But how certain can we be that the mean of our data is close to the true population mean?
- The Central Limit Theorem tells us that, whatever the distribution of the data, its *sample mean* behaves like a normal random variable.

Central Limit Theorem ii

- More precisely, if we have n independent observations that come from a distribution with mean μ and variance σ^2, then the sample mean is approximately normal:

$$\bar{X} \approx N(\mu, \sigma^2/n).$$

- The most important consequence: we can construct confidence intervals for the *population* mean.
  - For 95% CI: $\bar{X} \pm 1.96\hat{\sigma}/\sqrt{n}$, where $\hat{\sigma}$ is the *standard deviation* of the data (use the function `sd`).

Central Limit Theorem iii

- Important observations:
    - As we collect more data, the standard deviation of the data can go up or down. On the other hand, the confidence interval will become narrower and narrower.
    - In practice, we don't really know if our data is independent, or if it all comes from the same distribution. This uncertainty has to be reflected in the strength of our conclusions about the data.

Examples i

```r
# Recall
summarise(mtcars, mean(mpg), sd(mpg))
```

```
##   mean(mpg)  sd(mpg)
## 1  20.09062 6.026948
```

```r
# 95% Confidence interval
c(20.09062 - 1.96*6.026948/sqrt(32),
  20.09062 + 1.96*6.026948/sqrt(32))
```

```
## [1] 18.00239 22.17885
```

Examples ii

```r
# Alternative: save values in variables
# and use variables
mean_mpg <- mean(mtcars$mpg)
sd_mpg <- sd(mtcars$mpg)
n <- nrow(mtcars)

c(mean_mpg - 1.96*sd_mpg/sqrt(n),
  mean_mpg + 1.96*sd_mpg/sqrt(n))
```

```
## [1] 18.00239 22.17886
```

Exercise

Compute the *average and standard deviation* for `qsec`, which is the quarter-mile time (i.e. the time it takes the car to travel a quarter mile starting from a standstill).

Compute a 95% confidence interval for the average quarter-mile time.

Solution

```r
mean_qsec <- mean(mtcars$qsec)
sd_qsec <- sd(mtcars$qsec)
n <- nrow(mtcars)
mean_qsec
```

```
## [1] 17.84875
```

```r
sd_qsec
```

```
## [1] 1.786943
```

```r
c(mean_qsec - 1.96*sd_qsec/sqrt(n),
  mean_qsec + 1.96*sd_qsec/sqrt(n))
```

```
## [1] 17.22961 18.46789
```

Transforming your data

- Sometimes, you want to look at a subset of the data. Or perhaps you want to compute the mean of another variable, not defined in your dataset.
    - In other words, we need to transform the data first!
- All `tidyverse` functions take a `data.frame` as the first argument.
- A `data.frame` is a collection of vectors, all of the same length, but could be of different types.
    - This is the main way of organizing data in R.

Main tidyverse functions i

- `mutate`: Create a new variable as a function of the other variables

```r
# Switch to litres per 100km
mutate(mtcars, litres_per_100km = 235.215/mpg)
```

- `filter`: Keep only rows for which some condition is `TRUE`

```r
# Only keep rows where cyl is equal to 6 or 8
filter(mtcars, cyl %in% c(6, 8))
```

Examples i

- Let’s say we want to compute a 95% confidence interval for litres per 100km.

```
data1 <- mutate(mtcars, litres_per_100km = 235.215/mpg)
data2 <- summarise(data1,
                   avg_lit = mean(litres_per_100km),
                   sd_lit = sd(litres_per_100km))
data2
```

```
##    avg_lit   sd_lit
## 1 12.75506 3.863251
```

Examples ii

```r
data3 <- mutate(data2,
                low_bd = avg_lit - 1.96*sd_lit/sqrt(n),
                up_bd = avg_lit + 1.96*sd_lit/sqrt(n))
data3
```

```
##   avg_lit   sd_lit   low_bd    up_bd
## 1 12.75506 3.863251 11.41651 14.09361
```

Pipe operator

- One of the important features of the `tidyverse` is the pipe operator `%>%`
- It takes the output of a function (or of an expression) and uses it as input for the next function (or expression)

```r
library(tidyverse)

count(mtcars, cyl)
```

```
##   cyl  n
## 1   4 11
## 2   6  7
## 3   8 14
```

```r
# Or with the pipe
# mtcars becomes the first argument of count
mtcars %>% count(cyl)
```

Pipe operator

- In more complex examples, with multiple function calls, the pipe operator improves readability.

```
# Without pipe operator
fit_model(prepare_data(dataset))
# With pipe operator
dataset %>%
  prepare_data %>%
  fit_model
```

Examples i

```r
# Let's convert our previous example to use the pipe
mtcars %>%
  mutate(litres_per_100km = 235.215/mpg) %>%
  summarise(avg_lit = mean(litres_per_100km),
            sd_lit = sd(litres_per_100km)) %>%
  mutate(low_bd = avg_lit - 1.96*sd_lit/sqrt(n),
         up_bd = avg_lit + 1.96*sd_lit/sqrt(n))
```

```
##    avg_lit   sd_lit   low_bd    up_bd
## 1 12.75506 3.863251 11.41651 14.09361
```

Examples ii

- We didn't need intermediate datasets `data1`, `data2` and `data3`.
- It's easier to read.

Summaries by group

- We can combine `summarise` and `group_by` to create summaries for each group individually.

```r
# Average mpg for each value of cyl
mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg))
```

```
## # A tibble: 3 x 2
##     cyl avg_mpg
## * <dbl>   <dbl>
## 1     4    26.7
## 2     6    19.7
## 3     8    15.1
```

Examples i

```r
# Average mpg for each value of cyl + 95% CI
mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg),
            sd_mpg = sd(mpg),
            n = n()) %>%
  mutate(low_bd = avg_mpg - 1.96*sd_mpg/sqrt(n),
         up_bd = avg_mpg + 1.96*sd_mpg/sqrt(n))
```

Examples ii

```
## # A tibble: 3 x 6
##     cyl avg_mpg sd_mpg     n low_bd up_bd
## * <dbl>   <dbl>  <dbl> <int>  <dbl> <dbl>
## 1     4    26.7   4.51    11   24.0  29.3
## 2     6    19.7   1.45     7   18.7  20.8
## 3     8    15.1   2.56    14   13.8  16.4
```

- **Very important**: The number of observations in each group is different!
- This is why we computed the number of observations in each group using the function `n()`.

Examples iii

- When we compute the confidence interval, the variable `n` refers to the column `n` in the dataset, i.e. what we computed using summarise.
- Here, the "word" `n` refers to three different things:
  - A function, `n()`, which counts the number of observations.
  - A column in the dataset that we created using the function `n()`.
  - The number of rows of `mtcars` that we computed earlier.
- R keeps track of all of these (using something called "scoping rules"), but for a human this can be confusing... It's best to avoid it if we can.

Examples iv

```r
# Average mpg for each value of cyl + 95% CI
mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg),
            sd_mpg = sd(mpg),
            nobs = n()) %>%
  mutate(low_bd = avg_mpg - 1.96*sd_mpg/sqrt(nobs),
         up_bd = avg_mpg + 1.96*sd_mpg/sqrt(nobs))
```

Proportions are means too!

- It may not be obvious at first, but proportions are means!
- If I want the proportion of apples among fruits, I can take the mean of binary observations:
    - $X_i = 1$ if the i-th fruit is an apple.
    - $X_i = 0$ otherwise.
- This means we can use the CLT for proportions too.
    - **Note**: It doesn't work well when the proportion $\hat{p}$ and the number of observations n are small.

Examples i

```r
summarise(mtcars, prop = mean(cyl == 6))
```

```
##      prop
## 1 0.21875
```

- **Note**: If $\hat{p}$ is the proportion, then $\hat{\sigma} = \sqrt{\hat{p}(1-\hat{p})}$.

Examples ii

```r
n <- nrow(mtcars)
mtcars %>%
  summarise(prop = mean(cyl == 6)) %>%
  mutate(sigma = sqrt(prop*(1 - prop)),
         low_bd = prop - 1.96*sigma/sqrt(n),
         up_bd = prop + 1.96*sigma/sqrt(n))
```

```
##      prop     sigma     low_bd     up_bd
## 1 0.21875 0.4133986 0.07551468 0.3619853
```

Examples iii

```
# For all values of cyl
mtcars %>%
  group_by(cyl) %>%
  summarise(nobs = n(), prop = nobs/n) %>%
  mutate(sigma = sqrt(prop*(1 - prop)),
         low_bd = prop - 1.96*sigma/sqrt(n),
         up_bd = prop + 1.96*sigma/sqrt(n))
```

Examples iv

```
## # A tibble: 3 x 6
##     cyl  nobs  prop sigma low_bd up_bd
## * <dbl> <int> <dbl> <dbl>  <dbl> <dbl>
## 1     4    11 0.344 0.475 0.179  0.508
## 2     6     7 0.219 0.413 0.0755 0.362
## 3     8    14 0.438 0.496 0.266  0.609
```