STATS 2107
Statistical Modelling and Inference II
Practical 2: Manipulating Data

Sharon Lee, Matt Ryan

Semester 2 2022

In this practical, you start to look at how to

- manipulate the data,
- summarise data, and
- write functions.

Manipulating data

We are going to use the package `dplyr` to manipulate the `mpg` dataset. It is contained within the package `tidyverse` so let's load that:

```
library(tidyverse)
```

NOTE: This assumes `tidyverse` is installed, have a look at Practical 1.

Next, we load some data. I will use some built in data to make it easy to load. Load with

```
data("mpg")
mpg
```

```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 224 more rows
```

Filter

How can I filter subjects given a criteria? For example, how could I get all cars manufactured by `audi`? In the `mpg` dataset, there is a column called `manufacturer` that has the information about each manufacturer. We can see the number of cars from each manufacturer with:

```
table(mpg$manufacturer)
```

```
## 
##       audi  chevrolet      dodge       ford      honda    hyundai       jeep 
##         18         19         37         25          9         14          8 
## land rover    lincoln    mercury     nissan    pontiac     subaru     toyota 
##          4          3          4         13          5         14         34 
## volkswagen 
##         27 
```

So how do we get all the cars whose manufacturer is `audi`?

```
filter(mpg, manufacturer == "audi")
```

```
## # A tibble: 18 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## 11 audi         a4 quattro   2    2008     4 auto~ 4        19    27 p     comp~
## 12 audi         a4 quattro   2.8  1999     6 auto~ 4        15    25 p     comp~
## 13 audi         a4 quattro   2.8  1999     6 manu~ 4        17    25 p     comp~
## 14 audi         a4 quattro   3.1  2008     6 auto~ 4        17    25 p     comp~
## 15 audi         a4 quattro   3.1  2008     6 manu~ 4        15    25 p     comp~
## 16 audi         a6 quattro   2.8  1999     6 auto~ 4        15    24 p     mids~
## 17 audi         a6 quattro   3.1  2008     6 auto~ 4        17    25 p     mids~
## 18 audi         a6 quattro   4.2  2008     8 auto~ 4        16    23 p     mids~
```

Let's break this down:

- `filter()` is the command that we want,
- `mpg` is the first argument and should be the name of the dataframe that we are going to filter, and
- `manufacturer == "audi"` is the next argument which gives the constraints in the filter command: note the use of `==` which is the logical equals.

More than one constraint - no problem:

```
filter(mpg, manufacturer == "audi", year <= 2000)
```

```
## # A tibble: 9 x 11
##   manufacturer model      displ  year   cyl trans  drv     cty   hwy fl    class
##   <chr>        <chr>      <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
## 1 audi         a4           1.8  1999     4 auto(~ f        18    29 p     comp~
## 2 audi         a4           1.8  1999     4 manua~ f        21    29 p     comp~
## 3 audi         a4           2.8  1999     6 auto(~ f        16    26 p     comp~
## 4 audi         a4           2.8  1999     6 manua~ f        18    26 p     comp~
## 5 audi         a4 quattro   1.8  1999     4 manua~ 4        18    26 p     comp~
## 6 audi         a4 quattro   1.8  1999     4 auto(~ 4        16    25 p     comp~
## 7 audi         a4 quattro   2.8  1999     6 auto(~ 4        15    25 p     comp~
## 8 audi         a4 quattro   2.8  1999     6 manua~ 4        17    25 p     comp~
## 9 audi         a6 quattro   2.8  1999     6 auto(~ 4        15    24 p     mids~
```

Quiz questions

1. What does the command given above return?
2. Filter for cars with front-wheel drive and 6 cylinders. How many cars satisfy these requirements?
3. What commands can be used to filter for all cars from `audi` or `dodge`?
4. Filter for all cars from `audi` or `dodge`. How many cars satisfy this condition?

Select

How do we choose a smaller set of columns? The command `select()` is your friend.

```
select(mpg, model, trans)
```

```
## # A tibble: 234 x 2
##    model      trans
##    <chr>      <chr>
##  1 a4         auto(l5)
##  2 a4         manual(m5)
##  3 a4         manual(m6)
##  4 a4         auto(av)
##  5 a4         auto(l5)
##  6 a4         manual(m5)
##  7 a4         auto(av)
##  8 a4 quattro manual(m5)
##  9 a4 quattro auto(l5)
## 10 a4 quattro manual(m6)
## # ... with 224 more rows
```

If you want to select variables whose names contain a particular word, then use `contains()`:

```
select(mpg, contains("dis"))
```

```
## # A tibble: 234 x 1
##    displ
##    <dbl>
##  1   1.8
##  2   1.8
##  3   2
##  4   2
##  5   2.8
##  6   2.8
##  7   3.1
##  8   1.8
##  9   1.8
## 10   2
## # ... with 224 more rows
```

If you want to view a range of columns, use `:`

```
select(mpg, displ:cyl)
```

```
## # A tibble: 234 x 3
##    displ  year   cyl
##    <dbl> <int> <int>
##  1   1.8  1999     4
##  2   1.8  1999     4
##  3   2    2008     4
##  4   2    2008     4
##  5   2.8  1999     6
##  6   2.8  1999     6
##  7   3.1  2008     6
##  8   1.8  1999     4
##  9   1.8  1999     4
## 10   2    2008     4
## # ... with 224 more rows
```

Quiz questions

5. What commands can be used to get the model, displ, and year.

Mutate

We can create new columns with the `mutate()` command. To illustrate, we are going to create a new column that is the city efficiency in km per litre.

```
mutate(mpg, cty_km_l = cty * 0.425144)
```

```
## # A tibble: 234 x 12
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 224 more rows, and 1 more variable: cty_km_l <dbl>
```

If you look at the end of the output, you can see the new column `cty_km_l`. So excited by this, let's look at `mpg` again:

```
mpg
```

```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 224 more rows
```

ITS GONE!

The problem is that even though R created a new column, it did not save it. We need to tell R to do that:

```
mpg  <- mutate(mpg, cty_km_l = cty * 0.425144)
```

Now it is saved.

Magrittr

What if you want to filter and select and mutate? Well you could do something like this:

```
mpg_audi  <- filter(mpg, manufacturer == "audi")
mpg_trans_cty  <- select(mpg_audi, trans:cty)
mpg_trans_cty_km  <- mutate(mpg_trans_cty, cty_km_l = cty * 0.425144)
mpg_trans_cty_km
```

```
## # A tibble: 18 x 4
##    trans      drv     cty cty_km_l
##    <chr>      <chr> <int>    <dbl>
##  1 auto(l5)   f        18     7.65
##  2 manual(m5) f        21     8.93
##  3 manual(m6) f        20     8.50
##  4 auto(av)   f        21     8.93
##  5 auto(l5)   f        16     6.80
##  6 manual(m5) f        18     7.65
##  7 auto(av)   f        18     7.65
##  8 manual(m5) 4        18     7.65
##  9 auto(l5)   4        16     6.80
## 10 manual(m6) 4        20     8.50
## 11 auto(s6)   4        19     8.08
## 12 auto(l5)   4        15     6.38
## 13 manual(m5) 4        17     7.23
## 14 auto(s6)   4        17     7.23
## 15 manual(m6) 4        15     6.38
## 16 auto(l5)   4        15     6.38
## 17 auto(s6)   4        17     7.23
## 18 auto(s6)   4        16     6.80
```

This gets very annoying very quickly. Instead we can chain commands together using the command %>%. This command %>% is called a magrittr[1]. Let's see an example:

```
mpg %>%
  filter(manufacturer == "audi") %>%
  select(trans:cty) %>%
  mutate(cty_km_l = cty * 0.425144)
```

```
## # A tibble: 18 x 4
##    trans      drv     cty cty_km_l
##    <chr>      <chr> <int>    <dbl>
##  1 auto(l5)   f        18     7.65
##  2 manual(m5) f        21     8.93
##  3 manual(m6) f        20     8.50
##  4 auto(av)   f        21     8.93
```

[1] https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html

```
##  5 auto(l5)   f         16     6.80
##  6 manual(m5) f         18     7.65
##  7 auto(av)   f         18     7.65
##  8 manual(m5) 4         18     7.65
##  9 auto(l5)   4         16     6.80
## 10 manual(m6) 4         20     8.50
## 11 auto(s6)   4         19     8.08
## 12 auto(l5)   4         15     6.38
## 13 manual(m5) 4         17     7.23
## 14 auto(s6)   4         17     7.23
## 15 manual(m6) 4         15     6.38
## 16 auto(l5)   4         15     6.38
## 17 auto(s6)   4         17     7.23
## 18 auto(s6)   4         16     6.80
```

Read `%>%` as `then`. So this reads as

- `mpg`: get the dataframe,
- `%>%`: then,
- `filter(manufacturer == "audi")`: filter for Audi,
- `%>%`: then,
- `select(trans:cty)`: select from trans to cty,
- `%>%`: then,
- `mutate(cty_km_l = cty * 0.425144)`: create a new column that contains the city efficiency in km per litre.

Challenge

Get all cars from `audi` that were made before or including 2000. Keep only the information for manufacturer, model, displ, year, cty. Filter these for all cars with a city efficiency of greater than 8 km/litre.

Quiz questions

6. How many cars passed the above challenge?

Group_by and summarise

To get summary statistics we can use `mean()` etc, but what if we wanted the mean city efficiency for each manufacturer? This can be done with a mix of `group_by` and then `summarise()`.

```
mpg %>%
  group_by(manufacturer) %>%
  summarise(mean = mean(cty, na.rm = TRUE))
```

```
## # A tibble: 15 x 2
##    manufacturer  mean
##    <chr>        <dbl>
##  1 audi          17.6
##  2 chevrolet     15  
##  3 dodge         13.1
##  4 ford          14  
##  5 honda         24.4
##  6 hyundai       18.6
##  7 jeep          13.5
##  8 land rover    11.5
##  9 lincoln       11.3
## 10 mercury       13.2
## 11 nissan        18.1
## 12 pontiac       17
## 13 subaru        19.3
## 14 toyota        18.5
## 15 volkswagen    20.9
```

We can add more summary statistics:

```r
mpg %>%
  group_by(manufacturer) %>%
  summarise(mean = mean(cty, na.rm = TRUE),
            n = n(),
            sd = sd(cty, na.rm = TRUE))
```

```
## # A tibble: 15 x 4
##    manufacturer  mean     n    sd
##    <chr>        <dbl> <int> <dbl>
##  1 audi          17.6      18 1.97
##  2 chevrolet     15        19 2.92
##  3 dodge         13.1      37 2.49
##  4 ford          14        25 1.91
##  5 honda         24.4       9 1.94
##  6 hyundai       18.6      14 1.50
##  7 jeep          13.5       8 2.51
##  8 land rover    11.5       4 0.577
##  9 lincoln       11.3       3 0.577
## 10 mercury       13.2       4 0.5
## 11 nissan        18.1      13 3.43
## 12 pontiac       17         5 1
## 13 subaru        19.3      14 0.914
## 14 toyota        18.5      34 4.05
## 15 volkswagen    20.9      27 4.56
```

Functions

Often, you end up typing the same code again and again. Or at least copying and pasting a lot. Remember

Data analysis rule 3: Don't copy and paste - write a function.

So lets write a function to calculate the 95% confidence interval lower bound for the mean of some data.

First, type the following code into a script called `get_lwr_ci.R`.

```r
# A function to take a vector of numbers and give a CI lower bound
#
# Uses the standard one for normal data with an unknown variance
# That is, this calculates the lower bound for a one sample test
# Using a t-distribution.
#
# arguments:
# x: a vector of numbers
# level: The strength of your confidence interval, i.e. 0.95 for 95%
#
# returns:
# vector with the lower point
get_lwr_ci  <- function(x, level = 0.95){
  # get mean
  m  <- mean(x, na.rm = TRUE)
  # get SE
  s  <- sd(x, na.rm = TRUE)
  n <- length(x)
  se  <- s / sqrt(n)
  # get t cutoff point
  a  <- (1 - level) / 2
  t  <- qt(a, df = n-1, lower.tail = FALSE)
  # lower point
  lwr  <- m - t * se
  # return
  return(lwr)
}
```

Save this in the same folder as your main script file.

Now in your main script file for this practical, you need to add the line:

```
source("get_lwr_ci.R")
```

NOTE: This assumes that the folder you have saved your main script file in is your working directory. Have a look at Practical 1 if you don't remember how to set this up.

When you run this line, it reads the file into the environment of R and if it is type correctly should give you a new function.

We can now test this with

```
get_lwr_ci(mpg$cty)
```

```
## [1] 16.31083
```

If you change your function, remember to rerun:

```
source("get_lwr_ci.R")
```

so that R loads the latest version into the environment.

Functions are of the basic form

```
name <- function(variables) {
  # Do stuff
  return(value)
}
```

The `name` is the name of the function, the variables are the arguments that will be passed into the function. Note that we can set default values like we did with `level = 0.95` in the `get_lwr_ci()` function. If we do not give a value for level, then 0.95 is used. Finally we return the values.

Challenge

1. Write the function to give the upper bound.
2. Get the 95% CI for the mean `cty` for each manufacturer.