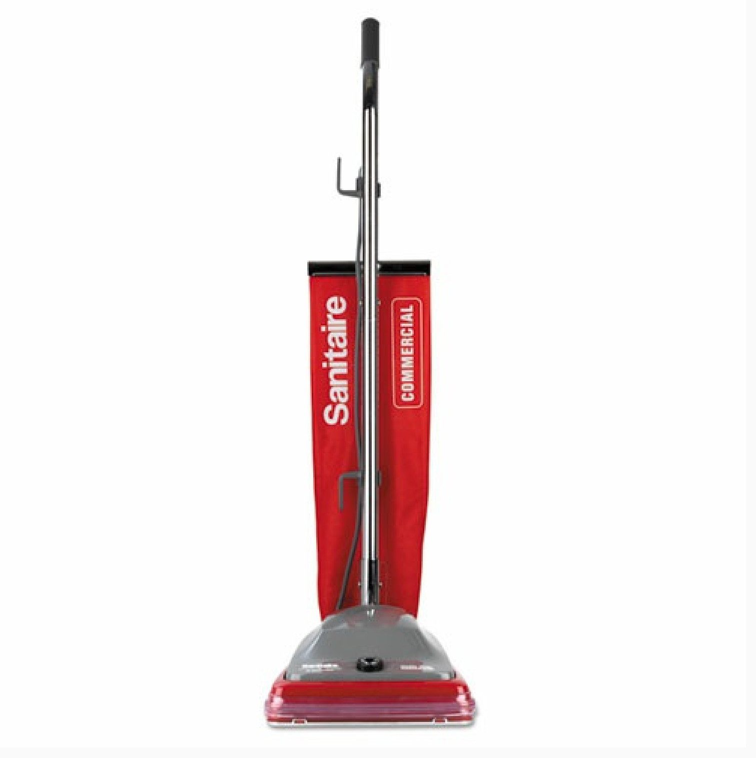
Data Analysis in R

Data Cleaning

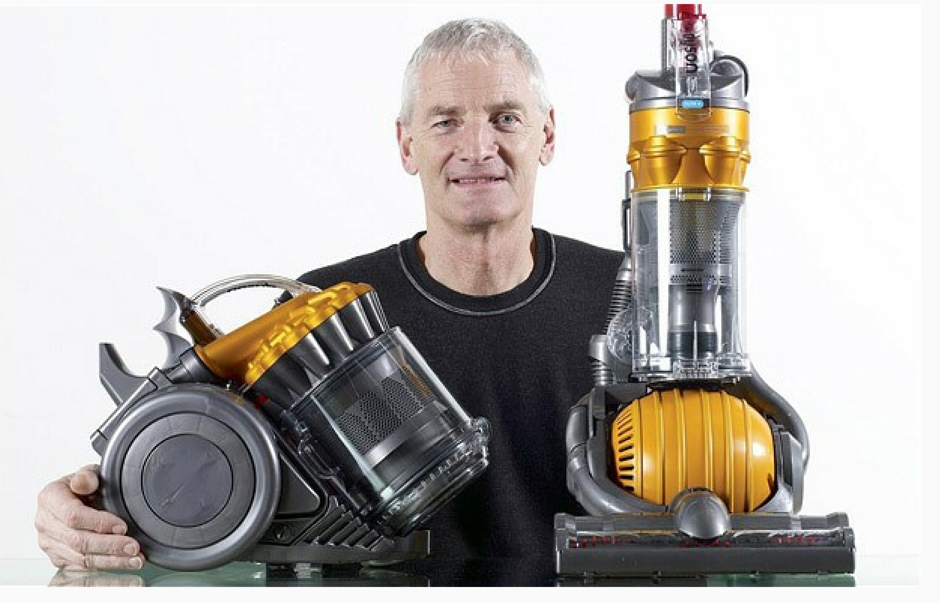
Michael E DeWitt Jr

2018-09-20 (updated: 2018-10-04)

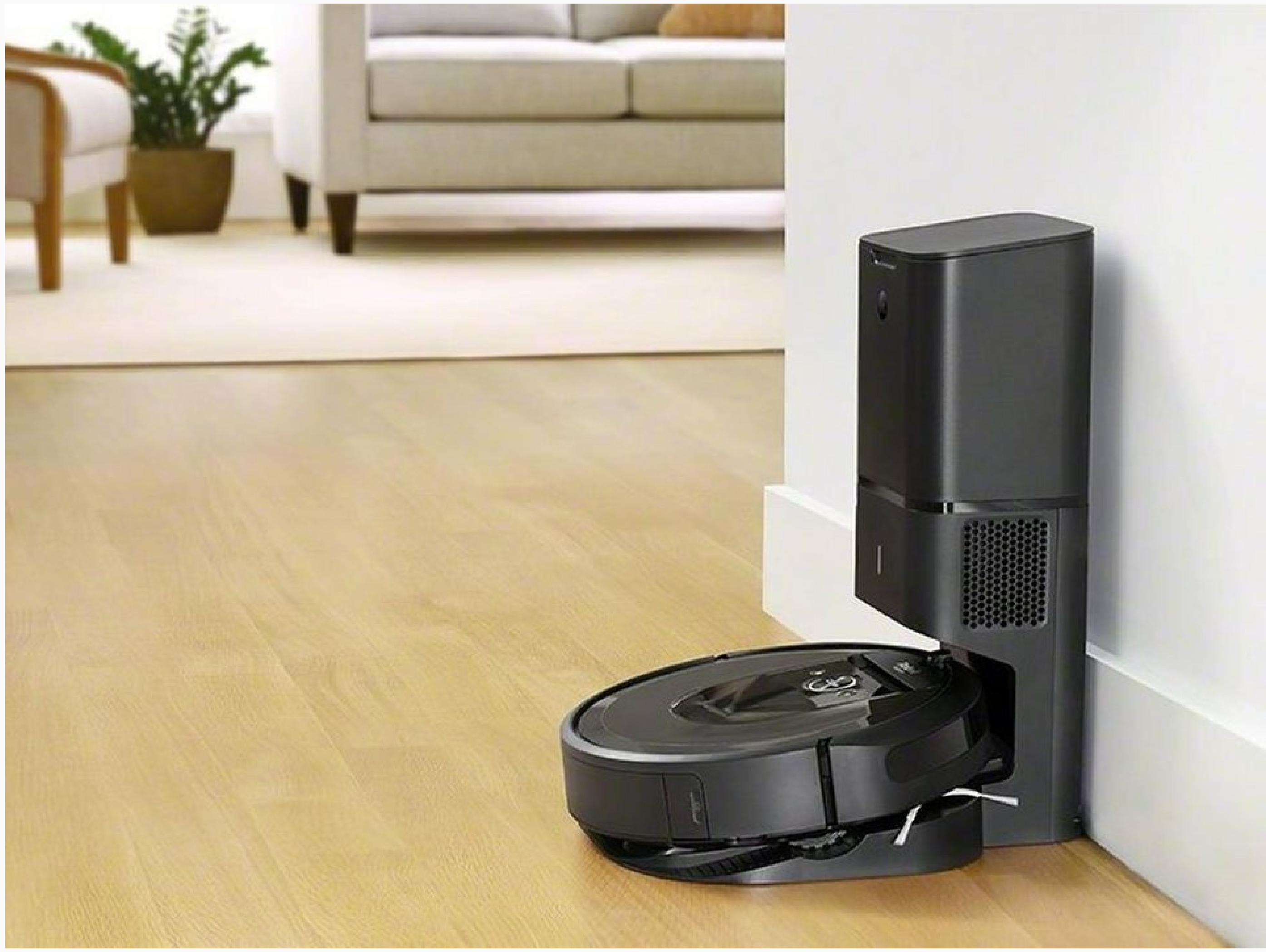
Cleaning is always a chore...



But wouldn't you rather be a Dyson?



Or even better...

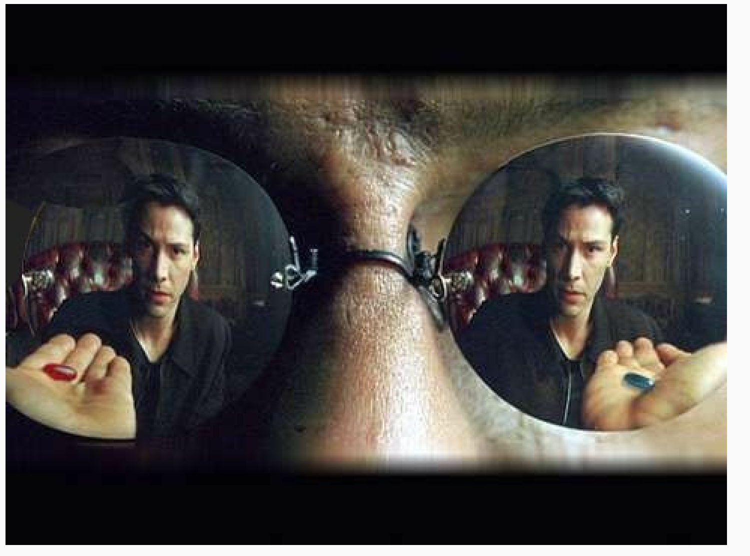


Get Started

Getting Data in...

The first part of the process in uploading data into R.

But the biggest question is what kind of data do you have?



Some Common Data Formats

- Delimited Files (fixed width, comma, tab, ...)
- Excel
- SPSS
- STATA
- Semi- structured (JSON, XML)

Enter `readr`, part of the `tidyverse`

`readr` is a package in the `tidyverse`

Enter `readr`, part of the `tidyverse`

`readr` is a package in the `tidyverse`

Designed with interfaces to handle most files

Enter `readr`, part of the `tidyverse`

`readr` is a package in the `tidyverse`

Designed with interfaces to handle most files

Has some nice defaults that make it easier than some base packages

Starting with CSVs

Syntax

```
data <- read_csv("path_to_csv.csv")
```

Starting with CSVs

Syntax

```r
data <- read_csv("path_to_csv.csv")
```

Options

- Specify column names
- Where data start
- Column types

Other delimiters

Tab Delimited Files

```
data <- read_tsv("path_to_csv.txt")
```

Any other delimiter

```
data <- read_delim("path_to_csv.txt", delim = ";")
```

Now to haven

The `haven` package can be used to read more proprietary data formats into R

SAS

```
my_sas <- read_sas("myfile.sas7bdat")
```

SPSS

```
my_sas <- read_spss("myfile.sav")
```

STATA

```
my_sas <- read_dta("myfile.dta")
```

Excel using `readxl`

And of course the most ubiquitous data form...

```
my_excel <-read_excel("my_excel_file.xlsx")
```

Excel using `readxl`

And of course the most ubiquitous data form...

```
my_excel <-read_excel("my_excel_file.xlsx")
```

- Specify cell ranges
- Name columns
- etc

And other formats

Json with `jsonlite`

XML with `xml2` or `XML` *(my preference is `xml2`)*

Now for cleaning

Tidy data

Most of the `tidyverse` relies on the "tidy" paradigm of data

Tidy data

Most of the `tidyverse` relies on the "tidy" paradigm of data

One observation per row, one attribute per column

Tidy data

Most of the `tidyverse` relies on the "tidy" paradigm of data

One observation per row, one attribute per column

Once data is in the format, visualisation and modeling becomes easier

Wide to Narrow and Back Again

We will use the `tidyr` package to help move from wide data to narrow data to move to the tidy format

Wide to Narrow and Back Again

We will use the `tidyr` package to help move from wide data to narrow data to move to the tidy format

Key functions

`spread`

Narrow to Wide

`gather`

Wide to narrow

Spread

Syntax

```
spread(data = data,
       key = "What You Want to Be Columns",
       value = "The value you want in the rows",
       fill = "What you want to appear if there are no values")
```

Spread

Syntax

```
spread(data = data,
       key = "What You Want to Be Columns",
       value = "The value you want in the rows",
       fill = "What you want to appear if there are no values")
```

Example

```
stocks
```

```
## # A tibble: 30 x 3
##    time       stock   price
##    <date>     <chr>   <dbl>
##  1 2010-01-01 X      0.892
##  2 2010-01-02 X      1.09
##  3 2010-01-03 X     -0.0525
##  4 2010-01-04 X      0.899
##  5 2010-01-05 X      0.326
##  6 2010-01-06 X      1.82
##  7 2010-01-07 X      0.0298
##  8 2010-01-08 X      0.164
##  9 2010-01-09 X      0.428
## 10 2010-01-10 X     -0.475
## # ... with 20 more rows
```

```
stocks %>%
  spread(stock, price)
```

```
## # A tibble: 10 x 4
##    time             X       Y      Z
##    <date>       <dbl>   <dbl>  <dbl>
##  1 2010-01-01  0.892  -0.0475  7.63
##  2 2010-01-02  1.09    2.79   -1.78
##  3 2010-01-03 -0.0525  1.72   -0.642
##  4 2010-01-04  0.899   3.03   -1.26
##  5 2010-01-05  0.326  -2.84    2.88
##  6 2010-01-06  1.82    1.75    5.27
##  7 2010-01-07  0.0298  1.22    4.13
##  8 2010-01-08  0.164  -1.81    2.52
##  9 2010-01-09  0.428  -1.43   -7.40
## 10 2010-01-10 -0.475  -2.44   -3.41
```

Gather

Syntax

```
gather(data = data,
       key = "new name of column",
       value = "what you want to call value",
       -"what you don't want grouped")
```

Gather

Syntax

```
gather(data = data,
       key = "new name of column",
       value = "what you want to call value",
       -"what you don't want grouped")
```

Example

```
mini_iris
```

```
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Spe
## *        <dbl>       <dbl>        <dbl>       <dbl> <fc
## 1          5.1         3.5          1.4         0.2 set
## 2          7           3.2          4.7         1.4 ver
## 3          6.3         3.3          6           2.5 vir
```

```
mini_iris %>%
  gather(attribute, measurement, -Species)
```

```
## # A tibble: 12 x 3
##    Species    attribute    measurement
##    <fct>      <chr>              <dbl>
##  1 setosa     Sepal.Length         5.1
##  2 versicolor Sepal.Length         7  
##  3 virginica  Sepal.Length         6.3
##  4 setosa     Sepal.Width          3.5
##  5 versicolor Sepal.Width          3.2
##  6 virginica  Sepal.Width          3.3
##  7 setosa     Petal.Length         1.4
##  8 versicolor Petal.Length         4.7
##  9 virginica  Petal.Length         6  
## 10 setosa     Petal.Width          0.2
## 11 versicolor Petal.Width          1.4
## 12 virginica  Petal.Width          2.5
```

Now Let's Practice `spread` and `gather`

Subset with `select`

Suppose you want to retain only a few columns.

This can be done with the `select` command from `dplyr`

```
iris %>%
  select(Sepal.Length, Petal.Width)
```

```
## # A tibble: 150 x 2
##    Sepal.Length Petal.Width
##           <dbl>       <dbl>
##  1          5.1         0.2
##  2          4.9         0.2
##  3          4.7         0.2
##  4          4.6         0.2
##  5          5           0.2
##  6          5.4         0.4
##  7          4.6         0.3
##  8          5           0.2
##  9          4.4         0.2
## 10          4.9         0.1
## # ... with 140 more rows
```

Helpers for `select`

`starts_with("something")` selects those columns that start with a specified string

`contains("something")` selects those columns that have a string *anywhere* in the column name

`ends_with("something")` selects those columns that end with a specified string

`everything()` selects everything else that was not explicitly selected

Helpers for `select`

`starts_with("something")` selects those columns that start with a specified string

`contains("something")` selects those columns that have a string *anywhere* in the column name

`ends_with("something")` selects those columns that end with a specified string

`everything()` selects everything else that was not explicitly selected

```
iris %>%
  select(starts_with("Sepal"))
```

```
## # A tibble: 150 x 2
##    Sepal.Length Sepal.Width
##           <dbl>       <dbl>
##  1          5.1         3.5
##  2          4.9         3
##  3          4.7         3.2
##  4          4.6         3.1
##  5          5           3.6
##  6          5.4         3.9
##  7          4.6         3.4
##  8          5           3.4
##  9          4.4         2.9
## 10          4.9         3.1
## # ... with 140 more rows
```

`unite` and `separate`

`unite` is a function that allows you to join (paste together) two columns

Syntax

```
unite(data = data, col = "new_column_name", sep = "separator",
      remove = "T/F if you want to drop the columns", -column_you_dont_want_joined)
```

unite and separate

`unite` is a function that allows you to join (paste together) two columns

Syntax

```
unite(data = data, col = "new_column_name", sep = "separator",
      remove = "T/F if you want to drop the columns", -column_you_dont_want_joined)
```

Example

```
sample_df
```

```
## # A tibble: 3 x 3
##   col1   col2 col3
##   <chr> <dbl> <chr>
## 1 A         1 X
## 2 B         2 Y
## 3 C         3 Z
```

```
sample_df %>%
  unite(col = "united", sep = "_",
        remove = FALSE, -col3)
```

```
## # A tibble: 3 x 4
##   united col1   col2 col3
##   <chr>  <chr> <dbl> <chr>
## 1 A_1    A         1 X
## 2 B_2    B         2 Y
## 3 C_3    C         3 Z
```

unite and separate

`separate` is a function that allows you to split a column into multiple columns

Syntax

```
separate(data = data, col = "what to separate", into = "new columns",
         sep = " what to separate by", remove = "T/F")
```

unite and separate

`separate` is a function that allows you to split a column into multiple columns

Syntax

```
separate(data = data, col = "what to separate", into = "new columns",
         sep = " what to separate by", remove = "T/F")
```

Example

```
sample_df
```

```
## # A tibble: 3 x 2
##   united col3
##   <chr>  <chr>
## 1 A_1    X
## 2 B_2    Y
## 3 C_3    Z
```

```
sample_df %>%
  separate(col = united, into = c("col1", "col2"),
           sep = "_", remove = TRUE)
```

```
## # A tibble: 3 x 3
##   col1  col2  col3
##   <chr> <chr> <chr>
## 1 A     1     X
## 2 B     2     Y
## 3 C     3     Z
```

Now Let's Practice `select`, `gather` and `unite`

Using `filter`

`filter` can be used to filter values to those ones that you would like

Syntax

```
filter(data = data, condition to test)
```

Using `filter`

`filter` can be used to filter values to those ones that you would like

Syntax

```
filter(data = data, condition to test)
```

Example

```
table(iris[["Species"]])
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
iris %>%
  filter(Species == "setosa") %>%
  select(Species, contains("Sepal"))
```

```
## # A tibble: 50 x 3
##    Species Sepal.Length Sepal.Width
##    <fct>          <dbl>       <dbl>
##  1 setosa           5.1         3.5
##  2 setosa           4.9         3
##  3 setosa           4.7         3.2
##  4 setosa           4.6         3.1
##  5 setosa           5           3.6
##  6 setosa           5.4         3.9
##  7 setosa           4.6         3.4
##  8 setosa           5           3.4
##  9 setosa           4.4         2.9
## 10 setosa           4.9         3.1
## # ... with 40 more rows
```

Logical Operators

- **==** for testing equivalence
- **!=** not equal to
- **>** / **<** greater than or less than
- **>=** / **<=** greater than or equal to
- **&**/ | and and or allows for combining conditions (e.g. Species == "Setosa" & Sepal.Length >5)

`rename` for changing column names

`rename` can be used

Syntax

```
rename(data = data, new_name = old_name)
```

`rename` for changing column names

`rename` can be used

Syntax

```
rename(data = data, new_name = old_name)
```

Example

```
iris
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Sp
##           <dbl>       <dbl>        <dbl>       <dbl> <f
##  1          5.1         3.5          1.4         0.2 se
##  2          4.9         3            1.4         0.2 se
##  3          4.7         3.2          1.3         0.2 se
##  4          4.6         3.1          1.5         0.2 se
##  5          5           3.6          1.4         0.2 se
##  6          5.4         3.9          1.7         0.4 se
##  7          4.6         3.4          1.4         0.3 se
##  8          5           3.4          1.5         0.2 se
##  9          4.4         2.9          1.4         0.2 se
## 10          4.9         3.1          1.5         0.1 se
## # ... with 140 more rows
```

```
iris %>%
  rename(sepal_length = Sepal.Length) %>%
  select(sepal_length, ends_with("Width"))
```

```
## # A tibble: 150 x 3
##    sepal_length Sepal.Width Petal.Width
##           <dbl>       <dbl>       <dbl>
##  1          5.1         3.5         0.2
##  2          4.9         3           0.2
##  3          4.7         3.2         0.2
##  4          4.6         3.1         0.2
##  5          5           3.6         0.2
##  6          5.4         3.9         0.4
##  7          4.6         3.4         0.3
##  8          5           3.4         0.2
##  9          4.4         2.9         0.2
## 10          4.9         3.1         0.1
## # ... with 140 more rows
```

`mutate` to add new columns

mutate can be used to create a *derrived* or *calculated* column

Syntax

```
mutate(data = data, new_column = operation you want to do)
```

`mutate` to add new columns

mutate can be used to create a *derrived* or *calculated* column

Syntax

```
mutate(data = data, new_column = operation you want to do)
```

Example

```
mtcars
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    a
##    * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl
##  1  21       6  160    110  3.9   2.62  16.5     0
##  2  21       6  160    110  3.9   2.88  17.0     0
##  3  22.8     4  108     93  3.85  2.32  18.6     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0
##  6  18.1     6  225    105  2.76  3.46  20.2     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0
##  8  24.4     4  147.    62  3.69  3.19  20       1
##  9  22.8     4  141.    95  3.92  3.15  22.9     1
## 10  19.2     6  168.   123  3.92  3.44  18.3     1
## # ... with 22 more rows
```

```
mtcars %>%
  mutate(mpg_per_wt = mpg / wt) %>%
  select(mpg, wt, mpg_per_wt)
```

```
## # A tibble: 32 x 3
##      mpg    wt mpg_per_wt
##    <dbl> <dbl>      <dbl>
##  1  21    2.62       8.02
##  2  21    2.88       7.30
##  3  22.8  2.32       9.83
##  4  21.4  3.22       6.66
##  5  18.7  3.44       5.44
##  6  18.1  3.46       5.23
##  7  14.3  3.57       4.01
##  8  24.4  3.19       7.65
##  9  22.8  3.15       7.24
## 10  19.2  3.44       5.58
## # ... with 22 more rows
```

Group Operations

`group_by` allows for group-wise observations

You can then feed these groups to the `sumarise/summarize` function to do group-wise calculations

Group Operations

`group_by` allows for group-wise observations

You can then feed these groups to the `sumarise/summarize` function to do group-wise calculations

```
iris %>%
  group_by(Species) %>%
  summarise(mu_length = mean(Sepal.Length),
           med_width = median(Sepal.Width))
```

```
## # A tibble: 3 x 3
##   Species    mu_length med_width
##   <fct>          <dbl>     <dbl>
## 1 setosa          5.01       3.4
## 2 versicolor      5.94       2.8
## 3 virginica       6.59       3
```

Add more groups with `group_by`

```
mtcars %>%
  group_by(cyl, am) %>%
  summarise(avg_mpg = mean(mpg),
            n = n(),
            min_wt = min(wt),
            max_wt = max(wt)) %>%
  mutate(perc_of_group = n/sum(n)) %>%
  ungroup() %>% # Needed to remove grouping
  mutate(perc_total = n/sum(n))
```

```
## # A tibble: 6 x 8
##     cyl    am avg_mpg     n min_wt max_wt perc_of_group perc_total
##   <dbl> <dbl>   <dbl> <int>  <dbl>  <dbl>         <dbl>      <dbl>
## 1     4     0    22.9     3   2.46   3.19         0.273     0.0938
## 2     4     1    28.1     8   1.51   2.78         0.727     0.25
## 3     6     0    19.1     4   3.22   3.46         0.571     0.125
## 4     6     1    20.6     3   2.62   2.88         0.429     0.0938
## 5     8     0    15.0    12   3.44   5.42         0.857     0.375
## 6     8     1    15.4     2   3.17   3.57         0.143     0.0625
```

To Recap

`gather` and `spread` to transform our data between wide and long forms

`select` to subset columns

`filter` to filter data to a specific condition

`unite` and `separate` to combine and separate columns

`rename` to rename our columns

`mutate` to add new derrived or calculated columns

`group_by` for group-wise operations

`summarise` for summary functions on grouped variables

Thanks!