Data handling in R using the *tidyverse*

Part 2 / 2

Joel Frischknecht

02.10.2019

Overview

Last week:

- Dataframe structure, tidy data, *tibble's*
- Data manipulation with *dplyr*
- Recommended ressources

Today:

- Importing data with *readr*
- Reshaping data with *tidyr*
- Creating some simple plots with *ggplot2*

Note: Applications today will be with "real" economic data, i.e. data from a household survey in Uganda.

Remember last week...

Our topics:

1. Dataframe structure (*tibble's*)
2. Data manipulation with *dplyr*:
   `filter()`, `arrange()`, `select()`, `summarise()` `mutate()` and `group_by()`.

- Any open questions, things to discuss, additional explanations you would like to have?
- Homework exercises (`rename()`, `count()`, `distinct()`)

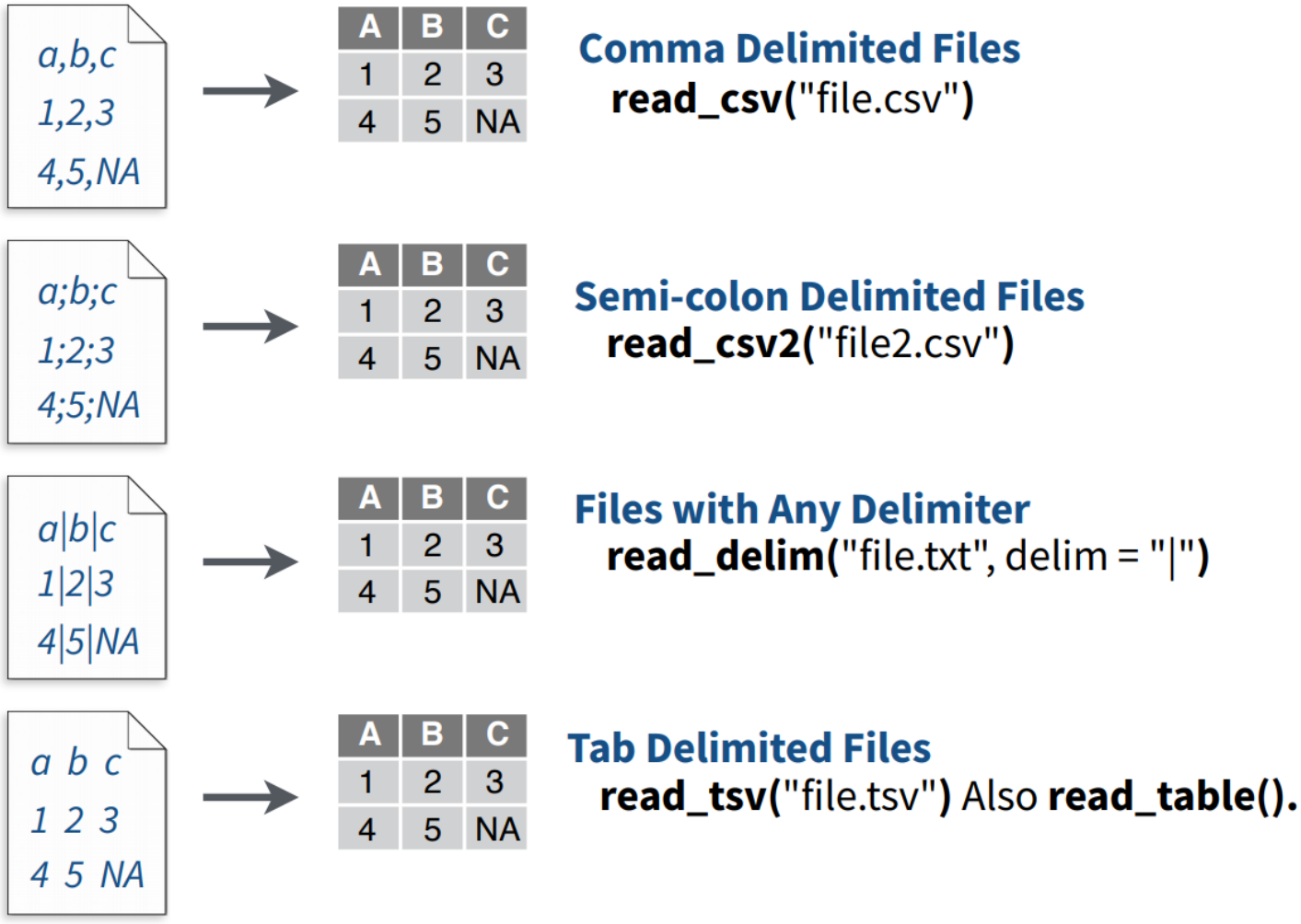
Importing data

We focus purely on tabular data, the most common type of data you will encounter.

1. Text files - with *readr* (our main focus)
2. "Foreign" files, e.g. STATA-, SPSS-, SAS- and Excel-Files (briefly)

Tabular text files

Most common text files and their corresponding import-verb in *readr*:



Paths:

Unless you specify a full path to your file (e.g. C:/...), paths are always intepreted relative to your working directory:

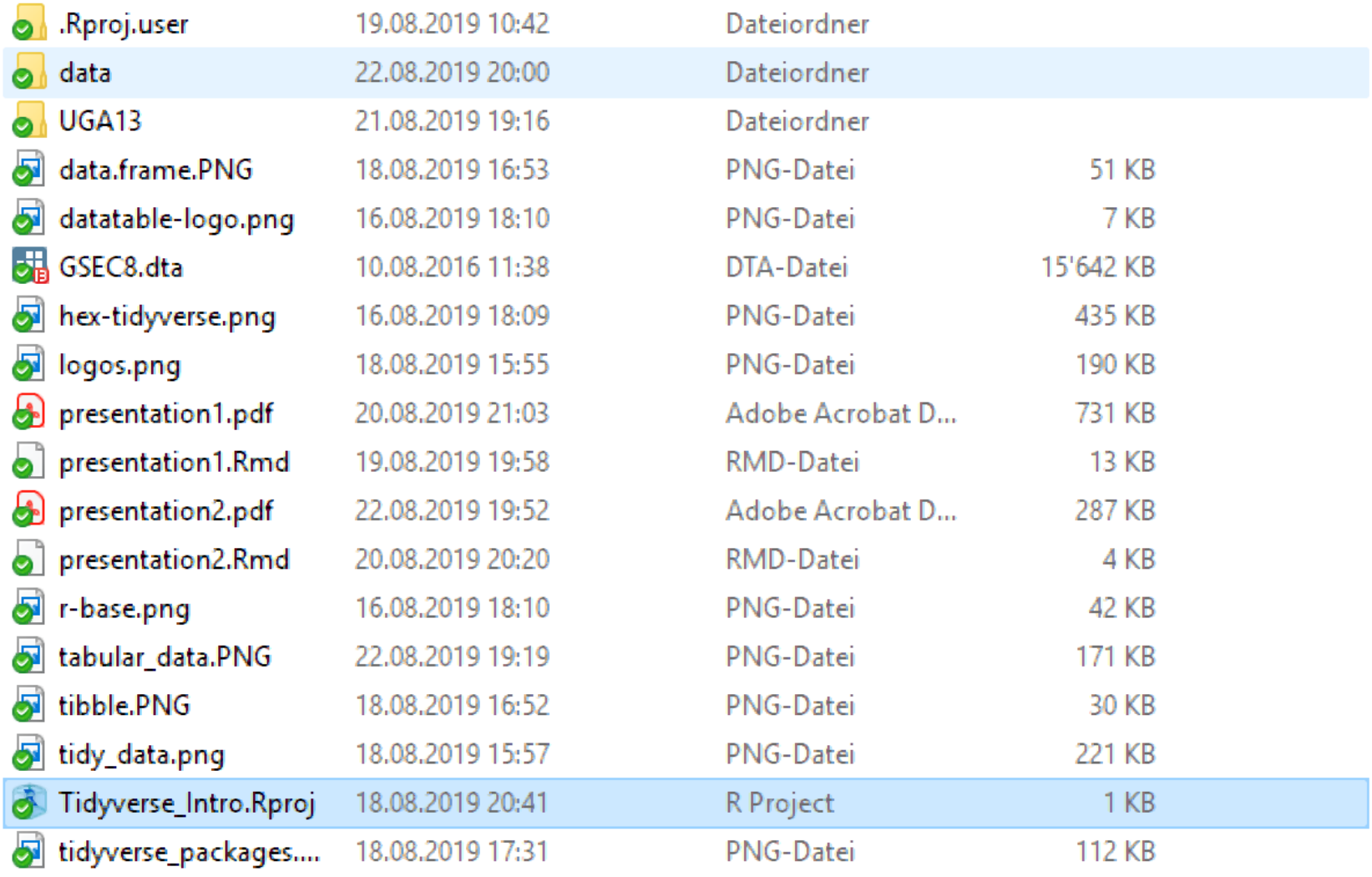
To determine your working directory:

```
getwd()
```

```
## [1] "D:/Onedrive/IT/R/Tidyverse_Intro"
```

When working with RStudio I highly recommend to work with RStudio-Projects. This will create a .Rproj-File and your working directory will automatically be set to the location of this .Rproj-File.

R-Studio Projects



.Rproj.user	19.08.2019 10:42	Dateiordner	
data	22.08.2019 20:00	Dateiordner	
UGA13	21.08.2019 19:16	Dateiordner	
data.frame.PNG	18.08.2019 16:53	PNG-Datei	51 KB
datatable-logo.png	16.08.2019 18:10	PNG-Datei	7 KB
GSEC8.dta	10.08.2016 11:38	DTA-Datei	15'642 KB
hex-tidyverse.png	16.08.2019 18:09	PNG-Datei	435 KB
logos.png	18.08.2019 15:55	PNG-Datei	190 KB
presentation1.pdf	20.08.2019 21:03	Adobe Acrobat D...	731 KB
presentation1.Rmd	19.08.2019 19:58	RMD-Datei	13 KB
presentation2.pdf	22.08.2019 19:52	Adobe Acrobat D...	287 KB
presentation2.Rmd	20.08.2019 20:20	RMD-Datei	4 KB
r-base.png	16.08.2019 18:10	PNG-Datei	42 KB
tabular_data.PNG	22.08.2019 19:19	PNG-Datei	171 KB
tibble.PNG	18.08.2019 16:52	PNG-Datei	30 KB
tidy_data.png	18.08.2019 15:57	PNG-Datei	221 KB
Tidyverse_Intro.Rproj	18.08.2019 20:41	R Project	1 KB
tidyverse_packages....	18.08.2019 17:31	PNG-Datei	112 KB

Data file is located in “data”.

Importing data correctly

All `readr::read_` verbs share the following arguments.

Most importantly:

- `col_names = TRUE`, use pre-existing column names or supply names yourself
- `na = c("", "NA")`, define how missing values are encoded

Sometimes useful:

- `skip = 0`, number of lines to skip before reading data
- `n_max = Inf`, maximum number of rows to read
- others, see `?readr::read_csv()`

For `read_delim()` specifically we also have the `delim = "|"` -argument

Fourth attempt

```
read_csv2("data/sw2.csv", na = ".")
```

```
## # A tibble: 87 x 10
##    name  height  mass hair_color skin_color eye_color
##    <chr>  <dbl> <dbl> <chr>      <chr>      <chr>
##  1 Luke~    172    77 blond      fair       blue
##  2 C-3PO    167    75 <NA>       gold       yellow
##  3 R2-D2     96    32 <NA>       white, bl~ red
##  4 Dart~    202   136 none       white      yellow
##  5 Leia~    150    49 brown      light      brown
##  6 Owen~    178   120 brown, gr~ light      blue
##  7 Beru~    165    75 brown      light      blue
##  8 R5-D4     97    32 <NA>       white, red red
##  9 Bigg~    183    84 black      light      brown
## 10 Obi-~    182    77 auburn, w~ fair       blue-gray
## # ... with 77 more rows, and 4 more variables:
## #   birth_year <dbl>, gender <chr>, homeworld <chr>,
## #   species <chr>
```

General procedure to import data

Checklist:

1. Make sure to select the correct delimiter/importing function.
2. Check if column-names are correctly imported.
3. Look whether there are any missing values, or whether there are odd values in your columns. Identify values that should be coded as `NA`.

Your turn:

- Click here for: Household Data
- Click here for: Individual Data

Challenge: Click on the Data and download the csv files from Github to your local computer. Open RStudio and try to import the datafiles.

Proposed solution

```
hh <-
  read_delim(paste0("https://raw.githubusercontent.com/",
                    "JFrischknecht/tidyverse-intro/",
                    "master/GSEC1.csv"),
             delim = "|", na = c("-99"))

ind <-
  read_delim(paste0("https://raw.githubusercontent.com/",
                    "JFrischknecht/tidyverse-intro/",
                    "master/GSEC8.csv"),
             delim = "|", na = c("-99"))
```

“Foreign” file types

Excel files with *readxl*:

- `readxl::read_excel()`
- `readxl::read_xlsx()` and `readxl::read_xls()`

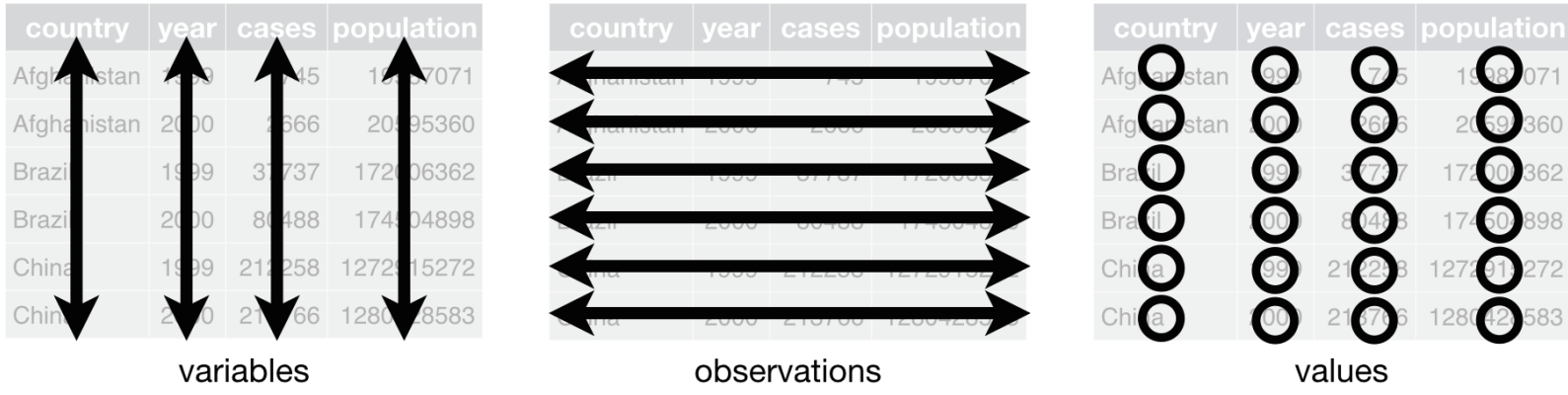
Other file types with *haven*:

- Stata-Files: `haven::read_dta()`
- SAS-Files: `haven::read_sas()`
- SPSS-Files: `haven::read_spss()`

Functions above have more or less similar arguments to *readr*-verbs.

Remember tidy data?

Our clean data should have the following structure:



This data is *wide*, i.e. all columns are variables.

Sometimes data is not tidy

```
## # A tibble: 4 x 4
##   country      year type           count
##   <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population 19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population 20595360
```

- The column "type" is a *key*-column. It identifies for each row what kind of value is stored in the column "count" (the *value*-column).
- This is *long* data, i.e. there is at least one *key*- and one *value*-column.

tidyr-verbs

We introduce two new data manipulation verbs from the *tidyr*-package. Similar to *dplyr*-verbs, the general form is:

$$f : (tibble, ...) \rightarrow tibble \tag{1}$$

- **`spread()`** - Move from long to wide format

```
spread(table2, key = type, value = count)
```

- **`gather()`** - Move from wide to long format

```
gather(table1, key = "key", value = "value", ...)
```

spread() - Move from long to wide format

```
table2
```

```
## # A tibble: 4 x 4
##    country      year type           count
##    <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population 19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population 20595360
```

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 2 x 4
##    country      year cases population
##    <chr>       <int> <int>      <int>
## 1 Afghanistan  1999   745   19987071
## 2 Afghanistan  2000  2666   20595360
```

gather() - Move from wide to long format

```
table1
```

```
## # A tibble: 2 x 4
##   country      year cases population
##   <chr>       <int> <int>      <int>
## 1 Afghanistan  1999   745   19987071
## 2 Afghanistan  2000  2666   20595360
```

```
gather(table1, key = "key", value = "value",
       cases, population)
```

```
## # A tibble: 4 x 4
##   country      year key            value
##   <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  2000 cases           2666
## 3 Afghanistan  1999 population 19987071
## 4 Afghanistan  2000 population 20595360
```

gather() - Move from wide to long format

gather() supports selection helpers similar to dplyr::select(), e.g.

```
gather(table1, key = "key", value = "value",
       3:4)
```

```
gather(table1, key = "key", value = "value",
       contains("case"), starts_with("pop"))
```

```
## # A tibble: 4 x 4
##    country      year key              value
##    <chr>       <int> <chr>            <int>
## 1 Afghanistan  1999 cases              745
## 2 Afghanistan  2000 cases             2666
## 3 Afghanistan  1999 population 19987071
## 4 Afghanistan  2000 population 20595360
```

Excercise:

1. Import data from Github
2. Column "hhid" is the household ID, column "PID" is the personal ID.
3. s8q36a , s8q36b , s8q36c , s8q36d , s8q36e , s8q36f , s8q36g are hours worked for each weekday
4. take individual dataset and construct individual id's (hint: use paste0())
5. stack the seven hours worked columns into a single one by using gather() or pivot_longer()
6. For each individual compute sum of hours worked across weekdays.
7. return tibble with single row per individual (hint: distinct()).

Proposed solution

```
ind %>%
  distinct(PID, .keep_all = T) %>%
  gather(key = weekday, value = hours,
         matches("h8q36")) %>%
  group_by(PID) %>%
  mutate(sum_hours = if_else(any(!is.na(hours)),
                             sum(hours, na.rm = T),
                             NA_real_)) %>%
  ungroup() %>%
  distinct(PID, .keep_all = T)
```

Joining *tibble's* with *dplyr*

Usually, not all the data we need is stored in a single *tibble*. This means we will have to combine, or *join* (*merge*) datasets by some kind of *key*-variable.

The most used *join's* in the *dplyr*-package are:

- **`left_join()`**
- **`inner_join()`**
- **`full_join()`**

See `?dplyr::join` for the lesser used *joins*.

left_join(x, y)

Return all rows from 'x' and all columns of 'x' and 'y':

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
left_join(members, instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

left_join(x, y)

```
?dplyr::left_join
```

```
left_join(x, y, by = NULL, copy = FALSE, suffix =
c(".x", ".y"), ...)
```

by: A character vector of variables to join by. If 'NULL', the default, will do a natural join, using all variables with common names across the two tables.

To join by different variables on x and y use a named vector. For example, 'by = c("a" = "b")' will match 'x.a' to 'y.b'.

- More restrictive code is in general less error prone:

```
left_join(members, instruments, by = c("name" = "name"))
left_join(members, instruments, by = "name") # The same
```

inner_join(x, y)

Return all rows in 'x' with matching rows in 'y' and all columns of 'x' and 'y':

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
inner_join(members, instruments, by = "name")
```

```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

full_join(x, y)

Return all rows and columns in 'x' and 'y'.

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
full_join(members, instruments, by = "name")
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

Your turn

We would like to link household level information to our individual level dataset.

1. Make sure to correctly import both datasets (see slides 14/15).
2. Join the household data to the individual data. What kind of join do you need? Can you identify a variable to join by?

Proposed solution

```
ind %>%
  left_join(hh, by = "HHID")
```

```
## # A tibble: 14,751 x 134
##    HHID    PID     h8q2 h8q3    h8q4  h8q5  h8q6  h8q7 
##    <chr>   <chr>  <dbl> <chr>  <dbl> <dbl> <dbl> <dbl> <
##  1 H00101~ P0010~     1 ""         2     2     1     1
##  2 H00101~ P0010~     2 P0010~     2     2     1     1
##  3 H00101~ P0010~     2 P0010~     2     2     1     1
##  4 H00102~ P0010~     1 ""         2     2     2     2
##  5 H00102~ P0010~     2 P0010~     2     2     2     2
##  6 H00102~ P0010~    NA ""        NA    NA    NA    NA
##  7 H00104~ P0010~     1 ""         2     2     1     1
##  8 H00104~ P0010~     1 ""         2     2     1     1
##  9 H00104~ P0010~     2 P0010~     2     2     2     2
## 10 H00110~ P0011~     1 ""         2     2     1     1
## # ... with 14,741 more rows, and 125 more variables:
## #   h8q9 <dbl>, h8q10 <dbl>, h8q11 <dbl>, h8q12 <dbl>,
```

Basic graphs with *ggplot2*

Grammar of **G**raphics:

A statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars) (Wickham, 2016).

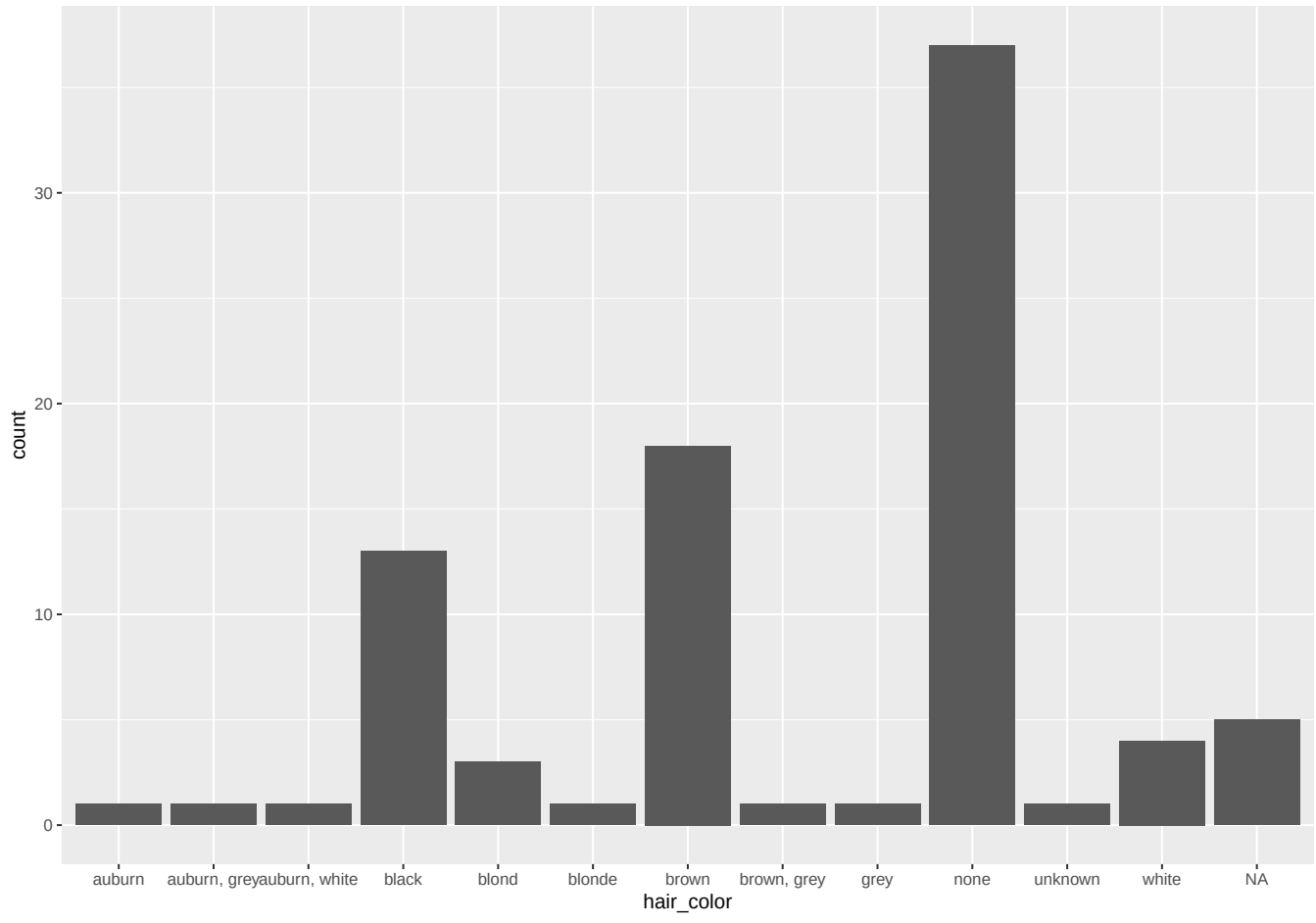
- Aesthetics `aes()`
- Geometric objects `geom_()`

A plot is built by layering geometric objects on top of each other, use + to add an additional layer.

```
ggplot(dataset, aes(x = x, y = y)) +
  geom_point(aes(colour = y)) + # Inherits `aes()` from `ggplot()`
  geom_line(colour = "black") # Not the same as `aes(color = )`
```

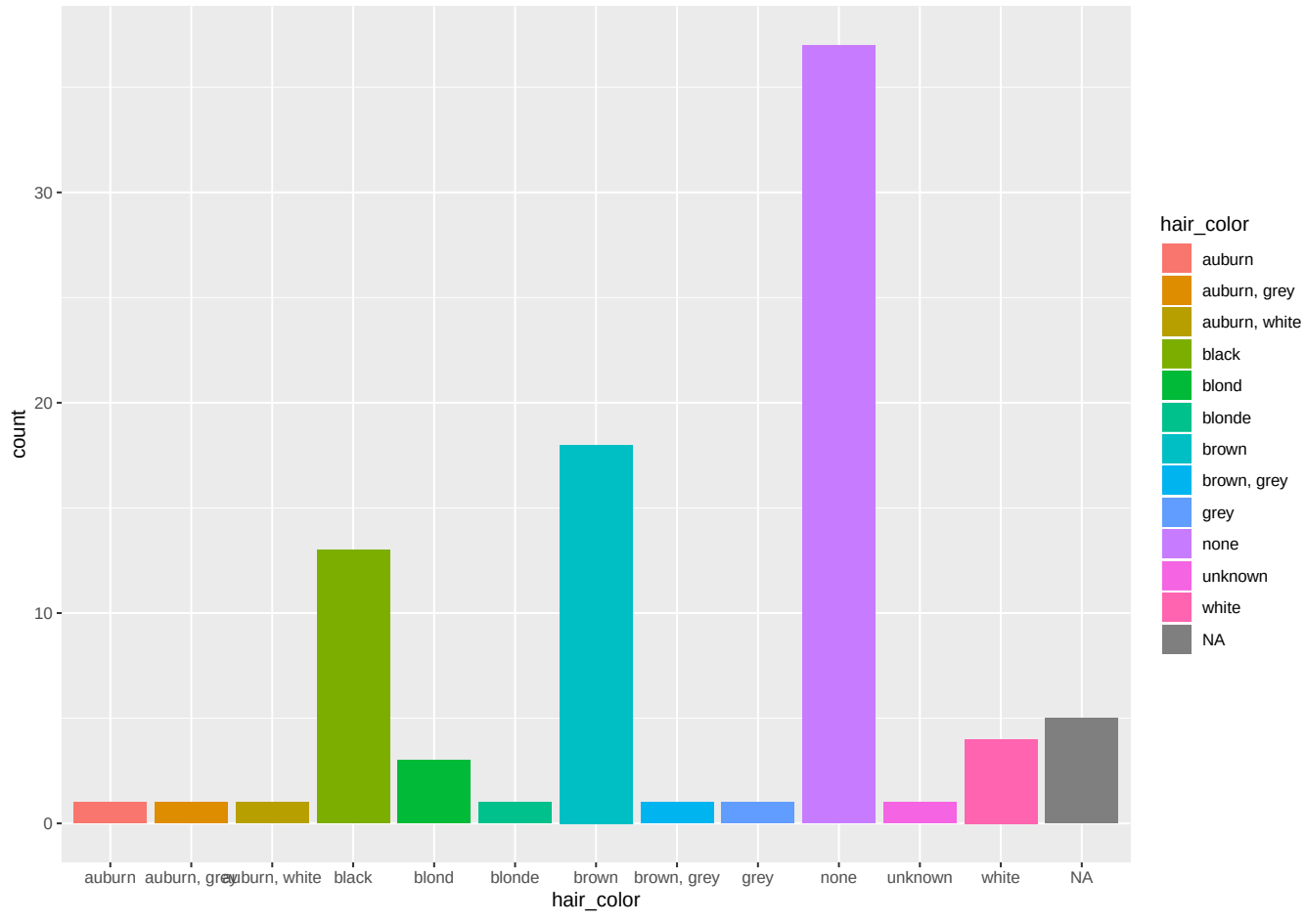
Bar-Plot with `geom_bar()`

```
starwars %>%
  ggplot(aes(x = hair_color)) +
  geom_bar()
```



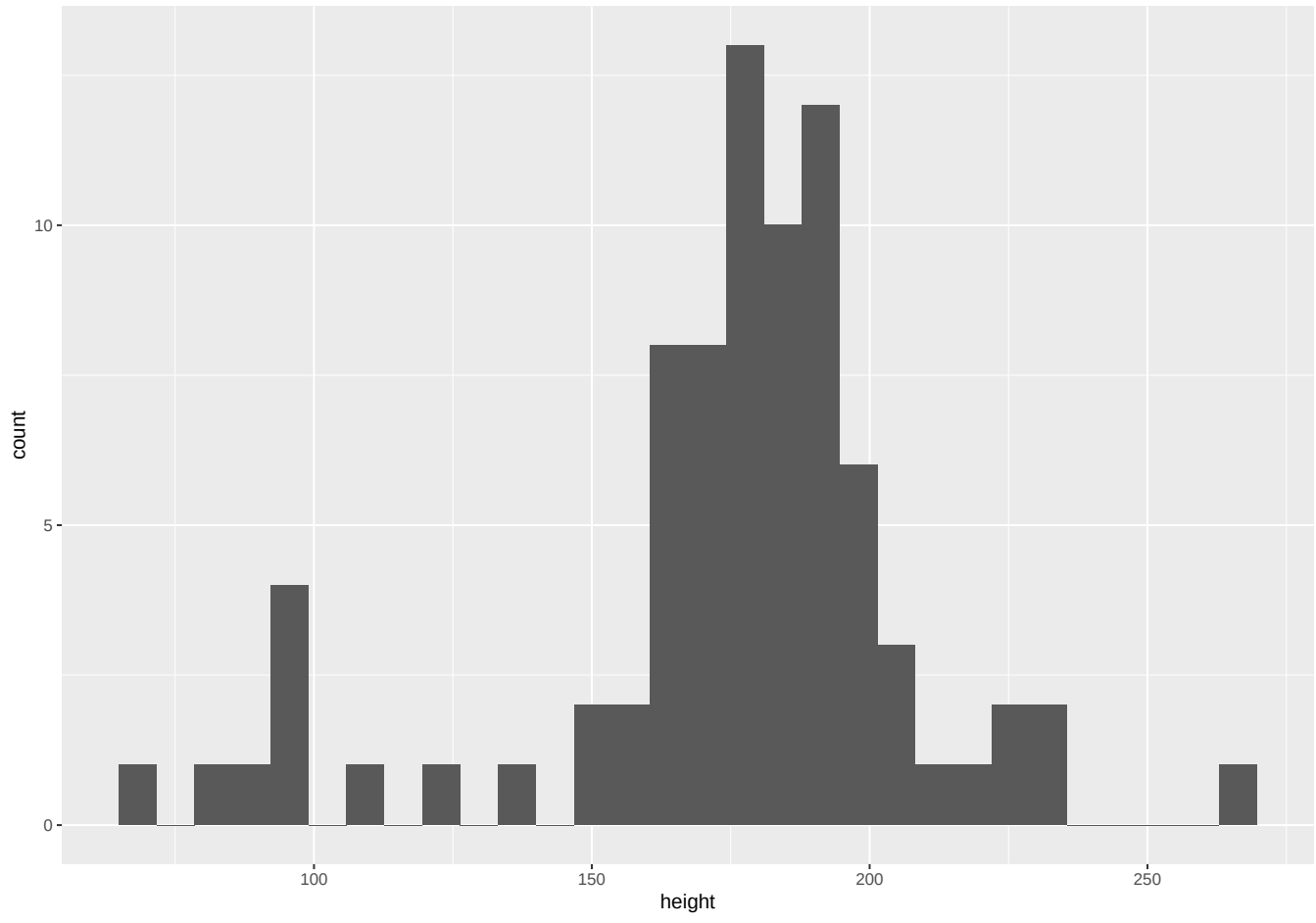
Add an additional aesthetic in `aes()`

```
starwars %>%
  ggplot(aes(x = hair_color, fill = hair_color)) +
  geom_bar()
```



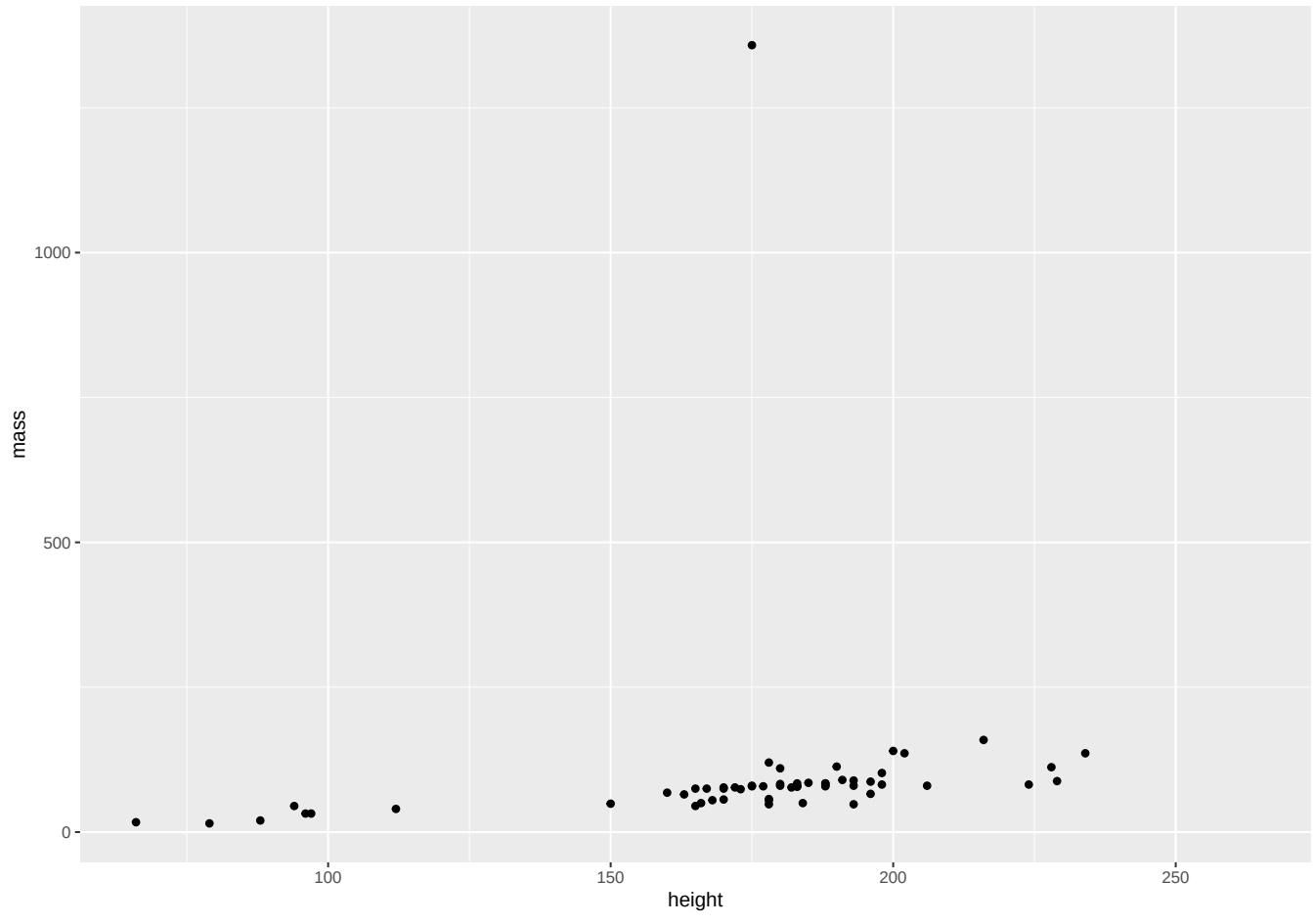
Histogram with `geom_histogram()`

```
starwars %>%
  ggplot(aes(x = height)) +
  geom_histogram()
```



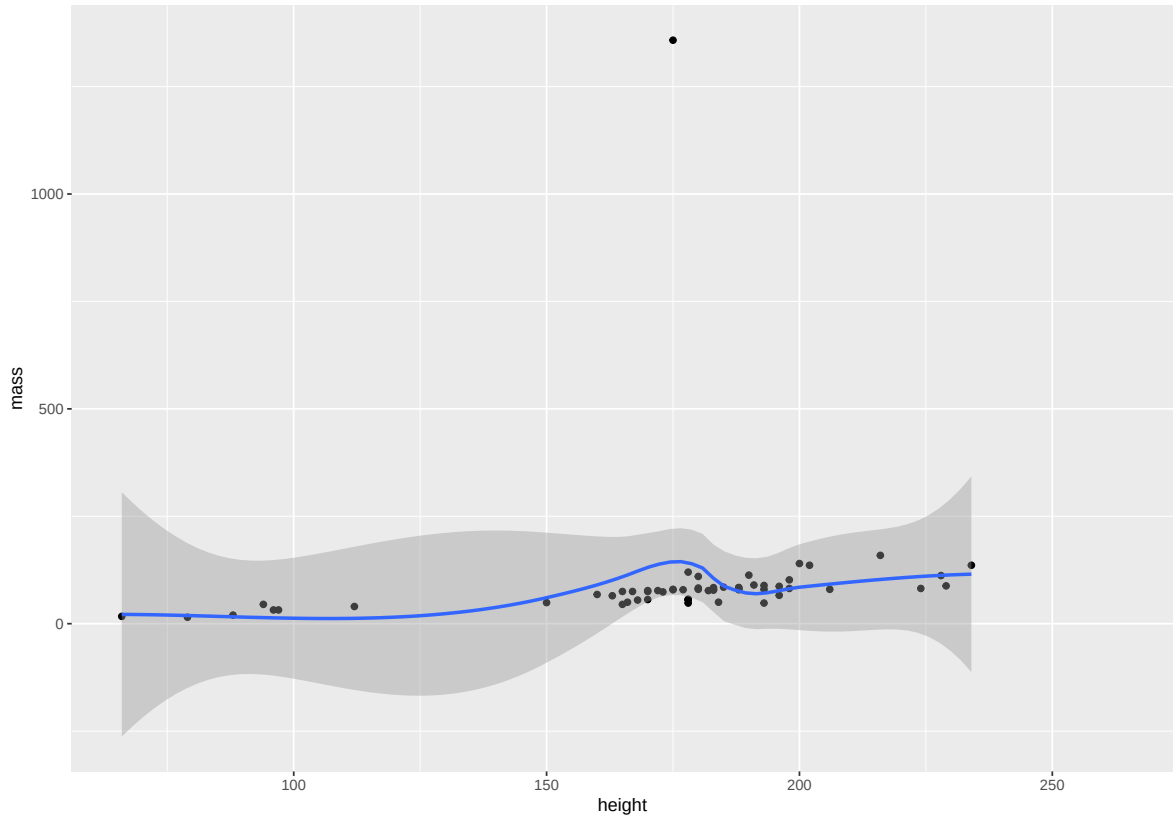
Scatter-Plot with `geom_point()`

```
starwars %>%
  ggplot(aes(x = height, y = mass)) +
  geom_point()
```



Adding a trend line with `geom_smooth()`

```
starwars %>%
  ggplot(aes(x = height, y = mass)) +
  geom_point() +
  geom_smooth()
```



Your turn:

Import the following cleaned dataset from:

https://raw.githubusercontent.com/JFrischknecht/tidyverse-intro/master/clean_data.csv

Create the following plots:

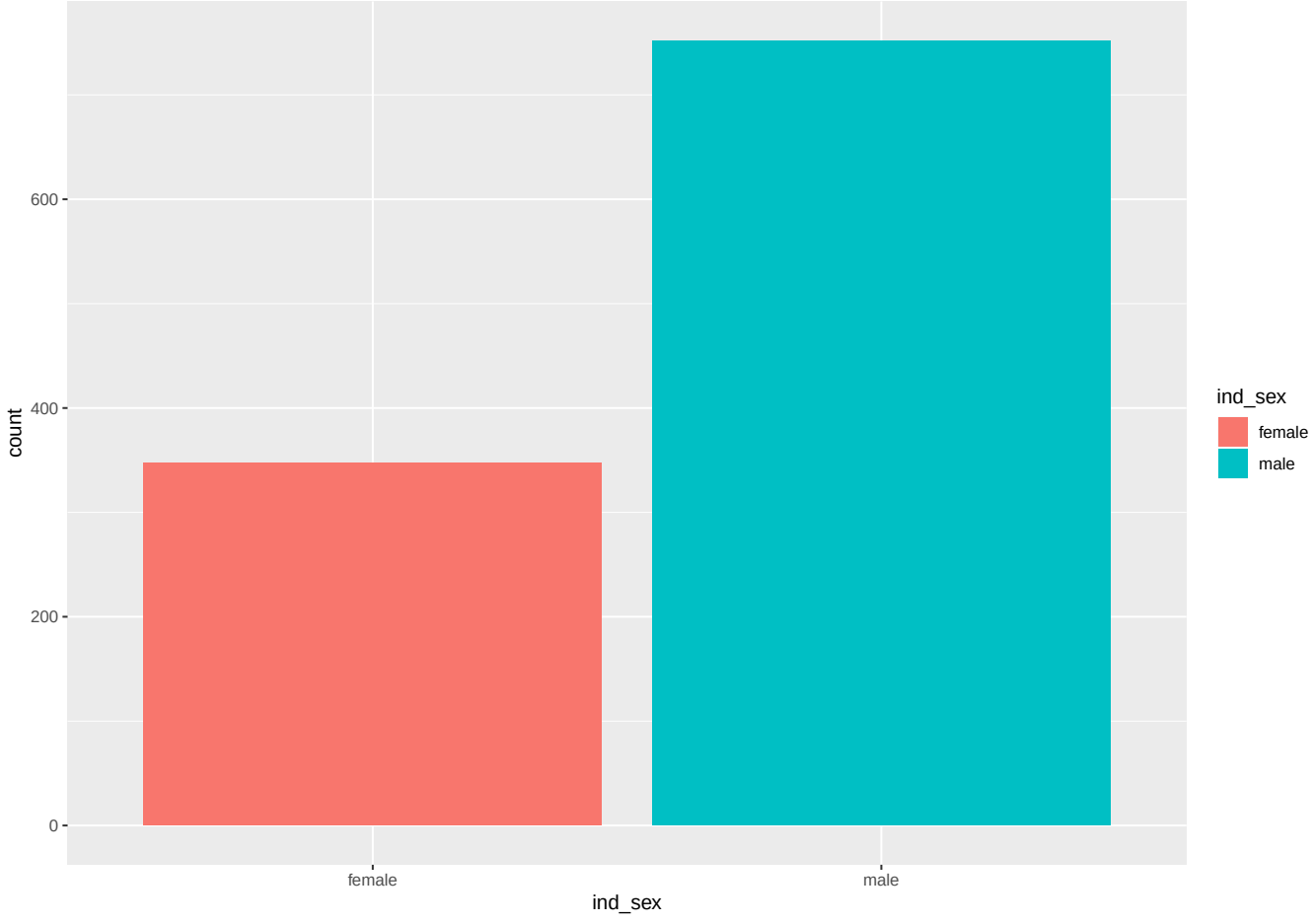
1. Distribution of gender.
2. Distribution of monthly wages.
3. Returns to education (in terms of wages).

Background information:

- "ind_sex" - Gender of respondents
- "ind_main_job_wage_month" - Monthly wage
- "ind_education_years" - Years of schooling

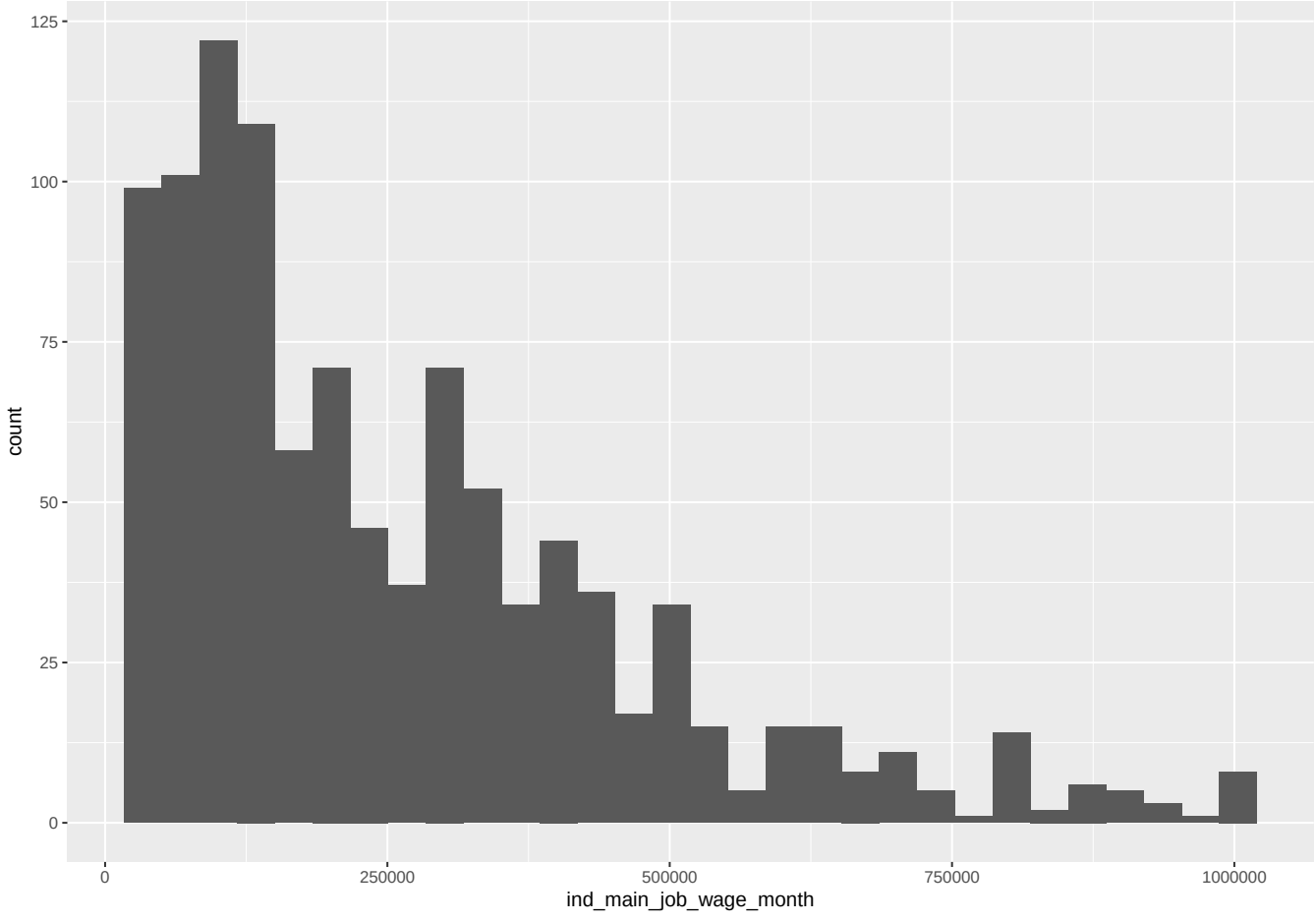
Proposed solution (1)

```
clean_data %>%
  ggplot(aes(x = ind_sex, fill = ind_sex)) +
  geom_bar()
```



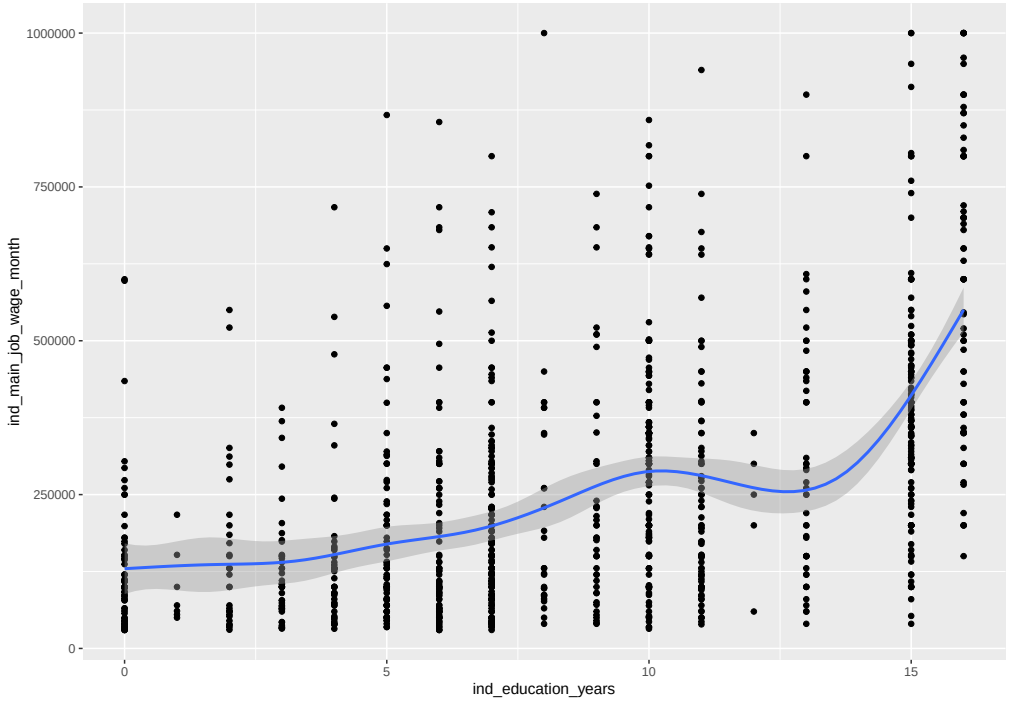
Proposed solution (2)

```
clean_data %>%
  ggplot(aes(x = ind_main_job_wage_month)) +
  geom_histogram()
```

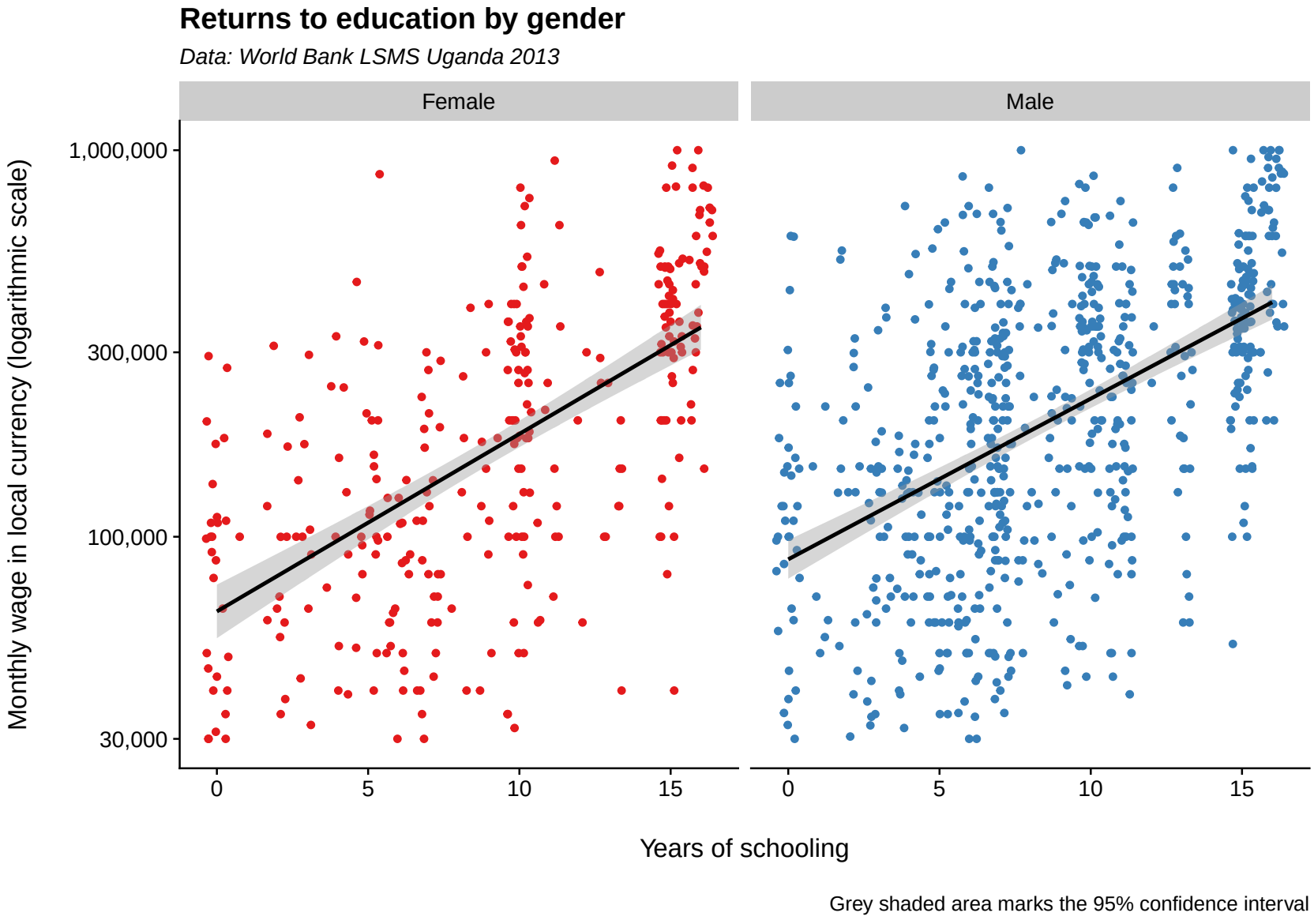


Proposed solution (3)

```
clean_data %>%
  ggplot(aes(x = ind_education_years,
             y = ind_main_job_wage_month)) +
  geom_point() +
  geom_smooth()
```



A quick glance at what you can do with *ggplot2*



And the underlying code . . .

```
clean_data %>%
  mutate(ind_sex = stringr::str_to_title(ind_sex)) %>%
  ggplot(aes(x = ind_education_years,
             y = ind_main_job_wage_month,
             colour = ind_sex)) +
  geom_jitter() +
  geom_smooth(colour = "black", method = "lm", se = 0.95) +
  scale_y_log10(labels = scales::comma) +
  scale_colour_brewer(type = "qual", palette = "Set1") +
  facet_wrap(. ~ ind_sex) +
  labs(title = "Returns to education by gender",
       subtitle = "Data: World Bank LSMS Uganda 2013",
       y = paste("Monthly wage in local currency",
                 "(logarithmic scale)"),
       x = "Years of schooling",
       caption = paste("Grey shaded area marks the 95%",
                       "confidence interval")) +
  guides(colour = F) +
  custom_theme
```

Recommended ressources

Importing/tidying data:

- "R for Data Science" (Wickham, Grolemund, 2018), free online version: https://r4ds.had.co.nz/
- Cheatsheets for the *tidyverse* packages: https://www.rstudio.com/resources/cheatsheets/

Plotting:

- "R Graphics Cookbook" (Chang, 2012)
- "ggplot2, Elegant Graphics for Data Analysis", (Wickham, 2016)

Main Take-aways

- readr, foreign, haven for importing
- R Projects
- pivot_longer(), pivot_wider()
- pipes (%>%)
- group_by(), mutate(), summarise()
- join
- ggplots
- case_when(), if_else(), distinct(), unique(), is.na(), filter(), write.csv()