Lab 1: Intro to R and data analysis

Practice session covering topics discussed in Lecture 1

M. Chiara Mimmi, Ph.D. | Università degli Studi di Pavia

July 25, 2024

GOAL OF TODAY'S PRACTICE SESSION

Motivate the choice of learning/using R for scientific quantitative analysis, and lay out some fundamental concepts in biostatistics with concrete R coding examples.

Lecture 1: topics

- **Introduction to R and R-studio**
  - Why R?
  - Principles of reproducible analysis with R + RStudio
- **R objects, functions, packages**
- **Understanding different types of variables**
  - Principles of "tidy data"
- **Descriptive statistics**
  - Measures of central tendency, measures of variability (or spread), and frequency distribution
- **Visual data exploration**
  - {`ggplot2`}

INTRO TO R AND RSTUDIO

R version

If you have previously installed R on your machine, you can check which version you are running by executing this command in R:

```r
# check your R version
R.Version()
```

```
$platform
[1] "x86_64-apple-darwin17.0"

$arch
[1] "x86_64"

$os
[1] "darwin17.0"

$system
[1] "x86_64, darwin17.0"

$status
[1] ""

$major
[1] "4"

$minor
[1] "2.2"
```

```r
# or just
#R.version.string
```

Install R

R is available for free for Windows, GNU/Linux, and macOS.

- To install **R**, go to this link **https://cloud.r-project.org/**. The latest available release is **R 4.3.3 "Angel Food Cake" released on 2024-02/29**, but any (fairly recent) version will do.

Install RStudio IDE

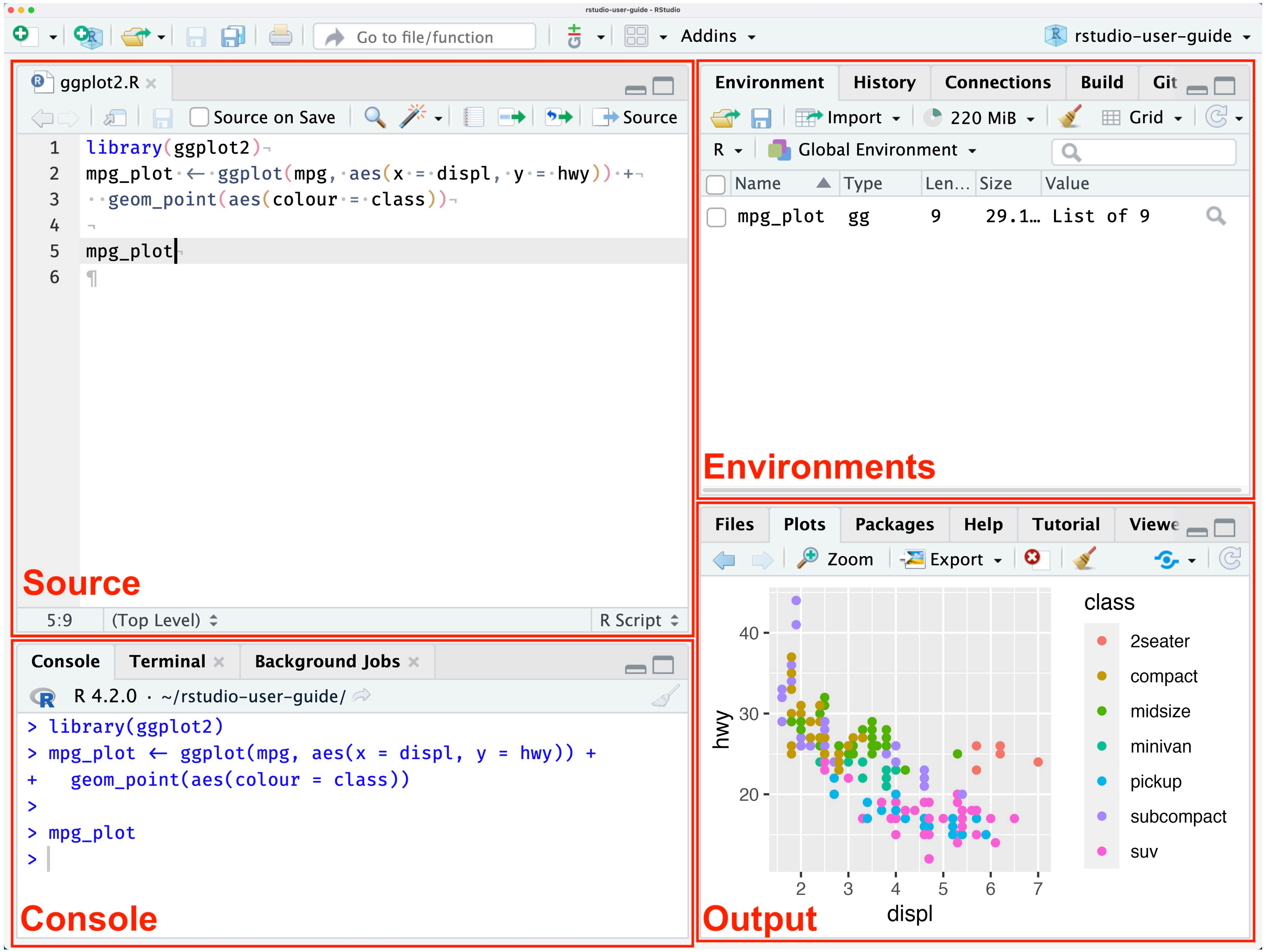
RStudio Desktop is an Integrated Development Editor (IDE), basically a graphical interface wrapping and interfacing R (which needs to be installed first).

Besides RStudio, R (which is a command line driven program) can be executed:

- via its native interface (**R GUI**)
- from many other code editors, like **VS Code**, **Sublime Text**, **Jupyter Notebook**

To install **RStudio**, go to this link **https://posit.co/download/rstudio-desktop/**. The free-version contains everything you need.

Use RStudio IDE



RStudio Pane Layout Source: Posit's RStudio User Guide

Creating an R Project [in Rstudio]

An **R Project** will keep all the files associated with a project (including invisible ones!) organized together – input data, R scripts, analytical results, figures. Besides being common practice, this has the advantage of implicitly setting the “working directory”, which is incredibly important when you need to load or output files, specifying their file path.

In Figure 1 you can see how easy it is just following RStudio prompts:

- Create a new directory for each project
- Select parent folder

Creating an R Project [in Rstudio] (cont.)

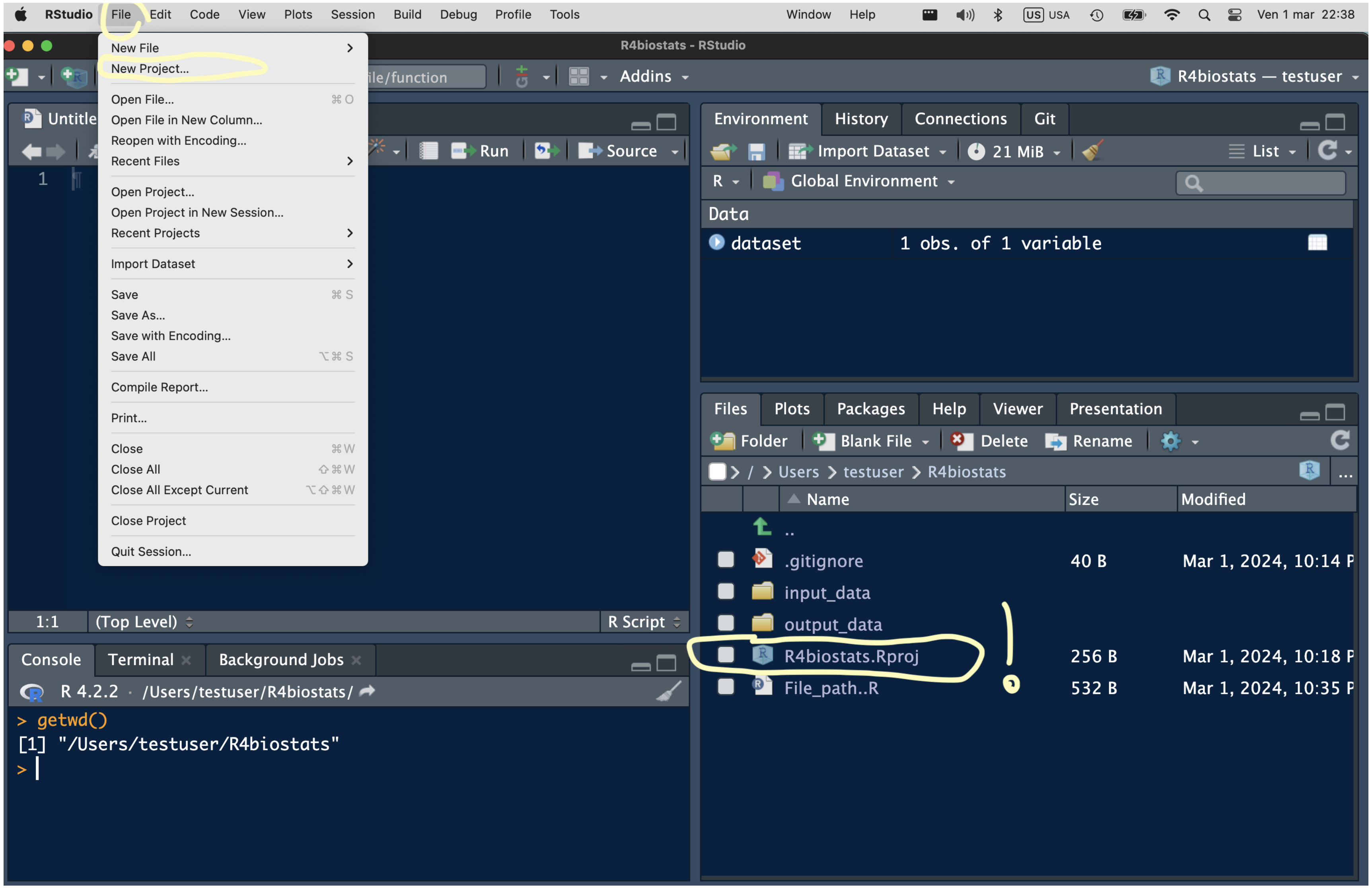


Figure 1: Creating an R project

Install R packages from CRAN (stable version)

An **R package*** is a shareable bundle of functions. Besides the basic built-in functions already contained in the program (i.e. the `base` package), many useful R functions come in free libraries of code (or *packages*) written by R's users. You can find them in different repositories:

To install a package use `utils` function `install.packages("package_name)`

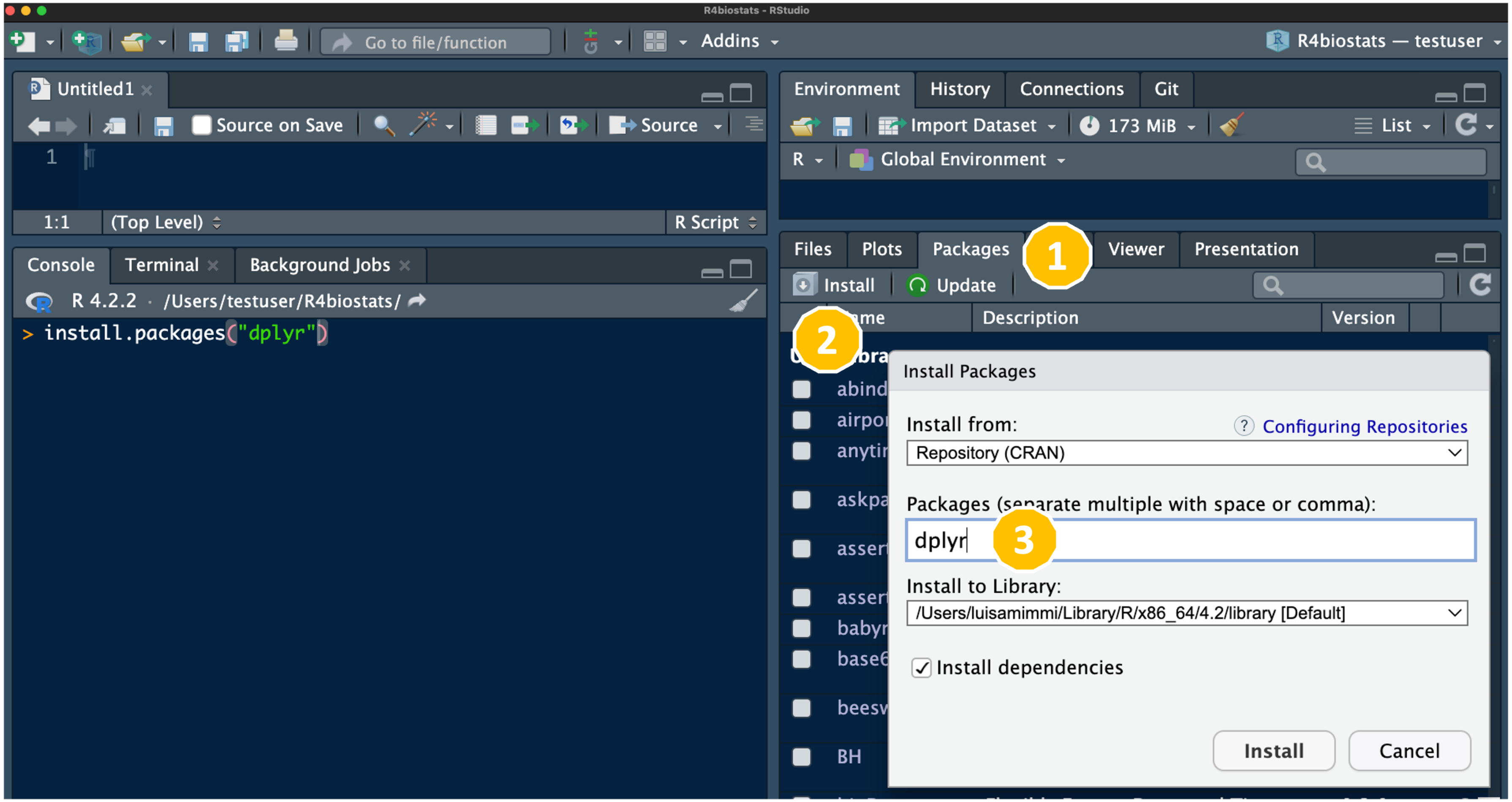
```
# Installing (ONLY the 1st time)
utils::install.packages('here')

# OR (same)
install.packages('here')
```

Here you are actually using a ***function*** (`install.packages`) of a pre-installed ***package*** (`utils`) using the syntax `packagename::function_name`. This prevents any ambiguity in case of duplicate function name… also helps you see

Install R packages RStudio pane

In alternative, you can install/update packages using the `Packages` tab on the lower right pane of RStudio.



Screenshot Install/Update pckgs from RStudio

Install R packages from GitHub (testing version

Use `install_github` from the package `devtools`.

EXAMPLE: let's install a little package `paint` (which colors the structure of dataset when printing).

Code

```r
# Installing devtools (ONLY the 1st time)
utils::install.packages('devtools')

# Installing paint from GitHub
library(devtools)
devtools::install_github("MilesMcBain/paint

# test paint out
library(paint)
```

Output {paint} function

```r
# Structure of a data.frame
paint::paint(mtcars)
```

```
> # it will show me the structure of a data.frame like this...
> paint(mtcars)
data.frame [32, 11]
mpg  dbl 21 21 22.8 21.4 18.7 18.1
cyl  dbl 6 6 4 6 8 6
disp dbl 160 160 108 258 360 225
hp   dbl 110 110 93 110 175 105
drat dbl 3.9 3.9 3.85 3.08 3.15 2.76
wt   dbl 2.62 2.875 2.32 3.215 3.44 3.46
qsec dbl 16.46 17.02 18.61 19.44 17.02 20.22
vs   dbl 0 0 1 1 0 1
am   dbl 1 1 1 0 0 0
gear dbl 4 4 4 3 3 3
carb dbl 4 4 1 1 2 1
```


[After `devtools::install_github("MilesMcBain/paint")`, R asks me if I want to update related packages...]

Use R Packages

- We will be using {base} & {utils} (pre-installed and pre-loaded)
- We will also use the packages below (specifying `package::function` for clarity).

```r
# Load them for this R session
library(here)      # tools find your project's files, based on working directory
library(janitor)   # tools for examining and cleaning data
library(skimr)     # tools for summary statistics
library(dplyr)     # {tidyverse} tools for manipulating and summarising tidy data
library(ggplot2)   # {tidyverse} tools for plotting
library(forcats)   # {tidyverse} tool for handling factors
library(ggridges)  # alternative to plot density functions
library(fs)        # file/directory interactions
```

Help on R package/function

To inquire about a package and/or its functions, you can again write in your console `?package_name` or `??package_name` and RStudio will open up the `Help` tab in the lower right pane.

```
# Opening Help page on package/function
?here

??here
```

File paths logistics

It is never good practice to “hard code” the file’s *absolute path*: most likely it will break your code as soon as you (or someone else) need to run it on a different computer, let alone within a different OS.

Let’s look at this example code using function `readr::read_csv()` (which reads a `*.csv` data file into the R workspace)

```
# [NOT REPRODUCIBLE] hard coding your file path  -----------------------

# File path on Mac:
dataset <- readr::read_csv("/Users/testuser/R4biostats/input_data/dataset.csv")
# Same file path on Windows:
dataset <- readr::read_csv("C:\Users\testuser\R4biostats\input_data\dataset.csv")
```

🙄 …it won’t work on any other computer since it won’t have that same file structure!

(Reproducible) file paths with `here` (in Rstudio)

The `here` package lets you reference file paths in a **reproducible** manner (anchored on the R Project's folder as the **root**).

Where is my Working Directory?

```
here::here()
```

You should get: **"/Users/YourName/RProj_Dir"**

Now, you can embed `here(dir,subdir)` specifications in other functions.
For example, create sub-directories (for saving input data and output data) with the `fs` package

```
## --- [check the function documentation]
?fs::dir_create
# with `here` I simply add subfolder names relative to my wd
fs::dir_create(here("practice", "data","data_input"))
# ...and a subfolder to put output files at the end
fs::dir_create(here("practice", "data","data_output"))

## --- [if I need to remove it (I have them already)]
fs::dir_delete(here("practice", "data"))
```

R OBJECTS, FUNCTIONS, PACKAGES

Importing data into R workspace

We are using real data provided by Thabtah,Fadi. (2017). Autism Screening Adult. UCI Machine Learning Repository. https://doi.org/10.24432/C5F019

We use `utils::read.csv` to load a csv file

```
?read.csv # to learn about function and arguments
```

Option 1: Importing from a url

```r
autism_data_url <- read.csv(
  file = "https://raw.githubusercontent.com/Sydney-Informatics-Hub/lessonbmc/gh-pages/_epis
  header = TRUE, # 1st line is the name of the variables
  sep = ",", # which is the field separator character.
  na.strings = c("?") # specific values R should interpret as NA
)
```

Option 2: Importing from my folder (if you previously downloaded the file)

- `here` lets me specify the complete path of the destination folder

> **Tip**
>
> Make sure to match your own folder structure the file path `here(...)`!

```r
# Check my working directory location
# here::here()

# Use `here` in specifying all the subfolders AFTER the working directory
autism_data_file <- read.csv(
  file = here("practice", "data_input", "01_datasets", "autism_data.csv"),
  header = TRUE, # 1st line is the name of the variables
  sep = ",", # which is the field separator character.
  na.strings = c("?"),# specific values R should interpret as NA
  row.names = NULL)
```

DATA OBSERVATION & MANIPULATION

Viewing the dataset and variables

```r
View(autism_data_file)
```

- Or click on the object in Environment tab (upper right pane of RStudio)

```r
# What data type is this data?
class(autism_data_file)
```

```
[1] "data.frame"
```

```r
# What variables are included in this dataset?
base::colnames(autism_data_file)
```

```
 [1] "id"              "A1_Score"        "A2_Score"        "A3_Score"       
 [5] "A4_Score"        "A5_Score"        "A6_Score"        "A7_Score"       
 [9] "A8_Score"        "A9_Score"        "A10_Score"       "age"            
[13] "gender"          "ethnicity"       "jaundice"        "autism"         
[17] "contry_of_res"   "used_app_before" "result"          "age_desc"       
[21] "relation"        "Class.ASD"      
```

- Notice the var name formatting inconsistency: `Class.ASD`

Manipulate / clean the dataframe

I want consistent name formatting for variables: no "`.`", only "`_`" separator. So, I use a very handy function `clean_names` from the `janitor` package

```r
autism_data <- janitor::clean_names(autism_data_file,
                                    case = "none")
# check change
colnames(autism_data)
```

```
 [1] "id"              "A1_Score"        "A2_Score"        "A3_Score"       
 [5] "A4_Score"        "A5_Score"        "A6_Score"        "A7_Score"       
 [9] "A8_Score"        "A9_Score"        "A10_Score"       "age"            
[13] "gender"          "ethnicity"       "jaundice"        "autism"         
[17] "contry_of_res"   "used_app_before" "result"          "age_desc"       
[21] "relation"        "Class_ASD"      
```

```r
dim(autism_data)
```

```
[1] 704  22
```

- By default `clean_names` renames cols into **"snake"** format (i.e. "abc_xyz")
- The option `case` is for capitalization preferences
  - `case = "none"` leaves the case as is, but only uses "`_`" separator

Isolate a variable (column)

You can use the $ sign to extract a variable (column name)

```
autism_data$id
autism_data$A1_Score
autism_data$gender
autism_data$autism
```

Add a new column

(I prefer to rename the dataframe when I make changes)

```r
# rename dataframe
autism_pids <- autism_data
```

Create a **new column**, using `paste` (function to concatenate strings)

```r
# create a new column
autism_pids$pids <- paste("PatientID_" , autism_data$id, sep = "")
```

Check results:

```r
# check change in df structure
base::colnames(autism_pids)
```

```
 [1] "id"              "A1_Score"        "A2_Score"        "A3_Score"
 [5] "A4_Score"        "A5_Score"        "A6_Score"        "A7_Score"
 [9] "A8_Score"        "A9_Score"        "A10_Score"       "age"
[13] "gender"          "ethnicity"       "jaundice"        "autism"
[17] "contry_of_res"   "used_app_before" "result"          "age_desc"
[21] "relation"        "Class_ASD"       "pids"
```

```r
dim(autism_data)
```

```
[1] 704  22
```

```r
dim(autism_pids)
```

```
[1] 704  23
```

(optional) Clean up my workspace

```r
# what do I have in the environment?
ls()
```

```
[1] "autism_data"      "autism_data_file" "autism_pids"
```

```r
# remove all EXCEPT for "autism_pids"
rm("autism_data", "autism_data_file", "autism_data_url" )
```

(Warning: mind that after `rm()`, you will not have these objects in your workspace anymore.)

Different ways to select rows &/or columns (from `base`)

Option 1 Extract cols with $

- (head only specifies to take the first 6 observations of the dataset)

```r
# With the `$` sign I extract a variable (column name)
head(autism_pids$id)
```

```
[1] 1 2 3 4 5 6
```

```r
head(autism_pids$pids)
```

```
[1] "PatientID_1" "PatientID_2" "PatientID_3" "PatientID_4" "PatientID_5"
[6] "PatientID_6"
```

```r
head(autism_pids$A1_Score)
```

```
[1] 1 1 1 1 1 1
```

```r
head(autism_pids$ethnicity)
```

```
[1] "White-European" "Latino"         "Latino"         "White-European"
[5] NA               "Others"
```

Option 2a Extract cols with `[,#col]`

- This is called "indexing"

```r
# Indexing to pick `[ , #col]`
head(autism_pids[ ,1] )# empty rows means all
```

```
[1] 1 2 3 4 5 6
```

```r
head(autism_pids[ ,23])
```

```
[1] "PatientID_1" "PatientID_2" "PatientID_3" "PatientID_4" "PatientID_5"
[6] "PatientID_6"
```

```r
head(autism_pids[ ,2])
```

```
[1] 1 1 1 1 1 1
```

```r
head(autism_pids[ ,14])
```

```
[1] "White-European" "Latino"         "Latino"         "White-European"
[5] NA               "Others"
```

Option 2b Extract rows with `[#row, ]`

```r
# Indexing to pick `[#row, ]`
head(autism_pids[1 , ] ) # empty cols means all
```

```
  id A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
1  1        1        1        1        1        0        0        1        1
  A9_Score A10_Score age gender      ethnicity jaundice autism contry_of_res
1        0         0  26      f White-European       no     no United States
  used_app_before result    age_desc relation Class_ASD        pids
1              no      6 18 and more     Self        NO PatientID_1
```

```r
head(autism_pids[50,])
```

```
   id A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
50 50        1        1        0        0        0        1        1        1
   A9_Score A10_Score age gender ethnicity jaundice autism contry_of_res
50        0         1  30      f     Asian       no     no    Bangladesh
   used_app_before result    age_desc relation Class_ASD         pids
50              no      6 18 and more     Self        NO PatientID_50
```

```r
head(autism_pids[25:26 ,])
```

```
   id A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
25 25        1        1        1        1        0        0        0        1
26 26        0        1        1        0        0        0        0        1
   A9_Score A10_Score age gender ethnicity jaundice autism contry_of_res
25        0         0  43      m      <NA>       no     no       Lebanon
26        0         0  24      f      <NA>      yes     no   Afghanistan
   used_app_before result    age_desc relation Class_ASD         pids
25              no      5 18 and more     <NA>        NO PatientID_25
26              no      3 18 and more     <NA>        NO PatientID_26
```

Option 3 Extract rows & cols with `[#row,#col`

```r
# Indexing to pick `[#row, #col]`
autism_pids[1:3,1]
```

```
[1] 1 2 3
```

```r
autism_pids[1:3,23]
```

```
[1] "PatientID_1" "PatientID_2" "PatientID_3"
```

```r
autism_pids[1:3,2]
```

```
[1] 1 1 1
```

```r
autism_pids[1:3,14]
```

```
[1] "White-European" "Latino"         "Latino"
```

What are the data types of the variables?

Option 1 using base functions

- on the whole dataset

```r
# What are the data types of the variables? ---------------------------------
str(autism_pids) # integer and character
```

```
'data.frame':	704 obs. of  23 variables:
 $ id             : int  1 2 3 4 5 6 7 8 9 10 ...
 $ A1_Score       : int  1 1 1 1 1 1 0 1 1 1 ...
 $ A2_Score       : int  1 1 1 1 0 1 1 1 1 1 ...
 $ A3_Score       : int  1 0 0 0 0 1 0 1 0 1 ...
 $ A4_Score       : int  1 1 1 1 0 1 0 1 0 1 ...
 $ A5_Score       : int  0 0 1 0 0 1 0 0 1 0 ...
 $ A6_Score       : int  0 0 0 0 0 0 0 0 0 1 ...
 $ A7_Score       : int  1 0 1 1 0 1 0 0 0 1 ...
 $ A8_Score       : int  1 1 1 1 1 1 1 0 1 1 ...
 $ A9_Score       : int  0 0 1 0 0 1 0 1 1 1 ...
 $ A10_Score      : int  0 1 1 1 0 1 0 0 1 0 ...
 $ age            : int  26 24 27 35 40 36 17 64 29 17 ...
 $ gender         : chr  "f" "m" "m" "f" ...
 $ ethnicity      : chr  "White-European" "Latino" "Latino" "White-European" ...
 $ jaundice       : chr  "no" "no" "yes" "no" ...
 $ autism         : chr  "no" "yes" "yes" "yes" ...
 $ contry_of_res  : chr  "United States" "Brazil" "Spain" "United States" ...
 $ used_app_before: chr  "no" "no" "no" "no" ...
 $ result         : int  6 5 8 6 2 9 2 5 6 8 ...
```

Option 1 using base functions (cont.)

- on specific columns

```r
# What values can the variables take? ---------------------------------
summary(autism_pids$pids)
```

```
   Length     Class      Mode 
      704 character character 
```

```r
length(unique(autism_pids$pids)) # N unique values
```

```
[1] 704
```

```r
sum(is.na(autism_pids$pids)) # N missing values
```

```
[1] 0
```

```r
summary(autism_pids$ethnicity)
```

```
   Length     Class      Mode 
      704 character character 
```

```r
length(unique(autism_pids$ethnicity)) # N unique values
```

```
[1] 12
```

```r
sum(is.na(autism_pids$ethnicity)) # N missing values
```

```
[1] 95
```

Option 2 using `skimr` function `skim`

- on specific columns

```r
autism_pids %>%
  skimr::skim(pids, ethnicity) %>%
  dplyr::select(#skim_variable,
                skim_type,
                complete_rate,
                n_missing,
                character.n_unique)
```

```
# A tibble: 2 × 4
  skim_type complete_rate n_missing character.n_unique
  <chr>             <dbl>     <int>              <int>
1 character         1             0                704
2 character         0.865        95                 11
```

Option 2 using `skimr` function `skim` (cont.)

- on the whole dataset

```r
autism_pids %>%
  skimr::skim()
```

```
── Variable type: character ────────────────────────────────────────────────────
   skim_variable   n_missing complete_rate min max empty n_unique whitespace
 1 gender                  0         1       1   1     0        2          0
 2 ethnicity              95         0.865   5  15     0       11          0
 3 jaundice                0         1       2   3     0        2          0
 4 autism                  0         1       2   3     0        2          0
 5 contry_of_res           0         1       4  20     0       67          0
 6 used_app_before         0         1       2   3     0        2          0
 7 age_desc                0         1      11  11     0        1          0
 8 relation               95         0.865   4  24     0        5          0
 9 Class_ASD               0         1       2   3     0        2          0
10 pids                    0         1      11  13     0      704          0

── Variable type: numeric ──────────────────────────────────────────────────────
   skim_variable n_missing complete_rate    mean      sd p0  p25  p50  p75 p100
 1 id                    0             1   352.    203.   1 177. 352. 528.  704
 2 A1_Score              0             1     0.722   0.449 0   0    1    1     1
 3 A2_Score              0             1     0.453   0.498 0   0    0    1     1
 4 A3_Score              0             1     0.457   0.499 0   0    0    1     1
```

Recoding variables

From character to factor using base R

```
#### char 2 factor -------------------------------------------------------------
# Say I want to treat some variables as factors
autism_pids$gender <- as.factor(autism_pids$gender)
autism_pids$ethnicity <- as.factor(autism_pids$ethnicity)
autism_pids$contry_of_res <- as.factor(autism_pids$contry_of_res)
autism_pids$relation <- as.factor(autism_pids$relation)

# check
class(autism_pids$gender)
```

```
[1] "factor"
```

```
class(autism_pids$ethnicity)
```

```
[1] "factor"
```

```
class(autism_pids$contry_of_res)
```

```
[1] "factor"
```

```
class(autism_pids$relation)
```

```
[1] "factor"
```

From character to factor using base R (n cols)

```
autism_pids_temp <- autism_pids # copy df for test

to_factor <- c("gender", "ethnicity", "contry_of_res", "relation") # vector of col names
autism_pids_temp[ ,to_factor] <-  lapply(X =  autism_pids[ ,to_factor], FUN = as.factor)

# check
class(autism_pids_temp$gender)
```

```
[1] "factor"
```

```
class(autism_pids_temp$ethnicity)
```

```
[1] "factor"
```

```
class(autism_pids_temp$contry_of_res)
```

```
[1] "factor"
```

```
class(autism_pids_temp$relation)
```

```
[1] "factor"
```

```
# now I have Variable type: factor
```

Inspect factors levels (3 different ways)

- using `base::levels` function

```
levels(autism_pids$ethnicity)
```

```
 [1] "Asian"            "Black"            "Hispanic"         "Latino"
 [5] "Middle Eastern "  "others"           "Others"           "Pasifika"
 [9] "South Asian"      "Turkish"          "White-European"
```

- using `base::table` function

```
table(autism_pids$ethnicity,useNA = "ifany")
```

```

          Asian           Black        Hispanic          Latino Middle Eastern 
            123              43              13              20              92 
         others          Others        Pasifika     South Asian         Turkish 
              1              30              12              36               6 
 White-European            <NA> 
            233              95 
```

Inspect factors levels – 3 different ways (cont.)

- using `janitor` function `tabyl`, which uses the "pipe" operator `%>%` which takes the output of a function as input of the next one

```r
janitor::tabyl(autism_pids$ethnicity) %>%
  adorn_totals() %>%
  adorn_pct_formatting()
```

```
 autism_pids$ethnicity   n percent valid_percent
                 Asian 123   17.5%         20.2%
                 Black  43    6.1%          7.1%
              Hispanic  13    1.8%          2.1%
                Latino  20    2.8%          3.3%
       Middle Eastern   92   13.1%         15.1%
                others   1    0.1%          0.2%
                Others  30    4.3%          4.9%
              Pasifika  12    1.7%          2.0%
           South Asian  36    5.1%          5.9%
               Turkish   6    0.9%          1.0%
        White-European 233   33.1%         38.3%
                  <NA>  95   13.5%             -
                 Total 704  100.0%        100.0%
```

Identify missing values

Use `is.na` to check if the 95 missing obs are the same missing for `ethnicity` and `relation`

```
which(is.na(autism_pids$ethnicity)) # indices of TRUE elements in vector
```

```
 [1]   5  13  14  15  20  21  25  26  63  80  81  82  92 217 222 239 258 271 277
[20] 278 286 307 316 325 338 339 340 341 342 343 344 345 346 347 348 349 350 351
[39] 352 353 354 355 356 362 366 370 371 373 379 380 381 382 383 384 385 386 387
[58] 388 389 391 396 400 401 402 404 424 428 429 430 433 439 454 486 506 519 528
[77] 535 536 537 538 557 565 572 573 589 594 637 643 646 652 653 659 660 667 702
```

```
which(is.na(autism_pids$relation))  # indices of TRUE elements in vector
```

```
 [1]   5  13  14  15  20  21  25  26  63  80  81  82  92 217 222 239 258 271 277
[20] 278 286 307 316 325 338 339 340 341 342 343 344 345 346 347 348 349 350 351
[39] 352 353 354 355 356 362 366 370 371 373 379 380 381 382 383 384 385 386 387
[58] 388 389 391 396 400 401 402 404 424 428 429 430 433 439 454 486 506 519 528
[77] 535 536 537 538 557 565 572 573 589 594 637 643 646 652 653 659 660 667 702
```

…indeed they are the same IDs!

From character to logical

I may prefer to code a variable as logical. For example, `age_desc` may be more explicit if coded as logical.

- I create a new column `age_desc_log`

```r
# observe a subset of some columns
autism_subset <- autism_pids [1:5, c("gender","jaundice", "autism","age_desc",
                                     "Class_ASD","pids")]
# View(autism_subset)

# recode "age_desc" as LOGICAL new var "age_desc_log"
autism_pids$age_desc_log <- ifelse(autism_pids$age_desc == "18 and more", TRUE, FALSE )
class(autism_pids$age_desc)
```

```
[1] "character"
```

```r
class(autism_pids$age_desc_log)
```

```
[1] "logical"
```

From character to dummy [0,1]

I also may need binary variables expressed as [0,1] (e.g. to incorporate nominal variables into regression analysis). Let's recode `autism`.

```r
autism_pids$autism_dummy <- ifelse(autism_pids$autism == 'yes', 1, 0)
class(autism_pids$autism)
```

```
[1] "character"
```

```r
class(autism_pids$autism_dummy)
```

```
[1] "numeric"
```

Subsetting the data for further investigation

Recall how to view the names of columns / variables

```r
colnames(autism_pids)
```

```
 [1] "id"              "A1_Score"        "A2_Score"        "A3_Score"       
 [5] "A4_Score"        "A5_Score"        "A6_Score"        "A7_Score"       
 [9] "A8_Score"        "A9_Score"        "A10_Score"       "age"            
[13] "gender"          "ethnicity"       "jaundice"        "autism"         
[17] "contry_of_res"   "used_app_before" "result"          "age_desc"       
[21] "relation"        "Class_ASD"       "pids"            "age_desc_log"   
[25] "autism_dummy"   
```

using head or tail from utils

- head or tail return the first or last parts of an object

```r
head(autism_pids)    #return fist 6 obs
tail(autism_pids)    #return last 6 obs
```

using head or tail from utils (cont.)

```r
head(autism_pids, n = 2) #return fist 2 obs
```

```
  id A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
1  1        1        1        1        1        0        0        1        1
2  2        1        1        0        1        0        0        0        1
  A9_Score A10_Score age gender      ethnicity jaundice autism contry_of_res
1        0         0  26      f White-European       no     no United States
2        0         1  24      m         Latino       no    yes        Brazil
  used_app_before result    age_desc relation Class_ASD        pids
1              no      6 18 and more     Self        NO PatientID_1
2              no      5 18 and more     Self        NO PatientID_2
  age_desc_log autism_dummy
1         TRUE            0
2         TRUE            1
```

```r
tail(autism_pids, n = 2) #return last 2 obs
```

```
     id A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
703 703        1        0        0        1        1        0        1        0
704 704        1        0        1        1        1        0        1        1
    A9_Score A10_Score age gender      ethnicity jaundice autism contry_of_res
703        1         1  35      m    South Asian       no     no      Pakistan
704        1         1  26      f White-European       no     no        Cyprus
    used_app_before result    age_desc relation Class_ASD          pids
703              no      6 18 and more     Self        NO PatientID_703
704              no      8 18 and more     Self       YES PatientID_704
    age_desc_log autism_dummy
703         TRUE            0
704         TRUE            0
```

Investigating a subset of observations

E.g. I learned that some patients have missing age… how many are they?

```r
# run...
sum(is.na(autism_pids$age))
```

```
[1] 2
```

```r
# or
skimr::n_missing(autism_pids$age)
```

```
[1] 2
```

So, next, I want to ID those patients with missing age.

New df (patients missing age) as SUBSET of the given c

I want to extract only the obs (*rows*) of interest with a few useful vars (*cols*)

Option 1) using [ ] from base

```r
missing_age_subset <- autism_pids[is.na(autism_pids$age),
                                  c("pids", "age", "autism_dummy") ]
missing_age_subset
```

```
           pids age autism_dummy
63 PatientID_63  NA            0
92 PatientID_92  NA            0
```

New df (patients missing age) as SUBSET of the given c

I want to extract only the obs (*rows*) of interest with a few useful vars (*cols*)

Option 2) using `which` from `base`

```r
missing_age_subset2 <- autism_pids[which(is.na(autism_pids$age)),
                                   c("pids", "age", "autism_dummy")]
missing_age_subset2
```

```
          pids age autism_dummy
63 PatientID_63  NA            0
92 PatientID_92  NA            0
```

New df (patients missing age) as SUBSET of the given c

I want to extract only the obs (*rows*) of interest with a few useful vars (*cols*)

Option 3) using `subset` from `base`

```r
# arguments allow me to specify rows and cols
missing_age_subset3 <- subset(x = autism_pids,
                              subset = is.na(autism_pids$age), # 1 logical condition
                              select = c("pids", "age", "autism_dummy") # which cols
                              )
missing_age_subset3
```

```
          pids age autism_dummy
63 PatientID_63  NA            0
92 PatientID_92  NA            0
```

New df (filtering on 2 conditions) as SUBSET of the given c

Option 1) using `base::subset`

```r
# Creates a SUBSET based on MORE conditions (`age` and `ethnicity`)
twocond_base_subset <- subset(x = autism_pids,
                              # 2 logical conditions
                              subset = age < 25 & contry_of_res == "Brazil",
                              # pick a few cols
                              select = c("pids", "age", "contry_of_res",
                                         "autism_dummy"))

twocond_base_subset
```

```
             pids age contry_of_res autism_dummy
2     PatientID_2  24        Brazil            1
54   PatientID_54  21        Brazil            1
94   PatientID_94  19        Brazil            1
429 PatientID_429  20        Brazil            0
587 PatientID_587  21        Brazil            0
588 PatientID_588  21        Brazil            0
```

New df (filtering on 2 conditions) as SUBSET of the given d

Option 2) using `dplyr` (`filter` + `select`)

Switching to the package `dplyr` and embracing the "pipe" (`%>%`) operator logic, in which the filtering (rows) and selecting (columns) is done in sequence

```r
## here the filtering (rows) and selecting (columns) is done in sequence
twocond_dplyr_subset <- autism_pids %>%
  dplyr::filter(age < 25 & contry_of_res == "Brazil") %>%  # which rows
  dplyr::select (pids, age, contry_of_res, autism_dummy)   # which cols

twocond_dplyr_subset
```

```
          pids age contry_of_res autism_dummy
1  PatientID_2  24        Brazil            1
2 PatientID_54  21        Brazil            1
3 PatientID_94  19        Brazil            1
4 PatientID_429 20        Brazil            0
5 PatientID_587 21        Brazil            0
6 PatientID_588 21        Brazil            0
```

Dealing with missing data

Input values where missing

⚠️ WARNING: This is a very delicate & risky step ⚠️

- any modified/imputed data (beyond the *original collection*) can affect subsequent analysis and statistical modeling
- it will be necessary to document and justify whichever approach is used to deal with missing data.

Let's assume we can get the missing data by cross-checking related clinical information

```r
# 1/2 create a new variable
autism_pids$age_inputed <- autism_pids$age
# 2/2 replace value (presumably taken from other source) of `aged_inputed`
  # CONDITIONAL on `pids`
autism_pids$age_inputed[autism_pids$pids == "PatientID_63"] <-  65
autism_pids$age_inputed[autism_pids$pids == "PatientID_92"] <-  45

# check
skimr::n_missing(autism_pids$age)
```

```
[1] 2
```

```r
skimr::n_missing(autism_pids$age_inputed)
```

```
[1] 0
```

DESCRIPTIVE STATISTICS

Summarizing all variables

Try these 2 options:

`base::c`

```r
summary(autism_pids)
```

`skimr::skim`

```r
skimr::skim(autism_pids)
```

Notice `summary` different behavior according to the variable's type

The function's results depend on the class of the object

- look at the output in case of `integer` (e.g. A1_Score)

```
summary(autism_pids$A1_Score)     # min, max quartiles, mean, median
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0000  0.0000  1.0000  0.7216  1.0000  1.0000 
```

- look at the output in case of `factor` (e.g. ethnicity)

```
summary(autism_pids$ethnicity)    # counts of levels' frequency (included NA!)
```

```
          Asian           Black        Hispanic          Latino Middle Eastern 
            123              43              13              20              92 
         others          Others        Pasifika     South Asian         Turkish 
              1              30              12              36               6 
 White-European            NA's 
            233              95 
```

Notice `summary` different behavior according to the variable's type (cont.)

- look at the output in case of `logical` (e.g. age_desc_log)

```
summary(autism_pids$age_desc_log) # counts of TRUE
```

```
   Mode    TRUE
logical     704
```

Frequency distributions with `table`

- Frequency distributions can be used for nominal, ordinal, or interval/ration variables

```r
table(autism_pids$gender)
```

```
  f   m 
337 367 
```

```r
table(autism_pids$age) # automatically drops missing...
```

```
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 
18 31 35 46 49 37 37 34 27 28 31 24 27 30 21 18 16 12 17 13 17 13  7 16  3 15 
43 44 45 46 47 48 49 50 51 52 53 54 55 56 58 59 60 61 64 
11 10  4  6  8  4  3  5  1  5  6  2  6  2  2  1  1  2  1 
```

```r
table(autism_pids$age, useNA = "ifany") #...unless specified
```

```
  17   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32 
  18   31   35   46   49   37   37   34   27   28   31   24   27   30   21   18 
  33   34   35   36   37   38   39   40   41   42   43   44   45   46   47   48 
  16   12   17   13   17   13    7   16    3   15   11   10    4    6    8    4 
  49   50   51   52   53   54   55   56   58   59   60   61   64 <NA> 
   3    5    1    5    6    2    6    2    2    1    1    2    1    2 
```

Cross tabulation with `table` (2 vars)

- Cross tabulation

```
table(autism_pids$gender, autism_pids$age_inputed)
```

```

   17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
f   7 11 22 22 18 14 17 10 11 14 18 15 16 13  8 14  6  7 12  7 11  6  5  9  0
m  11 20 13 24 31 23 20 24 16 14 13  9 11 17 13  4 10  5  5  6  6  7  2  7  3

   42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 58 59 60 61 64 65
f   6  5  4  5  4  3  2  2  2  0  2  4  1  1  0  1  0  1  1  0  0
m   9  6  6  0  2  5  2  1  3  1  3  2  1  5  2  1  1  0  1  1  1
```

```
table(autism_pids$ethnicity, autism_pids$autism_dummy)
```

```

                    0   1
  Asian           118   5
  Black            38   5
  Hispanic         12   1
  Latino           12   8
  Middle Eastern   83   9
  others            1   0
  Others           28   2
  Pasifika         10   2
  South Asian      34   2
  Turkish           5   1
  White-European  183  50
```

Grouping and summarizing with base R

E.g. I want to know the average age of men and women sub-groups.

Option 1) using by

```r
# by(data$column, data$grouping_column, mean)
by(data = autism_pids$age_inputed, INDICES = autism_pids$gender, FUN = mean)
```

```
autism_pids$gender: f
[1] 29.60237
------------------------------------------------------------
autism_pids$gender: m
[1] 28.98365
```

Grouping and summarizing with base R

Option 2) using tapply

```r
# i.e. apply a function to subsets of a vector or array, split by one or more factors.
tapply(X = autism_pids$age_inputed, INDEX = autism_pids$gender, FUN = mean)
```

```
       f        m 
29.60237 28.98365 
```

Option 3) using split + sapply

```r
# sapply(split(data$column, data$grouping_column), mean)
sapply(X = split(autism_pids$age_inputed, autism_pids$gender),FUN = mean) # returns a vecto
```

```
       f        m 
29.60237 28.98365 
```

Grouping and summarizing with `dplyr`

```
autism_pids %>%
  dplyr::group_by(gender) %>%
  dplyr::summarise(mean(age_inputed))  # returns a dataframe!
```

```
# A tibble: 2 × 2
  gender `mean(age_inputed)`
  <fct>                <dbl>
1 f                     29.6
2 m                     29.0
```

I could add more statistics to the grouped summary…

```
autism_pids %>%
  dplyr::group_by(gender) %>%
  dplyr::summarise(mean_age = mean(age_inputed),
                   N_obs = n(),
                   N_with_autism = sum(autism_dummy == 1)
  )
```

```
# A tibble: 2 × 4
  gender mean_age N_obs N_with_autism
  <fct>     <dbl> <int>         <int>
1 f          29.6   337            54
2 m          29.0   367            37
```

Measures of central tendency

Mean and median

Recall that:

Population MEAN $\mu = \frac{\sum_{i=1}^{n} x_i}{n}$

Sample MEAN

Sample MEDIAN

For uneven :

For even :

Mean/Median using base R

- Important to specify the argument `na.rm = TRUE` or the functions won't work

```r
## Let's use `age` and `age_inputed` to see what inputed missing values did
mean(autism_pids$age)
```

```
[1] NA
```

```r
median(autism_pids$age)
```

```
[1] NA
```

```r
mean(autism_pids$age, na.rm = TRUE)
```

```
[1] 29.20655
```

```r
median(autism_pids$age, na.rm = TRUE)
```

```
[1] 27
```

```r
mean(autism_pids$age_inputed)
```

```
[1] 29.27983
```

```r
median(autism_pids$age_inputed)
```

```
[1] 27
```

Create custom function to calculate statistical mode 1/2

R doesn't have a built-in function for the statistical mode, so we can create a custom one: `f_calc_mode`

Define the custom function

```
f_calc_mode  <- function(x) {
  # `unique` returns a vector of unique values
  uni_x <- unique(x)
  # `match` returns the index positions of 1st vector against 2nd vector
  match_x <- match(x, uni_x)
  # `tabulate` count the occurrences of integer values in a vector.
  tab_x  <-  tabulate(match_x)
  # returns element of uni_x that corresponds to max occurrences
  uni_x[tab_x == max(tab_x)]
}
```

Create custom function to calculate statistical mode 2/2

Call the custom function

```r
f_calc_mode(autism_pids$age)
```

```
[1] 21
```

```r
f_calc_mode(autism_pids$age_inputed)
```

```
[1] 21
```

```r
f_calc_mode(autism_pids$ethnicity)
```

```
[1] White-European
11 Levels: Asian Black Hispanic Latino Middle Eastern  others ... White-European
```

Measures of variability (or spread)

Variance and Standard deviation

Recall that:

Population Variance

Sample Variance

Population Standard deviation

Sample Standard deviation

Variance and Standard deviation using `base` R

- Important to specify the argument `na.rm = TRUE` or the functions won't work (or use the `age_inputed` variable)

```
var(autism_pids$age, na.rm = TRUE)
```

```
[1] 94.28966
```

```
var(autism_pids$age_inputed)
```

```
[1] 96.19328
```

```
sd(autism_pids$age, na.rm = TRUE)
```

```
[1] 9.710286
```

```
sd(autism_pids$age_inputed)
```

```
[1] 9.807817
```

VISUAL DATA EXPLORATION

Plotting with ggplot2

`ggplot2` provides a set of tools to map data to visual elements on a plot, to specify the kind of plot you want, and then subsequently to control the fine details of how it will be displayed. It basically allows to build a plot layer by layer (Figure 2).

- **data** -> specify what the dataset is
- **aesthetic mappings** (or just *aesthetics*) -> specify which dataset's variables will turn into the plot elements (e.g. and values, or categorical variable into colors, points, and shapes).
- **geom** -> the overall type of plot, e.g. `geom_point()` makes scatterplots, `geom_bar()` makes barplots, `geom_boxplot()` makes boxplots.

Additional (optional) pieces:

- information about the **scales**,
- the labels of **legends** and axes
- other **guides** that help people to read the plot,

Plotting with ggplot2 (cont.)

a layered approach!

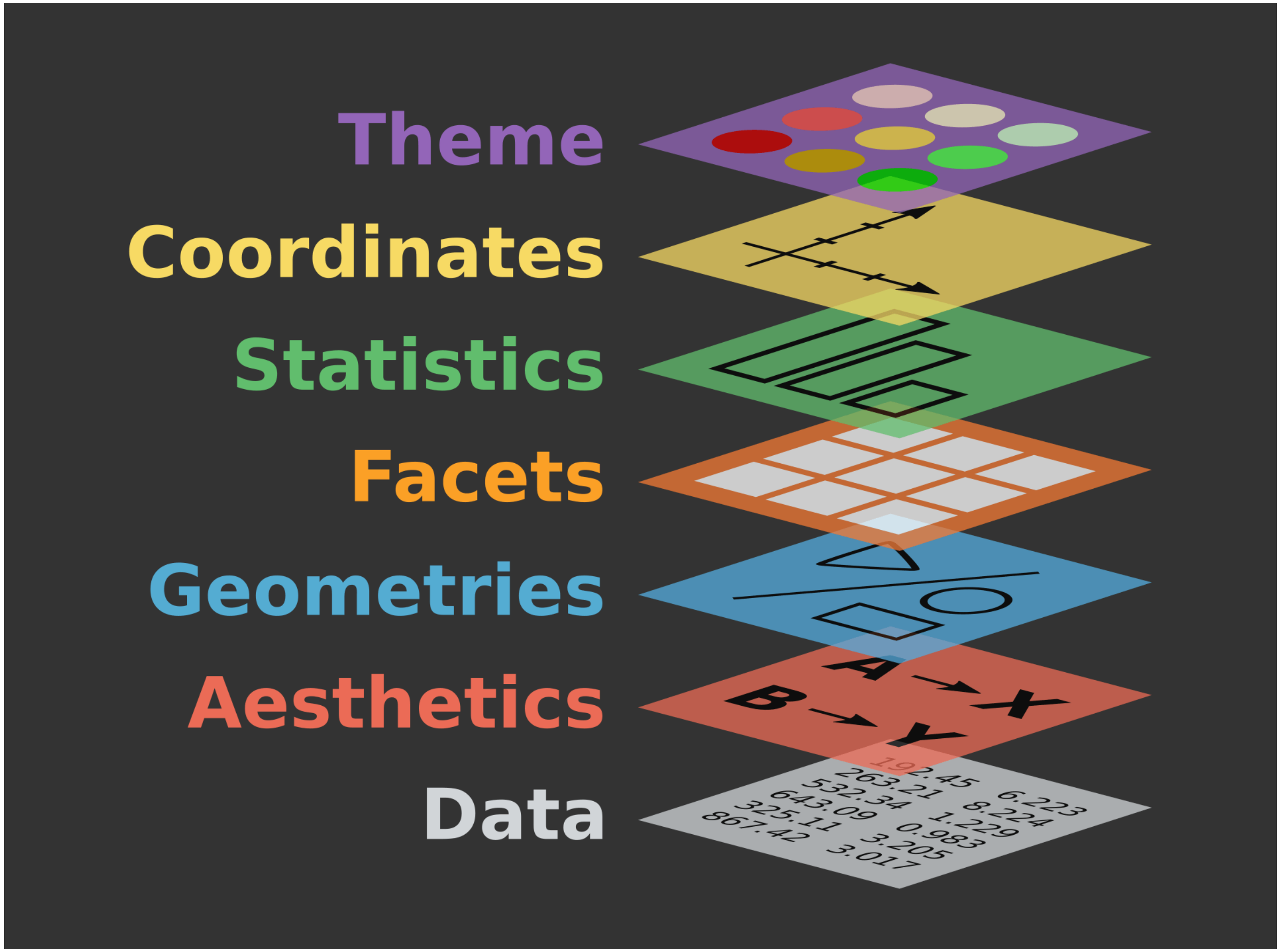


Figure 2: ggplot2 layers Source: Mine Çetinkaya-Rundel' Data Viz class

Save some colors (for customizing plots)

- Colors are defined in the form of **Hexadecimal color values**

```
two_col_palette <-  c("#9b2339", "#005ca1")

contrast_cols_palette <- c("#E7B800","#239b85", "#85239b", "#9b8523","#23399b",
                  "#d8e600", "#0084e6", "#399B23", "#e60066",
                  "#00d8e6", "#e68000")
```

Distribution of continuous var

Histograms

Histograms (and density plots) are often used to show the distribution of a continuous variable.

- Option 1) `data` inside the `ggplot()` function

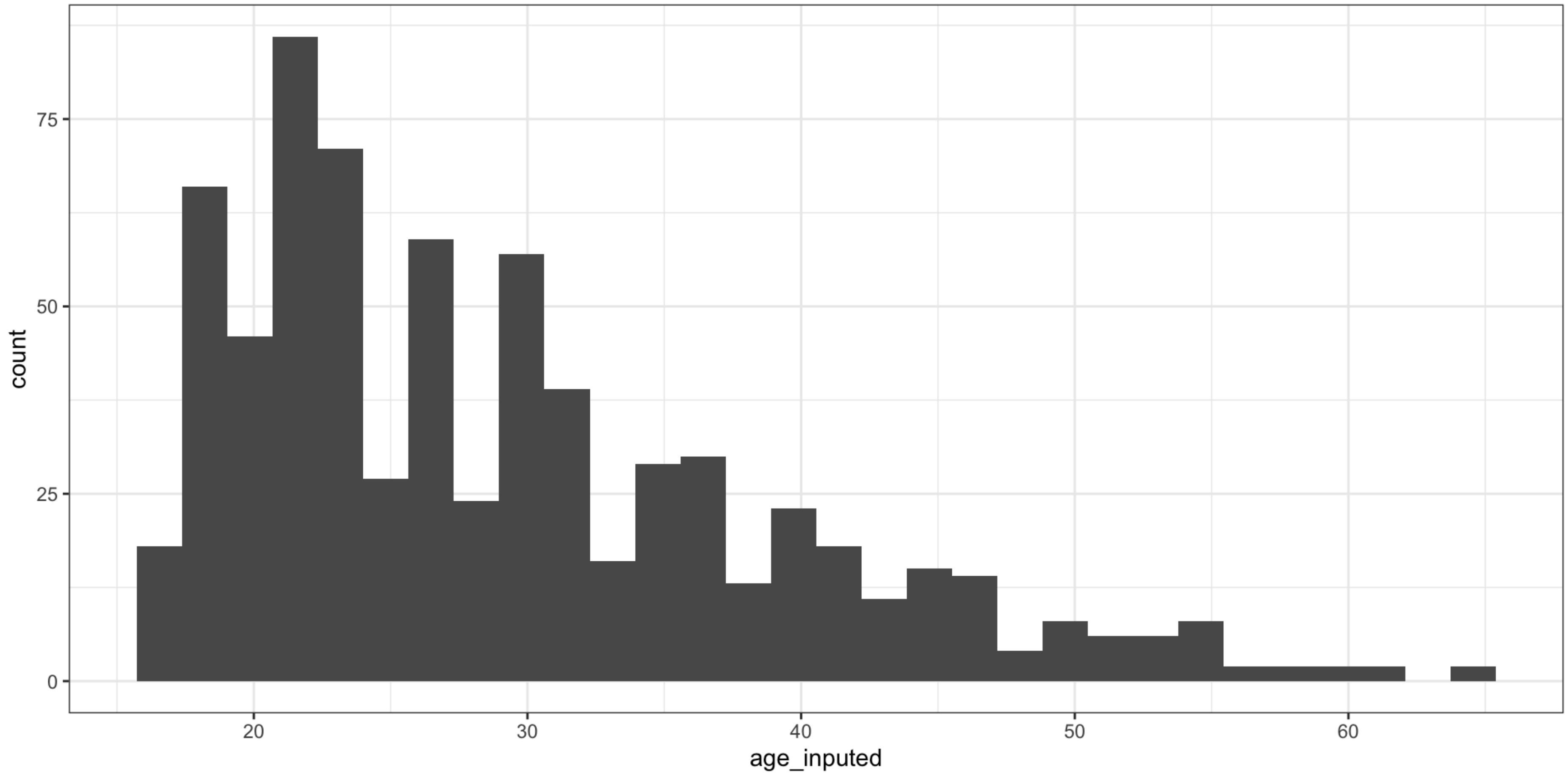
```
ggplot(data = autism_pids, mapping = aes(x=age_inputed)) +
  geom_histogram() +
  theme_bw()
```

- Option 2) `data` before the pipe `%>%`

```
autism_pids %>%
  ggplot(aes(x = age_inputed )) +
  geom_histogram() +
  theme_bw()
```

notice that after calling `ggplot()`, subsequent layers are added with `+`

Histograms



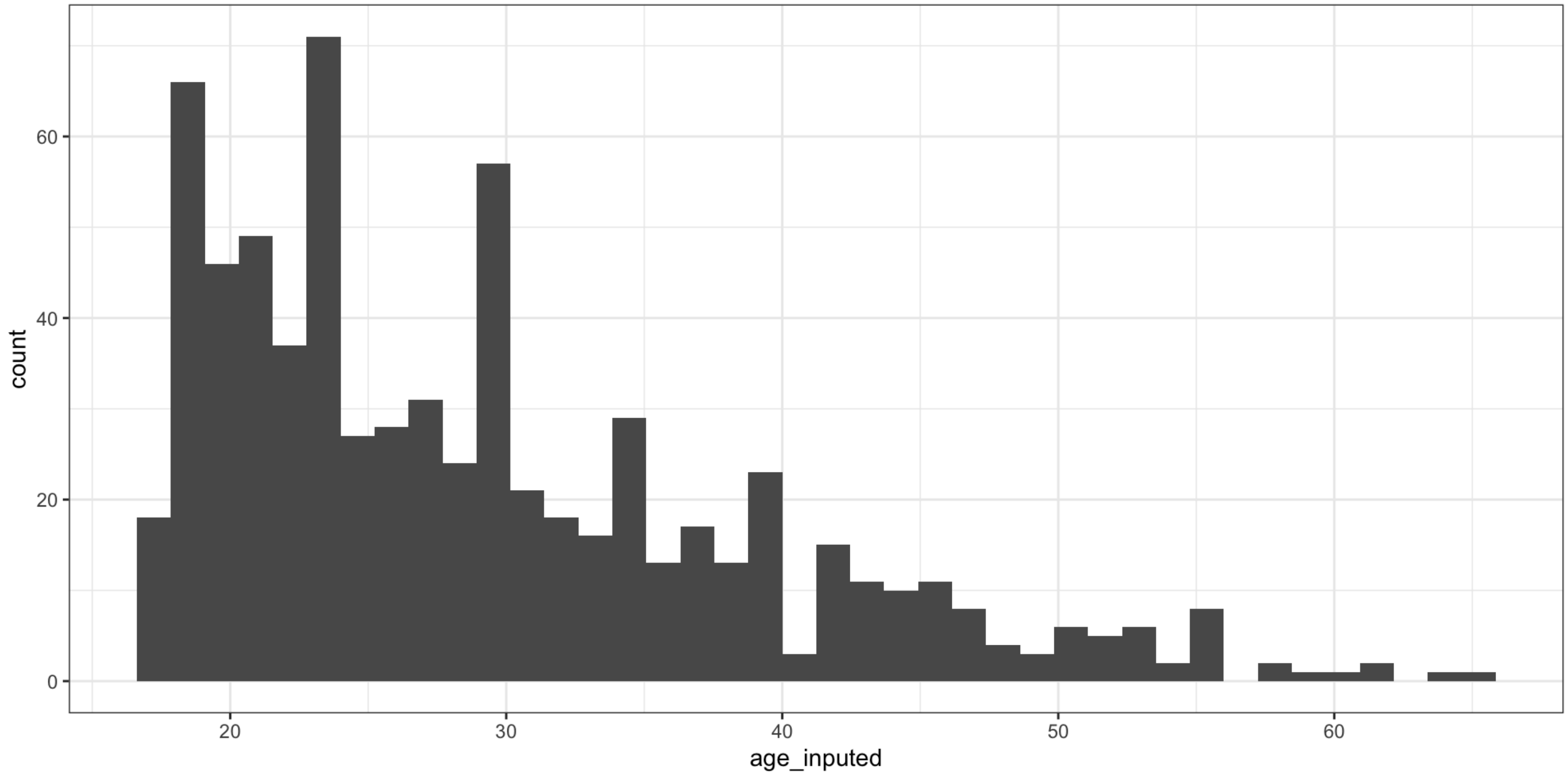
... define bin width

Histograms split the data into ranges (bins) and show the number of observations in each. Hence, it’s important to pick widths that represents the data well.

- The default value is 30
- We can change it using the argument `bins = #`

```
autism_pids %>%
  ggplot(aes(x = age_inputed )) +
  # specify to avoid warning if we fail to specify the number of bins
  geom_histogram(bins=40) +
  theme_bw()
```

… define bin width

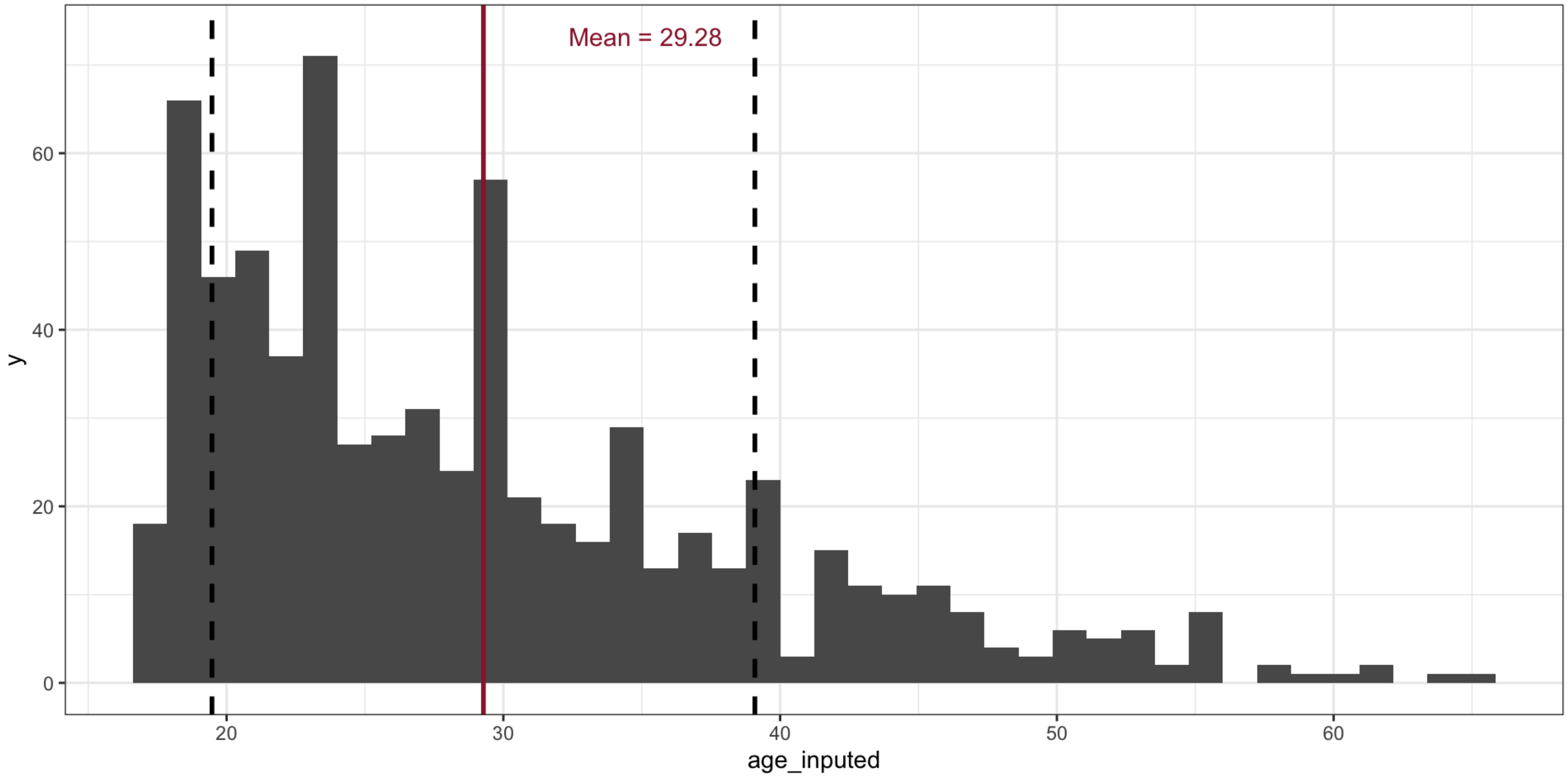


… add mean and std dev vertical lines

- using `geom_vline()` to add a vertical line for the *mean*, and the range between -1 and +1 *sd* from the mean.
- using `annotate()` for adding small annotations (such as text labels)

```
autism_pids %>%
  ggplot(aes(x = age_inputed )) +
  geom_histogram(bins=40) +
  # add mean vertical line
  geom_vline(xintercept = mean(autism_pids$age_inputed),
             na.rm = FALSE,
             lwd=1,
             color="#9b2339") +
  # add annotations with the mean value
  annotate("text",
           x = mean(autism_pids$age_inputed) * 1.2, # coordinates for positioning
           y = mean(autism_pids$age_inputed) * 2.5,
           label = paste("Mean =", round(mean(autism_pids$age_inputed), digits = 2)),
           col = "#9b2339",
           size = 4)+
  # add also sd +1 and -1
  geom_vline(aes(xintercept = mean(autism_pids$age_inputed) + sd(autism_pids$age_inputed)),
             color = "#000000", size = 1, linetype = "dashed") +
  geom_vline(aes(xintercept = mean(autism_pids$age_inputed) - sd(autism_pids$age_inputed)),
             color = "#000000", size = 1, linetype = "dashed") +
  theme_bw()
```

… add mean and std dev vertical lines

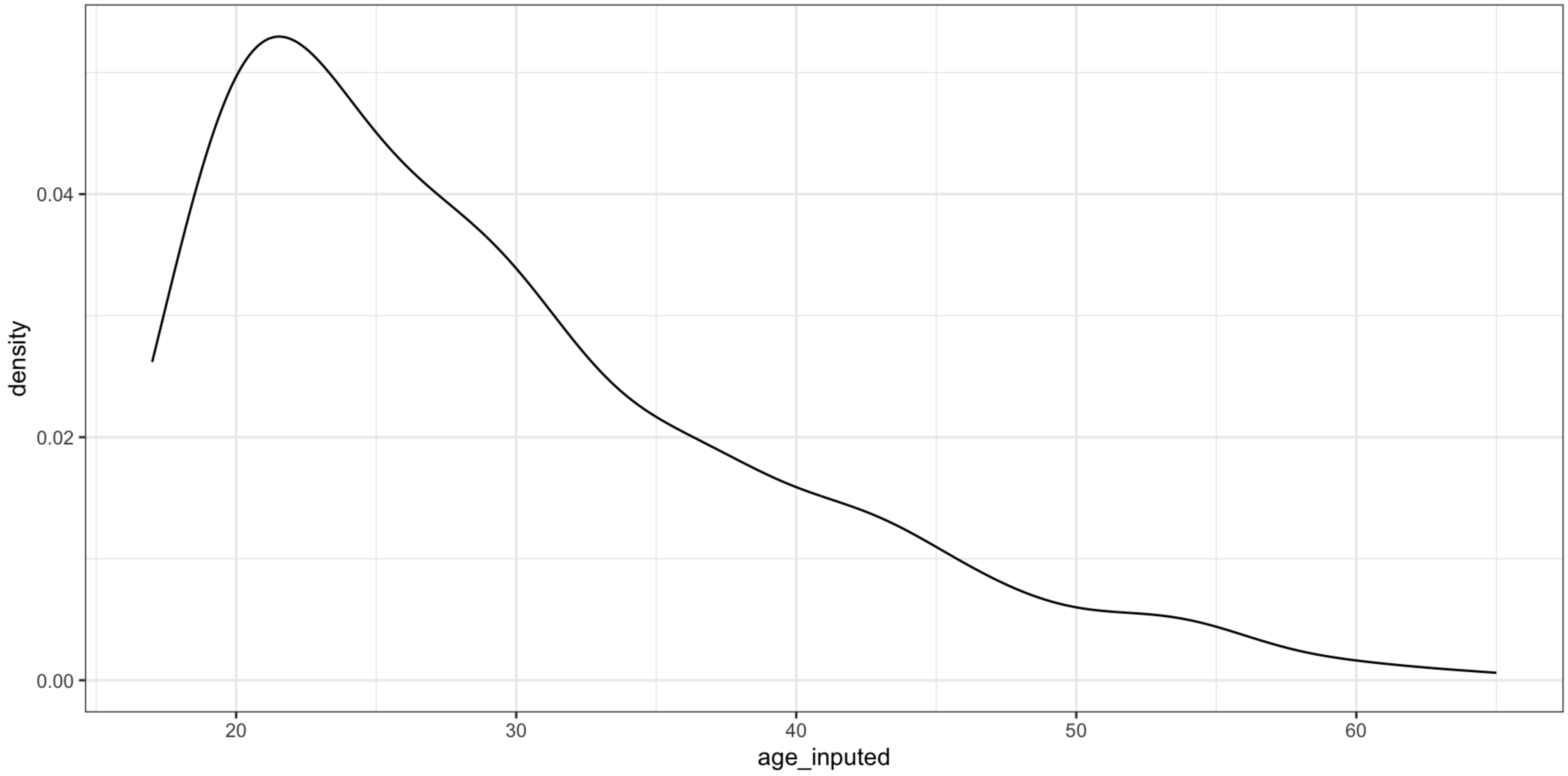


Density plot

- specifying `x` (the continuous variable)
- using `geom_density()`, in which we

```
autism_pids %>%
  ggplot(aes(x = age_inputed)) +
  geom_density()+
  theme_bw()
```

Density plot

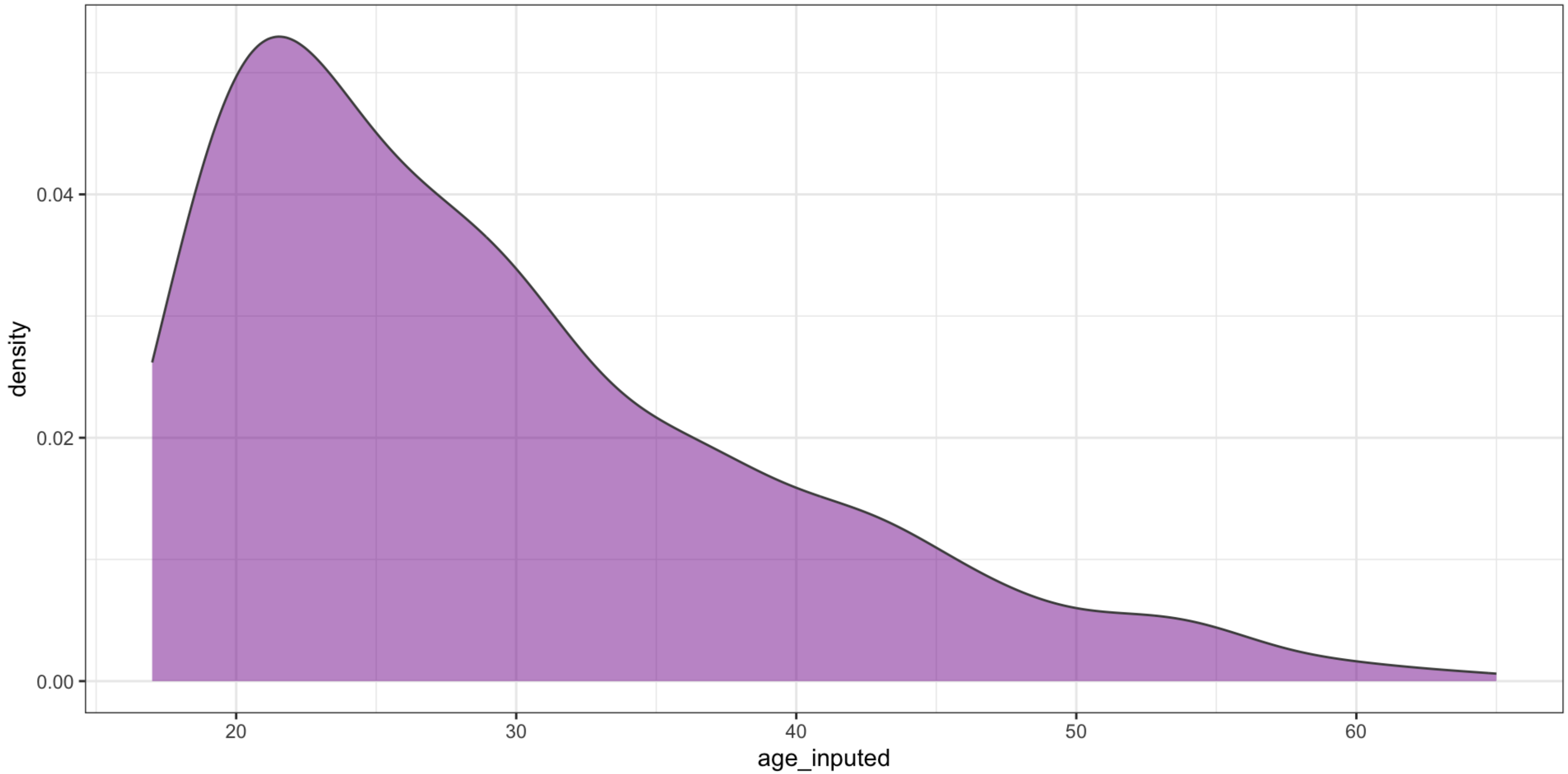


Density plot (cont.)

- specifying shape colors with the arguments inside `geom_density(...)`
- `color` for the line color
- `fill` for area color
- `alpha` to specify the degree of transparency in the density fill area

```r
autism_pids %>%
  ggplot(aes( x=age_inputed)) +
  geom_density(fill="#85239b", color="#4c4c4c", alpha=0.5)+
  theme_bw()
```

Density plot (cont.)

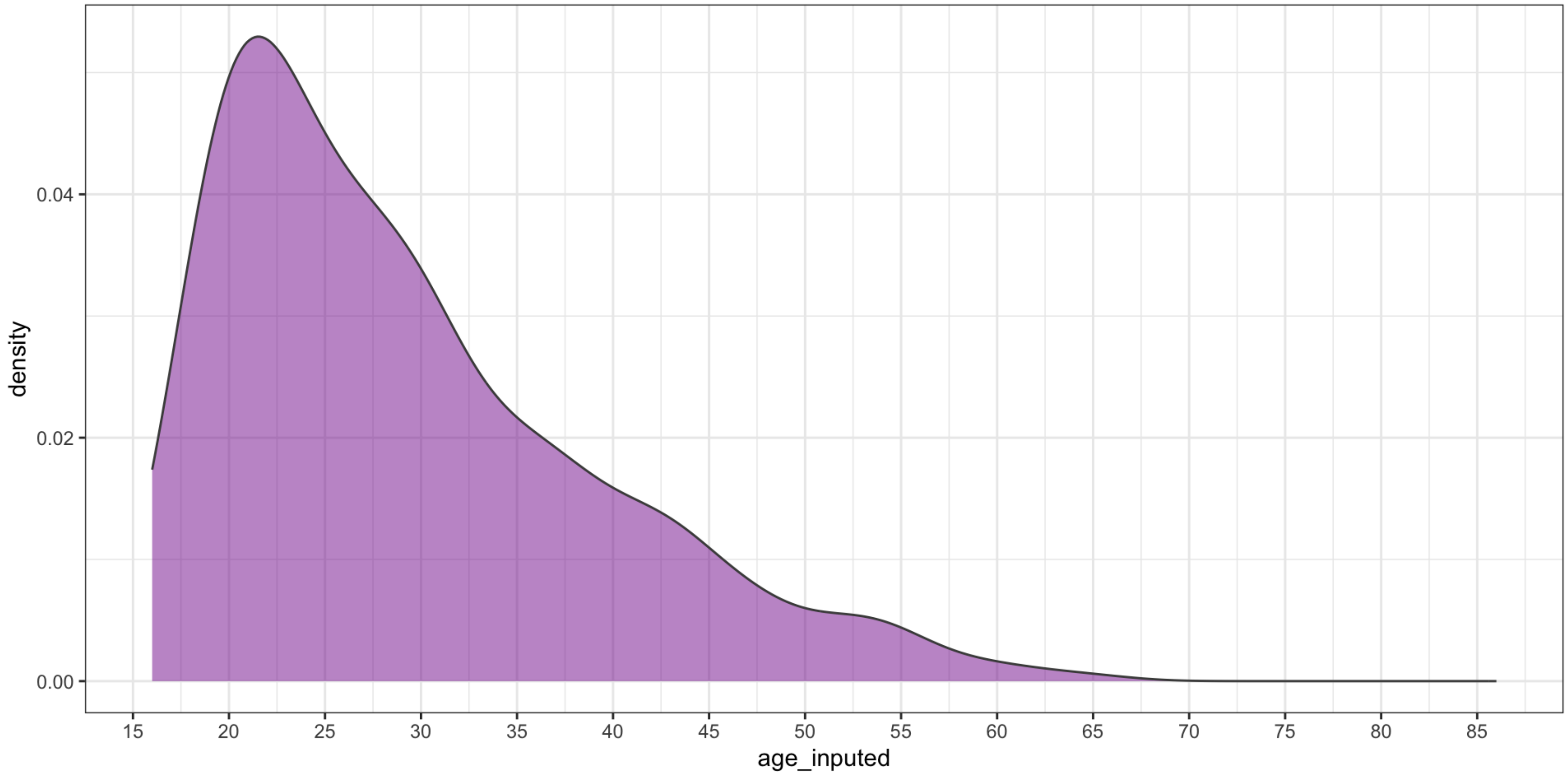


… increase # of x-axis ticks

- specifying the amount of breaks inside `scale_x_continuous()`

```r
autism_pids %>%
  ggplot(aes( x=age_inputed)) +
  geom_density(fill="#85239b", color="#4c4c4c", alpha=0.5)+
  theme_bw() +
  # increase number of x axis ticks
  scale_x_continuous(breaks = seq(10, 100,5 ), limits = c(16, 86))
```

… increase # of x-axis ticks



Distribution of continuous var split by categorical var

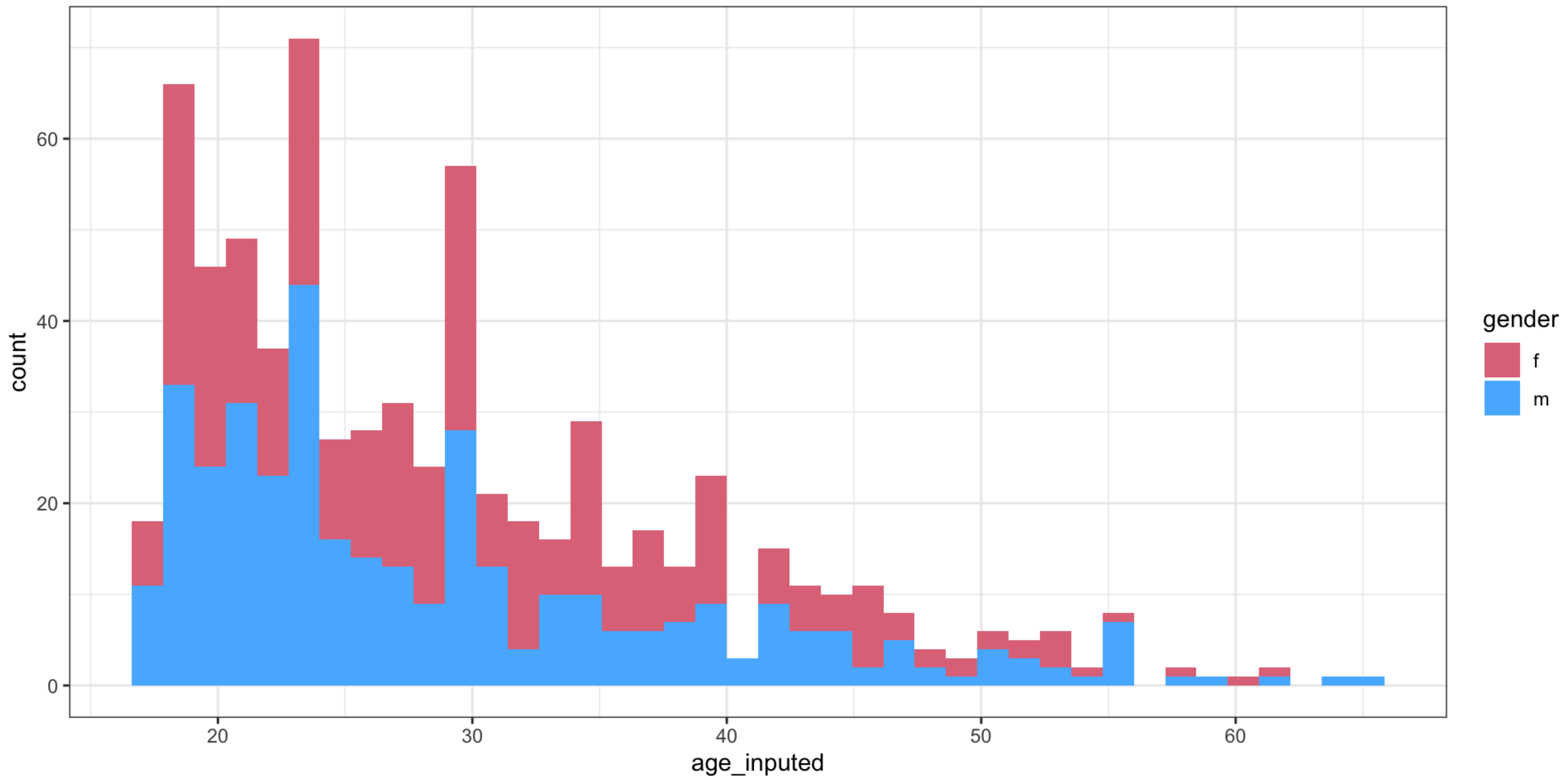
Histograms with `fill = category`

1. indicate the categorical group as `fill =` in the aesthetic mapping
2. specify custom colors for each group:

- use `scale_color_manual()` for changing line color
- use `scale_fill_manual()` for changing area fill colors.

```r
autism_pids %>%
  # specifying `fill` = gender
  ggplot(mapping = aes(x = age_inputed, fill = gender )) +
  geom_histogram(bins=40) +
  scale_fill_manual(values = c("#e07689","#57b7ff")) +
  scale_color_manual(values = c("#9b2339","#005ca1")) +
  theme_bw()
```

Histograms with `fill = category`

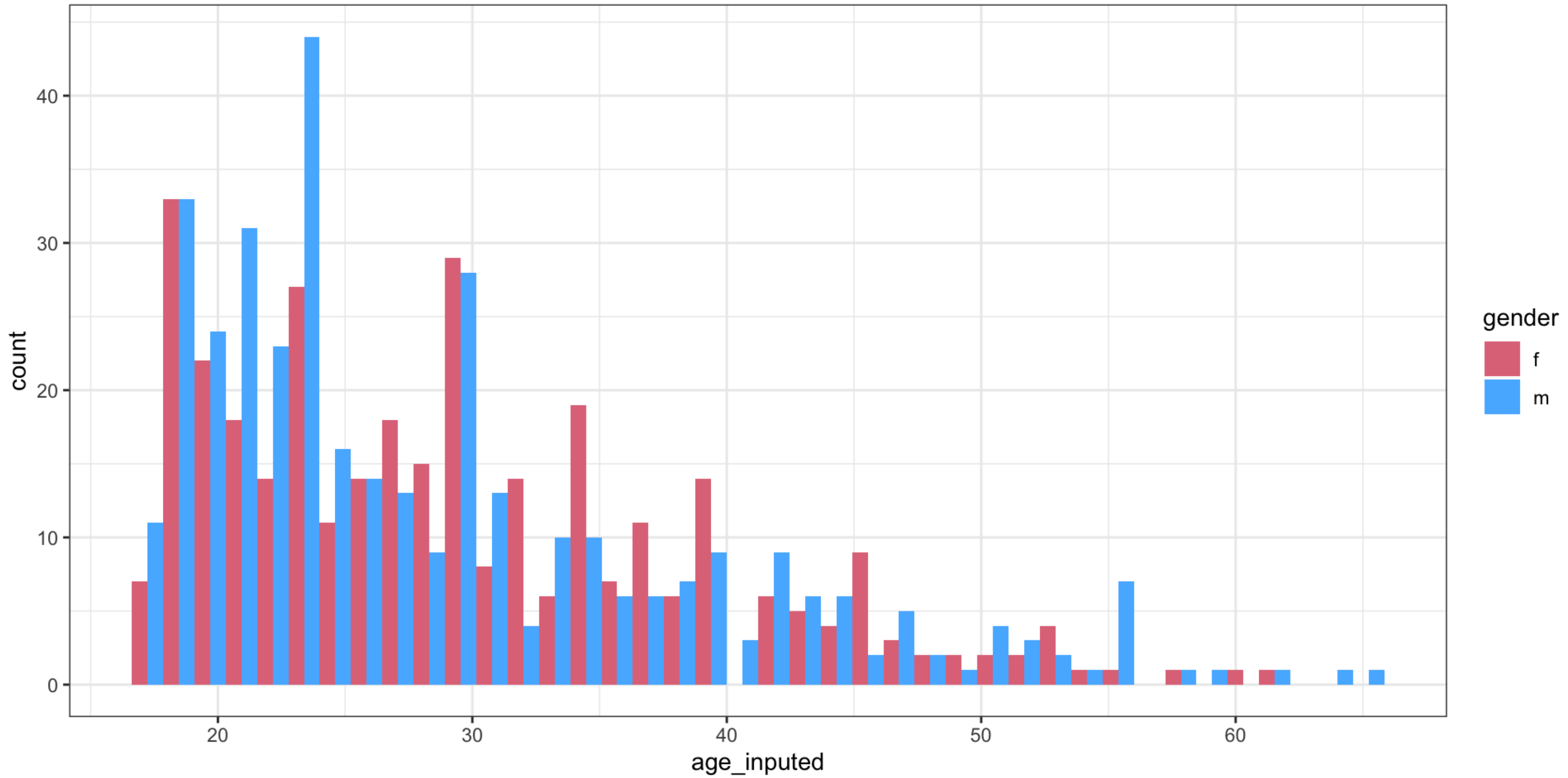


... shifting bars by group

- using the specification `position = 'dodge'` inside `geom_histogram()`

```r
# trying to improve readability
autism_pids %>%
  ggplot(mapping = aes(x = age_inputed, fill = gender )) +
  # bars next to each other with `position = 'dodge'`
  geom_histogram(bins=40, position = 'dodge')  +
  scale_fill_manual(values = c("#e07689","#57b7ff")) +
  scale_color_manual(values = c("#9b2339","#005ca1")) +
  theme_bw()
```

… shifting bars by group



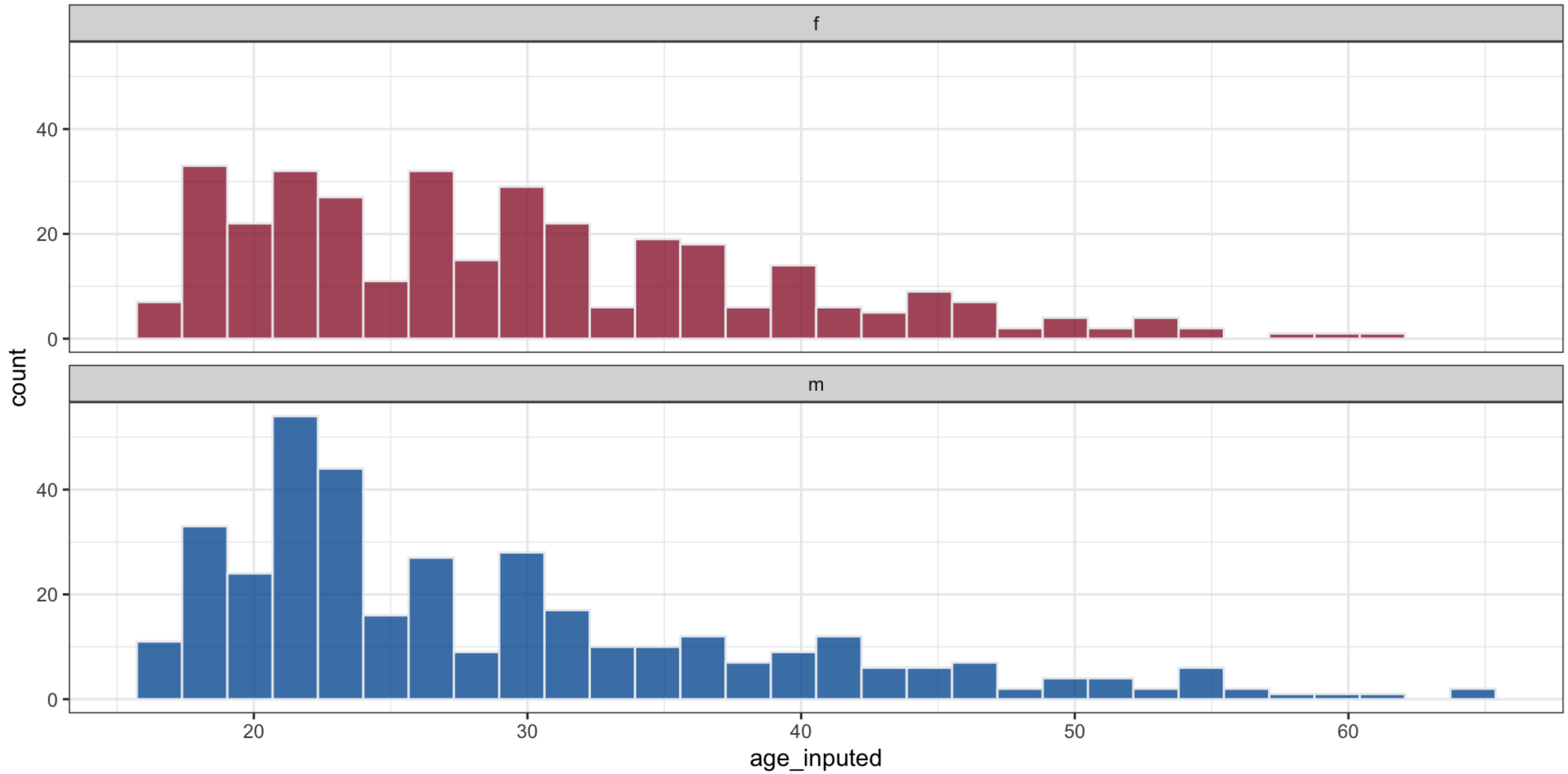
...facet by gender

That's still not very easy to digest. Instead of only filling, you can separate the data into multiple plots to improve readability

- adding `facet_wrap()` with the the specification of `~categ_var`
- also `ncol = 1` requires the subplot to be in 1 column

```
autism_pids %>%
  ggplot(aes(x = age_inputed, fill = gender )) +
  geom_histogram(color="#e9ecef", alpha=0.8, position = 'dodge') +
  theme_bw() +
  # splitting the gender groups, specifying `ncol` to see one above the other
  facet_wrap(~gender, ncol = 1)  +
  scale_fill_cyclical(values = c("#9b2339","#005ca1"))
```

...facet by gender



… adding 2 mean/median vert lines (by gender)

I want to see the mean vertical line for each of the subgroups, but in this case, I need to create a small dataframe of summary statistics (`group_stats`).

I do so by using `dplyr` add a column `mean_age` with the group mean

```r
group_stats <- autism_pids %>%
  dplyr::group_by(gender) %>%
  dplyr::summarize(mean_age = mean(age_inputed),
                   median_age = median (age_inputed))

group_stats
```

```
# A tibble: 2 × 3
  gender mean_age median_age
  <fct>     <dbl>      <dbl>
1 f          29.6         28
2 m          29.0         26
```

(Small digression on `tidyr::pivot_longer`

The new small dataframe `group_stats` offers an example of **reshaping**, i.e. turning a table from a "wide" form (with each variable in its own column) to a "long" form (one column for both the *measures names* and another for both the *measures values*).

- This can be done using `tidyr::pivot_longer` function, where these arguments must be specified:
  - `cols`: The names of the columns to pivot
  - `names_to`: The name for the new character column
  - `values_to`: The name for the new values column

```r
group_stats_long <- group_stats %>%
  tidyr::pivot_longer(cols = mean_age:median_age,
                      names_to = "Stat",
                      values_to = "Value") %>%
  dplyr::mutate(label = as.character(glue::glue("{gender}_{Stat}")))

group_stats_long
```

```
# A tibble: 4 × 4
  gender Stat       Value label
  <fct>  <chr>      <dbl> <chr>
1 f      mean_age    29.6 f_mean_age
2 f      median_age  28   f_median_age
3 m      mean_age    29.0 m_mean_age
4 m      median_age  26   m_median_age
```

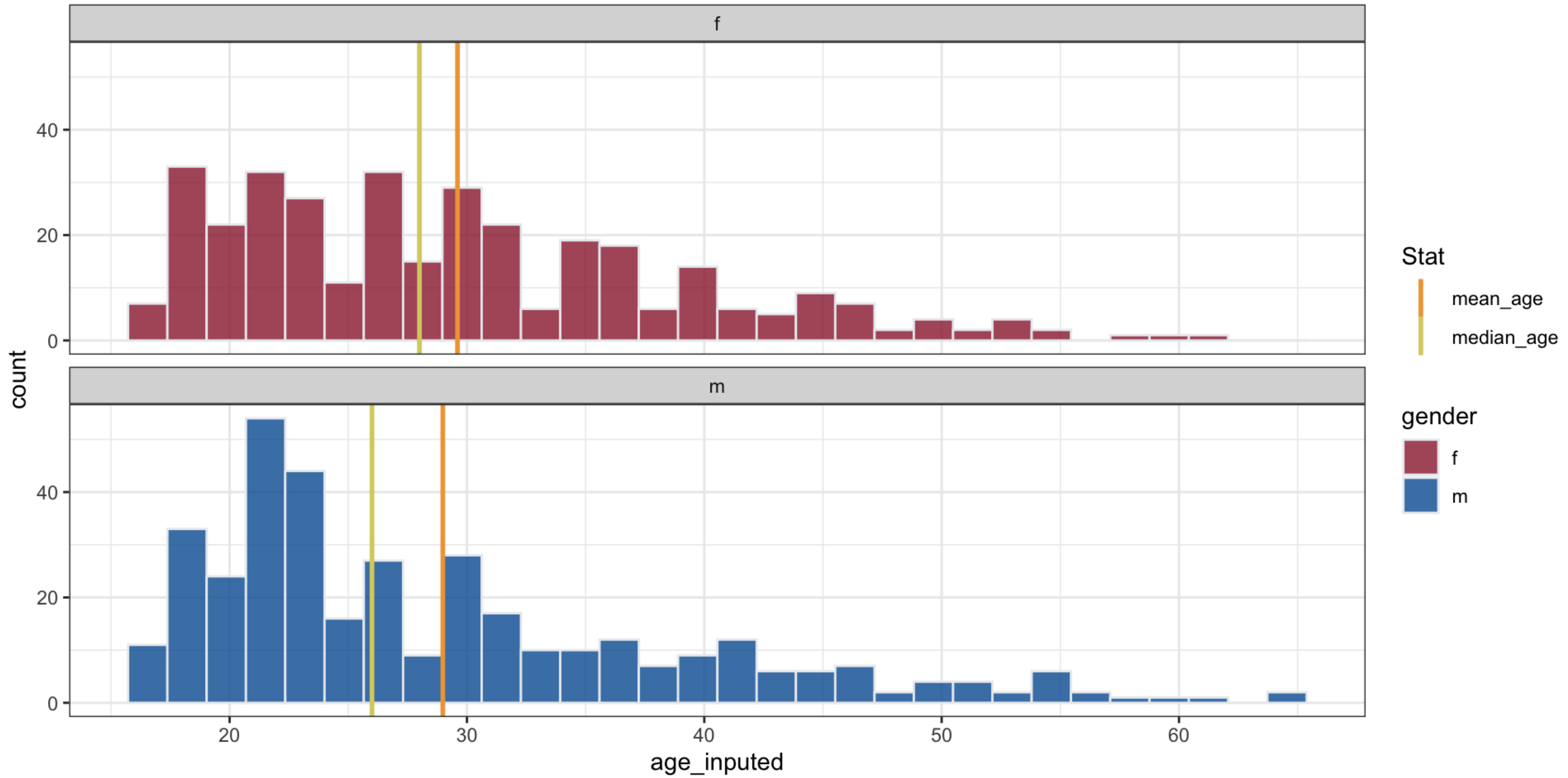
...facet by gender + vert lines by group

Notice that now the plot will have 2 `data` sources:

- `autism_pids`
- `group_stats_long`

```r
autism_pids %>%
  ggplot(aes(x = age_inputed, fill = gender)) +
  # geom_histogram from dataframe 1
  geom_histogram(bins=30,color="#e9ecef", alpha=0.8, position = 'dodge') +
  facet_wrap(~gender, ncol = 1) +
  scale_fill_manual(values = c("#9b2339","#005ca1"))  +
  # geom_vline from dataframe 2
  geom_vline(data = group_stats_long,
             mapping = aes(xintercept = Value, color = Stat),
             lwd=1,
             linetype=1) +
  scale_color_manual(values = c( "#f0a441" , "#d8cf71")) +
  theme_bw()
```

...facet by gender + vert lines by group

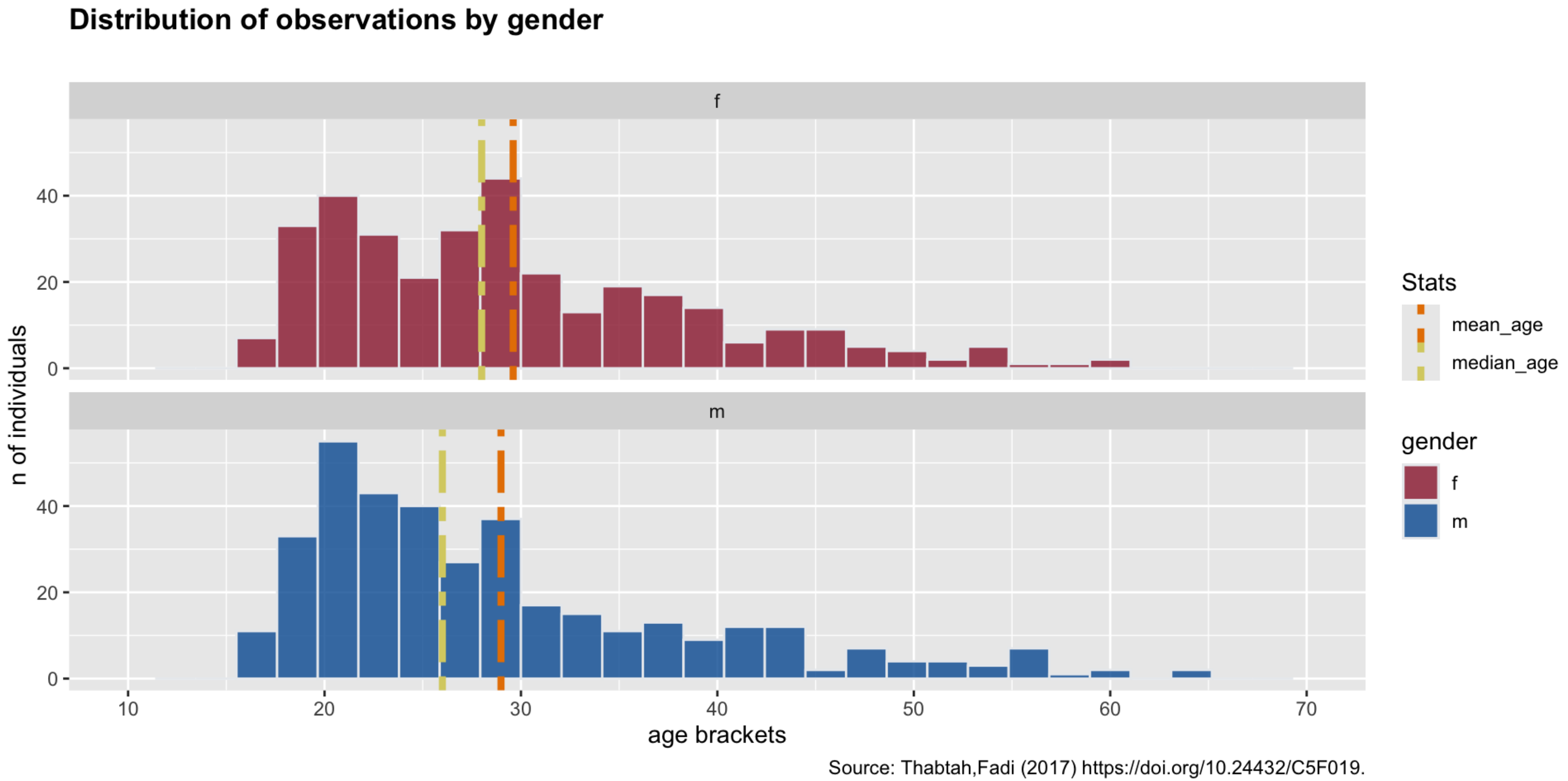


… finishing touches

- using `labs()` and `theme()` layers

```r
hist_plot <- autism_pids %>%
  ggplot(aes(x = age_inputed, fill = gender)) +
  # geom_histogram from dataframe 1
  geom_histogram(bins=30,color="#e9ecef", alpha=0.8, position = 'dodge') +
  facet_wrap(~gender, ncol = 1) +
  scale_fill_manual(values = c("#9b2339","#005ca1"))  +
  # geom_vline from dataframe 2
  geom_vline(data = group_stats_long,
             mapping = aes(xintercept = Value, color = Stat),
             lwd=1.5,
             linetype=6) +
  scale_color_manual(values = c( "#e68000", "#d8cf71")) +
  # increase number of x axis ticks
  scale_x_continuous(breaks = seq(10, 100,10 ), limits = c(10,70)) +
  # Additional theme details
  labs(x = "age brackets", y = "n of individuals",
       color = "Stats",
       title = "Distribution of observations by gender",
       subtitle = "",
       caption = "Source: Thabtah,Fadi (2017) https://doi.org/10.24432/C5F019.") +
  theme(legend.position = "right",
   plot.title = element_text(face = "bold"))
hist_plot
```

... finishing touches

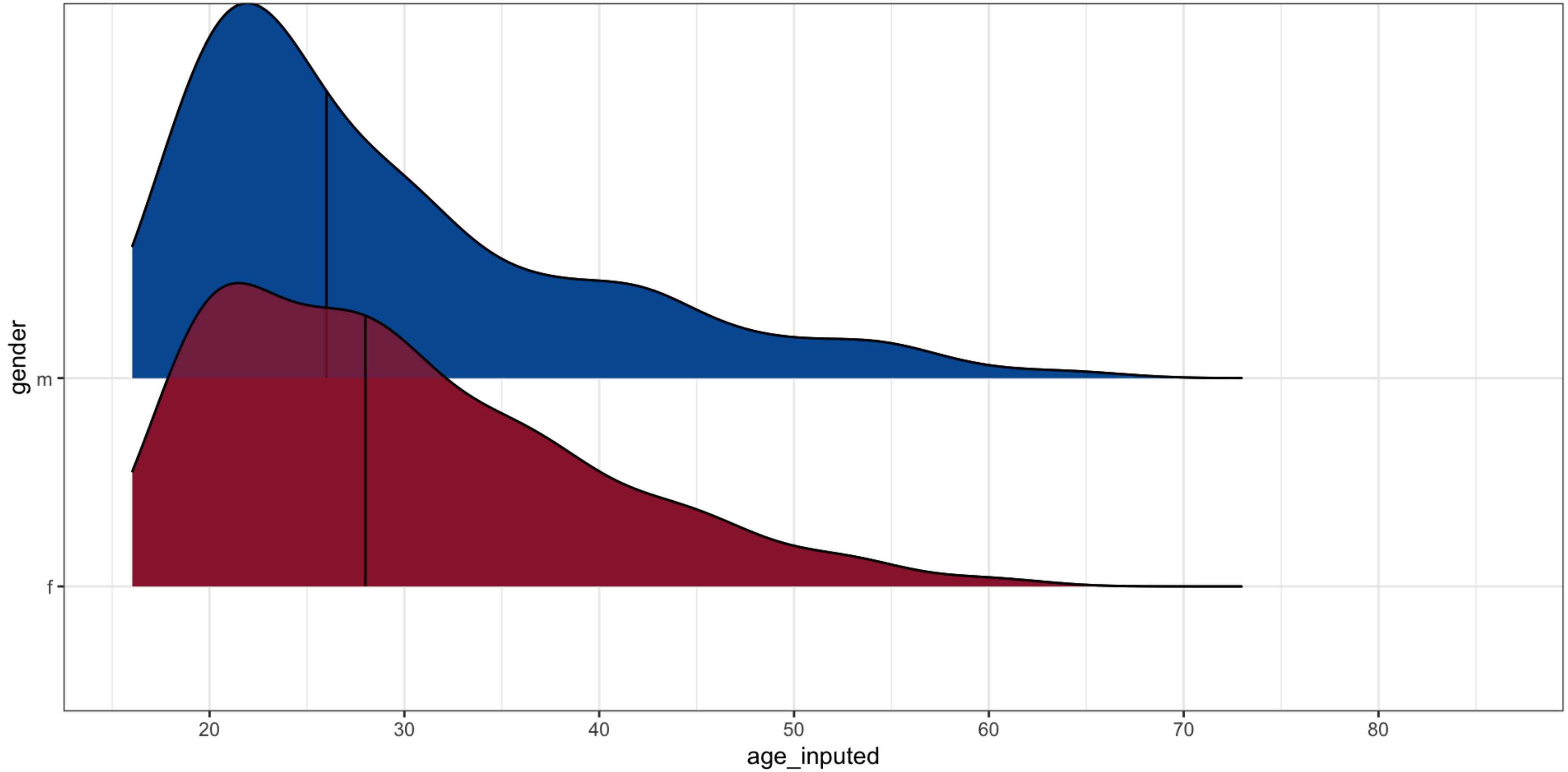


Density `ggridges` package

As an alternative, you can use the `ggridges` package to make ridge plots. The geom `geom_density_ridges` calculates density estimates from the provided data and then plots those, using the ridgeline visualization. In this case plots include a vertical median line.

```r
autism_pids %>%
  # this takes also `y` = group
  ggplot(aes(x=age_inputed, y = gender, fill = gender)) +
  ggridges::geom_density_ridges() +
  # I can add quantile lines (2 is the median)
  stat_density_ridges(quantile_lines = TRUE, quantiles = c(0.5), alpha = 0.75)+
  # increase number of x axis ticks
  scale_x_continuous(breaks = seq(10, 100,10 ), limits = c(16, 86)) +
  scale_fill_cyclical(values = c("#9b2339","#005ca1")) +
  theme_bw()
```

Density `ggridges` package



Barchart

Bar charts provide a visual presentation of categorical data, with `geom_bar()` (height of the bar proportional to the number of cases in each group)

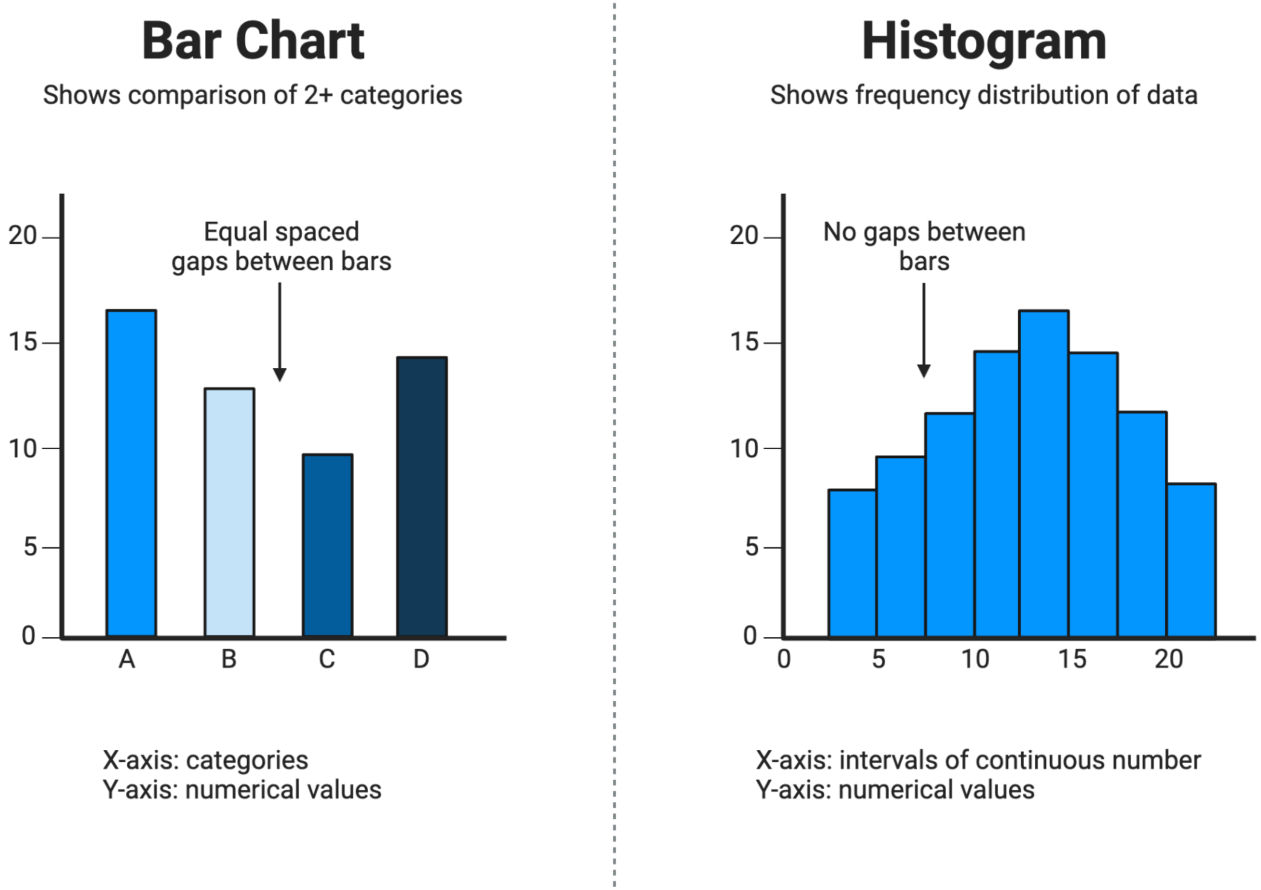


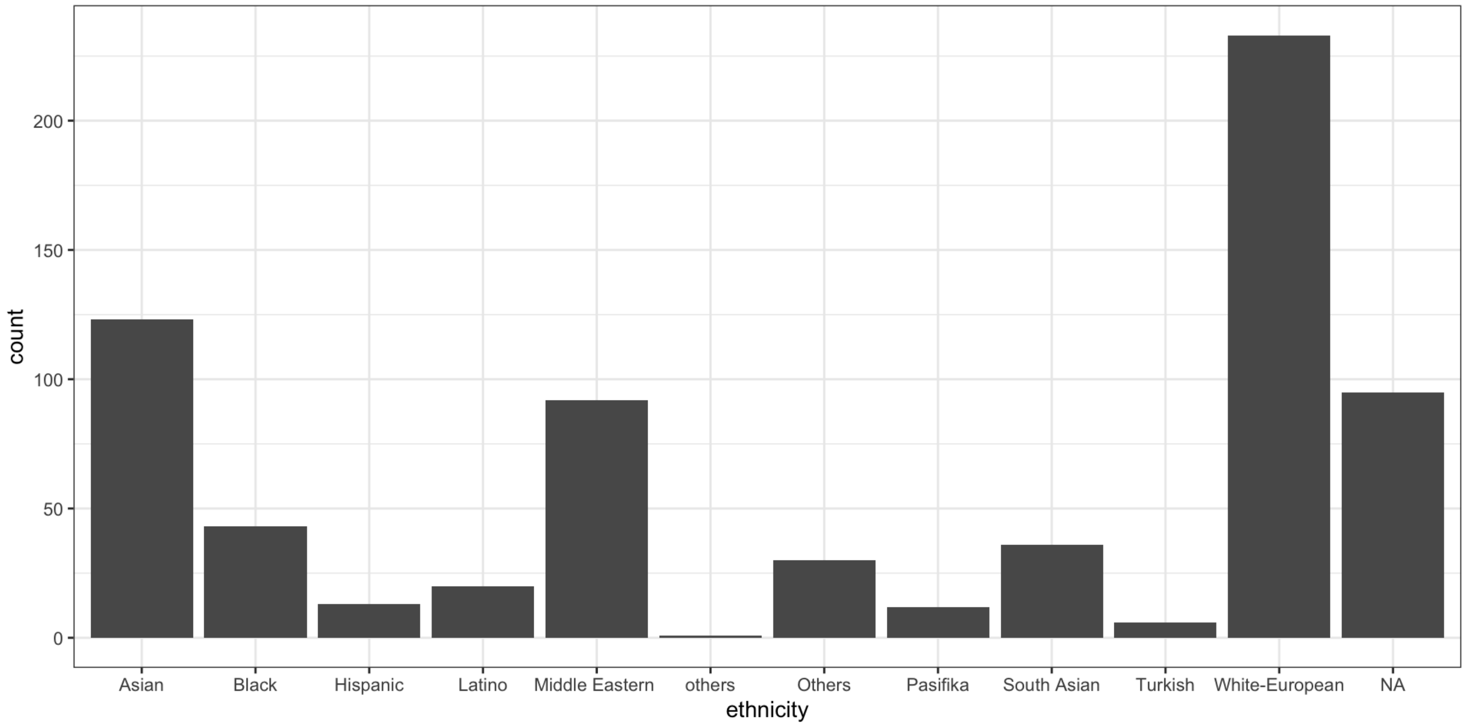
Figure 3: Difference barchart v. histogram Source: https://www.biorender.com/

Barchart (cont.)

```r
# Let's take a variable that we recoded as `factor`
class(autism_pids$ethnicity)
```

```
[1] "factor"
```

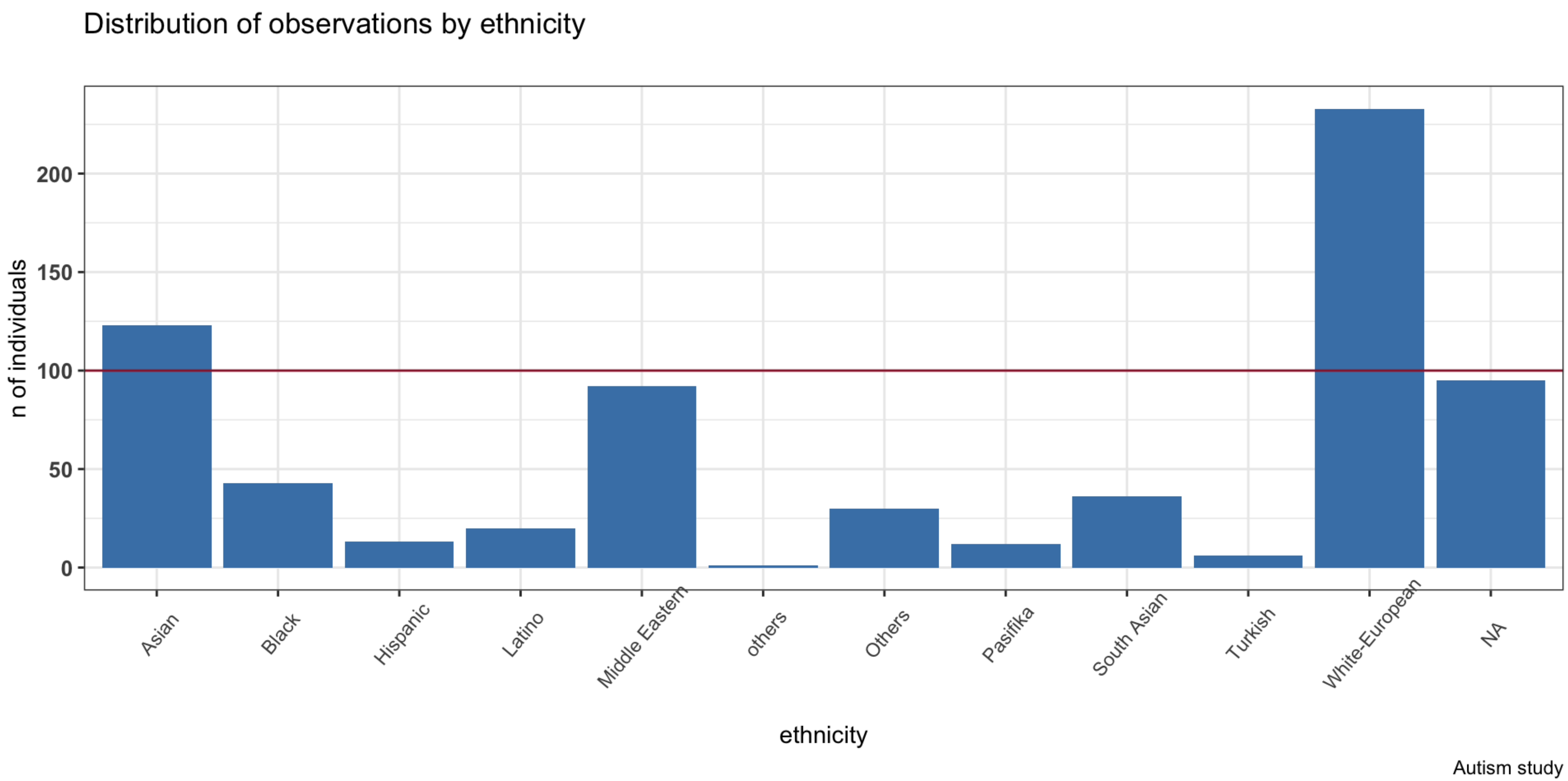
```r
#### ... no formatting ---------------------------------
autism_pids %>%
  ggplot(aes(x = ethnicity )) +
  geom_bar() +
  theme_bw()
```



...improve theme

```
autism_pids %>%
  ggplot(aes(x = ethnicity )) +
  geom_bar(fill = "steelblue") +
  # reference line
  geom_hline(yintercept=100, color = "#9b2339", size=0.5, ) +
  # labels, title, etc
  labs(x = "ethnicity", y = "n of individuals",
       color = "Stats",
       title = "Distribution of observations by ethnicity",
       subtitle = "",
       caption = "Autism study")  +
  theme_bw() +
  # specification son axis labels
  theme(axis.text.x = element_text(angle=50, vjust=0.75),
        axis.text.y = element_text(size=10,face="bold"))
```

...improve theme



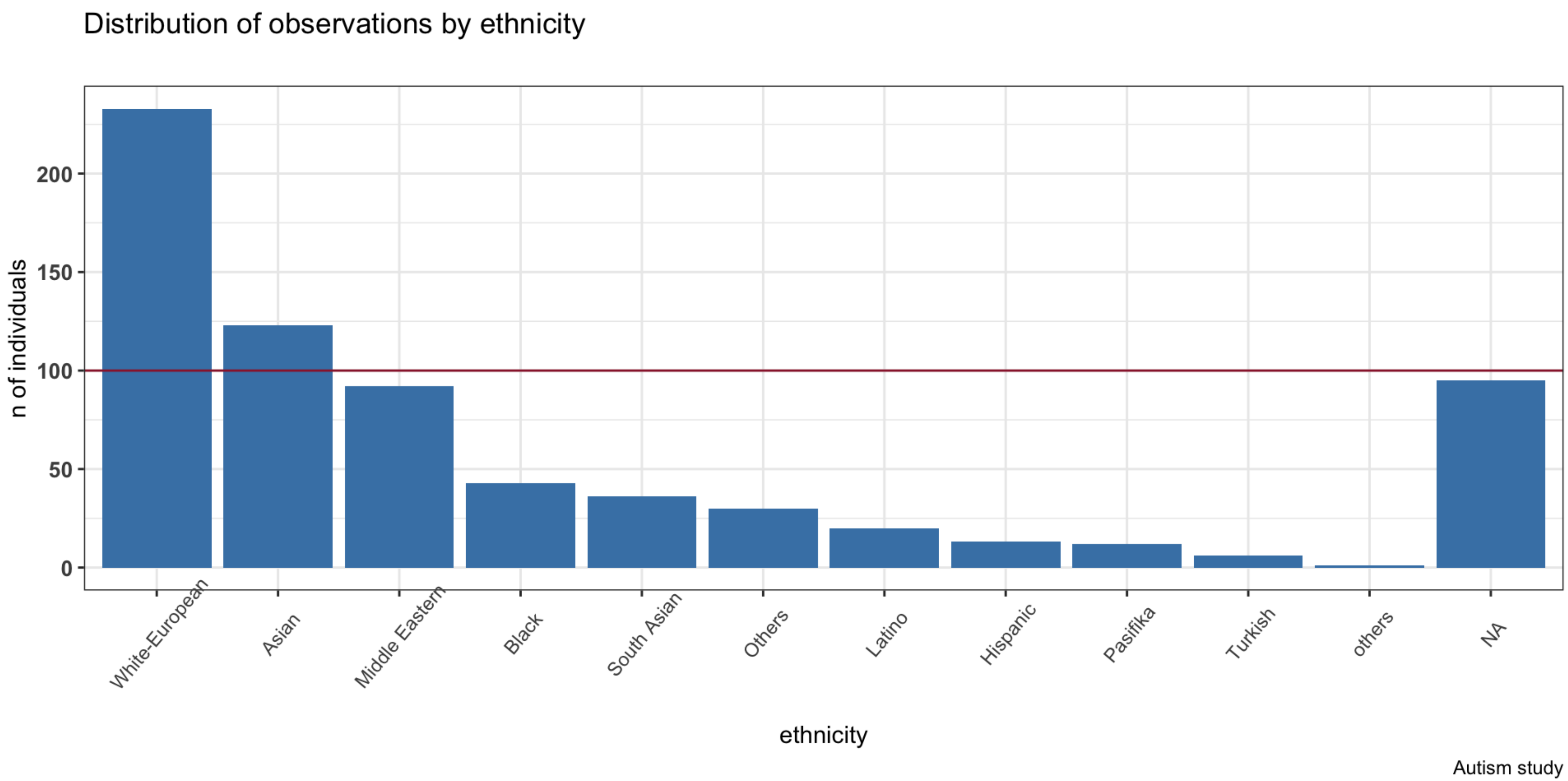
...improve readability (reorder bars)

Reordering the bars by count using the package `forcats` and its function `fct_infreq`

- (which we can do because ethnicity was recoded as `factor`)

```r
autism_pids %>%
    # we modify our x like so
    ggplot(aes(x = forcats::fct_infreq(ethnicity ))) +
    geom_bar(fill = "steelblue") +
    geom_hline(yintercept=100, color = "#9b2339", size=0.5, ) +
    labs(x = "ethnicity", y = "n of individuals",
         color = "Stats",
         title = "Distribution of observations by ethnicity",
         subtitle = "",
         caption = "Autism study")  +
    # --- wrap long x labels (flipped ) !!!
    #  scale_x_discrete(labels = function(x) stringr::str_wrap(x, width = 10)) +
    theme_bw() +
    theme(axis.text.x = element_text(angle=50, vjust=0.75),
          axis.text.y = element_text(size=10,face="bold"))
```

...improve readability (reorder bars)



...improve readability (highlight NA)

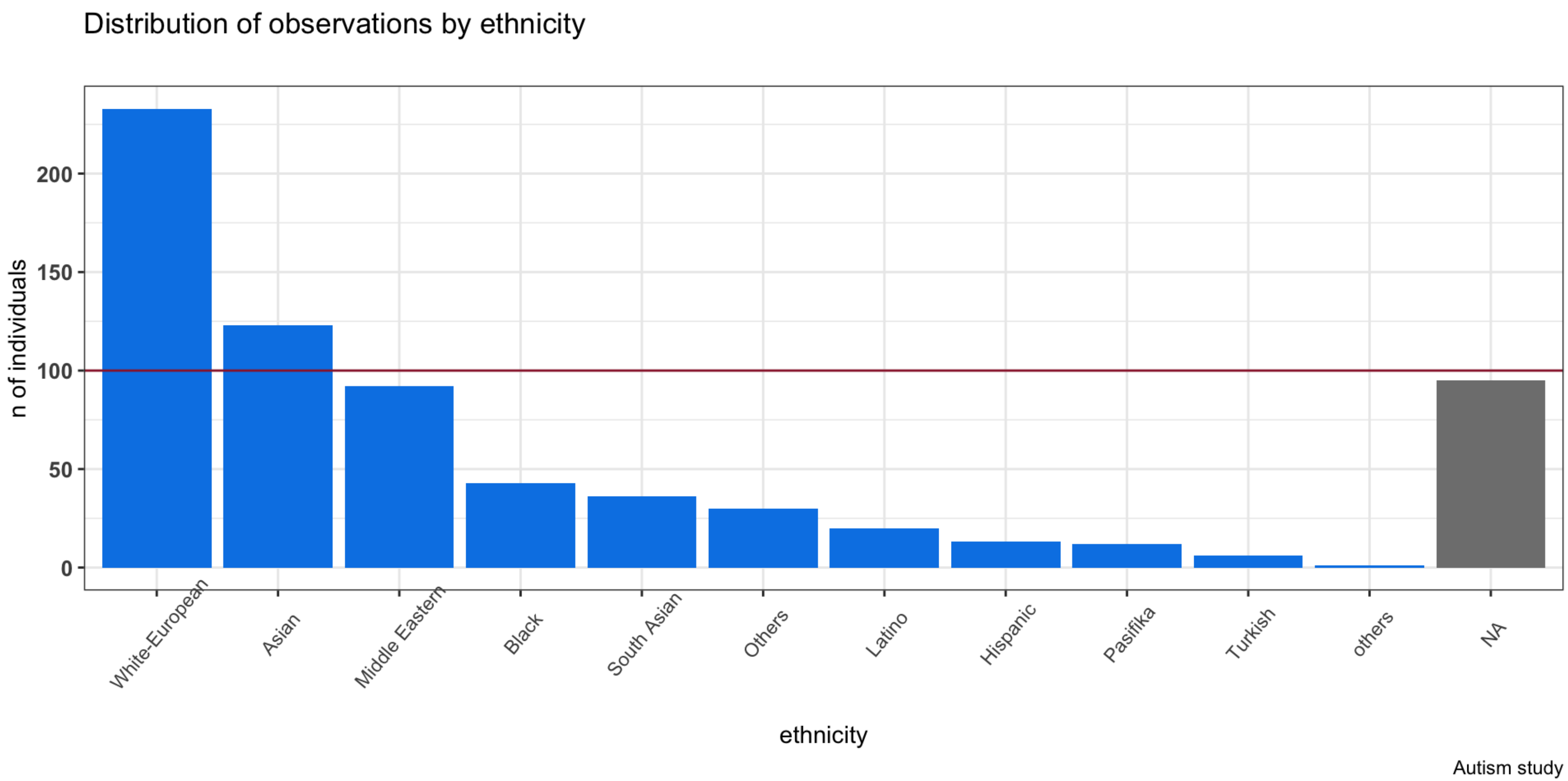
Let’s highlight the fact that the last column (`NA`) represents missing values.

1. Create the `highlight` variable
2. Map color to a variable (`fill = highlight`)

…improve readability (highlight NA) cod

```r
autism_pids %>%
  ## --- prep the dataframe
  dplyr::mutate(# Add a factor variable with two levels
    highlight = forcats::fct_other(ethnicity,
                                   keep = "NA",
                                   other_level = "All Groups")) %>%
  ## --- now plot
  # In `aes mapping` we map color to a variable (`fill = highlight`)
  ggplot(aes(x = forcats::fct_infreq(ethnicity), fill = highlight)) +
  geom_bar()+
  # Use custom color palettes
  scale_fill_manual(values=c("#0084e6")) +
  # Add a line at a significant level
  geom_hline(yintercept=100, color = "#9b2339", size=0.5, ) +
  theme_bw() +
  # make some more theme specifications
  labs(x = "ethnicity", y = "n of individuals",
       color = "Stats",
       title = "Distribution of observations by ethnicity",
       subtitle = "",
       caption = "Autism study")  +
  theme(axis.text.x = element_text(angle=50, vjust=0.75),
        axis.text.y = element_text(size=10,face="bold"))  +
  theme(legend.position = "none")
```

...improve readability (highlight NA) cod



Boxplot

The boxplot is one of the simplest ways of representing a distribution of a continuous variable and it is packed with information. It consists of two parts:

- **Box** — Extending from the 1st to the 3rd quartile (Q1 to Q3) with a line in the middle that represents the median.
- **Whiskers** — Lines extending from both ends of the box (minimum/maximum whisker values are calculated as Q1/Q3 -/+ 1.5 * IQR)
- Everything outside is represented as an **outlier**

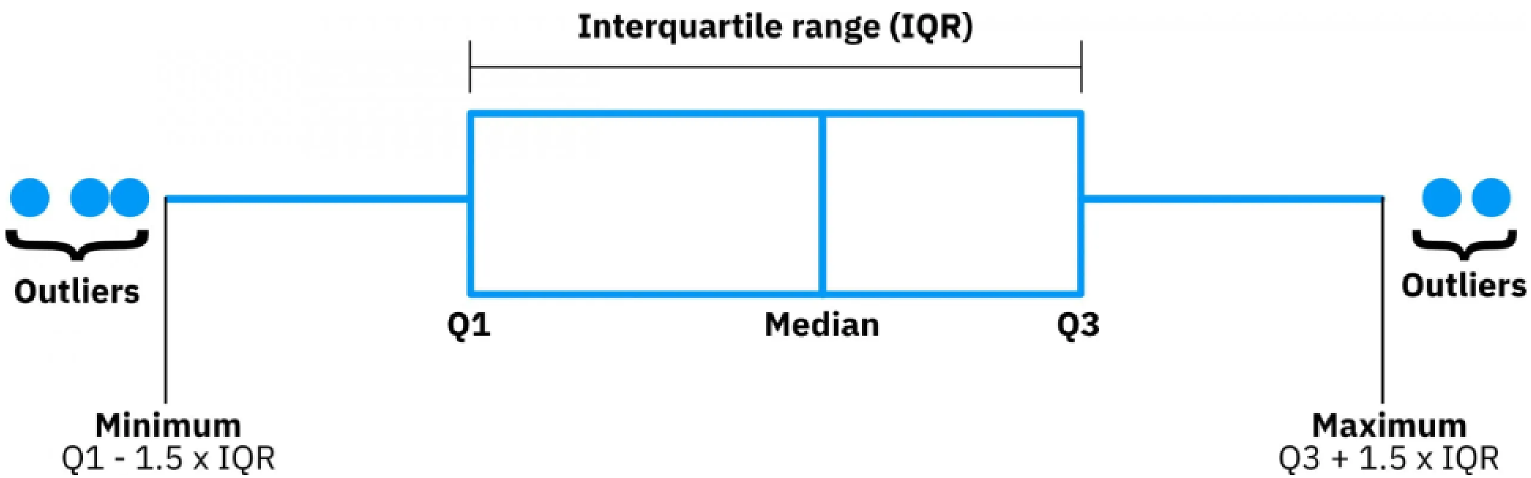


Figure 4: Boxplot Source: https://www.appsilon.com/post/ggplot2-boxplots

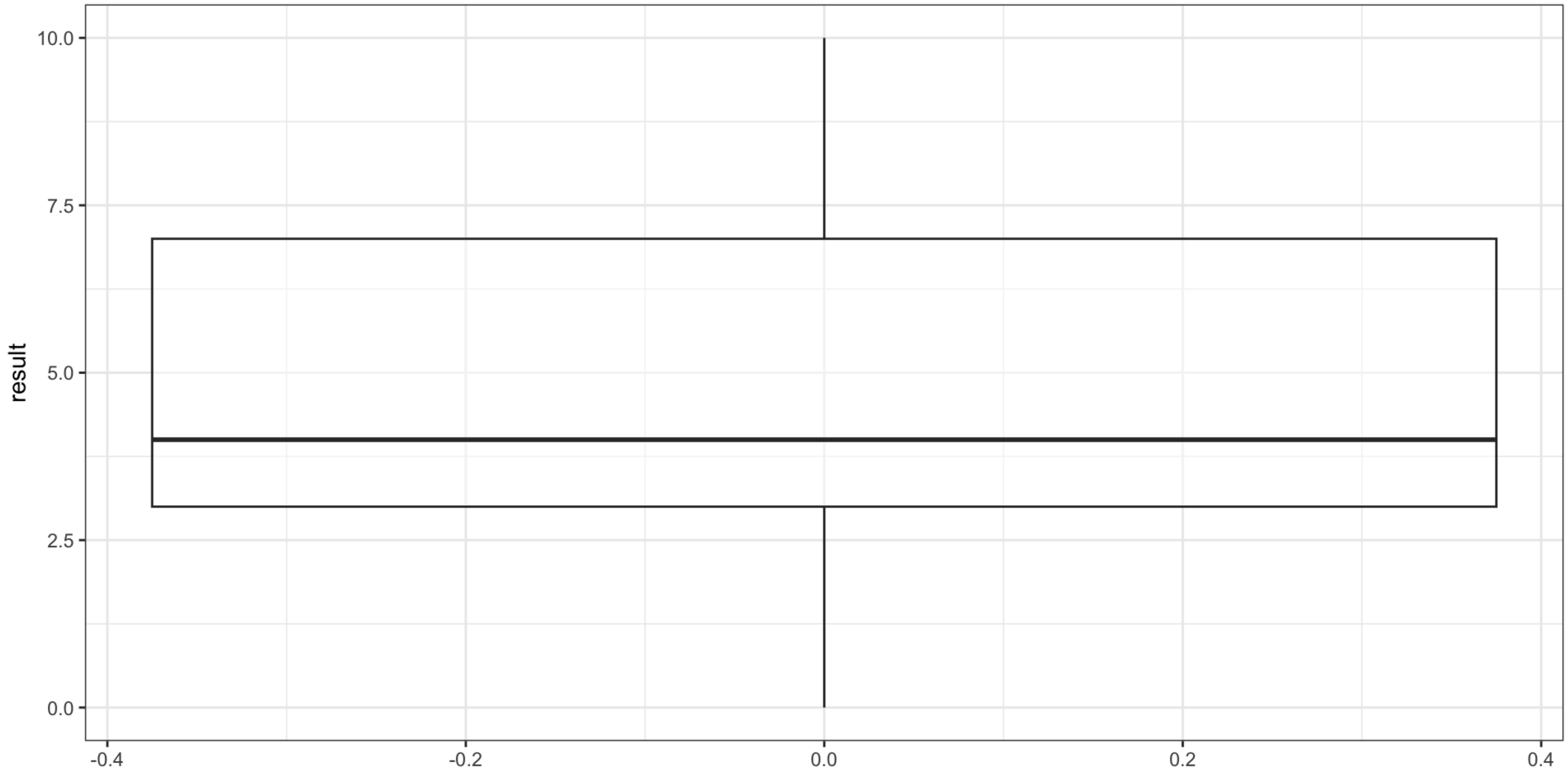
Boxplot example 1

Let's use a boxplot to explore how the continuous variable `result` is distributed in the autism dataset.

- in the aesthetic mapping we specify only `x` (continuous variable)
- switch to vertical orientation with `coord_flip()`

```r
autism_pids %>%
  ggplot(aes(x = result )) +
  geom_boxplot(alpha=0.5)+
  # switch to vertical orientation
  coord_flip() +
  theme_bw()
```

Boxplot example 1



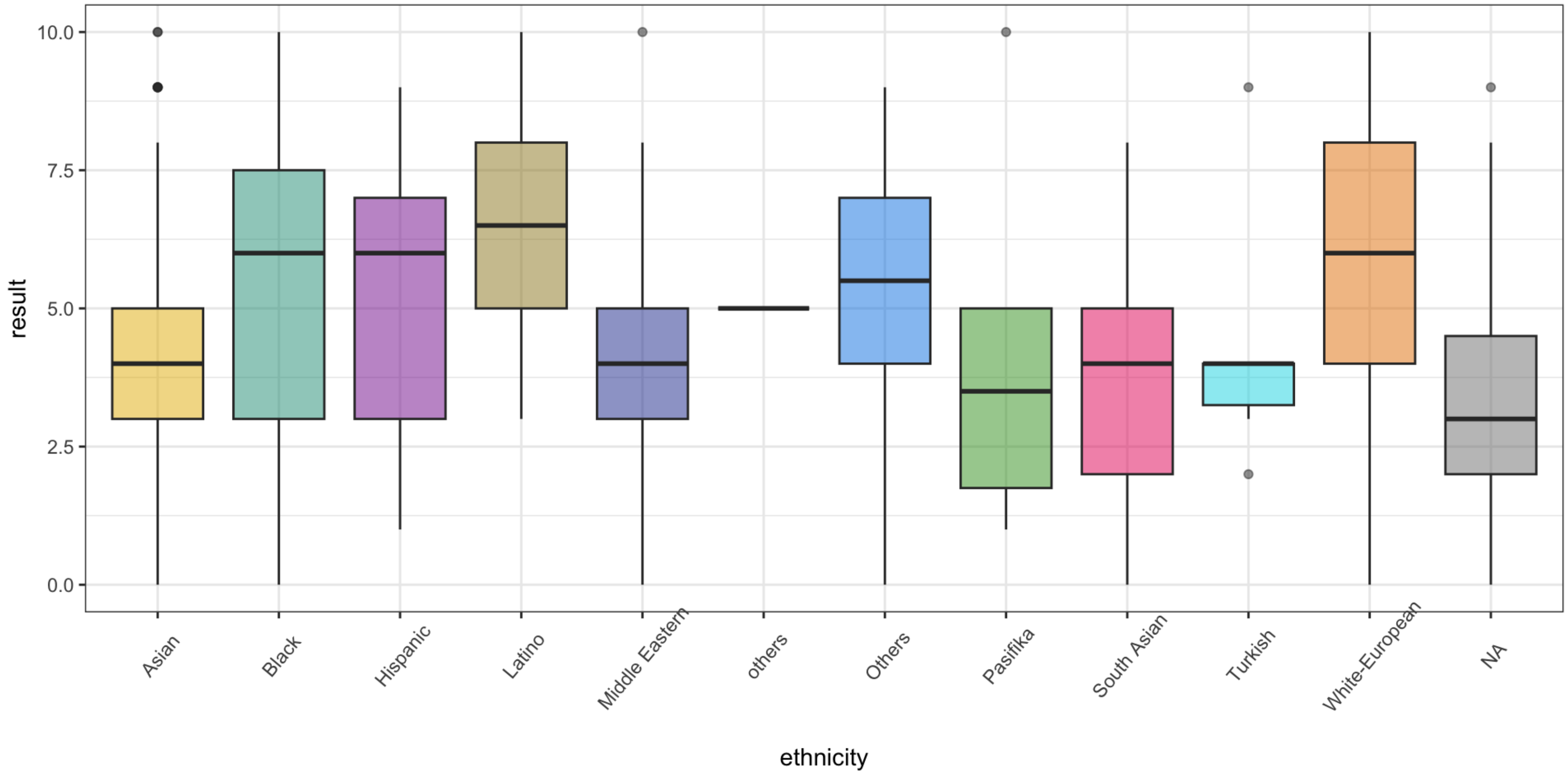
Boxplot example 2

Let's also explore how the continuous variable `result` is distributed by the categorical variable (factor) `ethnicity`.

- in the aesthetic mapping we specify `y` (continuous variable), plus `x` and `fill` (categorical variable)
- make x axis labels readable with `theme(axis.text.x (...))` layer
- I specify colors that I had previously saved in a color palette `contrast_cols_palette`

```r
autism_pids %>%
  ggplot(aes(x = ethnicity,  y= result, fill = ethnicity)) +
  geom_boxplot(alpha=0.5)+
  scale_fill_manual(values =  contrast_cols_palette)   +
  theme_bw()+
  # make x axis labes readable
  theme(axis.text.x = element_text(angle=50, vjust=0.75)) +
  # drop legend and Y-axis title
  theme(legend.position = "none")
```

Boxplot example 2



Violin plot

Similarly, the violin plot is an interesting alternative to show the distribution of a continuous variable along one or more categorical variables. Here, the kernel density plot shows the smoothed curve of the **probability density function (PDF)** of the data.

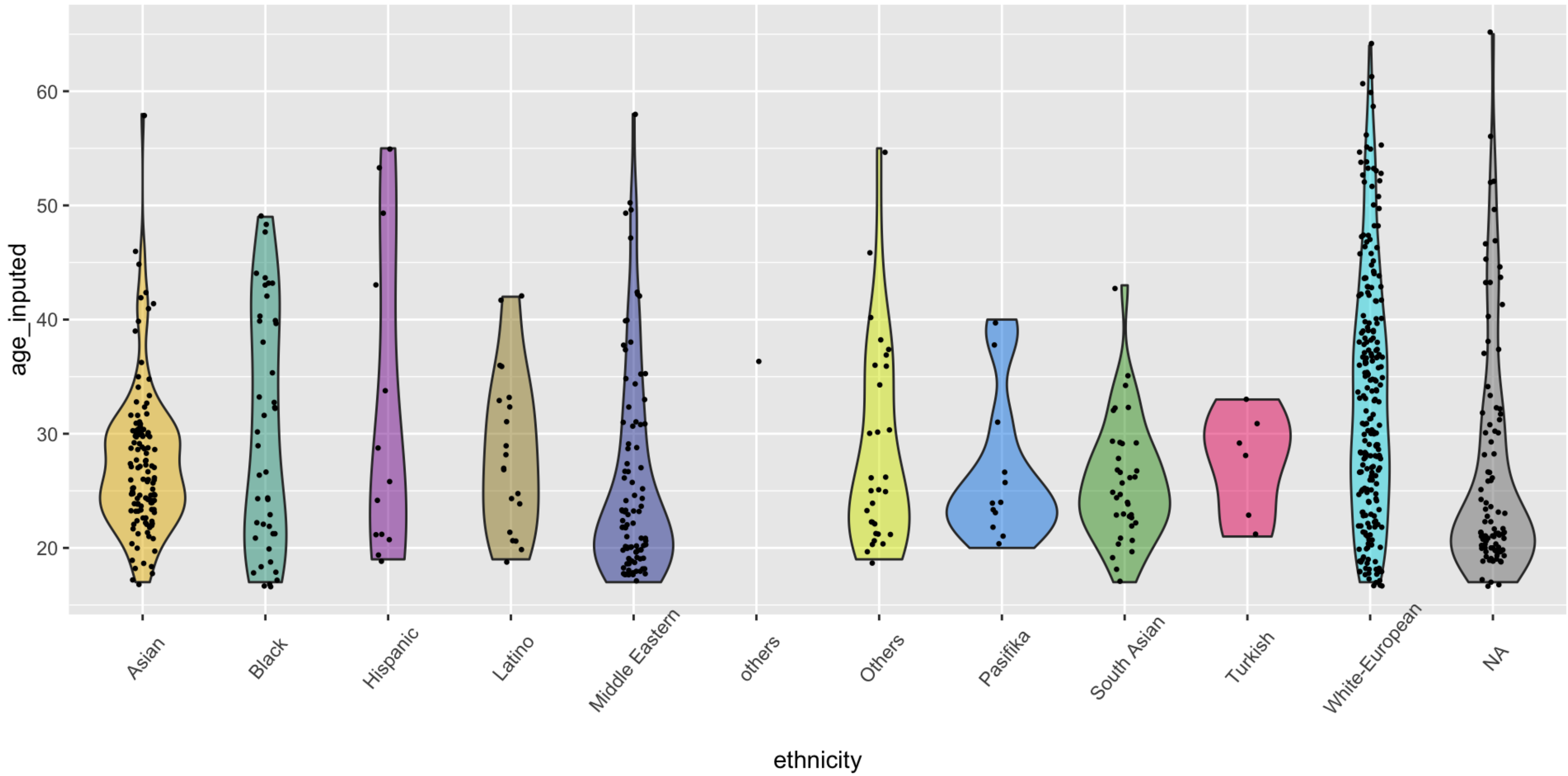
Compared to the box plot, a violin plot provides more information, as it shows not only the summary statistics but also the shape and variability of the data (i.e. helping to identify any **skewness** or **multimodality** in the data).

Violin plot example

- it requires the `geom_violin` function
- it can be enriched by adding with other `geoms`, such as points, lines, or box plots, to create more complex and informative plots
- let's add points with the `geom_point` layer

```
autism_pids %>%
  ggplot(mapping = aes(y = age_inputed, x = ethnicity, fill = ethnicity)) +
  geom_violin(alpha=0.5) +
  geom_point(position = position_jitter(width = 0.1), size = 0.5)+
  scale_fill_manual(values =  contrast_cols_palette)  +
  # make x axis labes readable
  theme(axis.text.x = element_text(angle=50, vjust=0.75)) +
  # drop legend and Y-axis title
  theme(legend.position = "none")
```

Violin plot example



SAVING & EXPORTING OUTPUT ARTIFACTS

Saving one plot

If I want to use these output files later, I can easily save in the output folder created at the beginning.

- save a plot with `ggplot2::ggsave`
- specifying the output directory with `here::here(...)`

```
ggsave (hist_plot,
        filename = here::here("practice",  "data_output", "hist_plot.png"))
```

Saving a `.Rds` data file.

- save a dataframe with `base::saveRDS`
- specifying the output directory with `here::here(...)`

```
saveRDS (object = autism_pids,
         file =  here::here("practice",  "data_output", "autism_pids_v2.Rds"))
```

- (later) load a saved dataframe with `base::readRDS`

```
# to load it later I will use
readRDS(here::here("practice",  "data_output", "autism_pids_v2.Rds"))
```


notice I renamed while saving: next time I load it it will be called “autism_pids_v2”

Final thoughts/recommendations

- Always **read the documentation** (`?package::function`, especially the examples at the bottom)
- Always **inspect the data** / variables **before** and **after** making changes
- **Better rename** (i.e. create a new R object) when you recode/manipulate a column or a dataset
    - *(this promotes reproducibility, since you will be able to retrace your coding steps)*
- Always plot distributions for **visual data exploration**
- Make changes in **small increments** (like we saw in building `ggplot2` graph in subsequent layers)