GC3 Workshop: Data Wrangling (Tutorial)

Winnie Wing-Yee Tse

2022-10-20

Contents

Objectives

1. Learn to use *dplyr* and *tidyr* for data manipulation
2. Use *ggplot2* to make graphs and plots
3. Make reproducible HTML/PDF reports with R

Data

We will use the R built-in datasets for all the following examples. Typically in R, a column of a data frame indicates a variable (e.g., `group`, `ID`), and a row contains the data for one observation.

```
# dataset 1: sleep (built in R)
data(sleep) # ?sleep for more details
# dataset 2: beliefs about crying scale
bacs <- read.csv("https://osf.io/6gsy8/download")
```

- Quick tip: as a sanity check, I always use `names()` or `colnames()` to have a quick glance of the names of all variables in a data frame

```
names(bacs)
```

```
##   [1] "V1"         "V5"         "Q6"         "IERQ__1"    "IERQ__2"
##   [6] "IERQ__3"    "IERQ__4"    "IERQ__5"    "IERQ__6"    "IERQ__7"
##  [11] "IERQ__8"    "IERQ__9"    "IERQ__10"   "IERQ__11"   "IERQ__12"
##  [16] "IERQ__13"   "IERQ__14"   "IERQ__15"   "IERQ__16"   "IERQ__17"
##  [21] "IERQ__18"   "IERQ__19"   "IERQ__20"   "BigV__1"    "BigV__2"
##  [26] "BigV__3"    "BigV__4"    "BigV__5"    "BigV__6"    "BigV__7"
##  [31] "BigV__8"    "BigV__9"    "BigV__10"   "BigV__11"   "BigV__12"
##  [36] "BigV__13"   "BigV__14"   "BigV__15"   "BigV__16"   "BigV__17"
##  [41] "BigV__18"   "BigV__19"   "BigV__20"   "BigV__21"   "BigV__22"
##  [46] "BigV__23"   "BigV__24"   "BigV__25"   "BigV__26"   "BigV__27"
##  [51] "BigV__28"   "BigV__29"   "BigV__30"   "BigV__31"   "BigV__32"
##  [56] "BigV__33"   "BigV__34"   "BigV__35"   "BigV__36"   "BigV__37"
##  [61] "BigV__38"   "BigV__39"   "BigV__40"   "BigV__41"   "BigV__42"
##  [66] "BigV__43"   "BigV__44"   "Jackson_1"  "Jackson_2"  "Jackson_3"
##  [71] "Jackson_4"  "Jackson_5"  "Jackson_6"  "Jackson_7"  "Jackson_8"
##  [76] "Jackson_9"  "Jackson_10" "Jackson_11" "Jackson_12" "Jackson_13"
##  [81] "Jackson_14" "Jackson_15" "Jackson_16" "Jackson_17" "Jackson_18"
##  [86] "Jackson_19" "Jackson_20" "Jackson_21" "Jackson_22" "Jackson_23"
##  [91] "Jackson_24" "Jackson_25" "Jackson_26" "Jackson_27" "Jackson_28"
##  [96] "Jackson_29" "Jackson_30" "DERSsf__1"  "DERSsf__2"  "DERSsf__3"
## [101] "DERSsf__4"  "DERSsf__5"  "DERSsf__6"  "DERSsf__7"  "DERSsf__8"
## [106] "DERSsf__9"  "DERSsf__10" "DERSsf__11" "DERSsf__12" "DERSsf__13"
## [111] "DERSsf__14" "DERSsf__15" "DERSsf__16" "DERSsf__17" "DERSsf__18"
## [116] "BACS_1"     "BACS_4"     "BACS_8"     "BACS_10"    "BACS_18"
## [121] "BACS_19"    "BACS_26"    "BACS_27"    "BACS_29"    "BACS_30"
## [126] "BACS_31"    "BACS_33"    "BACS_35"    "BACS_38"    "TAS_1"
## [131] "TAS_2"      "TAS_3"      "TAS_4"      "TAS_5"      "TAS_6"
```

```
## [136] "TAS_7"      "TAS_8"      "TAS_9"      "TAS_10"     "TAS_11"
## [141] "TAS_12"     "TAS_13"     "TAS_14"     "TAS_15"     "TAS_16"
## [146] "TAS_17"     "TAS_18"     "TAS_19"     "TAS_20"     "EES_1"
## [151] "EES_2"      "EES_3"      "EES_4"      "EES_5"      "EES_6"
## [156] "EES_7"      "EES_8"      "EES_9"      "EES_10"     "EES_11"
## [161] "EES_12"     "EES_13"     "EES_14"     "EES_15"     "EES_16"
## [166] "EES_17"     "CPS__1"     "CPS__2"     "CPS__3"     "CPS__4"
## [171] "CPS__5"     "CPS__6"     "CPS__7"     "CPS__8"     "CPS__9"
## [176] "CPS__10"    "CPS__11"    "CPS__12"    "CPS__13"    "CPS__14"
## [181] "CPS__15"    "CPS__16"    "CPS__17"    "CPS__18"    "CPS__19"
## [186] "CPS__20"    "CPS__21"    "CPS__22"    "CPS__23"    "CPS__24"
## [191] "CPS__25"    "CPS__26"    "CPS__27"    "Q13"        "Q14"
## [196] "Gender"     "Age"        "Ethnic"
```

Data Manipulation

The pipe operator, `%>%` (under the *magrittr* and *dplyr* package), allows us to pass the result of a function to the other function in sequence.

- Example: use the base R function `subset()` to select the data of group 1 and the person with ID 1

```
# without piping
subset(subset(sleep, group == 1), ID == 1)
```

```
##   extra group ID
## 1   0.7     1  1
```

```
# with piping
subset(sleep, group == 1) %>%
  subset(ID == 1)
```

```
##   extra group ID
## 1   0.7     1  1
```

The above is just a simple example for illustration. Without piping, we can use `subset(sleep, group == 1 & ID == 1)` to achieve the same purpose.

Manipulate cases

`filter()`: retains all rows that satisfy your conditions

```
# retain all observations in group 1
sleep %>%
  filter(group == 1)
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
```

```
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
```

```
# retain all observations that had a decrease in hours of sleep
# (i.e., `extra < 0`)
sleep %>%
  filter(extra < 0)
```

```
##   extra group ID
## 1  -1.6     1  2
## 2  -0.2     1  3
## 3  -1.2     1  4
## 4  -0.1     1  5
## 5  -0.1     2  5
```

```
# retain all observations in group 1 that had a decrease in hours of sleep
sleep %>%
  filter(group == 1, extra < 0)
```

```
##   extra group ID
## 1  -1.6     1  2
## 2  -0.2     1  3
## 3  -1.2     1  4
## 4  -0.1     1  5
```

```
# exclude all observations in group 2
sleep %>%
  filter(!group == 2)
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
```

`arrange()`: order the rows by the values of selected columns

```r
# sort the data by group in ascending order
sleep %>%
  arrange(group)
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
```

```r
# sort the data by group in descending order
sleep %>%
  arrange(desc(group))
```

```
##    extra group ID
## 1    1.9     2  1
## 2    0.8     2  2
## 3    1.1     2  3
## 4    0.1     2  4
## 5   -0.1     2  5
## 6    4.4     2  6
## 7    5.5     2  7
## 8    1.6     2  8
## 9    4.6     2  9
## 10   3.4     2 10
## 11   0.7     1  1
## 12  -1.6     1  2
## 13  -0.2     1  3
## 14  -1.2     1  4
## 15  -0.1     1  5
## 16   3.4     1  6
## 17   3.7     1  7
## 18   0.8     1  8
## 19   0.0     1  9
## 20   2.0     1 10
```

```r
# sort the data by group then by the descending order of ID
sleep %>%
  arrange(group, desc(ID))
```

```
##    extra group ID
## 1    2.0     1 10
## 2    0.0     1  9
## 3    0.8     1  8
## 4    3.7     1  7
## 5    3.4     1  6
## 6   -0.1     1  5
## 7   -1.2     1  4
## 8   -0.2     1  3
## 9   -1.6     1  2
## 10   0.7     1  1
## 11   3.4     2 10
## 12   4.6     2  9
## 13   1.6     2  8
## 14   5.5     2  7
## 15   4.4     2  6
## 16  -0.1     2  5
## 17   0.1     2  4
## 18   1.1     2  3
## 19   0.8     2  2
## 20   1.9     2  1
```

`add_row()`: add one or more rows of data to an existing data frame

```r
# add two observations
# (1) ID = 11, group = 2, extra = 0
# (2) ID = 12, group = 2, extra = 1
sleep %>%
  add_row(extra = c(0, 1),
          group = as.factor(c(2, 2)),
          ID = as.factor(c(11, 12)))
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
```

```
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21   0.0     2 11
## 22   1.0     2 12
```

Why do we need `as_factor()`? Read "Debugging tips" for more details.

Manipulate variables

`select()`: select variables in a data frame

- It also allows you to order the variables (columns)

```
# select ID and extra
sleep %>%
  select(ID, extra)
```

```
##    ID extra
## 1   1   0.7
## 2   2  -1.6
## 3   3  -0.2
## 4   4  -1.2
## 5   5  -0.1
## 6   6   3.4
## 7   7   3.7
## 8   8   0.8
## 9   9   0.0
## 10 10   2.0
## 11  1   1.9
## 12  2   0.8
## 13  3   1.1
## 14  4   0.1
## 15  5  -0.1
## 16  6   4.4
## 17  7   5.5
## 18  8   1.6
## 19  9   4.6
## 20 10   3.4
```

```
# exclude group and ID
sleep %>%
  select(!c(group, ID))
```

```
##    extra
## 1    0.7
## 2   -1.6
## 3   -0.2
## 4   -1.2
```

```
## 5   -0.1
## 6    3.4
## 7    3.7
## 8    0.8
## 9    0.0
## 10   2.0
## 11   1.9
## 12   0.8
## 13   1.1
## 14   0.1
## 15  -0.1
## 16   4.4
## 17   5.5
## 18   1.6
## 19   4.6
## 20   3.4
```

```
# reorder the variables: ID, group, extra
sleep %>%
  select(ID, group, extra)
```

```
##    ID group extra
## 1   1     1   0.7
## 2   2     1  -1.6
## 3   3     1  -0.2
## 4   4     1  -1.2
## 5   5     1  -0.1
## 6   6     1   3.4
## 7   7     1   3.7
## 8   8     1   0.8
## 9   9     1   0.0
## 10 10     1   2.0
## 11  1     2   1.9
## 12  2     2   0.8
## 13  3     2   1.1
## 14  4     2   0.1
## 15  5     2  -0.1
## 16  6     2   4.4
## 17  7     2   5.5
## 18  8     2   1.6
## 19  9     2   4.6
## 20 10     2   3.4
```

- `contains()`: select variables that match a pattern

```
# select all big 5 questions from bacs and demographic variables
# save the data subset to bacs_sub
bacs_sub <- bacs %>%
  select(contains("BigV"), Gender, Age, Ethnic)
names(bacs_sub)
```

```
##  [1] "BigV__1"  "BigV__2"  "BigV__3"  "BigV__4"  "BigV__5"  "BigV__6"
```

```
##  [7] "BigV__7"  "BigV__8"  "BigV__9"  "BigV__10" "BigV__11" "BigV__12"
## [13] "BigV__13" "BigV__14" "BigV__15" "BigV__16" "BigV__17" "BigV__18"
## [19] "BigV__19" "BigV__20" "BigV__21" "BigV__22" "BigV__23" "BigV__24"
## [25] "BigV__25" "BigV__26" "BigV__27" "BigV__28" "BigV__29" "BigV__30"
## [31] "BigV__31" "BigV__32" "BigV__33" "BigV__34" "BigV__35" "BigV__36"
## [37] "BigV__37" "BigV__38" "BigV__39" "BigV__40" "BigV__41" "BigV__42"
## [43] "BigV__43" "BigV__44" "Gender"   "Age"      "Ethnic"
```

- `everything()`: select all variables – save you time from copying and pasting!

```
# bring the demographic variables to the front
bacs_sub2 <- bacs_sub %>%
  select(Gender, Age, Ethnic, everything())
names(bacs_sub2)
```

```
##  [1] "Gender"   "Age"      "Ethnic"   "BigV__1"  "BigV__2"  "BigV__3"
##  [7] "BigV__4"  "BigV__5"  "BigV__6"  "BigV__7"  "BigV__8"  "BigV__9"
## [13] "BigV__10" "BigV__11" "BigV__12" "BigV__13" "BigV__14" "BigV__15"
## [19] "BigV__16" "BigV__17" "BigV__18" "BigV__19" "BigV__20" "BigV__21"
## [25] "BigV__22" "BigV__23" "BigV__24" "BigV__25" "BigV__26" "BigV__27"
## [31] "BigV__28" "BigV__29" "BigV__30" "BigV__31" "BigV__32" "BigV__33"
## [37] "BigV__34" "BigV__35" "BigV__36" "BigV__37" "BigV__38" "BigV__39"
## [43] "BigV__40" "BigV__41" "BigV__42" "BigV__43" "BigV__44"
```

`mutate()`: create and modify columns

```
# create a new column that standardizes `extra`
sleep_std <- sleep %>%
  mutate(extra_std = (extra - mean(extra)) / sd(extra))
# check the mean and sd of the standardized variable
mean(sleep_std$extra_std)
```

```
## [1] -1.110223e-16
```

```
sd(sleep_std$extra_std)
```

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

```
# dichotomous `extra`
# assign 1 to extra >= 0 and 0 to extra < 0
sleep %>%
  mutate(extra_bin = ifelse(extra >= 0, 1, 0))
```

```
##    extra group ID extra_bin
## 1    0.7     1  1         1
## 2   -1.6     1  2         0
## 3   -0.2     1  3         0
## 4   -1.2     1  4         0
## 5   -0.1     1  5         0
```

```
## 6    3.4     1  6         1
## 7    3.7     1  7         1
## 8    0.8     1  8         1
## 9    0.0     1  9         1
## 10   2.0     1 10         1
## 11   1.9     2  1         1
## 12   0.8     2  2         1
## 13   1.1     2  3         1
## 14   0.1     2  4         1
## 15  -0.1     2  5         0
## 16   4.4     2  6         1
## 17   5.5     2  7         1
## 18   1.6     2  8         1
## 19   4.6     2  9         1
## 20   3.4     2 10         1
```

```
# replace the original column with the standardized estimates
# ** not recommended **
sleep %>%
  mutate(extra = (extra - mean(extra)) / sd(extra))
```

```
##          extra group ID
## 1  -0.41627028     1  1
## 2  -1.55605794     1  2
## 3  -0.86227414     1  3
## 4  -1.35783400     1  4
## 5  -0.81271816     1  5
## 6   0.92174133     1  6
## 7   1.07040928     1  7
## 8  -0.36671429     1  8
## 9  -0.76316217     1  9
## 10  0.22795753     1 10
## 11  0.17840155     2  1
## 12 -0.36671429     2  2
## 13 -0.21804634     2  3
## 14 -0.71360619     2  4
## 15 -0.81271816     2  5
## 16  1.41730118     2  6
## 17  1.96241702     2  7
## 18  0.02973359     2  8
## 19  1.51641315     2  9
## 20  0.92174133     2 10
```

rename(): change the names of individual variables

- `new_name = old_name` (don't worry–I'm always confused with the order and have got plenty of error messages because of putting an incorrect order)

```
sleep %>%
  rename(ID_new = ID)
```

```
##    extra group ID_new
```

```
## 1    0.7     1      1
## 2   -1.6     1      2
## 3   -0.2     1      3
## 4   -1.2     1      4
## 5   -0.1     1      5
## 6    3.4     1      6
## 7    3.7     1      7
## 8    0.8     1      8
## 9    0.0     1      9
## 10   2.0     1     10
## 11   1.9     2      1
## 12   0.8     2      2
## 13   1.1     2      3
## 14   0.1     2      4
## 15  -0.1     2      5
## 16   4.4     2      6
## 17   5.5     2      7
## 18   1.6     2      8
## 19   4.6     2      9
## 20   3.4     2     10
```

Handl NA's

`drop_na()`: drop rows where any specified column contains NA

```
# create observations in the sleep data
sleep_NA <- sleep %>%
  add_row(extra = c(NA, 1, -1),
          group = as.factor(c(2, NA, 1)),
          ID = as.factor(c(13, 14, 15)))
# drop_na will drop ID 13 and 14 but not 15
sleep_NA %>%
  drop_na()
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
```

```
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21  -1.0     1 15
```

`replace_na()`: replace NAs with specified values

```
sleep_NA %>%
  mutate(extra = replace_na(extra, 999))
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21 999.0     2 13
## 22   1.0  <NA> 14
## 23  -1.0     1 15
```

`na_if()`: convert a value to NA

- I find this function particularly useful for datasets from SPSS, Mplus, Stata, etc., which have a convention of indicating missing values with 999, 9999, or some other numeric values.

```
sleep_999 <- sleep %>%
  add_row(extra = c(999, 1, -1),
          group = as.factor(c(2, 999, 1)),
          ID = as.factor(c(13, 14, 15)))
sleep_999 %>%
  na_if(999)
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
```

```
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21    NA     2 13
## 22   1.0  <NA> 14
## 23  -1.0     1 15
```

Combine tables: `left_join()`, `right_join()`, `replace_na()`, `na_if()`

```
band_members
```

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

```
band_instruments
```

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

```
# inner_join (include all rows in x AND y)
band_members %>%
  inner_join(band_instruments, by = "name")
```

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

```r
# full_join (include all rows in x OR y)
band_members %>%
  full_join(band_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
```

```r
# left_join (include all rows in x)
band_members %>%
  left_join(band_instruments, 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
```

```r
# right_join (include all rows in y)
band_members %>%
  right_join(band_instruments, by = "name")
```

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

Data Visualization

Let's wrangle the data a little bit before plotting. Below are two specific tasks.

1. Create two new variables—the individual sum score of the responses to big five inventory (43 questions) and that of the responses to beliefs about crying scale (14 questions).
2. Recode Gender from 1, 2 to male and female, respectively
3. Dichotomous Age by the median of it (`median(bacs$Age)` is 19)

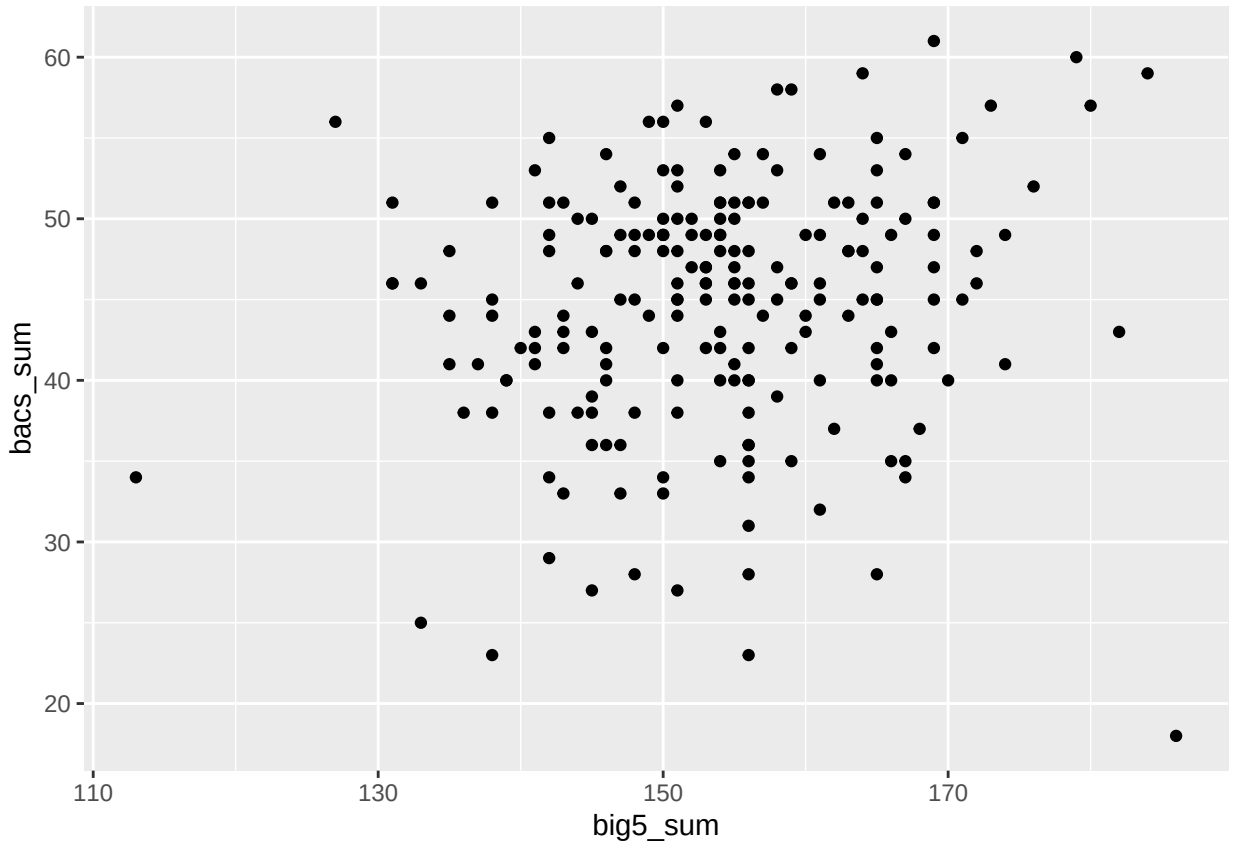
```
bacs_sum <- bacs %>%
  # calculate the sum score per individual
  mutate(big5_sum = rowSums(across(contains("BigV"))),
         bacs_sum = rowSums(across(contains("BACS"))),
         gender = recode(Gender, `1` = "Male", `2` = "Feamle"),
         age_bin = ifelse(Age >= 19, "Older than or at age 19",
                          "Younger than age 19"))
bacs_sum %>%
  select(big5_sum, bacs_sum) %>%
  head()
```

```
##   big5_sum bacs_sum
## 1      140       42
## 2      158       53
## 3      155       41
## 4      148       28
## 5      153       46
## 6      150       49
```

Scatter plot: `geom_point()`

```
bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



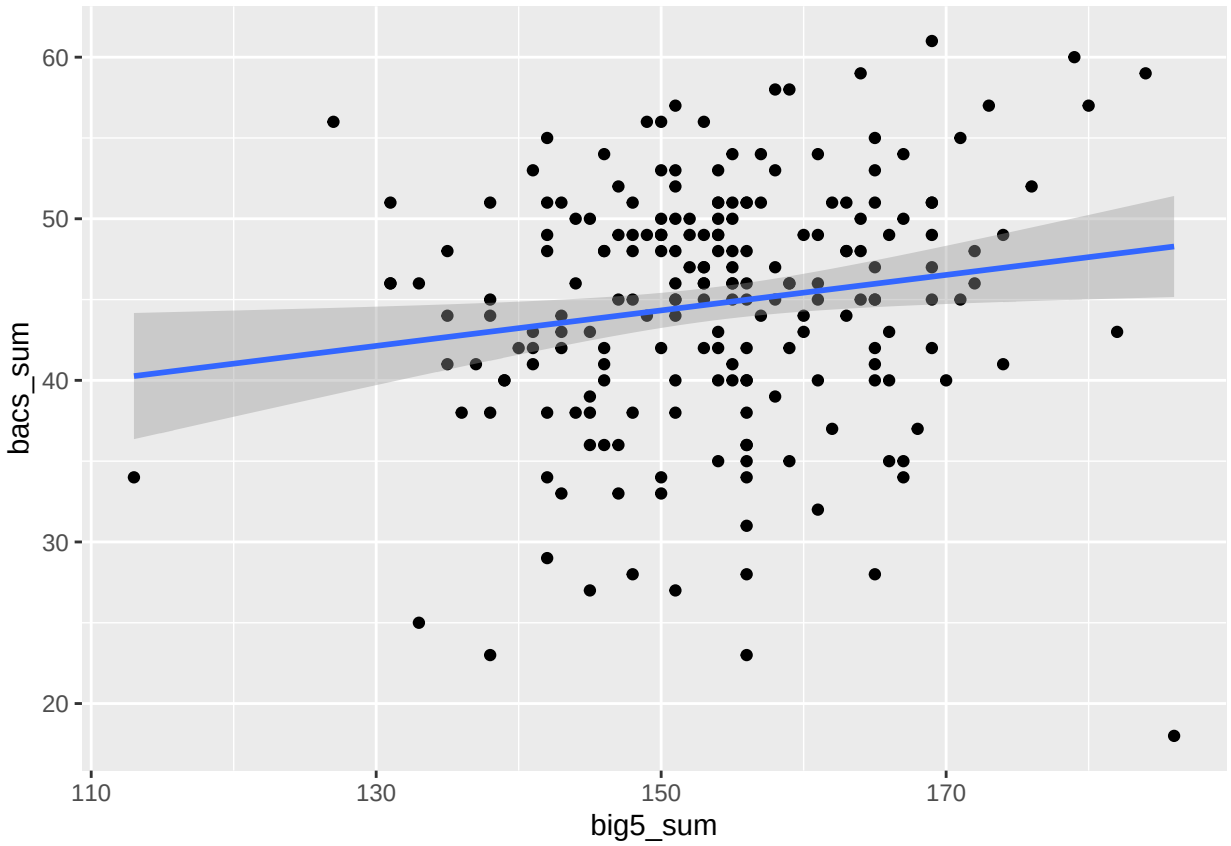
Adding a regression line: `geom_smooth()`

```
bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

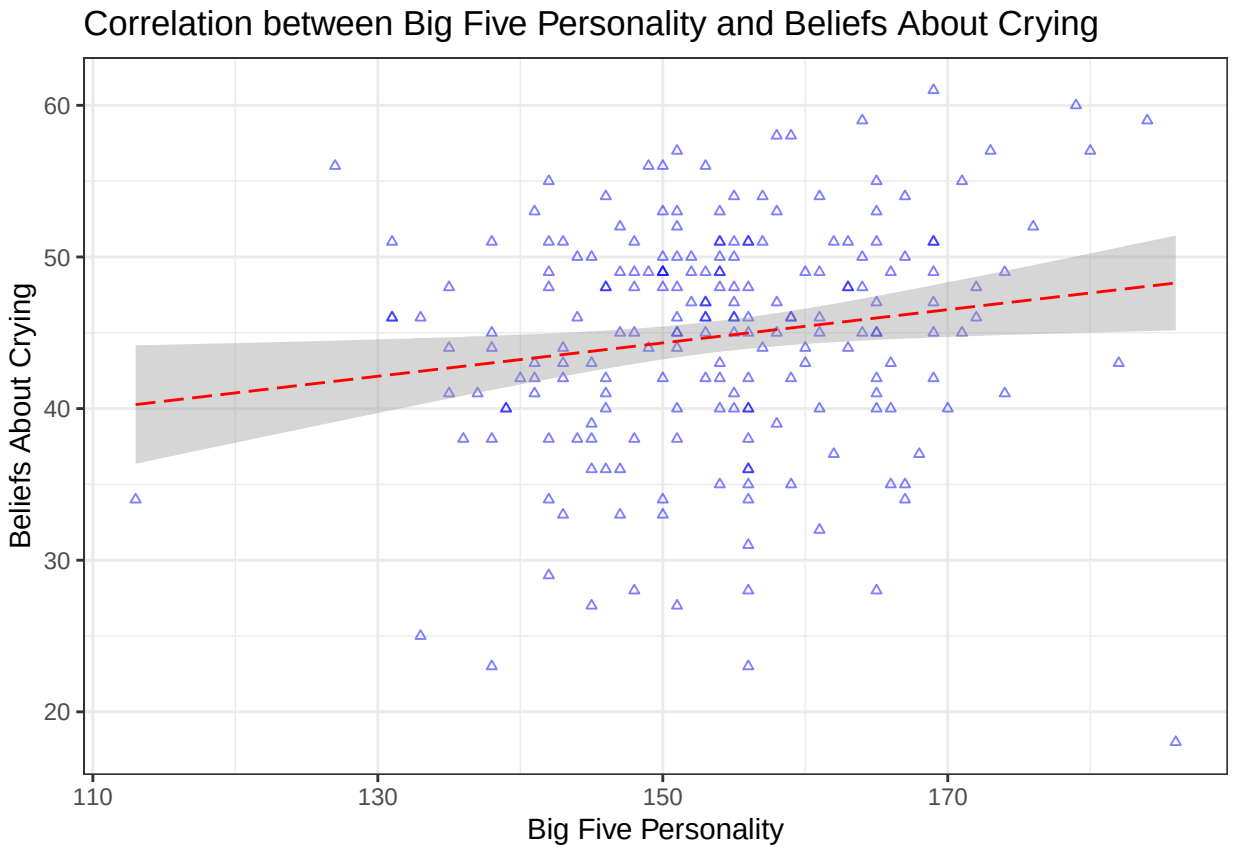


Changing Aesthetics

- Under `geom_point()`
  - `size` controls the size of the points
  - `alpha` controls the transparency of the points
  - `col` controls the color of the points
  - `shape` controls the shape of the points
- Under `geom_smooth()`
  - `size` controls the thickness of the line
  - `alpha` controls the transparency of uncertainty area
  - `col` controls the color of the line
  - `linetype` controls the shape of the points
- `labs()`: label of the axes and the title
- `theme_bw()`: one of the preset themes in ggplot2

```
(big5_bacs <- bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum)) +
  geom_point(size = 1, alpha = .5, col = "blue", shape = 2) +
  geom_smooth(method = "lm", size = .5, alpha = .4,
              col = "red", linetype = "longdash") +
  labs(x = "Big Five Personality", y = "Beliefs About Crying",
       title = "Correlation between Big Five Personality and Beliefs About Crying") +
  theme_bw())
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 2 rows containing non-finite values (stat_smooth).

## Warning: Removed 2 rows containing missing values (geom_point).
```



Export the figure: `ggsave()`

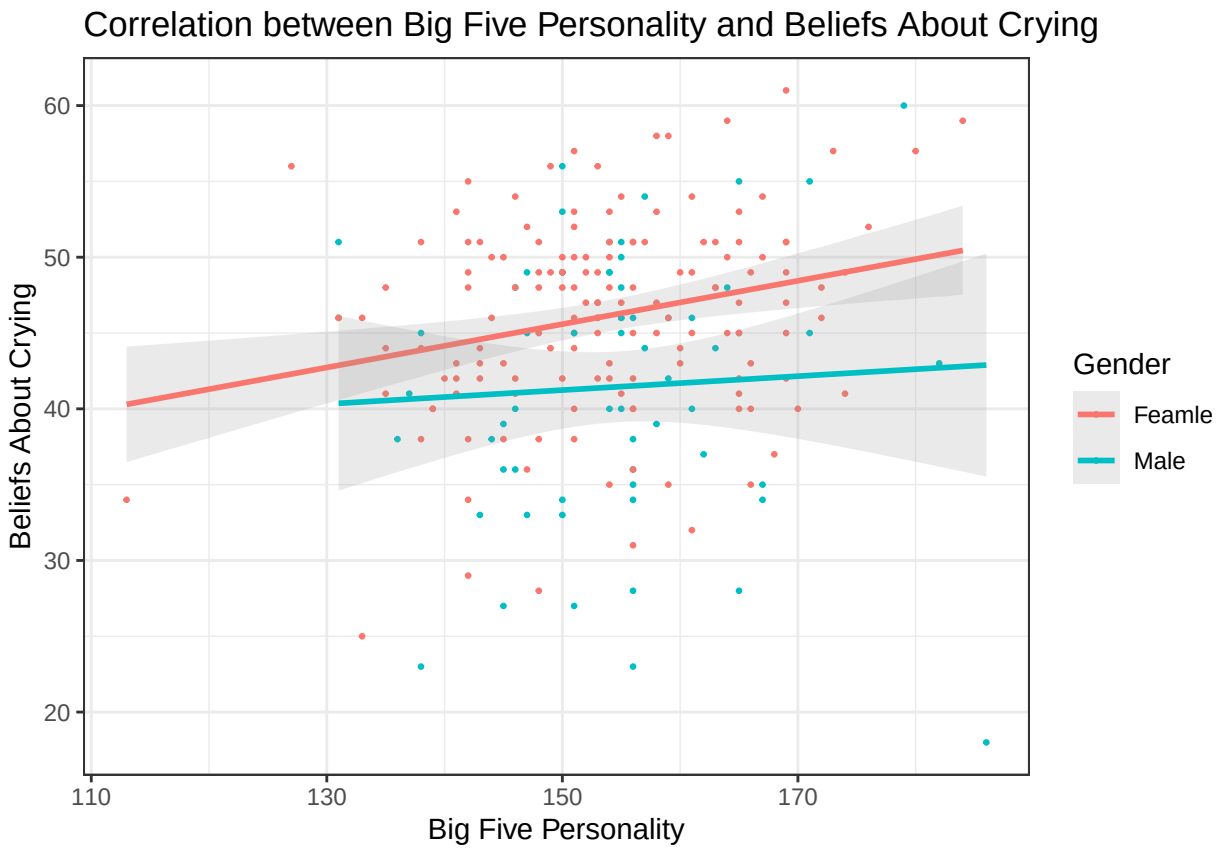
- This function is one of my personal favorites.

```
ggsave("big5_bacs.png", big5_bacs,
       width = 2000, height = 1600, units = "px")
```

Aesthetics by groups

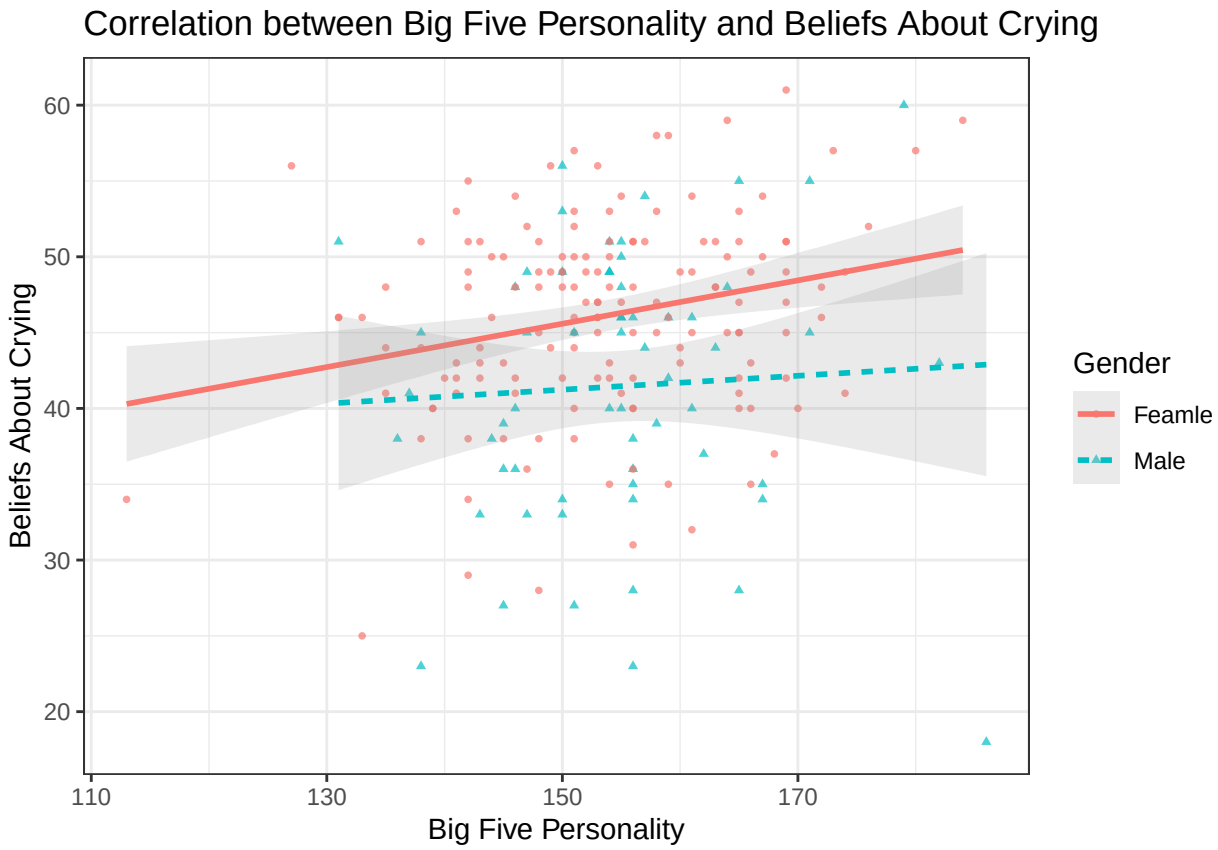
```
# colors by groups
bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum, col = gender)) +
  geom_point(size = .5) +
  geom_smooth(method = "lm", size = 1, alpha = .2) +
  labs(x = "Big Five Personality", y = "Beliefs About Crying",
       title = "Correlation between Big Five Personality and Beliefs About Crying",
       col = "Gender") +
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 2 rows containing non-finite values (stat_smooth).

## Warning: Removed 2 rows containing missing values (geom_point).
```




```
# shapes and linetypes by groups
bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum, col = gender)) +
  geom_point(size = 1, alpha = .7, aes(shape = gender)) +
  geom_smooth(method = "lm", size = 1, alpha = .2,
              aes(linetype = gender)) +
  labs(x = "Big Five Personality", y = "Beliefs About Crying",
       title = "Correlation between Big Five Personality and Beliefs About Crying",
       col = "Gender", shape = "Gender", linetype = "Gender") +
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 2 rows containing non-finite values (stat_smooth).

## Warning: Removed 2 rows containing missing values (geom_point).
```



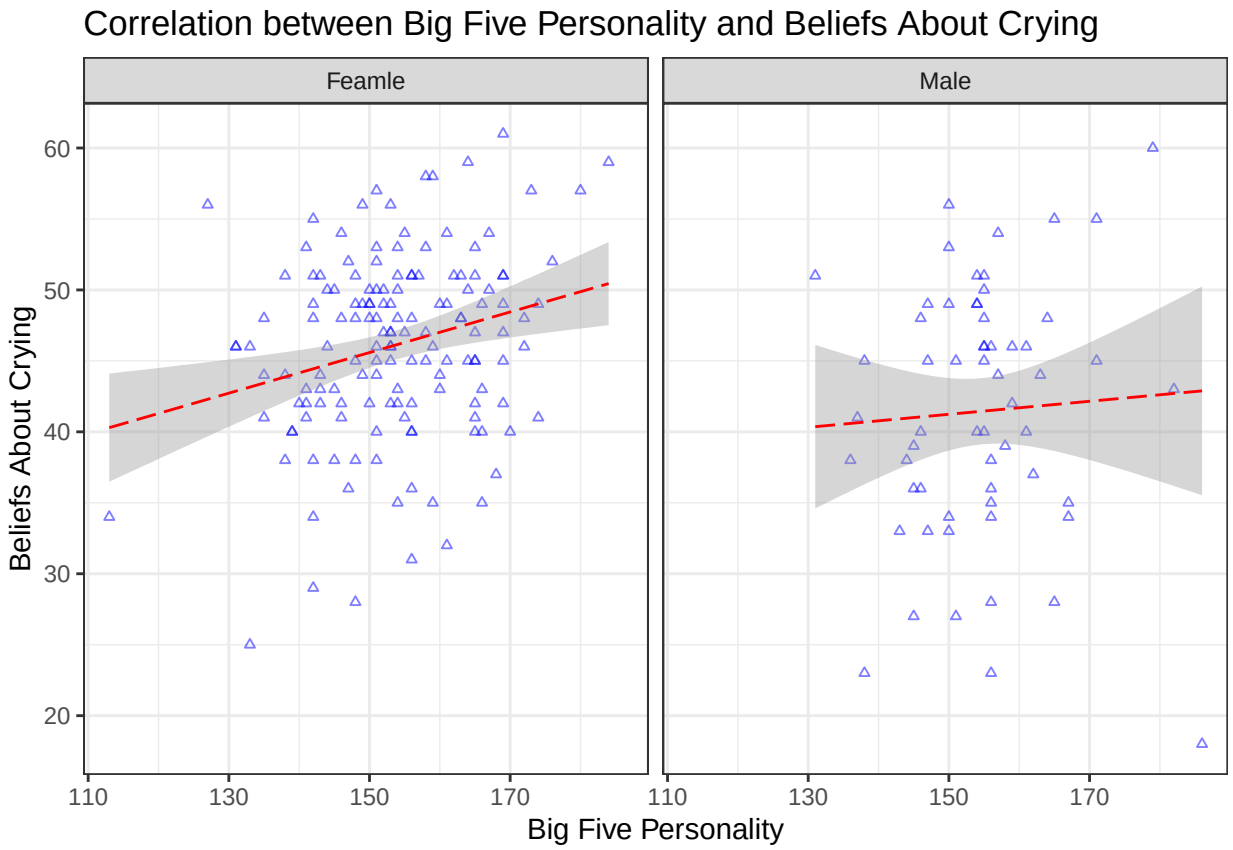
Grid of panels: `facet_grid()`

```
(bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum)) +
  geom_point(size = 1, alpha = .5, col = "blue", shape = 2) +
  geom_smooth(method = "lm", size = .5, alpha = .4,
              col = "red", linetype = "longdash") +
  facet_grid(~ gender) +
  labs(x = "Big Five Personality", y = "Beliefs About Crying",
       title = "Correlation between Big Five Personality and Beliefs About Crying") +
  theme_bw())
```

```
## `geom_smooth()` using formula 'y ~ x'
```

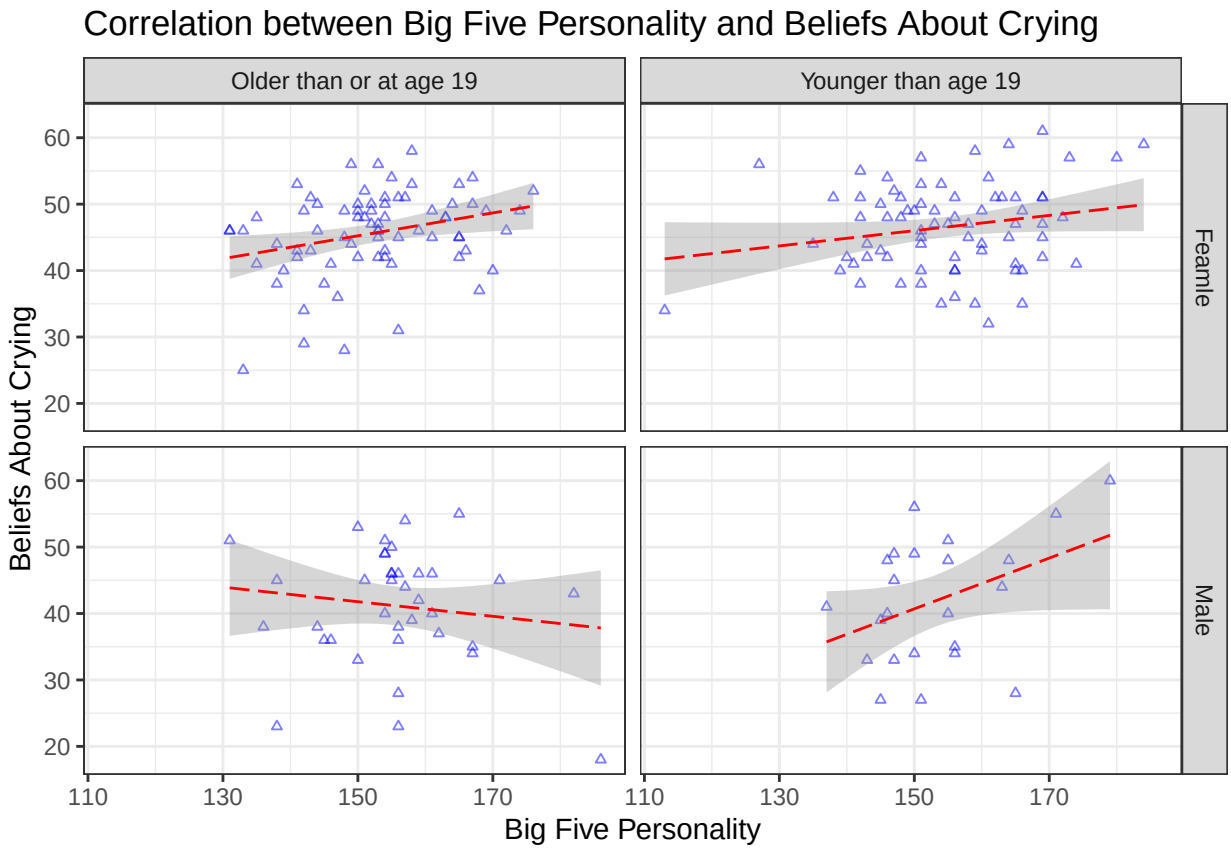
```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```




```
(bacs_sum %>%
  ggplot(aes(x = big5_sum, y = bacs_sum)) +
  geom_point(size = 1, alpha = .5, col = "blue", shape = 2) +
  geom_smooth(method = "lm", size = .5, alpha = .4,
              col = "red", linetype = "longdash") +
  facet_grid(gender ~ age_bin) +
  labs(x = "Big Five Personality", y = "Beliefs About Crying",
       title = "Correlation between Big Five Personality and Beliefs About Crying") +
  theme_bw())
```

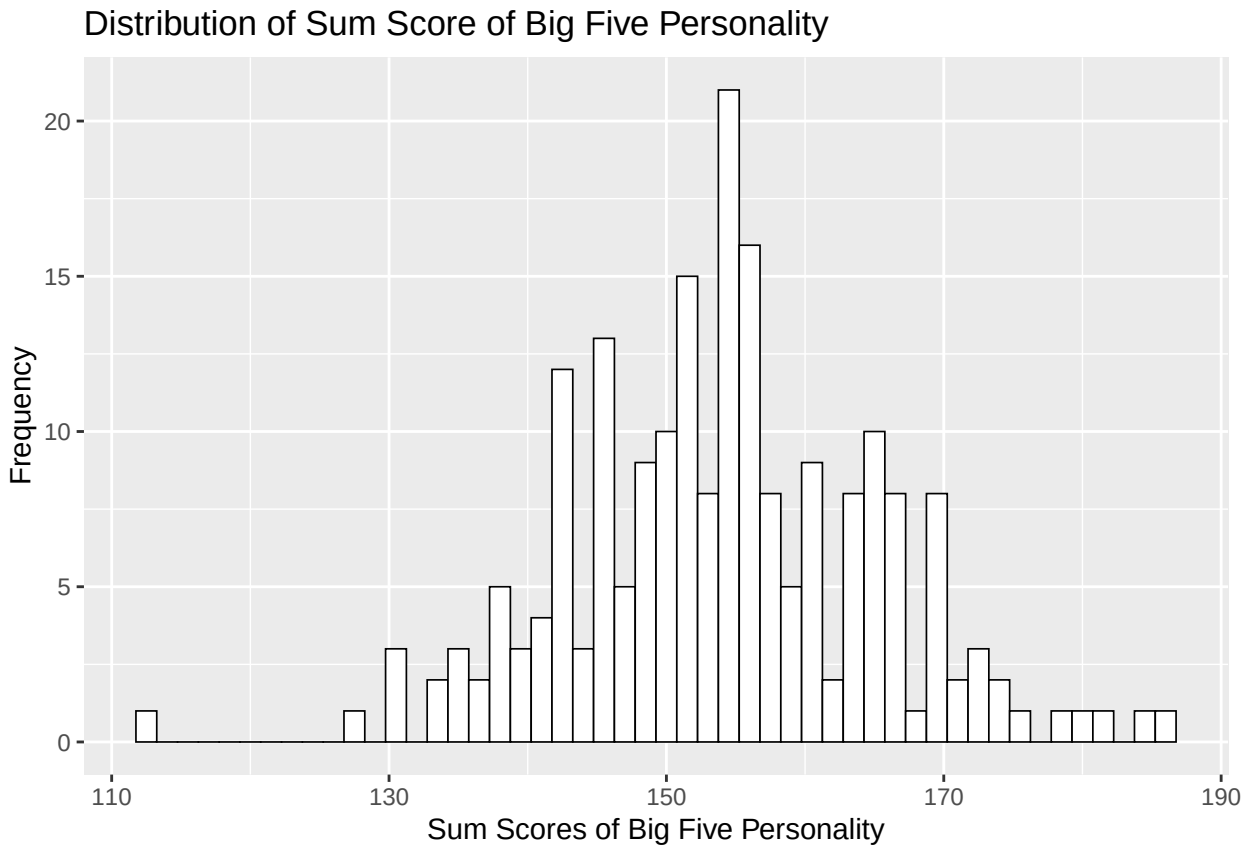
```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 2 rows containing non-finite values (stat_smooth).

## Warning: Removed 2 rows containing missing values (geom_point).
```



Histograms: `geom_histogram()`

```
bacs_sum %>%
  ggplot(aes(x = big5_sum)) +
  geom_histogram(binwidth = 1.5, col = "black", fill = "white",
                 size = .3) +
  labs(x = "Sum Scores of Big Five Personality",
       y = "Frequency",
       title = "Distribution of Sum Score of Big Five Personality")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



Debugging tips

1. Read the error message. *tidyverse* usually prints out an informative message that points you to the issue. If more help is needed, copy and paste the key message to google and you should be able to find relevant discussions.

Example: `add_row()`

```
sleep %>%
  add_row(extra = 0, group = 2, ID = 11)
```

```
## Error in ‘vec_rbind()‘:
## ! Can’t combine ‘..1$group‘ <factor<6ab52>> and ‘..2$group‘ <double>.
```

The error message tells us that R cannot combine something that is a factor with another thing that is a double.

2. The class of an R object matters. When there is a bug with data manipulation, the first thing I always do is to check what the class of a variable is using `class()` or `str()`.

```
str(sleep)
```

```
## 'data.frame':    20 obs. of  3 variables:
##  $ extra: num  0.7 -1.6 -0.2 -1.2 -0.1 3.4 3.7 0.8 0 2 ...
##  $ group: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ ID   : Factor w/ 10 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
```

```
class(2)
```

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

Structure tells us that `group` and `ID` have the class `factor`, but when we assign a value of `group = 2`, the "2" there has a class of `numeric`. The error tells us that R cannot combine a numeric value to a column that is defined as factor. The solution is to either change the class of the column or change the class of the new values.

```
# solution 1
sleep %>%
  add_row(extra = 0, group = as.factor(2), ID = as.factor(11))
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21   0.0     2 11
```

```
# solution 2
sleep %>%
  mutate_all(as.numeric) %>% # coerce all variables into numeric
  add_row(extra = 0, group = 2, ID = 11)
```

```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
```

```
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
## 21   0.0     2 11
```