Intro To Visualization In R - Static Plots - 2

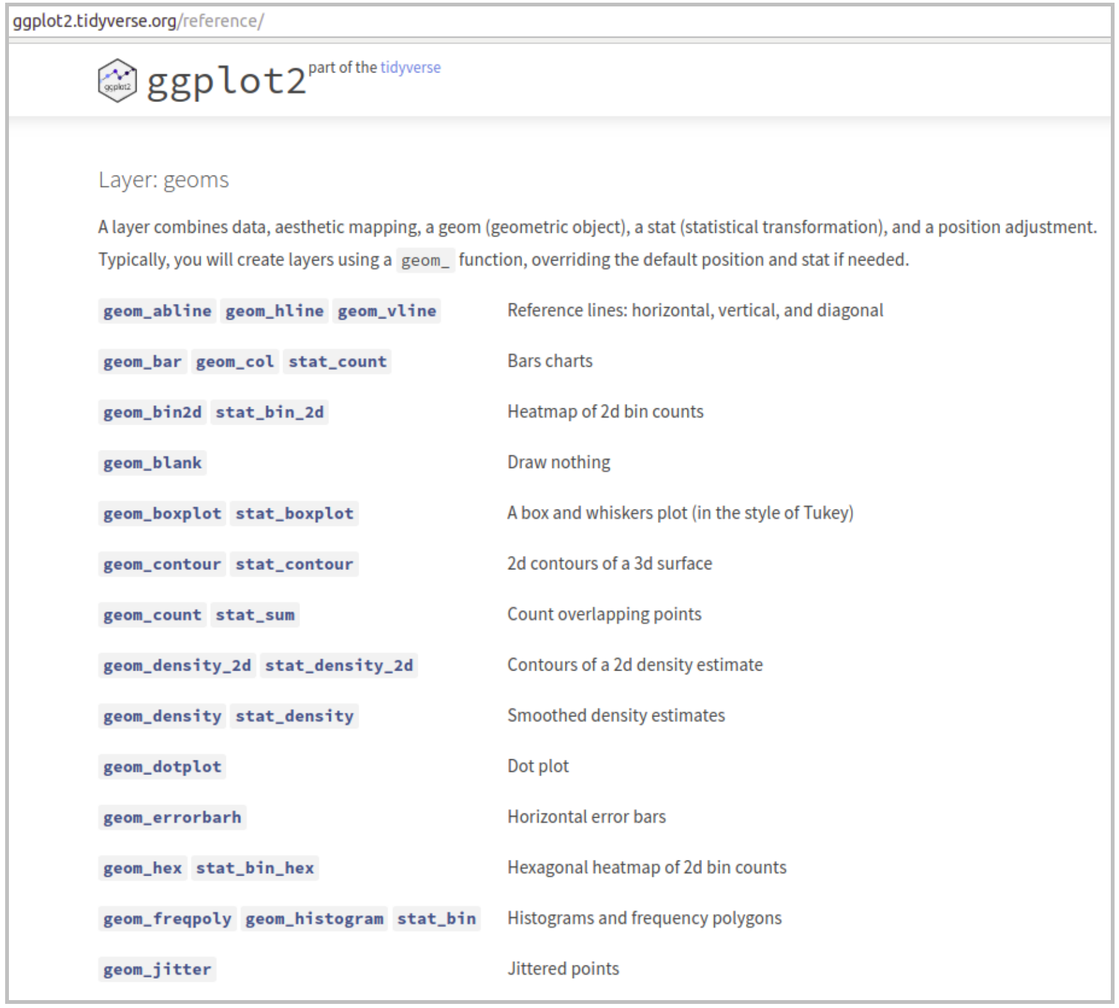
One should look for what is and not what he thinks should be – Albert Einstein

Module completion checklist

Objective	Complete
Transform data using tidyverse to prepare for compound visualizations	
Visualize the transformed data in a boxplot and a scatterplot with ggplot2	

Data prep for `ggplot2`

- As noticed earlier, each chart in a `ggplot2` package is like a layered cake, with elements built over a base
  - `ggplot2` is a part of the `tidyverse` collection in R packages
  - `ggplot2` works best with **tidy data**, allowing us to create complex multi-layered visualizations when **summarizing** and **transforming** data



Creating complex plots using `ggplot2`

- We have already created a simple histogram and scatterplot for a single variable from our dataset
- But the next step is to **compare** the distributions of variables
- To do so, we will create a series of **normalized boxplots**, allowing us to compare each variable's relative variation in data
- We will then create a **complex scatterplot** of the normalized values

Set up: load tidyverse

- Let's load the `tidyverse` package that includes `ggplot2`

```
# Load tidyverse library
library(tidyverse)
```

- We will use the same `ggplot2` theme

```
# Save our custom `ggplot` theme to a variable.
my_ggtheme = theme_bw() +
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        legend.text = element_text(size = 16),
        legend.title = element_text(size = 18),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

Set up: data conversion with `gather`

- Let's convert the dataset from **wide** to **long** using `gather()` for easy visualization
- Creating a boxplot in `ggplot2` requires the data to be in this format

```
health_subset_long = health_subset %>%
  gather(key = "variable",
         value = "value")
```

- The latest `tidyverse` version has `pivot_long()` which does the same work as `gather()` but is still under development

```
# Inspect the first few observations.
head(health_subset_long)
```

```
  variable value
1      age    67
2      age    61
3      age    80
4      age    49
5      age    79
6      age    81
```

```
# Inspect the last few observations.
tail(health_subset_long)
```

```
       variable    value
15325       bmi 18.60000
15326       bmi 28.89324
15327       bmi 40.00000
15328       bmi 30.60000
15329       bmi 25.60000
15330       bmi 26.20000
```

Set up: tidy data look

```
# Inspect the first few entries in the data.
head(health_subset_long)
```

```
  variable value
1      age    67
2      age    61
3      age    80
4      age    49
5      age    79
6      age    81
```

```
# Inspect the last few entries in the data.
tail(health_subset_long)
```

```
       variable    value
15325       bmi 18.60000
15326       bmi 28.89324
15327       bmi 40.00000
15328       bmi 30.60000
15329       bmi 25.60000
15330       bmi 26.20000
```

Module completion checklist

Objective	Complete
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize the transformed data in a boxplot and a scatterplot with ggplot2	

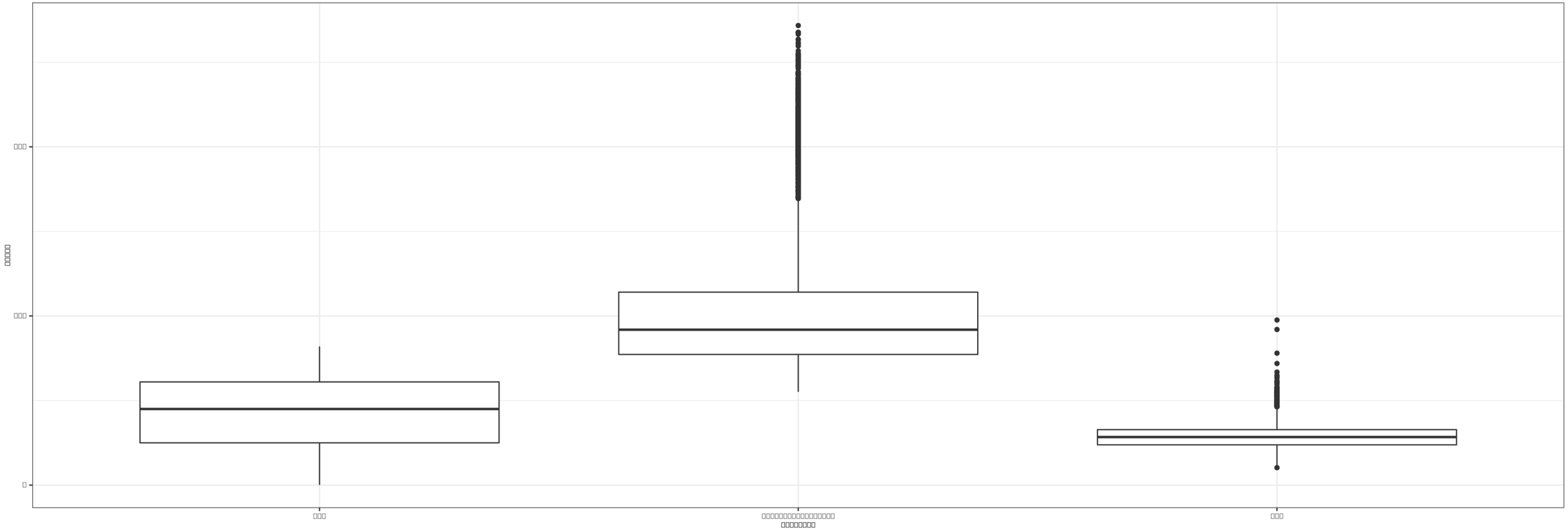
Set up & link data: make boxplots

- To add a **boxplot** layer to the base plot, use `geom_boxplot()`

```
# A basic box plot with pre-saved theme.
boxplots = ggplot(health_subset_long,   #<- set the base plot + data
                  aes(x = variable,  #<- map `variable` to x-axis
                      y = value)) +  #<- map `value` to y-axis
           geom_boxplot() +          #<- add boxplot geom
           my_ggtheme                #<- add pre-saved theme
```

Set up & link data: make boxplots (cont'd)

```
# View plot.
boxplots
```



Set up: normalize data with group_by + mutate

- The variables are on different scales, making it harder to compare them
- If we **normalize** the variables, we can more easily assess their **relative** distributions with respect to a numerical standard

```
# Let's normalize the data and then create boxplots.
health_subset_long = health_subset_long %>%
  group_by(variable) %>%                #<- group values by variable
  mutate(norm_value =                   #<- make `norm_value` column
           value/max(value,             #<- divide value by group max
                     na.rm = TRUE))     #<- don't forget the NAs!

head(health_subset_long)
```

```
# A tibble: 6 x 3
# Groups:   variable [1]
  variable value norm_value
  <chr>    <dbl>      <dbl>
1 age         67      0.817
2 age         61      0.744
3 age         80      0.976
4 age         49      0.598
5 age         79      0.963
6 age         81      0.988
```

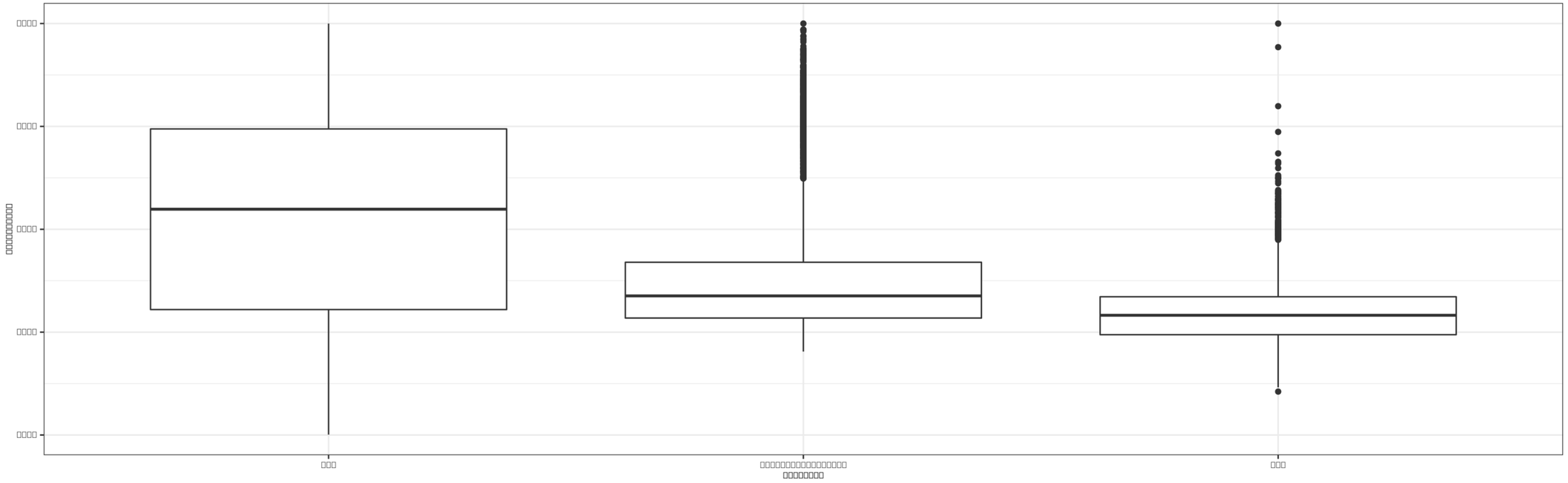
Set up: boxplot with normalized data

- Let's save the boxplot with normalized data to a variable
- Add `my_ggtheme` to it

```
# Construct the base plot for normalized data.
base_norm_plot = ggplot(health_subset_long,          #<- set data
                        aes(x = variable,          #<- map `variable`
                            y = norm_value)) +     #<- map `norm_value`
                 geom_boxplot() +                  #<- add boxplot layer
                 my_ggtheme                        #<- add theme
```

Set up: boxplot with normalized data (cont'd)

```
# View
base_norm_plot
```



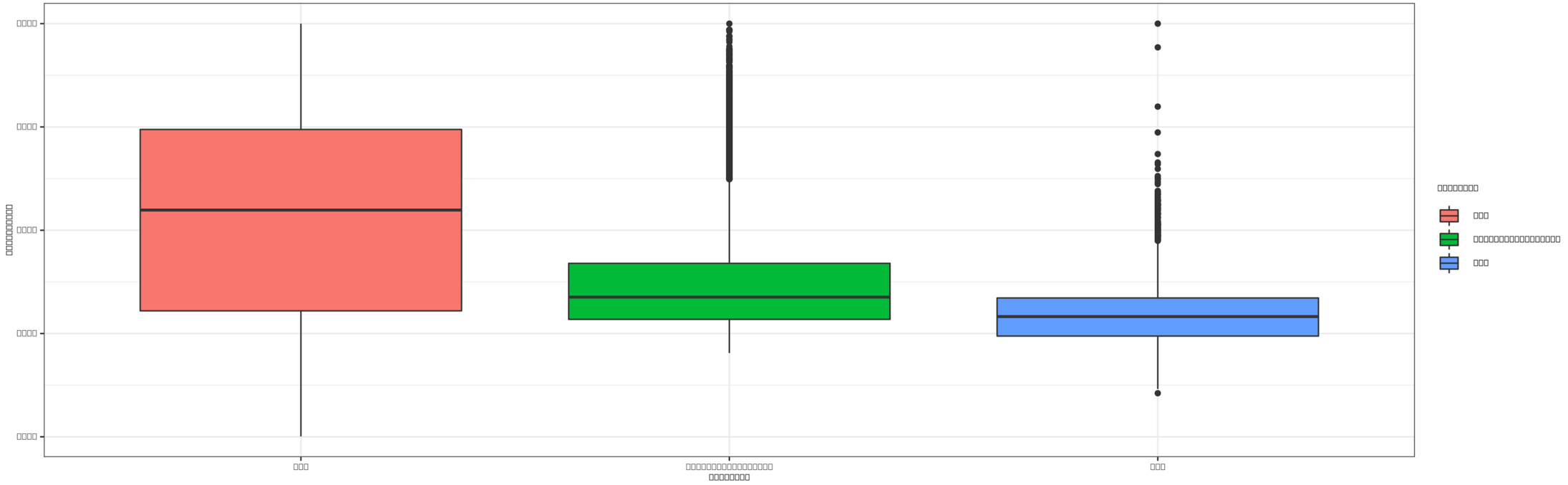
Adjust: normalized boxplot aesthetics

- Working in black and white alone, comparing the distributions of the variables may be more challenging

```
# Let's add a fill color to the plot.
boxplots_norm = ggplot(health_subset_long,          #<- set data
                       aes(x = variable,          #<- map `variable`
                           y = norm_value,        #<- map `norm_value`
                           fill = variable)) +    #<- set fill
                  geom_boxplot()  +               #<- boxplot layer
                  my_ggtheme                      #<- add theme
```

Adjust: normalized boxplot aesthetics (cont'd)

```
# View boxplot
boxplots_norm
```



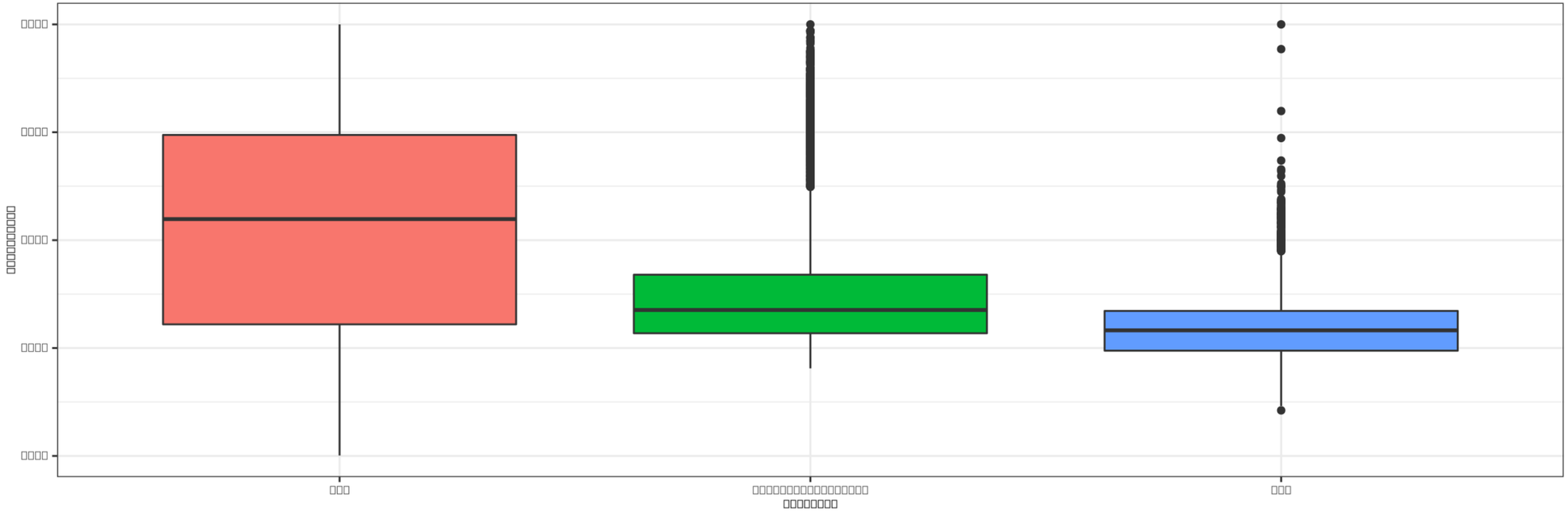
Adjust: normalized boxplot legends

- To remove the legend for `fill` color, use the function `guides()` and pass `fill = FALSE` to it
- ***Click here*** to read more about modifying legends with `guides()`

```
# Make color of fill based on variable.
boxplots_norm = boxplots_norm +     #<- set base plot
                guides(fill = FALSE) #<- remove redundant legend
```

Adjust: normalized boxplot legends (cont'd)

```
# View updated plot.
boxplots_norm
```



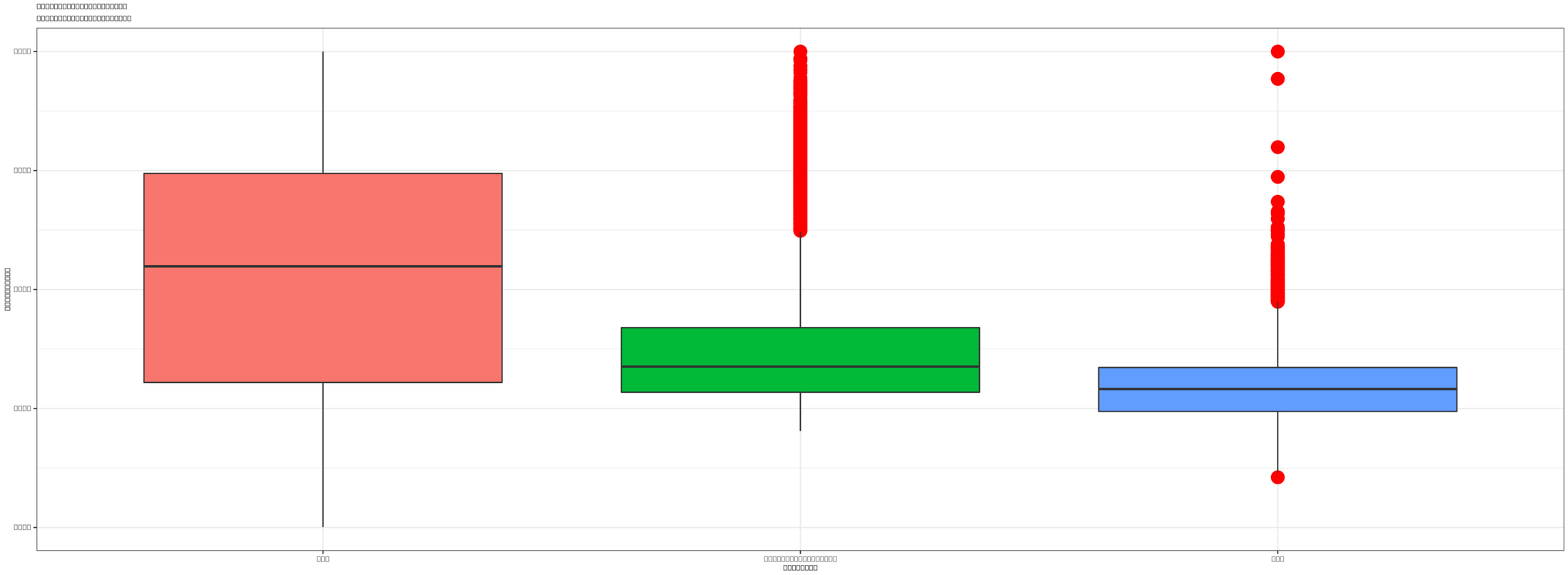
Polish: normalized boxplot details

- Finally, we can polish the visualization by adding features for improved readability and usability

```
# Make outliers stand out with red color and bigger size.
boxplots_norm = boxplots_norm +       #<- previously saved plot
  geom_boxplot(outlier.color = "red", #<- adjust outlier color
               outlier.size = 5) +    #<- adjust outlier size
  labs(title = "Health data variables",  #<- add title and subtitle
       subtitle = "Boxplot of scaled data")
```

Polish: normalized boxplot details (cont'd)

```
# View updated plot.
boxplots_norm
```



- **Chat question**: What **insights** can you gather based on this visualization?

Building a scatterplot with ggplot2

- Now that we have boxplots of the normalized variables, we can start to visualize their **relationship**
- We will use **scatterplots** to demonstrate such relationships, but we must prepare and transform our data again before building them
  - Separate `age` from all other variables
  - Ensure that each entry in `age` corresponds to an entry in all other variables
  - Convert the original wide data to a long format excluding the `age` variable

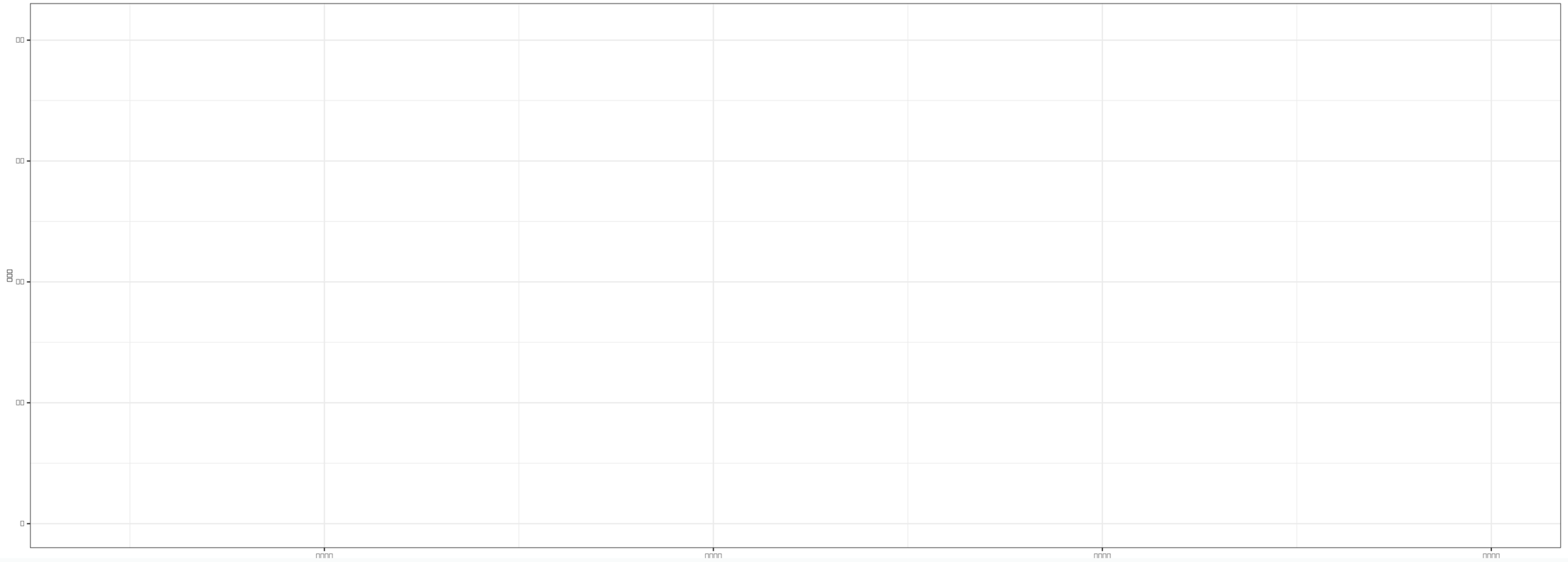
Set up: transform data for scatterplot

```
health_subset_long2 = health_subset %>%
  gather(avg_glucose_level:bmi, #<- gather all variables but `age`
         key = "variable",                          #<- set key to `variable`
         value = "value") %>%                       #<- set value to `value`
  group_by(variable) %>%
  mutate(norm_value = value/max(value, na.rm = TRUE))

# Inspect the data.
head(health_subset_long2)
```

```
# A tibble: 6 x 4
# Groups:   variable [1]
    age variable          value norm_value
  <dbl> <chr>             <dbl>      <dbl>
1    67 avg_glucose_level  229.      0.842
2    61 avg_glucose_level  202.      0.744
3    80 avg_glucose_level  106.      0.390
4    49 avg_glucose_level  171.      0.630
5    79 avg_glucose_level  174.      0.641
6    81 avg_glucose_level  186.      0.685
```

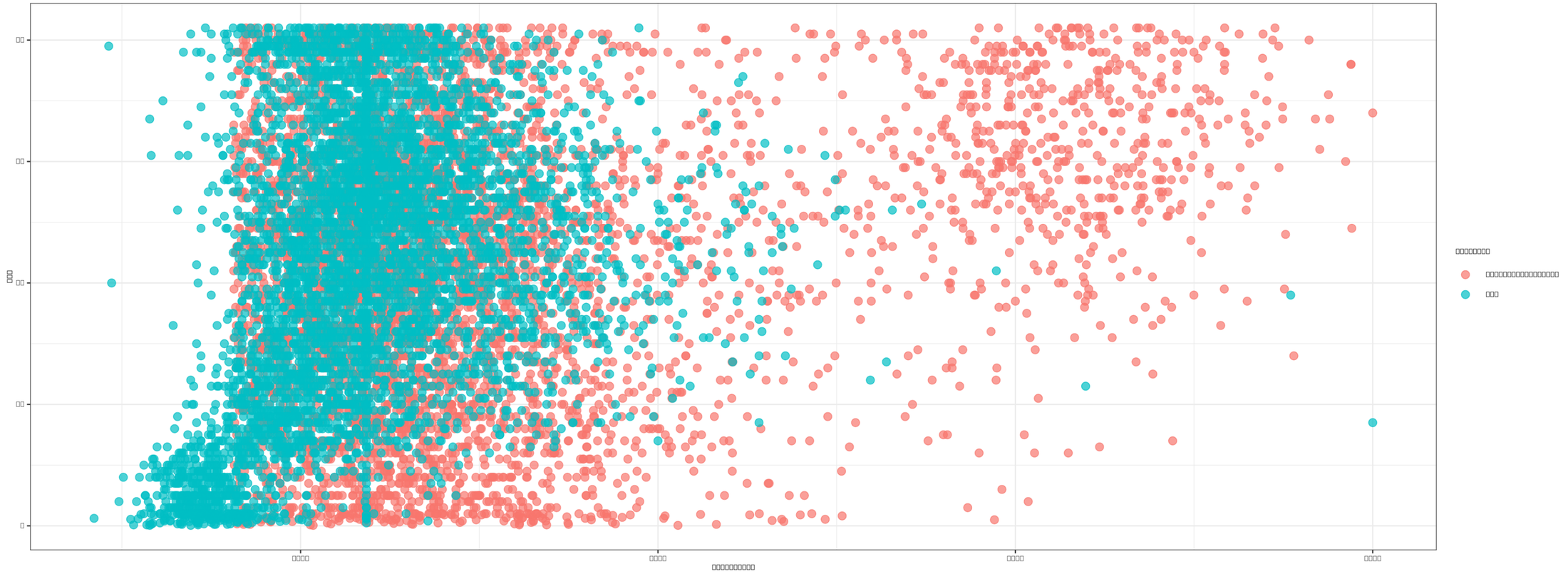
Set up & link: normalized data base plot

```
# Create a base plot.
base_norm_plot = ggplot(data = health_subset_long2,  #<- set data
                        aes(x = norm_value,        #<- set x-axis to represent normalized value
                            y = age,             #<-   y-axis to represent `age`
                            color = variable)) +  #<- set color to depend on `variable`
                        my_ggtheme                 #<- set theme

base_norm_plot
```



Set up & adjust: normalized data scatterplot

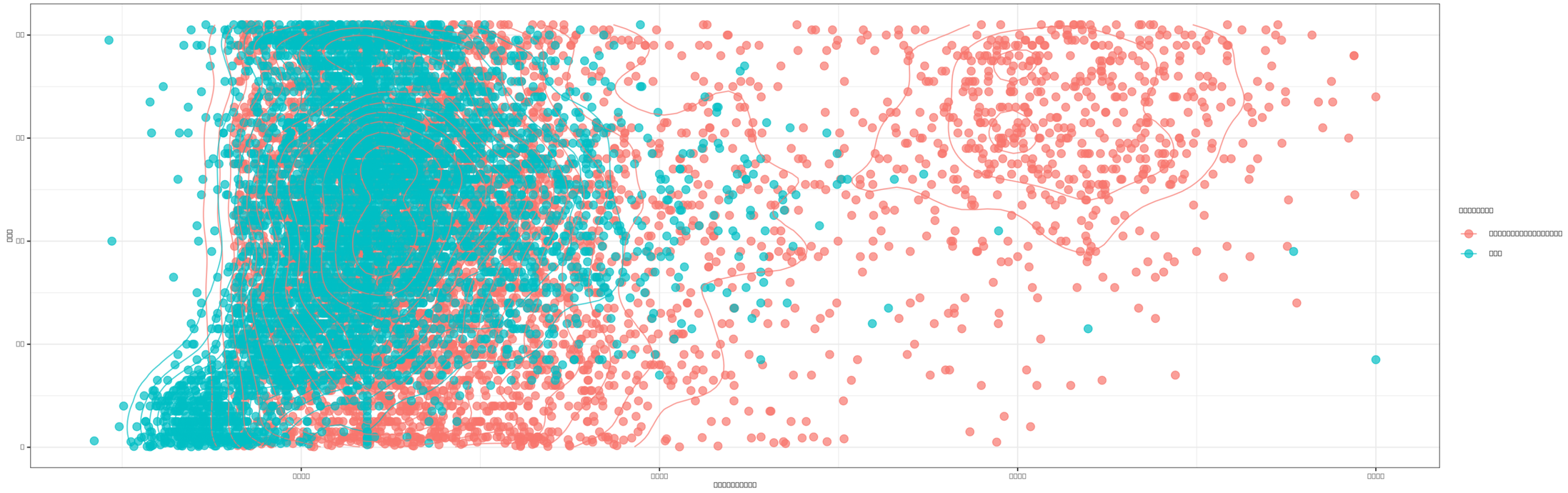
```
# Create a scatterplot.
scatter_norm = base_norm_plot +          #<- base plot
               geom_point(size = 3,      #<- add point geom with size of point = 3
                          alpha = 0.7)  #<-  make it 70% opaque

# View updated plot.
scatter_norm
```



Adjust: add density geom to scatterplot

- Now add a 2D density geom to our normalized scatterplot

```
# Adjust scatterplot to include 2D density.
scatter_norm = scatter_norm +                #<- previously saved plot
               geom_density2d(alpha = 0.7) #<- add 2D density geom with 70% opaque color

# View updated plot.
scatter_norm
```



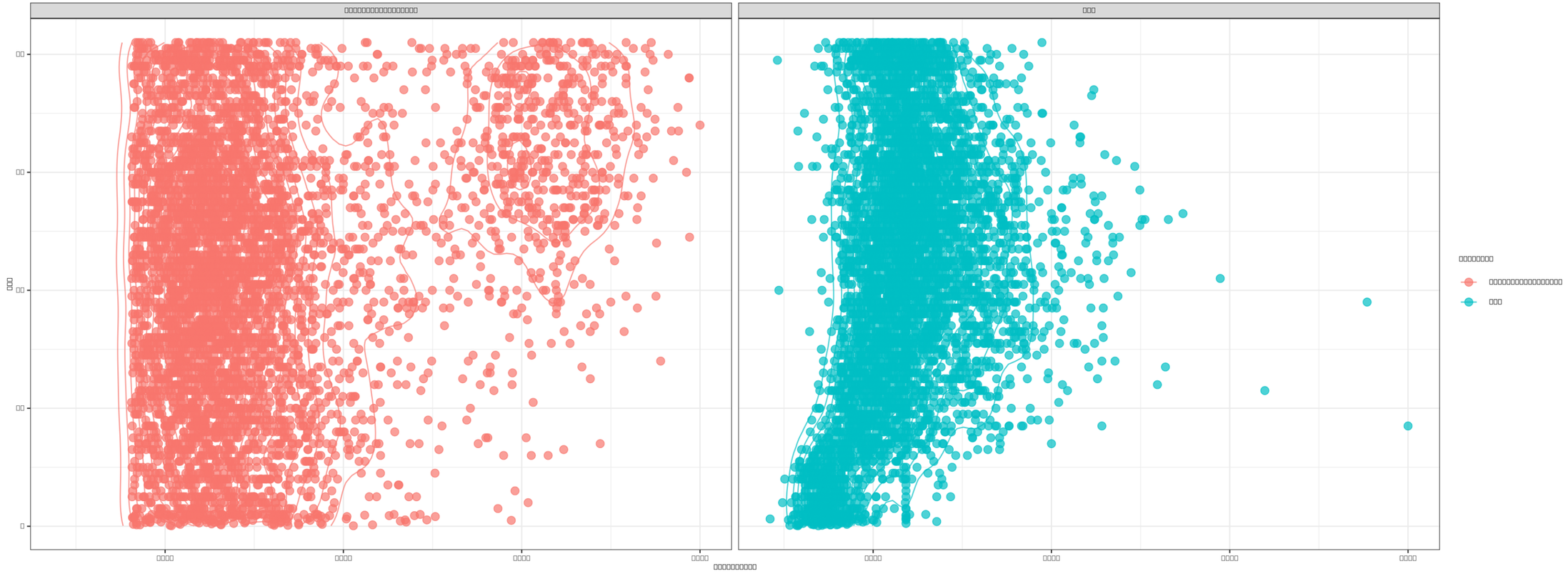
Adjust: wrap scatterplots in facets

- Notice that the previous plot is hard to decipher because of the overlapping data
- Instead of overlaying densities, let's split them into individual plots called **facets**
- These facets will appear as a tiled matrix of related visualizations
- We create them using the **`facet_wrap()`** function, which splits the data by one or more variables and plots the subsets together
- ***Click here*** to learn more about facets on the tidyverse website

```
# Wrap each scatterplot into a facet.
scatter_norm = scatter_norm +                    #<- previously saved plot
               facet_wrap(~ variable, ncol = 3) #<- wrap plots by variable into 3 columns
```

Adjust: wrap scatterplots in facets (cont'd)

```
# View updated plot.
scatter_norm
```

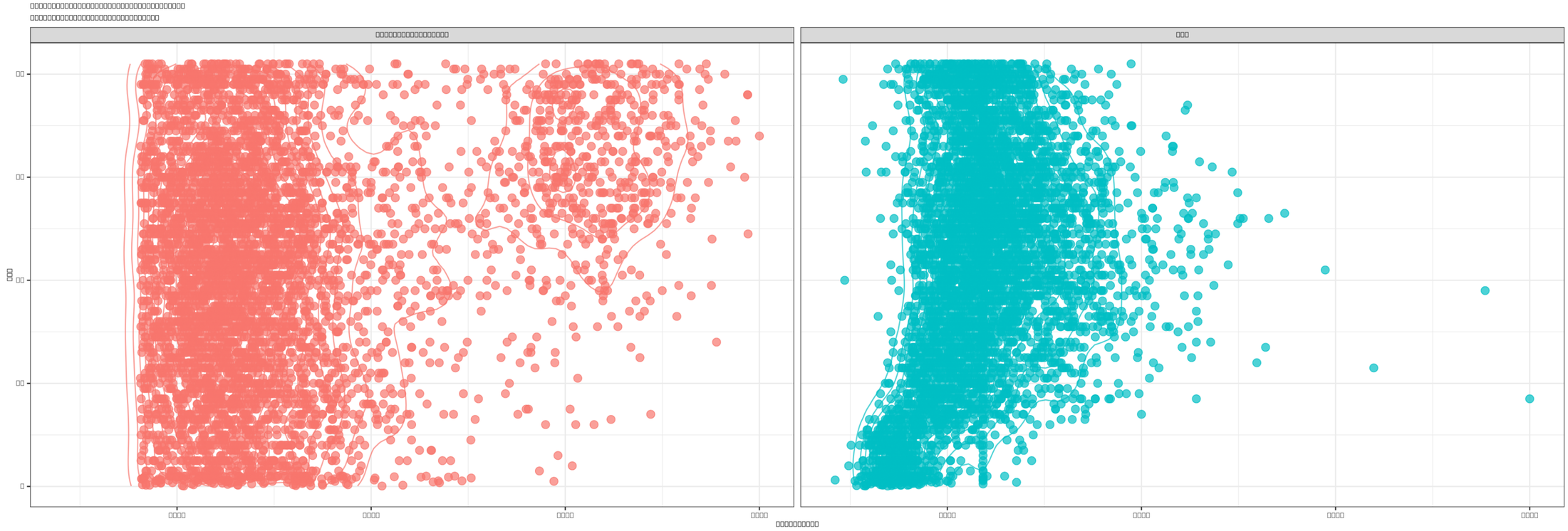


Adjust & polish: legends and text in scatterplot

```
# Add finishing touches to the plot.
scatter_norm = scatter_norm +                          #<- previously saved plot
  guides(color = FALSE) +                              #<- remove legend for color mappings
  theme(strip.text.x = element_text(size = 14)) +      #<- increase text size in strips of facets
  labs(title = "Health data: Age vs. other variables",#<- add title and subtitle
       subtitle = "2D distribution of scaled data")
```

Adjust & polish: legends and text in scatterplot

```
# View updated plot.
scatter_norm
```



Saving plots

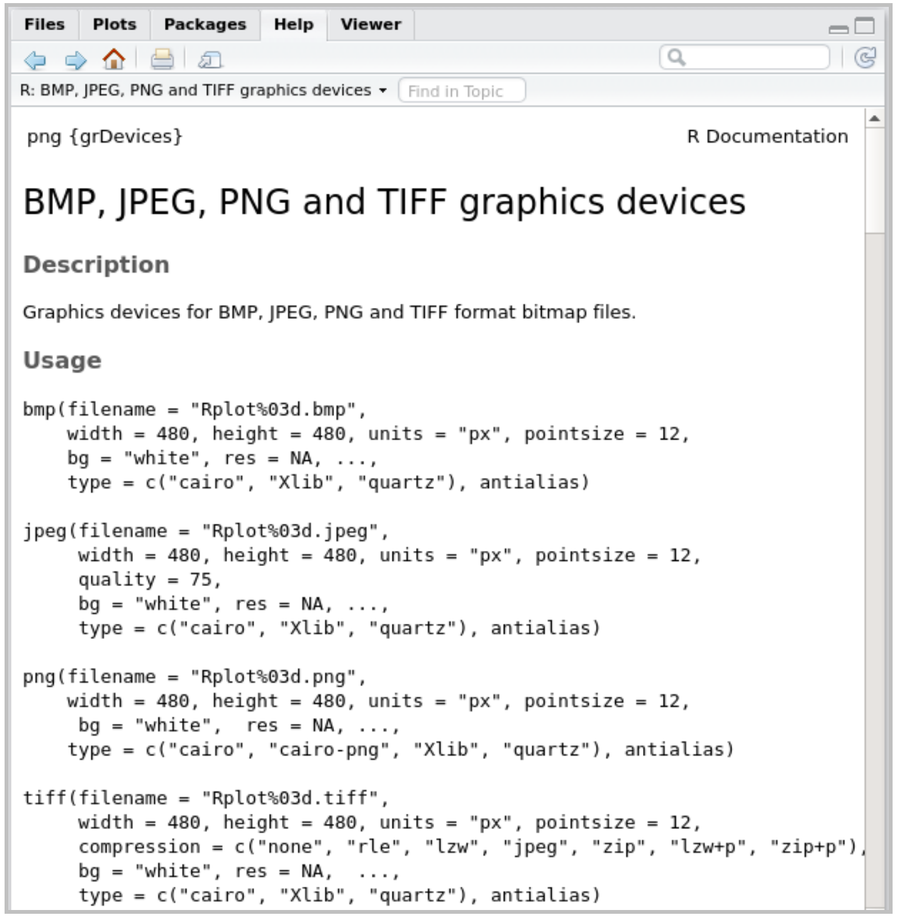
- Once we have created a plot in R, we may want to save it to a file so we can use it in another document
- The first step in saving plots is to decide which output format we want to use
- The available file formats are `PNG`, `JPEG`, `BMP`, and `PDF`
  - We will explore the `PNG` and `PDF` formats

Saving plots: export to PNG

- `png()` function opens the R graphics device and lets us save our plots in PNG file format
- The syntax of the function is:

```
png("Name_of_file.png", #<- name of file
    width = 400,        #<- width of image
    height = 300,       #<- height of image
    units = "px") #<- units for height & width

plot 1   #<- call the plot object you want to export

dev.off() #<- closes R graphics device
```

- `dev.off()` command allows clearing R graphics device so that we can continue working with our plots
- `bmp`, `jpeg`, and other graphic export commands use a similar command format



Saving plots: export to PNG (cont'd)

```
# Set working directory
# to where we want to save our plots.
setwd(plot_dir)

png("Heart_boxplots_norm.png",
    width = 1200,
    height = 600,
    units = "px")
boxplots_norm
dev.off()
```

- When the graphics device is cleared, we should get a similar message in our console

```
RStudioGD
        2
```

```
setwd(plot_dir)
png("Heart_scatterplot_norm.png",
    width = 1200,
    height = 600,
    units = "px")
scatter_norm
dev.off()
```

```
RStudioGD
        2
```

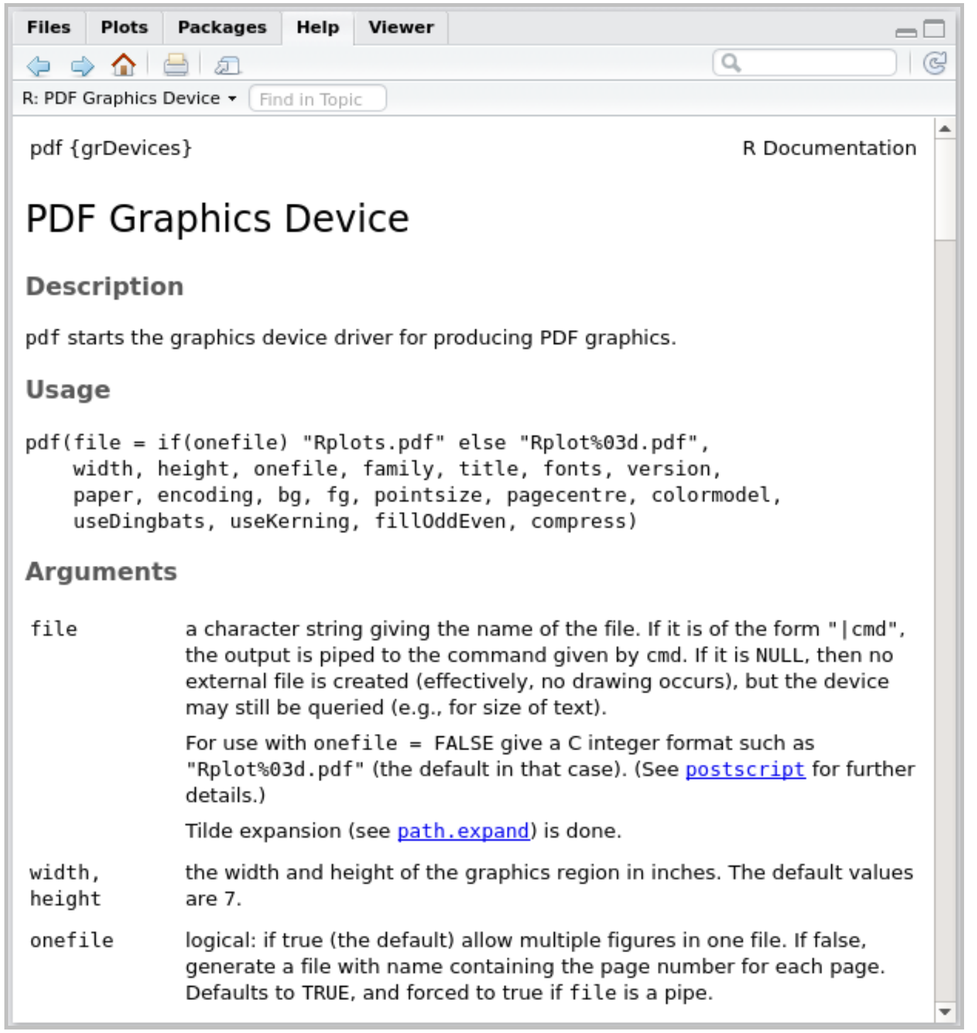
Saving plots: export to PDF

- There are a few advantages of saving plots to a PDF format as opposed to an image:
  - PDF documents allow R's native **vector graphics** to shine: the image quality will not be affected if we zoom in
  - We can **save multiple plots** into a single PDF document with one command

Saving plots: export to PDF (cont'd)

- `pdf()` function follows the same syntax as `png()`

```
pdf("Name_of_file.pdf", #<- name of file
    width = 16,         #<- width in inches
    height = 8)         #<- height in inches

plot 1    #<- plot object or plotting function
plot 2    #<- plot object or plotting function
...       #<- arbitrary number of plots

dev.off() #<- clear graphics device
```



Saving plots: export to PDF (cont'd)

- Let's save more than one plot together in a PDF

```
# Set working directory to where you want to save plots.
setwd(plot_dir)

pdf("health_plots.pdf", #<- name of file
    width = 16,       #<- width in inches
    height = 8)       #<- height ...

boxplots_norm         #<- plot 1
scatter_norm          #<- plot 2

dev.off()             #<- clear graphics device
```

```
RStudioGD
        2
```

Knowledge Check

Exercise

You are now ready to try tasks 8-16 in the Exercise for this topic

Module completion checklist

Objective	Complete
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize the transformed data in a boxplot and a scatterplot with ggplot2	✔

Static Plots: topic summary

In this part of the course, we have covered the following concepts:

- Discuss the process of constructing complex plots
- Visualize data with ggplot2

Congratulations on completing this module!