Intro To Visualization In R - Static Plots - 1

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

Static Plots: topic introduction

In this part of the course, we will cover the following concepts:

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

Chat question

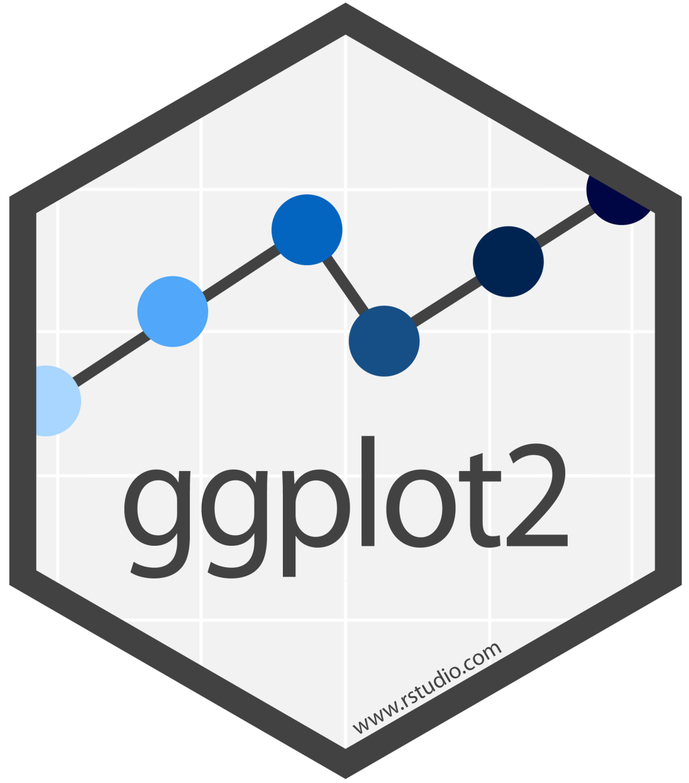
- Think about the data you analyze regularly at work
- Are you most likely to use univariate, bivariate, or multivariate plots to help interpret that data? Why?
- Share your responses in the chat

Module completion checklist

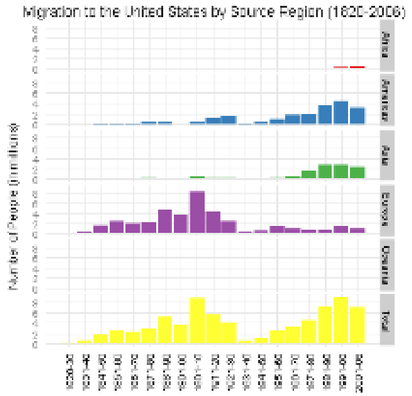
Objective	Complete
Formulate the process of using ggplot2 to build plots	
Build a histogram and a scatterplot with ggplot2	

Visualizing with `ggplot2`

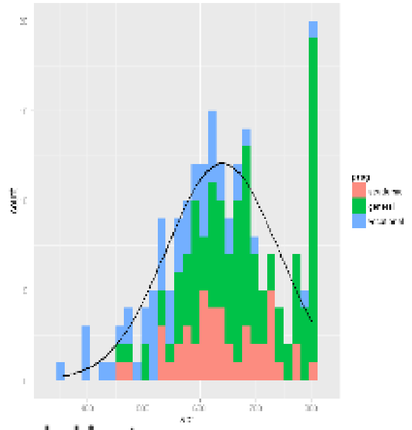
- **`ggplot2`** is a tidyverse package used for creating graphics grammatically (i.e., "grammar of graphics," or gg, plots)
- Building graphics **layer by layer** allows you to:
  - Explore your data efficiently
  - Communicate a visual story in a flexible and efficient way
  - Layer raw, summarized and contextual data
  - Reproduce and extend your work



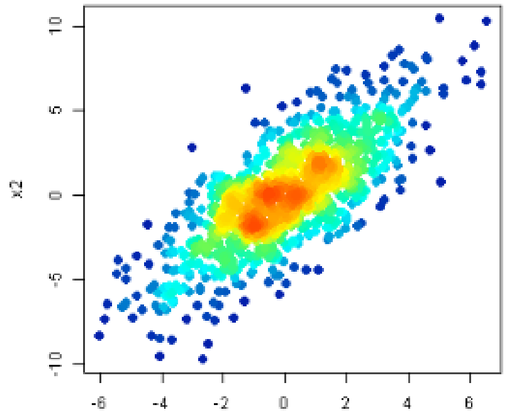
Visualizing with `ggplot2` (cont'd)



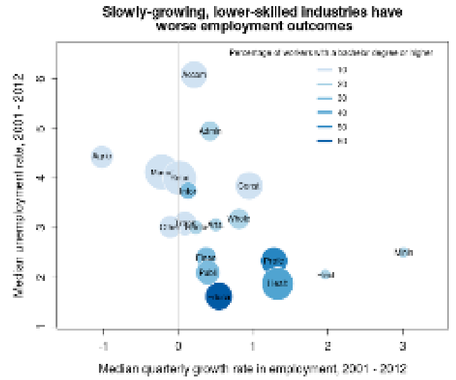
Bar plot



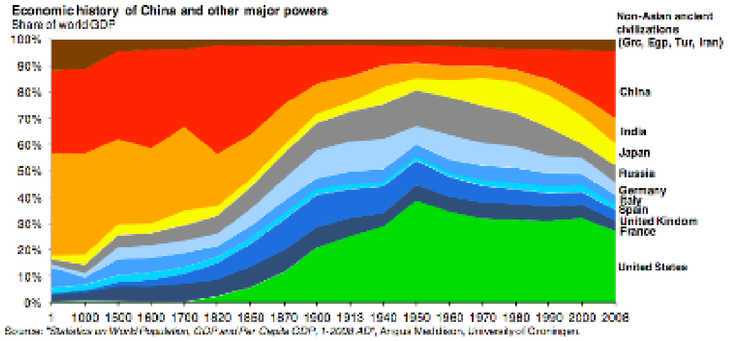
Histogram



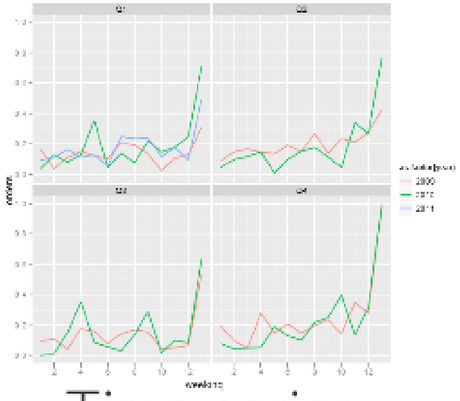
Scatterplot



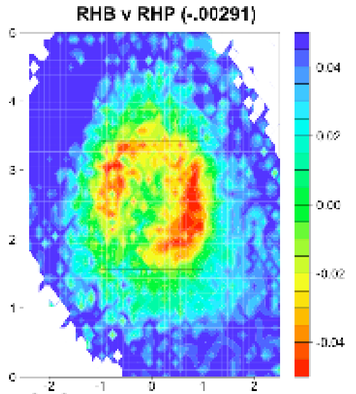
Bubble plot



Area plot



Time series



Heat map

Working with `ggplot2` package

- The `ggplot2` package offers the following features:

Set up

1. Specify data
2. Link data to visuals
3. Assign shapes

Adjust

1. Visual effects
2. Axes
3. Legend

Polish

1. Customize theme
2. Layer statistics
3. Text

Module completion checklist

Objective	Complete
Formulate the process of using ggplot2 to build plots	✔
Build a histogram and a scatterplot with ggplot2	

Directory settings

- Let's practice using `ggplot2` to visualize a sample dataset
- In order to maximize the efficiency of your workflow, we use the `box` package and encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your materials folder

```
# Set `main_dir` to the location of your materials folder.

path = box::file()
main_dir = dirname(dirname(path))
```

Directory settings (cont'd)

- Store all datasets in the `data` directory inside of the materials folder in your environment, so we'll save its path to a `data_dir` variable
- Save all of the plots in the `plots` directory corresponding to `plot_dir` variable
- To append a string to another string, use the `paste0` command and pass the strings you would like to paste together

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Case study: stroke survey

- According to the World Health Organization (WHO), stroke is the 2nd leading cause of death globally
- ***Click here*** for a dataset showing the results of a stroke drug survey clinical trial on a sample of adults in the U.S
- Each row in the data provides relevant information about the adult, including if they had a stroke

`Stroke` Dataset: attribute information

- *id*: unique identifier
- *gender*: “Male”, “Female” or “Other”
- *age*: age of the patient
- *hypertension*: 0 if the patient doesn’t have hypertension, 1 if the patient has hypertension
- *heart_disease*: 0 if the patient doesn’t have any heart diseases, 1 if the patient has a heart disease
- *ever_married*: “No” or “Yes”
- *work_type*: “children”, “Govt_job”, “Never_worked”, “Private” or “Self-employed”
- *Residence_type*: “Rural” or “Urban”
- *avg_glucose_level*: average glucose level in blood
- *bmi*: body mass index
- *smoking_status*: “formerly smoked”, “never smoked”, “smokes” or “Unknown”*
- *stroke*: 1 if the patient had a stroke or 0 if not

Load the dataset

- Let's load the dataset from our `data_dir` into R's environment

```
# Read CSV file called "healthcare-dataset-stroke-data.csv"
health_data = read.csv(file = file.path(data_dir,"healthcare-dataset-stroke-data.csv"), #<- provide
file path
                header = TRUE,             #<- if file has header set to TRUE
                stringsAsFactors = FALSE) #<- read strings as characters, not as factors
```

View data types

- Let's examine the data types of the columns in the dataset and handle the missing data, if there are any

```
str(health_data)
```

```
'data.frame':   5110 obs. of  12 variables:
 $ id                : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
 $ gender            : chr  "Male" "Female" "Male" "Female" ...
 $ age               : num  67 61 80 49 79 81 74 69 59 78 ...
 $ hypertension      : int  0 0 0 0 1 0 1 0 0 0 ...
 $ heart_disease     : int  1 0 1 0 0 0 1 0 0 0 ...
 $ ever_married      : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ work_type         : chr  "Private" "Self-employed" "Private" "Private" ...
 $ Residence_type    : chr  "Urban" "Rural" "Rural" "Urban" ...
 $ avg_glucose_level: num  229 202 106 171 174 ...
 $ bmi               : num  36.6 NA 32.5 34.4 24 29 27.4 22.8 NA 24.2 ...
 $ smoking_status    : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
 $ stroke            : int  1 1 1 1 1 1 1 1 1 1 ...
```

Impute missing data

- We will now impute missing values in the `bmi` column with the mean

```
# Convert BMI to numeric
health_data$bmi <- as.numeric(health_data$bmi)
# Replace N/A's in BMI column with mean
health_data$bmi[is.na(health_data$bmi)] <- mean(health_data$bmi,na.rm=TRUE)
```

```
str(health_data)
```

```
'data.frame':   5110 obs. of  12 variables:
 $ id                : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
 $ gender            : chr  "Male" "Female" "Male" "Female" ...
 $ age               : num  67 61 80 49 79 81 74 69 59 78 ...
 $ hypertension      : int  0 0 0 0 1 0 1 0 0 0 ...
 $ heart_disease     : int  1 0 1 0 0 0 1 0 0 0 ...
 $ ever_married      : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ work_type         : chr  "Private" "Self-employed" "Private" "Private" ...
 $ Residence_type    : chr  "Urban" "Rural" "Rural" "Urban" ...
 $ avg_glucose_level: num  229 202 106 171 174 ...
 $ bmi               : num  36.6 28.9 32.5 34.4 24 ...
 $ smoking_status    : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
 $ stroke            : int  1 1 1 1 1 1 1 1 1 1 ...
```

Subsetting data

- Suppose we only want to work on a few of these variables, not the entire dataset; we must restructure our data by taking a **subset** of the data with all observations of the following variables:
  - `age` variable
  - `avg_glucose_level` variable
  - `bmi` variable
- Since we know which columns we want to subset in the dataset, we can use their column names to extract them

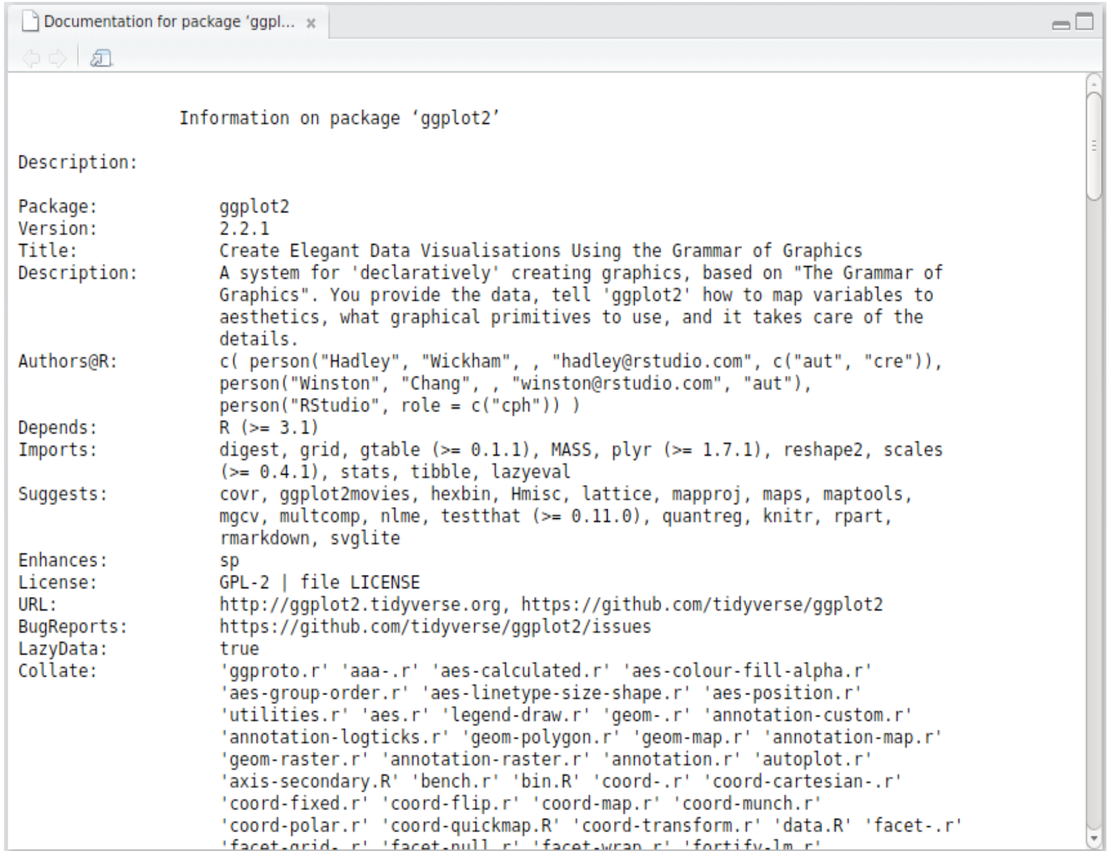
```
health_subset <- health_data[, c("age", "avg_glucose_level", "bmi")]
str(health_subset)
```

```
'data.frame':   5110 obs. of  3 variables:
 $ age               : num  67 61 80 49 79 81 74 69 59 78 ...
 $ avg_glucose_level: num  229 202 106 171 174 ...
 $ bmi               : num  36.6 28.9 32.5 34.4 24 ...
```

Installing `ggplot2`

- Since `ggplot2` is an external package, we need to install it first

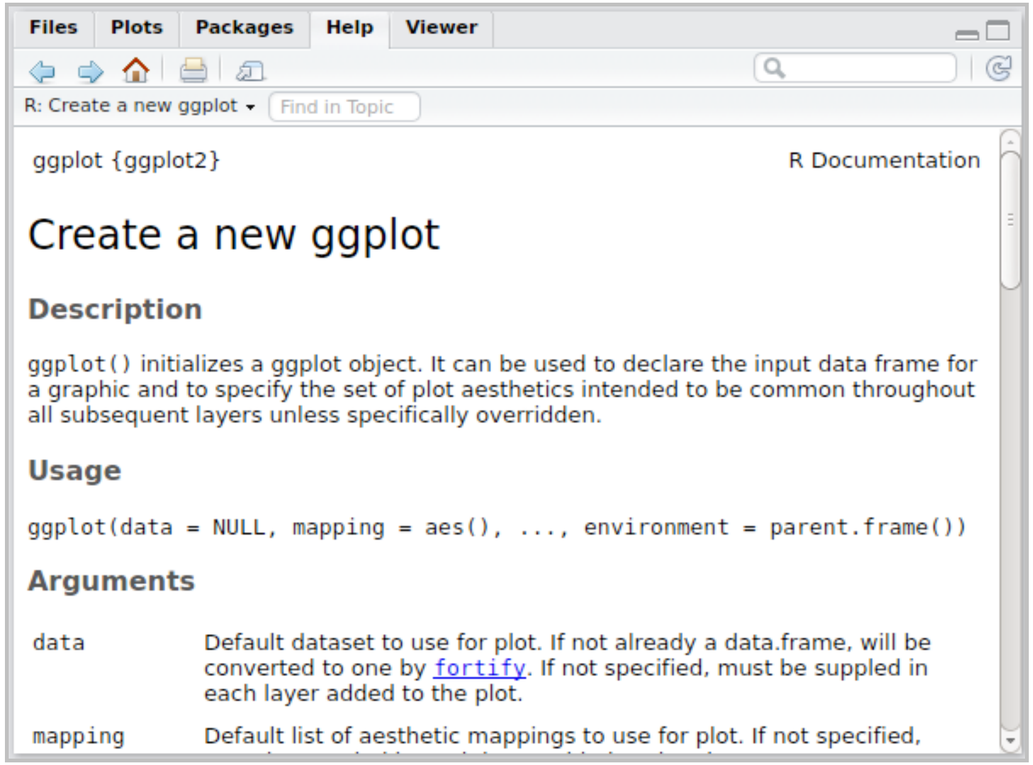
```
# First let's install `ggplot2`. if not installed
#install.packages("ggplot2")

# Then we need to load it to our environment.
library(ggplot2)

# Take a look at the documentation.
library(help = "ggplot2")
```



Working with `ggplot2` package

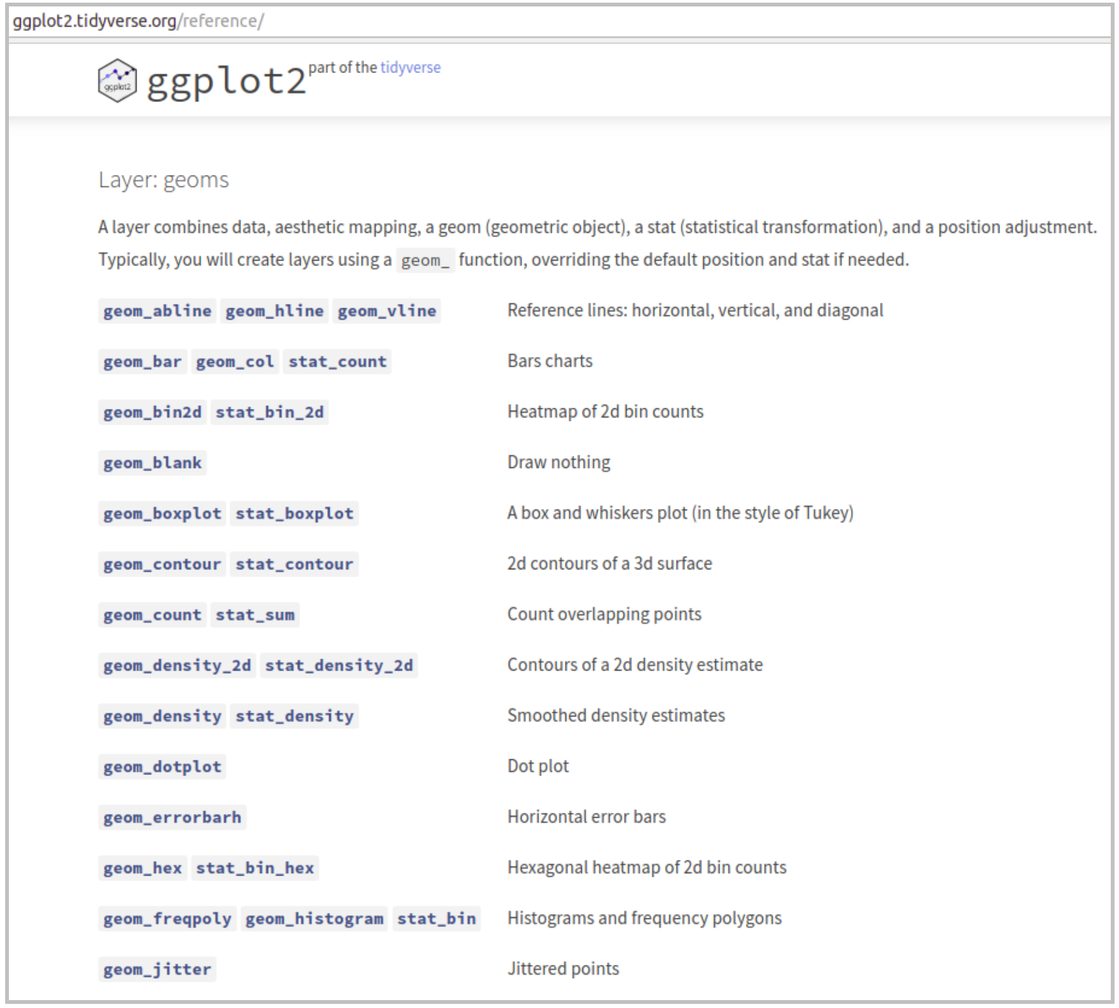
```
?ggplot
```

- The two main arguments in the function are:
  - `data`: it tells `ggplot` which data is to be plotted
  - `mapping`: a list of `aes[thetics]` that tells `ggplot` which variables are to be mapped to which axes



Working with `ggplot2` package (cont'd)

- After the initial plot object has been created as a base, other layers are added to it
- Each chart type corresponds to a `geom` (i.e., a layer)
- ***Click here to view the entire list of `geoms` at the tidyverse website***


ggplot2.tidyverse.org/reference/

ggplot2 part of the tidyverse

Layer: geoms

A layer combines data, aesthetic mapping, a geom (geometric object), a stat (statistical transformation), and a position adjustment. Typically, you will create layers using a `geom_` function, overriding the default position and stat if needed.

`geom_abline` `geom_hline` `geom_vline`	Reference lines: horizontal, vertical, and diagonal
`geom_bar` `geom_col` `stat_count`	Bars charts
`geom_bin2d` `stat_bin_2d`	Heatmap of 2d bin counts
`geom_blank`	Draw nothing
`geom_boxplot` `stat_boxplot`	A box and whiskers plot (in the style of Tukey)
`geom_contour` `stat_contour`	2d contours of a 3d surface
`geom_count` `stat_sum`	Count overlapping points
`geom_density_2d` `stat_density_2d`	Contours of a 2d density estimate
`geom_density` `stat_density`	Smoothed density estimates
`geom_dotplot`	Dot plot
`geom_errorbarh`	Horizontal error bars
`geom_hex` `stat_bin_hex`	Hexagonal heatmap of 2d bin counts
`geom_freqpoly` `geom_histogram` `stat_bin`	Histograms and frequency polygons
`geom_jitter`	Jittered points

Plotting with `ggplot2`: setup

- Let's start by creating a **histogram** to show the distribution of the `age`

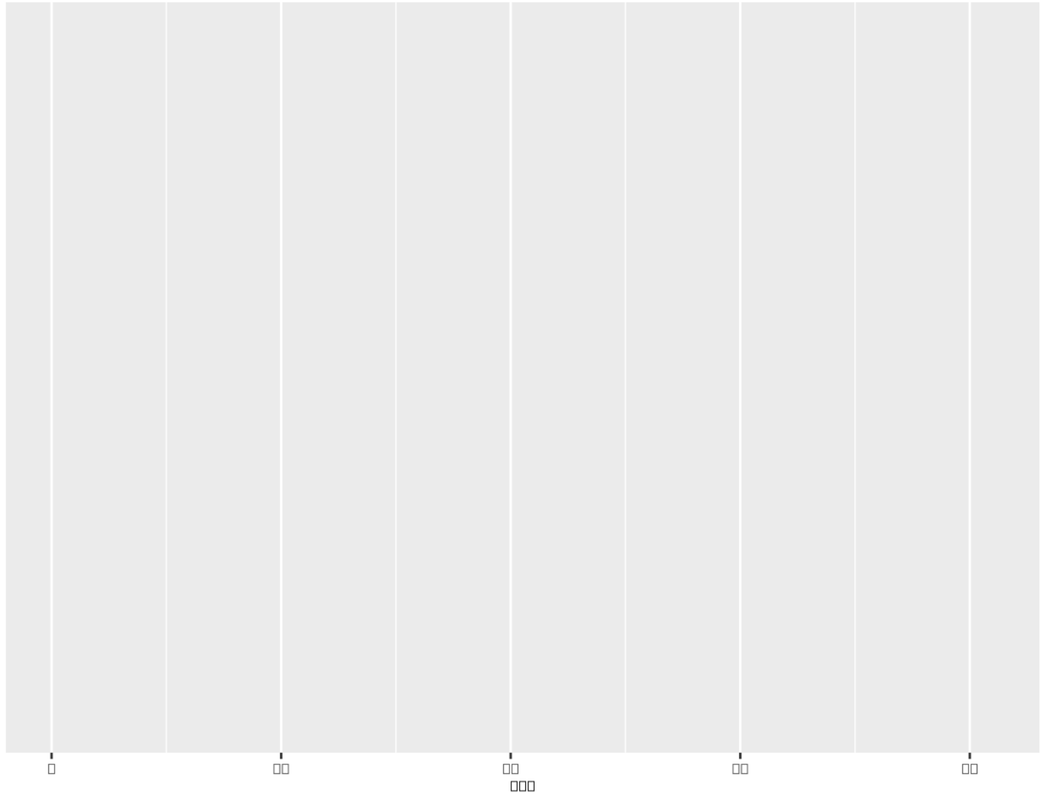
- In order to **make a base plot** we need to:
  - Specify data
  - Link data to visuals
  - Assign shapes

Plotting with `ggplot2`: setup

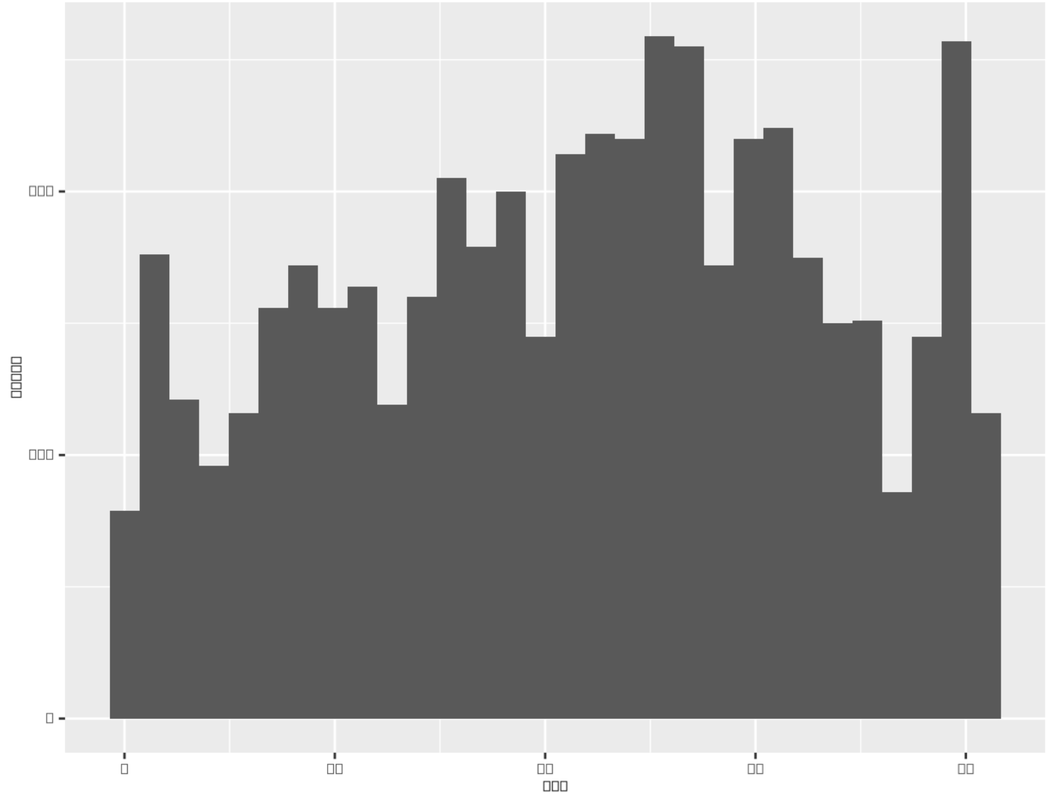
- Translated into `ggplot2` syntax, this will require us to call a `ggplot` function and:
  - Set its first argument to the data we want to use (e.g. `health_subset`)
  - Set its second argument to the mapping of our choice (e.g. `Age` to be plotted on `x-axis`)
  - Add a `geom` layer, in our case it will be a histogram (e.g. add `geom_histogram` to the base plot)

Making a plot with `geom_histogram`

```
# Let's create a base plot.
ggp1 = ggplot(health_subset,    #<-Set data
              aes(x = age))#<-Set aesthetics
ggp1
```



```
# Let's add a layer.
# Each layer is called a `geom`, we need to use
# `geom_histogram` to add a histogram layer.
ggp1 + geom_histogram()
```

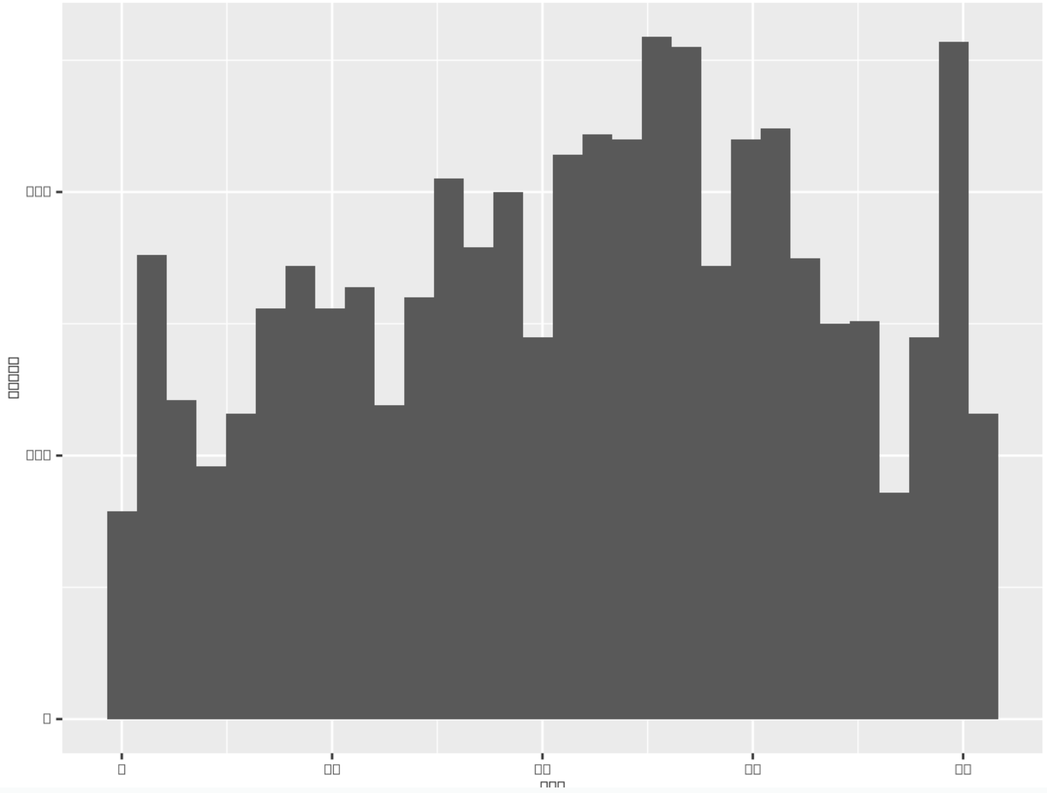


Building a histogram using density with `ggplot2`

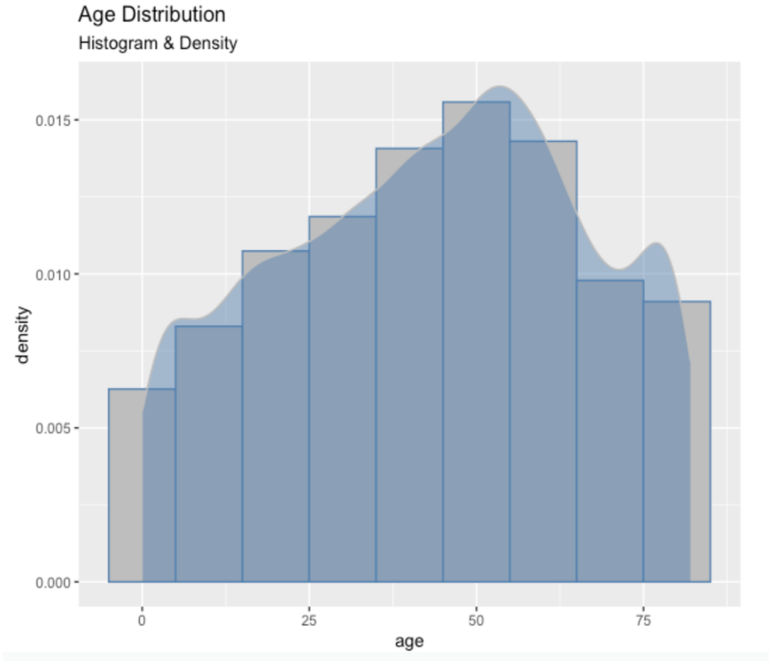
- What we have

```
ggp1 + geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better
value with `binwidth`.
```



- What we need like



Plotting with `ggplot2`: adjust

- To make the visualization **prominent**,we need to adjust:
  - Visual effects
  - Axes
  - Legend

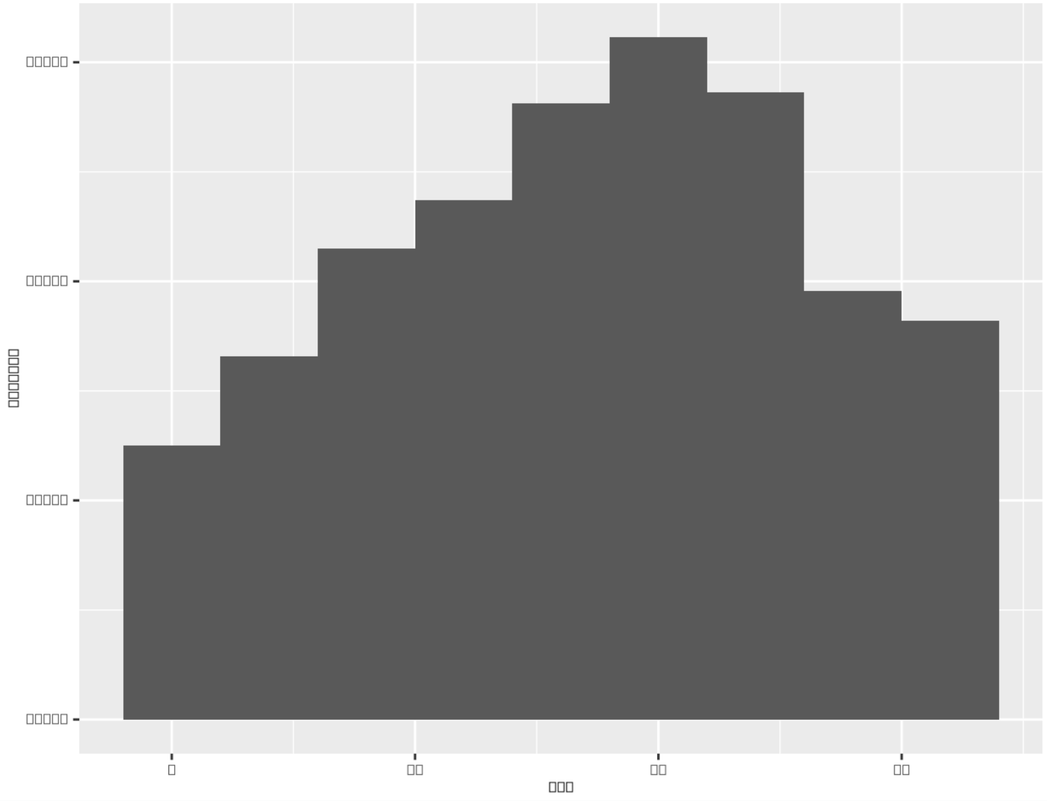
Plotting with `ggplot2`: adjust (cont'd)

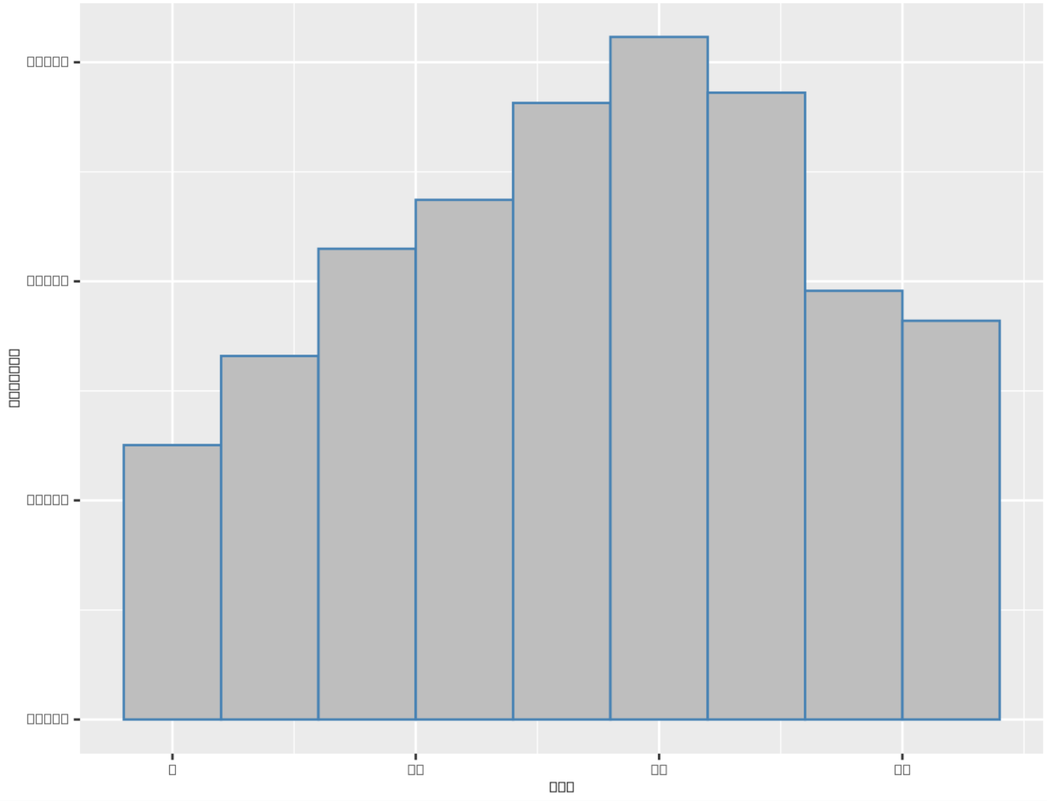
- Depending on the plot and the final goal, you may want to:
  - Change colors, add more layers with extra information such as additional data sources, reference lines or points, etc.
  - Add axes labels and adjust the breaks in data (i.e. change scale, bins in the histogram)
  - Add a legend if there are more variables or datasets than one combined on the plot

Making a plot with `geom_histogram`: adjust

```
# Say instead of frequency counts we want
# probability density estimates. We can:
ggp1 +
  # 1. Make `y-axis` of type `density`
  geom_histogram(aes(y = ..density..),
  # 2. Adjust binwidth for better smoothness.
                 binwidth = 10)
```

```
ggp1 = ggp1 +     #<- save adjusted plot
  geom_histogram(aes(y = ..density..),
                 binwidth = 10,
                 # 3. Add color & fill.
                 color = "steelblue",
                 fill = "gray")
ggp1               #<- view saved plot
```



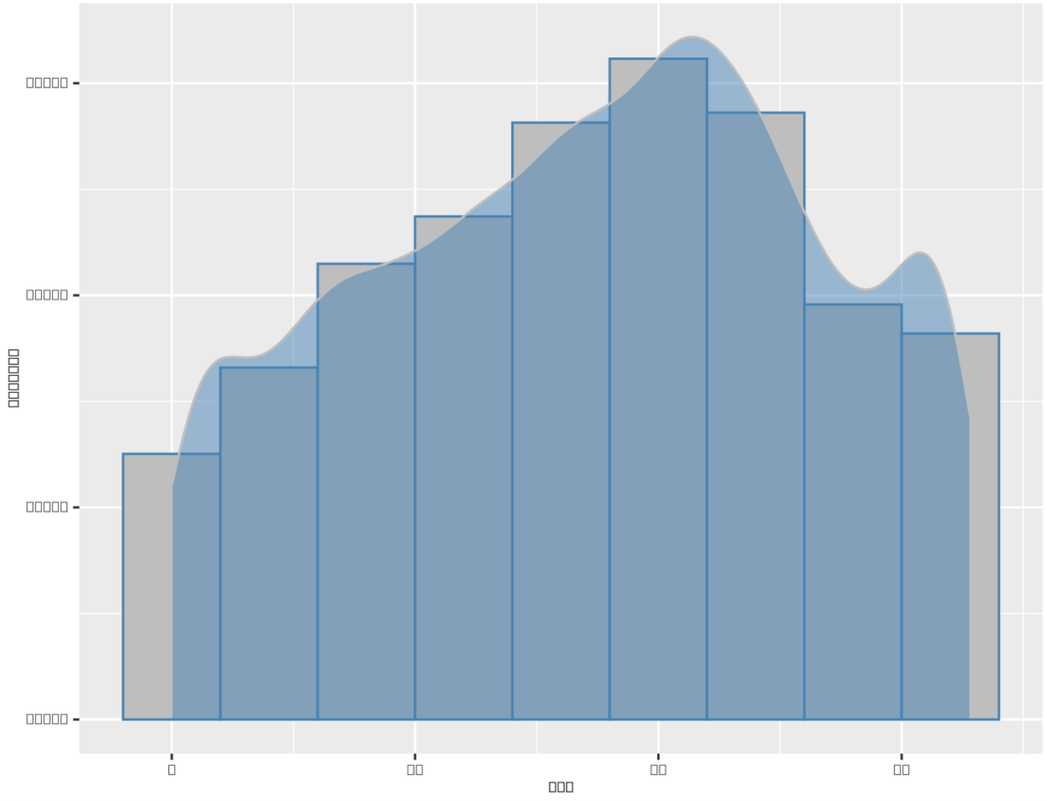


Plotting with `ggplot2`: polish

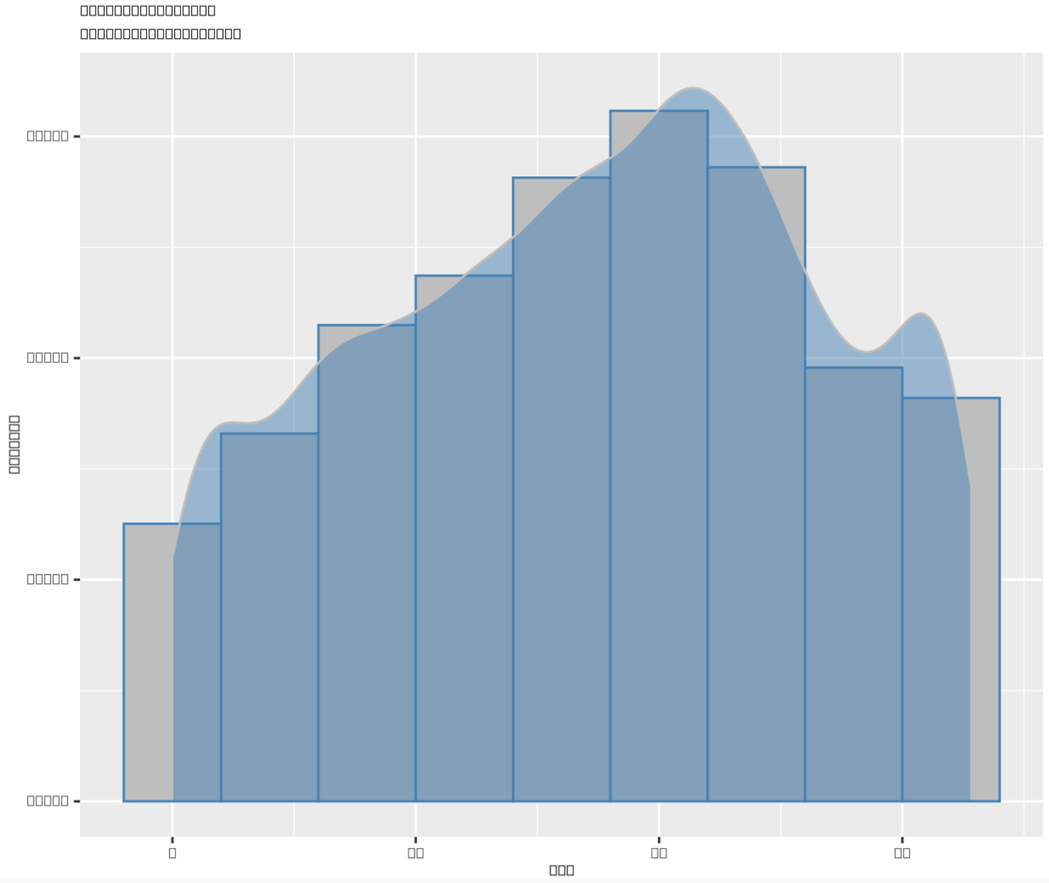
- In order to make visualization complete, we need to polish by adding more elements:
  - Layer statistics
  - Custom theme
  - Text
- Depending on the plot and the final goal, you may want to:
  - Add statistical layers such as density, mean markers, loess curves, etc.
  - Customize plot theme
  - Add/change any text labels or add free-form text to the plot

Plotting with `geom_histogram`: polish

```
ggp1 = ggp1 +
  # Add density layer with 50% opaque fill
  # in "steelblue" and gray color (border).
  geom_density(alpha = .5,
               color = "gray",
               fill = "steelblue")
ggp1
```



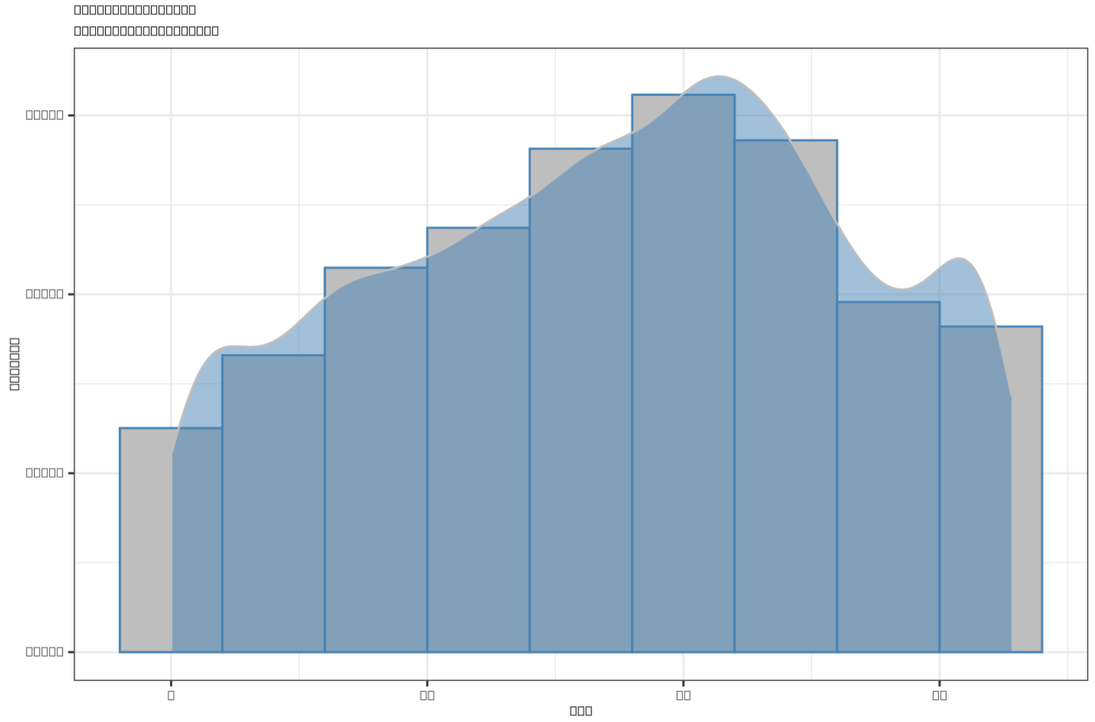
```
ggp1  = ggp1 +
# Add plot title and subtitle.
labs(title = "Age Distribution",
     subtitle = "Histogram & Density")
ggp1
```



Plotting with `geom_histogram`: polish (cont'd)

```
# Add a black and white theme to
# overwrite default.
ggp1 = ggp1 +

  # Add a black and white theme.
  theme_bw() +

  # Customize elements of the theme.
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

```
# Display polished plot.
ggp1
```

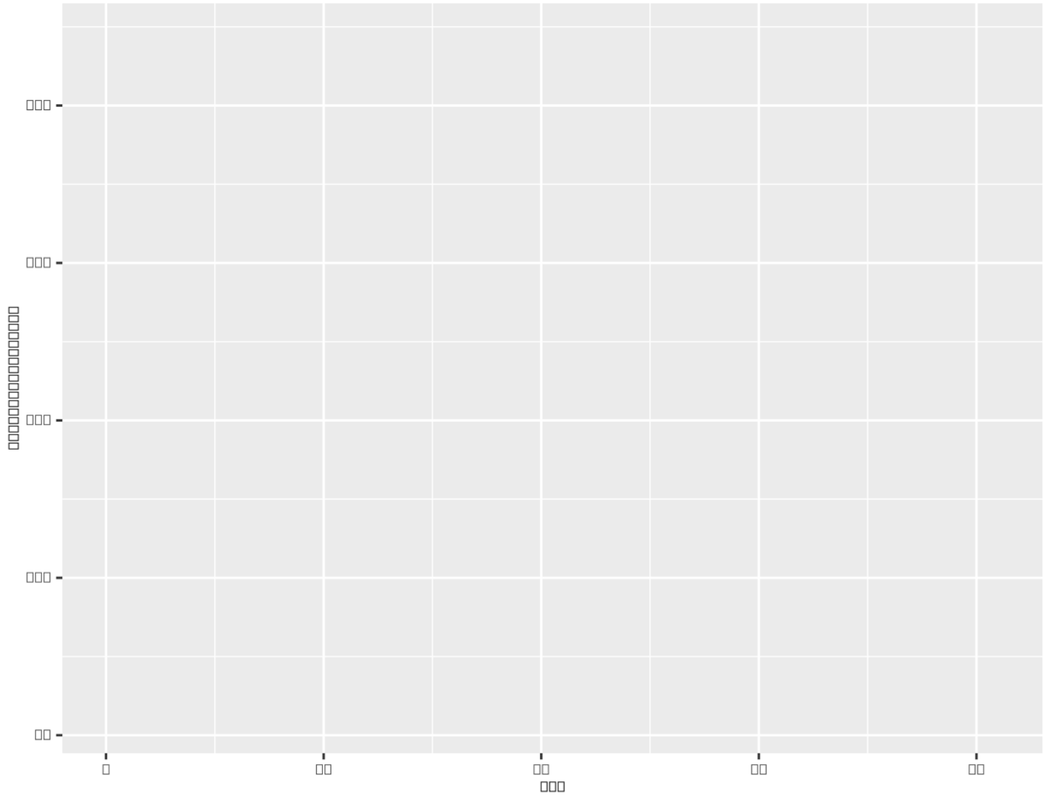


Scatterplot with `geom_point`: set up

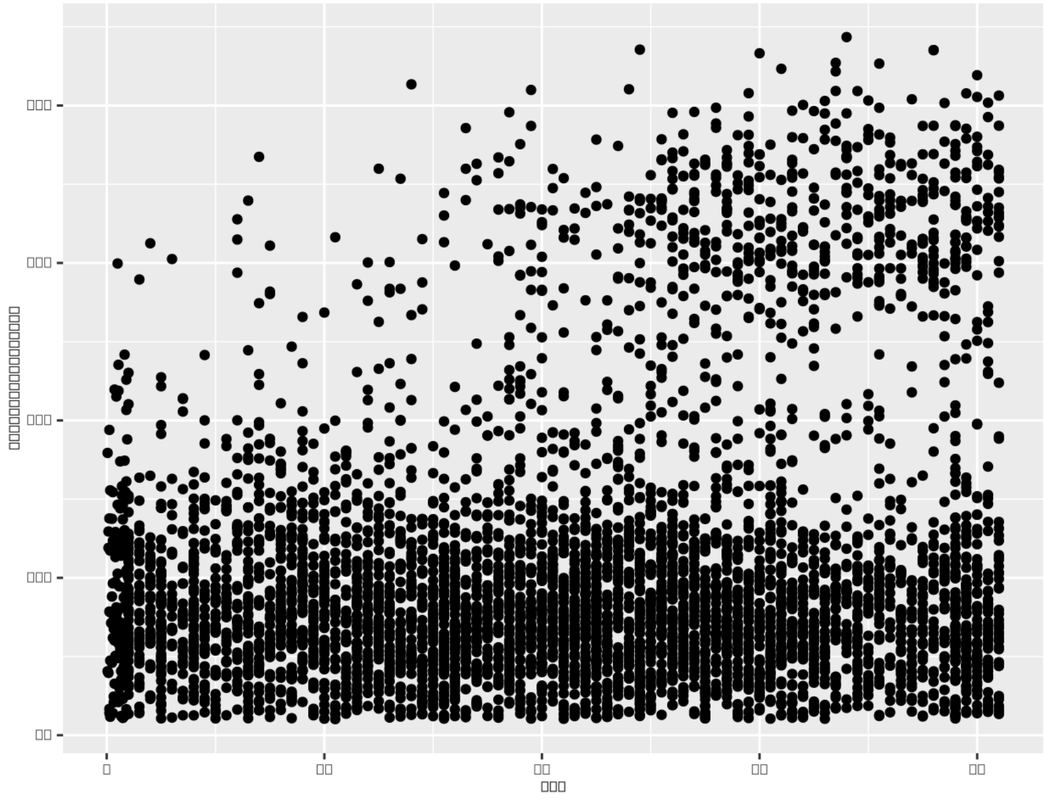
- The histogram was a **univariate** visualization, so we only had to define a single axis
- But for **bivariate** visualizations, like **scatterplots**, we need to define 2 axes:
  - `x-axis`
  - `y-axis`
- In `ggplot` terms, this means that we need to map 2 `aes` parameters (`x` and `y` respectively)
- Let's plot the relationship between `age` on `x-axis` and `avg_glucose_level` on `y-axis`

Scatterplot with `geom_point`: set up

```
# Let's create a base plot.
ggp2 = ggplot(health_subset,
              aes(x = age,
                  y = avg_glucose_level))
ggp2
```



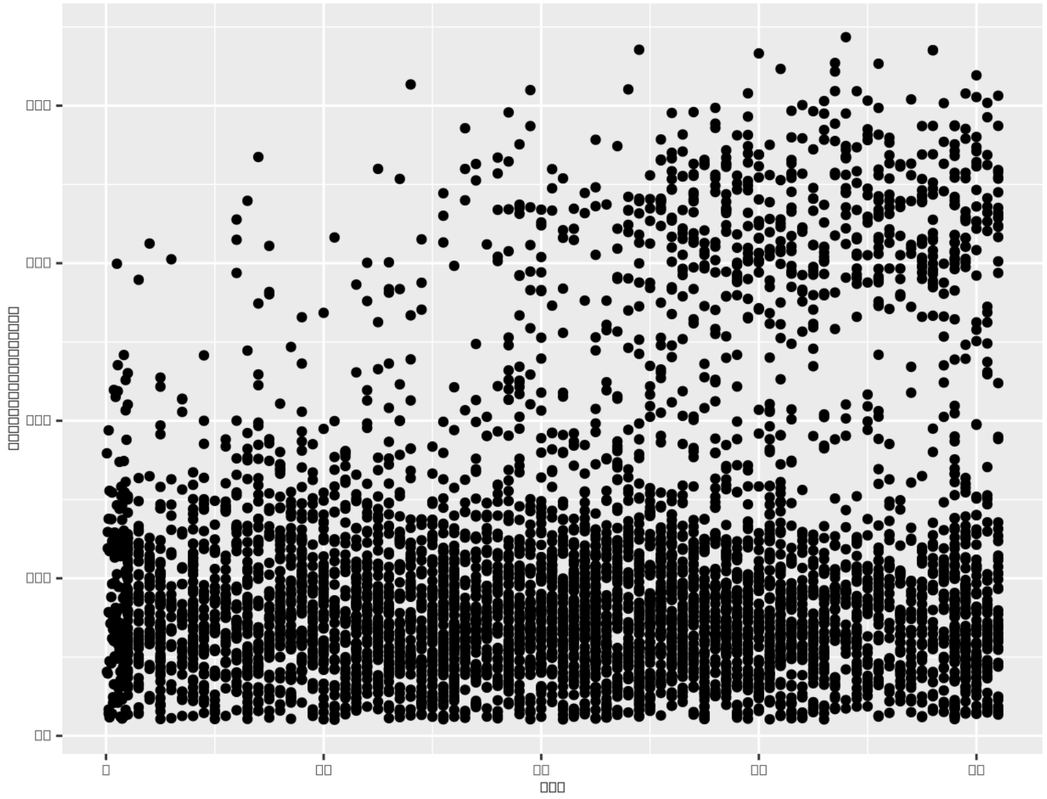
```
# Now, let's add a layer.
# Each layer is called a `geom`, we need to use
# `geom_point` to add a histogram layer.
ggp2 = ggp2 + geom_point()
ggp2
```



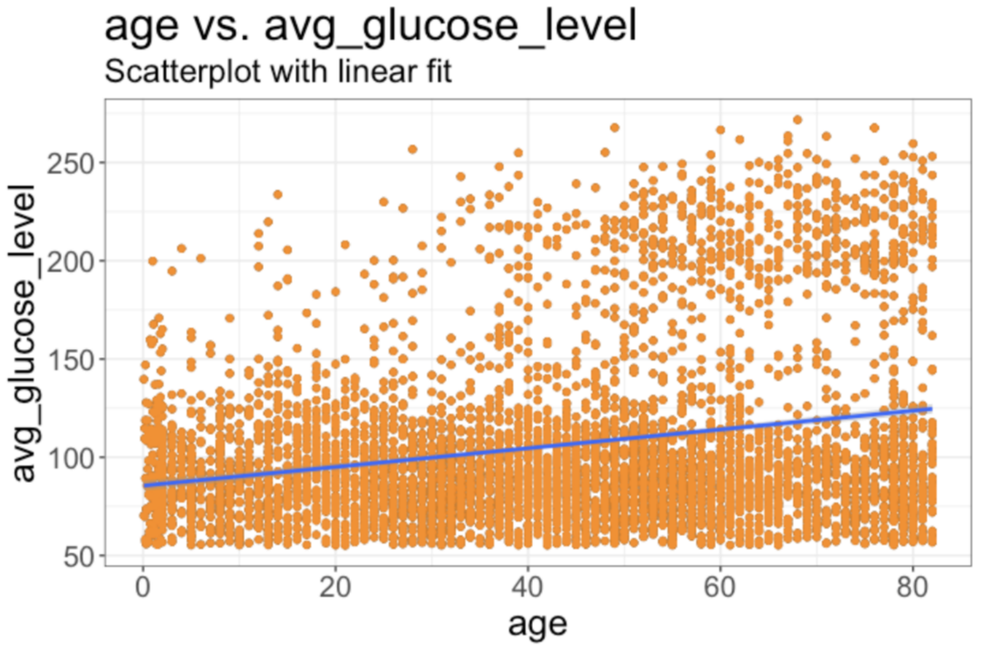
Building a scatterplot with fitted line

- What we have

```
ggp2
```



- What we need



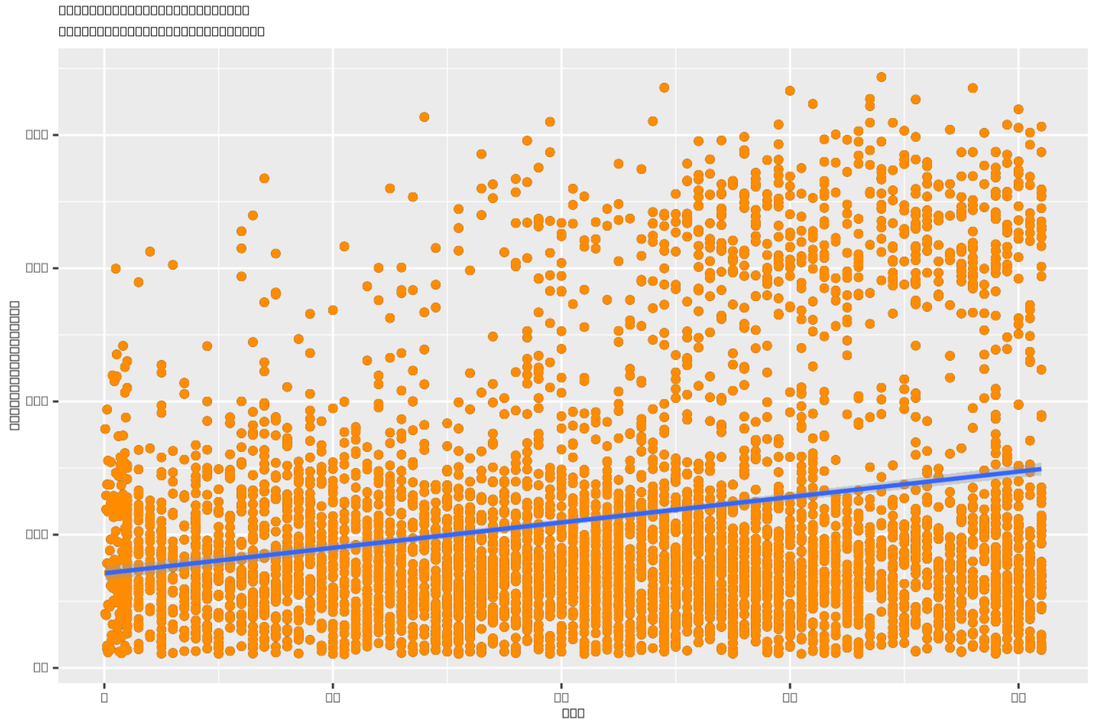
Adjusting a scatterplot with `geom_point`

- Let's adjust the scatterplot with `geom_point`

```
ggp2 = ggp2 +

  # Adjust the color of the points.
  geom_point(color = "darkorange") +

  # Add linear regression line (`lm`).
  geom_smooth(method = lm) +

  # Add a title and a subtitle.
  labs(title = "age vs. avg_glucose_level",
       subtitle = "Scatterplot with linear fit")
```

```
# View the plot.
ggp2
```

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

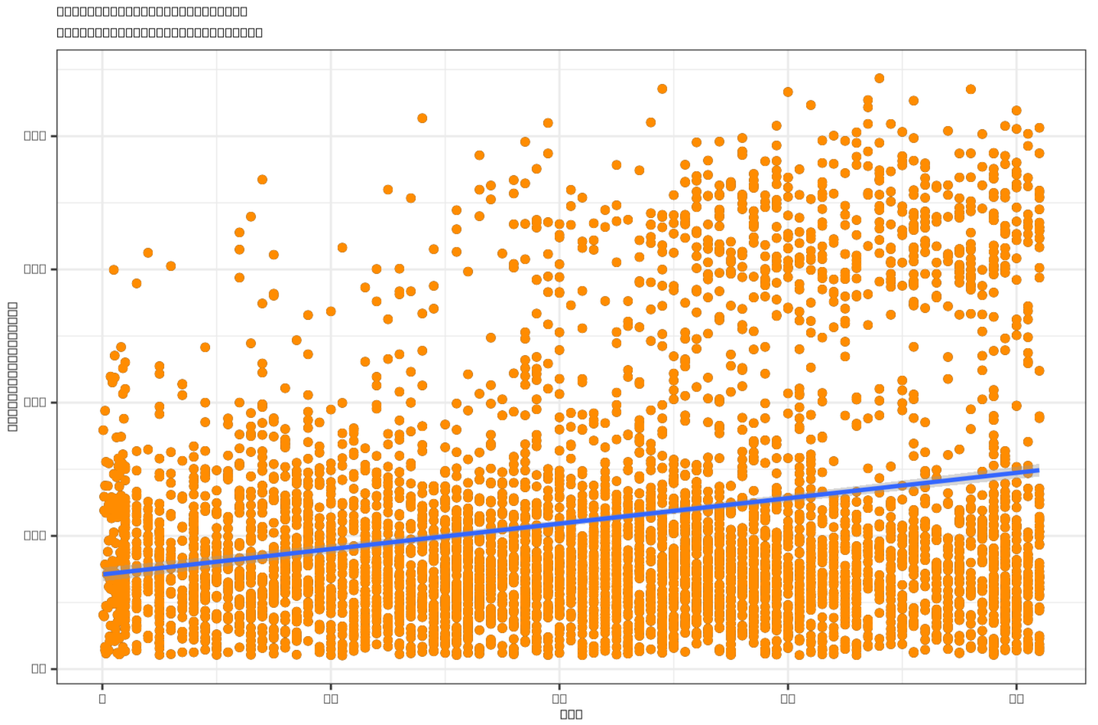


Polishing a scatterplot with `geom_point`

- Now, let's polish it with `geom_point`

```
ggp2 +

  # Add black & white theme, adjust labels.
  theme_bw() +

  # Customize elements of the theme.
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

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



Saving a theme to a variable

- We've used the following 3 lines of code for both the charts:

```
theme_bw() +
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

```
List of 93
 $ line                          :List of 6
  ..$ colour        : chr "black"
  ..$ size          : num 0.5
  ..$ linetype      : num 1
  ..$ lineend       : chr "butt"
  ..$ arrow         : logi FALSE
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_line" "element"
 $ rect                          :List of 5
  ..$ fill          : chr "white"
  ..$ colour        : chr "black"
  ..$ size          : num 0.5
  ..$ linetype      : num 1
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_rect" "element"
 $ text                          :List of 11
  ..$ family        : chr ""
  ..$ face          : chr "plain"
```

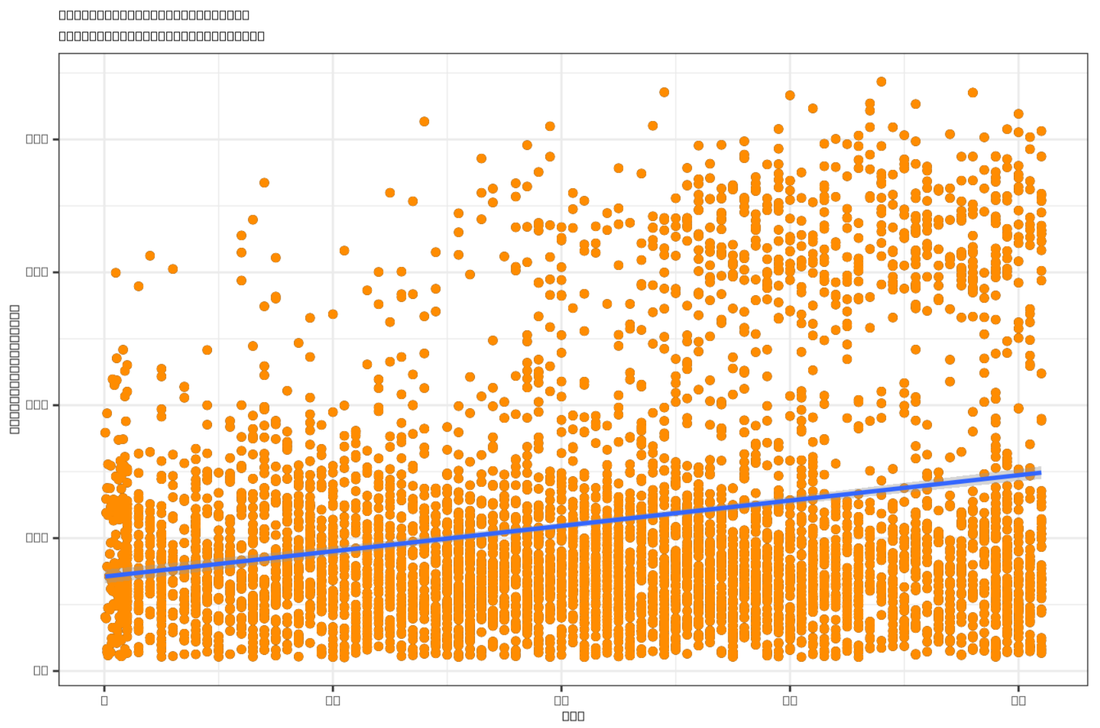
Saving a theme to a variable (cont'd)

- It would been easier to save these theme adjustments to a variable and add a variable to our `ggplot`

```
my_ggtheme = theme_bw() +
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

```
# Add saved theme and re-save the plot.
ggp2 = ggp2 + my_ggtheme
ggp2
```

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



Knowledge Check

Module completion checklist

Objective	Complete
Formulate the process of using ggplot2 to build plots	✔
Build a histogram and a scatterplot with ggplot2	✔

Congratulations on completing this module!

You are now ready to try tasks 1-7 in the Exercise for this topic