Intro To Visualization In R - Basic Data Visualization - 1

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

Basic Data Visualization: topic introduction

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

- Discuss basics of Univariate, Bivariate & Multivariate plots and their use cases
- Build plots with base r

Chat question

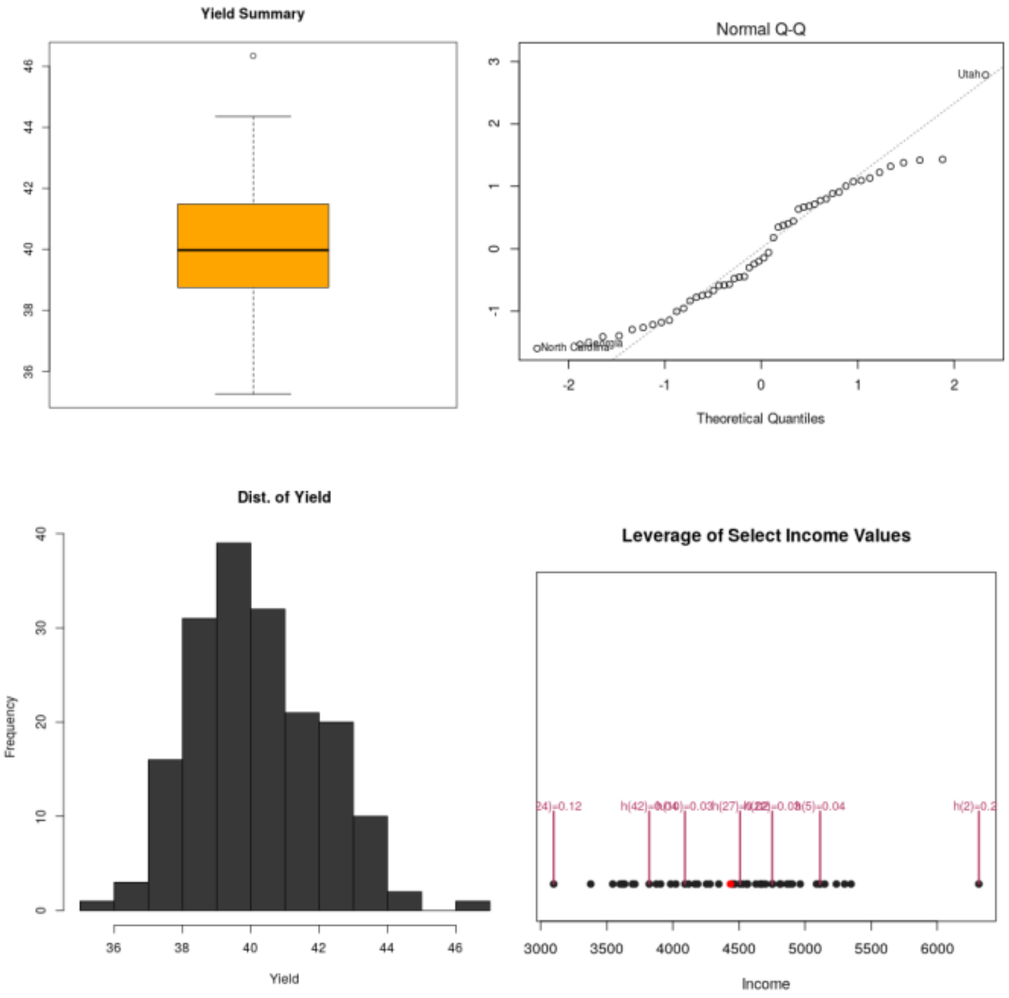
- **Exploratory Data Analysis** refers to the analysis process which is used to discover patterns, spot anomalies, test hypotheses, and check assumptions
- During EDA, you will likely generate visual representations of data to see trends, outliers, and patterns
- In your own work, how do you approach exploratory data analysis?
- Share your responses in the chat

Module completion checklist

Objective	Complete
Describe and build univariate plots to illustrate patterns in the data	
Discuss and create bivariate and multivariate plots to illustrate patterns in data	

Univariate plots

- **Univariate plots** are visualizations that represent a **single variable**
  - These include `boxplot`, `histogram`, `line plot`, `density curve`, `dot plot`, `QQ plot`, `bar plot`
- It can be helpful for profiling a dataset in the **initial stages of EDA** to learn more about individual variables
- **Combining** univariate plots to **compare** the distributions of different variables can also be done



Directory settings

- 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 the materials folder in your environment
- 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 `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 ...
```

Univariate plots: line plot

- **Line plots** are the graphs that can be used to identify **trends** in the data
- The syntax of the line plot includes:

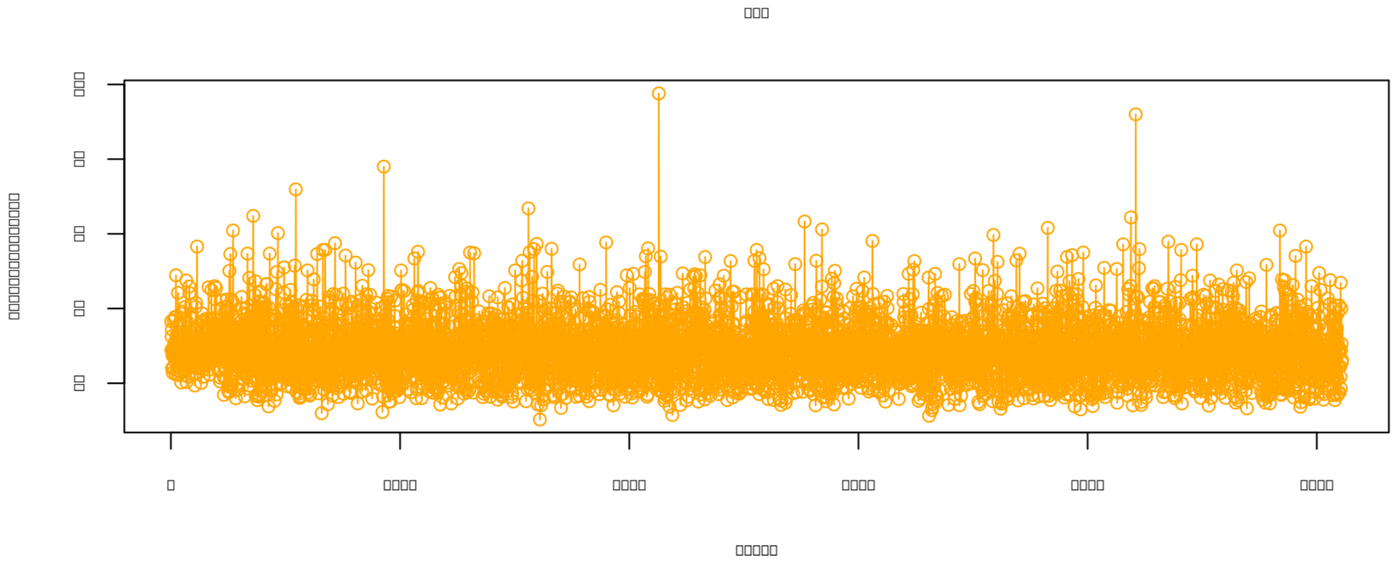
```
# Line plot.
plot(col_vector,type,col,xlab,ylab)
```

- `col_vector` is the vector of the column with the numeric values
- `type` indicates the type of plot and is set to either:
  - `p` to create a plot with points
  - `l` to create a plot with a single line
  - `o` to create a line chart with the combination of both
- `col` denotes the color of the lines and points
- `xlab` and `ylab` denote the label for x-axis and y-axis respectively

Univariate plots: line plot (cont'd)

- Let's create a simple line plot for the `bmi` column to observe the trend in it

```
# Line plot.
plot(health_data$bmi,
     type = "o",
     col = "orange",
     main = "bmi")
```



Univariate plots: boxplot

- Suppose we want to visualize the **distribution** of the `bmi`
- To do so, we will need to create a **boxplot**, which represents the minimum, maximum, median, and 1st and 3rd quartile values as a box and whiskers
- The `summary` of the `bmi` reveals the parameters we will add to a plot

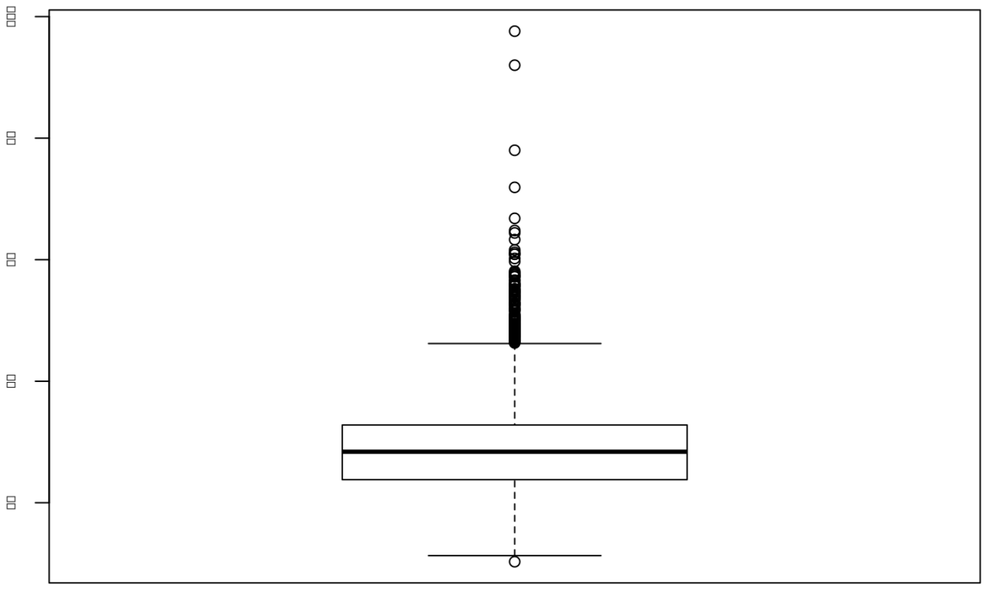
```
# bmi summary.
summary(health_data$bmi)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.
Max.
  10.30   23.80   28.40   28.89   32.80
97.60
```

Univariate plots: boxplot

- Creating a boxplot of the summary values requires just one line of code

```
# Univariate plot: box-and-whisker plot.
boxplot(health_data$bmi)
```

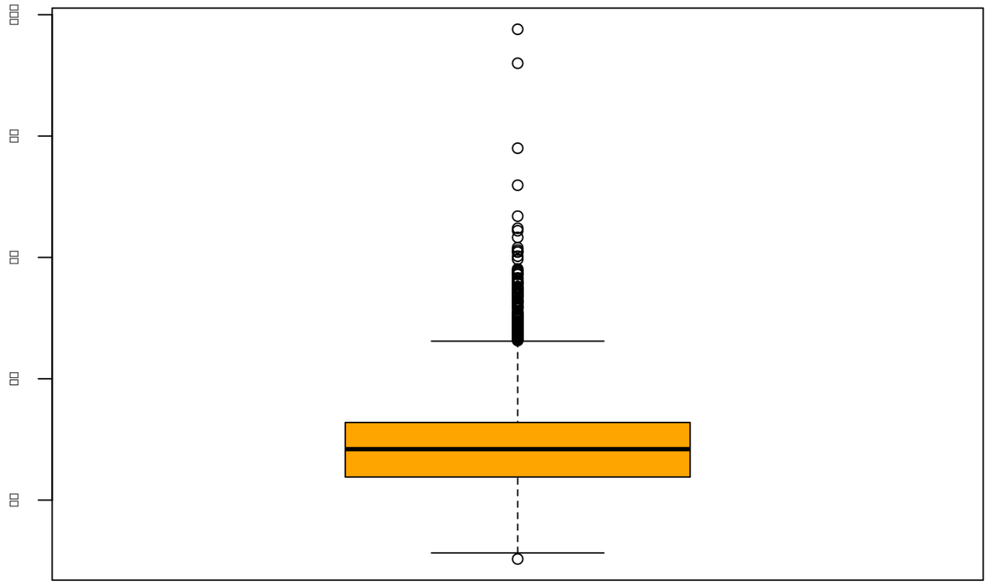


- Min value (`10.30`): bottom `whisker` of the boxplot
- Max value (`97.60`): circle (it's an outlier!)
- 1st quartile value (`23.80`): bottom of the box
- Median (2nd quartile) value (`28.40`): line in the box
- 3rd quartile value (`32.80`): top of the box

Univariate plots: boxplot (cont'd)

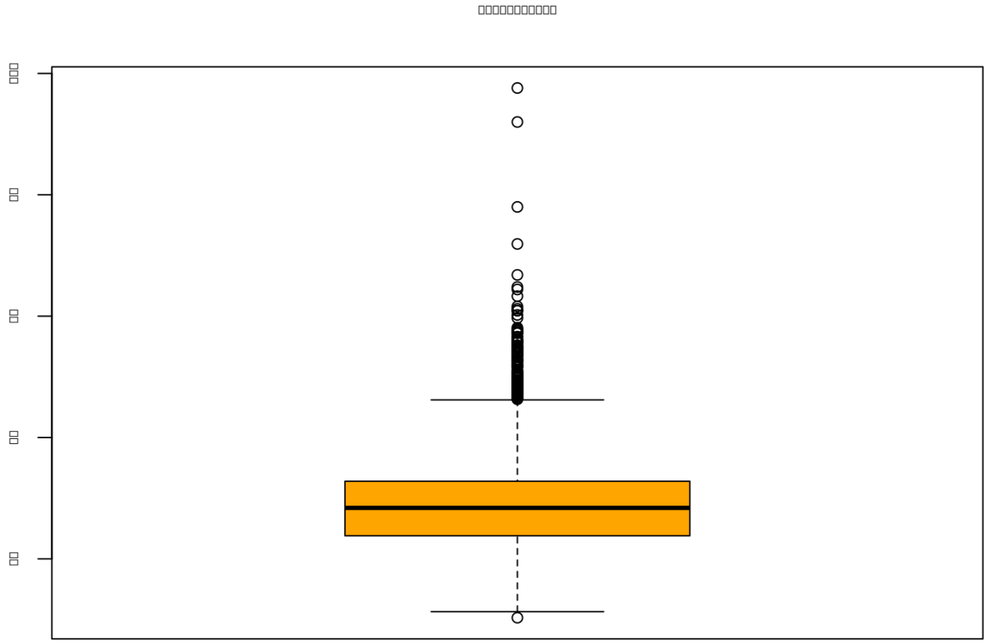
- We can specify parameters for the color

```
# Customize your boxplot.
boxplot(health_data$bmi,
        col = "orange") #<- set color
```



- We can also add a title

```
# Customize your boxplot
boxplot(health_data$bmi,
        col = "orange",
        main = "BMI Boxplot") #<- add title
```



R base colors

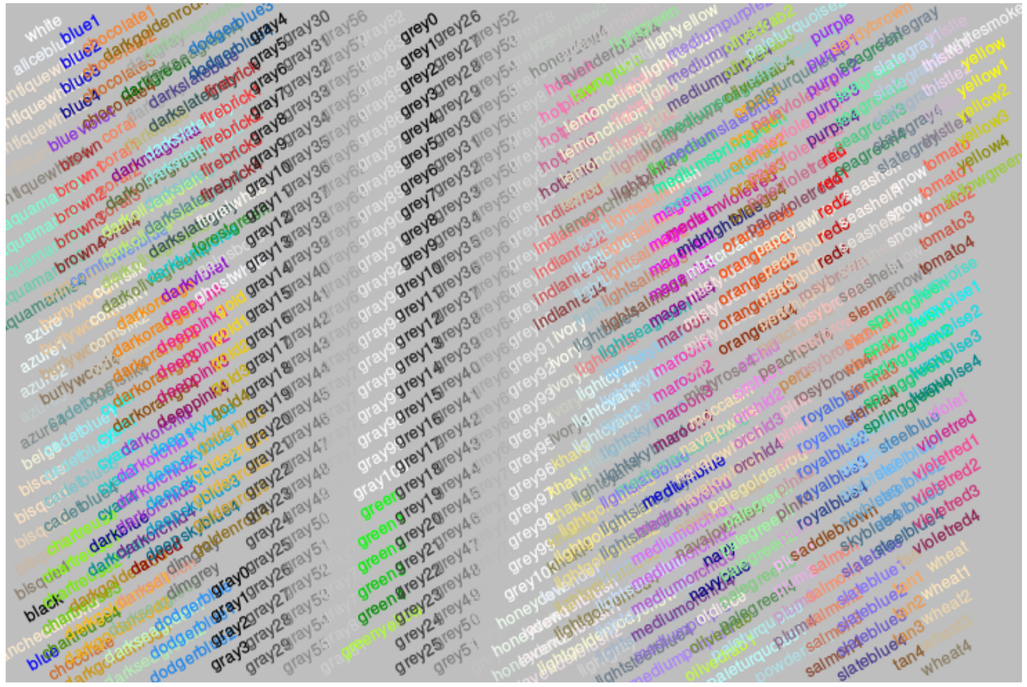
- We can also customize using the wide variety of colors available in R

```
# R has many colors in its native
# `grDevices` package.
?colors
```

```
# There are a total of 657 colors in
# the `colors()` vector.
str(colors())
```

```
 chr [1:657] "white" "aliceblue"
"antiquewhite" "antiquewhite1" ...
```

```
demo("colors")
```

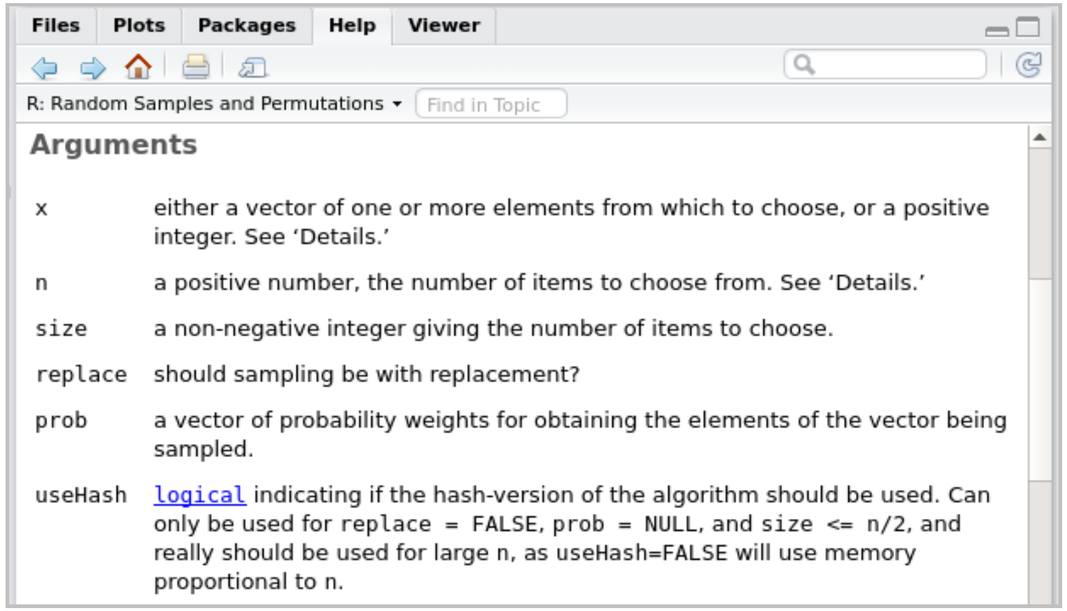


Create variable colors

- However, if you do not want to select specific color values, R can generate a random `sample` as a color palette

```
?sample
```

- `x` is a vector from which we can sample elements
- `n` is the number to indicate how many elements to sample



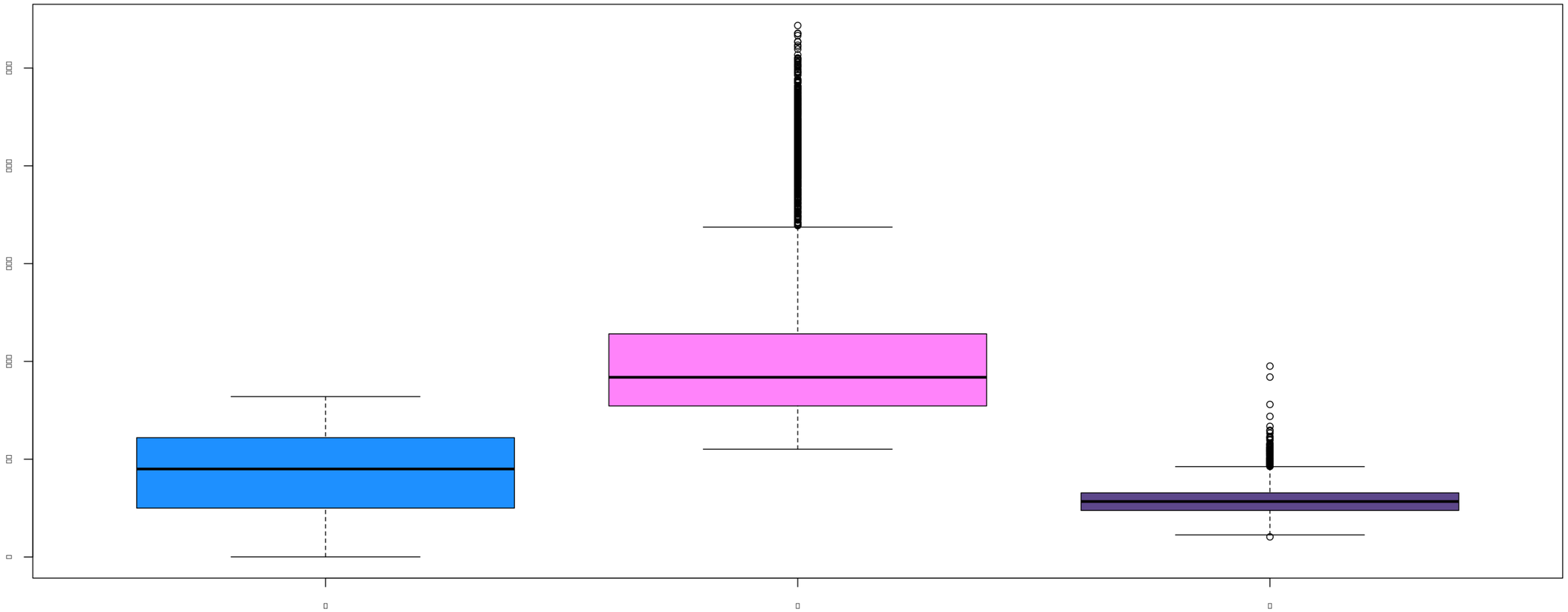
Create variable colors (cont'd)

```
# Set random seed to get the same sample every time!
set.seed(1)

# We have as many variables as columns in our data.
# Save number of columns to a variable using `ncol` function.
n_cols = ncol(health_data)

# Pick a sample of colors for each
# variable in our data set
col_sample = sample(colors(),#<- vector of colors
                    n_cols)  #<- n elements to sample
col_sample
```

```
 [1] "dodgerblue1"   "orchid1"       "mediumpurple4" "grey38"
 [5] "grey9"         "gray34"        "grey46"        "slateblue3"
 [9] "grey16"        "olivedrab1"    "grey69"        "skyblue2"
```

Compare variables: boxplots combined

```
# Make a box plot of relevant variables and give a vector of colors for each of them.
boxplot(health_data$age, health_data$avg_glucose_level, health_data$bmi, col = col_sample)
```



Univariate plots: histogram

- A **histogram** is another way of visualizing the distribution of the data
- While a boxplot shows the total range as quartiles, a histogram represents the **count of observations** within specific **bins** or segments of the range
- We can return specific values by telling R to create a histogram without an associated plot

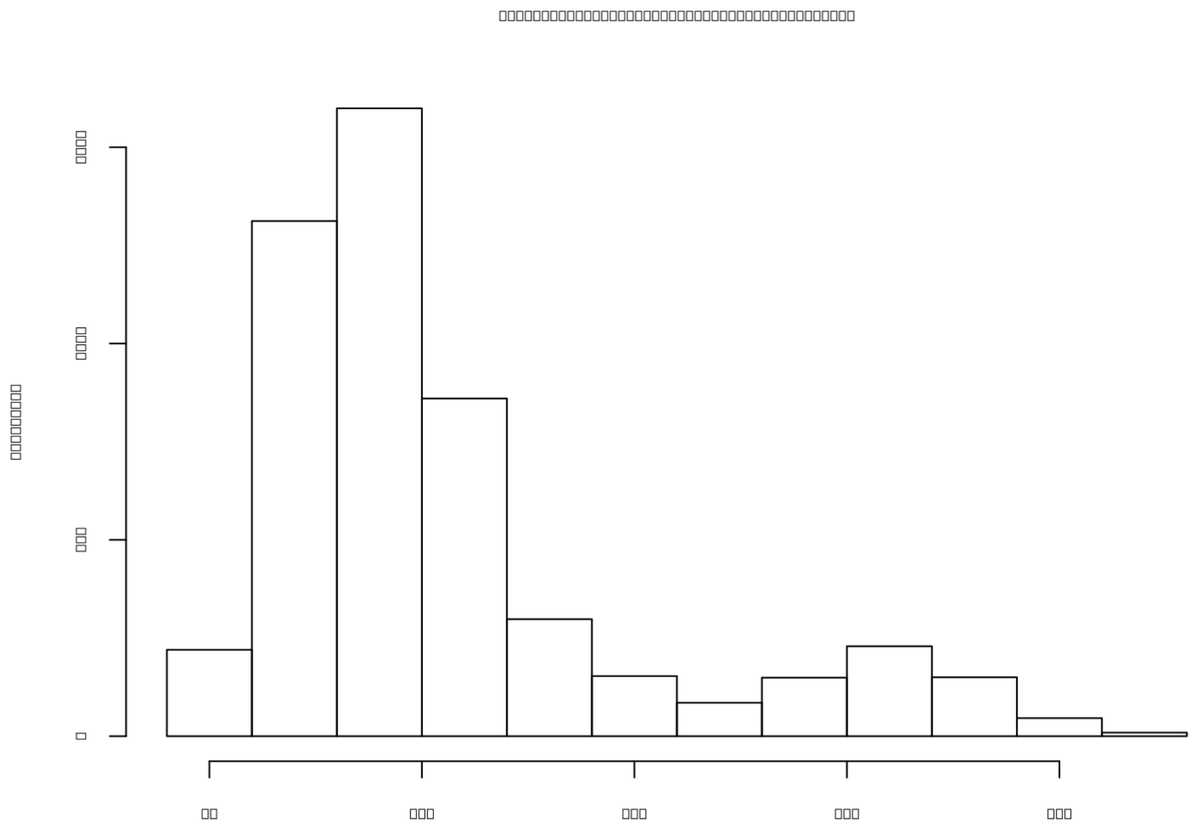
```
# Histogram data without plot.
hist(health_data$avg_glucose_level, plot = FALSE)
```

```
$breaks
 [1]  40  60  80 100 120 140 160 180 200 220
240 260 280

$counts
 [1]  220 1312 1599  860  298  153   85  149
229  150   46    9

$density
 [1] 2.152642e-03 1.283757e-02 1.564579e-02
8.414873e-03 2.915851e-03
 [6] 1.497065e-03 8.317025e-04 1.457926e-03
2.240705e-03 1.467710e-03
[11] 4.500978e-04 8.806262e-05

$mids
 [1]  50  70  90 110 130 150 170 190 210 230
250 270
```

Univariate plots: histogram (cont'd)

- However, we can also produce a fully visualized histogram

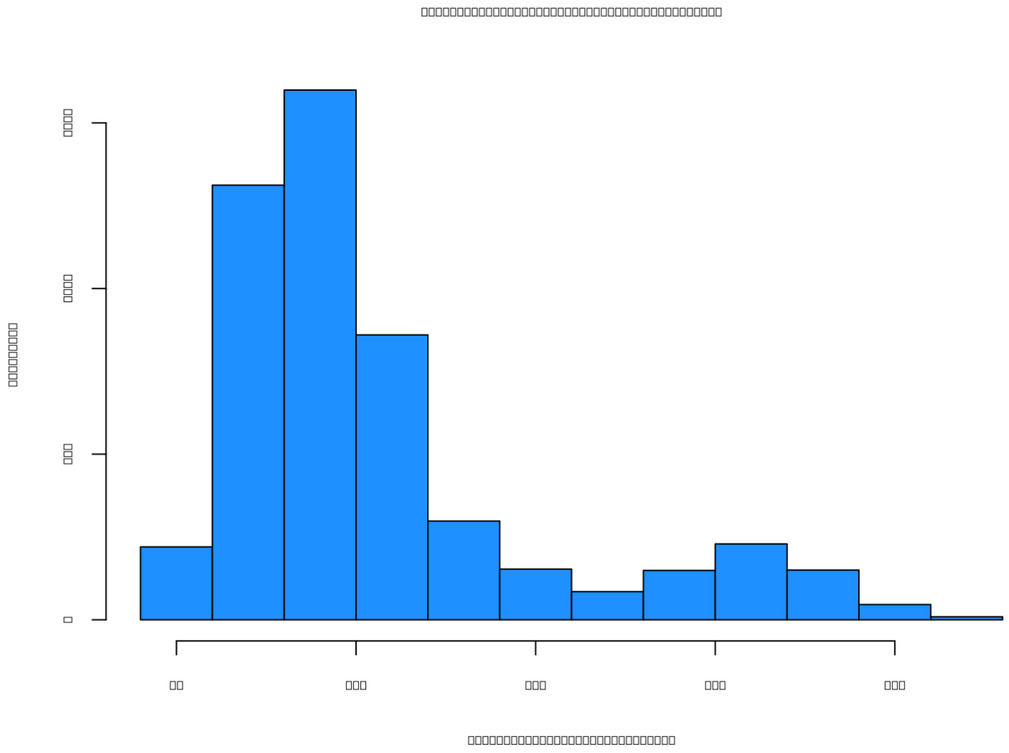
```
# Univariate plot: histogram.
hist(health_data$avg_glucose_level)
```



Univariate plots: histogram (cont'd)

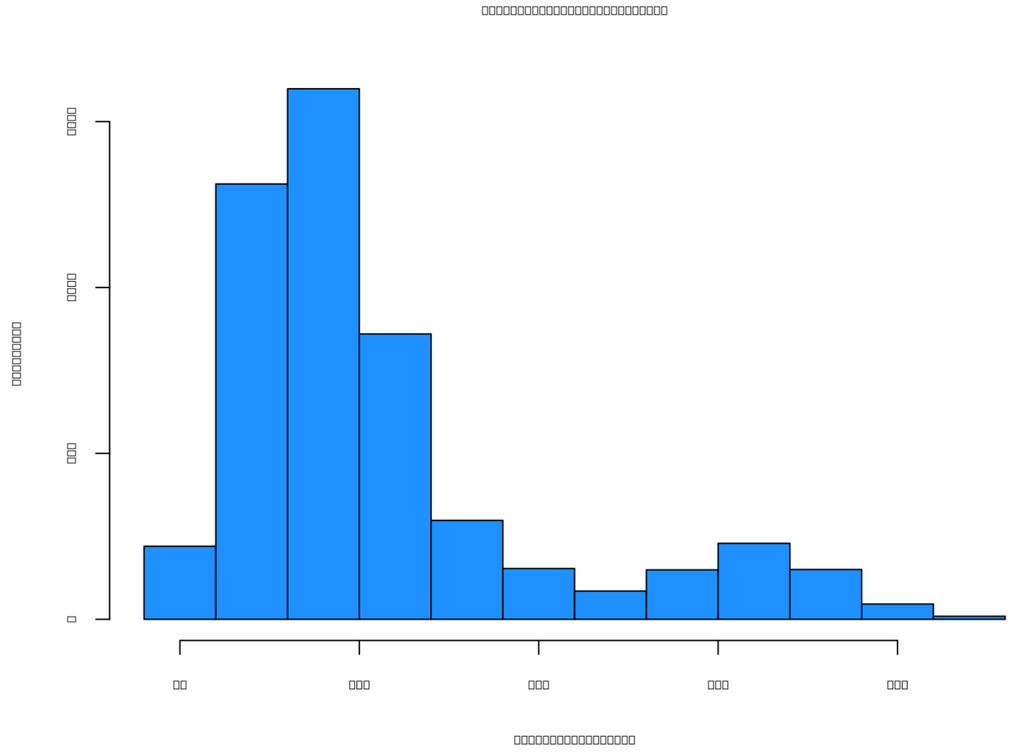
- We can add color

```
# Customize your histogram.
hist(health_data$avg_glucose_level,
     col = col_sample[1]) #<- set color
```



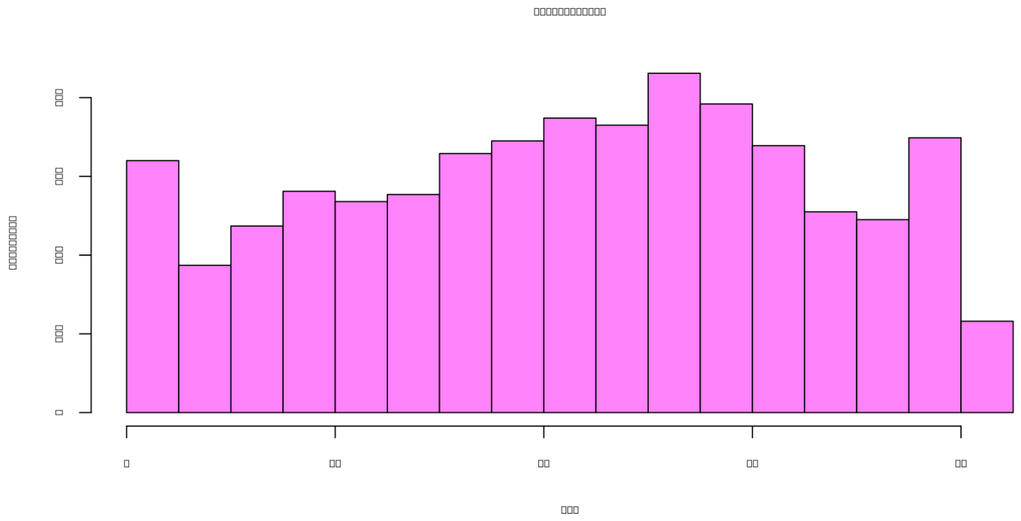
- We can also add a title

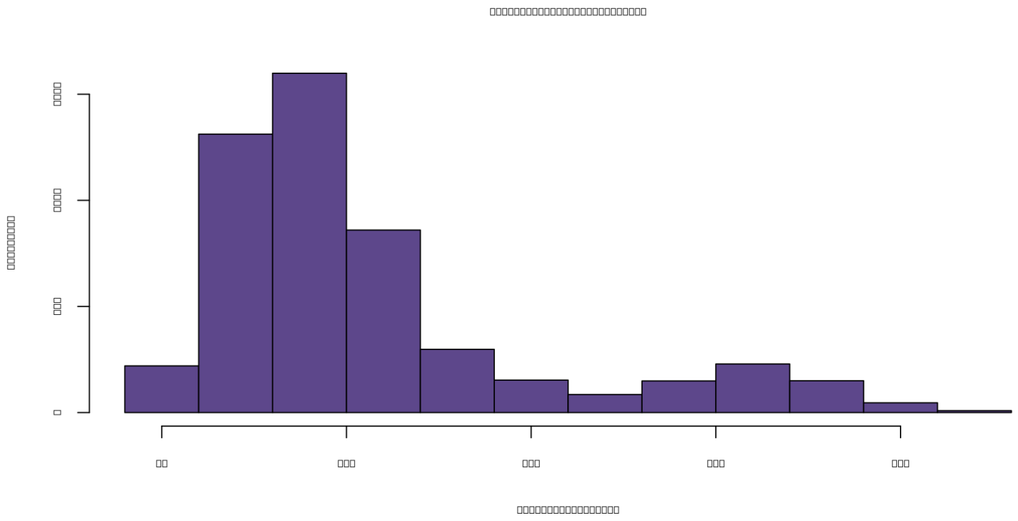
```
hist(health_data$avg_glucose_level,
     col = col_sample[1],
     xlab = "avg_glucose_level",   # x-axis label
     main = "Dist. of avg_glucose_level") # title
```



Compare variables: histograms combined

```
# Make a combined histogram plot.
par(                                    #<- set plot area parameters with `par`
    mfrow = c(1, 2))                    #<- split area into 1 row 2 columns with `mfrow`
hist(health_data$age,   #<- add 1st histogram
     col = col_sample[2],
     xlab = "age",
     main = "Dist. of age")
hist(health_data$avg_glucose_level,   #<- add 2nd histogram
     col = col_sample[3],
     xlab = "avg_glucose_level",
     main = "Dist. of avg_glucose_level")
```





Univariate plots: bar chart

- Bar charts are used to represent the data in the form of columns and rectangles
- The syntax of the bar chart is:

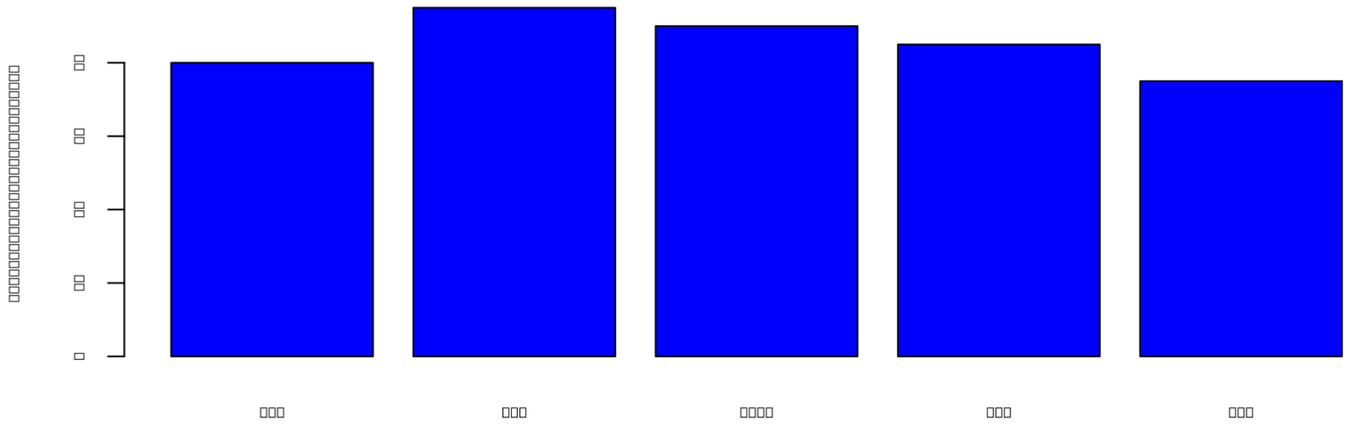
```
# Bar chart.
barplot(input,xlabel,ylabel,main,names.arg,col)
```

- It includes:
  - `input` represents the vector or matrix containing the numeric values
  - `xlabel` and `ylabel` denote the labels for x-axis and y-axis respectively
  - `names.arg` represents the vector of names that would appear under the bar
  - `col` parameter denotes the color

Univariate plots: bar chart (cont'd)

- Let's create a bar chart with a sample vector of a patient's pulse rate at different intervals

```
par(                                    #<- set plot area parameters with `par`
  mfrow = c(1, 1))
# Create the data for the chart
input <- c(80, 95, 90, 85, 75)
time_i <- c("6AM", "9AM", "12PM", "3PM", "6PM")

# Plotting the bar chart
barplot(input, names.arg=time_i, xlab="Time intervals", ylab="pulse rate of sample patient",
col="blue")
```



Module completion checklist

Objective	Complete
Describe and build univariate plots to illustrate patterns in the data	✔
Discuss and create bivariate and multivariate plots to illustrate patterns in data	

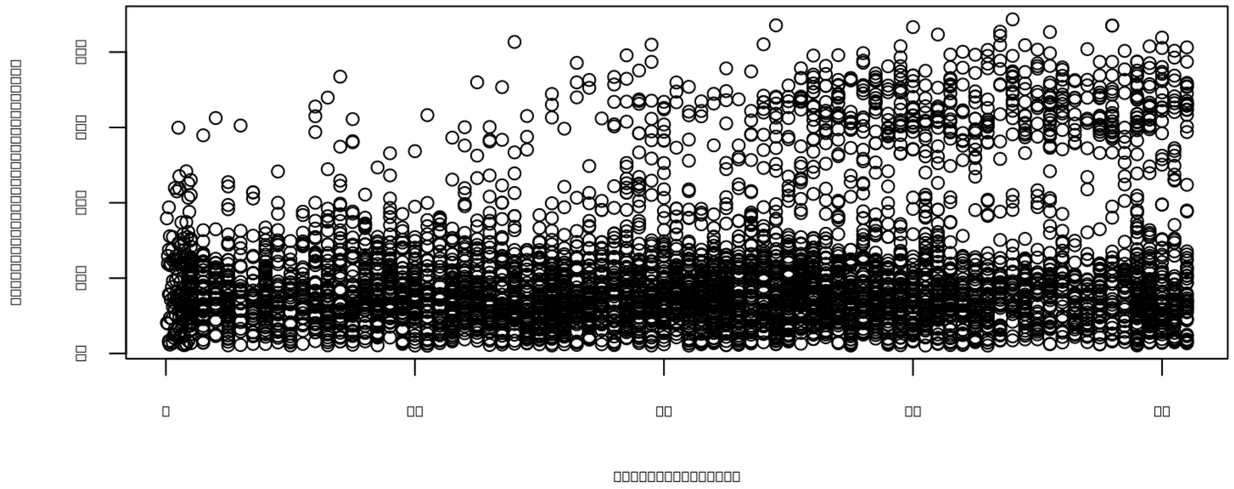
Bivariate plots

- **Bivariate plots** are visualizations that represent the **relationship between two variables**
- As was the case when examining single variables, there are several essential characteristics of the relationship between two variables that are of interest:
  - the `form of` the relationship
  - the `strength` of the relationship
  - the `dependence` of the relationship on external circumstances.
- By plotting two variables within the same space, we can add other features, like regression lines, that assist in predicting

Bivariate plots: scatterplot

- One of the main bivariate plots is the **scatterplot**, representing each observation of two variables as a point in a coordinate plane
- Let's create a generic scatterplot using one variable as the x and the other as the y

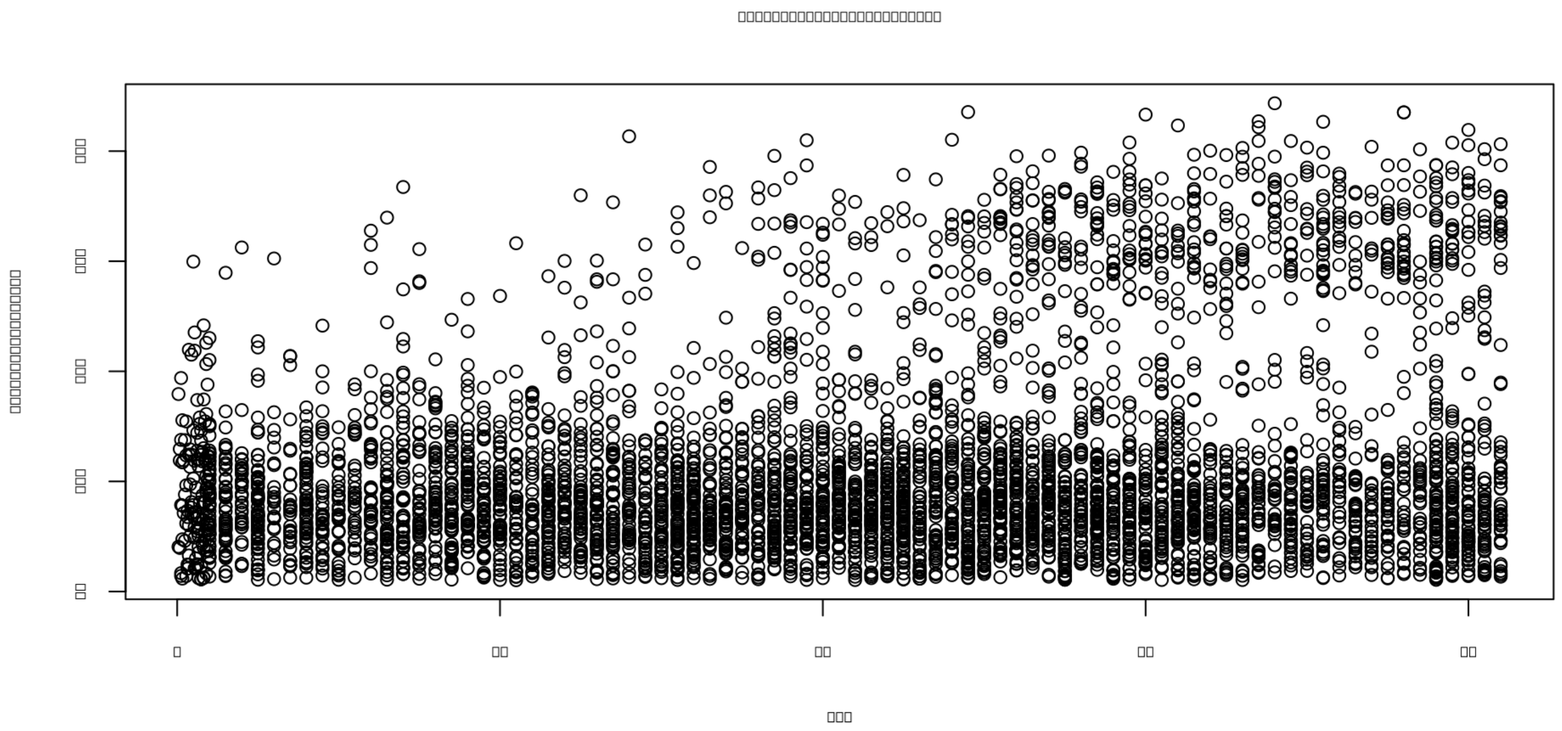
```
plot(health_data$age, #<- variable for x-axis
     health_data$avg_glucose_level) #<- variable for y-axis
```



Bivariate plots: scatterplot (cont'd)

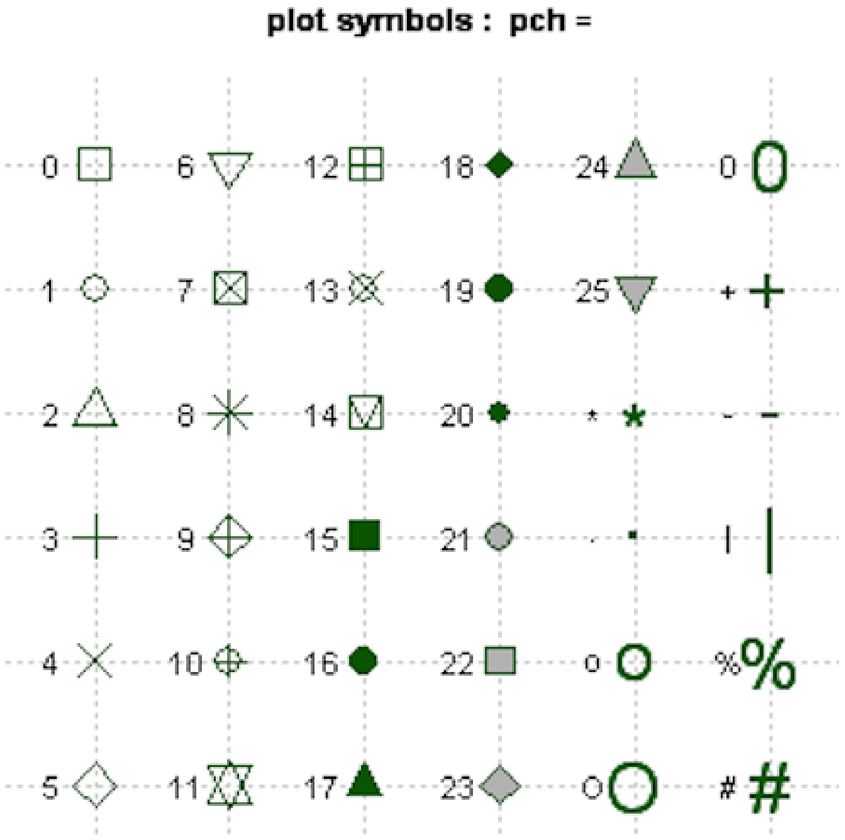
- We can build on our base by adding labels and a title

```
plot(health_data$age,
     health_data$avg_glucose_level,
     xlab = "age",
     ylab = "avg_glucose_level",
     main = "age vs avg_glucose_level")
```

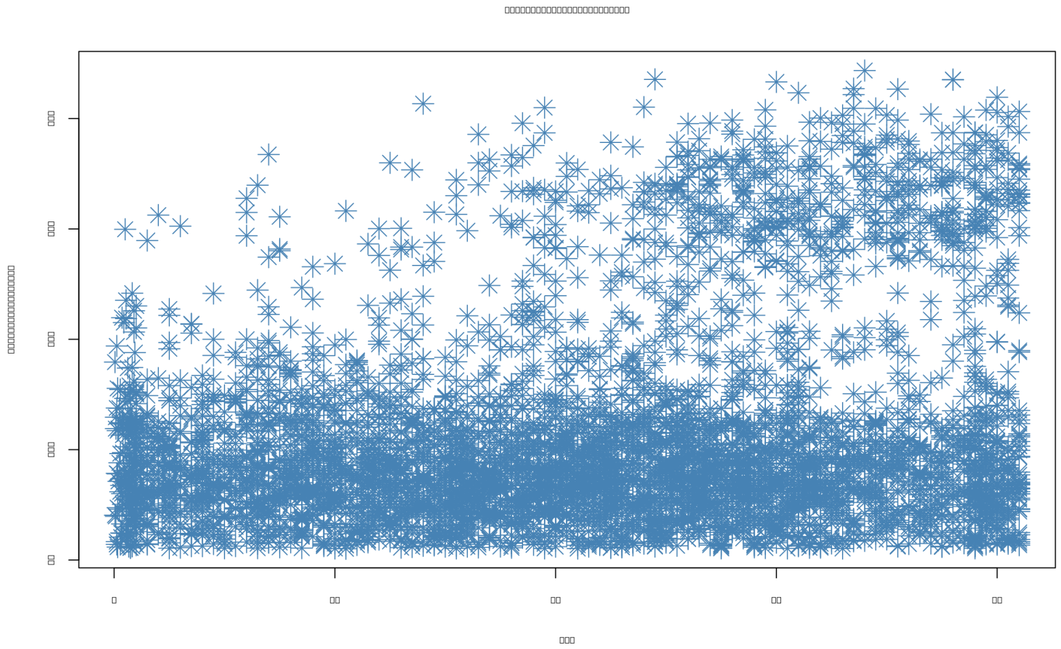


Bivariate plots: scatterplot (cont'd)

- Select a symbol for the observations with `pch` option from the following options

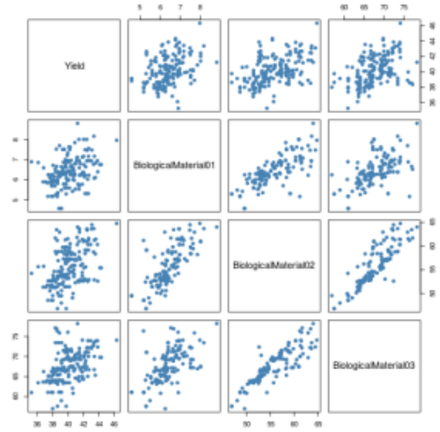


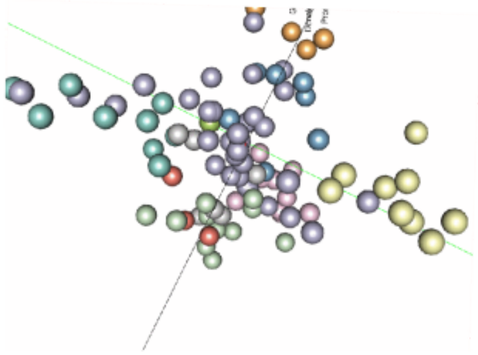

```
# Customize your scatterplot.
plot(health_data$age,
     health_data$avg_glucose_level,
     xlab = "age",
     ylab = "avg_glucose_level",
     main = "age vs avg_glucose_level",
     pch = 8,    #<- type of symbol to use
     cex = 2,    #<- scale of the point
     col = "steelblue")
```

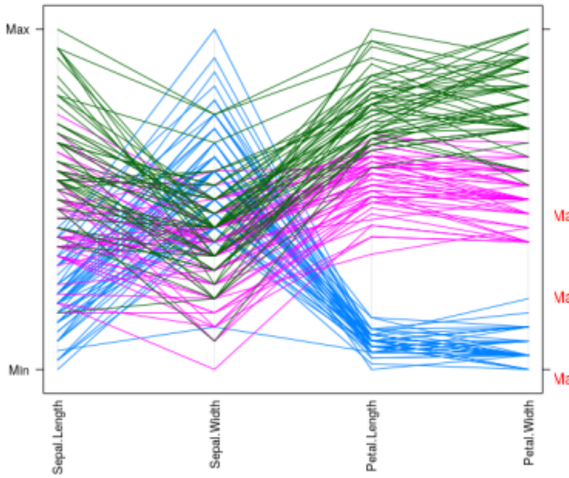


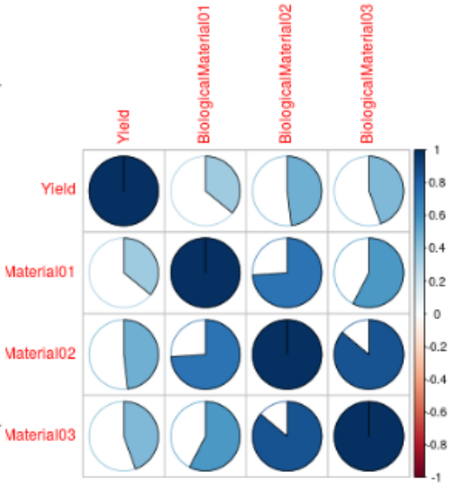
Multivariate plots

- **Multivariate plots** allow you to visualize the data distribution and relationships among **three or more variables**
- Examples include `scatterplot matrix`, `correlation plot`, `parallel coordinates plot`, various compound plots using multiple juxtaposed variables
- Multivariate plots are most helpful in learning how **groups of variables** are related, as well as in combination with **machine learning methods** that require specific complex inputs





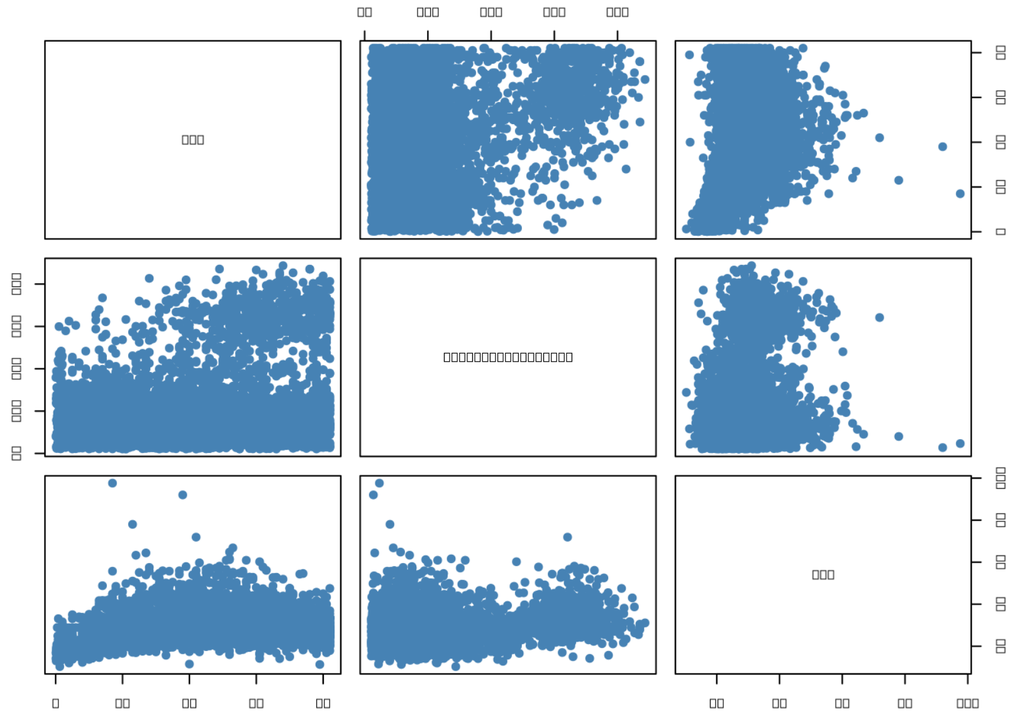




Multivariate plots: scatterplot matrix

- A **scatterplot matrix** allows us to glance at relationships between multiple variables quickly
  - The matrix is symmetrical around the diagonal
  - Each variable occupies the same row and column
- To see the scatterplot for variables 1 and 2, select the tile in row 1 and column 2
- To create this plot, use the `pairs` function to pass a range of variables

```
# Plot scatterplots for many variables.
# Select quantitative variables
health_quant <- health_data[, c("age",
"avg_glucose_level", "bmi")]

pairs(health_quant , #<- select variables
      pch = 19,           #<- set symbol
      col = "steelblue") #<- set color
```

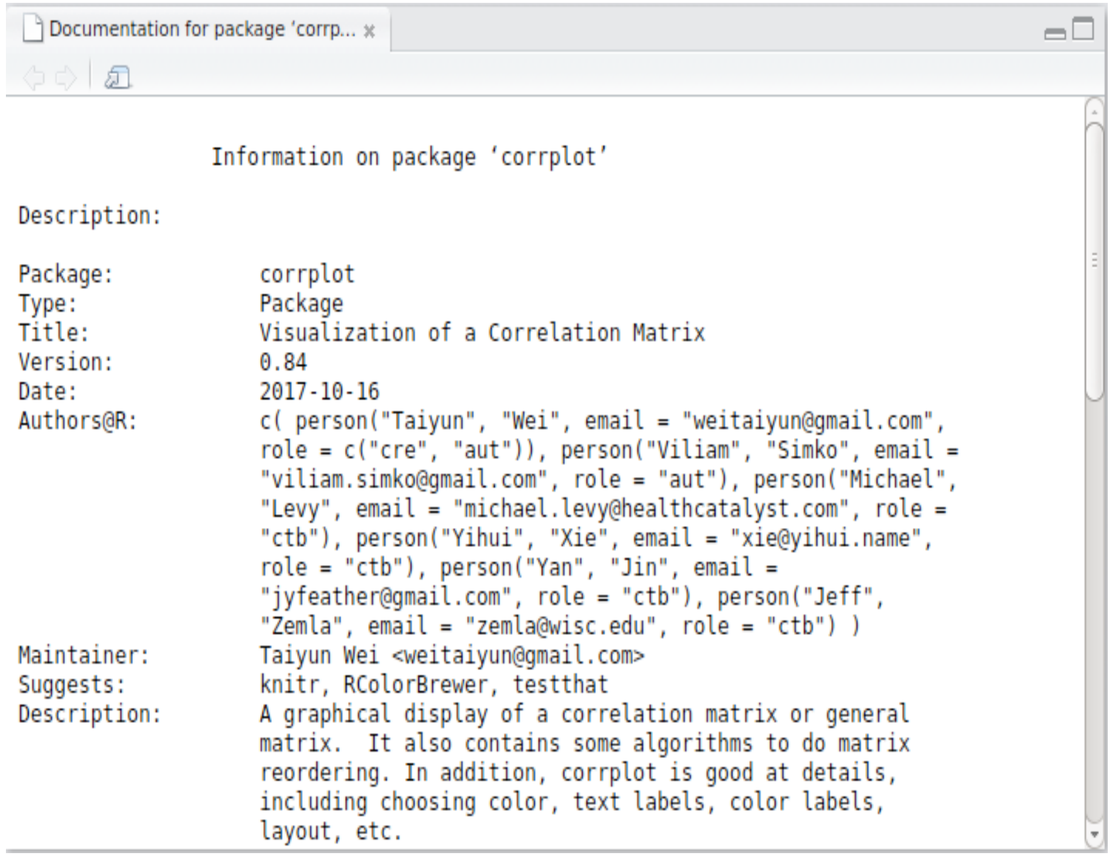


Installing `corrplot` package

- A **correlation plot** is a matrix representing the **correlation** between variables in each tile rather than the point distribution
- We can use the `corrplot` package to create these visualizations

```
# Install package through command line.
#install.packages("corrplot")

# Load the package into the environment.
library(corrplot)
```

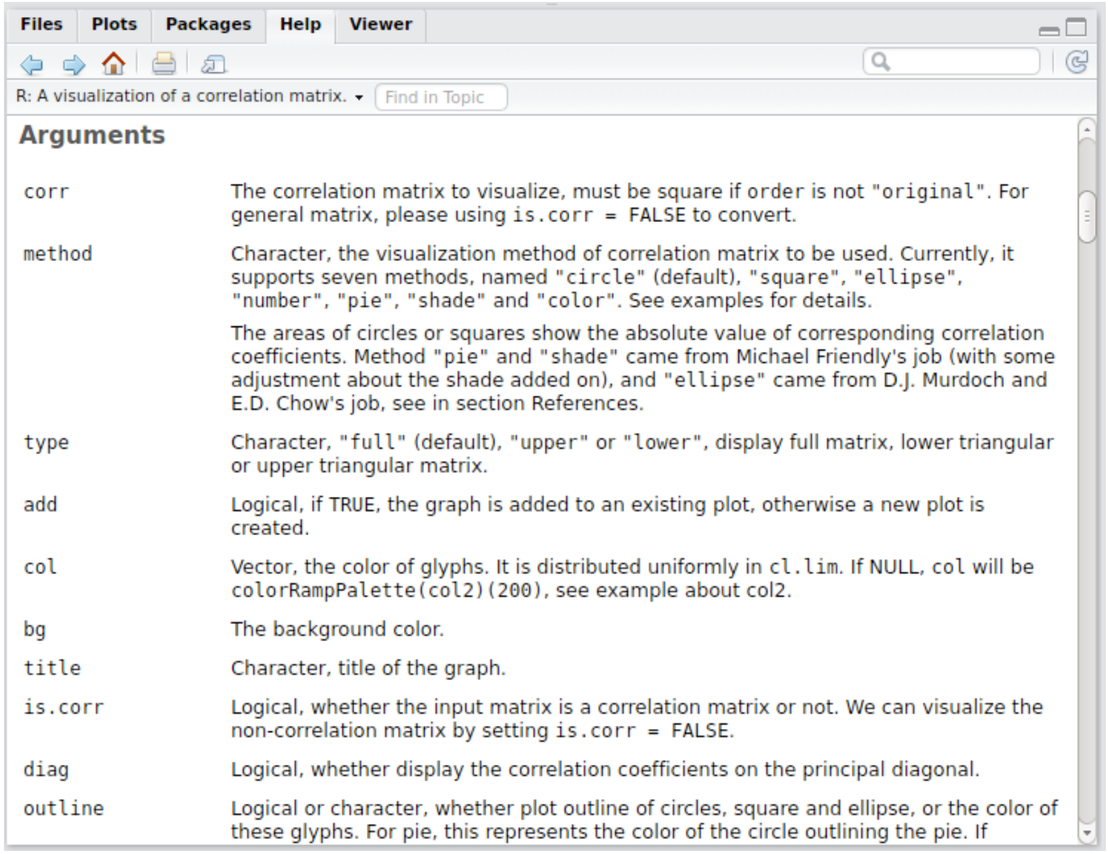
```
corrplot 0.88 loaded
```



Multivariate plots: correlation plot

```
# View package documentation.
library(help = "corrplot")

?corrplot
```

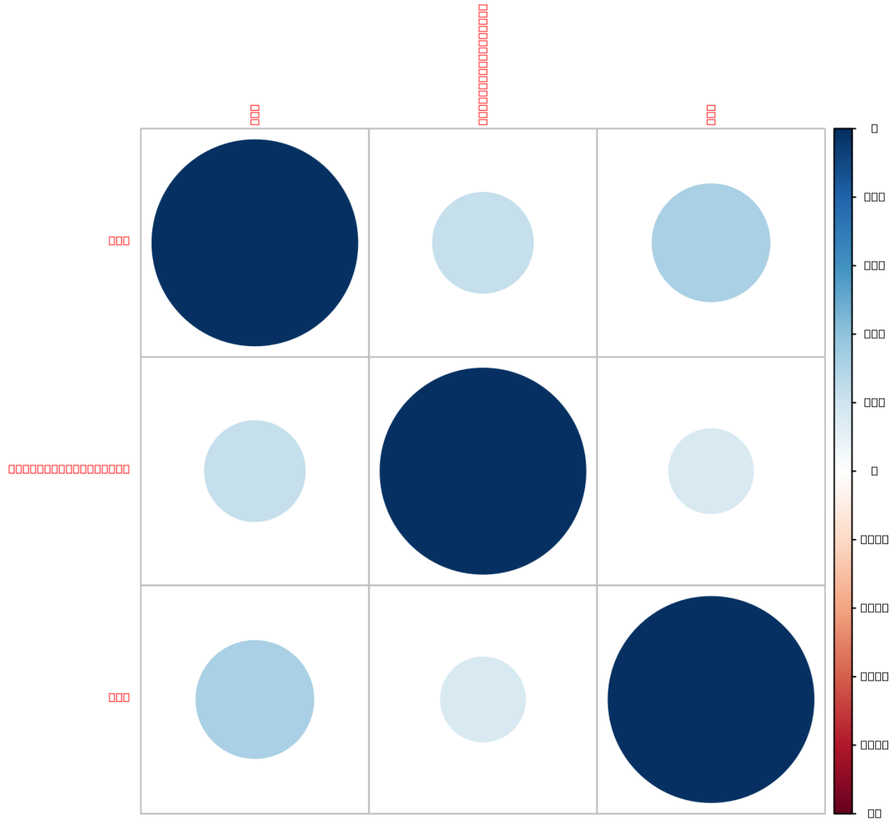
- The only argument required for `corrplot` is a **correlation matrix** of the variables we want the plot to represent
- We can generate this correlation matrix using `cor`



Multivariate plots: correlation plot (cont'd)

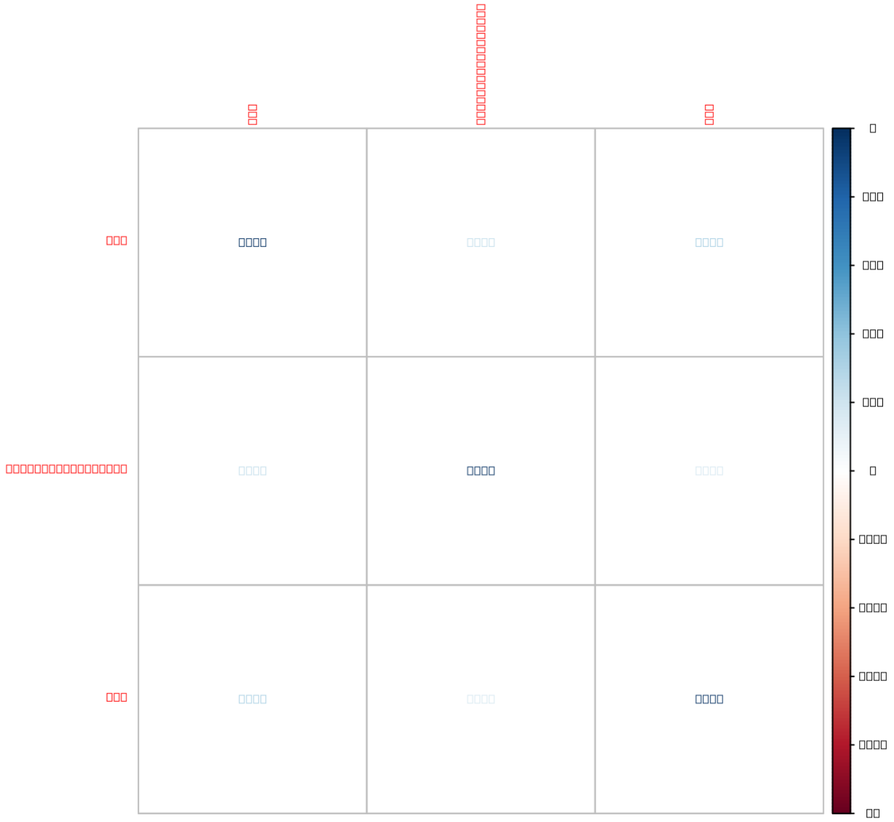
- The size of the circle represents the abs. value of the corr. coefficient: $0 \leq cor \leq |1|$
- The color of the circle represents the `sign` (i.e., **positive** vs. **negative**)
- From this plot we notice that:
  - All variables are positively correlated
  - `age` & `bmi` have a relatively higher corr. coefficient of about `0.33`
  - `bmi` & `avg_glucose_level` have relatively lower coefficient of over `0.17`

```
# Create correlation plot.
health_cor = cor(health_quant)
corrplot(health_cor)
```

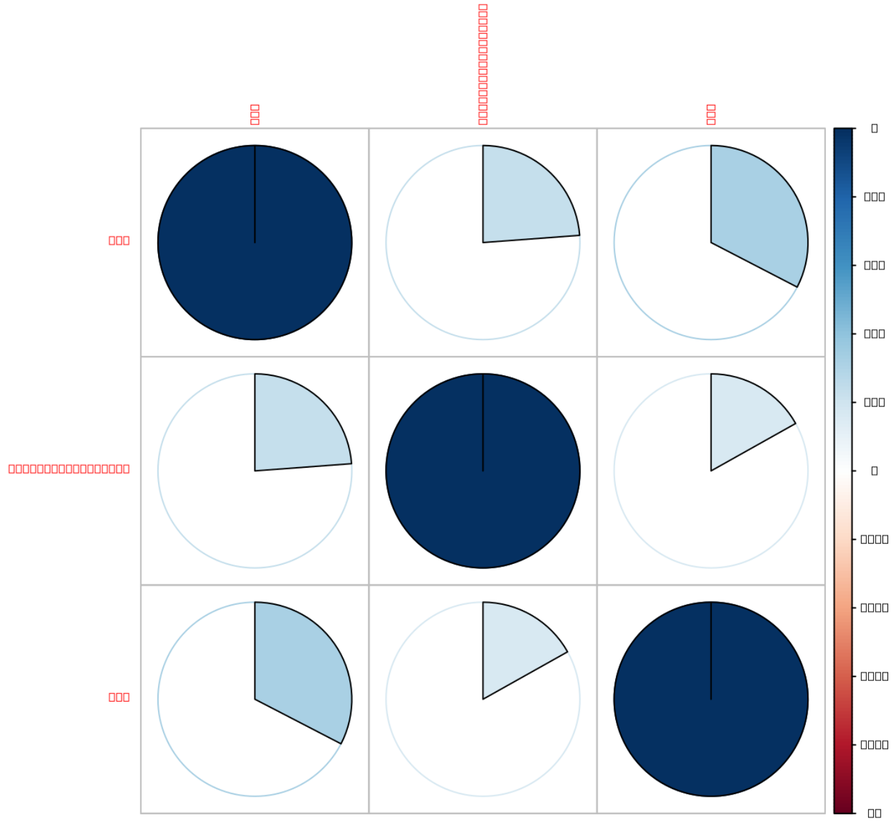


Multivariate plots: correlation plot (cont'd)

```
# The default method is "circle", to
# display the correlation coefficient
# instead, use method "number".
corrplot(health_cor,method = "number")
```



```
# To display correlation coefficient as
# a portion of a circle proportionate to
# correlation coefficient use method "pie".
corrplot(health_cor,method = "pie")
```



Knowledge check

Exercise

You are now ready to try tasks 1-8 in the exercise for this topic

Module completion checklist

Objective	Complete
Describe and build univariate plots to illustrate patterns in data	✔
Discuss and create bivariate and multivariate plots to illustrate patterns in data	✔

Basic Data Visualization: Topic Summary

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

- Basics of Univariate, Bivariate & Multivariate plots
- Building plots with base r

Congratulations on completing this module!