Lab 2 – Descriptive statistics

PROBABILITY AND STATISTICS - LABORATORY

BACHELOR'S DEGREE IN COMPUTER SCIENCE

A.Y. 2022/2023

DOTT. MATTEO CALGARO
matteo.calgaro_01@univr.it

Descriptive statistics and visualization

Descriptive statistics is a branch of statistics that involves the **collection**, **organization**, **summarization**, and **presentation** of data. It provides a way to describe and summarize the characteristics of a dataset, such as its **central tendency**, **dispersion**, **shape**, and **distribution**.

Descriptive statistics can be used to provide insight into the data. For example, we can use descriptive statistics to determine the average age of a group of people, the spread of their ages, and how their ages are distributed across the group.

When a dataset contains more than one variable, depending on their types (*e.g.*, discrete, continous, categorical…), descriptive statistics can be used to study **associations** between those variables (*e.g.*, covariance and correlation).

Visualization can help make patterns and trends in the data more apparent, and can provide a more intuitive understanding of the data than a table of numbers alone. Some common types of visualizations include **scatter plots**, **bar charts**, **line graphs**, and **histograms**.

Descriptive statistics in R

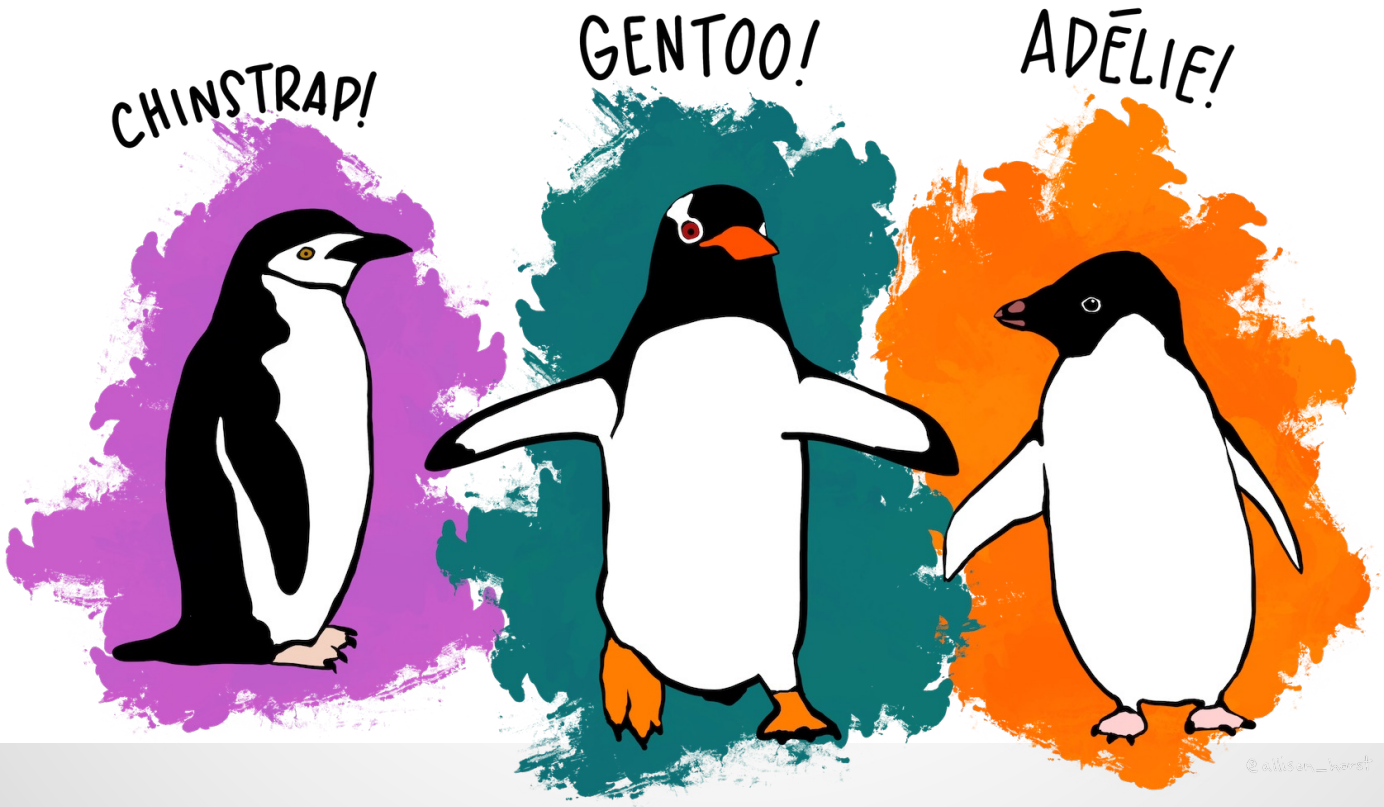
Function	Description
`mean()`	Mean: calculates the average value of a dataset
`median()`	Median: calculates the middle value of a dataset
`range()`	Calculates the difference between the minimum and the maximum
`IQR()`	Interquartile Range: calculates the spread of a dataset by computing the difference between the third and the first quartiles
`var()`	Variance: measures the spread of a dataset by calculating the average squared deviation from the mean
`sd()`	Standard deviation: measures the spread of a dataset by calculating the square root of the variance
`skewness()`	Measures the degree of asymmetry in a dataset (requires the "e1071" package)
`kurtosis()`	Measures the degree of peakedness in a dataset (requires the "e1071" package)
`cov()`	Measures the degree of linear association between two variables in a dataset
`cor()`	Correlation coefficient: measures the degree of linear association between two variables in a dataset, ranging from -1 to 1 (it is a standardized covariance)

Other useful functions

Function	Description
`summary()`	Provides a summary of a dataset, including the minimum and maximum values, quartiles, mean, and median for each variable
`table()`	Creates a contingency table of categorical variables, showing the frequency of each combination of categories
`prop.table()`	Creates a contingency table of categorical variables, showing the proportion of each combination of categories
`quantile()`	Calculates the value of a specific quantile (*e.g.*, the 90th percentile) of a dataset

Meet the penguins

The palmerpenguins data contains size measurements for three penguin species observed on three islands in the Palmer Archipelago, Antarctica.



The Palmer Archipelago penguins.
Artwork by @allison_horst.

Meet the penguins

Data are contained into the `palmerpenguins` package available in CRAN. To obtain it we need to install the package and load it.

```
install.packages("palmerpenguins")
library("palmerpenguins")

data(package = "palmerpenguins")
data("penguins")
```

The package contains two datasets: `penguins` and `penguins_raw`. We will use `penguins`. You can find the dataset as a .RData file at moodle. Download it and import the dataset by placing it in the right directory. Then use the **`load()`** function.

To inspect the data, just type the name of the dataset, use the function **`head()`** (to see the first rows of the dataset), the **`str()`** function (to get some basic information for each column of the dataset), or click the name of the dataset from the environment.

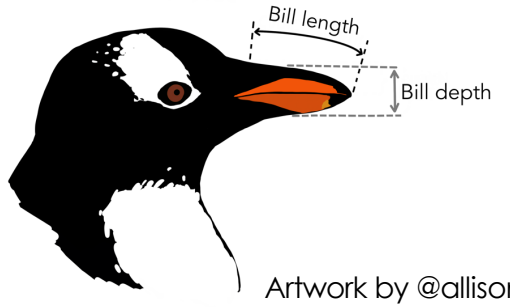
```
> penguins                    > str(penguins)
> head(penguins)              > View(penguins)
```

Meet the penguins – first inspection

Looking at a full table is not always very straightforward. Let's see, instead, the output of the **`str()`** function:

```
tibble [344 × 8] (S3: tbl_df/tbl/data.frame)
 $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
 $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
 $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
 $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
 $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
 $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

The dataset in the package is of type tibble (a sort of data.frame). It has 344 observations and 8 variables. For each variable in the dataset, the name, type, and the first values are returned.



Artwork by @allison_horst.

Exercise 1 – Create a new mean and sd function

A. Create the **Lab2** project. Use the same structure used for **Lab1**: **scripts**, **plots**, and **data** directories.

B. Install the **palmerpenguins** package, load the **penguins** dataset or, alternatively, download the .RData object from moodle and import it after placing it inside the data directory of the project (hint: use the **`load()`** function).

C. Compute the mean, the standard deviation, and the median for the numeric variables of the dataset.

D. Create a function called **`stat_auto`** that simultaneously returns both the mean and the standard deviation of a given vector (hint: return an object of type list or simply a vector). Then try it on the same numeric variables in C. to check the results (hint: if you obtain NA maybe you forgot to remove NA terms in the vector).

E. Create a function called **`stat_manual`** that simultaneously returns both the mean and the standard deviation of a given vector without using the `mean()` and the `sd()` functions (hint: you can use **`length()`**, **`sum()`**, and **`na.omit()`** functions). Then try it on the same numeric variables in C. to check the results.

SOLUTION

Exercise 1 – Create a new mean and sd function

B. Install the **palmerpenguins** package, load the **penguins** dataset or, alternatively, download the .RData object from moodle and import it after placing it inside the data directory of the project.

```
# Install the package
install.packages("palmerpenguins")
data("penguins")
penguins

# Or load the file you can find in moodle
load("./data/penguins.RData")
```

SOLUTION

Exercise 1 – Create a new mean and sd function

c. Compute the mean, the standard deviation, and the median for the numeric variables of the dataset.

```
# To inspect the type of each column
str(penguins)

# For the bill_length_mm variable
mean(penguins$bill_length_mm, na.rm = TRUE)
sd(penguins$bill_length_mm, na.rm = TRUE)
median(penguins$bill_length_mm,na.rm = TRUE)

# For the bill_depth_mm variable
mean(penguins$bill_depth_mm, na.rm = TRUE)
sd(penguins$bill_depth_mm, na.rm = TRUE)
median(penguins$bill_depth_mm,na.rm = TRUE)
```

SOLUTION

Exercise 1 – Create a new mean and sd function

D. Create a function called **`stat_auto`** that simultaneously returns both the mean and the standard deviation of a given vector. Then try it on the same numeric variables in C. to check the results.

```
stat_auto <- function(vec, removeNA = TRUE){
    avg <- mean(vec, na.rm = removeNA)
    stddev <- sd(vec, na.rm = removeNA)
    output <- list("Mean" = avg, "SD" = stddev)
    return(output)
}

stat_auto(penguins$bill_length_mm)
stat_auto(penguins$bill_depth_mm)
```

SOLUTION

Exercise 1 – Create a new mean and sd function

E. Create a function called **`stat_manual`** that simultaneously returns both the mean and the standard deviation of a given vector without using the `mean()` and the `sd()` functions. Then try it on the same numeric variables in C. to check the results.

```
stat_manual <- function(vec, removeNA = TRUE){
    if(removeNA == TRUE) # if(removeNA) also valid
        vec <- na.omit(vec)
    n <- length(vec)
    avg <- sum(vec)/n
    stddev <- sqrt(sum((vec - avg)^2) / (n - 1))
    return(list("Mean" = avg, "SD" = stddev))
}

stat_manual(vec = penguins$bill_length_mm, removeNA = TRUE)
stat_manual(vec = penguins$bill_depth_mm, removeNA = TRUE)
```

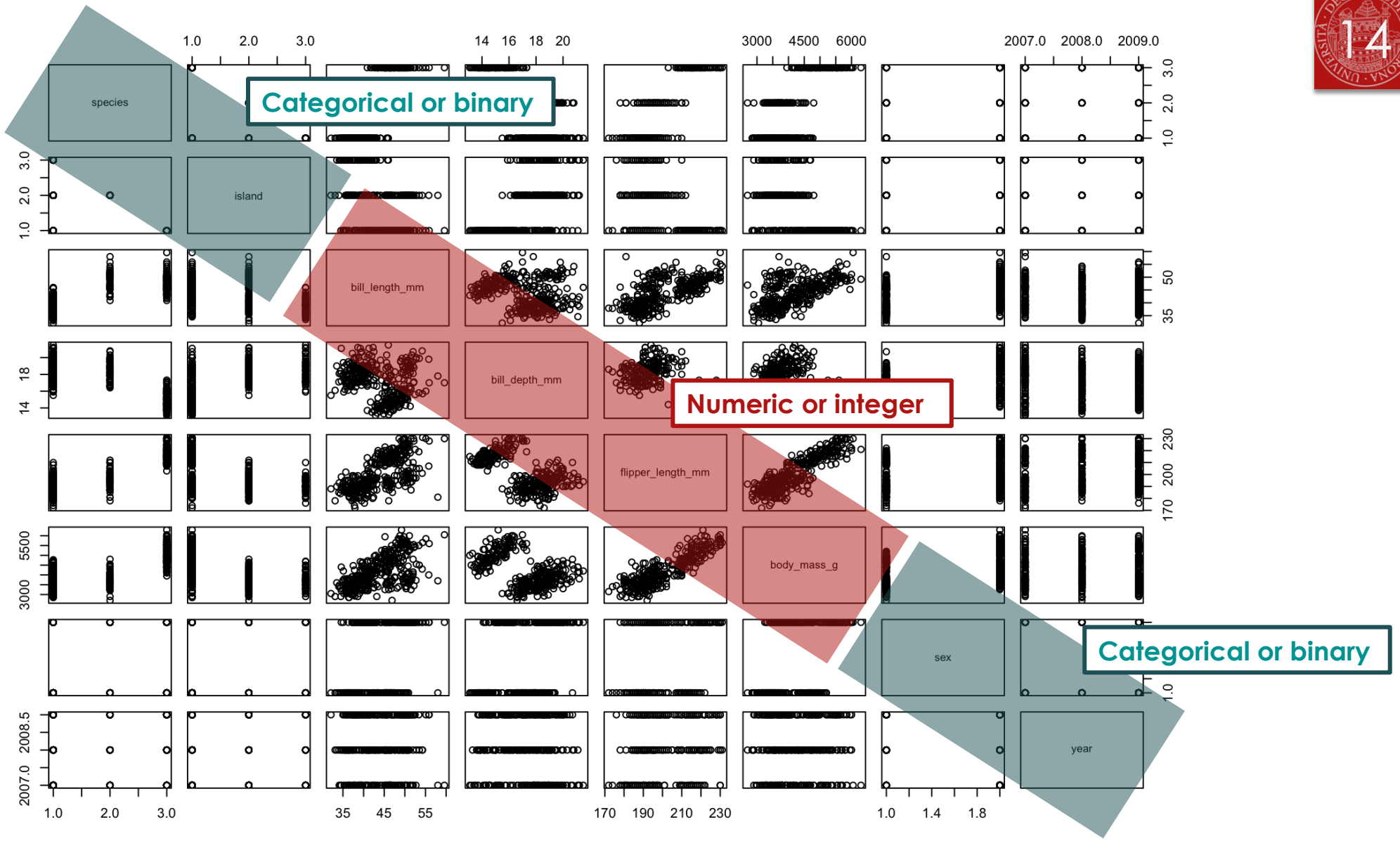
Meet the penguins – graphical inspection

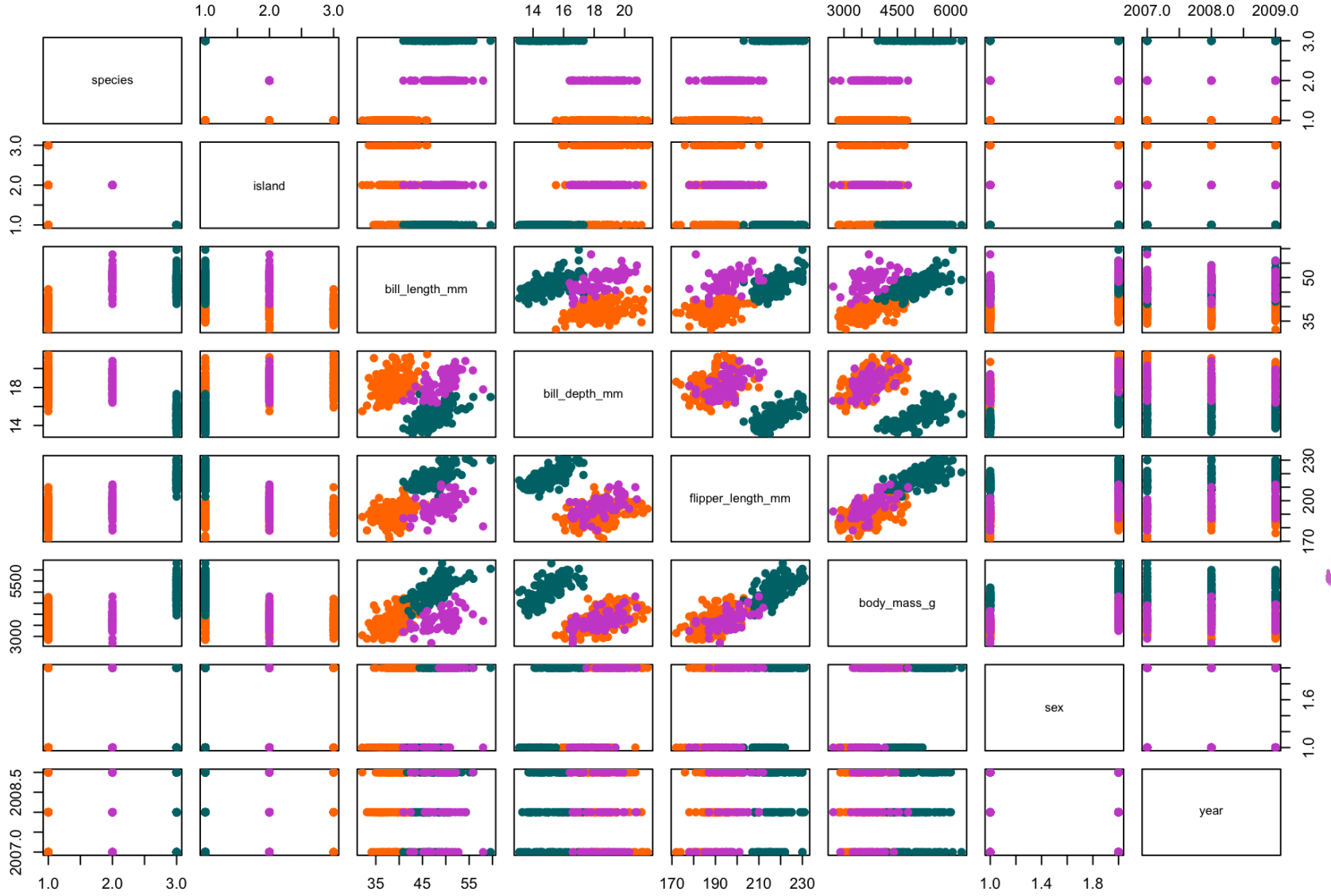
Even after calculating summary statistics, it can be difficult to understand the data. On the contrary a simple graph could bring more information to our mind.

```
plot(penguins)
```

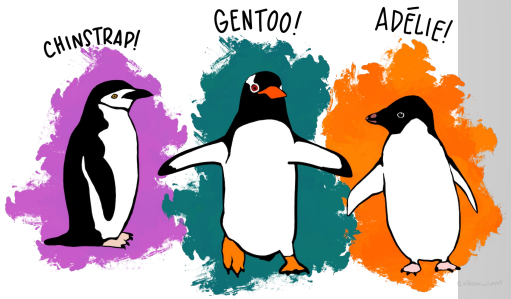
It goes even better adding color values for each penguin to see better the species.

```
pCol <- c('#ff8301', '#bf5ccb', '#057076')
names(pCol) <- c('Adelie', 'Chinstrap', 'Gentoo')
plot(penguins, col = pCol[penguins$species], pch = 19)
```


Categorical or binary
Numeric or integer
Categorical or binary
species
island
bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
sex
year



Species is important



Refined graphical representations

During the last lesson we learnt how to do simple plots using the **`plot()`**, **`barplot()`**, **`hist()`**, and **`boxplot()`** functions. Today we will introduce the **ggplot2** package. It offers:

- A **layered approach** which allows you to build a plot step-by-step, adding different layers of information, such as points, lines, or labels, to create a final plot.
- The possibility to easily **map variables** to different aesthetics of the plot, such as color, shape, or size.
- A range of built-in **themes** that make it easy to create professional-looking plots with consistent font sizes, colors, and layout.
- A wide range of **plot types**, including scatter plots, line plots, bar plots, histograms, and more.
- The possibility to split plots into multiple panels (**facets**), based on one or more variables.

ggplot2 offers a more streamlined and flexible approach to data visualization, making it a popular choice among R users.

Graphically represent the penguins

```
mass_flipper <- ggplot(data = penguins,
                       aes(x = flipper_length_mm, y = body_mass_g)) +
    geom_point(aes(color = species, shape = species), size = 3, alpha = 0.8) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    labs(title = "Flipper length and body mass",
         subtitle = "Colored by Adelie, Chinstrap and Gentoo Penguins",
         x = "Flipper length (mm)",
         y = "Body mass (g)",
         color = "Penguin species",
         shape = "Penguin species") +
    theme(legend.position = "bottom")

mass_flipper
```

ggplot: this is the starting point. Here we have to choose the **data** and the basic **aes**thetics (x, y axes)

geom_point: here we are asking to plot data as **points**. The coordinates are inherited from the ggplot aes. Here we can define other aesthetics for those points such as the **color** and the **shape**.

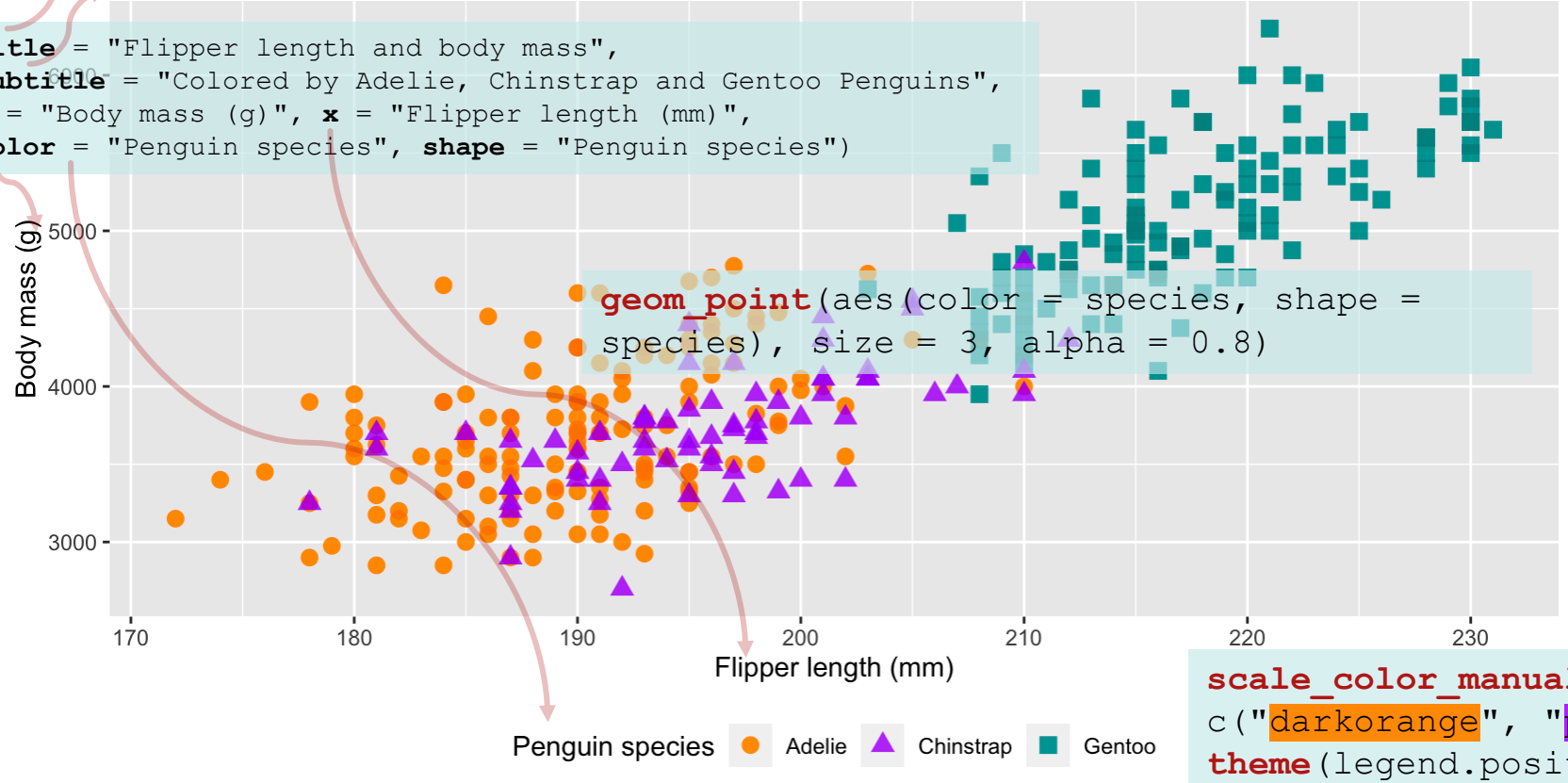
labs: here we define all the labels. From the title and subtitle, to the axis labels, to the legend labels about colors and shapes.

theme: here we choose where to plot the legend.

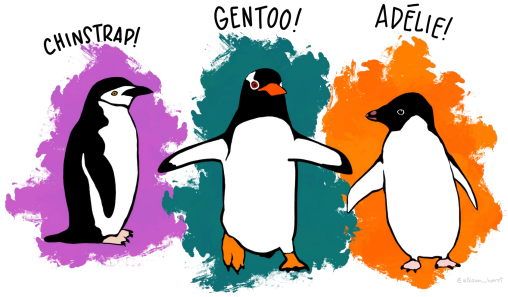
Graphically represent the penguins

```
ggplot(data = penguins,
       aes(x = flipper_length_mm, y = body_mass_g))
```

```
labs(title = "Flipper length and body mass",
     subtitle = "Colored by Adelie, Chinstrap and Gentoo Penguins",
     y = "Body mass (g)", x = "Flipper length (mm)",
     color = "Penguin species", shape = "Penguin species")
```

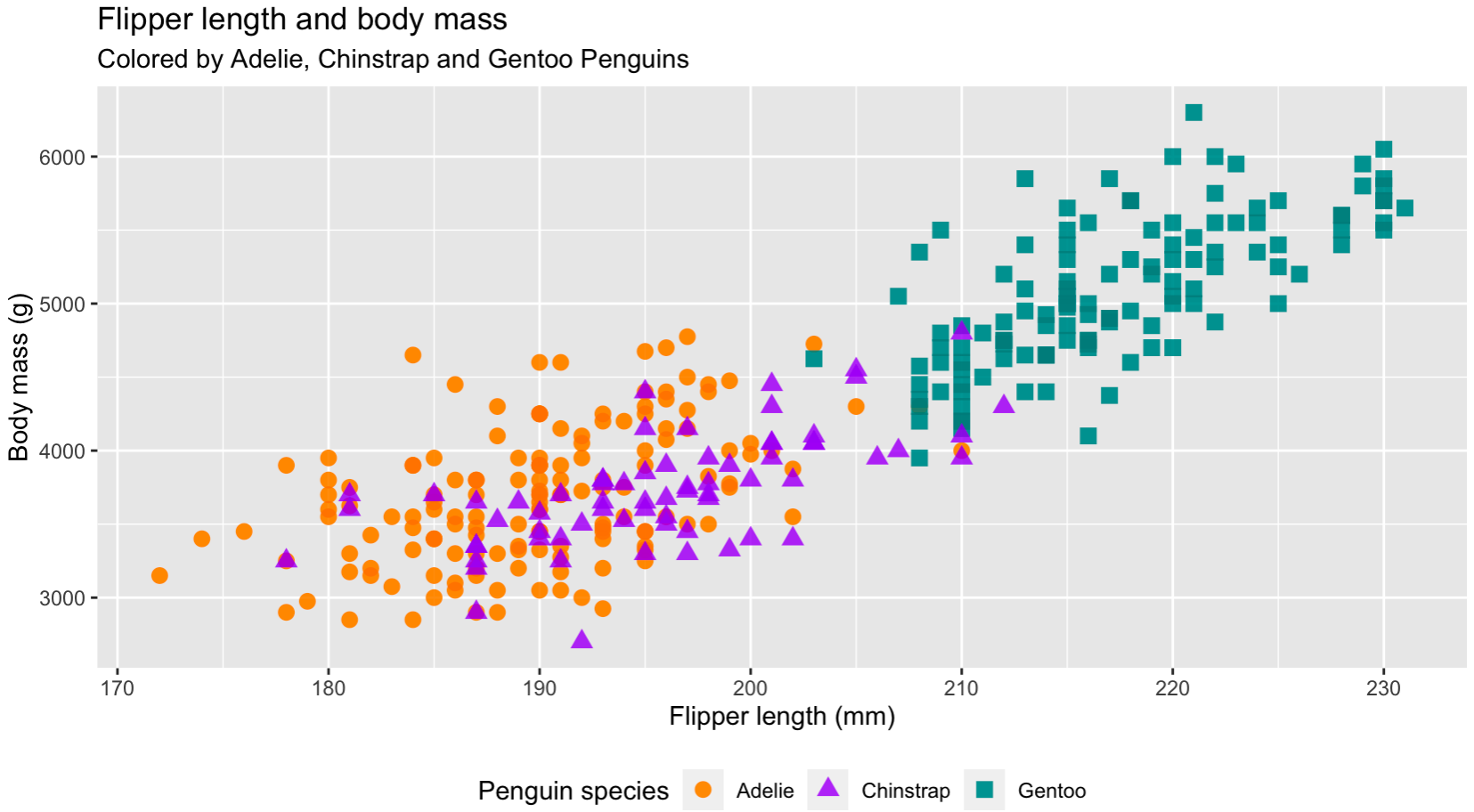


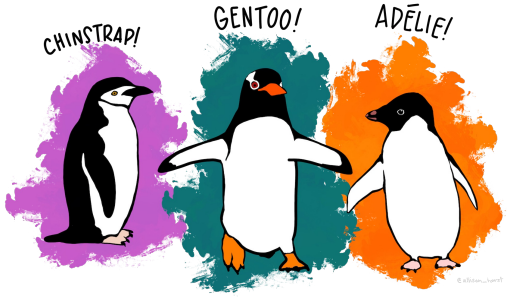

```
geom_point(aes(color = species, shape =
species), size = 3, alpha = 0.8)
```




```
scale_color_manual(values =
c("darkorange", "purple", "cyan4")) +
theme(legend.position = "bottom")
```

What is this plot telling us?





ggplot2 useful functions

Function	Description
`ggplot()`	This is the main function in ggplot2 that initializes a plot object and sets the data and aesthetics for the plot.
`aes()`	This function maps variables in the data to different aesthetic properties of the plot, such as color, shape, or size.
`geom_*()`	These are the geometric objects that define the type of plot to be created, such as points (`geom_point()`), lines (`geom_line()`), bars (`geom_bar()`, `geom_col()`), or histograms (`geom_histogram()`).
`facet_*()`	These functions split the plot into multiple panels based on one or more variables in the data, such as `facet_wrap()` or `facet_grid()`.
`scale_*()`	These functions customize the appearance of the plot, such as changing the axis labels, adjusting the color palette, or setting the range of the axis values.
`labs()`	This function is used to add titles and labels to the plot, such as a main title, axis labels, or legends.
`theme()`, `theme_*()`	These functions customize the overall appearance of the plot, such as changing the font size, background color, or grid lines.
`stat_*()`	These functions calculate summary statistics for the data and add them to the plot, such as the mean or standard deviation.
`coord_*()`	These functions control the coordinate system of the plot, such as setting the x and y axis limits or adjusting the aspect ratio

Exercise 2 – Table of frequencies

A. In the penguins dataset, transform a numeric variable to a categorical one by aggregating values into classes. Consider the flipper length variable and create 10mm wide classes using the **`cut()`** function (hint: use the **`range()`** function to determine the min and max values of the variable, then define a sequence for the cuts).

B. Use the **`table()`** function on the new variable generated by `cut()`. Then transform it into a data.frame object. Rename the columns accordingly using the **`colnames()`** function (hint: the second column correspond to the absolute frequencies).

C. Add the the columns for: relative frequencies, cumulative absolute frequencies, and cumulative relative frequencies.

D. Use the **`geom_col()`** function to plot the frequence of each class. Then, using the **`geom_text(aes(label = …))`** function, add the relative frequence as a percentage above each column (hint: substitute the … with the relative frequency values. Use the **`round()`** function to choose the appropriate number of digits).

SOLUTION

Exercise 2 – Table of frequencies

A. In the penguins dataset, transform a numeric variable to a categorical one by aggregating values into classes. Consider the flipper length variable and create 10mm wide classes using the **`cut()`** function.

```
flipper <- penguins$flipper_length_mm[!is.na(penguins$flipper_length_mm)]
range(flipper)
# 172 231

# For the break points, let's start from 170 instead of 172
classes <- seq(170, 235, 10)

flipper_cut <- cut(flipper, breaks = classes, include.lowest = TRUE,
                   right = FALSE)
# The output will be a vector of intervals, right exluded, left included
head(flipper_cut)
[1] [180,190) [180,190) [190,200) [190,200) [190,200) [180,190)
Levels: [170,180) [180,190) [190,200) [200,210) [210,220) [220,230]
```

SOLUTION

Exercise 2 – Table of frequencies

B. Use the **`table()`** function on the new variable generated by `cut()`. Then transform it into a data.frame object. Rename the columns accordingly using the **`colnames()`** function

```
flipper_df <- data.frame(table(flipper_cut))
colnames(flipper_df) <- c("flipper_length_mm", "absolute_freq")
```

C. Add the the columns for: relative frequencies, cumulative absolute frequencies, and cumulative relative frequencies.

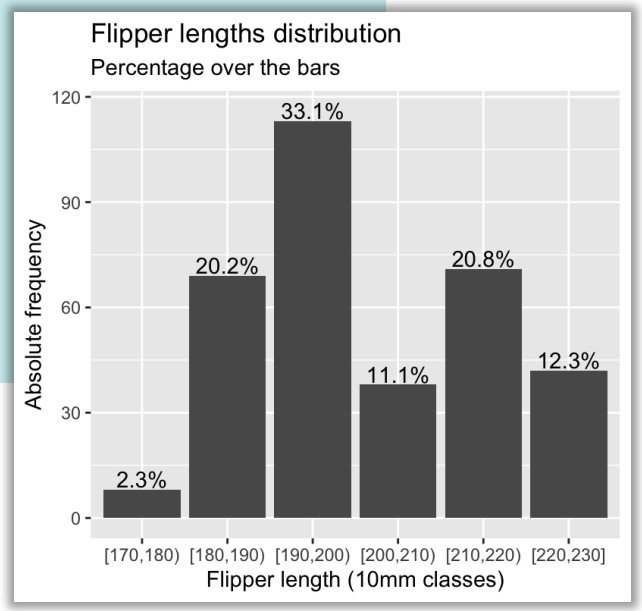
```
flipper_df$cumulative_absolute_freq <- cumsum(flipper_df$absolute_freq)
flipper_df$relative_freq <- flipper_df$absolute_freq /
                            sum(flipper_df$absolute_freq)
flipper_df$cumulative_relative_freq <- cumsum(flipper_df$relative_freq)
```

SOLUTION

Exercise 2 – Table of frequencies

D. Use the **`geom_col()`** function to plot the frequence of each class. Then, using the **`geom_text(aes(label = …))`** function, add the relative frequence as a percentage above each column.

```
ggplot(flipper_df, aes(x = flipper_length_mm, y = absolute_freq)) +
    geom_col() +
    geom_text(aes(label = paste0(
                round(relative_freq * 100, digits = 1), "%")),
                nudge_y = 3) +
    labs(x = "Flipper length (10mm classes)",
         y = "Absolute frequency",
         title = "Flipper lengths distribution",
         subtitle = "Percentage over the bars")
```



Exercise 3 – Histogram, Boxplot and quartiles

A. Using the **`geom_histogram()`** function of the ggplot2 package plot the flipper length distribution coloring each species with a different color (hint: use the fill argument of the **`aes()`** function to fill the histogram area and the position = "identity" argument of the **`geom_histogram()`**). Play with the binwidth argument. Try to insert `y = ..density..` in `aes()`. Do you notice any change?

B. About the flipper length, for each species of penguins compute the:

   1. sample mean;
   2. sample median;
   3. sample standard deviation (use a division by n-1);
   4. sample variance.

   (hint: to choose only a specific species use **`penguins[penguins$species == "Gentoo",]`**)

C. Using the **`geom_boxplot()`** function of the ggplot2 package plot the boxplot for the flipper length variable coloring each species with a different color (hint: use the color argument of the **`aes()`** function).

D. Compute the flipper length quartiles for the "Gentoo" penguins (Q1, Q2, Q3).

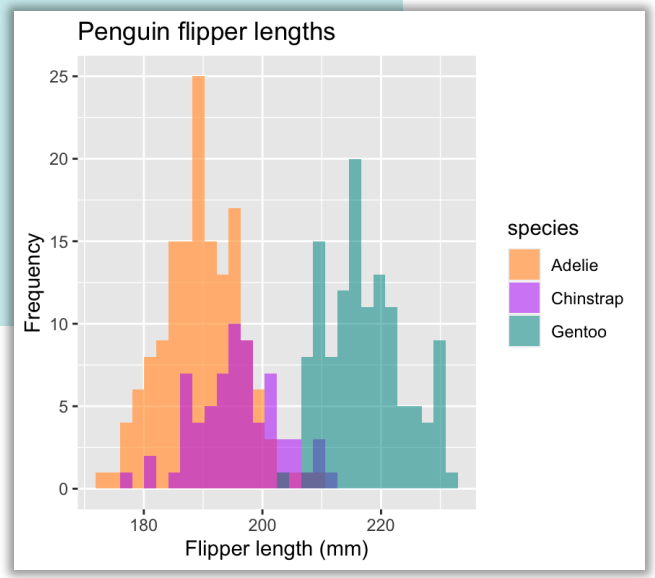
E. Calculate the flipper length 40th percentile for the "Adelie" penguins.

SOLUTION

Exercise 3 – Histogram, Boxplot and quartiles

A. Using the **`geom_histogram()`** function of the ggplot2 package plot the flipper length distribution coloring each species with a different color.

```
flipper_hist <- ggplot(data = penguins, aes(x = flipper_length_mm)) +
    geom_histogram(aes(fill = species),
        alpha = 0.5, position = "identity") +
    scale_fill_manual(values = c("darkorange",
        "purple", "cyan4")) +
    labs(x = "Flipper length (mm)", y = "Frequency",
        title = "Penguin flipper lengths")

flipper_hist
```



SOLUTION

Exercise 3 – Histogram, Boxplot and quartiles

B. About the flipper length, for each species of penguins compute the: sample mean, sample median, sample variance, sample standard deviation.

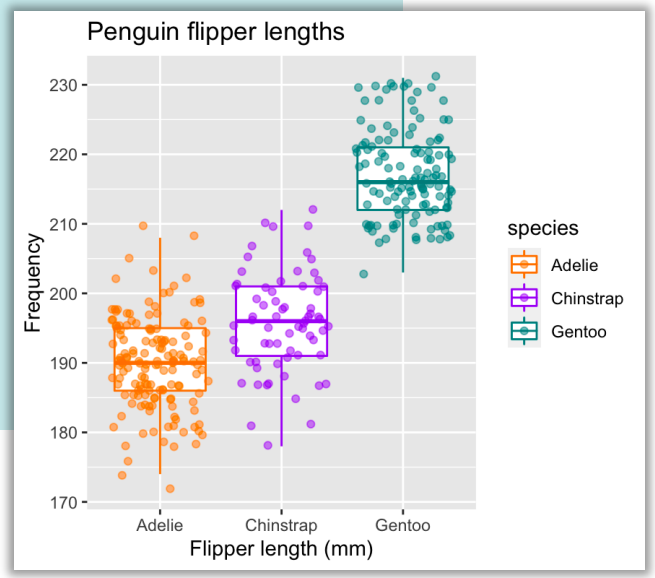
```
# Extract the value for the flipper length
# Then choose the elements corresponding to the specific species
fl_gentoo <- penguins$flipper_length_mm[penguins$species == "Gentoo"]

# Finally compute the statistics
mean(fl_gentoo, na.rm = TRUE)
median(fl_gentoo, na.rm = TRUE)
sd(fl_gentoo, na.rm = TRUE)
var(fl_gentoo, na.rm = TRUE)

# Repeat the process for each species, or create a function for not
# repeating the same code 3 times
```

SOLUTION

Exercise 3 – Histogram, Boxplot and quartiles

c. Using the **`geom_boxplot()`** function of the ggplot2 package plot the boxplot for the flipper length variable coloring each species with a different color.

```
flipper_box <- ggplot(data = penguins,
                      aes(x = species, y = flipper_length_mm,
                      color = species)) +
    geom_boxplot(outlier.alpha = 0) +
    geom_jitter(alpha = 0.5) +
    scale_color_manual(values = c("darkorange", "purple",
                                  "cyan4")) +
    labs(x = "Flipper length (mm)", y = "Frequency",
         title = "Penguin flipper lengths")

flipper_box
```



SOLUTION

Exercise 3 – Histogram, Boxplot and quartiles

D. Compute the flipper length quartiles for the "Gentoo" penguins (Q1, Q2, Q3).

E. Calculate the flipper length 40th percentile for the "Adelie" penguins.

```
quantile(penguins$flipper_length_mm[penguins$species == "Gentoo"],
         probs = c(0.25, 0.5, 0.75),
         na.rm = TRUE)

quantile(penguins$flipper_length_mm[penguins$species == "Adelie"],
         probs = 0.4,
         na.rm = TRUE)
```

Long and wide data format

Long format and wide format are two ways to structure data in a dataset.

In long format, each observation is represented by a single row, and each variable has its own column. This format is useful for datasets where each observation has multiple measurements or values for each variable.

In wide format, each variable has its own column, and each observation is represented by a single row. This format is useful for datasets where each observation has only one measurement or value for each variable.

Exercise 4 – Multiple boxplots from scratch

A. Generate random data with some structure, and create one data set for each day of the week (hint: use the **`for()`** cycle, data should have 7 columns). At the end you should obtain a matrix with N rows (N = number of random number to generate each time) and 7 columns (one for each day of the week).

B. Go from a wide to a long data format. You should create a data.frame object with exactly two columns. One contains the values created in A., the other contains the corresponding day of the week.

C. Plot the seven boxplots (one for each day of the week) in one graph, horizontally oriented (hint: **`coord_flip()`** function translates the axes, the "limits" argument of **`scale_x_discrete()`** allows you to reorder the axis labels).

SOLUTION

Exercise 4 – Multiple boxplots from scratch

A. Generate random data with some structure, and create one data set for each day of the week. At the end you should obtain a matrix with N rows (N = number of random number to generate each time) and 7 columns (one for each day of the week).

```
# Number of random numbers to generate
N = 100
# Number of days in a week
days = 7
# Create an empty matrix
data = matrix(0, ncol = days, nrow = N);
# Fill the matrix
for (day in seq_len(days)) {
    data[, day] <- 7 + 6*sin(day/5) + runif(N)
}
```

SOLUTION

Exercise 4 – Multiple boxplots from scratch

B. Go from a wide to a long data format. You should create a data.frame object with exactly two columns. One contains the values created in A., the other contains the corresponding day of the week.

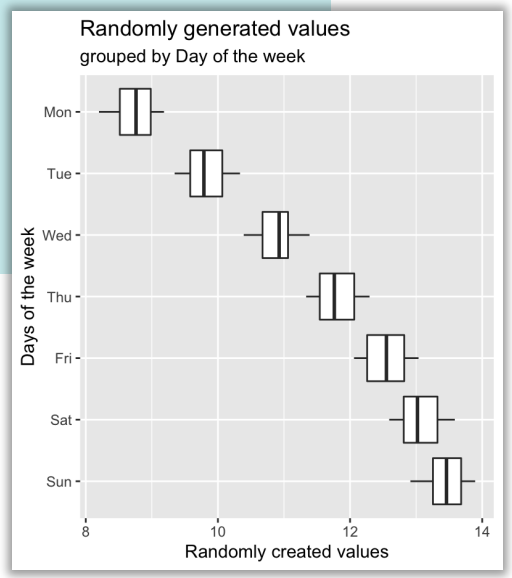
```
# Put all the matrix values in a vector
data_as_vector <- c(data)

# Then we have to add information about the day of the week
day_of_the_week <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
data_long <- data.frame("Value" = data_as_vector,
                        "Day" = rep(day_of_the_week, each = N))
```

SOLUTION

Exercise 4 – Multiple boxplots from scratch

c. Plot the seven boxplots (one for each day of the week) in one graph, horizontally oriented

```
ggplot(data = data_long, aes(x = Day, y = Value)) +
    geom_boxplot() +
    coord_flip() +
    scale_x_discrete(limits = rev(day_of_the_week)) +
    labs(y = "Randomly created values",
         x = "Days of the week",
         title = "Randomly generated values",
         subtitle = "grouped by Day of the week")
```



Exercise 5 – Exploratory analysis of data

Penguins dataset does not contain their weights and flipper lengths only. Many other variables are available. Let's explore it a little more:

A. How many islands are there? And how many penguins are present in each isle? Are the 3 species of penguins living together? (hint: use the **`table()`** function).

B. Try to use the **`geom_bar()`** or **`geom_col()`** functions to graphically represent the population of each island, colored by species (hint: islands in the x-axis, number of penguins in the y-axis).

C. Use a scatter plot to represent flipper length vs. body mass. Color the point according to the "sex" variable. Try to use facets to see if there are differences across species (hint: use **`facet_grid(~ species)`** function to add facets for species).

D. The numeric variables shows some interesting relationships. Are they correlated? Use the **`cor()`** and the **`corrplot()`** functions to study correlations between numeric variables (hint: try to google `corrplot()` to see which package you have to install to use it).

E. Choose a pair of numeric variables, compute the correlation between them without using the `cor()` function (hint: remember the formula).

F. Plot the scatter plot for bill length vs. bill depth. Color the points by species. Use the function **`geom_smooth(formula = "y ~ x")`** to add a line to represent the linear relationship between the two variables. Then, again, use **`geom_smooth(formula = "y ~ x")`** colored by species. What are you noticing?

SOLUTION

Exercise 5 – Exploratory analysis of data

A. How many islands are there? And how many penguins are present in each isle? Are the 3 species of penguins living together?

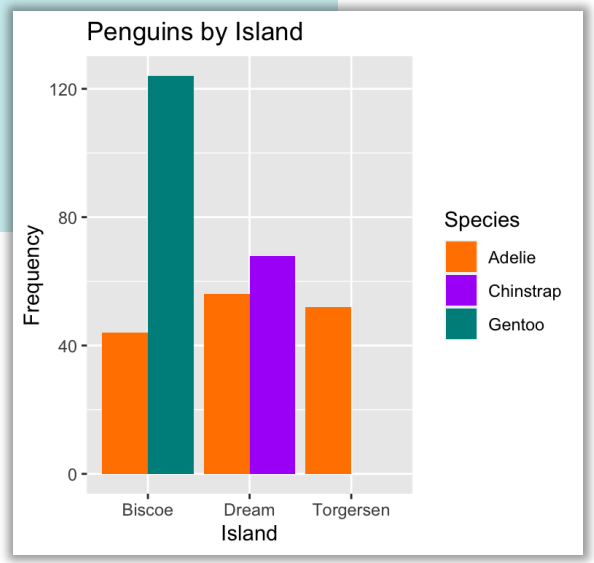
```
library(penguins)
data("penguins")
head(penguins)

table(penguins$species, penguins$island)

# This returns the table below:
#           Biscoe Dream Torgersen
# Adelie        44    56        52
# Chinstrap      0    68         0
# Gentoo       124     0         0
```

SOLUTION

Exercise 5 – Exploratory analysis of data

B. Try to use the **`geom_bar()`** or **`geom_col()`** functions to graphically represent the population of each island, colored by species.

```
library(ggplot2)
ggplot(data = penguins, aes(x = island)) +
    geom_bar(aes(fill = species),
             position = position_dodge(preserve = "single")) +
    labs(x = "Island", y = "Frequency", fill = "Species",
         title = "Penguins by Island") +
    scale_fill_manual(values = c("darkorange", "purple",
                      "cyan4"))
```

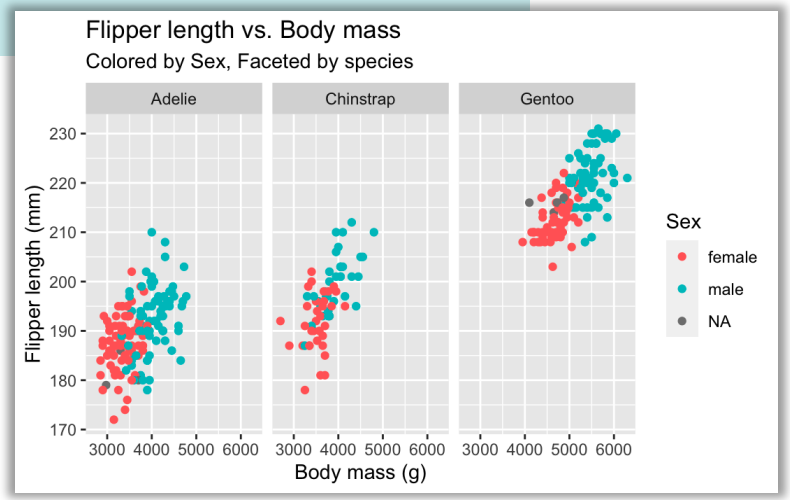


SOLUTION

Exercise 5 – Exploratory analysis of data

c. Use a scatter plot to represent flipper length vs. body mass. Color the point according to the "sex" variable. Try to use facets to see if there are differences across species.

```
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm)) +
    geom_point(aes(color = sex)) +
    facet_grid(~ species) +
    labs(x = "Body mass (g)", y = "Flipper length (mm)", color = "Sex",
         subtitle = "Colored by Sex, Faceted by species",
         title = "Flipper length vs. Body mass")
```



SOLUTION

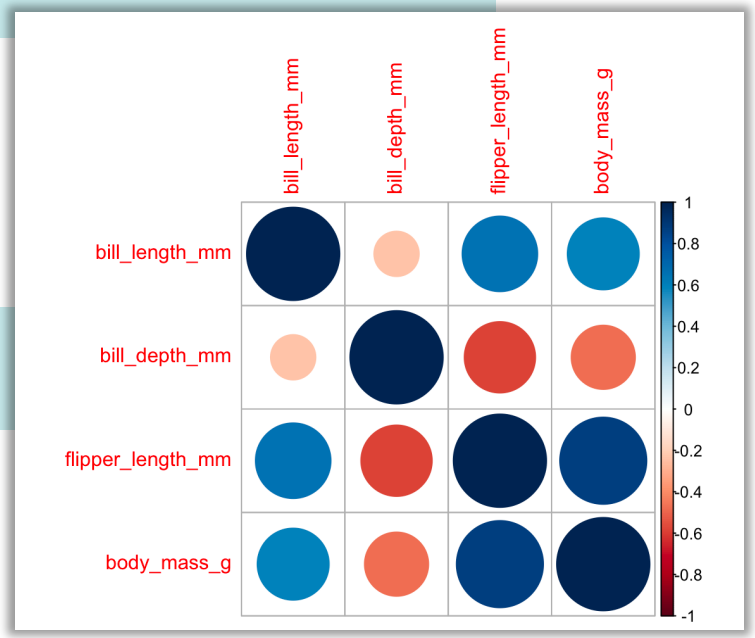
Exercise 5 – Exploratory analysis of data

D. The numeric variables shows some interesting relationships. Are they correlated? Use the **`cor()`** and the **`corrplot()`** functions to study correlations between numeric variables.

```
penguins_numeric <- penguins[, 3:6]
cors <- cor(penguins_numeric, use = "complete.obs")
cors
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.0000000	-0.2350529	0.6561813	0.5951098
bill_depth_mm	-0.2350529	1.0000000	-0.5838512	-0.4719156
flipper_length_mm	0.6561813	-0.5838512	1.0000000	0.8712018
body_mass_g	0.5951098	-0.4719156	0.8712018	1.0000000

```
# install.packages("corrplot")
corrplot::corrplot(corr = cors)
```



SOLUTION

Exercise 5 – Exploratory analysis of data

E. Choose a pair of numeric variables, compute the correlation between them without using the `cor()` function.

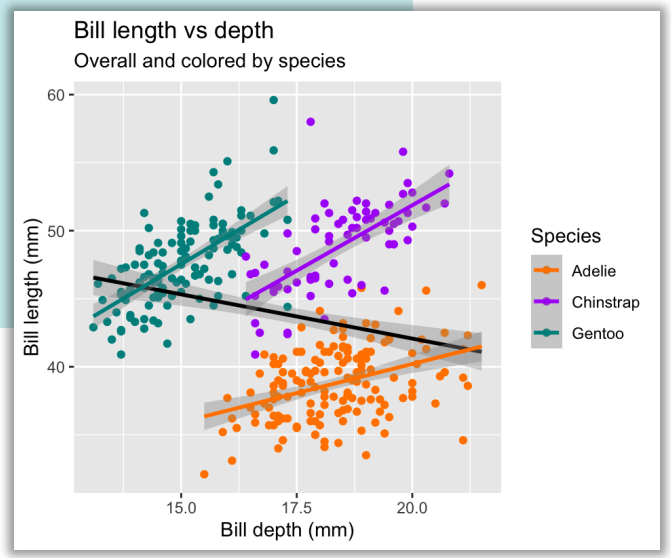
```
var1 <- penguins_numeric$bill_length_mm
var2 <- penguins_numeric$bill_depth_mm

# Using cov() and sd()
r_short <- cov(var1, var2, use = "complete.obs") /
    (sd(var1, na.rm = TRUE) * sd(var2, na.rm = TRUE))

# Using elementary functions
v1 <- na.omit(var1)
v2 <- na.omit(var2)
covariance <- sum((v1 - sum(v1)/length(v1)) * (v2 - sum(v2)/length(v2))) /
    (length(v1) - 1)
sd1 <- sqrt(sum((v1 - sum(v1)/length(v1)) ^ 2) / (length(v1) - 1))
sd2 <- sqrt(sum((v2 - sum(v2)/length(v2)) ^ 2) / (length(v2) - 1))
r_long <- covariance / (sd1 * sd2)
```

SOLUTION

Exercise 5 – Exploratory analysis of data

F. Plot the scatter plot for bill length vs. bill depth. Color the points by species. Use the function **`geom_smooth(formula = "y ~ x")`** to add a line to represent the linear relationship between the two variables. Then, again, use **`geom_smooth(formula = "y ~ x")`** colored by species. What are you noticing?

```
ggplot(data = penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +
    geom_point(aes(color = species)) +
    geom_smooth(method = "lm", color = "black") +
    geom_smooth(aes(color = species), method = "lm") +
    scale_color_manual(values = c("darkorange", "purple",
                       "cyan4")) +
    labs(x = "Bill depth (mm)", y = "Bill length (mm)",
         color = "Species", title = "Bill length vs depth",
         subtitle = "Overall and colored by species")
```



Simpson's Paradox: the Simpson's Paradox is a statistical phenomenon in which a trend that is observed in subgroups of data can disappear or even reverse when those subgroups are combined into a larger group due to differences in subgroup sizes and variable relationships. It can lead to incorrect conclusions and interpretations of data if not carefully analyzed.