Turning Numbers into Love Letters

Love Data Week 2026

Statistical Consulting and Survey Research Unit
University of Waterloo

2026-02-10

Agenda

In this workshop, we will cover:

1. What is Exploratory Data Analysis (EDA)
2. Type of variables (continuous, discrete, nominal, ordinal)
3. Manipulation of categorical data
4. Exploring categorical and numerical data: tables, numerical summaries and various charts

1. Planning and conducting a study

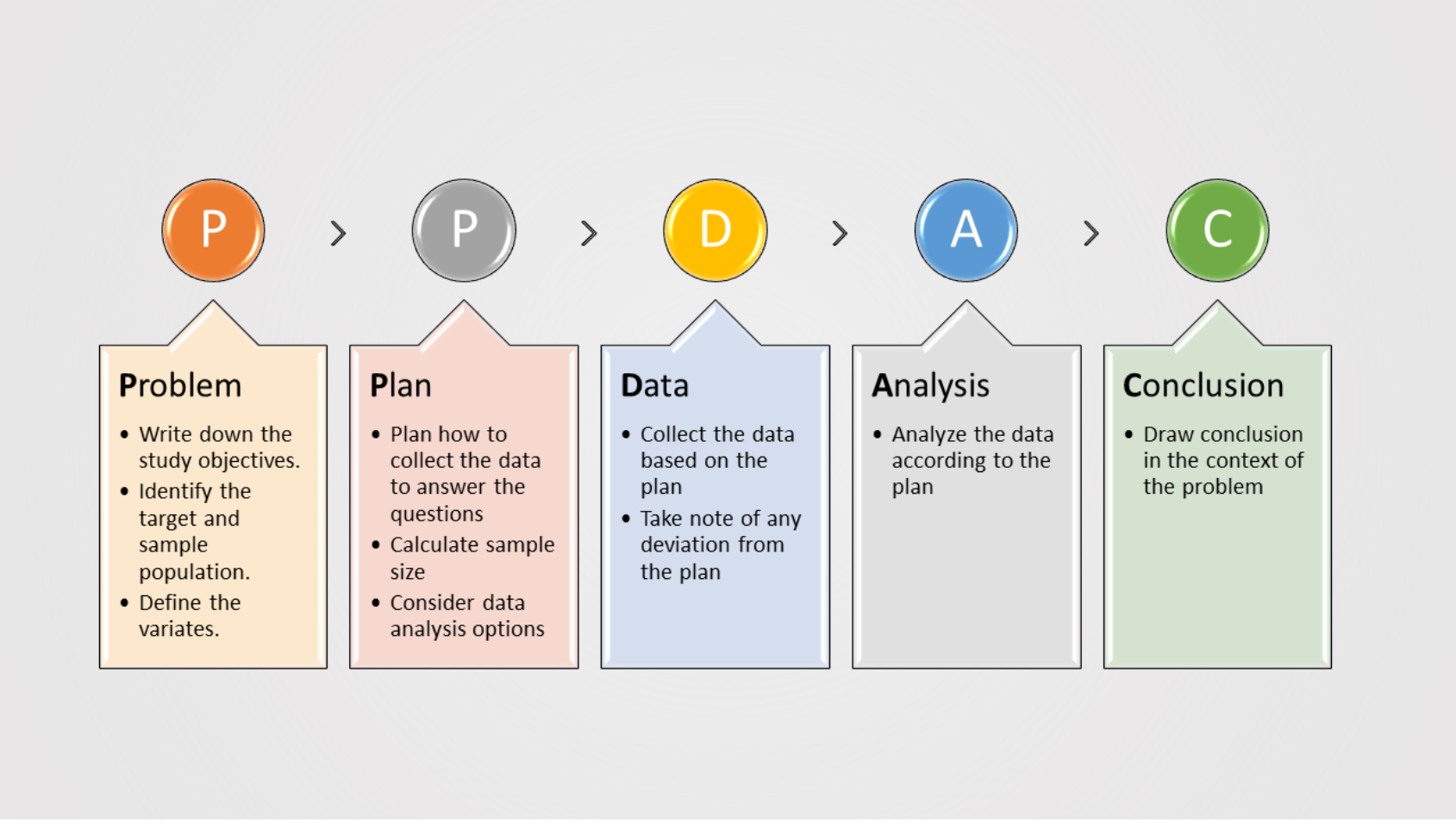


Figure 1: Consider PPDAC when planning and conducting a study

1.1 After collecting the data

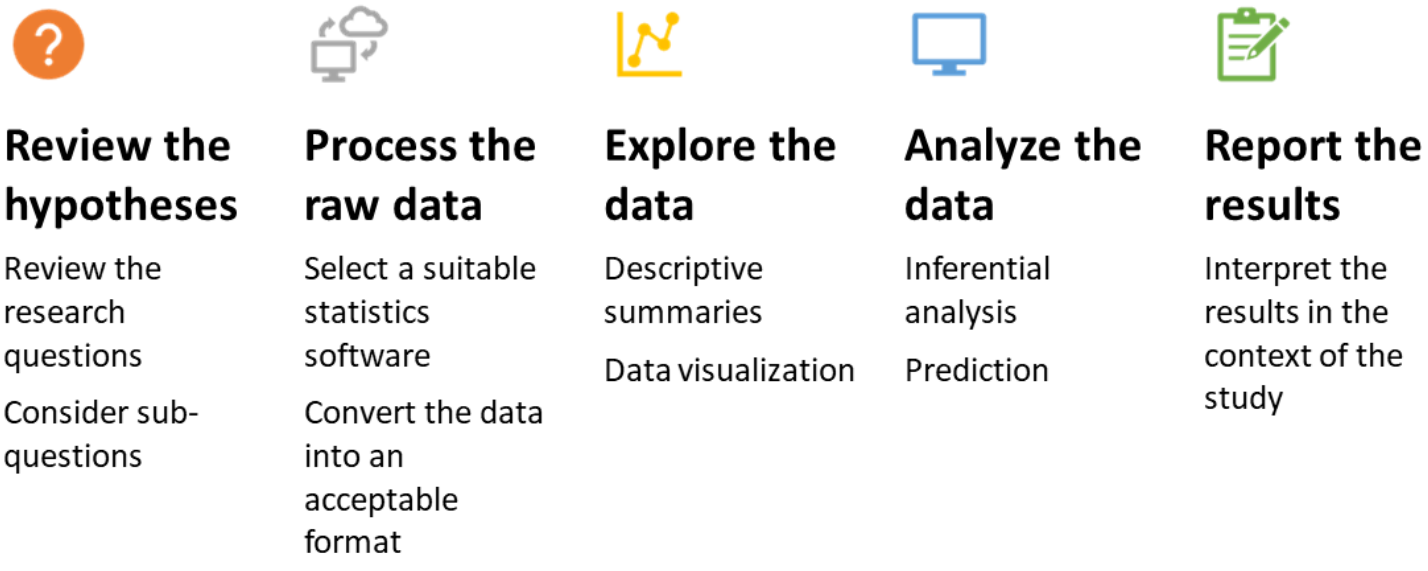


Figure 2: Recommended process

1.2 Exploratory data analysis

- Exploratory data analysis (EDA) was developed by John Tukey in the 1970s. Nowadays, the EDA techniques are used to analyze and investigate data and summarize their main characteristics numerically and graphically.
- The main purpose of EDA is to:
  - uncover the data structure,
  - discover patterns,
  - spot anomalies,
  - check assumptions, and
  - find interesting relationships among the variables of interest.

More sophisticated data analysis is usually performed after EDA is completed.

2. The data set and R libraries

- Throughout this workshop, we will be looking at the Pokemon data set.
- This data set was obtained from Kaggle. A little background about Pokemon:

The original Pokemon is a role-playing game based around building a team of monsters to battle other monsters to become the best team. Pokemon are divided into types with different strengths.

The data set can be downloaded from our website. Please set your working directory to where you saved the data set and load the data set into your R environment.

Import data and libraries

- Load the dataset.

```
pokemon <- read.csv("pokemon.csv")
```

- Load the libraries.

```
library(ggplot2) # visualize dataset with fancy plots
library(dplyr) # manipulate data frames
library(agrmt) # for likert scale data
library(likert) # also for likert scale data
```

3. Types of variables

It is important to understand what each variable is and its types. We also need to ensure that the types are correctly recognized by the software before further analysis.

- *Continuous (numerical)* variable: takes any value over a continuous range. Usually, the variable will have a measurement unit.
  - In the `pokemon` data set, examples of continuous variables include:
    - `height_m`: The height of the Pokemon in metres.
    - `weight_kg`: The weight of the Pokemon in kilograms.
- *Discrete (categorical/qualitative)* variable: takes values over a finite set of values (or levels).
  - *Binary variable*: categorical variables with only 2 levels.
  - In the `pokemon` data set, the variable `is_legendary` which denotes whether the Pokemon is legendary, only take 2 values: 0 and 1. We call `is_legendary` a binary variable.

3.1 Nominal variables

A categorical variable with no specific order is also called a nominal variable. Examples include:

- A university student's major.
- A person's blood type.
- The type of drinks at Starbucks.
- A person's eye colour.

3.2 Ordinal variables

A categorical variable with natural ordering is also called an ordinal variable. Examples include

- A person's eye colour.
- A person's level of agreement about a statement.

Notice that the example "a person's eye colour" shows up as nominal and ordinal variable. Why?

3.3 Discussion: Is the variable type fixed?

- We cannot determine the variable type by its name. To accurately categorize a variable, we need to consider how it is recorded.
- A common example is age.
  - In some studies, age is recorded an exact value, e.g. 25, 35.5, 80, etc. This age variable is considered a numeric variable.
  - Other studies may require respondents to select the category in which their age falls in, e.g. $<$20, 21-25, 80$+$, etc. This age variable is considered a categorical variable.

4. Data manipulation

- Before performing any kind of analysis, it is important to take a look at the data set in the environment of the software.
- We can use the `View()` function to quickly view the data set in spreadsheet format.

```
View(pokemon)
```

- To understand how the software R understands the data, we will need to use the `str()` function.

```
str(pokemon) # output the name, type and first few entries
```

4.1 Factorize categorical variables

- Recall that the variable `is_legendary` is a binary variable. However, it is recorded as an integer.
- We will need to correct this, using the `factor()` function.

```
pokemon$is_legendary_f <- factor(pokemon$is_legendary,
                                 levels = c(0,1),
                                 labels = c("No", "Yes"))
```

- Notice that we created a new variable `is_legendary_f` for the categorical variable `is_legendary`. This allows us to review and recover the original values easily as needed.

More examples

- The variable `generation` denotes the numbered generation which the Pokemon was first introduced. Although it is recorded as integer, it is not an integer in theory. We should also "inform" the statistics software that this is a categorical variable.

```
pokemon$generation_f <- factor(pokemon$generation)
```

- The same can be say for the `type` variable.

```
pokemon$type1_f <- factor(pokemon$type1)
```

4.2 View levels of a categorical variable

- Another useful function is `levels()` which allows us to investigate the categories within a variable.
- This is particularly useful when there are many categories.
- For example, the variable `type1` (which we stored as `type1_f`) has 18 levels:

```
levels(pokemon$type1_f)
```

```
##  [1] "bug"      "dark"     "dragon"   "electric" "fairy"
##  [6] "fighting" "fire"     "flying"   "ghost"    "grass"
## [11] "ground"   "ice"      "normal"   "poison"   "psychi
## [16] "rock"     "steel"    "water"
```

5. Exploring categorical variables

- A common first step to understanding categorical variable is to summarize the variable using a table.

```
table(pokemon$type1_f)
```

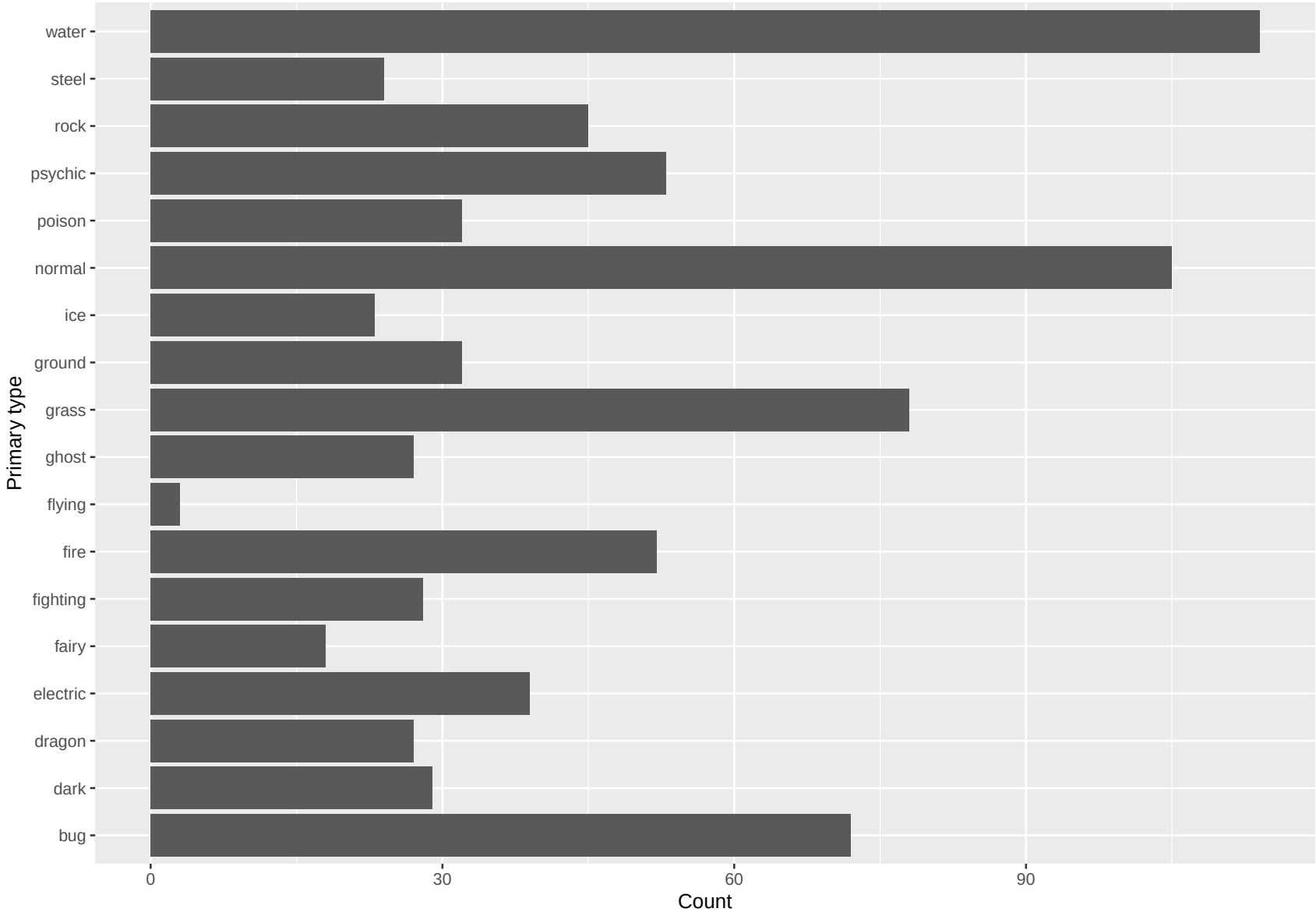
```
##
##      bug     dark   dragon electric     fairy fighting
##       72       29       27       39       18       28
##     fire   flying    ghost    grass   ground      ice
##       52        3       27       78       32       23
##   normal   poison  psychic     rock    steel    water
##      105       32       53       45       24      114
```

When there are many categories, it is more appealing to look at the table produced using `dplyr` library, e.g., the `count()` function.

5.1 Bar charts

- Notice that the tables are not appealing when there are many categories. This is where graphical tools shine.
- A common way to visualize a categorical variable is the bar chart.

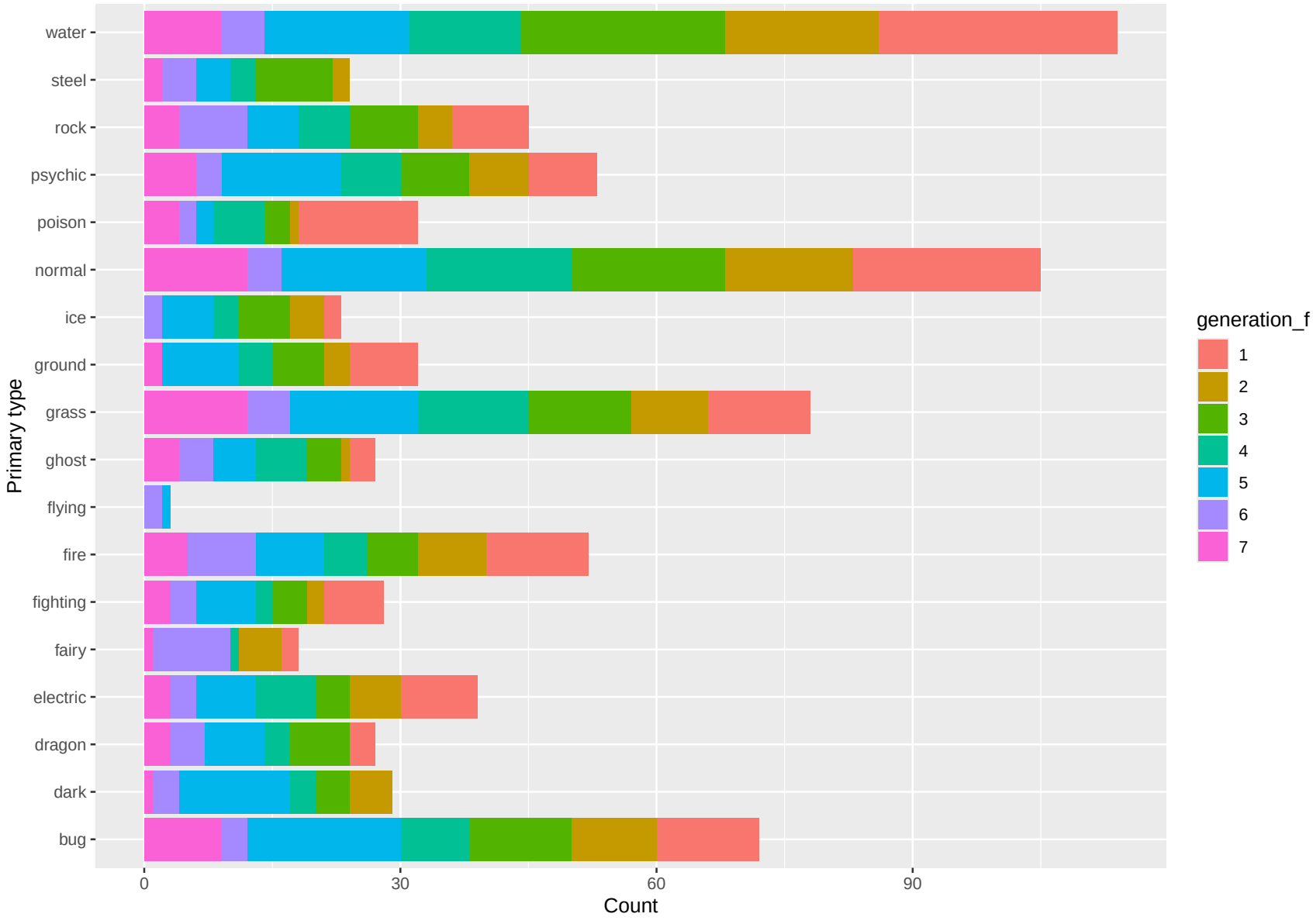
```
ggplot(pokemon, aes(x=type1_f)) +
  geom_bar()+ # Type of chart
  xlab("Primary type") + ylab("Count")+ # Label the xy-axes
  coord_flip() # Flip the xy-axes
```


water
steel
rock
psychic
poison
normal
ice
ground
grass
ghost
flying
fire
fighting
fairy
electric
dragon
dark
bug
Primary type
0
30
60
90
Count

5.1.1 Stacked bar charts

- To better visualize the contingency table and relationship between two categorical variables, we can use a stacked bar chart.

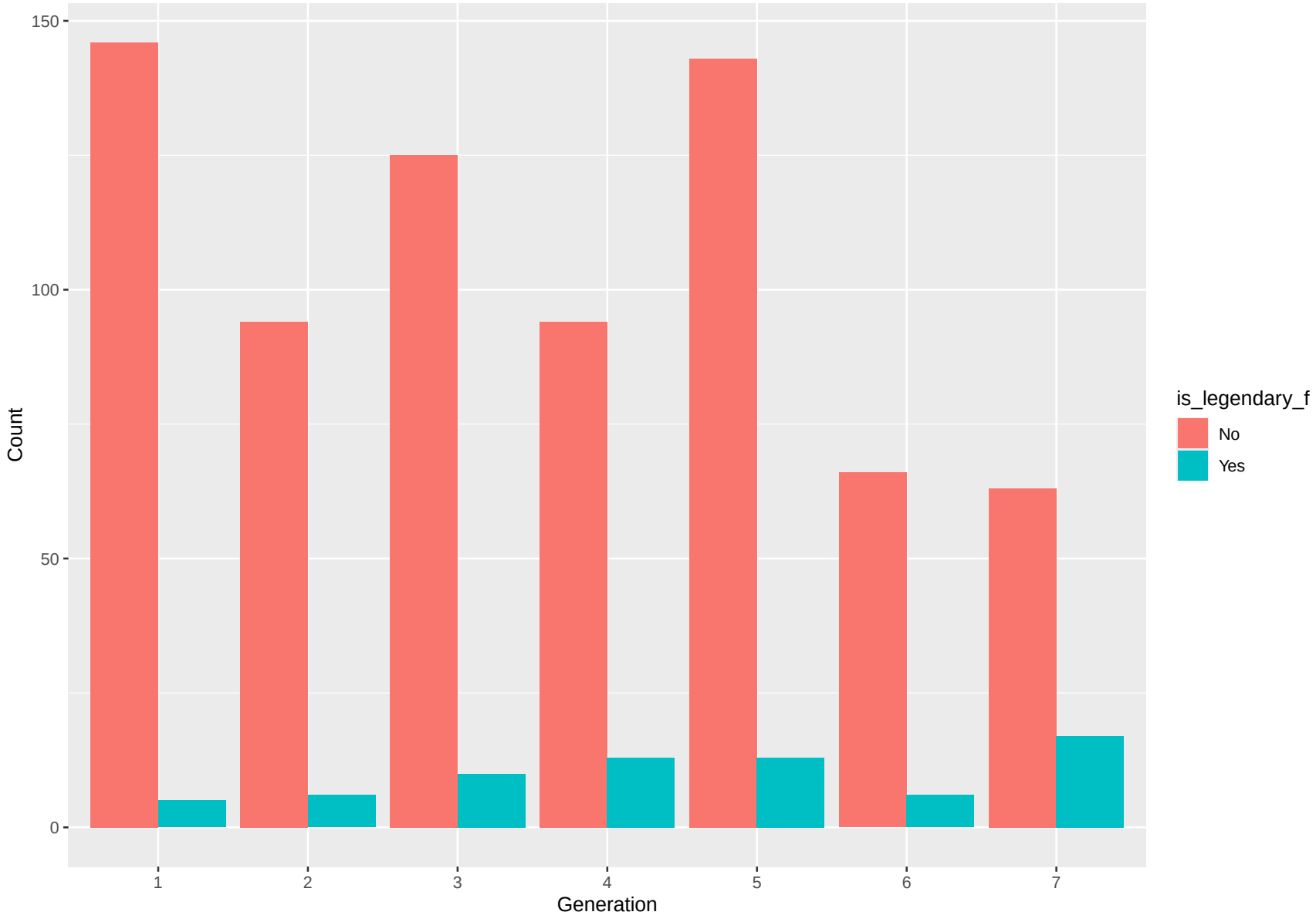
```
ggplot(pokemon, aes(x=type1_f, fill=generation_f)) +
  geom_bar()+ # Type of chart
  xlab("Primary type") + ylab("Count")+ # Label the xy-axes
  coord_flip() # Flip the xy-axes
```


water
steel
rock
psychic
poison
normal
ice
ground
grass
ghost
flying
fire
fighting
fairy
electric
dragon
dark
bug
Primary type
0
30
60
90
Count
generation_f
1
2
3
4
5
6
7

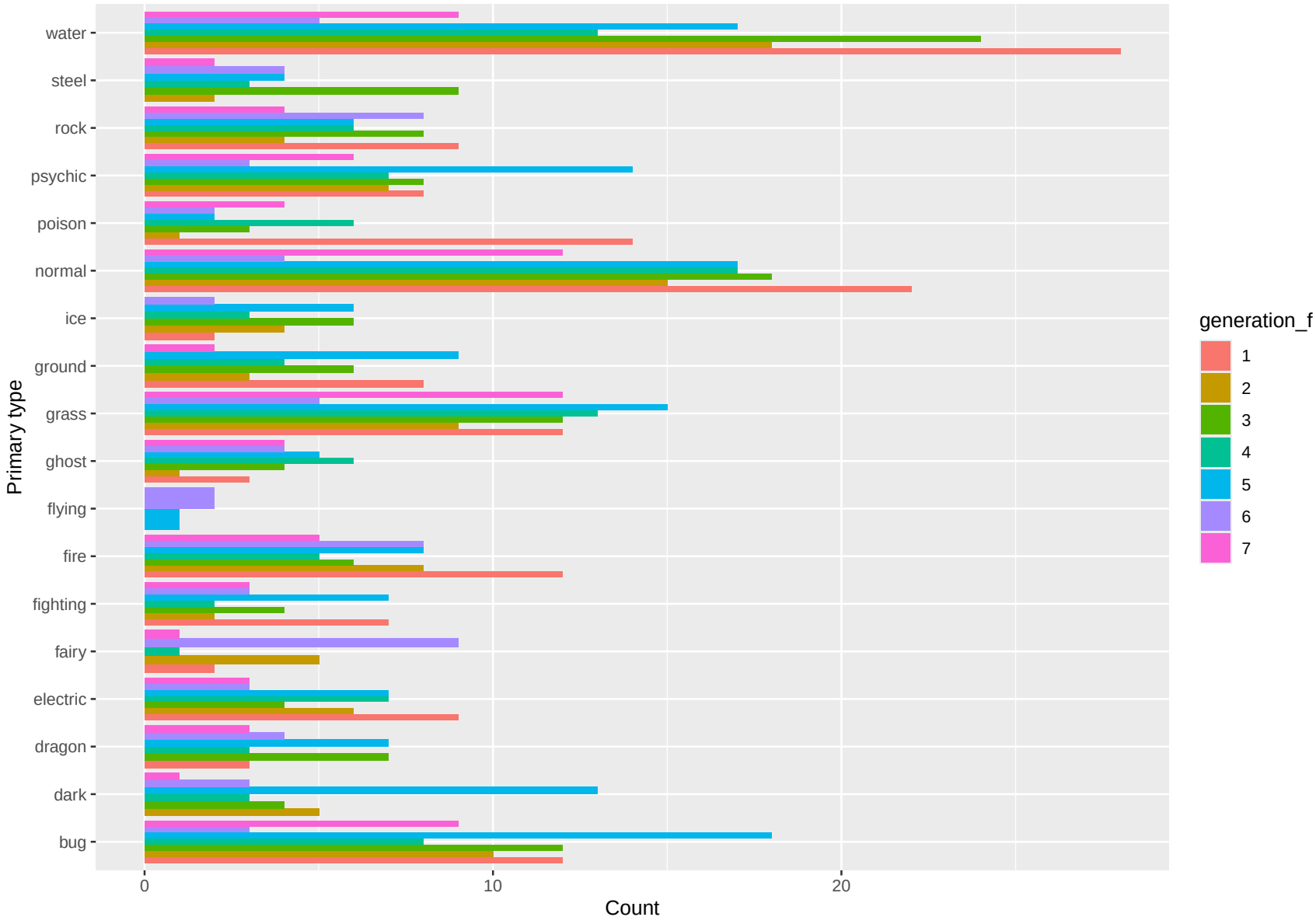
5.1.2 Side-by-side bar charts

- There can often be a few options to present the same data.
- The side-by-side bar charts are also useful for representing a contingency table graphically.

```
ggplot(pokemon, aes(x=generation_f, fill=is_legendary_f)) +
  geom_bar(position="dodge")+ # Type of chart
  xlab("Generation") + ylab("Count") # Label the xy-axes
```


150
100
50
0
Count
1
2
3
4
5
6
7
Generation
is_legendary_f
No
Yes

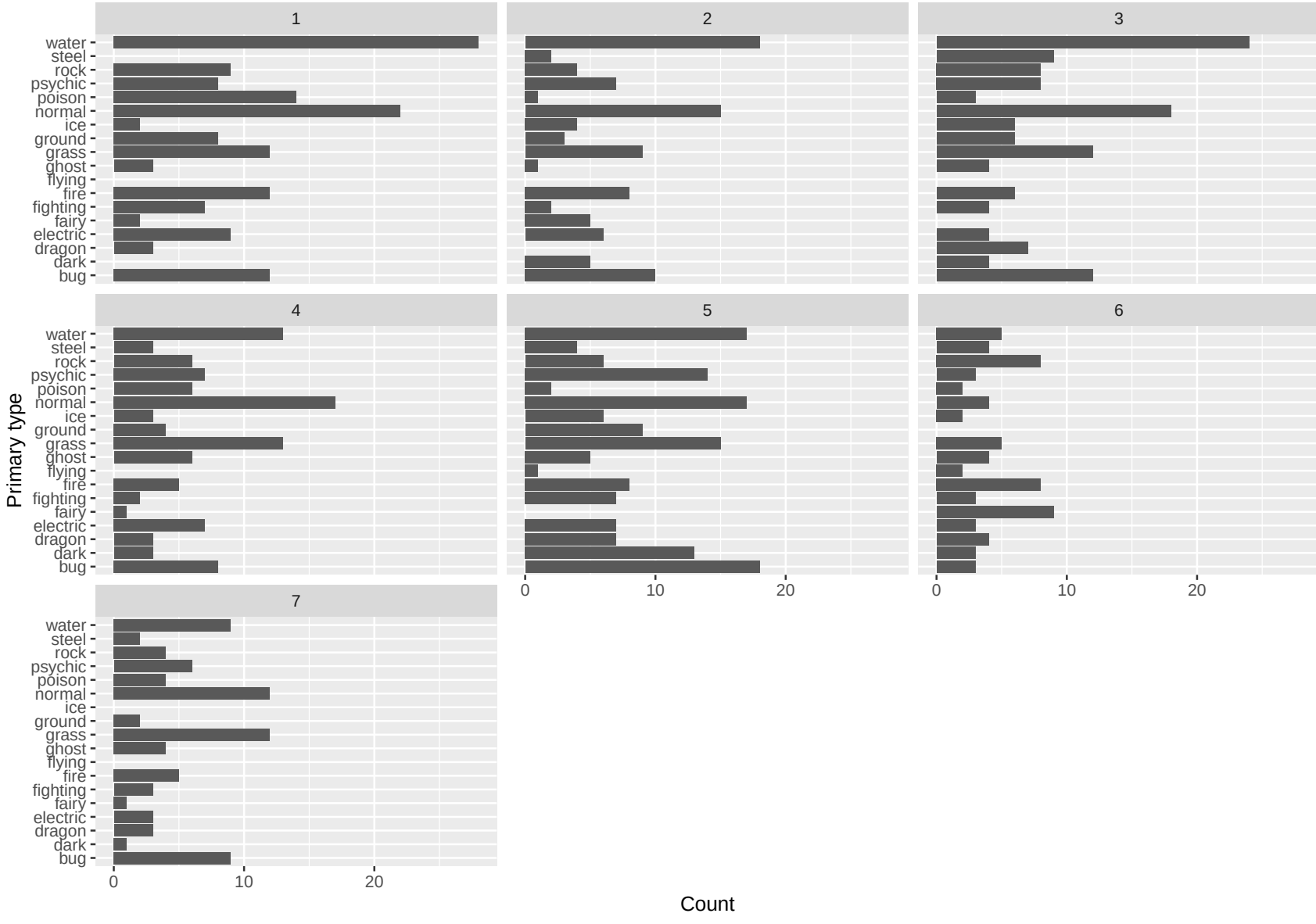
When side-by-side bar chart does not work



5.2 Faceting

- When side-by-side bar chart does not convey the message very well, we can consider the option of using the facet grids.
- The facet grids form a matrix of panels defined by row and column faceting variables.
- It is most useful when you have two discrete variables.

```
ggplot(pokemon, aes(x=type1_f)) +
  geom_bar(position="dodge") +
  facet_wrap(~generation_f) + # Facet
  xlab("Primary type") + # Label the x-axes
  ylab("Count")+ # Label the y-axes
  coord_flip() # Flip the xy-axes
```


1
2
3
4
5
6
7
water
steel
rock
psychic
poison
normal
ice
ground
grass
ghost
flying
fire
fighting
fairy
electric
dragon
dark
bug
0
10
20
Primary type
Count

Discussion: Stacked vs side-by-side vs facet

- Each type of bar charts have their advantages and disadvantages. How do we decide which to use?
- Recall that the saying that a picture is worth a thousand words. If a graph needs to be accompanied by long caption, then it defeats the purpose of a graph.

5.3 Proportion table

- Proportions can suggest more information when investigating relationships with other variables than counts, especially when analyzing the difference between treatment and control groups.
- Use `proportions()` or `prop.table()`

```
tab_type1 <- table(pokemon$type1_f)
round(prop.table(tab_type1),2)
```

```
##
##      bug     dark   dragon electric    fairy fighting
##     0.09     0.04     0.03     0.05     0.02     0.03
##     fire   flying    ghost    grass   ground      ice
##     0.06     0.00     0.03     0.10     0.04     0.03
##   normal   poison  psychic     rock    steel    water
##     0.13     0.04     0.07     0.06     0.03     0.14
```

- Notice that the proportions sum up to 1.

5.3.1 Conditional proportions

- If we are curious about systematic associations between variables, we will consider the conditional proportions, which can be obtained by specifying the second argument in `prop.table()`.
- Adding 1 as the second argument will specify to condition on the rows. The proportions in every row will sum to 1.
- Adding 2 as the second argument will specify to condition on the columns. The proportions in every column will sum to 1.

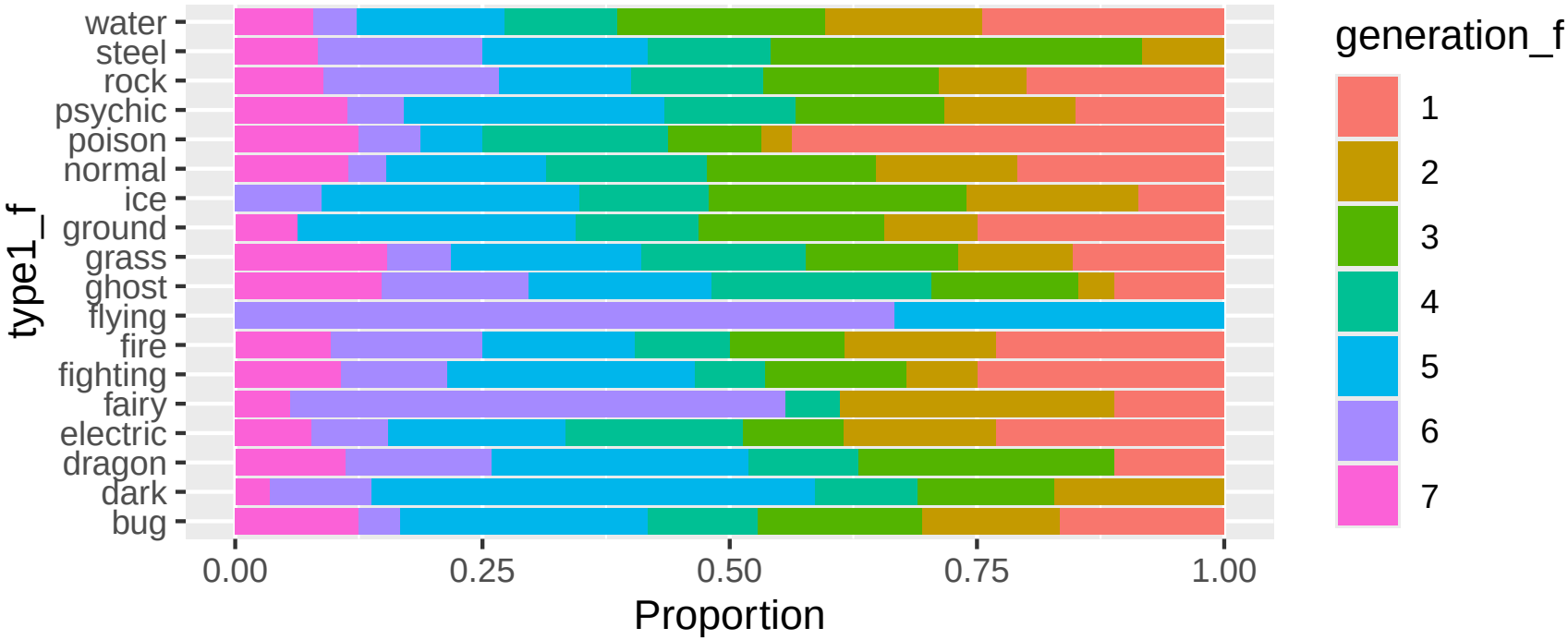
5.3.1 Conditional proportions: Example

```
tab_type1_gen <- table(pokemon$type1_f, pokemon$generation_f)
round(prop.table(tab_type1_gen,1),2)
```

```
##
##                1    2    3    4    5    6    7
##   bug       0.17 0.14 0.17 0.11 0.25 0.04 0.12
##   dark      0.00 0.17 0.14 0.10 0.45 0.10 0.03
##   dragon    0.11 0.00 0.26 0.11 0.26 0.15 0.11
##   electric  0.23 0.15 0.10 0.18 0.18 0.08 0.08
##   fairy     0.11 0.28 0.00 0.06 0.00 0.50 0.06
##   fighting  0.25 0.07 0.14 0.07 0.25 0.11 0.11
##   fire      0.23 0.15 0.12 0.10 0.15 0.15 0.10
##   flying    0.00 0.00 0.00 0.00 0.33 0.67 0.00
##   ghost     0.11 0.04 0.15 0.22 0.19 0.15 0.15
##   grass     0.15 0.12 0.15 0.17 0.19 0.06 0.15
##   ground    0.25 0.09 0.19 0.12 0.28 0.00 0.06
##   ice       0.09 0.17 0.26 0.13 0.26 0.09 0.00
##   normal    0.21 0.14 0.17 0.16 0.16 0.04 0.11
##   poison    0.44 0.03 0.09 0.19 0.06 0.06 0.12
##   psychic   0.15 0.13 0.15 0.13 0.26 0.06 0.11
##   rock      0.20 0.09 0.18 0.13 0.13 0.18 0.09
##   steel     0.00 0.08 0.38 0.12 0.17 0.17 0.08
##   water     0.25 0.16 0.21 0.11 0.15 0.04 0.08
```

The table shows us that among the Pokemons that are of bug type, 17% are from Generation 1, 14% are from Generation 2, etc. Visually,

```
ggplot(pokemon, aes(x = type1_f, fill=generation_f))+
  geom_bar(position="fill") +
  ylab("Proportion") +
  coord_flip()
```



5.4 Ordinal

- Ordinal variable is a type of categorical variable with natural ordering.
- We can turn numeric variables into ordinal variable too. For example, age recorded in groupings of 10-19, 20 - 29, 80+, etc.

Creating ordinal variables

- We often turned continuous variables into categorical variable.
- The Pokemons have different level of happiness. We will create a `happiness_level` with three categories:
  - `low`: `base_happiness` $<$ 70
  - `normal`: `base_happiness` $=$ 70
  - `high`: `base_happiness` $>$ 70

```
pokemon <- pokemon %>%
  mutate(happiness_level=
           cut(base_happiness,
               breaks=c(-Inf, 69,71, Inf),
               labels=c("low","normal","high")))
head(pokemon$happiness_level)
```

```
## [1] normal normal normal normal normal normal
## Levels: low normal high
```

- As we can see, there is no "$<$" in the R output of `Levels`. The variable is still a nominal variable.

5.4.1 Measures of central tendency

- To indicate that this variable has a natural ordering, we use the `factor()` function:

```
pokemon$happiness_level <-
  factor(pokemon$happiness_level, ordered = TRUE,
         levels = c("low", "normal", "high"))
head(pokemon$happiness_level)
```

```
## [1] normal normal normal normal normal normal
## Levels: low < normal < high
```

- We can summarize ordinal variable numerically the same way as any categorical variable.
- However, since ordinal variables are often skewed, we recommend using the median or the mode.

5.4.2 Measure of spread: consensus

- A measure of spread for ordinal data is called *consensus* (Tastle & Wierman, 2007).
- The consensus ranges from 0 to 1.
- 0 indicates a lack of consensus, 1 indicates all the respondents are in agreement.
- Use the `consensus()` function from the library `agrmt`.

```
# library(agrmt)
tab_happiness<- table(pokemon$happiness_level)
consensus(tab_happiness)
```

```
## [1] 0.786256
```

- The consensus is around 0.79, which means that the Pokemons are pretty agreeable on their levels of happiness.
- This will not be surprising after we take a look at the distribution of `happiness_level` visually.

5.4.3 Visualizing likert scale data

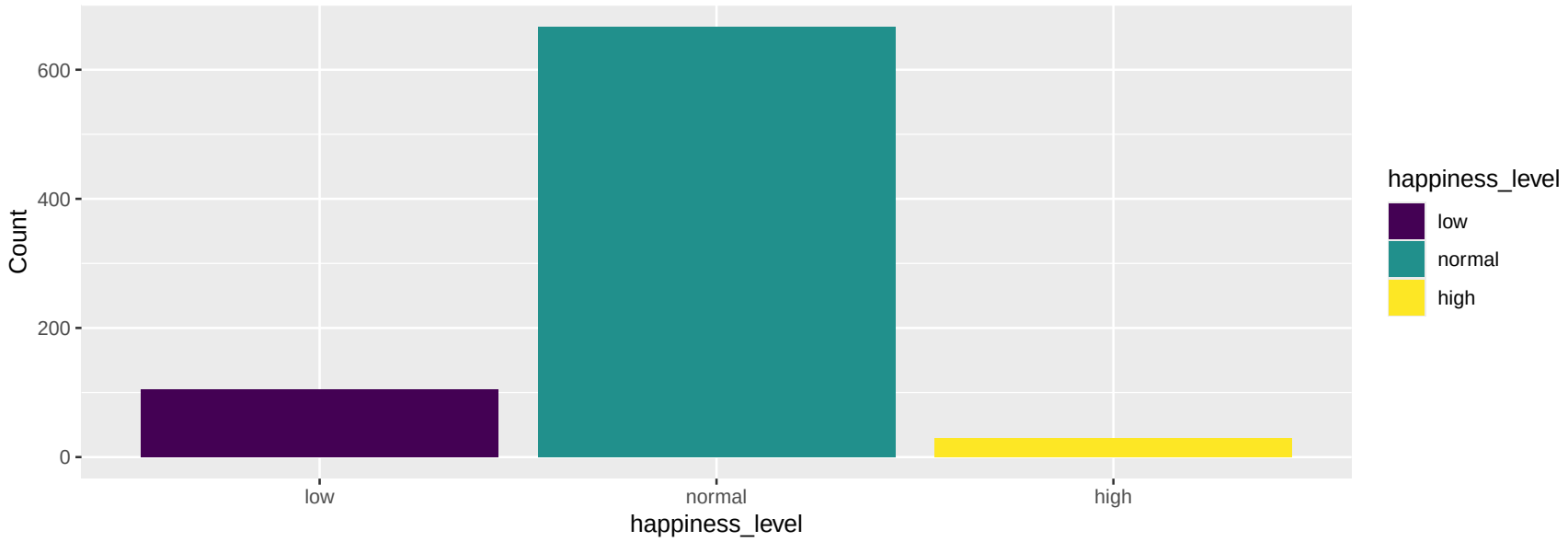
- There are two popular ways to visualize likert scale data:

1. Bar charts
2. Diverging stacked bar charts

- There are much debate about which is more superior.
- However, both are equally great options as long as the graph sends the intended message to the readers.

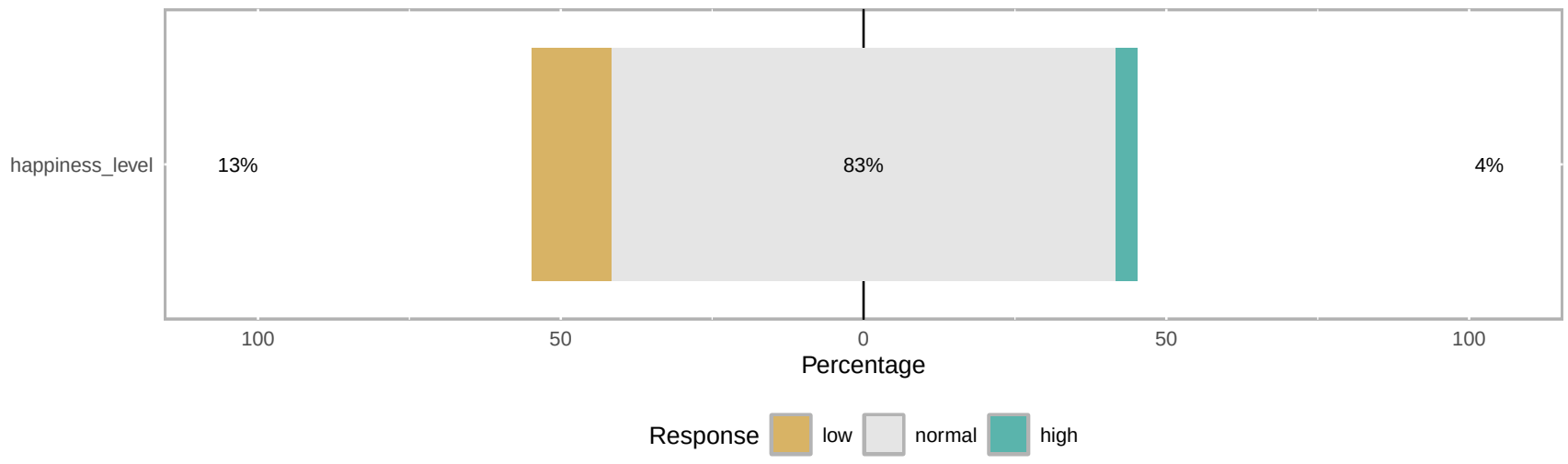
5.4.3 Visualizing likert scale data: Bar charts

```
ggplot(pokemon,
       aes(x= happiness_level, fill=happiness_level))+
  geom_bar() +
  ylab("Count")
```



5.4.3 Visualizing likert scale data: Diverging stacked bar charts

```
plot(likert(pokemon["happiness_level"]))
```



- The diverging stacked bar charts are centered at zero with a reference line at zero. - It is important that the reference line lie behind the bars; otherwise, the neither agree nor disagree group is split and appears to be two groups - The construction of diverging stacked bar charts in R requires a lot of programming knowledge.

6 Numerical summaries of numerical data

- The variable `height_m` records the height of the Pokemons in metres. Some of the Pokemons' height were unavailable and were left blank. We will attempt to summarize the mean, median and variance of the height.

```
pokemon %>%
  filter(height_m!="") %>%
  summarise(mean_height = mean(height_m),
            median_height = median(height_m),
            var_height = var(height_m))
```

```
##   mean_height median_height var_height
## 1    1.163892             1   1.167105
```

Visualization of numerical data

- From the previous examples, without the help of a graph, it was challenging to determine a suitable numerical measures to describe the data set.
- There are many ways to visualize a set of numeric values. In this workshop, we will focus on histogram, density plots and boxplots.

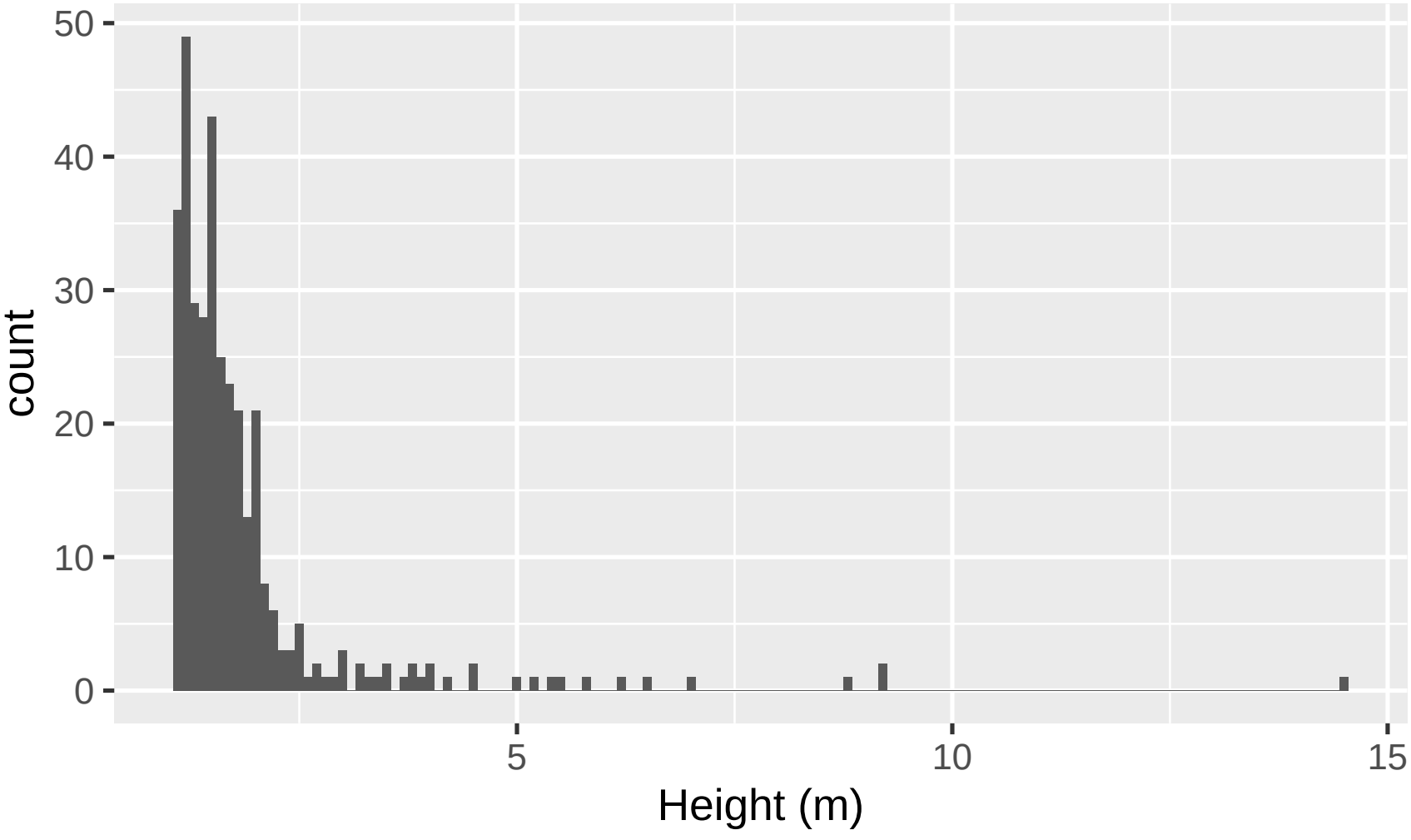
6.1 Histograms

- The dot plots are also a commonly used graphical tool to demonstrate the distribution of numerical variables. However, the dot plots can become difficult to understand when the number of values become very large.
- The histogram, which is another common visualization tool, solves this problem by aggregating the dots into bins on the x-axis and mapping the height of the bar to the number of cases that fall into that bin.
- The downside is that it becomes impossible to reconstruct the data perfectly.

6.1.1 Histograms of Pokemon's height

Lets graph a histogram with a bin width of 0.1.

```
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=height_m)) +
  geom_histogram(binwidth = 0.1) +
  xlab("Height (m)") #Label x-axes
```

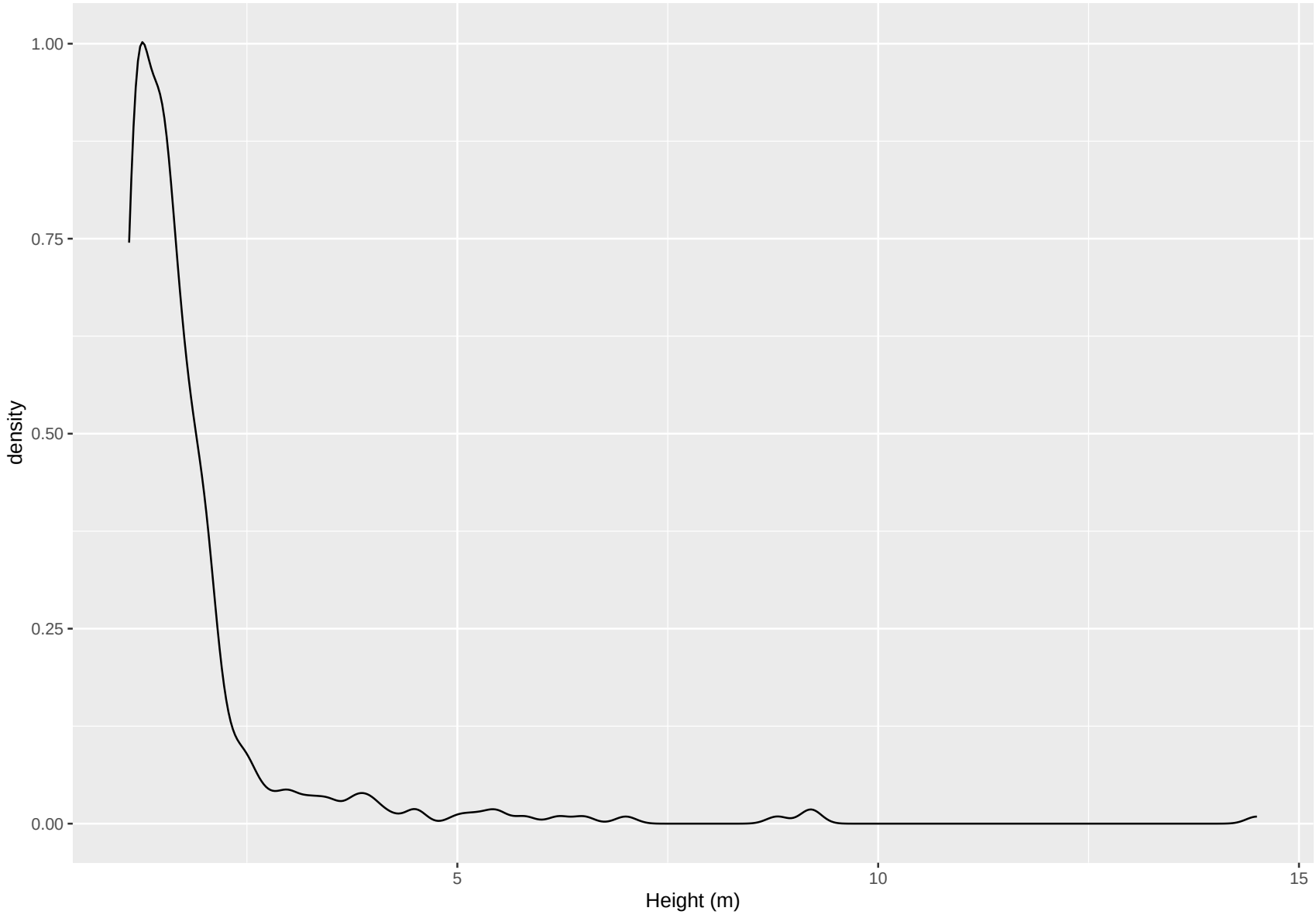

50
40
30
20
10
0
count
5
10
15
Height (m)

6.2 Density plots

- The step-like histograms may not be appealing to some readers.
- The density plot will resolve the issue.

```
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=height_m)) +
  geom_density() +
  xlab("Height (m)") #Label x-axes
```

6.2 Density plot showing the distribution of Pokemon's height



6.3 Boxplots

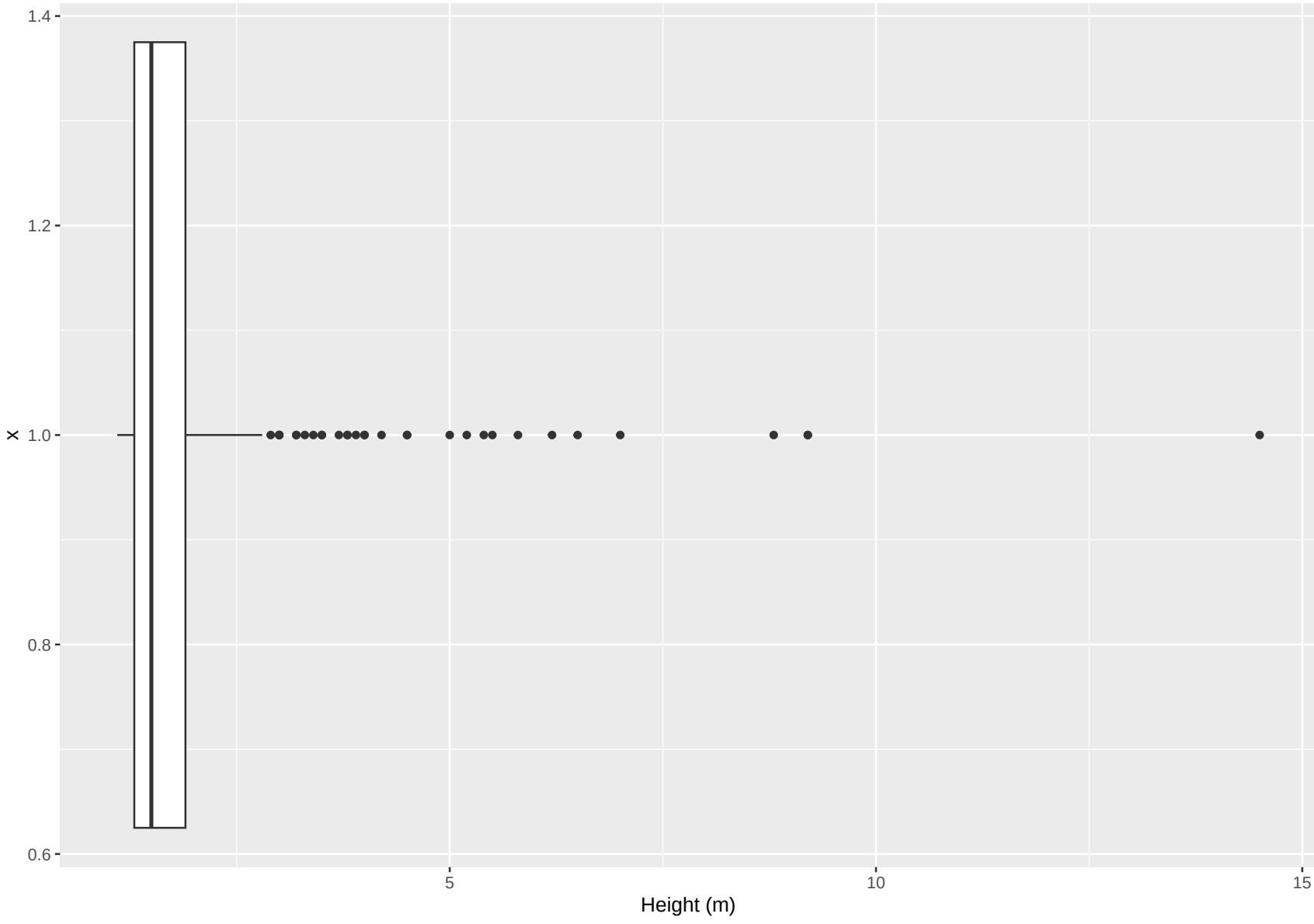
The boxplot is a visual representation of the five-number summary:

- Minimum
- First quartile, Q_1
- Second quartile, Median
- Third quartile, Q_3
    - Maximum

Potential outliers are shown as dots outside the boxplots.

6.3 Boxplot showing the distribution of Pokemon's height

```
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=1, y=height_m)) +
  geom_boxplot() +
  ylab("Height (m)") + #Label xy-axes
  coord_flip()
```

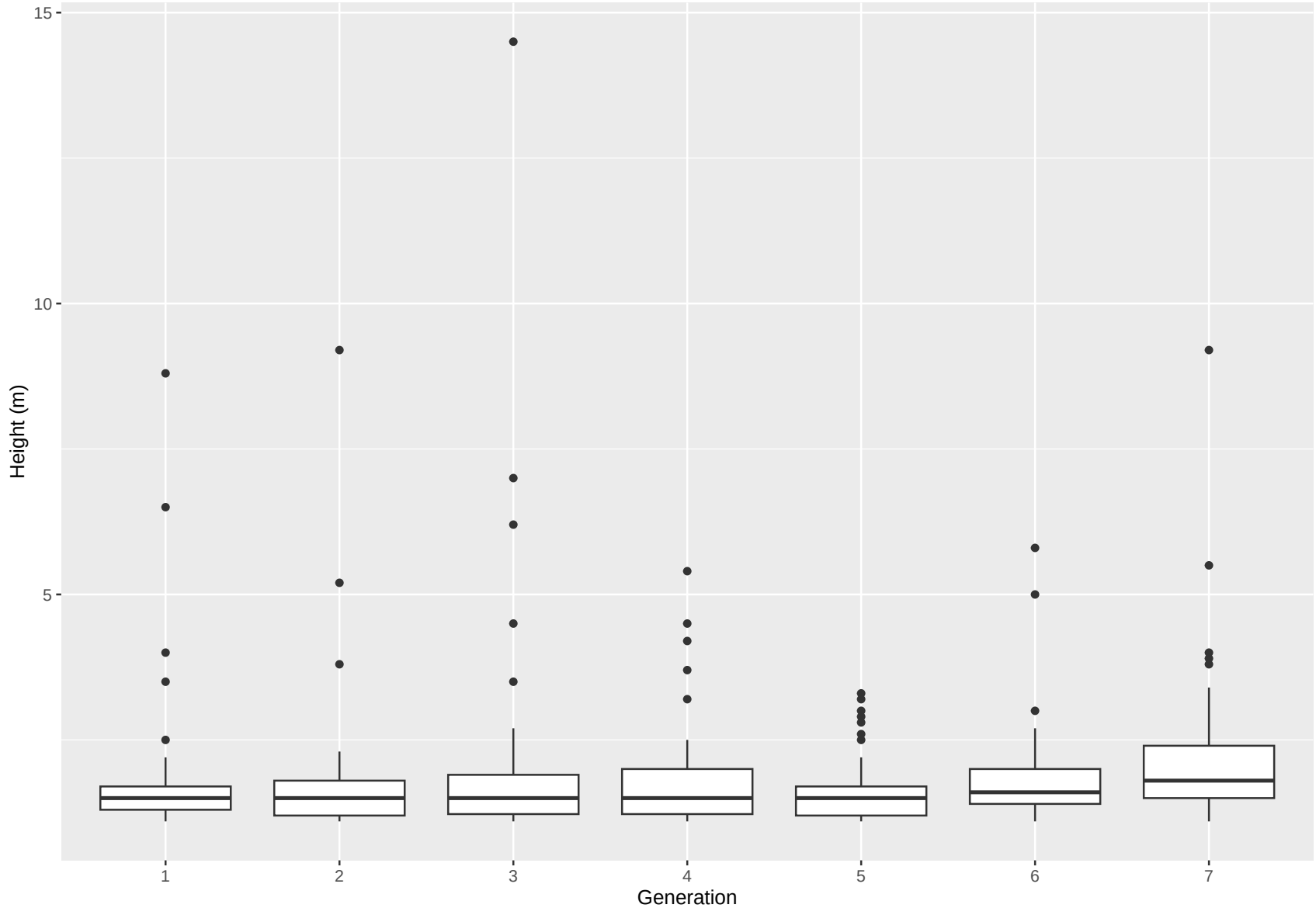

1.4
1.2
1.0
0.8
0.6
x
5
10
15
Height (m)

6.3.1 Side-by-side boxplots

- The side-by-side boxplots are used to display the distribution of several numerical variables, or a single numerical variable along with a categorical variable.
- Since the boxplots show the five-number summary, we can quickly visualize the similarities and differences between the distributions.
- Example: side-by-side boxplots (Pokemons' height by generation)

```
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=generation_f, y=height_m)) +
  geom_boxplot() +
  xlab("Generation") + ylab("Height (m)") #Label xy-axes
```

6.3.1 Side-by-side boxplots (Pokemons' height by generation)



7. Next steps

- Exploratory data analysis requires skills such as data visualization and data manipulation.
- We recommend exploring R libraries such as `ggplot2` and `dplyr`.
- At the beginning of the workshop, we mentioned that EDA is usually followed by more sophisticated analysis such as linear models, generalized linear models, machine learning, etc. We encourage you to explore the variety of statistical models later on.

Beyond this workshop

For those who are interested to learn more, the SCSRU hosts statistics seminars and workshops focusing on topics commonly encountered by researchers on campus. Please check our website for future events.

Thank you!

The Statistical Consulting and Survey Research Unit (SCSRU) is the unit through which the Department of Statistics and Actuarial Science provides statistical advice to those working on research problems.