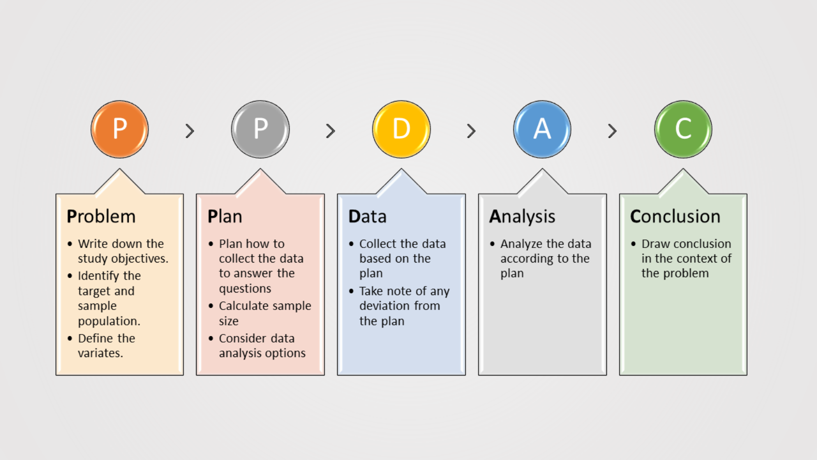
Exploratory Data Analysis with R

SCSRU Workshop

Statistical Consulting and Survey Research Unit
University of Waterloo

2024-02-14

1. Planning and conducting a study



Problem

- Write down the study objectives.
- Identify the target and sample population.
- Define the variates.

Plan

- Plan how to collect the data to answer the questions
- Calculate sample size
- Consider data analysis options

Data

- Collect the data based on the plan
- Take note of any deviation from the plan

Analysis

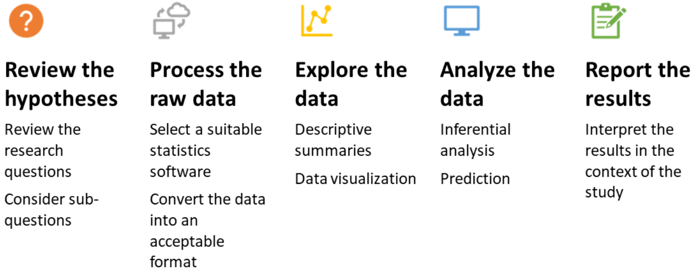
- Analyze the data according to the plan

Conclusion

- Draw conclusion in the context of the problem

Figure 1: Consider PPDAC when planning and conducting a study

After collecting the data



Review the hypotheses

Review the research questions

Consider sub-questions

Process the raw data

Select a suitable statistics software

Convert the data into an acceptable format

Explore the data

Descriptive summaries

Data visualization

Analyze the data

Inferential analysis

Prediction

Report the results

Interpret the results in the context of the study

Figure 2: Recommended process

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.

3.1 The data set

- 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.

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

3.2.1 The R libraries, ggplot2

In this workshop, we will be graphing with the package ggplot2. We find that this library to be

- consistent with the underlying grammar of graphics (Wilkinson, 2005),
- visually appealing,
- very flexible, and
- able to handle multiple outputs better.

Every ggplot requires that you specify:

- (1) the dataset
- (2) the variables and
- (3) the layers to describe how the variables are plotted.

Please install the `ggplot2` package and load it into your current work environment.

```r
# install.packages("ggplot2")
library(ggplot2)
```

3.2.2 The R libraries, dplyr

- As we explore the data, we may need to manipulate the entries and summarize some variables.
- The dplyr package contains functions designed to enable data frame manipulation in an intuitive and user-friendly manner.
- The functions covered in this workshop can be found in the cheatsheet, including:
  - group_by()
  - filter()
  - summarize()
  - mutate()

Please install the dplyr package and load it into your current work environment.

```r
# install.packages("dplyr")
library(dplyr)
```

3.2.3 The R libraries, agrmt and likert

- Lastly, we will be using two packages for the likert scale data:
- Please install both the agrmt and likert packages and load them into your current work environment.

```
# install.packages("agrmt")
library(agrmt)

# install.packages("likert")
library(likert)
```

4. Types of variables

Most data sets contain a variety of variables. In general, we can group the variables into discrete and continuous.

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.

4.1 Numeric variables

- A continuous variable is a variable that can take any value over a continuous range. Usually, the variable will have a measurement unit. Examples include:
  - Age in years.
  - Number of works hours.
  - Midterm scores.
- 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.

4.2 Discrete variables

- Discrete variables are sometimes known as *categorical or qualitative* variables. A categorical variable is a variable that can only take values over a finite set of values (or levels). Examples include:
  - A university student's major.
  - A person's blood type.
  - The type of drinks at Starbucks.
  - A person's eye colour.
  - A person's level of agreement about a statement.
- Some categorical variables can only take 2 levels. Categorical variables with only 2 levels are called *binary variable*.
  - 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.

4.2.1 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.

4.2.2 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?

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.

5. 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 take see the data set in spreadsheet format.

```
View(pokemon)
```

5.1 The function `str()`

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

```
str(pokemon)
```

- What does the output tell you?

5.2 The function `factor()`

- 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.

```r
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.

Practice 5.1

- The variable `generation` denotes the numbered generation which the Pokemon was first introduced. It is currently recorded as an integer. Do you think it is appropriate?

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

- What about the variable `type1` which records the primary type of the Pokemon? Create the a new variable called `type1_f` to denote the categorized variable.

5.3 The function, `levels()`

- 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"
```

6. 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.

```r
pokemon %>% count(type1_f)
```

```
##     type1_f   n
## 1       bug  72
## 2      dark  29
## 3    dragon  27
## 4  electric  39
## 5     fairy  18
## 6  fighting  28
## 7      fire  52
## 8    flying   3
## 9     ghost  27
## 10    grass  78
## 11   ground  32
## 12      ice  23
## 13   normal 105
## 14   poison  32
## 15  psychic  53
## 16     rock  45
## 17    steel  24
## 18    water 114
```

6.1 Contingency tables

- The contingency table, sometimes known as a two-way frequency table is a table with at least 2 rows and 2 columns to present the categorical data in terms of frequency counts. We can achieve this in R using `table()`.
- Example, we tabularize the pokemon's type (variable `type1_f`) against their generations (variable `generation_f`) by:

```
table(pokemon$type1_f, pokemon$generation_f)
```

6.1 Contingency tables (Continued)

```
##
##              1  2  3  4  5  6  7
##   bug       12 10 12  8 18  3  9
##   dark       0  5  4  3 13  3  1
##   dragon     3  0  7  3  7  4  3
##   electric   9  6  4  7  7  3  3
##   fairy      2  5  0  1  0  9  1
##   fighting   7  2  4  2  7  3  3
##   fire      12  8  6  5  8  8  5
##   flying     0  0  0  0  1  2  0
##   ghost      3  1  4  6  5  4  4
##   grass     12  9 12 13 15  5 12
##   ground     8  3  6  4  9  0  2
##   ice        2  4  6  3  6  2  0
##   normal    22 15 18 17 17  4 12
##   poison    14  1  3  6  2  2  4
##   psychic    8  7  8  7 14  3  6
##   rock       9  4  8  6  6  8  4
##   steel      0  2  9  3  4  4  2
##   water     28 18 24 13 17  5  9
```

6.1 Contingency tables (Continued)

- Using `dplyr`, we can obtain a similar contingency table:

```
pokemon %>%
  group_by(generation_f) %>%
  count(type1_f)
```

```
## # A tibble: 114 x 3
## # Groups:   generation_f [7]
##    generation_f type1_f      n
##    <fct>        <fct>    <int>
##  1 1            bug         12
##  2 1            dragon       3
##  3 1            electric     9
##  4 1            fairy        2
##  5 1            fighting     7
##  6 1            fire        12
##  7 1            ghost        3
##  8 1            grass       12
##  9 1            ground       8
## 10 1            ice          2
## # i 104 more rows
```

Practice 6.1

Let's create a contingency table of the `generation` and `is_legendary` to find out whether the number of legendary pokemon differs by generation.

Do you see?

```
##
##         1   2   3   4   5   6   7
##   No  146  94 125  94 143  66  63
##   Yes   5   6  10  13  13   6  17
```

Practice 6.1

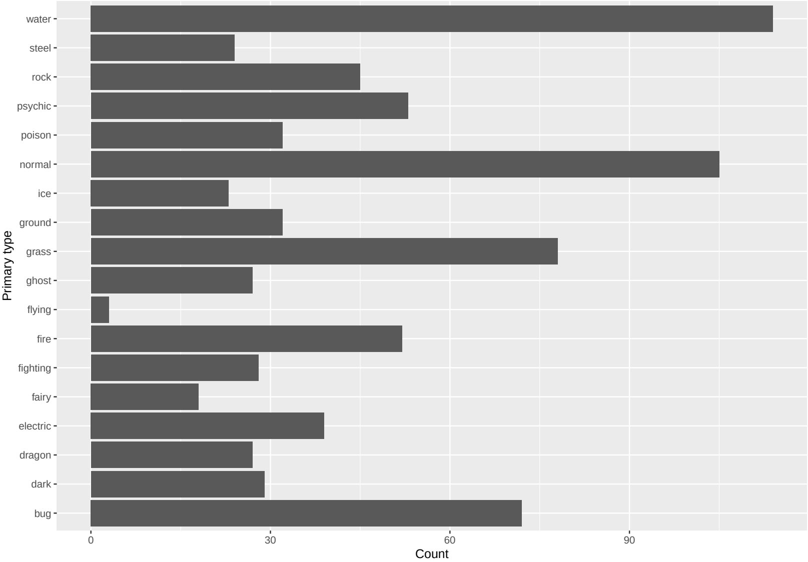
If you use `dplyr`, do you see?

```
## # A tibble: 14 x 3
## # Groups:   is_legendary_f [2]
##    is_legendary_f generation_f     n
##    <fct>          <fct>        <int>
##  1 No             1              146
##  2 No             2               94
##  3 No             3              125
##  4 No             4               94
##  5 No             5              143
##  6 No             6               66
##  7 No             7               63
##  8 Yes            1                5
##  9 Yes            2                6
## 10 Yes            3               10
## 11 Yes            4               13
## 12 Yes            5               13
## 13 Yes            6                6
## 14 Yes            7               17
```

6.2 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.

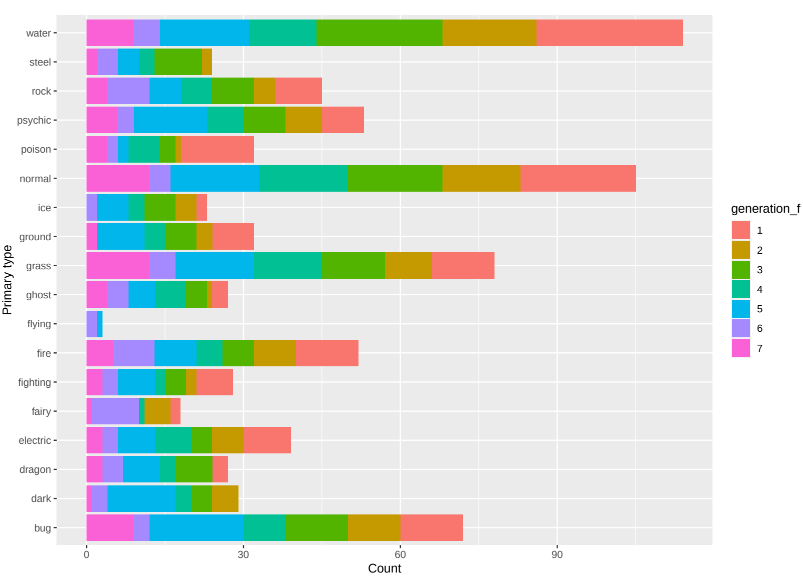
```r
ggplot(pokemon, aes(x=type1_f)) +
  geom_bar()+ # Type of chart
  xlab("Primary type") + ylab("Count")+ # Label the xy-axes
  coord_flip() # Flip the xy-axes
```


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

6.2.1 Stacked bar charts

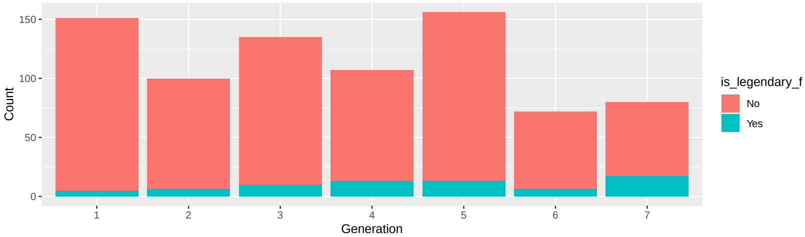
- To better visualize the contingency table and relationship between two categorical variables, we can use a stacked bar chart.

```
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
```


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

Practice 6.2

- Recall the contingency table of the `generation` and `is_legendary` created in Practice 6.1.
- Create a bar graph to visualize the relationship between the two variables.
- Do you have a similar graph?



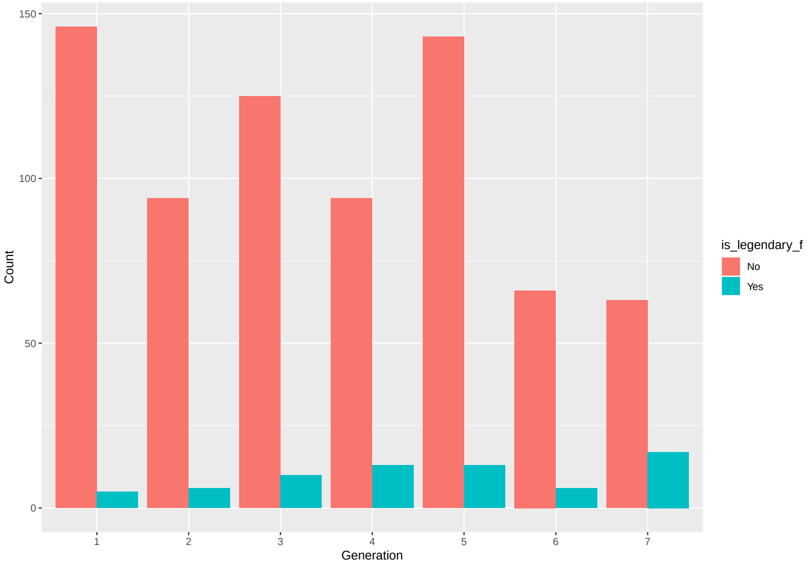
Practice 6.2

- What do you see from the bar chart?
- Did you notice the trend when looking at the tables earlier?

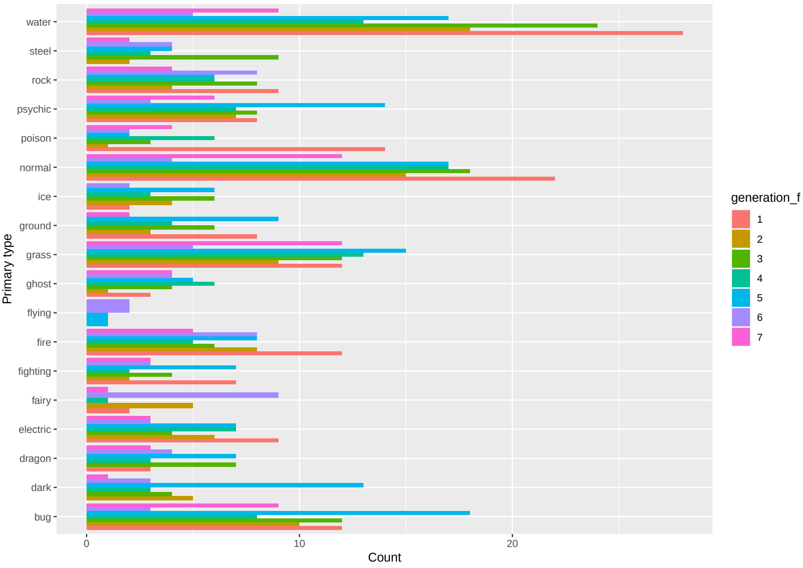
6.2.2 Side-by-side bar charts

- 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.
- The trend in the stacked bar chart in Practice 6.2 are shown in the side-by-side bar chart as well.

```
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

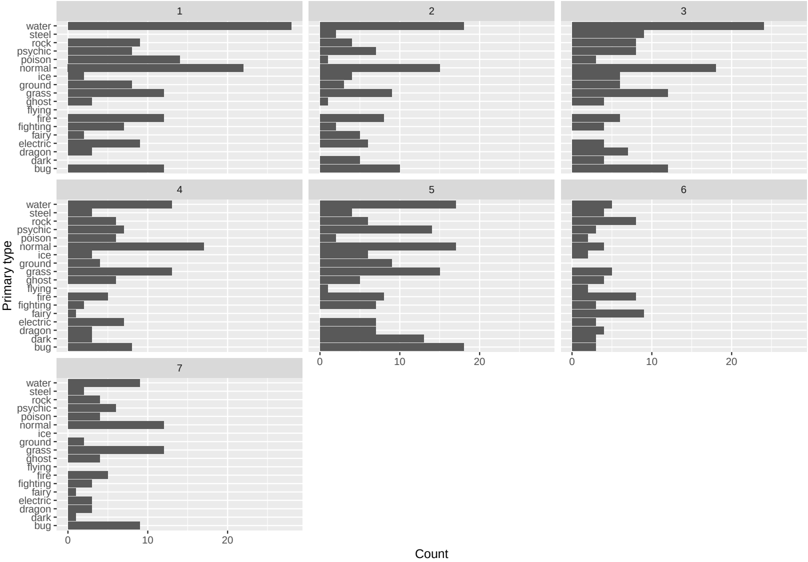
When side-by-side bar chart does not work



6.2.3 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.

```r
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
Primary type
0
10
20
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.

Practice 6.3

- Pokemons have different level of happiness. Create a categorical variable called `happiness_level` with three categories:
    - `low`: `base_happiness` < 70
    - `normal`: `base_happiness` $= 70$
    - `high`: `base_happiness` > 70
- Create a stacked bar chart, side-by-side bar chart and a faceted bar chart to investigate the relationship between `generation` and `happiness_level`.
- Which chart do you recommend?

6.3 Proportion

- The tables we have seen so far summarized the count of each category within a variable.
- However, the proportions can suggest more information when investigating relationships with other variables.
- This is particularly important when we are interested in the difference between the treatment and control groups.
- The functions `proportions()` or `prop.table()` will produce table with proportions about the variable.

Example 6.1

```
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.

6.3.1 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.

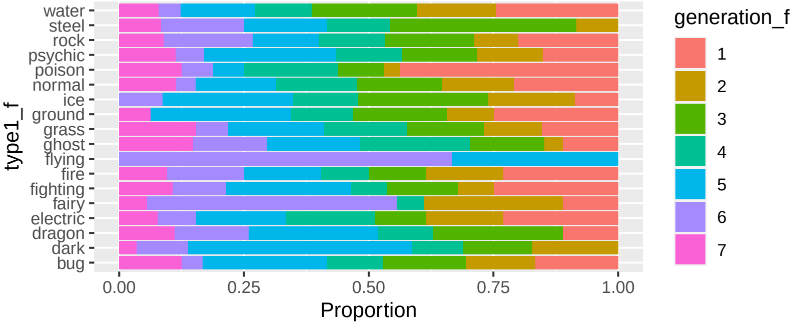
Example 6.2

```r
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()
```



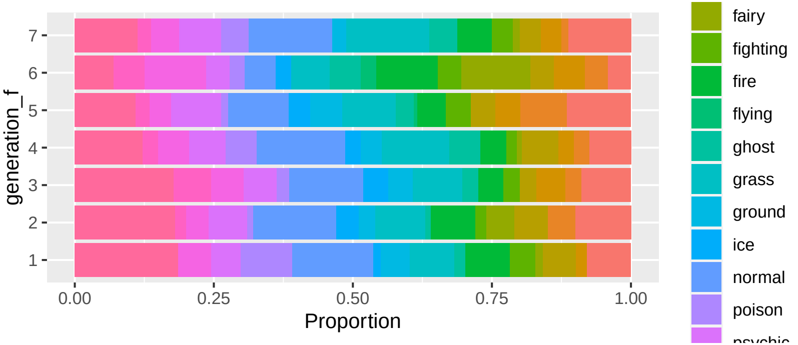
Example 6.3

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

```
##
##               1    2    3    4    5    6    7
##   bug      0.08 0.10 0.09 0.07 0.12 0.04 0.11
##   dark     0.00 0.05 0.03 0.03 0.08 0.04 0.01
##   dragon   0.02 0.00 0.05 0.03 0.04 0.06 0.04
##   electric 0.06 0.06 0.03 0.07 0.04 0.04 0.04
##   fairy    0.01 0.05 0.00 0.01 0.00 0.12 0.01
##   fighting 0.05 0.02 0.03 0.02 0.04 0.04 0.04
##   fire     0.08 0.08 0.04 0.05 0.05 0.11 0.06
##   flying   0.00 0.00 0.00 0.00 0.01 0.03 0.00
##   ghost    0.02 0.01 0.03 0.06 0.03 0.06 0.05
##   grass    0.08 0.09 0.09 0.12 0.10 0.07 0.15
##   ground   0.05 0.03 0.04 0.04 0.06 0.00 0.03
##   ice      0.01 0.04 0.04 0.03 0.04 0.03 0.00
##   normal   0.15 0.15 0.13 0.16 0.11 0.06 0.15
##   poison   0.09 0.01 0.02 0.06 0.01 0.03 0.05
##   psychic  0.05 0.07 0.06 0.07 0.09 0.04 0.07
##   rock     0.06 0.04 0.06 0.06 0.04 0.11 0.05
##   steel    0.00 0.02 0.07 0.03 0.03 0.06 0.03
##   water    0.19 0.18 0.18 0.12 0.11 0.07 0.11
```

The table shows us that among the Generation 1 pokemons. 17% are bug type, 0% are `dark` type, 11% are `dragon` type, etc.

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



Practice 6.4

Consider the `happiness_level` created earlier. What is the proportion of Pokemons from Generation 1 that have

- `low` happiness level?
- `normal` happiness level?
- `high` happiness level?

Create a suitable graph to represent the above information.

6.4 Pie chart

- The pie chart is a common way to display categorical data where the size of the slice corresponds to the proportion of cases that are in that category.
- However, it is difficult to compare relative size of the slices.
- For that reason, it is generally a good idea to stick to bar charts.

6.5 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.
- Recall the variable `happiness_level` that contains three levels: low, normal and high.

```r
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.

6.5.1 Measures of central tendency

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

```r
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.

6.5.2 Measure of spread: consensus

- A measure of spread for ordinal data is called *consensus* (Tastle & Wierman, 2007).
- The consensus ranges from 0 to 1.
- A 0 indicates a lack of consensus, whereas a consensus of one would indicate all the respondents are in agreement.
- To calculate consensus, we can use the `consensus()` function from the library `agrmt`.

Example 6.4

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

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

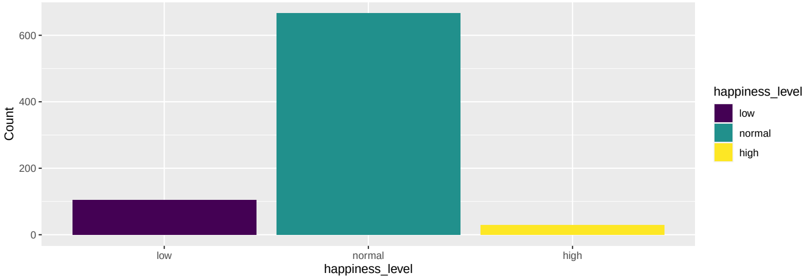
- The consensus is around 0.79.
- This means that the Pokemons are pretty agreeable on their levels of happiness.
- This will not be surprising after we take a look at the distribution of `happiness_level` visually.

6.5.3 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.

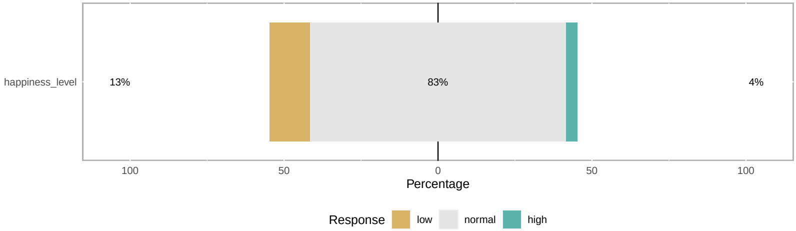
6.5.3 Visualizing likert scale data: Bar charts

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



6.5.3 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.

7. Numerical data

- Compared to categorical variables, numerical variables have more numerical summaries.
- We want to look at how the numeric values are distributed and typically summarized them with:
  - measures of centre
  - measures of spread

7.1 Measures of centre

- The measures of centre is a measure used to summarize a set of values with a single value that represents the middle or centre of its distribution.
- Common measures include:
  - mean: the average of all the values.
  - median: the value that falls in the middle when the data are ordered from smallest to largest.
  - mode: the value that occurs most often.

7.2 Measures of spread

- The measures of spread describe the variability of the set of values.
- Variability is often an extremely important consideration. Common measures include:
    - range: Maximum - Minimum.
    - interquartile range: the spread of the middle half when the data are ordered from smallest to largest.
    - variance: squared distance from the mean.
    - standard deviation: the square root of variance.

Example 7.1

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
```

Practice 7.1

The variable `weight_m` records the weight of the Pokemons in kilograms. Without the help of a graph, summarize the mean, median, variance and interquartile range of this variable.

- Comment on the spread of this variable?
- What are your recommended measures of centre and spread?

7.3 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 cover:
    - dot plots
    - histograms
    - density plots
    - boxplots

7.3.1 Dot plots

- A dot plot is also known as a strip plot or a dot chart.
- It is a simple form of data visualization that consists of data points plotted as dots on a graph.
- The plot groups the data as little as possible so that the individual observations are not lost.

Example 7.1 revisit

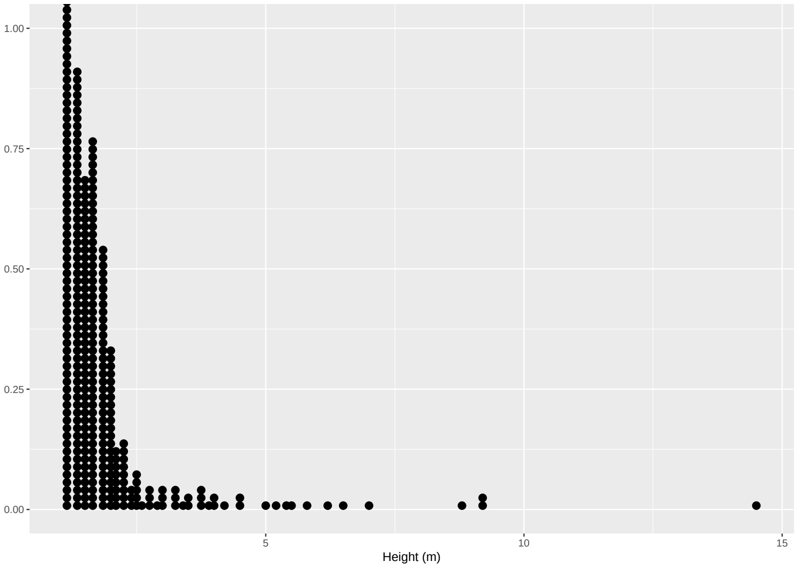
- Suppose we are interested to look at the distribution of Pokemons' height but focusing on those that are taller than 1m.

```r
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=height_m)) +
  geom_dotplot(binwidth = 0.1, dotsize = 1.5) +
  xlab("Height (m)")+ylab("")  #Label xy-axes
```

Notes:

- The numbers on y axis are not meaningful, due to technical limitations of ggplot2. You can hide the y axis, as in one of the examples, or manually scale it to match the number of dots.
- There is a reason we did not use the entire data set to demonstrate the dot plot. What is it?

Dot plot showing the distribution of Pokemon's height



7.4 Histograms

- The dot plots 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.

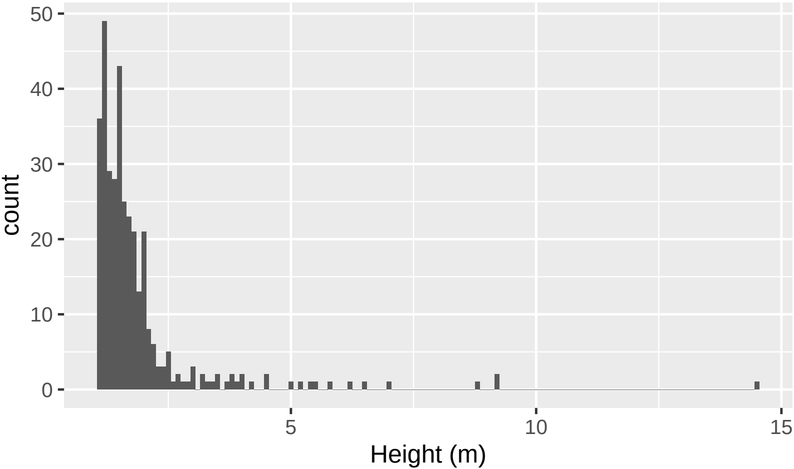
Example 7.1 revisit

Let's replace the dotplot earlier with a histogram with a binwidth of 0.1.

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

Example 7.1 revisit

- Let's replace the previous dot plot with a histogram with a bin width of 0.1.



Practice 7.2

- The shape of the distribution is affected by the width of the bin.
- Try adjusting the `binwidth` to find a recommended `binwidth`.

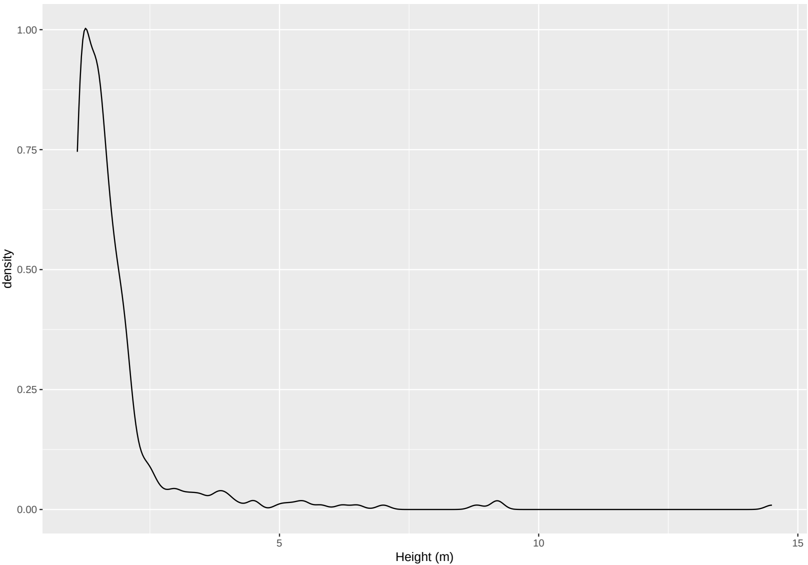
```r
pokemon %>%
  filter(height_m > 1) %>%
  ggplot(aes(x=height_m)) +
  geom_histogram(binwidth = 0.1) +
  xlab("Height (m)") #Label x-axes
```

7.5 Density plots

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

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

Density plot showing the distribution of Pokemon's height



7.6 Boxplots

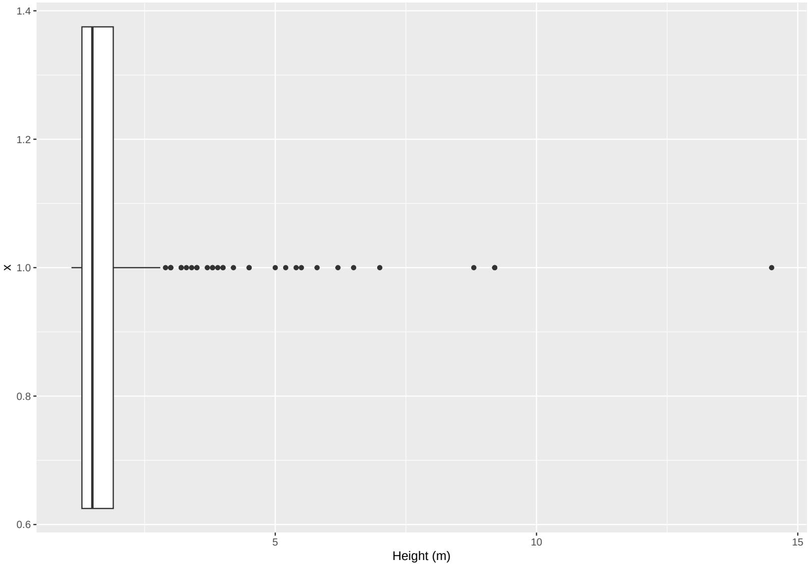
The boxplot is a visual representation of the five-number summary:

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

Potential outliers are shown as dots outside the boxplots.

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)

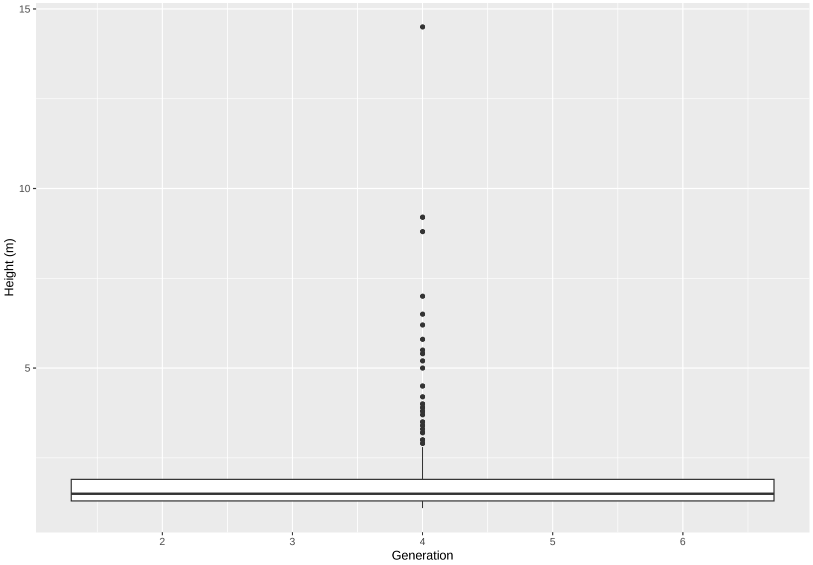
7.6.1 Side-by-side boxplots

- 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.

Side-by-side boxplots comparing Pokemons' height by generation

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

```
## Warning: Continuous x aesthetic
## i did you forget 'aes(group = ...)'?
```

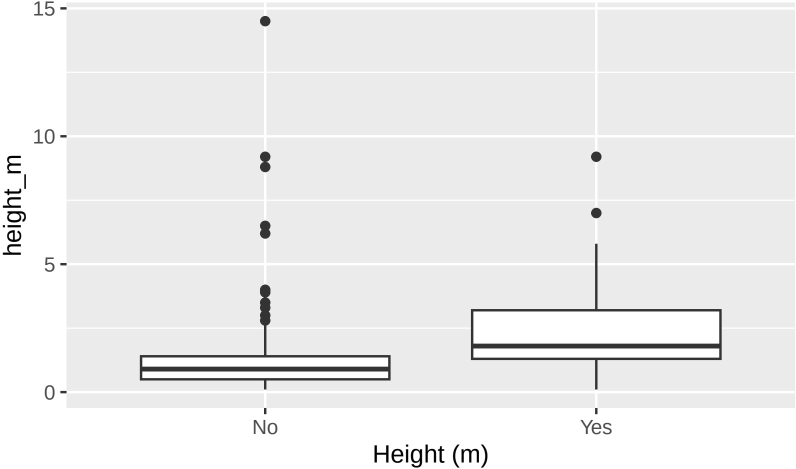


7.7 Outliers

- An important aspect of EDA is to look for outliers.
- Outliers are values that are extremely far away from the bulk of distribution.
- They are often interesting cases that the experts want to investigate.
- Instead of removing the outliers, a better idea is to create a column to indicate whether the observation is an outlier. When performing analysis, we can filter out the outliers as needed.

Practice 7.3

- Create side-by-side boxplots to compare the distribution of Pokemon's height by whether the Pokemons' are legendary, `is_legendary`.



Practice 7.4

- Based on the boxplots, recommend cut-off values and create a column `is_outlier` to indicate whether the a Pokemon's height is potentially an outlier.

8. Visualization in higher dimensions

So far, the discussion is centered around 2 variables such as:

- How does X affect Y?
- Are A and B correlated?

Can we expand on that and explore the relationship of multiple variables at the same time?

The answer is yes. For the remaining time, we will take a look at summarizing three variables at the same time.

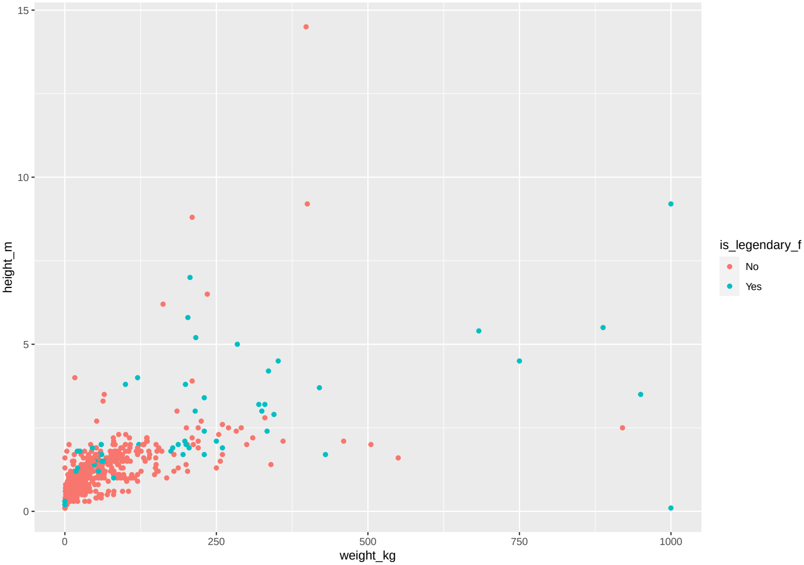
8.1 Multi-dimensional plots

- When considering three variables, many will think about the 3D-plots.
- Higher dimensional plots are theoretically correct, but explaining and presenting high dimensional plots can be challenging.
- Instead of high dimensional plots, we can consider options such as shapes, sizes, colours, and patterns.
- Keep in mind that “simple is more”.

Example 8.1

- Suppose we want to inspect the relationship between the Pokemon's weight (`weight_kg`), height (`height_m`) and whether they are legendary (is_legendary‘).
- All three variables are continuous.
- It is natural to consider a 3D scatterplot but we will take a different route by changing the shape and sizes of a 2D scatterplot.

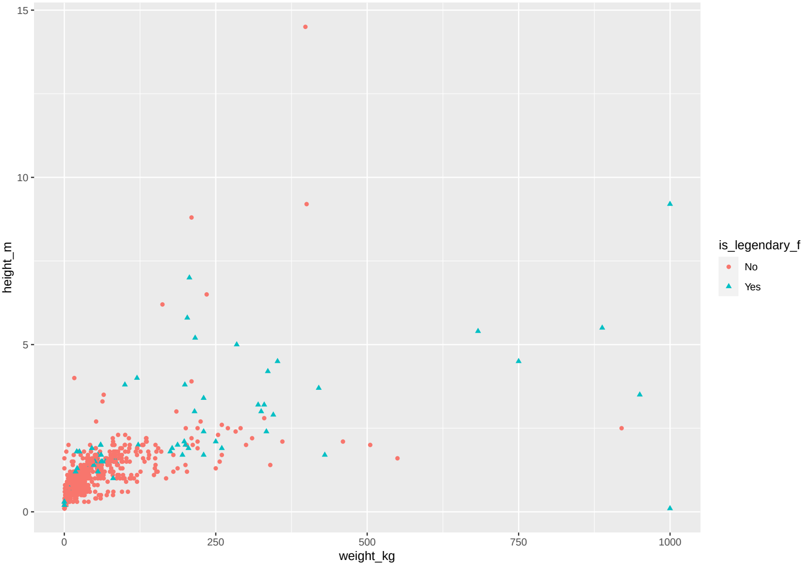
```r
ggplot(pokemon, aes(x=weight_kg, y=height_m)) +
  geom_point(aes(col = is_legendary_f))
```


15
10
5
0
height_m
0
250
500
750
1000
weight_kg
is_legendary_f
● No
● Yes

- We can make this a little better (or not) by changing the shape of the points.

```
ggplot(pokemon, aes(x=weight_kg, y=height_m)) +
  geom_point(aes(col = is_legendary_f,
                 shape = is_legendary_f))
```

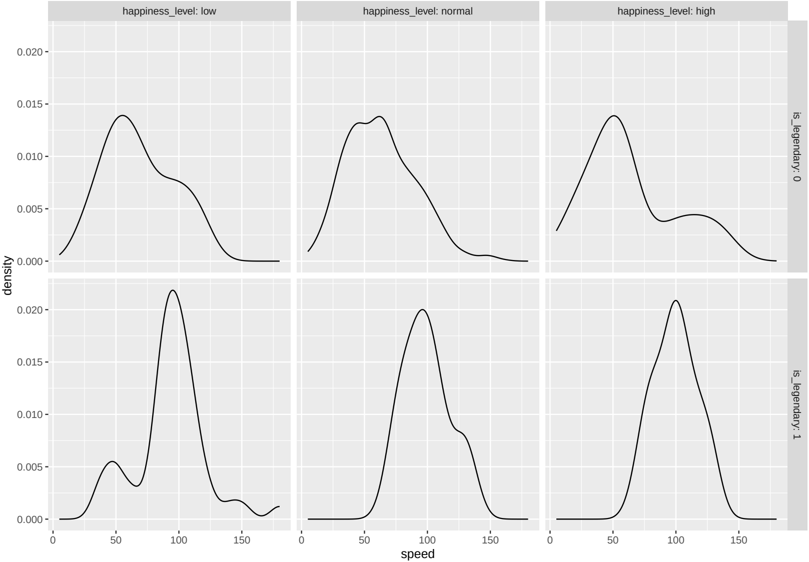
- Keep in mind: Simple is more.


15
10
5
0
height_m
0
250
500
750
1000
weight_kg
is_legendary_f
No
Yes

8.2 Faceting

- Faceting is also a great option when dealing with multiple models.
- Suppose we are interested to know about the relationship between the Pokemon's speed (speed), their happiness level (happiness_level) and whether they are legendary (is_legendary). Since speed is a numerical value, we can use the density plot.

```r
ggplot(pokemon, aes(x=speed)) +
  geom_density() +
  facet_grid(is_legendary~happiness_level,
             labeller = label_both)
```


happiness_level: low
happiness_level: normal
happiness_level: high
is_legendary: 0
is_legendary: 1
density
0.000
0.005
0.010
0.015
0.020
0
50
100
150
speed

Practice 8.2

- Recall that the histogram is also suitable for numeric variable. How do we visualize the relationship of the three variables: `speed`, `happiness_level` and `is_legendary`?

9. 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.