Data Visualisation using ggplot2

Contents

Introduction

In this practical, we will learn how to visualise data after we have cleaned up our datasets using the `dplyr` verbs from the previous practical: `filter()`, `arrange()`, `mutate()`, `select()`, and `summarise()`. For the visualisations, we will be using a package that implements the grammar of graphics: `ggplot2`. Please review the lecture slide for week2 beforehand.

Don't forget to open the project file `03_Data_visualisation.Rproj` and to create your `.Rmd` or `.R` file to work in.

```
library(ISLR)
library(tidyverse)
```

An excellent reference manual for `ggplot` can be found on the tidyverse website: https://ggplot2.tidyverse.org/reference/

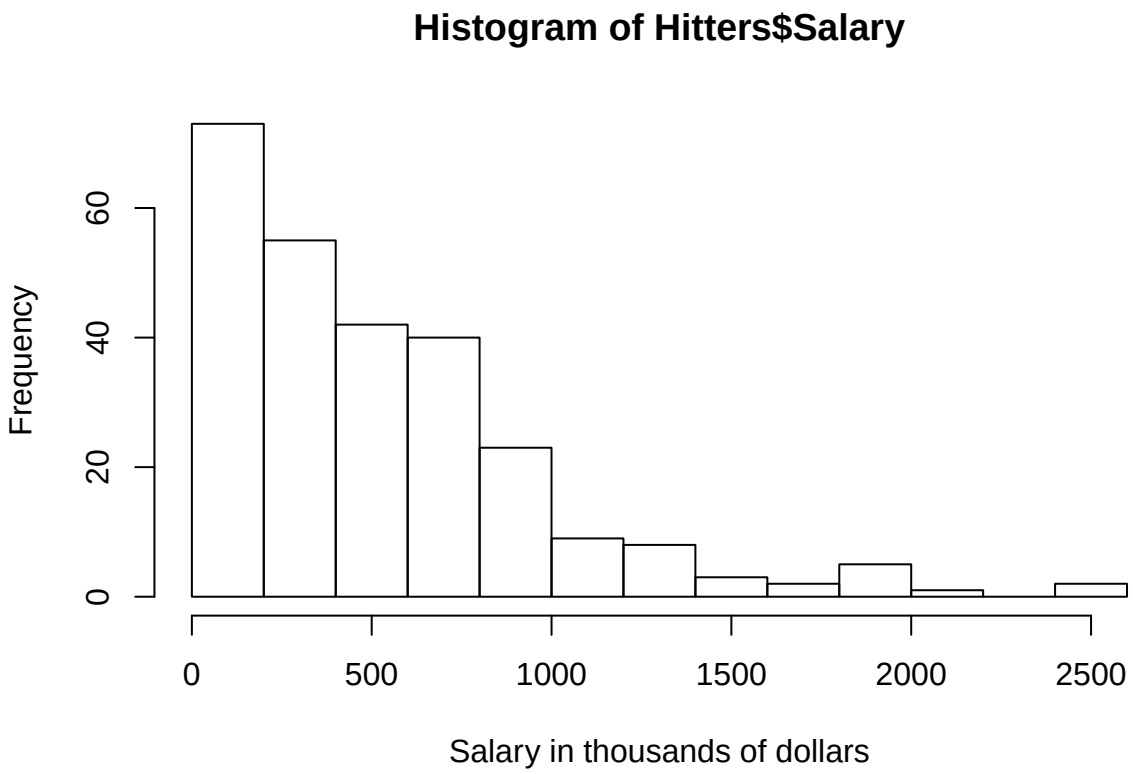
What is `ggplot`?

Plots can be made in `R` without the use of `ggplot` using `plot()`, `hist()` or `barplot()` and related functions. Here is an example of each on the `Hitters` dataset from `ISLR`:

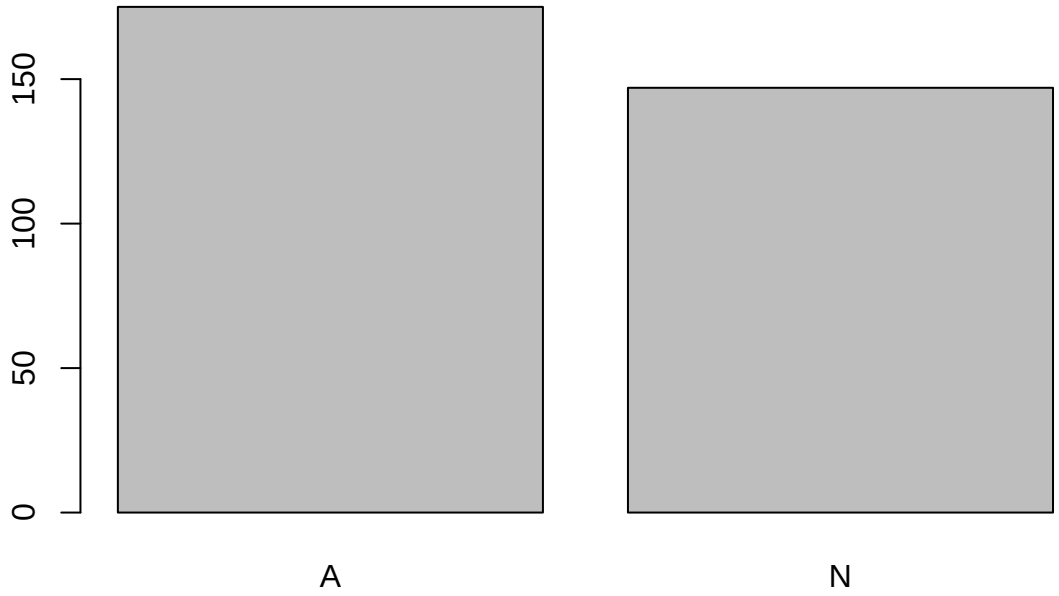
```
# Get an idea of what the Hitters dataset looks like
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
## -Andy Allanson      293   66     1   30  29    14     1    293    66
## -Alan Ashby         315   81     7   24  38    39    14   3449   835
## -Alvin Davis        479  130    18   66  72    76     3   1624   457
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575
## -Andres Galarraga   321   87    10   39  42    30     2    396   101
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133
##                   CHmRun CRuns CRBI CWalks League Division PutOuts Assists
## -Andy Allanson         1    30   29     14      A        E     446      33
## -Alan Ashby           69   321  414    375      N        W     632      43
## -Alvin Davis          63   224  266    263      A        W     880      82
## -Andre Dawson        225   828  838    354      N        E     200      11
## -Andres Galarraga     12    48   46     33      N        E     805      40
## -Alfredo Griffin      19   501  336    194      A        W     282     421
##                   Errors Salary NewLeague
## -Andy Allanson        20     NA         A
## -Alan Ashby           10  475.0         N
## -Alvin Davis          14  480.0         A
## -Andre Dawson          3  500.0         N
## -Andres Galarraga      4   91.5         N
## -Alfredo Griffin      25  750.0         A
```

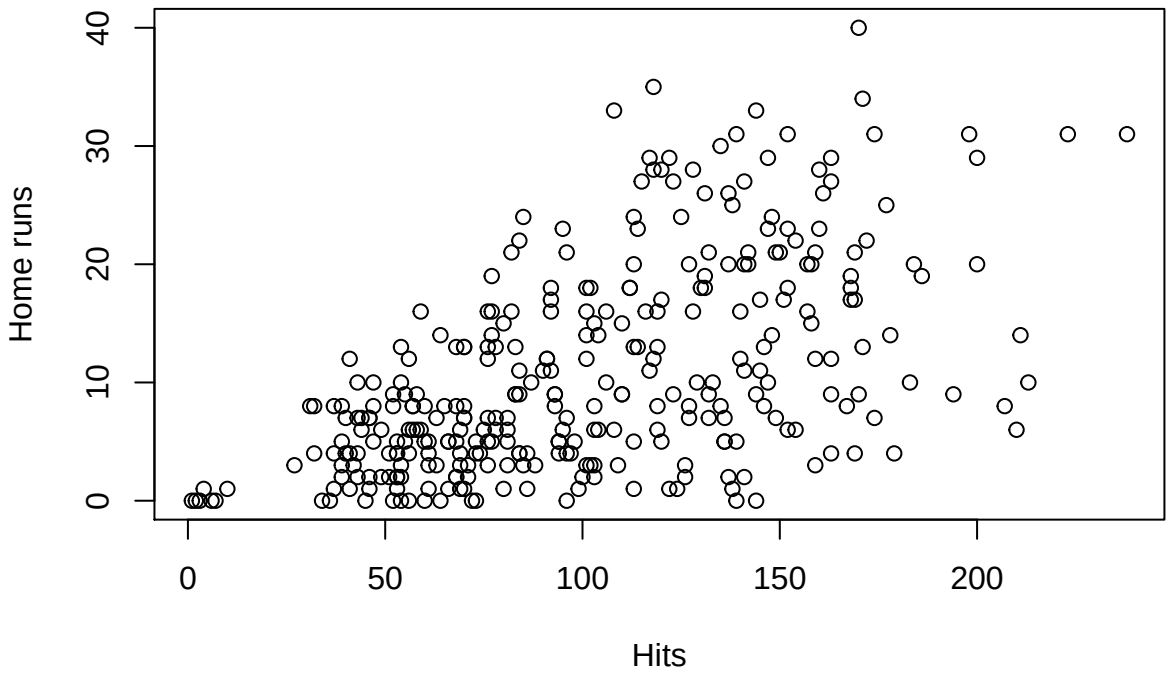
```r
# histogram of the distribution of salary
hist(Hitters$Salary, xlab = "Salary in thousands of dollars")
```




```
# barplot of how many members in each league
barplot(table(Hitters$League))
```



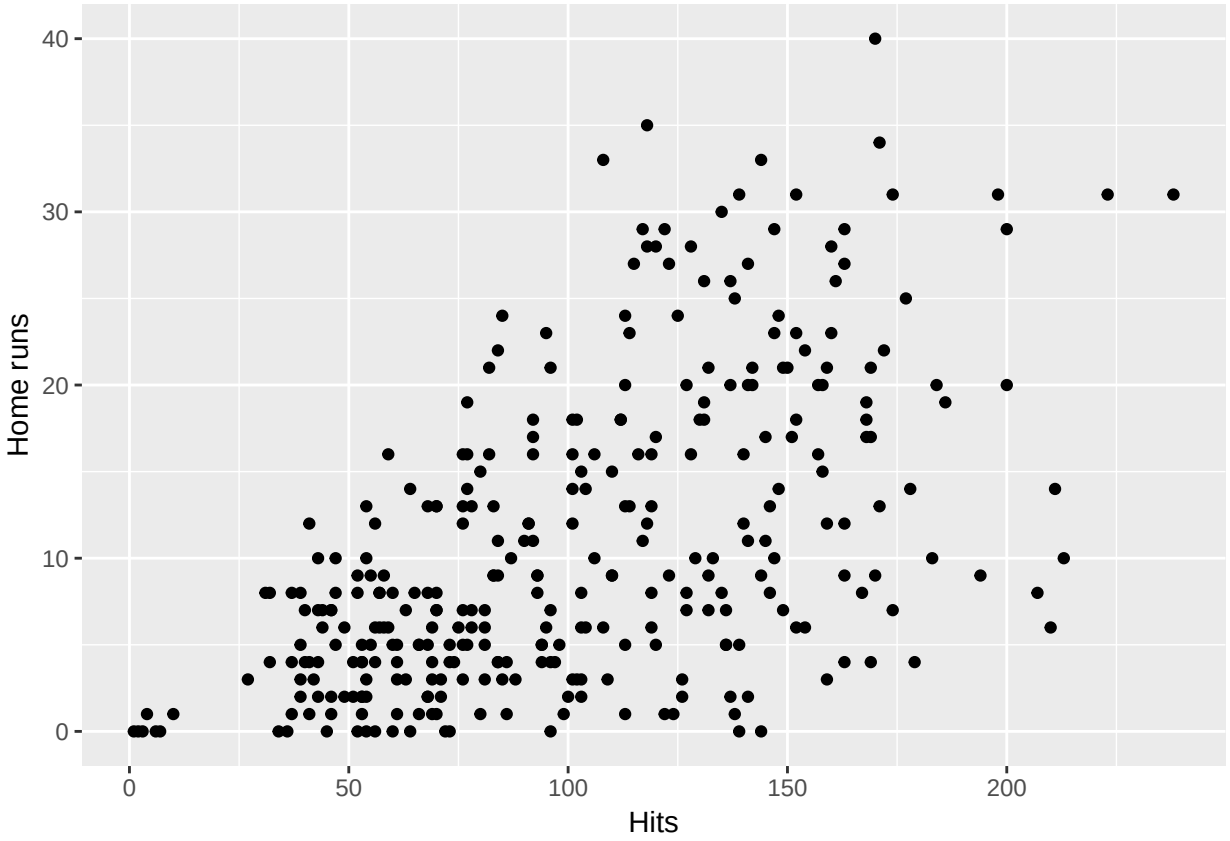

```
# number of career home runs vs 1986 home runs
plot(x = Hitters$Hits, y = Hitters$HmRun,
     xlab = "Hits", ylab = "Home runs")
```



These plots are informative and useful for visually inspecting the dataset, and they each have a specific syntax associated with them. `ggplot` has a more unified approach to plotting, where you build up a plot layer by layer using the + operator:

```
homeruns_plot <-
  ggplot(Hitters, aes(x = Hits, y = HmRun)) +
  geom_point() +
  labs(x = "Hits", y = "Home runs")

homeruns_plot
```

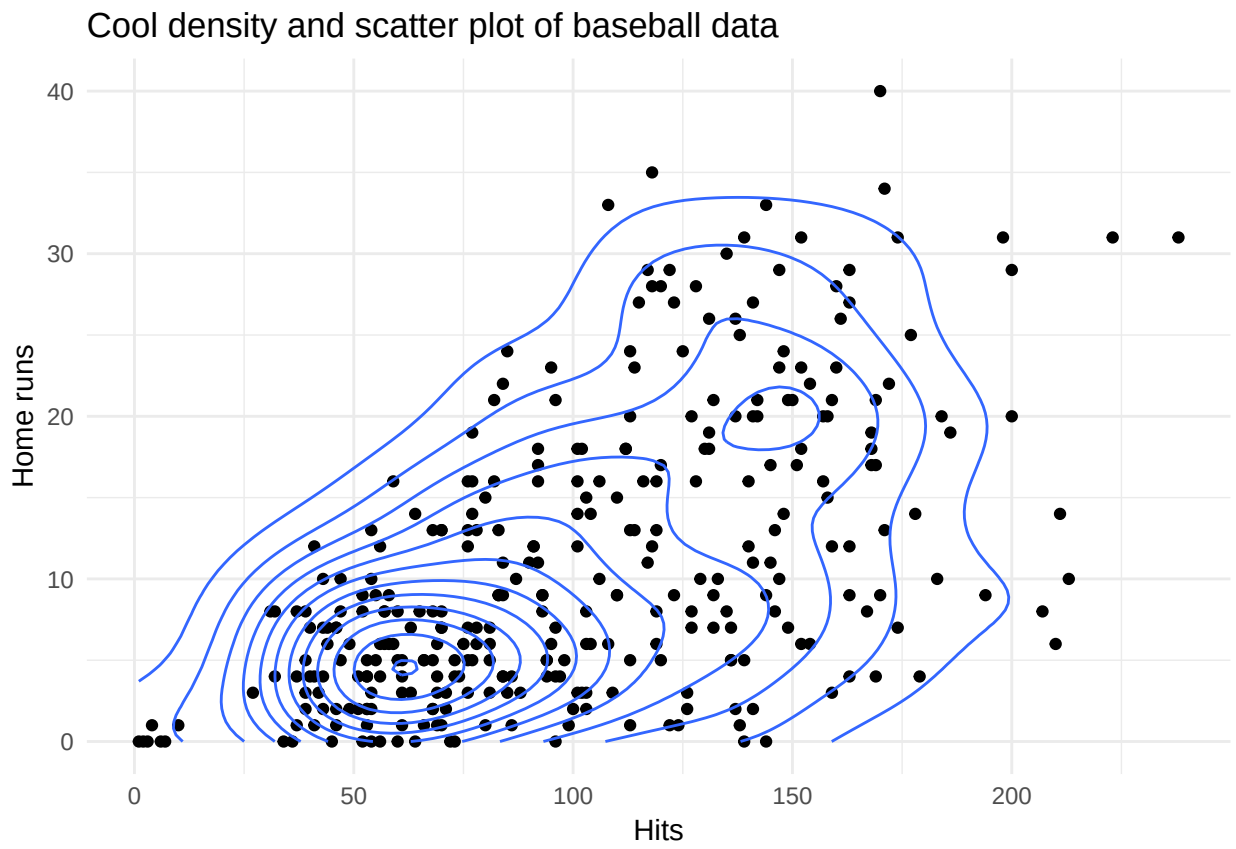


As introduced in the lectures, a `ggplot` object is built up in different layers:

1. input the dataset to a `ggplot()` function call
2. construct aesthetic mappings
3. add (geometric) components to your plot that use these mappings
4. add labels, themes, visuals.

Because of this layered syntax, it is then easy to add elements like these fancy density lines, a title, and a different theme:

```
homeruns_plot +
  geom_density_2d() +
  labs(title = "Cool density and scatter plot of baseball data") +
  theme_minimal()
```



In conclusion, `ggplot` objects are easy to manipulate and they force a principled approach to data visualisation. In this practical, we will learn how to construct them.

---

1. **Name the aesthetics, geoms, scales, and facets of the above visualisation. Also name any statistical transformations or special coordinate systems.**

Aesthetics and data preparation

The first step in constructing a `ggplot` is the preparation of your data and the mapping of variables to aesthetics. In the `homeruns_plot`, we used an existing data frame, the `Hitters` dataset.

The data frame needs to have proper column names and the types of the variables in the data frame need to be correctly specified. Numbers should be `numerics`, categories should be `factors`, and names or identifiers should be `character` variables. `ggplot()` *always* expects a data frame, which may feel awfully strict, but it allows for excellent flexibility in the remaining plotting steps.

---

2. **Run the code below to generate data. There will be three vectors in your environment. Put them in a data frame for entering it in a `ggplot()` call using either the `data.frame()` or the `tibble()` function. Give informative names and make sure the types are correct (use the `as.<type>()` functions). Name the result `gg_students`**

```
set.seed(1234)
student_grade  <- rnorm(32, 7)
student_number <- round(runif(32) * 2e6 + 5e6)
programme      <- sample(c("Science", "Social Science"), 32, replace = TRUE)
```

Mapping aesthetics is usually done in the main `ggplot()` call. Aesthetic mappings are the second argument to the function, after the data frame.

---

3. **Plot the first `homeruns_plot` again, but map the `Hits` to the y-axis and the `HmRun` to the x-axis instead.**

---

4. **Recreate the same plot once more, but now also map the variable `League` to the `colour` aesthetic and the variable `Salary` to the `size` aesthetic.**

Examples of aesthetics are:

- x
- y
- alpha (transparency)
- colour
- fill
- group
- shape
- size
- stroke

Geoms

Up until now we have used two geoms: contour lines and points. The geoms in `ggplot2` are added via the `geom_<geomtype>()` functions. Each geom has a required aesthetic mapping to work. For example, `geom_point()` needs at least and x and y position mapping, as you can read here. You can check the required aesthetic mapping for each geom via the `?geom_<geomtype>`.

---

5. **Look at the many different geoms on the reference website.**

There are two types of geoms:

- geoms which perform a transformation of the data beforehand, such as `geom_density_2d()` which calculates contour lines from x and y positions.
- geoms which do not transform data beforehand, but use the aesthetic mapping directly, such as `geom_point()`.

Visual exploratory data analysis

Several types of plots are useful for exploratory data analysis. In this section, you will construct different plots to get a feel for the two datasets we use in this practical: `Hitters` and `gg_students`. One of the most common tasks is to look at the distributions of variables in your dataset.

Histogram

---

6. **Use `geom_histogram()` to create a histogram of the grades of the students in the `gg_students` dataset. Play around with the `binwidth` argument of the `geom_histogram()` function.**

Density

The continuous equivalent of the histogram is the density estimate.

---

7. **Use `geom_density()` to create a density plot of the grades of the students in the `gg_students` dataset. Add the argument `fill = "light seagreen"` to `geom_density()`.**

The downside of only looking at the density or histogram is that it is an abstraction from the raw data, thus it might alter interpretations. For example, it could be that a grade between 8.5 and 9 is in fact impossible. We do not see this in the density estimate. To counter this, we can add a raw data display in the form of rug marks.

---

8. **Add rug marks to the density plot through `geom_rug()`. You can edit the `colour` and `size` of the rug marks using those arguments within the `geom_rug()` function.**

9. **Increase the data to ink ratio by removing the y axis label, setting the theme to `theme_minimal()`, and removing the border of the density polygon. Also set the limits of the x-axis to go from 0 to 10 using the `xlim()` function, because those are the plausible values for a student grade.**

Boxplot

Another common task is to compare distributions across groups. A classic example of a visualisation that performs this is the boxplot, accessible via `geom_boxplot()`. It allows for visual comparison of the distribution of two or more groups through their summary statistics.

10. **Create a boxplot of student grades per programme in the `gg_students` dataset you made earlier: map the programme variable to the x position and the grade to the y position. For extra visual aid, you can additionally map the programme variable to the fill aesthetic.**

11. **What do each of the horizontal lines in the boxplot mean? What do the vertical lines (whiskers) mean?**

```
# From the help file of geom_boxplot:

# The middle line indicates the median, the outer horizontal
# lines are the 25th and 75th percentile.

# The upper whisker extends from the hinge to the largest value
# no further than 1.5 * IQR from the hinge (where IQR is the
# inter-quartile range, or distance between the first and third
# quartiles). The lower whisker extends from the hinge to the
# smallest value at most 1.5 * IQR of the hinge. Data beyond
# the end of the whiskers are called "outlying" points and are
# plotted individually.
```

Two densities

12. **Comparison of distributions across categories can also be done by adding a fill aesthetic to the density plot you made earlier. Try this out. To take care of the overlap, you might want to add some transparency in the `geom_density()` function using the `alpha` argument.**

Bar plot

We can display amounts or proportions as a bar plot to compare group sizes of a factor.

---

13. **Create a bar plot of the variable `Years` from the `Hitters` dataset.**

`geom_bar()` automatically transforms variables to counts (see `?stat_count`), similar to how the function `table()` works:

```
table(Hitters$Years)
```

```
##
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 23 24
## 22 25 29 36 30 30 21 16 15 14 10 14 12 13  9  7  7  7  1  2  1  1
```

Line plot

The `Smarket` dataset contains daily return rates and trade volume for 1250 days on the S&P 500 stock market.

---

14. **Use `geom_line()` to make a line plot out of the first 200 observations of the variable `Volume` (the number of trades made on each day) of the `Smarket` dataset. You will need to create a `Day` variable using `mutate()` to map to the x-position. This variable can simply be the integers from 1 to 200. Remember, you can select the first 200 rows using `Smarket[1:200, ]`.**

We can edit properties of the line by adding additional arguments into the geom_line() function.

---

15. **Give the line a nice `colour` and increase its `size`. Also add points of the same colour on top.**

---

16. **Use the function `which.max()` to find out which of the first 200 days has the highest trade volume and use the function `max()` to find out how large this volume was.**

17. **Use `geom_label(aes(x = your_x, y = your_y, label = "Peak volume"))` to add a label to this day. You can use either the values or call the functions. Place the label near the peak!**

This exercise shows that aesthetics can also be mapped separately per geom, in addition to globally in the `ggplot()` function call. Also, the data can be different for different geoms: here the data for geom_label has only a single data point: your chosen location and the "Peak volume" label.

Faceting

18. **Create a data frame called `baseball` based on the `Hitters` dataset. In this data frame, create a factor variable which splits players' salary range into 3 categories. Tip: use the `filter()` function to remove the missing values, and then use the `cut()` function and assign nice `labels` to the categories. In addition, create a variable which indicates the proportion of career hits that was a home run.**

19. **Create a scatter plot where you map `CWalks` to the x position and the proportion you calculated in the previous exercise to the y position. Fix the y axis limits to (0, 0.4) and the x axis to (0, 1600) using `ylim()` and `xlim()`. Add nice x and y axis titles using the `labs()` function. Save the plot as the variable `baseball_plot`.**

20. **Split up this plot into three parts based on the salary range variable you calculated. Use the `facet_wrap()` function for this; look at the examples in the help file for tips.**

Faceting can help interpretation. In this case, we can see that high-salary earners are far away from the point (0, 0) on average, but that there are low-salary earners which are even further away. Faceting should preferably be done using a `factor` variable. The order of the facets is taken from the `levels()` of the factor. Changing the order of the facets can be done using `fct_relevel()` if needed.

Final exercise

---

21. **Create an interesting data visualisation based on the `Carseats` data from the `ISLR` package.**
