Computer exercise 1: Introduction to `ggplot2`

Data Analysis and Visualization / Måns Thulin

Contents

Preparations

In this computer exercise you will get started with the `ggplot2` package for R, allowing you to create good-looking plots using the grammar of graphics.

To begin with, launch RStudio. Throughout the text, there will be code chunks that you can paste into the Console window, or into a script file (use File > New File > R Script to create a script file for this exercise). The benefit of using a script file is that you can save it and return to your code at a later point. Here is an example of what a code chunk can look like:

```
x <- c(1+1, 0, pi)
x
mean(x)
```

If you have not already done so, the next thing you need to do is to install `ggplot2`:

```
install.packages("ggplot2")
```

Alternatively, you can install the `tidyverse` package, of which `ggplot2` is a part.

Next, load the package:

```
library(ggplot2)
```

In this exercise, we will study two datasets that are shipped with the `ggplot2` package:

- `diamonds`: describing the prices of more than 50,000 cut diamonds.
- `msleep`: describing the sleep times of 83 mammals.

These, as well as some other datasets, are automatically loaded when you load `ggplot2`. Before we get started, you should familiarise yourself with these two datasets. This is your task in the first exercise, given below. You can find a solution to to this and other exercises at the end of this document.

Exercise 1

1. View the documentation for the two datasets.
2. Check the data structures: how many observations and variables are there and what type of variables (numeric, categorical, etc.) are there?
3. Compute summary statistics (means, median, min, max, standard deviations, counts for categorical variables). Are there any missing values?

(Click here to go to the solution.)

Continuous data

The three key components to grammar of graphics plots are:

- **Data**: the observations in your dataset,
- **Aesthetics**: mappings from the data to visual properties, and
- **Geoms**: geometric objects, e.g. lines, representing what you see in the plot.

When we create plots using `ggplot2`, we must define what data, aesthetics and geoms to use. To begin with, we will illustrate how this works by visualising some continuous variables in the `msleep` data.

Our first plot

As a first example, let's make a scatterplot by plotting the total sleep time of an animal against the REM sleep time of an animal. The code for doing this using `ggplot2` is:

```
ggplot(msleep, aes(x = sleep_total, y = sleep_rem)) + geom_point()
```

The code consists of three parts:

- **Data**: given by the first argument in the call to `ggplot`: `msleep`
- **Aesthetics**: given by the second argument in the `ggplot` call: `aes`, where we map `sleep_total` to the x-axis and `sleep_rem` to the y-axis.
- **Geoms**: given by `geom_point`, meaning that the observations will be represented by points.

That is, the syntax to create the plot is `ggplot(data, aes) + geom`. All plots created using `ggplot2` follow this pattern. The plus sign is important, as it implies that we can add more geoms to the plot, for instance a trend line, and perhaps other things as well. We will return to that shortly.

Unless the user specifies otherwise, the first two arguments to `aes` will always be mapped to `x` and `y`, meaning that we can simplify the code above by removing the `x =` and `y =` bits (at the cost of a slight reduction in readbility). Moreover, it is considered good style to insert a line break after the `+` sign. The resulting code is:

```
ggplot(msleep, aes(sleep_total, sleep_rem)) +
      geom_point()
```

Note that this does not change the plot in any way - the difference is merely in the style of the code.

Exercise 2

Create a scatterplot with total sleeping time along the x-axis and time awake along the y-axis (using the `msleep` data). What pattern do you see? Can you explain it?

(Click here to go to the solution.)

Colours, shapes and axis labels

You now know how to make scatterplots, but if you plan to show your plot to someone else, there are probably a few changes that you'd like to make. For instance, it's usually a good idea to change the label for the x-axis from the variable name "sleep_total" to something like "Total sleep time (h)". This is done by using the `+` sign again, adding a call to `xlab` to the plot:

```
ggplot(msleep, aes(sleep_total, sleep_rem)) +
      geom_point() +
      xlab("Total sleep time (h)")
```

Note that the plus signs must be placed at the end of a row rather than at the beginning. To change the y-axis label, add `ylab` instead.

To change the colour of the points, you can set the colour in `geom_points`:

```
ggplot(msleep, aes(sleep_total, sleep_rem)) +
      geom_point(colour = "red") +
      xlab("Total sleep time (h)")
```

Alternatively, you may want to use the colours of the point to separate different categories. This is done by adding a `colour` argument to `aes`, since you are now mapping a data variable to a visual property. For instance, we can use the variable `vore` to show differences between herbivores, carnivores and omnivores:

```
ggplot(msleep, aes(sleep_total, sleep_rem, colour = vore)) +
      geom_point() +
      xlab("Total sleep time (h)")
```

What happens if we use a continuous variable, such as the sleep cycle length `sleep_cycle` to set the colour?

```
ggplot(msleep, aes(sleep_total, sleep_rem, colour = sleep_cycle)) +
      geom_point() +
      xlab("Total sleep time (h)")
```

In the next two exercises, you will create scatterplots for the `diamonds` data.

Exercise 3

1. Create a scatterplot with carat along the x-axis and price along the y-axis. Change the x-axis label to read "Weight of the diamond (carat)" and the y-axis label to "Price (USD)". Use `cut` to set the colour of the points.
2. Try adding the argument `alpha = 1` to `geom_points`, i.e. `geom_points(alpha = 1)`. Does anything happen? Try changing the `1` to `0.5` and `0.25` and see how that affects the plot.

(Click here to go to the solution.)

Exercise 4

Similar to how you changed the colour of the points, you can also change their size and shape. The arguments for this are called `size` and `shape`.

1. Change the scatterplot from Exercise 3 so that diamonds with different cut qualities are represented by different shapes[1].
2. Then change it so that the size of each point is determined by the diamond's length (i.e. the variable `x`).

(Click here to go to the solution.)

Axis limits and scales

Next, assume that we wish to study the relationship between animals' brain sizes and their total sleep times. We create a scatterplot using:

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore)) +
      geom_point() +
      xlab("Brain weight") +
      ylab("Total sleep time")
```

There are two animals with brains that are much heavier than the rest (African elephant and Asian elephant). These outliers distort the plot, making it difficult to spot any patterns. We can try changing the x-axis to only go from 0 to 1.5 by adding `xlim` to the plot, to see if that improves it:

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore)) +
      geom_point() +
      xlab("Brain weight") +
```

[1] You may receive a warning message saying that `Using shapes for an ordinal variable is not advised`. This is a good point - shapes, unlike ordinal variables, are not ordered, and so the order of the categories is lost when ordinal variables are represented using shapes. A colour gradient or ordered facetting (introduced on the next page) is probably a better idea.

```
      ylab("Total sleep time") +
      xlim(0, 1.5)
```

This is slightly better, but we still have a lot of points clustered near the y-axis. If instead we wished to change the limits of the y-axis, we would have used `ylim` in the same fashion.

Another option is to resacle the x-axis by applying a log transform to the brain weights, which we can do directly in `aes`:

```
ggplot(msleep, aes(log(brainwt), sleep_total, colour = vore)) +
      geom_point() +
      xlab("log(Brain weight)") +
      ylab("Total sleep time")
```

This is a better-looking scatterplot, with a weak declining trend. We didn't have to remove the outliers (the elephants) to create it, which is good. The downside is that the x-axis now has become difficult to interpret. A third option that mitigates this is to add `scale_x_log10` to the plot, which changes the scale of the x-axis to a $\log_{10}$ scale (which increases interpretability because the values shown at the ticks still are on the original x-scale).

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore)) +
      geom_point() +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

Exercise 5

Using the `msleep` data, create a plot of log-transformed body weight versus log-transformed brain weight. Use total sleep time to set the colours of the points. Change the text on the axes to something informative.

(Click here to go to the solution.)

Comparing groups

We frequently wish to make visuals comparison of different groups. One way to display categorical variables in plots is to use *facetting*, i.e. to create a grid of plots corresponding to the different categories. For instance, in our plot of animal brain weight versus total sleep time, we may wish to separate the different feeding behaviours (omnivores, carnivores, etc.) in the `msleep` data using facetting instead of different coloured points. In `ggplot2` we do this by adding a call to `facet_wrap` to the plot:

```
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10() +
      facet_wrap(~ vore)
```

Note that the x-axes and y-axes of the different plots in the grid all have the same scale and limits.

Exercise 6

1. Using the `diamonds` data, create a scatterplot with carat along the x-axis and price along the y-axis, facetted by `cut`.

2. Read the documentation for `facet_wrap` (`?facet_wrap`). How can you change the number of rows in the plot grid? Create the same plot as in part 1, but with 5 rows.

(Click here to go to the solution.)

Boxplots

Another option for comparing groups is boxplots (also called box-and-whiskers plots). Using `ggplot2`, we create boxplots for animal sleep times, grouped by feeding behaviour, with

```
ggplot(msleep, aes(vore, sleep_total)) +
      geom_boxplot()
```

Note that just as for a scatterplot, the code consists of three parts:

- **Data**: given by the first argument in the call to `ggplot`: `msleep`
- **Aesthetics**: given by the second argument in the `ggplot` call: `aes`, where we map `vore` to the x-axis and `sleep_total` to the y-axis.
- **Geoms**: given by `geom_boxplot`, meaning that the data will be visualised with boxplots.

Exercise 7

1. Using the `diamonds` data, create boxplots of diamond prices, grouped by `cut`.
2. Read the documentation for `geom_boxplot`. How can you change the colours of the boxes and their outlines?
3. Replace `cut` by `reorder(cut, price)` in the plot's aestethics. What does `reorder` do? What is the result?
4. Add `geom_jitter(size = 0.1, alpha = 0.2)` to the plot. What happens?

(Click here to go to the solution.)

Histograms

To show the distribution of a continuous variable, we can use a histogram, in which the data is split into a number of bins and the number of observations in each bin is shown by a bar. The `ggplot2` code for histograms follows the same pattern as other plots:

```
ggplot(msleep, aes(sleep_total)) +
      geom_histogram()
```

As before, the three parts are:

- **Data**: given by the first argument in the call to `ggplot`: `msleep`
- **Aesthetics**: given by the second argument in the `ggplot` call: `aes`, where we map `sleep_total` to the x-axis.
- **Geoms**: given by `geom_histogram`, meaning that the data will be visualised by a histogram.

Exercise 8

1. Using the `diamonds` data, create a histogram of diamond prices.
2. Create histograms of diamond prices for different cuts, using facetting.

3. Add a suitable argument to `geom_histogram` to add black outlines around the bars[2].

(Click here to go to the solution.)

Categorical data

When visualising categorical data, we typically try to show the counts, i.e. the number of observations, for each category. The most common plot for this type of data is the bar chart.

Bar charts

Bar charts are discrete analogues to histograms, where the category counts are represented by bars. The code for creating them is:

```
ggplot(msleep, aes(vore)) +
      geom_bar()
```

As always, the three parts are:

- **Data**: given by the first argument in the call to `ggplot`: `msleep`
- **Aesthetics**: given by the second argument in the `ggplot` call: `aes`, where we map `vore` to the x-axis.
- **Geoms**: given by `geom_bar`, meaning that the data will be visualised by a bar chart.

Exercise 9

1. Using the `diamonds` data, create a bar chart of cuts.
2. Add different colours to the bars by adding a `fill` argument to `geom_bar`.
3. Check the documentation for `geom_bar`. How can you decrease the width of the bars?
4. Return to the code you used for part 1. Add `fill = clarity` to the `aes`. What happens?
5. Next, add `position = "dodge"` to `geom_bar`. What happens?
6. Return to the code you used for part 1. Add `coord_flip()` to the plot. What happens?

(Click here to go to the solution.)

Saving your plot

When you create a `ggplot2` plot, you can save it as a plot object in R:

```
myPlot <- ggplot(msleep, aes(sleep_total, sleep_rem)) +
      geom_point()
```

To plot a saved plot object, use

```
print(myPlot)
```

If you like, you can add things to the plot, just as before:

```
print(myPlot + xlab("I forgot to add a label!"))
```

[2]Personally, I don't understand why anyone would ever plot histograms without outlines!

To save your plot object as an image file, use `ggsave`. The `width` and `height` arguments allows us to control the size of the figure (in inches, unless you specify otherwise using the `units` argument).

```
ggsave("filename.pdf", myPlot, width = 5, height = 5)
```

In addition to pdf, you can save image files e.g. as jpg, tif, eps, svg and png files.

Exercise 10

1. Create a plot object and save it as a 4 by 4 inch png file.
2. When printing images, you may want to increase their resolution. Check the documentation for `ggsave`. How can you increase the resolution of your png file to 600 dpi?

(Click here to go to the solution.)

Preparations for the next computer exercise

In the next computer exercise, we will use some datasets and functions from the `nlme`, `plotly`, `forecast` and `fpp2` packages. Ahead of the next computer exercise, you should therefore install them:

```
install.packages(c("nlme", "plotly", "forecast", "fpp2"), dependencies = TRUE)
```

Solutions

Exercise 1

Code for examining the `diamonds` dataset is provided below. To examine the other dataset, simply replace `diamonds` with `msleep`.

```
# View the documentation, where the data is described.
?diamonds

# Have a look at the first ten rows of the data:
diamonds
# diamonds is a tibble, in which case the above code yields the first ten rows.
# If it were a data frame, we would use the following code instead:
head(diamonds, n = 10)

# Have a look at the structure of the data:
str(diamonds)
# This shows you the number of observations (53,940) and variables (10) and the
# variable types (carat is numeric, cut is an ordered factor, and so on.)

# Summary statistics:
summary(diamonds)
# In the summary, missing values show up as NA's. There are no NA's here, and
# hence no missing values.
```

(Click here to return to the exercise.)

Exercise 2

```
ggplot(msleep, aes(sleep_total, awake)) +
      geom_point()
```

The points follow a declining line. The reason for this is that at any given time, an animal is either awake or asleep, so the total sleep time plus the awake time is always 24 hours for all animals. Consequently, the points lie on the line given by `awake=24-sleep_total`.

(Click here to return to the exercise.)

Exercise 3

```
ggplot(diamonds, aes(carat, price, colour = cut)) +
      geom_point() +
      xlab("Weight of diamond (carat)") +
      ylab("Price (USD)")
```

We can change the opacity of the points by adding an `alpha` argument to `geom_point`. This is useful when the plot contains overlapping points:

```
ggplot(diamonds, aes(carat, price, colour = cut)) +
      geom_point(alpha = 0.25) +
      xlab("Weight of diamond (carat)") +
      ylab("Price (USD)")
```

(Click here to return to the exercise.)

Exercise 4

To set different shapes for different values of `cut` we use:

```
ggplot(diamonds, aes(carat, price, colour = cut, shape = cut)) +
      geom_point(alpha = 0.25) +
      xlab("Weight of diamond (carat)") +
      ylab("Price (USD)")
```

We can then change the size of the points as follows. The resulting figure is unfortunately not that informative in this case.

```
ggplot(diamonds, aes(carat, price, colour = cut, shape = cut, size = x)) +
      geom_point(alpha = 0.25) +
      xlab("Weight of diamond (carat)") +
      ylab("Price (USD)")
```

(Click here to return to the exercise.)

Exercise 5

Using the `scale_axis_log10` options:

```
ggplot(msleep, aes(bodywt, brainwt, colour = sleep_total)) +
      geom_point() +
      xlab("Body weight (logarithmic scale)") +
      ylab("Brain weight (logarithmic scale)") +
      scale_x_log10() +
      scale_y_log10()
```

(Click here to return to the exercise.)

Exercise 6

We use `facet_wrap(~ cut)` to create the facetting:

```
ggplot(diamonds, aes(carat, price)) +
      geom_point() +
      facet_wrap(~ cut)
```

To set the number of rows, we add an `nrow` argument to `facet_wrap`:

```
ggplot(diamonds, aes(carat, price)) +
      geom_point() +
      facet_wrap(~ cut, nrow = 5)
```

(Click here to return to the exercise.)

Exercise 7

```
ggplot(diamonds, aes(cut, price)) +
      geom_boxplot()
```

To change the colours of the boxes, we add `colour` (outline colour) and `fill` (box colour) arguments to `geom_boxplot`:

```
ggplot(diamonds, aes(cut, price)) +
      geom_boxplot(colour = "magenta", fill = "turquoise")
```

(No, I don't really recommend using this particular combination of colours.)

`reorder(cut, price)` changes the order of the `cut` categories based on their `price` values.

```
ggplot(diamonds, aes(reorder(cut, price), price)) +
      geom_boxplot(colour = "magenta", fill = "turquoise")
```

`geom_jitter` can be used to plot the individual observations on top of the histogram. Because there are so many observations in this dataset, we must set a small size and a low alpha in order not to cover the boxes completely.

```
ggplot(diamonds, aes(reorder(cut, price), price)) +
      geom_boxplot(colour = "magenta", fill = "turquoise") +
      geom_jitter(size = 0.1, alpha = 0.2)
```

(Click here to return to the exercise.)

Exercise 8

```
ggplot(diamonds, aes(price)) +
      geom_histogram()
```

Next, we facet the histograms using `cut`:

```
ggplot(diamonds, aes(price)) +
      geom_histogram() +
      facet_wrap(~ cut)
```

Finally, by reading the documentation `?geom_histogram` we find that we can add outlines using the `colour` argument:

```
ggplot(diamonds, aes(price)) +
      geom_histogram(colour = "black") +
      facet_wrap(~ cut)
```

(Click here to return to the exercise.)

Exercise 9

```
ggplot(diamonds, aes(cut)) +
      geom_bar()
```

To set different colours for the bars, we can use `fill`:

```
ggplot(diamonds, aes(cut)) +
      geom_bar(fill = c("red", "yellow", "blue", "green", "purple"))
```

`width` lets us control the bar width:

```
ggplot(diamonds, aes(cut)) +
      geom_bar(fill = c("red", "yellow", "blue", "green", "purple"), width = 0.5)
```

By adding `fill = clarity` to `aes` we create a stacked bar chart:

```
ggplot(diamonds, aes(cut, fill = clarity)) +
      geom_bar()
```

By adding `position = "dodge"` to `geom_bar` we obtain a grouped bar chart:

```
ggplot(diamonds, aes(cut, fill = clarity)) +
      geom_bar(position = "dodge")
```

`coord_flip` flips the coordinate system, yielding a horizontal bar plot:

```
ggplot(diamonds, aes(cut)) +
      geom_bar() +
      coord_flip()
```

(Click here to return to the exercise.)

Exercise 10

To save the png file, use

```
myPlot <- ggplot(msleep, aes(sleep_total, sleep_rem)) +
      geom_point()

ggsave("filename.png", myPlot, width = 4, height = 4)
```

To change the resolution, we use the `dpi` argument:

```
ggsave("filename.png", myPlot, width = 4, height = 4, dpi=600)
```

(Click here to return to the exercise.)