Computer exercise 4: Exploratory data analysis, part II

Data Analysis and Visualization / Måns Thulin

Contents

Preparations

In this computer exercise, we will focus on multivariate exploratory data analysis, meaning that we will study relationships and interactions between variables.

We start by loading `ggplot2`:

```
library(ggplot2)
```

In this exercise, we will study the following datasets from the `ggplot2`, `gapminder` and `nycflights13` packages:

- `diamonds`: describing the prices of more than 50,000 cut diamonds; from `ggplot2`.
- `msleep`: describing the sleep times of 83 mammals; from `ggplot2`.
- `gapminder`: describing population and economic growth in different countries; from `gapminder`.
- `planes`: describing various properties of airplanes; from `nycflights13`.

Visualising multiple variables

Scatterplot matrices

When we have a large enough number of numeric variables in our data, plotting scatterplots of all pairs of variables becomes tedious. Luckily there are some R functions that speed up this process.

The `GGally` package is an extension to `ggplot2` which contains several functions for plotting multivariate data. They work similar to the `autoplot` functions that we have used in previous exercises. One of these is `ggpairs(data)`, which creates a scatterplot matrix, that is, a grid with scatterplots of all pairs of variables in `data`. In addition, it also plots density estimates (along the diagonal) and shows the (Pearson) correlation for each pair.

If we only want to include some of the variables in a dataset, we can do so by providing a vector with variable names. Here is an example for the animal sleep data `msleep`:

```
library(GGally)
ggpairs(msleep[, c("sleep_total", "sleep_rem", "sleep_cycle", "awake",
                   "brainwt", "bodywt")])
```

Optionally, if we wish to create a scatterplot involving all numeric variables, we can replace the vector with variable names with some R code that extracts the columns containing numeric variables:

```
ggpairs(msleep[, which(sapply(msleep, class) == "numeric")])
```

The resulting plot is identical to the previous one, because the list of names contained all numeric variables. The grab-all-numeric-variables approach is often convenient, because we don't have to write all the variable names. On the other hand, it's not very helpful in case we only want to include some of the numeric variables.

If we include a categorical variable in the list of variables (such as the feeding behaviour `vore`), the matrix will include a bar plot of the categorical variable as well as boxplots and facetted histograms to show differences between different categories in the continuous variables:

```
ggpairs(msleep[, c("vore", "sleep_total", "sleep_rem", "sleep_cycle",
                   "awake", "brainwt", "bodywt")])
```

Alternatively, we can use a categorical variable to colour points and density estimates using `aes(colour = ...)`. The syntax for this is follows the same pattern as that for a standard `ggplot` call - `ggpairs(data, aes)`. The only exception is that if the categorical variable is not included in the `data` argument, we much specify which data frame it belongs to:

```
ggpairs(msleep[, c("sleep_total", "sleep_rem", "sleep_cycle", "awake",
                   "brainwt", "bodywt")],
        aes(colour = msleep$vore, alpha = 0.5))
```

Exercise 1

Create a scatterplot matrix for all numeric variables in `diamonds`. Differentiate different cuts by colour. Add a suitable title to the plot. (`diamonds` is a fairly large dataset and it may take a minute or so for R to create the plot.)

(Click here to go to the solution.)

Correlograms

Scatterplot matrices are not a good choice when we have too many variables, partially because the plot window needs to be very large to fit all variables and partially because it becomes difficult to get a good overview of the data. In such cases, a correlogram, where the strength of the correlation between each pair of variables is plotted instead of scatterplots, can be used instead. It is effectively a visualisation of the correlation matrix of the data, where the strength and sign of the correlations are represented by different colours.

The `GGally` package contains the function `ggcorr`, which can be used to create a correlogram:

```
library(GGally)

ggcorr(msleep[, c("sleep_total", "sleep_rem", "sleep_cycle", "awake",
                  "brainwt", "bodywt")])
```

Exercise 2

Using the `diamonds` dataset and the documentation for `ggcorr`, do the following:

1. Create a correlogram for all numeric variables in the dataset.
2. Change the type of correlation used to create the plot to the Spearman correlation.
3. Change the colour scale from a categorical scale with 5 categories.
4. Change the colours on the scale to go from yellow (low correlation) to black (high correlation).

(Click here to go to the solution.)

Adding more variables to scatterplots

We have already seen how scatterplots can be used to visualise two continuous and one categorical variable, by plotting the two continuous variables against each other and using the categorical variable to set the colours of the points. There are however more ways we can incorporate information about additional variables into a scatterplot.

So far, we have set three aesthethics in our scatterplots: `x`, `y` and `colour`. Two other important aesthetics are `shape` and `size` which, as you'd expect, allow us to control the shape and size of the points. As a first example using the `msleep` data, we use feeding behaviour (`vore`) to set the shapes used for the points:

```
ggplot(msleep, aes(brainwt, sleep_total, shape = vore)) +
      geom_point() +
      scale_x_log10()
```

The plot looks a little nicer if we increase the `size` of the points:

```
ggplot(msleep, aes(brainwt, sleep_total, shape = vore, size = 2)) +
      geom_point() +
      scale_x_log10()
```

Another option is to let `size` represent a continuous variable, in what is known as a bubble plot:

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore, size = bodywt)) +
      geom_point() +
      scale_x_log10()
```

Because some animals are much heavier (i.e. have higher `bodywt` values) than most others, almost all points are quite small. There are a couple of things we can do to remedy this. First, we can transform `bodywt`, e.g. using the square root transformation `sqrt(bodywt)`, to decrease the differences between large and small animals. This can be done by adding `scale_size(trans = "sqrt")` to the plot. Second, we can also use `scale_size` to control the range of point sizes (e.g. from size 1 to size 20). This will cause some points to overlap, so we add `alpha = 0.5` to the geom, to make the points transparent.

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore, size = bodywt)) +
      geom_point(alpha = 0.5) +
      scale_x_log10() +
      scale_size(range = c(1, 20), trans = "sqrt")
```

This produces a fairly nice-looking plot, but it'd look even better if we changed the axes labels and legend texts. We can change the legend text for the size scale by adding the argument `name` to `scale_size`. Including a `\n` in the text lets us create a line break. Similarly, we can use `scale_color_discrete` to change the legend text for the colours:

```
ggplot(msleep, aes(brainwt, sleep_total, colour = vore, size = bodywt)) +
      geom_point(alpha = 0.5) +
      xlab("log(Brain weight)") +
      ylab("Sleep total (h)") +
      scale_x_log10() +
      scale_size(range = c(1, 20), trans = "sqrt", name = "Square root of\nbody weight") +
      scale_color_discrete(name = "Feeding behaviour")
```

Exercise 3

Using the bubble plot created above, do the following:

1. Load the `hrbrthemes` package and add `theme_ipsum()` to the plot. What happens?
2. Replace `colour = vore` in the `aes` by `fill = vore` and add `colour = "black", shape = 21` to `geom_point`. What happens?
3. Use `ggplotly` to create an interactive version of the bubble plot above, where variable information and the animal name are displayed when you hover a point.

(Click here to go to the solution.)

Overplotting

Let's make a scatterplot of `table` versus `depth` based on the `diamonds` datset:

```
ggplot(diamonds, aes(table, depth)) +
      geom_point()
```

This plot is cluttered. There are too many points, which makes it difficult to see if, for instance, high `table` values are more common than low `table` values. In this section, we'll look at some ways to deal with this problem, known as overplotting.

The first thing we can try is to decrease the point size:

```
ggplot(diamonds, aes(table, depth)) +
      geom_point(size = 0.1)
```

This helps a little, but now the outliers become a bit difficult to spot. We can try changing the opacity using `alpha` instead:

```
ggplot(diamonds, aes(table, depth)) +
      geom_point(alpha = 0.2)
```

This is also better than the original plot, but neither plot is great. Instead of plotting each individual point, maybe we can try plotting the counts or densities in different regions of the plot instead? Effectively, this would be a 2D version of a histogram. There are several ways of doing this in `ggplot2`.

First, we bin the points and count the numbers in each bin, using `geom_bin2d`:

```
ggplot(diamonds, aes(table, depth)) +
      geom_bin2d()
```

By default, `geom_bin2d` uses 30 bins. Increasing that number can sometimes give us a better idea about the distribution of the data:

```
ggplot(diamonds, aes(table, depth)) +
      geom_bin2d(bins = 50)
```

If you prefer, you can get a similar plot with hexagonal bins by using `geom_hex` instead:

```
ggplot(diamonds, aes(table, depth)) +
      geom_hex(bins = 50)
```

As an alternative to bin counts, we could create a 2-dimensional density estimate and create a contour plot showing the levels of the density:

```
ggplot(diamonds, aes(table, depth)) +
      stat_density_2d(aes(fill = ..level..), geom = "polygon", colour="white")
```

The `fill = ..level..` bit above probably looks a little strange to you. It means that an internal function (the level of the contours) is used to choose the fill colours.

We can use a similar approach to show a summary statistic for a third variable in a plot. For instance, we may want to plot the average price as a function of `table` and `depth`. This is called a tile plot:

```
ggplot(diamonds, aes(table, depth, z = price)) +
      geom_tile(binwidth = 1, stat = "summary_2d", fun = mean) +
      ggtitle("Mean prices for diamonds with different depths and tables")
```

Exercise 4

1. Create a tile plot of table versus depth, showing the highest price for a diamond in each bin.
2. Create a bin plot of `carat` versus `price`. What type of diamonds have the highest bin counts?

(Click here to go to the solution.)

Categorical data

When visualising a pair of categorical variables, plots similar to those in the previous section prove to be useful. One way of doing this is to use the `geom_count` geom. We illustrate this with an example using `diamonds`, showing how common different combinations of colour and cuts are:

```
ggplot(diamonds, aes(color, cut)) +
      geom_count()
```

However, it is often better to use colour rather than point size to visualise counts, which we can do using a tile plot. First we have to compute the counts though:

```
# We use count from the dplyr package to count how many occurences there are
# of different combinations of colours and cut
library(dplyr)
diamonds2 <- diamonds %>% count(color, cut)
# The count is saved in diamonds2 as a new variable called n

# Then, we plot the counts using geom_tile
ggplot(diamonds2, aes(color, cut, fill = n)) +
      geom_tile()
```

It is also possible to combine point size and colours:

```
ggplot(diamonds2, aes(color, cut, colour = n, size = n)) +
      geom_count()
```

Exercise 5

Using the `diamonds` dataset, do the following:

1. Use a plot to find out what the most common combination of cut and clarity is.
2. Use a plot to find out which combination of cut and clarity that has the highest average price.

(Click here to go to the solution.)

Putting it all together

In the final two exercise, you will investigate the `gapminder` and `planes` datasets. First, load the corresponding libraries and have a look at the documentation for each dataset:

```
library(gapminder)
?gapminder

library(nycflights13)
?planes
```

Exercise 6

Do the following using the `gapminder` dataset:

1. Create a scatterplot matrix showing life expectancy, population and GDP per capita for all countries, using the data from the year 2007. Use colours to differentiate countries from different continents.
2. Create an interactive bubble plot, showing information about each country when you hover the points. Use data from the year 2007. Put log(GDP per capita) on the x-axis and life expetancy on the y-axis. Let population determine point size. Plot each country in a different colour and facet by continent. Tip: the `gapminder` package provides a pretty color scheme for different countries, called `country_colors`. You can use that scheme by adding `scale_colour_manual(values = country_colors)` to your plot.

Nice, you just visualised 5 variables in a facetted bubble plot! (Click here to go to the solution.)

Exercise 7

Use graphics to answer the following questions regarding the `planes` dataset:

1. What is the most common combination of manufacturer and plane type in the dataset?
2. Which combination of manufacturer and plane type has the highest average number of seats?
3. Does the number of seats on planes change over time? Which plane had the highest number of seats?
4. Does the type of engine used change over time?

(Click here to go to the solution.)

Solutions

Exercise 1

As for all `ggplot2` plots, we can use `ggtitle` to add a title to the plot:

```
ggpairs(diamonds[, which(sapply(diamonds, class) == "numeric")],
        aes(colour = diamonds$cut, alpha = 0.5)) +
      ggtitle("Numeric variables in the diamonds dataset")
```

(Click here to return to the exercise.)

Exercise 2

We create the correlogram using `ggcorr` as follows:

```
ggcorr(diamonds[, which(sapply(diamonds, class) == "numeric")])
```

`method` allows us to control which correlation coefficient to use:

```
ggcorr(diamonds[, which(sapply(diamonds, class) == "numeric")],
       method = c("pairwise", "spearman"))
```

`nbreaks` is used to create a categorical colour scale:

```
ggcorr(diamonds[, which(sapply(diamonds, class) == "numeric")],
       method = c("pairwise", "spearman"),
       nbreaks = 5)
```

`low` and `high` can be used to control the colours at the endpoints of the scale:

```
ggcorr(diamonds[, which(sapply(diamonds, class) == "numeric")],
       method = c("pairwise", "spearman"),
       nbreaks = 5,
       low = "yellow", high = "black")
```

(Yes, the default colours are a better choice!)

(Click here to return to the exercise.)

Exercise 3

Adding `theme_ipsum()` to the plot changes the theme used for the figure. Some important differences to the standard `ggplot2` theme is that the background is white and that different default fonts are used:

```
library(hrbrthemes)
ggplot(msleep, aes(brainwt, sleep_total, colour = vore, size = bodywt)) +
      geom_point(alpha = 0.5) +
      xlab("log(Brain weight)") +
      ylab("Sleep total (h)") +
      scale_x_log10() +
      scale_size(range = c(1, 20), trans = "sqrt", name = "Square root of\nbody weight") +
      scale_color_discrete(name = "Feeding behaviour") +
      theme_ipsum()
```

Next, we replace `colour = vore` in the `aes` by `fill = vore` and add `colour = "black", shape = 21` to `geom_point`. The points now get black borders, which makes them a bit sharper:

```
library(hrbrthemes)
ggplot(msleep, aes(brainwt, sleep_total, fill = vore, size = bodywt)) +
      geom_point(alpha = 0.5, colour = "black", shape = 21) +
      xlab("log(Brain weight)") +
      ylab("Sleep total (h)") +
      scale_x_log10() +
      scale_size(range = c(1, 20), trans = "sqrt", name = "Square root of\nbody weight") +
      scale_color_discrete(name = "Feeding behaviour") +
      theme_ipsum()
```

Finally, we can use `ggplotly` to create an interactive version of the plot. Adding `text` to the `aes` allows us to include more information when hovering points:

```
library(plotly)
myPlot <- ggplot(msleep, aes(brainwt, sleep_total, fill = vore, size = bodywt,
                             text = name)) +
      geom_point(alpha = 0.5, colour = "black", shape = 21) +
      xlab("log(Brain weight)") +
      ylab("Sleep total (h)") +
      scale_x_log10() +
      scale_size(range = c(1, 20), trans = "sqrt", name = "Square root of\nbody weight") +
      scale_color_discrete(name = "Feeding behaviour") +
      theme_ipsum()

ggplotly(myPlot)
```

(Click here to return to the exercise.)

Exercise 4

We create the tile plot using `geom_tile`. By setting `fun = max` we obtain the highest price in each bin:

```
ggplot(diamonds, aes(table, depth, z = price)) +
      geom_tile(binwidth = 1, stat = "summary_2d", fun = max) +
      ggtitle("Highest prices for diamonds with different depths and tables")
```

We can create the bin plot using either `geom_bin2d` or `geom_hex`:

```
ggplot(diamonds, aes(carat, price)) +
      geom_bin2d(bins = 50)
```

Diamonds with carat around 0.3 and price around 1000 have the highest bin counts.

(Click here to return to the exercise.)

Exercise 5

VS2 and Ideal is the most common combination:

```
diamonds2 <- diamonds %>% count(clarity, cut)
ggplot(diamonds2, aes(clarity, cut, fill = n)) +
      geom_tile()
```

As for continuous variables, we can use `geom_tile` with the arguments `stat = "summary_2d", fun = mean` to display the average prices for different combinations. SI2 and Premium is the combination with the highest average price:

```
ggplot(diamonds, aes(clarity, cut, z = price)) +
      geom_tile(binwidth = 1, stat = "summary_2d", fun = mean) +
      ggtitle("Mean prices for diamonds with different clarities and cuts")
```

(Click here to return to the exercise.)

Exercise 6

We create the scatterplot using:

```
gapminder2007 <- filter(gapminder, year == 2007)

ggpairs(gapminder2007[, c("lifeExp", "pop", "gdpPercap")],
        aes(colour = gapminder2007$continent, alpha = 0.5))
```

The interactive facetted bubble plot is created using:

```
library(plotly)

gapminder2007 <- filter(gapminder, year == 2007)

myPlot <- ggplot(gapminder2007, aes(gdpPercap, lifeExp, size = pop, colour = country)) +
      geom_point(alpha = 0.5) +
      scale_x_log10() +
      scale_size(range = c(2, 15)) +
      scale_colour_manual(values = country_colors) +
      theme(legend.position = "none") +
      facet_wrap(~ continent)

ggplotly(myPlot)
```

(Click here to return to the exercise.)

Exercise 7

Fixed wing multi engine Boeings are the most common planes:

```
planes2 <- planes %>% count(type, manufacturer)
ggplot(planes2, aes(type, manufacturer, fill = n)) +
      geom_tile()
```

The fixed wing multi engine Airbus has the highest average number of seats:

```
ggplot(planes, aes(type, manufacturer, z = seats)) +
      geom_tile(binwidth = 1, stat = "summary_2d", fun = mean) +
      ggtitle("Number of seats for different planes")
```

The number of seats seems to have increased in the 1980's, and then reached a plateau:

```
ggplot(planes, aes(year, seats)) +
      geom_point(aes(colour = engine)) +
      geom_smooth()
```

The plane with the largest number of seats is not an Airbus, but a Boeing 747-451. It can be found using `planes[which.max(planes$seats),]` or visually using `plotly`:

```
myPlot <- ggplot(planes, aes(year, seats, text = paste("Tail number:", tailnum,
                                                       "<br>Manufacturer:",
                                                       manufacturer))) +
      geom_point(aes(colour = engine)) +
      geom_smooth()

ggplotly(myPlot)
```

Finally, we can investigate what engines were used during different time periods in several ways, for instance by differentiating engines by colour in our previous plot:

```
ggplot(planes, aes(year, seats)) +
      geom_point(aes(colour = engine)) +
      geom_smooth()
```

(Click here to return to the exercise.)