Plotting with ggplot2

David García Heredia

2019-12-01

1 Intro to ggplot2

1.1 What is ggplot2?

`Ggplot2` is an `R` library designed to create statistical graphics which, opposite to the basic plot-functions provided in `R`, allows us to have more control of our graphs. That is, we can manipulate almost any feature of a plot to get the best visual representation for our problem. In addition to that, with a small amount of effort, we will make really beautiful plots which will enhance the presentation of our results.

Note: Have in mind that the goal in visualization is not to make beautiful pictures, but to help us to convey our results in a more simple and straightforward way.

1.2 How does ggplot2 work?

The strength of `ggplot2` lies in the fact that it defines each plot as a collection of **independent layers** which are later combined to produce the desired graph. This allows us, not only to use a set of pre-specified graphics, but also to create our own graphs to better describe the problem that we are facing.

The layers which form any plot are:

- The **data** that you want to visualize and a **mapping** of it to the variables that will form the plot (e.g.: what column/variable of the data frame corresponds to the x-axis and which one to the y-axis).
- The **geometric** objects that you will see in the plot: points, lines, polygons...
- The **statistical** transformations that are used to summarize the data (e.g.: counting observations to make a histogram).
- The **scales** or visual features of the plot (e.g.: color, size, shape...).
- The **coordinate** system: Cartesian, polar, map projection.
- **Faceting**: specifications on how to break up the data into subsets and how to display them, e.g.: make 5 plots, 1 per continent, showing the historical evolution of their Gross Domestic Product (GDP). In the example, continents would be the splitting specification. Notice that we will not always want to make a partition of the data set so, in those cases, we will not define this layer.

To better understand the idea of the layers, let's check the following example which will also help us to show the kind of plots that we will be able to do by the end of this document.

1.2.1 Example

Let's start loading the `ggplot2` library and a data set called "diamond".

```
library(ggplot2)

set.seed(1410) # Fix a random seed to make the experiment reproducible
```

```r
dsmall <- diamonds[sample(nrow(diamonds), 1000), ] # Select a subset of 1K observations

head(dsmall) # To check the info in the data set
```

```
## # A tibble: 6 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  1.35 Ideal   J     VS2      61.4    57  5862  7.1   7.13  4.37
## 2  0.3  Good    G     VVS1     64      57   678  4.23  4.27  2.72
## 3  0.75 Ideal   F     SI2      59.2    60  2248  5.87  5.92  3.49
## 4  0.26 Ideal   F     VS1      60.9    57   580  4.13  4.11  2.51
## 5  0.33 Premium H     VVS1     61.4    59   752  4.42  4.44  2.72
## 6  1.52 Ideal   G     VVS1     62.4    55 15959  7.3   7.39  4.58
```

The meaning of each of the variables of the data set is:

- *Carat*: Weight of the diamond. **Continuous variable**.
- *Cut*: Quality of the cut (Fair, Good, Very Good, Premium, Ideal). **Categorical variable**.
- *Color*: Diamond color, from D (best) to J (worst). **Categorical variable**.
- *Clarity*: A measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)). **Categorical variable**.
- *Price*: Price in US Dollars of the diamond. Although just integer numbers are collected, it makes more sense to treat it as a **continuous variable**.

As we will not use the rest of the variables showed in the table, we skip their description (which can be found on the Internet).

Now, imagine that we want to make the following plot (pay attention to the words in bold): We want, for **each type of cut**, a **scatter plot** to visualize the **price** of the diamonds according to their **weight**. Furthermore, depending on the variable `color` (see above in the table), we also want to **assign a color** to each observation. Let's check what would be the layers of the plot in this example:

- The **data** will be the set `dsmall`.
- The **mapping** will be the assignation of variable x to the column `carat`, and variable y to the column `price` (check below in the code).
- The **geometric** object will be points (we want a scatter plot).
- In this case, there will be no **statistical** layer.
- The color rule for the points will make the **scale** layer.
- We will use the default **coordinate** system: Cartesian.
- Variable `cut` will define our **faceting** layer: we want to split the data so we make a plot for each type of cut.

The code to make the desired plot is the following one (notice how we use + to add the layers):

```r
ggplot(dsmall, aes(x = carat, y = price, color = color)) + # data,
                                                           # mapping (x, y) and
                                                           # scale (color) layers.
  facet_wrap(~ cut, ncol = 3) + # faceting layer.
  geom_point(alpha = 0.5) + # geometric layer
                            # alpha is the transparency, so it is part of the scale layer.
  theme(legend.position = "bottom")
```

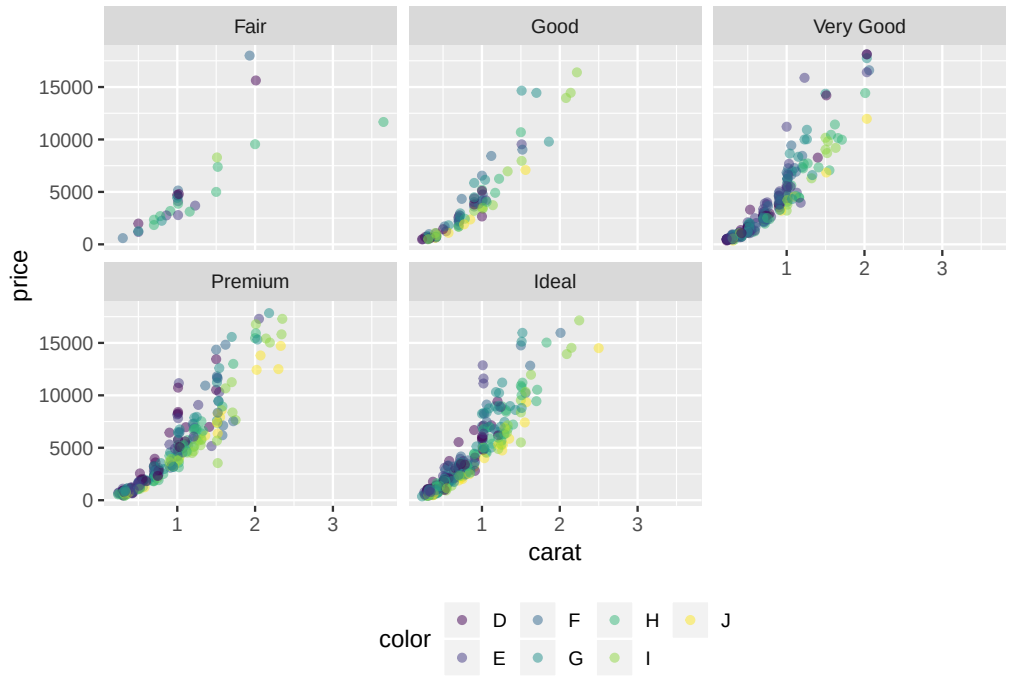

Figure 1: Our first plot with ggplot2.

Note: It is quite common to define the mapping layer inside the `ggplot()` function. However, this can also be defined inside the `geom()` functions. This is useful if, for instance, we want to add different `geom` layers, each of them representing different information.

Preliminary conclusion from the plot? It does not seem to exist a relationship between the quality of the cut and the price, i.e.: for the same weight value, prices are similar for the different cut qualities. **However**, to actually draw a conclusion, we would have to use the statistical techniques learned during the course. Have in mind that plots are great to get an idea of the situation and to convey results, **but they are not maths**. So for instance, if the relationship between variables was linear, we could use Linear Regression techniques to study the significance of the variable `cut`.

1.3 Notes about the color scales

Before showing how to use `ggplot2`, it is important to make a few comments about what colors to use in our plots. When using a color scale, we should be aware of using one that: 1) Shows uniformity in the color gradient, 2) It does not lead to misinterpretation of results when printed in black-and-white, and 3) If possible, it is colorblind-friendly.

Classic color scales, like the ones shown in Figure 2, do not respect the abovementioned principles, so they should be avoided. For instance, when a heatmap that uses `jet.colors` is printed in black-and-white, it will produce a misinterpreation of the results because it is impossible to distinguish blue and red areas!

To solve these problems, there is a package called `viridisLite` which contains different color scales having the three mentioned properties (see in Figure 3). Special mention to the color scale Cividis, which is specifically aimed at people with color vision deficiency.

As a conclusion of all this, the color scales that we will be using in this document are those of the `viridisLite` package.

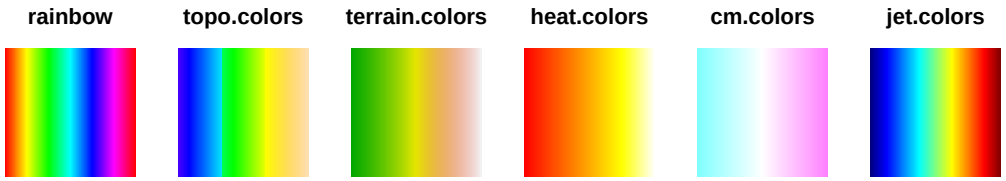

Figure 2: Color scales to avoid.

Figure 3: Recommended color scales.

2 Hands on ggplot2

Now that we know what is `ggplot2` and that it works through **layers**, let's start making some plots and learning how to modify different features of them. For that, we will start studying, using a box plot, the spread in diamonds' price according to their cut quality.

```
# Notice how we can save a plot in a variable like if it was a number!
plot1 <- ggplot(dsmall, aes(x = cut, y = price)) # data and mapping
plot1 <- plot1 + geom_boxplot() # statistical layer
print(plot1)
```

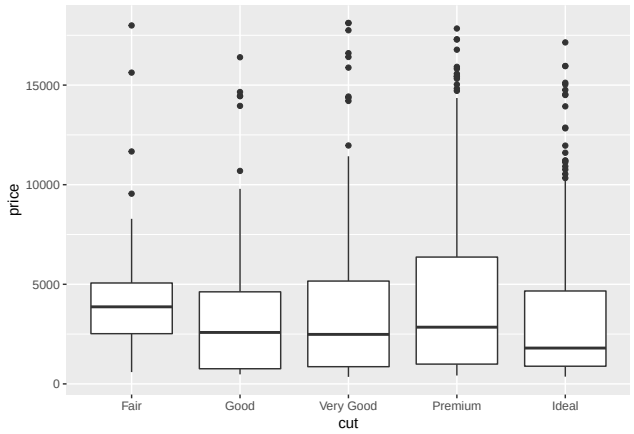

Figure 4: Box plots with ggplot2.

As you can see, just with two lines of code we got to produce the desired plot, but with the advantage of making it look great. Nevertheless, we might want to modify that plot in different ways, e.g.: adding color, a title, a legend, changing the background, etc. Let's see how to do that.

Note: Almost anything you want to do to a plot can be done with `ggplot2`. So, whenever you want to do something not explained in this manual, just check on the Internet and you will find how to do it. For example, the boxplots of Figure 4 can be reordered in ascending order using function `reorder()` to make the trend easier to see. Check on the Internet how!

2.1 Adding colors

2.1.1 Changing the color of the borders

```
library(viridisLite)
plot2 <- ggplot(dsmall, aes(x = cut, y = price))
plot2 <- plot2 + geom_boxplot(aes(color = cut)) # one color for each boxplot
plot2 <- plot2 + scale_color_viridis_d(option = "magma") # using magma color scale
print(plot2)
```

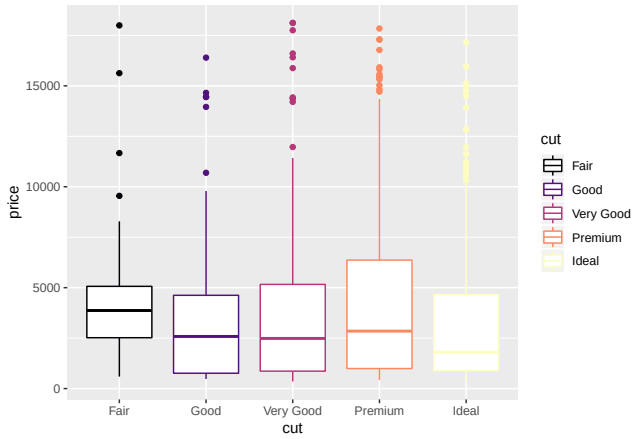

Figure 5: Changing the color of the box plots.

As we are coloring according to a discrete value (the `cut`), we use the function `scale_color_viridis_d()`. For continuous values, we would use `scale_color_viridis_c()` (the last letter changes: "d" for discrete and "c" for continuous). Notice how, when coloring, a legend is automatically added.

2.1.2 Filling the box plots

```
plot3 <- ggplot(dsmall, aes(x = cut, y = price))
plot3 <- plot3 + geom_boxplot(aes(fill = cut))
plot3 <- plot3 + scale_fill_viridis_d(alpha = 0.6) # adding also transparency
print(plot3)
```

Now, as we are filling the box plots, we do not use `scale_color_viridis_d()`, but `scale_fill_viridis_d()` (we changed the word *color* by *fill*).

Note: This way of coloring taking into account if we want just to color the borders or fill the plot, and if the variables are discrete or continuous, might seem a little bit confusing at the beginning, but it is extremely powerful and it is not difficult to get used to it.

2.1.3 Changing the theme

In `ggplot2` there exist predefined themes that establish some of the aesthetic properties of the final plot, e.g.: the initial color of the background, how the axes will look, etc. As an example, compare the plot shown in Figure 7, where a different theme has been used, with the previous ones.

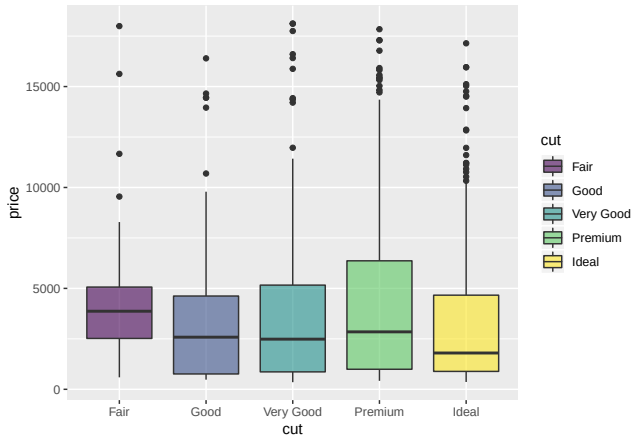

Figure 6: Filling with color the box plots.

```
plot4 <- ggplot(dsmall, aes(x = cut, y = price))
plot4 <- plot4 + geom_boxplot(aes(fill = cut, color = cut)) # fill and color
plot4 <- plot4 + scale_color_viridis_d(option = "cividis")
plot4 <- plot4 + scale_fill_viridis_d(alpha = 0.5, option = "cividis") # using cividis
plot4 <- plot4 + theme_light() # https://ggplot2.tidyverse.org/reference/ggtheme.html
print(plot4)
```

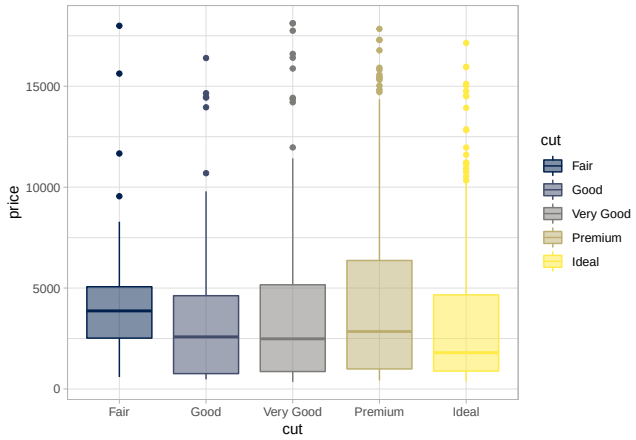

Figure 7: Changing the theme in ggplot2.

Now that we know how to change the color of a plot, let's see how to change the title, axes etc.

2.2 Title, axes and more

Let's add the following modifications to the plot saved in variable `plot3`:

1. A title.
2. A different name for the axes.
3. Different labels for the x-axis: instead of the type of cut, just one letter will appear. Notice that in this case, this last change is counterproductive because the name of the cut is informative, while a letter is not. However, we will do it anyway because most of the times the situation is the opposite: we have non-informative labels and we want to change them.

```r
# Title and name of the axes
plot3 <- plot3 + labs(title = "Spread of price for each type of cut",
                      y = "Price ($)",
                      x = "Type of cut"
                      ) # We could also add a subtitle and a caption.

# Labels of x-axis
labelsX <- c("F", "G", "VG", "P", "I") # save new labels in a variable
plot3 <- plot3 + scale_x_discrete(labels = labelsX)
print(plot3)
```

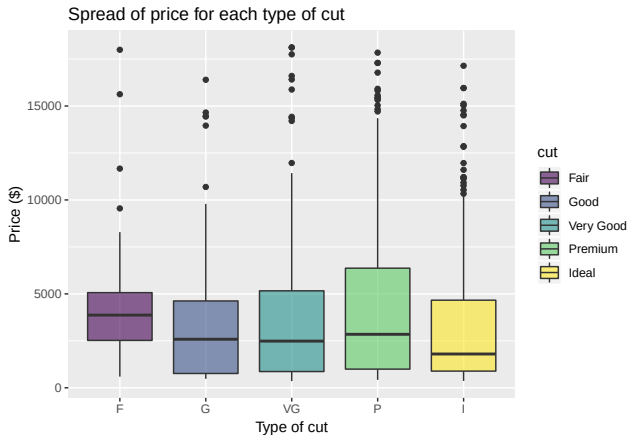

Figure 8: Adding a title to the plot.

Notice that changing the labels of the x-axis **does not** change the labels of the legend (recall that in `ggplot2` we work with independent layers). In the next section we will see how to also change the names in the legend, but here you can see that, to avoid code repetition, it was a good idea to save the new labels in variable `labelsX`.

As you can see, in the previous plot, the title does not appear in the center and it has the same color that the axes. Let's see how to change that and, for instance, how to rotate the labels of the x-axis too.

```r
# Note: check color names in http://sape.inf.usi.ch/quick-reference/ggplot2/colour
plot3 <- plot3 + theme(plot.title = element_text(size   = 15,
                                                 face   = "bold",
                                                 family = "serif",
                                                 color  = "tomato",
                                                 hjust  = 0.5),
                       axis.text.x = element_text(angle = 30,
                                                  vjust = .5)
                       )
print(plot3)
```

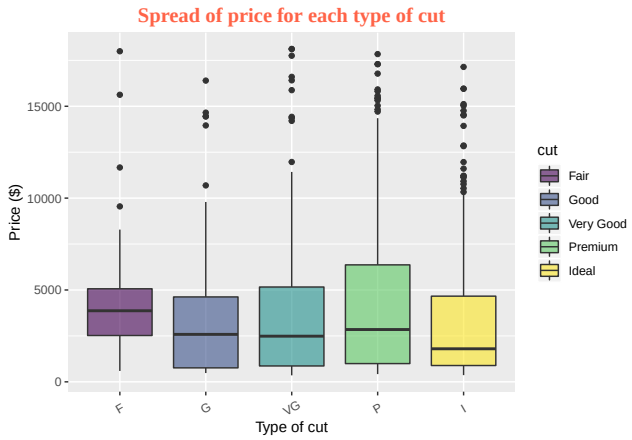

Figure 9: Changing fonts.

2.2.1 Legend

Let's start learning how to modify the title and labels of the legend. Again, notice that in this case, modifying the labels makes no sense because the names that appear (those of the data set) are already good.

```
plot3 <- plot3 + scale_fill_viridis_d(alpha  = 0.6,
                                      name   = "Cut quality",
                                      labels = labelsX)
print(plot3)
```

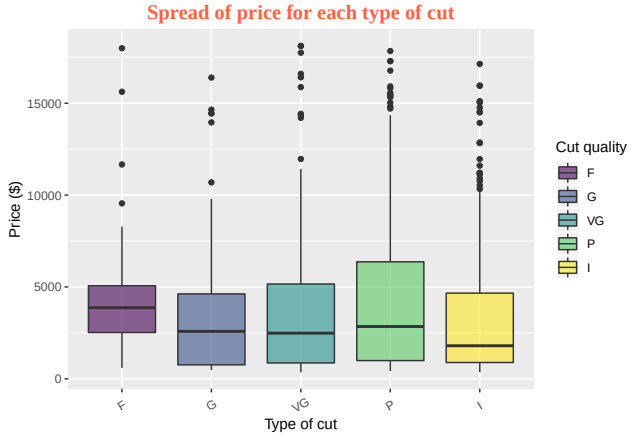

Figure 10: Modifying the legend.

Important! If instead of having filled the box plots, we had colored the borders, we would have used the function `scale_color_viridis_d`.

Now check the following code to see how we can modify different features of the legend:

```
plot3 <- plot3 + theme(legend.title = element_text(colour = "blue",
                                                   size   = 10,
                                                   face   = "bold"),
                       legend.background = element_rect(fill = "gray90",
                                                        size = .5,
                                                        linetype = "dotted"),
```

```
                        legend.position = "top")
print(plot3)
```

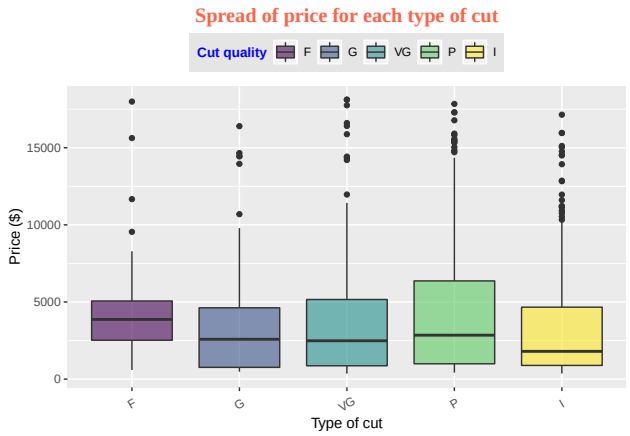

Figure 11: More modifications to the legend.

Note: Use `legend.title = element_blank()` to remove the title of the legend and `legend.position = "none"` to remove the legend from the plot.

2.3 Combining plots

One of the consequences of making a plot through the addition of independent layers is that it allows us to easily combine different graphical representations in a single framework. For instance, imagine that we want to make visible all the points in each box plot and not only the outliers. That is, we want to have in the same plot two graphical representations at the same time: The observations (points) and the box plot. We can achive that by adding to the plot the corresponding two layers as shown below:

```
# Change the properties of the outliers to easily distinguish them
plot3 <- plot3 + geom_boxplot(aes(fill = cut),
                              outlier.fill = "turquoise1",
                              outlier.alpha = 0.4, # some transparency to the outliers
                              outlier.shape = 23)

# Make all the points visibles. Add transparency to better visualize them
plot3 <- plot3 + geom_point(alpha = 0.1) # Affects to all the points but the outliers
print(plot3)
```

2.4 Saving our plots

Once we have all the code to make our plots, for sure we do not want to save them manually one by one clicking on `Export -> save as Image` etc. This is especially relevant when we have made several plots and/or when we want to make the same plots for different data sets. To save the plots automatically we can use the function `ggsave`.

```
ggsave(file = "myggplot1.png",  plot = plot1)
# We can indicate the folder where we want to save our plots
folderPath  = "Folder1/.../ProjectFolder/Plots/"
```

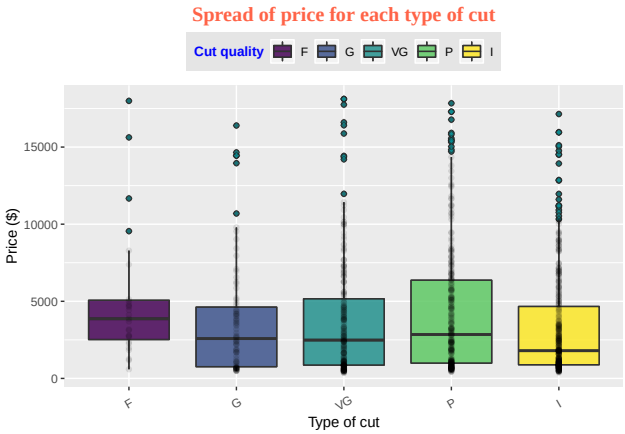

Figure 12: Multiple geom in one plot.

```
ggsave(file = folderPath + "myggplot2.pdf", plot = plot2)
ggsave(file = "myggplot3.jpeg", plot = plot3, width = 4, height = 4)
ggsave(file = "myggplot4.jpeg", plot = plot4, width = 12, height = 12, units = "cm")
```

Note: When exporting a plot with some extensions like EPS, we might obtain a warning message saying: *Transparence not preserved*. If that is the case, the solution is to install the package `Cairo` and add to the `ggsave` function the following argument: `device = cairo_ps`.

3 More plots in ggplot2

Now that we know how to manipulate, combine and save plots, let's check some other graphs that can be made using `ggplot2`. In particular, we will present those that we have been using in our computer labs.

3.1 Histogram and bar plot

3.1.1 Histogram

Imagine that we want to see how the price of the diamonds is distributed, i.e.: what ranges of prices occur more frequently in our data sample. For that, we can make a histogram with the following code.

```
plotHist1 <- ggplot(dsmall)
plotHist1 <- plotHist1 + geom_histogram(aes(price,
                                            y = ..density.. # remove for absolute values
                                            ),
                                        color = "black",
                                        fill  = "lightblue",
                                        bins  = 10 # no need to establish this value
                                       )
plotHist1 <- plotHist1 + labs(title = "Histogram")
print(plotHist1)
```

Notice that the part related with `aes`, instead of being defined inside `geom_histogram`, it could have been defined inside the function `ggplot`, as we did in the previous examples. Depending on the situation, one

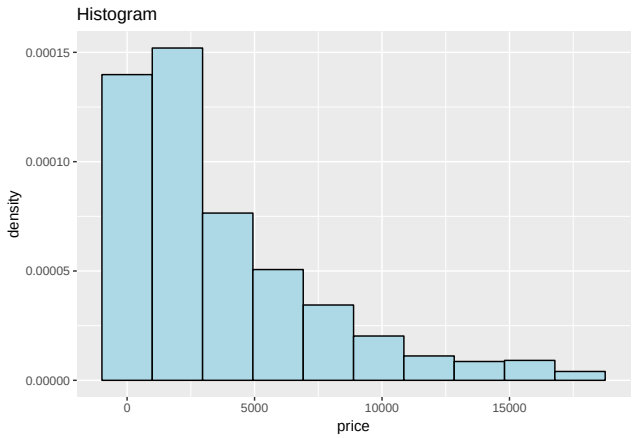

Figure 13: A histogram with ggplot2.

option will be better than the other. Practice will tell you.

3.1.2 Stacked Histogram

Sometimes it might happen that, in addition to see how a variable is distributed, we also want to check, for each bar of the plot, which elements are contributing and how, i.e.: we want a stacked histogram. As an example, imagine that in our previous plot we wanted to see, for each bar in the histogram, what is the contribution per type of cut. The code for that is:

```
plotHist2 <- ggplot(dsmall)
plotHist2 <- plotHist2 + geom_histogram(aes(price, fill = cut),
                                        color = "black",
                                        bins = 10 # no need to stablish this value
                                       )
plotHist2 <- plotHist2 + scale_fill_viridis_d(option = "magma", alpha = 0.8)
plotHist2 <- plotHist2 + labs(title = "Stacked histogram")
print(plotHist2)
```

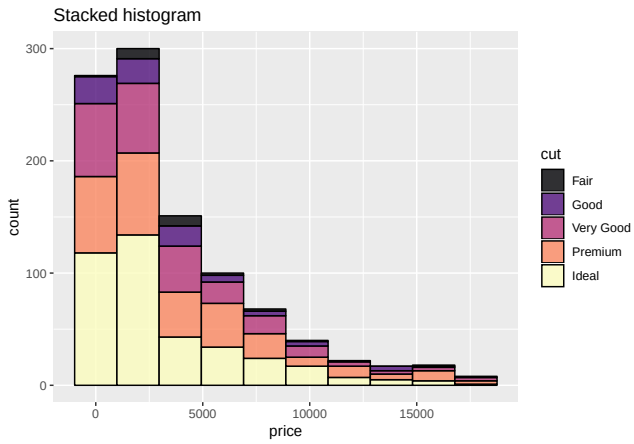

Figure 14: A stacked histogram with ggplot2.

3.1.3 Bar plot

In case that we wanted, for example, to visualize the proportion of diamonds in the data set per type of cut, we would use a bar plot instead of a histogram because variable `cut`, opposite to variable `price`, is discrete.

```
plotBar <- ggplot(dsmall)
plotBar <- plotBar + geom_bar(aes(cut, fill = color),
                              width = 0.5) # width of the bars
plotBar <- plotBar + scale_fill_viridis_d()
plotBar <- plotBar + labs(title = "Stacked bar plot")

# Let's also rotate the x-labels
plotBar <- plotBar + theme(axis.text.x = element_text(angle = 65, vjust = 0.6))
print(plotBar)
```

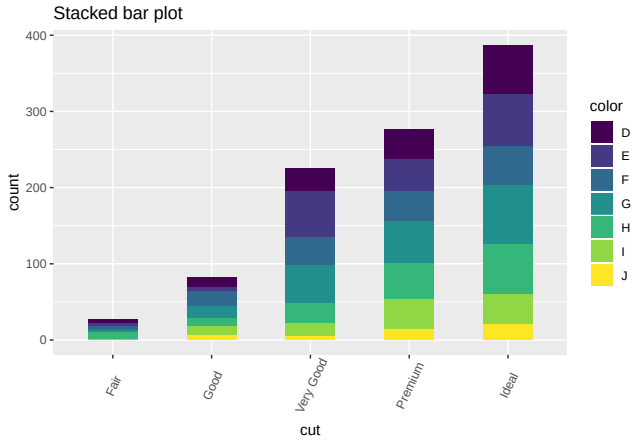

Figure 15: A stacked bar plot with ggplot2.

Notice that we have also split the composition of each bar according to the variable `color` in the data set.

3.2 Density plot

In the histogram of Figure 13, the distribution of the variable `price` seemed to be exponential, but can we assume that? During the course, we have studied different non-parametrical tests to answer that kind of questions. However, to have a graphical intuition about what answer we can expect, we could just add the density line of the corresponding distribution to the histogram and see how well it fits the data.

So, as an example, let's add to the first histogram an exponential and a normal density line. The required parameters λ, μ and σ will be estimated from the data. Also, we will add the corresponding equations of those distributions using `annotate`, a function which allows us to add any kind of text to a plot.

```
plotDensity <- plotHist1 + labs(title = "Histogram and density line")

# Exponential distribution
plotDensity <- plotDensity + stat_function(fun   = dexp,
                                           args  = list(rate = 1/mean(dsmall$price)),
                                           color = "darkred", size = 0.7)

plotDensity <- plotDensity + annotate("text", x = 5000, y = 0.00020,
                                      parse = TRUE, size  = 4,
                                      label = "y == lambda*e^{-lambda*x}",
```

```
                                    color = "darkred")

# Normal distribution
plotDensity <- plotDensity + stat_function(fun  = dnorm,
                                           args = list(mean = mean(dsmall$price),
                                                       sd   = sd(dsmall$price)),
                                           color = "darkgreen", size = 0.7)

plotDensity <- plotDensity + annotate("text", x = 10000, y = 0.00010,
                                      parse = TRUE, size = 4,
                                      label = "y==frac(1, sqrt(2*pi))*e^{-x^2/2}",
                                      color = "darkgreen")

print(plotDensity)
```

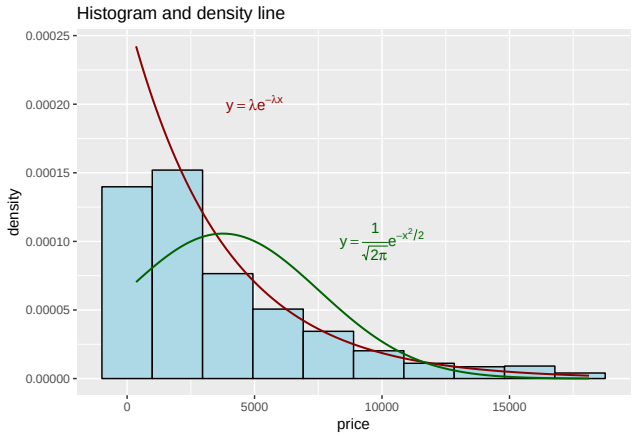

Figure 16: Adding density lines to a plot.

As you can see, none of the distributions really fits the data. For the normal distribution that was expected, but, what about the exponential one? Although the shape of the data fits the density line, the empirical frequencies do not match with the theoretical ones (at least in the first quantiles). However, as mentioned in other parts of this document, our conclusions should be based on maths, not in plots.

Note: Check on the Internet what other things, besides "text", can be added using `annotate`.

3.3 Regression line

Now let's learn how to make a scatter plot containing the corresponding regression line using `ggplot2`. For that, consider the example that we used in the first plot that we made in this document (Figure 1). In the mentioned example we made: "for each type of cut, a scatter plot to visualize the price of the diamonds according to their weight." So, let's make that plot, but this time, adding the corresponding regression line and changing the shape and color of the points whose price is above 15K.

```
plotReg1 <- ggplot(dsmall)
plotReg1 <- plotReg1 + facet_wrap(~ cut, ncol = 3)
plotReg1 <- plotReg1 + geom_point(aes(x = carat, y = price,
                                      shape = price > 15000, # condition for the shape
                                      color = price > 15000),
```

```
                                  alpha = 0.5)
plotReg1 <- plotReg1 + scale_shape_manual(values = c(16,8)) # choose the shape
plotReg1 <- plotReg1 + scale_color_manual(values = c("black", "red")) # choose the color
plotReg1 <- plotReg1 + geom_smooth(aes(x = carat, y = price), method = lm, se = FALSE)
plotReg1 <- plotReg1 + theme(legend.position = "none")
print(plotReg1)
```

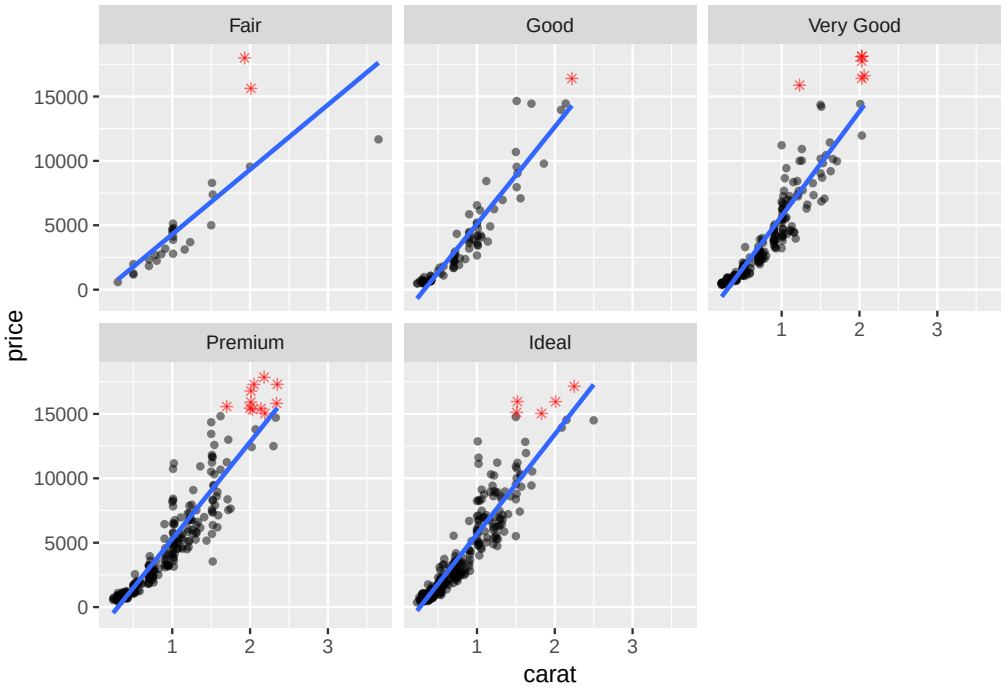

Figure 17: Plotting OLS with ggplot2.

Note: If 'se = FALSE' is removed, the plot will show the confidence intervals for $\hat{y}$. Additionally, it is possible to change the color of the line adding `col = "nameOfTheColor"` inside `geom_smooth`.

3.3.1 Multiple regression lines

Sometimes, we might want to add, in the same plot, a different regression line according to a categorical variable. As an example of that, check the following code:

```
# We use a smaller data set (for clarity) considering 3 types of cut
dsmall2  <- dsmall[dsmall$cut %in% c("Fair", "Good", "Very Good"), ]

# The plot
plotReg2 <- ggplot(dsmall2) + labs(title = "Multiple lines")
plotReg2 <- plotReg2 + geom_point(aes(x = carat, y = price, color = cut, shape = cut),
                                  alpha = 0.5)
plotReg2 <- plotReg2 + geom_smooth(aes(x = carat, y = price, color = cut),
                                   method = lm, se = FALSE)
plotReg2 <- plotReg2 + scale_color_viridis_d(option = "inferno",
                                             end = 0.75) # to not use the yellow color
print(plotReg2)
```

```
## Warning: Using shapes for an ordinal variable is not advised
```

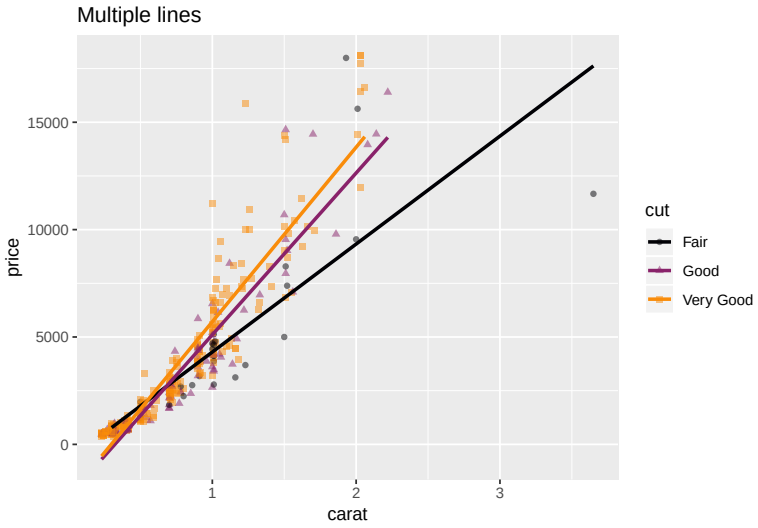

Figure 18: Multiple regression lines.

3.4 Laying out multiple plots on the same frame

There are occasions in which we might want to make several plots on a unique frame to simultaneously expose different ideas about the same data set. In this section we will see a brief introduction to that. However, before doing it, let's change the text size appearing in the three plots (Figures 14, 16, 18) that we will be using in our example. This is to avoid the final plot not looking good (e.g.: having an overlap of the values in the axes).

```
plotHist2 <- plotHist2 + theme(plot.title  = element_text(size = 10),
                               axis.text.x = element_text(size = 6),
                               axis.text.y = element_text(size = 6),
                               text = element_text(size = 7)
                              )

plotDensity <- plotDensity + theme(plot.title   = element_text(size = 10),
                                   legend.title = element_text(size = 7),
                                   axis.text.x  = element_text(size = 6),
                                   axis.text.y  = element_text(size = 6),
                                   text = element_text(size = 7)
                                  )

plotReg2 <- plotReg2 + theme(plot.title   = element_text(size = 10),
                             legend.title = element_text(size = 7),
                             axis.text.x  = element_text(size = 6),
                             axis.text.y  = element_text(size = 6),
                             text = element_text(size = 7)
                            )
```

Now, imagine that we want to represent the three previous plots on the same frame. For that, we need to provide how we want to divide the frame (e.g.: 2 columns and 2 rows) and put the plots on it. **Additionally**, it is also important to label each of the plots so we can properly reference them in the text afterwards. Check the following code to see how to achieve this.

```
library(cowplot)
multiplePlots <- plot_grid(plotHist2, plotDensity, plotReg2,
                           labels = "AUTO",  # use "auto" for lowercase letters
                           label_size = 8,
                           ncol = 2, nrow = 2)

title <- ggdraw() + draw_label("General Title", fontface = 'bold')

plot_grid(title, multiplePlots, ncol = 1,
          rel_heights = c(0.1, 1)) # rel_heights values control title margins
```

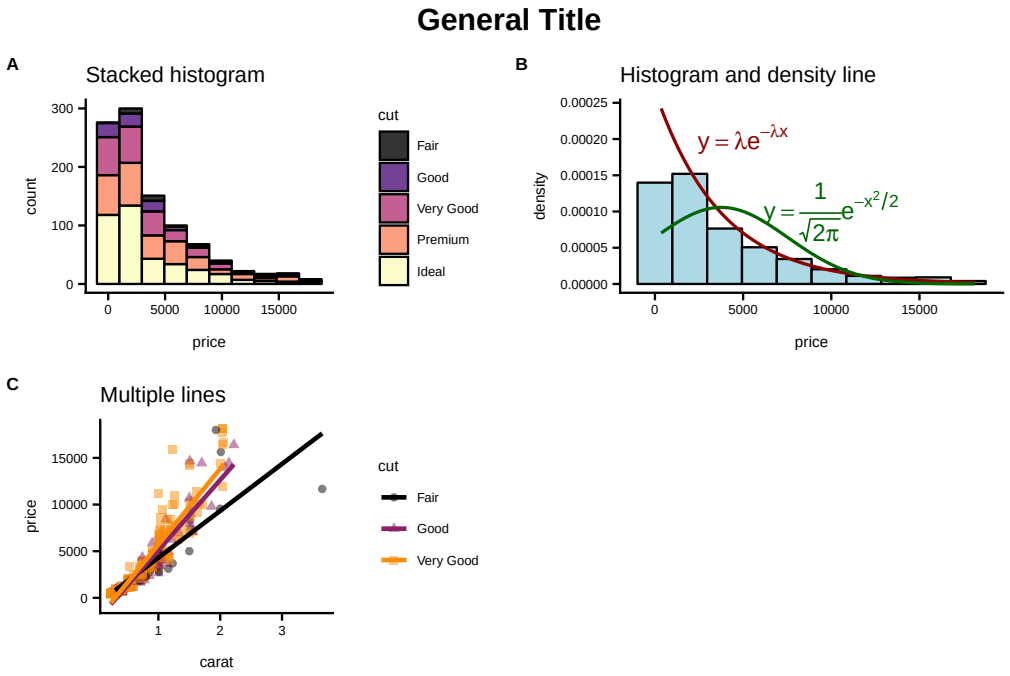

Figure 19: Multiple plots in the same frame.

Note: Try to remove `rel_heights` to see what happens.

Now check the same example, but when making one of the plots bigger than the other two.

```
multiplePlots2 <- plot_grid(plotHist2,
                           plot_grid(plotDensity, plotReg2,
                                     ncol = 2, nrow = 1,
                                     labels = c("(B)", "(C)"), label_size = 8),
                           labels = c("(A)"), label_size = 8,
                           ncol = 1, nrow = 2)

plot_grid(title, multiplePlots2, ncol = 1, rel_heights = c(0.1, 1))
```

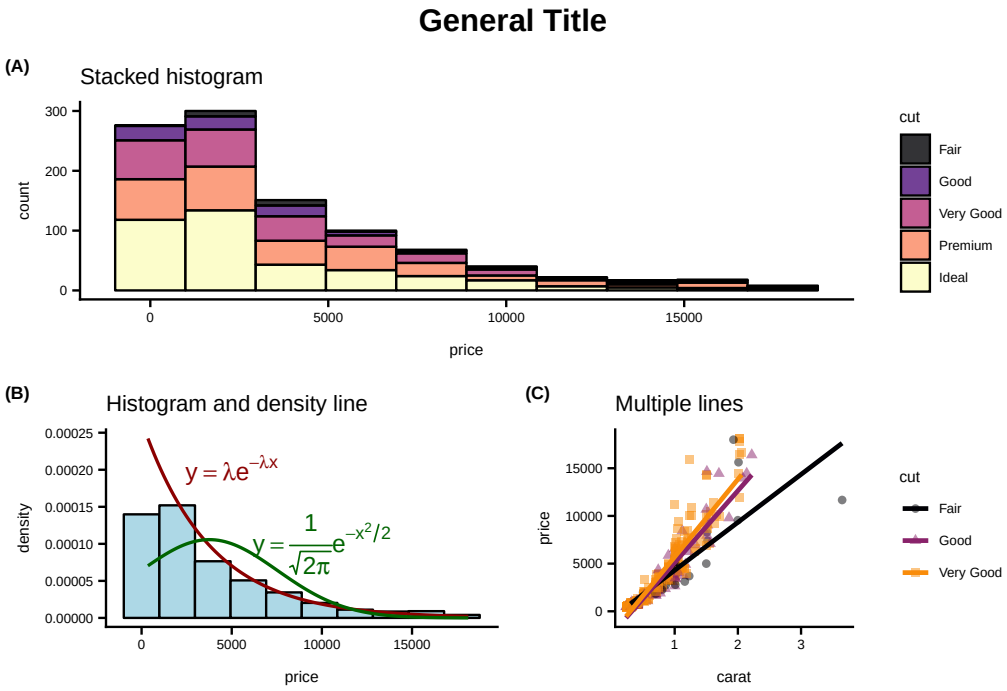

Figure 20: Multiple plots in the same frame II.

4 References

We conclude this document citing the references employed to create it, and providing some comments about what can be found in each of them.

1. *Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc.* In the third chapter of this fantastic book, there is a great introduction to `ggplot2`, where some extra content not addressed here can be found. Furthermore, the book is legally available online for free!!!

2. The following links provide more information about: 1) the color scales and 2) The types of graphs that can be made with `ggplot2`:

   i) https://codingclubuc3m.rbind.io/post/2018-05-10/
   ii) https://codingclubuc3m.github.io/20190305_slides/#1
   iii) http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

3. For a complete (and advanced) guide in `ggplot2`, check out the book: *Wickham, H. (2016). ggplot2: elegant graphics for data analysis. Springer.*