DR. OSAMA MAHMOUD

ADVANCED GRAPHICS USING R

Contents

“THE GREATEST VALUE OF A PICTURE IS WHEN IT FORCES US TO NOTICE WHAT WE NEVER EXPECTED TO SEE.”

JOHN TUKEY

“VISUALIZATION GIVES YOU ANSWERS TO QUESTIONS YOU DIDN’T KNOW YOU HAD.”

BEN SCHNEIDERMAN

“VISUALIZATIONS ACT AS A CAMPFIRE AROUND WHICH WE GATHER TO TELL STORIES.”

AL SHALLOWAY

1
Background

1.1 Installing packages

Installing packages in R is straightforward. To install a package from the command line we use the `install.packages` command. For example, to install the 'ggplot2' package, you need:

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

NB: Installation is only required before your 1st use of a package. Then, you can simply load it to your R session whenever needed using the `library` command, i.e.

```
library("ggplot2")
```

1.2 Types of R graphics

1.2.1 Base graphics

Base graphics were written by Ross Ihaka based on his experience of implementing the S graphics driver. If you have created a boxplot, histogram or scatter plot, you may have used base graphics. They are generally fast, but have limited scope, e.g. you can not edit or alter existing graphics but only draw on top of the plot. For example, if you want to add points - using `points` commands - to a basic graph plotted by `plot` command, you have to work out the x- and y- limits before adding the points!

1.2.2 Grid graphics

Grid graphics were developed by Paul Murrell[1] to have more advanced scope than the basic graphics. In Grid graphics, graphical objects can be represented independently and modified later. Grid graphics associated system, called viewports, makes it easier to construct complex plots. Grid per se does not provide tools for graphics, it rather provides primitives for creating plots. Hence, grid is the basis for several advanced graphics, e.g. Lattice and ggplot2.

[1] P Murrell. *R Graphics*. CRC Press, 2 edition, 2011

1.2.3 *Lattice graphics*

The lattice graphics[2] produce better and automated plots than base graphics, e.g. their legends are automatically generated. I have not used lattice intensively as I found `ggplot2` is much more easier for me!

[2] D Sarkar. *Lattice: Multivariate Data Visualization with R (Use R!)*. Springer, 1st edition, 2008

1.2.4 *ggplot2 graphics*

The `ggplot2` follows the "*GrammarofGraphics*"[3], a philosophy of visualisation developed by Leland Wilkinson[4], to produce advanced graphics.[5] Like `lattice`, `ggplot2` uses grid to draw graphics, which means you can enjoy low-level control over the plot appearance.

[3] We'll come on to that later.

[4] L Wilkinson. *The Grammar of Graphics*. Springer, 1st edition, 1999

[5] H Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2009. ISBN 978-0-387-98140-6

1.3 *Data sets*

Throughout the course, we will use many datasets. Detailed descriptions of these data, along-with course materials are available from the website of the course materials.[6] Here, we give a simple introduction to each data set:

[6] http://bristol-medical-stat.bristol.ac.uk:3838/RVis/.

1.3.1 *Pain medication data*

This data is a modified data set from a randomized trial for 200 individuals with 2 treatment arms. It is included with the `BristolVis` package which can be installed using:

```r
install.packages("drat")
drat::addRepo("statcourses")
install.packages("BristolVis", type="source")
```

Then, the data is loaded using the `data` function:

```r
data(med, package="BristolVis")
```

1.3.2 *Fuel economy data*

This dataset includes car make, model, class, engine size and fuel economy for a selection of US cars in 1999 and 2008. It is included with the `ggplot2` package[7] and is then loaded using:

[7] The data originally comes from the EPA fuel economy website, http://fueleconomy.gov

```r
data(mpg, package="ggplot2")
```

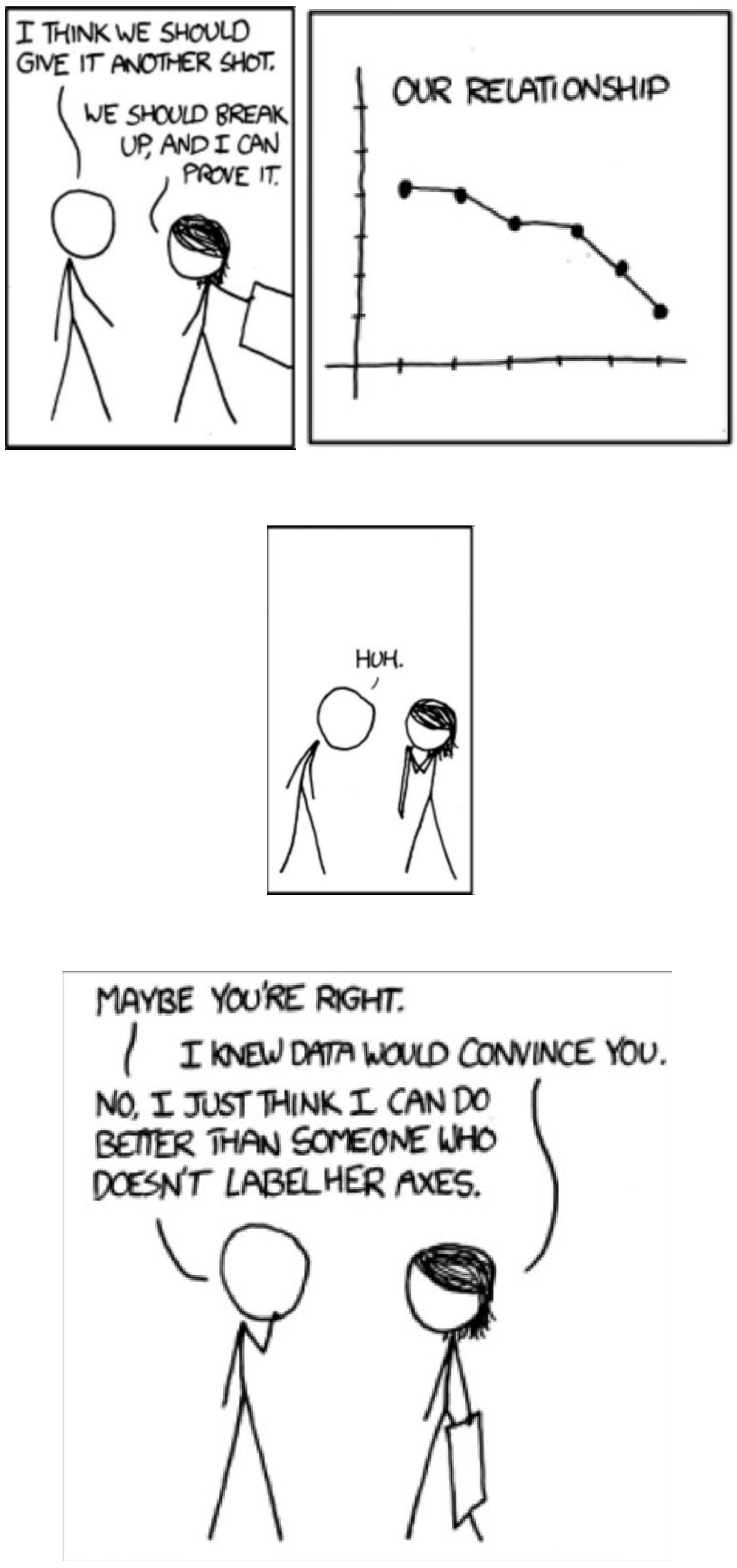
I THINK WE SHOULD GIVE IT ANOTHER SHOT.
WE SHOULD BREAK UP, AND I CAN PROVE IT.
OUR RELATIONSHIP
HUH.
MAYBE YOU'RE RIGHT.
I KNEW DATA WOULD CONVINCE YOU.
NO, I JUST THINK I CAN DO BETTER THAN SOMEONE WHO DOESN'T LABEL HER AXES.

Figure 1.1: http://xkcd.com/833/

2
ggplot2 overview

The `ggplot2` thinks about plots as layers. This makes it easier for the user to iteratively add new components.[1]

[1] and for a developer to add new functionality.

2.1 *A basic plot using base graphics*

One might first attempt to analyse the `med`[2] data set by producing a scatter plot of e.g., time to relief against age. Using `base` graphics, we would first construct a basic scatter plot of the data for subjects with poor health condition:[3]

[2] See §1.3.1.

[3] We've cheated here and pretended that we know the x- and y- limits by setting the `xlim` and `ylim` parameters in advance for the whole data set being plotted.

```
plot(med[med$health=="Poor",]$age,
     med[med$health=="Poor",]$time,
     xlim=c(30,72), ylim=c(0.5,13))
```

Next we add in the other subjects corresponding to different health conditions:

```
points(med[med$health=="Fair",]$age,
       med[med$health=="Fair",]$time, col=2)
points(med[med$health=="Good",]$age,
       med[med$health=="Good",]$time, col=4)
```

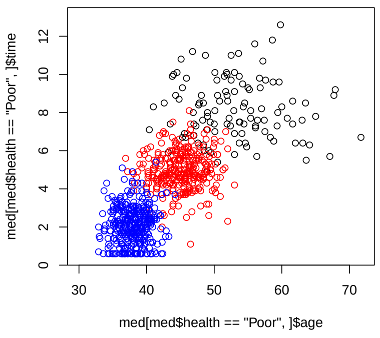

Figure 2.1: A scatter plot, using `base` graphics, for time to relief vs age. The coloured points correspond to different health conditions.

This would produce figure 2.1. Here, there are a few points to note:

- We had to manually set the scales in the `plot` command using the `xlim` and `ylim` parameters.
- We had not created a legend. We would need to use the `legend` function to create one.
- The default axis labels were terrible, e.g. `med[med$health=="Poor",]$age`.

2.2 *Equivalent graphics using ggplot2*

Let's now consider the equivalent `ggplot2` graphic - figure 2.2. After loading the necessary library, the plot is generated using the following code:

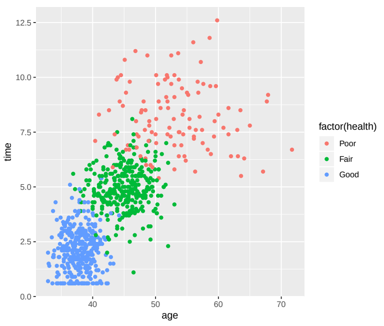

Figure 2.2: As figure 2.1, but created using `ggplot2`.

```
library(ggplot2)
g = ggplot(data=med, aes(x=age, y=time))
g + geom_point(aes(colour=health))
```

Plot Name	Geom	Base graphic
Barchart	bar	`barplot`
Box-and-whisker	boxplot	`boxplot`
Histogram	histogram	`hist`
Line plot	line	`plot` and `lines`
Scatter plot	point	`plot` and `points`

Table 2.1: Basic `geom`'s and their corresponding standard plot names.

The philosophy behind the `ggplot2` code is fundamentally different from the `base` code. The `ggplot` function sets the default data set, and attributes called **aesthetics** which represent the properties that are perceived on the graphic. A particular aesthetic can be mapped to a variable or set to a constant value. In figure 2.2, the variable `age` is mapped to the x-axis and `time` variable is mapped to the y-axis.

The other function, `geom_point` adds a layer to the plot. The `x` and `y` variables are inherited (in this case) from the first function, `ggplot`, and the colour aesthetic is set to the `health` variable. Other possible aesthetics include size, shape and transparency. In figure 2.2 these additional aesthetics are left at their default value.

This approach is very powerful and enables us to easily create complex graphics. For example, we could create a plot where the size of the points depends on an additional factor:

```
(p = g + geom_point(aes(size=health)))
```

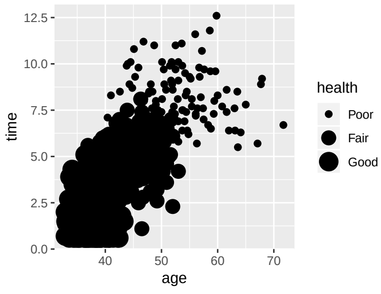

Figure 2.3: As figure 2.2, but where the size aesthetic depends on health condition.

which gives figure 2.3 or we could create a line chart

```
(p = g + geom_line(aes(colour=health, size = health)))
```

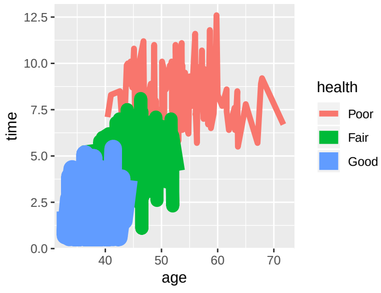

Figure 2.4: As figure 2.2, but using `geom_line`.

to get figure 2.4. Of course, figures 2.3 and 2.4 are not particular good plots, they just illustrate the general idea.

Points, bars and lines are examples of **geom**'s or geometric objects. Typically, if we use a single `geom`, we get a standard plot. Table 2.1 summarises some standard geoms and their equivalent base graphic counter part.

Using the idea of a graphical layer, we can construct more complicated functions. For example, this code uses `mpg`[4] data

[4] see 1.3.2

```
(p = ggplot(mpg, aes(x = displ, y = cty)) +
  geom_point(aes(colour=factor(cyl))) +
  stat_smooth(aes(colour=factor(cyl))))
```

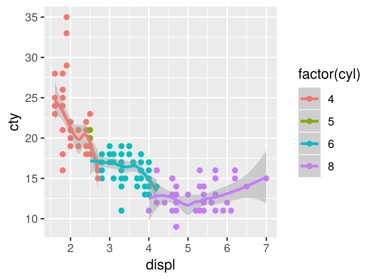

Figure 2.5: Geom points plot with LOESS regression smooth curves fitted.

to produce figure 2.5, which combines things beyond simple graphics.

NOTE THAT: Using each `ggplot2` command, we are adding (multiple) layers. A single layer may comprise of four elements:

- an aesthetic and data mapping;
- a statistical transformation (**stat**);

- a geometric object (**geom**);
- and a position adjustment, i.e. how should objects that overlap be handled.

  When we use the command:

```
g + geom_point(aes(colour=health))
```

we are actually calling (in the background) the command:

```
g + layer(
     data = med, #inherited
     mapping = aes(color=health), #x and y are inherited.
     stat = "identity",
     geom = "point",
     position = "identity",
     params = list(na.rm=FALSE)
)
```

In practice, we **never** use the layer function. Instead, we use:

- `geom_*` which creates a layer for the `geom`;
- `stat_*` which create a layer for the `stat`;
- `qplot` which creates together a ggplot and a layer.

`qplot` is short for 'quick plot'. We do not cover `qplot` in this course. However, If you will use `ggplot2` intensively, then it is worth the time investment.

3
Plot building-blocks

3.1 The basic plot object

To create an initial ggplot object, we use the `ggplot()` function which has two arguments:

- **`data`** and
- an aesthetic **`mapping`**.

These arguments set up the defaults for the various layers that are added to the plot. For each plot layer, these arguments can be overwritten.

The `data` argument takes a data frame. The `mapping` argument creates default aesthetic attributes. For example, the command:

```
g = ggplot(data=mpg,
      mapping=aes(x=displ, y=cty, colour=factor(cyl)))
```

or equivalently,

```
g = ggplot(mpg, aes(displ, cty, colour=factor(cyl)))
```

doesn't actually produce anything to be displayed, it just sets the intial plot object. We need to add layers for that to happen.

3.2 The geom_ functions

The `geom_` functions are used to perform the actual rendering in a plot, e.g. a line geom will create a line plot and a point geom creates a scatter plot. Each geom has a list of aesthetics that it accepts.[1] However, some geoms have unique elements. The error-bar geom requires arguments `ymax` and `ymin`. Table 3.1 gives some commonly used geoms.[2] The full list of aesthetics can be shown by:

[1] For example, `x`, `y`, `colour` and `size`.

[2] For a full list, see `https://ggplot2.tidyverse.org/reference/`.

```
ggplot2:::.all_aesthetics
```

Name	Description
`abline`	Line, specified by slope and intercept
`boxplot`	Box and whiskers plot
`density`	Kernel density plot
`histogram`	Histograms
`jitter`	Individual points are jittered to avoid overlap
`step`	Connect observations by stairs

Table 3.1: A few standard `geom_` functions in `ggplot2`.

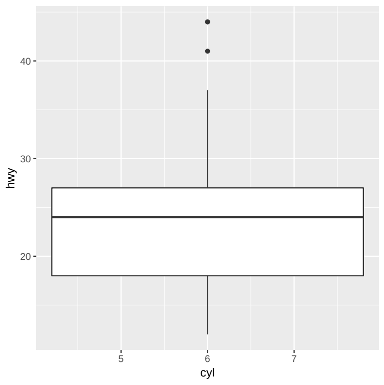

Figure 3.1: A boxplot of highway miles per gallon.

3.2.1 *combining geoms*

Here, I will show how to combine more that one `geom` function to produce a bit more complex plots. If we consider the `mpg` data set and begin with creating a base ggplot object

```
g = ggplot(mpg, aes(x=cyl, y=hwy))
```

Remember, the above piece of code doesn't do anything. Now we'll create a boxplot using the boxplot geom:

```
(g1 = g + geom_boxplot())
```

This produces figure 3.1. Notice that the default axis labels are the column headings of the associated data frame. Figure 3.1 is a boxplot of all the mpg data, a more useful plot would be to have individual boxplots conditional on number of cylinders.

```
(g2 = g + geom_boxplot(aes(x=factor(cyl),group=cyl)))
```

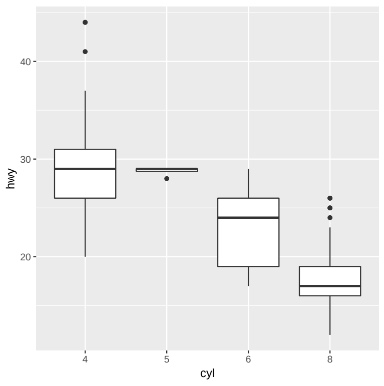

Figure 3.2: A boxplots of highway miles per gallon, conditional on number of cylinders.

Notice that we have included a group aesthetic to the boxplot `geom`. Many `geom`'s have this aesthetic. For example, if we used `geom_line`, then we would have individual lines for each number of cylinders - this doesn't make much sense in this situation.

We are not restricted to a single geom - we can add multiple geoms. When data sets are reasonably small, it is useful to display the data on top of the boxplots:

```
(g3 = g2  + geom_dotplot(aes(x=factor(cyl), group=cyl),
            binaxis="y", stackdir="center",
            binwidth=0.25, stackratio=2))
```

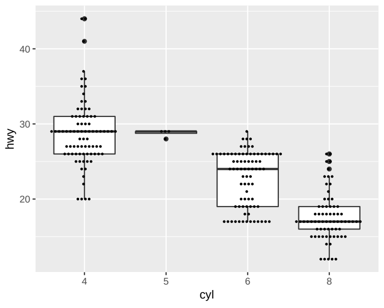

Figure 3.3: As figure 3.2, but including the data points.

This generates figure 3.3. The dotplot geom produces a sort of histogram. Notice that we can start picking off some patterns.

3.3 *Standard plots*

There are a few standard geom's that are particular useful:

- geom_boxplot: produces a boxplot - see figure 3.1.
- geom_point: a scatter plot - see figure 2.2.
- geom_dotplot: a dot plot - see figure 3.3
- geom_bar: produces a standard barplot that counts the x values. For example, to generate a bar plot in figure 3.4 of the status of effect observations, in the med data set, the following code can be used:

```
(h = ggplot(med, aes(x=status)) + geom_bar())
```

- geom_line: a line plot
- geom_text: adds labels to specified points. This has an additional (required) aesthetic: label. Other useful aesthetics, such as hjust and vjust control the horizontal and vertical position. The angle aesthetic controls the text angle.
- geom_raster: Similar to levelplot or image. For example,

```
(g_rast = ggplot(med, aes(gender, health)) +
   geom_raster(aes(fill=time)))
```

generates figure 3.5.

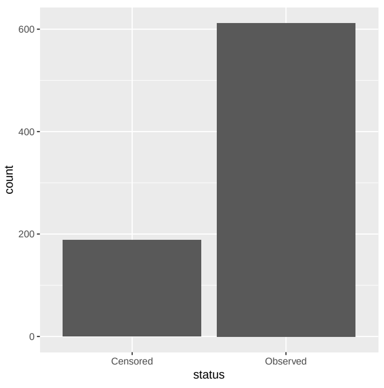

Figure 3.4: A bar chart of the status of effect observation.

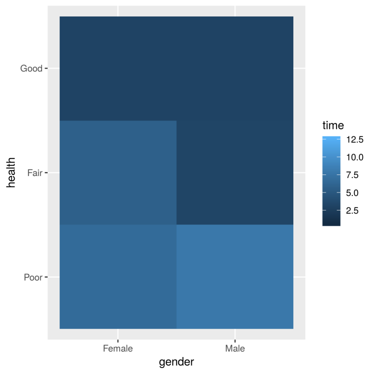

Figure 3.5: A simple heatmap of health condition against gender using geom_raster.

4
Facets

Faceting is a mechanism for automatically laying out multiple plots on a page. The data is split into subsets, with each subset plotted onto a different panel. `ggplot2` has two types of faceting:

- `facet_grid`: produces a 2d panel of plots where variables define rows and columns.
- `facet_wrap`: produces a 1d ribbon of panels which can be wrapped into 2d.

4.1 *Facet grid*

The function `facet_grid` lays out the plots in a 2d grid. The faceting formula specifies the variables that appear in the columns and rows. Suppose we are interested in time to relief. A first plot we could generate is a basic histogram:

```
(g = ggplot(med, aes(x=time)) +
  geom_histogram(aes(y=..density..), binwidth=0.2))
```

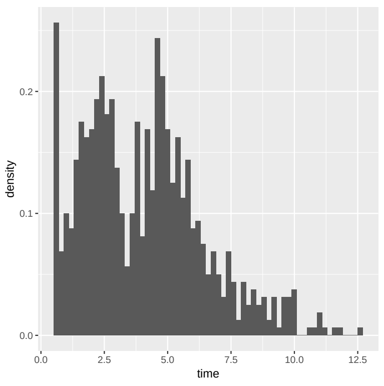

Figure 4.1: A histogram of time to relief.

This produces figure 4.1. Notice that we adjusted the binwidth in the histogram and used density as the y-axis scale. This just means that the area under the histogram sums to one. The data is clearly positively skewed and some subjects reliefed in fairly short, whilst others have an average time to relief around five hours.

We will now use faceting to explore the data further.

- `y ~ .`: a single column with multiple rows, whose number is the number of `levels(y)`. This can be handy for some situations, e.g. double column journals. For example, to create histograms conditional on gender, we use:

  ```
  (g1 = g + facet_grid(gender ~ .))
  ```

  This gives figure 4.2.

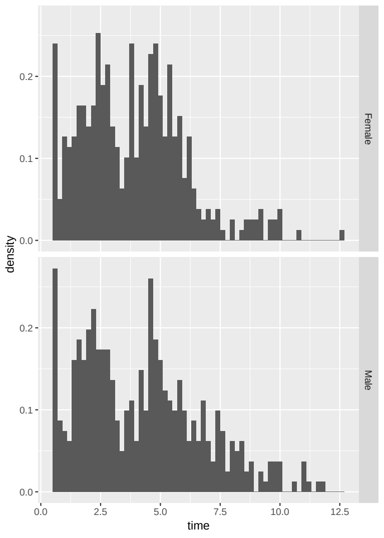

Figure 4.2: time to relief conditional on gender.

- `. ~ x`: a single row with multiple columns (Very useful in wide screen monitors). In this piece of code, we create histograms conditional on health condition:

```
(g2 = g + facet_grid(. ~ health) +
   scale_x_continuous(breaks = c(0, 5, 10)))
```

From figure 4.3, it's clear that subjects with better health conditions tend to relief faster. For illustration purposes, we have used the `geom_density` function in figure 4.3. Moreover, we set the breaks of the x axis using the argument `scale_x`

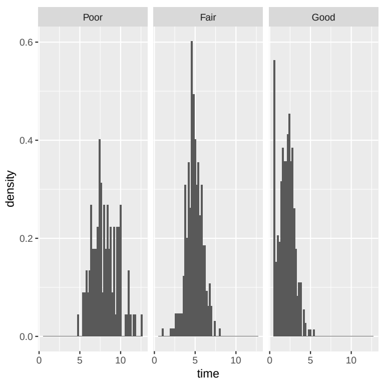

Figure 4.3: Histograms of time to relief conditional on health condition.

- y ∼ x: multiple rows and columns.

```
(g3 = g + facet_grid(gender ~ health) +
theme(axis.text.x = element_text(angle = 90, hjust = 1)))
```

Figure 4.4 splits time to relief by gender and health condition.

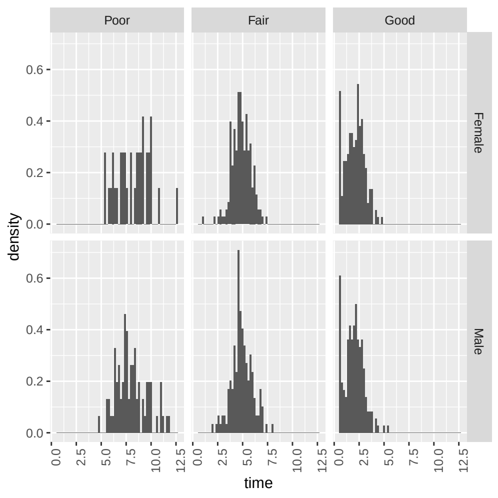

Figure 4.4: Time to relief conditional on gender and health condition.

4.2 *Facet wrap*

The `facet_wrap` function creates a 1d ribbon of plots. For example,

```
(g4 = ggplot(mpg, aes(x=cty)) +
geom_histogram(binwidth=1) + facet_wrap(~class, ncol=4) +
  coord_flip())
```

produce the figure 4.5.

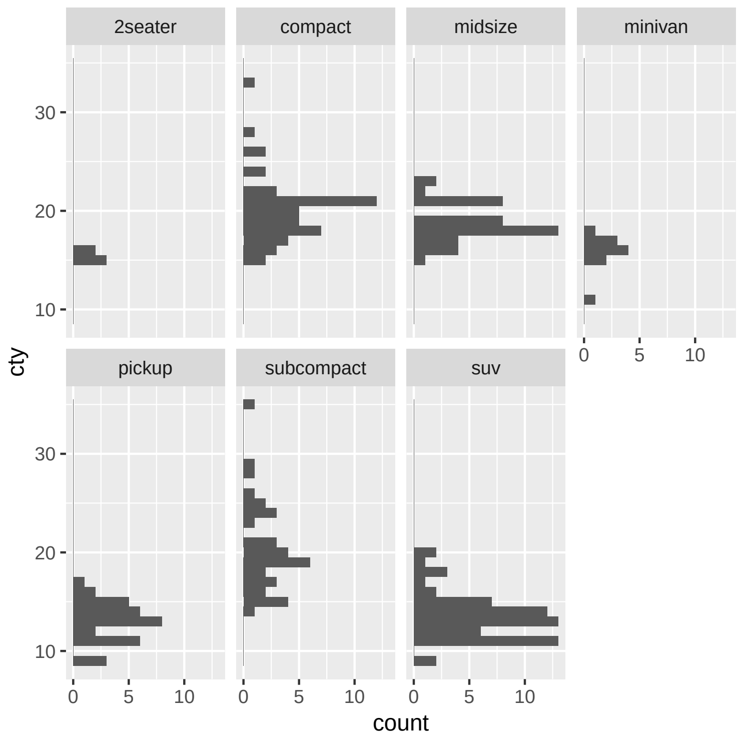

Figure 4.5: average city miles per gallon on the car class.

5
Themes

5.1 Introduction

If you want to make consistent changes to all your plots, e.g. change the font size, background colour, or axis label angels[1], then you need to use themes.

[1] as done in figure 4.4.

5.2 Choice of themes

A few standard useful themes are available in the `ggthemes` package which can be loaded using:

```
require(ggthemes)
```

Once the package is loaded, a range of standard themes can be used to control the plot appearance:

- `theme_bw()`
- `theme_economist()`
- `theme_stata()` - useful for `stata` users
- `theme_pander()`
- `theme_hc(style = "darkunica")`

For example:

```
data(movies, package="ggplot2movies")
(h = ggplot(movies, aes(year)) +
  geom_histogram(binwidth = 1, fill="#2b8cbe", alpha=0.6) +
  xlab("Year")+ylab("Number of movies produced"))
```

produces figure 5.1 which uses the `ggplot2` default theme. The `theme_bw()` is used to produce figure 5.2 as follows:

```
(h1 = h + theme_bw())
```

Using different theme can have a huge influence on the appearance. For instance, the following line code gives the figure 5.3

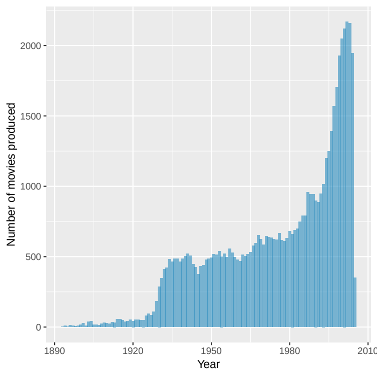

Figure 5.1: Histograms of movie length per year.

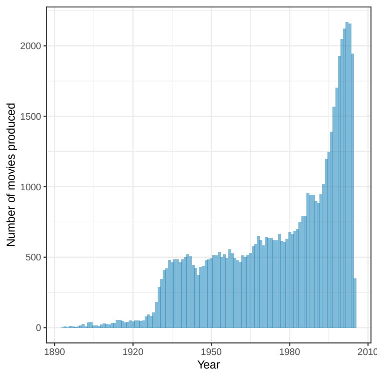

Figure 5.2: As figure 5.1, but with `theme_bw`.

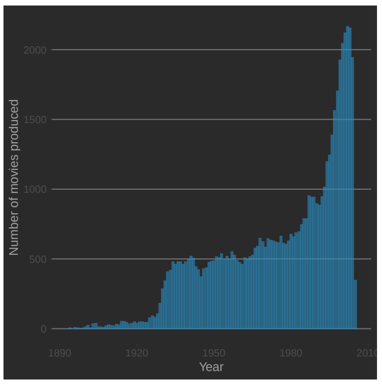

Figure 5.3: As figure 5.1, but with `theme_hc`.

```
(h2 = h + theme_hc(style = "darkunica"))
```

Moreover, We can customise our own theme:

```
My_theme <-
  theme(axis.text=element_text(size=10, face="bold"),
  axis.text.x=element_text(angle=20, vjust=0.5),
  axis.text.y=element_text(vjust=0.5, hjust = 0.5),
  plot.margin = unit(c(0.5,0.5,0.5,0.5), "cm"),
  axis.title=element_text(size = rel(1.2)),
  axis.title.x=element_text(vjust=-0.5),
  axis.title.y=element_text(vjust=1.5))
```

Then, it can be used to produce a customised plot. The following line code gives figure 5.4.

```
(h3 = h1 + My_theme)
```

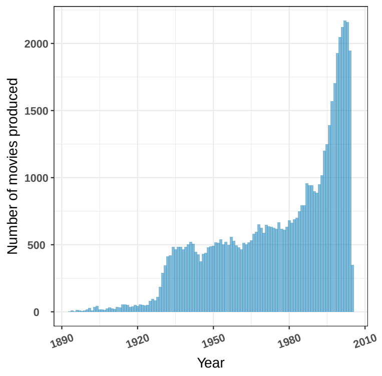

Figure 5.4: As figure 5.2, but with self-customised theme's element.

Appendix

Course R package: BristolVis

This course has an associated R package. Installing this package is straightforward[2]. First install the `drat` package

```
install.packages("drat")
```

Then run the command[3]

```
drat::addRepo("statcourses")
```

Then install the package as usual

```
install.packages("BristolVis")
```

To load the package, use

```
require("BristolVis")
```

[2] See also: `http://osmahmoud.com/R-courses/`.

[3] This adds a new repo URL to you list of repositories.

Acknowledgement

I would like to thank Dr. Andrew Harrison[4] for giving me the first opportunity to teach R at the CODATA summer school in Trieste, Italy, 2016. My deepest gratitude goes to Dr. Colin Gillespie from whom I have learnt many programming techniques[5].

[4] Senior lecturer, Essex Univeristy

[5] C Gillespie and R Lovelace. *Efficient R Programming*. O'Reilly, 1st edition, 2016

Bibliography

C Gillespie and R Lovelace. *Efficient R Programming*. O'Reilly, 1st edition, 2016.

P Murrell. *R Graphics*. CRC Press, 2 edition, 2011.

D Sarkar. *Lattice: Multivariate Data Visualization with R (Use R!)*. Springer, 1st edition, 2008.

H Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2009. ISBN 978-0-387-98140-6.

L Wilkinson. *The Grammar of Graphics*. Springer, 1st edition, 1999.