Data Science: Visualisations in R - Assignment 3

Healy (2018) Data visualization, Chapters 5 and 8

dr. Mariken A.C.G. van der Velden

November 15, 2019

Instructions for the tutorial

Read the *entire* document that describes the assignment for this tutorial. Follow the indicated steps. Each assignment in the tutorial contains several exercises, divided into numbers and letters (1a, 1b, 1c, 2a, 2b, etc.).

Write down your answers with **the same number-letter combination** in a separate document (e.g., Google Doc), and submit this document to Canvas in a PDF format.

Note: this is the first time that this course has ever been taught. Because of rapidly changing techniques, the course tries to keep up/ reflect those changes. As a result, the assignments may contain errors or ambiguities. If in doubt, ask for clarification during the hall lectures or tutorials. All feedback on the assignments is greatly appreciated!

Refining Your Graphs

Today, we will build upon the knowledge gathered in the last two weeks in three ways:

1. We will learn about how to transform data *before* we send it to ggplot to be turned into a figure. As we saw in last week's assignment, ggplot's geoms will often summarize data for us. While convenient, this can sometimes be awkward or even a little opaque. Often, it's better to get things into the right shape before we send anything to ggplot. This is a job for another tidyverse component, the `dplyr` library. We will learn how to use some of its "action verbs" to select, group, summarize and transform our data.
2. We will expand the number of geoms, and learn more about how to choose between them. The more we learn about ggplot's geoms, the easier it will be to pick the right one given the data we have and the visualization we want. As we learn about new geoms, we will also get a little more adventurous and depart from some of ggplot's default arguments and settings. We will learn how to reorder the variables displayed in our figures, and how to subset the data we use before we display it.
3. The process of gradual customization will give us the opportunity to learn a little more about the `scale`, `guide`, and `theme` functions that we have mostly taken for granted until now. These will give us even more control over the content and appearance of our graphs. Together, these techniques can be used to make plots much more legible to readers. They allow us to present our data in a more structured and easily comprehensible way, and to pick out the elements of it that are of particular interest. We will begin to use these techniques to layer geoms on top of one another, a technique that will allow us to produce very sophisticated graphs in a systematic, comprehensible way.

Still, no matter how complex our plots get, or how many individual steps we take to layer and tweak their features, underneath we will always be doing the same thing. We want a table of tidy data (see example below), a mapping of variables to aesthetic elements, and a particular type of graph. If you can keep sight of this, it will make it easier to confidently approach the job of getting any particular graph to look just right!

```
library(tidyverse)
library(socviz)

table(gss_sm$bigregion, gss_sm$religion, useNA = "ifany")
```

```
##
##             Protestant Catholic Jewish None Other <NA>
##   Northeast        158      162     27  112    28    1
##   Midwest          325      172      3  157    33    5
##   South            650      160     11  170    50   11
##   West             238      155     10  180    48    1
```

Summarize & Transform Data

In Chapter 4 we began making plots of the distributions and relative frequencies of variables. Cross-classifying one measure by another is one of the basic descriptive tasks in data analysis. Tables 5.1 and 5.2 show two common ways of summarizing our GSS data on the distribution of religious affiliation and region. Table 5.1 shows the column marginals, where the numbers sum to a hundred by column and show, e.g., the distribution of Protestants across regions. Meanwhile in Table 5.2 the numbers sum to a hundred across the rows, showing for example the distribution of religious affiliations within any particular region.

As shown schematically in Figure 5.1, we will start with our individual-level table of about 2,500 GSS respondents. Then we want to summarize them into a new table that shows a count of each religious preference, grouped by region. Finally, we will turn these within-region counts into percentages, where the denominator is the total number of respondents within each region. The `dplyr` library provides a few tools to make this easy and clear to read. We will use a special operator,‘%>%, to do our work. This is the *pipe* operator. It plays the role of the yellow triangle in Figure 5.1, in that it helps us perform the actions that get us from one table to the next.

We have been building our plots in an *additive* fashion, starting with a `ggplot` object and layering on new elements. By analogy, think of the `%>%` operator as allowing us to start with a data frame and perform a *sequence* or *pipeline* of operations to turn it into another, usually smaller and more aggregated table. Data goes in one side of the pipe, actions are performed via functions, and results come out the other. A pipeline is typically a series of operations that do one or more of four things:

- *Group* the data into the nested structure we want for our summary, such as “Religion by Region” or “Authors by Publications by Year”. This can be done with the function `group_by()`.
- *Filter* or *select* pieces of the data by row, column, or both. This gets us the piece of the table we want to work on. This can be done with the functions `filter()` (for rows) and `select()` (for columns).
- *Mutate* the data by creating new variables at the current level of grouping. This adds new columns to the table without aggregating it. This can be done with the function `mutate()`.
- *Summarize* or *aggregate* the grouped data. This creates new variables at a higher level of grouping. For example we might calculate means with `mean()` or counts with `n()`. This results in a smaller, summary table, which we might do more things on if we want. This can be done within the function `summarize()`.

We use the `dplyr` functions `group_by()`, `filter()`, `select()`, `mutate()`, and `summarize()` to carry out these tasks within our *pipeline*. They are written in a way that allows them to be easily piped. That is, they understand how to take inputs from the left side of a pipe operator and pass results along through the right side of one. The dplyr documentation has some useful vignettes that introduce these grouping, filtering, selection, and transformation functions. There is also a more detailed discussion of these tools, along with many more examples, in Wickham & Grolemund (https://r4ds.had.co.nz/).

We will create a new table called `rel_by_region`. Here’s the code:

```
(rel_by_region <- gss_sm %>% #Create a new object, rel_by_region from the gss_sm data
    group_by(bigregion, religion) %>% # Subsequently group the rows by bigregion and,
  #within that, by religion
    summarize(N = n()) %>% #Summarize this table to create a new, much smaller table,
  #with three columns: bigregion, religion, and a new summary variable, N, that is a
```

```
  #count of the number of observations within each religious group for each region.
    mutate(freq = N / sum(N), # With this new table, use the N variable to calculate
  #the relative proportion (freq)
           pct = round((freq*100), 0))) #and percentage (pct) for each religious
```

```
## # A tibble: 24 x 5
## # Groups:   bigregion [4]
##    bigregion religion       N    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324      32
##  2 Northeast Catholic     162 0.332      33
##  3 Northeast Jewish        27 0.0553      6
##  4 Northeast None         112 0.230      23
##  5 Northeast Other         28 0.0574      6
##  6 Northeast <NA>           1 0.00205     0
##  7 Midwest   Protestant   325 0.468      47
##  8 Midwest   Catholic     172 0.247      25
##  9 Midwest   Jewish         3 0.00432     0
## 10 Midwest   None         157 0.226      23
## # ... with 14 more rows
```

```
  #category, still grouped by region. Round the results to the nearest percentage
  #point

### Putting brackets () around a statement with a data set, makes R visualize the
### first 10 rows
```

In this way of doing things, objects passed along the pipeline and the functions acting on them carry some assumptions about their context. For one thing, you don't have to keep specifying the name of the underlying data frame object you are working from. Everything is implicitly carried forward from `gss_sm`. Within the pipeline, the transient or implicit objects created from your summaries and other transformations are carried through, too.

When trying to grasp what each additive step in a `ggplot()` sequence does, it can be helpful to work backwards, removing one piece at a time to see what the plot looks like when that step is not included. In the same way, when looking at pipelined code it can be helpful to start from the end of the line, and then remove one `%>%` step at a time to see what the resulting intermediate object looks like.

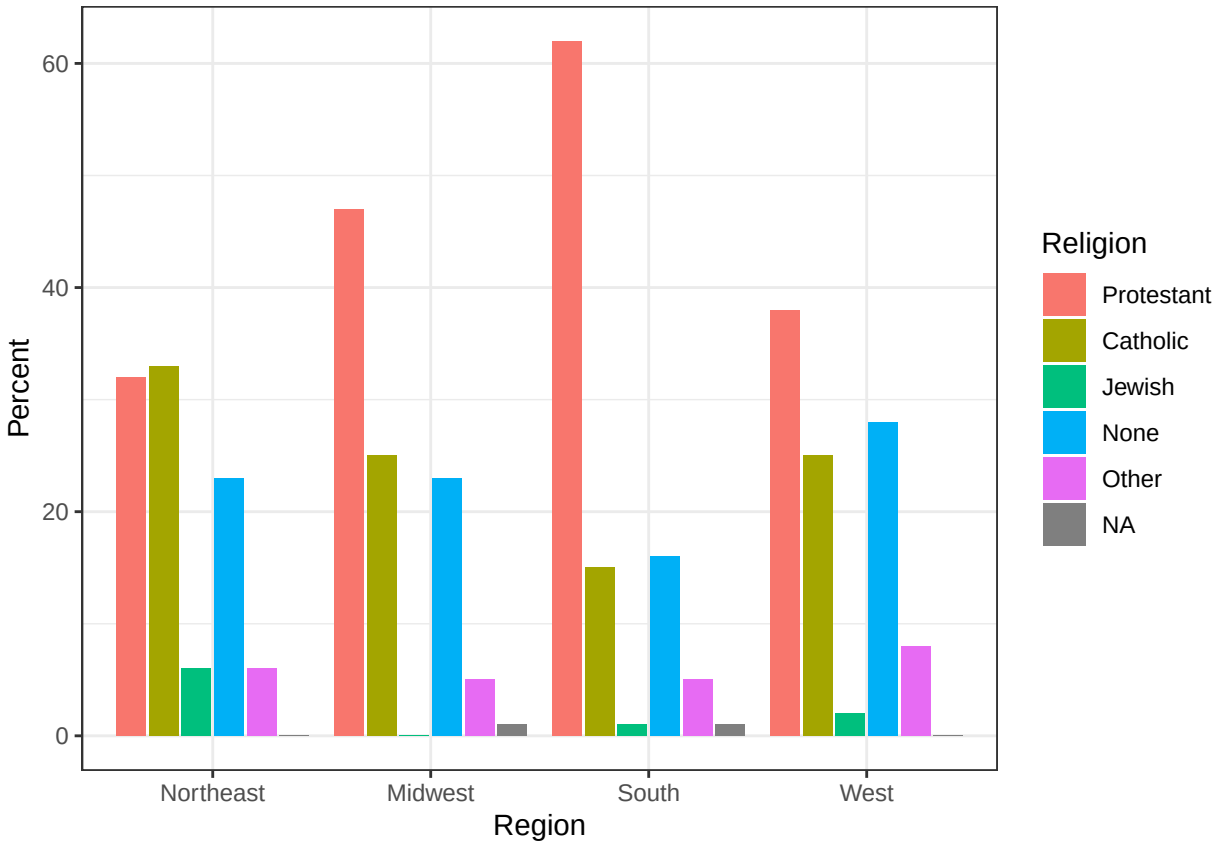
Assignment 1: Deconstructing a Pipeline

1a) What happens if we remove the `mutate()` step from the code above? Provide the script and a table in your assignment.

1b) Subsequently, what happens if you also remove `summarize()` step? How big is the table returned at each step? What level of grouping is it at? What variables have been added or removed?

1c) Replicate the code below. `geom_col()` is used instead of `geom_bar()`. What is changed in the plot if you use `geom_bar()`?

```
ggplot(rel_by_region, aes(x = bigregion, y = pct, fill = religion)) +
  geom_col(position = "dodge2") +
    labs(x = "Region",y = "Percent", fill = "Religion") +
    theme(legend.position = "top") +
  theme_bw()
```



1d) Moreover `dodge2` *is used instead of* `dodge` *like in last week's assignment. What is the difference?*

1e) Add the functions `coord_flip()` *and* `facet_grid()` *to the* `ggplot` *object. Explain what happens and whether this aids or impedes the interpretation of the visual.*

Continue with Grouping

Let's move to a new dataset, the `organdata` table. Like `gapminder`, it has a country-year structure. It contains a little more than a decade's worth of information on the donation of organs for transplants in seventeen OECD countries. The organ procurement rate is a measure of the number of human organs obtained from cadaver organ donors for use in transplant operations. Along with this donation data, the dataset has a variety of numerical demographic measures, and several categorical measures of health and welfare policy and law. Unlike the `gapminder` data, some observations are missing. These are designated with a value of `NA`, R's standard code for missing data. The `organdata` table is included in the `socviz` library. Load it up and take a quick look. Instead of using `head()`, for variety this time we will make a short pipeline to select the first six columns of the dataset, and then pick five rows at random using a function called `sample_n()`. This function takes two main arguments. First we provide the table of data we want to sample from. Because we are using a pipeline, this is implicitly passed down from the beginning of the pipe. Then we supply the number of draws we want to make.

```
organdata %>% select(1:6) %>% sample_n(size = 10)
```

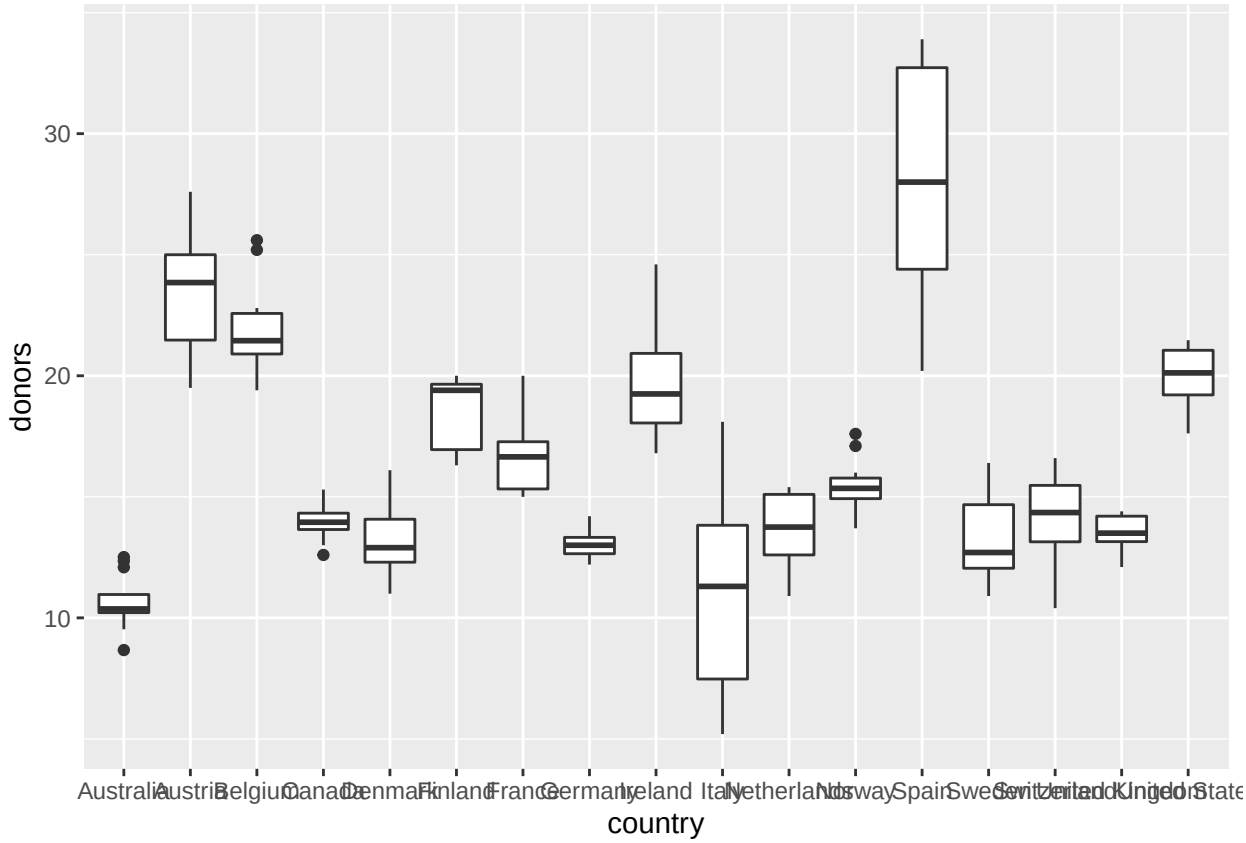
```
## # A tibble: 10 x 6
##    country        year       donors    pop pop_dens   gdp
##    <chr>          <date>      <dbl>  <int>    <dbl> <int>
##  1 Australia      1998-01-01   10.5  18711    0.242 24148
##  2 France         1995-01-01   15.1  57844   10.5   21283
##  3 Switzerland    2002-01-01   10.4   7290   17.7   30725
```

```
##  4 Switzerland    2000-01-01   14     7184   17.4   29837
##  5 Sweden         NA           NA     8559    1.90  18660
##  6 Netherlands    1997-01-01   14.4  15611   37.6   23753
##  7 Australia      2000-01-01   10.2  19153    0.247 26545
##  8 United Kingdom 1995-01-01   14.4  58005   23.9   19998
##  9 Germany        1997-01-01   13.2  82035   23.0   22589
## 10 United States  2000-01-01   21.2 282224    2.93  34590
```

Let's imagine we are interested in country-level variation, without paying attention to the time trend. We can use `geom_boxplot()` to get a picture of variation by year across countries. Just as `geom_bar()` by default calculates a count of observations by the category you map to `x`, the `stat_boxplot()` function that works with `geom_boxplot()` will calculate a number of statistics that allow the box and whiskers to be drawn. We tell `geom_boxplot()` the variable we want to categorize by (here: `country`) and the continuous variable we want summarized (here: `donors`).

```
ggplot(data = organdata,
       mapping = aes(x = country, y = donors)) +
  geom_boxplot()
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

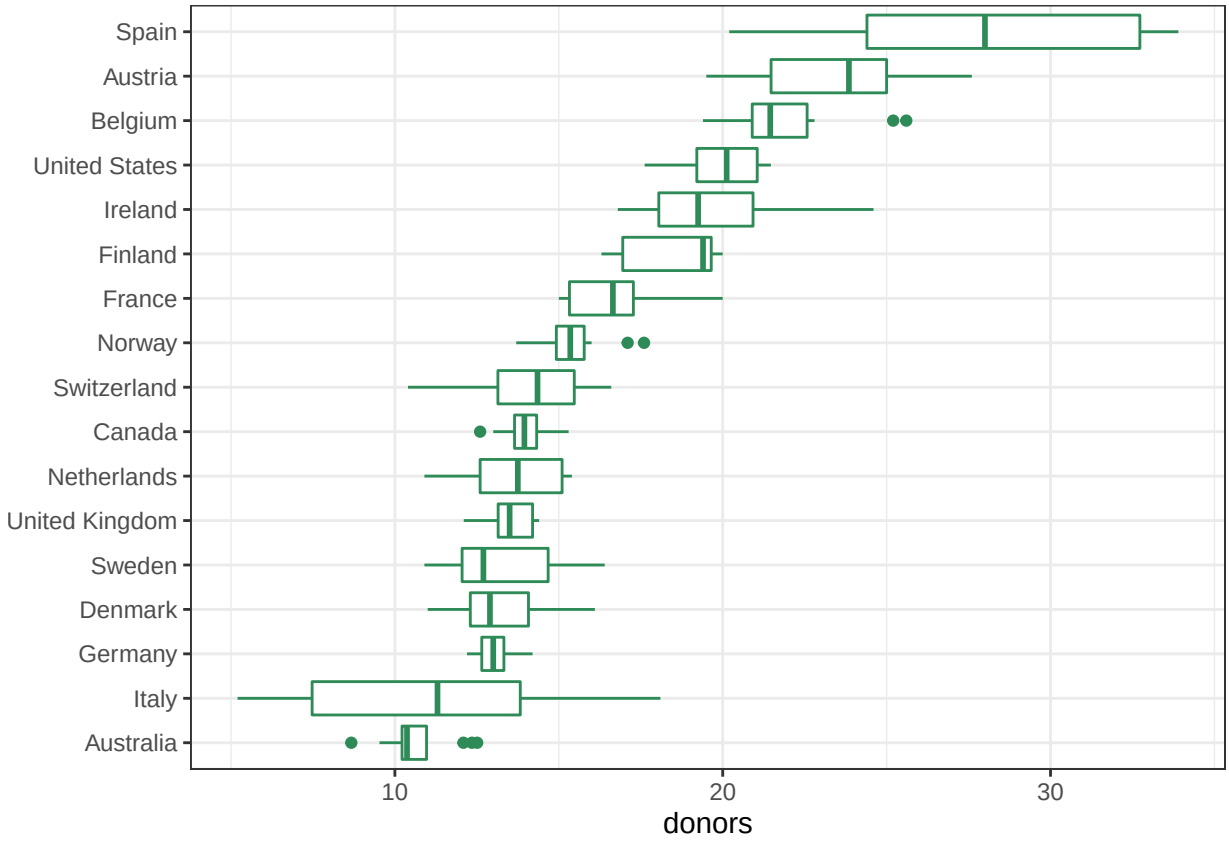


The boxplots look interesting but two issues could be addressed. First, as we saw in the previous chapter, it is awkward to have the country names on the x-axis because the labels will overlap. We should use `coord_flip()` again to switch the axes (but not the mappings). While that will make the graph more legible, it's still not ideal. We generally want our plots to present data in some meaningful *order*. An obvious way is to have the countries listed from high to low average donation rate. We accomplish this by reordering the country variable by the mean of donors. The `reorder()` function will do this for us. It takes two required arguments:

1. The categorical variable or factor that we want to reorder. In this case, that's country.
2. The variable we want to reorder it by. Here that is the donation rate, donors.

The third and optional argument to `reorder()` is the function you want to use as a summary statistic. If you only give `reorder()` the first two required arguments, then by default it will reorder the categories of your first variable by the mean value of the second. You can name any sensible function you like to reorder the categorical variable (e.g., `median`, or `sd`). There is one additional wrinkle. In R, the default `mean()` function will fail with an error if there are missing values in the variable you are trying to take the average of. You must say that it is OK to remove the missing values when calculating the mean. This is done by supplying the `na.rm=TRUE` argument to `reorder()`, which internally passes that argument on to `mean()`. We are reordering the variable we are mapping to the `x` aesthetic, so we use `reorder()` at that point in our code:

```
ggplot(data = organdata,
       mapping = aes(x = reorder(country, donors, na.rm=TRUE),
                     y = donors)) +
  geom_boxplot(colour = "seagreen") +
  labs(x=NULL) +
  coord_flip() +
  theme_bw()
```



Assignment 2: Apply Transformations & Different Ways of Visualizing Data

2a) Replicate the code above, but used `geom_violin()` *instead of* `geom_boxplot()`

2b) What is the difference with the `geom_boxplot()`*? What do the shapes of the* `geom_violin()` *tell you?*

2c) The `subset()` *function is very useful when used in conjunction with a series of layered geoms. Go back to the book's code for the* Presidential Elections plot *(Figure 5.18) and redo it so that it shows all the data*

points but only labels elections since 1992. You might need to look again at the `elections_historic` *data to see what variables are available to you.*

2d) Continue with the plot created in 2c and colour the data points to reflect the winning party.

2e) Use `geom_point()` *and* `reorder()` *to make a Cleveland dot plot (e.g. Figure 5.13) of all Presidential elections, ordered by share of the popular vote.*

2f) Install the packages `ggthemes` *and replicate the graph of 2e with various themes. What kind of backgrounds aid interpretation? Why? Include the code and plots in the assignment.*

Good Luck

The deadline for the assignment is **November 21, 10am**!

References

Healy, K. (2018). *Data visualization: a practical introduction.* Princeton University Press.