QStep — Week 4 Lab: Data Visualisation and Wrangling

Leonardo Carella

1. Installing and Loading Packages

Packages are bundles of code and data that the R community has produced, and that you can use in your own code to expand the functionalities of R. They are currently over 19,000 of them available on the CRAN repository (https://cran.r-project.org/web/packages/available_packages_by_name.html), ranging from packages for cutting-edge statistical modelling to repositories of data on members of parliament, to packages like `fortunes`, whose only function is to return random fortune cookie-style quotes. In most cases, the point of installing packages is using additional functions that base R does not have.

Today we are going to install and use `tidyverse`: the most popular collection of packages in R. In particular, we will use two packages that come with the `tidyverse` bundle: `ggplot2`, which is widely used for data visualisation, and `dplyr`, which will come handy to manipulate dataframes. `tidyverse` has developed something of its own language, and often provides alternative ways of doing things in R than the 'base' R language we learnt last time (for instance, `tidyverse` almost never uses the `$` operator). In practice, most R users these days are aware of both options, and use a combination of `tidyverse` and base according to their needs.

To install packages from the CRAN repository, which is where the things you need are going to be in 99% of the cases, you simply have to pass the name of the package as a character in the `install.packages()` function:

```
install.packages("tidyverse")
```

At this point, you may get a message in the console asking you if you want to install dependencies (other packages that are needed to make your package work). Just type 'yes' in the console, and leave it to compile until it says it's done. The package is now installed on your device's library. You won't have to run this command again in the future for `tidyverse`. An alternative way to install packages is to click on the 'Packages' window on the bottom-right panel of your scree, then selecting 'Install', and searching manually for the package you need. Make sure you have 'install dependencies' ticked.

To use `tidyverse` in this R session, we need to load it form our library, with the function:

```
library(tidyverse)
```

You will have to load your package from your library in every R session where you want to use it (otherwise the functions within it won't work).

2. The Quality of Government 2022 dataset

Recall that last time we set our working directory to the folder where you downloaded the `brexit.csv`. This time, we are going to work with a dataset of social, political and economic indicators for all the coun-

tries of the world: the Quality of Government dataset. The file you should have downloaded, `qog2022.csv` is a slightly cleaned up version of the 'QOG Basic Dataset 2022'.

Once again, set your working directory: select the 'Session' drop-down menu at the top of the RStudio interface, select `Set Working Directory`, then select `Choose Directory`, then search for the folder you've saved the file in, and click `Open`. Something like this should appear in your console:

```
setwd("~/Desktop/QStep")
```

Alternatively, you can manually set the working directory by copying the file path of the dataset you want to import and passing it through the `setwd()` function. Make sure to enclose it within quote marks. It will look something like this:

```
setwd("/Users/yourname/Desktop/QStep")
```

Now load the data, and store it as an object called `qog` (for 'quality of government'):

```
qog <- read.csv("qog2022.csv")
```

The dataset is maintained by a team of researchers at the University of Gothenburg You can learn more about the project, download more data etc. at the Quality of Government institute website (https://www.gu.se/en/quality-government). A codebook has been provided alongside the data, detailing the variables included in the full database. Moreover, the original source for each of these indicators has been provided in the codebook (say, the UN, or the 'Freedom House' think tank).

The version you just loaded onto the R environment contains 242 variables, covering scores and statistics on themes from corruption to health, from the economy to gender party. The variables we are going to work with in this lab and in the homework are the following:

Variable	Description
`cname`	Country name
`ccodealp`	Country code (isocode-3)
`region`	UN Region (continent)
`fh_status`	Freedom House Rating (Free, Partly Free, Not Free)
`wdi_wip`	Proportion of women in parliament (World Development Indicators)
`undp_hdi`	Human Development Index (UN Development Programme)
`dr_ig`	Index of Globalisation (1-100 scale: higher = more globalised)
`fh_ipolity2`	Freedom House Polity Score (0-10 scale: higher = more democratic)
`mad_gdppc`	Real GDP per capita (Maddison Project)
`vdem_corr`	V-Dem corruption index (higher = more corr.)

Later on in the lab, we are also going to work with the `ppi.csv` dataset. This contains a `ccodealp` country code variable, a `country` name variable, and a `parl_power_index` variable with the values of the 'Parliamentary Powers Index' scores computed by Fish and Kroeger. For now, make sure that this additional dataset is in your working directory.

3. Visualising Distributions: Barplots and Histograms

Data visualisation is a core, if often overlooked, skill of modern data scientists. Our brains tend to pull information from visual patterns much more easily than from verbal or numerical abstractions. Being able to turn numbers into images allows in first instance the researcher to gain insights into the data they are working with, and then to convey effectively and succinctly those insights to their audience.

Often at the start of our analysis we find ourselves in the situation where we would like to understand how a variable is *distributed* in a dataset. Visualisation is very effective at describing data; but to pick the right approach, we should first understand the type of variable we are working with. There are, of course, many valid ways to plot the same data (visualisation is part art and part science), but for the sake of brevity we are going to stick to the most commonly used. For categorical variables – those that convey no obvious mathematical meaning – we are going to use *barplots*. For continuous variables – those that express real quantities – we are going to look at *histograms* in the lab, as well as *box plots* and *violin plots* (covered in the homework).

Let's use the `ggplot` function to make a barplot of the variable `region` in the dataframe `qog`. Remember, you need to have installed and loaded `tidyverse` for the following code to work:

```
ggplot(data = qog) +
  geom_bar(mapping = aes(x = region))
```

What's going on here? In `ggplot2`, you always start a plot with the function `ggplot()` and pass your dataframe as the argument; this creates an empty graph to which you can add layers. In this case we added a layer with the `+` operator and the function `geom_bar()`. `geom_bar()` tells R that the layer we are building is a barplot. Within the function we specify the `aes` (aesthetics) arguments: in this case, the only aesthetic parameter is the position; i.e. we are asking R to put the variable `region` in the x-axis.

What if the variable we want to describe is continuous? Take for instance the variable `wdi_wip` (proportion of national parliament seats held by women). It obviously wouldn't make sense to have a bar of height one for each individual value (27.9, 25.8, 32.1, 30.5, 11.1 etc.) that appears in our dataset. Histograms solve this problem by *binning* numbers that are close to each others into rectangles of equal width. Let's make one:

```
ggplot(data = qog) +
  geom_histogram(mapping = aes(x = wdi_wip))
```

What information can you gather from this visualisation?

Note that the width of the bins is arbitrary: `ggplot` tends to choose a sensible value, but you can specify how thin or chunky you want the bins to be by passing the argument `binwidth` within the `geom_histogram()` function:

```
ggplot(data = qog) +
  geom_histogram(mapping = aes(x = wdi_wip), binwidth = 10)
```

Finally, use the functions `ggtitle`, `xlab` and `ylab` to label your plot. Remember to add them as layers with the `+` sign, and to use quote marks to create text objects.

```
ggplot(data = qog) +
  geom_histogram(mapping = aes(x = wdi_wip), binwidth = 10) +
  ggtitle("Women's Representation in World Parliaments") +
  xlab("percentage of seats held by women") +
  ylab("number of countries")
```

To save a plot you just made in `ggplot2`, simply use the `ggsave()` function. Use a string to give your plot a name (it supports formats like `.jpeg`, `.pdf`, `.png`). You can also set the dimensions of the plot with the arguments `width =` and `heigth =`; units are (inexplicably) inches. The plot will appear in your working directory.

```
ggsave("my_first_plot.jpg")
```

4. Faceting

Sometimes you want to visualise how a variable is distributed across a second (categorical) variable. For instance, is the geographical distribution of countries in my dataset different for democracies and non-democracies? A simple way of conveying information from two different variables at once is *faceting*. Let's try this with the categorical variable `fh_status`, which records the Freedom House rating of countries as 'Free', 'Partly Free' and 'Not Free'. Go back to the region barplots, and add the layer `facet_wrap(~fh_status)` (read the tilde sign `~` as 'by')

```
ggplot(data = qog) +
  geom_bar(mapping = aes(x = region)) +
  facet_wrap(~fh_status)
```

To make the plot more easily readable, we can also use the 'fill' aesthetic within the `geom_bar` function, so as to make the bars of a different colour according to the value of the variable `region`. (Note: `ggplot2` uses the language of `fill` when it comes to colouring in areas, and `color` or `colour` when it comes to lines and points; more about this in the homework).

```
ggplot(data = qog) +
  geom_bar(mapping = aes(x = region, fill = region)) +
  facet_wrap(~fh_status)
```

Of course, you can use faceting with histograms as well:

```
ggplot(data = qog) +
  geom_histogram(mapping = aes(x = wdi_wip), binwidth = 10) +
  ggtitle("Women's Representation in World Parliaments") +
  xlab("percentage of seats held by women") +
  ylab("number of countries") +
  facet_wrap(~region)
```

5. Scatter plots

Faceting is useful to show how a categorical variable (region) varies across another categorical variable (democracy) or a continuous one (women's representation). But what if we wanted to show how two continuous variables vary with each other? Scatter plots are the obvious solution. To make one in `ggplot2`, you want to use the `geom_point()` function, and to specify both the `x` and the `y` variables in `aes()`. Let's see for example how a continuous measure of human development `undp_hdi` (a composite index of health, education and standards of living) is related to our women-in-parliament variable.

```
ggplot(data = qog) +
  geom_point(mapping = aes(x = undp_hdi, y = wdi_wip)) +
  xlab("Human Development Index") +
  ylab("Percentage of Women in Parliament")
```

If you want to add information from a *third* variable to the scatter plot, you can use colour-coding (though it might get messy) within `aes()` or use faceting (probably more sensible).

```
ggplot(data = qog) +
  geom_point(mapping = aes(x = undp_hdi, y = wdi_wip, color = region)) +
  xlab("Human Development Index") +
  ylab("Percentage of Women in Parliament")
```

```
ggplot(data = qog) +
  geom_point(mapping = aes(x = undp_hdi, y = wdi_wip)) +
  xlab("Human Development Index") +
  ylab("Percentage of Women in Parliament") +
  facet_wrap(~region)
```

How would you characterise the relationship between human development and female representation in parliament?

Another approach that can be useful to 'eyeball' the data is to plot country names/codes instead of points. This can give you an idea of what kind of observations we encounter in different parts of the plot. This can be done with the `geom_text()` function, with an additional `label` argument to pass within `aes()`. In this case we'll use the `ccodealp` variable, which stores 3-character country codes. The parameter `size` controls the size of the font. Can you guess which country has the highest female representation in their national parliament?

```
ggplot(data = qog) +
  geom_text(mapping = aes(x = undp_hdi, y = wdi_wip, label = ccodealp),
            size = 3) +
  xlab("Human Development Index") +
  ylab("Percentage of Women in Parliament")
```

6. Data wrangling 1: Filtering and Selecting

The `dplyr` package that comes with `tidyverse` allows you to manipulate dataframes with only a few commands. A particularly useful operator is the pipe operator `%>%` (read: 'and then'). This allows you

to concatenate functions. For instance, let's take the `filter()` function. This allows you to reduce a dataset conditional on the value of some variable. For instance, if we wanted to reduce `qog` to a dataset with only Asian countries, we would run:

```
qog %>% filter(region == "Asia")
```

This can be used in combination with the `<-` operator to store the new dataframe under some new name:

```
asia <- qog %>%
  filter(region == "Asia")
```

Now say we wanted to reduce the number of columns in the dataset, as opposed to the number of rows. In this case, we would select the columns we want with the function `select()`:

```
qog %>% select(cname, region, wdi_wip, undp_hdi)
```

With the pipe we can concatenate functions like `filter()` and `select()` in a single chunk of code:

```
high_hdi_democracies <- qog %>%
  filter(undp_hdi > 0.9 & fh_status == "Free") %>%
  select(cname, region, wdi_wip, undp_hdi)

high_hdi_democracies
```

7. Data wrangling 2: Merging Dataframes

Another helpful set of commands in `dplyr` allow you to **merge** two dataframes. This will be very useful if you want to collect more information for your own research, and add it to existing datasets. Given two dataframes `x` and `y` sharing a common variable `matching_variable`, you have various options:

- `inner_join(x, y, by = "matching_variable")`: includes all rows in x **and** y.
- `left_join(x, y, by = "matching_variable")`: includes all rows in x.
- `right_join(x, y, by = "matching_variable")`: includes all rows in y.
- `full_join(x, y, by = "matching_variable")`: includes all rows in x **or** y.

Let's try and merge the `qog` dataset we already have with the dataset in `ppi.csv`, which you should have as part of the material for this lab. First, let's load `ppi`, which contains the "Parliamentary Power Index", a score recording the formal powers of the national legislature compiled by Steven Fish and Matthew Kroenig for their book 'The Handbook of National Legislatures: A Global Survey' (New York: Cambridge University Press, 2009). Conveniently, the dataset includes a variable `ccodealp` with standard three-letter alphabetic country codes that match exactly those in `qog`. For instance, both datasets have a row with the code "GBR" (for Britain), "USA" (for the United States), "BRA" (for Brazil) etc. The join functions match those rows and merge them into a new dataset with columns from both original dataframes.

Let's first get the data in, and have a look at it:

```
ppi <- read.csv("ppi.csv")

dim(ppi)
colnames(ppi)
View(ppi)
```

Now let's use `full_join()` to merge `qog` and `ppi` into a new data frame:

```
new_dataframe <- full_join(qog, ppi, by = "ccodealp")
```

How many rows and columns does the new dataframe have? How can we visualise the relationship between the Human Development Index variable from `qog` and the Parliamentary Power Index (`parl_power_index`) variable from `ppi`?

```
ggplot(data = new_dataframe) +
  geom_point(mapping = aes(x = undp_hdi, y = parl_power_index)) +
  xlab("Human Development Index (WDI)") +
  ylab("Parliamentary Power Index (Fish-Kroeger)")
```

Homework

- Load the `brexit` data from week 2's lab. Choose and implement a suitable data visualisation for describing (1) the `region` variable, (2) the `percent_degree` variable, (3) the `percent_degree` variable separately for majority-Leave and majority-Remain areas, (4) the relationship between `percent_degree` and `percent_leave`.
- Box and Violin Plots

Box plots and violin plots serve a similar function as histogram, in that they show the distribution of a continuous variable. However, these approaches tend to be more synthetic and therefore lend themselves to visualisations that *compare* various distributions. Run the following code for a box plot of the `wdi_wip` variable by UN region. Then repeat the same syntax substituting `geom_boxplot()` with `geom_violin()` for a violin plot.

```
ggplot(data = qog) +
  geom_boxplot(mapping = aes(x = region, y = wdi_wip)) +
  xlab("Region") + ylab("Percentage of women in parliament")
```

To understand what information box and violin plots are conveying refer to Claus Wilke's excellent chapter on these types of plots: https://clauswilke.com/dataviz/boxplots-violins.html#boxplots-violins-vertical. There's also a useful description of the information conveyed by a box plot in the textbook, 'The Fundamentals of Political Science Research', in section 6.4.1.

- Colour scales in `ggplot`

Colour in plots is a very powerful aesthetic to convey additional information:

1. It can be used as a *qualitative tool* to distinguish categories. Use the data in the `qog` database to build a scatterplot with GDP per capita (`mad_gdppc`) on the x-axis, the V-Dem corruption indicator (`vdem_corr`) on the y-axis, color-coded by Freedom House rating (`fh_status`).

2. It can also be used to *represent quantities on a scale*. Using the same scatterplot, swap `fh_status` (a categorical variable) for `fh_ipolity2` (a continuous indicator of democracy).

A word of caution: where possible, it is good practice to avoid conveying information purely through colour. People see colour differently from each other, and about 1 in 20 people are colourblind in some way. While there are colour combinations that do 'work' for most people (see: https://cran.r-project.org/web/packages/colorBlindness/vignettes/colorBlindness.html), alternative solutions are generally a safer bet. For instance, how would you convey the information in the scatterplots above without relying on colour?

Resources for Further Self-Learning

If you're enjoying working with data, at this point you should have just about enough familiarity with R language to try your hand at some self-learning. This is, of course, entirely optional. In your own time, you could try working through **R for Data Science**, a free online textbook by Hadley Wickham (the statistician who developed `ggplot2` and `dplyr`) and Garret Grolemund, available at https://r4ds.had.co.nz/index.html. At this point, you should be able to work through chapters 3 (data visualisation), 4 (workflow: basics) and 5 (data transformation). After next lab, you should also try chapter 7 (exploratory data analysis). Subsequent chapters are beyond the scope of an introductory course (but they're not rocket science either).

If you're particularly interested in data visualisation, you could have a look at Wickham, Navarro and Pedersen's **ggplot2: Elegant Graphics for Data Analysis**, a relatively advanced book-length treatment of the various uses of `ggplot2`. It is available at https://ggplot2-book.org/.

We have only scratched the surface in terms of all the options that `dplyr` offers for data wrangling. `filter()`, `select()` and the `_join()` family of functions — alongside the other 'base R' techniques we've learnt last time — should be enough for your immediate needs. However, in your own time you might consider looking at how to use the functions `mutate()`, `summarise()`, `arrange()` and `group_by()`. You could start with this tutorial: https://genomicsclass.github.io/book/pages/dplyr_tutorial.html.