Lab 3: More Data Loading and Data Visualization

Tom Hanna

2026-02-20

Table of contents

1 Getting ready

1.1 Proper project

Be sure you are in the project for the class github repository. You can check this by looking at the top of the RStudio window, it should show the name of the project as "pols3316_spring2026". If not, you can open the project by going to `File > Open Project` and selecting the `pols3316_spring2026.Rproj` file in the folder where you cloned the repository or by going to `File > Recent Projects` and selecting it from the list.

1.2 Clear the console and environment

Next, we are going to clear the console and remove all objects from the environment.

To clear the console, you can use the shortcut `Ctrl + L` (Windows/Linux) or `Cmd + L` (Mac).

To clear the environment, you can use the broom icon in the upper right or use following command in R by typing it in the console:

```r
# this command erases all objects from the environment

rm(list = ls())
```

1.3 Git Pull

Last, we are going to do a *"Git Pull"* to update the local repository with the latest material from the class github repository on GitHub.

You can do this by going to the "Git" tab in the upper right pane of RStudio and clicking the "Pull" button or by using the following command in R by typing it in the console:

```r
# this executes a git pull

system("git pull")
```

If you get errors, it may be because you have uncommitted changes in your local repository. If this is the case, you can either move them out of the folder or *stash* them before doing the pull using the following code.

```
# 1) Temporarily save local changes
git stash save "my local work"

# 2) Get the latest changes
git pull origin main

# 3) Re-apply your local work on top
git stash pop
```

1.4

Now, let's create a new R script file for this lab. You can do this by going to `File > New File > R Script`. Save the new file in your `My_Labs` folder with the name `lab_3.R`. (If you named your folder something different, just save it there.)

2 More Data Loading

Before we start data visualization, we are going to reload some of the data we worked with before and explore a couple of other ways to load data. This will give us several sources of data for the data visualization step.

2.1 Lab 1 Data Loading

In Lab 1, we created a data frame called `my_data` and saved it to a CSV file in the `data` folder. We are going to load that data back into R like we did in Lab 2, but this time we are going to use a slightly more efficient function from the tidyverse called `read_csv()` from the `readr` package.

First, you will need to load the `readr` package by typing the function in your script and running it.

```
# this command loads the reader package and needs to be in the script

library(readr)
```

If you get a *"there is no package called 'readr'"* error, you will need to install the package first by running the following command in the console (since you only want to do this once, not every time the script runs).

```
# this command installs the readr package and only needs to be run once, so
# run it in the console, not in the script

install.packages("readr")
```

Now you can use the `read_csv()` function to load the data from the CSV file into a new data frame called `my_data_loaded`.

```
# This command will load the data from the csv file into a new data frame
# called my_data_loaded

my_data_loaded <- read_csv("data/my_data.csv")
```

2.2 Web data loading

In Lab 2, we also loaded some data from Posit/R Studio developer Jenny Bryan, who famously threatened to set people's computers on fire if they kept putting rm(list = ls()) in their scripts.

Here is the code you need in one block. Remember that you can't copy/paste quotation marks or other punctuation from Word to R Studio:

```
# load the tidyverse

library(tidyverse)

# define the web address or URL to the Gapminder dataset

url <-
"https://raw.githubusercontent.com/jennybc/gapminder/master/inst/extdata/gapm
inder.tsv"

# read the data from the URL into a dataframe called gapminder_data

gapminder_data <- read_tsv(url)
```

2.3 The V-Dem Project Package

The Varieties of Democracy Project (V-Dem) is a research project that provides high-quality data on democracy and governance around the world. They have an R package called `vdemdata` that allows you to easily access their data directly from R and another for their Episodes of Regime Transition Dataset as well. We're going to practice with the `vdemdata` package, which contains the main V-Dem dataset.

One of the lessons of the `vdemdata` package is that it is not on CRAN, so you have to install it from GitHub using the `remotes` package. You can install the `remotes` package from CRAN since you probably don't have it already and then use it to install the `vdemdata` package from GitHub.

```
# install the remotes package if you don't have it already
# do this in the console, since you only want to do it once
```

```
install.packages("remotes")
```

We will need to load the library in the console, since this is just for installing and we only need to do it once:

```
# load the remotes package with the libary() function

library(remotes)
```

Now you can use the `install_github()` function in the console from the `remotes` package to install the vdemdata package from GitHub.

```
install_github("vdeminstitute/vdemdata") # in the console
```

Next, we will load the `vdemdata` package in our script so we can use it to access the V-Dem data and then add some of the data from the package to a dataframe in our environment.

We could load the main V-Dem dataset into a dataframe called vdem_data with the following command, but it is very large, so we will just load a subset of it instead `vdem_data <- vdemdata::vdem`.

```
# load the vdemdata package
library(vdemdata)

# load the 5 main vdem indices and some economic data for the year 2014 into
a dataframe called vdem_indices_2014
vdem_indices_2014 <- vdemdata::vdem %>%
  filter(year == 2014) %>%
  select(country_name, v2x_polyarchy, v2x_libdem, v2x_partipdem,
v2x_delibdem, v2x_egaldem, e_gdppc, e_cow_exports, e_cow_imports, e_gdp,
e_pop)

View(vdem_indices_2014)
```

The %>% symbol is called the "pipe" operator. It comes from the `tidyverse` and it allows you to chain together multiple operations in a more readable way. In this case, we are filtering the V-Dem dataset to only include rows where the year is 2023 and then selecting only the columns we are interested in.

Next, since we are going to be doing some data visualization, it would be great if those variable names were a bit more meaningful. We can use the `rename()` function from the `dplyr` package to rename the columns in our dataframe to something more descriptive.

```
# rename the columns in the vdem_indices_2014 dataframe to something more
descriptive

vdem_indices_2014 <- vdem_indices_2014 %>%
  rename(
    country = country_name,
    polyarchy = v2x_polyarchy,
    liberal_democracy = v2x_libdem,
    participatory_democracy = v2x_partipdem,
    deliberative_democracy = v2x_delibdem,
    egalitarian_democracy = v2x_egaldem,
    gdp_per_capita = e_gdppc,
    exports = e_cow_exports,
    imports = e_cow_imports,
    gdp = e_gdp,
    population = e_pop
  )
```

Here we are assigning the output of the `rename()` function back to the `vdem_indices_2023` dataframe, which will overwrite the original dataframe with the new column names. Now we have a dataframe with more meaningful column names that we can use for our data visualization in the next step.

3 Data Visualization

Now that we have some data loaded into R, we can start doing some data visualization. We will be using the `ggplot2` package from the `tidyverse` for this.

```
# load ggplot2

library(ggplot2)
```

We are going to start with a couple of really simple examples from our simplest data. First, a histogram of the values in X5 in the Lab 1 data.

```
# create a histogram of the values in X5 in my_data_loaded


ggplot(my_data_loaded, aes(x = X1)) +
        geom_histogram(binwidth = 1) +
        labs(title = "Histogram of X5", x = "X5", y = "Frequency")
```

ggplot() is the main function in the `ggplot2` package for creating plots. The first argument is the data frame we want to use, and the second argument is the aesthetics (aes) which tells ggplot which variables to use for the x and y axes. In this case, we are using X5 for the x-axis and we are not specifying a y-axis because we are creating a histogram.

Next, geom_histogram() is the function that tells ggplot we want to create a histogram. The binwidth argument specifies the width of the bins in the histogram. You can adjust this to make the histogram look better depending on your data. A smaller binwidth will create more bins and a larger binwidth will create fewer bins. You can experiment with different values to see what looks best for your data.

Finally, labs() is a function that allows us to add labels to our plot, including a title and labels for the x and y axes.

Next, we will do a simple scatterplot of X1 versus X2 in the Lab 1 data.

```
# from my_data_loaded create a simple scatterplot of X1 versus X2

ggplot(my_data_loaded, aes(x = X1, y = X2)) +
        geom_point() +
        labs(title = "Scatterplot of X1 vs X2", x = "X1", y = "X2")
```

Again, we start with the basic ggplot() function, specifying the data frame and the aesthetics - aes() - for the x and y axes. This time we are using X1 for the x-axis and X2 for the y-axis. We are using a different geometry function, geom_point(), which tells ggplot we want to create a scatterplot. Finally, we are adding labels to the plot with labs().

Next, we'll create a dot plot.

```
# create a simple dot plot of gdp per capita from vdem_indices_2014

ggplot(vdem_indices_2014, aes(x = gdp_per_capita)) +
        geom_dotplot(binwidth = 5) +
        labs(title = "Dot Plot of GDP per Capita", x = "GDP per Capita", y = 
"Frequency")
```

The first difference from our previous plots is the geometry function, which is now geom_dotplot() instead of geom_histogram() or geom_point(). The y-axis is not specified because it is a dot plot, which does not have a y-axis in the traditional sense. Finally, we are adding labels to the plot with labs(). The binwidth argument specifies the width of the bins in the dot plot, which represents the number of observations in each bin.

This is very similar to a histogram, but it uses dots instead of bars to represent the frequency of observations in each bin.

```
# create a histogram of the gdp per capita variable logged from 
vdem_indices_2014

ggplot(vdem_indices_2014, aes(x = gdp_per_capita)) +
        geom_histogram(binwidth = 5) +
        labs(title = "Histogram of Log GDP per Capita", x = "Log GDP per 
Capita", y = "Frequency")
```

Here, we change the geometry function to geom_histogram() to create a histogram, and we specify the x-axis as the log of gdp_per_capita. The binwidth argument specifies the width of the bins in the histogram, which represents the number of observations in each bin. Finally, we are adding labels to the plot with labs().

Part of good data visualization is using contrast to show differences and also just making the plot appealing to the eye.

```
# create a histogram of the polyarchy variable from vdem_indices_2014 with
some color

ggplot(vdem_indices_2014, aes(x = polyarchy)) +
        geom_histogram(binwidth = 0.1, fill = "blue", color = "black") +
        labs(title = "Histogram of Polyarchy", x = "Polyarchy", y =
"Frequency")
```

Here, fill specifies the color of the bars in the histogram and color specifies the color of the borders of the bars. You can experiment with different colors to see what looks best for your data.

One of our major goals for this semester is to fit a linear equation to our data with the lowest possible error. We can have the ggplot package do this graphically so we can visualize it.

```
# create a scatter plot of gdp per capita  vs polyarchy
# from vdem_indices_2014 with some color and size aesthetics
# and add a regression line

ggplot(vdem_indices_2014, aes(x = gdp_per_capita, y = polyarchy)) +
        geom_point(aes(color = country, size = population)) +
        geom_smooth(method = "lm", se = FALSE) +
        labs(title = "Scatterplot of GDP per Capita vs Polyarchy with LM
Line", x = "GDP per Capita", y = "Polyarchy") +
        theme(legend.position = "none")
```

This uses the same ggplot() function with the same scatterplot geometry (geom_point) and the same labeling. In the geom_point() function we add an additional aes() function to specify that we want to vary the color of the points based on the country and the size of the points based on population. Then, we add a new geometry function, geom_smooth(), which adds a smoothed line to the plot. The method = "lm" argument specifies that we want to fit a linear model to the data, and se = FALSE specifies that we do not want to include a shaded area representing the standard error of the regression line.

```
## create a scatter plot of liberal democracy vs exports from
vdem_indices_2014 with some color and size aesthetics create a scatter plot
# with an lm line

ggplot(vdem_indices_2014, aes(x = liberal_democracy, y = imports)) +
        geom_point(aes(color = country, size = population)) +
        geom_smooth(method = "lm", se = FALSE) +
        labs(title = "Scatterplot of Liberal Democracy vs Imports",
        x = "Liberal Democracy", y = "Imports") +
        theme(legend.position = "none")
```

Here we use the same functions, but put the measure of democracy on the X-axis. This implies to the viewer that we consider Liberal Democracy as the cause of the changes in Imports. In the previous, we implied to the viewer that GDP per Capita is the cause of changes in Polyarchy. This isn't about proof, it's about how we present our data and the story we want to tell with it.

Now, let's use the Gapminder data to tell a little more complicated story with a histogram and a different sort of line.

```
## create a histogram of lifeExp from the gapminder_data dataset
## overlay it with a normal distribution curve with the same mean and
## standard deviation as the lifeExp variable

ggplot(gapminder_data, aes(x = lifeExp)) +
        geom_histogram(aes(y = ..density..), binwidth = 5, fill = "green",
color = "black") +
        stat_function(fun = dnorm, args = list(mean =
mean(gapminder_data$lifeExp), sd = sd(gapminder_data$lifeExp)), color =
"red", size = 1) +
        labs(title = "Histogram of Life Expectancy with Normal Curve", x =
"Life Expectancy", y = "Density")
```

First, ggplot() is our main function and its first argument is the dataframe we are referencing.

The aes() function specifies that we want to use the lifeExp variable for the x-axis.

In the geom_histogram() function, we specify that we want to plot the density of the data instead of the frequency by using aes(y = ..density..). This allows us to overlay a normal distribution curve on top of the histogram later.

The binwidth argument specifies the width of the bins in the histogram, which represents the number of observations in each bin. The fill and color arguments specify the colors of the bars and their borders, respectively.

Next, we add a normal distribution curve to the plot using the stat_function() function.

The *fun = dnorm* argument specifies the *dnorm()* function - the *norm*al *d*istribution.

The *args = list(mean = mean(gapminder_data*$lifeExp), sd = sd(gapminder_data$*lifeExp))* tells R that we want the distribution to be based on the mean and standard deviation of the lifeExp variable in our data. The color and size arguments specify the color and thickness of the normal distribution curve.

Finally, we add labels to the plot with labs().

4 Lab assignment

Before we meet for Lab 4, practice creating plots by loading a dataset of your choice - preferably the one you will use for your semester project - then creating at least *3* different types of plot. Also, find the *mean, median, variance, and standard deviation* of at least 2 variables from the data you loaded. Put the Export the Plots and copy the statistics, placing everything in a Word document, Google Doc, etc. Turn this in on Canvas in PDF or Word format.

Extra points opportunity: If you can put the mean, median, variance, and standard deviation in a table using the method from Lab 1 (or any other *R method*), it is worth 5 extra points. Include the code you used to do it in your Word document.

Time saving hint #1: For your semester project, you will need to produce *3* plots for your data at a minimum:

1. A histogram or dot plot of your dependent variable

2. histogram of or dot plot of your main independent variable

   ```
    - If you choose to have more than one independent variable, histograms
   or dots plot of each of those make for a better project
   ```

3. A scatterplot of your dependent variable versus your main independent variable with a regression line

Time saving hint # 2: For your semester project, you will need the the *mean, median, variance, and standard deviation* for your dependent variable and independent variable(s) as well.

5 Word on datasets

There is information in Canvas about datasets. If you would like to use the Gapminder dataset we used here, that is fine. If you would like to use the V-Dem dataset, the V-Dem Codebook is here, so you can select variables that interest you. If you would like to use the Gapminder data with other variables from V-Dem, we will talk about how to merge datasets next time.