# Econ 366 demo

## R Econ 366

You should be able to do anything you need to do for this class in R and a special feature called R Markdown. I'm bascially going to teach you to code in R in *code chunks* that you can adapt to the tasks you need and this will make it easy for me and others to see how you've done the work, and will teach you to write self-contained code.

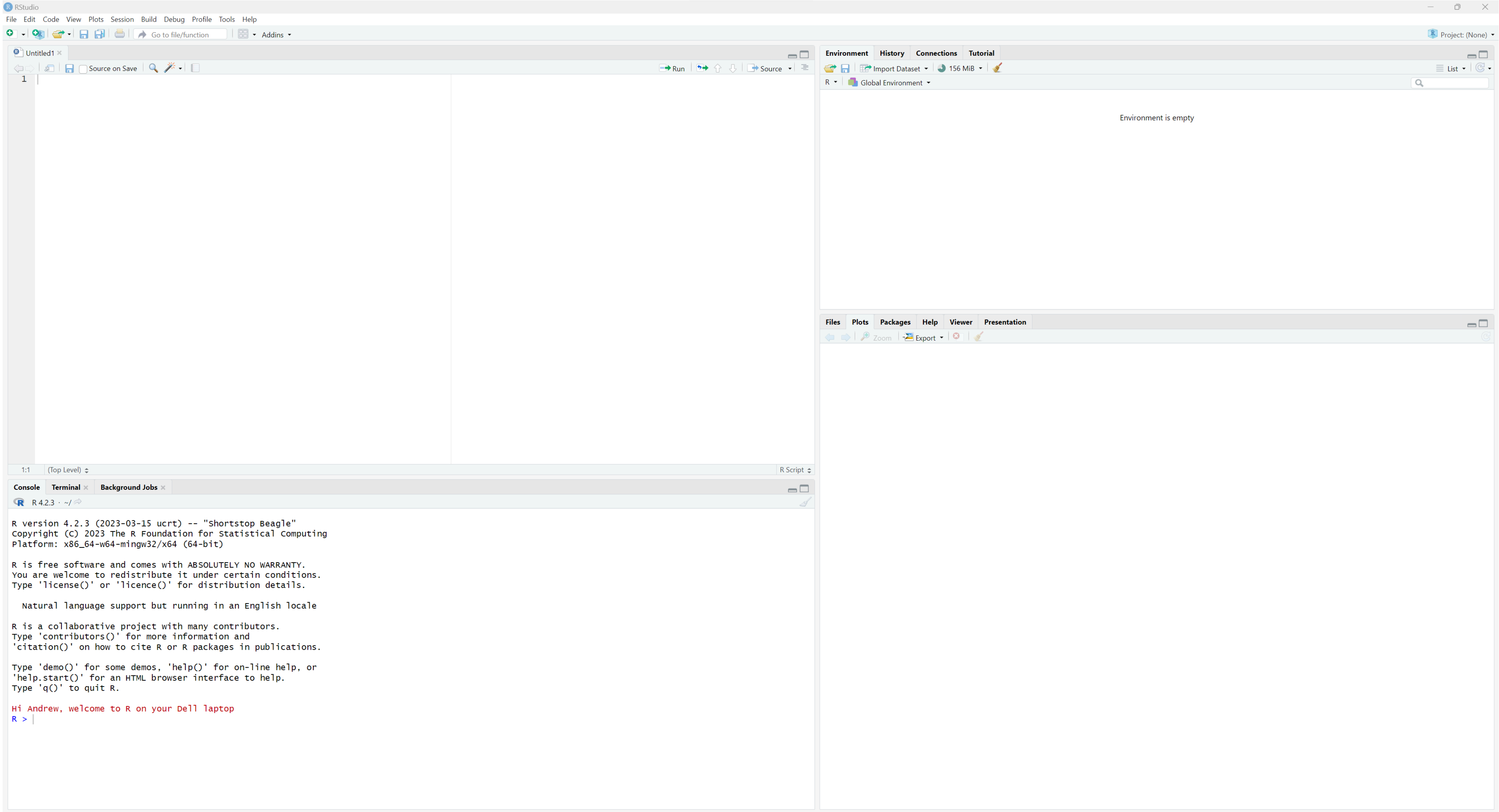
For today's demo, we're going to go through a quick exercise to download some basic data from an open-data website and use that data to make a quick graph, with everything produced in an HTML file. Sound cool?

## Installing R and R Studio

The first thing you'll want to do, if you haven't done so already, is to install R and the editor RStudio. Once you have those installed, you're ready to create a basic R Markdown document. This document from Earth Lab provides a great introduction, on which I'll base some of what follows here on their introduction. You can also watch this video. If you can create a new RMarkdown document and render (knit) it, and that's the skill you'll use for all the assignments in the class.

## The RStudio Environment

When you open RStudio, you'll see a 4-tile scree with an editor (top-left), a command-line (Console) interface (bottom-right), an envirionment and history pane (top-right) and a file and package explorer and viewer pane (bottom right).

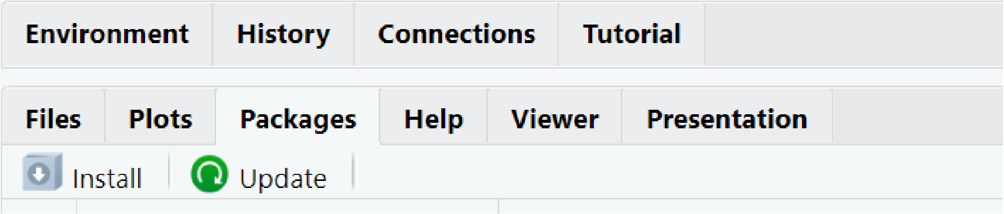


```
R version 4.2.3 (2023-03-15 ucrt) -- "Shortstop Beagle"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Hi Andrew, welcome to R on your Dell laptop
R > 
```

You can try to create a new R markdown document from the File menu and then knit it just to see how it works.

# R Packages

Base R has a lot of functionality, but one of the strengths (and challenges) of R is that it relies on packages which make lots of very powerful data work possible, but also mean that two people can attack a problem in R in very different ways. When I look back at some of my old R code, it's almost incomprehensible because I used to use a very different set of packages than I use today. For most of the work in this class, we'll rely on *the tidyverse* so you can install that package to get going. You can check with packages you have installed in this window:



If you don't have the tidyverse package installed, install it now. I would also recommend that you also install a package called janitor for the purpose of data cleaning, and I'll use it in this code.

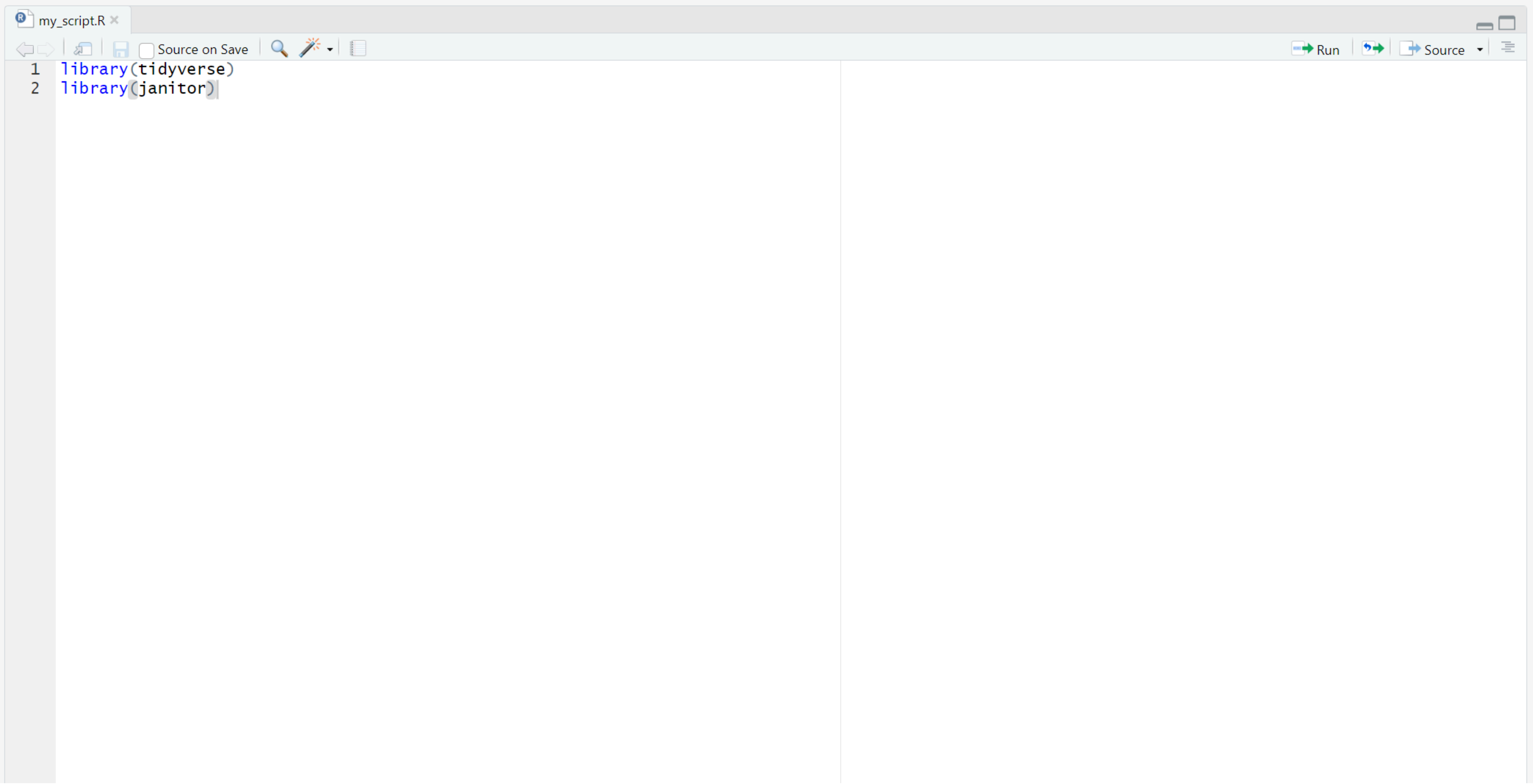
Now, there are basically three ways you can run commands in R to process data: 1) scripts; 2) console or command line; and 3) Markdown documents (RMarkdown or Quarto).

A script is like a recipe card: a sequence of commands in a text file that you can execute one at a time or all at once. When you execute code from a script, it's the equivalent of cutting and pasting each command into the console in sequence. And, markdown documents are combinations of code and text that allow you to produce integrated documents that read data and present output all in one place.

Let's use the next section for a little demo. When you start R in RStudio, it loads some basic packages (the commands and features you'll use) but there's a lot of customisation that comes from packages. But, you need to tell R which packages to load, and you load packages using the **library** command. Unless a package is loaded, or referenced in a command (we'll come to that), it won't be available for your code. You could do that in a script, at the command line, or in a code chunk in an RMarkdown document. Let me show you two examples with the following packages:

```r
library(tidyverse)
library(janitor)
```

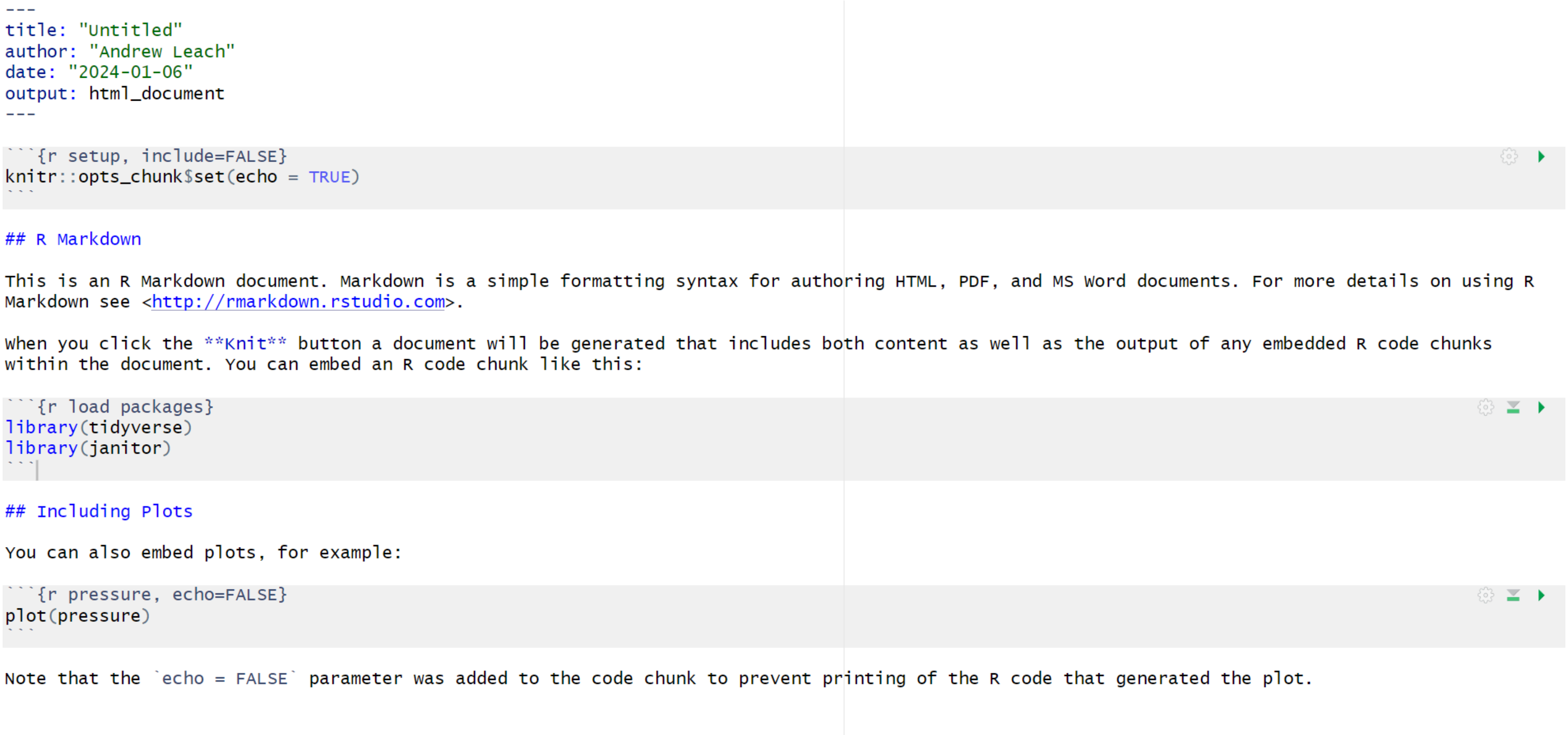
In a script, I could have the two commands like this and then, when I click run, it will execute the commands in order from my cursor. Or, I can highlight a single or multiple commands and press [crtl+enter] to run them.



```r
library(tidyverse)
library(janitor)
```

Alternatively, I could use a command line entry to read in a package, although I'd almost never recommend doing this. Keep your commands in scripts.

And, I could also have the commands in a code chunk in a markdown document and the commands will be executed when I *knit* the file.



~~~
---
title: "Untitled"
author: "Andrew Leach"
date: "2024-01-06"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R
Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks
within the document. You can embed an R code chunk like this:

```{r load packages}
library(tidyverse)
library(janitor)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
~~~

I thought, for today's demo, we would start with a data set that lets me introduce you to some of the features of R and R Markdown: Canada's Greenhouse Gas Emissions Inventory, for which an open data page is available. We're going to download data by economic sector that is available here.

My recommendation would be just to do this one in a script, and leave R-markdown for next time. So, for each of the commands I am showing you, copy them into a script, run them, and see if you can see what they're doing.

The first thing we're going to ask R to do is to download those data. You don't *have* to download a local copy of the data, but I tend to find it's helpful, so we'll do that here. To get the link, use a right-click on the data you want to download (although I have given you the link below):

```
# I'm giving it a simpler name ghg_data.csv
# old link was"https://data.ec.gc.ca/data/substances/monitor/canada-s-official-greenhouse-gas-inventory/B-Economic-Sector/EN_GHG_
# mode= wb is "windows binary" and works more reliably than the default in my experience
download.file(
```

```
  "https://data-donnees.az.ec.gc.ca/api/file?path=/substances%2Fmonitor%2Fcanada-s-official-greenhouse-gas-inventory%2FB-Economic
  destfile = "ghg_data.csv",mode="wb")
```

Next, we want to read the data into R from the file we created. I am also going to use janitor to clean the data names (it will take out spaces, upper case letters, and other things that are hard to deal with in code).

```
ghg_data<-read.csv("ghg_data.csv") %>% clean_names()
```

That code also lets me demonstrate to you a key attribute of the *tidyverse*, the pipe (`%>%`) which allows you to *pass* data from one command to another. So, in that code, I am defining `ghg_data` to be the product of the data I read in from the csv, passed through the janitor function `clean_names()`. Trust me, this will be natural for you in no time.

And, since we have the data in, we can start to do some work with the data. You can, as you get used to working in R, have a look at the CSV file so you know what you're dealing with, or you can use some R commands to have a look. For example (yes, I know, I told you not to look at the command line) you could type this in the console:

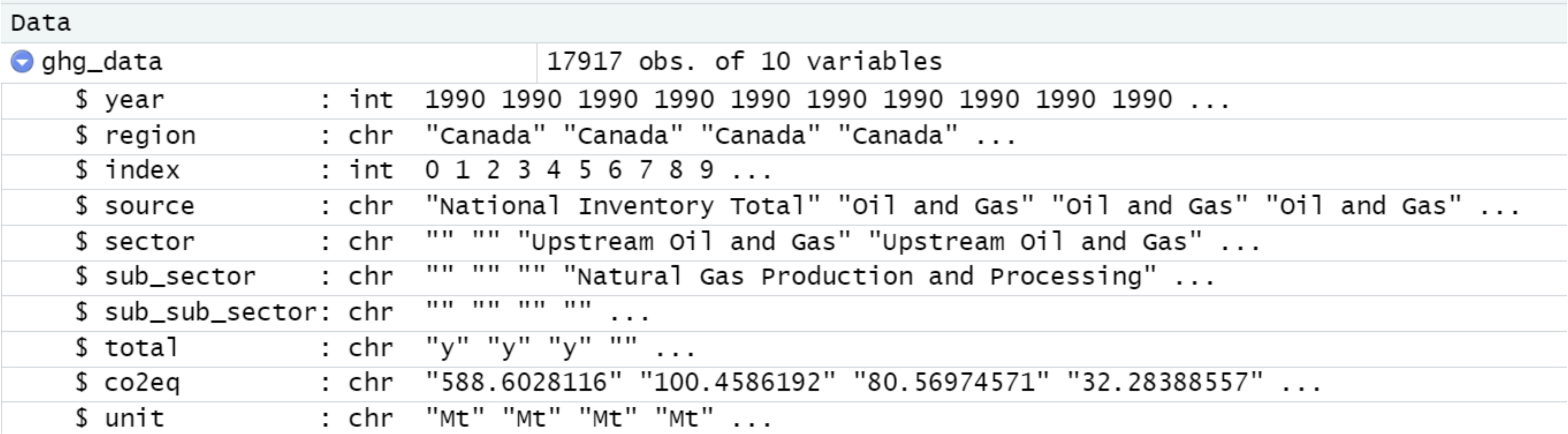
```
head(ghg_data)
```

```
  year region index                   source               sector
1 1990 Canada     0 National Inventory Total
2 1990 Canada     1              Oil and Gas
3 1990 Canada     2              Oil and Gas Upstream Oil and Gas
4 1990 Canada     3              Oil and Gas Upstream Oil and Gas
5 1990 Canada     4              Oil and Gas Upstream Oil and Gas
6 1990 Canada     5              Oil and Gas Upstream Oil and Gas
                             sub_sector                    sub_sub_sector total
1                                                                              y
2                                                                              y
3                                                                              y
4 Natural Gas Production and Processing
5           Conventional Oil Production                                        y
6           Conventional Oil Production Conventional Light Oil Production
        co2eq unit
1 588.6028116   Mt
2 100.4586192   Mt
```

```
3 80.56974571   Mt
4 32.28388557   Mt
5 21.04013617   Mt
6 12.62694028   Mt
```

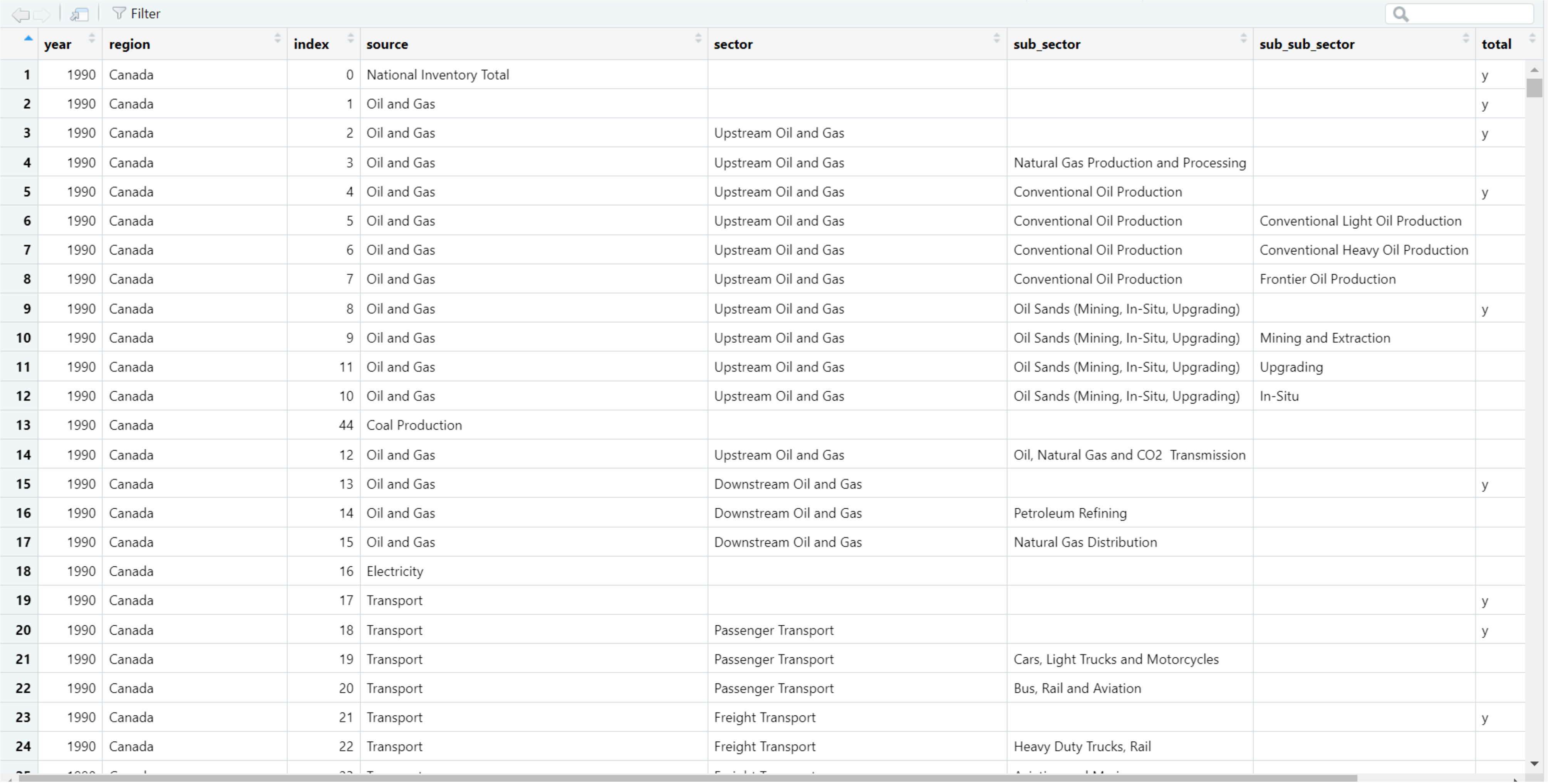
That's not pretty, but it tells you a bit about what you've got: emissions by year, region, source (sector), and then three sub-sector categories with various levels of aggregation. There's an indicator telling you whether or not the observation is a sector total, and finally the emissions and units columns.

You can also look at the data in the data window in the top right hand corner of r-studio to see what you're looking at.



```
Data
ghg_data                        17917 obs. of 10 variables
    $ year          : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
    $ region        : chr  "Canada" "Canada" "Canada" "Canada" ...
    $ index         : int  0 1 2 3 4 5 6 7 8 9 ...
    $ source        : chr  "National Inventory Total" "Oil and Gas" "Oil and Gas" "Oil and Gas" ...
    $ sector        : chr  "" "" "Upstream Oil and Gas" "Upstream Oil and Gas" ...
    $ sub_sector    : chr  "" "" "" "Natural Gas Production and Processing" ...
    $ sub_sub_sector: chr  "" "" "" "" ...
    $ total         : chr  "y" "y" "y" "" ...
    $ co2eq         : chr  "588.6028116" "100.4586192" "80.56974571" "32.28388557" ...
    $ unit          : chr  "Mt" "Mt" "Mt" "Mt" ...
```

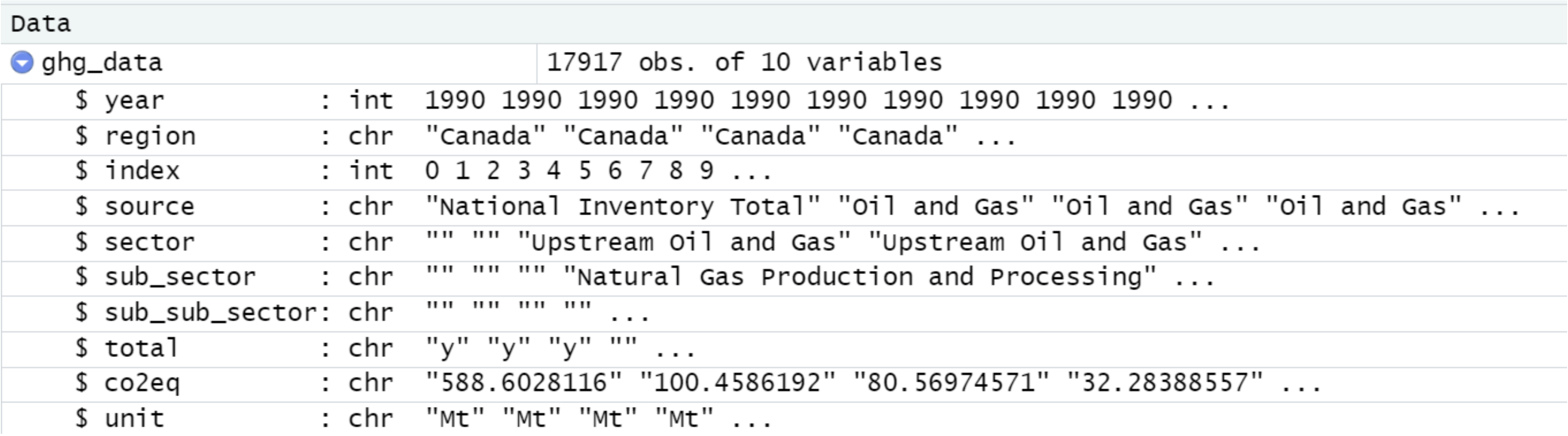
If you double-click on that, you'll also open the data-viewer which can be handy, but also cumbersome for large data sets:



| | year | region | index | source | sector | sub_sector | sub_sub_sector | total |
|---|---|---|---|---|---|---|---|---|
| 1 | 1990 | Canada | 0 | National Inventory Total | | | | y |
| 2 | 1990 | Canada | 1 | Oil and Gas | | | | y |
| 3 | 1990 | Canada | 2 | Oil and Gas | Upstream Oil and Gas | | | y |
| 4 | 1990 | Canada | 3 | Oil and Gas | Upstream Oil and Gas | Natural Gas Production and Processing | | |
| 5 | 1990 | Canada | 4 | Oil and Gas | Upstream Oil and Gas | Conventional Oil Production | | y |
| 6 | 1990 | Canada | 5 | Oil and Gas | Upstream Oil and Gas | Conventional Oil Production | Conventional Light Oil Production | |
| 7 | 1990 | Canada | 6 | Oil and Gas | Upstream Oil and Gas | Conventional Oil Production | Conventional Heavy Oil Production | |
| 8 | 1990 | Canada | 7 | Oil and Gas | Upstream Oil and Gas | Conventional Oil Production | Frontier Oil Production | |
| 9 | 1990 | Canada | 8 | Oil and Gas | Upstream Oil and Gas | Oil Sands (Mining, In-Situ, Upgrading) | | y |
| 10 | 1990 | Canada | 9 | Oil and Gas | Upstream Oil and Gas | Oil Sands (Mining, In-Situ, Upgrading) | Mining and Extraction | |
| 11 | 1990 | Canada | 11 | Oil and Gas | Upstream Oil and Gas | Oil Sands (Mining, In-Situ, Upgrading) | Upgrading | |
| 12 | 1990 | Canada | 10 | Oil and Gas | Upstream Oil and Gas | Oil Sands (Mining, In-Situ, Upgrading) | In-Situ | |
| 13 | 1990 | Canada | 44 | Coal Production | | | | |
| 14 | 1990 | Canada | 12 | Oil and Gas | Upstream Oil and Gas | Oil, Natural Gas and CO2 Transmission | | |
| 15 | 1990 | Canada | 13 | Oil and Gas | Downstream Oil and Gas | | | y |
| 16 | 1990 | Canada | 14 | Oil and Gas | Downstream Oil and Gas | Petroleum Refining | | |
| 17 | 1990 | Canada | 15 | Oil and Gas | Downstream Oil and Gas | Natural Gas Distribution | | |
| 18 | 1990 | Canada | 16 | Electricity | | | | |
| 19 | 1990 | Canada | 17 | Transport | | | | y |
| 20 | 1990 | Canada | 18 | Transport | Passenger Transport | | | y |
| 21 | 1990 | Canada | 19 | Transport | Passenger Transport | Cars, Light Trucks and Motorcycles | | |
| 22 | 1990 | Canada | 20 | Transport | Passenger Transport | Bus, Rail and Aviation | | |
| 23 | 1990 | Canada | 21 | Transport | Freight Transport | | | y |
| 24 | 1990 | Canada | 22 | Transport | Freight Transport | Heavy Duty Trucks, Rail | | |

## Fixing data types

Something I noticed after loading those data is that the my data set has ghg emissions ( `co2eq` ) stored a character ( `chr` ) variable. You can see this in the data window of your R-Studio session:



```
Data
ghg_data                        17917 obs. of 10 variables
    $ year          : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
    $ region        : chr  "Canada" "Canada" "Canada" "Canada" ...
    $ index         : int  0 1 2 3 4 5 6 7 8 9 ...
    $ source        : chr  "National Inventory Total" "Oil and Gas" "Oil and Gas" "Oil and Gas" ...
    $ sector        : chr  "" "" "Upstream Oil and Gas" "Upstream Oil and Gas" ...
    $ sub_sector    : chr  "" "" "" "Natural Gas Production and Processing" ...
    $ sub_sub_sector: chr  "" "" "" ""  ...
    $ total         : chr  "y" "y" "y" "" ...
    $ co2eq         : chr  "588.6028116" "100.4586192" "80.56974571" "32.28388557" ...
    $ unit          : chr  "Mt" "Mt" "Mt" "Mt" ...
```

So, R thinks that the variable co2eq is a string (text). That’s not going to work well for graphing, so we’re going to have to make a quick change to that variable. This lets me introduce another command that you’ll use a lot: `mutate`. We’re going to use a command to **overwrite our existing data set** in memory called `ghg_data`. We’re going to use that pipeline `%>%` command (the *pass to*), so what you see below is, in text:

> over-write my data called ghg_data by taking ghg_data and changing the co2_eq column to a numeric data point

```
#talk yourself through it
# ghg_data is equal to ghg_data with co2eq converted to a numeric variable
ghg_data<-ghg_data %>% mutate(co2eq=as.numeric(co2eq))
```

Any time you want to do a calculation on a variable (like a formula in Excel), you’ll use mutate. You can also use it, as I did above, with a self-reference to re-define a variable.

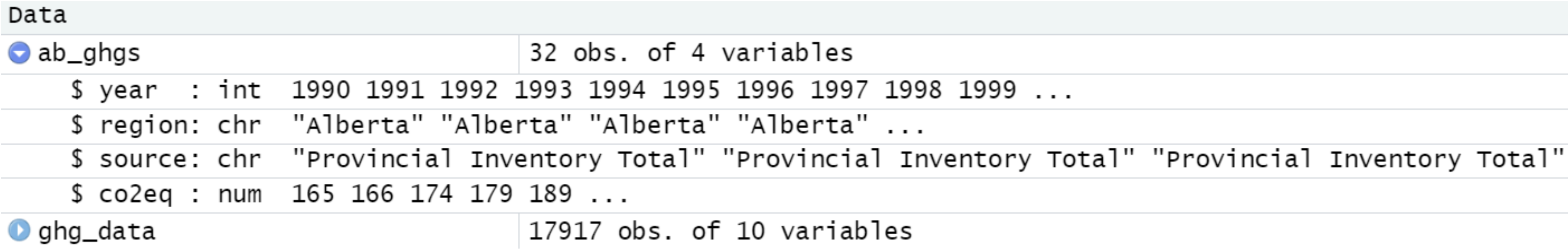
## Filtering Data

Now, we know those data are a bit of a mess, but what if we wanted to extract on piece of information from those data: total emissions from Alberta for each year. We could use the concept of a filter. We’re going to use a command to create a **new data set** in memory called `ab_ghgs` and we’ll build it by grabbing only some parts of our existing `ghg_data` dataset. We’re again going to use the pipeline `%>%` (or pass to) command so what you see below is, in text:

> create a new data set called ab_ghgs by taking ghg_data and passing it through a filter what will keep only observations that have region equal to Alberta and source equal to Provincial Inventory Total, then select only the columns year, region, source, and co2eq from that filtered data

```
#talk yourself through it
# ab_ghgs is equal to ghg_data passed through a filter to catch only the observations that meet these conditions...
ab_ghgs<-ghg_data %>% filter(region=="Alberta",source=="Provincial Inventory Total")%>%
#and then we'll select four columns
    select(year,region,source,co2eq)
```

You'll notice that, after you *run* these commands, you should have a new data set in memory called ab_ghgs. Take a look at it.

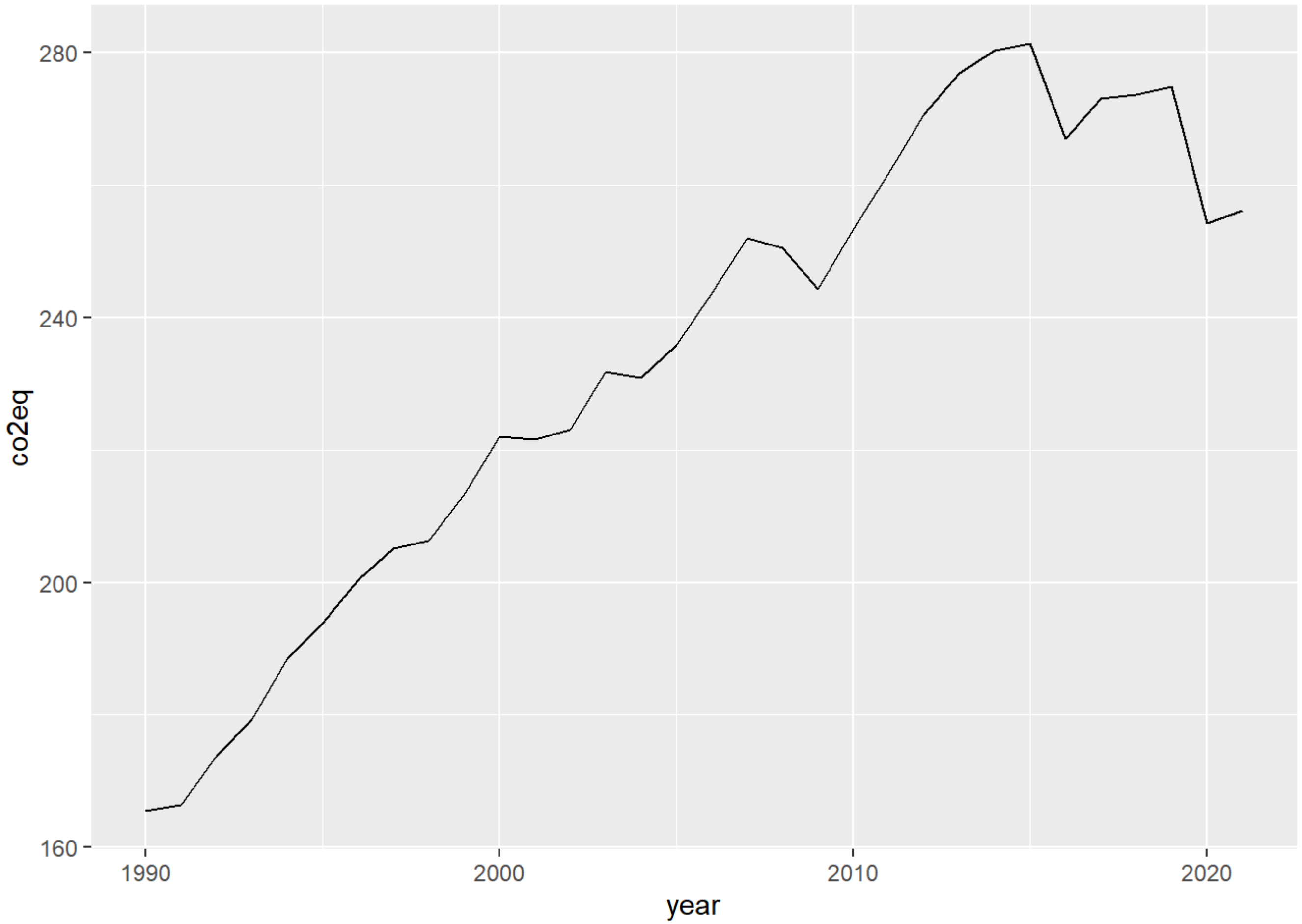


```
Data
ab_ghgs                         32 obs. of 4 variables
    $ year  : int  1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 ...
    $ region: chr  "Alberta" "Alberta" "Alberta" "Alberta" ...
    $ source: chr  "Provincial Inventory Total" "Provincial Inventory Total" "Provincial Inventory Total"
    $ co2eq : num  165 166 174 179 189 ...
ghg_data                        17917 obs. of 10 variables
```

## Making Plots

So, now let's use that Alberta data to make a basic graph. We'll use ggplot (Grammar of Graphics plot), which effectively creates graphs in layers. All the data visualization in this class will use ggplot.

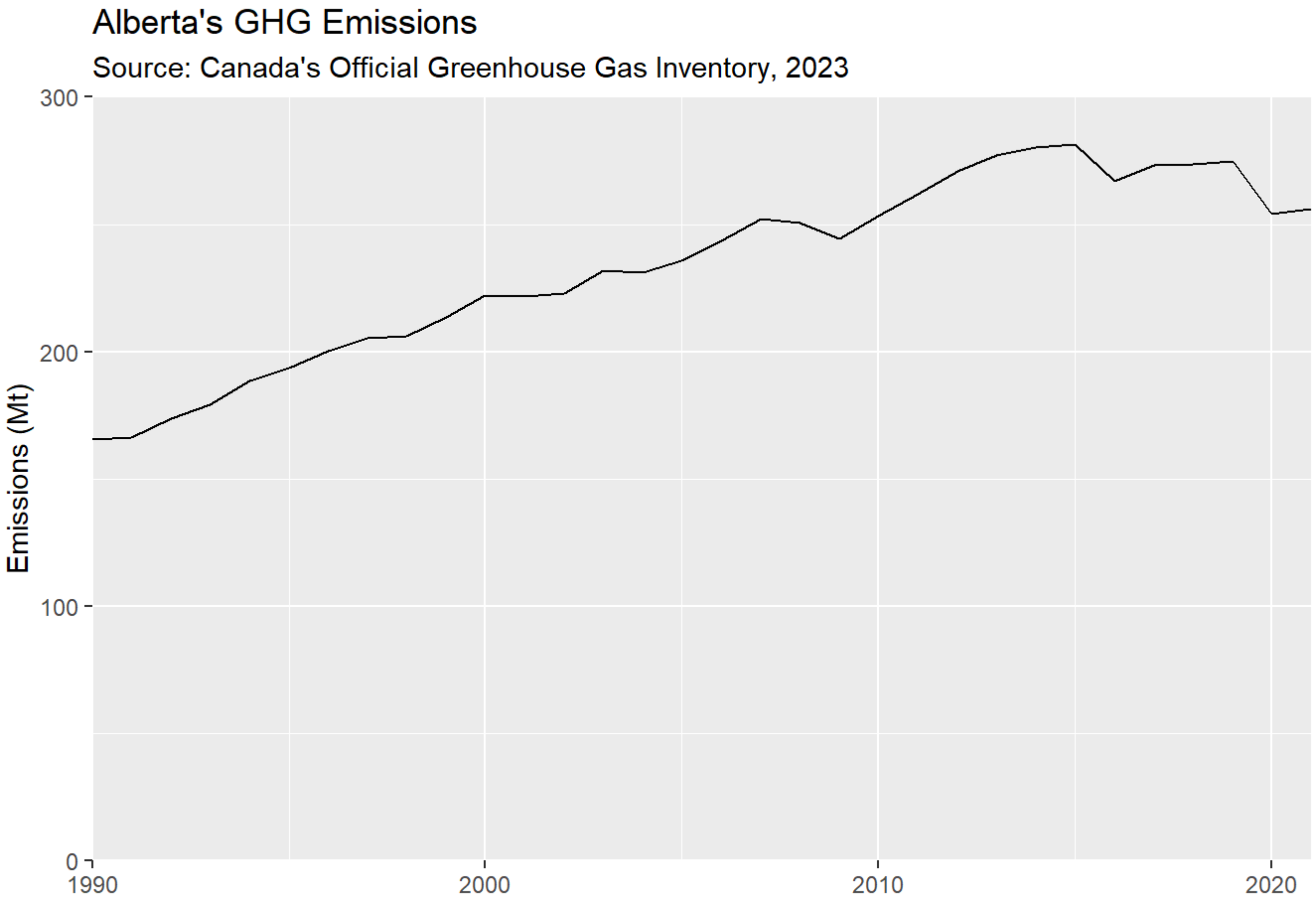
```
#ggplot is part of the tidyverse
ggplot(ab_ghgs)+ #create a plot
  #add a line graph "geom"
  geom_line(aes(x=year,y=co2eq))
```



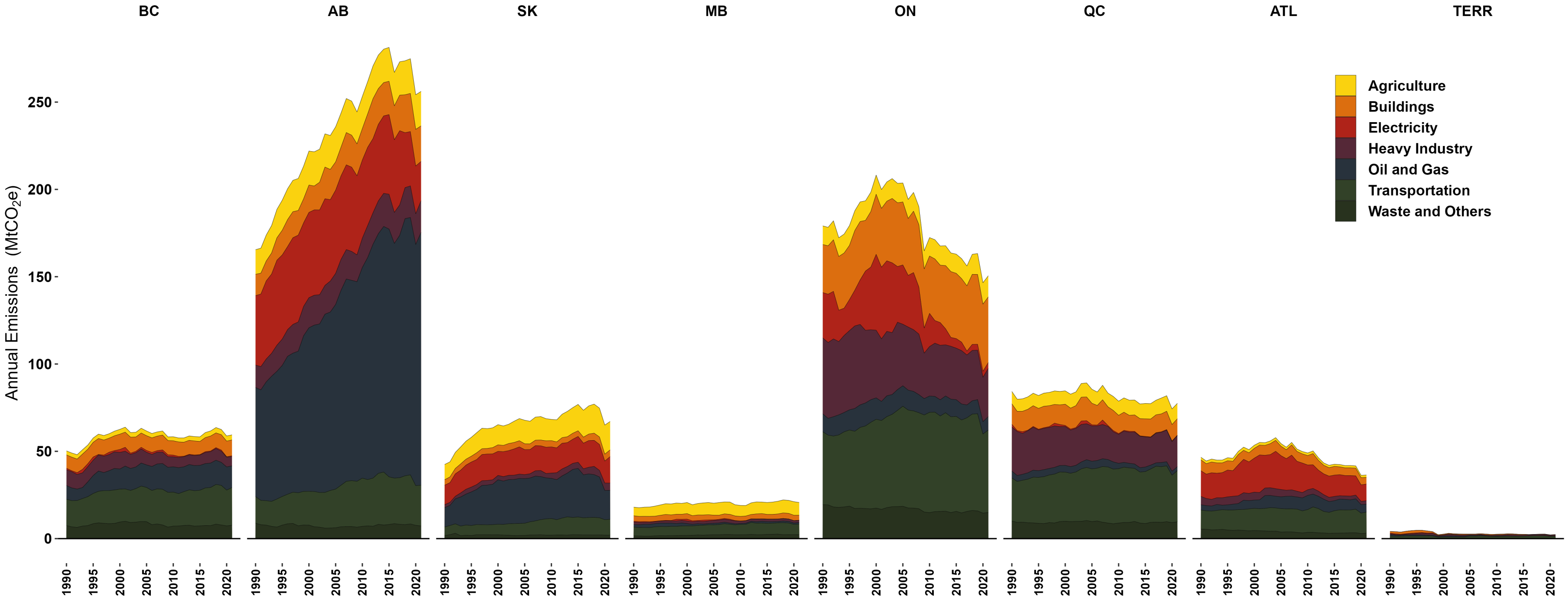
Not bad - you've got a graph of Alberta's official GHG emissions inventory from 1990 to 2020. But, it's a bit ugly.

We can make it a bit nicer with a few additions: title, subtitle, axis labels, and fixing the x and y axes.

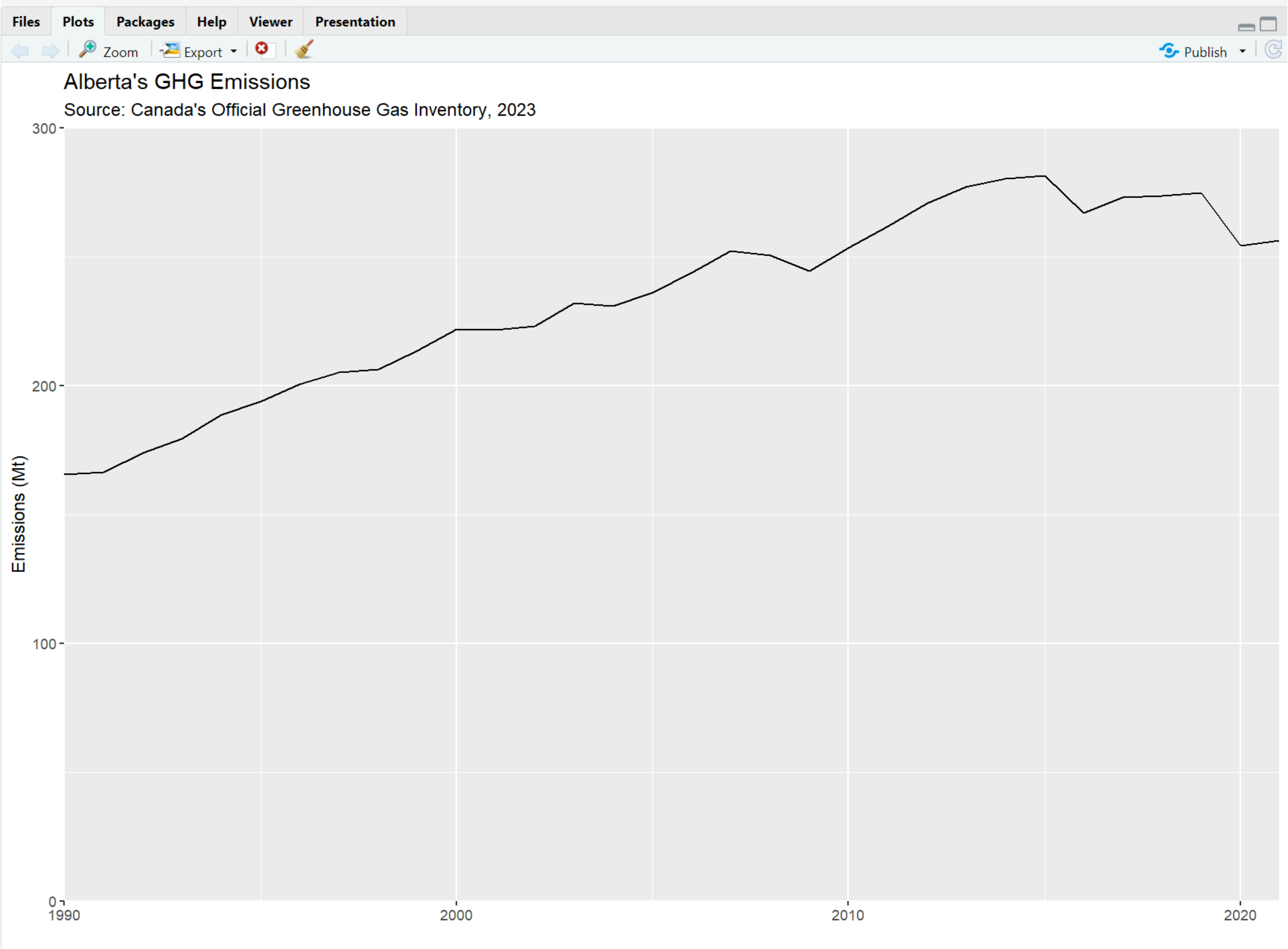
```
#ggplot is part of the tidyverse
ggplot(ab_ghgs)+ #create a plot
  #add a line graph "geom"
  geom_line(aes(year,co2eq))+
  #fix the x scales so that they don't add buffer spaces
  scale_x_continuous(expand=c(0,0))+
  scale_y_continuous(expand=c(0,0))+
  expand_limits(y=c(0,300))+ #make sure the y axis goes to zero
  #now add some titles
  labs(title="Alberta's GHG Emissions",
        subtitle="Source: Canada's Official Greenhouse Gas Inventory, 2023",
      x="",
      y="Emissions (Mt)")
```



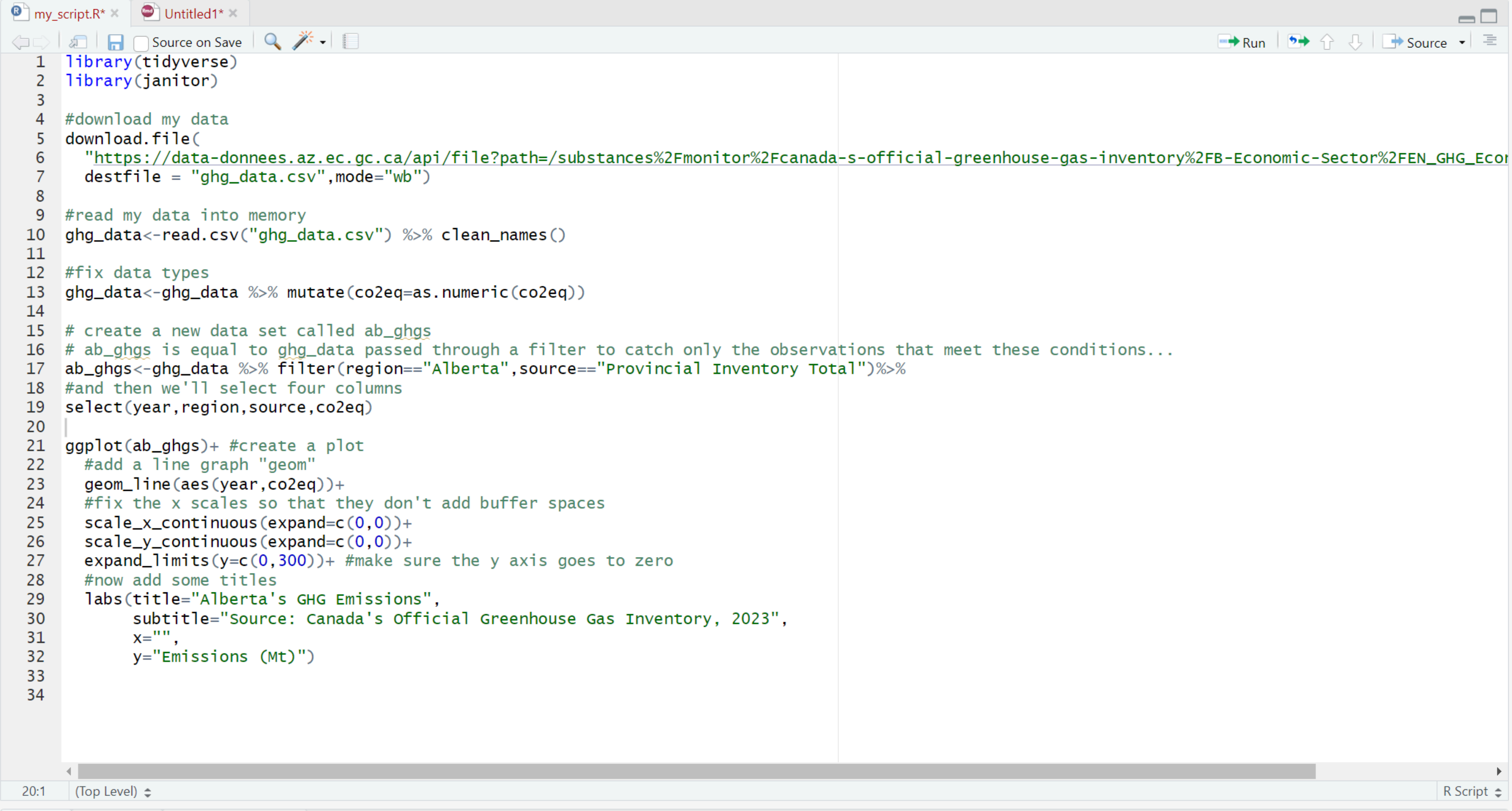
Eventually, we’ll have you making some nicer graphs like this one from my data projects page made from those same data:



The big things I want you to take away from this are 1) I got R working on my computer; 2) I used code to download data from the internet; 3) I used code to clean those data; 4) I made a graph with those data. How many of you could say more than 2 of those things before today?

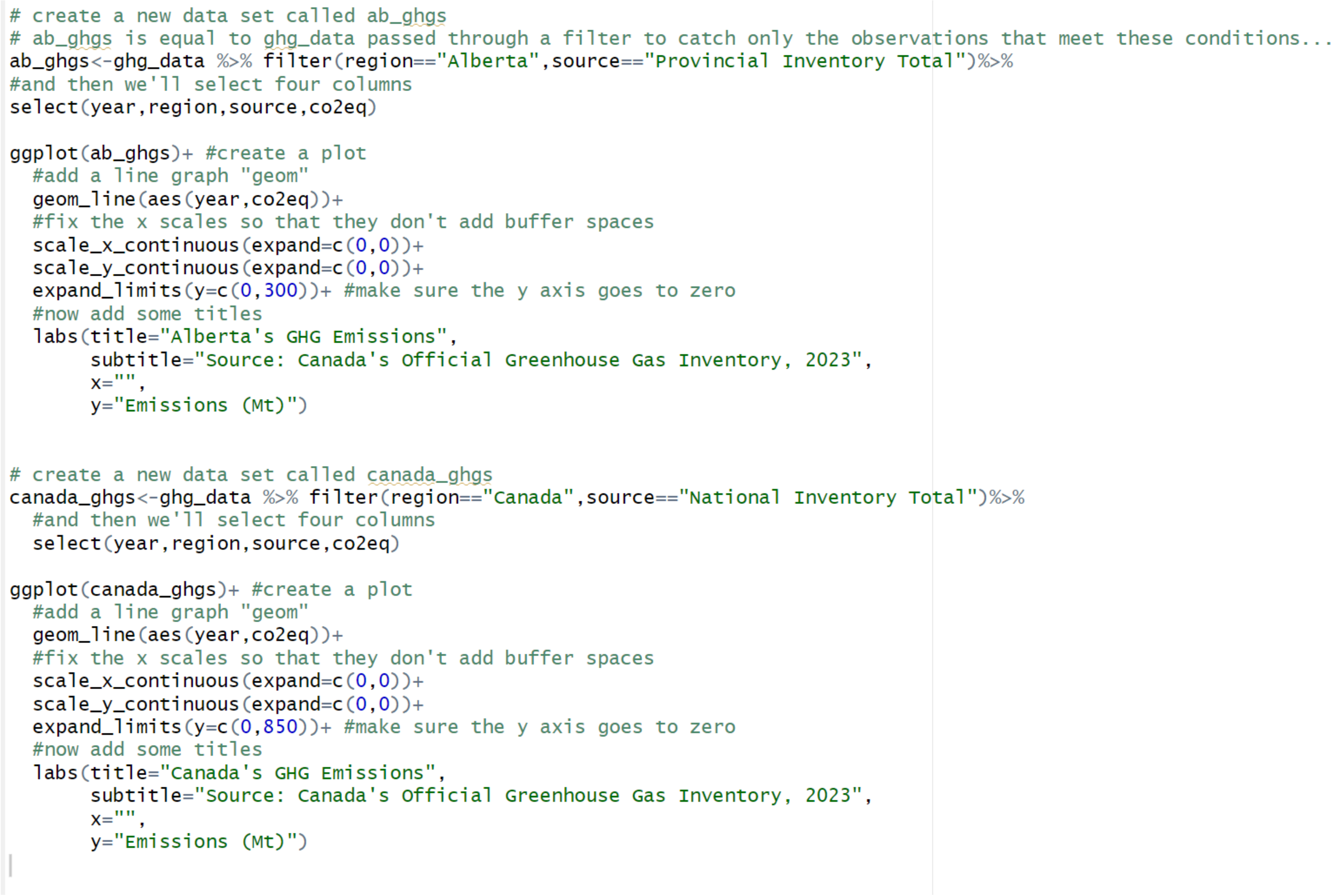


And, if you did that all correctly, you've got a nice recipe card in your script window of all the commands to do this.



```r
library(tidyverse)
library(janitor)

#download my data
download.file(
  "https://data-donnees.az.ec.gc.ca/api/file?path=/substances%2Fmonitor%2Fcanada-s-official-greenhouse-gas-inventory%2FB-Economic-Sector%2FEN_GHG_Ecor
  destfile = "ghg_data.csv",mode="wb")

#read my data into memory
ghg_data<-read.csv("ghg_data.csv") %>% clean_names()

#fix data types
ghg_data<-ghg_data %>% mutate(co2eq=as.numeric(co2eq))

# create a new data set called ab_ghgs
# ab_ghgs is equal to ghg_data passed through a filter to catch only the observations that meet these conditions...
ab_ghgs<-ghg_data %>% filter(region=="Alberta",source=="Provincial Inventory Total")%>%
#and then we'll select four columns
select(year,region,source,co2eq)

ggplot(ab_ghgs)+ #create a plot
  #add a line graph "geom"
  geom_line(aes(year,co2eq))+
  #fix the x scales so that they don't add buffer spaces
  scale_x_continuous(expand=c(0,0))+
  scale_y_continuous(expand=c(0,0))+
  expand_limits(y=c(0,300))+ #make sure the y axis goes to zero
  #now add some titles
  labs(title="Alberta's GHG Emissions",
       subtitle="Source: Canada's Official Greenhouse Gas Inventory, 2023",
       x="",
       y="Emissions (Mt)")
```

Now, if you want to see the power of this tool, copy and paste the code you used to make a graph of Alberta emissions to make a graph for Canada too:



```
# create a new data set called ab_ghgs
# ab_ghgs is equal to ghg_data passed through a filter to catch only the observations that meet these conditions...
ab_ghgs<-ghg_data %>% filter(region=="Alberta",source=="Provincial Inventory Total")%>%
#and then we'll select four columns
select(year,region,source,co2eq)

ggplot(ab_ghgs)+ #create a plot
  #add a line graph "geom"
  geom_line(aes(year,co2eq))+
  #fix the x scales so that they don't add buffer spaces
  scale_x_continuous(expand=c(0,0))+
  scale_y_continuous(expand=c(0,0))+
  expand_limits(y=c(0,300))+ #make sure the y axis goes to zero
  #now add some titles
  labs(title="Alberta's GHG Emissions",
       subtitle="Source: Canada's Official Greenhouse Gas Inventory, 2023",
       x="",
       y="Emissions (Mt)")


# create a new data set called canada_ghgs
canada_ghgs<-ghg_data %>% filter(region=="Canada",source=="National Inventory Total")%>%
  #and then we'll select four columns
  select(year,region,source,co2eq)

ggplot(canada_ghgs)+ #create a plot
  #add a line graph "geom"
  geom_line(aes(year,co2eq))+
  #fix the x scales so that they don't add buffer spaces
  scale_x_continuous(expand=c(0,0))+
  scale_y_continuous(expand=c(0,0))+
  expand_limits(y=c(0,850))+ #make sure the y axis goes to zero
  #now add some titles
  labs(title="Canada's GHG Emissions",
       subtitle="Source: Canada's Official Greenhouse Gas Inventory, 2023",
       x="",
       y="Emissions (Mt)")
```

That’s what I mean by teaching you through the use of code chunks that you can modify for your own work.

Made with R and Quarto
View the source at GitHub