Functions Demo

Functions are a very powerful programming tool, allowing you to partially automate repetitive tasks. Since there are a couple of repetitive tasks in Data Assignment 2, it seems like the right time to introduce the concept. I'll use the Canadian Energy Regulator pipeline profiles data from here for this demonstration.

In what you see below, I've used the following packages:

```
library(kableExtra)
library(readxl)
library(janitor)
library(tidyverse)
library(lubridate)
library(scales)
library(viridis)
```

These data contain information for a variety of pipelines including the Enbridge Mainline, Trans-Mountain, and Keystone pipelines that are the focus of the assignment. Since the data are stored in a consistent way, with a consistent naming convention, and you need to download basically the same data for three pipelines, this is ripe for functionalization.

> A function is a short *program* that receives inputs and accomplishes a particular task. Once you create and store the function in memory, you can call it again and again to execute that task.

You create a function in r as `my_function<-function(parameter1,parameter2,etc.){code}` where the parameters are values (like a pipeline name) that you send to the function, and the code is the program you have the function execute.

> Importantly, variables you create inside a function **will not** be loaded into memory globally (i.e. they won't end up visible in your environment tab), and will only be in use for the time the function is running. If you call the same function again, it won't have access to variables or files it created before, but code running in a function will be able to access variables stored in your (global) enviroment.

In a normal R script or R markdown document, if I wanted to download data for the Cochin pipeline, I might use the following commands:

```
file_name<-"https://www.cer-rec.gc.ca/open/energy/throughput-capacity/Cochin-throughput-and-capacity.csv"
download.file(file_name,"local_file_name.csv",mode="wb")
cochin_data <- read_csv("local_file_name.csv")%>%clean_names()
```

But, since I know I am going to have to do this for a few pipelines, I might make myself a function:

```
get_pipe_data<-function(pipe_name){
#construct the file name based on the naming convention of the open govt site
#make sure you use the CORRECT name for the pipelines
# paste is like concatenate in Excel
# MAKE SURE you include the sep="" portion, otherwise it will include spaces by default
    file_name<-paste("https://www.cer-rec.gc.ca/open/energy/throughput-capacity/",pipe_name,"-throughput-and-capacity.csv",sep="")
    #create a standard local file name convention
    local_file<-paste(pipe_name,".csv",sep="")
    #download the file and give it the local name based on the name of the pipeline
    download.file(file_name,local_file,mode="wb")
    #read in the data, and add a column for the pipe name (so you can easily combine data for different pipes)
    pipe_data <- read_csv(file = local_file)%>%clean_names()%>%mutate(pipe_name=pipe_name)
#the last line in a function here will send this data back to wherever you called it. You'll see how it works in a second
  pipe_data
}
```

So, now I've created a function called `get_pipe_data` for which I need to provide a pipeline name. As long as that name is one that appears on the open data site, the function will download and return the data I want.

Try it out: run the code to create the function and then run `get_pipe_data(pipe_name="Cochin")` (you can also run `get_pipe_data("Cochin")`, since `pipe_name` is the first and only parameter being sent.

You might also choose to build some error-checking or other parameters into your functions. And, you can also give parameters a default value (using the equals sign in the function definition) if there's one task you do often.

For example, in downloading data for the assignment, I used this function:

```
get_pipe_data<-function(pipe_name="Enbridge-Mainline"){
#check the names on the open data site - notice they are bad about naming consistently
    names<-c("Alliance","Cochin","Enbridge-Mainline","Norman-Wells",
        "Keystone","MNP","Trans-Mountain","TQM","tcpl-mainline","Westcoast","ngtl")
#start with a default of returning "no data"
pipe_data<-"No data"
#is the name I've sent one of the specified pipeline names?
if(pipe_name %in% names)
  {
  file_name<-paste("https://www.cer-rec.gc.ca/open/energy/throughput-capacity/",pipe_name,"-throughput-and-capacity.csv",sep="")
    local_file<-paste(pipe_name,".csv",sep="")
    download.file(file_name,local_file,mode="wb")
    pipe_data <- read_csv(file = local_file)%>%clean_names()%>%mutate(pipe_name=pipe_name)
}

#return pipe_data, which will either be the data I just made, if I had a correct name, or the string "No data" if I didn't
pipe_data
}
```

Try it out:

```
enbridge_data<-get_pipe_data(pipe_name = "Enbridge-Mainline")
```

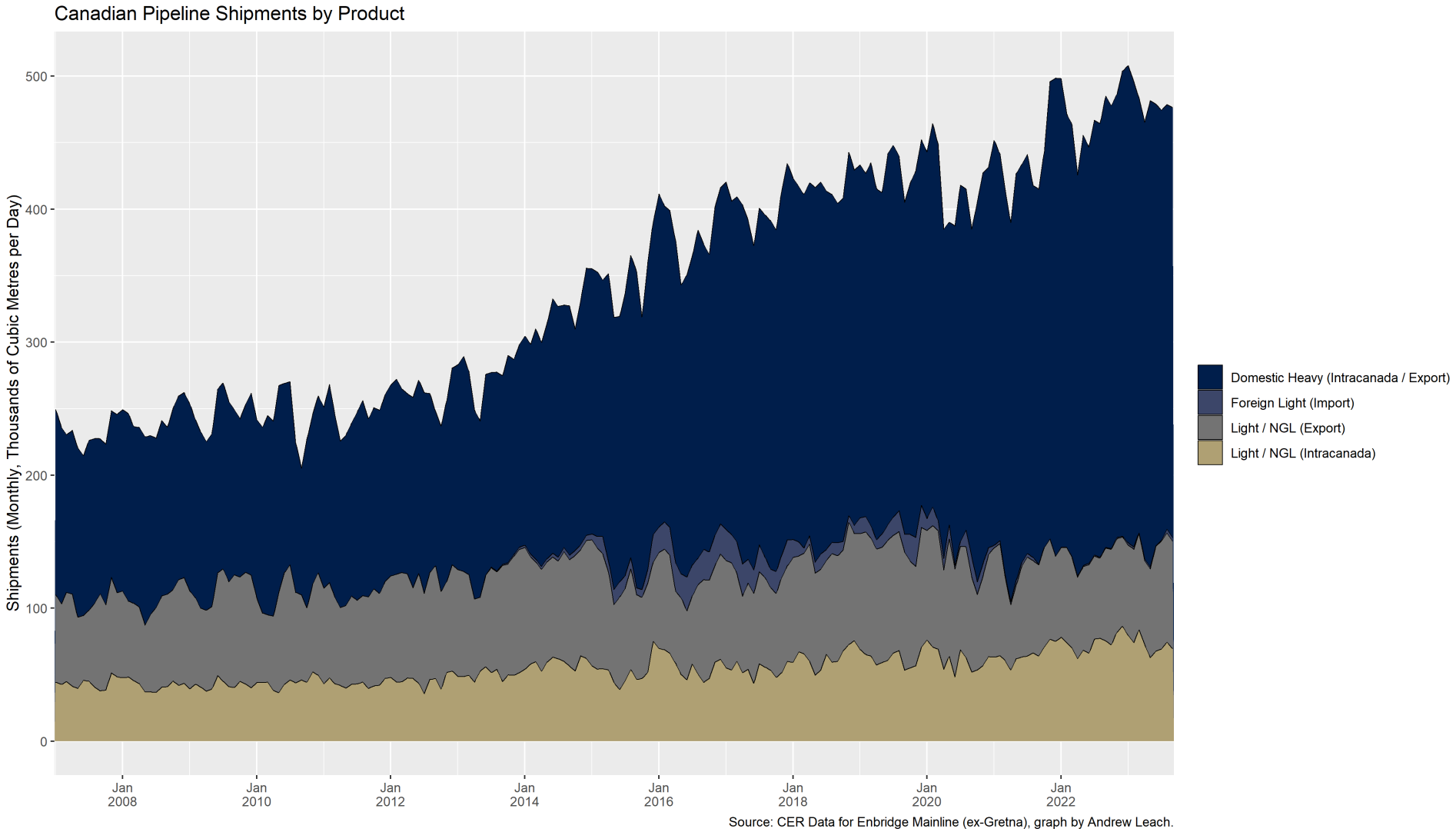
```
Rows: 1915 Columns: 15
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (7): Company, Pipeline, Key Point, Direction Of Flow, Trade Type, Produ...
dbl  (7): Month, Year, Latitude, Longitude, Throughput (1000 m3/d), Nameplat...
date (1): Date
```

```
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

You can also use a function to store the graph parameters you use most often. For example, I could have code like this:

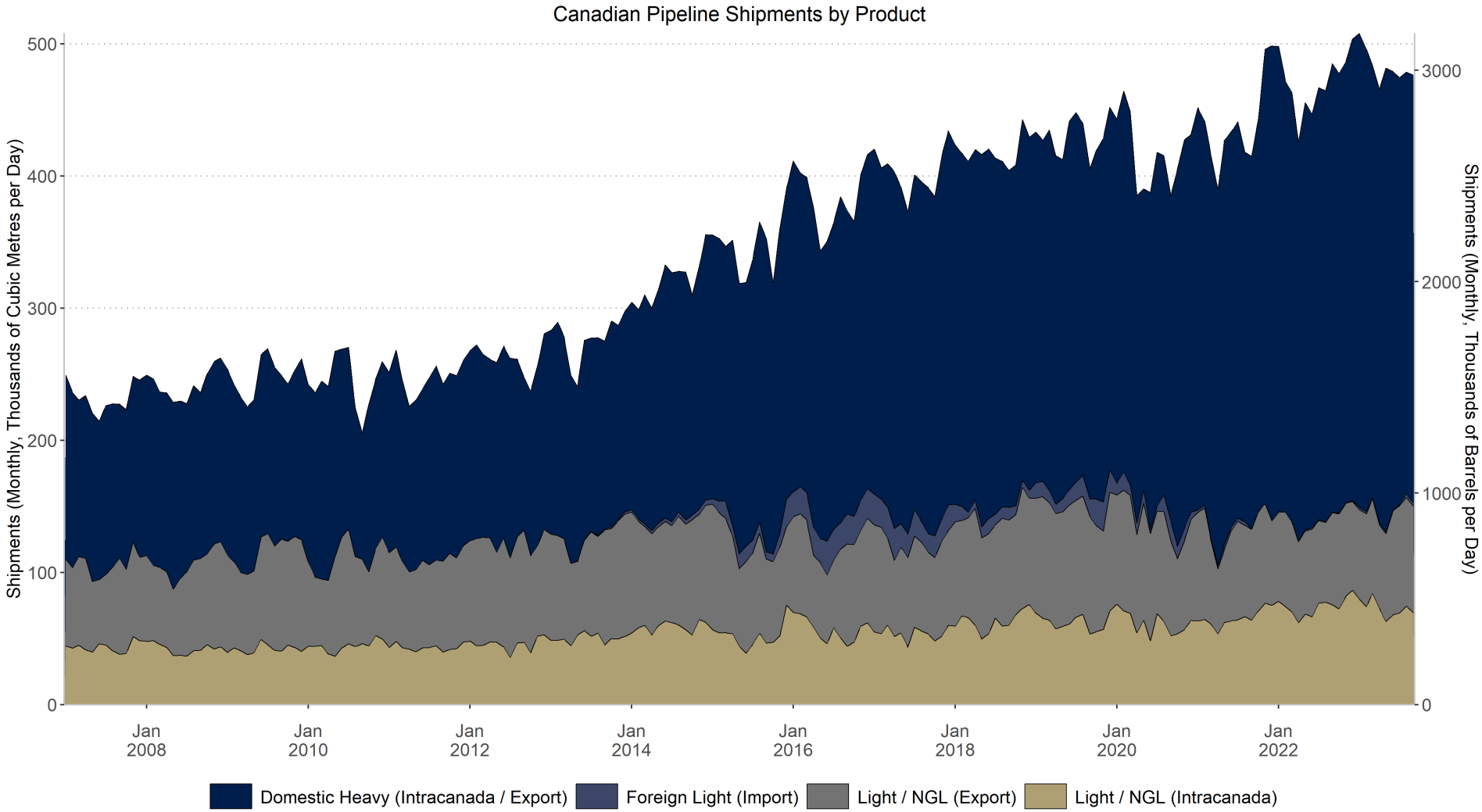
```
leach_graph<-function(){
  theme_classic() +
  theme(panel.border = element_blank(),
        panel.grid = element_blank(),
        panel.grid.major.y = element_line(color = "gray",linetype="dotted"),
        axis.line.x = element_line(color = "gray"),
        axis.line.y = element_line(color = "gray"),
        axis.text = element_text(size = 12),
        axis.text.x = element_text(margin = margin(t = 10)),
        axis.title = element_text(size = 12),
        plot.subtitle = element_text(size = 12,hjust=0.5),
        plot.caption = element_text(face="italic",size = 12,hjust=0),
        legend.key.width=unit(2,"line"),
        legend.position = "bottom",
        legend.box = "vertical",
        legend.text = element_text(size = 12),
        plot.title = element_text(hjust=0.5,size = 14))
}
```

and, suppose I had a basic graph called `enb_plot` made from the Enbridge data that looked like this:



I could clean it up by using my function (and one of the hints from the assignment) to look like this:

```
enb_plot+leach_graph()+
  scale_y_continuous(expand = c(0, 0),
    sec.axis = sec_axis( trans=~.*1/.16, name="Shipments (Monthly, Thousands of Barrels per Day)"))
```



Source: CER Data for Enbridge Mainline (ex-Gretna), graph by Andrew Leach.

I hope some of this is helpful for you.