Data Exercise, Week 8

This exercise has two purposes: first, to introduce you to electricity data, and second to build some new skills in R.

For this exercise, you're going to apply a couple of new and really important skills. First, you're going to learn how to download data using an Application Programming Interface (API). Second, you're going to use the functions skill we learned last week with the pipeline tolls assignment. You're going to apply those skills to make some graphs of Alberta electricity prices and loads.

Registering for an API

If you're going to use and API, you'll need to register with the data provider for a key. That key is your unique identifier, and ties any activity you have on the site to your email address and identity.

For this exercise, you'll need to sign up for a key for the Alberta Electric System Operator (AESO) API, which you can do here. Once you have your API key, you're ready for the next step.

Using your .Rprofile file

Generally, you don't want to share your API key, but you also want it to be loaded into R so that you can use it. So, how can you make this happen? One solution is to use your `.Rprofile` file that you'll see is in the home directory for R.

> How do you find the home directory for R? Open a clean session of R and type `getwd()` and you'll see it.

In your home directory, there will be a blank `.Rprofile` file that is, basically, R code that gets read each time you open an R session. You can store lots of stuff in there, and you can use it to create new variables with things like your AESO API key like this:

```
aeso_key<-"sdlkfjslkfjskldjhvklsHDFKJSDHFKJDSHFLDHFLDHFSHDKJShFCDKJshfhsjfkasHdjhsafkjshgvHDKJHSDkjbvDKJV"
```

If you do this, each time you open an R session, you can use `aeso_key` where you need to use your key to access data.

You can also use other methods to store keys: The `.Renviron` file gives you another way to manage keys, but I find it more cumbersome to use environment variables than plain variables, so this is not how I use it. But, if you want to try this approach, you can set an environment variable by including in the `.Renviron` file entries like this:

```
Key1=value1
Key2=value2
```

and then you would use `Sys.getenv("Key1")` to return "value1" in an R session. See here for more details.

Basically, I don't care what method you use, but **do not** make an RMarkdown document that contains your API key. On future assignments, I will deduct marks if you do so (and, to be fair, there may be bonus marks if you can find any of my keys online).

You won't necessarily get every aspect of this, and that's fine. I want you to come away with a general knowledge that APIs exist, that R can scrape them, and that they are an efficient way to access data online. I won't expect you to build your own API access code, but some of you might like to do so.

On with the AESO data

As usual, we'll start with a set of packages for this exercise, but in this case we're going to add two new ones: `jsonlite` and `httr`. `httr` lets you access web-based data and `jsonlite` lets you read .json files which are a common and efficient output format for API data. You'll need to install these packages if you haven't done so already. We've seen `lubridate` before, but you're really going to want to use it for this exercise too, so if you haven't already installed it, you should.

```
library(tidyverse)
library(httr)
library(jsonlite)
library(lubridate)
library(janitor)
library(kableExtra)
library(scales)
```

Now, before we get to the data, what are we looking at? The Alberta Electric System Operator (AESO) runs Alberta's wholesale electricity market, its transmission system (the *grid*), and ensures reliability of our electricity supply. At each instant, the AESO acts to ensure that electricity supply equals demand by dispatching new generating assets into the market as needed. Those generating assets each offer their generation into the market in block prices for each hour, so the AESO is **literally** working with a supply curve that we'll look at in class.

In each hour, the quantity of electricity demand is referred to as load, and the market price that allows offered supply to equal load is what each generator (and each distributor) pays for electricity.

In this exercise, we're going to graph prices and loads and some other bits and pieces using the AESO API.

Now, we'll start by *pulling* 1 year of pool price data, current to today(!!!), from the AESO API (you can also download these data from here, but I'd like you to try it using the API).

API calls rely on web links with headers to access data. The headers send information to the API, and tell you what you want back. In this case, we're going to use start dates (today's date), and end date (one year ago), we're going to tell it to give us json data, and you'll need to know which API site you want to access:

> For the pool price report, as you can see here, we're going to need the `/v1.1/price/poolPrice` API link.

And, we can access the data like so:

```
# we're going to need to convert our start and end dates to date format using ymd from lubridate
start_date<-Sys.Date()-years(1)
end_date<-Sys.Date()-days(1)

#using the GET command from httr, we'll access the API site
data <- GET(url="https://api.aeso.ca/report/v1.1/price/poolPrice",
    #add aeso key using the variable name you used in your .Rprofile
    add_headers("X-API-Key" = aeso_key),
    #we want to add some other headers
    query = list(
      startDate = format(start_date, "%Y-%m-%d"),
      endDate = format(end_date, "%Y-%m-%d"),
      contentType = "application/json"))
# the json output is a list with a bunch of information, but for this purpose, we only want the content
```

```
data <- data %>%
  httr::content()

#and then we want to extract the data and convert it to a data frame

data<-data$return$`Pool Price Report` %>% enframe()%>%unnest_auto(value)

#you could also use this method to get to the same place

#data <- as.data.frame(fromJSON(rawToChar(data$content)))%>%clean_names()
```

If you run your code chunks to here, you should see that you have, in memory, an 8761 (or, if it's a leap year, 8785) hour file with some pretty gross variable names and such. We're going to clean that up, but since this is code we're going to use over and over, let's make a **function** to do it. Here, you're basically creating a new command that you can run in R, you can send it some information, and it will send you output back. We'll make a function to return some cleaned-up output from the AESO API, using the pool price report we just generated as a default:

```
aeso_prices<-function(key,start=ymd("2023-01-01"),end=ymd("2023-12-31")){ #these are default dates
  #default dates mean that if you run aeso_prices(key), it will use the 2023 year by default.

data <- GET(url="https://api.aeso.ca/report/v1.1/price/poolPrice",
    add_headers("X-API-Key" = key), #use the variable that's sent to the function, just called key
    #we want to add some other headers
    query = list(
      startDate = format(start, "%Y-%m-%d"),
      endDate = format(end, "%Y-%m-%d"),
      contentType = "application/json"))
# the json output is a list with a bunch of information
data<- data %>%
  httr::content()

#and then we want to extract the data and convert it to a data frame

data<-data$return$`Pool Price Report` %>% enframe()%>%unnest_auto(value)%>% select(-1)


names(data)<-gsub("return_pool_price_report_","",names(data))
# recode the times into posixct objects
```

```
data<-data %>%   mutate(begin_datetime_utc=ymd_hm(begin_datetime_utc,tz="UTC"),
         begin_datetime_mpt=ymd_hm(begin_datetime_mpt,tz="America/Denver"))
data <- data %>% rename(time_mt=begin_datetime_mpt,
                        time_utc=begin_datetime_utc)%>%
  #code for hour ending, since a lot of electricity works on this
  mutate(he=hour(time_mt)+1,
         pool_price=as.numeric(pool_price),
         forecast_pool_price=as.numeric(forecast_pool_price),
         rolling_30day_avg=as.numeric(rolling_30day_avg)
         )
#you could also use return(data) to send the data back to the calling part of your code, but you don't need to
data
}

#you can now call this function anywhere below this chunk in your code to get a data frame of pool price data, like this:

#test_data<-aeso_prices(key=aeso_key)
```

Let's try it to get today's (!!!) data:

```
aeso_prices(key=aeso_key,start=Sys.Date(),end=Sys.Date()) %>% #get data for today!
     kbl(col.names = c('Time (UTC)', 'Time (Mountain)', 'Pool Price', 'Forecast Pool Price', 'Rolling 30 day avg','Hour ending'))%
    kable_styling(fixed_thead = T,bootstrap_options = c("hover", "condensed","responsive"),full_width = T)%>%
      I()
```

Time (UTC)	Time (Mountain)	Pool Price	Forecast Pool Price	Rolling 30 day avg	Hour ending
2024-03-12 06:00:00	2024-03-12 00:00:00	74.7	119.8	68.8	1
2024-03-12 07:00:00	2024-03-12 01:00:00	81.6	107.5	68.9	2
2024-03-12 08:00:00	2024-03-12 02:00:00	46.1	47.6	68.9	3
2024-03-12 09:00:00	2024-03-12 03:00:00	44.6	48.2	69.0	4
2024-03-12 10:00:00	2024-03-12 04:00:00	34.8	41.9	69.0	5
2024-03-12 11:00:00	2024-03-12 05:00:00	39.1	42.5	69.0	6
2024-03-12 12:00:00	2024-03-12 06:00:00	74.2	79.9	69.1	7

Time (UTC)	Time (Mountain)	Pool Price	Forecast Pool Price	Rolling 30 day avg	Hour ending
2024-03-12 13:00:00	2024-03-12 07:00:00	198.7	279.1	69.3	8
2024-03-12 14:00:00	2024-03-12 08:00:00	54.4	130.3	69.3	9
2024-03-12 15:00:00	2024-03-12 09:00:00	23.2	23.0	69.2	10
2024-03-12 16:00:00	2024-03-12 10:00:00	NA	24.0	NA	11
2024-03-12 17:00:00	2024-03-12 11:00:00	NA	104.3	NA	12
2024-03-12 18:00:00	2024-03-12 12:00:00	NA	23.3	NA	13

So, what are you actually seeing? Two time variables, one in local time and one in UTC, an actual and 3-hour-ahead forecast of the pool price, a rolling 30-day average pool price, and lastly an important thing about electricity markets: we tell time at the end of the hour. So he12 is the hour that ends at noon.

The fun thing about using a function is that now that we've made one, it's easy to make a new version for different reports, like this one to get actual and forecast internal loads (or demand) measured in MW:

```
aeso_forecasts<-function(key,starting=ymd("2023-01-01"),ending=ymd("2023-12-31")){

data <- GET(url="https://api.aeso.ca/report/v1/load/albertaInternalLoad",
    add_headers("X-API-Key" = aeso_key,"contentType" = "application/json"), #use the variable that's sent to the function, just c
    #we want to add some other headers
    query = list(
      startDate=format(starting, "%Y-%m-%d"),
      endDate=format(ending, "%Y-%m-%d")))
# the json output is a list with a bunch of information
# the json output is a list with a bunch of information
data<- data %>%
  httr::content()

#and then we want to extract the data and convert it to a data frame

data<-data$return$`Actual Forecast Report` %>% enframe()%>%unnest_auto(value)%>%select(-1)

names(data)<-gsub("return_actual_forecast_report_","",names(data))
names(data)<-gsub("alberta_","",names(data))
```

```
# recode the times into posixct objects
data<-data %>%   mutate(begin_datetime_utc=ymd_hm(begin_datetime_utc,tz="UTC"),
         begin_datetime_mpt=ymd_hm(begin_datetime_mpt,tz="America/Denver"))
data <- data %>% rename(time_mt=begin_datetime_mpt,
                        time_utc=begin_datetime_utc)%>%
  #code for hour ending, since a lot of electricity works on this
  mutate(he=hour(time_mt)+1,
         internal_load=as.numeric(internal_load),
         forecast_internal_load=as.numeric(forecast_internal_load)
         )
data
}
```

Let's try this one too:

```
aeso_forecasts(key=aeso_key,start=Sys.Date(),end=Sys.Date()) %>%
   kbl(col.names = c('Time (UTC)', 'Time (Mountain)', 'Internal Load', 'Forecast Load','Hour ending')) %>%
    kable_styling(fixed_thead = T,bootstrap_options = c("hover", "condensed","responsive"),full_width = T)%>%
      I()
```

Time (UTC)	Time (Mountain)	Internal Load	Forecast Load	Hour ending
2024-03-12 06:00:00	2024-03-12 00:00:00	9683	9747	1
2024-03-12 07:00:00	2024-03-12 01:00:00	9547	9592	2
2024-03-12 08:00:00	2024-03-12 02:00:00	9521	9518	3
2024-03-12 09:00:00	2024-03-12 03:00:00	9548	9511	4
2024-03-12 10:00:00	2024-03-12 04:00:00	9585	9596	5
2024-03-12 11:00:00	2024-03-12 05:00:00	9721	9736	6
2024-03-12 12:00:00	2024-03-12 06:00:00	10194	10095	7
2024-03-12 13:00:00	2024-03-12 07:00:00	10584	10570	8
2024-03-12 14:00:00	2024-03-12 08:00:00	10621	10636	9
2024-03-12 15:00:00	2024-03-12 09:00:00	10693	10639	10

Time (UTC)	Time (Mountain)	Internal Load	Forecast Load	Hour ending
2024-03-12 16:00:00	2024-03-12 10:00:00	NA	10665	11
2024-03-12 17:00:00	2024-03-12 11:00:00	NA	10642	12
2024-03-12 18:00:00	2024-03-12 12:00:00	NA	10594	13
2024-03-12 19:00:00	2024-03-12 13:00:00	NA	10574	14
2024-03-12 20:00:00	2024-03-12 14:00:00	NA	10529	15
2024-03-12 21:00:00	2024-03-12 15:00:00	NA	10508	16
2024-03-12 22:00:00	2024-03-12 16:00:00	NA	10536	17
2024-03-12 23:00:00	2024-03-12 17:00:00	NA	10531	18
2024-03-13 00:00:00	2024-03-12 18:00:00	NA	10420	19
2024-03-13 01:00:00	2024-03-12 19:00:00	NA	10459	20
2024-03-13 02:00:00	2024-03-12 20:00:00	NA	10593	21
2024-03-13 03:00:00	2024-03-12 21:00:00	NA	10444	22
2024-03-13 04:00:00	2024-03-12 22:00:00	NA	10207	23
2024-03-13 05:00:00	2024-03-12 23:00:00	NA	9941	24

So, now we have two separate functions that pull in price and load data, but I would rather we build a single data set, so let's do that next using the `left_join` command. This will combine the data sets we pull in using common elements (times, he, etc) if they have common names, or you can tell it which variables to use to execute the merge.

Are you ready for this? It's pretty cool and one of the more powerful things about using scripted statistical programming. We're going to grab the 2023 year:

```
start_date<-ymd("2023-01-01")
end_date<-ymd("2023-12-31")
#since we have the same variable names, it will automatically join by common elements c("time_utc", "time_mt", "he")
aeso_combined<-aeso_forecasts(key=aeso_key,starting = start_date,ending=end_date)%>%
  left_join(aeso_prices(key=aeso_key,start=start_date,end=end_date))
```

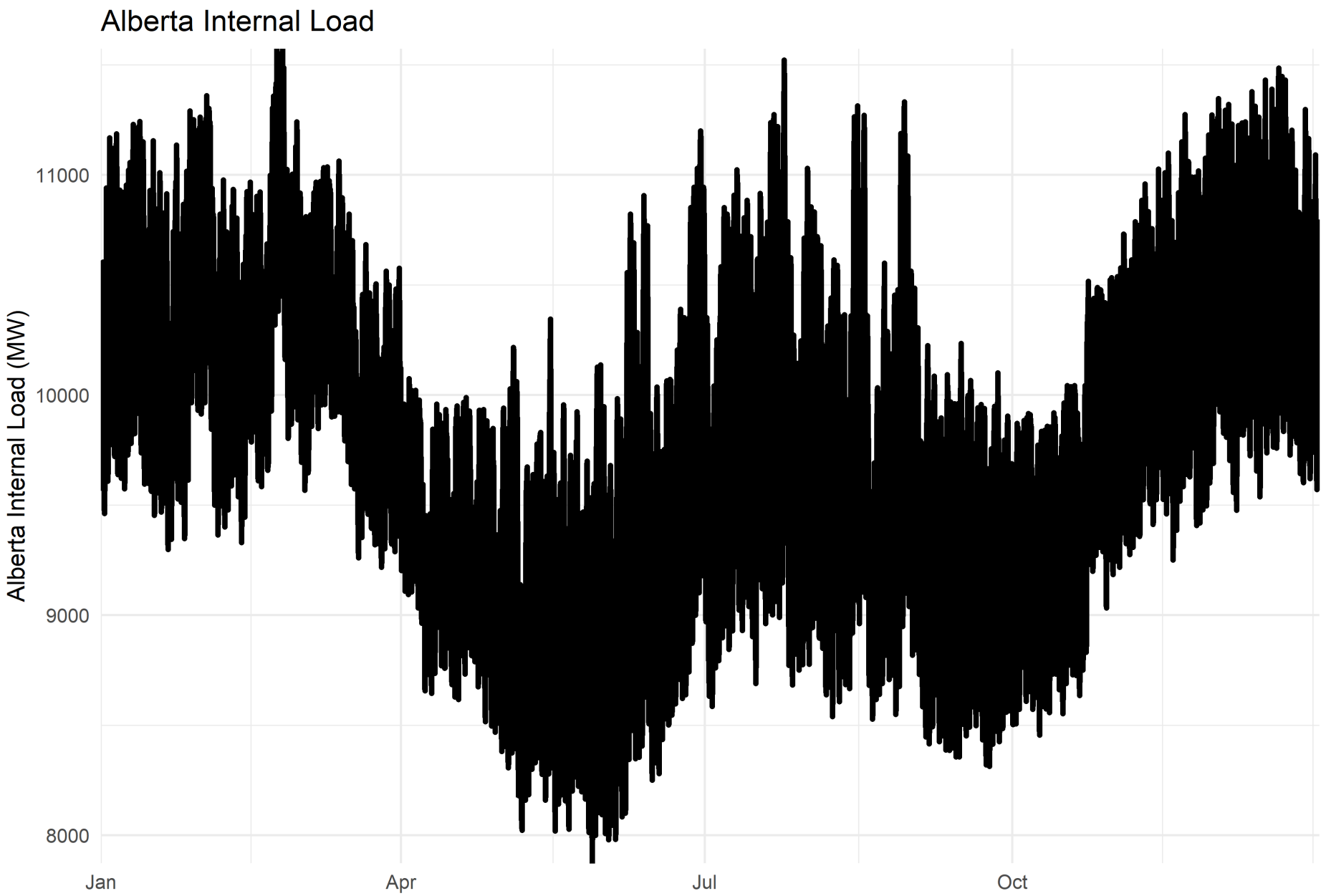
If you run all your chunks to here, you should have an 8760x8 (or 8784x8 if it's a leap year) data frame with prices and loads. We can show a subset (Christmas day from 10am-5pm) of our load and price data.

```
aeso_combined %>% filter(as_date(time_mt)==ymd("2023-12-25"),he>10,he<19)%>%
   kbl() %>%
    kable_styling(fixed_thead = T,bootstrap_options = c("hover", "condensed","responsive"),full_width = T)%>%
      I()
```

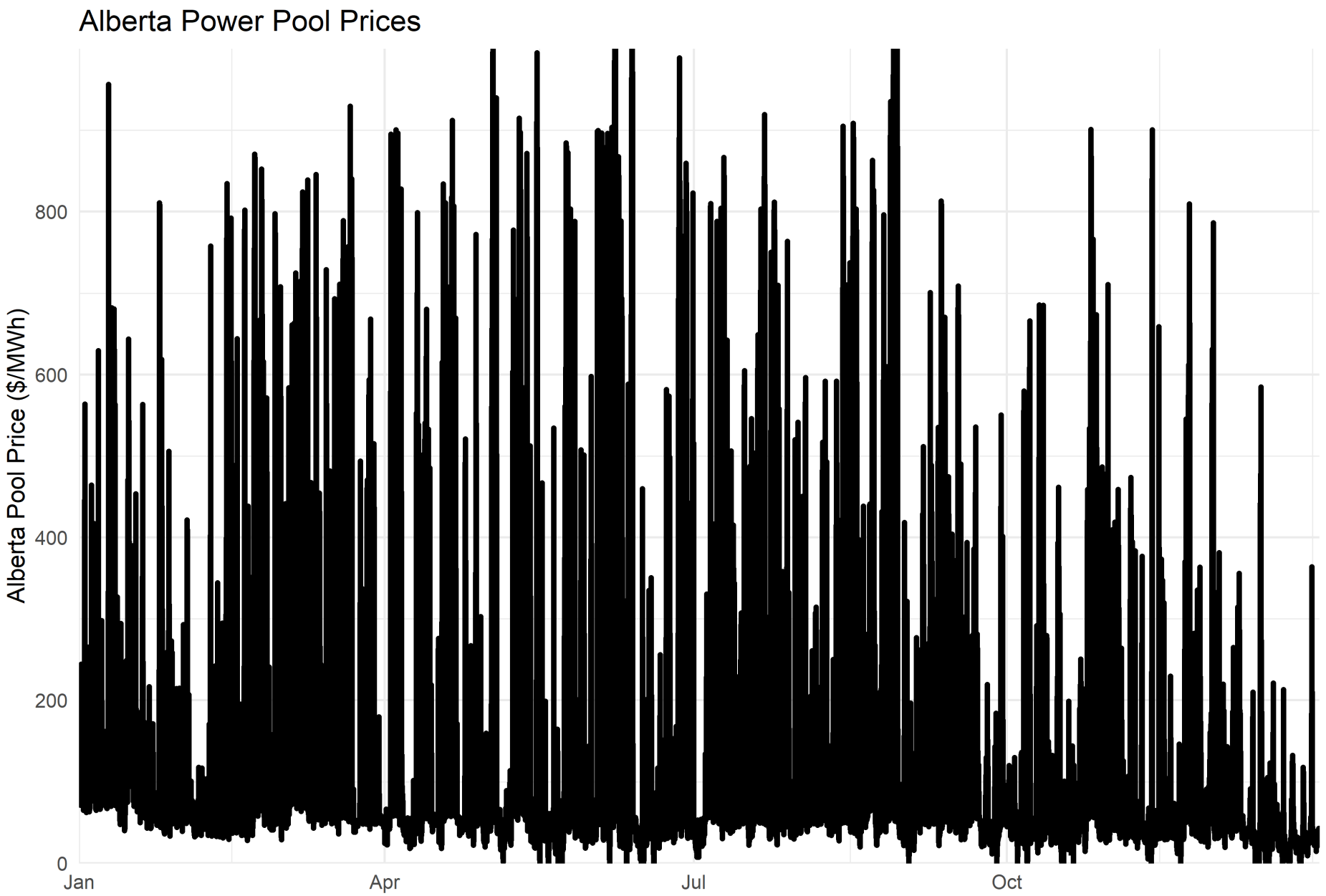
time_utc	time_mt	internal_load	forecast_internal_load	he	pool_price	forecast_pool_price	rolling_30day_avg
2023-12-25 17:00:00	2023-12-25 10:00:00	10507	10539	11	23.7	25.3	65.3
2023-12-25 18:00:00	2023-12-25 11:00:00	10469	10493	12	27.0	23.8	65.3
2023-12-25 19:00:00	2023-12-25 12:00:00	10431	10448	13	29.2	26.3	65.3
2023-12-25 20:00:00	2023-12-25 13:00:00	10434	10399	14	27.0	26.0	65.3
2023-12-25 21:00:00	2023-12-25 14:00:00	10418	10417	15	26.4	25.2	65.3
2023-12-25 22:00:00	2023-12-25 15:00:00	10448	10492	16	31.0	30.8	65.3
2023-12-25 23:00:00	2023-12-25 16:00:00	10712	10670	17	28.5	31.6	65.2
2023-12-26 00:00:00	2023-12-25 17:00:00	10830	10807	18	22.6	27.7	65.2

And, we can also make quick graphs of the data you've downloaded:

```
aeso_combined%>%ggplot()+
  geom_line(aes(time_mt,internal_load),color="black",linewidth=1.2)+
  scale_x_datetime(breaks=pretty_breaks(), expand=c(0,0))+
  scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  labs(
      title = "Alberta Internal Load",
      y="Alberta Internal Load (MW)",x=NULL)+
    theme_minimal()+
  theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```


Alberta Internal Load
Alberta Internal Load (MW)
11000
10000
9000
8000
Jan
Apr
Jul
Oct

```
aeso_combined%>%ggplot()+
  geom_line(aes(time_mt,pool_price),color="black",linewidth=1.2)+
  scale_x_datetime(breaks=pretty_breaks(), expand=c(0,0))+
  scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  labs(
      title = "Alberta Power Pool Prices",
      y="Alberta Pool Price ($/MWh)",x=NULL)+
    theme_minimal()+
  theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```



So, now you've learned three new skills: API data pulls, creating functions, and merging data files.

And we've made some ugly graphs.

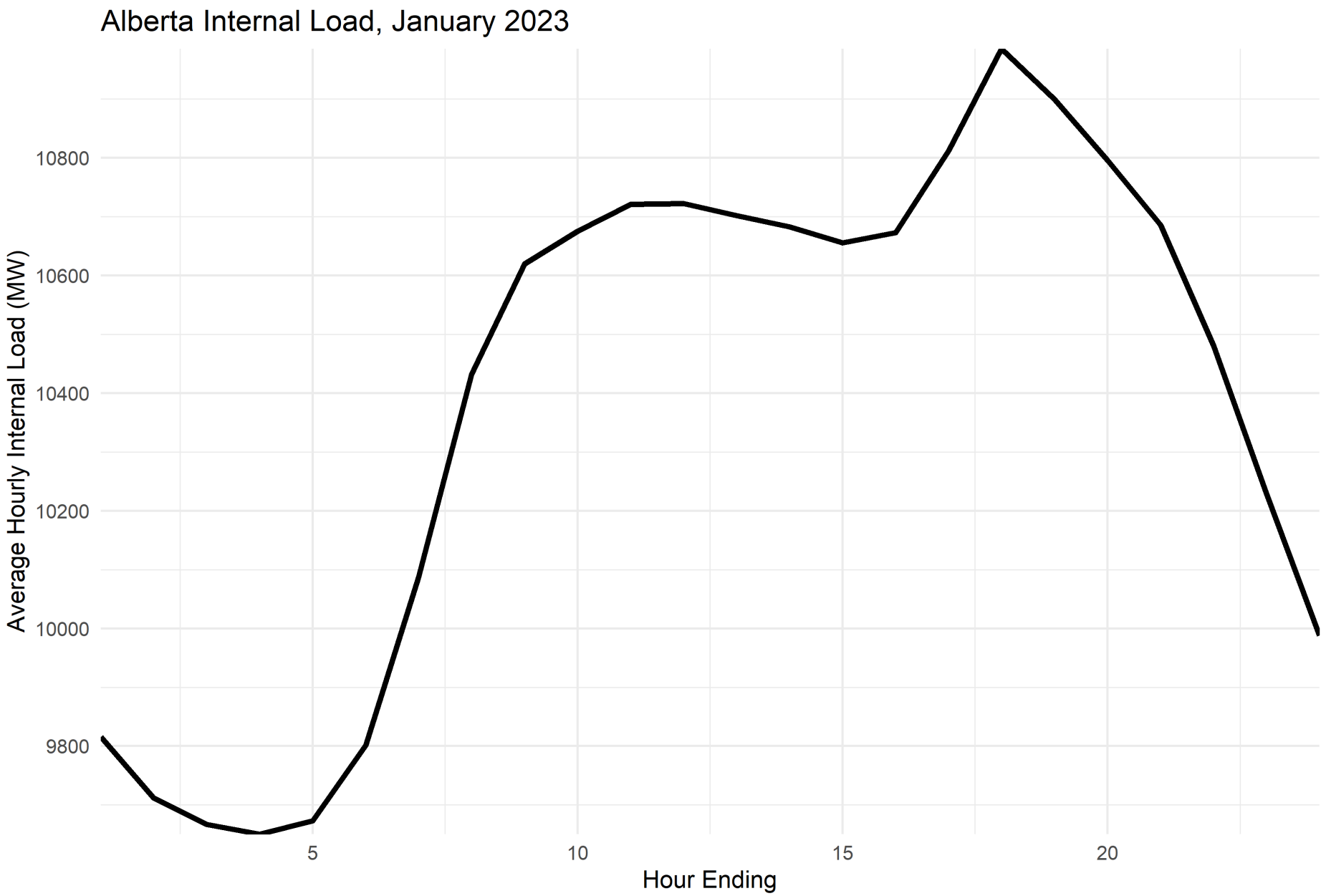
Now, it's time to clean this up using a really important data wrangling skill: grouping. And, a related graphing skill: ribbon plots.

As you've seen before, `group_by` (and the related command `ungroup`) let you execute commands on data grouped by time, by another factor, etc. I use it often with `mutate` (to create group averages or deviations from the average) or `summarize` (to create monthly data from daily data, for example). In the example below, we'll group by month and hour, and then summarize to get hourly means, min and max by month (e.g the mean, min, and max value at 2pm in January).

If you look at the graphs you've made, you probably already had the feeling that electricity load is seasonal and has daily variation too. We'll use `group_by` and `summarize` to have a look at that.

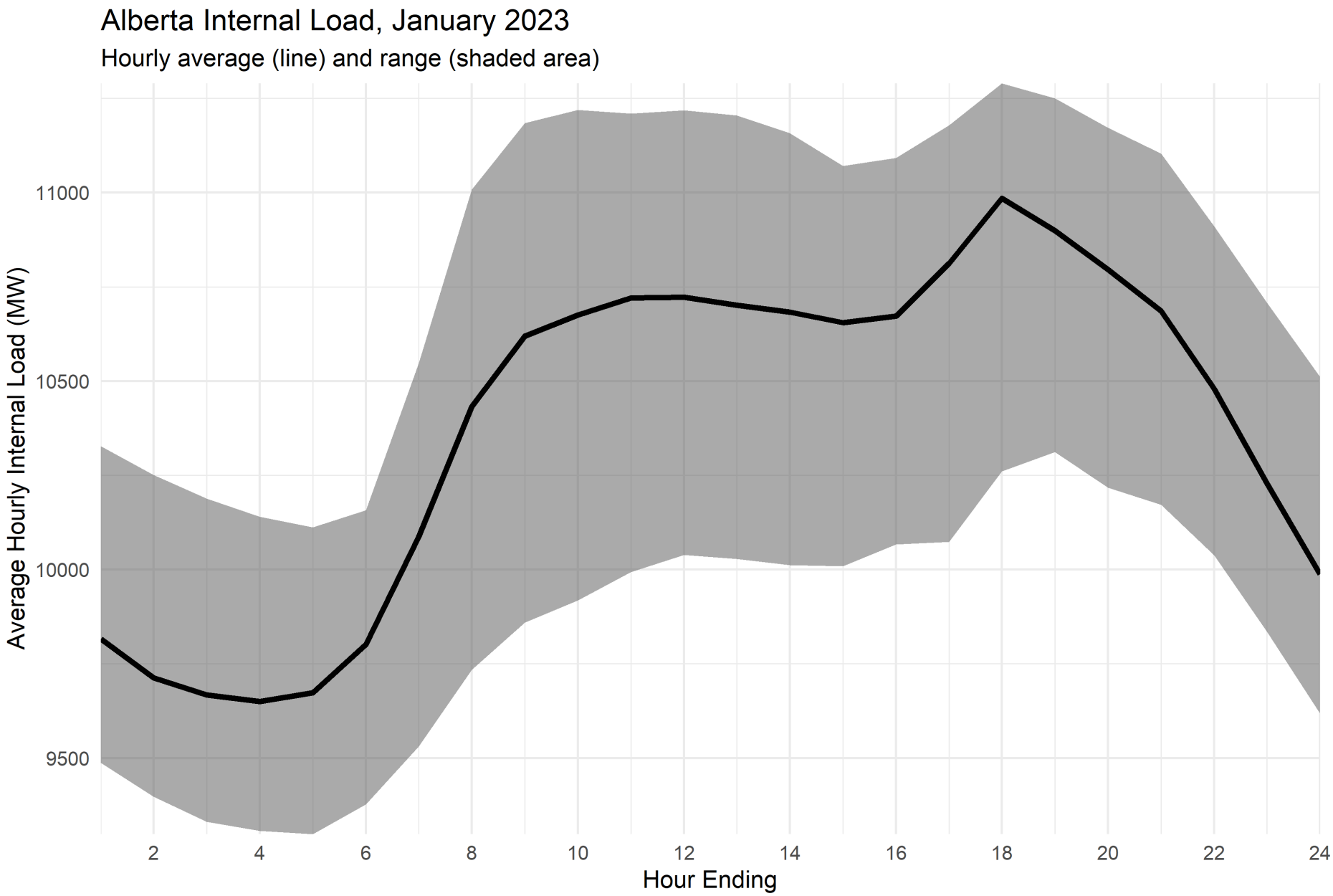
Pay close attention to the steps in this code:

```
aeso_combined %>%
  mutate(month=month(time_utc))%>% # create month indicator variables
  filter(month==1)%>% #select january
  group_by(month,he) %>% #group data by month and he
  summarize(load=mean(internal_load,na.rm = T),# for each month and he pair, create a summary variable called internal_load that
            load_min=min(internal_load), #and one for the miniumum
            load_max=max(internal_load), #and one for the maximum
  )%>%
  ggplot()+
  geom_line(aes(he,load),color="black",linewidth=1.2)+
  scale_x_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  labs(
      title = "Alberta Internal Load, January 2023",
      y="Average Hourly Internal Load (MW)",x="Hour Ending")+
    theme_minimal()+
  theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```


Alberta Internal Load, January 2023
Average Hourly Internal Load (MW)
10800
10600
10400
10200
10000
9800
5
10
15
20
Hour Ending

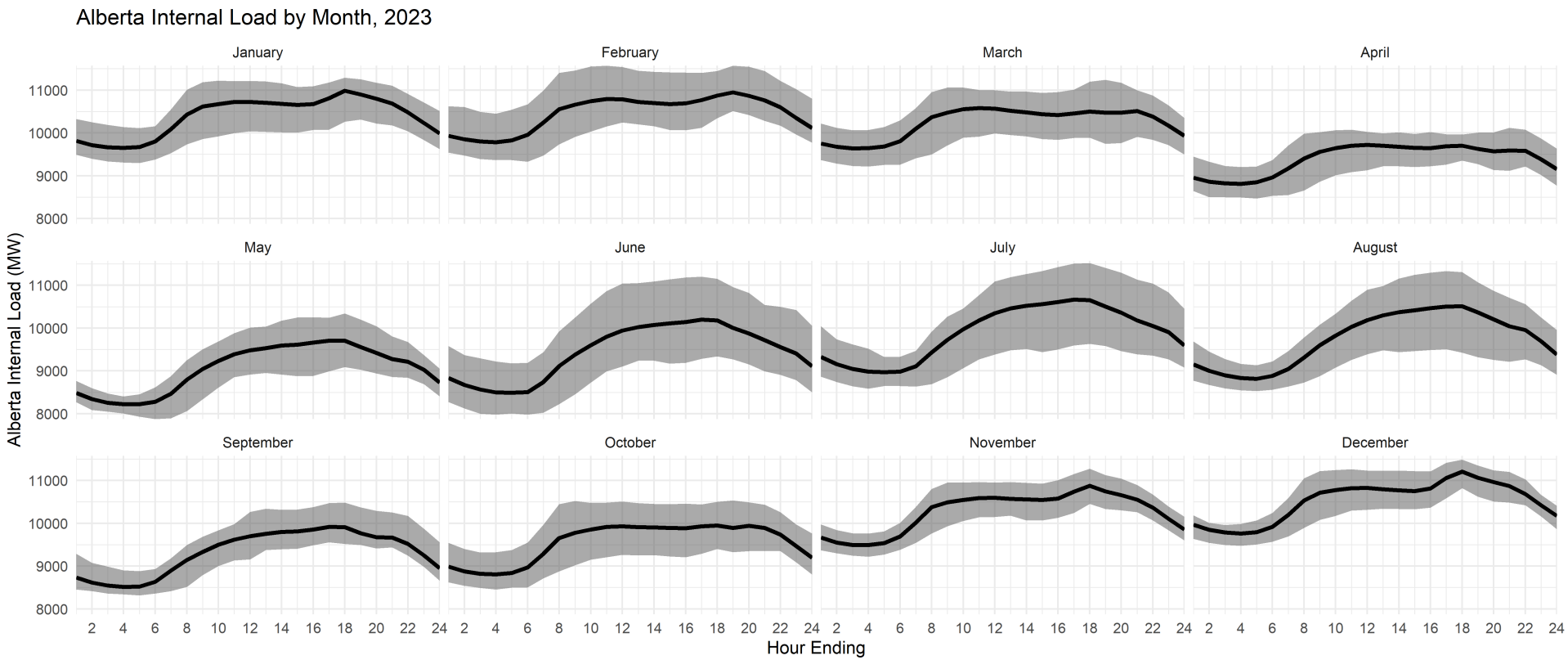
Now, that's definitely seasonal, but there's a bit that's getting lost in the mix here - the range of possible values. Let's use a ribbon plot for this and take advantage of the minima and maxima that we made earlier.

```
aeso_combined %>%
  mutate(month=month(time_utc))%>% # create month indicator variables
  filter(month==1)%>% #select january
  group_by(month,he) %>% #group data by month and he
  summarize(load=mean(internal_load,na.rm = T),# for each month and he pair, create a summary variable called internal_load that
            load_min=min(internal_load), #and one for the miniumum
            load_max=max(internal_load), #and one for the maximum
  )%>%
  ggplot()+
  #LOOK HERE
  #adding in the ribbon
  geom_ribbon(aes(min=load_min,max=load_max,x=he),alpha=0.4)+
  geom_line(aes(he,load),color="black",linewidth=1.2)+
  scale_x_continuous(breaks=pretty_breaks(12), expand=c(0,0))+
  scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  labs(
      title = "Alberta Internal Load, January 2023",
      subtitle = "Hourly average (line) and range (shaded area)",
      y="Average Hourly Internal Load (MW)",x="Hour Ending")+
    theme_minimal()+
  theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```



But, you know that there's also variation across months, so let's use a faceted plot to show that information.

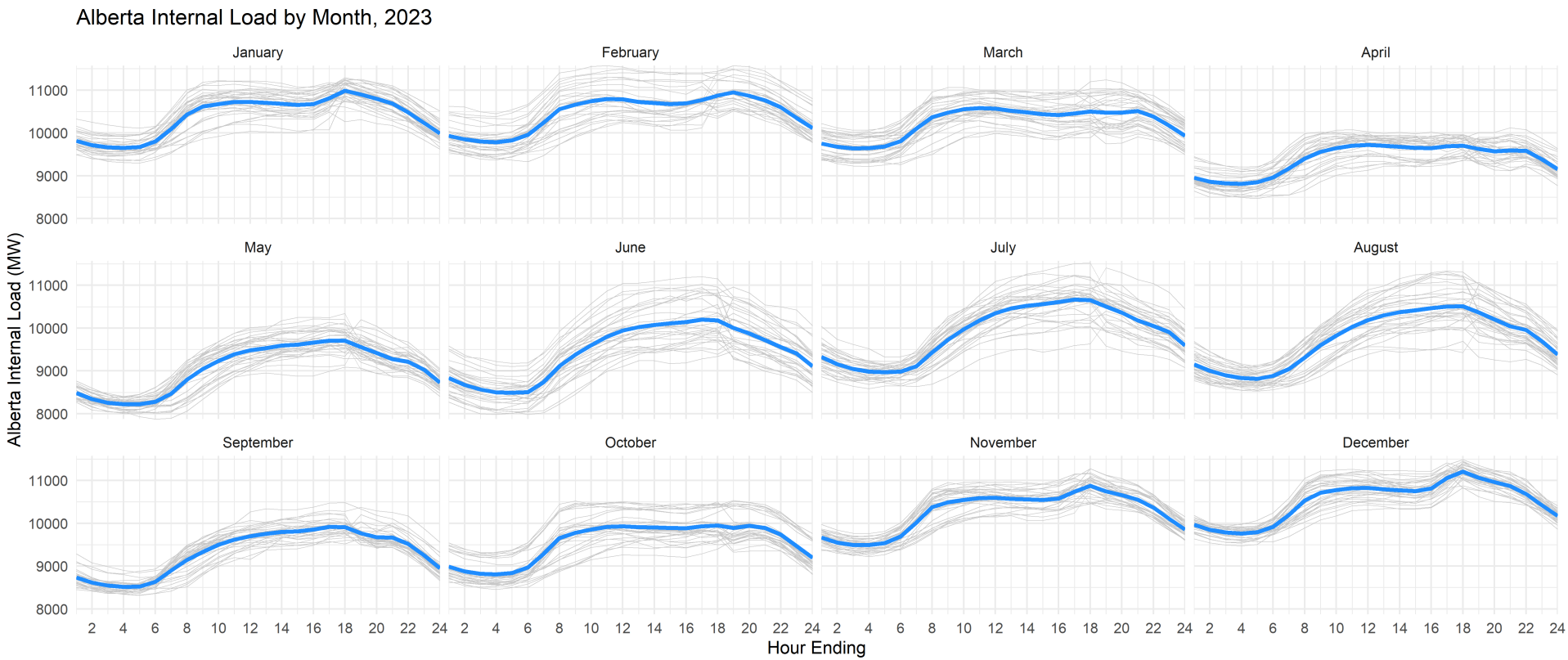
```
# a function to label months
month_string <- function(month_sent) {
  #substr(string, 1, 1) <- toupper(substr(string, 1, 1))
  month.name[as.numeric(month_sent)]
}

aeso_combined %>%
  mutate(month=month(time_utc))%>% # create month indicator variables
  group_by(month,he) %>% #group data by month and he
  summarize(load=mean(internal_load,na.rm = T),# for each month and he pair, create a summary variable called internal_load that
            load_min=min(internal_load), #and one for the miniumum
            load_max=max(internal_load), #and one for the maximum
  )%>%
  ggplot()+
  geom_ribbon(aes(min=load_min,max=load_max,x=he),alpha=0.4)+
  geom_line(aes(he,load),color="black",linewidth=1.2)+
  scale_x_continuous(breaks=pretty_breaks(12), expand=c(0,0))+
  scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
  #look here, adding in the facet wrap and the labeler - you send the variable month to the function month_string to get the labe
  facet_wrap(~month,labeller = labeller(month=month_string))+
  labs(
      title = "Alberta Internal Load by Month, 2023",
      y="Alberta Internal Load (MW)",x="Hour Ending")+
    theme_minimal()+
  theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```



There's another way to do this which, in some circumstances, you might prefer: instead of using the solid ribbon, you can trace individual daily lines in a lighter shade with thinner lines. Let me show you that alternative here:

```
month_string <- function(month_sent) {
  #substr(string, 1, 1) <- toupper(substr(string, 1, 1))
  month.name[as.numeric(month_sent)]
}
aeso_combined %>%
  #LOOK HERE: added days
  mutate(month=month(time_utc),
         day=day(time_utc))%>% # create month and day indicator variables
  group_by(month,he) %>% #group data by month and he
  #LOOK HERE
```

```
#use mutate to create month/hour averages while keeping the original data
mutate(load=mean(internal_load,na.rm = T),# for each month and he pair, create a summary variable called internal_load that is
          load_min=min(internal_load), #and one for the miniumum
          load_max=max(internal_load), #and one for the maximum
)%>%
ggplot()+
#use thin lines with a lighter shade to graph each daily curve
geom_line(aes(he,internal_load,group=day),color="grey80",linewidth=0.2)+
# thicker lines for the means
geom_line(aes(he,load),color="dodgerblue",linewidth=1.2)+
scale_x_continuous(breaks=pretty_breaks(12), expand=c(0,0))+
scale_y_continuous(breaks=pretty_breaks(), expand=c(0,0))+
#look here, adding in the facet wrap and the labeler
facet_wrap(~month,labeller = labeller(month=month_string))+
labs(
    title = "Alberta Internal Load by Month, 2023",
    y="Alberta Internal Load (MW)",x="Hour Ending")+
  theme_minimal()+
theme(plot.margin = unit(c(1,1,0.2,1), "cm"))
```



That's a little inefficient, since it effectively draws a version of the mean line for each day, but it won't change your life too much here.

Now, for your exercise, I'd like to see you reproduce a similar set of ribbon and *spaghetti* graphs for prices. Here's the spaghetti plot for prices for your inspiration: