ECON 366 — Energy Economics Syllabus Schedule Course Content Course Deliverables Data and Resources

Data Exercise, Week 2

This week, our 💻 **Data Exercise** focuses on another key souce of energy data, the US Energy Information Administration and, as ever, we'll introduce a couple of new commands in R for you to learn.

I'd love to have you use the API from the EIA, but it's a mess, so we're going to use spreadsheet data again. In particular, we're going to use the data on oil and refined product prices here, although this week I am going to let you work out how to download the linked excel spreadsheet and read the data into R.

By this point, you should be working to prepare an RMarkdown file that compiles to an html document. You'll need that skill for the first data assignment, coming next week, so please make sure you've got it running. You can use `week2.Rmd` as your starting point. Knit it before you start playing with it, just so you see that it works, then make a clean version to use for this week's exercise.

Don't forget to include your basic R packages. We'll add two new ones this week: `scales` for nicer graphs and `kableExtra` for nice tables. The order can matter in cases where two packages have commands with the same names. The last one you load will be the one that defines what that particular command means when you run it.

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

When downloading the data, you might want a couple of hints:

- use `sheet="Data 1"` to get the WTI and Brent prices;
- use `skip=2` in your `read_excel` command to skip the first two lines of the spreadsheet.

So, if you've got that right, you should be able to generate a table that looks like this, showing that you've got two price series (WTI and Brent) and dates (I named by data `price_data` but you might have called it something else):

```
  price_data<-price_data %>%
  rename(wti=2,brent=3)
price_data%>%#start with my price data
  tail(10)%>% #grab the last 10 observations and pass it to kbl
  #everything below here just makes a nice table
   kbl(escape = FALSE,table.attr = "style='width:40%;'") %>%
    kable_styling(fixed_thead = T,bootstrap_options = c("hover", "condensed","responsive"),full_width = T)%>%
  scroll_box(width = "1000px", height = "400px")%>%
      I()
```

date	wti	brent
2023-12-18	72.2	78.9
2023-12-19	73.2	79.8
2023-12-20	73.9	81.1
2023-12-21	73.6	80.7
2023-12-22	73.3	80.2
2023-12-26	75.8	NA
2023-12-27	74.3	81.0
2023-12-28	72.0	79.0
2023-12-29	71.9	77.7
2024-01-02	70.6	76.2

You'll notice that I also used a couple of new bits of code in there:

- I used `price_data<-price_data %>% rename(wti=2,brent=3)` to rename columns by number;
- I used `tail(10)` to send the last 10 rows of data to the `kbl` table.

Like last week, you'll want to transform your data from ***wide***, with the variables one in each column, to ***long*** with data stacked vertically and an indicator variable (date) in this case. Try it out.

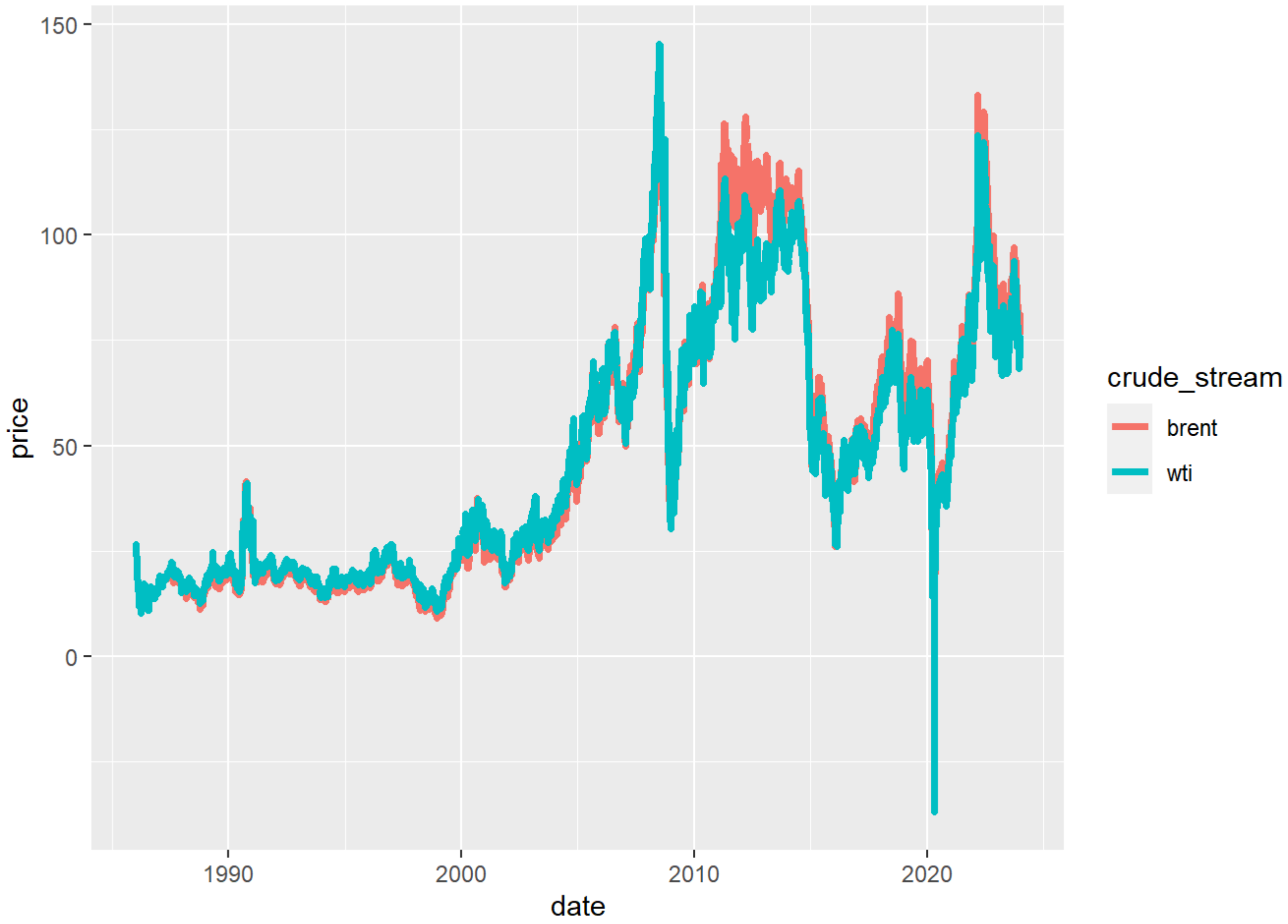
If you did it right, and you used `names_to = "crude_stream", values_to = "price"` in your `pivot_longer` command, you should get data that look something like this:

```
price_data %>% tail(10)%>%
  kbl(escape = FALSE,table.attr = "style='width:40%;'") %>%
    kable_styling(fixed_thead = T,bootstrap_options = c("hover", "condensed","responsive"),full_width = T)%>%
  scroll_box(width = "1000px", height = "400px")%>%
      I()
```

date	crude_stream	price
2023-12-26	wti	75.8
2023-12-26	brent	NA
2023-12-27	wti	74.3
2023-12-27	brent	81.0
2023-12-28	wti	72.0
2023-12-28	brent	79.0
2023-12-29	wti	71.9
2023-12-29	brent	77.7
2024-01-02	wti	70.6
2024-01-02	brent	76.2

The first new skill we want to learn this week is making a graph with multiple data series. Here's a basic example, with comments in the code:

```
price_data %>%
  ggplot()+ #make a graph
geom_line(aes(date,price,group=crude_stream,color=crude_stream),linewidth=1.25)
```



```
#the grouping tells ggplot that each different crude stream is a different line
# the colour tells ggplot to assign a different colour to each of the crude streams
```

Now we're going to get to one of the more powerful aspects of R graphs - dealing with dates and times. First, check your data: use the command from `janitor` to `describe_class` of your date variable:

```
describe_class(price_data$date)
```

```
[1] "POSIXct, POSIXt"
```

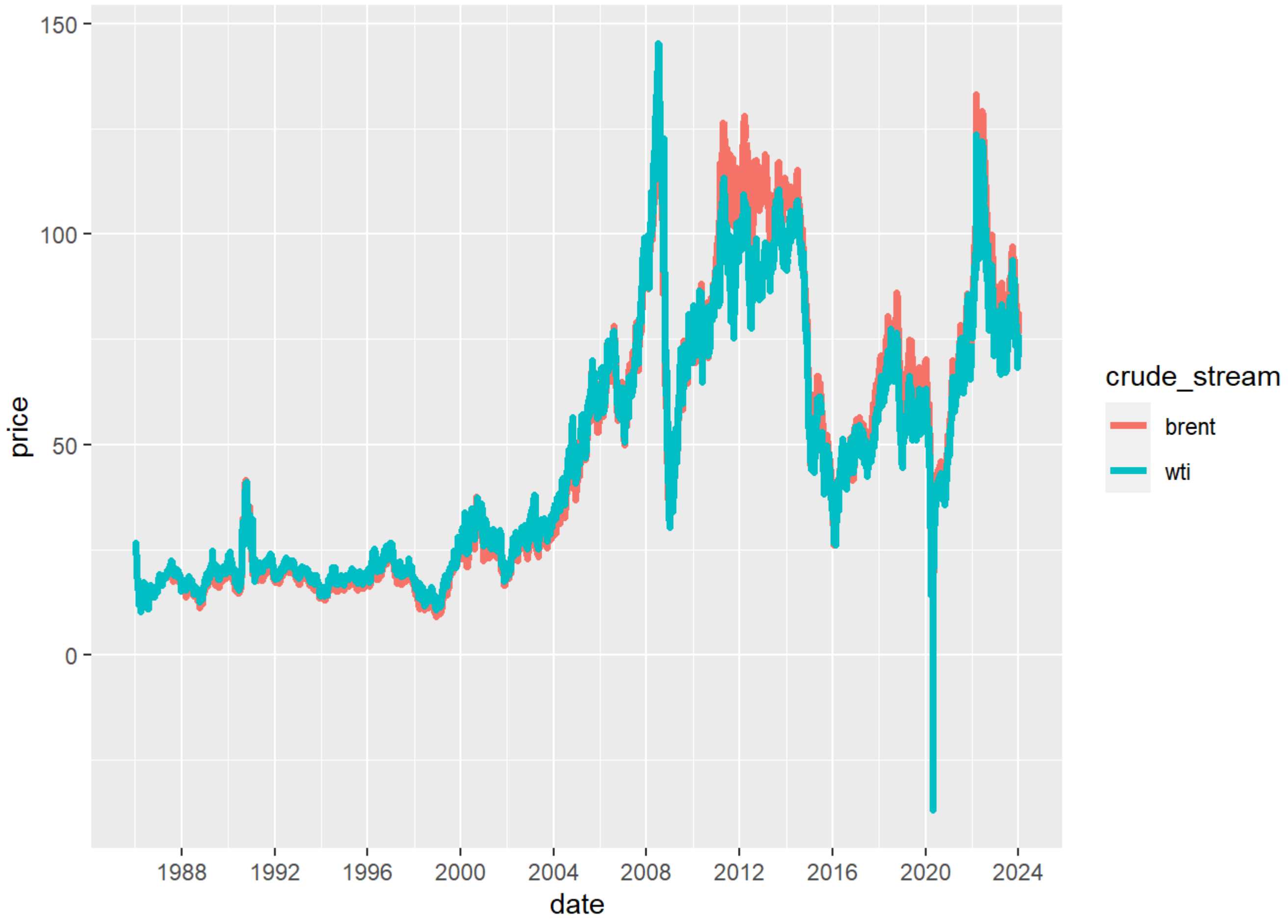
The output tells you that you're dealing with a `POSIXct` or `POSIXt` date-time class variable - this class allows you to store a lot of information including, as in your case, just a YMD date. So, since we're dealing with date-time classes, we would need to modify our graph using the `scale_x_datetime` command. Or, we can convert it to just a date, which I think it easier since we won't use time for this.

```
#notice here I am over-writing my price_data with the altered data set
price_data<-price_data %>% mutate(date=as_date(date))

describe_class(price_data$date)
```

```
[1] "Date"
```

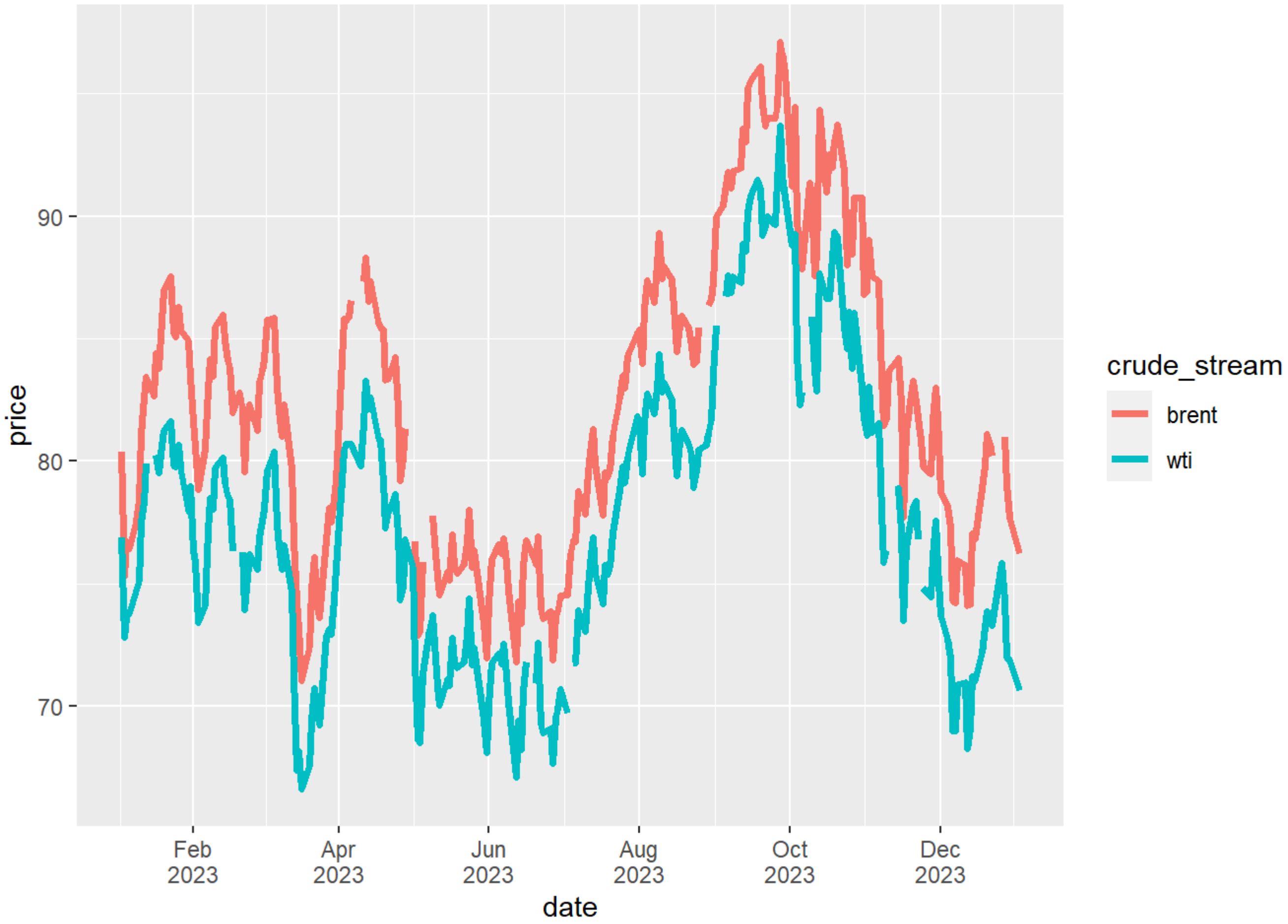
```
#notice here I am just sending price_data through to the graph command, not altering price_data at all
  price_data %>%
    ggplot()+ #make a graph
  geom_line(aes(date,price,group=crude_stream,color=crude_stream),linewidth=1.25)+
  #the grouping tells ggplot that each different crude stream is a different line
  # the colour tells ggplot to assign a different colour to each of the crude streams
  scale_x_date(date_labels = "%Y",date_breaks = "4 years")
```



In this command, `date_labels` tells you how to label your dates and/or times while `date_breaks` tells you the frequency of your labels.

For example, if I wanted to graph crude prices over the last year with axis labels by month, I could do something like this:

```
  price_data %>% filter(date>=max(date)-years(1))%>% #keep the last year of data
    ggplot()+ #make a graph
  geom_line(aes(date,price,group=crude_stream,color=crude_stream),linewidth=1.25)+
#the grouping tells ggplot that each different crude stream is a different line
# the colour tells ggplot to assign a different colour to each of the crude streams
    scale_x_date(date_labels = "%b\n%Y",date_breaks = "2 months")+
# use labels every 2 months, with labels being short-month and 4-digit year on two lines
  NULL #just a placeholder
```

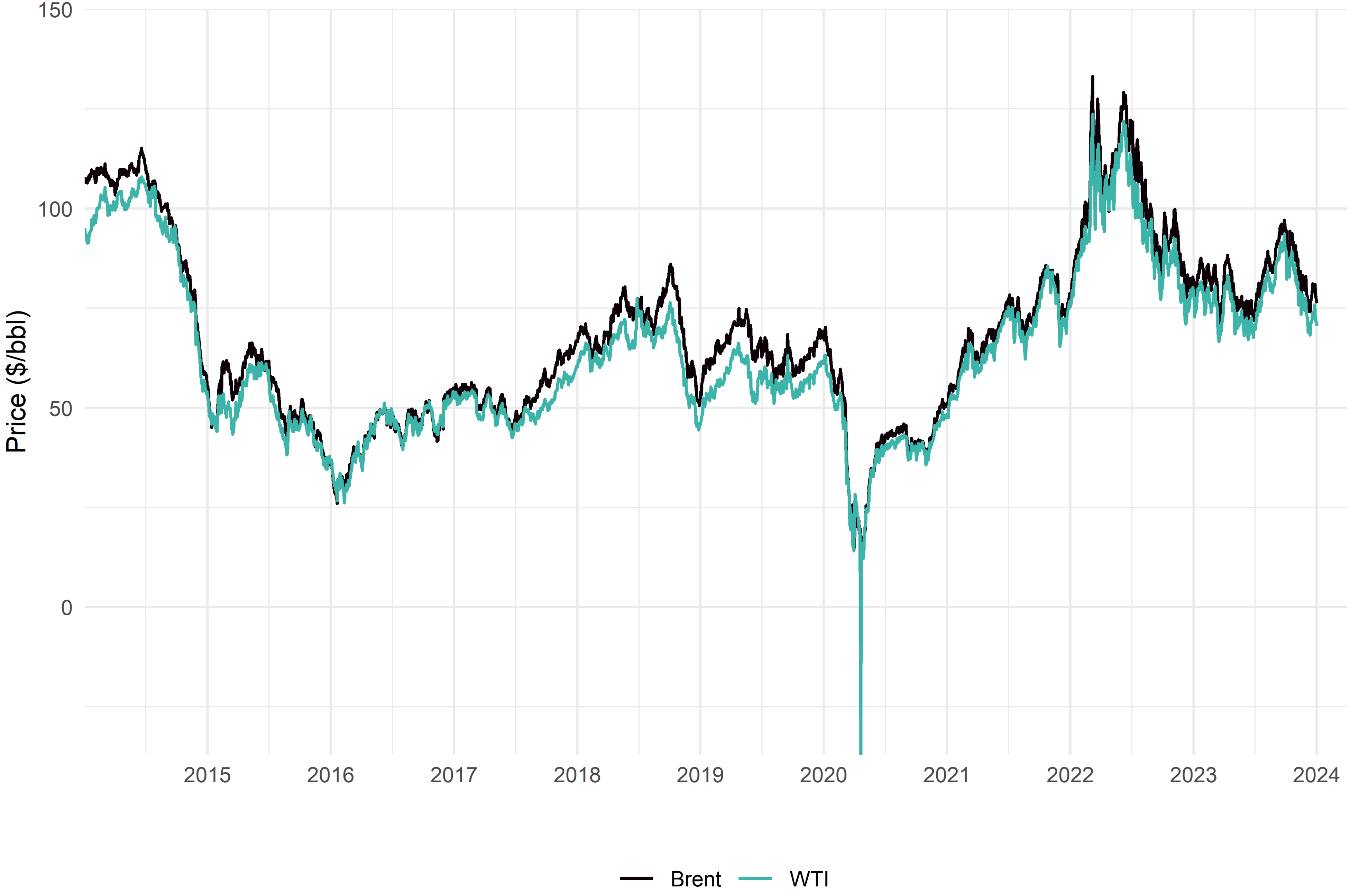


The graph is still ugly, but you get the point.

The different codes you might use to format your dates (the `%Y` and `%b`) are all listed here and the `\n` is a new line command.

Okay, now I'd like to see you try to fix a graph up using some of the code that we've used before, and create a graph showing the last 10 years of WTI and Brent crude oil prices.

Here's my graph for this week:



Crude Oil Spot Prices

Daily West Texas Intermediate (WTI) and Brent prices in dollars per barrel (2014-2024)

Data via EIA, graph by Andrew Leach

I'll give you a couple of hints for how to make this happen:

- I used a colour-blind friendly palette from the `viridis` package (you'd need to install it to replicate this exactly) using the code `scale_color_viridis("",discrete = T,option="mako",direction = -1,begin=.7,end = 0)`
- I stretched the y limits using `expand_limits(y=c(0,150))`
- I expanded the x-axis right limit with `expand_limits(x=ymd("2024-03-31")))` . In this one, I take my text date, `2024-03-31` and convert it to a date object to match the dates in our data using `ymd` from the `lubridate` package.
- I used `theme_minimal()`
- I filtered out any `NA` data points so that I don't get a choppy line.

Good luck!

Made with R and Quarto
View the source at GitHub