Answers for EES 3310/5310 Lab #2

Exercises in Data Manipulation

Jonathan Gilligan

Lab: Mon. Sept. 3. Due: Mon. Sept. 10.

Instructions

Worked Example

Downloading CO_2 Data from Mauna Loa Observatory

In 1957, Charles David Keeling established a permanent observatory on Mauna Loa, Hawaii to make continuous measurements of atmospheric carbon dioxide. The observatory has been running ever since, and has the longest record of direct measurements of atmospheric carbon dioxide levels. The location was chosen because the winds blow over thousands of miles of open ocean before reaching Mauna Loa, and this means the CO_2 measurements are very pure and uncontaminated by any local sources of pollution.

We can download the data from http://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/in_situ_co2/monthly/monthly_in_situ_co2_mlo.csv. We can download the file and save it to the local computer using the R function `download.file`

```
mlo_url = "http://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/in_situ_co2/month
download.file(mlo_url, file.path(data_dir, "mlo_data.csv"))
```

Try opening the data file in Excel or a text editor.

The first 54 lines of the data file are comments describing the data. These comments describe the contents of each column of data and explain that this data file uses the special value `-99.99` to indicate a missing value. The Mauna Loa Observatory started recording data in March 1958, so the monthly averages for January and February are missing. Other months are missing for some months in the record when the instruments that monitor CO_2 concentrations were not working properly.

The `read_csv` function from the `tidyverse` package can read the data into R and convert it into a `tibble` data structure (like a fancy data table). However, when R reads in `.csv` files, it expects column names to be on a single row, and lines 55–57 of the data file are column headings that are split across multiple rows, so R will get confused if we tell it to use those rows as column names.

Thus, we will tell `read_csv` to read this data file, but skip the first 57 lines. We will also tell it not to look for column names in the data file, so we will supply the column names, and we will tell it

that whenever it sees `-99.99`, it should interpret that as indicating a missing value, rather than a measurement.

Finally, R can guess what kinds of data each column contains, but for this file, things work a bit more smoothly if we provide this information explicitly.

`read_csv` lets us specify the data type for each column by providing a string with one letter for each column. The letters are `i` for integer numbers, `d` for real numbers (i.e., numbers with a decimal point and fractional parts), `n` for an unspecified number, `c` for character (text) data, `l` for logical (`TRUE` or `FALSE`), `D` for calendar dates, `t` for time of day, and `T` for combined date and time.

```
mlo_data = read_csv(file.path(data_dir, "mlo_data.csv"),
                    skip = 57,  # skip the first 57 rows
                    col_names = c("year", "month", "date.excel", "date",
                                  "co2.raw", "co2.raw.seas",
                                  "co2.fit", "co2.fit.seas",
                                  "co2.filled", "co2.filled.seas"),
                     col_types = "iiiddddddd", # the first three columns are integers
                                               # and the next 7 are real numbers
                     na = "-99.99" # interpret -99.99 as a missing value
                    )
```

Let's look at the first few rows of the data:

Here is how it looks in R:

```
head(mlo_data)
```

```
## # A tibble: 6 x 10
##    year month date.excel  date co2.raw co2.raw.seas co2.fit co2.fit.seas
##   <int> <int>      <int> <dbl>   <dbl>        <dbl>   <dbl>        <dbl>
## 1  1958     1      21200 1958.     NA           NA      NA           NA
## 2  1958     2      21231 1958.     NA           NA      NA           NA
## 3  1958     3      21259 1958.    316.         314.    316.         315.
## 4  1958     4      21290 1958.    317.         315.    317.         315.
## 5  1958     5      21320 1958.    318.         315.    318.         315.
## 6  1958     6      21351 1958.     NA           NA     317.         315.
## # ... with 2 more variables: co2.filled <dbl>, co2.filled.seas <dbl>
```

And here is how we can use the `kable` function to format the data nicely as a table in an RMarkdown document:

```
head(mlo_data) %>% knitr::kable()
```

year	month	date.excel	date	co2.raw	co2.raw.seas	co2.fit	co2.fit.seas	co2.filled	co2.filled.se
1958	1	21200	1958.041	NA	NA	NA	NA	NA	N
1958	2	21231	1958.126	NA	NA	NA	NA	NA	N
1958	3	21259	1958.203	315.70	314.43	316.18	314.90	315.70	314.
1958	4	21290	1958.288	317.45	315.16	317.29	314.98	317.45	315.

year	month	date.excel	date	co2.raw	co2.raw.seas	co2.fit	co2.fit.seas	co2.filled	co2.filled.se
1958	5	21320	1958.370	317.51	314.72	317.84	315.06	317.51	314.
1958	6	21351	1958.455	NA	NA	317.23	315.14	317.23	315.

There are six different columns for the CO_2 measurements:

- `co2.raw` is the raw measurement from the instrument. The measurements began in March 1958, so there are `NA` values for January and February. In addition, there are missing values for some months when the instrument was not working well.
- `co2.fit` is a smoothed version of the data, which we will not use in this lab.
- `co2.filled` is the same as `co2.raw`, except that where there are missing values in the middle of the data, they have been filled in with interpolated estimates based on measurements before and after the gap.

For each of these three data series, there is also a *seasonally adjusted* version, which attempts to remove the effects of seasonal variation in order to make it easier to observe the trends.

For this lab, we will focus on the `co2.filled` data series. To keep things simple, we can use the `select` function from `tidyverse` to keep only certain columns in the tibble and get rid of the ones we don't want.

```
mlo_simple = mlo_data %>% select(year, month, date, co2 = co2.filled)

head(mlo_simple)
```

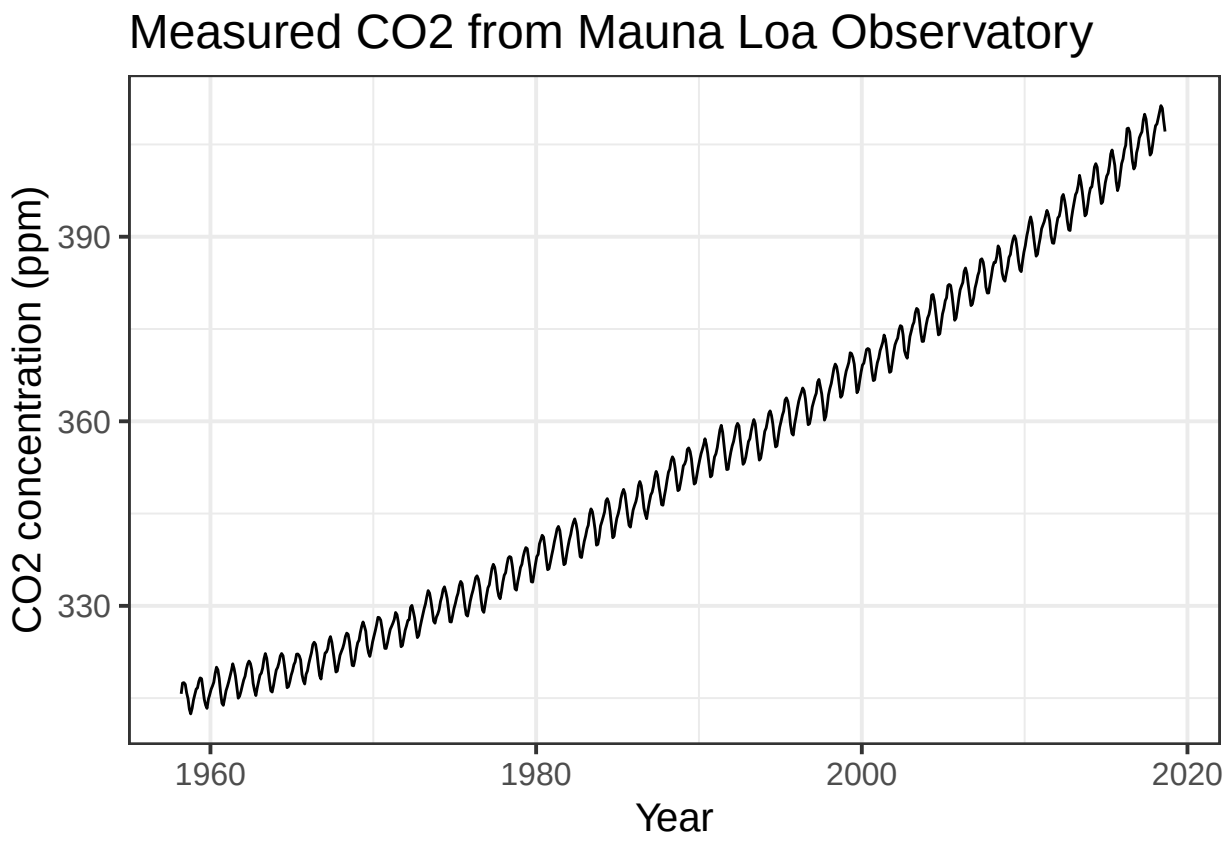
```
## # A tibble: 6 x 4
##    year month  date   co2
##   <int> <int> <dbl> <dbl>
## 1  1958     1 1958.   NA
## 2  1958     2 1958.   NA
## 3  1958     3 1958.  316.
## 4  1958     4 1958.  317.
## 5  1958     5 1958.  318.
## 6  1958     6 1958.  317.
```

Note how we renamed the `co2.filled` column to just plain `co2` in the select function.

Now, let's plot this data:

```
ggplot(mlo_simple,
       aes(x = date, y = co2)) + # This line specifies the data to plot and
                                 # the aesthetics that define which variables to
                                 # use for the x and y axes
  geom_line() +   # This line says to plot lines between the points
  labs(x = "Year", y = "CO2 concentration (ppm)",
       title = "Measured CO2 from Mauna Loa Observatory") # This line gives the
```

```
## Warning: Removed 6 rows containing missing values (geom_path).
```



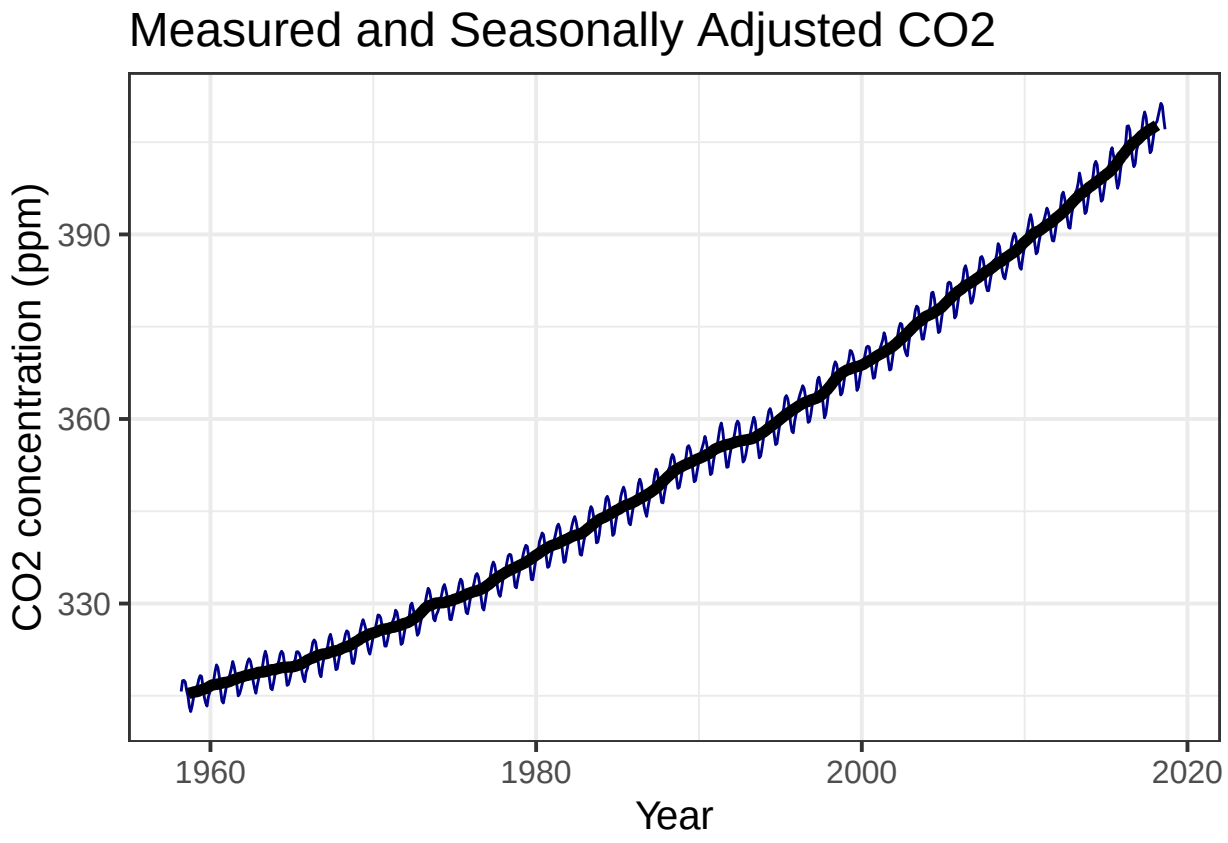

```
                                                    # names of the axes
  # Earlier in this .Rmd file, I called set_theme(theme_bw(base_size = 15)) to
  # set the default plot style. If you call ggplot() without this,
  # you will get a different style, but you can either call theme_set
  # or you can add a theme specification (such as "+ theme_bw(base_size = 15)")
  # end of the sequence of plotting commands in order to apply a specific style
  # to an individual plot.
```

Notice the seasonal variation in CO_2. Every year, there is a large cycle of CO_2, but underneath is a gradual and steady increase from year to year. If we wanted to look at the trend without the seasonal variation, we could use the `co2.filled.seas` column of the original tibble, but instead, let's look at how we might estimate this ourselves.

The seasonal cycle is 12 months long and it repeats every year. This means that if we average the values in our table over a whole year, this cycle should average out. We can do this by creating a new column `trend` where every row represents the average over a year centered at that row (technically, all the months from 5 months before through six months after that date):

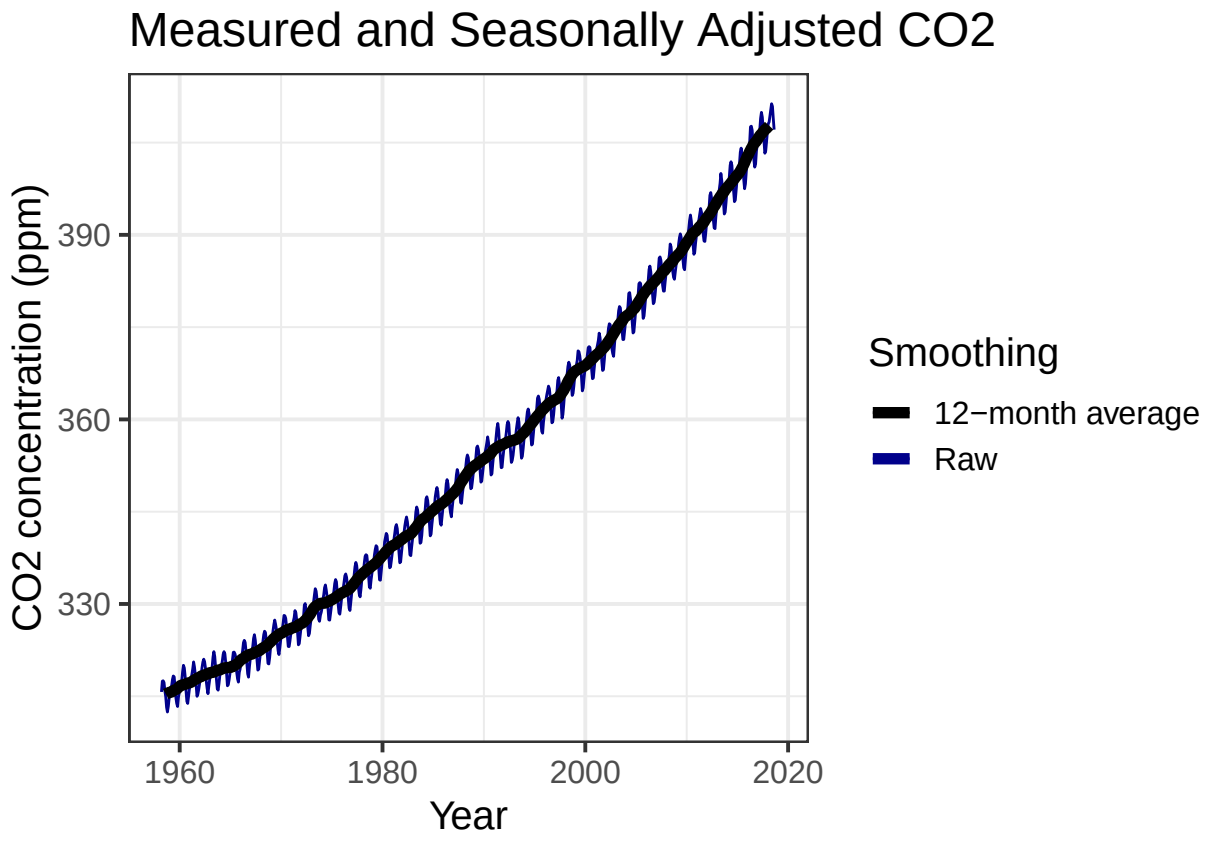
```
mlo_simple %>% mutate(trend = rollapply(data = co2, width = 12, FUN = mean,
                                        fill = NA, align = "center")) %>%
  ggplot(aes(x = date)) +
```

```
  geom_line(aes(y = co2), color = "dark blue") +
  geom_line(aes(y = trend), color = "black", size = 2) +
  labs(x = "Year", y = "CO2 concentration (ppm)",
       title = "Measured and Seasonally Adjusted CO2")
```



But wait: we might want a legend to tell the reader what each colored line represents. We can create new aesthetics for the graph mapping to do this:

```
mlo_simple %>% mutate(trend = rollapply(data = co2, width = 12, FUN = mean,
                                        fill = NA, align = "center")) %>%
  ggplot(aes(x = date)) +
  geom_line(aes(y = co2, color = "Raw")) +
  geom_line(aes(y = trend, color = "12-month average"), size = 2) +
  scale_color_manual(values = c("Raw" = "dark blue", "12-month average" = "black"),
                     name = "Smoothing") +
  labs(x = "Year", y = "CO2 concentration (ppm)",
       title = "Measured and Seasonally Adjusted CO2")
```



We can also anlyze this data to estimate the average trend in CO_2. We use the `lm` function in R to fit a straight line to the data, and we use the `tidy` function from the `broom` package to print the results of the fit nicely.

R has many powerful functions to fit data, but here we will just use a very simple one. We specify the linear relationship to fit using R's formula language. If we want to tell R that we think `co2` is related to `date` by the linear relationship $co2 = a + b \times \text{date}$, then we write the formula `co2 ~ date`. The intercept is implicit, so we don't have to spell it out.

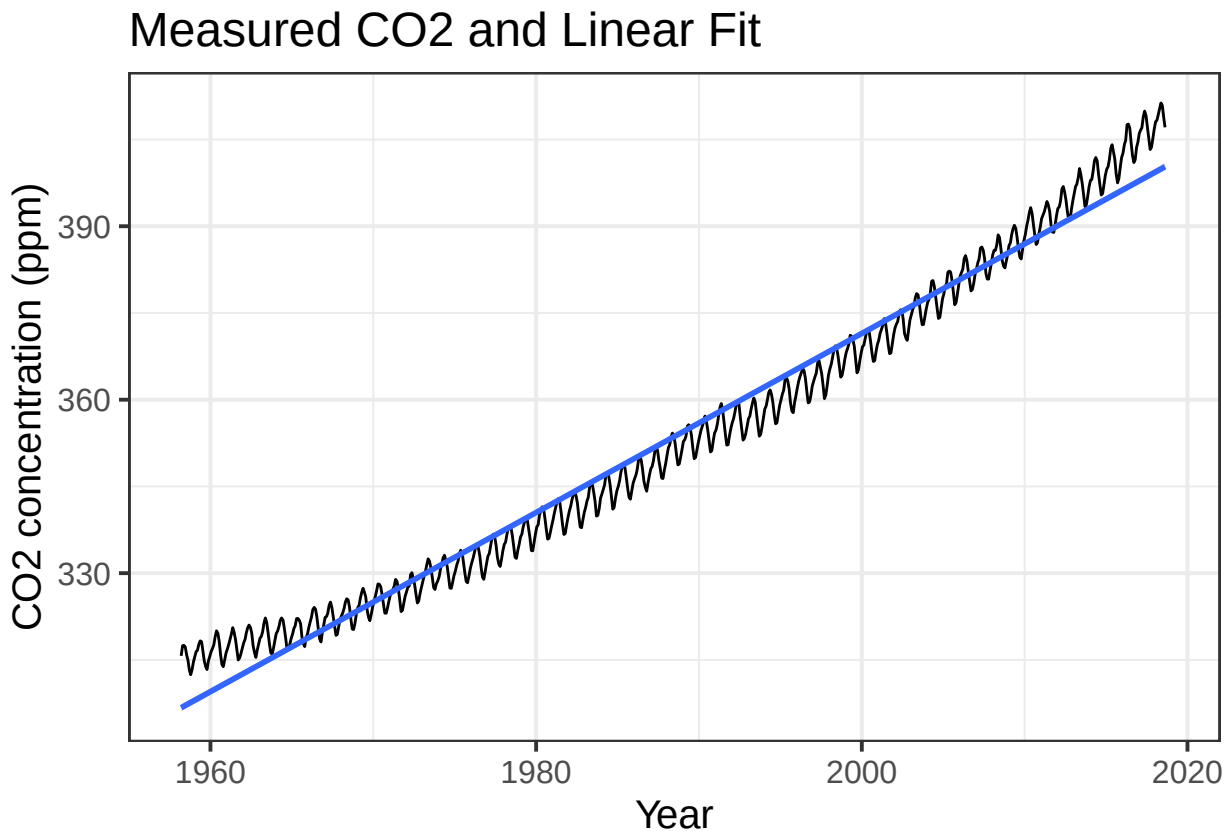
```
trend = lm(co2 ~ date, data = mlo_simple)

library(broom)

tidy(trend)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept) -2726.    17.6         -155.        0
## 2 date            1.55   0.00884      175.        0
```

This shows us that the trend is for CO_2 to rise by 1.55 ppm per year, with an uncertainty of plus or minus 0.009.

We can also plot a linear trend together with the data:

```
mlo_simple %>% mutate(trend = rollapply(data = co2, width = 12, FUN = mean,
                                        fill = NA, align = "center")) %>%
  ggplot(aes(x = date, y = co2)) +
  geom_line() +
  geom_smooth(method = "lm") +
  labs(x = "Year", y = "CO2 concentration (ppm)",
       title = "Measured CO2 and Linear Fit")
```



Exercises

Exercises with CO_2 Data from the Mauna Loa Observatory

Using the `select` function, make a new data tibble called `mlo_seas`, from the original `mlo_data`, which only has two columns: `date` and `co2.seas`, where `co2.seas` is a renamed version of `co2.filled.seas` from the original tibble.

```
# We only need to load the libraries once and they will be loaded for all
# subsequent code chunks
```

```r
library(tidyverse)
library(zoo)

mlo_seas = select(mlo_data, date, co2.filled.seas)
mlo_seas = rename(mlo_seas, co2.seas = co2.filled.seas)

# Alternately, you can simplify with the pipe operator:
#
# mlo_seas = select(mlo_data, date, co2.filled.seas) %>% rename(co2.seas = co2.filled.se
#
# or you can rename as part of the select operation:
#
# mlo_seas = select(mlo_data, date, co2.seas = co2.filled.seas)

# Display the first few rows:
head(mlo_seas)
```

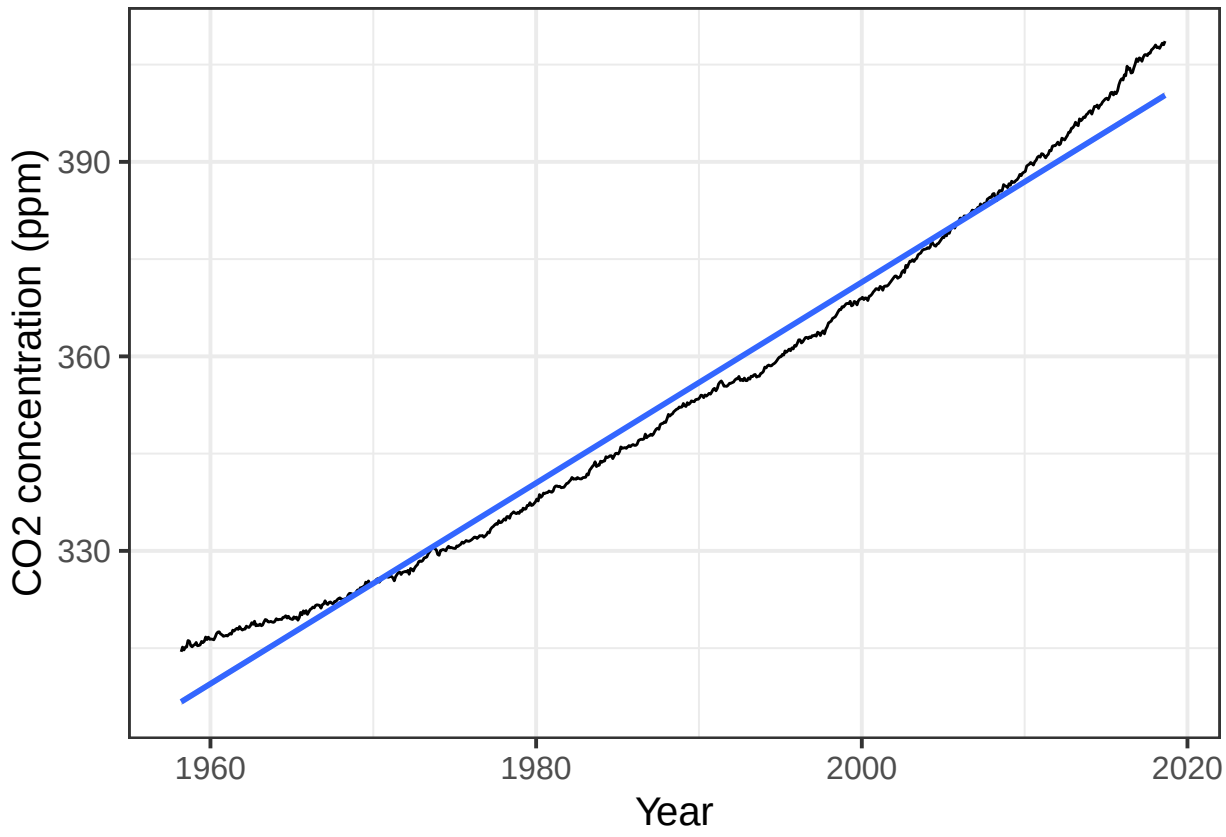
```
## # A tibble: 6 x 2
##    date co2.seas
##   <dbl>    <dbl>
## 1 1958.      NA
## 2 1958.      NA
## 3 1958.     314.
## 4 1958.     315.
## 5 1958.     315.
## 6 1958.     315.
```

Now plot this with `co2.seas` on the *y* axis and `date` on the *x* axis, and a linear fit:

```r
ggplot(mlo_seas, aes(x = date, y = co2.seas)) +
  geom_line() +
  geom_smooth(method="lm") +
  labs(x = "Year", y = "CO2 concentration (ppm)")
```

```
## Warning: Removed 6 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 6 rows containing missing values (geom_path).
```



Now fit a linear function to find the annual trend of `co2.seas`. Save the results of your fit in a variable called `trend.seas`.

```
trend.seas = lm(co2.seas ~ date, data = mlo_seas)

tidy(trend.seas)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept) -2725.     15.1         -181.       0
## 2 date            1.55    0.00758      204.       0
```

Compare the trend you fit to the raw `co2.filled` data to the trend you fit to the seasonally adjusted data.

Note: I just intend students to informally look at the trend in the graph and estimate its slope by eye to compare to the results in `trend.seas`.

Exercises with Global Temperature Data from NASA

We can also download a data set from NASA's Goddard Institute for Space Studies (GISS), which contains the average global temperature from 1880 through the present.

The URL for the data file is https://data.giss.nasa.gov/gistemp/tabledata_v3/GLB.Ts+dSST.csv

Download this file and save it in the directory `_data/global_temp_land_sea.csv`.

```
giss_url = "https://data.giss.nasa.gov/gistemp/tabledata_v3/GLB.Ts+dSST.csv"

download.file(giss_url, file.path(data_dir, "global_temp_land_sea.csv"))
```

- Open the file in Excel or a text editor and look at it.
- Unlike the CO_2 data file, this one has a single line with the data column names, so you can specify `col_names=TRUE` in `read_csv` instead of having to write the column names manually.
- How many lines do you have to tell `read_csv` to skip?

Answer: 1 line: the first line is "Land-Ocean: Global Means" and we want to skip it.

- `read_csv` can automatically figure out the data types for each column, so you don't have to specify `col_types` when you call `read_csv`
- This file uses `***` to indicate missing values instead of `-99.99`, so you will need to specify `na="***"` in `read_csv`.

  For future reference, if you have a file that uses multiple different values to indicate missing values, you can give a vector of values to `na` in `read_csv`: `na = c("***","-99.99", "NA", "")` would tell `read_csv` that if it finds any of the values "***","-99.99","NA", or just a blank with nothing in it, any of those would correspond to a missing value, and should be indicated by `NA` in R.

Now read the file into R, using the `read_csv` function, and assign the resulting tibble to a variable `giss_temp`

```
giss_temp = read_csv(file.path(data_dir, "global_temp_land_sea.csv"), skip = 1, na = "**
                     col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Year = col_integer(),
##   Jan = col_double(),
##   Feb = col_double(),
##   Mar = col_double(),
##   Apr = col_double(),
##   May = col_double(),
##   Jun = col_double(),
##   Jul = col_double(),
```

```
##   Aug = col_double(),
##   Sep = col_double(),
##   Oct = col_double(),
##   Nov = col_double(),
##   Dec = col_double(),
##   `J-D` = col_double(),
##   `D-N` = col_double(),
##   DJF = col_double(),
##   MAM = col_double(),
##   JJA = col_double(),
##   SON = col_double()
## )
```

```
# show the first 5 lines of giss_temp
head(giss_temp, 5)
```

```
## # A tibble: 5 x 19
##    Year    Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct    Nov
##   <int>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1  1880 -0.290 -0.18 -0.11 -0.19 -0.11 -0.23 -0.2  -0.09 -0.15 -0.23 -0.2
## 2  1881 -0.15  -0.17  0.04  0.04  0.02 -0.2  -0.06 -0.02 -0.13 -0.2  -0.21
## 3  1882  0.15   0.15  0.04 -0.18 -0.16 -0.25 -0.2  -0.05 -0.09 -0.24 -0.16
## 4  1883 -0.31  -0.38 -0.12 -0.17 -0.2  -0.12 -0.08 -0.15 -0.2  -0.14 -0.22
## 5  1884 -0.15  -0.08 -0.37 -0.42 -0.36 -0.4  -0.34 -0.26 -0.27 -0.24 -0.290
## # ... with 7 more variables: Dec <dbl>, `J-D` <dbl>, `D-N` <dbl>,
## #   DJF <dbl>, MAM <dbl>, JJA <dbl>, SON <dbl>
```

Something is funny here: Each row corresponds to a year, but there are columns for each month, and some extra columns called "J-D", "D-N", "DJF", "MAM", "JJA", and "SON". These stand for average values for the year from January through December, the year from the previous December through November, and the seasonal averages for Winter (December, January, and February), Spring (March, April, and May), Summer (June, July, and August), and Fall (September, October, and November).

The temperatures are recorded not as the thermometer reading, but as *anomalies*. If we want to compare how temperatures are changing in different seasons and at different parts of the world, raw temperature measurements are hard to work with because summer is hotter than winter and Texas is hotter than Alaska, so it becomes difficult to compare temperatures in August to temperatures in January, or temperatures in Texas to temperatures in Alaska and tell whether there was warming.

To make it easier and more reliable to compare temperatures at different times and places, we define anomalies: The temperature anomaly is the difference between the temperature recorded at a certain location during a certain month and a baseline reference value, which is the average temperature for that month and location over a period that is typically 30 years.

The GISS temperature data uses a baseline reference period of 1951–1980, so for instance, the temperature anomaly for Nashville in July 2017 would be the monthly average temperature measured

in Nashville during July 2017 minus the average of all July temperatures measured in Nashville from 1951–1980.

The GISS temperature data file then averages the temperature anomalies over all the temperature-measuring stations around the world and reports a global average anomaly for every month from January 1880 through the latest measurements available (currently, July 2017).

Let's focus on the months only. Use `select` to select just the columns for "Year" and January through December (if you are selecting a consecutive range of columns between "Foo" and "Bar", you can call `select(Foo:Bar)`). Save the result in a variable called `giss_monthly`

```
giss_monthly = select(giss_temp, Year:Dec)
#
# alternately, you could remove unwanted columns:
#
# giss_monthly = select(giss_temp, -(`J-D`:SON))
#
# You have to use back-quotes for the column `J-D` because its name includes
# characters other than "a"-"z", "A"-"Z", "0"-"9", ".", and "_".
# You can give columns names with other characters than these, but it becomes
# more complicated to indicate them to R.

head(giss_monthly)
```

```
## # A tibble: 6 x 13
##    Year    Jan    Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct
##   <int>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  1880 -0.290 -0.18  -0.11 -0.19 -0.11 -0.23 -0.2  -0.09 -0.15 -0.23
## 2  1881 -0.15  -0.17   0.04  0.04  0.02 -0.2  -0.06 -0.02 -0.13 -0.2
## 3  1882  0.15   0.15   0.04 -0.18 -0.16 -0.25 -0.2  -0.05 -0.09 -0.24
## 4  1883 -0.31  -0.38  -0.12 -0.17 -0.2  -0.12 -0.08 -0.15 -0.2  -0.14
## 5  1884 -0.15  -0.08  -0.37 -0.42 -0.36 -0.4  -0.34 -0.26 -0.27 -0.24
## 6  1885 -0.59  -0.290 -0.25 -0.42 -0.42 -0.44 -0.35 -0.31 -0.23 -0.19
## # ... with 2 more variables: Nov <dbl>, Dec <dbl>
```

Next, it will be difficult to plot all of the data if the months are organized as columns. What we want is to transform the data tibble into one with three columns: "year", "month", and "anomaly". We can do this easily using the `gather` function from the `tidyverse` package: `gather(df, key = month, value = anomaly, -Year)` or `df %>% gather(key = month, value = anomaly, -Year)` will gather all of the columns except `Year` (the minus sign in `select` or `gather` means to include all columns except the ones indicated with a minus sign) and:

- Make a new tibble with three columns: "Year", "month", and "anomaly"
- For each row in the original tibble, make rows in the new tibble for each of the columns "Jan" through "Dec", putting the name of the column in "month" and the anomaly in "anomaly".

Here is an example of using `gather`, using the built-in data set `presidents`, which lists the quarterly approval ratings for U.S. presidents from 1945–1974:

```
df = presidents@.Data %>% matrix(ncol=4, byrow = TRUE) %>%
  as_tibble() %>% set_names(paste0("Q", 1:4)) %>% mutate(year = 1944 + seq(n()))


print("First 10 rows of df are")
```

```
## [1] "First 10 rows of df are"
```

```
print(head(df, 10))
```

```
## # A tibble: 10 x 5
##       Q1    Q2    Q3    Q4  year
##    <dbl> <dbl> <dbl> <dbl> <dbl>
##  1    NA    87    82    75  1945
##  2    63    50    43    32  1946
##  3    35    60    54    55  1947
##  4    36    39    NA    NA  1948
##  5    69    57    57    51  1949
##  6    45    37    46    39  1950
##  7    36    24    32    23  1951
##  8    25    32    NA    32  1952
##  9    59    74    75    60  1953
## 10    71    61    71    57  1954
```

For each year, the table has a column for the year and four columns (Q1 ... Q4) that hold the quarterly approval ratings for the president in that quarter. Now we want to gather these data into three columns: one column for the year, one column to indicate the quarter, and one column to indicate the approval rating.

We do this with the `gather` function from the `tidyverse` package.

"'{ r gather_example} dfg <- df %>% gather(key = quarter, # create a column called "quarter" to store # the names of the columns that are gathered value = approval, # create a column called "approval" to # store the values from those columns # (i.e., the approval ratings in that quarter) -year # the minus sign means gather all columns EXCEPT year.) %>% arrange(year, quarter) # sort the rows of the resulting tibble to put # the years in ascending order, from 1945 to 1971 # and within each year, sort the quarters from Q1 # to Q4

head(dfg) # print the first few rows of the tibble. "'

Now you try to do the same thing to:

- First select just the columns of `giss_monthly` for the year and the individual months.
- Next, gather all the months togeher, so there will be three columns: one for the year, one for the name of the month, and one for the temperature anomaly in that month.
- Store the result in a new variable called `giss_g`

```
giss_g = gather(giss_monthly, key = month, value = anomaly, -Year)
```

Remember how the CO_2 data had a column `date` that had a year plus a fraction that corresponded to the month, so June 1960 was 1960.4548?

Here is a trick that lets us do the same for the `giss_g` data set. R has a data type called `factor` that it uses for managing categorical data, such as male versus female, Democrat versus Republican, and so on. Categorical factors have a textual label, but are silently represented as integer numbers. Normal factors don't have a special order, so R sorts the values alphabetically. However, there is another kind of factor called an ordered factor, which allows us to specify the order of the values.

We can use a built-in R variable called `month.abb`, which is a vector of abbreviations for months.

The following command will convert the `month` column in `giss_g` into an ordered factor that uses the integer values 1, 2, ..., 12 to stand for "Jan", "Feb", ..., "Dec", and then uses those integer values to create a new column, `date` that holds the fractional year, just as the `date` column in `mlo_data` did:

```
giss_g = giss_g %>%
  mutate(month = ordered(month, levels = month.abb),
         date = Year + (as.integer(month) - 0.5) / 12) %>%
  arrange(date)`
```

In the code above, `ordered(month, levels = month.abb)` converts the variable `month` from a character (text) variable that contains the name of the month to an ordered factor that associates a number with each month name, such that "Jan" = 1 and "Dec" = 12.

Then we create a new column called `date` to get the fractional year corresponding to that month. We have to explicitly convert the ordered factor into a number using the function `as.integer()`, and we subtract 0.5 because the time that corresponds to the average temperature for the month is the middle of the month.

Below, use code similar to what I put above to add a new `date` column to `giss_g`.

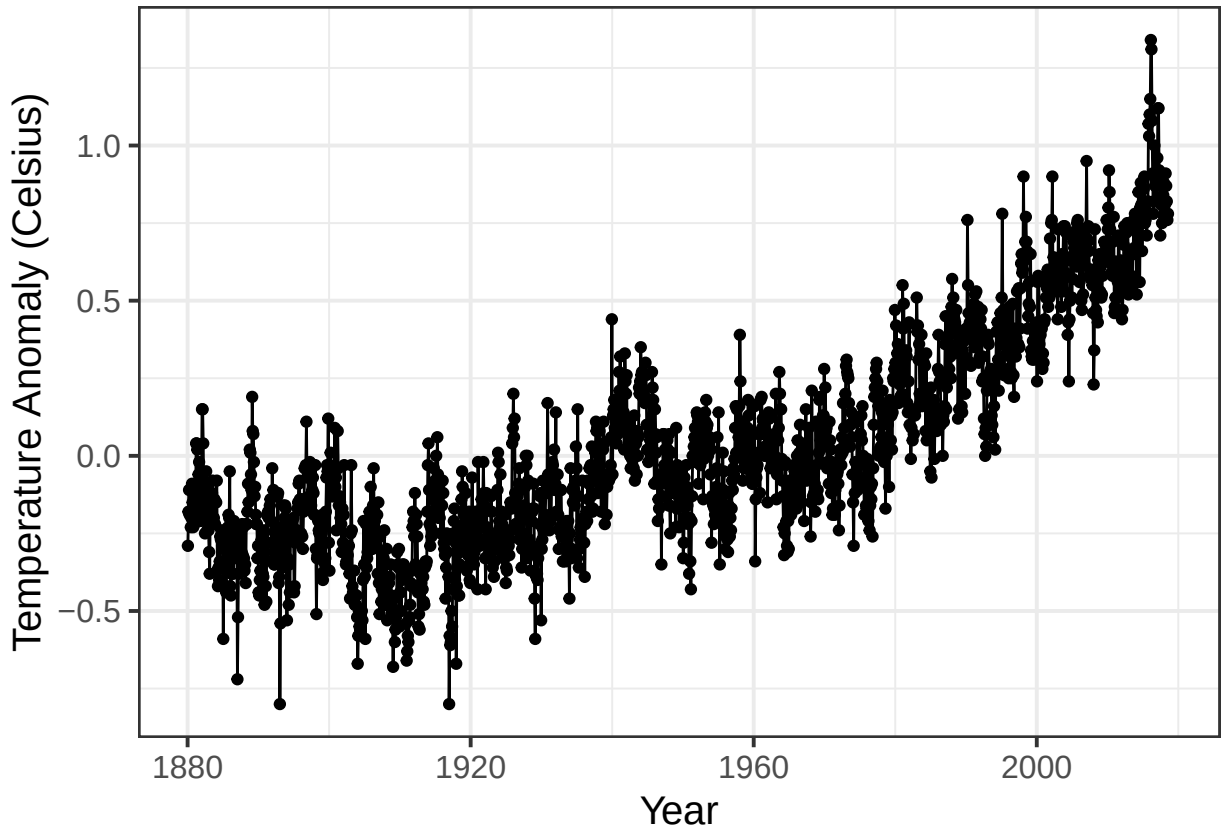
```
# Here, you just copy the code from above and run it.
#
giss_g = giss_g %>%
  mutate(month = ordered(month, levels = month.abb),
         date = Year + (as.integer(month) - 0.5) / 12) %>%
  arrange(date)
```

Now plot the monthly temperature anomalies versus date:

```
ggplot(giss_g, aes(x = date, y = anomaly)) +
  geom_line() +
  geom_point() +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```

```
## Warning: Removed 5 rows containing missing values (geom_path).
```

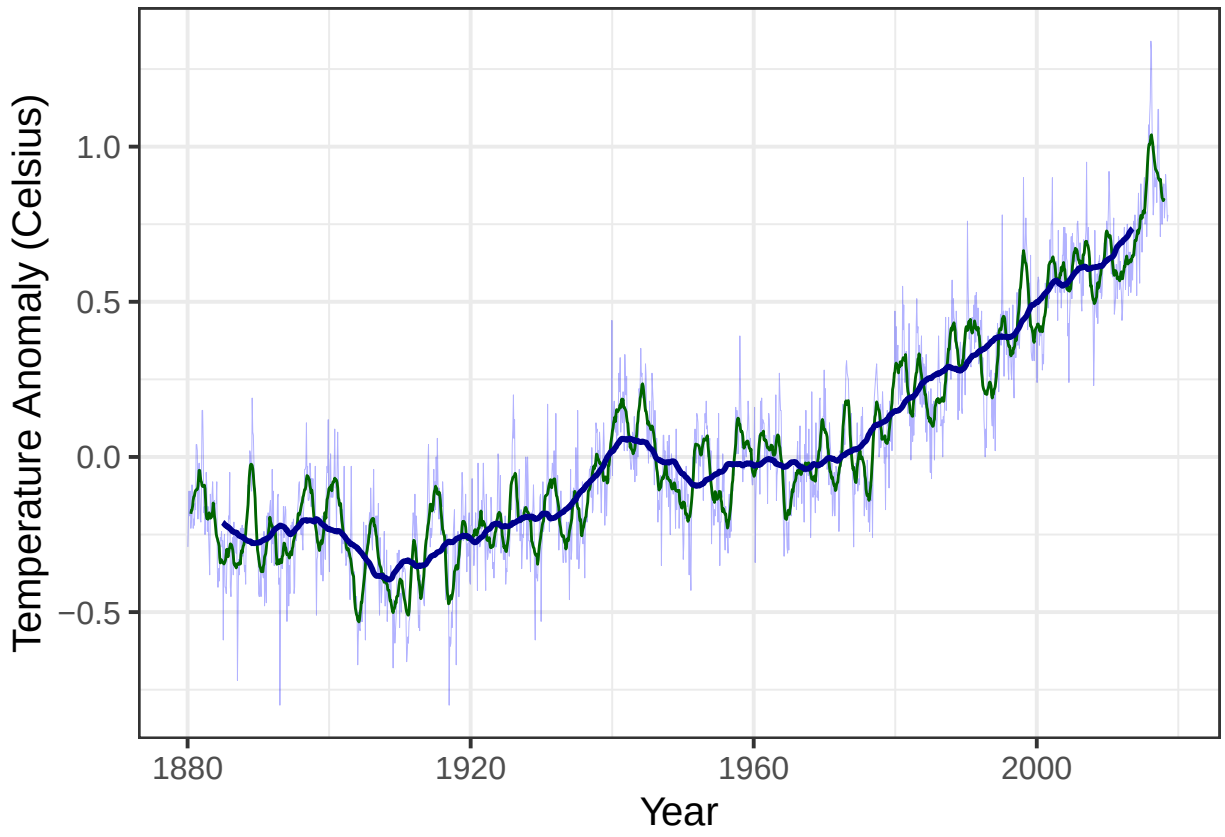
```
## Warning: Removed 5 rows containing missing values (geom_point).
```



That plot probably doesn't look like much, because it's very noisy. Use the function `rollapply` from the package `zoo` to create new columns in `giss_g` with 12-month and 10-year (i.e., 120-month) rolling averages of the anomalies.

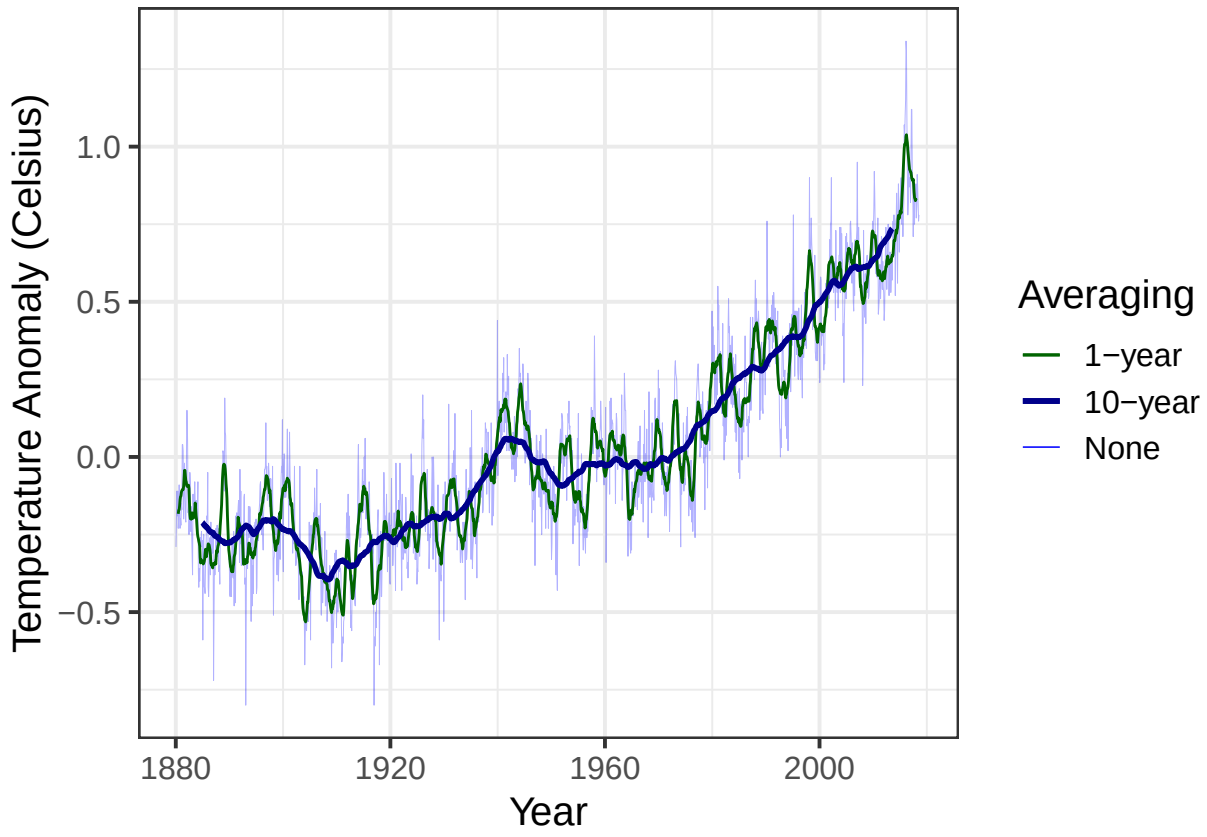
Make a new plot in which you plot a thin blue line for the monthly anomaly (use `geom_line(aes(y = anomaly), color = "blue", alpha = 0.3, size = 0.1)`; alpha is an optional specification for transparency where 0 means invisible (completely transparent) and 1 means opaque), a medium dark green line for the one-year rolling average, and a thick dark blue line for the ten-year rolling average.

```
giss_g %>%
   mutate( smooth.1 = rollapply(data = anomaly, width = 12, FUN = mean,
                                fill = NA, align = "center"),
           smooth.10 = rollapply(data = anomaly, width = 120, FUN = mean,
                                 fill = NA, align = "center")) %>%
  ggplot(aes(x = date)) + # Put code here to map variables to aesthetics
  geom_line(aes(y = anomaly), alpha = 0.3, size = 0.1, color = "blue") +
  geom_line(aes(y = smooth.1), color = "dark green", size = 0.5) +
  geom_line(aes(y = smooth.10), color = "dark blue", size = 1) +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```



Alternately, we could do this fancier version:

```
giss_g %>%
   mutate( smooth.1 = rollapply(data = anomaly, width = 12, FUN = mean,
                                fill = NA, align = "center"),
           smooth.10 = rollapply(data = anomaly, width = 120, FUN = mean,
                                 fill = NA, align = "center")) %>%
  ggplot(aes(x = date)) + # Put code here to map variables to aesthetics
  geom_line(aes(y = anomaly, size = "None", color = "None"), alpha = 0.3) +
  geom_line(aes(y = smooth.1, size = "1-year", color = "1-year")) +
  geom_line(aes(y = smooth.10, size = "10-year", color = "10-year")) +
  scale_color_manual(values = c("None" = "blue", "1-year" = "dark green",
                                "10-year" = "dark blue"), name = "Averaging") +
  scale_size_manual(values = c("None" = 0.1, "1-year" = 0.5,
                               "10-year" = 1.0), name = "Averaging") +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```



The graph shows that temperature didn't show a steady trend until starting around 1970, so we want to isolate the data starting in 1970 and fit a linear trend to it.

To select only rows of a tibble that match a condition, we use the function `filter` from the `tidyverse` package:

`data_subset = df %>% filter( conditions )`, where `df` is your original tibble and `conditions` stands for whatever conditions you want to apply. You can make a simple condition using equalities or inequalities:

- `data_subset = df %>% filter( month == "Jan")` to select all rows where the month is "Jan"
- `data_subset = df %>% filter( month != "Aug")` to select all rows where the month is not August.
- `data_subset = df %>% filter( month %in% c("Sep", "Oct", "Nov")` to select all rows where the month is one of "Sep", "Oct", or "Nov".
- `data_subset = df %>% filter(year >= 1945)` to select all rows where the year is greater than or equal to 1945.
- `data_subset = df %>% filter(year >= 1951 & year <= 1980 )` to select all rows where the year is between 1951 and 1980, inclusive.

- `data_subset = df %>% filter(year >= 1951 | month == "Mar")` to select all rows where the year is greater than or equal to 1951 or the month is "Mar". this will give all rows from January 1951 onward, plus all rows before 1951 where the month is March.

Below, create a new variable `giss_recent` and assign it a subset of `giss_g` that has all the data from January 1970 through the present. Fit a linear trend to the monthly anomaly and report it.

What is the average change in temperature from one year to the next?

```
giss_recent = filter(giss_g, date >= 1970)

recent_trend = lm(anomaly ~ date, data = giss_recent)

tidy(recent_trend)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) -35.4     0.847        -41.8 2.64e-177
## 2 date          0.0179  0.000425      42.3 2.13e-179
```

Did Global Warming Stop after 1998?

It is a common skeptic talking point that global warming stopped in 1998. In years with strong El Niños, global temperatures tend to be higher and in years with strong La Niñas, global temperatures tend to be lower. We will discuss why later in the semester.

The year 1998 had a particularly strong El Niño, and the year set a record for global temperature that was not exceeded for several years. Indeed, compared to 1998, it might look as though global warming paused for many years.

We will examine whether this apparent pause has scientific validity.

To begin with, we will take the monthly GISS temperature data and convert it to annual average temperatures, so we can deal with discrete years, rather than separate temperatures for each month.

We do this with the `group_by` and `summarize` functions.

We also want to select only recent data, so we arbitrarily say we will look at temperatures starting in 1979, which gives us 19 years before the 1998 El Ni~o.

We don't have a full year of data for 2017, so we want to discard that because we won't get a full year average from it.

If we go back to the original `giss_g` data tibble, run the following code:

```
giss_annual = giss_g %>%
  filter(Year >= 1979 & Year < 2017) %>%
  group_by(Year) %>%
```

```
  summarize(anomaly = mean(anomaly)) %>%
  ungroup() %>%
  mutate(date = Year + 0.5)

head(giss_annual)
```

```
## # A tibble: 6 x 3
##    Year anomaly  date
##   <int>   <dbl> <dbl>
## 1  1979   0.159 1980.
## 2  1980   0.271 1980.
## 3  1981   0.33  1982.
## 4  1982   0.131 1982.
## 5  1983   0.312 1984.
## 6  1984   0.161 1984.
```

This code groups the giss data by the year, so that one group will have January–December 1979, another will have January–December 1980, and so forth.

Then we replace the groups of 12 rows for each year (each row represents one month) with a single row that represents the average of those 12 months.

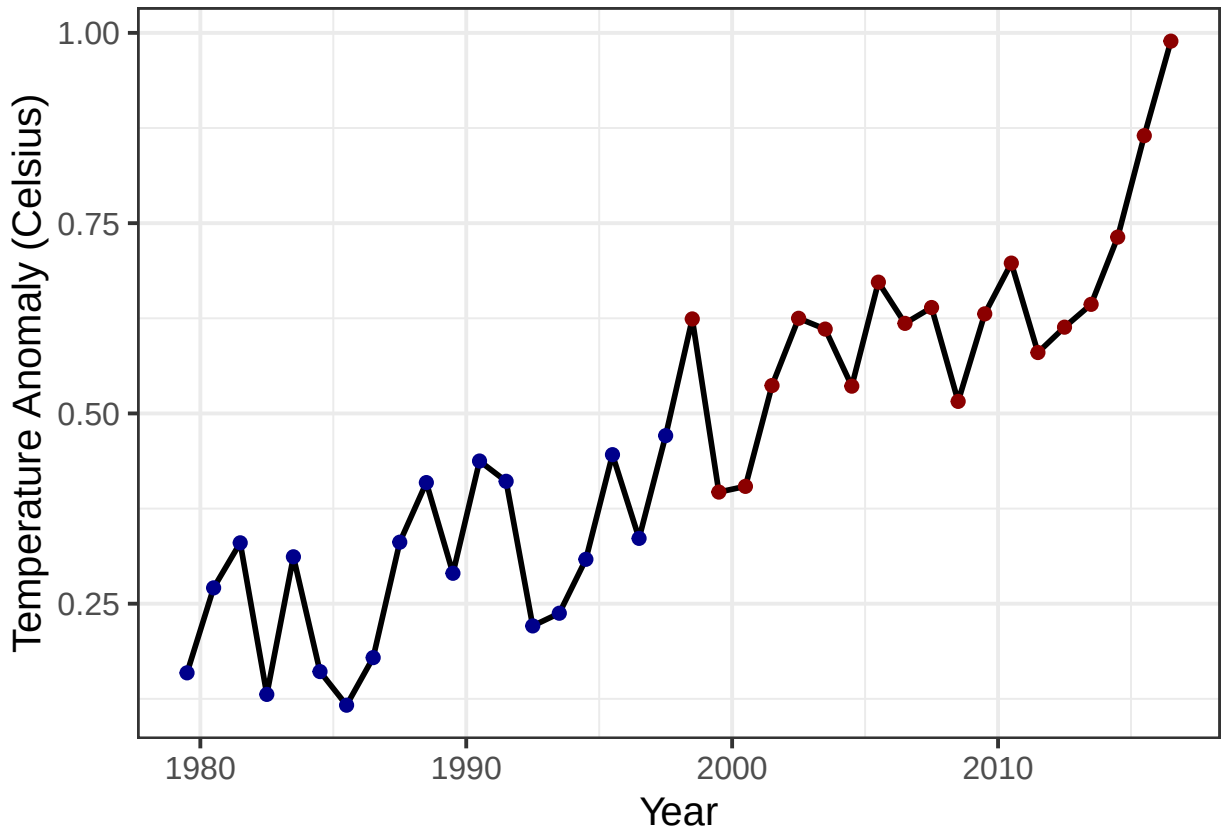
It is important to tell R to `ungroup` the data after we're done working with the groups.

Finally, we set `date` to `year + 0.5` because the average of a year corresponds to the middle of the year, not the beginning.

Now, let's introduce a new column `after`, which indicates whether the data is after the 1998 El Niño:

Now plot the data and color the points for 1998 and afterward dark red to help us compare before and after 1998.

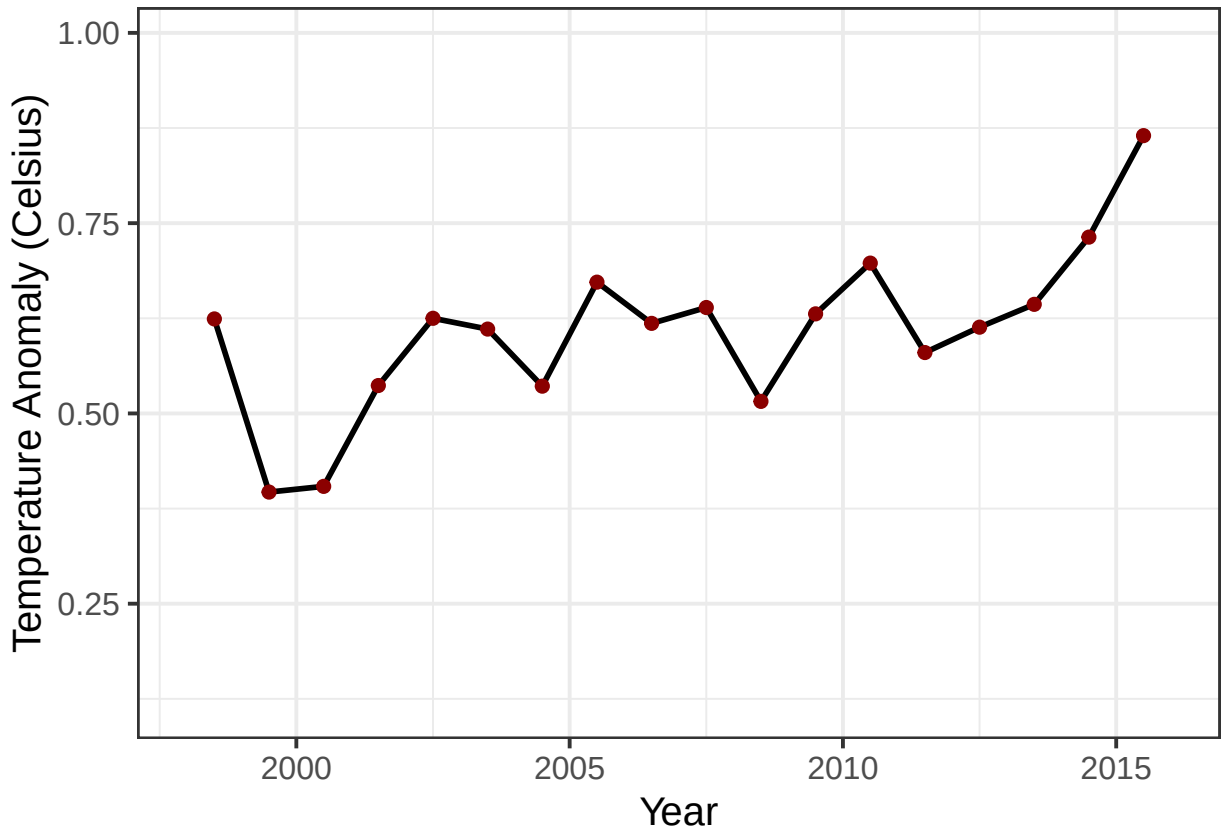
```
ggplot(giss_annual, aes(x = date, y = anomaly)) +
  geom_line(size = 1) +
  # I didn't include it in the original instructions, but the
  # following version of geom_line is nicer than what's above:
  #
  # geom_line(aes(color = Year >= 1998), size = 1) +
  #
  geom_point(aes(color = Year >= 1998), size = 2) +
  scale_color_manual(values = c("TRUE" = "dark red", "FALSE" = "dark blue"),
                     guide = "none") + # color "before" points dark blue,
                                       # "after" points dark red
                                       # don't use a legend
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```



Does it look as though the red points are not rising as fast as the blue points?

Let's just plot the data from 1998 on:

```
ggplot(giss_annual, aes(x = date, y = anomaly)) +
  geom_line(size = 1) +
  # I didn't include it in the original instructions, but the
  # following version of geom_line is nicer than what's above:
  #
  # geom_line(aes(color = Year >= 1998), size = 1) +
  #
  geom_point(aes(color = Year >= 1998), size = 2) +
  scale_color_manual(values = c("TRUE" = "dark red", "FALSE" = "dark blue"),
                     guide = "none") + # color "before" points dark blue,
                                       # "after" points dark red
                                       # don't use a legend
  xlim(1998,2016) +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```



Now how does it look?

Let's use the `filter` function to break the data into two different tibbles: `giss_before` will have the data from 1979–1998 and the other, `giss_after` will have the data from 1998 onward (note that the year 1998 appears in both tibbles).

```
giss_before = filter(giss_annual, Year <= 1998)
giss_after = filter(giss_annual, Year >= 1998)
```

Now use `lm` to fit a linear trend to the temperature data in `giss_before` (from 1979–1998) and assign it to a variable `giss_trend`.

Next, add a column `timing` to each of the split data sets and set the value of this column to "Before" for `giss_before` and "After" for `giss_after`.

```
giss_before = mutate(giss_before, timing = "Before")
giss_after = mutate(giss_after, timing = "After")

giss_trend = lm(anomaly ~ date, data = giss_before)
tidy(giss_trend)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
```

```
## 1 (Intercept) -28.3      7.91        -3.58 0.00213
## 2 date          0.0144   0.00398      3.62 0.00196
```

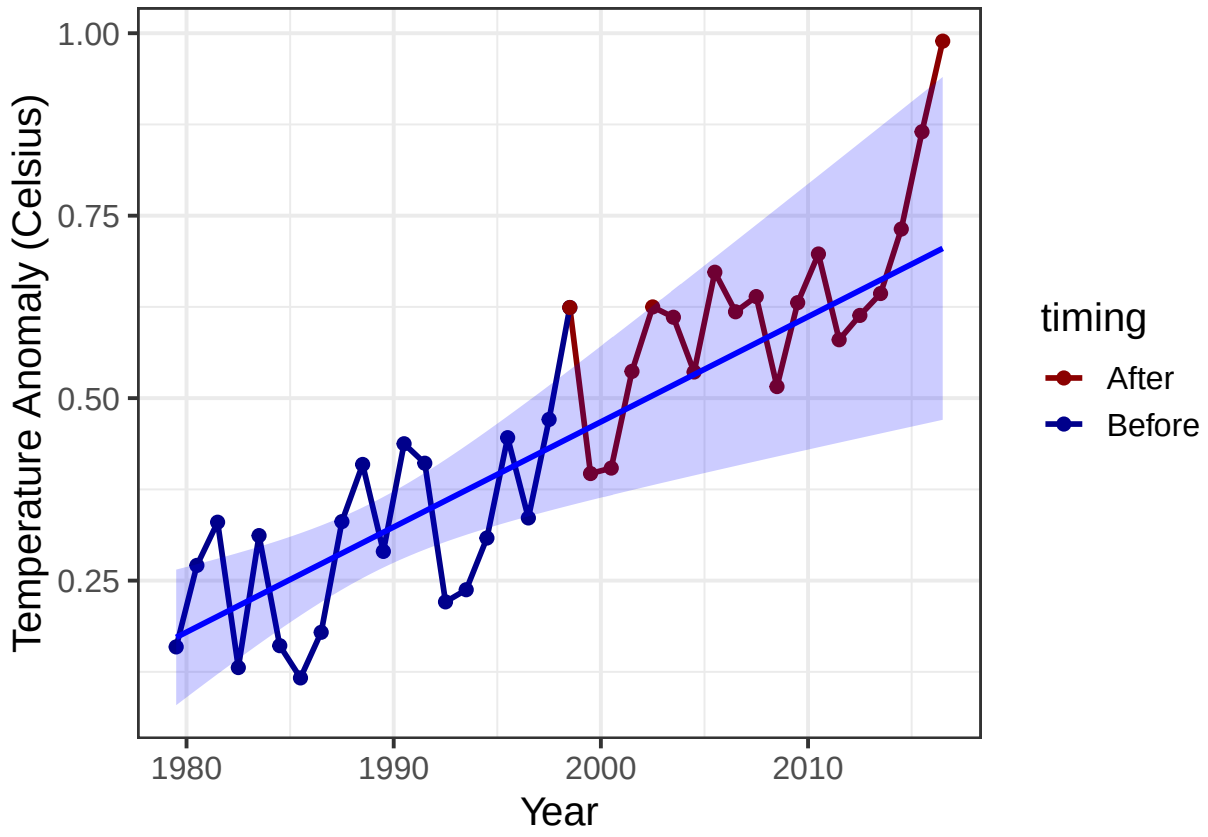
Now, combine the two tibbles into one tibble:

```
giss_combined <- bind_rows(giss_before, giss_after)
```

Now let's use ggplot to plot `giss_combined`:

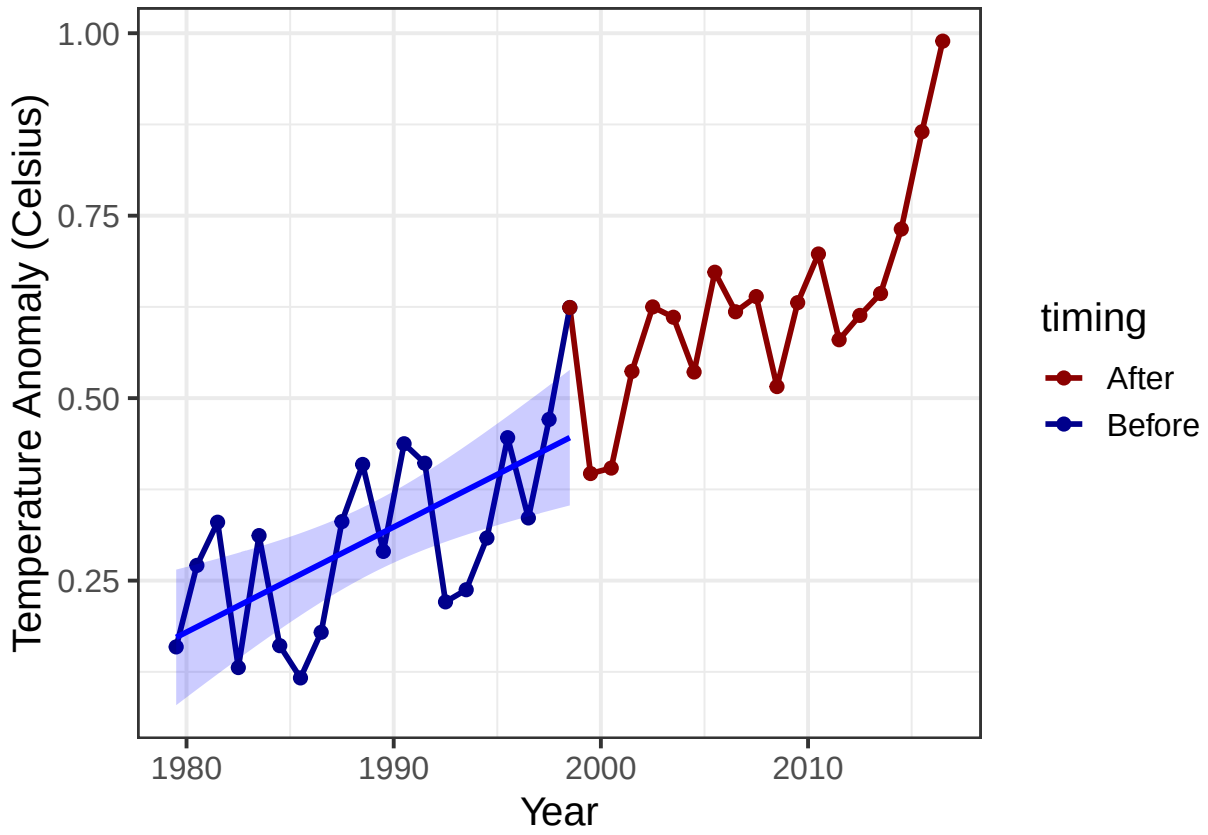
- Aesthetic mapping:
  - Use the `date` column for the *x* variable.
  - Use the `anomaly` column for the *y* variable.
  - Use the `timing` column to set the color of plot elements
- Plot both lines and points.
  - Set the `size` of the lines to 1
  - Set the `size` of the points to 2
- Use the `scale_color_manual` function to set the color of "Before" to "dark blue" and "After" to "dark red"
- Use `geom_smooth(data = giss_before, method="lm", color = "blue", fill = "blue", alpha = 0.2, fullrange = TRUE)` to show a linear trend that is fit just to the `giss_before` data.

```
ggplot(giss_combined, aes(x = date, y = anomaly, color = timing)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  geom_smooth(data = giss_before, method = "lm", color = "blue", fill = "blue",
              alpha = 0.2, fullrange = TRUE) +
  scale_color_manual(values = c(Before = "dark blue", After = "dark red")) +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```

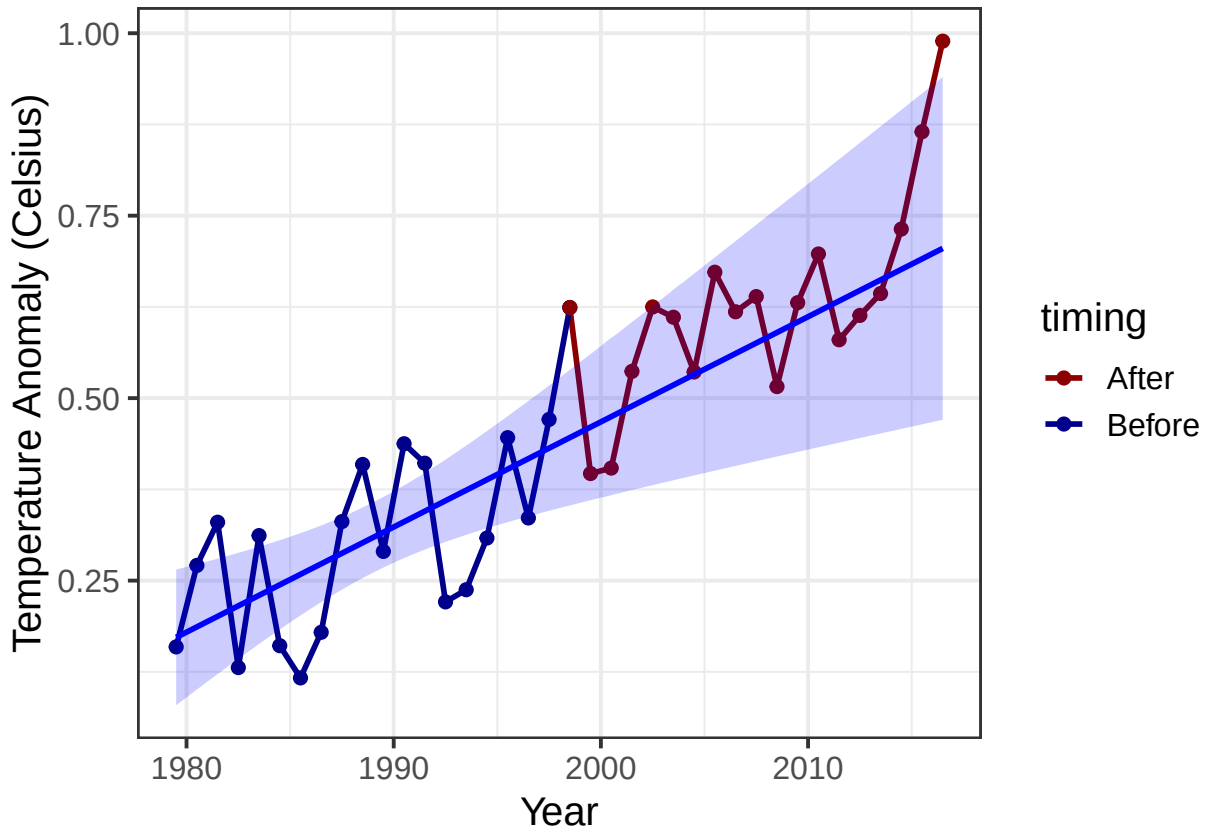


Try this with the parameter `fullrange` set to `TRUE` and `FALSE` in the `geom_smooth` function. What is the difference?

```
ggplot(giss_combined, aes(x = date, y = anomaly, color = timing)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  geom_smooth(data = giss_before, method = "lm", color = "blue", fill = "blue",
              alpha = 0.2, fullrange = FALSE) +
  scale_color_manual(values = c(Before = "dark blue", After = "dark red")) +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```




```
ggplot(giss_combined, aes(x = date, y = anomaly, color = timing)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  geom_smooth(data = giss_before, method = "lm", color = "blue", fill = "blue",
              alpha = 0.2, fullrange = TRUE) +
  scale_color_manual(values = c(Before = "dark blue", After = "dark red")) +
  labs(x = "Year", y = "Temperature Anomaly (Celsius)")
```



Answer: Both plots show the full data set, and a linear trend that is fit just to the "before" data. The trend line shows both the best fit for a trend (that's the solid line) and the range of uncertainty in the fit (that's the light blue shaded area around the line).

But, when `fullrange = FALSE`, the line is only drawn for the data to which the trend was fit, whereas when `fullrange = TRUE`, the trend line is drawn for the full range of the graph, even though the trend was only fit to the data in part of the graph.

If the temperature trend changed after 1998 (e.g., if the warming paused, or if it reversed and started cooling) then we would expect the temperature measurements after 1998 to fall predominantly below the extrapolated trend line, and our confidence that the trend had changed would depend on the number of points that fall below the shaded uncertainty range.

How many of the red points fall below the trend line?

Answer: 6 points: 1999, 2000, 2008, 2011, 2012, and 2013.

How many of the red points fall above the trend line?

Answer: Not counting 1998, 12 points: 2001, 2002, 2003, 2004 (barely), 2005, 2006, 2007, 2009, 2010, 2014, 2015, and 2016.

What do you conclude about whether global warming paused or stopped after 1998?

Answer: Most of the years after 1998 were warmer than we would have predicted if temperatures

had continued to follow the warming trends of 1979–1998, so this is not evidence of any slow-down or pause in the warming.