

EES 3310/5310 Lab #2 Report, Part 1

Exercises in Data Manipulation

put your name here

Lab: Mon. Jan. 31. Due: Wed. Feb. 9

Contents

Instructions	1
Introduction	2
Downloading CO ₂ Data from Mauna Loa Observatory	2
Exercises	3
Exercises with CO ₂ Data from the Mauna Loa Observatory	3
Exercises with Global Temperature Data from NASA	4

Instructions

This lab report picks up from where the instructions and the worked examples left off.

The section “Exercises” has instructions about what to do and places where you will fill in R code, following the instructions, in order to perform these analysis.

To make it easier for you to find the places where you have to fill in R code, I have put the comment `# TODO` at the beginning of every code chunk where you need to fill in some code. You can search for this using RStudio’s search (press Ctrl+F or Cmd+F to open up the search bar, and type the text you want to search for, such as “TODO”, into the box that says “Find”)

If you wish, you may delete this “Instructions” section before you knit and turn in your final version of the report, so the report will begin with the “Introduction” section. If you do, **be sure that you have staged and committed all your files to git before you delete this section**, so that if you accidentally delete too much, you will be able to recover the original material.

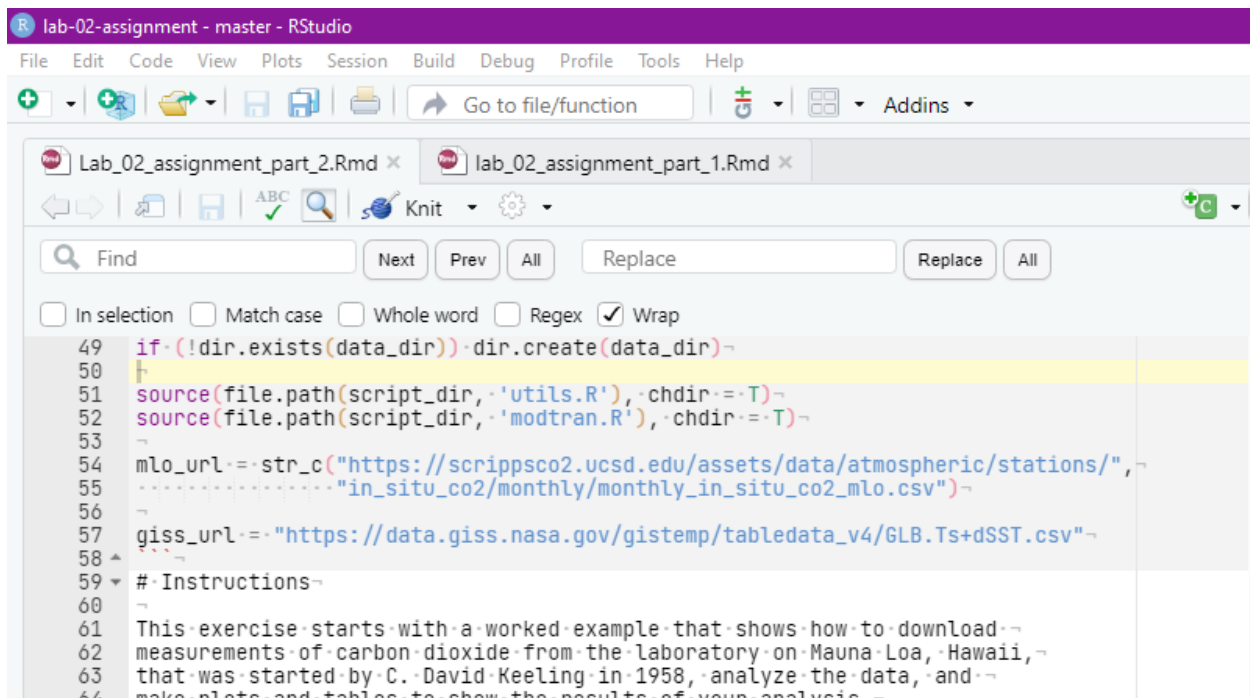


Figure 1: RStudio find bar

Introduction

This exercise uses measurements of carbon dioxide from the laboratory on Mauna Loa, Hawaii, that was started by C. David Keeling in 1958.

Downloading CO₂ 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 CO₂ measurements are very pure and uncontaminated by any local sources of pollution.

We start by downloading the data from https://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/in_situ_co2/monthly/monthly_in_situ_co2_mlo.csv. The code here checks to see if the file already exists, so we don't download it again every time we build this report.

```
if (!file.exists('_data/mlo_data.csv')) {
  download.file(mlo_url, '_data/mlo_data.csv')
}
```

After downloading the data, we process it, following the worked example in the instructions:

```

mlo_data = read_csv('_data/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
) %>% clean_names()

```

```

mlo_simple = mlo_data %>% select(year, month, date, co2 = co2_filled)

```

```

mlo_data_adjusted = mlo_simple %>%
  mutate(co2_annual = slide_vec(co2, mean, .before = 5, .after = 6))

```

```

co2_fit = lm(co2 ~ date, data = mlo_simple)

```

```

library(broom)

```

Exercises

Exercises with CO₂ 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.

```

# TODO
# put your R code here

```

Now plot this with `co2_seas` on the y axis and `date` on the x axis, and a linear fit:

```

# TODO
# put your R code here
# remember to use geom_smooth to include a linear fit.

```

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`.

```
# TODO
# put your R code here to set trend_seas
```

Compare the trend you fit to the raw `co2_filled` data to the trend you fit to the seasonally adjusted data.

Answer: *put your value for the trend here and discuss how it compares to the trend you fit to the seasonally adjusted data.*

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 stored in the variable `giss_url` so you don't have to type it in here.

Download this file and save it in the directory `_data/global_temp_land_sea.csv`. You may want to use `if (! file.exists(...))`, as in the example above of downloading the CO₂ data, to avoid downloading the file again if it already exists in your `_data` directory.

```
# TODO
# Put your R code here
```

- Open the file in Excel or a text editor and look at it.
- Unlike the CO₂ 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?
- `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`

```
# TODO
# Put your R code here to call read_csv and read "global_temp_land_sea.csv"
# and assign the data to a variable giss_temp

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

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 December 2020).

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” in the tibble `df`, you can call `select(df, Foo:Bar)` or `df %>% select(Foo:Bar)`). Save the result in a variable called `giss_monthly`

```
# TODO
# put your R code here
```

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 `pivot_longer` function from the `tidyverse` package. See the worked examples for an example of using `pivot_longer()`.

Here are examples of using `pivot_longer` on a variable `df`: `pivot_longer(df, cols = -Year, names_to = "month", values_to = "anomaly")` or `df %>% pivot_longer(cols = -Year, names_to = "month", values_to = "anomaly")` will gather all of the columns except `Year` (the minus sign in the argument to `cols` 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”.

Now you try to do the same thing to the `giss_monthly` data. Use `pivot_longer` to organize the data to have 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`

```
# TODO
# put your R code here to make the variable giss_g
```

Remember how the `CO2` 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 behind the scenes, R thinks of them 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 = factor(month, levels = month.abb, ordered = TRUE),
         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`.

```
# TODO
# put your R code here
```

Now plot the monthly temperature anomalies versus date:

```
# TODO
# put your R code here
#
# Here in the comments is an example of the kind of thing you might want to
# use, but you will need to fill in some details, such as the data and
# aesthetics for ggplot() and which geometries you want to plot (geom_XXX is not
# a real geometry).
#
# ggplot( ) +
# # specify the data and the mappings of variables to plot aesthetics
# geom_XXX() +
# # put the geometries (lines, points, etc) that you want to plot
# labs() +
# # label the axes
# ... # put any other characteristics here.
```

That plot probably doesn't look like much, because it's very noisy. Use the function `slide_vec` from the package `slider` to create new columns in `giss_g` with 12-month and 10-year (i.e., 120-month) sliding 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 sliding average, and a thick dark blue line for the ten-year sliding average.

```
# TODO
# put your R code here
#
# Here is an example of the outline of the kind of code you might want to
# use, but you will need to fill in the details to make this code work.
#
# giss_g %>%
#   mutate( ... ) %>%
#   # ^^ fill in code for "..." in "mutate()" to add a columns called
#   # "smooth.1" for the one-year smoothed anomaly
#   # and "smooth.10" for the ten-year smoothed anomaly.
#   ggplot(aes( ... )) +
#   # ^^ Then we send the result of mutate to ggplot() where it becomes the
#   # data to plot.
#   # Add code for the aesthetics ("...") to map variables to aesthetics.
```

```
# geom_line(aes(y = anomaly), alpha = 0.3, size = 0.1) +
# # ^^^ plot a thin blue line with the un-smoothed anomaly
# geom_line(...) +
# # ^^^ Now add a medium "darkgreen" line for the one-year smoothed data
# geom_line(...) +
# # ^^^ And a thick "darkblue" line for the ten-year smoothed data
# labs( ...) + # Label your axes
# # ... # add any other plot specifications you need.
```

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?

```
# TODO
# put your R code here
#
# create giss_recent
#
# fit a linear trend to the data using lm()
#
# print the coefficients of the trend.
```

Write some text here giving the answer

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 (see the examples and explanation in the worked examples document).

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 Niño.

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

```
# TODO
# When you are ready to run the code below, you can un-comment it in RStudio
# by deleting the "#" at the beginning of each line.

# giss_annual = giss_g %>%
#   filter(Year >= 1979) %>%
#   group_by(Year) %>%
#   summarize(anomaly = mean(anomaly)) %>%
#   ungroup() %>%
#   mutate(date = Year + 0.5, before = Year < 1998)
```

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 and we introduce a new column `before`, which indicates whether the data is before 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.

```
# TODO
# Here is more example code that you can uncomment and run after you get the
# code in the preceding chunks to run properly.

# ggplot(giss_annual, aes(x = date, y = anomaly)) +
#   geom_line(size = 1) +
#   geom_point(aes(color = before), size = 2) +
#   scale_color_manual(values = c("TRUE" = "darkblue", "FALSE" = "darkred"),
#     guide = "none") +
#   # ^^^ color "before" points dark blue, "after" points dark red.
#   # guide = "none" tells ggplot not to show a legend explaining the colors.
#   labs(x = "Year", y = "Temperature Anomaly")
```

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

Let's just plot the data from the years 1998–2011. Use the `filter` function to select just the date from the years 1998–2011 and pass that to `ggplot`.

```
# TODO
# Put your R code here
# Filter the giss_annual data to select only the years >= 1998
# plot the data
```

Now how does it look?

Let's use the `filter` function to break the data into two different data sets, which we will store in tibbles called `giss_before` and `giss_after`: `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 data sets).

Also, use the `mutate` function to add a column called `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`.

```
# TODO
# Put your R code here
```

Now use `lm` to find the trend in 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`.

```
# TODO
# Put your R code here
```

Next, combine the two data frames (or tibbles) into one, using the `bind_rows` function. If you have created the data frames `giss_before` and `giss_after`, then you can un-comment the code below to combine them.

```
# TODO
# After you have created two data frames or tibbles, giss_before and giss_after,
# then you can un-comment the line of code below and it will run.
# I have commented it because it will cause an error if you knit the document
# before you add code to create those data frames.
#
# 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 “darkblue” and “After” to “darkred”
- 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.

```
# TODO
# Put your R code here.
```

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

What this plot shows is 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).

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: *put your answer here.*

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

Answer: *put your answer here.*

If we just look at the years 1998–2012, how many of the red points fall above vs. below the trend line?

Answer: *put your answer here.*

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

Answer: *put your answer here.*