Lab 3: Plotting Data

Learning Outcomes

1. Put into practice your knowledge of cases and variables
2. Hone your understanding of levels vs. rates
3. Import basic publicly available data into R with date variables
4. Plot time series data as a line graph
5. Shape and change your data so to make it fit for purpose…
6. … allowing you to superimpose time series line graphs
7. Envisioning the skill to develop dynamic graphs like the Federal Reserve uses

Loading the packages

Call on the following packages, using the `install.packages("")` command if necessary to install the packages. Remember to open a code chunk and put these in the code chunk.

```
library(mosaic)
library(tidyverse)
library(ggthemes)
```

Levels of Labor Force Participation

The Federal Reserve Bank, St. Louis has an economic research branch called “FRED”. You can visit the website here: research.stlouisfed.org (https://research.stlouisfed.org). FRED makes a variety of fantastic data available to the public. In Labs 1 and 2, we saw a table that had information about the total number of men and women employed in the United States.

```
##          date  male female
## 1  1948-01-01 43214  16881
## 2  1948-02-01 43400  17124
## 3  1948-03-01 43080  16990
## 4  1948-04-01 43215  17462
## 5  1948-05-01 43002  16970
## 6  1948-06-01 43257  17700
## 7  1948-07-01 43429  17752
## 8  1948-08-01 43403  17403
## 9  1948-09-01 43240  17575
## 10 1948-10-01 43396  17250
```

As we saw in Chapter 2, many economists want to understand the behavioral responses of people to

incentives, for example, what happens to labor supply when people are offered the opportunity to work, such as with women in the second half of the twentieth century. You can see the FRED graph of labor force participation rate at the following link: research.stlouisfed.org/fred2/graph/?id=LNS11300001,LNS11300002,# (https://research.stlouisfed.org/fred2/graph/?id=LNS11300001,LNS11300002,#) (accessed on 2015.07.15.) The FRED graph is interactive, which is more advanced than we need right now. We simply want to look at the data for the moment and get a sense of its structure.

Importing Data

The first thing you need to do is read in the data. To read in comma separated value (csv) data, we need to use one of two commands (which we saw in the previous lab):

1. `read.csv()`
2. `read_csv()`

In the parentheses we can call the url online where the file is or specify the filepath where the file is. the `readr` package using `read_csv()` is very useful for big csv files or for doing it more quickly (along with greater customization).

You should go through the following process:

1. Download the Excel file `labor_force_participation.xls` from the following url:https://drive.google.com/file/d/0B9jjwkjdUJU7R2hXVGtvTEZmLVU/view?usp=sharing (https://drive.google.com/file/d/0B9jjwkjdUJU7R2hXVGtvTEZmLVU/view?usp=sharing)

And then… either

1. Open the file in MS Excel and save the file as a .csv file.
2. Import the data by assigning a name to the object, e.g. `Participation <-read.csv(YOUR FILE PATH HERE)` (notice how, because we are creating a data table, we have started it with an uppercase letter).

Here's an example of how you might read in the .csv file:

```
Participation <- read.csv("../more/labor_force_participation.csv")
```

OR… you can read the `readxl` package (install it using `install.packages("readxl")`)

```
library(readxl)
ParticipationXL <- read_excel("../more/labor_force_participation.xls")
```

Remember: If you have the csv/Excel file *in the same folder* as your R Markdown file, then you don't need to specify the folders `../more` and so on, you could just have:
`ParticipationXL <- read_excel("labor_force_participation.xls")`.

Getting R to see Dates

Remember that R needs to be told a variable is a date. We shall solve it using the function `mutate` as we did previously:

```
Participation <-
  Participation %>%
  mutate(date = as.Date(date))
```

We shall use the data to create a variety of graphs using the command `ggplot`. At the end of the lab, you'll see that you can also create a graph that look startlingly similar to the one created by the Federal Reserve Bank. That'll be more than what you need to learn for now, but it's important to see to understand the power of R.

Generating a Line Graph

Let's use these data to create two graphs, one for each of men and women. You can use many commands to generate graphs in R, but we shall use the `ggplot` command. If you want to search for help, you need to look for "ggplot2" because an old version of `ggplot` existed. Remember to look for help simply put a question mark `?` in front of a command's name, e.g. `?ggplot2`.

To generate a graph in `ggplot` you first need to tell R what data you want to use. We shall do this by putting the data on a line *before* we call on the ggplot command and we shall follow the name of the data with the following symbol: `%>%`. Remember from last week that the `%>%` symbol is called a **pipe** in programming, statistics or econometrics. You should read the pipe as saying "Then" or "And then".

Next we will add in the call to `ggplot` and tell it that the x-axis maps to a date and that the y-axis maps to a measure of the female participation rate.

You will get an error if you try to knit this in a chunk without the option `eval = FALSE` in the curly brackets after a comma after `{r}`, e.g. `{r, eval = FALSE}`. By including this command you are telling RMarkdown *not* to evaluate the command as an actual command, but simply to print it. We're doing it here to add things together step by step.

```
Participation %>%
  ggplot(aes(x = date, y = female))
```

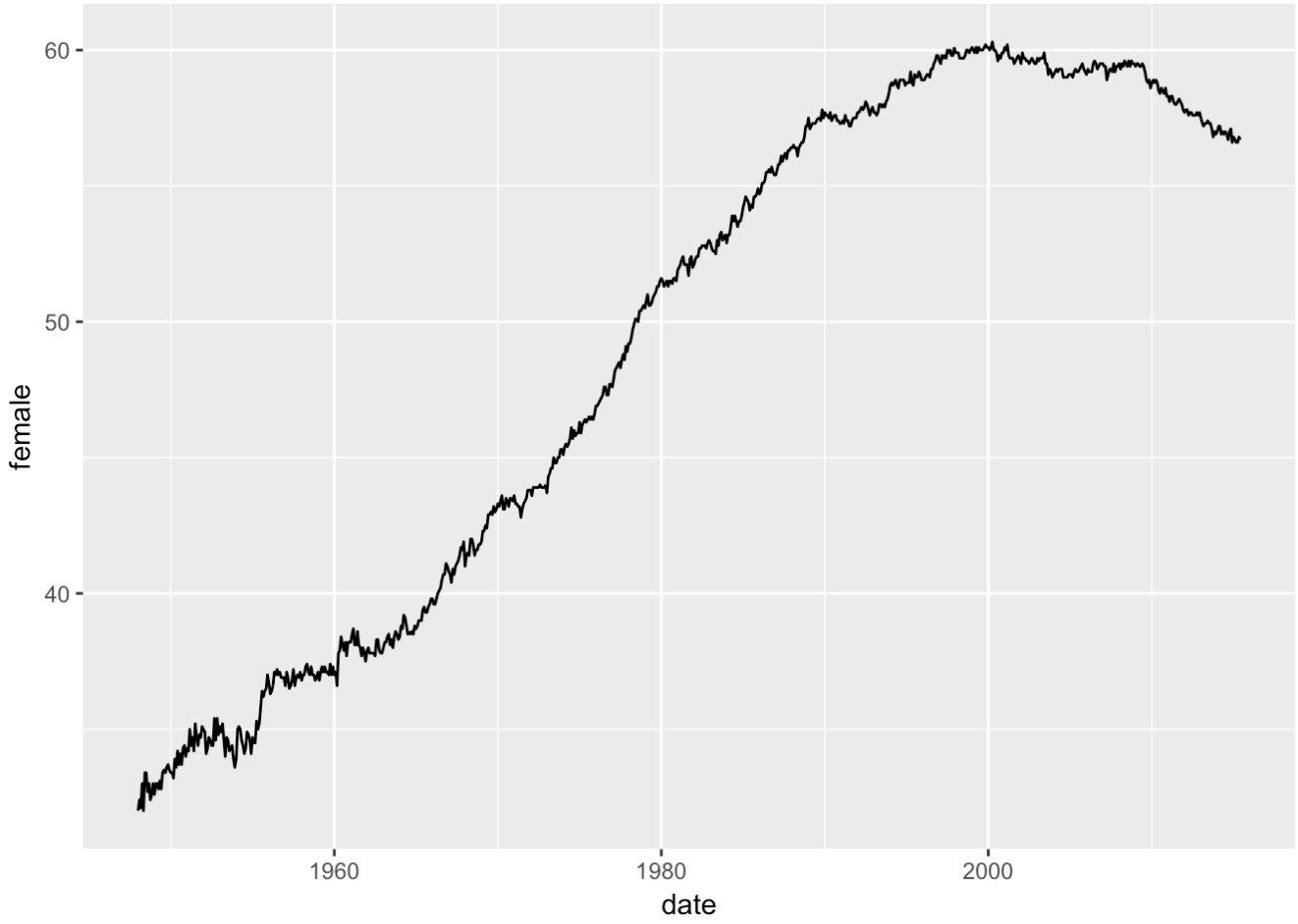
For your reference, the line above would be the equivalent of doing the following:

```
ggplot(Participation, aes(x = date, y = female))
```

The commands above won't do anything yet because we haven't yet told `ggplot` what kind of plot we want. We want a line plot. To make a plot we have to specify a kind of `geom` for `ggplot`. A `geom` is a type of object that talks to R and says "Make Me A Line Plot" or "Make Me A Scatter Plot" or "Make Me a Density Plot". We shall use `geom_line` as a line graph is the one we want. Check the notes on using `ggplot` if you want to be sure of this (see the Moodle uploads for these documents or the `ggplot` cheatsheet under the help menu).

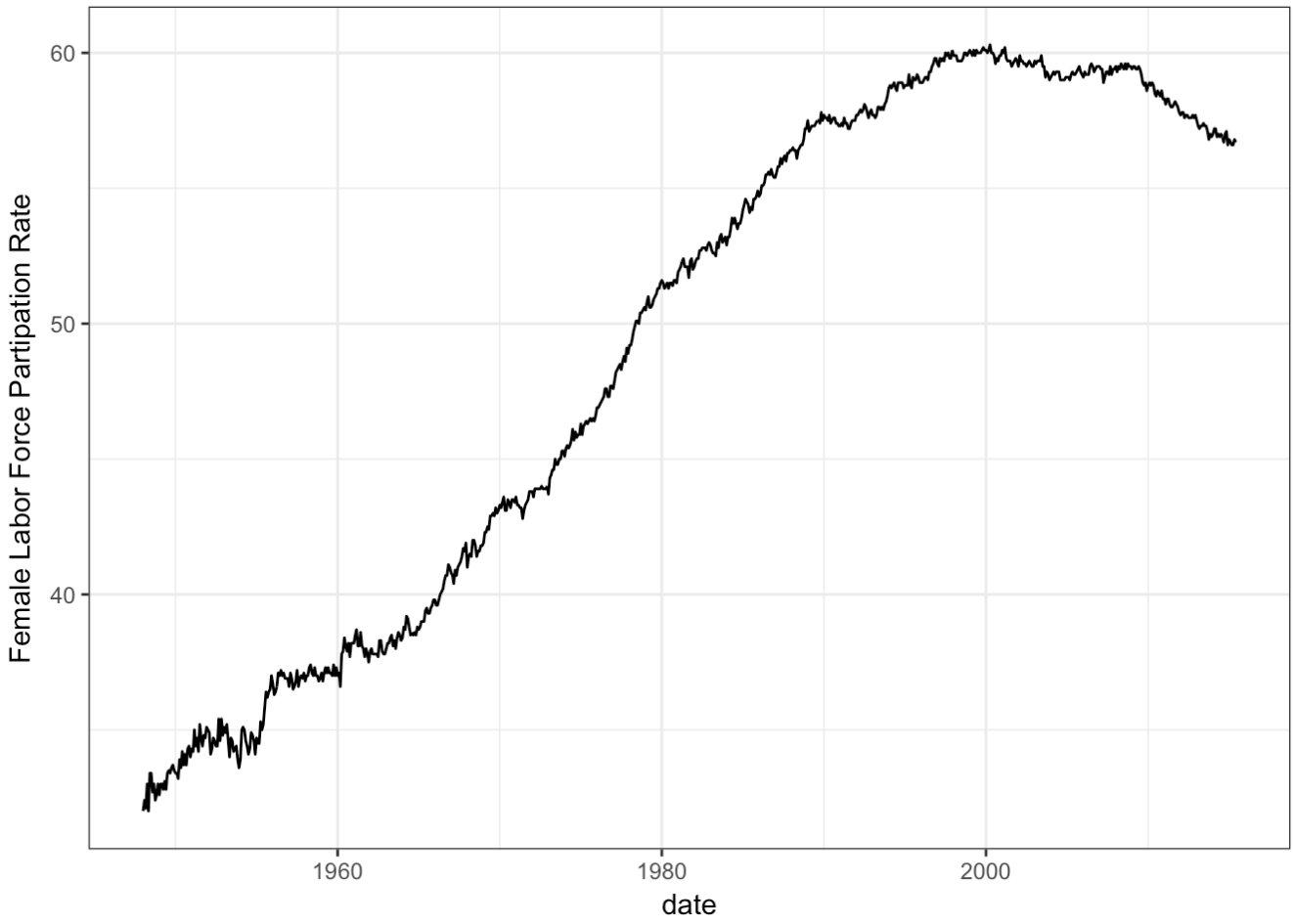
Now, `ggplot` also needs you to end each line with a `+` sign. Think backto the 6-year-old from Lab 2 who was telling you about the `%>%` symbol: she might say 'and' a lot. We have to treat ggplot like a 6-year-old who says 'and' a lot. So it doesn't just want to be told what the x-axis and y-axis are, it also needs to be told "and I want a line plot".

```
Participation %>% #Put in data data and then...
  ggplot(aes(x = date, y = female)) + #ggplot x and y AND
  geom_line() #I want a line plot
```

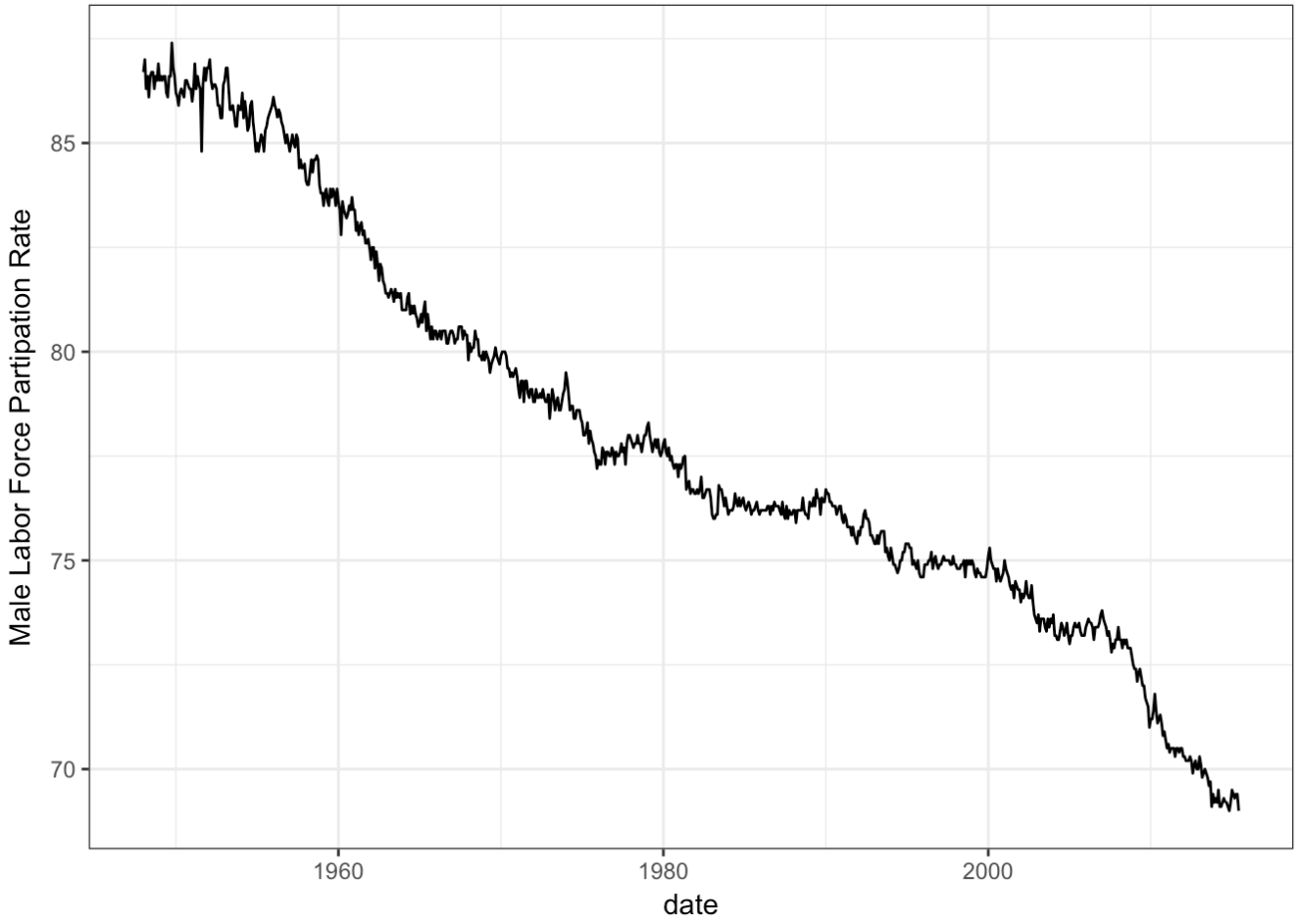


I have created the graph for women below again, but with a label for the y-axis and a different background.

```
#Female Participation Rate Graph
Participation %>%
  ggplot(aes(x = date, y = female)) +
  geom_line() +
  ylab("Female Labor Force Partipation Rate") + #This is a label for the y-axis
  theme_bw() #this is a "theme" for the ggplot from the ggthemes package
```



1. Now it's your turn. Use the commands you've learned so far to create a graph for the labor force participation rate of **men**. Your graph should look like the one below.
2. You only need to change two parts of the code above.



1. Is it easy to compare the two graphs? What makes it easier or harder?

Making Data fit for Purpose

As it stands, with the data as it exists in the way we get it from the Federal Reserve, we can create the graphs for male and female participation rate independently, but we have to stop there. We can't superimpose the male and female graphs on each other or create facets of them in one command (more on faceting later). Why is that?

As we learned previously, to construct useful visualizations and to conduct worthwhile models, our data ought to be **tidy**. **Tidy data** typically has unique **variables** in each column and unique **cases** in each row. So let's take a moment to think about the data that we have.

---

1. What is measured in each column in the existing data?
2. Does each column measure a *unique* variable?
3. What variables ought we to have in its own columns? Why?

To super-impose the two graphs, we shall change the "shape" of the data or `gather` it. `gather` is a very useful command in the `dplyr` package (part of the `tidyverse`) that allows us to change the cases and variables we're interested in analyzing.

When you use `gather` you need to be aware of several things:

1. gather wants a **key-value** pair.
2. the **key** is a variable that you name which tells you what you're doing to the data; our key is **sex** as we're interested in males and females.
3. the **value** is the labor force participation rate that we call **rate** to corresponds to the labor force participation rate.
4. we then tell gather to gather up the male and female variables we had previously and stack them up on top of each other making the data **longer** than it was before.

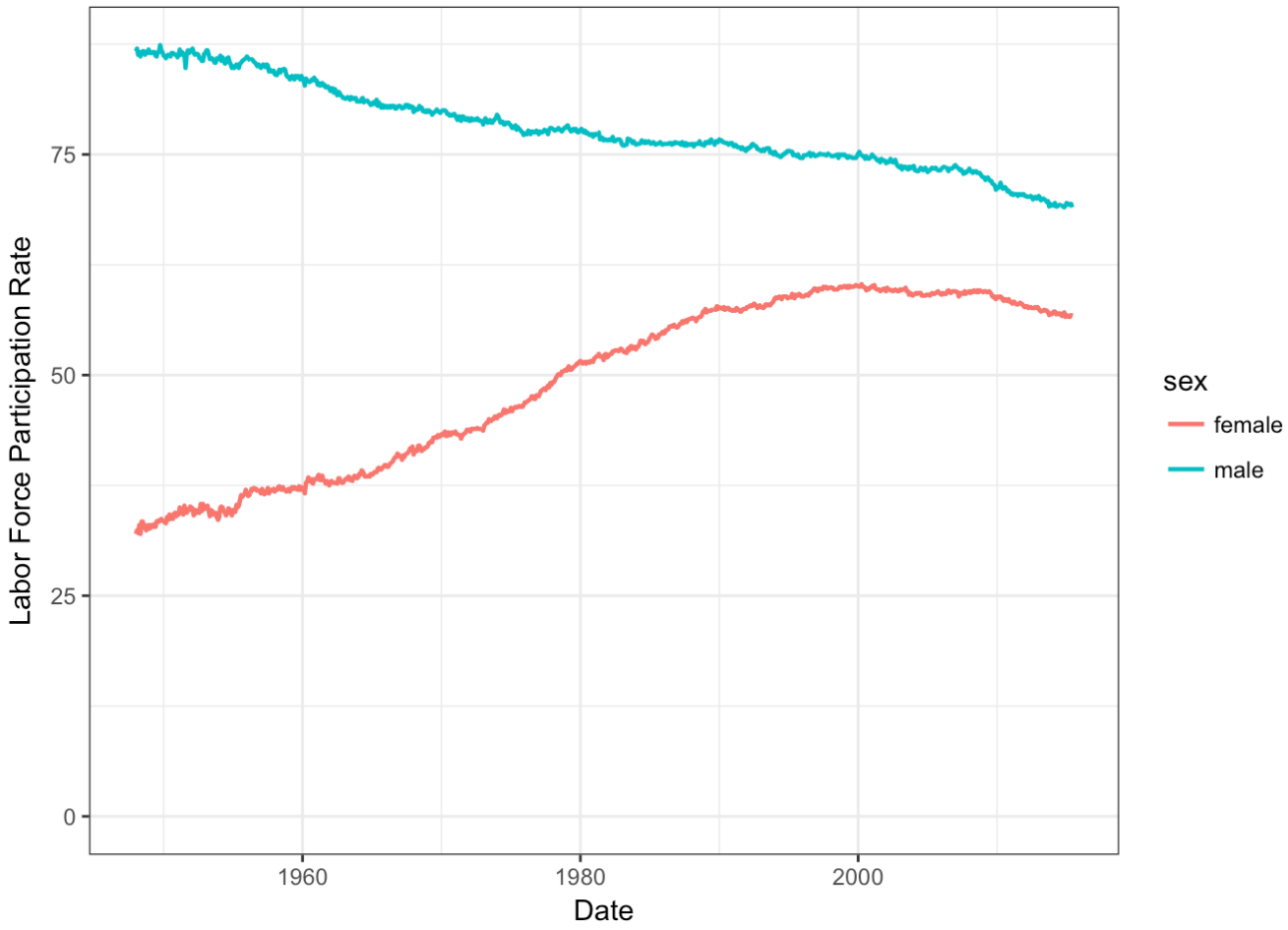
Let's run the code now:

```
Participation2 <- #notice the name of this new object
 Participation %>% #pipe in the original data
 gather(sex, rate, male, female)
```

Superimposing Lines and Faceting Graphs

With our new data we shall superimpose the male and female participation rates on the same graph and give each its own color.

```
Participation2 %>%
  ggplot(aes(x = date, y = rate, color = sex)) +
  geom_line(size = 0.8) +
  ylim(0, NA) +
  ylab("Labor Force Participation Rate") +
  xlab("Date") +
  theme_bw()
```



Notice that this graph includes zero (0) on the y-axis, whereas our previous graphs did not. This happened because I specified an option called `ylim()` (the limites of the y-axis) with 0 as the low end and NA (not applicable) as the top end.

---

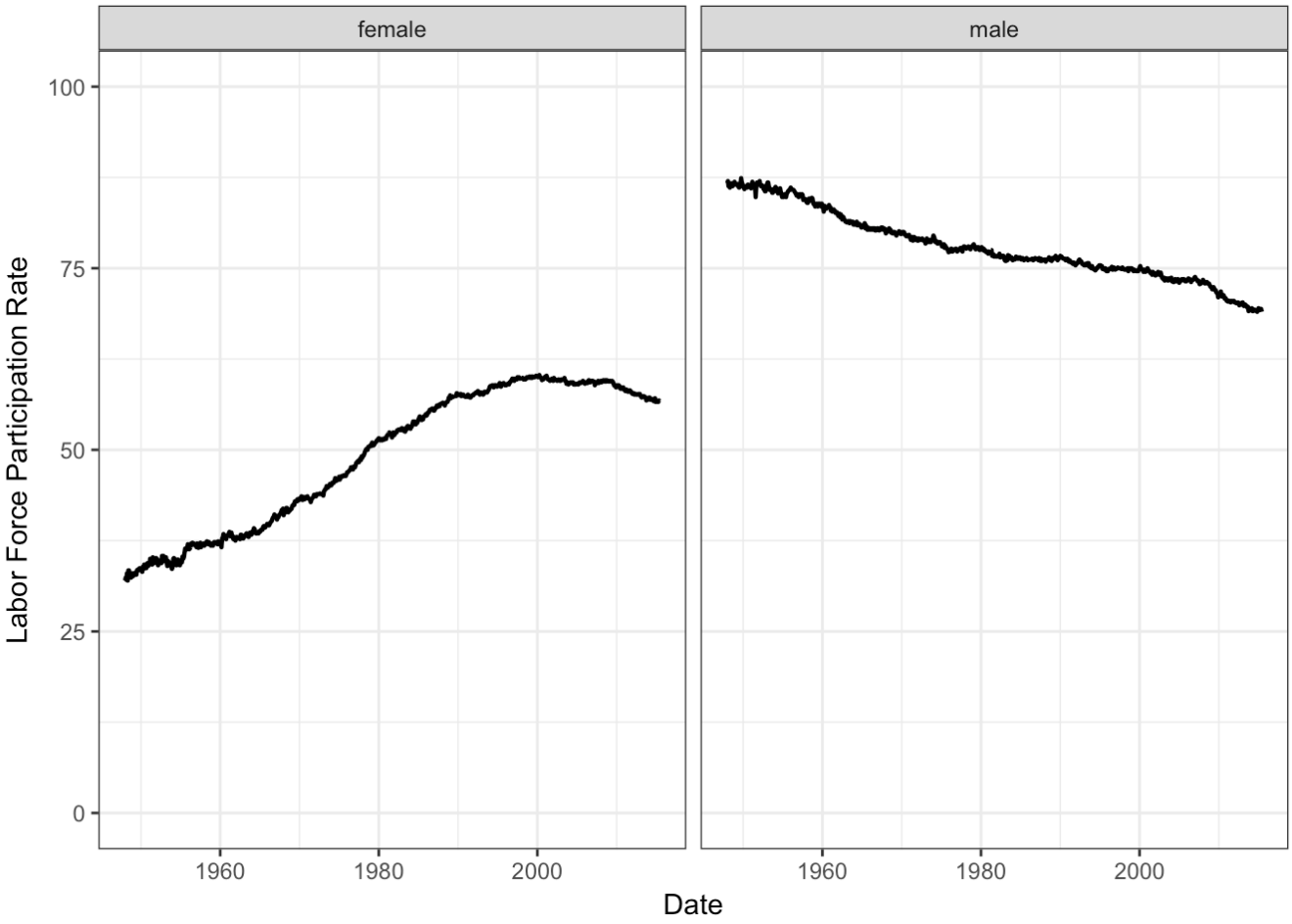
1. What changes when we add zero to the graph?
2. Why do you think zero is important to include in the graph?

You have now created a *static* version of the graph from FRED. Well done!

Faceting

Another way of presenting graphs is to present them in *facets* that are side-by-side or above each other. `ggplot` allows you to do this easily with the option `facet_grid()`. In the ggplot below, we add the option `facet_grid(.~sex)` to have R *facet* the ggplots on the `sex` variable we created with `gather`.

```
Participation2 %>%
  ggplot(aes(x = date, y = rate)) +
  geom_line(size = 0.8) +
  facet_grid(. ~sex) +
  ylim(0, 100) +
ylab("Labor Force Participation Rate") +
xlab("Date") +
theme_bw()
```



R can make a dygraph like FRED

We'll leave it at that for now, but here you can see a lovely `dygraph` that can be generated using htmlwidgets (go to www.htmlwidgets.org (http://www.htmlwidgets.org/) to see a variety of other advanced options for what you can do when you become and R Champion).

Spreading your data

`spread` and `gather` are **complementary** commands: they go in opposite directions. Earlier, we took our data and `gather` ed it so that we had a male and a female rate of labor force participation for each year. If, for some odd reason, you wanted to go back to the original shape of the data, you can use the command `spread`, which spreads the data back to what it looked like previously.

Here is an example:

```
ParticipationSpread <- #assign the name
Participation2 %>% #Tell it to put in the data we used for the graphs
  spread(sex, rate) #spread on the key `sex` with the value `rate`
head(ParticipationSpread) #Check that the data looks like we want it to
```

```
##         date female male
## 1 1948-01-01   32.0 86.7
## 2 1948-02-01   32.4 87.0
## 3 1948-03-01   32.1 86.3
## 4 1948-04-01   33.0 86.6
## 5 1948-05-01   32.0 86.1
## 6 1948-06-01   33.4 86.6
```

Next steps

The data that we've looked at above is just on example of a situation in which we can visualize data. There are many sources of data out there that you will want to use in your data analysis. In the next part of today's lab you will take a look at importing data from GapMinder.org to draw graphs looking at HIV. In it you will learn about using `filter` and `select` in more complicated ways than previously. We don't merely have to stipulate what specific variables we want, we can stipulate variables that meet a certain condition, such as starting with a certain letter or word. If we have time, we shall look at data from GapMinder in order to test out our ideas. You'll

also pracise on DataCamp.