Lab 2: Basics of Data

Learning Outcomes

At the end of today's lab you should be able to do the following:

1. Demonstrate knowledge of how to interact with R using RStudio
2. Identify the basic strutures of a data frame: the number and names of variables and cases
3. Create a new variable from existing variables in a data frame.
4. Filter information out of a data frame and compare summary statistics across filtered observations
5. Select out one or more variables from all of the variables in a data frame.

Preliminaries

Before you start, open up a new R Notebook: using the menus go to: File -> New File -> R Notebook.

At the top of the notebook, create a code chunk (CMD + OPTION + I on MacOS or CTRL + ALT + I on Windows) and load these packages (you could also use the command `require()`):

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

Remember that you have to tell R Markdown what packages you plan to use in order to use them in the file. We are going to use functions from the `mosaic` and `tidyverse` packages and so we have to tell our R Markdown file to use them.

Loading Data

We now want to start to work with some data. We shall first download a dataset that I have already accessed from the Federal Reserve Bank of St. Louis using the FRED database. You can access the `.csv` (comma separated value) file of the data here: https://drive.google.com/file/d/0B17bVUW1HMv-SGNheF9haXJ0bVk/view?usp=sharing (https://drive.google.com/file/d/0B17bVUW1HMv-SGNheF9haXJ0bVk/view?usp=sharing). The file is called `labor_force_levels.csv`.

Download the file and save it in a logical place. I would recommend that you save it in your Behavioral Economics folder under a folder called Lab2 or 'more' and then create a backup of the file in that folder too (name it 'NAMEOFFILE_backup'. Open the file if you want to see what a csv looks like in Excel or Google Sheets. Do not save any changes you make to it.

Now, we are going to open the file in R using RStudio or RStudio Server. Use the command below, but be sure to change the filepath to the one where you have put the `labor_force_levels.csv` file. Note, that in the filepath below I put two periods `..` at the start to tell R to look 'back' (to one level up in the folder structure). I then told it to use the folder called 'more'. Inside the folder called 'more' there is a file called `labor_force_levels.csv` which is what I want it to open.

RStudio Server Note: If you are using RStudio Server, then you will need to *upload* the csv file to your folder. Simply click on 'upload' in the bottom right tab and navigate to the file and click to upload it. You will then have it available in your workspace.

```
ParticipationLevels <- read_csv("../more/labor_force_levels.csv")
```

```
## Parsed with column specification:
## cols(
##   date = col_character(),
##   male = col_integer(),
##   female = col_integer()
## )
```

This command instructs R to read the csv file and assign it to an object called `ParticipationLevels` (hence `read_csv()`). Importantly, we are using a command from the `tidyverse()` package to do this. There is also a base R function called `read.csv()` but it is much slower than `read_csv()`. ParticipationLevels is a data table that we shall use to practise some basic analysis.

The data table contains the Bureau of Labor Statistics' counts for men and women participating in the labor force each period. You should see that the workspace area in the upper righthand corner of the RStudio window now lists a data set called `ParticipationLevels` that has 810 observations of 3 variables. As you interact with R, you will create a series of **objects**. Sometimes you load an object as we have done here, and sometimes you create an object yourself as the result of a computation or some analysis you have performed. Because you access the data from the web, this command (and the entire assignment) will work in a computer lab, in the library, or in your dorm room; anywhere you have access to the internet.

The Data: Bureau of Labor Statistics (BLS) counts of participants in the labor force

The Bureau of Labor Statistics has collected data regularly since 1948 on the numbers of men and women who participate in the labor force. These days, the counts are published monthly. The BLS needs to understand whether someone participates in the labor force to come to judge employment in the US. You might remember from ECO153 or ECO253 that the government measures employment only from those people participating in the labor force, that is, either working or looking for work. Someone who does not have a job and is not actively searching for work is not unemployed. Rather, they are a **discouraged worker**. Discouraged workers do not participate in the labor force. Why are we looking at macro-variables in a microeonomics-heavy course? Look at Cartwright pp 77-84 for an idea about why.

We and the government are interested in understanding what percentages of men and women are working at a given point in time. To calculate those proportions we need the counts. We shall use the percentages in later labs. For now, we want to understand the counts to get a sense of the how the data works.

```
ParticipationLevels
```

You should see four columns of numbers, each row representing a different date. The first entry in each row, corresponding the first column, is simply the row number. The second column is the date which follows international conventions and is written as YYYY-MM-DD (note this is different to how many people write dates

in the USA, where most people write dates as MM-DD-YYYY). R is used internationally and so adheres to the international scientific dating conventions. This system is often called 'unambiguous' because of the order. The fully specified date and time, called a POSIX time would be YYYY-MM-DD HH:MM:SS with a 24-hour clock, e.g. 2015-07-18 14:17:32).

The third and fourth columns are the numbers of men and women who participated in the labor force at the given date. Use the scrollbar on the right side of the console window to examine the complete data set.

Note that the row numbers in the first column are not part of the BLS data. R adds them as part of its printout to help you make visual comparisons. You can think of them as the index that you see on the left side of a spreadsheet. In fact, the comparison to a spreadsheet will generally be helpful. R has stored the BLS data in a kind of spreadsheet or table called a **data frame** or **data table**.

You would have seen that looking at the data frame in its entirety is quite cumbersome. We would probably rather just take a look at some portion of our data, such as the top or the bottom of the data. We can do this with the `head` and `tail` commands. Run them now to check out the first few rows of the data. By default, head and tail show you 6 rows. You can change that by adding a command and a number after the the name of the data frame.

```
head(ParticipationLevels)
```

```
## # A tibble: 6 x 3
##       date  male female
##      <chr> <int>  <int>
## 1 1948/1/1 43214  16881
## 2 1948/2/1 43400  17124
## 3 1948/3/1 43080  16990
## 4 1948/4/1 43215  17462
## 5 1948/5/1 43002  16970
## 6 1948/6/1 43257  17700
```

```
head(ParticipationLevels, 8)
```

```
## # A tibble: 8 x 3
##       date  male female
##      <chr> <int>  <int>
## 1 1948/1/1 43214  16881
## 2 1948/2/1 43400  17124
## 3 1948/3/1 43080  16990
## 4 1948/4/1 43215  17462
## 5 1948/5/1 43002  16970
## 6 1948/6/1 43257  17700
## 7 1948/7/1 43429  17752
## 8 1948/8/1 43403  17403
```

```
tail(ParticipationLevels)
```

```
## # A tibble: 6 x 3
##       date  male female
##      <chr> <int>  <int>
## 1 2015/1/1 83771  73408
## 2 2015/2/1 83772  73230
## 3 2015/3/1 83694  73211
## 4 2015/4/1 83805  73267
## 5 2015/5/1 83892  73577
## 6 2015/6/1 83490  73547
```

```
tail(ParticipationLevels, 12)
```

```
## # A tibble: 12 x 3
##         date  male female
##        <chr> <int>  <int>
##  1  2014/7/1 83017  73031
##  2  2014/8/1 83010  73008
##  3  2014/9/1 82983  72862
##  4 2014/10/1 82950  73293
##  5 2014/11/1 82961  73442
##  6 2014/12/1 83210  72919
##  7  2015/1/1 83771  73408
##  8  2015/2/1 83772  73230
##  9  2015/3/1 83694  73211
## 10  2015/4/1 83805  73267
## 11  2015/5/1 83892  73577
## 12  2015/6/1 83490  73547
```

You can see the dimensions (`dim`) of this data frame by typing:

```
dim(ParticipationLevels)
```

```
## [1] 810   3
```

This command should output `[1] 810 3` , indicating that there are 810 rows and 3 columns (we'll get to what the `[1]` means in a bit), just as it says next to the object in your workspace. Consult with a neighbor and feel free to type `?dim` and see what the help tells you to find out more (see the lower right pane of your RStudio for the Help panel).

You can see the names of these columns (orvariables) by typing:

```
names(ParticipationLevels)
```

```
## [1] "date"   "male"   "female"
```

You should see that the data frame contains the columns `date` , `male` , and `female` .

Another way to obtain similar information is to ask for the structure of the data using the command `str()` :

```
str(ParticipationLevels)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    810 obs. of  3 variables:
##  $ date  : chr  "1948/1/1" "1948/2/1" "1948/3/1" "1948/4/1" ...
##  $ male  : int  43214 43400 43080 43215 43002 43257 43429 43403 43240 43396 ...
##  $ female: int  16881 17124 16990 17462 16970 17700 17752 17403 17575 17250 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 3
##   .. ..$ date  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ male  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ female: list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

As we saw with the `dim` command, the `str` command tells us that the data has 810 observations and 3 variables. As with the `names` command, `str` also tells us the names of the three variables (`date` , `male` and `female`) and the format of these variables. It tells us that `date` has the format of a “chr” (which is short for “character” which means a word or character string, this is a problem we shall have to solve later). It tells us that male and female have the format of “int”, which is short for “integer”.

At this point, you might notice that many of the commands in R look a lot like functions from math class. So, invoking R commands means supplying a function with some number of arguments that appear in parentheses. For example, when you say that a variable y is a function of another variable x you write: $y = f(x)$. The `dim` , `names` and `str` commands each took a single argument, the name of a data frame `ParticipationLevels` . The idea of an R command working like a function will remain helpful in the future.

One advantage of RStudio is that it comes with a built-in data viewer. Click on the name `ParticipationLevels` in the *Environment* pane (upper right window) that lists the objects in your workspace. This will bring up an alternative display of the data set in the *Data Viewer* (upper left window). You can close the data viewer by clicking on the *x* in the upper lefthand corner.

Summary Statistics

In economics, we are often interested in knowing basic summary statistics of variables and also in generating new variables that contain new information. One very useful command in R, using the `mosaic` package, is the `favstats` command (read as “fave stats”). What, you might ask, are an economist’s favorite statistics? How about the following?

1. mean
2. median
3. standard deviation
4. minimum
5. maximum

Let's try the `favstats` command with the ParticipationLevels data frame. Make sure that you use the `library()` command to load the `mosaic` package before trying to run `favstats`. The function `favstats()` uses the following structure `favstats(~VARIABLENAME, data = DATATABLENAME)`. That is, we use a tilde `~` before the name of our variable, and we also specify the data as being a particular data object we plan to use.

```
favstats(~female, data = ParticipationLevels)
```

As you can see we have information about the mean number of females who participated in the labor force over the period (in thousands). Each month, approximately 45,567,830 women participated in the labor force, or about 46 million. That should sound about right.

Try the `favstats` command with the male variable in ParticipationLevels

1. What was the mean number of mean employed each month over the period 1948 to 2015?
2. What was the standard deviation of the number of males employed each month?
3. What were the minimum and maximum numbers of males employed each month?

When you ran the `favstats` command, the final column was labeled as 'missing'. Missing data exist when there are no observations of that variable for a particular case. For both males and females in the US, we have no missing data. This makes sense for aggregate employment data, but if you were asking each person individually what their income was, people would be less likey to report that, and data would likely be missing. One of the jobs of the statistician or econometrician is to work out what missing data look like. Dealing with that is beyond the scope of this course. We will mainly deal with 'non-missing' data.

Data Verbs

During this course we shall use a variety of data verbs, commands that "do" things to data (hence "verb"). The list of data verbs includes:

1. `mutate()` : change the date, thus 'mutate' it in some way
2. `filter()` : filter some subset of cases out from the total list of variables
3. `select()` : select some subset of variables out from the total list of variables
4. `join()` : join two different data tables together
5. `group_by()` : group a data table using a variable (e.g. gender) to understand some patterns in the data
6. `summarise()` : summarise the data to find some useful statistic; often combined with `group_by` to contrast differences across groups
7. `gather()` and `spread()` : gather and spread re-shape the data by making the data narrower (for gather) or wider (for spread) depending on what we need. Normally, we need to re-shape data when we want to change the particular cases that we have so that we can draw insights based on alternative cases to those we already have. For example, to draw a graph, we might need to change the case into a summary statistic, such as the average number of people engaged in an activity, rather than having the case be each unique individual in a dataset.

Piping

Before we do much else, we want to think about how to write good code. One of doing that is by using "pipes".

In R, the `%>%` symbol is called a **pipe**. You should read the pipe as saying “Then” or “And then”. A pipe “pipes in” an argument so that you are able to use it in a function.

I like the to think of my programs that include the pipe as stories told by a 6-year-old:

That way, I don’t get confused about what it means because I always get the functions in the right sequence.

So the command we will use to tell R what data to use is the following:

```
Participation %>%
```

> *Don’t leave this command by itself - R will look for what you’re trying to “pipe” the data into.*

Filtering and Mutating

We might be interested in seeing the total number of participants in the labor force each year. First, though, we have to tell R that the variable we have called `date` is, in fact, a date.

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

`as.Date()` is a function which lets R know that a variable we are concerned with is a date. Now re-run `str()` on `ParticipationLevels`. What does it say about the variable `date`?

Now, to undertand a subset of the cases, we could do one of two operations. We will do asimple one first. Let’s say we’re interested in finding out what the total number of labor force participants was in January of 1948. We shall use the command `filter` to do this.

```
Jan1948 <- #name the object & assign something to it
ParticipationLevels %>% #use the object ParticipationLevels
filter(date == "1948-01-01") #Filter on the date
Jan1948 #check the new object
```

```
## # A tibble: 1 x 3
##         date  male female
##       <date> <int>  <int>
## 1 1948-01-01 43214  16881
```

Let's go through a step-by-step interpretation of the string of code:

1. First, we tell R we want to assign the name `Jan1948` to our new variable.
2. Second, we tell R to pipe in (to use) the `ParticipationLevels` data.
3. Third, we then (%>%) tell R to use the filter command where it takes the variable `date` and only selects the case(s) for which date is equal to "1948-01-01" (the use of two equals signs `==` is very important here! You will get an error if you only use one equals sign)
4. Fourth, we tell R to display our new dataframe `Jan1948` . (notice, there's no pipe %>% at the end of the line before `Jan1948`)

We can see that there were 43,214 male labor force participants (in thousands) and 16,881 female (also in thousands). Now, we can treat R like a calculator and add these together

```
43214 + 16881
```

```
## [1] 60095
```

We can see that there are 60,095 total participants in the labor force (in thousands).

With R, as with your calculator, you need to be conscious of the order of operations. Here, we want to divide the number of boys by the total number of newborns, so we have to use parentheses. Without them, R will first do the division, then the addition, giving you something that is not a proportion.

How else could we do this? We could add the male and female totals together to create a new variable. To create a new variable we use the `mutate` command.

```
ParticipationLevels <-
  ParticipationLevels %>%
  mutate(total = male + female)
```

What does this string of code mean? Let's go through it step-by-step:

1. First, we tell R the name we want to assign our data frame. We're going to stick with old name of `ParticipationLevels` .
2. Second, we tell R we want to pipe in the data from the existing data frame `ParticipationLevels` to tell it we're using that data in our function.
3. Third, we then (`%>%`) call the mutate function and tell it that we're creating a new variable called `total` which is `male + female` .
4. We can now `head(ParticipationLevels)` to check what our new variable looks like. Go ahead and do

that on your own (it's not included in the code above)

We didn't create the total variable just because. Rather, we want to identify the proportions of men and women who are in the labor force. Let's attempt that now using the `mutate` command again and generate a new data frame that has the variable `femprop` for the proportion of the labor force that is made up of females.

```
ParticipationLevels <-
  ParticipationLevels %>%
  mutate(femprop = female / total)
head(ParticipationLevels)
```

```
## # A tibble: 6 x 5
##         date  male female total   femprop
##       <date> <int>  <int> <int>     <dbl>
## 1 1948-01-01 43214  16881 60095 0.2809052
## 2 1948-02-01 43400  17124 60524 0.2829291
## 3 1948-03-01 43080  16990 60070 0.2828367
## 4 1948-04-01 43215  17462 60677 0.2877861
## 5 1948-05-01 43002  16970 59972 0.2829654
## 6 1948-06-01 43257  17700 60957 0.2903686
```

---

1. Write out a point by point interpretation of the command like we did for the commands to generate `total`.
2. Now you generate a new variable called `maleprop` which is the proportion of male participants as a total of all the labor force participants.
3. Look at the head and the tail of the data frame (you can look at more than 6 rows if you want). What has happened to the proportion of males and females as a total of the labor force participants in the United States?
4. Create two new data frames Jan1960 and Jan2000 which `filter` out all other months except those ones in each frame. Display them and comment.

```
## # A tibble: 6 x 6
##         date  male female total   femprop  maleprop
##       <date> <int>  <int> <int>     <dbl>     <dbl>
## 1 1948-01-01 43214  16881 60095 0.2809052 0.7190948
## 2 1948-02-01 43400  17124 60524 0.2829291 0.7170709
## 3 1948-03-01 43080  16990 60070 0.2828367 0.7171633
## 4 1948-04-01 43215  17462 60677 0.2877861 0.7122139
## 5 1948-05-01 43002  16970 59972 0.2829654 0.7170346
## 6 1948-06-01 43257  17700 60957 0.2903686 0.7096314
```

```
## # A tibble: 6 x 6
##         date  male female  total   femprop  maleprop
##       <date> <int>  <int>  <int>     <dbl>     <dbl>
## 1 2015-01-01 83771  73408 157179 0.4670344 0.5329656
## 2 2015-02-01 83772  73230 157002 0.4664272 0.5335728
## 3 2015-03-01 83694  73211 156905 0.4665944 0.5334056
## 4 2015-04-01 83805  73267 157072 0.4664549 0.5335451
## 5 2015-05-01 83892  73577 157469 0.4672475 0.5327525
## 6 2015-06-01 83490  73547 157037 0.4683419 0.5316581
```

```
## # A tibble: 1 x 6
##         date  male female total   femprop  maleprop
##       <date> <int>  <int> <int>     <dbl>     <dbl>
## 1 1960-01-01 46296  22666 68962 0.3286738 0.6713262
```

```
## # A tibble: 1 x 6
##         date  male female  total   femprop  maleprop
##       <date> <int>  <int>  <int>     <dbl>     <dbl>
## 1 2000-01-01 76136  66131 142267 0.4648372 0.5351628
```

You can also use the command `filter` to create a new dataframe where the dates are in a particular range or only after a given date. For example, let's say that we were only interested in looking at dates since January 2000. We could easily tell R this by using the greater than sign, `>=` or the `>`
therefore and doing the following:

```
RecentParticipation <-
  ParticipationLevels %>%
  filter(date >= "2000-01-01")
```

Check that you ran the code correctly by checking the `head` of the data frame. We could have looked at all dates *prior* to the 2000s by using the less than sign `<` .

How about checking dates in a given 10 year period? Let's say we were only interested in the 1960s and we want to check ou that ten-year period.

```
The1960s <-
  ParticipationLevels %>%
  filter(date >= "1960-01-01" & date < "1970-01-01")
  head(The1960s)
```

```
## # A tibble: 6 x 6
##         date  male female total   femprop  maleprop
##       <date> <int>  <int> <int>     <dbl>     <dbl>
## 1 1960-01-01 46296  22666 68962 0.3286738 0.6713262
## 2 1960-02-01 46213  22736 68949 0.3297510 0.6702490
## 3 1960-03-01 45957  22442 68399 0.3281042 0.6718958
## 4 1960-04-01 46400  23179 69579 0.3331321 0.6668679
## 5 1960-05-01 46322  23304 69626 0.3347026 0.6652974
## 6 1960-06-01 46337  23597 69934 0.3374181 0.6625819
```

Again, we're not just filtering data because we think filtering is cool, we might be interested in what the mean proportions and variance were across decades because we have some hypothesis about different decades. Or we could lookat 5-year periods, or the periods before and after financial crises.

Complete the following exercise using `filter` and `favstats`:

1. What was the mean proportion of women in the labor force for the 1960s?
2. What was the mean proportion of women in the labor force in the 2000s?

TIP: Make sure that you have an 'n' of 120 when you run the favstats command. Why should it be 120?

```
##        min        Q1    median        Q3       max      mean        sd   n
##  0.3281042 0.3403247 0.3489816 0.3658732 0.3807091 0.3531088 0.0145727 120
##  missing
##        0
```

```
##        min        Q1    median        Q3       max      mean          sd
##  0.4615541 0.4638255 0.4648642 0.4655445 0.4681476 0.4648457 0.001339235
##    n missing
##  120       0
```

Selecting variables

Let's start to examine the data a little more closely. We can access the data in a single column of a data frame separately using a command called `select`:

```
ParticipationFemale <-
  ParticipationLevels %>%
  select(female)
head(ParticipationFemale)
```

This command will only show the number of females in the labor force during each period.

[exercise number] What command would you use to extract just the counts of **males** in the labor force? Try it!

Tip: If you use the up and down arrow keys, you can scroll through your previous commands, your so-called command history. You can also access it by clicking on the history tab in the upper right panel. This will save you a lot of typing in the future.

This seems like a fair bit for your [nth] lab, so let's stop here. To exit RStudio you can click the *x* in the upper

right corner of the whole window.
You will be prompted to save your workspace. If you click *save*, RStudio will save the history of your commands and all the objects in your workspace so that the next time you launch RStudio, you will see `ParticipationLevels` and you will have access to the commands you typed in your previous session. For now, click *save*, then re-start RStudio.

Conclusion

That was an introduction to R and RStudio, but we will provide you with more functions and a more complete sense of the language as the course progresses. Feel free to browse around the websites for R (http://www.r-project.org) and RStudio (http://rstudio.org) if you're interested in learning more, or find more labs for practice at http://openintro.org (http://openintro.org).