Quantitative Methods - Computer Lab 1

Ralf Becker

1 February 2021

Before you start you should create a space (i.e. a folder) on your computer from where you are planning to do all your R work. Make sure you understand where that folder is and that you know the path to that folder (Apple users, if you do not know how to find the path to a particular folder, then look at this https://piazza.com/class/k5wtxg2vebk5or?cid=7). If you are using a computer lap computer you should create a folder for your R work on the P drive. This will be accessible to you whenever you log on to a University computer, or from anywhere via https://pdrives.manchester.ac.uk/horde/login.php.

Let's say you named your folder `Rwork` directly on the P drive and in that folder created a subfolder for this unit `QM` , then the path to your folder will be `P:\Rwork\QM`.

For this computer lab we are using exactly the same data as for the first lecture. So please make sure that you have the datafile `CK_public.xlsx` in the folder you just created and want to work from. It will also be useful to have `CK_codebook.txt` readily available (also from BB, Lecture 1).

Preparing your script file

Via the RStudio menu (FILE - NEW FILE - R SCRIPT) open a new script file and safe it in the folder from which you want to work (see above).

Start by creating a comment line at the top of that file which may say something like

```
# File for first QM Computer Lab
# February 2020
```

If not mentioned otherwise all the following code should be added to that script file and executed from there.

Next you should ensure that you set the working directory to the directory where your scriptfile and datafile is in. If your folder was `P:\Rwork\QM` then the command below should read `setwd("P:/Rwork/QM")`. Note that all backward slashes (`\`) have to be replcaed by forward slashes (`/`). Don't ask why, just accept ;-).

```
setwd("XXXX:/XXXX")    # replace the XXXX with your drive and path
```

Note that R only understands forward slashes `/`.

Load the libraries which we want to use.

```
library(tidyverse)    # for almost all data handling tasks
```

```
## -- Attaching packages --------------------------------------------------------------- tidyverse 1.3.0

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(readxl)       # to import Excel data
library(ggplot2)      # to produce nice graphiscs
library(stargazer)    # to produce nice results tables
```

```
##
## Please cite as:
## 
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
## 
##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

If RStudio tells you that one or more of these libraries are not installed then install these (not any others) on the machine you are working from. For instance, if `stargazer` was not installed you would receive an error message like "Error in library(stargazer) : there is no package called 'stargazer' ", and in that case you should run:

```
install.packages("stargazer")
```

You can run this straight from the command/console or you could instead install the package via the packages tab on the right hand side. **Do not do this on computer lab computers as these packages should be preinstalled**.

YOu will need to do this only once on your computer and once you have done that you can call `library(stargazer)` again without running into problems.

Data Upload

Make sure you have set the working directory to the directory in which you saved your script file (see above). As we are dealing with data in an excel file we will use the `read_excel` function to load the data. Before you run this command open the actual spreadsheet and try to see how missing data are coded. In this case they are coded as dots ("."). But in other datafiles you may find "NA", "n.a.", "-9999" or other ways to code missing data. It is best to let R know how missing data look. This will simplfy your life tremendously.

```
CKdata <- read_excel("CK_public.xlsx",na = ".")
```

Make sure that your data upload is successful. Easiest to run the `str` (structure) function.

```
str(CKdata)  # prints some basic info on variables
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    410 obs. of  46 variables:
##  $ SHEET   : num  46 49 506 56 61 62 445 451 455 458 ...
##  $ CHAIN   : num  1 2 2 4 4 4 1 1 2 2 ...
##  $ CO_OWNED: num  0 0 1 1 1 1 0 0 1 1 ...
##  $ STATE   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SOUTHJ  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CENTRALJ: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NORTHJ  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ PA1     : num  1 1 1 1 1 1 0 0 0 1 ...
##  $ PA2     : num  0 0 0 0 0 0 1 1 1 0 ...
##  $ SHORE   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NCALLS  : num  0 0 0 0 0 2 0 0 0 2 ...
##  $ EMPFT   : num  30 6.5 3 20 6 0 50 10 2 2 ...
##  $ EMPPT   : num  15 6.5 7 20 26 31 35 17 8 10 ...
##  $ NMGRS   : num  3 4 2 4 5 5 3 5 5 2 ...
##  $ WAGE_ST : num  NA NA NA 5 5.5 5 5 5 5.25 5 ...
##  $ INCTIME : num  19 26 13 26 52 26 26 52 13 19 ...
##  $ FIRSTINC: num  NA NA 0.37 0.1 0.15 0.07 0.1 0.25 0.25 0.15 ...
##  $ BONUS   : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ PCTAFF  : num  NA NA 30 0 0 45 0 0 0 0 ...
##  $ MEALS   : num  2 2 2 2 3 2 2 2 1 1 ...
##  $ OPEN    : num  6.5 10 11 10 10 10 6 0 11 11 ...
##  $ HRSOPEN : num  16.5 13 10 12 12 12 18 24 10 10 ...
##  $ PSODA   : num  1.03 1.01 0.95 0.87 0.87 0.87 1.04 1.05 0.73 0.94 ...
##  $ PFRY    : num  1.03 0.9 0.74 0.82 0.77 0.77 0.88 0.84 0.73 0.73 ...
##  $ PENTREE : num  0.52 2.35 2.33 1.79 1.65 0.95 0.94 0.96 2.32 2.32 ...
##  $ NREGS   : num  3 4 3 2 2 2 3 6 2 4 ...
##  $ NREGS11 : num  3 3 3 2 2 2 3 4 2 4 ...
##  $ TYPE2   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ STATUS2 : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ DATE2   : num  111792 111292 111292 111492 111492 ...
##  $ NCALLS2 : num  1 NA NA NA NA NA NA 2 NA 1 ...
##  $ EMPFT2  : num  3.5 0 3 0 28 NA 15 26 3 2 ...
##  $ EMPPT2  : num  35 15 7 36 3 NA 18 9 12 9 ...
##  $ NMGRS2  : num  3 4 4 2 6 NA 5 6 2 2 ...
##  $ WAGE_ST2: num  4.3 4.45 5 5.25 4.75 NA 4.75 5 5 5 ...
##  $ INCTIME2: num  26 13 19 26 13 26 26 26 13 13 ...
##  $ FIRSTIN2: num  0.08 0.05 0.25 0.15 0.15 NA 0.15 0.2 0.25 0.25 ...
##  $ SPECIAL2: num  1 0 NA 0 0 0 0 0 0 0 ...
##  $ MEALS2  : num  2 2 1 2 2 2 2 2 2 1 ...
##  $ OPEN2R  : num  6.5 10 11 10 10 10 6 0 11 11 ...
##  $ HRSOPEN2: num  16.5 13 11 12 12 12 18 24 11 10.5 ...
##  $ PSODA2  : num  1.03 1.01 0.95 0.92 1.01 NA 1.04 1.11 0.94 0.9 ...
##  $ PFRY2   : num  NA 0.89 0.74 0.79 0.84 0.84 0.86 0.84 0.84 0.73 ...
##  $ PENTREE2: num  0.94 2.35 2.33 0.87 0.95 1.79 0.94 0.94 2.32 2.32 ...
##  $ NREGS2  : num  4 4 4 2 2 3 3 6 4 4 ...
##  $ NREGS112: num  4 4 3 2 2 3 3 3 3 3 ...
```

You should now see the `CKdata` object in your Environment (right hand pane).

There are 410 observations, each representing one fast-food restaurant. Each restaurant has some variables which characterise the store and then two sets of variables which are observed before the new minimum wage was introduced to New Jersey (Wave 1: Feb 15 to Mar 14, 1992) and after the policy change (Wave 1: Nov 5 to Dec 31, 1992). Variables which relate to the second wave will have a `2` at the end of the variable name.

The following variables will be important for the analysis here:

- `SHEET`, this is a unique identifier for each restaurant
- `STATE`, 1 if New Jersey (NJ); 0 if Pennsylvania (Pa)
- `WAGE_ST`, starting wage ($/hr), Wave 1
- `WAGE_ST2`, starting wage ($/hr), after policy, Wave 2
- `STATUS2`, the status of the Wave 2 interview, 0 = refused, 1 = completed, 3, permanently closed, 2, 4 and 5 = temporarily closed
- `CHAIN`, 1 = Burger King; 2 = KFC; 3 = Roy Rogers; 4 = Wendy's
- `EMPFT`, # full-time employees before policy implementation
- `EMPFT2`, # full-time employees after policy implementation
- `EMPPT`, # part-time employees before policy implementation
- `EMPPT2`, # part-time employees after policy implementation
- `NMGRS`, # managers/ass't managers before policy implementation
- `NMGRS2`, # managers/ass't managers before policy implementation

Accessing data

The `CKdata` item in your environment basically contains the entire spreadsheet of data. You can look at the entire spreadsheet by clicking on the tiny spreadsheet in the `CKdata` line. This will open a new tab with the spreadsheet. Have a look and then close the tab again.

It is important to know how you can access subsets of data. Run through the following commands to see what happens. Perhaps also experiment a bit by changing the commands and predicting what the outcome should be.

```
CKdata[1,]
CKdata[,2]
CKdata[3,4]
CKdata[,4:6]
CKdata[4:6]
CKdata[2:4,5:10]
CKdata$SOUTHJ
CKdata$SOUTHJ[1:5]
CKdata[c("CHAIN","STATE")]
```

These are all different ways to select particular observations (rows in a spreadsheet) and variables (columns in a spreadsheet). There were two particular techniques which are important

- `CKdata$CHAIN` did address the `CHAIN` variable in the `CKdata` dataset (`DATASET$VARIABLE`).
- `c("CHAIN","STATE")` allowed us to access two very particular variables, ignoring all other 44 variables. It is useful to know what `c("CHAIN","STATE")` actually does.

```
c("CHAIN","STATE")
```

```
## [1] "CHAIN" "STATE"
```

It creates a list with two elements, here two text strings. But you can also create lists with numbers.

```
c(3.5,2/3)
```

```
## [1] 3.5000000 0.6666667
```

In fact you could even create lists which mixed datatypes.

```
c("Countries in the EU", 28-1, ":-(")
```

```
## [1] "Countries in the EU" "27"                  ":-("
```

Lists are quite fundamental to the way how R stores data.

Data Formats

We can see (from the output to `str(CKdata)`) that all variables are numeric (`num`) variables, i.e. the software will treat them as numbers. If you check the `CK_codebook.txt` on BB you will find what all the numbers mean. For instance an entry of `2` in the `CHAIN` variable means that this is a "KFC" restaurant.

For later it will be convenient to have `STATE` and `CHAIN` variables which aren't numeric, but categorical variables. In R these are called `factor` variables. Hence we will create two new variables, `STATEf` and `CHAINf` which contain the same info, but just in a way easier to understand.

```
CKdata$STATEf <- as.factor(CKdata$STATE)  # translates a variable into a factor variable
levels(CKdata$STATEf) <- c("Pennsylvania","New Jersey") # changes the names of the categories
```

There are a number of important elements in these two lines

- `CKdata$STATEf <-` creates a new variable in the `CKdata` object, called `STATEf`. `<-` is called the assignment operator and it assigns some value to that new variable. It will assign whatever comes to the right hand side of `<-`
- `as.factor()` is a function called `as.factor`. If you want to find out what a function does you can call the help function `?as.factor`. This one ensures that creates a variable as a factor (categorical) variable. But it requires an input (Whatever comes inside tha parenthesis). Here we ask R to change the `num` variable `CKdata$STATE` into a factor variable.
- `levels(CKdata$STATEf) <- c("Pennsylvania","New Jersey")` then changes the names of the levels (categories) into sensible names. Here we use again the assignment operator `<-`.

Now create a new variable `CHAINf` which achieves the same for the `CHAIN` variable. Replace the `XXXX` to make this code work.

```
CKdata$XXXX <- as.factor(XXXX$XXXX)
XXXX(CKdata$CHAINf) XXXX c("XXXX","KFC", "XXXX", "Wendy's")
```

Confirm that you have added the new variables and that they are indeed of the factor type.

Or to avoid such a long output

Some summary statistics

Whenever you deal with real life data you should ensure that you understand the data characteristics. The easiest way to get a first impression of your data is to create some summary statistics

```
summary(CKdata$EMPFT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   2.000   6.000   8.203  12.000  60.000       6
```

We can see that, on average, the restaurants employed 8.2 full-time staff. The largest restaurant had 60 full-time staff members.

We can also look at several variables together. Try and run the command below. It has two mistakes. But before you try and fix it, run it as it is so that you can see the error messages and try and use them to figre out what is wrong.

You will constantly have to deal with error messages and that is totally normal. Once you see one, your inner Sherlok Holmes needs to come out!

```
sumary(CKdata[c("EMPPT","EMPPT2"))
```

Here we used a selection technique we've seen earlier, calling variables from a list (`c("EMPPT","EMPPT2")`). We learn that the average number of part-time employees hardly changed during 1992 (check the codebook to understand what these two variables are).

We could also look at the summary statistics for the `CHAIN` variable.

```
summary(CKdata$CHAIN)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   2.000   2.117   3.000   4.000
```

We know that the `CHAIN` variable tells us something about which restaurant we are looking at. This is a categorical variable and means and standard deviatiosn don't really make too much sense here, which is why we created the `CHAINf` variable.

When looking at categorical variables we are interested in the frequency or proportion of observations in each category.

```
table(CKdata$CHAINf)
```

```
##
## Burger King         KFC  Roy Rogers     Wendy's
##         171          80          99          60
```

Or the table with proportions.

```
prop.table(table(CKdata$CHAINf))
```

```
##
## Burger King         KFC  Roy Rogers     Wendy's
##   0.4170732   0.1951220   0.2414634   0.1463415
```

We fed the result of the `table` function straight into the `prop.table()` function which translates frequencies into proportions.

All these data are identical to those in the Card and Krueger paper.

Now we replicate some of the summary statistics in Table 2. We use the same functions as above, but now we feed two categorical variables into the `table` function. The addition of the `margin = 2` option ensures that proportions are calculated by state (2=columns). Try for yourself what changes if you either set `margin = 1` (1 for rows) or leave this option out.

```
prop.table(table(CKdata$CHAINf,CKdata$STATEf,dnn = c("Chain", "State")),margin = 2)
```

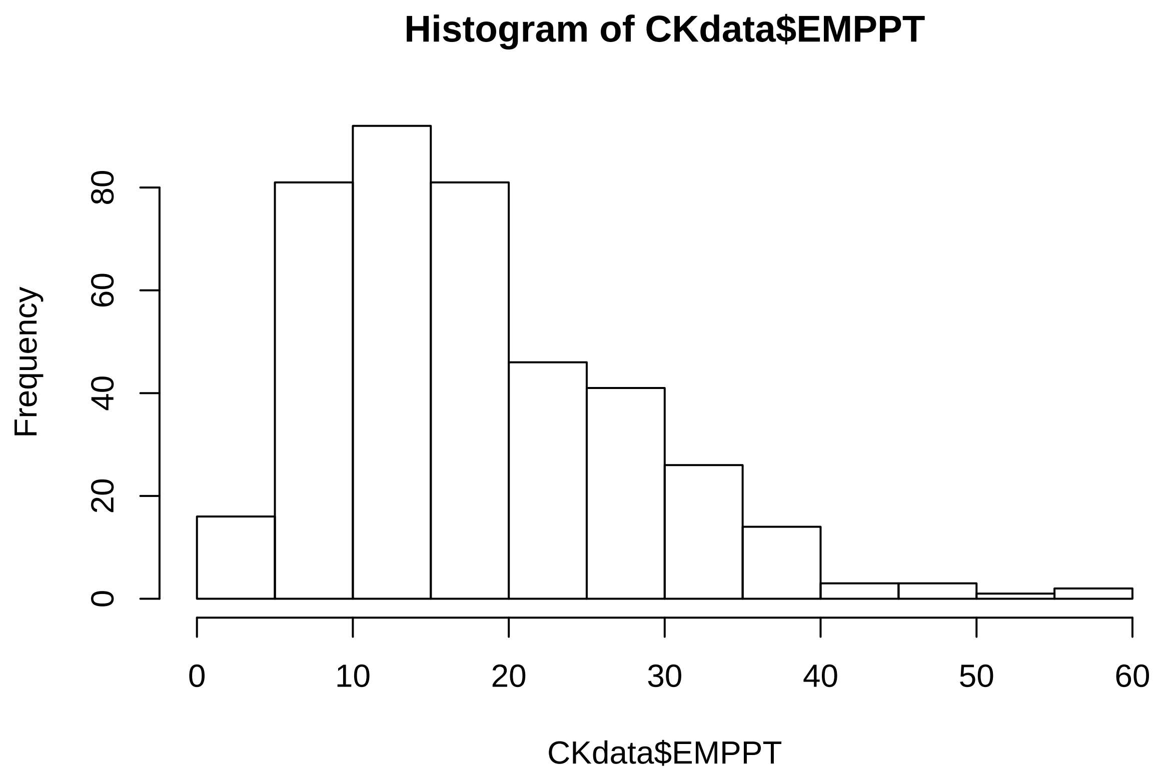
```
##              State
## Chain          Pennsylvania New Jersey
##   Burger King    0.4430380  0.4108761
##   KFC            0.1518987  0.2054381
##   Roy Rogers     0.2151899  0.2477341
##   Wendy's        0.1898734  0.1359517
```

Also google ("R table dnn") or use the help function (`?table`) to figure out what the `dnn` optional input into the table function achieves.

Graphical Data Representation

Histograms are an extremely useful representation of categorical data and numerical data. Here we will look at the distribution of PT employee numbers. The simplest way to create a simple histogram is by using the `hist` function.

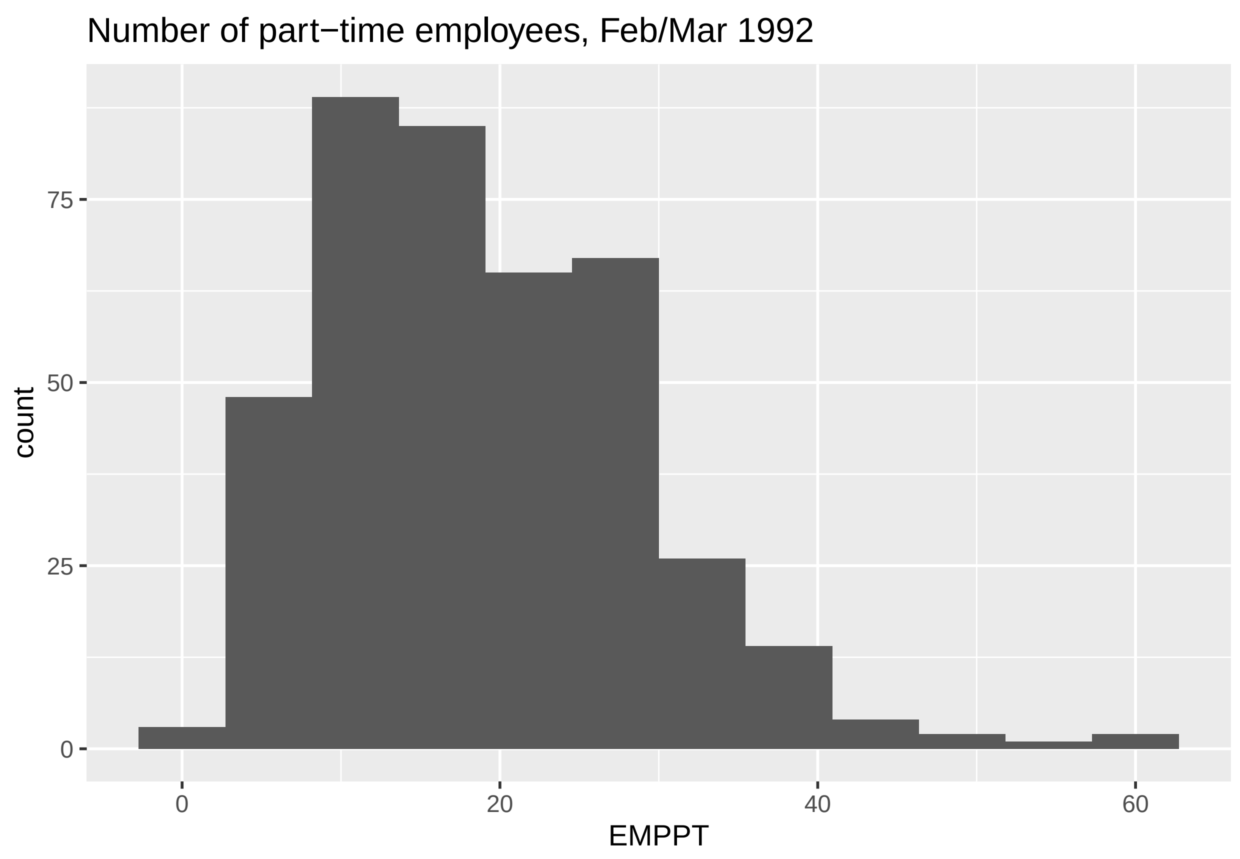
```
hist(CKdata$EMPPT)
```



You can clearly see the regularity in this plot.

Let's introduce a different way to produce these plots. The function we use here is the `ggplot` function. That function is incredibly flexible as we will see in a moment. And that added flexibility is potentially worth the extra complication. So let's check it out.

```
ggplot(CKdata,aes(x=EMPPT)) +
    geom_histogram(bins = 12) +
    ggtitle("Number of part-time employees, Feb/Mar 1992")
```



How this function works is as follows.

- Call the `ggplot` function
- Tell `ggplot` where it should get the data from, `ggplot(CKdata)`
- Then we set the aesthetics (`aes`), here we specify that we want `EMPPT` on the x-axis, `ggplot(CKdata,aes(x=EMPPT))`

At this stage no graph is plotted. All we have done is to tell R where to get the data from and what variable it should use on the x-axis. We are yet to tell R what type of graph we should produce for the `CKdata$EMPPT` data. We do this by

- adding the instruction to produce a histogram with 12 bins `+ geom_histogram(bins = 12)`

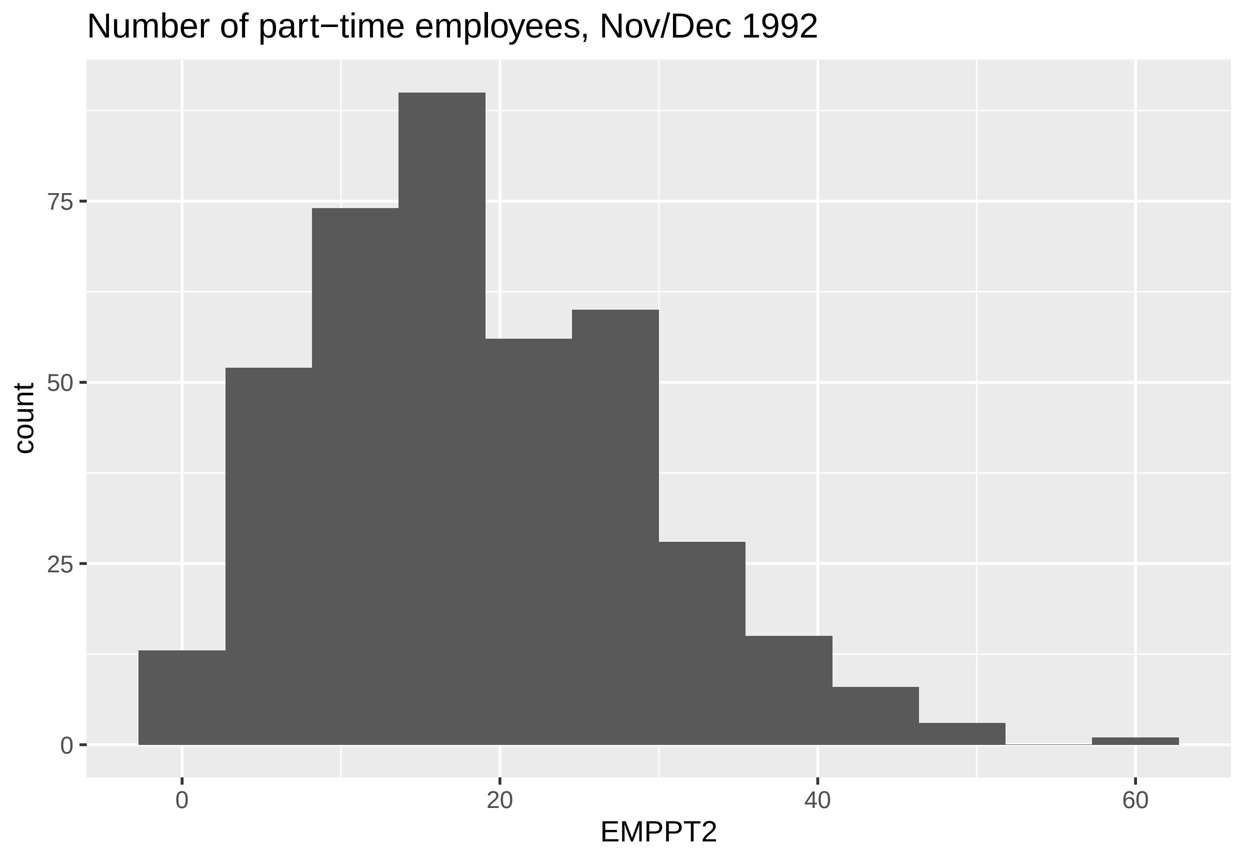
As we want a nice title we the also

- add the instruction to give our graph a useful title, `+ ggtitle("Number of part-time employees, Feb/Mar 1992")`.

One of the very useful features of the `ggplot` function is that you can keep adding extra flourishes in a similar manner.

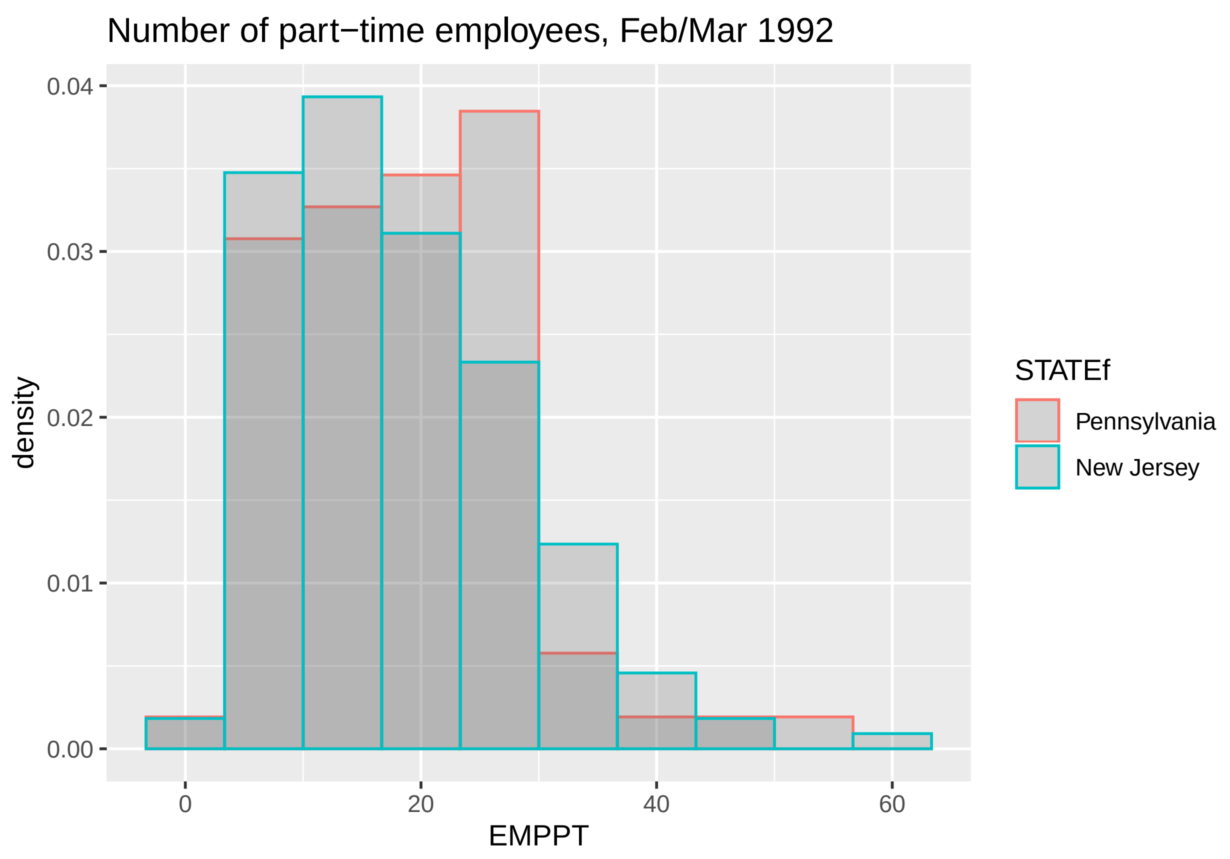
Now practice by creating a similar histogram for `EMPPT2`.

```
XXXX(XXXX,aes(x=XXXX)) +
    geom_XXXX(bins = XXXX) +
    ggtitle("Number of part-time employees, XXXX 1992")
```



To illustrate how powerful this function can be run the following command.

```
ggplot(CKdata,aes(EMPPT, colour = STATEf)) +
    geom_histogram(position="identity",
                   aes(y = ..density..),
                   bins = 10,
                   alpha = 0.2) +
    ggtitle(paste("Number of part-time employees, Feb/Mar 1992"))
```



So, what happened here. We took the part-time employee data but split them into the two states (using the `colour = STATEf` option in the aesthetics).

Not everything in this command is super intuitive. In fact you should expect that you need to google (something like "R ggplot histogram two variables") to find someone who achieved what you wanted and then you nick and adsjust their code! **This is super important. You will hardly ever need to know commands by heart. You can usually find help fairly easily.**

Try to change a few things in the above code to see what different elements in the code do. When you do this remember, you cannot break the computer!!!!!