

# ECON 1190: Econometrics 2:

## Slides 2: Getting started in R

Claire Duquennois

## Starting up in R

## Setting up your account

- ▶ Make an account on R Studio Cloud (aka Posit Cloud)
- ▶ Join the course work space by clicking the link in the canvas announcement
- ▶ upgrade your account to a student account for \$5/month to get access to more RAM
- ▶ If you want to follow along with the coding in class today:
  - ▶ Start the Slides2\_Started\_inR assignment
  - ▶ Open and download the Slides2\_chinks.txt file from canvas

# R Markdown

- ▶ We're going to do most of our work using R Markdown.
- ▶ Why it's nice
  - Run multiple lines of code at once.
  - But don't have to run all code at once.
  - Add comments between chunks of code.
  - Compile output into pretty documents.

All of my lecture slides and notes are made in R Markdown!

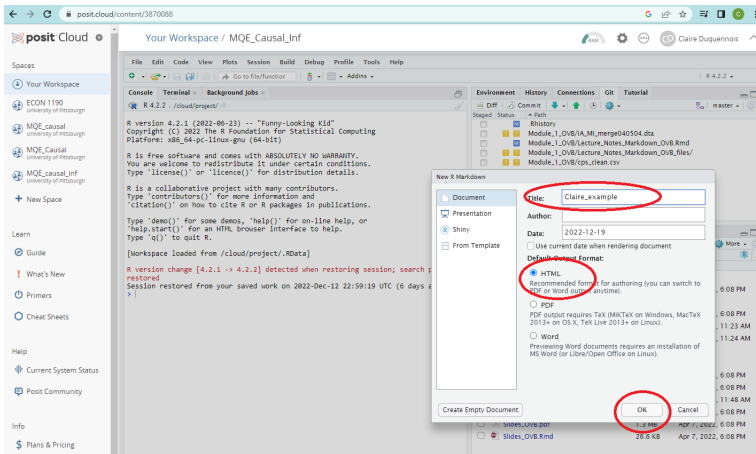
# R Markdown: Getting started

- Create file: File -> New File -> R Markdown...

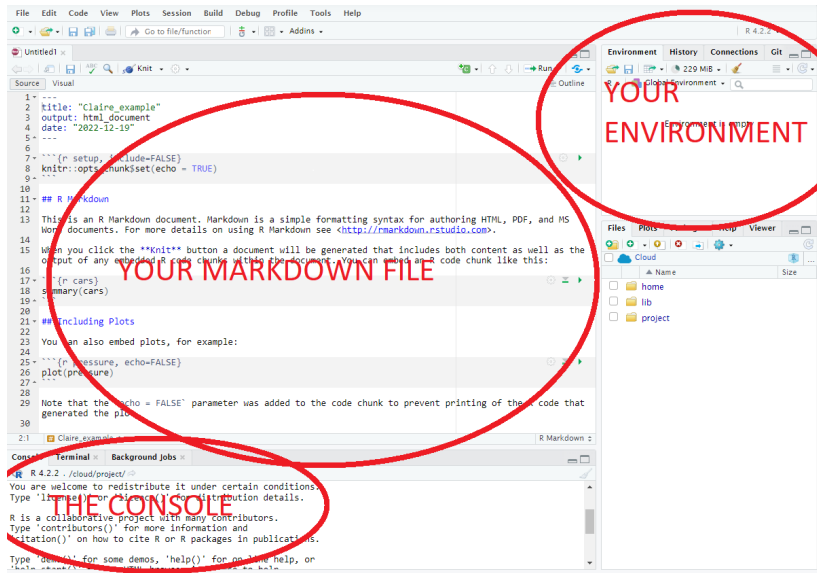
The screenshot displays the Posit Cloud web interface. On the left, the 'Your Workspace' sidebar lists several projects, including 'MQE\_causal\_inf'. The main area shows the 'File' menu open, with 'R Markdown...' selected and circled in red. The menu also includes options like 'R Script', 'Quarto Document...', 'R Notebook', 'Shiny Web App...', 'Plumber API...', 'Text File', 'C++ File', 'Python Script', 'SQL Script', 'Stan File', 'D3 Script', 'R Swave', 'R HTML', and 'R Documentation...'. The background shows a workspace named 'MQE\_Causal\_Inf' with a file explorer on the right listing files like '.Rhistory', 'code\_output', 'Code\_OVB.R', 'cps\_clean.csv', and 'IA\_MI\_merge040504.dta'.

# R Markdown: Getting started

- ▶ Give your file a title
- ▶ Set it to compile to PDF
- ▶ Click OK



# R Markdown: The workspace

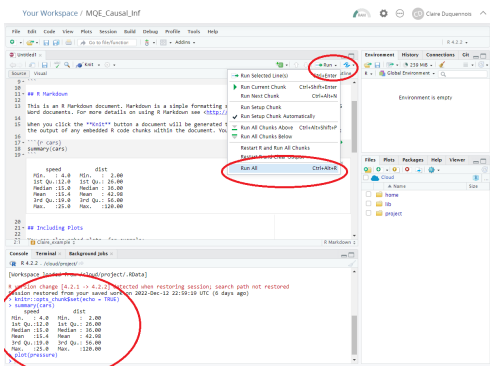


## R Markdown: The Console

When you run code, in the console you will see:

- the lines that were run
- called output
- errors and warnings

You can type commands directly into the console **BUT** these will not be saved





# R Markdown: Your R Markdown File

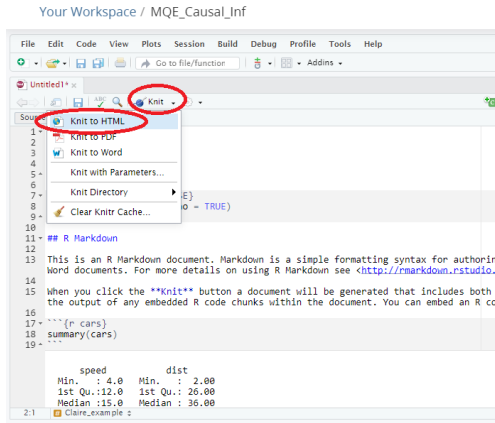
This is where the action is!

This file is a combination of:

- ▶ Markdown text (in the white parts)
- ▶ R code in the chunks (the grey parts)

When you “Knit” an R markdown file, you will produce a PDF document (or HTML or WORD...) that combines your text, code and code output.

# R Markdown: Your R Markdown File



And when prompted, save your PDF file to your workspace

# R Markdown: Your R Markdown File

Congrats! You now have a PDF document with the default Rmarkdown example code and text!

Let's move beyond the default. . .

# R Markdown File Elements

The screenshot displays the RStudio IDE with an R Markdown file named 'Claire\_example.Rmd' open. The interface is divided into several panes: a source editor on the left, a console at the bottom, and environment/plot/packages panes on the right. Red, green, and blue circles highlight specific elements in the source code, with corresponding text labels placed next to them.

**Annotations:**

- Markdown header:** Points to the header section of the document: `---  
title: "Claire_example"  
output: html_document  
date: "2022-12-19"  
---`
- R and knitr setup:** Points to the knitr setup code chunk: `{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)`
- Markdown Text:** Points to the main text block: `## R Markdown  
this is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.  
When you click the Knit button a document will be generated that includes both content as well as the output of all embedded R code chunks within the document. You can embed an R code chunk like this:`
- R code:** Points to the R code chunk: `## cars  
> summary(cars)`

**Console Output:**

```
R 4.2.2 . /cloud/project/ >  
[workspace loaded from /cloud/project/.RData]  
  
R version change [4.2.1 -> 4.2.2] detected when restoring session; search path not restored  
> knitr::opts_chunk$set(echo = TRUE)  
> summary(cars)  
      speed      dist  
Min.   : 4.0   Min.   : 2.00  
1st Qu.:12.0   1st Qu.: 26.00  
Median :15.0   Median : 36.00
```

**Environment Pane:** Shows the environment is empty.

**Files Pane:** Lists files in the project directory, including .gitignore, RData, .Rhistory, first\_day, GitHub instructions.Rmd, GitHub\_names.txt, GitHub-instructions.pdf, images, Module\_1\_OVG, Module\_2\_FE, Module\_3\_IV, Module\_4\_RCT, Module\_5\_DID, Module\_6\_RD, Module\_7\_Misc, project.Rproj, README.md, and Claire\_example.Rmd.

# R Markdown: Edit the markdown text

- ▶ Edit the Markdown text
- ▶ Re-knit your document and see your changes

Your Workspace / MQE\_Causal\_Inf

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Claire\_example.Rmd x

Knit Run

```
1 ---
2 title: "Claire_example"
3 output: html document
4 date: "2022-12-19"
5 ---
6
7 {r setup, include=FALSE}
8 knitr::opts_chunk$set(echo = TRUE)
9
10 ## Claire's DOC
11 My writing
12
13 {r cars}
14 summary(cars)
15
16 ## Claire's Plots
17 Isn't this pretty!
18
19 {r pressure, echo=FALSE}
20 plot(pressure)
21
22
23
24
25
26
27
```

Environment History

R Global Environment

Files Plots Packages

Cloud project

Name

ignore

.RData

.Rhistory

first\_day

GitHub instructions

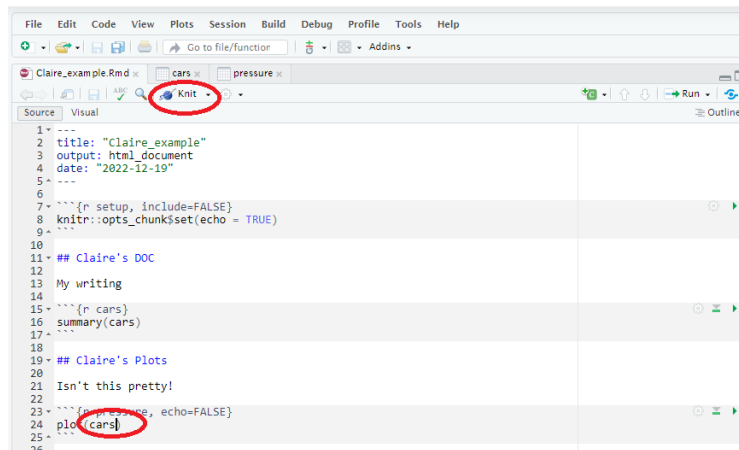
GitHub name

# R Markdown: Edit the R code

Why are we summarizing the data in the cars data set but plotting the data in the pressure data set?

Let's fix that. Replace the word pressure with cars in the `plot()` function and re-knit your document

Your Workspace / MQE\_Causal\_Inf



```
1 ---
2 title: "Claire_example"
3 output: html_document
4 date: "2022-12-19"
5 ---
6
7 ```{r setup, include=FALSE}
8 knitr::opts_chunk$set(echo = TRUE)
9 ```
10
11 ## Claire's DOC
12
13 My writing
14
15 ```{r cars}
16 summary(cars)
17 ```
18
19 ## Claire's Plots
20
21 Isn't this pretty!
22
23 ```{r pressure, echo=FALSE}
24 plot(cars)
25 ```
26
```

## R Markdown: Edit the R code

Congrats! Your PDF file should now show a new scatterplot of the cars data.

## R Markdown: Code “chunks”

- ▶ Always write code in “grey” regions called chunks
- ▶ Creating a new chunk
  - Green +C box
  - option-command-i (Mac)
  - control-alt-i (Windows)
- ▶ the first line of the chunk, in brackets is the chunk header



# R Markdown: Chunk headers

The UNIQUE chunk name:

- ▶ the 1st element (before first comma)
- ▶ If you use the same name for multiple chunks your file will not knit
- ▶ Ex: try replacing *r pressure* with *r cars* in the third chunk: your file will not knit

## R Markdown: Chunk headers

Other chunk options that determine how your code shows up in your document (separate multiple options with commas):

- ▶ `include = FALSE` prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
- ▶ `echo = FALSE` prevents code, but not the results from appearing in the finished file. This is useful to embed figures.
- ▶ `message = FALSE` prevents messages that are generated by code from appearing in the finished file.
- ▶ `warning = FALSE` prevents warnings that are generated by code from appearing in the finished file.

Notice: the line `knitr::opts_chunk$set(echo = TRUE)` is setting the default for all the chunks to have `echo=TRUE`.

## R Markdown: Code “chunks”

### ► Execute code

- Green arrow in upper right of chunk.
- `command-shift-enter` (Mac) [click inside chunk first]
- `control-shift-enter` (Windows) [click inside chunk first]
- Let's run some code!

# R Code

Create a chunk and type the following:

```
x <- 2 # Assign the value 2 to the variable x
y <- 3 # Assign the value 3 to the variable y
z <- x + y # Add x and y
z
```

```
## [1] 5
```

- ▶ The operator “<-” is how we assign the value to a variable. The value could be a number but also a list, string, matrix, etc.
- ▶ To add comments, put a “#” before comment. Comments appear green. R ignores these when it executes the code.
- ▶ R printed 5 under the chunk. If you perform an operation and don't store the result as a variable, R will print it.
- ▶ Note that the code and output was printed in the Console.
- ▶ The values you created x, y, z now appear in your Environment.

# R Packages

- ▶ A package is a set of functions/commands which have already been programmed.
- ▶ To use a package you must load it from the library: use `library()` function.

```
library(stats) #loading the stats package
```

- ▶ If a package is not in the library it is not yet installed. To install packages use the `install.packages()` command.

```
#install.packages("ggplot2")
```

- ▶ We can now load `ggplot2` from the library when we want to use it.

# Loading data

Let's load the olympics\_data.csv data:

- ▶ this is already in the RStudio cloud workspace
- ▶ it is also on the course canvas page
- ▶ this is a csv (Comma Separated Vales) file, to load it we need the function read.csv

```
olympics <- read.csv("olympics_data.csv")
```

```
View(olympics) #opens the data in the data viewer
```

```
head(olympics,5) #prints the first 5 rows of data in the console (or knitr doc)
```

```
##      country country_abbrev continent year  type gold silver bronze
## 1 Afghanistan          AFG      Asia 2000 summer    0      0      0
## 2 Afghanistan          AFG      Asia 2002 winter    0      0      0
## 3 Afghanistan          AFG      Asia 2004 summer    0      0      0
## 4 Afghanistan          AFG      Asia 2006 winter    0      0      0
## 5 Afghanistan          AFG      Asia 2008 summer    0      0      1
##   population      gdp g8
## 1   21.60699      NA  0
## 2   22.60077  7.228792  0
## 3   24.72669  7.978512  0
## 4   26.43306  9.349917  0
## 5   27.72228 11.060389  0
```

# Data Management with Dplyr

You're going to be using a lot of data. Dplyr makes working with data less overwhelming.

- ▶ Dplyr is a data management package.
- ▶ Dplyr allows us to apply multiple transformations to our data at once using the pipe operator: `%<%`
- ▶ First let's load the dplyr package (we installed it earlier)

```
# Load Dplyr  
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
## filter, lag
```

```
## The following objects are masked from 'package:base':
```

```
##
```

```
## intersect, setdiff, setequal, union
```

# Selecting Data

When working with a large data set, it can be helpful to select the columns you will use.

Create a new dataset `olympics2` that only contains the columns we will use (drops `country_abbrev`, `continent`, `g8`)

```
olympics2<-olympics %>% select(country, year, type, gold, silver, bronze, population, gdp)  
head(olympics2)
```

##	country	year	type	gold	silver	bronze	population	gdp
## 1	Afghanistan	2000	summer	0	0	0	21.60699	NA
## 2	Afghanistan	2002	winter	0	0	0	22.60077	7.228792
## 3	Afghanistan	2004	summer	0	0	0	24.72669	7.978512
## 4	Afghanistan	2006	winter	0	0	0	26.43306	9.349917
## 5	Afghanistan	2008	summer	0	0	1	27.72228	11.060389
## 6	Afghanistan	2010	winter	0	0	0	29.18551	15.354605

- ▶ First we write the data set name, then the pipe operator (`%>%`), and then the `select()` function.
- ▶ In the `select()` function I list the variables I want to keep (... there are other ways to do this)
- ▶ this smaller data is saved as `olympics2` in the environment



# Filtering Data

You may want to limit your data to certain rows.

Create a new dataset `us_olympics` that only contains observations for the United States, using the `filter` function

```
us_olympics<-olympics2 %>% filter(country == "United States")  
head(us_olympics)
```

```
##      country year  type gold silver bronze population      gdp  
## 1 United States 2000 summer   37     24     32   284.9690 13754.30  
## 2 United States 2002 winter   10     13     11   287.6252 14121.05  
## 3 United States 2004 summer   36     39     26   292.8053 15075.14  
## 4 United States 2006 winter    9      9      7   298.3799 16034.37  
## 5 United States 2008 summer   36     39     37   304.0940 16376.73  
## 6 United States 2010 winter    9     15     13   309.3271 16383.04
```

- ▶ First we write the data set name, then the pipe operator (`%>%`), and then the `filter()` function.
- ▶ The `filter()` function takes a logical statement. In this case `country == "United States"`. `country` is the variable we are filtering on, `==` is a logical operator, `"United States"` is a value.

# Variable types

## String variable:

- ▶ Country is a string variable: to select rows we write "United States" in quotation marks
- ▶ Main logical operators: == (is equal to) and != (is not equal to)

## Numeric variables:

- ▶ Year is a numeric variable: no need for quotation marks.
  - ▶ With numeric variables we can use additional logical operators
    - ▶ < Less than and > Greater than
    - ▶ <= Less than or equal to and >= Greater than or equal to
    - ▶ %in% is in a particular vector (more about vectors later)

## Factor variables:

- ▶ variable is a set of numeric codes
- ▶ the numeric value does not matter, it just defines a category

# Filtering Data

## Filter the United States at Summer Olympics since 2010

```
uss10_olympics<-olympics2 %>% filter(country == "United States" & type == "summer" & year >= 2010)  
head(uss10_olympics)
```

```
##           country year   type gold silver bronze population      gdp  
## 1 United States 2012 summer   47    27    30   313.8777 17016.39  
## 2 United States 2016 summer   46    37    38   323.0718 18509.60  
## 3 United States 2020 summer   39    41    33   331.5011 19247.06
```

- ▶ We can combine multiple conditions using the & operator.
- ▶ If we want to use or, we can use | (that is a vertical line not a capital I)

# Filtering Data

Filter observations with more than 10 silver medals or more than 4 gold medals.

```
olympics_SG<-olympics2 %>% filter(silver > 10 | gold > 4)  
head(olympics_SG)
```

##	country	year	type	gold	silver	bronze	population	gdp
## 1	Australia	2000	summer	16	25	17	19.41300	872.5868
## 2	Australia	2004	summer	17	16	17	20.12740	995.0810
## 3	Australia	2008	summer	14	15	17	21.24920	1133.4026
## 4	Australia	2012	summer	8	15	12	22.73346	1256.1287
## 5	Australia	2016	summer	8	11	10	24.19091	1387.5610
## 6	Australia	2020	summer	17	7	22	25.69327	1490.9678

# Filtering Data

Filter observations from Argentina, Colombia, Brazil, Mexico.

```
olympics_LA <- olympics2 %>% filter(country %in% c("Argentina", "Colombia", "Brazil", "Mexico"))
```

- ▶ Use %in% operator
- ▶ c( , , , ) defines a vector
- ▶ we are telling R to keep observations where the variable country is the same as one of the strings in the given vector

# Missing values

- ▶ Missing values in R are indicated with NA
- ▶ Being aware of missing values in a variable is important since they tend to mess things up. We'll talk more about this later.

We can filter on rows where none of the population values of a variable are missing.

```
olympics_nomiss <- olympics2 %>% filter(!is.na(population))  
head(olympics_nomiss)
```

##	country	year	type	gold	silver	bronze	population	gdp
## 1	Afghanistan	2000	summer	0	0	0	21.60699	NA
## 2	Afghanistan	2002	winter	0	0	0	22.60077	7.228792
## 3	Afghanistan	2004	summer	0	0	0	24.72669	7.978512
## 4	Afghanistan	2006	winter	0	0	0	26.43306	9.349917
## 5	Afghanistan	2008	summer	0	0	1	27.72228	11.060389
## 6	Afghanistan	2010	winter	0	0	0	29.18551	15.354605

# Create New Variables

Create a new variable called total that is the sum of all medals

*#method 1: use the mutate function in dplyr*

```
olympics2 <- olympics2 %>% mutate(total = gold + silver + bronze)
```

*#method 2: using base R*

```
olympics2$total2<-olympics2$gold+olympics2$silver+olympics2$bronze  
head(olympics2)
```

##	country	year	type	gold	silver	bronze	population	gdp	total	total2
## 1	Afghanistan	2000	summer	0	0	0	21.60699	NA	0	0
## 2	Afghanistan	2002	winter	0	0	0	22.60077	7.228792	0	0
## 3	Afghanistan	2004	summer	0	0	0	24.72669	7.978512	0	0
## 4	Afghanistan	2006	winter	0	0	0	26.43306	9.349917	0	0
## 5	Afghanistan	2008	summer	0	0	1	27.72228	11.060389	1	1
## 6	Afghanistan	2010	winter	0	0	0	29.18551	15.354605	0	0

- ▶ I store my output using the same name. This overwrites the original data set.
- ▶ Here I use only a single = sign. This is not a logical statement but rather a formula.
- ▶ to call a variable in base R: dataset\_name\$variable\_name

# Summarize Data

To summarize data, we combine `group_by()` and `summarize()`.

Get the total count of medals by country and season for all years

```
olympics_totals<-olympics2 %>% group_by(country, type) %>% summarize(total_medals = sum(total))  
head(olympics_totals)
```

```
## # A tibble: 6 x 3  
## # Groups:   country [3]  
##   country    type total_medals  
##   <chr>      <chr>         <dbl>  
## 1 Afghanistan summer          2  
## 2 Afghanistan winter          0  
## 3 Algeria     summer         10  
## 4 Algeria     winter          0  
## 5 Argentina   summer         27  
## 6 Argentina   winter          0
```

- ▶ We divided the data into groups and got the sum of each group.
- ▶ The groups are defined by country and season so we write `group_by(country, type)`.
- ▶ `summarize` took the sum of the variable `total` and stored it in a variable called `total_medals`.
- ▶ Other `summarize` operations: `mean()`, `median()`, `sd()`, `min()`, `max()`...



# Multiple operations

Now let's put it all together. Calculate a country's average score at the summer olympics.

```
olympics_sumavg<-olympics2 %>% filter(type == "summer")%>%  
  group_by(country) %>%  
  summarize(avg_total = mean(total))  
  
head(olympics_sumavg)
```

```
## # A tibble: 6 x 2  
##   country      avg_total  
##   <chr>         <dbl>  
## 1 Afghanistan  0.333  
## 2 Algeria      1.67  
## 3 Argentina    4.5  
## 4 Armenia      2.67  
## 5 Australia    44  
## 6 Austria      3.5
```

This is the beauty of dplyr.

# GGplot2

A great tool to visualize, and learn about, your data

First we will need to load ggplot

```
# Install GGplot2  
# install.packages("ggplot2")
```

```
# Load GGplot2  
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1
```

## Step 1: Prepare the data

Before working with ggplot, you need to prepare your data for graphing.

- Is there a relationship between population and medals in the summer olympics?

```
olympics_plot1<-olympics2 %>% filter(type == "summer" & !is.na(population))  
  
head(olympics_plot1)
```

##	country	year	type	gold	silver	bronze	population	gdp	total	total2
## 1	Afghanistan	2000	summer	0	0	0	21.60699	NA	0	0
## 2	Afghanistan	2004	summer	0	0	0	24.72669	7.978512	0	0
## 3	Afghanistan	2008	summer	0	0	1	27.72228	11.060389	1	1
## 4	Afghanistan	2012	summer	0	0	1	31.16138	17.386481	1	1
## 5	Afghanistan	2016	summer	0	0	0	35.38303	19.566705	0	0
## 6	Afghanistan	2020	summer	0	0	0	38.92834	20.621946	0	0

## Step 2: Understand your data

It is wise to look at the summary statistics of variables.

```
summary(olympics_plot1$population)
```

```
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
##    0.0278    3.7227   10.3255   49.5113   34.7306  1411.1000
```

```
sd(olympics_plot1$population)
```

```
## [1] 167.3171
```

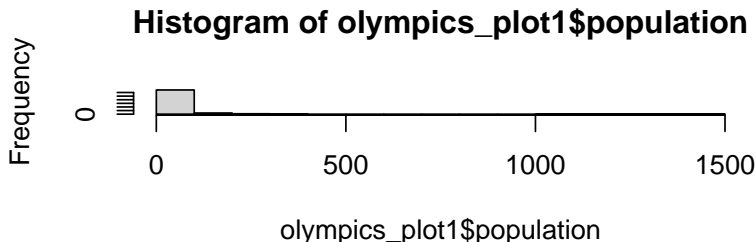
## Step 2: Understand your data

Histograms can be particularly informative to get a sense of your data.

► Here I use a simple command but could also use ggplot

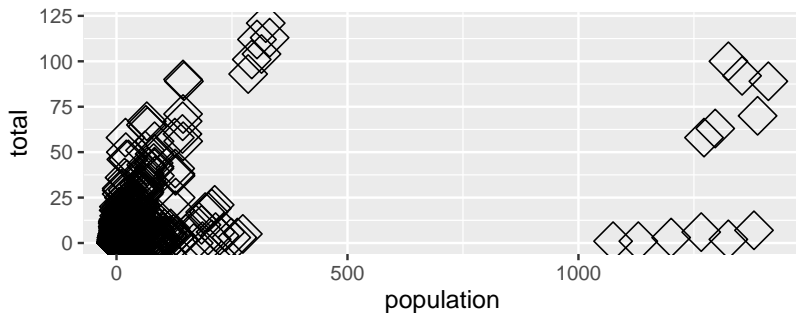
Looks like there are some big outliers. . .

```
hist(olympics_plot1$population)
```



## Step 3: Build your figure

```
my_plot1<-ggplot(data = olympics_plot1, aes(x = population, y = total))+  
  geom_point(size=6, shape=23)  
my_plot1
```



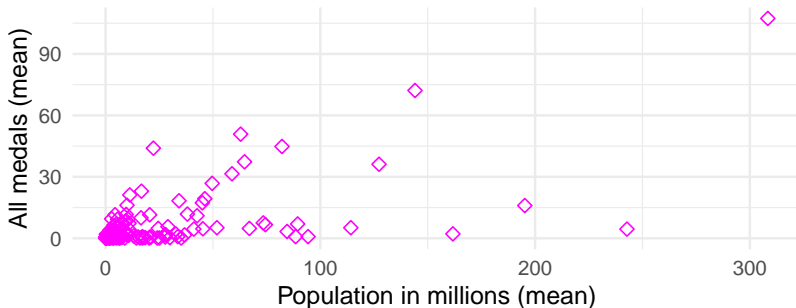
## Step 5: Critiquing our figure

Not bad BUT...

- ▶ kind of ugly
- ▶ can't see much where the action is (bottom left)
- ▶ have multiple observations for each country. Maybe averages over this time period would be better?

## Step 6: Improve your figure

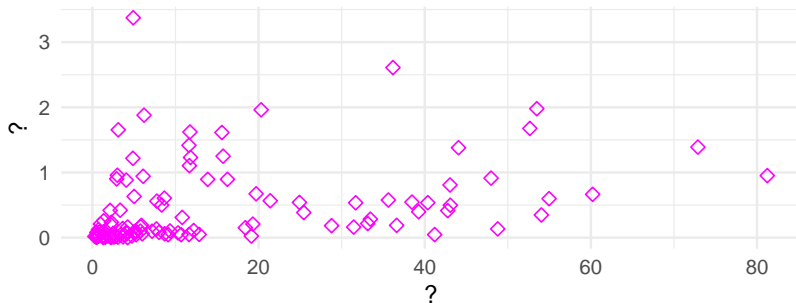
```
olympics_plot2<-olympics2 %>%  
  filter(type == "summer" & !is.na(population)& population<400)%>%  
  group_by(country)%>%  
  mutate(total = gold + silver + bronze) %>%  
  summarize(avg_total = mean(total), avg_pop=mean(population))  
  
my_plot2<-ggplot(data = olympics_plot2, aes(x =avg_pop, y = avg_total))+  
  geom_point(size=2,shape=23, color="magenta1")+  
  labs(x = "Population in millions (mean)", y = "All medals (mean)")+  
  theme_minimal()  
my_plot2
```





## A more interesting figure

```
olympics_plot3<-olympics2 %>%  
  filter(type == "summer" & !is.na(population)& !is.na(gdp))%>%  
  group_by(country)%>%  
  mutate(total = gold + silver + bronze) %>%  
  summarize(avg_total = mean(total), avg_pop=mean(population), avg_gdp=mean(gdp))%>%  
  mutate(avg_total_cap=avg_total/avg_pop, avg_gdp_cap=avg_gdp/avg_pop)%>%  
  filter(avg_pop>=1)  
  
my_plot3<-ggplot(data = olympics_plot3, aes(x =avg_gdp_cap, y =avg_total_cap ))+  
  geom_point(size=2,shape=23, color="magenta1")+  
  labs(x = "?", y = "?")+  
  theme_minimal()  
my_plot3
```



## Top Hat question

How should I label the axes?

⇒ Top Hat

# GGplot basics

- ▶ Always prepare your data before making a graph.
- ▶ We start a ggplot with the `ggplot()` function, typically with two arguments:
  - ▶ `data`: this is the name of your dataset
  - ▶ `aes`: this controls how your data relates to the graph.
    - ▶ `x`: the variable on the x-axis
    - ▶ `y`: the variable on the y-axis
- ▶ `ggplot()` only creates the chart area. To add layers we use the `+` operator. Here we added a
- ▶ `geom_point()` layer (the dots)
- ▶ `labs()` layer (defines the axis labels)
- ▶ the `theme_minimal()` layer (sets the background colors/format)

There are many (MANY!) options with ggplot with lots (LOTS!) of online example to help you make great figures

# Regressions in R

- ▶ Can use the `lm()` function
- ▶ for more advanced regressions we will be running use `felm()` which is part of the `lfe` package (same syntax)

```
reg<-felm(avg_total_cap~avg_gdp_cap, olympics_plot3)
summary(reg)
```

```
##
## Call:
##   felm(formula = avg_total_cap ~ avg_gdp_cap, data = olympics_plot3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.71306 -0.30524 -0.25759  0.09904  3.02881
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.289600   0.073164   3.958 0.000135 ***
## avg_gdp_cap  0.011413   0.003102   3.680 0.000365 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5836 on 109 degrees of freedom
## Multiple R-squared(full model): 0.1105   Adjusted R-squared: 0.1023
## Multiple R-squared(proj model): 0.1105   Adjusted R-squared: 0.1023
## F-statistic(full model):13.54 on 1 and 109 DF, p-value: 0.0003647
## F-statistic(proj model): 13.54 on 1 and 109 DF, p-value: 0.0003647
```

# Stargazer

Lets make the regression results more presentable.

```
reg<-felm(avg_total_cap~avg_gdp_cap, olympics_plot3)
stargazer(reg, type = "latex", header=FALSE)
```

Table 1:

	<i>Dependent variable:</i>
	avg_total_cap
avg_gdp_cap	0.011*** (0.003)
Constant	0.290*** (0.073)
Observations	111
R <sup>2</sup>	0.110
Adjusted R <sup>2</sup>	0.102
Residual Std. Error	0.584 (df = 109)
Note:	* p<0.1; ** p<0.05; *** p<0.01

# Stargazer

Stargazer is like the table version of ggplot.

- ▶ It will help you generate nice presentable table
- ▶ has lots (LOTS!) of options to customize your presentation
- ▶ for the table to render you need to add the chunk option:  
`results='asis'` in the chunk header
- ▶ you should specify output type “latex”
- ▶ there are lots of example online of how to format stargazer tables.
- ▶ you might need to install a latex editor  
(<https://miktex.org/download>)

# R resources for troubleshooting

Help function:

- ▶ The `help()` function will bring up the documentation for a given function
- ▶ Let's look at the documentation for `rnorm`

```
#help(rnorm)
```

Web search:

- ▶ Half of coding is knowing how to Google your questions.
- ▶ **stackoverflow.com**: A particularly helpful website

GGplot image web search:

- ▶ Describe the kind of figure you want +ggplot and look at image results. Sometimes the code for the image is given
- ▶ Lots of tutorial websites on ggplot visualizations