- The data is Public Domain and available from my webpage.

```
Pearson <-
read.table("https://users.pfw.edu/yorgovd/IntroR/
Pearson.txt") # will not work
# There is a header; tab delimited…
```

- Recall, scatterplots are a great way to explore relationships between a single response and a single predictor variable (if both are *quantitative* variables).

- We are interested in the inheritance of heights from fathers to sons.
- Therefore:
  - the father's height (*Father* column in the dataset) is used as the predictor variable (input).

    and
  - the son's height (*Son* column in the dataset) is used as the response variable (output).
- Let's create a scatterplot.
  - Does there appear to be a linear relationship between the predictor and the response?

- Let's proceed with fitting the linear model.
  - Now, using the derived model parameters, let's overlay a line representing the fitted mean function.
- Let's consider the "results".
  - How strong is the relationship?
  - The parameters here are two numbers(coefficients), intercept, and slope.
  - What is the p-value of the slope coefficient?
    - The p-value is the probability to observe the data under if the actual slope coefficient $\boldsymbol{\beta_1 = 0}$.
      - Did we just stare in the noise? Compare p-value to 5% chance typically.

- The single-number (point) estimates of the parameters are indeed noisy.
  - A confidence interval gives a "margin of error" around the estimates…
- One must study if the residuals are approximately normally distributed and more…
  - Some will be covered later in the semester.

18. Importing and Exporting Data

- Recall that we could save all the objects in the memory (environment) when we exit or when we explicitly execute `save.image()` function.
  - The "`.RData`" file is updated in the current (working) directory.

- You can also save just some of the local objects/variables with the `save()` function.
  - The format is **an internal to R format** that changes between major versions; newer versions can read older formats but you need to specify the specific R version.

  - To be consistent, I suggest using a filename with extension `.RData` too, for instance `test.RData`

```
A<-44:89; b<-"R is great"
save(A,b,file="test.RData")
```

  - Note that these variables will not be automatically loaded when you start your R session.
    - R automatically loads only the "`.RData`" file in the default working directory only.
    - You must use the `load()` function to load the file.

```
rm(A,b) # delete the two variables
load("test.RData")  # reload the two
```


***

- You could also call an `.R` script files from within R with the **`source()` function**.
  - Content of files accessed by `source()` is executed as R commands in the current R session.
  - Note that if an error is encountered, the `source()` function will halt and just stop without a message (it will not execute the rest of the commands). Provide an **`echo=T`** argument to **see any issues or errors**.
    - No warnings will print without **`echo=T`** either.

  - `source()` is useful for instance when debugging the scripts that you run remotely in the Cloud or on a supercomputer via a terminal connection.
  - You could also keep some pre-processing steps in an ".R" file that you source when you need, e.g. the `cleanPima.R` that we have.
  - We learned how to write our own functions in R. You could keep some user-defined functions in ".R" files that you call/source with the `source()` function only when you need them.
    - This is easier and faster than building your own package (which you could also do if needed).

- Let us source our `cleanPima.R` file that we created.

```
source("cleanPima.R")
```

- Note that the above command immediately started executing; the dataset was loaded and cleaned but nothing was printed out in the console.
  - In this case, just new object was created in the memory.
  - To see the console commands, provide the option:

```
source("cleanPima.R", echo=T)
```


- The `source()` command accepts file names and web addresses (URLs) too.
  - You can call and execute the .R file directly from its web location.
- You used my version of `cleanPima.R` online at:

https://users.pfw.edu/yorgovd/IntroR/CleanPima.R

  - Remove the `pima` data frame from the memory.
  - Source the file from the web to create a new instance.

  - Next, let us create a histogram and a density plot for the glucose level recorded.

- We created an `MPG.R` file with the code previously, creating the "fancy" MPG plot.

```
source("MPG.R", echo=T)
```

Something is not good… TRUNCATED…
Let us try again…

***

- Besides using the export image dialog box, you can also direct an image to be saved to a file format of your choice with a command (function).
  - First, you open a "graphic device" file connection.
  - Next, you plot with one or more commands.
  - Third, you close the graphic device with `dev.off()`
- Some of the built-in formats you can save to are *BMP*, *JPEG*, *PNG*, *TIFF*, *PostScript*, and *PDF* graphics devices with R functions `bmp()`, `jpeg()`, `png()`, `tiff()`, `postscript()`, and `pdf()`.
  - Many options are available that give you a lot of control of your exported graphics.

- Let us save the MPG vs Displacement plot produced with `ggplot2` to a PNG file.
  - PNG stands for *Portable Network Graphics.*
  - We will just write in the current directory(folder). Make sure you have write permissions.

```
png(filename="MPG.png")
# source the old code
# here
dev.off()
```

  - Of course, if you are sourcing a file, you need the file with the actual code to modify any options for the graphic.

- Many options are available…
- You could change the resolution, the color from *yellow* to *lemonchiffon* color, etc.
  - If you want to dive-in R color palettes topics, start from the help first:
- `?color`

http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf

***

- How about importing/exporting external data (not in the internal R file format)?
- The `read.table()` function that we already covered allows for a fast and easy way to import the so called
  - *flat files: text files containing data (typically) in rows and columns.*
  - Usually, each row is an observation. Columns are different variables.

- Tiny functions exist in R that call `read.table()` with default arguments pre-set for different type of files like `read.csv()`, `read.csv2()`, `read.delim()`, etc.
  - You can use these to avoid the need to figure out the arguments.
  - Sometimes these files are labeled one way but the actual file format differ from the defaults in the functions.
- Let us specifically review all options to import the csv file provided (Refer to Lab 2 for details).

Bring on a dialog box to open the file:

```
dat <- read.table(file = file.choose(), header = TRUE, sep
= "," )
```

```
file_path <- "c:/buff/Practicum/R_lab6.csv"
```

Or explicitly provide the path to the location on the local device:

```
dat <- read.table(file= file_path, header = T, sep =
"," )
```

Or change the working directory to the location of the file:

```
setwd(file_path)
dat <- read.table(file="R_lab6.csv",header=T,sep=",")
```

Or download directly from the web:

```
dat <-
read.table("https://users.pfw.edu/yorgovd/IntroR/R_la
b6.csv", header = TRUE, sep = "," )
```


Or use the "shortcut `read.csv()`:

```
dat <-
read.csv("https://users.pfw.edu/yorgovd/IntroR/R_lab6
.csv")
```

- Again, flat files are very easy to handle with different software packages and very fast to access directly in Linux/Windows/Mac Terminal.
- Importantly, most software packages / data repositories will allow you to export to a *flat text files* that you could then import into R.

- Many huge data sets are actually *flat files* that are additionally compressed to save space (very often gzipped).
  - For MAC/Linux gzip is available in terminal.
    - If you use gzipped files, you should install and use `pigz` as it uses all cores in your machine's CPU(s).
    - For PC you need to install a Linux file subsystem and a Linux distribution to access such files.
    - Alternatively, you could download 7-zip, a free and open-source file archiver.

- Our examples are small, but you can deal with big and huge files in R too.
  - Some files are so big you would need to use external R packages that do not load all the data in the memory.
  - For some projects, you do not even save or store the data as it is a prohibitively large stream.
    - You process the data stream, and you update your models. If you want to use the term *big data*, perhaps that would be the correct place to do nowadays.

- Again, for more “modest” file sizes, say a few Gigabytes, you can automatically import compressed, gzipped flat files, with file extension `.gz`.
  - R will automatically recognize files with an extension `.gz` and decompress when loading.
  - In a Linux/Mac terminal you can directly use the .gz flat files with bash and/or AWK scripts, etc.
  - Much of the data in my research field, statistical genetics, has been **flat files** (benefit: simple format that everybody can use) that are **compressed** (to save storage space ant to to speed up Disk I/O).

- To write a data frame or a matrix to a flat file format use `write.table()` or the child functions `write.csv()` and `write2.csv()`.
  - Let us write the `iris` data frame to a CSV file.
    - If you will share your files, avoid names with spaces. Use dots instead, e.g., "`this.is.a.file.csv`", or capitalizations, e.g., "`ThisIsaFile.csv`", or underscores, e.g., "`this_is_a_file.csv`"

```
write.csv(iris,"iris.csv")
```

    - Let’s check the content of the file in MS Excel.
    - Let’s check the content of the file in the terminal.

- Using the `row.names=F` option will remove the row labels or indices.

- In the next example, let us create a big CSV file and save it.
  - Let’s create a medium data set, 150MB.
  - Start small first.

```
n<-250 # will change to n=10^8 for 0.1GB
a<-sample(1:n,n) # permute the first n int.
```


- Create data frame with `a` but split in two columns and a third column of all 1s.
  - Let’s write this data frame to a file `temp.csv`.
    - Remove the row numbers.
  - Let’s try with a bigger n.
    - As the data frame size increases, saving the data frame to a file could take longer.
      - In R, you could break the execution with CTRL-C or CMND-C if you need to.
      - RStudio provides a stop button too.
      - The interruption might not be immediate.
  - Let’s try again, timing the execution of the command.