The objective of this section is to review the syllabus and to introduce the `R` environment by way of an example that will (hopefully) help you with the first problem set. If there is remaining time, I'll work through some basic code puzzles that will require you to work in `R`. The first two or three sections will feel slow for those of you with substantial experience in `R`[1], but I promise we'll speed up soon.

Installing `R`

Download `R`: The download of `R` will vary by operating system, but it will begin here in any event:

`cran.r-project.org`

The online documentation and installer routines are comprehensive. If you are new to `R`, then it might make sense to use the Mac or Windows distribution, along with the built-in editor to write and evaluate code. `Rstudio` is a popular IDE that provides a somewhat more user-friendly interface than the base `R` installation. For the tech-oriented, the Linux distribution is very flexible; I use Emacs with the ESS package for editing. If you are interested in using the Linux distribution and are having trouble with the setup, please see me.

Learning `R`

I have included links in the syllabus to a few of the many resources on the web that provide gentle introductions to the `R` language. Those of you who have no experience with `R` or with programming in general will find it well worth your time to spend a few hours browsing these, in particular the starter resources for `R` from the UCLA Statistics department. In section I will focus on presenting examples of code piece-by-piece in order to illustrate certain concepts. My intention is to expose you to a small part of the `R` language in section so that you'll feel more comfortable exploring the rest.

Once you feel sufficiently competent in the language you will find that the optimal strategy for learning how to put together a particular piece of code is usually to search the web for "R [whatever you want to do]". RSeek is also an immensely useful resource. If you want to learn about a specific function, simply type `?func` into your `R` console, where `func` is the name of the function you want to look up.

[1]If you're bored, skip ahead to the puzzles. I won't tell.

Auto data example

The first step is to load the `foreign` package[2] Once `foreign` is loaded, you'll have access to all of its constituent functions, including `read.csv` which will convert a comma-separated value worksheet (.csv) into a data frame[3]. A data frame is a particular type of variable in `R`; it might be helpful to think of it as a frame that holds our data[4] Using `read.csv`, we will load `auto.csv` into a data frame called `data` [5]. We set the option `header` to `TRUE`, which tells the `R` interpreter to read the first line of the CSV file as variable names, rather than an observation.

```
library(foreign)
data <- read.csv("auto.csv", header=TRUE)
```

We can read the names from the data set; but they aren't much help.

```
names(data)
```

```
[1] "V1" "V2" "V3"
```

We can replace the column headers with more descriptive variable names. There are three elements in both the original and new list; and each element in the original list is reassigned based on its position in the list, e.g., the `V1` header is replaced by `price`.

```
names(data) <- c("price", "mpg", "weight")
```

To get a sense of the data, list the first six observations using `head()`:

```
head(data)
```

```
  price mpg weight
1  4099  22   2930
2  4749  17   3350
3  3799  22   2640
4  4816  20   3250
5  7827  15   4080
6  5788  18   3670
```

With the columns named, we use the `$` operator to request a particular variable from `data`.

```
head(data$mpg)
```

```
[1] 22 17 22 20 15 18
```

Suppose we want to sort this data. This is not quite as easy as it is in Stata, but it is still fairly easy. To sort by `price`, we use the command `order()`.

```
data.sorted <- data[order(data$price), ]
head(data.sorted)
```

[2]If the foreign package isn't installed, we would install it first using `install.packages("foreign")`

[3]Note that it is also possible to read in `xls`, `dta`, tab-delimited, and many other types of data using similar functions.

[4]"I yam what I yam." -Popeye

[5]`auto.csv` can be downloaded here.

```
   price mpg weight
34  3291  20   2830
14  3299  29   2110
18  3667  24   2750
68  3748  31   2200
66  3798  35   2050
3   3799  22   2640
```

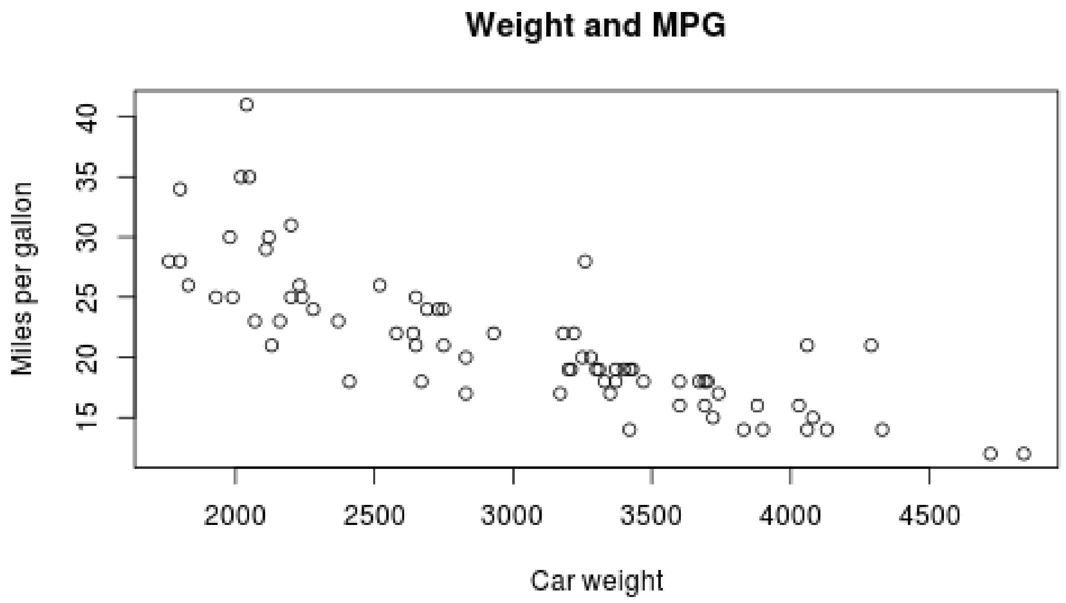
If you don't understand how this command works right now, don't worry! We'll go into the guts of it later. We also might want to take an average of these values. There are many ways to do this, but for our application here the simplest is probably by using the function `mean()`.

```
mean(data$price)
```

```
[1] 6165.257
```

One of the Max-ims[6] of data analysis is to examine your data in as many different ways as possible to find outliers, mistakes, non-linearities, or other irritating/useful idiosyncracies in the data. A standard first step is to look for interesting relationships using a scatterplot. Here, we might think that there is some relationship between `mpg` and `weight`:

```
plot(data$weight,data$mpg, main="Weight and MPG", xlab="Car weight", ylab="Miles per gallon")
```

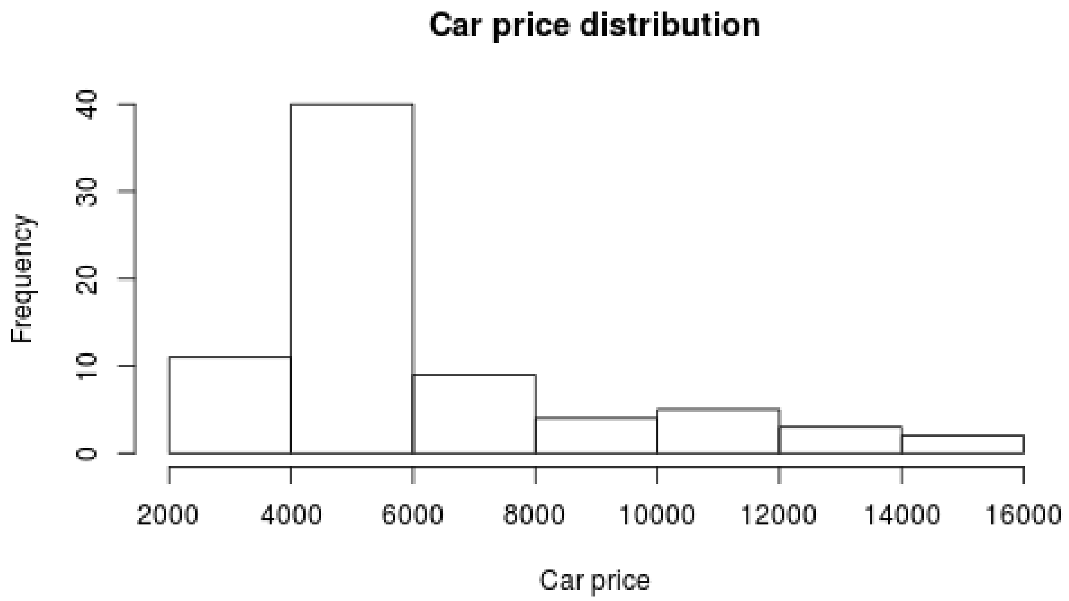


As you might expect, miles per gallon is negatively correlated with car weight.

[6]I just made that up. It's a Max Auffhammer maxim. Tip your bartender.

It may also be useful to think about how single variables are distributed. One good way to do this is with a histogram.

```
hist(data$price, main="Car price distribution", xlab="Car price")
```



`R` is an incredibly powerful tool for creating informative and beautiful graphics with relatively little coding effort. `ggplot2` is a particularly nice package for this.

That's it for this section. Next week we will take a break from real data to introduce basic matrix algebra commands in `R`.

Linear algebra puzzles

These notes will provide a code illustration of the Linear Algebra review in Chapter 1 of the lecture notes. Don't worry if you can't solve these puzzles, they all require commands that we have not covered in section. Come back to them later, once we have gone over `R` code in more detail. There are many correct ways to solve these puzzles. If time remains, I will go over a couple of these next week.

1. Let $\mathbf{I}_5$ be a 5×5 identity matrix. Demonstrate that $\mathbf{I}_5$ is symmetric and idempotent using simple functions in `R`.

2. Generate a 2×2 idempotent matrix $\mathbf{X}$, where $\mathbf{X}$ is not the identity matrix. Demonstrate that $\mathbf{X} = \mathbf{XX}$.

3. Generate two random variables, $\mathbf{x}$ and $\mathbf{e}$, of dimension $n = 100$ such that $\mathbf{x}, \mathbf{e} \sim N(0, 1)$. Generate a random variable $\mathbf{y}$ according to the data generating process $y_i = x_i + e_i$. Show that if you regress $\mathbf{y}$ on $\mathbf{x}$ using the canned linear regression routine `lm()`, then you will get an estimate of the intercept β_0 and the coefficient on $\mathbf{x}$, β_1, such that $\beta_0 = 0$ and $\beta_1 = 1$.

4. Show that if $\lambda_1, \lambda_2, \ldots, \lambda_5$ are the eigenvectors of a 5×5 matrix $\mathbf{A}$, then $\text{tr}(\mathbf{A}) = \sum_{i=1}^{5} \lambda_i$.