This section is intended to introduce linear regression in . The first step is to read the data set `auto.csv` from the relative path. You can also download it from here. We set the option `header` to `TRUE`, which treats the first line of the CSV file as variable names, rather than an observation.

```
data <- read.csv("../data/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. The next command is an example of destructuring. 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:

```
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 appropriately names, we can refer to particular variables within the data set using the unique indexing in `R`, where data objects tend to be variants of lists and nested lists.

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

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

Now for the analysis, a linear model using `lm()`. We specify the model by referring to the column headers within the specified data set — along with a 1 to indicate a column of ones. `R` is unique for intelligently snapping the dimensions of comparable matrices.

```
lm(price ~ 1 + mpg + weight, data=data)
```

```

Call:
lm(formula = price ~ 1 + mpg + weight, data = data)

Coefficients:
(Intercept)          mpg       weight
   1946.069      -49.512        1.747
```

Note that we do not need to refer to the data set in calling a vector, e.g., `data$price`. Instead, we "attach" the data within the linear model. We can do the same thing for the `R` session, which tends to simplify notation. The columns can be referenced directly after the data set has been attached.

```
attach(data)
head(mpg)
```

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

The use of *canned* routines is not permitted for most of this class; you'll have to write the econometric routines from first principles. First, create matrices of the data, since we will be working mainly with matrix operations. Let $\mathbf{y}$ be the dependent variable, price, and let $\mathbf{X}$ be a matrix of the other car characteristics, along with a column of ones prepended. The `cbind()` function binds the columns horizontally and coerces the `matrix` class.

```
y <- matrix(price)
X <- cbind(1, mpg, weight)
```

Check that the number of observations are the same in both the dependent vector $\mathbf{y}$ and the cofactor matrix $\mathbf{X}$.

```
dim(X)[1] == nrow(y)
```

```
 [1] TRUE
```

Before preceding, we should `detach` the data frame, so that if we reload the `R` script, there won't be any naming conflicts. The `X` and `y` objects will persist, as will `data$mpg`, but we will no longer be able to call the variables without referencing the data frame.

```
detach(data)
```

Using the matrix operations described in the previous section, we can quickly estimate the ordinary least squared parameter vector.

```
beta <- solve(t(X) %*% X) %*% t(X) %*% y
print(beta)
```

```
               [,1]
       1946.068668
mpg     -49.512221
weight    1.746559
```

This vector matches the coefficient vector from the canned routine, thankfully. Digging deeper into the numbers, consider the projection matrix $\mathbf{P} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ and the residual maker matrix $\mathbf{M} = \mathbf{I}_n - \mathbf{P}$

```
n <- nrow(y)
P <- X %*% solve(t(X) %*% X) %*% t(X)
M <- diag(n) - P
```

`R` is useful for checking the properties of these matrices, including whether $\mathbf{M}$ is symmetric, that is, whether $\mathbf{M} = \mathbf{M}'$. The function `all.equal()` does not test **exact** equality, but instead whether the supplied objects are "close enough" to be considered the same. The problem is the limits of machine precision, and rounding at the tail ends of floating point numbers.

```
all.equal(M, t(M))
```

```
 [1] TRUE
```

If we want to test for exact equality, we set the tolerance to zero, and the function will return a message with the mean relative difference between elements — which is clearly very close to zero.

```
all.equal(M, t(M), tol=0)
```

```
 [1] "Mean relative difference: 7.266263e-15"
```

The residual maker matrix should also be idempotent, or $\mathbf{M} = \mathbf{M}\mathbf{M}'$.

```
all.equal(M, M %*% t(M))
```

```
[1] TRUE
```

Finally, we can examine the different components of the variation in the dependent variable, as they relate to the OLS estimate. Specifically, we can show that the total sum of square is equal to the sum of the residual and estimated sum of squares:

$$\mathbf{y}'\mathbf{y} = \hat{\mathbf{y}}'\hat{\mathbf{y}} + \mathbf{e}'\mathbf{e} \tag{1}$$

First, define the relevant variables:

```
e <- M %*% y
y.hat <- P %*% y
rss <- t(e) %*% e
ess <- t(y.hat) %*% y.hat
tss <- t(y) %*% y
```

Then check the condition in Eq. (1):

```
all.equal(tss, ess + rss)
```

```
[1] TRUE
```