You can also tell **R** to download the file for you so that you have a physical copy on your disk. You can then load it with the same function:

```
> download.file(url = "http://www.peterhaschke.com/Teaching/R
  -Course/FE2013.csv", destfile = "z:/FE2013.csv")
> FE2013 <- read.csv(file = "z:/FE2013.csv")
>
```

To save your dataset you can use the `save()` function we used before or the `write.csv()` function:

```
> save(FE2013, file = "z:/Dataset.Rdata")
> # or
> write.csv(FE2013, file = "z:/Dataset.csv")
>
```

Stata Data `.dta`

Importing datasets created in different formats is relatively straightforward. All you have to do is install the `foreign` package. This should be installed already in the THE STAR LAB but you will have to load it before being able to access its functions. The `foreign` package extends **R**'s `read()` and `write()` functions. You now have access to `read.dta()` and `write.dta()` to read and write Stata files, and many others (e.g. `read.spss()`, etc).[3]

```
> library(foreign)
> Students <- read.dta("http://www.peterhaschke.com/Teaching/
  R-Course/Students.dta")
>
```

5.2 Manipulating Data Frames

Once you have your datasets loaded fun ensues. At this point we have three datasets in **R**'s active memory, the diamonds dataset, the one on fuel economy and one on current Political Science Ph.D. students at the University of Rochester. Let's verify just to make sure:

```
> ls()
[1] "diamonds"  "FE2013"  "Students"
```

[3]Use the `help.start()` function to find out more about this package.

The first thing to note about data frames, is that it is usually not terribly helpful to print the object to the screen. Just for the heck of it, try it with the fuel economy data:

```
> FE2013
```

If you pressed enter, R will literally printed all the data to the screen. Unless you are dealing with tiny datasets containing only two or three variables, this is a waste of time and won't tell you anything. The best thing to do first is to use the `names()` and the `dim()` functions. This will tell you all the variable names of the data frame and give you some idea about the size of the dataset.

```
> names(FE2013)
 [1] "ModelYear"              "Manufacturer"
 [3] "Division"               "Model"
 [5] "Displacement"           "Cylinder"
 [7] "FEcity"                 "FEhighway"
 [9] "FEcombined"             "Guzzler"
[11] "AirAspiration1"         "AirAspiration2"
[13] "Gears"                  "LockupTorqueConverter"
[15] "DriveSystem1"           "DriveSystem2"
[17] "FuelType"               "FuelType2"
[19] "AnnualFuelCost"         "IntakeValvesPerCyl"
[21] "ExhaustValvesPerCyl"    "Class"
[23] "OilViscosity"           "StopStartSystem"
[25] "FErating"               "CityCO2"
[27] "HighwayCO2"             "CombinedCO2"

> dim(FE2013)
[1] 1082   28
```

And ...

```
> names(Students)
[1] "Name" "Year"

> dim(Students)
[1] 47  2
```

5.2.1 Extraction

Most of the basic extraction principles – namely the [ ] – we used for matrices also work for data frames. But you should remember that data frames are a special type of list and as such the $ will come in handy. For example, lets try to extract the variable called `"Gears"`. Unless you knew that `"Gears"` was the 13th variable you'd be trying around a bit. But either way works. Let's test it

```
> FE2013[,13]
  [1] 6 8 6 7 6 7 7 6 6 6 6 7 7 7 7 7 6 6 6 6 6 5 6 7 7
 [26] 7 7 6 7 7 7 7 7 7 5 5 6 6 6 6 6 6 6 6 6 6 6 6 7 6
 [51] 7 6 7 6 7 6 7 7 6 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6
 [76] 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 6
[101] 7 6 7 7 6 6 1 6 6 8 8 8 8 8 6 7 7 6 7 6 6 8 6 8 6
  ...

FE2013$Gears
  [1] 6 8 6 7 6 7 7 6 6 6 6 7 7 7 7 7 6 6 6 6 6 5 6 7 7
 [26] 7 7 6 7 7 7 7 7 7 5 5 6 6 6 6 6 6 6 6 6 6 6 6 7 6
 [51] 7 6 7 6 7 6 7 7 6 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6
 [76] 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 6
[101] 7 6 7 7 6 6 1 6 6 8 8 8 8 8 6 7 7 6 7 6 6 8 6 8 6
  ...
```

You can combine the $ notation and the [ ] notation since the $ extracts a vector and vectors are indexed via [ ].

```
> FE2013$Gears[1:3] # returns the first three elements of $
   Gears
[1] 6 8 6
```

This of course also means that you can do anything to a data frame's variables that you can do to vectors.

```
> log(FE2013$Gears) + (FE2013$Gears + 100)
  [1] 107.7918 110.0794 107.7918 108.9459 107.7918
  [6] 108.9459 108.9459 107.7918 107.7918 107.7918
 [11] 107.7918 108.9459 108.9459 108.9459 108.9459
 [16] 108.9459 107.7918 107.7918 107.7918 107.7918
 [21] 107.7918 106.6094 107.7918 108.9459 108.9459
 [26] 108.9459 108.9459 107.7918 108.9459 108.9459
 [31] 108.9459 108.9459 108.9459 108.9459 106.6094
  ...
```

The with() Function

The problem with datasets being lists is that working with the $ notation is kind of tedious. Luckily there exists a function that makes dealing data frame names easier. Whenever you need to extract or index multiple variables and don't feel like typing `dataset$variable.name` each time, use the `with()` function.

```
> FE2013$Gears + FE2013$ModelYear / FE2013$Cylinder
  [1] 509.2500 511.2500 509.2500 342.5000 341.5000
  [6] 342.5000 132.8125 257.6250 257.6250 257.6250
 [11] 257.6250 258.6250 258.6250 258.6250 258.6250
 [16] 174.7500 207.3000 207.3000 207.3000 207.3000
 [21] 509.2500 508.2500 509.2500 258.6250 258.6250
 [26] 174.7500 510.2500 509.2500 342.5000 258.6250
   ...

> with(FE2013, Gears + ModelYear / Cylinder)
  [1] 509.2500 511.2500 509.2500 342.5000 341.5000
  [6] 342.5000 132.8125 257.6250 257.6250 257.6250
 [11] 257.6250 258.6250 258.6250 258.6250 258.6250
 [16] 174.7500 207.3000 207.3000 207.3000 207.3000
 [21] 509.2500 508.2500 509.2500 258.6250 258.6250
 [26] 174.7500 510.2500 509.2500 342.5000 258.6250
   ...
```

Warning NEVER use the `attach()` or `detatch()` to add a dataset to the search path of available R objects. If somebody tells you otherwise. They are wrong. These functions create all sorts of trouble.

5.2.2 Subsetting

You already know about the `subset()` function from our treatment of vectors. The `subset()` function works the same way for datasets. Let's look at the other data frame.

```
> Students
                          Name Year
1               Jeffrey Arnold    6
2        Sergio Ascencio Bonfil    2
3            Chitralekha Basu    3
4            Jonathan Bennett    1
5                  Peter Bils    1
6              Robert Carroll    4
7                   Hun Chung    1
8           Casey Crisman-Cox    2
9                  Trung Dang    3
10               Mason DeLang    2
11               David Gelman    1
```

```
12           Michael Gibilisco    2
13               Peter Haschke    6
14             YeonKyung Jeong    1
15                Doug Johnson    1
16                Gleason Judd    1
17                Kerim Kavakli   7
18           Brenton J. Kenkel    5
19                 HyeSung Kim    5
20           Jonathan Klingler    6
21                Patrick Kuhn    7
22                Ben Laughlin    2
23              Youngchae Lee    7
24                 Rabia Malik    3
25              Paulina Marek    6
26            Jeffrey Marshall    4
27            Justin Nicholson    1
28            Jonathan Olmsted    6
29                 Lukas Pfaff    6
30          Barbara Piotrowska    1
31               Shawn Ramirez    8
32                 Luke Reilly    2
33      Miguel R. Rueda Robayo    5
34          Kristin K. Rulison    5
35                 Jeheung Ryu    1
36                 Yoji Sekiya    8
37           Mattan Sharkansky    3
38               Bradley Smith    1
39              Jennifer Smith    2
40             William Spaniel    3
41               Jessica Stoll    7
42                   Ian Sulam    6
43             Susanna Supalla    4
44             Matthew Sweeten    1
45 Svanhildur Thorvaldsdottir    3
46          Ioannis Vassiliadis   2
47                     Jie Wen    1
```

Notice also that data frames are not printed the same as matrices (i.e. there are no [ ]).

Let's subset this dataset such that only first year students and myself are included:

```
> Rcourse <- subset(Students, Students$Year == 1 | Students$
   Name == "Peter Haschke")
> Rcourse
                 Name Year
4    Jonathan Bennett    1
5          Peter Bils    1
7           Hun Chung    1
11       David Gelman    1
13      Peter Haschke    6
14    YeonKyung Jeong    1
15       Doug Johnson    1
16       Gleason Judd    1
27   Justin Nicholson    1
30 Barbara Piotrowska    1
35        Jeheung Ryu    1
38      Bradley Smith    1
44    Matthew Sweeten    1
47            Jie Wen    1
```

5.2.3 Editing

R is not very good for editing datasets or data entry generally. This is not surprising since R is not a spreadsheet. If you want to use Excel to edit your datasets, feel free to do so. You know how to load and save from and to the .csv format which Excel can deal with. If you really feel so inclined – I do not advise this – you can use the edit() function. This will open up an interactive spreadsheet like environment for data entry and data manipulation. Again, just use Excel.

Let's do some manual data entry, anyway. We will use the $ operator to create a new variable in our Rcourse data frame.

```
> Rcourse$GreatFirstName <-  "No"
> Rcourse$GreatFirstName[c(2,5)] <- "Yes"
> Rcourse$New <- 1:length(Rcourse$Name)
> Rcourse$New2 <- rep(c("A", "B"), 7)
> Rcourse
                 Name Year GreatFirstName New New2
4    Jonathan Bennett    1             No   1    A
5          Peter Bils    1            Yes   2    B
7           Hun Chung    1             No   3    A
11       David Gelman    1             No   4    B
13      Peter Haschke    6            Yes   5    A
14    YeonKyung Jeong    1             No   6    B
15       Doug Johnson    1             No   7    A
```

```
16        Gleason Judd    1              No   8    B
27    Justin Nicholson    1              No   9    A
30 Barbara Piotrowska    1              No  10    B
35         Jeheung Ryu    1              No  11    A
38       Bradley Smith    1              No  12    B
44     Matthew Sweeten    1              No  13    A
47             Jie Wen    1              No  14    B
```

The `transform()` Function

To bulk edit or transform a number of variables at once, the `transform()` function can be used:

```
> Rcourse <- transform(Rcourse, Months = Year * 12
                              , New = New / 10 )
> Rcourse
                Name Year GreatFirstName New New2 Months
4    Jonathan Bennett    1              No 0.1    A     12
5          Peter Bils    1             Yes 0.2    B     12
7           Hun Chung    1              No 0.3    A     12
11       David Gelman    1              No 0.4    B     12
13      Peter Haschke    6             Yes 0.5    A     72
14    YeonKyung Jeong    1              No 0.6    B     12
15       Doug Johnson    1              No 0.7    A     12
16       Gleason Judd    1              No 0.8    B     12
27   Justin Nicholson    1              No 0.9    A     12
30 Barbara Piotrowska    1              No 1.0    B     12
35        Jeheung Ryu    1              No 1.1    A     12
38      Bradley Smith    1              No 1.2    B     12
44    Matthew Sweeten    1              No 1.3    A     12
47            Jie Wen    1              No 1.4    B     12
```

5.3 More on *Objects*, *Modes* and other *Lies*

In Chapter 2 we talked rather loosely about *objects* and *modes*. I claimed that there exist a variety of different object types and that various objects can store elements of various modes.

For example. I claimed that there exists an object type called a vector. Moreover, I insisted that every vector can at most store elements of one mode. To determine what type of object and what mode we are dealing with the `class()`, `mode()`, and `is()` functions can be employed. Technically things are much more complicated and my treatment of *objects* and *modes* is quite forced.

Be that as it may. Our fancy data frame Rcourse can be understood as a special type of object (namely a special list) storing other objects (namely vectors and factors) of varying modes. Let's extract four of its components.

```r
> Name <- Rcourse$Name
> Year <- Rcourse$Year
> New  <- Rcourse$New
> New2 <- Rcourse$New2
```

When we are using the `class()` function on the four vectors we have just created we will find the following.

```r
> class(Name) # this tells us that the elements stored in
    this object are factors
[1] "factor"

> class(Year) # this tells us that the elements stored in
    this object are of mode integer
[1] "integer"

> class(New)  # this tells us that the elements stored in
    this object are of mode numeric
[1] "numeric"

> class(New2) # this tells us that the elements stored in
    this object are of mode character
[1] "character"
```

It is very important to realize that different types and modes affect the behavior of all R functions. For example factors are special vectors that contain an attribute called level. They are different from character vectors. To see this just print Name and New2:

```r
> Name
 [1] Jonathan Bennett   Peter Bils
 [3] Hun Chung          David Gelman
 [5] Peter Haschke      YeonKyung Jeong
 [7] Doug Johnson       Gleason Judd
 [9] Justin Nicholson   Barbara Piotrowska
[11] Jeheung Ryu        Bradley Smith
47 Levels: Barbara Piotrowska ... Youngchae Lee

> New2
 [1] "A" "B" "A" "B" "A" "B" "A" "B" "A" "B" "A"
[12] "B" "A" "B"
```

In the above example we can see that printing each object produces different results. Printing a `factor` returns an abbreviated listing of its `levels`. To get the full list, type `levels(Name)`. You will see that this listing contains all the names of current graduate students even though none of them are part of the `Rcourse` data frame. In other words a `factor` is more than a vector of `characters` elements but and indicator vector. To beat a horse to death, type `summary(Name)` as well as `summary(New2)`. You can see that the summary of the `character` vector was not terribly useful.

Luckily, R is capable of changing mode and object types rather seamlessly. You will be bound to use many of the functions below:

Function	Description
`as.numeric()`	turns vectors, and matrices of other modes into a numeric ones
`as.character()`	turns vectors, and matrices of other modes into character ones
`as.integer()`	turns vectors, and matrices of other modes into integer ones[4]
`as.factor()`	will turn a vector, or matrices into factors
`as.matrix()`	will turn a vector, or data frame into a matrix[5]
`as.vector()`	will turn matrices into vectors
`as.data.frame()`	will turn vectors and matrices into data frames
`as.list()`	will turn vectors and matrices into lists

For our example above let's turn `New2` into a `factor` and try the summary command again. Instead of jibberish, the summary now produces a nice tabulation of frequencies.

```
> New2 <- as.factor(New2) # we are overwriting the old New2
> New2
 [1] A B A B A B A B A B A B A B
Levels: A B

> summary(New2)
A B
7 7
```

5.4 Data Summaries

Since you know how to extract and recall components of data frames, summaries can be computed with the functions you have used for vectors and matrices.

```
> mean(Rcourse$New) # the mean of the New variable for
   example
[1] 0.75

> summary(as.factor(Rcourse$GreatFirstName))
 No Yes
 12   2
```

You can also use the summary() function on the whole data frame.

```
> summary(Rcourse)   \\
       Names                  Year
 Barbara Piotrowska:1   Min.   :1.000
 Bradley Smith     :1   1st Qu.:1.000
 David Gelman      :1   Median :1.000
 Doug Johnson      :1   Mean   :1.357
 Gleason Judd      :1   3rd Qu.:1.000
 Hun Chung         :1   Max.   :6.000
 (Other)           :8

  GreatFirstName          New
 Length:14          Min.   :0.100
 Class :character   1st Qu.:0.425
 Mode  :character   Median :0.750
                    Mean   :0.750
                    3rd Qu.:1.075
                    Max.   :1.400

     New2               Months
 Length:14          Min.   :12.00
 Class :character   1st Qu.:12.00
 Mode  :character   Median :12.00
                    Mean   :16.29
                    3rd Qu.:12.00
                    Max.   :72.00
```

Another useful feature is the table() function. It allows you to create contingency tables. Use ?table to find out more on this.

```
> table(Students$Year)
 1  2  3  4  5  6  7  8
13  8  6  3  4  7  4  2

> table(Rcourse$Year, Rcourse$GreatFirstName)
    No Yes
  1 12   1
  6  0   1
```

A more complex table:

```
> table(FE2013$FErating, FE2013$Cylinder)
       3   4   5   6   8  10  12  16
  1    0   0   0   0   8   2   0   1
  2    0   0   0   0  53   1  15   0
  3    0   0   0   3  73   3   6   0
  4    0   6   0  86 106   0   1   0
  5    0  31   2 200  23   0   0   0
  6    0 121   8  59   0   0   0   0
  7    0 110   7   7   0   0   0   0
  8    0 119   0   4   0   0   0   0
  9    2  14   0   0   0   0   0   0
  10   0  11   0   0   0   0   0   0

> cor(FE2013$FErating, FE2013$Cylinder)
[1] -0.8114269
```

The str() function summarizes the structure of a dataset.

```
> str(Rcourse)
'data.frame':   14 obs. of  6 variables:
 $ Name          : Factor w/ 47 levels "Barbara Piotrows ...
 $ Year          : int  1 1 1 1 6 1 1 1 1 1 ...
 $ GreatFirstName: chr  "No" "Yes" "No" "No" ...
 $ New           : num  0.1 0.2 0.3 0.4 0.5 0.6 0.7 ...
 $ New2          : chr  "A" "B" "A" "B" ...
 $ Months        : num  12 12 12 12 72 12 12 12 12 12 ...
```

The describe() Function

For an even more detailed summary for our dataset, load the Hmisc package. You may have to install it first. As always check out: ?describe after loading the package.

```
> library(Hmisc)
> describe(Rcourse)
Rcourse

 6  Variables      14  Observations
---------------------------------------------------------
Name
      n missing  unique
     14       0      14
```

```
Barbara Piotrowska (1, 7%), Bradley Smith (1, 7%)
David Gelman (1, 7%), Doug Johnson (1, 7%)
Gleason Judd (1, 7%), Hun Chung (1, 7%)
Jeheung Ryu (1, 7%), Jie Wen (1, 7%)
Jonathan Bennett (1, 7%), Justin Nicholson (1, 7%)
Matthew Sweeten (1, 7%), Peter Bils (1, 7%)
Peter Haschke (1, 7%), YeonKyung Jeong (1, 7%)
---------------------------------------------------------
Year
      n missing  unique    Mean
     14       0       2   1.357

1 (13, 93%), 6 (1, 7%)
---------------------------------------------------------
GreatFirstName
      n missing  unique
     14       0       2

No (12, 86%), Yes (2, 14%)
---------------------------------------------------------
New
      n missing  unique    Mean     .05     .10     .25
     14       0      14    0.75   0.165   0.230   0.425
    .50     .75     .90     .95
  0.750   1.075   1.270   1.335

          0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 1.1 1.2
Frequency   1   1   1   1   1   1   1   1   1 1   1   1
%           7   7   7   7   7   7   7   7   7 7   7   7
          1.3 1.4
Frequency   1   1
%           7   7
---------------------------------------------------------
New2
      n missing  unique
     14       0       2

A (7, 50%), B (7, 50%)
---------------------------------------------------------
Months
      n missing  unique    Mean
     14       0       2   16.29

12 (13, 93%), 72 (1, 7%)
---------------------------------------------------------
```