R Tips

Professor Rossi

Getting course data

You need to install the course package "PERregress." This package contains all of the data for the course as well as some customized functions written by me.

Once the package is installed, you need to tell R to access a dataset and then tell R that you want it to "look" in the dataset for objects (such as variables).

```
> library(PERregress)
> data(mfunds)
> attach(mfunds)
> head(windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222
```

Here we have looked at the "head" or "beginning" of the variable `windsor` which is the returns on the Vanguard Windsor fund.

Getting Help

You can always get help on a dataset or a command by using the `help()` function or the shortcut ?

To get help on `mfunds` or `lm`

```
> ?mfunds
> ?lm
```

You can also use the help menu on the main R window bar. Note: there is a very nice introductory manual under "manuals." If you click on packages, you can get help on any of the datasets or functions in PERregress.

Understanding the R environment

Data and Functions

In the R world, there are two types of things (they call "objects") data and functions. Data is self-explanatory. Data can be stored either in a dataframe (R's version of a spreadsheet) or as a variable (sometimes called a vector). Functions are designed to do things with data.

Every command in R is something like

```
>out=Rfunction(data)
```

"Rfunction" is a function and "data" is some sort of data.

Examples:

```
> descStat(mfunds)
          Mean Median    SD   IQR SE Mean 95% CI-L 95% CI-U NMissing
drefus   0.007  0.007 0.047 0.064   0.004    0.000    0.014        0
fidel    0.005  0.002 0.057 0.080   0.004   -0.004    0.013        0
keystne  0.007  0.006 0.084 0.102   0.006   -0.006    0.019        0
Putnminc 0.006  0.005 0.030 0.031   0.002    0.001    0.010        0
scudinc  0.004  0.004 0.036 0.040   0.003   -0.001    0.010        0
windsor  0.010  0.008 0.049 0.067   0.004    0.003    0.017        0
eqmrkt   0.011  0.004 0.069 0.082   0.005    0.001    0.021        0
valmrkt  0.007  0.004 0.048 0.067   0.004    0.000    0.014        0
tbill    0.006  0.005 0.003 0.003   0.000    0.006    0.006        0
Number of Observations =  180

> var(drefus)
[1] 0.002231345

> new_var=rnorm(10)
```

In the first example above, we are using the function "PERregress" to compute descriptive statistics for the data in the mfunds dataset (note you must first access the dataset by using `data(mfunds).`

In the second example, we are using a built-in function `var()` to compute the sample variance of the variable drefus.

In the third example, we are creating a new variable by drawing 10 standard normal random numbers using rnorm.

The R session and the "workspace"

Everything that is created in an R "session" (the period of time from when you invoke R by double clicking the icon to when you quit) is saved in the memory (NOT DISK) of your computer.

For example, if we create a new variable by drawing 100 random variable. This new variable will be stored in memory and can be reused.

Ex:

```
> normrv=rnorm(100)
> var(normrv)
[1] 1.023766
> objects()
[1] "normrv"
> ls()
[1] "normrv"
```

The first command creates the variable normrv using the `rnorm()` function to draw from the std normal random variable. The next command computes the sample variance of the variable we have created.

The variable, normrv, is now available in our R session memory (called the "workspace") and can be used by another command in our session. The function "objects" lists all objects (dataframes, variables or functions) currently found in the workspace.

Another example will illustrate this point:

```
> normrv_sq=normrv**2
> objects()
[1] "normrv"    "normrv_sq"
>
```

In this example we created a new variable, normrv_sq, by squaring the old variable. New now have two variables in our workspace.

When you leave R (using the function `q()` for quit), you will be prompted to see if you want to save the variables you have created. If you say yes, then the variables will be there the next time you start up R.

Understanding how R looks up variables/dataframes

Be careful about creating variables with the same name as variables in the datasets.

```
> data(mfunds)
> attach(mfunds)
> head(windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222

> objects()
[1] "mfunds"

> windsor=1
> windsor
[1] 1
> objects()
[1] "mfunds"  "windsor"
> head(mfunds$windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222
```

What happened here?
First, we accessed and attached the mfunds dataset and showed the contents of the beginning of the windsor variable. `objects()` now shows that mfunds is available to us for use.

Second, we created a new variable with the same name as one of the variables in mfunds. When we type the name of the variable, R prints out the contents of our new variable.

Does this mean we have erased or over-written the variable windsor in the mfunds dataset/dataframe? NO! R is not dumb. But unless we tell R differently, it will access our new variable which is in the workspace.

To reference the windsor fund returns in mfunds, we have to use the full name, mfunds$windsor.

Removing or Erasing Variables Created in the Workspace

We often create something that we don't want to use. We can always leave it in the workspace (but that's like leaving a useless paper on your desk). To remove it from the workspace, we can say

```
>rm(normrv)
```

DON'T WORRY IF YOU MAKE A MISTAKE! It is impossible to erase the datasets in the course, you can only remove those variables and dataframes that you create.

Using R as a Powerful Calculator

R has many built-in functions to sum things up, multiply, divide, exponentiate, log, draw random numbers, etc. You can create your own variables with the results. Let's look a few examples:

```
> var(windsor)
[1] 0.002365798
> vwindsor=sum((windsor-mean(windsor))**2)/179
> vwindsor
[1] 0.002365798
> vwindsor=sum((windsor-mean(windsor))**2)/(length(windsor-1))
> vwindsor
[1] 0.002352655
```

Here I'm summing up squared deviations from the mean and dividing by N-1. I'm using

`var()` -- compute sample variance
`mean()` -- compute sample mean
`length()` -- compute the length of a variable
`sum()` -- sum up the contents of a variable (just like the summation notation)

Other useful built-in functions include

`log()`
`sd()`
`cov()`
`exp()`
`round()`

Accessing Variables and Subsetting Variables and Dataframes

We have already seen that we can access variables in a dataframe by using `attach` to make the variables directly accessible or using the $ method. We can also access the variables by using their column number in the data frame:

```
> head(windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222

> head(mfunds$windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222

> str(mfunds)
'data.frame':     180 obs. of  9 variables:
 $ drefus  : num  -0.0628 -0.0423 0.0217 0.1004 0.0214 ...
 $ fidel   : num  -0.0779 -0.0395 0.0165 0.0926 0.0308 ...
 $ keystne : num  -0.072 -0.0997 0.0169 0.1906 0.0762 ...
 $ Putnminc: num  -0.01289 -0.04244 -0.00795 0.07416 0.02481 ...
 $ scudinc : num  -0.0285 -0.027 0.0115 0.0717 0.0352 ...
 $ windsor : num  -0.02976 -0.02355 0.00224 0.08842 0.02512 ...
 $ eqmrkt  : num  0.0228 -0.0571 -0.0124 0.1417 0.0903 ...
 $ valmrkt : num  -0.03654 -0.03386 0.00514 0.09381 0.02639 ...
 $ tbill   : num  0.004 0.0039 0.0038 0.0043 0.0045 ...
> head(mfunds[,6])
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222
```

Note: in the last command above, we are using the square brackets to access the 6th column of the spreadsheet mfunds.

Very often, we might only wish to use some of the observations in a variable. To access specific observations in a variable, we can use the "[ ]" command.

```
> windsor[2]
[1] -0.023549

> windsor[2:6]
[1] -0.023549  0.002243  0.088416  0.025118  0.035222

> mfunds[1,]
     drefus      fidel    keystne  Putnminc   scudinc   windsor   eqmrkt
valmrkt tbill
1 -0.062783 -0.077937 -0.071979 -0.012889 -0.028539 -0.029756 0.022792
-0.03654 0.004


> head(windsor)
[1] -0.029756 -0.023549  0.002243  0.088416  0.025118  0.035222
> head(windsor[-1])
[1] -0.023549  0.002243  0.088416  0.025118  0.035222 -0.004931
```

In the first example, we access the 2nd observation in windsor.

In the second example, we access the 2-6 observations.

In the third, we access the first observation for all variables in the mfunds spreadsheet. mfunds has two subscripts [row,col].

In the fourth, we are accessing all observations except the first in windsor (we feed it thru the head() function to avoid printing 179 values.

Using Other Variables to Subset

Sometimes we want to use two or more variables to subset the data. For example, supposed we want to compute the mean price for only LED tvs in the Flat_Panel_TV dataset.

```
> data(Flat_Panel_TV)
> attach(Flat_Panel_TV)
> Price[Type=="LED"]
 [1] 1527  499 1697 2300  792  989 2000 1088  896  849 1800 1316 1600
1780  750 1355  657
[18] 1497 1000 1910 1850  999 2100 1400 1600 1200 1518  899 2497 1958
750  744 4049 1099
[35] 1140 1699
> mean(Price[Type=="LED"])
[1] 1439
```

What about something even more complicated, like all TVs with size between 50 and 58 inches?

```
> Price[Size >= 50 & Size <=58]
 [1] 1697 2300 1060 1104  800 2000 1377 1650 1780 1910 1000 1850  907
1470 1186 2100 2498
[18]  994  899 1199 1600 2058  839 1518 1499 1897 1958  899 2029 1155
1699
```

Here we are using what are called logical expressions.

Some points:

==	means exactly equal to
!=	means not equal to
<=	less than or equal to
<	less than
>=	
>	
&	means a logical "and"
|	means a logical "or"

Using the Regression function, lm

The R function, `lm()`, is an extraordinarily powerful function that will run many types of linear regression models.

However, the basic idea is always the same. `lm()` takes data and "output" statistics of computations on this data (like least squares coefficients).

The basic form of the lm function is to specify a regression with variables currently accessible.

```
> out=lm(Price~Size)
> out=lm(Price~Size,data=Flat_Panel_TV)
```

The top form assumes the Flat_Panel_TV is already "attached." Make sure there isn't another variable in your workspace called "Price" or "Size" or R will use that one instead.

The second form is "safety first" and will always access the right variables.

Let's look at what is in "out."

```
> names(out)
 [1]  "coefficients"    "residuals"          "effects"              "rank"
"fitted.values"] "assign"       "qr"              "df.residual"   "xlevels"
"call"          "terms"          "model"

> out$coefficients
(Intercept)        Size
-1408.92604    57.13165

> out$coef
(Intercept)        Size
-1408.92604    57.13165
```

out is what R calls a list. Each element in the list can be accessed using the "$" convention as we did above to print-out the least squares coefficients. Note that you only have to give enough letters of the element of the list to identify it.

You can also use various functions to summarize the contents of out in a variety of diffentet ways (we illustrate these in the course).

```
summary(out)
anova(out)
predict(out)
influence.measures(out)
```

The heart of the lm command is the "formula" which tells R which is the dependent variable and which are the independent variables.

Here are some examples:

Y~X	regression Y on X
Y~X1+X2	regress Y on X1 and X2
Y ~X1*X2	regress Y on X1, X2 and X1*X2 (interaction)
Log(Y)~log(X)	regress log(Y) on log(X)
Y~.	regress Y on all of the other variables in the dataframe (requires data argument).