Intro to R for MATH 109

About R

R is an open source software environment for statistical computing and graphics. It is compatible with Linux, Windows, and Apple operating sysyems; i.e. if you have a computer with a standard operating system it should work just fine. It doesn't take a lot of time to install and it doesn't take a lot of time to learn the basics. As far as programming languages go it is relatively intuitive. Additionally, since it is so popular there are lots of excellent tutorials online. Here is the R webpage and an information booklet with more details:

https://www.r-project.org/

https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

Dowloading and Installing R

To download R you have to choose a CRAN mirror, which is just a fancy way of saying distribution site for software. You want to pick a location close to you.

https://cran.r-project.org/mirrors.html

So if you're in Kentucky you might want to choose the Indiana University CRAN mirror.

http://ftp.ussg.iu.edu/CRAN/

You will be prompted to select the file that is compatible with your OS, then open and run the installer. If you are using Linux you will most likely have to install through the terminal.

Once you have it installed, you're ready to start computing, just click on the R icon.

If you think you might want to try out some other programming languages Anaconda (https://www.anaconda.com/products/individual) is an open-source platform wich includes RStudio (R) as well as Spyder (Python) which is another popular and intuitive language to learn.

RStudio

Once you have installed R, there is an excellent and free GUI (graphical user interface) that you can use to enhance your R capabilities called RStudio. It can be downloaded here

https://rstudio.com/products/rstudio/download/#download

and the installation process is similar to installing R, except that you do not have to choose a CRAN mirror.

Using R Without Installation

You can run R code without installing the software, this is one place:

https://www.tutorialspoint.com/execute_r_online.php

How to Save Your Work

If you are using an online simulator you will want to have a text file open (like Notepad) so you can save your code. If you are using R or RStudio you can work from and save an R script file. In R click `File > New script` or `CTRL+N`. For RStudio click on the green `+` on the top left and select `R script` or click `CTRL+SHIFT+N`. The code itself will be the same regardless of which format of R you are using.

When you're coding, do not forget to save your work frequently: `CTRL+S` is your best friend and can save you a lot of headache.

Basic Coding

If you are using R or an online simulator you will see a "console" which is simply a place to input code and run it. So you can type your code in a text file and copy `CTRL+C` and paste `CTRL+V` it into the console, then click `ENTER` or `Run` or `execute`. If you are using RStudio you should have a script open and you can highlight your code and click `Run` or `CTRL+ENTER`.

Now that we have our text files and R scripts ready to go, we can try some coding. Starting with something simple, let's tell the computer to say hi.

```
print("hello world")
```

```
## [1] "hello world"
```

Be careful, R is case sensitive. So if you tried

```
Print("hello world")
```

you would get an error:

```
Error in Print("hello world") : could not find function "Print"
```

and it's nice of R to tell you what the problem is. There's no function called `Print`. Sometimes you will get an error that is a little bit more cryptic, and you can search the error to get more insight on what went wrong. The majority of the time with simple code, it's a syntax error, meaning the computer didn't understand what you typed because it doesn't follow it's grammar rules. This is the type of error we witnessed above. Other common syntax errors are mispellings and missing or extra punctuation. However, R is more forgiving than other programming languages because it will often let things like extra spaces or indentations slide.

If you have a question about a command, you can use the help function and it should pull up a page about the function you are trying to use.

```
help("print")
```

You can assign values to letters using an `=` or `<-`, and then we can recall those values by typing the variable. This is extremely useful when we're storing more than one value to a variable.

```
x=5
x
```

```
## [1] 5
```

```
y<-3
y
```

```
## [1] 3
```

R can be used as a basic calculator, and remember to use parentheses when appropriate.

```
# addition
x+y
```

```
## [1] 8
```

```
# subtraction
x-y
```

```
## [1] 2
```

```
# multiplication
x*y
```

```
## [1] 15
```

```
# division
x/y
```

```
## [1] 1.666667
```

```
# exponent
x^y
```

```
## [1] 125
```

```
# natual exponent
exp(y)
```

```
## [1] 20.08554
```

```
# natural log
log(x)
```

```
## [1] 1.609438
```

```
# combinations of standard operations
(x+y)*log(x)/(exp(y)-y)
```

```
## [1] 0.7535908
```

Above I used the `#` to include "comments" to "annotate" or "document" the code. The `#` tells the computer that what follows in that line is not code, and should not be exectuted. All programming languages have a command for comments, however, it is often a different symbol like a `%`. This annotation is extremely useful so that when you or someone else reads your code later on they know what that line is doing.

Data Structures

Normally we will have a list of data, like exam scores, and we want to save it in R as a variable so that we can find some summary statistics. Here I will assign `e1` as a vector - simply a list of values using the command combine, `c()`, and putting commas in between each value.

```
# assign a list of data to a variable
e1=c(100,25,85,73,67,80,87,85,73,99,45,79,87,25,67,80,87,96,92);e1
```

```
##  [1] 100  25  85  73  67  80  87  85  73  99  45  79  87  25  67  80  87
## [18]  96  92
```

```
# how many entries are in the vector
length(e1)
```

```
## [1] 19
```

Above, to recall the data I used a semicolon followed by the variable to save on space. The semicolon here tells the computer that there is a new line of code following.

Now suppose I have another set of data that I want to combine with another set of data. There's a couple of ways to do that. I could create a new vector of data and then combine the data into a matrix using the `cbind` function which stands for column bind. Be careful, the number of rows have to match. So I inputed the second exams scores for each student in the same order and then combined that with the first exam scores and assigned them to a new variable. You can do this with as many vectors as you need.

```
e2=c(92,20,86,73,77,86,80,89,90,30,55,80,90,25,60,89,89,99,90)
E=cbind(e1,e2);E
```

```
##         e1 e2
##  [1,] 100 92
##  [2,]  25 20
##  [3,]  85 86
##  [4,]  73 73
##  [5,]  67 77
##  [6,]  80 86
##  [7,]  87 80
##  [8,]  85 89
##  [9,]  73 90
## [10,]  99 30
## [11,]  45 55
## [12,]  79 80
## [13,]  87 90
## [14,]  25 25
## [15,]  67 60
## [16,]  80 89
## [17,]  87 89
## [18,]  96 99
## [19,]  92 90
```

We could also combine the vectors in rows, using `rbind`.

```
E2=rbind(e1,e2);E2
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## e1  100   25   85   73   67   80   87   85   73    99    45    79    87
## e2   92   20   86   73   77   86   80   89   90    30    55    80    90
##    [,14] [,15] [,16] [,17] [,18] [,19]
## e1    25    67    80    87    96    92
## e2    25    60    89    89    99    90
```

However, this format isn't very friendly. We normally want our indexes (in this case students) to be the rows and the data labels (which exam) to be the columns.

There is also a `matrix` command so you can combine data in this way also. There are three things that go into this command, first is the data, second is the the number of rows, and the third is the number of columns. The information must go into the function in this order.

Additionally, if we have a lot of data we can ask to look at only a little bit of it by using the command `head` or `tail` which show the first few or last few rows, respectively. You can also just look at one row or one entry at a time using `data name[desired row(s), desired column(s)]`. Using a colon like `x:y` indicates everything between `x` and `y`, and leaving a blank indicates you want to look at the entire row or column.

```
# create a matrix with existing vectors
E3=matrix(c(e1,e2),length(e1),2)
# look at first few rows
head(E3)
```

```
##      [,1] [,2]
## [1,]  100   92
## [2,]   25   20
## [3,]   85   86
## [4,]   73   73
```

```
## [5,]   67   77
## [6,]   80   86
```

```
# look at the entry at row 1, column 2
E3[1,2]
```

```
## [1] 92
```

```
# look at rows 7, 8, 9, and 10 and both columns
E[7:10,]
```

```
##      e1 e2
## [1,] 87 80
## [2,] 85 89
## [3,] 73 90
## [4,] 99 30
```

```
# this creates the same matrix without making the vectors first
E4=matrix(c(100,25,85,73,67,80,87,85,73,99,45,79,87,25,67,80,87,
            96,92,92,20,86,73,77,86,80,89,90,30,55,80,90,25,60,89,89,99,90),19,2)
# look at last few rows
tail(E4)
```

```
##       [,1] [,2]
## [14,]   25   25
## [15,]   67   60
## [16,]   80   89
## [17,]   87   89
## [18,]   96   99
## [19,]   92   90
```

Another option is to create a data frame, and you can specify the names of your columns.

```
df=data.frame(exam1 = e1,exam2= e2)
df[1:3,]
```

```
##   exam1 exam2
## 1   100    92
## 2    25    20
## 3    85    86
```

Data frames are better than matricies at handling data that is not numeric, and in particular, combining numeric and non-numeric data. So let's suppose I create a vector of the names of the students in this class, and I want to combine it with their two exam scores in a data frame. Note that if the vector is already called what you want the column named, you do not have to specify the column name.

```
Students=c("Sally", "George", "Tonya", "Greg", "Josh", "Lindsay", "Scott", "Tina", "Emily",
           "Matt", "Israel", "Sida", "Jacob", "Mary", "Aaron", "Seth", "Cindy","Daniel","Ryan")
df1=data.frame(Students, Exam1=e1,Exam2=e2)
df1[1:3,]
```

```
##   Students Exam1 Exam2
## 1    Sally   100    92
## 2   George    25    20
## 3    Tonya    85    86
```

If you have a data file from somewhere else, you just need the file path and then you can read in the data directly into R. The most common file formats data comes in is a table where the data is separated by tabs or a comma separated value (CSV) file where the entries are separated by commas.

```
# if you have a text file where the data is separated by tabs
A=read.table("filepath\file.txt or .dat")
# if you have a text file where the data is separated by commas
B=read.csv("filepath\file.csv")
```

Summary Statistics

R was built for statisical computing and graphics, and the commands are intuitive. Some of the basics are demonstrated below.

```
# mean
mean(e1)
```

```
## [1] 75.36842
```

```
# median
median(e1)
```

```
## [1] 80
```

```
#standard deviation
sd(e1)
```

```
## [1] 21.93731
```

```
# variance
var(e1)
```

```
## [1] 481.2456
```

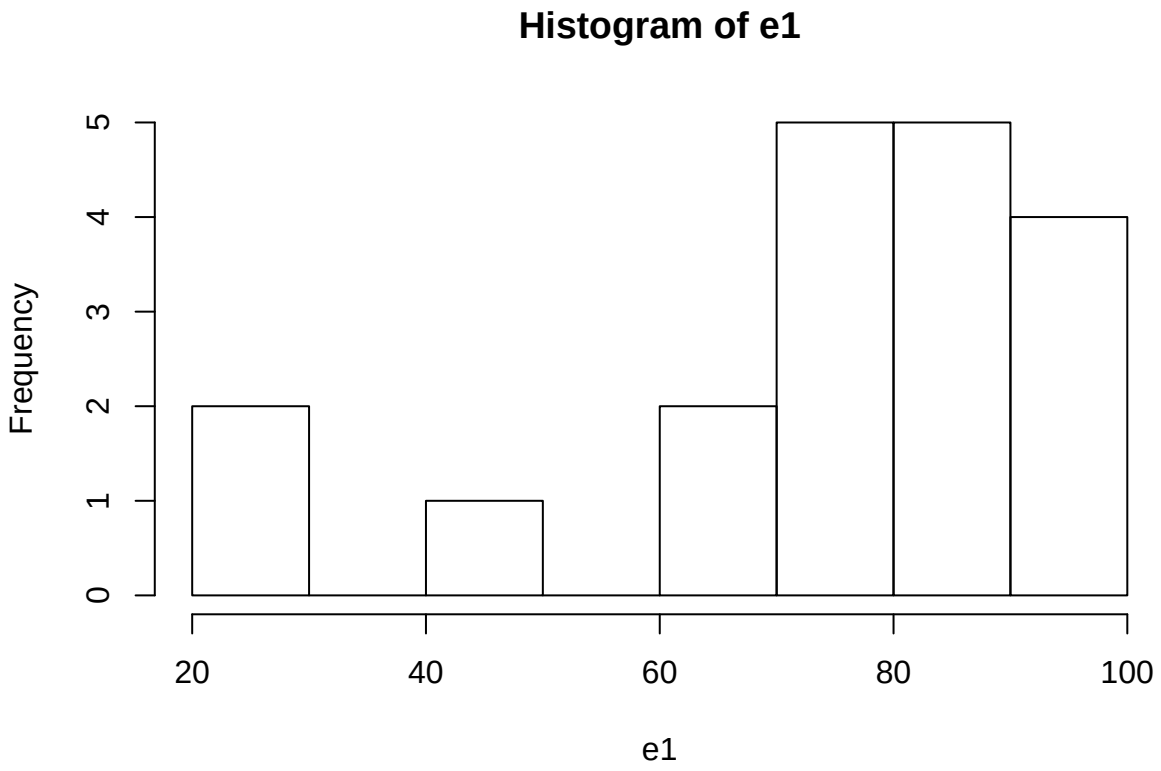
```
# IQR
IQR(e1)
```

```
## [1] 17
```

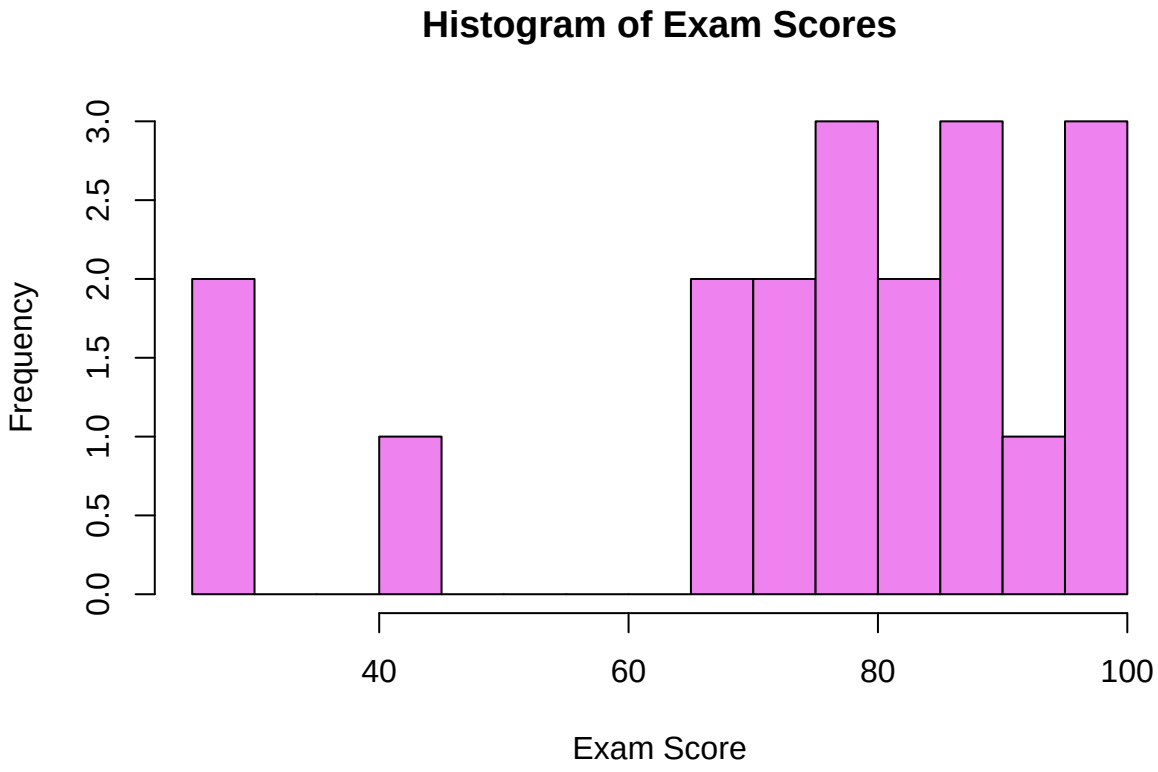
Graphics

R has some built in functions that cover basic graphics, like histograms, boxplots and scatterplots. You can customize them to have labels, thicker or thinner lines, and different colors.

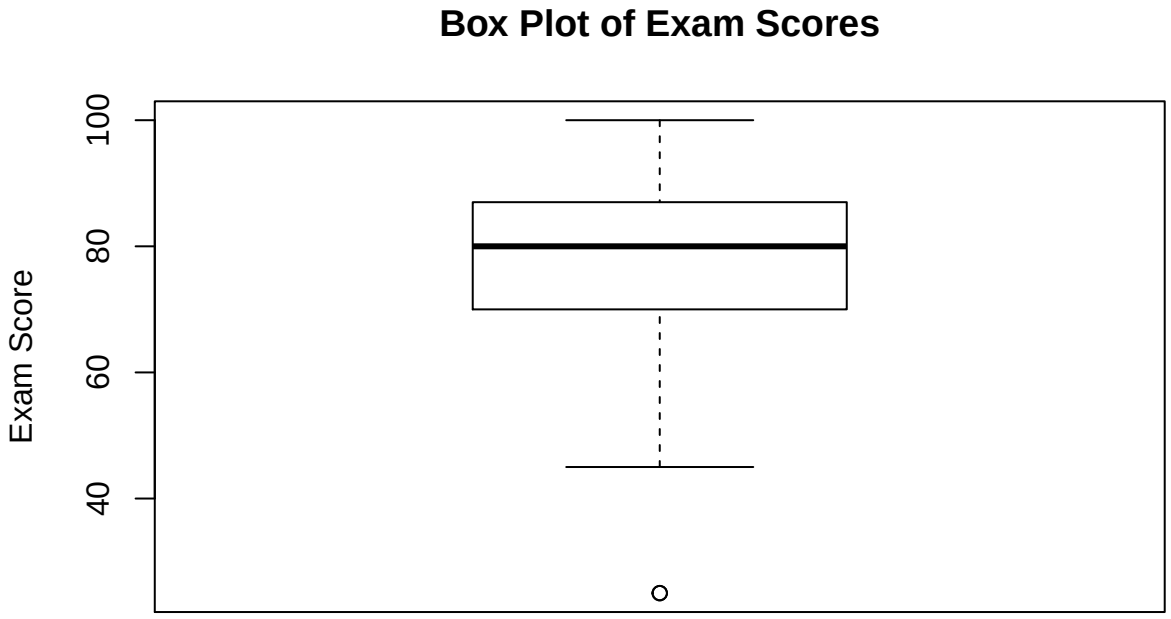
```
# histogram with no customizations
hist(e1)
```



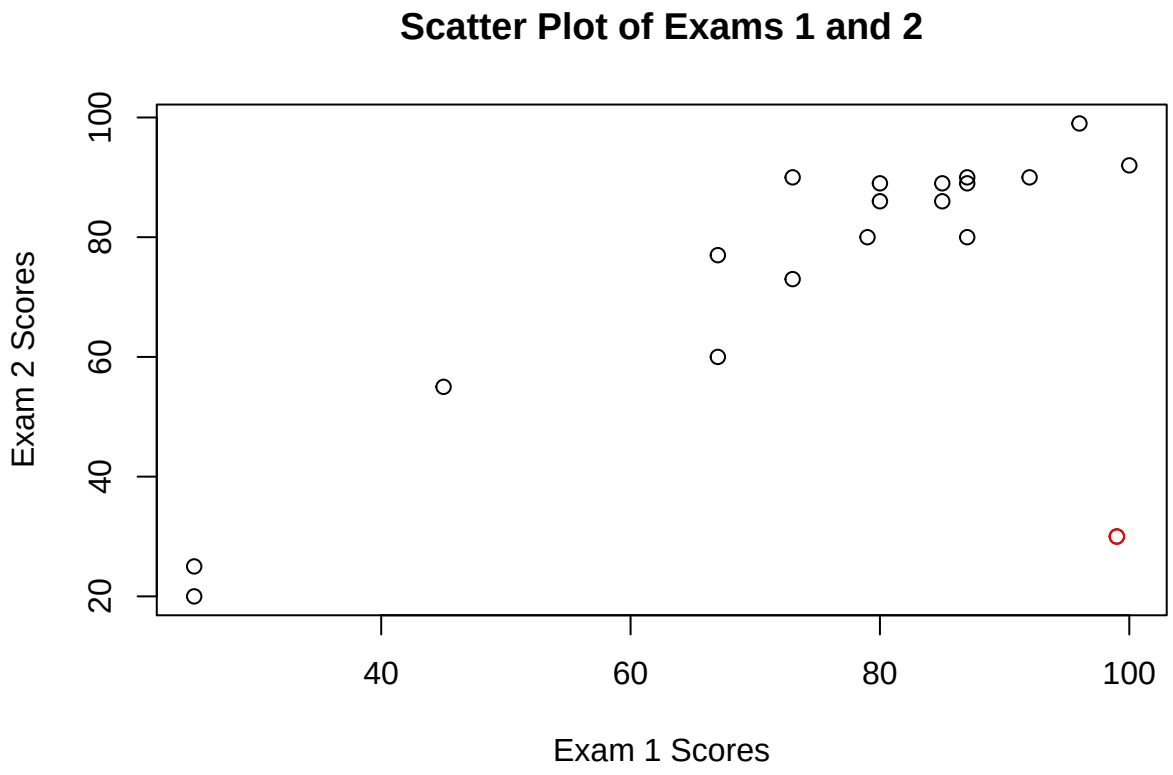
```r
# histogram with customizations
hist(e1,
     # breaks changes the bin width
     breaks = 15,
     # main changes the title of the histogram
     main = "Histogram of Exam Scores",
     # xlab changes the label on the x axis
     xlab = "Exam Score",
     # ylab changes the label on the y axis
     ylab = "Frequency",
     # col changes the color of the histogram
     col = "violet")
```



```r
# box plot
boxplot(e1, main = "Box Plot of Exam Scores",ylab = "Exam Score")
```



```r
# scatter plot
plot(df,
     main = "Scatter Plot of Exams 1 and 2",
     xlab = "Exam 1 Scores",
     ylab = "Exam 2 Scores")
# we can select a point, like an outlier, and highlight it in a different color
points(df[10,],col="red")
```



Special Topics

Working Directories

R is always using a "directory" on your computer: it's the place where it saves files and picks files from without assigning a filepath. If you want to know what directory your computer is working from just use the get working directory command,

```
getwd()
```

```
## [1] "/home/tina/Desktop/2020Math109"
```

If you want to change the directory use the set working directory command. The syntax here has to be precise: you need to use quotation marks, correct capitalization, have the correct file path, and use forward slashes. Then you can get the working directory again to check and make sure that you did it correctly.

```
setwd("/home/tina/Desktop/2020Math109")
```

Packages

Sometimes we want to do something a little more complex than just create a histogram or find some basic summary statistics. Since R has been around a while and is open source people have developed supplimentary "packages." The packages include functions that don't come with basic R.

We will use the `lattice` package as an example. Most useful packages come with good documentation, so the documentation that comes with `lattice` is here:

https://cran.r-project.org/web/packages/lattice/lattice.pdf

In the console, tell R to install the package. This only needs to be done once. If you are using RStudio, then the CRAN mirror will be chosen automatically. In R you will have to pick the CRAN mirror, and just like when you installed R, just pick a location close to you.

```
install.packages("lattice")
```

If the installation is successful, you will get a message like this,

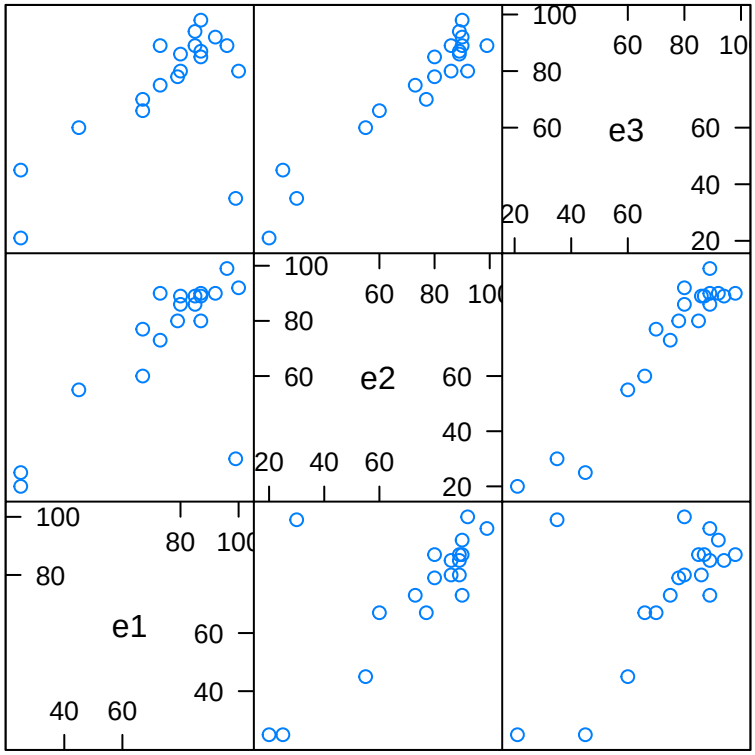
```
package 'lattice' successfully unpacked and MD5 sums checked The downloaded binary packages are in your filepath
```

To use the package you need to tell the computer that you want to use that package with the command `library(package name)` or `require(package name)`.

```
library(lattice)
```

Now we can use some of the things that this package has to offer. Suppose now that I have data for a third exam, and I want to look at the scatter plots comparing exam 1 and 2, exam 1 and 3 and exam 2 and 3 all at the same time. The function `splom` which stands for "scatter plot matrix" from the package `lattice` allows us to do just that.

```r
# assign the third vector of data
e3=c(80,21,89,75,70,80,85,94,89,35,60,78,98,45,66,86,87,89,92)
# create a data matrix
exam=cbind(e1,e2,e3)
# create the scatter plot matrix
splom(exam)
```



Scatter Plot Matrix

Functions

If we can't find a package or built in function that will perform a desired function, we can write our own. Suppose I wanted to define a function $f(x)$ such that

$$f(x) = \sqrt{|x - 20|}.$$

No such function exists in R, so we have to create our own with the `function` function.

```r
# name the function my.f
# inside the parantheses we tell the function what we will input
my.f=function(x){
  f=sqrt(abs(x-20)) # the operation the function performs
  print(f) # what the function outputs
}
# try your new function and find f(45)
my.f(-45)
```

```
## [1] 8.062258
```

There are plenty of customizations you can make to functions, like output multiple values.