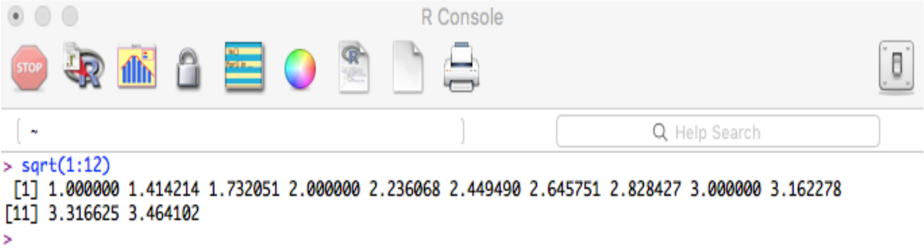
Marcus Crowder

September 2nd 2017

Lecture Notes 1 HW

1.1 Let *n* be equal to the total number of letters in your middle and last name. *If that total is less than 10, add 17 to the total.* Create a single line R command that will return the square roots of all the natural numbers from 1 to *n*.

Answer: My First Name is "Marcus" and my middle name is "Kellon" which makes n = 12.
My one line R command was
Command: sqrt(1:12)
This command uses the sqrt function in conjunction with the slicing mechanism implemented in R to print out all the square roots of the desired values.

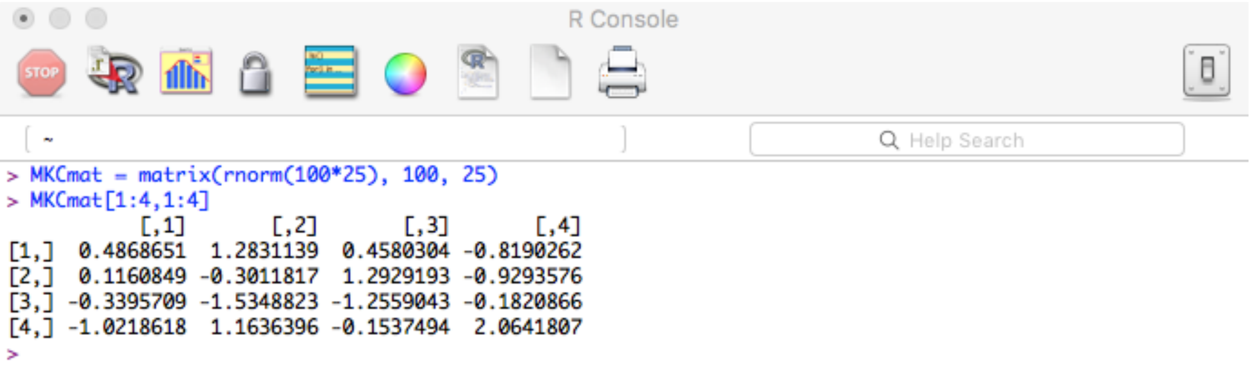


```
> sqrt(1:12)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000 3.162278
[11] 3.316625 3.464102
>
```

1.2 Create a matrix with 100 rows and 25 columns, filling it with standard Normal deviates. (Thus, you will need to place a call for 100*25 deviates.)

Commands: MKCmat = matrix(rnorm(100*25), 100, 25)
MKCmat[1:4,1:4]

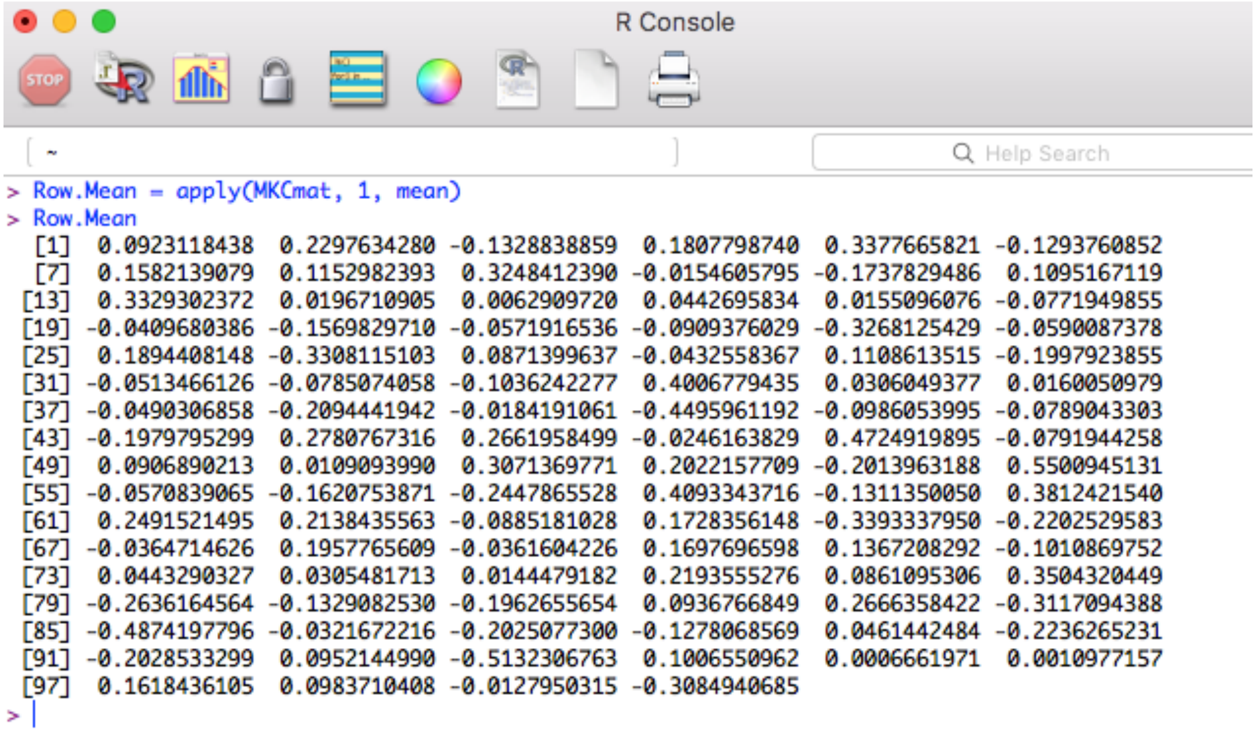
Answer: I used the matrix function to create a 100 by 25 matrix with 2500 values of the Standard Normal Deviates.



```
> MKCmat = matrix(rnorm(100*25), 100, 25)
> MKCmat[1:4,1:4]
           [,1]       [,2]       [,3]       [,4]
[1,]  0.4868651  1.2831139  0.4580304 -0.8190262
[2,]  0.1160849 -0.3011817  1.2929193 -0.9293576
[3,] -0.3395709 -1.5348823 -1.2559043 -0.1820866
[4,] -1.0218618  1.1636396 -0.1537494  2.0641807
>
```

1.3 Using the *apply* function, find the row averages of XYZmat. Give that output a name.
Commands: apply(MKCmat, 1, mean)
              Row.Mean

Answer: By using the apply function I was able to find the means all the row values of my matrix. Using apply with “1” for the row and “mean” for mean.

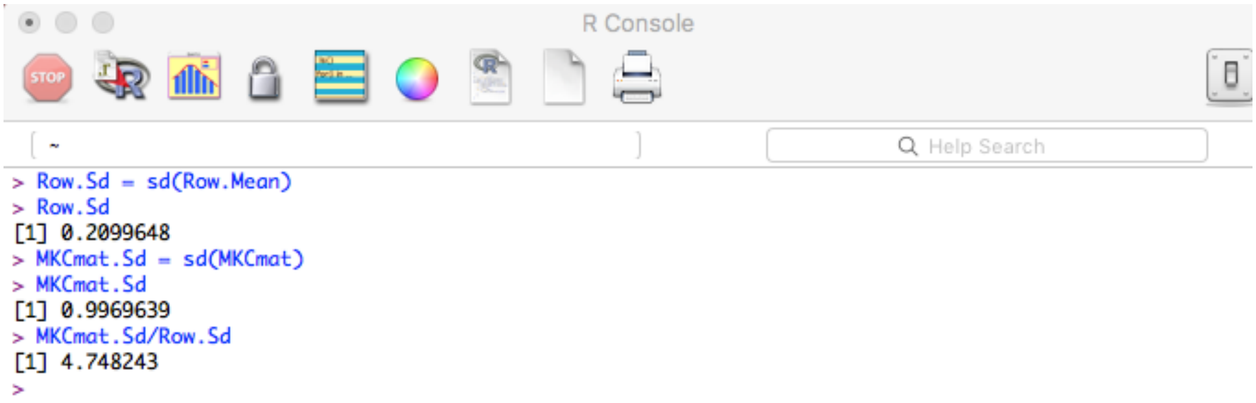


```
> Row.Mean = apply(MKCmat, 1, mean)
> Row.Mean
  [1]  0.0923118438  0.2297634280 -0.1328838859  0.1807798740  0.3377665821 -0.1293760852
  [7]  0.1582139079  0.1152982393  0.3248412390 -0.0154605795 -0.1737829486  0.1095167119
 [13]  0.3329302372  0.0196710905  0.0062909720  0.0442695834  0.0155096076 -0.0771949855
 [19] -0.0409680386 -0.1569829710 -0.0571916536 -0.0909376029 -0.3268125429 -0.0590087378
 [25]  0.1894408148 -0.3308115103  0.0871399637 -0.0432558367  0.1108613515 -0.1997923855
 [31] -0.0513466126 -0.0785074058 -0.1036242277  0.4006779435  0.0306049377  0.0160050979
 [37] -0.0490306858 -0.2094441942 -0.0184191061 -0.4495961192 -0.0986053995 -0.0789043303
 [43] -0.1979795299  0.2780767316  0.2661958499 -0.0246163829  0.4724919895 -0.0791944258
 [49]  0.0906890213  0.0109093990  0.3071369771  0.2022157709 -0.2013963188  0.5500945131
 [55] -0.0570839065 -0.1620753871 -0.2447865528  0.4093343716 -0.1311350050  0.3812421540
 [61]  0.2491521495  0.2138435563 -0.0885181028  0.1728356148 -0.3393337950 -0.2202529583
 [67] -0.0364714626  0.1957765609 -0.0361604226  0.1697696598  0.1367208292 -0.1010869752
 [73]  0.0443290327  0.0305481713  0.0144479182  0.2193555276  0.0861095306  0.3504320449
 [79] -0.2636164564 -0.1329082530 -0.1962655654  0.0936766849  0.2666358422 -0.3117094388
 [85] -0.4874197796 -0.0321672216 -0.2025077300 -0.1278068569  0.0461442484 -0.2236265231
 [91] -0.2028533299  0.0952144990 -0.5132306763  0.1006550962  0.0006661971  0.0010977157
 [97]  0.1618436105  0.0983710408 -0.0127950315 -0.3084940685
>
```

1.4 Using the *sd* function,

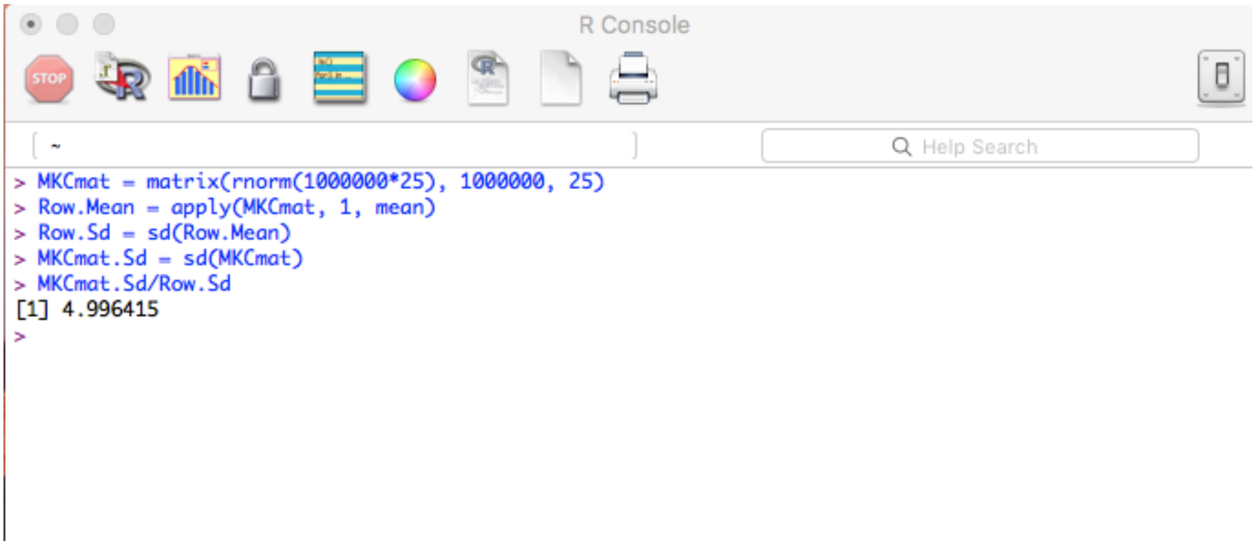
(a) find the sample standard deviation of your row averages from above;
(b) find the sample standard deviation for XYZmat as a whole;
(c) Divide the sd of the XYZmat by the sd of the row averages. Statistical theory says that ratio will *on average* be close to the square root of the sample size, which is *n*, here.
(d) **Do your results agree with theory?** (That is a non-trivial question.) Show your work.

Answer: To find the sample standard deviated of the row averages I used the sd(Row.Mean) command. And the sd(MKCmat.Sd) to find the standard deviation of MKCmat as a whole. When dividing the the sd of MKCmat by my row averages my answer came out to 4.748243. Assuming the n being refered too are the number of columns then sqrt(100) would equal to 10 or more than 2x my answer. In conclusion I do not agree that my results agree with theory.



```
> Row.Sd = sd(Row.Mean)
> Row.Sd
[1] 0.2099648
> MKCmat.Sd = sd(MKCmat)
> MKCmat.Sd
[1] 0.9969639
> MKCmat.Sd/Row.Sd
[1] 4.748243
>
```

1.5 Try it again, but with *one million rows.* (Hold onto your XYZmat for the next problem.) Repeat the steps in 1.4 and see if your results agree with theory.
The commands are similar to my Answer above with the exception of rewriting MKCmat to hold one million rows. Again my results do not match up with theory if my sample size is one million and my answer comes out to be closed to 5.



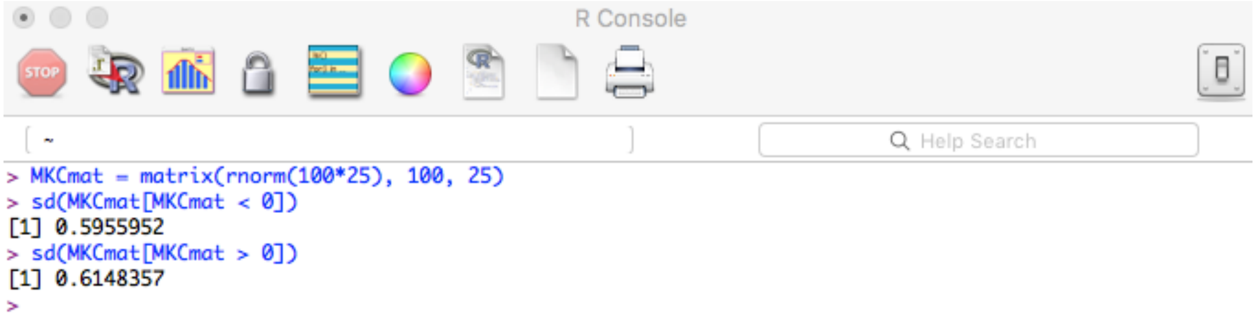
```
> MKCmat = matrix(rnorm(1000000*25), 1000000, 25)
> Row.Mean = apply(MKCmat, 1, mean)
> Row.Sd = sd(Row.Mean)
> MKCmat.Sd = sd(MKCmat)
> MKCmat.Sd/Row.Sd
[1] 4.996415
>
```

1.6 For XYZmat, find the sample mean and the sample standard deviation for

(a) the negative values within XYZmat; and

(b) for the non-negative values. Show your command and results.

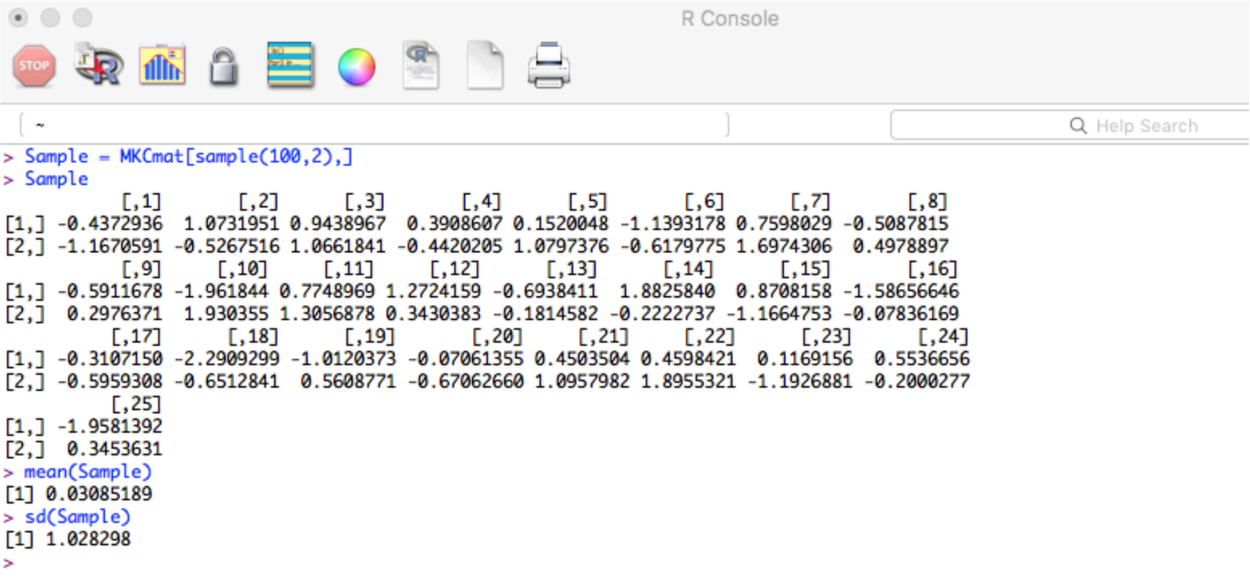
Answer: I generated the results by creating a normalized matrix of 100 rows and 25 columns. Then using the sample deviation function (sd) for our matrix MKCmat but for only the values below <0 or only negative values. A similar solution is used for the non-negative values by switching the operand around to >0.



```
> MKCmat = matrix(rnorm(100*25), 100, 25)
> sd(MKCmat[MKCmat < 0])
[1] 0.5955952
> sd(MKCmat[MKCmat > 0])
[1] 0.6148357
>
```

1.7 Find the sample mean and sample standard deviation of two *randomly selected* rows of XYZmat. Show your commands and results.

Answer: By using the sample function within the matrix we were able to generate a separate matrix with only 2 randomly selected Rows. Then by using the mean() and sd() function I was able to fine the Mean and Standand deviation of the rows selected.



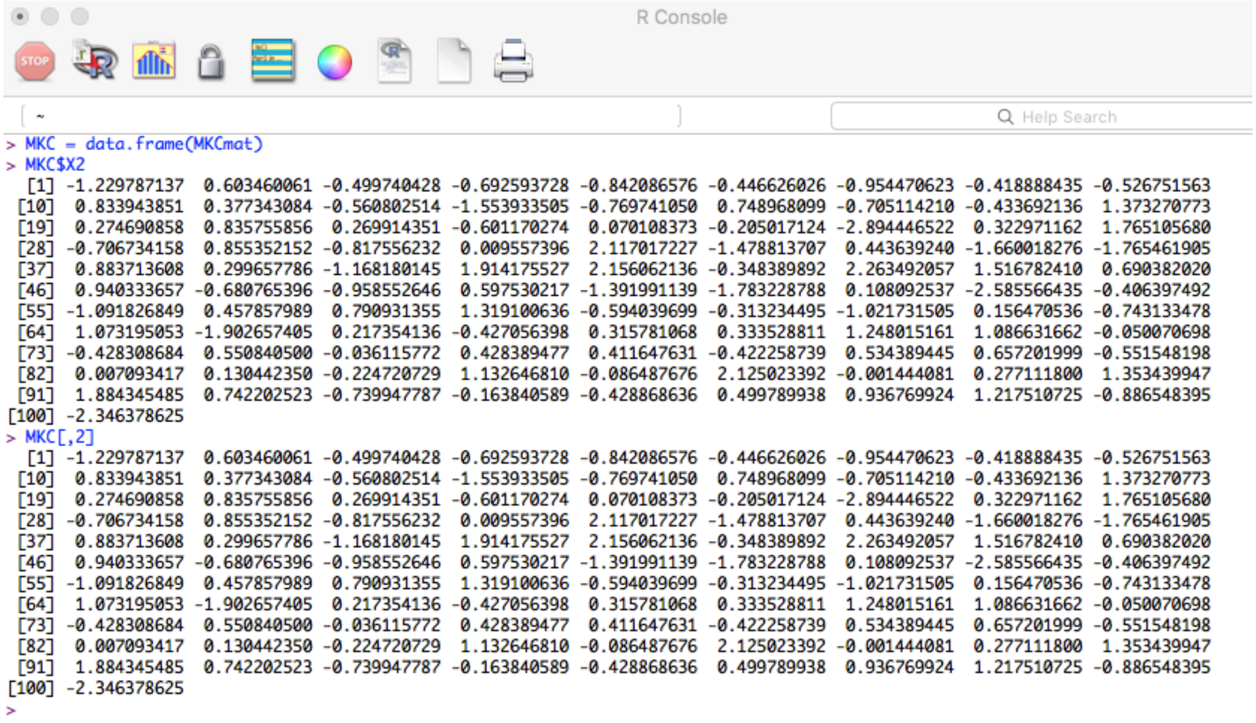
```
> Sample = MKCmat[sample(100,2),]
> Sample
           [,1]       [,2]      [,3]       [,4]      [,5]       [,6]      [,7]       [,8]
[1,] -0.4372936  1.0731951 0.9438967  0.3908607 0.1520048 -1.1393178 0.7598029 -0.5087815
[2,] -1.1670591 -0.5267516 1.0661841 -0.4420205 1.0797376 -0.6179775 1.6974306  0.4978897
           [,9]      [,10]     [,11]     [,12]      [,13]      [,14]      [,15]       [,16]
[1,] -0.5911678 -1.961844 0.7748969 1.2724159 -0.6938411  1.8825840  0.8708158 -1.58656646
[2,]  0.2976371  1.930355 1.3056878 0.3430383 -0.1814582 -0.2222737 -1.1664753 -0.07836169
          [,17]      [,18]      [,19]       [,20]     [,21]     [,22]      [,23]      [,24]
[1,] -0.3107150 -2.2909299 -1.0120373 -0.07061355 0.4503504 0.4598421  0.1169156  0.5536656
[2,] -0.5959308 -0.6512841  0.5608771 -0.67062660 1.0957982 1.8955321 -1.1926881 -0.2000277
          [,25]
[1,] -1.9581392
[2,]  0.3453631
> mean(Sample)
[1] 0.03085189
> sd(Sample)
[1] 1.028298
>
```

1.8 Convert XYZmat into a data frame. Show that you can call the second column by

(a) using the $ method and

(b) the [ ] method.

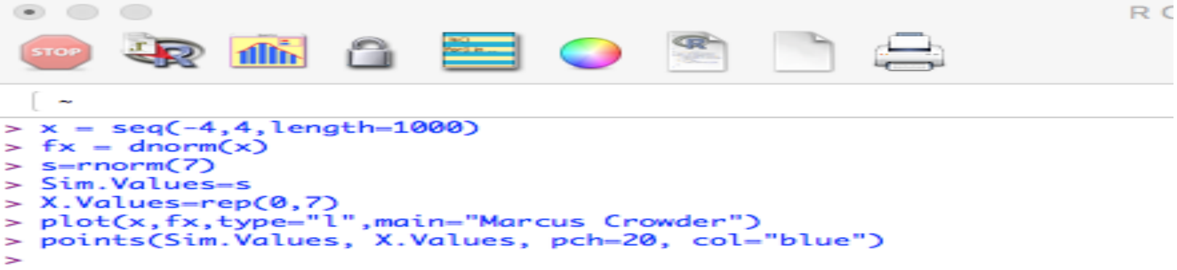
Answer: I converted my matrix into a data frame using the data.frame() function. Then used $ and [] selection methods to grab column 2.



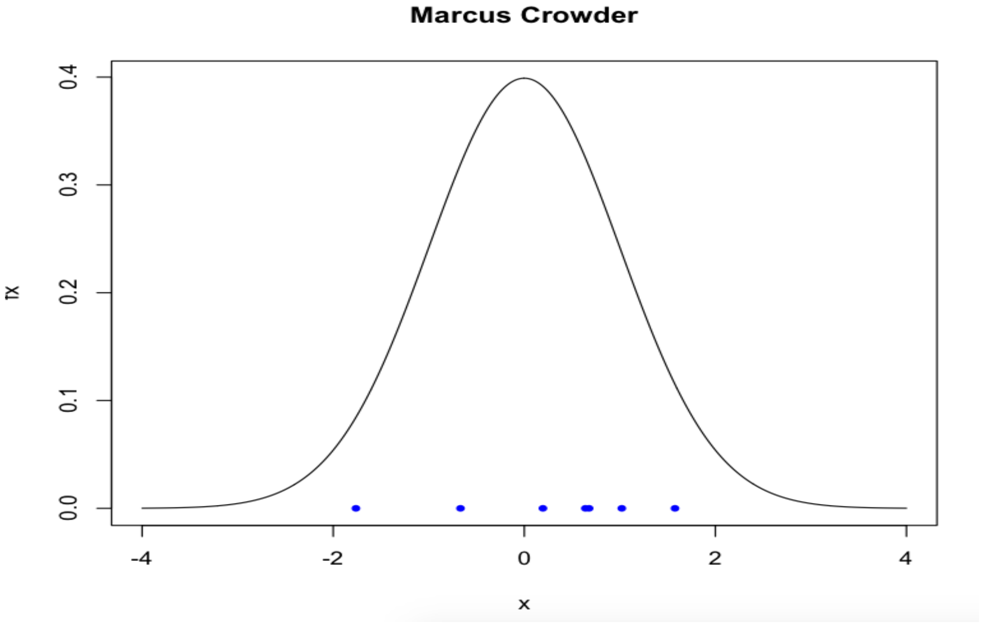
```
> MKC = data.frame(MKCmat)
> MKC$X2
  [1] -1.229787137  0.603460061 -0.499740428 -0.692593728 -0.842086576 -0.446626026 -0.954470623 -0.418888435 -0.526751563
 [10]  0.833943851  0.377343084 -0.560802514 -1.553933505 -0.769741050  0.748968099 -0.705114210 -0.433692136  1.373270773
 [19]  0.274690858  0.835755856  0.269914351 -0.601170274  0.070108373 -0.205017124 -2.894446522  0.322971162  1.765105680
 [28] -0.706734158  0.855352152 -0.817556232  0.009557396  2.117017227 -1.478813707  0.443639240 -1.660018276 -1.765461905
 [37]  0.883713608  0.299657786 -1.168180145  1.914175527  2.156062136 -0.348389892  2.263492057  1.516782410  0.690382020
 [46]  0.940333657 -0.680765396 -0.958552646  0.597530217 -1.391991139 -1.783228788  0.108092537 -2.585566435 -0.406397492
 [55] -1.091826849  0.457857989  0.790931355  1.319100636 -0.594039699 -0.313234495 -1.021731505  0.156470536 -0.743133478
 [64]  1.073195053 -1.902657405  0.217354136 -0.427056398  0.315781068  0.333528811  1.248015161  1.086631662 -0.050070698
 [73] -0.428308684  0.550840500 -0.036115772  0.428389477  0.411647631 -0.422258739  0.534389445  0.657201999 -0.551548198
 [82]  0.007093417  0.130442350 -0.224720729  1.132646810 -0.086487676  2.125023392 -0.001444081  0.277111800  1.353439947
 [91]  1.884345485  0.742202523 -0.739947787 -0.163840589 -0.428868636  0.499789938  0.936769924  1.217510725 -0.886548395
[100] -2.346378625
> MKC[,2]
  [1] -1.229787137  0.603460061 -0.499740428 -0.692593728 -0.842086576 -0.446626026 -0.954470623 -0.418888435 -0.526751563
 [10]  0.833943851  0.377343084 -0.560802514 -1.553933505 -0.769741050  0.748968099 -0.705114210 -0.433692136  1.373270773
 [19]  0.274690858  0.835755856  0.269914351 -0.601170274  0.070108373 -0.205017124 -2.894446522  0.322971162  1.765105680
 [28] -0.706734158  0.855352152 -0.817556232  0.009557396  2.117017227 -1.478813707  0.443639240 -1.660018276 -1.765461905
 [37]  0.883713608  0.299657786 -1.168180145  1.914175527  2.156062136 -0.348389892  2.263492057  1.516782410  0.690382020
 [46]  0.940333657 -0.680765396 -0.958552646  0.597530217 -1.391991139 -1.783228788  0.108092537 -2.585566435 -0.406397492
 [55] -1.091826849  0.457857989  0.790931355  1.319100636 -0.594039699 -0.313234495 -1.021731505  0.156470536 -0.743133478
 [64]  1.073195053 -1.902657405  0.217354136 -0.427056398  0.315781068  0.333528811  1.248015161  1.086631662 -0.050070698
 [73] -0.428308684  0.550840500 -0.036115772  0.428389477  0.411647631 -0.422258739  0.534389445  0.657201999 -0.551548198
 [82]  0.007093417  0.130442350 -0.224720729  1.132646810 -0.086487676  2.125023392 -0.001444081  0.277111800  1.353439947
 [91]  1.884345485  0.742202523 -0.739947787 -0.163840589 -0.428868636  0.499789938  0.936769924  1.217510725 -0.886548395
[100] -2.346378625
>
```

1.9 Create a graph like that on page 3, but with a sample size of n=7 and with your name in the title to the graph. Do your points fall symmetrically? Do you think you could infer the shape of the distribution those points came from by looking at just those points? Why is this of concern?

Answer: By using the example to create a graph given, I created bell curve distribution with 7 simulated plot points. My points were not symmetrical with points leading to the right. I do not believe I would be able to tell the shape of the distribution these points originated from. This is concerning because knowing the shape of the curve would produce better observations.

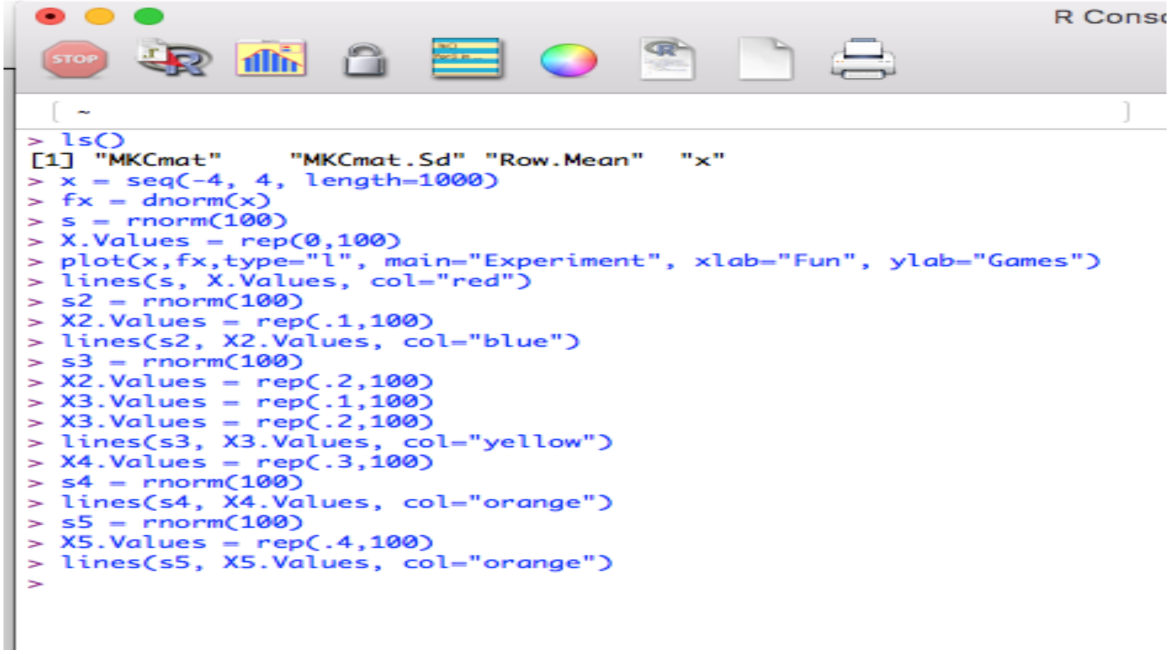


```
[ ~
> x = seq(-4,4,length=1000)
> fx = dnorm(x)
> s=rnorm(7)
> Sim.Values=s
> X.Values=rep(0,7)
> plot(x,fx,type="l",main="Marcus Crowder")
> points(Sim.Values, X.Values, pch=20, col="blue")
>
```

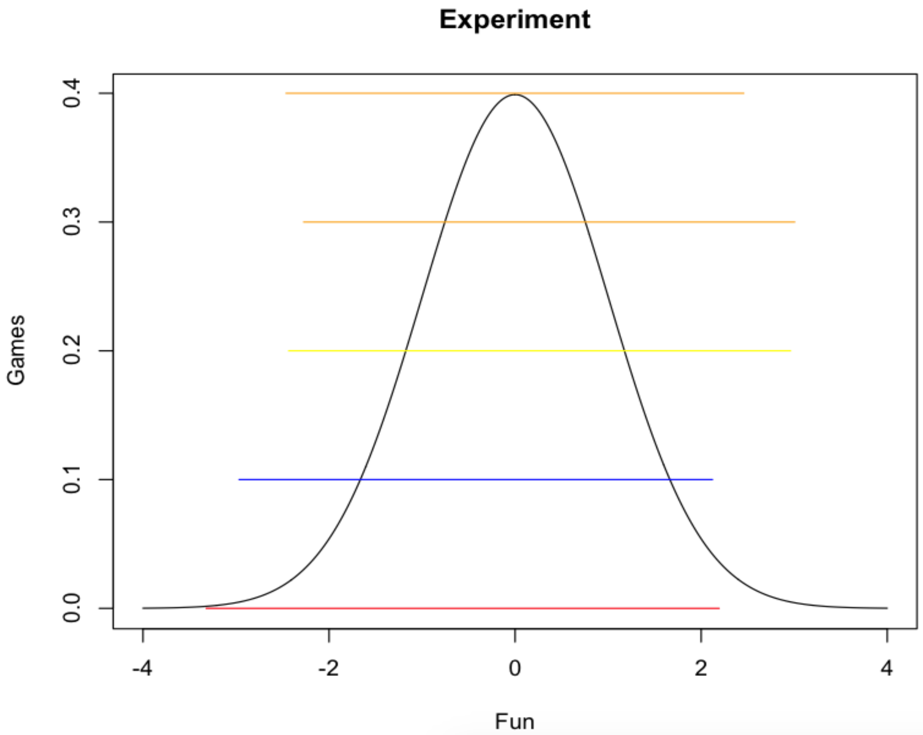


1.10.

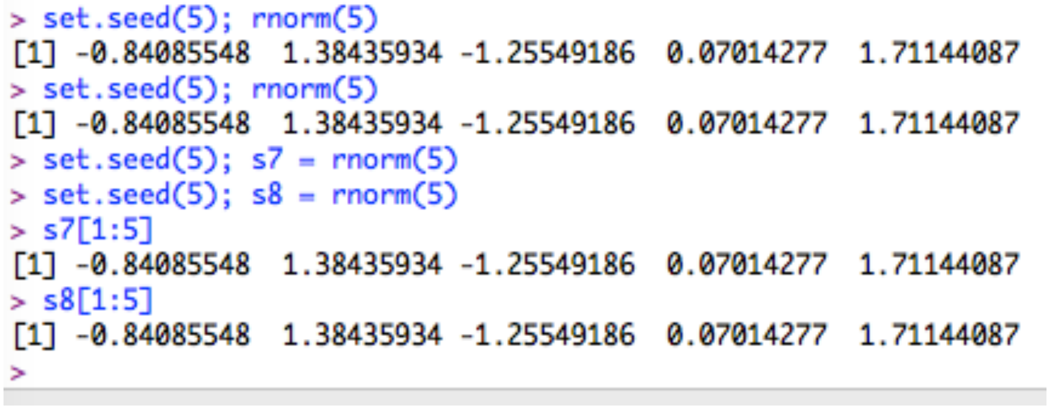
I decided to look into the API of the plot function to emphasize the randomness that can be found when plotting random points or in my case lines. I used the line function to plot 5 lines on my graph with different colors. To no surprise the lines of the graph do not seem in place with each other. I then researched how to make random numbers less random (for future debugging purposes) and got experience using the set.seed() which makes it possible to reproduce random numbers that were generated.



```
> ls()
[1] "MKCmat"      "MKCmat.Sd" "Row.Mean"   "x"
> x = seq(-4, 4, length=1000)
> fx = dnorm(x)
> s = rnorm(100)
> X.Values = rep(0,100)
> plot(x,fx,type="l", main="Experiment", xlab="Fun", ylab="Games")
> lines(s, X.Values, col="red")
> s2 = rnorm(100)
> X2.Values = rep(.1,100)
> lines(s2, X2.Values, col="blue")
> s3 = rnorm(100)
> X2.Values = rep(.2,100)
> X3.Values = rep(.1,100)
> X3.Values = rep(.2,100)
> lines(s3, X3.Values, col="yellow")
> X4.Values = rep(.3,100)
> s4 = rnorm(100)
> lines(s4, X4.Values, col="orange")
> s5 = rnorm(100)
> X5.Values = rep(.4,100)
> lines(s5, X5.Values, col="orange")
>
```



set.seed(5)



```
> set.seed(5); rnorm(5)
[1] -0.84085548  1.38435934 -1.25549186  0.07014277  1.71144087
> set.seed(5); rnorm(5)
[1] -0.84085548  1.38435934 -1.25549186  0.07014277  1.71144087
> set.seed(5); s7 = rnorm(5)
> set.seed(5); s8 = rnorm(5)
> s7[1:5]
[1] -0.84085548  1.38435934 -1.25549186  0.07014277  1.71144087
> s8[1:5]
[1] -0.84085548  1.38435934 -1.25549186  0.07014277  1.71144087
>
```

Information for question 10 found on https://www.programiz.com/r-programming/plot-function
And usage for the set.seed() function found on stack overflow(of-course)
https://stackoverflow.com/questions/13605271/reasons-for-using-the-set-seed-function