Week 1 – R Warm-Ups for Finance

Imagine this . . .

You work for the division president of an aerospace company that makes testing equipment for high-tech manufacturers. Orders arrive "lumpy" at best as some quarters are big producers, others stretch the company's credit revolver.

The president, call her Nancy, found a new way to make money: lease equipment. This would help finance operations and smooth cash flow. You had a follow-on idea: build a captive insurance company to insure certain aspects of quality in the manufactured product.

"Nice ideas," Nancy quips. "Show me how we can make money."

For starters we want to borrow low and sell (leases) high! How?

1. The "money factor"

   - Sets the monthly payments
   - Depends on the length and frequency of payment of the lease
   - Also depends on the value of money at monthly forward borrowing rates

2. Residual value of the equipment

   - Uncertain based on competitors' innovations, demand for manufacturers' products, etc.
   - Uncertain based on quality of equipment at end of lease (is there a secondary market?)

Also consider

3. Portfolio of leases

- By maturity
- By Equipment class
- By customer segment

Try this...

Let's begin the modelling process:

Name three leasing cash flow components and financial considerations we might use to begin to build reasonable scenarios.

Thinking. . .

Result

Four cash flow elements with models from our time together in this course:

1. Lease payments

   - Week 4: Term structure of interest rates
   - Week 6: Credit risk of customers
   - Week 3: Impact of economy on customers' market value

2. Residual cashflow

   - Week 7: Operational risk
   - Week 10: Aggregating risks

3. Borrowing costs

   - Week 4: Term structure of interest rates
   - Week 6: Our own credit risk

In addition,

4. Collateral

   - Week 9: Portfolio optimization
   - Week 5: Market risk

5. Regulatory issues

   - Week 7: Operational risk

6. Shareholder tolerance for risk

   - Week 5: Market risk
   - Week 8: Hedging
   - Week 10: Capital requirements

What we are about

. . . to do that is. In this first week we will review some aspects of R programming to whet our appetite for further work in finance and risk. There are many, many compendia of programming practices available. Here we will focus on a few aspects that we will directly use later on.

Purpose

1. Build on pre-course practice with R
2. Use some finance theory with R
3. Introduce arrays using regression analysis

Process

1. Introduce R calculations, arrays, text handling, graphics
2. Review basic finance and statistics content
3. Use introductory R calculations in financial and statistical examples
4. Extend introductory calculations with further examples

Product

1. Handful of useful `R` statements
2. Present value calculator
3. Regression coefficients and standard error calculator

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE

Hot and cold running resources

Much is available in books, e-books, and online for free. This is an extensive online community that links expert and novice modelers globally.

The standard start-up is at CRAN

`http://cran.r-project.org/manuals.html`. A script in the appendix can be dropped into a workspace and played with easily.

Other resources include

- Julian Faraway's `https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf` complete course on regression where you can imbibe deeply of the many ways to use `R` in statistics.
- Along econometrics lines is Grant Farnsworth's `https://cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf`.
- Winston Chang's `http://www.cookbook-r.com/` and Hadley Wickham's example at `http://ggplot2.org/` are terrific online graphics resources.

Try this . . .

Navigate to this site:

Cookbook-R

1. Click `http://www.cookbook-r.com/`
2. Navigate to `Packages`
3. Install package `EIAdata`
4. Click `http://www.eia.gov/opendata/register.cfm`
5. Register and get your key
6. Report on the data you can retrieve

Thinking. . .

Result

We will be using data from this site to examine relationships among prices in the energy sector. These will be input to a market spread hedging routine we will build together in future weeks.

EIA Data examples

- 408,000 electricity series organized into 29,000 categories
- 30,000 State Energy Data System series organized into 600 categories
- 115,052 petroleum series and associated categories
- 34,790 US crude imports series and associated categories
- 11,989 natural gas series and associated categories
- 132,331 coal series and associated categories
- 3,872 Short-Term Energy Outlook series and associated categories
- 368,466 Annual Energy Outlook series and associated categories
- 92,836 International Energy series

Tickling the ivories

Or if you paint and draw, the 2-minute pose will warm you up. In the RStudioio console panel (in the SW pane of the IDE) play with these by typing these statements at the > symbol:

```
1 + (1:5)
```

```
## [1] 2 3 4 5 6
```

This will produce a vector from 2 to 6.

Use `alt-` (hold alt and hyphen keys down simultaneously) to produce <-, and assign data to an new object. This is a from `R`'s predecessor James Chamber's `S` (ATT Bell Labs) that was ported from the single keystroke ← in Ken Iverson's `APL` (IBM), where it is reserved as a binary logical operator. We can now also use = to assign variables in `R`. But, also a holdover from `APL`, we will continue to use = only for assignments within functions. [Glad we got that over!] - Now try these.

```
x <- 1 + (1:5)
sum(x)
```

```
## [1] 20
```

```
prod(x)
```

```
## [1] 720
```

This will assign a calculation to a variable `x`

and sum and multiply the elements. `x` is stored in the active workspace. You can verify that by typing `ls()` in the console to list the objects in the workspace.

```
ls()
```

```
## [1] "x"
```

```
length(x)
```

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

```
x[1:length(x)]
```

```
## [1] 2 3 4 5 6
```

```
x[6:8]
```

```
## [1] NA NA NA
```

```
x[6:8] <- 7:9
x/0
```

```
## [1] Inf Inf Inf Inf Inf Inf Inf Inf
```

```
(x1 <- x - 2)
```

```
## [1] 0 1 2 3 4 5 6 7
```

```
x1
```

```
## [1] 0 1 2 3 4 5 6 7
```

```
x/x1
```

```
## [1]      Inf 3.000000 2.000000 1.666667 1.500000 1.400000 1.333333 1.285714
```

`x` has length of 5 ...

- ... and we use that to index all of the current elements of `x`.
- Trying to access elements 6 to 8 produces `na` because they do not exist yet.
- Appending 7 to 9 will fill the spaces.
- Dividing by 0 produces `inf`.
- Putting parentheses around an expression is the same as printing out the result of the expression.
- Element-wise division (multiplication, addition, subtraction) produces `inf` as the first element.

Try this . . .

Suppose you have a gargleblaster machine that produces free cash flows of $10 million each year for 8 years. The machine will be scrapped and currently you believe you can get $5 million at the end of year 8 as salvage value. The forward curve of interest rates for the next 1 to 8 years is 0.06, 0.07, 0.05, 0.09, 0.09, 0.08, 0.08, 0.08.

1. What is the value of $1 received at the end of each of the next 8 years? Use this script to begin the process. Describe each calculation.

```
rates <- c(0.06, 0.07, 0.05, 0.09, 0.09,
    0.08, 0.08, 0.08)
t <- seq(1, 8)
(pv.1 <- sum(1/(1 + rates)^t))
```

2. What is the present value of salvage? Salvage would be at element 8 of an 8-element cash flow vector, and thus would use the eighth forward rate, `rate[8]`, and `t` would be 8 as well. Eliminate the sum in the above script. Make a variable called `salvage` and assign salvage value to this variable. Use this variable in place of the 1 in the above script for `pv.1`. Call the new present value `pv.salvage`.

3. What is the present value of the gargleblaster machine? Type in these statements. The `rep` function makes an 8 element cash flow vector. We change the value of the 8th element of the cash flow vector to include salvage. Now use the `pv.1` statement above and substitute `cashflow` for `1`. You will have your result.

```
cashflow <- rep(10, 8)
cashflow[8] <- cashflow[8] + salvage
```

Thinking. . .

Result

The present value of $1 is:

```
rates <- c(0.06, 0.07, 0.05, 0.09, 0.09,
    0.08, 0.08, 0.08)
t <- seq(1, 8)
(pv.1 <- sum(1/(1 + rates)^t))
```

```
## [1] 5.792958
```

The present value of salvage is

```
salvage <- 5
(pv.salvage <- salvage/(1 + rates[8])^8)
```

```
## [1] 2.701344
```

The present value of the gargleblaster machine is

```
cashflow <- rep(10, 8)
cashflow[8] <- cashflow[8] + salvage
(pv.machine <- sum(cashflow/(1 + rates)^t))
```

```
## [1] 60.63092
```

What just happened?

Write down your thoughts

Look at your notes for this last segment. Write down the two or three things you least understood from this section. This will help us prepare for the live sessions this week.

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE

Building Some Character

Type these into the console at the > prompt:

```
x[length(x) + 1] <- "end"
x[length(x) + 1] <- "end"
x.char <- x[-length(x)]
x <- as.numeric(x.char[-length(x.char)])
str(x)
```

```
##  num [1:8] 2 3 4 5 6 7 8 9
```

We have appended the string "end" to the end of x, twice.

- We use the - negative operator to eliminate it.
- By inserting a string of characters into a numeric vector we have forced `R` to transform all numerical values to characters.
- To keep things straight we called the character version `x.char`.
- In the end we convert `x.char` back to numbers that we check with the `str`(ucture) function.
- We will use this procedure when comparing distributions of variables such as stock returns.

Now for something completely different with characters.

Here's a useful set of statements for coding and classifying variables.

```
set.seed(1016)
n.sim <- 10
x <- rnorm(n.sim)
y <- x/(rchisq(x^2, df = 3))^0.5
# help('distribution') will give you
# lots more information
z <- c(x, y)
indicator <- rep(c("normal", "abnormal"),
    each = length(x))
xy.df <- data.frame(Variates = z, Distributions = indicator)
```

```r
head(xy.df, n = 5)
```

```
##     Variates Distributions
## 1  0.7773788        normal
## 2  1.3733067        normal
## 3  1.3025762        normal
## 4  0.1482796        normal
## 5 -1.8251426        normal
```

```r
tail(xy.df, n = 5)
```

```
##      Variates Distributions
## 16 -0.3918201      abnormal
## 17 -0.2841416      abnormal
## 18  0.2774302      abnormal
## 19 -0.7409946      abnormal
## 20  0.8757421      abnormal
```

We did a lot of `R` here. First, we set a random seed to reproduce the same results every time we run this simulaton. Then, we store the number of simulations in `n.sim` and produced two new variables with normal and a weirder looking distribution (almost a Student's t distribution). Invoking `help` will display help with distributions in the SE pane of the `RStudio` IDE. Next we concatenate the two variables into a new variable `z`.

We built into the variable `indicator` the classifier to indicate which is `x` and which is `y`. But let's visualize what we want. (Paint in words here.) We want a column the first `n.sim` elements of which are `x` and the second are `y`. We then want a column the first `n.sim` elements of which are indicated by the character string "normal", and the second `n.sim` elements by "abnormal".

The `rep` function replicates the concatenation of "normal" and "abnormal" 10 times (the `length(x)`). The `each` feature concatenates 10 replications of "normal" to 10 replications of "abnormal". We concatenate the variates into `xy` with the `c()` function.

We can see the first 5 components of the data frame components using the $ subsetting notation as below.

```
str(xy.df)
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ Variates     : num  0.777 1.373 1.303 0.148 -1.825 ...
##  $ Distributions: Factor w/ 2 levels "abnormal","normal": 2 2 2 2 2 2 2 2 2 2 ...
```

```
head(xy.df$Variates, n = 5)
```

```
## [1]  0.7773788  1.3733067  1.3025762  0.1482796 -1.8251426
```

```
head(xy.df$Distributions, n = 5)
```

```
## [1] normal normal normal normal normal
## Levels: abnormal normal
```

The `str` call returns the two vectors inside of `xy`. One is numeric and the other is a "factor" with two levels. `R` and many of the routines in `R` will interpret these as zeros and ones in developing indicator and dummy variables for regressions and filtering.

Try this...

1. We will want to see our handiwork, so load the `ggplot2` library.
2. Visit Hadley Wickham's examples at `http://ggplot2.org/`.
3. Report on your progress by pasting the `R` statement you used to install the package.

Thinking. . .

Result

This plotting package requires data frames. A data frame simply put is a list of vectors and arrays with names. We define a data frame `xy.df`. All of the variates are put into one part of the frame, and the distribution indicator into another. For all of this to work in a plot the two arrays must be of the same length. Thus we use the common `n.sim` and `length(x)` to insure this. We always examine the data, here using the `head` and `tail` functions.

What just happened?

Write it down

Look at your notes for this last segment. Write down the two or three things you least understood from this section. This will help us prepare for the live sessions this week.

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE

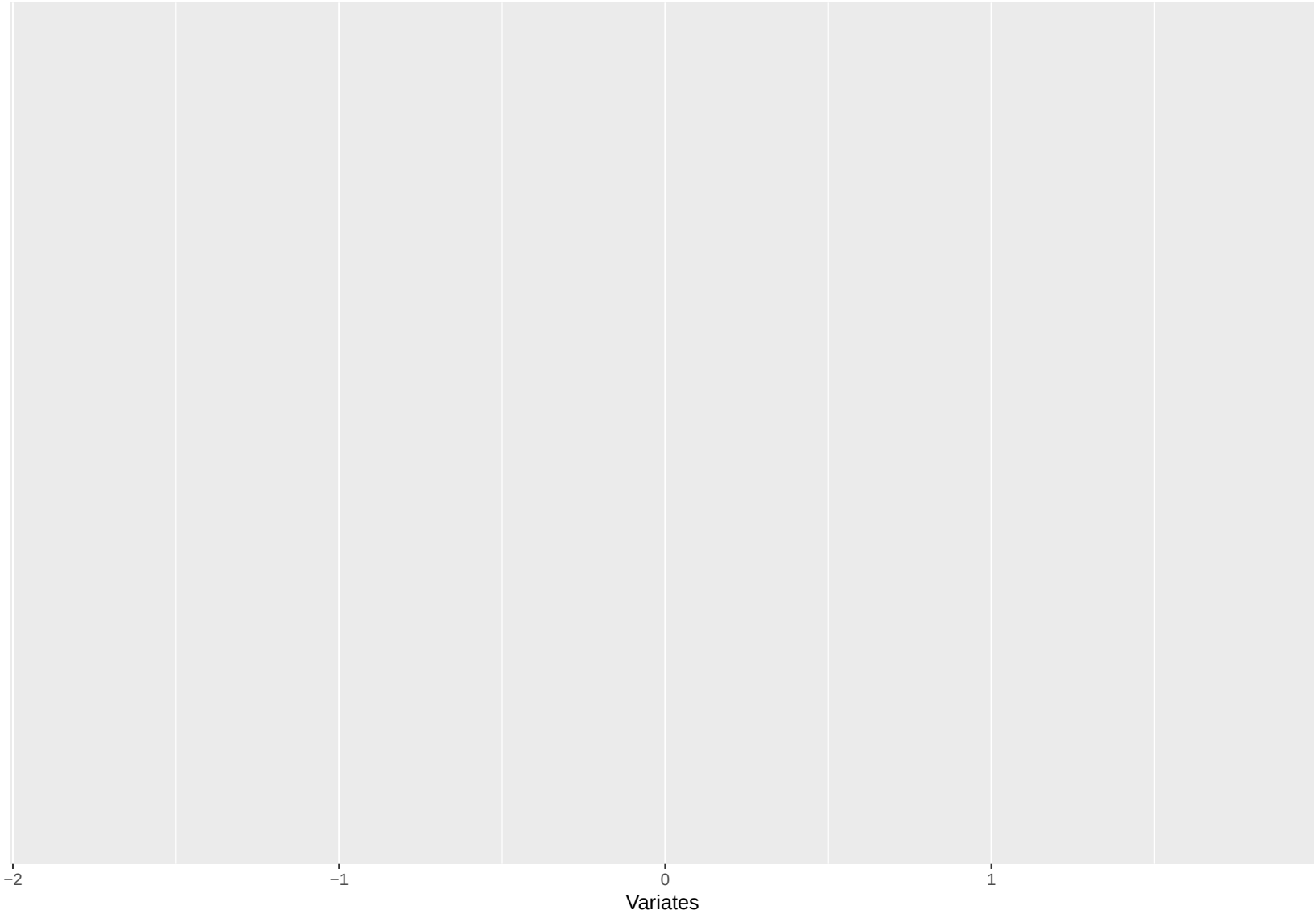
The plot thickens

Look at `help(ggplot)` for details. The `ggplot2` graphics package embodies Hadley Wickham's "grammar of graphics" we can review at `http://ggplot2.org`. Hadley Wickham has a very useful presentation with numerous examples at `http://ggplot2.org/resources/2007-past-present-future.pdf`.

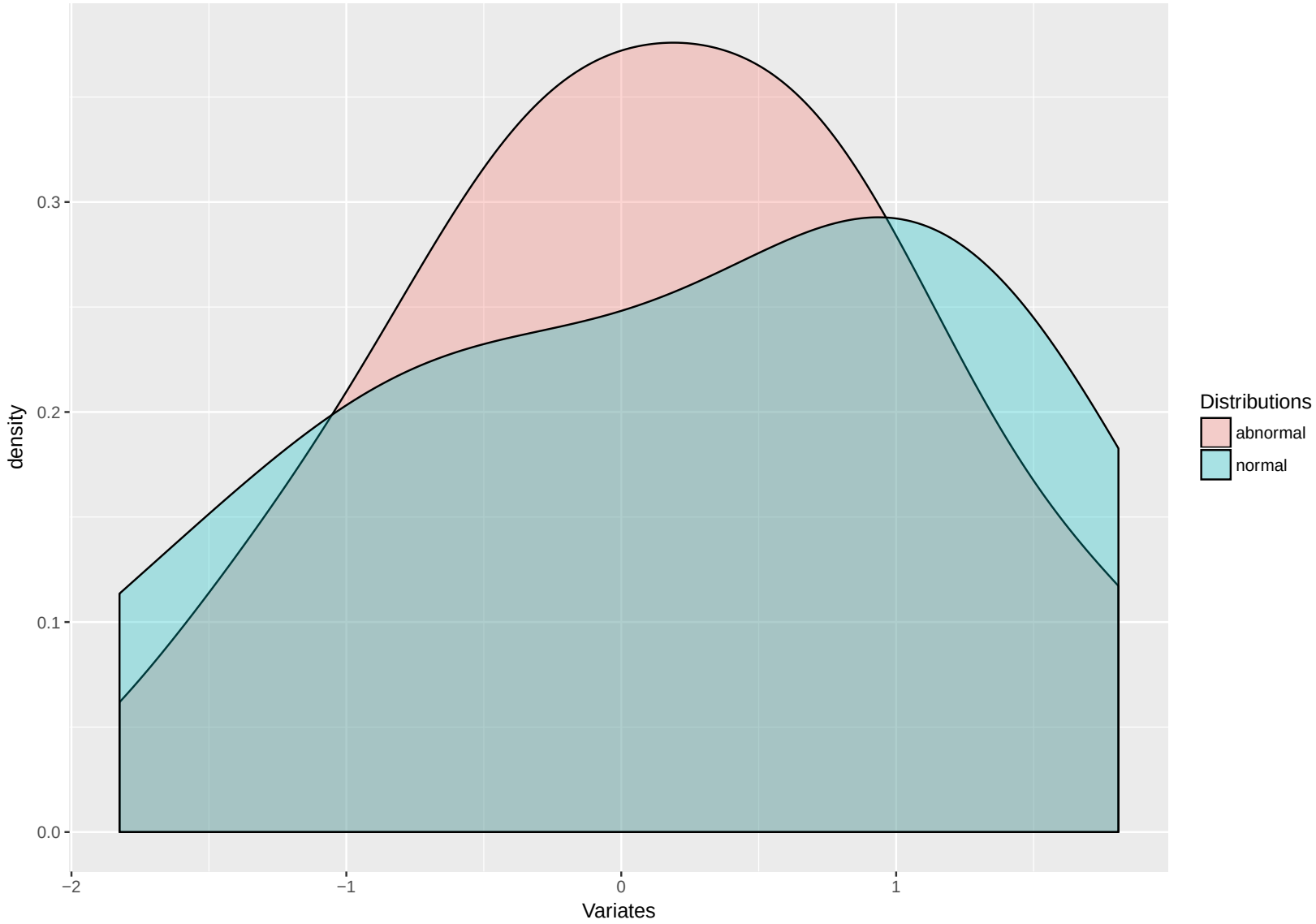
As mentioned above, the package uses data frames to process graphics. A lot of packages other than `ggplot2`, including the base `stats` package, require data frames.

We load the library first. The next statement sets up the blank but all too ready canvas (it will be empty!) on which a density plot can be rendered.

```
library(ggplot2)
ggplot(xy.df, aes(x = Variates, fill = Distributions))
```



The data frame is first followed by graphics grammar element #1 aesthetics mapping of data. The next statement inserts a geometrical element, here a density curve, which has a transparency parameter aesthetic `alpha`.



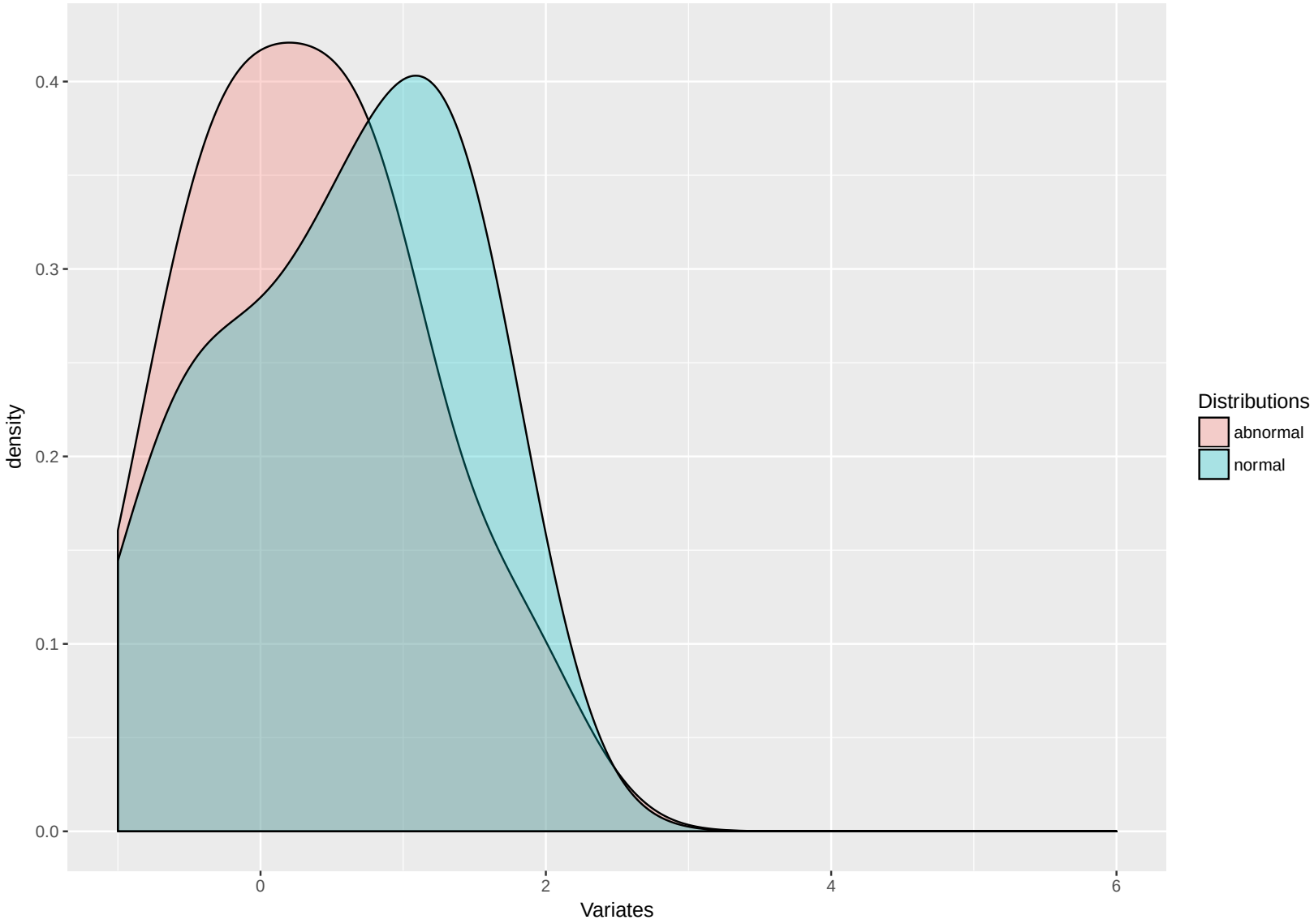
Try this on for size . . .

Zoom in with `xlim` and lower x-axis and upper x-axis limits using the following statement:

```
ggplot(xy.df, aes(x = Variates, fill = Distributions)) +
    geom_density(alpha = 0.3) + xlim(-1,
    6)
```

Thinking. . .

Results



Now we are getting to extreme finance!

What just happened?

Again write it down

Look at your notes for this last segment. Write down the two or three things you least understood from this section. This will help us prepare for the live sessions this week.

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE

An array of good things to come

Arrays have rows and columns and are akin to tables. All of Excel's worksheets are organized into cells that are tables with columns and rows. Data frames are more akin to tables in data bases. Here are some simple matrix arrays and functions. We start by making a mistake:

```
(A.error <- matrix(1:11, ncol = 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9    1
```

```r
(A.row <- matrix(1:12, ncol = 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
(A.col <- matrix(1:12, ncol = 4, byrow = FALSE))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

`A.error` throws an interesting error that we should all heed. In `A` we take 16 integers in a row and specify they be organized into 4 columns, and in `R` this is by row. `A.col` and column binding are equivalent.

```
(R <- rbind(1:4, 5:8, 9:12))  # Concatenate rows
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```
(C <- cbind(1:3, 4:6, 7:9, 10:12))  # concatenate columns
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
A.col == C
```

```
##      [,1] [,2] [,3] [,4]
## [1,] TRUE TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE TRUE
```

Using the `outer` product allows us to operate on matrix elements, first picking the minimum, then the maximum of each row. The `pmin` and `pmax` compare rows element by element. If you used `min` and `max` you would get the minimum and maximum of the whole matrix.

```
(A.min <- outer(3:6/4, 3:6/4, FUN = pmin))  #
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.75 0.75 0.75 0.75
## [2,] 0.75 1.00 1.00 1.00
## [3,] 0.75 1.00 1.25 1.25
## [4,] 0.75 1.00 1.25 1.50
```

```
(A.max <- outer(3:6/4, 3:6/4, FUN = pmax))  #
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.75 1.00 1.25  1.5
## [2,] 1.00 1.00 1.25  1.5
## [3,] 1.25 1.25 1.25  1.5
## [4,] 1.50 1.50 1.50  1.5
```

```
(A.sym <- A.max - A.min - 0.5)
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] -0.50 -0.25  0.00  0.25
## [2,] -0.25 -0.50 -0.25  0.00
## [3,]  0.00 -0.25 -0.50 -0.25
## [4,]  0.25  0.00 -0.25 -0.50
```

We build a symmetrical matrix and replace the diagonal with 1. `A.sym` looks like a correlation matrix. Here all we were doing is playing with shaping data.

```
diag(A.sym) <- 1
A.sym
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,]  1.00 -0.25  0.00  0.25
## [2,] -0.25  1.00 -0.25  0.00
## [3,]  0.00 -0.25  1.00 -0.25
## [4,]  0.25  0.00 -0.25  1.00
```

Try this . . .

The `inner` product %*% cross multiplies successive elements of a row with the successive elements of a column. If there are two rows with 5 columns, there must be a matrix at least with 1 column that has 5 rows in it. Run these statements.

```
n.sim <- 100
x.1 <- rgamma(n.sim, 0.5, 0.2)
x.2 <- rlnorm(n.sim, 0.15, 0.25)
X <- cbind(x.1, x.2)
```

1. Plot the histograms of each simulated random variate using `hist()`.
2. The `cbind` function binds into matrix columns the row arrays `x.1` and `x.2`. These might be simulations of operational and financial losses. Look up `rgamma` and `rlnorm` for more information.
3. The `X` matrix could look like the "design" matrix for a regression. Suppose we have a response vector, say equity, and call it `y` and look at its histogram.

```
y <- 1.5 * x.1 + 0.8 * x.2 + rnorm(n.sim,
    4.2, 5.03)
```

4. Now we have a model for y:

$$y = X\beta + \epsilon$$

where y is a 100×1 (rows $\times$ columns) vector, X is a 100×2 matrix, β is a 2×1 vector, and ϵ is a 100×1 vector of disturbances (a.k.a., "errors").

Multiplying out the matrix term $X\beta$ we have

$$y = \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where y, x_1, x_2, and ϵ are all vectors with 100 rows for simulated observations.

If we look for β to minimize the sum of squared ϵ we would find that the solution is

$$\hat{(}\beta) = (X^T X)^{-1} X^T y.$$

Where $\hat{(}\beta)$ is read as "beta hat".

Code this into your console. Calculate the estimated ϵ and run its histogram.

```
XTX.inverse <- solve(t(X) %*% X)
(beta.hat <- XTX.inverse %*% t(X) %*%
    y)
```

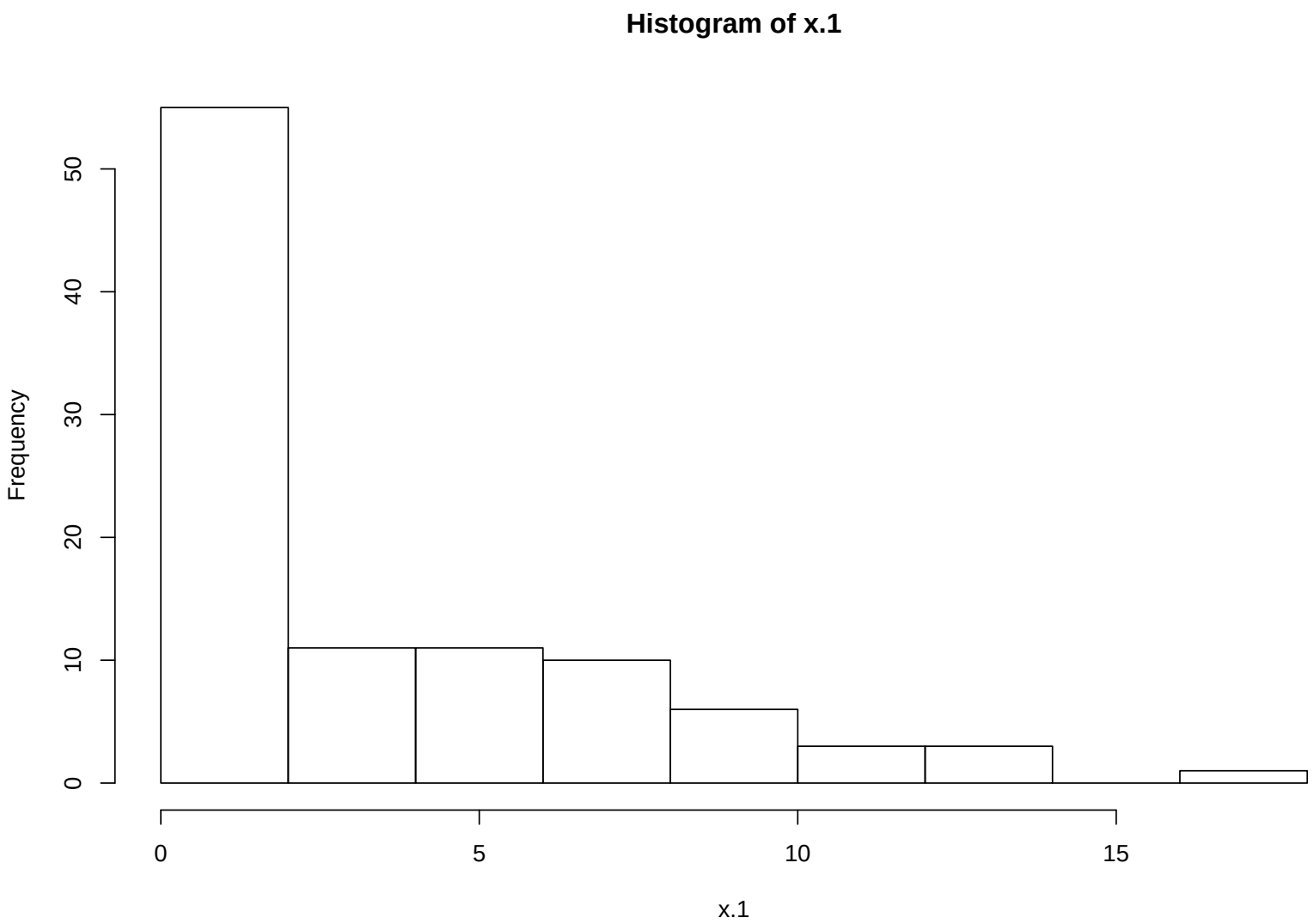
Thinking. . .

Results

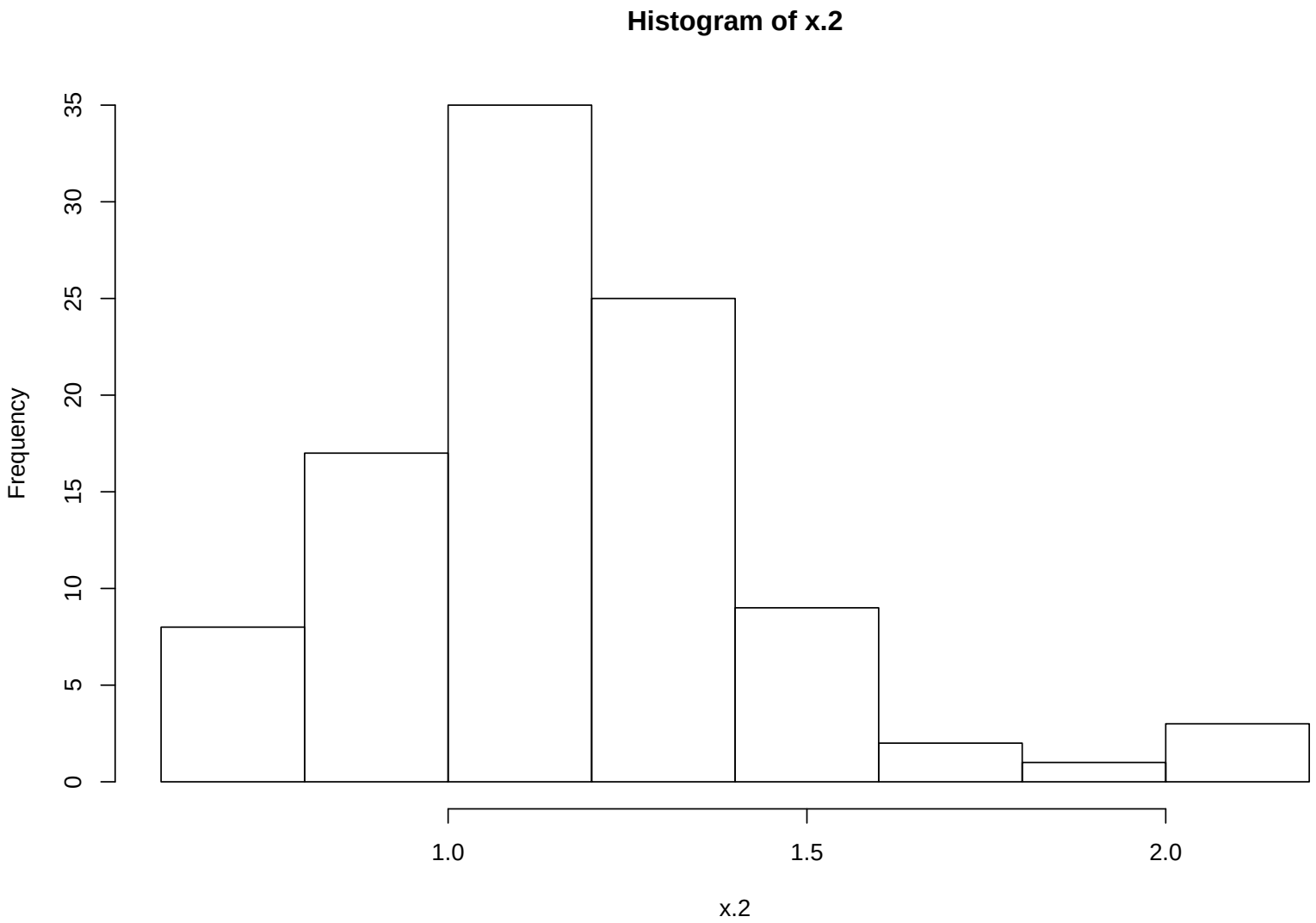
Histograms for the two variates:

```
set.seed(1016)
n.sim <- 100
x.1 <- rgamma(n.sim, 0.5, 0.2)
x.2 <- rlnorm(n.sim, 0.15, 0.25)
```

```
hist(x.1)
```



```
hist(x.2)
```



- More results

Gamma distribution

- The gamma distribution is concentrated in the low end of the spectrum with some fairly high values in the tail of the distribution and is nonnegative.
- It is often used to model loan default amounts or insurance claims.
- See `http://wiki.stat.ucla.edu/socr/index.php/AP_Statistics_Curriculum_2007_Gamma` for more details

Lognormal distribution

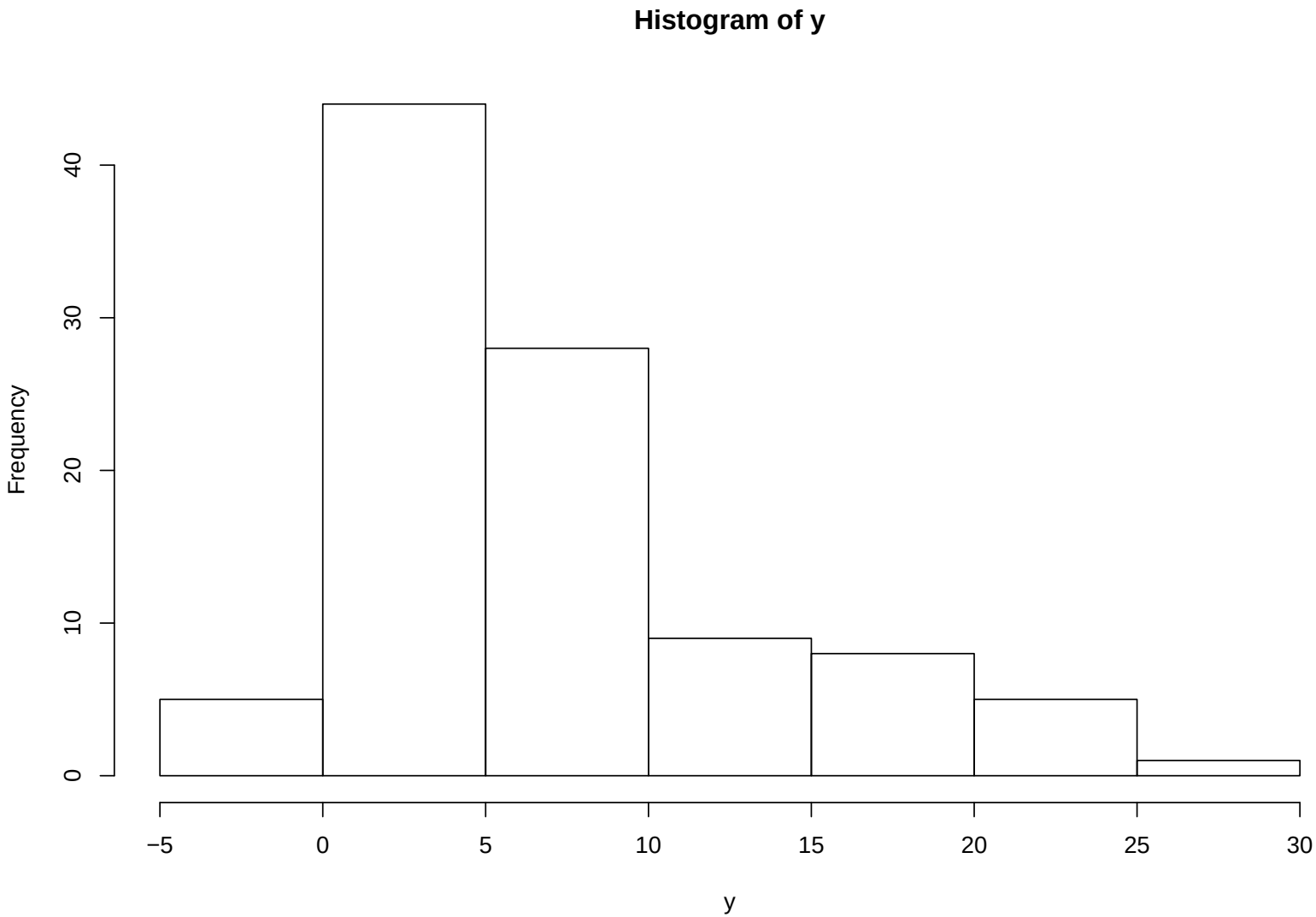
- The `lognormal` distribution is almost, but not quite,symmetrical around the mean value of 1 with more mass out into the tail, and is also non-negative.
- It is often used to model income of most individuals (the "1 percenters are usually modeled with the `pareto` distribution).
- See `https://en.wikipedia.org/wiki/Log-normal_distribution` for more information.

Yet more results

The result y with its `hist()`:

```
y <- 1.5 * x.1 + 0.8 * x.2 + rnorm(n.sim,
    1, 2)
```

```
hist(y)
```



- Still more results

Rubber meets the road here

```
X <- cbind(x.1, x.2)
XTX.inverse <- solve(t(X) %*% X)
(beta.hat <- XTX.inverse %*% t(X) %*%
    y)
```

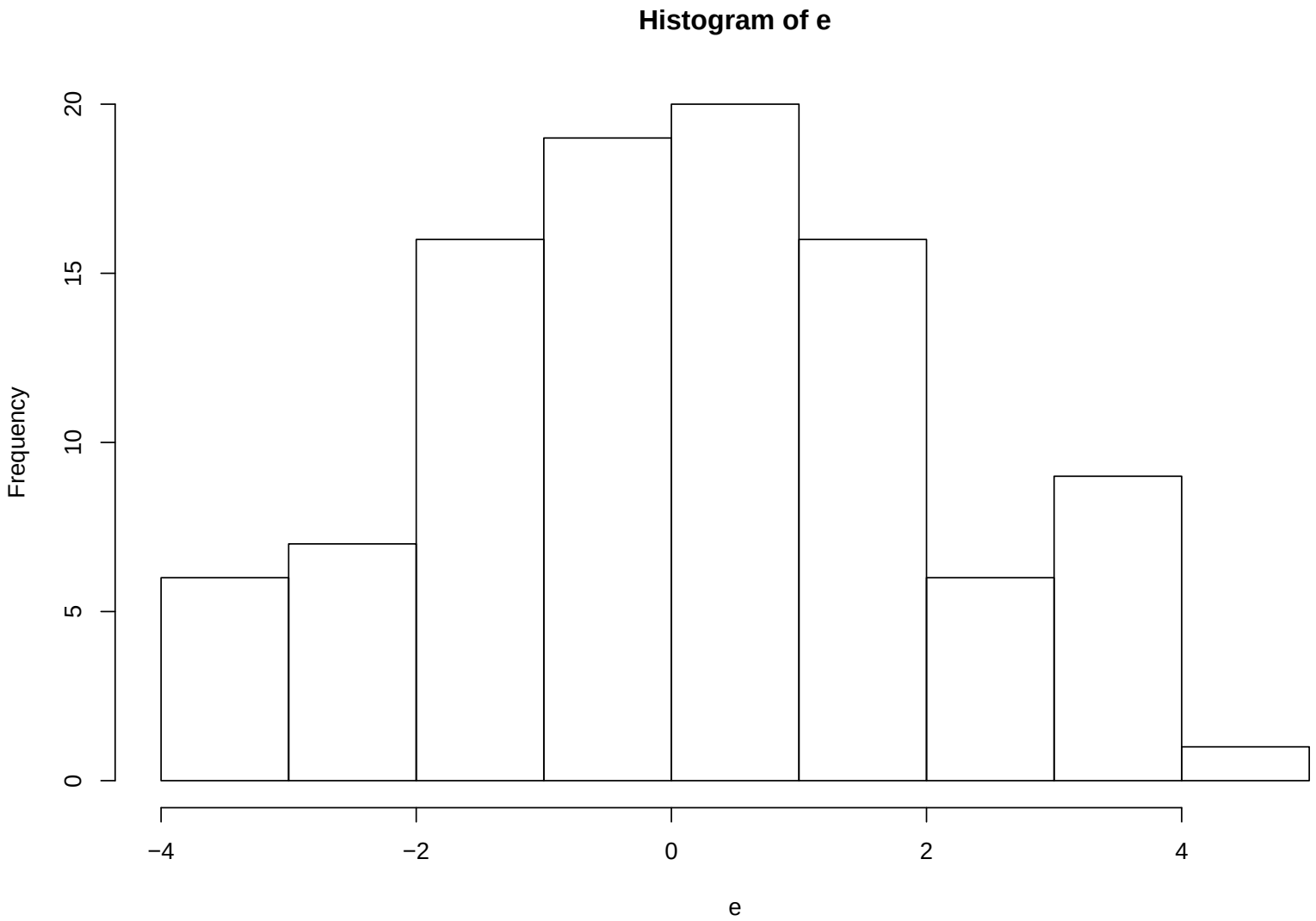
```
##          [,1]
## x.1 1.596584
## x.2 1.262823
```

The `beta.hat` coefficients are much different than our model for `y`. Why? Because of the innovation, error, disturbance term `rnorm(n.sim, 1, 2)` we added to the `1.5*x.1 + 0.8 * x.2` terms.

Now for the estimated ϵ where we use the matrix inner product %*%. Be sure to *pre*-multiply `beta.hat` with `X`!

```
e <- y - X %*% beta.hat
```

```
hist(e)
```



. . . almost centered at 0.

For no charge at all let's calculate the sum of squared errors in matrix talk, along with the number of obervations `n` and degrees of freedom `n - k`, all to get the standard error of the regression `e.se`:

```r
(e.sse <- t(e) %*% e)
```

```
##           [,1]
## [1,] 351.7062
```

```r
(n <- dim(X)[1])
```

```
## [1] 100
```

```r
(k <- nrow(beta.hat))
```

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

```r
(e.se <- (e.sse/(n - k))^0.5)
```

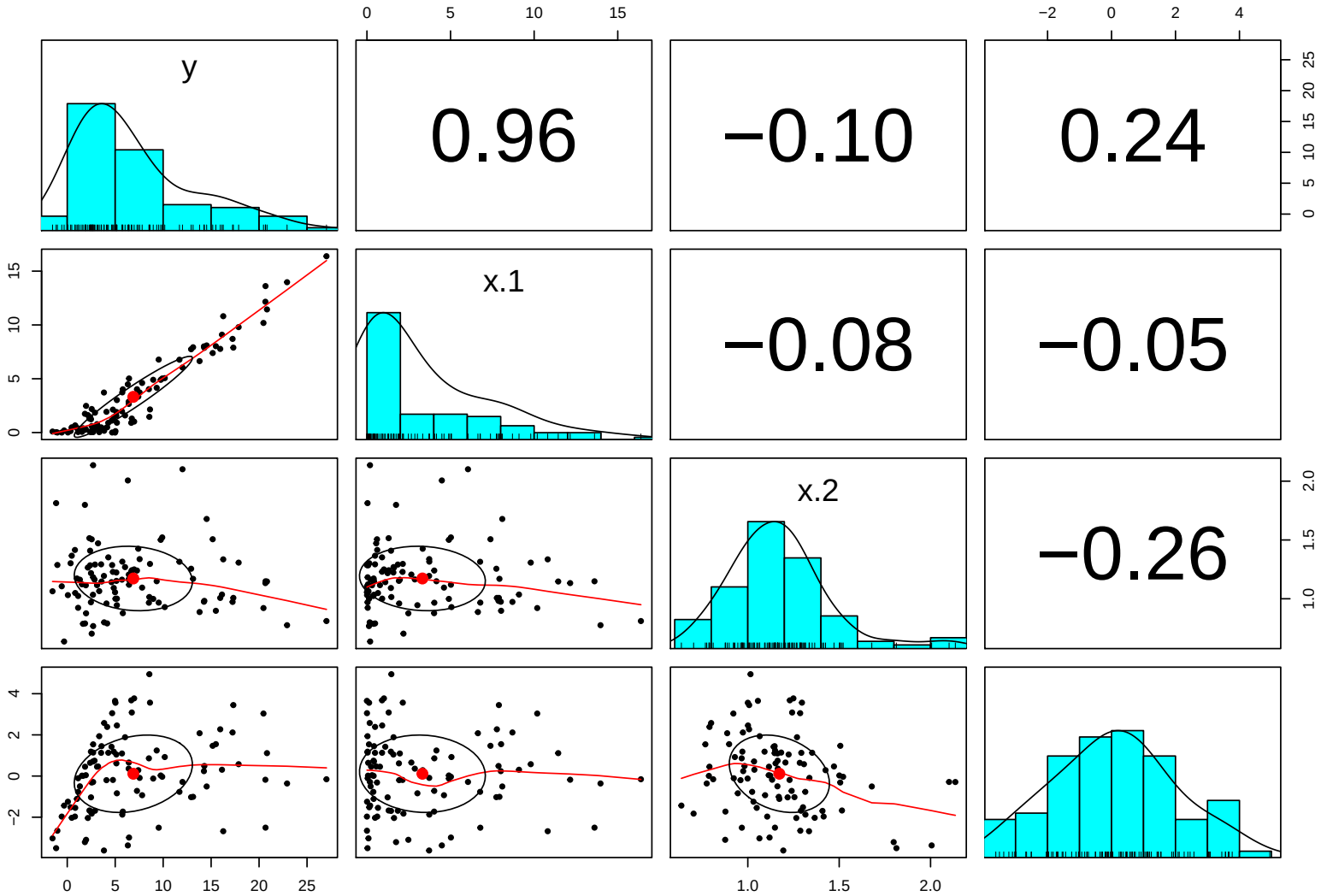
```
##           [,1]
## [1,] 1.894423
```

The statement `dim(X)[1]` returns the first of two dimensions of the matrix `X`.

Finally, again for no charge at all, lets load library `psych` (use `install.packages("psych")` if you need to...) and use `pairs.panels()` for a pretty picture of our work in this try out. First column bind `cbind()` the `y`, `X`, and `e` arrays to create a data frame for `pairs.panel()`.

```
library(psych)
all <- cbind(y, X, e)
```

```
pairs.panels(all)
```



We will use this tool again and again to explore the multivariate relationships among our data in our domain of choice: finance.

More array work here

We show off some more array operations in the following statements.

```
nrow(A.min)
```

```
## [1] 4
```

```
ncol(A.min)
```

```
## [1] 4
```

```
dim(A.min)
```

```
## [1] 4 4
```

```
rowSums(A.min)
```

```
## [1] 3.00 3.75 4.25 4.50
```

```
colSums(A.min)
```

```
## [1] 3.00 3.75 4.25 4.50
```

```
apply(A.min, 1, sum)
```

```
## [1] 3.00 3.75 4.25 4.50
```

```
apply(A.min, 2, sum)
```

```
## [1] 3.00 3.75 4.25 4.50
```

First we find the dimensions of `A.min`. These are the same as the number of rows and columns. We also calculate the sums of each row and each column.
Alternatively we can use the `apply` function on the first dimension (rows) and then on the second dimension (columns) of the matrix.

```
(A.inner <- A.sym %*% t(A.min[, 1:dim(A.min)[2]]))
```

```
##       [,1]   [,2]   [,3]  [,4]
## [1,] 0.750 0.7500 0.8125 0.875
## [2,] 0.375 0.5625 0.5000 0.500
## [3,] 0.375 0.5000 0.6875 0.625
## [4,] 0.750 0.9375 1.1250 1.375
```

Then, starting from the inner circle of embedded parentheses we pull every row (the `[,col]` piece) for columns from the first to the second dimension of the `dim()` of `A.min`. We then transpose (row for column) the elements of `A.min` and cross left multiply in an inner product this transposed matrix with `A.sym`.

```
(A.inner.invert <- solve(A.inner))
```

```
##                [,1]      [,2]      [,3]      [,4]
## [1,]  4.952381e+00 -3.047619 -1.142857 -1.523810
## [2,] -2.285714e+00  6.857143 -2.285714  0.000000
## [3,]  1.268826e-16 -2.285714  6.857143 -2.285714
## [4,] -1.142857e+00 -1.142857 -3.428571  3.428571
```

We then invert the square `A.inner` matrix with `solve`.

```
(A.result <- A.inner %*% A.inner.invert)
```

```
##               [,1]          [,2]          [,3]         [,4]
## [1,] 1.000000e+00  0.000000e+00  0.000000e+00 0.000000e+00
## [2,] 2.220446e-16  1.000000e+00  0.000000e+00 0.000000e+00
## [3,] 1.110223e-16  1.110223e-16  1.000000e+00 4.440892e-16
## [4,] 2.220446e-16 -4.440892e-16 -8.881784e-16 1.000000e+00
```

Then we cross multiply `A.inner` with its inverse. We should, and do, get the identity matrix that is a matrix of ones in the diagonal and zeros in the off-diagonal elements.

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE

What have we done this week?

- Lots of R
- Arithmetic
- Character manipulation
- Graphics
- Present value
- Probability distributions
- Matrix operations
- Regression

To prepare for the live session:

List these:

1. What are the top three key learnings for you?
2. What pieces are still a mystery?
3. What parts would you like more practice on?
4. Review the assignment. What questions do you have about the assignment for the Live Session?

Thanks!

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE