Introduction to Running R and Rmpi

L. Shawn Matott

Center for Computational Research
University at Buffalo, SUNY
701 Ellicott St
Buffalo, NY 14203

ccr-help at ccr.buffalo.edu

Spring 2015

CENTER FOR **COMPUTATIONAL RESEARCH**

University at Buffalo
The State University of New York

Outline

- Basics of R
- Graphics and Plotting in R
- R for HPC
- R Resources

The Basics of 

- What is R?
  - An open-source scripting environment
  - Geared toward statistics and data analysis
  - Base R + *many, many* extension packages
- R extensions
  - Package repositories make it easy to locate and install add-ons
  - Most popular repos: CRAN and Bioconductor

The Basics of 

- R is case sensitive (R != r)
- Command line prompt is >
- To run R code: use command line, or save script (.R) and
  > source("script_name")
  or (from Linux prompt or bash script):
  Rscript script_name.R
- To separate commands, use ; or a newline

The Basics of 

- Use the # character for *comments*
- To display help:

  ?<command-name>

  or

  ??<command-name>

R Output

```
> 2 + 3 * 5
[1] 17
```

Q: What's that [1] about?

A: R numbers outputs with [*n*]

```
# Try this in the command line:
> 1:500
```

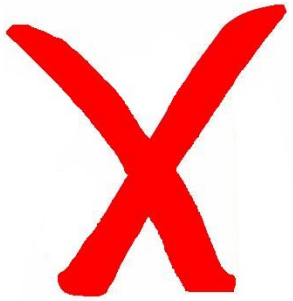
Naming Variables in R

A variable name may consist of letters, numbers and the dot or underline characters. It should start with a letter.

Good:

```
> y = 2
> try.this = 33.3
> oneMoreTime = “woohoo”
```

Bad:



```
> 2y = 2
> _z = 33.3
> function = “woohoo”
```

* *function* is a reserved word

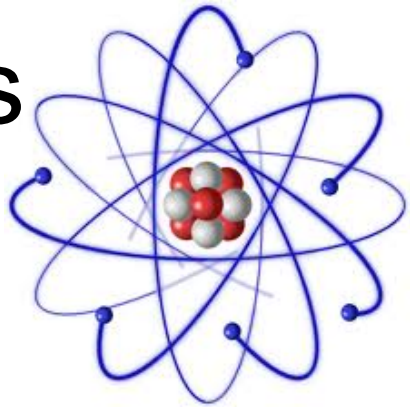
Variables Assignment

- You may use '=' or '<-' to assign values to a variable.

```
> y =  2 + 3 * 5
> z <- 2 + 3 * 5
```

(old fashioned way)

R's Atomic Data Types



Let's take a look at some available data types:

- Numeric (includes integer)
  3.14, 1, 2600

- Character (string)
  “hey, I'm a string”

- Logical
  TRUE or FALSE

- NA
  No value known

Using Logical Operators


```
1==2                    # equivalence test
9 != 19                 # “not equal” test
3 < 204                 # less-than test
18 > 44                 # greater-than test
“tree”==89      # comparing mixed data types
```

Objects in R

R stores everything, variables included, in Objects.

```
> x = 2.71
> print(x) # print the value of the object
[1] 2.71
> class(x)  # what data type or object type?
[1] "numeric"
> is.na(x) # tests whether x has a known value
[1] FALSE
```

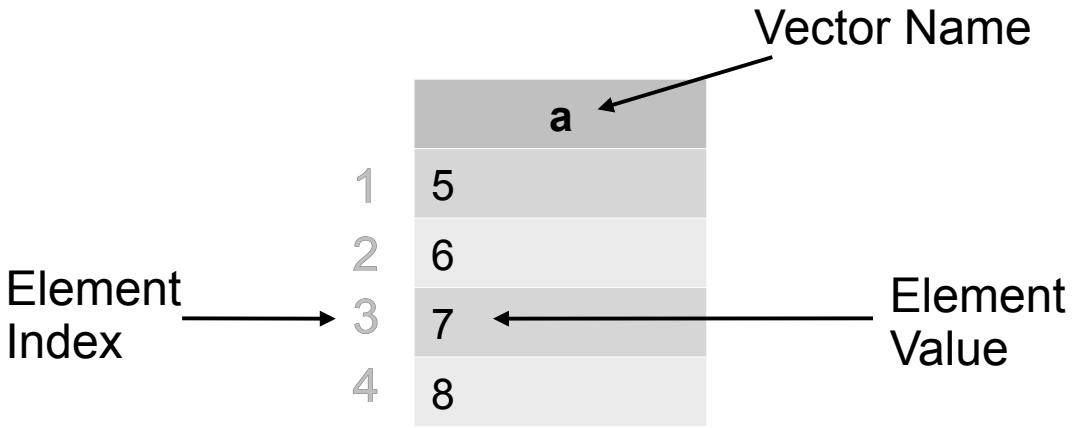
Objects in R

Vectors and Data Frames are the bread and butter of R

- Vector - a list of elements with the same type
- Data Frame - a structure consisting of columns of various types

Vector Data Object

A vector is a list of elements having the *same type*.



Vector Construction

	a
1	5
2	6
3	7
4	8

Use the c() function:

```
> a = c(5,6,7,8) # vector with 4 numeric values
> d = c("red", "orange", "green")  # char vector
```

Accessing Vector Data

	d
1	“red”
2	“orange”
3	“green”

	a
1	5
2	6
3	7
4	8

Access by index or range:

```
> d[1]      # retrieves “red”
> a[3]      # retrieves 7
> d[1:2]  # retrieves “red”, “orange”
```

Element numbering starts at 1 in R

Some Vector Operations

- sum() # Sum of all element values
- length() # Number of elements
- unique() # Generate vector of distinct values
- diff() # Generate vector of first differences
- sort() # Sort elements, omitting NAs
- order() # Sort indices, with NAs last
- rev() # Reverse the element order
- summary() # Information about object contents

Handling Missing Data

Remove NAs prior to calculation:

```
> y = c(1, NA, 3, 2, NA)  # [1, ?, 3, 2, ?]
sum(y, na.rm=TRUE) # removes NAs, then sums
[1] 6  # sum of 1 + 3 + 2
```

Data Frames

- Data frames are handy containers for experimental data.
- Columns - data fields (vectors)
- Rows - tuples of data
- Rows and columns can be named

A Simple Data Frame

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)
```

source: http://www.r-tutor.com/r-introduction/data-frame

Data Frame Information

```
str(data)   # structure (NOT string!)
dim(data)  # dimensions

View(data)        # open View window of data
head(data)        # print beginning of the data frame
tail(data)        # print end of the data frame

names(data)           # names of the columns
rownames(data)        # names of the rows
colnames(data)        # names of the columns
```

User-Defined Functions in R

Basic structure:

```
my_function_name = function(arg1, arg2, ... ){
   statements
   return(object)
}
```

A concrete example:

```
toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32; # do something
  return(f); # return the result
}
```

Invoking Functions in R

```
temps = c(20:25); # define input temperatures
# define a function to convert temperatures
toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32;  # perform the conversion
  return(f);
}
# invoke the function:
toFahrenheit(temps);
[1] 68.0  69.8  71.6  73.4  75.2  77.0
```

Loading Functions in R

```
# Functions can be loaded from a separate R file.
# Analogous to an "include" statement in C/C++/FORTRAN
# or an "import" statement in python
source("myFunctions.R")
```

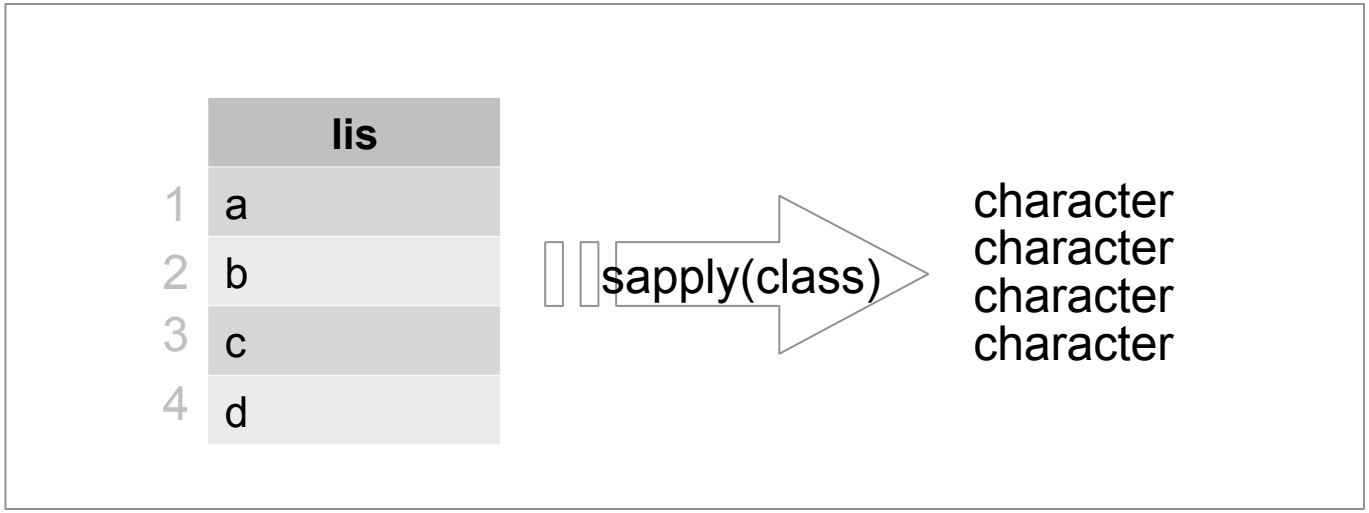
Control Structures: apply() Family

- What if we want to call a function over and over?
- We can do this with a single line of R code!
- Use it on native R functions, or functions you wrote yourself.

sapply(*vector*, *function*)

Control Structures: sapply()

```
> lis = c("a", "b", "c", "d")
> sapply(lis, class)
        a         b         c         d
"character" "character" "character" "character"
```



Statistical Functions in R

- mean()
- median()
- range()
- var() and sd() # variance, standard deviation
- summary() # combination of measures
- *plus many, many others!*

Outline

Basics of R

Graphics and Plotting in R

R for HPC

R Resources

Some Plot Types

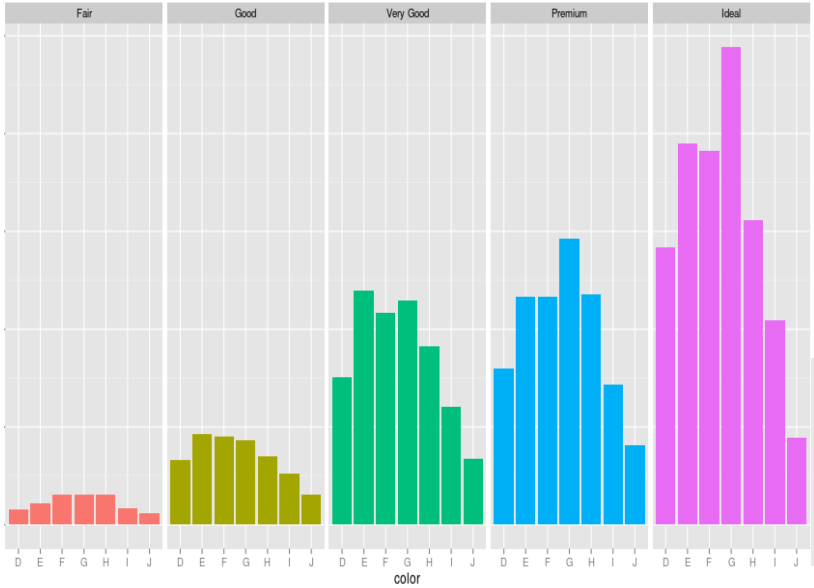
- Pie Chart
  - Display proportions of different values for a variable
- Bar Plot
  - Display counts of values for a categorical variable
- Histogram, Density Plot
  - Display counts of values for a binned, numeric variable
- Scatter Plot
  - y vs. x
- Box Plot
  - Display distributions over different values of a variable

Plotting Packages

- 3 Main Plotting Packages
  - Base graphics, lattice, and ggplot2
- Base graphics
  - Legacy approach (FORTRAN and S)
  - http://www.statmethods.net/graphs/creating.html
- lattice
  - Extends base graphs with support for trellis graphs.
  - http://www.statmethods.net/advgraphs/trellis.html
- ggplot2
  - The "cadillac" of plotting packages. Beautiful plots but complex.
  - http://ggplot2.org/

Plotting Packages

- The best approach: learn by doing but *don't start from scratch*.
- Start with an example that is similar in appearance to what you are trying to achieve.
- Modify this example to use your own data.



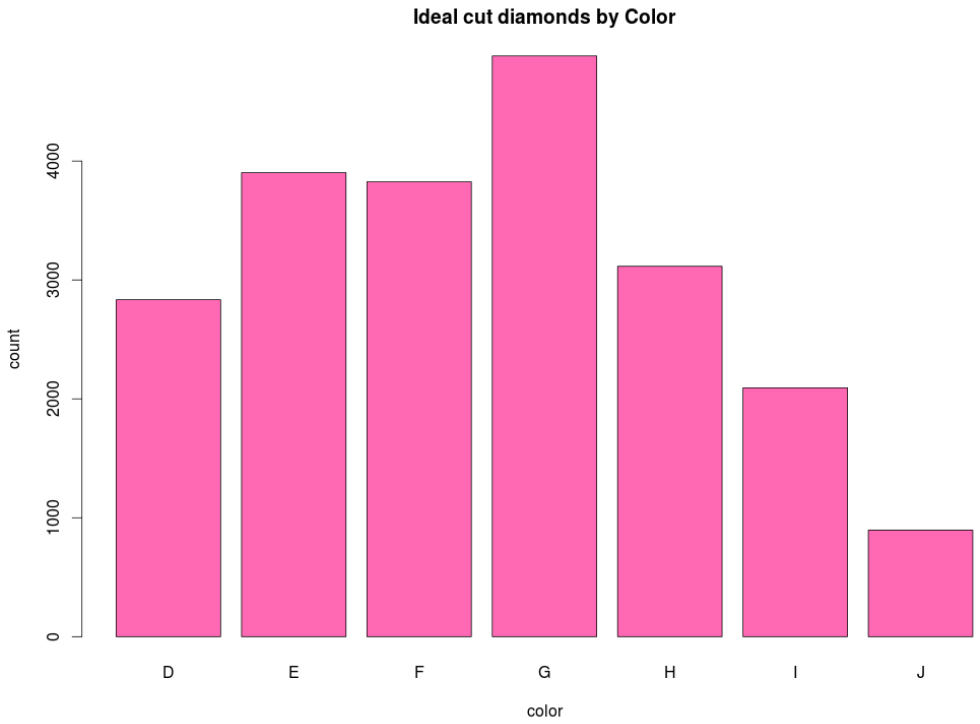
Dataset:
http://docs.ggplot2.org/0.9.3.1/diamonds.html

Tutorial:
http://ggplot2.org/book/qplot.pdf

Code Snippet:

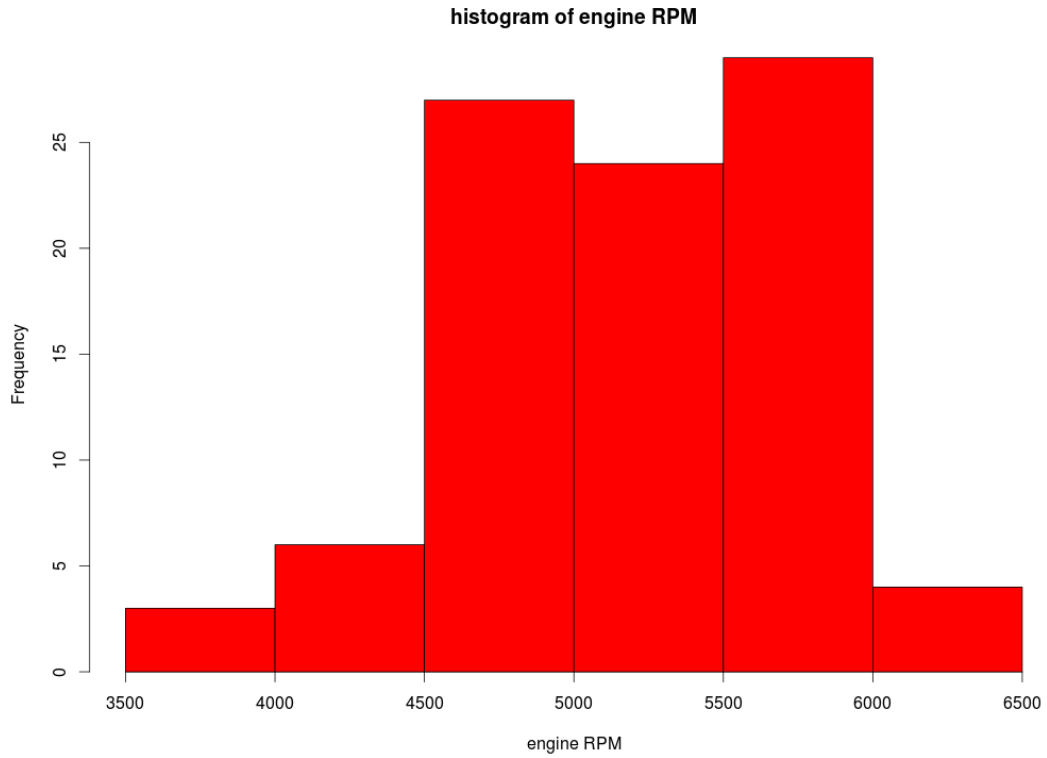
```
ggplot(small)
        +geom_point(aes(x=carat,y=price,colour=cut))
                +scale_y_log10()
                        +facet_wrap(~cut)
                                +ggtitle("First example")
```

Bar Plot: Counts of Categorical Values



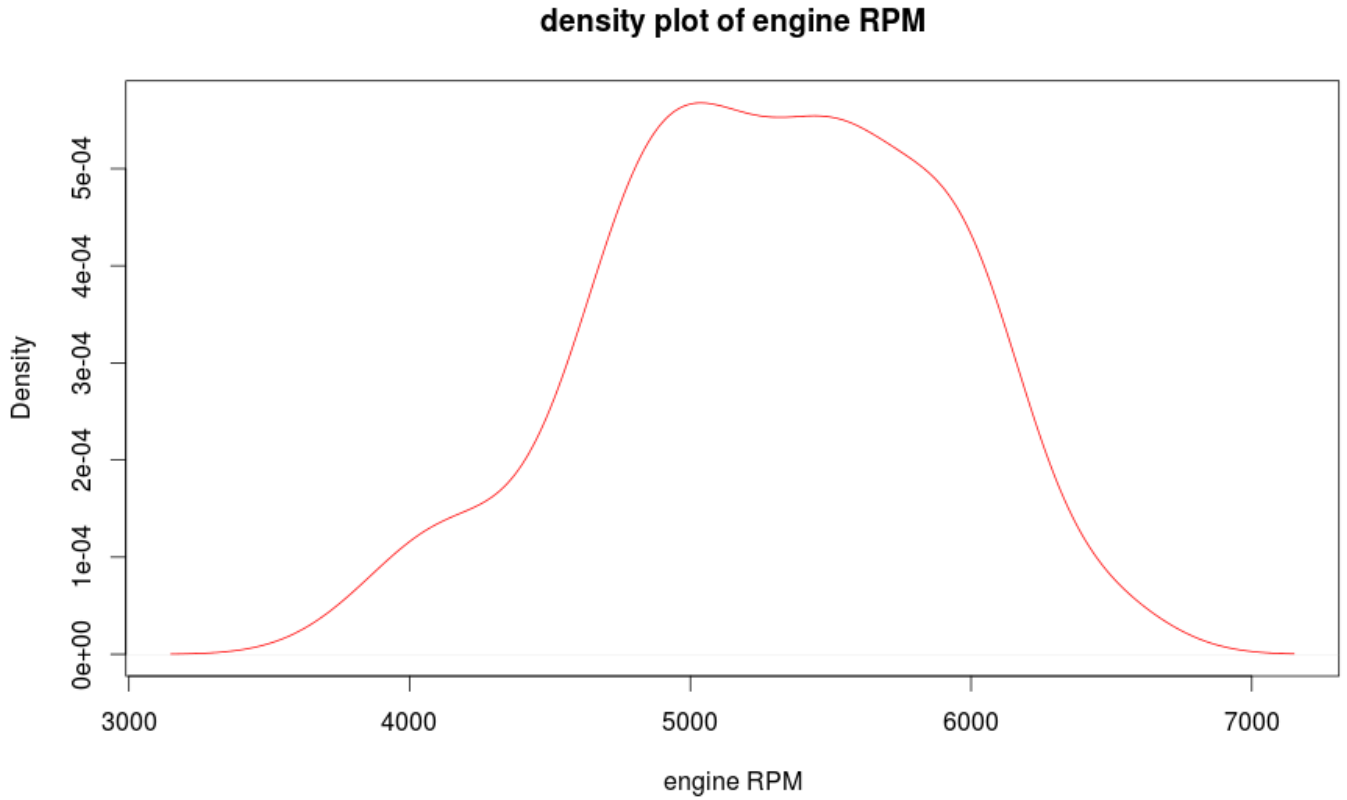

```
ideal=diamonds[diamonds$cut=="Ideal","color"]

barplot(table(ideal), xlab="color", ylab="count",
        main="Ideal cut diamonds by Color",
        col="hotpink")
```

Histogram: Frequencies of Numeric Values



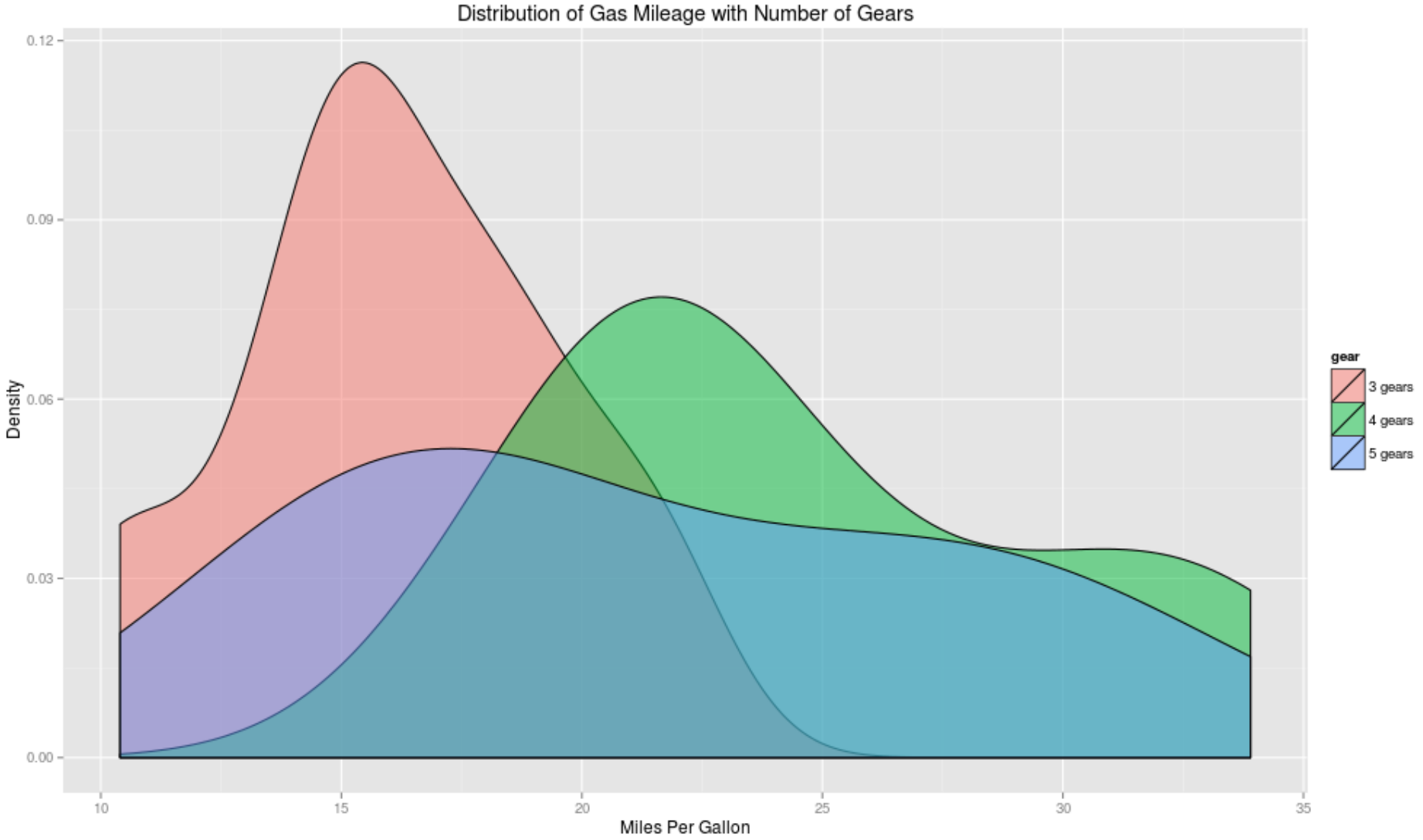

```
hist(Cars93$RPM,
     xlab="engine RPM",
     main="histogram of engine RPM",
     col="red")
```

Kernel Density Plot

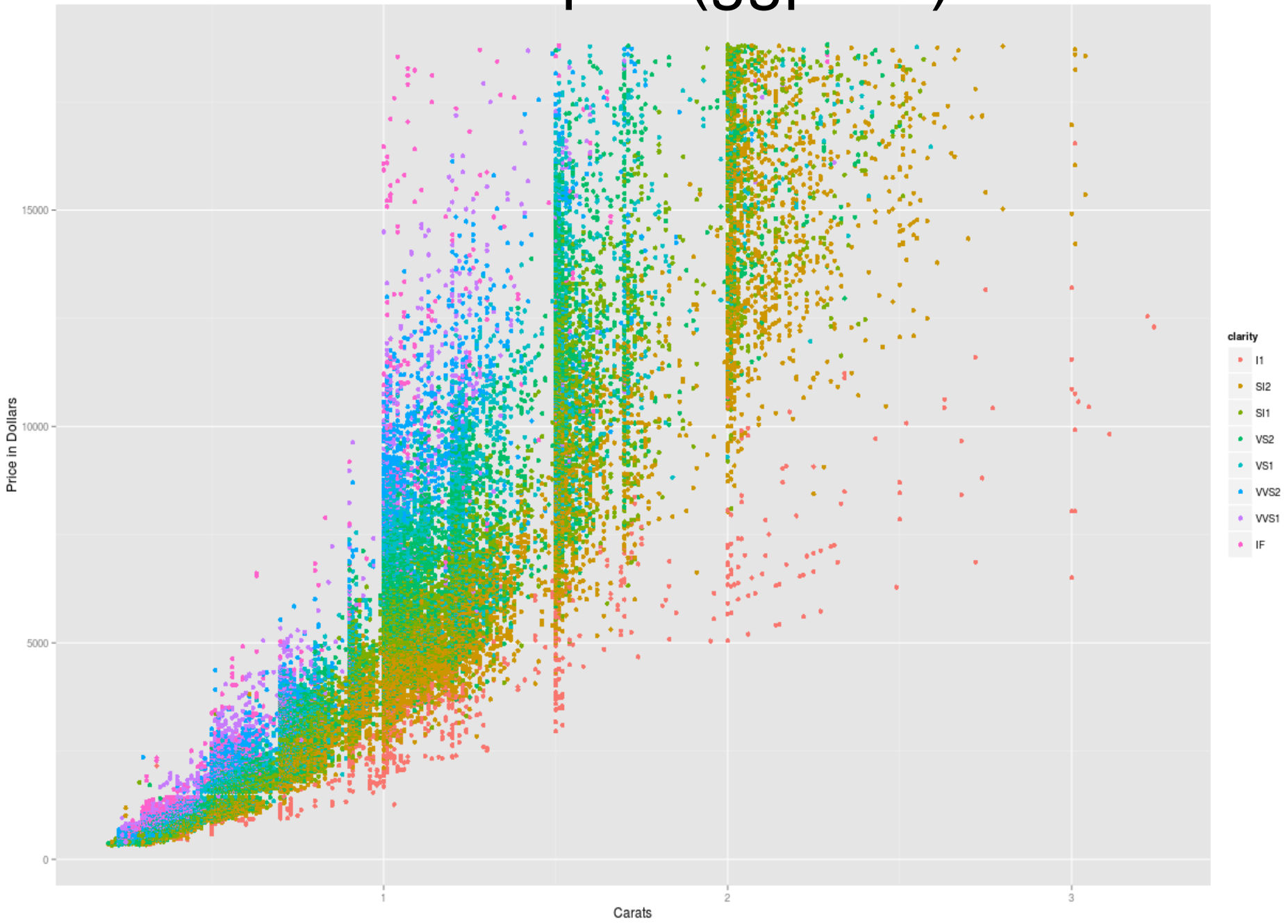



```
plot(density(Cars93$RPM),
     xlab="engine RPM",
     main="density plot of engine RPM",
     col="red")
```

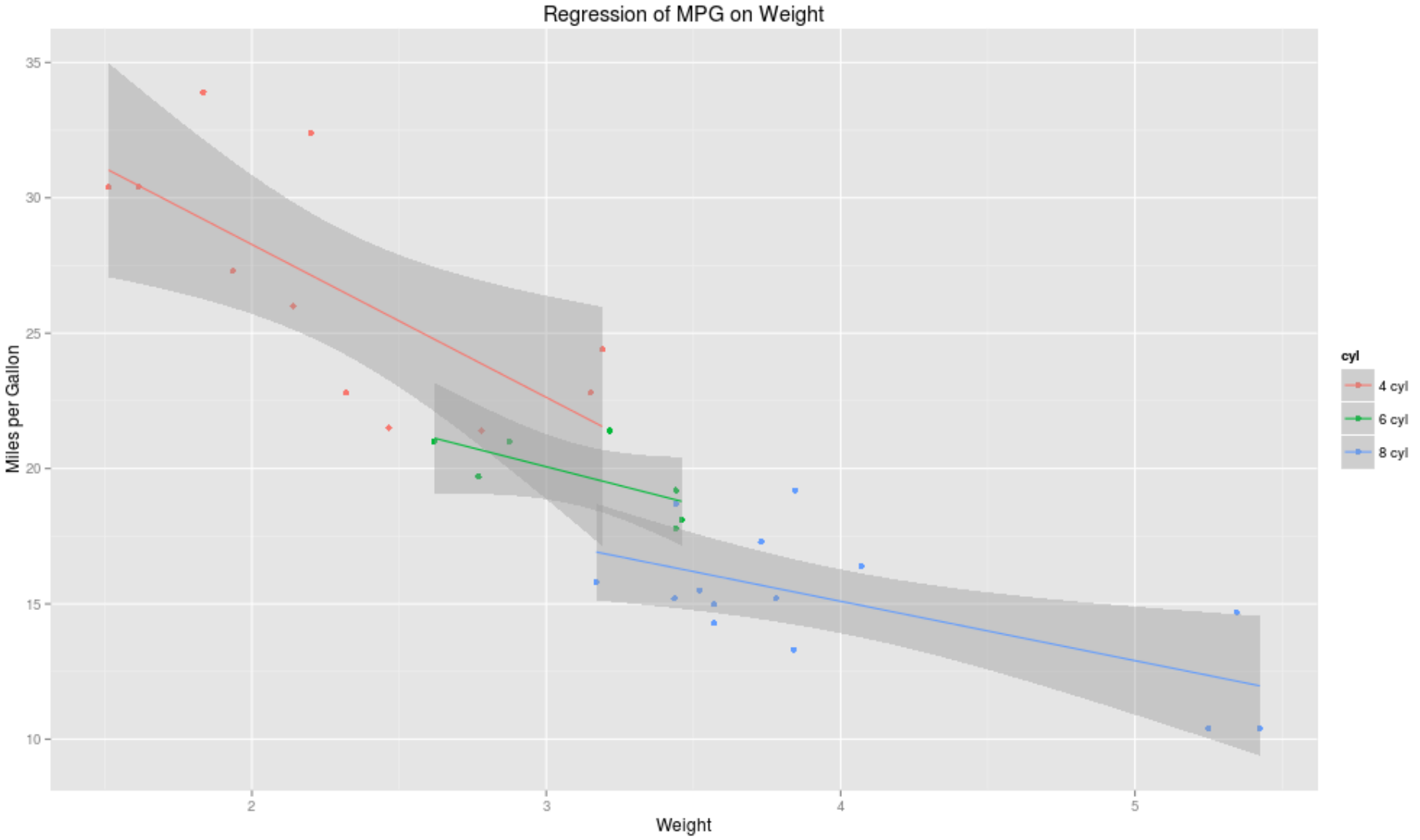
Density Plot (ggplot2)



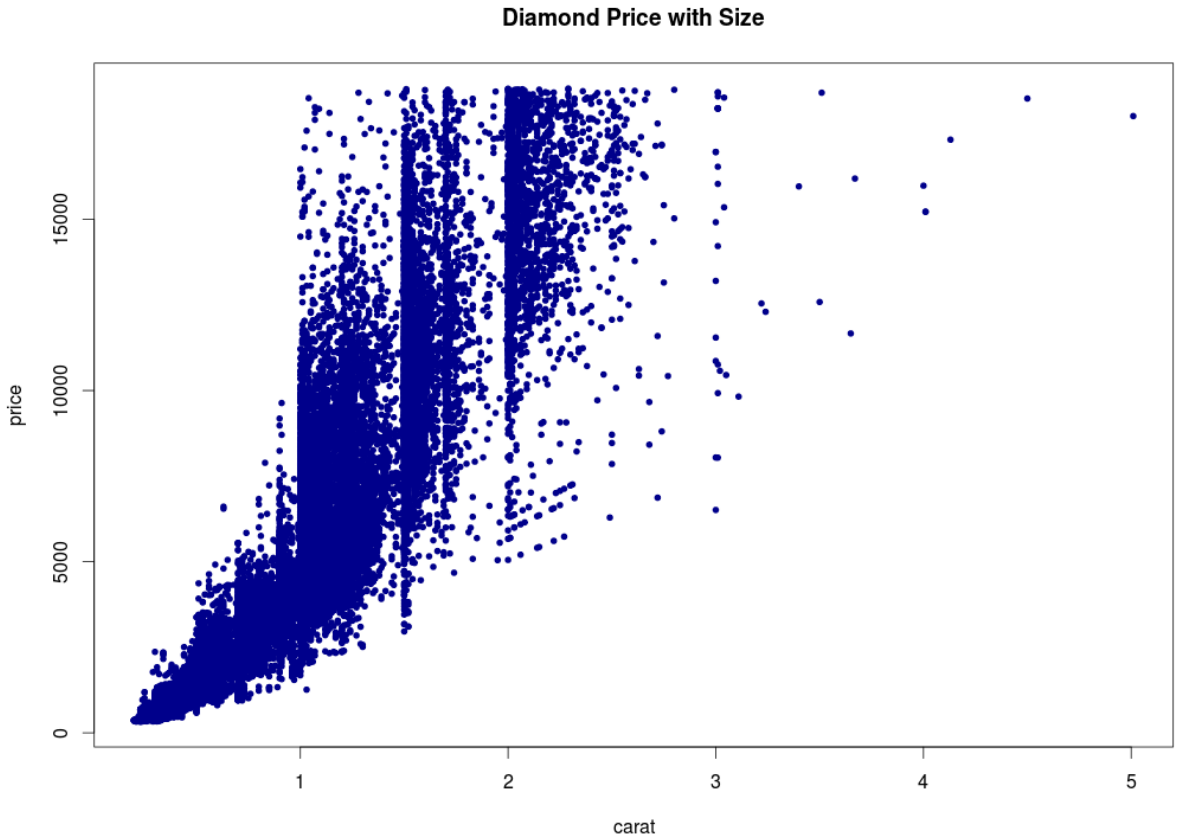
Scatterplot (ggplot2)



Scatterplot w/ Regression Lines (ggplot2)

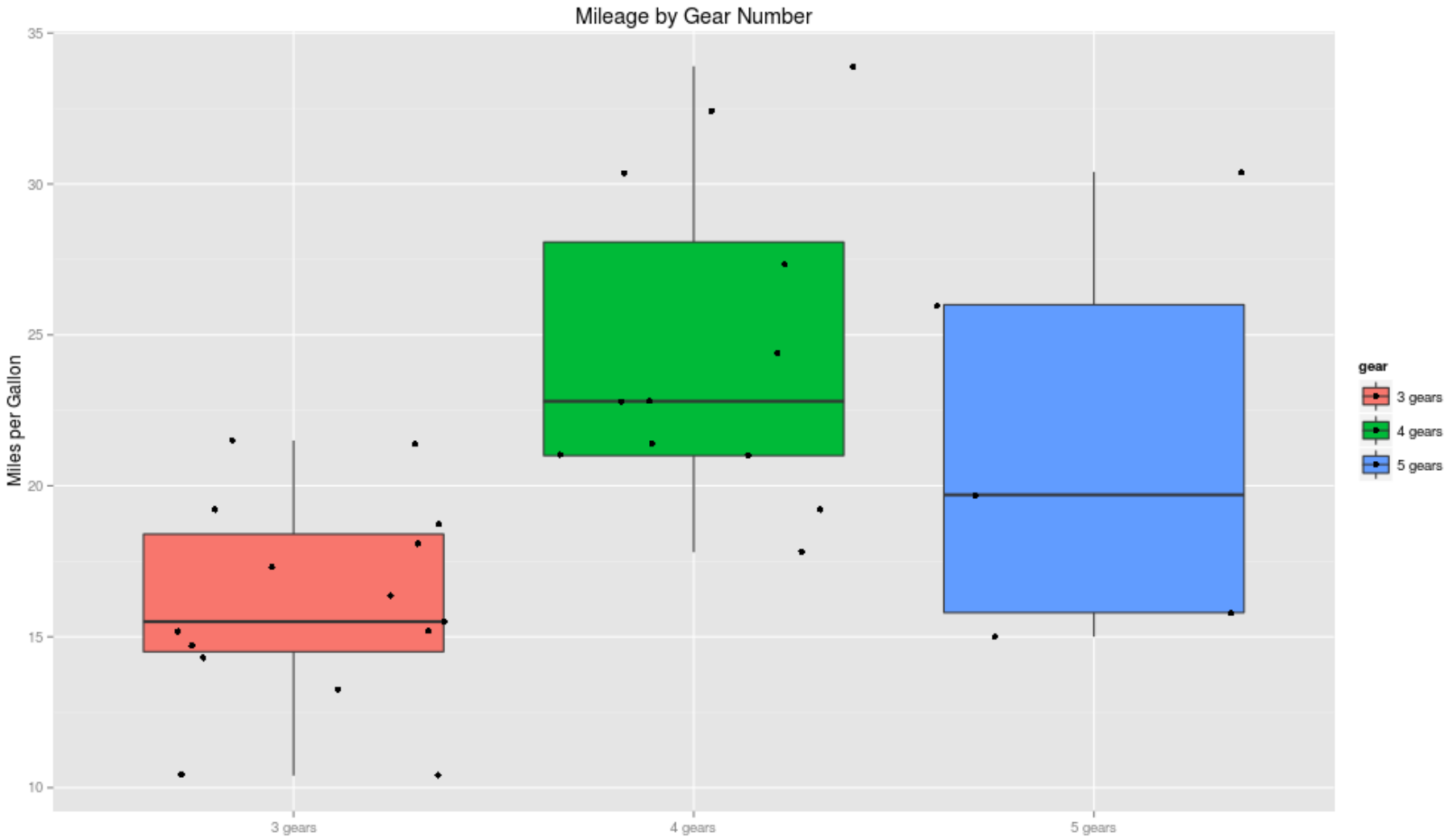


Scatterplot (base graphics)

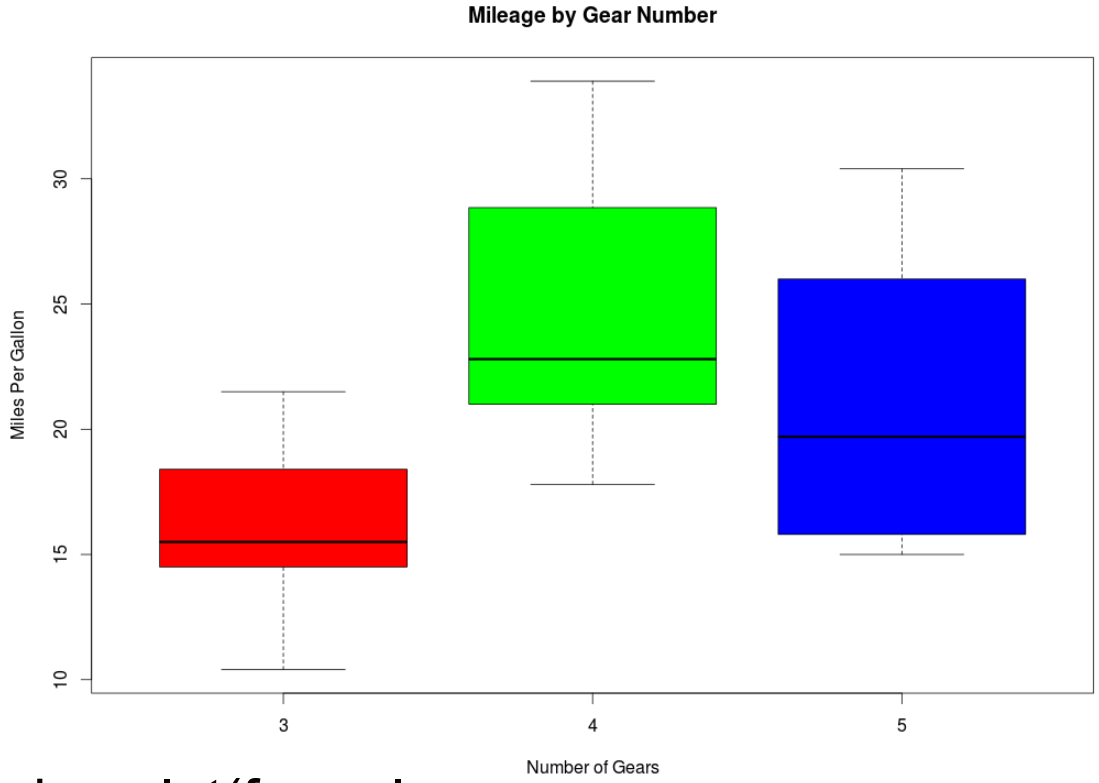



```
plot(formula=price~carat,
     data=diamonds,
     col="darkblue",
     pch=20,
     main="Diamond Price with Size")
```

Box (and Whisker) Plot (ggplot2)



Box (and Whisker) Plot (base graphics)




```
boxplot(formula=mpg~gear,
                data=mtcars,
                main="Mileage by Gear Number",
                xlab="Number of Gears",
                ylab="Miles Per Gallon",
                col=c("red","green","blue"))
```

Outline

Basics of R

Graphics and Plotting in R

R for HPC

R Resources

R for HPC

- Many options for parallelizing R code
- Some of the more commonly used:
  - Rmpi - an MPI implementation
  - snow/snowfall - Simple Network of Workstations
  - foreach – like parfor in MatLab
  - doMC – do MultiCore
  - papply - parallel version of sapply (deprecated)
- Full list of options:

http://cran.r-project.org/web/views/HighPerformanceComputing.html

Rmpi Examples

- Available on CCR in the /util storage area
- /util/academic/R/Rmpi-simple
  - “Hello World”
- /util/academic/R/Rmpi-example
  - Data analysis
- /util/academic/R/Rmpi-julia
  - Fractal generation

Rmpi Examples

- All CCR examples contain:
  - R script using Rmpi for parallelization
  - SLURM script for launching parallel job using srun
  - .Rprofile for configuring Rmpi
- .Rprofile
  - You must have a copy of this in your CCR work dir
  - Loads and initializes Rmpi for use on the cluster
- SLURM script
  - Loads R and intel-mpi modules
  - Simply copy and edit to invoke your Rmpi code

Rmpi – Julia Example

- Simple algorithm for creating fractals

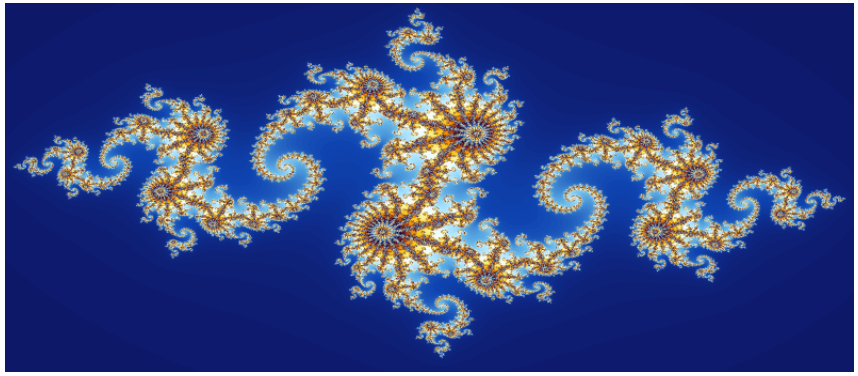
Pick a point (Z_p) in the complex plane:

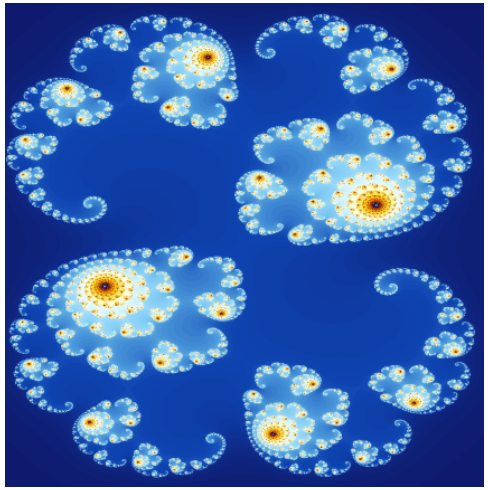
$$Z_0 = Z_p$$

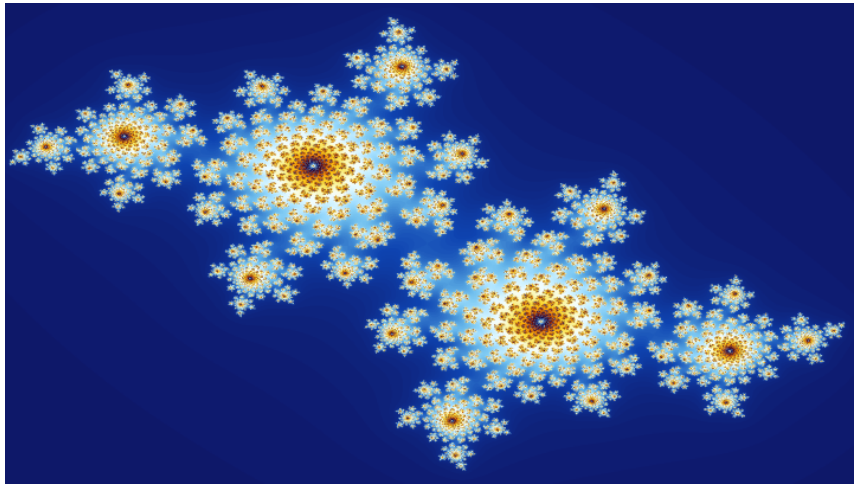
Iterate according to:

$$Z_{n+1} = Z_n^{\,2} + C$$

If Z_{n+1} diverges, discard point
Else, plot point







A serial implementation

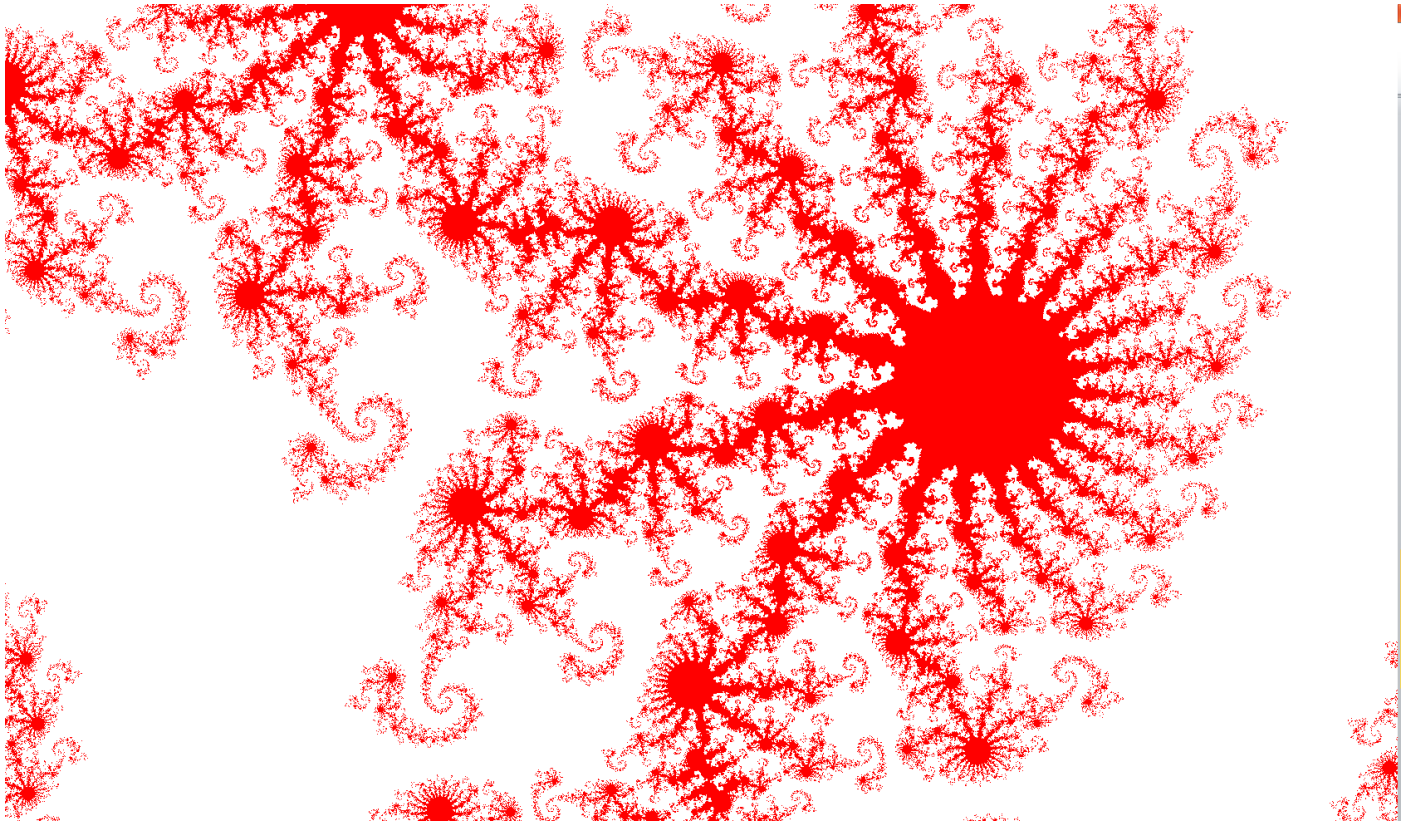
```
# grab start time
start_time = proc.time();

# create a gray bitmap, flattened to a vector
Z = rep(gryPxl, DIM*DIM);

# compute Julia set coordinates
Range=seq(from=JMIN,to=JMAX,length=DIM)
XY=expand.grid(Range,Range);
X=XY[[1]];
Y=XY[[2]];

# compute julia set values
Z = sapply(1:(DIM*DIM), function(i, x, y, cR, cI) julia(x[i], y[i], cR, cI), x=X, y=Y, cR=cR, cI=cI);

# grab end time
end_time = proc.time();

# elapsed time
elapsed_time = end_time - start_time;
paste("Timing info:");
print(elapsed_time);

# save bitmap to file
BMP = matrix(Z, nrow = DIM, ncol = DIM);
paste("rendering bitmap");
grid.raster(BMP);
```

Main program loop – Julia function does actual work!

A serial implementation

- **`julia_serial.R`**

  Run time is ~140 seconds



Rplots.pdf

Rmpi Parallel Implementation

```
# Now, send the data to the children
mpi.bcast.Robj2slave(DIM);
mpi.bcast.Robj2slave(X);
mpi.bcast.Robj2slave(Y);
mpi.bcast.Robj2slave(cR);
mpi.bcast.Robj2slave(cI);

# Send the functions to the children
mpi.bcast.Robj2slave(cxMag2);
mpi.bcast.Robj2slave(cxMult);
mpi.bcast.Robj2slave(julia);
mpi.bcast.Robj2slave(juliaRmpi);

# Call the function in all the children, and collect the results
Zlist = mpi.remote.exec(juliaRmpi());

# combine julia set results
paste("Combining results");
for(i in 1:length(Zlist)) {
  ZZ=Zlist[[i]];
  for(j in 1:(DIM*DIM)) {
    if( ZZ[j] == redPxl ) {
      Z[j] = redPxl;
    }
  }
}
```

Rmpi Parallel Implementation

- `julia_rmpi.R` ← Rmpi version
  - Run Time is ~30 seconds on 16 processors
- Speedup = (serial time) / (parallel time) = 4.67
- Efficiency = (Speedup / nprocs) = 0.29
- Typically want efficiency >= 0.7

Why such poor efficiency?

```
# deermine whether point lies on julia set
julia = function(x, y, cR, cI){
  #*************************************************************************
  # Evaluate julia set for a given point (x,y) and given constant (cR, cI),
  #   0 -- diverges, don't plot
  #   1 -- converges, plot the point
  #*************************************************************************

  redPxl = "#FF0000"; # red pixel
  grnPxl = "#00FF00"; # green
  bluPxl = "#0000FF"; # blue
  gryPxl = "#999999"; # grey pixel

  #simple julia set algorithm
  zR = x;
  zI = y;
  for(i in 1:200) {
    # Z = Z^2 + C
    Z=cxMult(zR, zI, zR, zI);
    zR=Z[[1]];
    zI=Z[[2]];
    zR = zR + cR;
    zI = zI + cI;
    if (cxMag2(zR, zI) > 1000) {
      return(gryPxl);
    }
  }
  return(redPxl);
}
```

←Some processors will finish early and wait (long-pole problem)

Outline

Basics of R

Graphics and Plotting in R

R for HPC

R Resources

...is free

If you want to experiment further with R and RStudio, you can install them on your favorite operating system at home.

First, install R:

http://cran.r-project.org/

Then, install the Rstudio IDE:

http://www.rstudio.com/ide/

Basic R References

Verzani's Simple R pdf book (R, statistics):
http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf

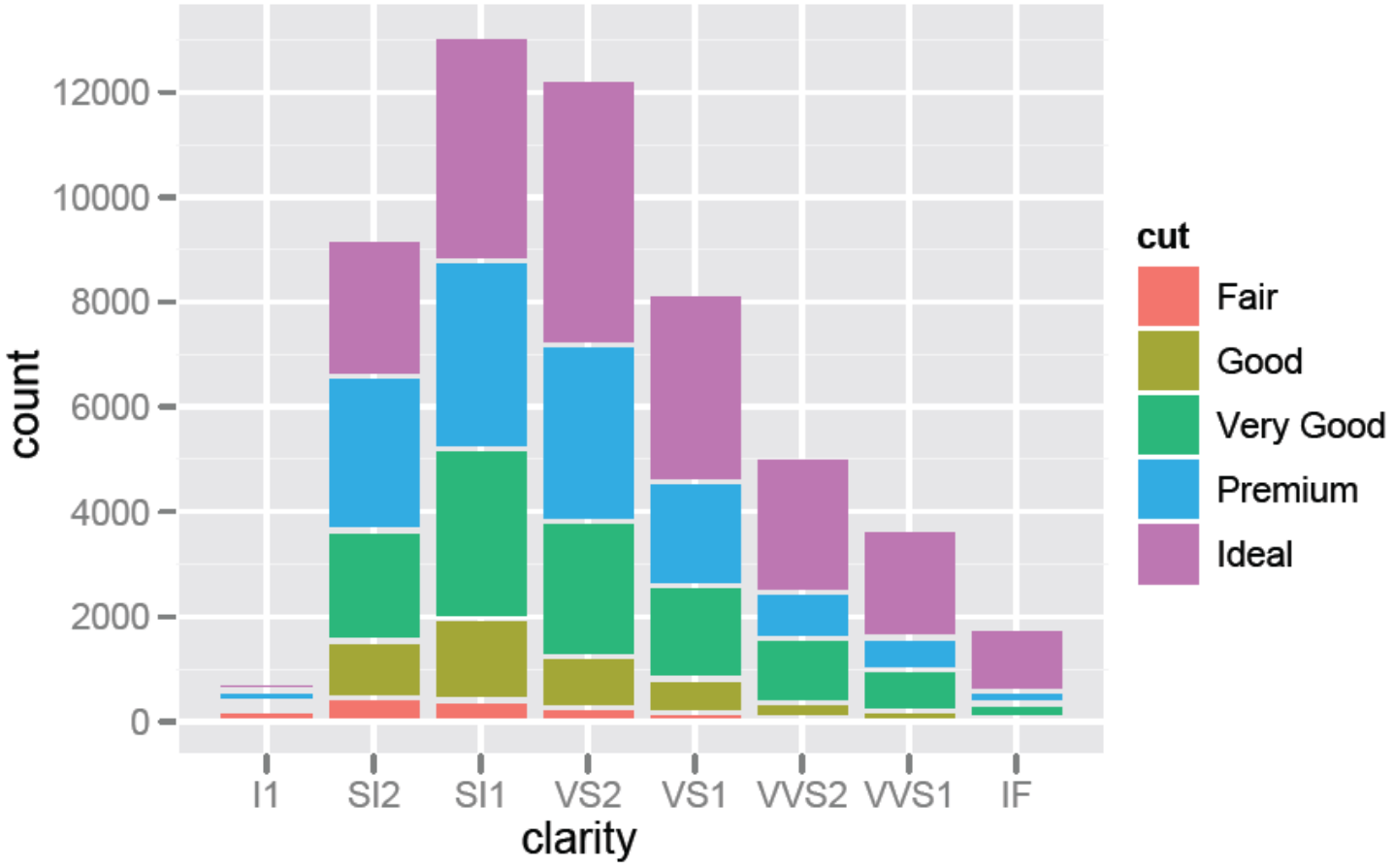
Maindonald's Using R pdf book (R, basic plotting):
http://cran.r-project.org/doc/contrib/usingR.pdf

R Bootcamp Tutorial:
http://jaredknowles.com/s/Tutorial1_Intro.html

A Basic R Vocabulary:
https://github.com/hadley/devtools/wiki/vocabulary

Fancy Graphics (ggplot2)

www.ceb-institute.org/bbs/wp-content/uploads/2011/09/handout_ggplot2.pdf



On The apply() Family

```
base::apply    Apply Functions Over Array Margins
base::by       Apply a Function to a Data Frame Split by Factors
base::eapply   Apply a Function Over Values in an Environment
base::lapply   Apply a Function over a List or Vector
base::mapply   Apply a Function to Multiple List or Vector Arguments
base::rapply   Recursively Apply a Function to a List
base::tapply   Apply a Function Over a Ragged Array
```

https://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/

http://www.r-bloggers.com/the-r-apply-function-a-tutorial-with-examples/

http://www.r-bloggers.com/using-apply-sapply-lapply-in-r/

On The apply() Family

See also the plyr package:

Maintainer Hadley Wickham <h.wickham@gmail.com>

Description plyr is a set of tools that solves a common set of problems: you need to break a big problem down into manageable pieces, operate on each pieces and then put all the pieces back together. For example, you might want to fit a model to each spatial location or time point in your study, summarise data by panels or collapse high-dimensional arrays to simpler summary statistics. The development of plyr has been generously supported by BD (Becton Dickinson).

URL http://had.co.nz/plyr