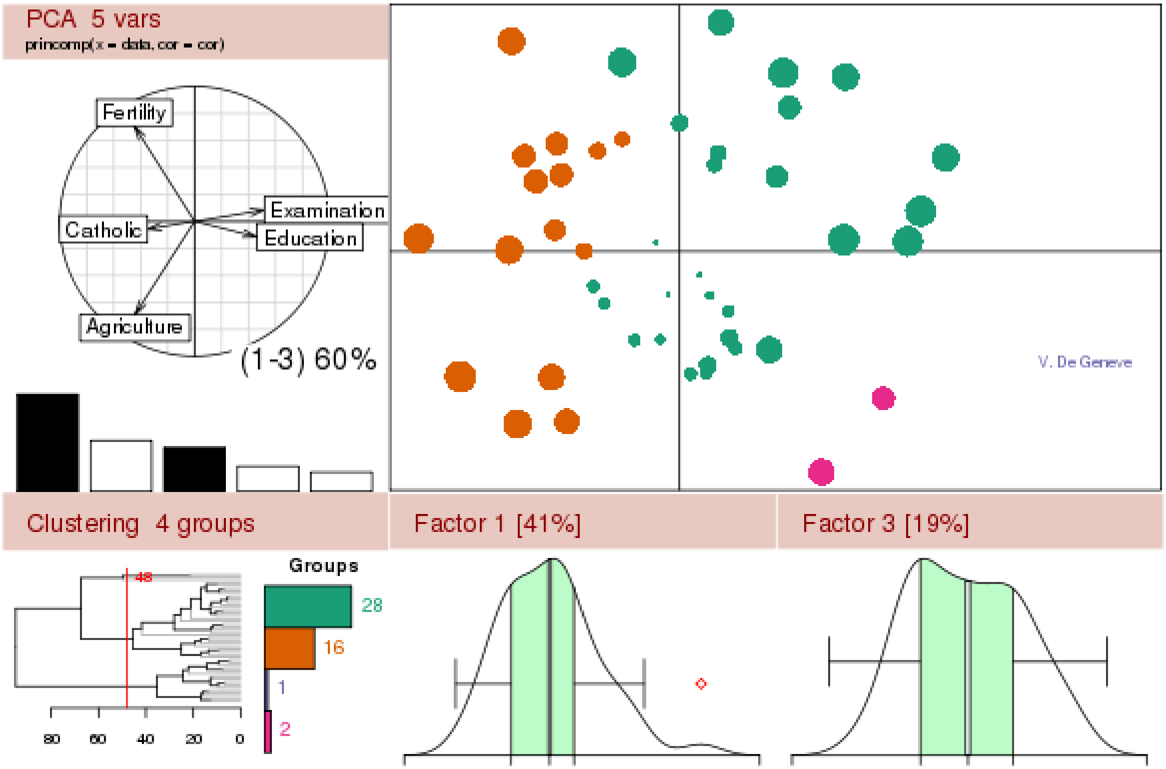
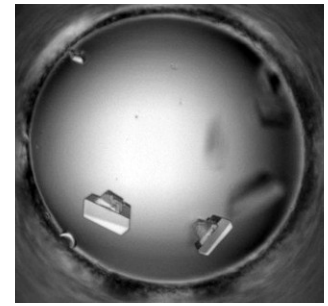
2013 Eric Pitman Summer Workshop in Computational Science



...an introduction to R, statistics, programming, and getting to know datasets

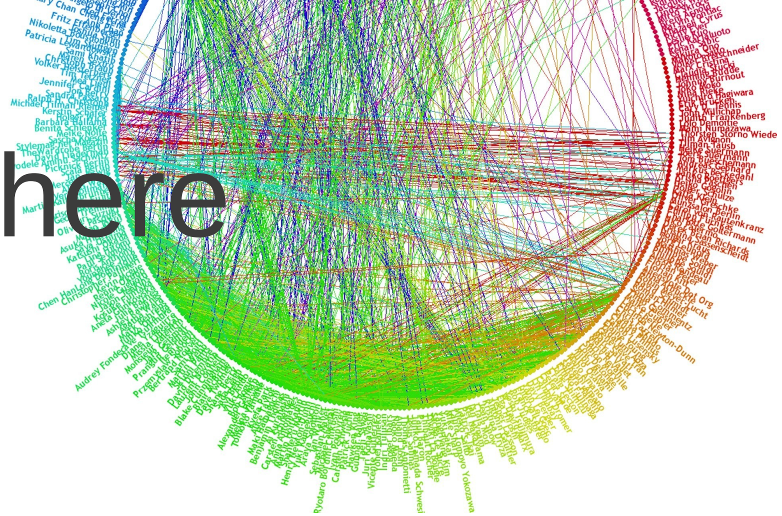
Jeanette Sperhac

Today's Plan



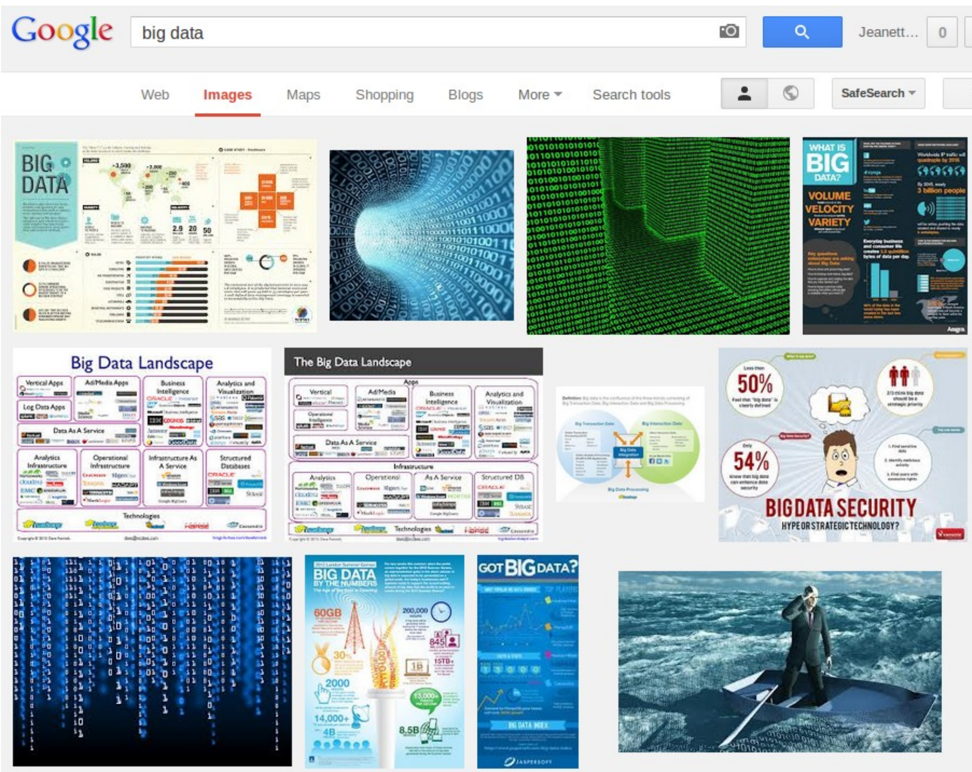
- Overview of the workshop
- Get on the command line!
- Variables lecture
- Variables exercises
- Lunch
- HWI tour

Data is Everywhere



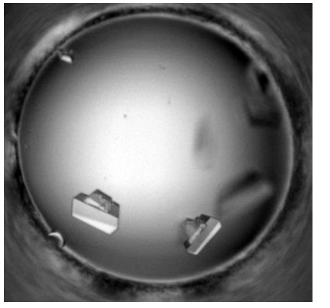
- For example:
  - Science/engineering/medicine
  - Environmental science/Social science/Law enforcement
  - Finance and marketing
  - Social media
- How do we come to an understanding of what a dataset contains?
- Can we draw conclusions from a dataset?
- Let's taste the complexities for ourselves

“Big Data” means three things



- *Volume*: lots of data
- *Velocity*: coming at you fast—Twitter, 7TB/day
- *Variety*: text, pictures, video, etc.

Our Plan for the Workshop

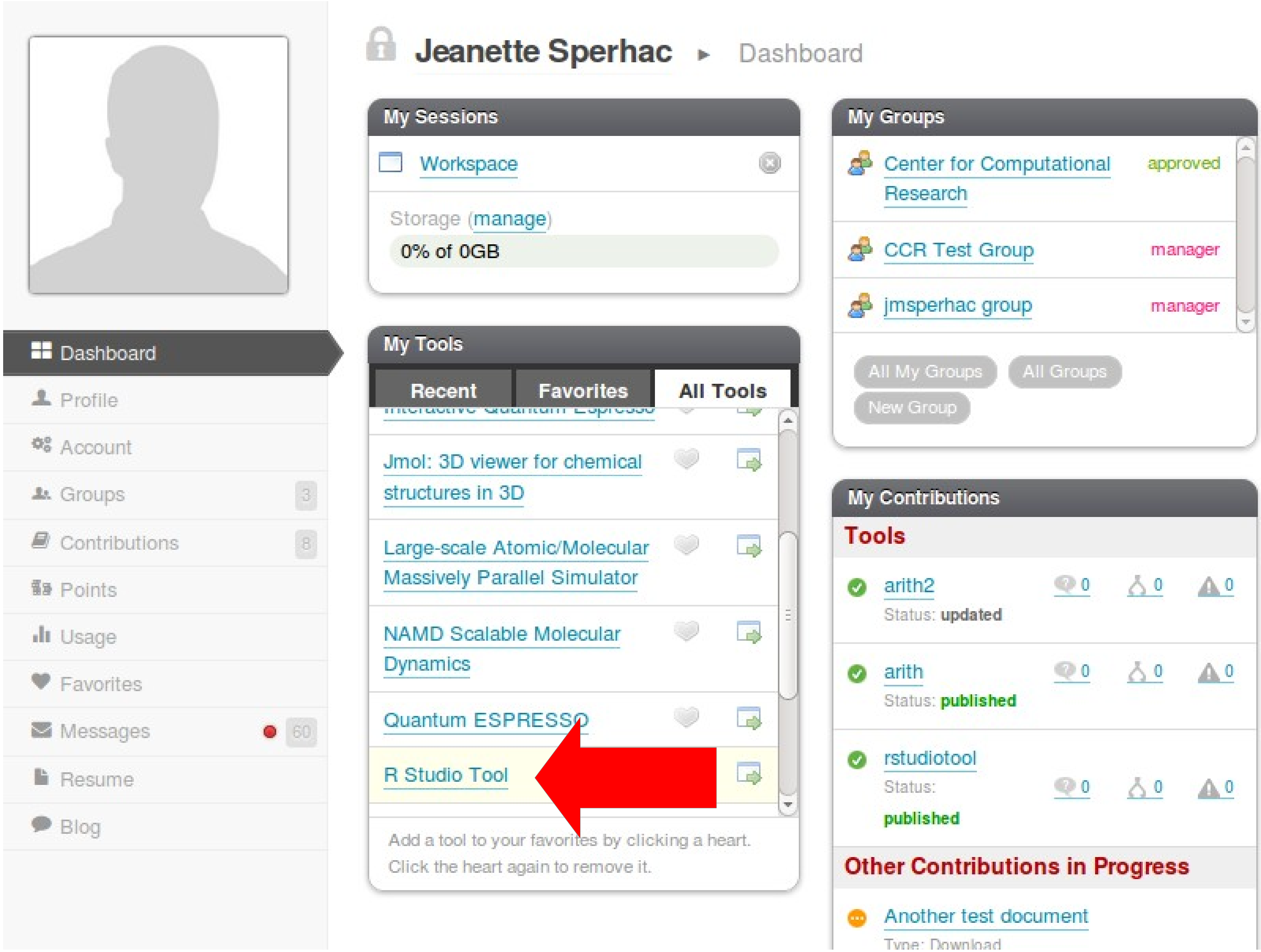


- Introduce the R language
- Do some programming
- Examine, model, and visualize datasets
- Project: explore and characterize protein crystallization data from HWI

Intro to

1. Using the command line;
variables;
and a test flight

hpc2 My Tools: R Studio Tool



...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/

R Practical Matters

- R is case sensitive (R != r)
- Command line prompt is >
- To run R code: use command line, or save script and
  source(“script_name”)
- To separate commands, use ; or a newline
- The # character marks a non-executed *comment*
- To display help files:

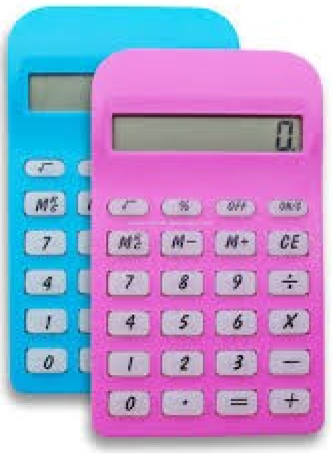
  ?<command-name> or ??<command-name>

R as a calculator

```
> 2 + 3 * 5        # Order of operations.
> (2 + 3) * 5

> 2+3*5            # Equivalent to the above!
                   # Spaces are optional.
> (2+3)*5
```

On the command line...

R output



```
> 2 + 3 * 5      # In the console
[1] 17
```

Q: What's that [1] about?

A: R numbers outputs with [*n*]

Try this in the command line:

```
> 1:500
```

About Comments

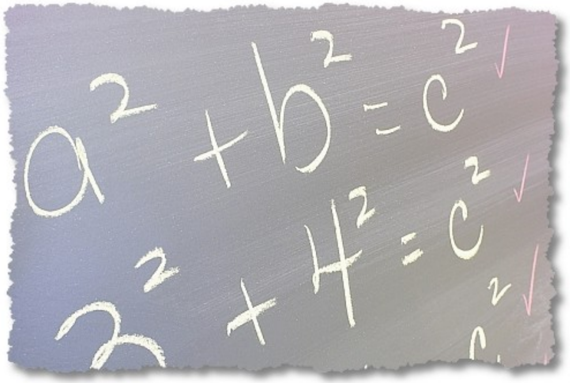
```
> 2 + 3 * 5      # Order of operations.

#  A comment is:
#  Text useful to humans, ignored by computer
#  Helps you understand what code does, or why
#  Denoted by a pound sign in R
```

Use them!!

R as a calculator

Try these in your RStudio console:

```
> 4^2          # 4 raised to the second power
> 3/2          # Division
> sqrt(16)     # Square root
> 3 - 7        # Subtraction
> log(10)      # Natural logarithm
               #        with base e=2.718282
```



Variables: save it

How do we keep a value for later use?

Variable assignment!

```
> y = 2 + 3 * 5        # Do some arithmetic
> y                    # R stores this value as y
[1] 17
```

y can be found under Values in the Workspace window

Variable Assignment

> y = 2 + 3 * 5   # R stores this value as *y*

17 ← 2 + 3 * 5

y

y can be found under Values in the Workspace window

Naming Variables in R

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

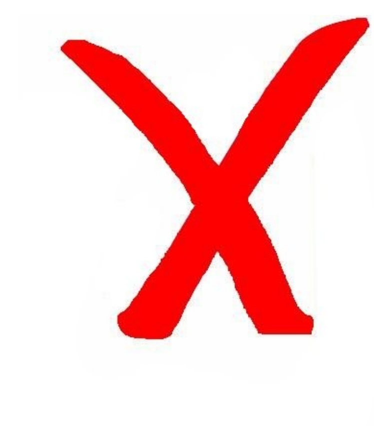
Good:

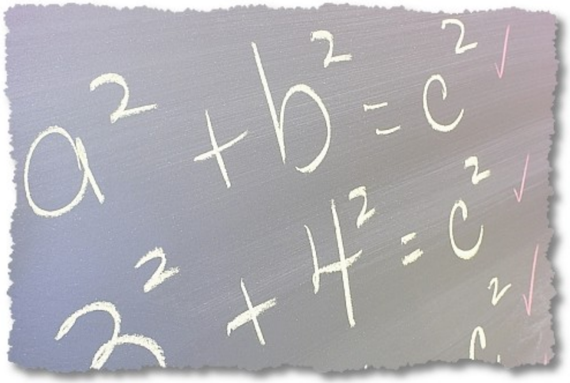
```
> y = 2
> try.this = 33.3
> oneMoreTime = “woohoo”
```

Bad:

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

* *function* is a reserved word in R



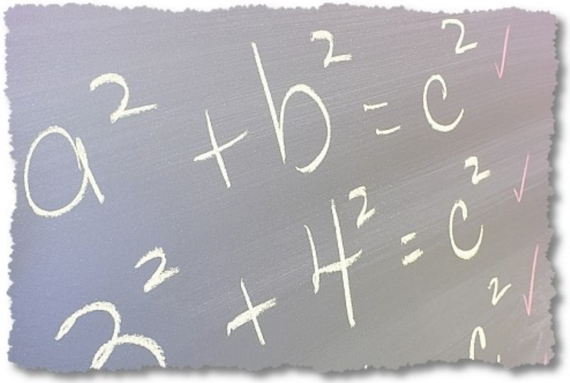


Assign Variables

Try these in your RStudio console:

```
# make variable assignments
> abc = 3
> Abc = log(2.8) * pi
> ABC = "fiddle"
```

Now, check Workspace: Values



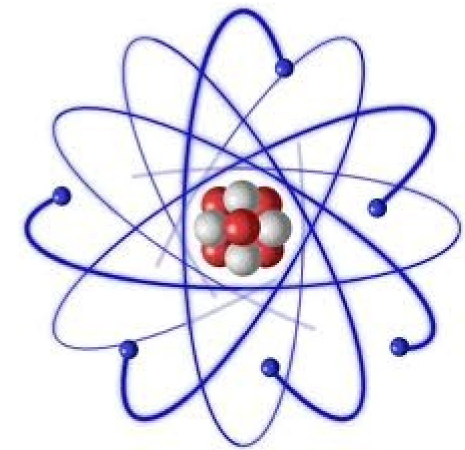
Variables: save it

Alternate R syntax for assignment

```
> y =  2 + 3 * 5
> z <- 2 + 3 * 5          # Same thing as y
```

Variable assignment: Use = or <-

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

Numeric data

Find the type of a variable using class()

```
> class(8)                  # numeric type
[1] "numeric"

> class(6.02e+24)           # numeric type
[1] "numeric"

> class(pi)                 # numeric type (predefined in R)
[1] "numeric"
```

Character and Logical data

Find the type of a variable using class()

```
> class("phooey")  # character type:
[1] "character"    # notice the quotes

> class(TRUE)      # logical type: no quotes
[1] "logical"

> class(NA)        # NA (no quotes) means “no value known”
[1] "logical"
```

RStudio test flight

To whet your appetite for RStudio, let's try:

- Using the editor
- Entering data
- Making a plot in R
- Sourcing a file

The M&M Exercise

On your workstation:

- Sign in to hpc2.org
- Start the RStudio tool
- Create/Access Project from GitHub

  git://github.com/jsperhac/workshop-dev.git
- Files pane: click *examples*, then

  mm-single-example.R

The M&M Exercise

Inside mm-single-example.R:

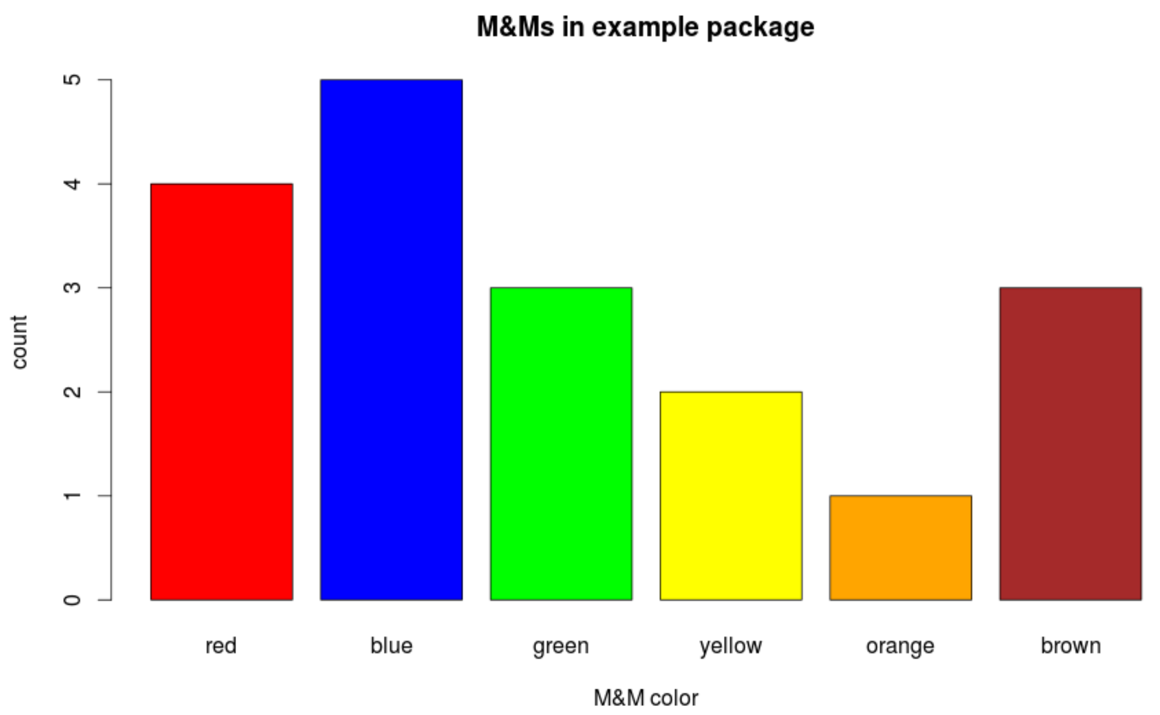
- Change the M&M color counts in the mv variable
- Edit ptitle, if you want

```
# EDIT HERE: ...
mvl = c("red", "blue", "green", "yellow", "orange", "brown")
mv = c(   4,      5,       3,        2,        1,       3)
ptitle = "M&Ms in example package"
```

The M&M Exercise

Inside mm-single-example.R:

- Save the file (File:Save)
- Source the file (Source button)



The M&M Exercise

Questions:

- What have you plotted?
- What outputs does R provide in the console?
- What variables were created?
- What else happens inside this source file?

OK, now you can eat...

The M&M Exercise

- Distribution of colors across many samples
- Increase the number of samples—reveal the underlying distributions
- Barplot
  - Counts of colors in one sample
- Histogram
  - Instances of color counts across all samples

Using Logical Operators

1==2 # equivalence test: *double equals*

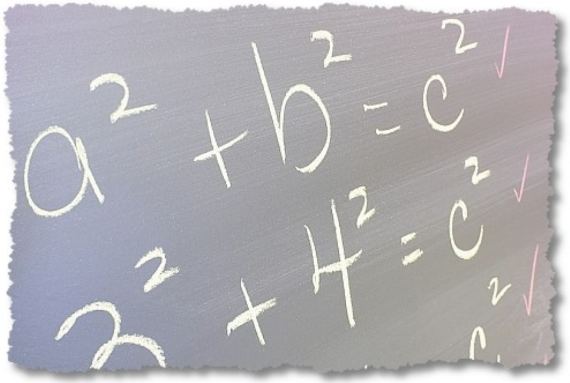
9 != 19 # “not equal” test

3 < 204 # less-than test

18 > 44 # greater-than test

“tree”==89 # comparing mixed data types

What should the results of these tests be?



A logical test

Compare R syntax for assignment

```
> y =  2 + 3 * 5
> z <- 2 + 3 * 5      # Same thing as y

> y==z                 # Here's the test...
[1] TRUE
```

Logical data

A logical value is often created from a comparison between variables.

```
u & v        # Are u AND v both true?
u | v        # Is at least one of u OR v true?
!u           # "NOT u" flips the logical value of
             #          variable u
```

Learning about Object x

R stores everything, variables included, in Objects.

Object x

```
> 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)          # is.na() tests whether a value has a
                    # known value
[1] FALSE
```

Interlude

Complete variable/atomic datatype exercises.

Open in the RStudio source editor:

<workshop>/exercises/exercises-variables-atomic-datatypes.R

Interlude++

Browse some Resources:

- http://jaredknowles.com/s/Tutorial1_Intro.html
- https://github.com/hadley/devtools/wiki/vocabulary

Intro to

2. Data Structures:
Vectors and Data Frames

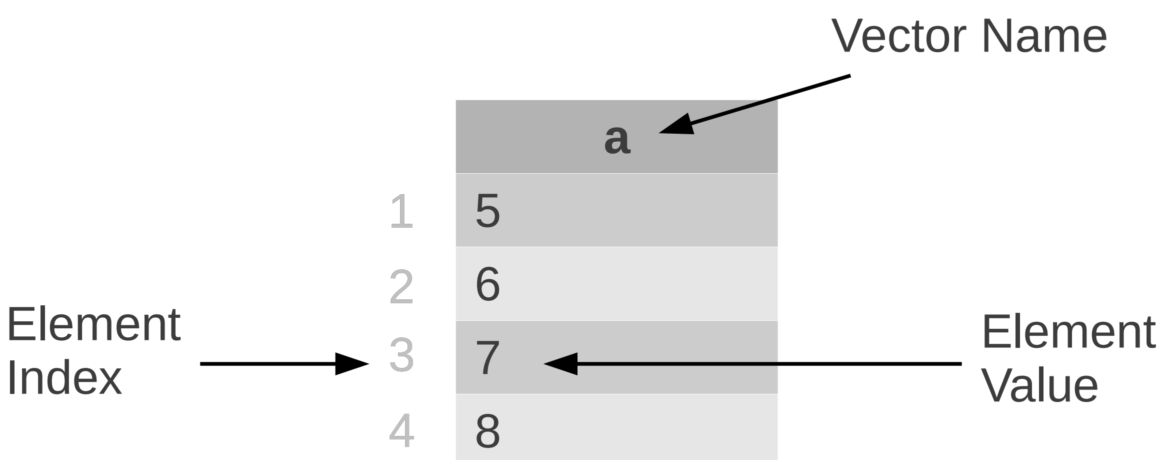
Data Objects in R

These objects, composed of multiple atomic data elements, are the bread and butter of R:

- Vectors
- Data Frames

Vector Data Object

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



Construct a Vector Data Object

	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")  # character 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

Information about a vector

	y
1	3
2	5
3	7
4	9

```
> y <- c(3,5,7,9)   # vector with 4 numeric values

> length(y)         # how many elements?
> class(y)          # class of a vector object is the class
                    # of its elements
```

Information about a vector

	y
1	3
2	5
3	7
4	9

```
> str(y)      #  structure of the vector: number of
              #  elements, type, and contents
```

num [1:4] 3 5 7 9

Type of
elements

Number
(and positions)
of elements

contents

Some operations on vectors

- 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

Repercussions of NA

Any arithmetic operation on a structure containing an NA generates NA!

```
# NA means "no value known"
> y = c(1, NA, 3, 2, NA)
> sum(y)
[1] NA
```

We must *remove NAs* to make calculations. How?

Finding NAs in a data structure

```
> y = c(1, NA, 3, 2, NA)

> summary(y)
   Min.  1st Qu.  Median    Mean  3rd Qu.    Max.    NA's
    1.0      1.5     2.0     2.0      2.5     3.0       2
```

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
```

rm = “remove”

Data Frames

- A data frame is a structure consisting of columns of *various modes* (numeric, character, etc).
- Its rows and columns can be named.
- Data frames are handy containers for experimental data.

Data Frame Example

Data frames are handy containers for data that describe experimental subjects.

Student population data:

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

Constructing a Data Frame

1. Construct the vectors that hold column data:

```
height   = c(68, 75, 60)          # inches
age      = c(16, 17, 16)          # years
handed = c("L", "R", "R")       # dominant hand: R=right, L=left
```

2. Construct the data frame by associating the columns:

```
data = data.frame(Height=height,
                  Age=age,
                  Hand=handed)
```

Name of the column!

Data Frame

Organized in rows and columns:

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

Rows

Columns (formed from vectors)

Accessing by Index

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

First index is row, second index is column:

```
> data[1,1]  # retrieves subject A's Height
```

Accessing by Index

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

```
> data[1, ]   # retrieves all subject A data
    Height Weight  Age Hand
A       68       120    16       L
```

Comma is a placeholder in the [row, column] notation

```
> data[ ,1]   # retrieves all Height data
[1] 68 75 60
```

Try it:
Accessing by Index

```
> source("data-frame-simple-example.R")
> data[2,3]  # retrieves subject B's Age
> data[2, ]   # retrieves all subject B data
> data[ ,3]   # retrieves all Age data
```

Accessing by Name

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

First is row, second is column:

```
> data["A","Height"]  # retrieves subject A's Height
                      # Notice the quotes!
```

Accessing by Name

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

```
> data["A", ]     # retrieves all subject A data.
                  # Notice the comma!
```

Accessing by Name

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

To fetch Height column:

```
> data$Height
```

Try it:
Accessing by Name

```
> source("data-frame-simple-example.R")
> data["B","Age"]  # retrieves B's Age
> data["B", ]      # retrieves all B data
> data$Age         # retrieves all Age data
```

Conditional Access

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

Subjects who are taller than 65 inches:

```
> data[data$Height > 65, ]  # subset of the data frame
                            # (notice the comma!)
```

Conditional Access

	Height	Weight	Age	Hand
A	68	120	16	L
B	75	160	17	R
C	60	118	16	R

Heights over 65 inches:

```
> data$Height[data$Height > 65]   # subset of a column
                                  # of the data frame
```

Try it:
Conditional Access

> source(“data-frame-simple-example.R”)

subset of the data frame having age<17 years:

> data[data$Age < 17,]

subset of a *column* of data frame, age<17 years:

> data$Age[data$Age < 17]

Data Frame Information

```
str(data)        # structure
dim(data)        # dimensions

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

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

```
> class(data)
[1] "data.frame"
```

Interlude

Complete vector/data frame exercises.

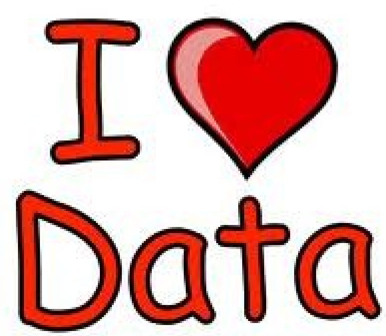
Open in the RStudio source editor:

<workshop>/exercises/exercises-vectors-matrices-dataframes.R

Intro to R

3. Descriptive Statistics

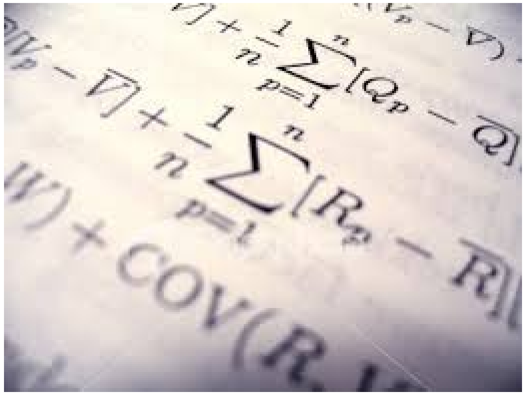
Descriptive Statistics



Explore a dataset:

- What's in the dataset?
- What does it mean?
- What if there's *a lot* of it?

Basic statistical functions in R



Wanted: measures of the center and the spread of our numeric data.

- mean()
- median()
- range()
- var() and sd()   # variance, standard deviation
- summary()   # combination of measures

mean()

A measure of the data's “most typical” value.

- Arithmetic mean == average
- Divide sum of values by number of values

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

mean()

A measure of the data's “most typical” value.

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
> f <- c(3, 2, 4, 1)
> mean(f)     # == sum(f)/length(f) == (3+2+4+1)/4
[1] 2.5
```

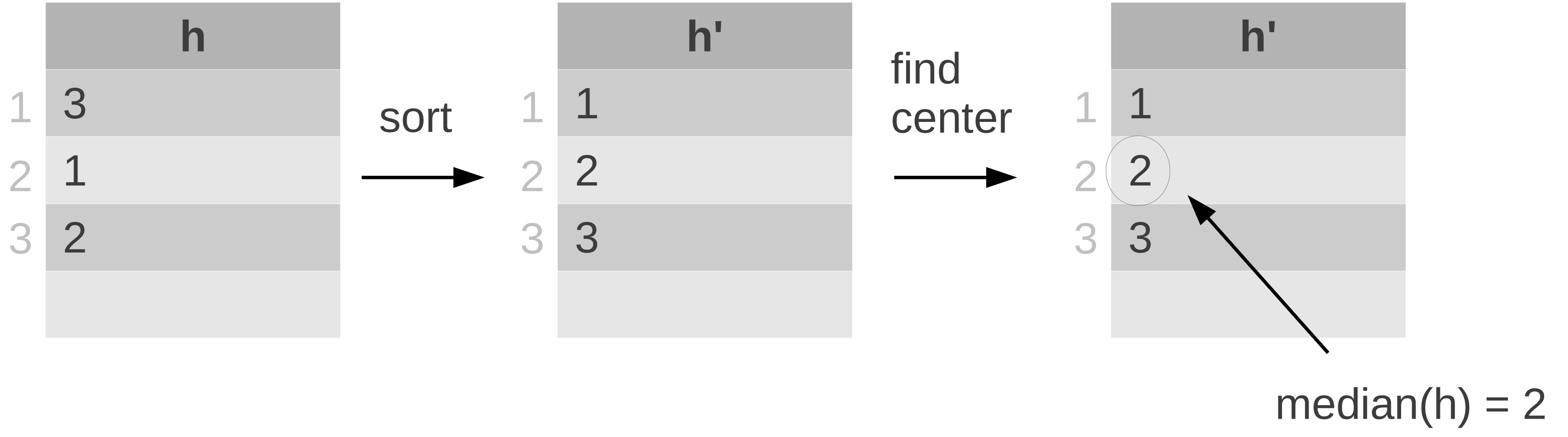
median()

A measure of the data's center value. To find it:

- Sort the contents of the data structure
- Compute the value at the center of the data:
  - For odd number of elements, take the center element's value.
  - For even number of elements, take mean around center.

median()

Odd number of values:



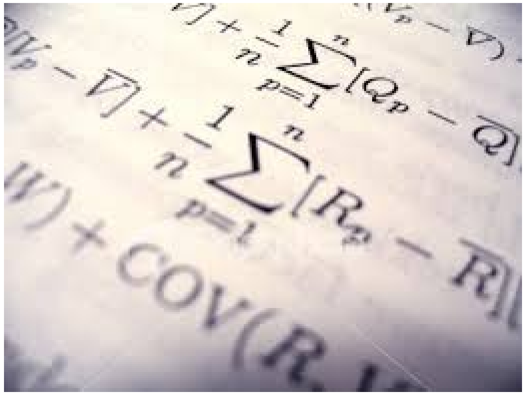

```
> h <- c(3, 1, 2)
> median(h)
[1] 2
```

median()

Even number of values: need to find mean()


```
> f <- c(3, 2, 4, 1)
> median(f)
[1] 2.50
```

range():
min() and max()



range() reports the minimum and maximum values found in the data structure.

```
> f <- c(3, 2, 4, 1)
> range(f)   # reports min(f) and max(f)
  [1] 1   4
```

var() and sd()

- *Variance*: a measure of the spread of the values relative to their mean:

$$Var = s_n^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

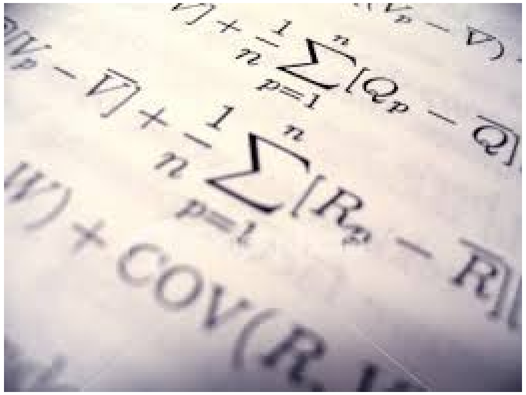
Sample variance

- *Standard deviation*: square root of the variance

$$s_n = \sqrt{Var}$$

Sample standard deviation

R's summary() function

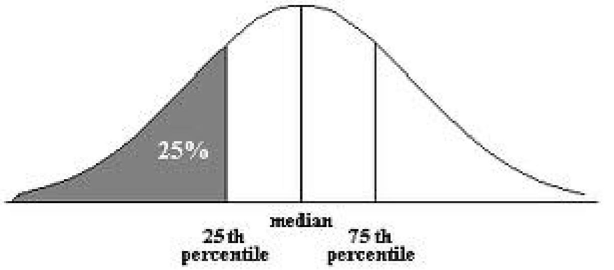


Provides several useful descriptive statistics about the data:

```
> g <- c(3, NA, 2, NA, 4, 1)
> summary(g)
 Min.   1st Qu.  Median   Mean   3rd Qu.   Max.   NA's
 1.00     1.75     2.50   2.50      3.25   4.00      2
```

Quartiles: Sort the data set and divide it up into quarters...

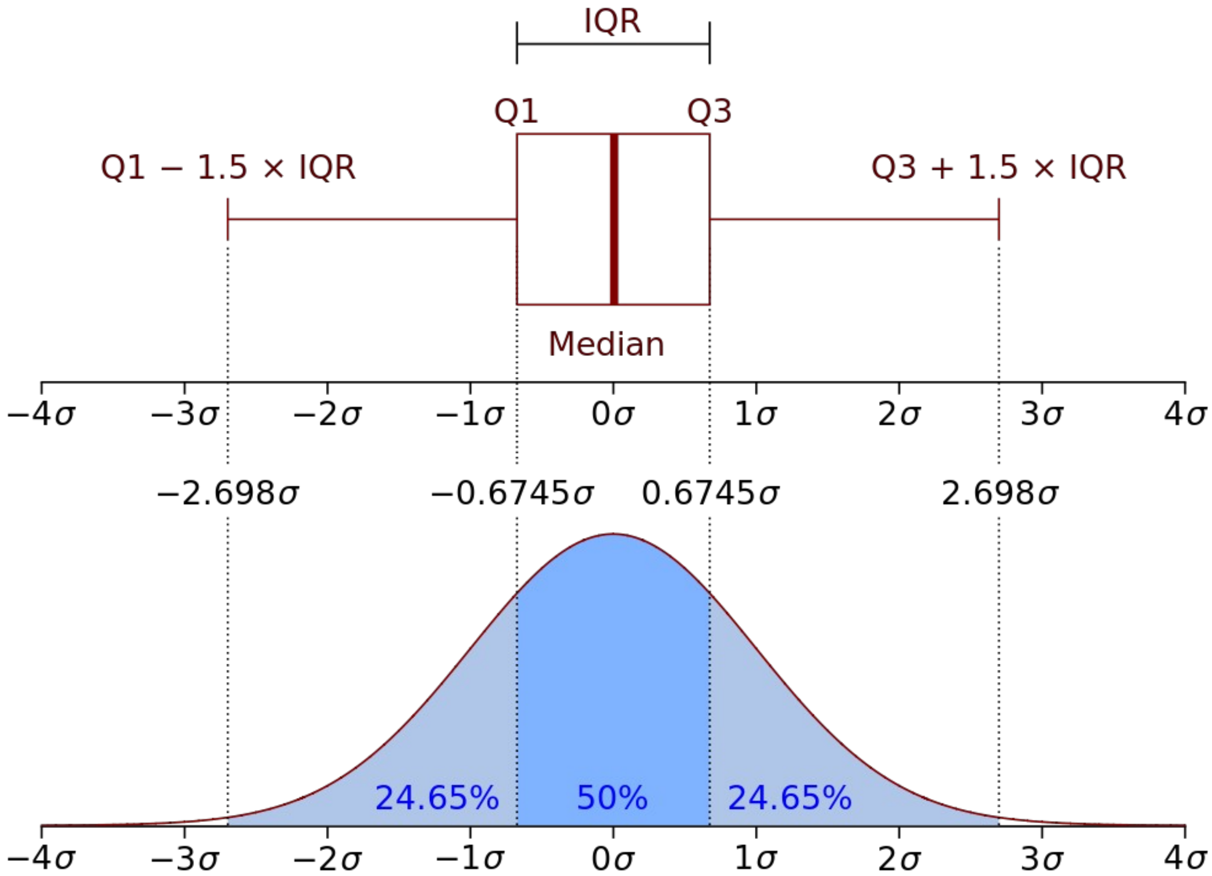
Quartiles



Quartiles are the *three points* that divide ordered data into four equal-sized groups:

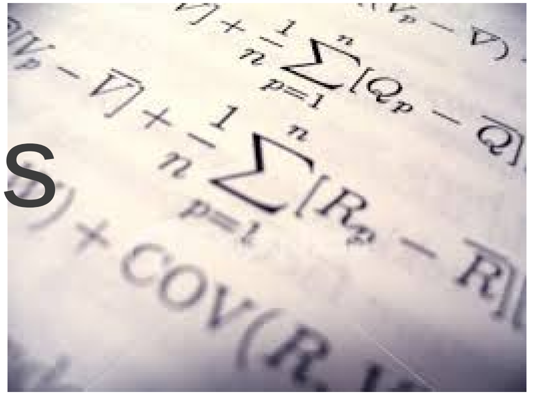
- Q1 marks the boundary just above the lowest 25% of the data
- Q2 (the *median*) cuts the data set in half
- Q3 marks the boundary just below the highest 25% of data

Quartiles



Boxplot and probability distribution function of Normal N(0,1σ^2) population

Summary: basic statistical functions



- Characterize the center and the spread of our numeric data.
- Comparing these measures can give us a good sense of our dataset.

Statistics and Missing Data

If NAs are present, specify na.rm=TRUE to call:

- mean()
- median()
- range()
- sum()
- ...and some other functions

R disregards NAs, then proceeds with the calculation.

diamonds data

50,000 diamonds, for example:

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31

What can we learn about these data?

diamonds data
summary()

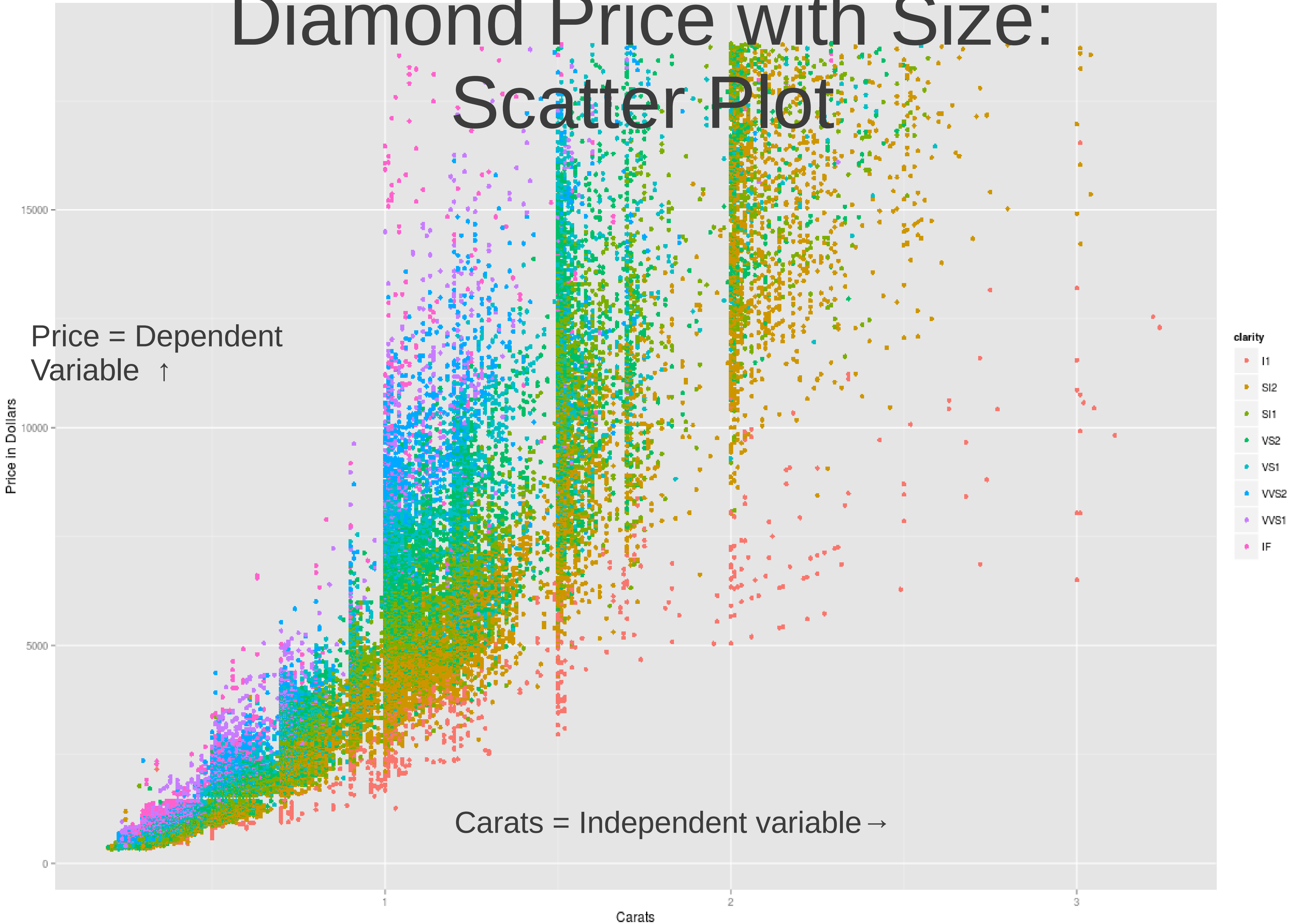
Information provided by summary() depends on the type of data, by column:

```
    carat               cut           color         price
Min.   :0.2000   Fair     : 1610   D: 6775   Min.   :  326
1st Qu.:0.4000   Good     : 4906   E: 9797   1st Qu.:  950
Median :0.7000   Very Good:12082   F: 9542   Median : 2401
Mean   :0.7979   Premium  :13791   G:11292   Mean   : 3933
3rd Qu.:1.0400   Ideal    :21551   H: 8304   3rd Qu.: 5324
Max.   :5.0100                     I: 5422   Max.   :18823
                                   J: 2808
```

numeric data:
statistical summary

categorical (factor) data:
counts

Diamond Price with Size: Scatter Plot



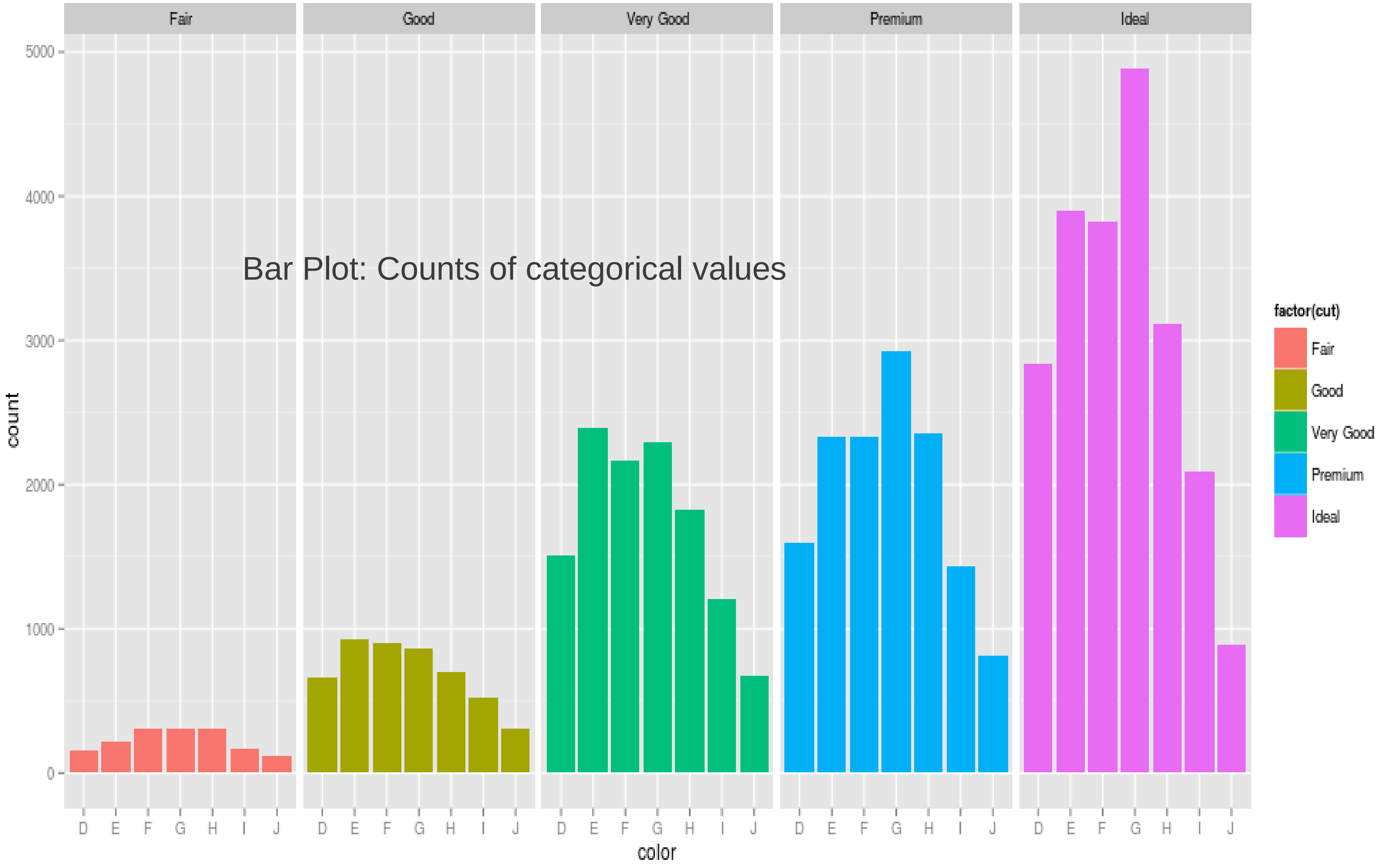
table() function

Contingency table: counts of categorical values for selected columns

```
> table(diamonds$cut, diamonds$color)
```

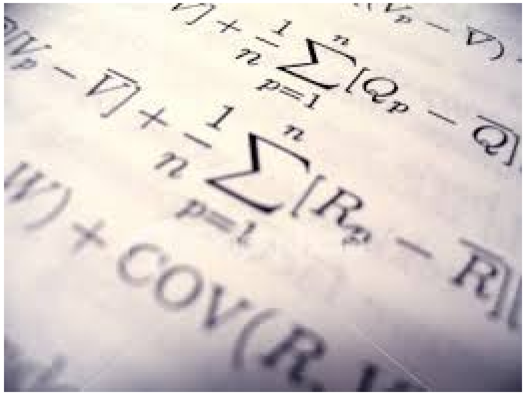
	D	E	F	G	H	I	J
Fair	163	224	312	314	303	175	119
Good	662	933	909	871	702	522	307
Very Good	1513	2400	2164	2299	1824	1204	678
Premium	1603	2337	2331	2924	2360	1428	808
Ideal	2834	3903	3826	4884	3115	2093	896

Diamond Color and Cut



Bar Plot: Counts of categorical values

Correlation



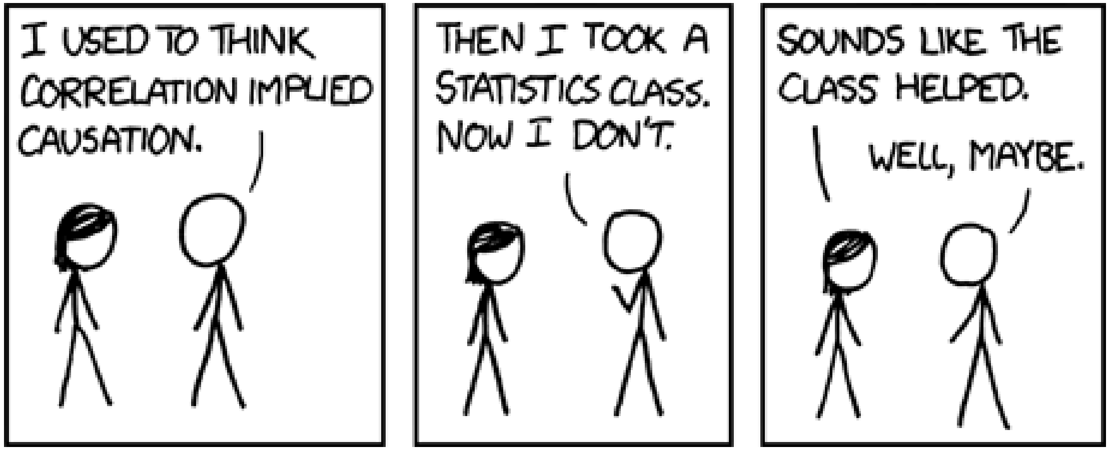
Do the two quantities X and Y vary together?

- Positively: $0 < \rho < 1$
- Or negatively: $-1 < \rho < 0$

$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

A pairwise, *statistical* relationship between quantities

Correlation



$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

NOTE: Correlation does not imply causation...

Looking for correlations

diamonds data frame: 50,000 diamonds

- carat: weight of the diamond (0.2–5.01)
- table: width of top of diamond relative to widest point (43–95)
- price: price in US dollars
- x: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)

cor() function

Look at pairwise, *statistical* relationships between numeric data:

```
> cor(diamonds[c(1,6:10)])
```

	carat	table	price	x	y	z
carat	1.0000000	0.1816175	0.9215913	0.9750942	0.9517222	0.9533874
table	0.1816175	1.0000000	0.1271339	0.1953443	0.1837601	0.1509287
price	0.9215913	0.1271339	1.0000000	0.8844352	0.8654209	0.8612494
x	0.9750942	0.1953443	0.8844352	1.0000000	0.9747015	0.9707718
y	0.9517222	0.1837601	0.8654209	0.9747015	1.0000000	0.9520057
z	0.9533874	0.1509287	0.8612494	0.9707718	0.9520057	1.0000000

-1.0: perfectly anticorrelated

↕

0 : uncorrelated

↕

1.0: perfectly correlated

Interlude

Complete descriptive statistics exercises.

Open in the RStudio source editor:

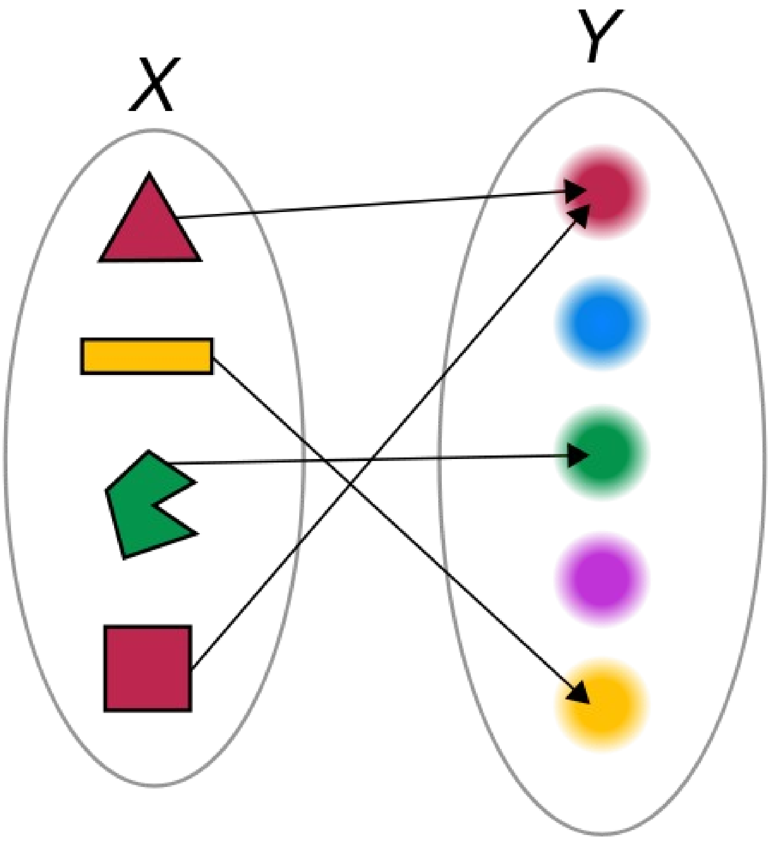
<workshop>/exercises/exercises-descriptive-statistics.R

Intro to R

4. Writing Functions in R

Functions

A function generates an output (Y), given an input (X).



Control Structures: if/else

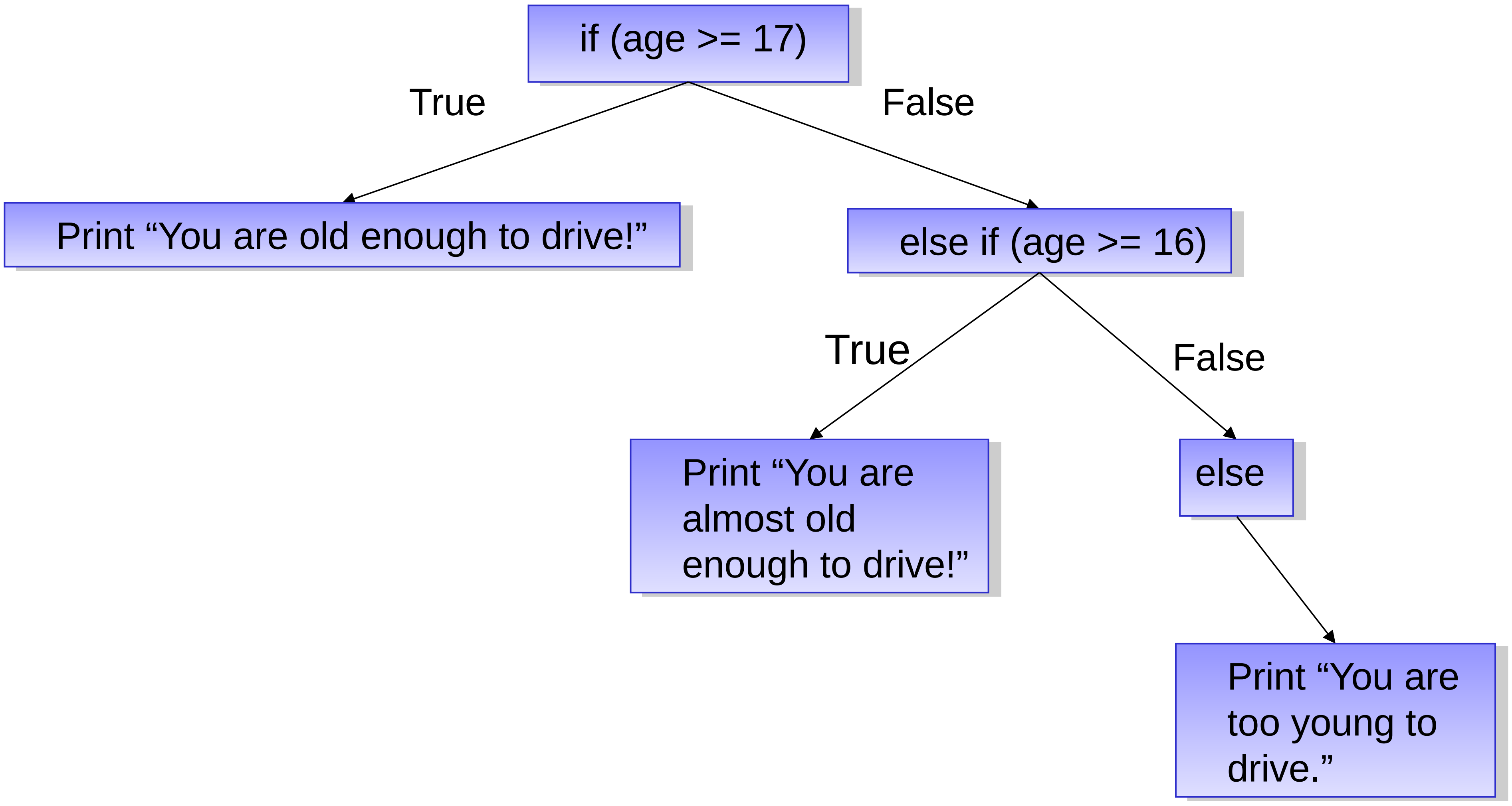
- Make a logical test
- Perform operations based on the outcome

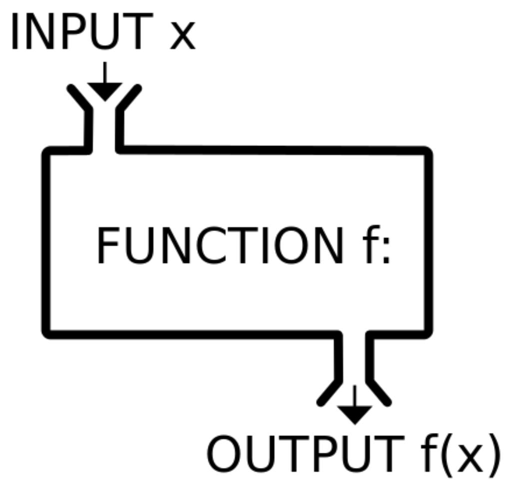
```
if (condition is true)
{
    # do something
}
```

Control Structures: if/else

```
age = 21;

if (age >= 17) {

    print(“You can drive!”);

} else if (age >= 16) {

    print(“You are almost old enough to drive!”);

} else {

    print(“You are not old enough to drive.”);
}
```

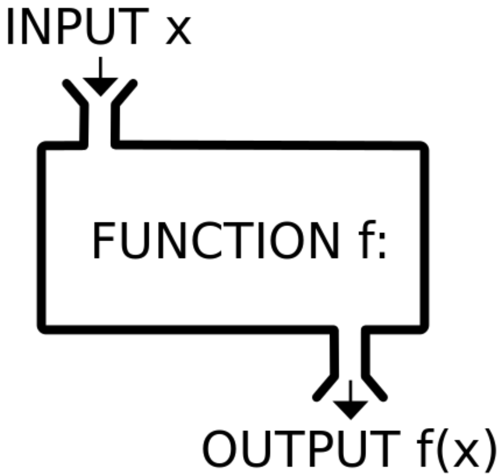
if/else flowchart





Functions

- A function *f* takes an input, x, and returns an output *f*(x).
- It's like a machine that converts an input into an output.



Functions

Function: a piece of code that can be called again and again

To call it, specify:

- Function name
- Input values

It may return an output value

Functions in R

Name of function

Input parameter(s)

```
functionName = function(inputs) {
  # do something
  # return the result
}
```

Declaration
(start of function)

End of function

Functions in R

Name of function

Input parameter(s)

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

Declaration
(start of function)

Output value

End of function

Functions in R

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

Functions in R

```
celsius = c(20:25); # define input temperatures

toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32;  # perform the conversion
  return(f);
}

# call the function to convert temperatures to Fahrenheit:
toFahrenheit(celsius);
[1] 68.0  69.8  71.6  73.4  75.2  77.0
```

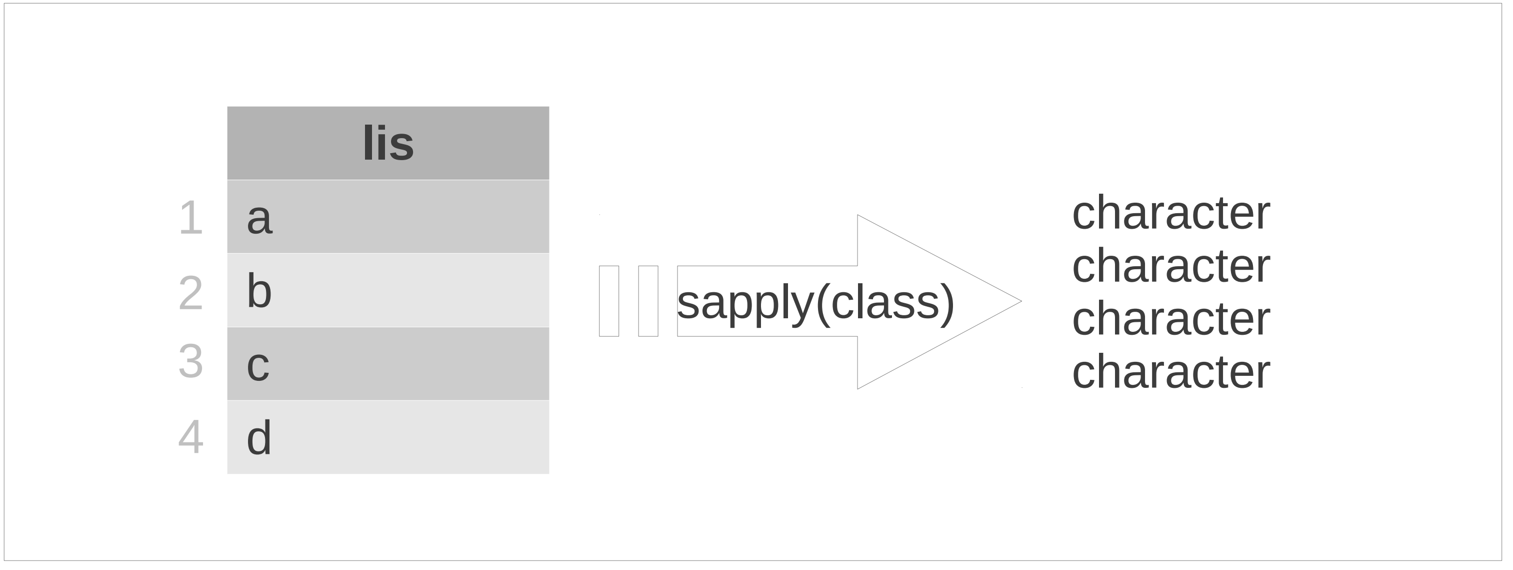
Control Structures: apply() family

- 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*)

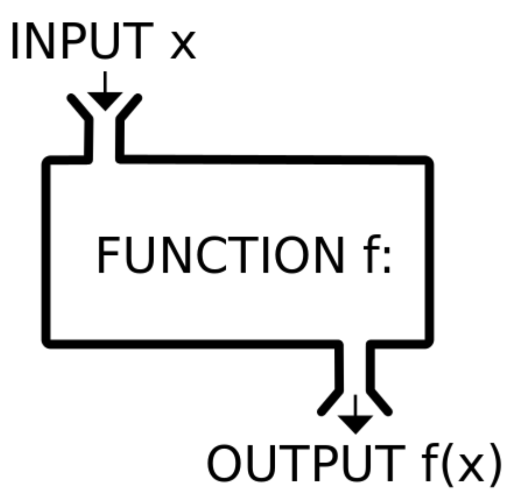
Control Structures: sapply()

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



Tips: Writing Functions

- Use an editor window (not the command line) to compose functions
- Try out one line at a time, and test!
- Comment your function to indicate:
  - input
  - output
  - purpose



Interlude

Complete function exercises.

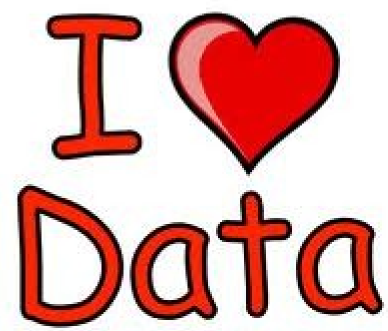
Open in the RStudio source editor:

workshop/exercises/exercises-functions.R

Intro to R

5. Visualizing Data in R

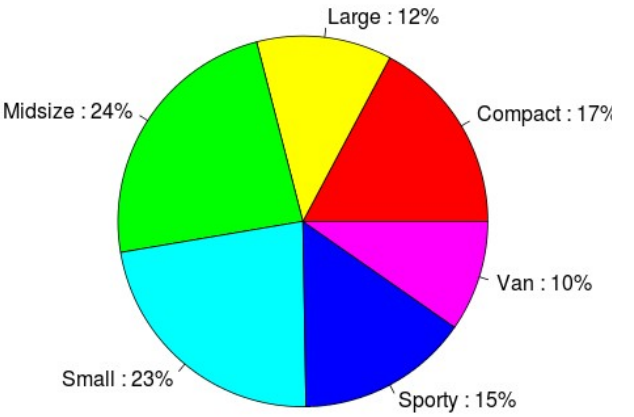
Plotting Data



Plotting is another way to explore a dataset, visually:

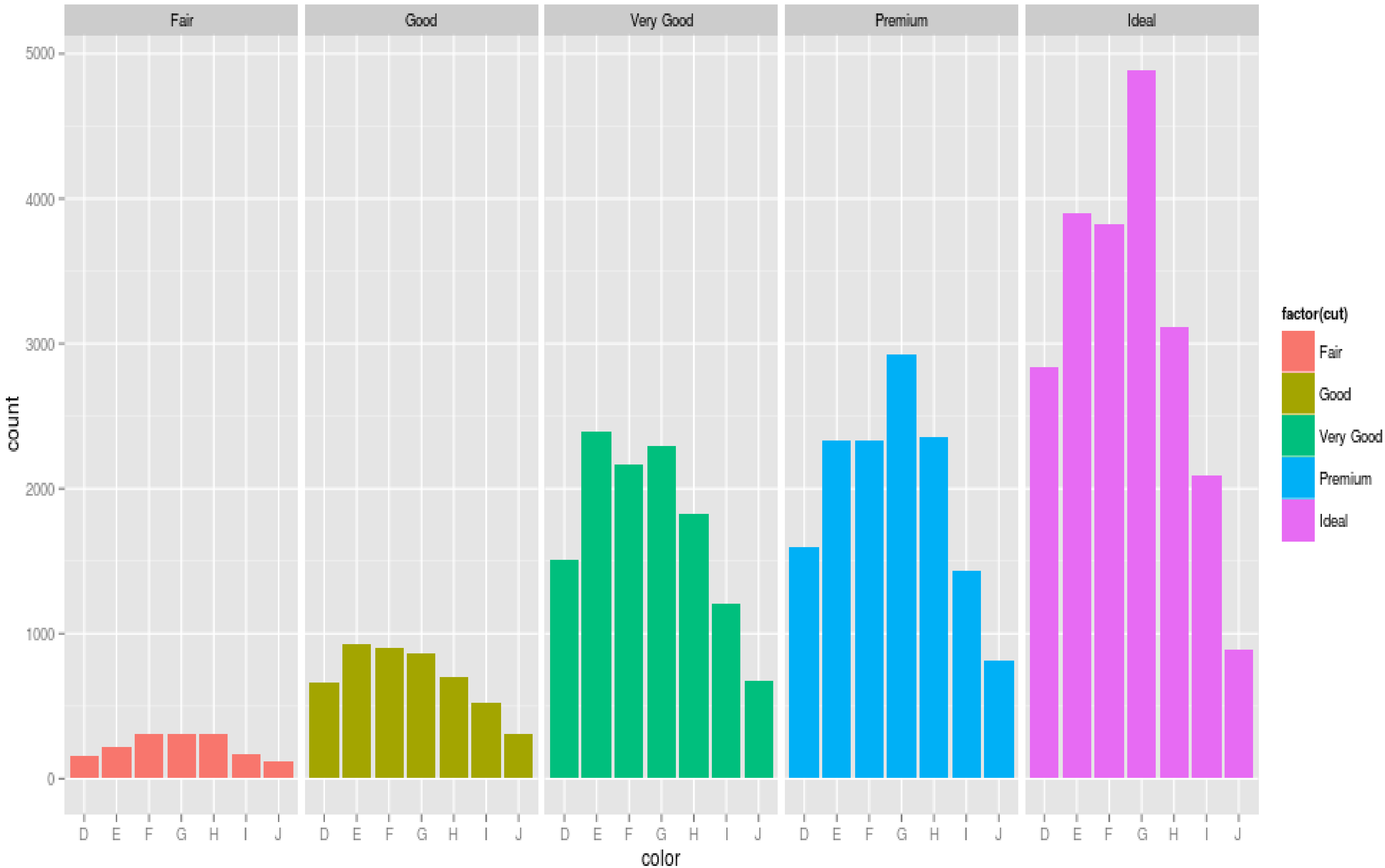
- What's in the dataset?
- What does it mean?
- What if there's *a lot* of it?

Some Plot Types



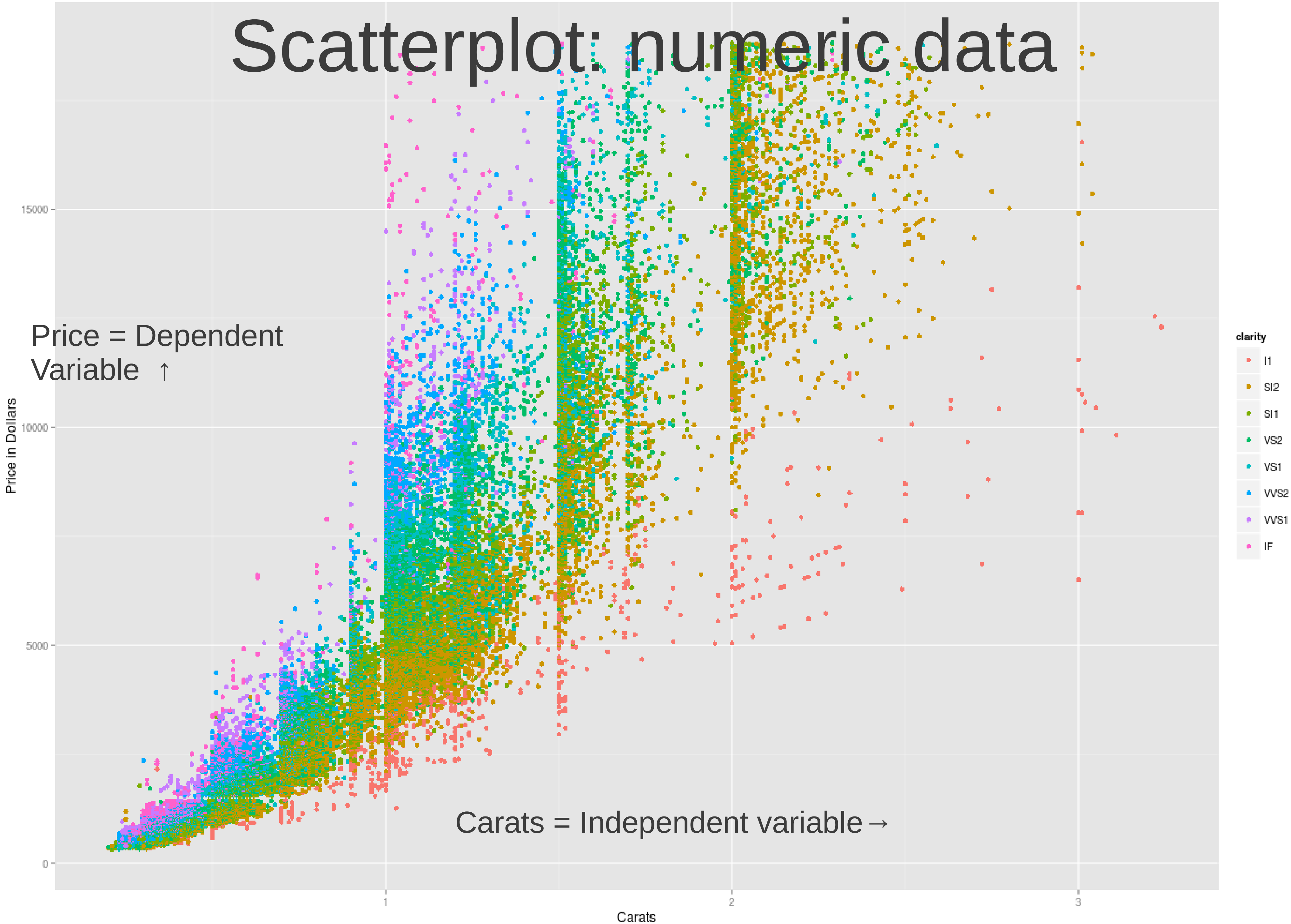
- Pie Chart
  - Display proportions of different values for a variable
- Bar Plot
  - Display counts of values for a categorical variable
- Histogram, Density Plot
  - Display counts of values for a binned, numeric variable
- Scatter Plot
  - y vs. x
- Box Plot
  - Display distributions over different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

Barplot: counts of categorical values



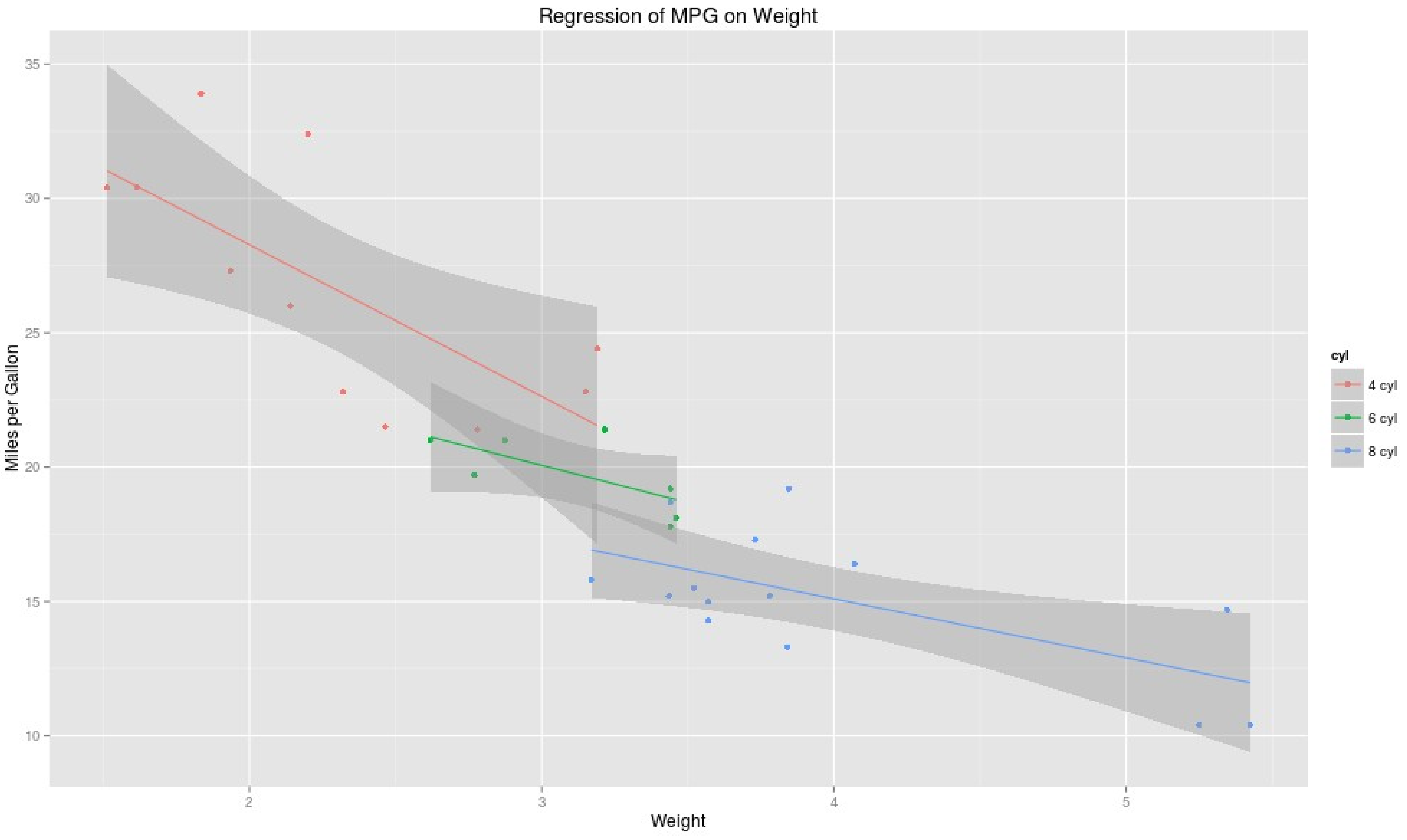
Scatterplot: numeric data

Price = Dependent Variable ↑

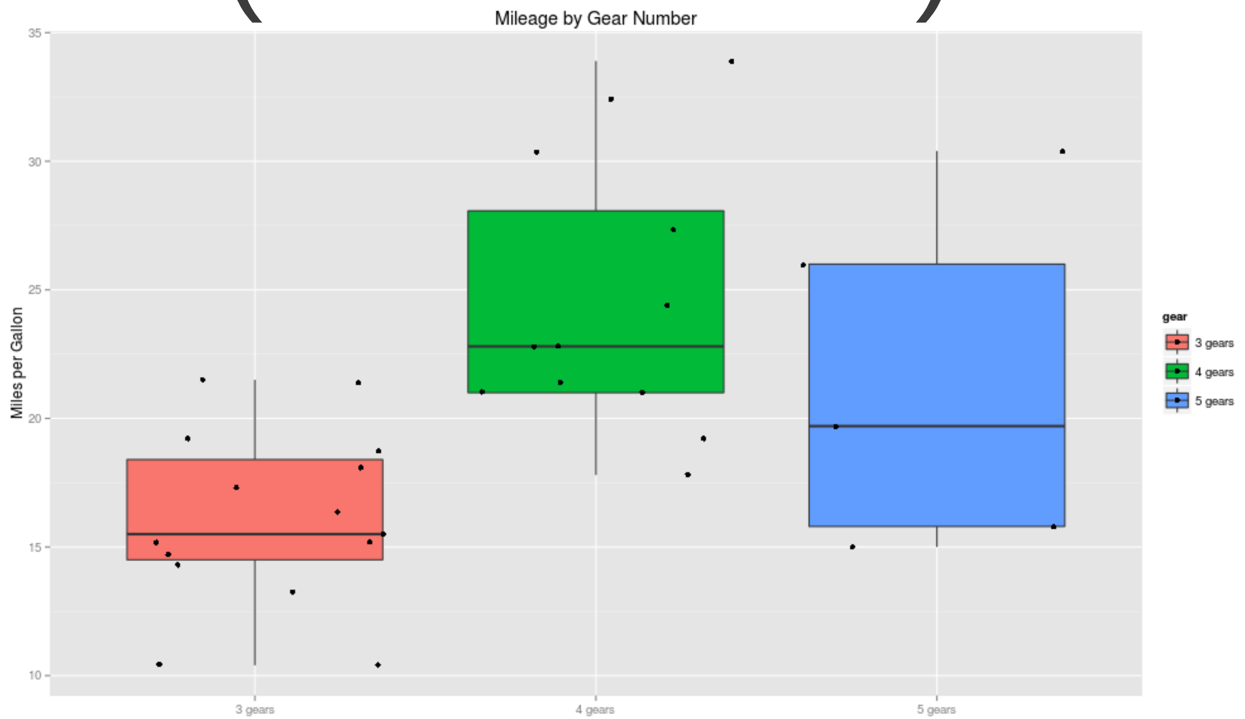


Carats = Independent variable→

Scatterplot with Regression Lines

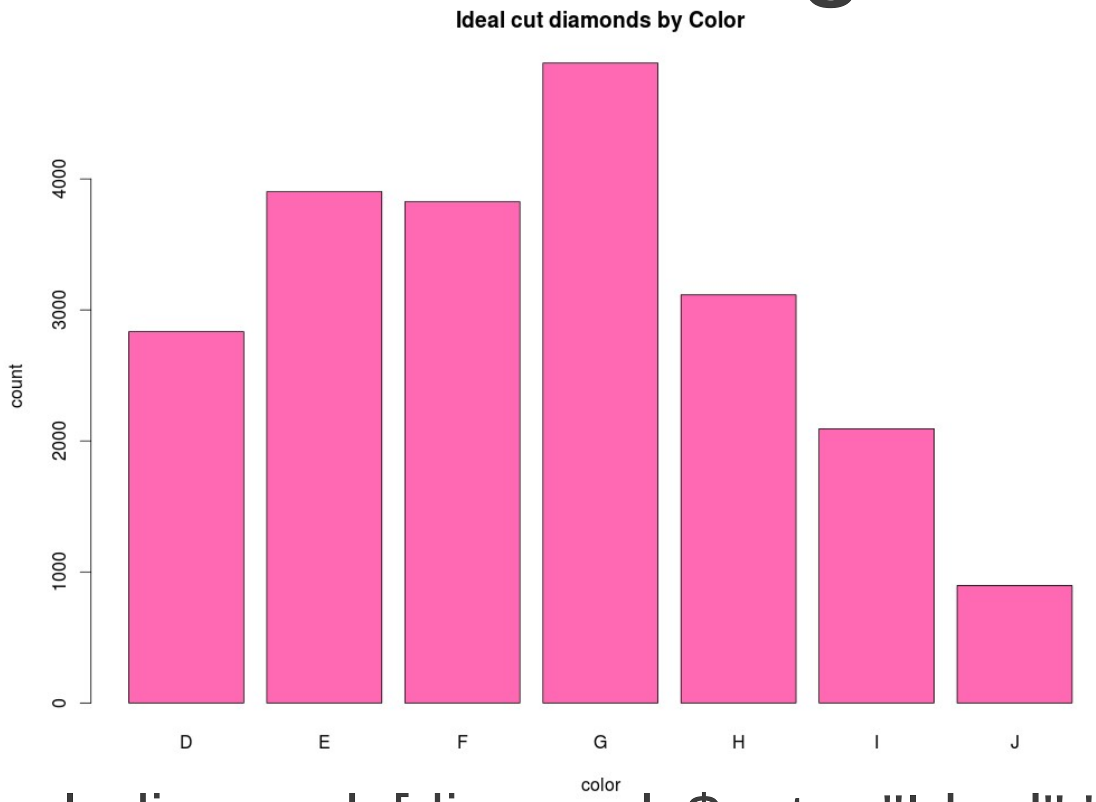


Box (and Whisker) Plot



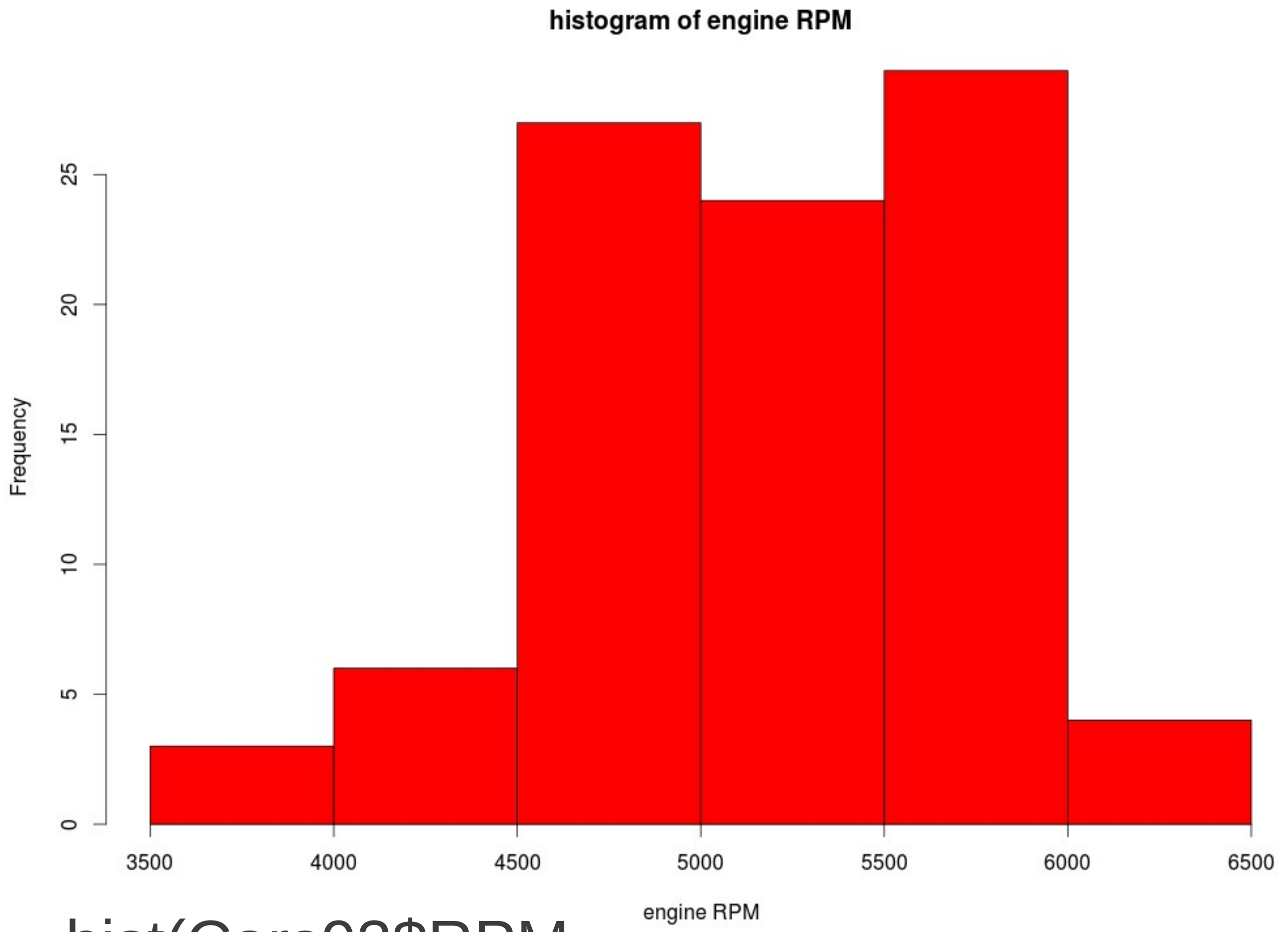
- The *box* extends from Q1 to Q3
- The *median*, Q2, is marked inside the box
- The *whiskers* extend to the min and max
  - Whiskers: required to lie within 1.5×(IQR)
  - *Outliers*: beyond 1.5×(IQR)

Barplot: counts of categorical values



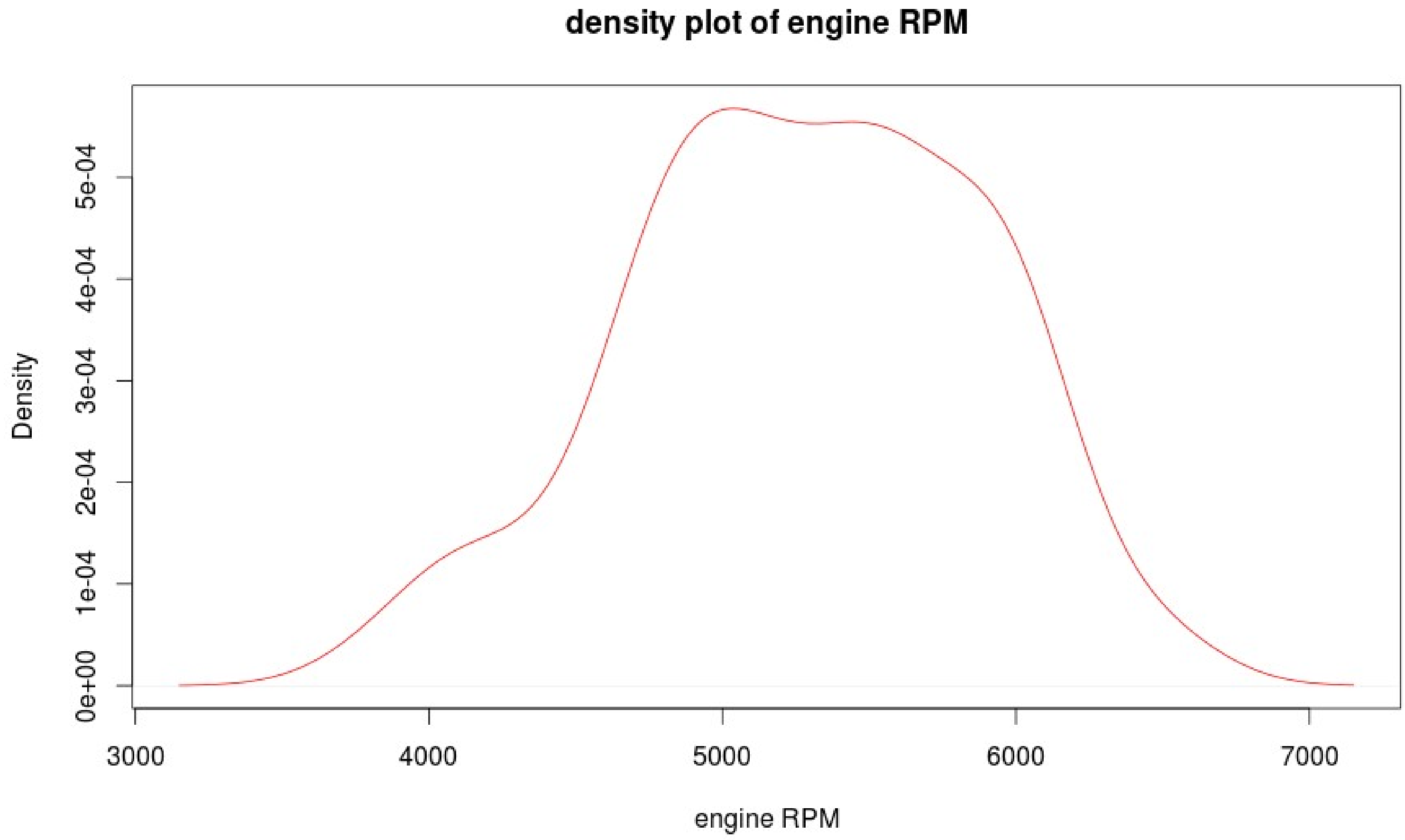

```
ideal=diamonds[diamonds$cut=="Ideal","color"]

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

Histogram: frequencies of numeric values



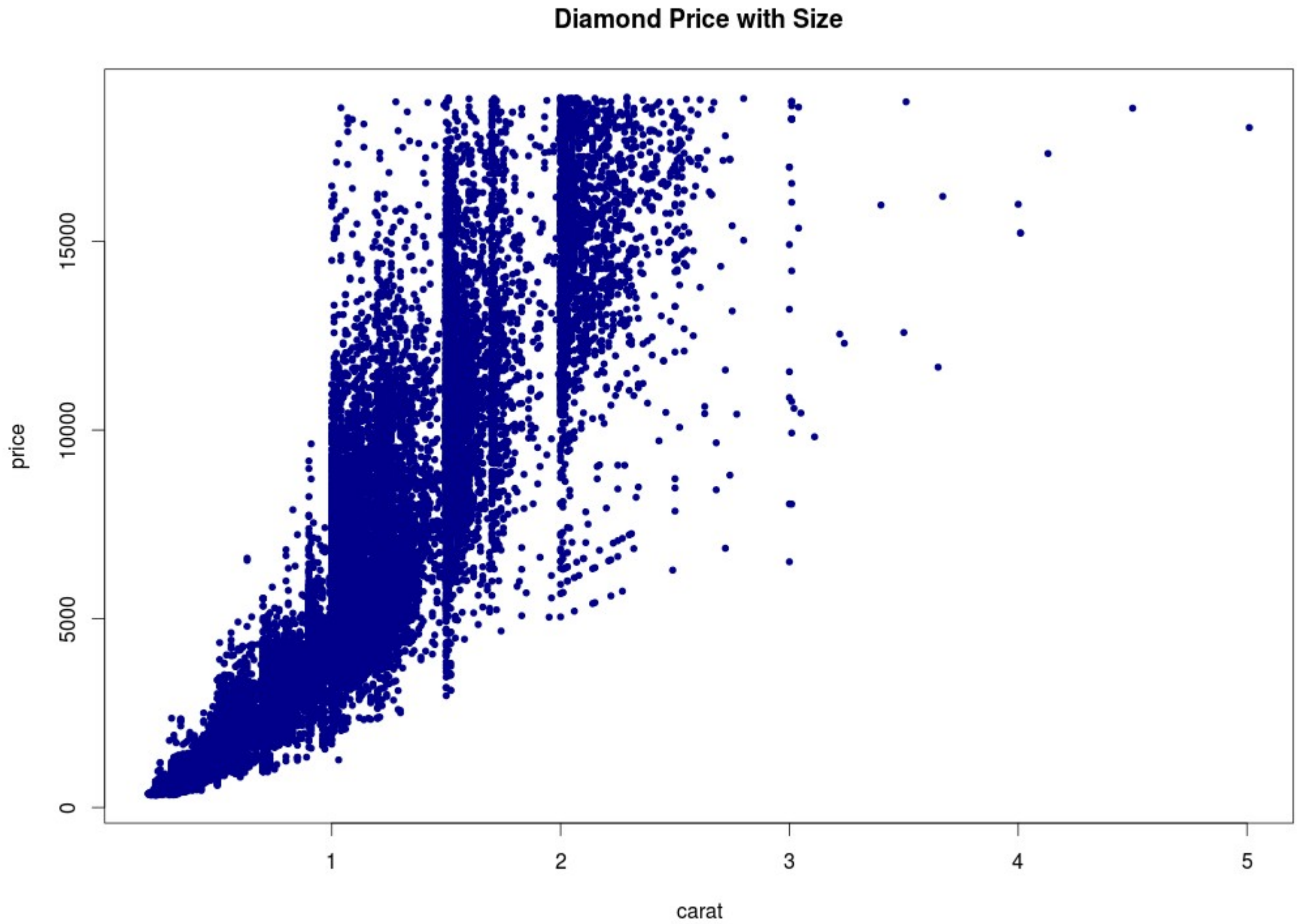

```
hist(Cars93$RPM,
     xlab="engine RPM",
     main="histogram of engine RPM",
     col="red")
```

Kernel Density Plot



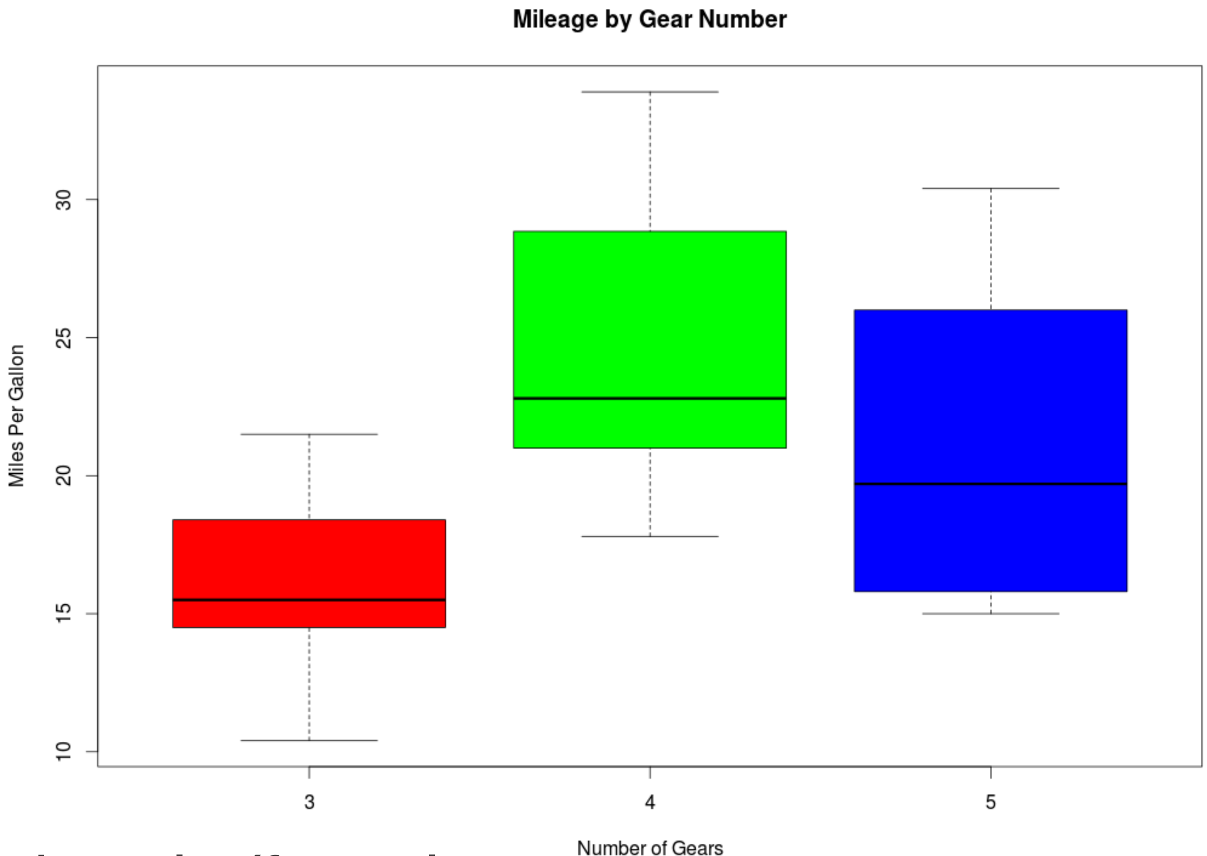

```
plot(density(Cars93$RPM),
     xlab="engine RPM",
     main="density plot of engine RPM",
     col="red")
```

Scatterplot: numeric data, y vs. x




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

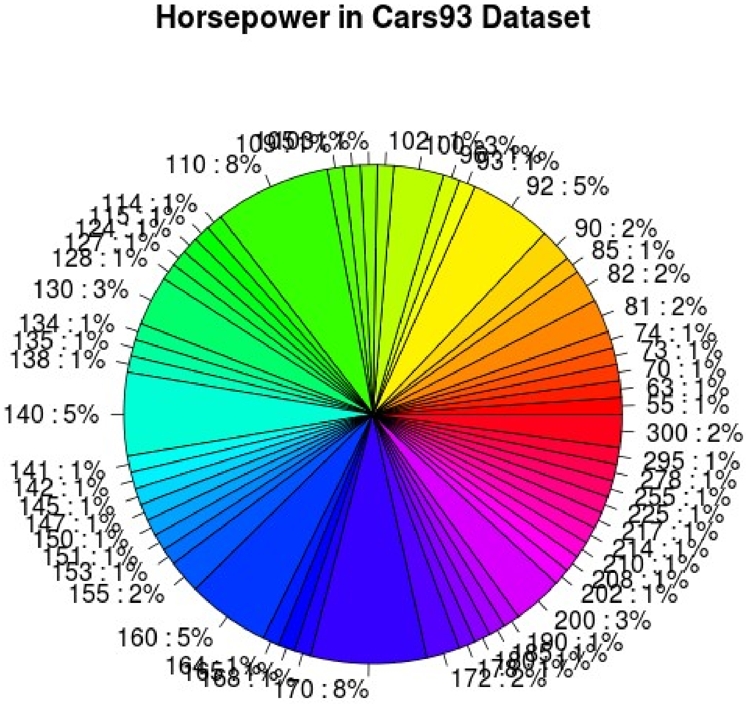
Box (and Whisker) Plot




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

Approach to Plotting

- Remember, you're getting to know your data.
- Don't be afraid to tinker and play.
- Sometimes the outcomes are silly (make sure you learn something!)



pie(table(Cars93$Horsepower))

Interlude

Complete plotting exercises.

Open in the RStudio source editor:

workshop/exercises/exercises-plotting-basic.R

...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/