An Introduction to R

A Programming Environment for Data Analysis and Graphics

Giovanni Seni

September 19, 2018

Table of Contents

R is an environment for...

Data Manipulation

- Connecting to data sources
- Slicing and dicing data

R is an environment for...

Data Manipulation

- Connecting to data sources
- Slicing and dicing data

Modeling and Computation

- Statistical modeling
- Numerical simulation

R is an environment for...

Data Manipulation

- Connecting to data sources
- Slicing and dicing data

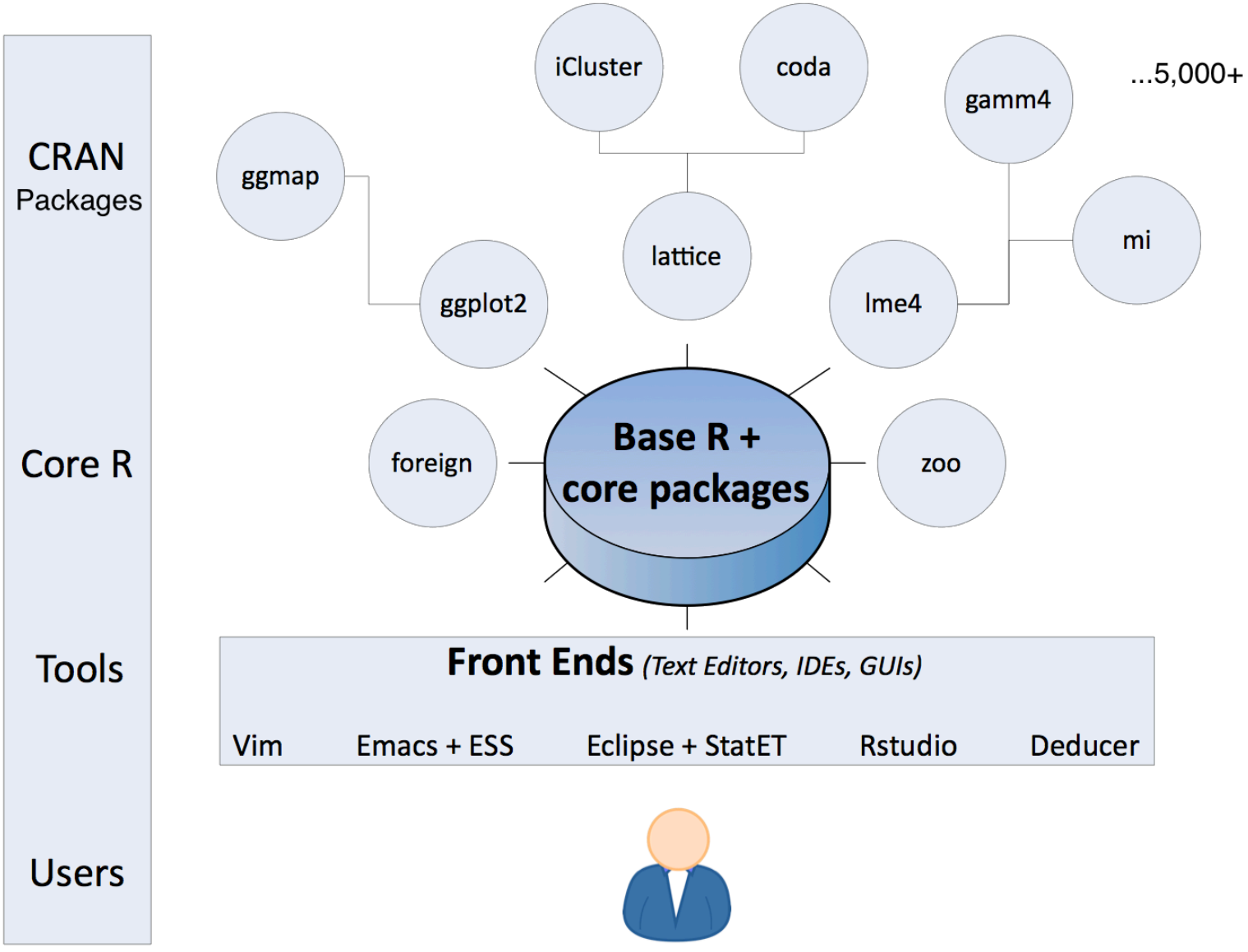
Modeling and Computation

- Statistical modeling
- Numerical simulation

Data Visualization

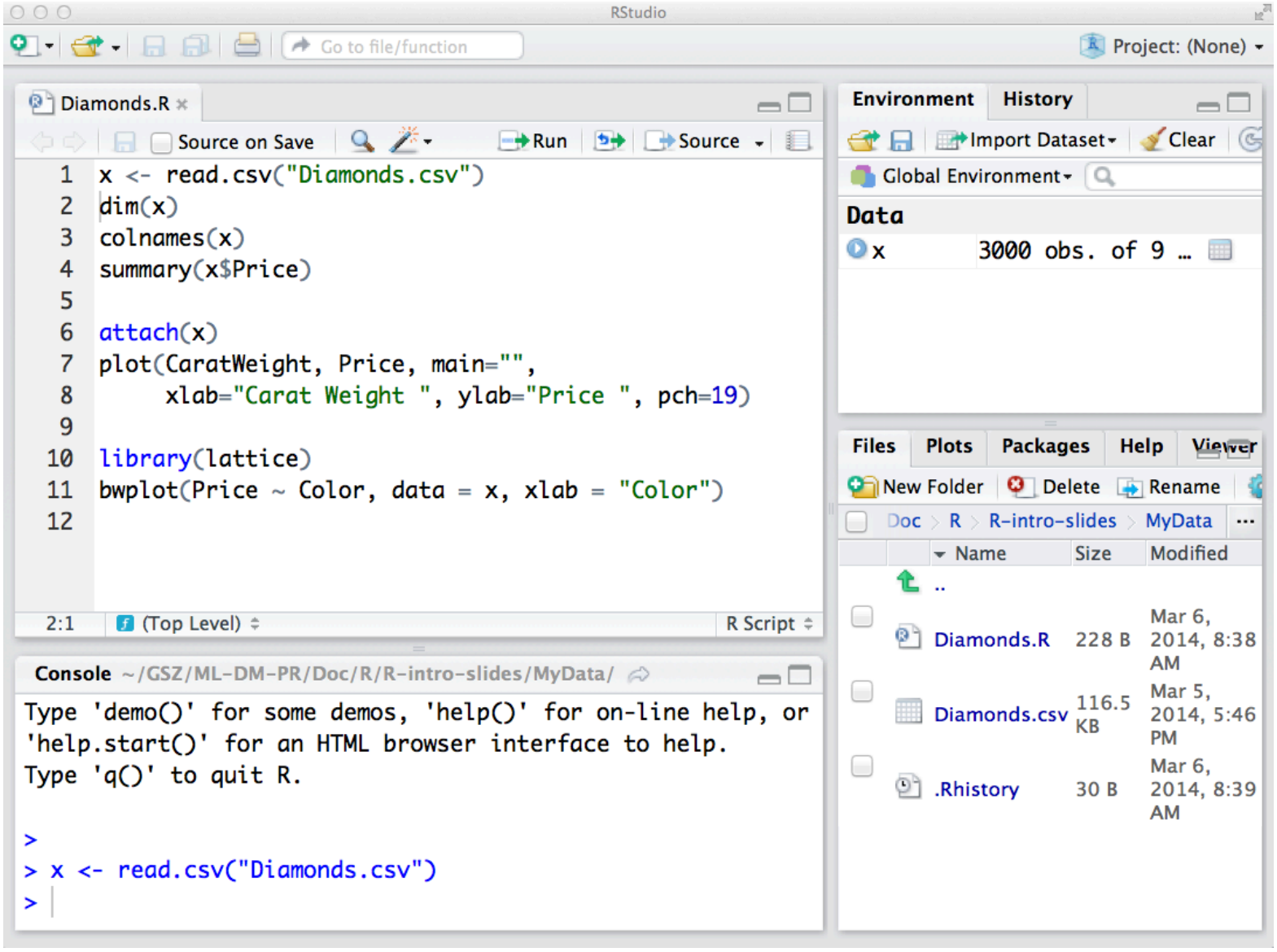
- Visualizing fit of models
- Composing statistical graphics

The R Ecosystem



A Sample Session

```
open -a RStudio Diamonds.R
```



Simple manipulations; numbers and vectors

- Simple math

```
> 2+2
```

[1] 4

- Storing results in variables

```
> x <- 2+2  ## R syntax for '=' or assignment
> x^2
```

[1] 16

Simple manipulations; numbers and vectors

- Simple math

```
> 2+2
```

[1] 4

- Storing results in variables

```
> x <- 2+2  ## R syntax for '=' or assignment
> x^2
```

[1] 16

- Vectorized math

```
> weight <- c(110, 180, 240)  ## vector of 3 weights
> height <- c(5.5, 6.1, 6.2)  ## vector of 3 heights
> bmi <- (weight*4.88)/height^2  ## element-wise op
```

Sequences

- : operator

```
> -5:5
```

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

Sequences

- : operator

```
> -5:5
```

[1] -5 -4 -3 -2 -1 0 1 2 3 4 5

- function *seq()*

```
> seq(-1, 1, by=.5)
```

[1] -1.0 -0.5 0.0 0.5 1.0

Sequences

- : operator

```
> -5:5
```

[1] -5 -4 -3 -2 -1 0 1 2 3 4 5

- function *seq()*

```
> seq(-1, 1, by=.5)
```

[1] -1.0 -0.5 0.0 0.5 1.0

- function *rep()*

```
> rep(c(-1, 1), times=2)
```

[1] -1 1 -1 1

Logical Vectors and Missing Values

- conditions

```
> heavy <- weight > 200
```

[1] FALSE FALSE TRUE

Logical Vectors and Missing Values

- conditions

```
> heavy <- weight > 200
```

[1] FALSE FALSE TRUE

- NA : “not available”

```
> z <- c(1:3,NA)
```

[1] 1 2 3 NA

Logical Vectors and Missing Values

- conditions

```
> heavy <- weight > 200
```

[1] FALSE FALSE TRUE

- NA : “not available”

```
> z <- c(1:3,NA)
```

[1] 1 2 3 NA

- function *is.na(x)*

```
> ind <- is.na(z)
```

[1] FALSE FALSE FALSE TRUE

Character vectors

Character strings are entered using either matching double (") or single (') quotes, but are printed using double quotes

```
> labs <- paste(c('X', 'Y'), 1:3, sep="-")
```

[1] "X-1" "Y-2" "X-3"

Note that recycling of short lists takes place...

Index Vectors

```
> x[1:10]
```

selects the first 10 elements of x (assuming length(x) is not less than 10)

Index Vectors

```
> x[1:10]
```

selects the first 10 elements of x (assuming length(x) is not less than 10)

```
> x[-(1:5)]
```

gives all but the first five elements of x

Index Vectors

```
> x[1:10]
```

selects the first 10 elements of x (assuming length(x) is not less than 10)

```
> x[-(1:5)]
```

gives all but the first five elements of x

Objects can have a names attribute to identify its components

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> fruit[c("apple","orange")]
```

apple orange 1 5

Factors, or categorical variables

Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.

- Create a factor corresponding to weight, with three equally spaced levels:

```
> weight.factor <- cut(weight, 3)
> summary(weight.factor)

## (115,131] (131,148] (148,164]
##         6         5         4
```

Factors, or categorical variables

Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.

- Create a factor corresponding to weight, with three equally spaced levels:

```
> weight.factor <- cut(weight, 3)
> summary(weight.factor)

## (115,131] (131,148] (148,164]
##         6         5         4
```

- The *class* and *levels* of a factor

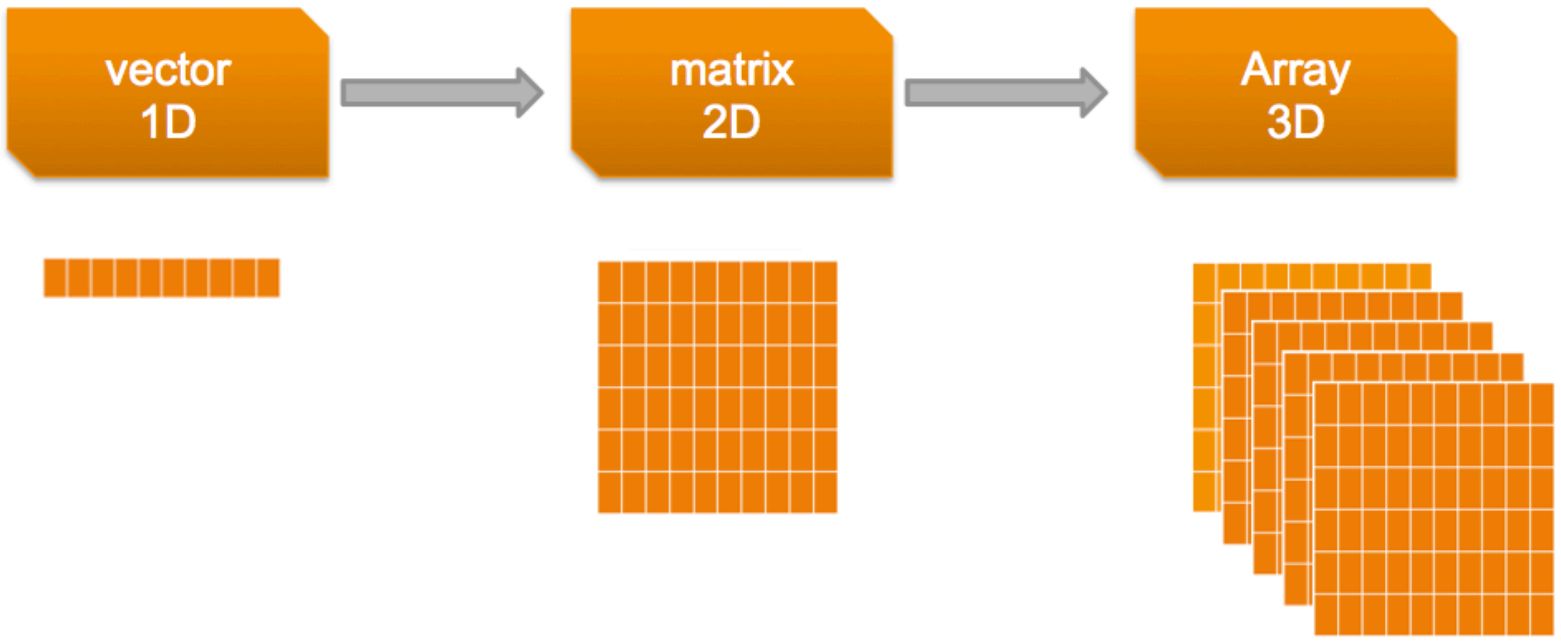
```
> class(weight.factor)
```

[1] "factor"

```
> levels(weight.factor)
```

[1] "(115,131]" "(131,148]" "(148,164]"

Arrays and matrices



Arrays and matrices

- The *matrix()* function

```
> M <- matrix(1:9,3,3, byrow = T)

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

Arrays and matrices

- The *matrix()* function

```
> M <- matrix(1:9,3,3, byrow = T)

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

- Negative subscripts remove elements:

```
> M[-1,-2]

##      [,1] [,2]
## [1,]    4    6
## [2,]    7    9
```

Arrays and matrices

Matrix Facilities

- Row and column name assignments:

```
> colnames(M) <- c("C1", "C2")
> rownames(M) <- c("R1", "R2")

##    C1 C2
## R1  4  6
## R2  7  9
```

Arrays and matrices

Matrix Facilities

- Row and column name assignments:

```
> colnames(M) <- c("C1", "C2")
> rownames(M) <- c("R1", "R2")

##    C1 C2
## R1  4  6
## R2  7  9
```

- Subsetting with names:

```
> M[,"C1"]

## R1 R2
##  4  7
```

Arrays and matrices

Matrix Facilities

- Matrix dimensions:

```
> nrow(A)
> ncol(A)
```

Arrays and matrices

Matrix Facilities

- Matrix dimensions:

```
> nrow(A)
> ncol(A)
```

- Matrix product:

```
> A %*% B
```

Arrays and matrices

Matrix Facilities

- Matrix dimensions:

```
> nrow(A)
> ncol(A)
```

- Matrix product:

```
> A %*% B
```

- Matrix transpose:

```
> t(A)
```

Arrays and matrices

Matrix Facilities

- Matrix dimensions:

```
> nrow(A)
> ncol(A)
```

- Matrix product:

```
> A %*% B
```

- Matrix transpose:

```
> t(A)
```

- Matrix inverse:

```
> solve(A)
```

Arrays and matrices

Matrix Facilities

- *cbind()*: forms matrices by binding together vectors horizontally, or column-wise

```
> X1 <- c(1, 2)
> X2 <- c(3, 4)
> X <- cbind(0, X1, X2)

##         X1 X2
## [1,] 0  1  3
## [2,] 0  2  4
```

Arrays and matrices

Matrix Facilities

- *cbind()*: forms matrices by binding together vectors horizontally, or column-wise

```
> X1 <- c(1, 2)
> X2 <- c(3, 4)
> X <- cbind(0, X1, X2)

##         X1 X2
## [1,] 0  1  3
## [2,] 0  2  4
```

- *rbind()*: corresponding operation for rows

Lists

An R *list* is an object consisting of an ordered collection of objects of possibly different types

```
> Lst <- list(name="Fred", wife="Mary",
+             no.children=3,
+             child.ages=c(4,7,9))
```

Lists

An R *list* is an object consisting of an ordered collection of objects of possibly different types

```
> Lst <- list(name="Fred", wife="Mary",
+             no.children=3,
+             child.ages=c(4,7,9))
```

- Accessing components:

```
> Lst$name == Lst[[1]]
```

[1] TRUE

```
> Lst$child.ages[1] == Lst[[4]][1]
```

[1] TRUE

Data Frames

A *data frame* is a list where components are vectors, factors, lists, or other data frames

```
> char <- letters[1:5]
> m <- 1:5
> log <- c(TRUE,FALSE,TRUE,FALSE,TRUE)
> df <- data.frame(char, m, log)

##   char m   log
## 1    a 1  TRUE
## 2    b 2 FALSE
## 3    c 3  TRUE
## 4    d 4 FALSE
## 5    e 5  TRUE
```

Data Frames

A *data frame* is a list where components are vectors, factors, lists, or other data frames

```
> char <- letters[1:5]
> m <- 1:5
> log <- c(TRUE,FALSE,TRUE,FALSE,TRUE)
> df <- data.frame(char, m, log)

##   char m   log
## 1    a 1  TRUE
## 2    b 2 FALSE
## 3    c 3  TRUE
## 4    d 4 FALSE
## 5    e 5  TRUE
```

A data frame may for many purposes be regarded as a matrix

Getting Data In/Out

- from Files

```
> x <- read.csv("Diamonds.csv")
> x <- read.table("Diamonds.txt", header=T, sep=",")
> x <- read.xlsx(Diamonds.xlsx, 2) # read the second sh
```

Getting Data In/Out

- from Files

```
> x <- read.csv("Diamonds.csv")
> x <- read.table("Diamonds.txt", header=T, sep=",")
> x <- read.xlsx(Diamonds.xlsx, 2) # read the second sh
```

- from Databases

```
> library(RODBC)
> con <- odbcConnect(db.dsn, uid="", pwd="")
> x <- sqlQuery(con, "SELECT * FROM Diamonds")
> close(con)
```

Getting Data In/Out

- from Files

```
> x <- read.csv("Diamonds.csv")
> x <- read.table("Diamonds.txt", header=T, sep=",")
> x <- read.xlsx(Diamonds.xlsx, 2) # read the second sh
```

- from Databases

```
> library(RODBC)
> con <- odbcConnect(db.dsn, uid="", pwd="")
> x <- sqlQuery(con, "SELECT * FROM Diamonds")
> close(con)
```

- from R objects

```
> load("Diamonds.RData")
```

Getting Data In/Out

- from Files

```
> x <- read.csv("Diamonds.csv")
> x <- read.table("Diamonds.txt", header=T, sep=",")
> x <- read.xlsx(Diamonds.xlsx, 2) # read the second sh
```

- from Databases

```
> library(RODBC)
> con <- odbcConnect(db.dsn, uid="", pwd="")
> x <- sqlQuery(con, "SELECT * FROM Diamonds")
> close(con)
```

- from R objects

```
> load("Diamonds.RData")
```

- from the Web

```
> con <- url("http://...")
> x <- read.csv(con)
```

Control Flow Basics

- *for* loops, *repeat* and *while*

```
> for (i in 1:nrow(x)) {
+   print(x[i,])
+ }
```

Warning: for() loops are used in R code much less often than in compiled languages. Often *apply*-type functions preferred.

Control Flow Basics

- *for* loops, *repeat* and *while*

```
> for (i in 1:nrow(x)) {
+   print(x[i,])
+ }
```

Warning: for() loops are used in R code much less often than in compiled languages. Often *apply*-type functions preferred.

- *if* statements

```
> if (expr_1) expr_2 else expr_3
> ifelse(condition, a, b)
```

operators && and || are often used as part of the condition

Control Flow Basics

- *for* loops, *repeat* and *while*

```
> for (i in 1:nrow(x)) {
+   print(x[i,])
+ }
```

Warning: for() loops are used in R code much less often than in compiled languages. Often *apply*-type functions preferred.

- *if* statements

```
> if (expr_1) expr_2 else expr_3
> ifelse(condition, a, b)
```

operators && and || are often used as part of the condition

- *break* and *next* statements available

Functions

A function is defined by an assignment of the form

```
> name <- function(arg_1,arg_2, ...) {
+   expression(s)
+ }
```

Functions

A function is defined by an assignment of the form

```
> name <- function(arg_1,arg_2, ...) {
+   expression(s)
+ }
```

if there is a function fun1 defined by

```
> fun1 <- function(data, data.frame, graph, limit) {...}
```

then the function may be invoked in several ways, for example

```
> ans <- fun1(d, df, TRUE, 20)
> ans <- fun1(d, df, graph=TRUE, limit=20)
> ans <- fun1(data=d, limit=20, graph=TRUE,
+             data.frame=df)
```

Navigating within the R Environment

- Listing all variables

```
> ls()
```

Navigating within the R Environment

- Listing all variables

```
> ls()
```

- Examining a variable x

```
> str(x)
> class(x)
> typeof(x)
> attributes(x)
> head(x)
> tail(x)
```

Navigating within the R Environment

- Listing all variables

```
> ls()
```

- Examining a variable x

```
> str(x)
> class(x)
> typeof(x)
> attributes(x)
> head(x)
> tail(x)
```

- Removing variables

```
> rm(x)
> rm(list=ls())
```

Navigating within the R Environment

- Listing all variables

```
> ls()
```

- Examining a variable x

```
> str(x)
> class(x)
> typeof(x)
> attributes(x)
> head(x)
> tail(x)
```

- Removing variables

```
> rm(x)
> rm(list=ls())
```

- Get or set working directory

```
> getwd()
> setwd(dir)
```

Data Processing

Data Subsetting

- *which*: used to identify values in a vector or array that satisfy a list of criteria
  e.g. which car in *mtcars* has the highest horse power:

```
> which(mtcars$hp == max(mtcars$hp))
```

[1] 31

Data Processing

Data Subsetting

- *which*: used to identify values in a vector or array that satisfy a list of criteria
  e.g. which car in *mtcars* has the highest horse power:

```
> which(mtcars$hp == max(mtcars$hp))
```

[1] 31

- *subset*: allows us to do the same thing, but different notation

```
> subset(mtcars, mtcars$mpg > 20 & mtcars$cyl == 4,
+         select = c("mpg","cyl","disp"))
```

Data Processing

Data Aggregation

- *table*:build a contingency table of the counts at each combination of factor levels.

```
>
table(mtcars[,"gear"])

##
##  3  4  5
## 15 12  5
```

```
> table(mtcars[,c("gear","cyl")])

##      cyl
## gear  4  6  8
##    3  1  2 12
##    4  8  4  0
##    5  2  1  2
```

Data Processing

Data Aggregation

- *table*:build a contingency table of the counts at each combination of factor levels.

```
>
table(mtcars[,"gear"])

##
##  3  4  5
## 15 12  5
```

```
> table(mtcars[,c("gear","cyl")])

##      cyl
## gear  4  6  8
##    3  1  2 12
##    4  8  4  0
##    5  2  1  2
```

- *xtabs* allows you to do the same thing, but using a formula

```
> xtabs(~gear+cyl,mtcars)
```

Data Processing

Data Aggregation

- *aggregate*: Splits the data into subsets, computes summary statistics for each (like a sql group by)

```
> aggregate(hp~cyl+gear, data=mtcars, mean)

##   cyl gear       hp
## 1   4    3  97.0000
## 2   6    3 107.5000
## 3   8    3 194.1667
## 4   4    4  76.0000
## 5   6    4 116.5000
## 6   4    5 102.0000
## 7   6    5 175.0000
## 8   8    5 299.5000
```

Data Processing

Data Aggregation

- *reshape*: from repeated measurements in separate records to repeated measurements in separate columns

```
> df.wide <- reshape(movie.data, idvar="user.id",
+                    timevar="item.id",
+                    direction="wide")
```

user.id	item.id	rating
196	242	3
196	302	1
186	302	3
...	...	...

row.names	user.id	rating.242	rating.302
1	196	3	1
2	186	NA	3
...	...	...	...

Data Processing

Merging

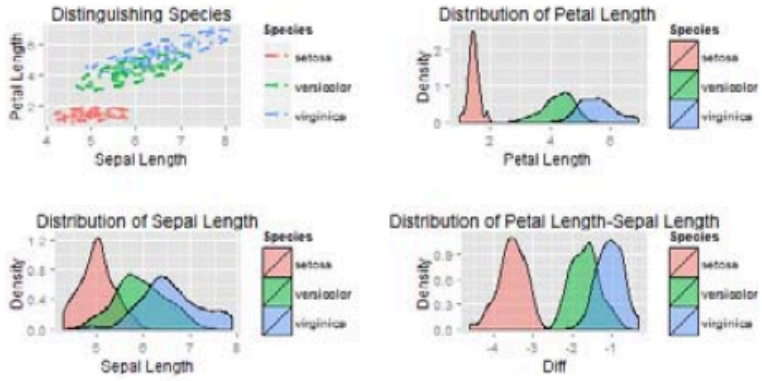
merge: equivalent to database join operations to combine data sets

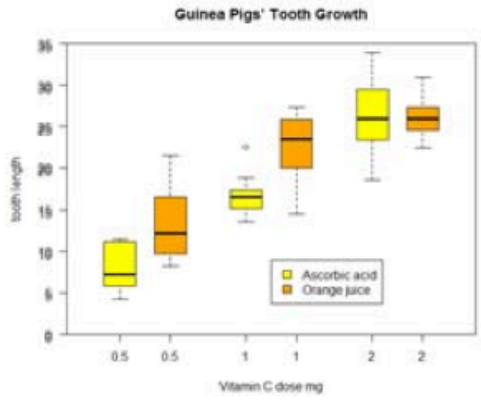
```
> df.all <- merge(df.1, df.2, by = "id", all = TRUE)
```

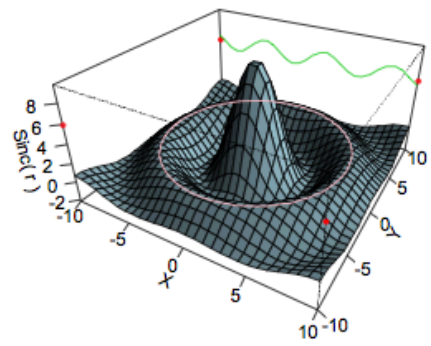
- all = FALSE: gives a *natural* (inner) join
- all.x = TRUE: gives a left (outer) join
- all.y = TRUE: gives a right (outer) join
- all = TRUE: gives a (full) outer join

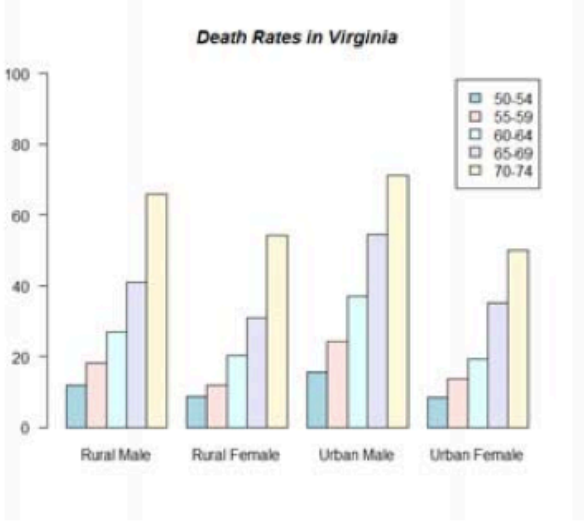
Graphical Procedures

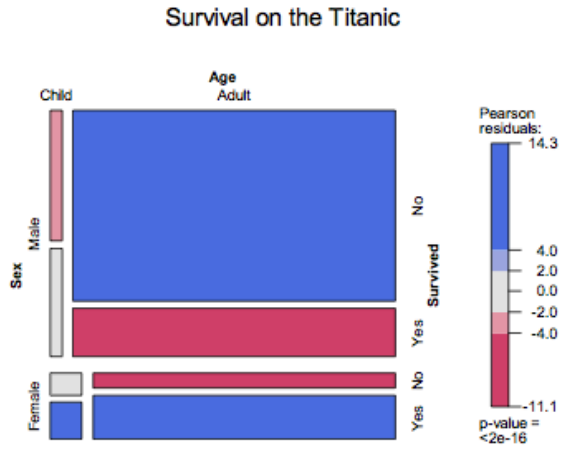
R does graphs

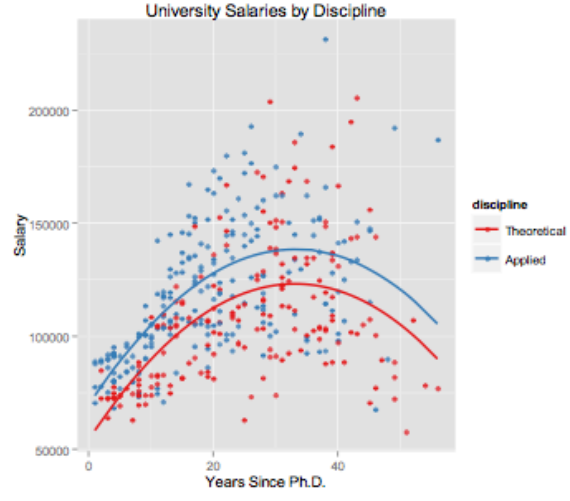












[0] From *Practical Data Visualization with R* by Rob Kabacoff

Graphical Procedures

Basic Graphs

plot() is a generic function: the type of plot produced is dependent on the type or class of the first argument

```
> plot(CaratWeight, Price,
+      main="",
+      xlab="Carat Weight",
+      ylab="Price ", pch=19)
```

Graphical Procedures

Basic Graphs

plot() is a generic function: the type of plot produced is dependent on the type or class of the first argument

```
> plot(CaratWeight, Price,
+      main="",
+      xlab="Carat Weight",
+      ylab="Price ", pch=19)
```

```
> plot(Color)
```

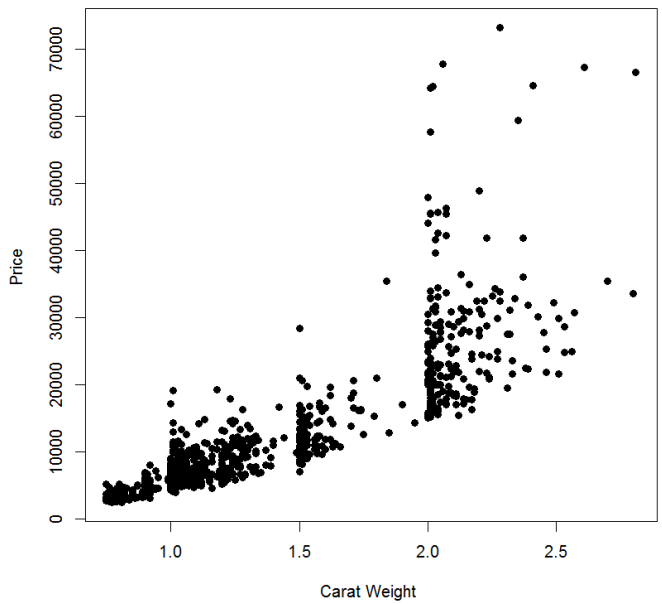
generates a bar plot of diamond's *Color*

Graphical Procedures

Basic Graphs

plot() is a generic function: the type of plot produced is dependent on the type or class of the first argument

```
> plot(CaratWeight, Price,
+      main="",
+      xlab="Carat Weight",
+      ylab="Price ", pch=19)
```




```
> plot(Color)
```

generates a bar plot of diamond's *Color*

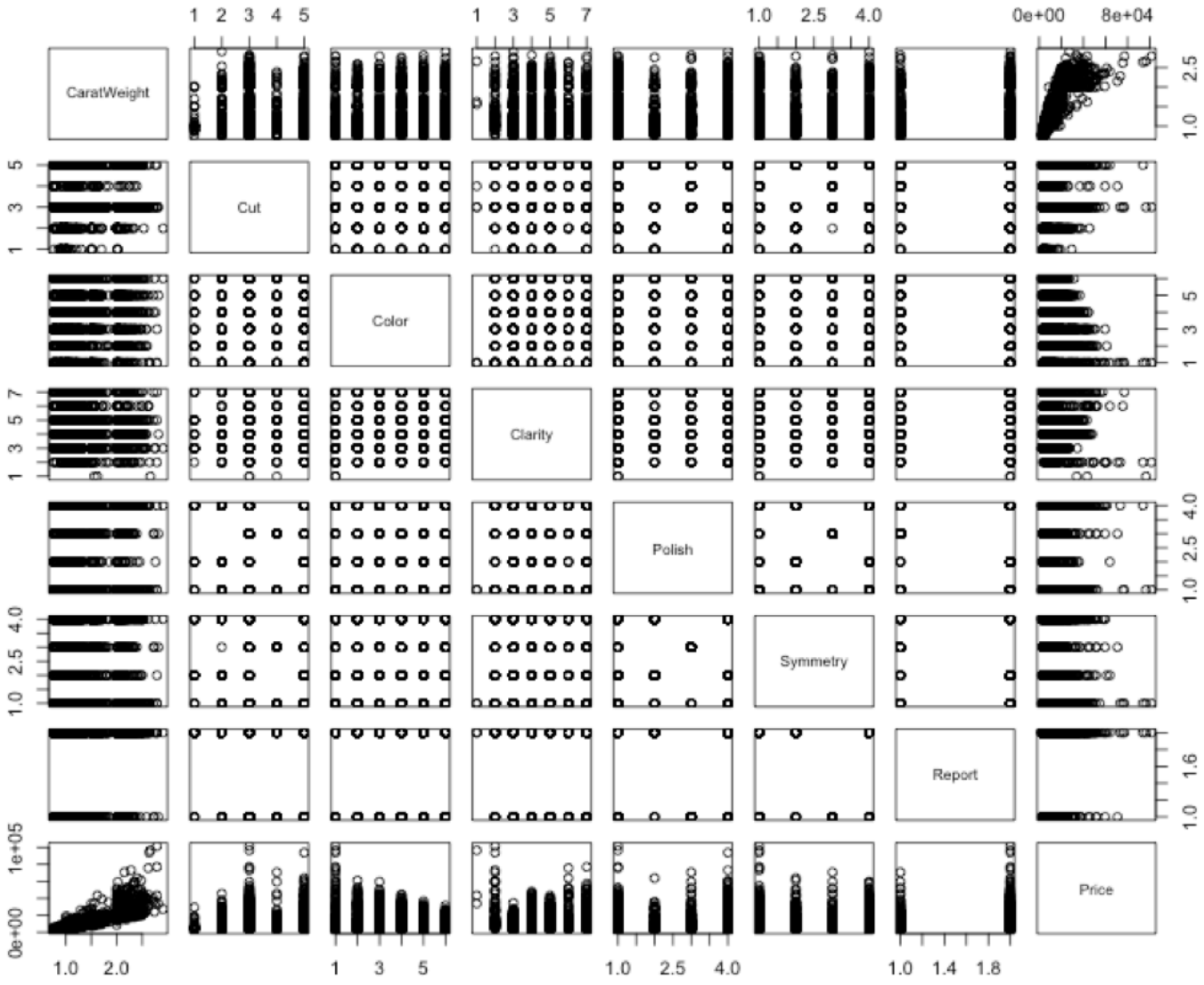
```
> plot(Color, Price)
```

produces boxplots of *Price* for each level of *Color*

Graphical Procedures

Scatter Plot Matrix

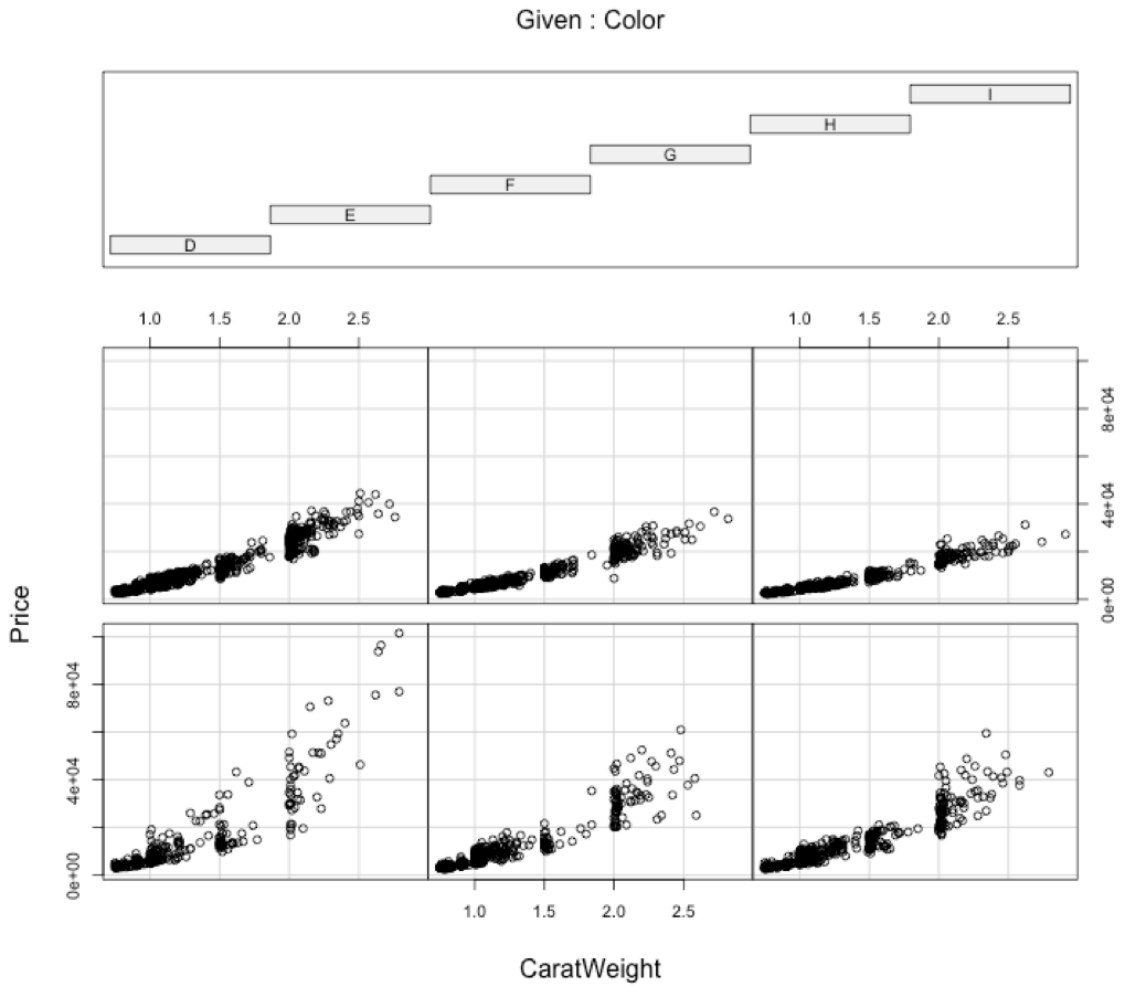
```
> pairs(x[,2:9])
```



Graphical Procedures

Conditioning Plots

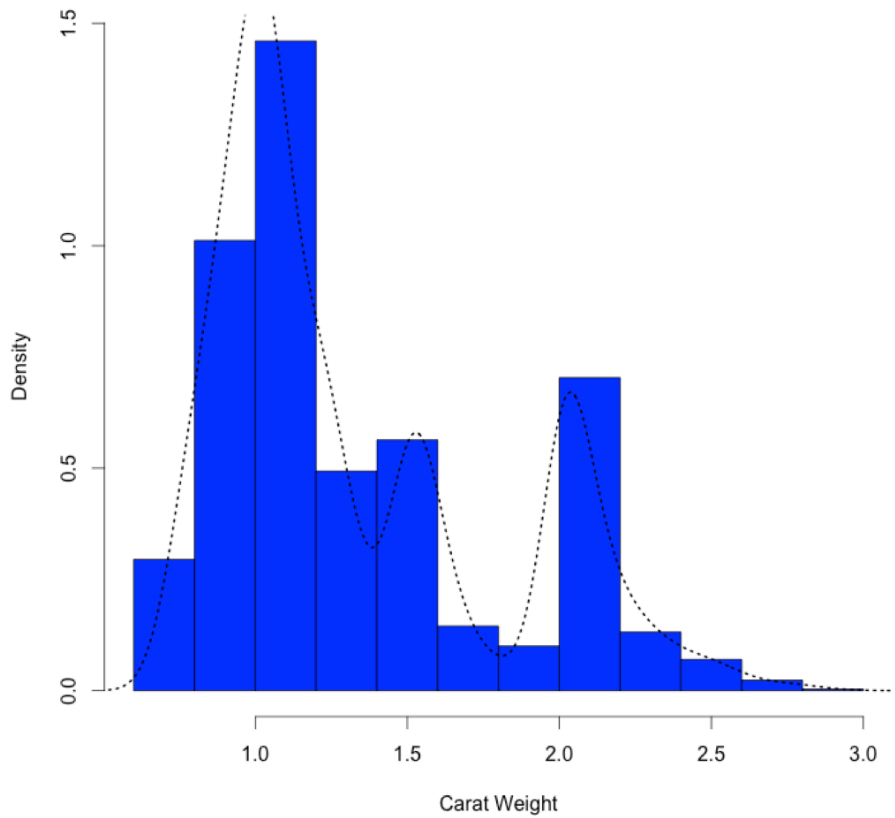
```
> coplot(Price ~ CaratWeight | Color)
```



Graphical Procedures

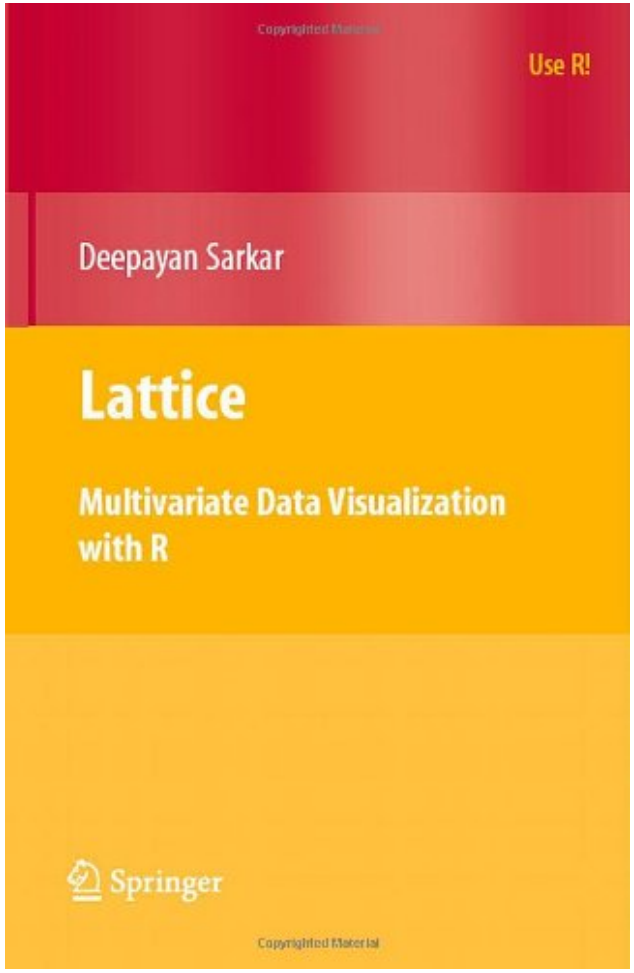
Histograms and Density Plots

```
> hist(x$CaratWeight,breaks=12, col="blue", main="",
+      xlab="Carat Weight", prob=T)
> lines(density(x$CaratWeight), lty="dotted", lwd=2)
```



Graphical Procedures

Lattice vs ggplot



Graphical Procedures

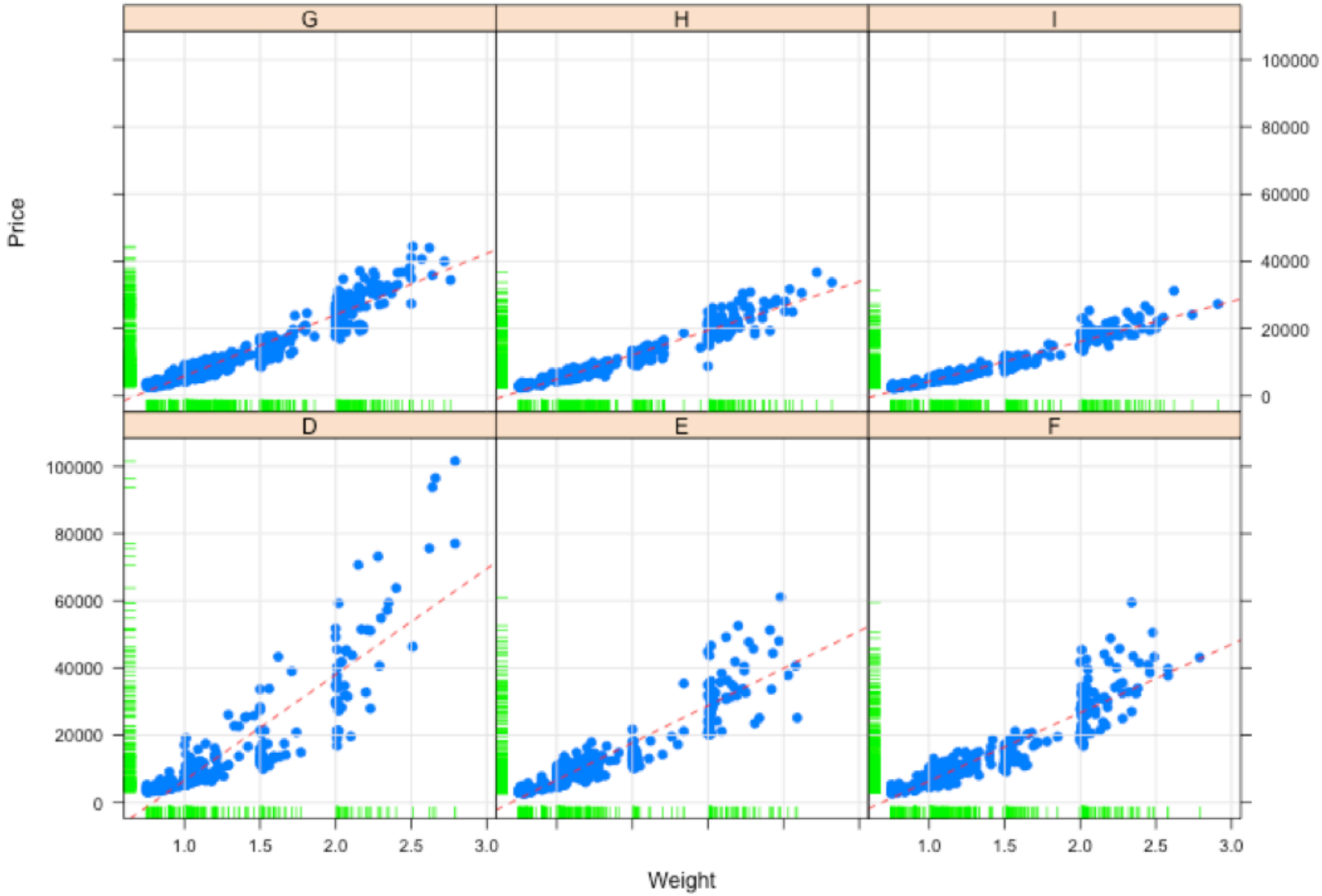
Customizing Lattice Graphs

```
> library(lattice)
> xyplot(Price ~ CaratWeight | Color, data = x,
+        layout=c(3,3),
+        xlab="Weight", ylab="Price",
+        panel=mypanel)
```

```
> mypanel <- function(x, y) {
+   panel.xyplot(x, y, pch=19)
+   panel.rug(x, y, col="green")
+   panel.grid(h=-1, v=-1)
+   panel.lmline(x, y, col="red", lwd=1, lty=2)
+ }
```

Graphical Procedures

Customizing Lattice Graphs



Graphical Procedures

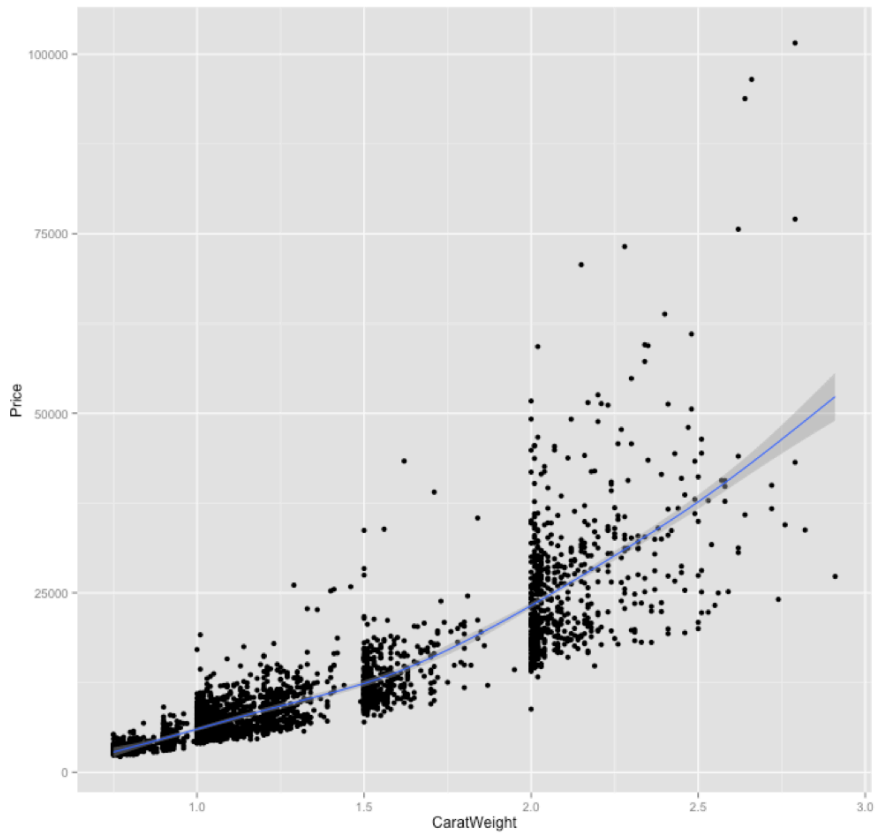
Lattice Graphs

	Graph.type	Function	Formula.examples
1	3D contour plot	contourplot()	z~x*y
2	3D level plot	levelplot()	z~y*x
3	3D scatter plot	cloud()	z~x*y|A
4	3D wireframe graph	wireframe()	z~y*x
5	Bar chart	barchart()	x~A or A~x
6	Box plot	bwplot()	x~A or A~x
7	Dot plot	dotplot()	~x|A
8	Histogram	histogram()	~x
9	Kernel density plot	densityplot()	~x|A*B
10	Parallel coordinates plot	parallel()	dataframe
11	Scatter plot	xyplot()	y~x|A
12	Scatter plot matrix	splom()	dataframe
13	Strip plots	stripplot()	A~x or x~A

Graphical Procedures

GGplot scatter plot with loess fit

```
> library(ggplot2)
> qplot(Price, CaratWeight, data = x,
+       geom = c("point", "smooth"))
```



Statistical Models

Fit a Linear Model

```
> diamonds.lm <- lm(Price ~., data = diamonds.df)
```

- examine it

```
> summary(diamonds.lm)
```

Statistical Models

Fit a Linear Model

```
> diamonds.lm <- lm(Price ~., data = diamonds.df)
```

- examine it

```
> summary(diamonds.lm)
```

- plot it

```
> plot(diamonds.lm)
> plot(density(resid(diamonds.lm)))
```

Statistical Models

Fit a Linear Model

```
> diamonds.lm <- lm(Price ~., data = diamonds.df)
```

- examine it

```
> summary(diamonds.lm)
```

- plot it

```
> plot(diamonds.lm)
> plot(density(resid(diamonds.lm)))
```

- add interactions

```
> diamonds.lm <- lm(Price ~ CaratWeight + Cut*Color,
+                   data = diamonds.df)
```

Statistical Models

Fit a Linear Model

```
> diamonds.lm <- lm(Price ~., data = diamonds.df)
```

- examine it

```
> summary(diamonds.lm)
```

- plot it

```
> plot(diamonds.lm)
> plot(density(resid(diamonds.lm)))
```

- add interactions

```
> diamonds.lm <- lm(Price ~ CaratWeight + Cut*Color,
+                   data = diamonds.df)
```

- predict

```
> y.hat <- predict(diamonds.lm, newdata)
```

Statistical Models

Fit a Logistic Regression

- need a binary outcome variable

```
> diamonds.df$Price01 <- diamonds.df$Price > 10000
> summary(diamonds.df$Price01)

##    Mode   FALSE    TRUE
## logical    1837    1163
```

Statistical Models

Fit a Logistic Regression

- need a binary outcome variable

```
> diamonds.df$Price01 <- diamonds.df$Price > 10000
> summary(diamonds.df$Price01)

##    Mode   FALSE    TRUE
## logical    1837    1163
```

- fit model... exclude original *Price*

```
> diamonds.glm <- glm(Price01 ~., data =
+                       subset(diamonds.df,
+                              select = -c(Price, ID)))
```

Statistical Models

Fit a Decision Tree

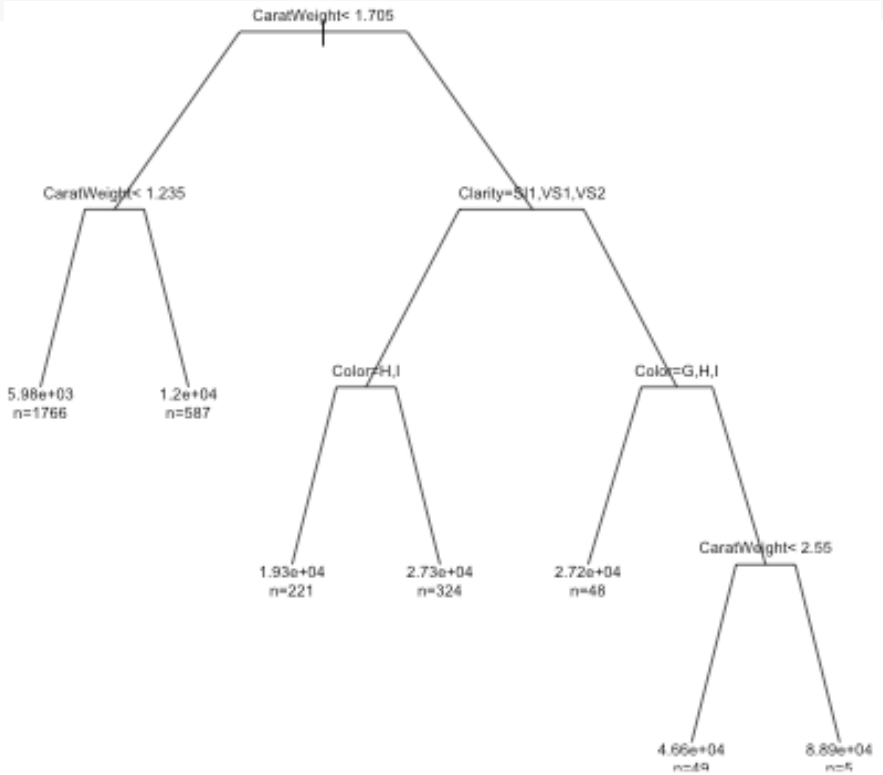
```
> library(rpart)
> set.seed(123)
> diamonds.rpa <- rpart(Price ~ .,
+                       method = "anova",
+                       data = diamonds.df[,-1],
+                       cp = 0,
+                       minsplit=2,
+                       minbucket=1,
+                       maxdepth=7)
```

- *cp* pre-pruning parameter: Any split that does not decrease the overall lack of fit by a factor of cp is discarded

Statistical Models

Fit a Decision Tree

```
> diamond.rpa1 <- prune(diamonds.rpa, cp = 0.02)
> plot(diamond.rpa1, branch = 0.4, uniform = T,
+      compress = T)
> text(diamond.rpa1, use.n = T, digits=3, cex = 0.5,
+      pretty=0)
```



R and Hadoop

- Install Cloudera ODBC Driver for Hive
- Get Kerberos authentication ticket

```
kinit -V user-admin-name
```

- Within R

```
> library(RODBC)
> ch = odbcConnect("HaaS_Hive_DSN")
> sqlQuery(ch, "select * from db_name.table_name")
```

Getting Help

Function	**Action**
help.start()	General help
?foo	Help on function foo
??foo	Search the help system for instances of the string foo
example("foo")	Examples of function foo
RSiteSearch("foo")	Search for the string foo in online help manuals and archived mailing lists
apropos("foo")	List all available functions with foo in their name
data()	List all available datasets for currently loaded packages
vignette(topic)	Open a vignette for topic or package