Predictive Analytics I - 15008 - MATH 3050 - 001

Associated Term: Fall 2020
Special Approval: Department Approval
Registration Dates: Mar 30, 2020 to Sep 14, 2020
Levels: Undergraduate

Main/Uptown Center Campus
Lecture Schedule Type
Online: Specific Mtg Times Instructional Method
3.000 Credits
View Catalog Entry

Course Description:
This course will introduce you to the R programming language and you will apply statistical theory to build insurance predictive models. We will cover techniques such as linear, multivariate, and logistic regression, generalized linear modeling, decision trees, cluster analysis and ensemble modeling techniques. We will study shrinkage and feature selection methods and model validation techniques.

Scheduled Meeting Times

Type	Time	Days	Where	Date Range	Schedule Type	Instructors
Class	11:15 am - 2:00 pm	F	None INTRNET	Sep 07, 2020 - Dec 15, 2020	Lecture	Dorothy L Andrews

Test Dates: First Quiz – Friday, October 2, 2020
Second Quiz – Friday, November 20, 2020
Mid-Term – Friday, October 23, 2020
Final Exam – Friday, December 18, 2020

1

1

Topic 1: The R Programming Language – Part 1

- ❑ Installing R
- ❑ The 4 Console Windows
- ❑ The R Stats Package
- ❑ The GLM Function
- ❑ Built-in Functions
- ❑ Operators
- ❑ Getting Data into R
- ❑ Basic Plotting in R
- ❑ Assignment

2

2

Educational Objectives :

The course is intended to help you

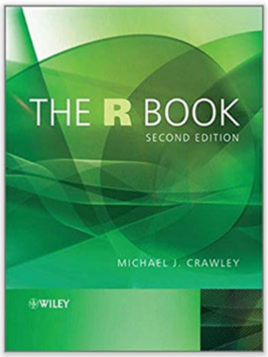
1. Develop a conceptual understanding of statistical techniques relevant to machine learning techniques common to statistical modeling;
2. Apply statistical techniques to real data sets to develop a working understanding of the techniques so that you can assess their relevancy to modeling problems;
3. Perform exploratory data analysis of data to assess modeling data sets for biases and deficiencies which may impact results of machine learning techniques; and
4. Provide you with resources to continue to deepen your statistical acumen as you gain more experience building advanced modeling techniques.

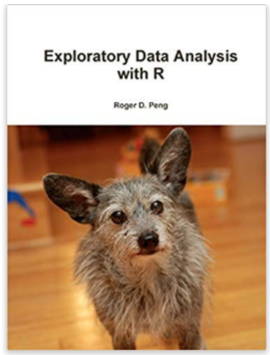
Course Requirements:

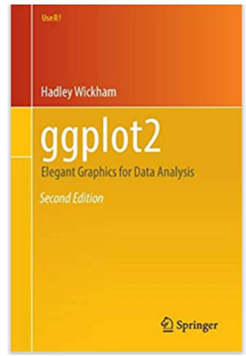
- A laptop with R and R Studio installed that you bring to class. This will facilitate you running code along side the lectures.
- There will be homework assigned after each session and a mechanism for sending it in to be reviewed, graded and commented on. I want to stress the importance of keeping up with it. R is our tool for all the statistics in this training
- Need to set up directory C:\Rdata on your C drive for the class data files. We will use this directory for all class examples to make it easy to share code. Extract the data zip file to this directory. You will be given additional file to store in this directory. Set up a directory C:\Rscripts for all your scripts.
- The software R & R Studio.

Course Textbooks:

The following texts will be provided in pdf format. Additional references will be provided as needed.







5

Objectives of this Lesson:

By the end of this lesson you should be able to:

- Navigate through the different console windows in R Studio.
- Create variables and preform mathematical operations using them.
- Import data files into R manually and through code from directories and the web.
- Subset data using base R functions and data frame and data.table functions
- Write R code programs to read datafiles into R and create exploratory analyses using basic statistical functions, boxplots and other graphical techniques.

6

What is R?

This is an easy question to answer. R is a dialect of S. S is a language that was developed by John Chambers and others at the old Bell Telephone Laboratories, originally part of AT&T Corp.

S was initiated in 1976 as an internal statistical analysis environment—originally implemented as Fortran libraries. Early versions of the language did not even contain functions for statistical modeling.

The primary R system is available from the Comprehensive R Archive Network, also known as CRAN.

R functionality is divided into several *packages*.

- The "base" R system contains, among other things, the base package which is required to run R
- The other packages contained in the "base" system include:
  - utilise, stats, datasets graphics, grDevices
  - grid, methods, tools, parallel, compiler, splines, tcltk, stats4

- There are over 30,000 packages and counting that can be installed in R.

Installing R & R Studio:

To Install R:

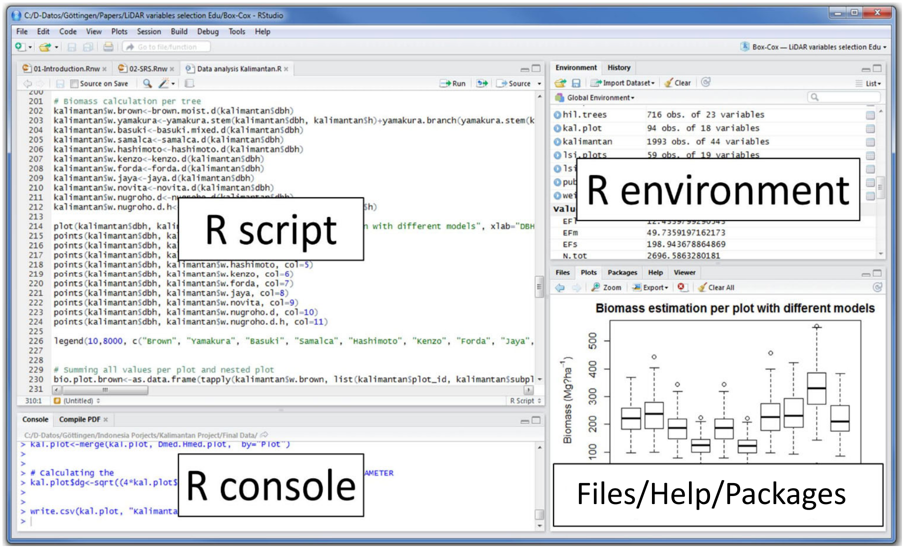
1. Open an internet browser and go to www.r-project.org
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select the CRAN location closet to you and click the corresponding link.
4. Click on the "Download R for Windows" link at the top of the page.
5. Click on the "Install R for the First Time" link at the to of the page.
6. Click "Download R for Windows" and save the executable file on your computer & rn it.

To Install RStudio

1. Go to rstudio.com and click on the "Download" button.
2. Click on "Download RStudio Desktop" – Pick the "Free" version.
3. Click on the version recommended for your system, download the.exe file, and install it.

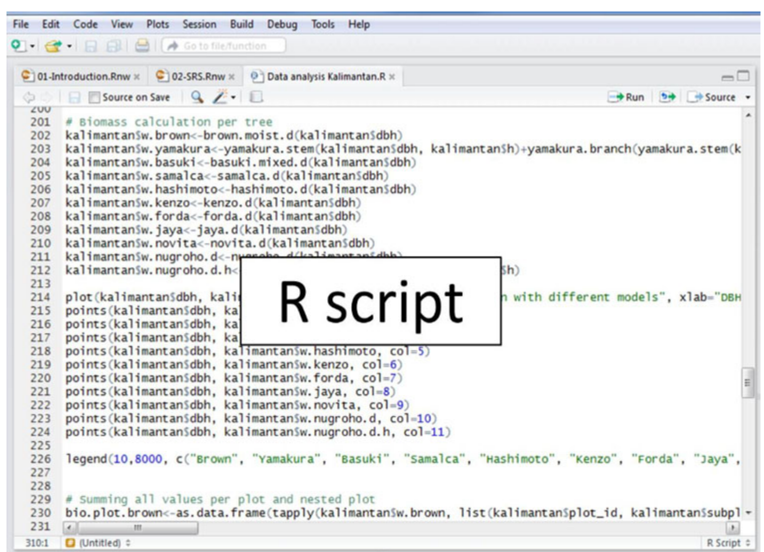
4 Console Windows

1. R Script
2. R Environment
3. R Console
4. Files/Help/Packages



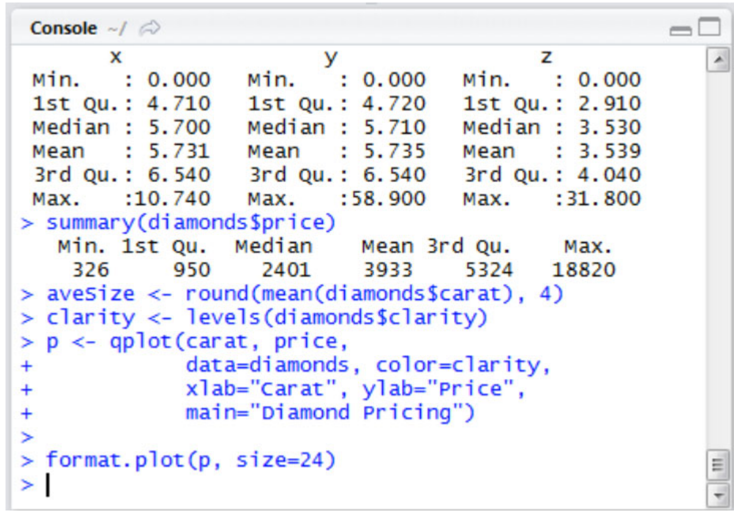
This is where you write the code for your analysis.

Each tab represents a different R *script* file



R console

This is where R prints the output of your code when it's run. You can also write code directly in the console after the > symbol.

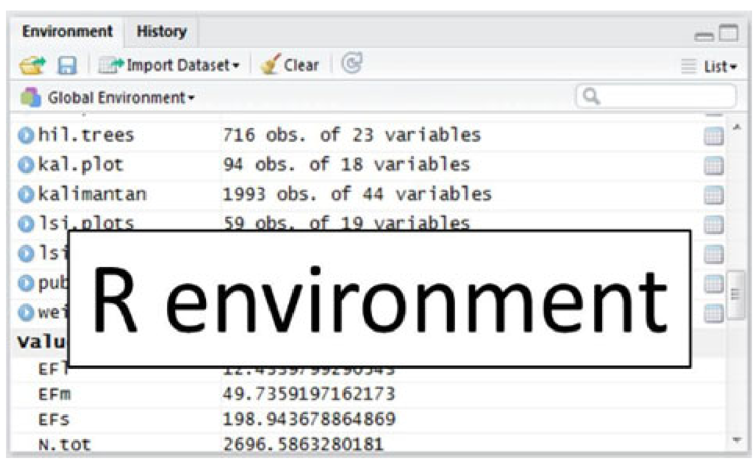


11

Environment/History - This panel generally has the following two tabs:

- **Environment** - Displays all your data, variables, and user-defined functions. These are created by the user either in the code editor or directly in the console.
- **History** - A list of your command history

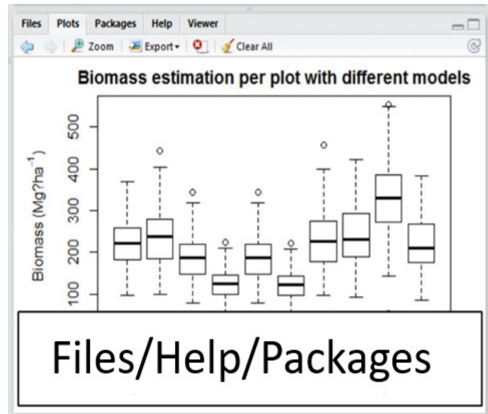
You can also import datasets from other formats using the "Import Dataset" dropdown.



12

•**Files/Plots/Packages/Help/Viewer** - This panel contains numerous helpful panels:

- **Files** - The list of all files contained in your current *working directory*. You can also navigate to different folders on your computer. This is where you can click on different R scripts to open them in the code editor.
- **Plots** - When you produce plots with your code, they will be displayed here
- **Packages** - The list of packages (groups of functions) currently installed on your computer. You can install new packages or update existing packages from this tab by clicking Install or Update.
- **Help -** Where you can search the R documentation to get help using the different R functions. This is a very useful feature of RStudio! You can also get help for a function by typing ? followed by the function name in the console (e.g. ?data.frame()).



13

The Console as a Calculator:

Type the following in the console window. Hit return after each line

```
>4 + 3
>a <-100   # The "<-" is the less than operator and the hypen.
>a
>b<-200
>a + b
>c<-print("Hello World")
>c
>x <- list (c(1, 2), c("Sleep Well"))
>x
>X  #This is an upper-case X
>d<-10:30  #This is the technique of sequencing
>d
```

The "<-" operator is one of the most used functions in R. It is one of a family of assignment operators.

- # Used to denote comments.
- Comments are a best practice in programming.
- R is very case sensitive!

14

Try these at home

Do the following in the console window :

1. Calculate 10 times 3
2. Make a variable x equal to 5
3. Calculate 10 times x
4. Assign (1, 2,3,4) to a and then
   a. take the square root of a
   b. exponentiate a
   c. take the log of a
   d. find the min of a
   e. find the max of a
5. Assign (-1, -2, -3, 4) to b and then
   a. multiply a by b
   b. subtract b from a
   c. divide b by a
   d. divide a by b

Limitations of R

- R is essentially based on almost 50-year-old technology, going back to the original S system developed at Bell Labs. There was originally little built in support for dynamic or 3-D graphics (but things have improved greatly since the “old days”).
- Another commonly cited limitation of R is that objects must generally be stored in physical memory. This means R generally is more of a **memory hog** than other statistical packages.
- At a higher level one “limitation” of R is that its functionality is based on consumer demand and (voluntary) user contributions. If no one feels like implementing your favorite method, then it’s *your* job to implement it (or you need to pay someone to do it).

The R Stats Package

R statistical functions.

Details

Priority	base
License	Part of R 3.6.1
NeedsCompil...	yes
Imports	graphics , grDevices , utils
Suggests	MASS , Matrix , methods , stats4 , SuppDists
Contributors	contributors worldwide, R team

https://www.rdocumentation.org/packages/stats/versions/3.6.1

Item	Name	Description
1	.checkMFClasses	Functions to Check the Type of Variables passed to Model Frames
2	acf	Auto- and Cross- Covariance and -Correlation Function Estimation
3	acf2AR	Compute an AR Process Exactly Fitting an ACF
4	add1	Add or Drop All Possible Single Terms to a Model
5	addmargins	Puts Arbitrary Margins on Multidimensional Tables or Arrays
6	aggregate	Compute Summary Statistics of Data Subsets
7	AIC	Akaike's An Information Criterion
8	alias	Find Aliases (Dependencies) in a Model
9	anova	Anova Tables
10	anova.glm	Analysis of Deviance for Generalized Linear Model Fits
11	anova.lm	ANOVA for Linear Model Fits
12	anova.mlm	Comparisons between Multivariate Linear Models
13	ansari.test	Ansari-Bradley Test
14	aov	Fit an Analysis of Variance Model
15	approxfun	Interpolation Functions
16	ar	Fit Autoregressive Models to Time Series
17	ar.ols	Fit Autoregressive Models to Time Series by OLS
18	arima	ARIMA Modelling of Time Series
19	arima.sim	Simulate from an ARIMA Model
20	arima0	ARIMA Modelling of Time Series -- Preliminary Version
21	ARMAacf	Compute Theoretical ACF for an ARMA Process
22	ARMAtoMA	Convert ARMA Process to Infinite MA Process
23	as.hclust	Convert Objects to Class hclust
24	asOneSidedFormula	Convert to One-Sided Formula
25	ave	Group Averages Over Level Combinations of Factors

```
>#Use the following command to list all the
>#functions in a package
>ls("package:stats")
```

Try it!

17

Search Functions in R

To search for all available linear modeling (lm) functions:

```
>apropos("lm")

 [1] ".colMeans"      ".lm.fit"        "colMeans"       "confint.lm"     "contr.helmert"
 [6] "dummy.coef.lm"  "getAllMethods"  "glm"            "glm.control"    "glm.fit"
[11] "KalmanForecast" "KalmanLike"     "KalmanRun"      "KalmanSmooth"   "kappa.lm"
[16] "lm"             "lm.fit"         "lm.influence"   "lm.wfit"        "model.matrix.lm"
[21] "nlm"            "nlminb"         "predict.glm"    "predict.lm"     "residuals.glm"
[26] "residuals.lm"   "summary.glm"    "summary.lm"
```


18

Examples of Functions in R

To see an example of a function:

>**example**(lm)

```
lm> require(graphics)

lm> ## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
lm> ## Page 9: Plant Weight Data.
lm> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)

lm> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)

lm> group <- gl(2, 10, 20, labels = c("Ctl","Trt"))

lm> weight <- c(ctl, trt)

lm> lm.D9 <- lm(weight ~ group)

lm> lm.D90 <- lm(weight ~ group - 1) # omitting intercept
```


Examples of Functions in R

To see an example of a function:

>**example**(lm)

Note: I cleaned up the code from the above output to aid our discussion.

```
>require(graphics)

>## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
>## Page 9: Plant Weight Data.
>ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
>trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)

>group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
>weight <- c(ctl, trt)

>lm.D9 <- lm(weight ~ group)
>lm.D90 <- lm(weight ~ group - 1) # omitting intercept

>## No test:
>anova(lm.D9)

>summary(lm.D9)
>par(mfrow = c(2, 2))
>plot(lm.D9)      # Residuals, Fitted, ...
```

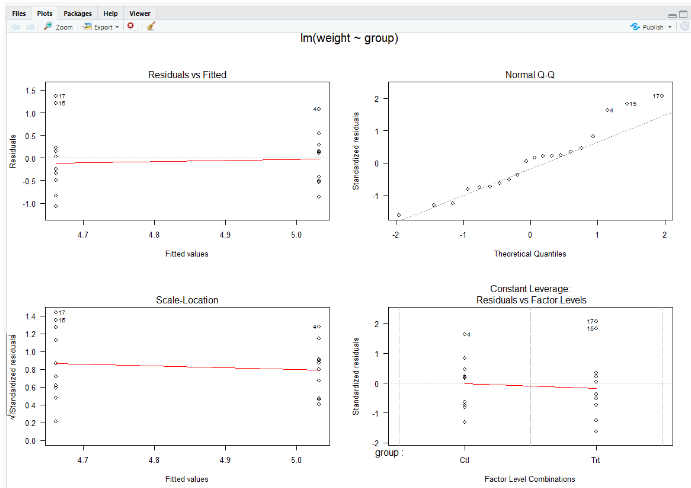
Examples of Functions in R

The code that produces the graph.
>par(mfrow = c(2, 2))
>plot(lm.D9)   # Residuals, Fitted, ...

par(mfrow = c(2, 2)) – defines the organization of the graphs – two rows of graphs and two graphs per row. These parameters can be changed to other values.

plot(lm.D9) – plots the standard graphs available to the plot(lm) function.

We will revisit these functions later.



GLM Function in R

Try the follow command:

>**example**(glm)

GLM Function in R

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)

print(d.AD <- data.frame(treatment, outcome, counts))

glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())

anova(glm.D93)
summary(glm.D93)

## Computing AIC in 3 ways:
A0 <- AIC(glm.D93)            #No. 1
ll <- logLik(glm.D93)
A1 <- -2*c(ll) + 2*attr(ll, "df") #No. 2
A2 <- glm.D93$family$aic(counts, mu=fitted(glm.D93), wt=1) + 2 *
length(coef(glm.D93))         #No. 3
A0
ll
A1
A2
```

Output

```
> counts <- c(18,17,15,20,10,20,25,13,12)
> outcome <- gl(3,1,9)
> treatment <- gl(3,3)

> outcome
[1] 1 2 3 1 2 3 1 2 3
Levels: 1 2 3

> treatment
[1] 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
```


GLM Function in R

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)

print(d.AD <- data.frame(treatment, outcome, counts))

glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
anova(glm.D93)
summary(glm.D93)

## Computing AIC [in many ways]:
A0 <- AIC(glm.D93)
ll <- logLik(glm.D93)
A1 <- -2*c(ll) + 2*attr(ll, "df")
A2 <- glm.D93$family$aic(counts, mu=fitted(glm.D93), wt=1) + 2 *
length(coef(glm.D93))
A0
ll
A1
A2
```

```
  treatment outcome counts
1         1       1     18
2         1       2     17
3         1       3     15
4         2       1     20
5         2       2     10
6         2       3     20
7         3       1     25
8         3       2     13
9         3       3     12
```

GLM Function in R

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)

print(d.AD <- data.frame(treatment, outcome, counts))

glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
anova(glm.D93)
summary(glm.D93)

## Computing AIC [in many ways]:
A0 <- AIC(glm.D93)
ll <- logLik(glm.D93)
A1 <- -2*c(ll) + 2*attr(ll, "df")
A2 <- glm.D93$family$aic(counts, mu=fitted(glm.D93), wt=1) + 2 *
length(coef(glm.D93))
A0
ll
A1
A2
```

```
Analysis of Deviance Table

Model: poisson, link: log

Response: counts

Terms added sequentially (first to last)


          Df Deviance Resid. Df Resid. Dev
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291
treatment  2   0.0000         4     5.1291
```


GLM Function in R

```
>glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
>summary(glm.D93)
```

```
Call:
glm(formula = counts ~ outcome + treatment, family = poisson())

Deviance Residuals:
       1         2         3         4         5         6         7         8         9
-0.67125   0.96272  -0.16965  -0.21999  -0.95552   1.04939   0.84715  -0.09167  -0.96656

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.045e+00  1.709e-01  17.815   <2e-16 ***
outcome2    -4.543e-01  2.022e-01  -2.247   0.0246 *
outcome3    -2.930e-01  1.927e-01  -1.520   0.1285
treatment2   1.338e-15  2.000e-01   0.000   1.0000
treatment3   1.421e-15  2.000e-01   0.000   1.0000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: 56.761

Number of Fisher Scoring iterations: 4
```

GLM Function in R: Deviance

```
## Computing AIC [in many ways]:
>A0 <- AIC(glm.D93)
>ll <- logLik(glm.D93)
>A1 <- -2*c(ll) + 2*attr(ll, "df")
>A2 <- glm.D93$family$aic(counts, mu=fitted(glm.D93), wt=1)
        + 2 * length(coef(glm.D93))
>A0
>ll
>A1
>A2
```

```
> ## Computing AIC [in many ways]:
> A0 <- AIC(glm.D93)
> ll <- logLik(glm.D93)
> A1 <- -2*c(ll) + 2*attr(ll, "df")
> A2 <- glm.D93$family$aic(counts, mu=fitted(glm.D93), wt=1) + 2 * length(coef(glm.D93))
> A0
[1] 56.76132
> ll
'log Lik.' -23.38066 (df=5)
> A1
[1] 56.76132
> A2
[1] 56.76132
```


Built-in Functions in R

Numeric Functions

Function	Description
abs(x)	absolute value
sqrt(x)	square root
ceiling(x)	ceiling(3.475) is 4
floor(x)	floor(3.475) is 3
trunc(x)	trunc(5.99) is 5
round(x, digits=n)	round(3.475, digits=2) is 3.48
signif(x, digits=n)	signif(3.475, digits=2) is 3.5
cos(x), sin(x), tan(x)	also acos(x), cosh(x), acosh(x), etc.
log(x)	natural logarithm
log10(x)	common logarithm
exp(x)	e^x
log(x, n)	$\log_n x$

This is just a sampling of functions.

We will investigate the effect of floor, ceiling, trunc and round in an exercise.

Built-in Functions in R

This is just a sampling of functions.

Character Functions

Function	Description
substr(x, start=n1, stop=n2)	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "222" is "a222ef"
grep(pattern, x , ignore.case=FALSE, fixed=FALSE)	Search for pattern in x. If fixed =FALSE then pattern is a regular expression. If fixed=TRUE, then pattern is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2 sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE) Find pattern in x and replace with replacement text. If fixed=FALSE then pattern is a regular expression. If fixed = T then pattern is a text string. sub("\\s",".","Hello There") returns "Hello.There"

Built-in Functions in R

Character Functions

This is just a sampling of functions.

Function	Description
strsplit(x, split)	Split the elements of character vector x at split. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using sep string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(x)	Uppercase
tolower(x)	Lowercase

Built-in Functions in R

Statistical Functions

Function	Description
mean	average
sd	standard deviation
median	median
range	range of a set of numbers
min	minimum value
max	maximum value
quantile	quantile of distribution
summary	summary statistics

This is just a sampling of functions.

Built-in Functions in R

Other Useful Functions

Function	Description	
seq(from , to, by)	generate a sequence	#indices is c(1, 3, 5, 7, 9)
rep(x, ntimes)	repeat x n times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)	
cut(x, n)	divide continuous variable in factor with n levels y <- cut(x, 5)	

This is just a sampling of functions.

Topic 1: The R Programming Language – Part 1

Operators in R

Important assignment Operators:

<-
-> Assigns objects to variables in the environment in which they are evaluated.

<<-
->> Used in functions and assignments can be made from the parent environment.

Note: "<-" is the most prevalent assignment function

Operator Precedence in R

Operator	Description	Associativity
^	Exponent	Right to Left
-x, +x	Unary minus, Unary plus	Left to Right
%%	Modulus	Left to Right
*, /	Multiplication, Division	Left to Right
+, –	Addition, Subtraction	Left to Right
<, >, <=, >=, ==, !=	Comparisions	Left to Right
!	Logical NOT	Left to Right
&, &&	Logical AND	Left to Right
|, ||	Logical OR	Left to Right
->, ->>	Rightward assignment	Left to Right
<-, <<-	Leftward assignment	Right to Left
=	Leftward assignment	Right to Left

Topic 1: The R Programming Language – Part 1

Operators in R

The Difference Between "=" and "==" in R

"=" : Assignment operator. Works like "<-" operator.

"==" : A logical "**Equal to"** operator that compares if two things are exactly equal.

The Difference Between "==" and "%in%" in R

"==" : Checks the assignment of a variable

"%in%" : Compares an element to all elements in a vector

Examples:

```
ac <- c("a", "b", "c")
ae <- c("a", "b", "c", "d", "e")
ac %in% ae
[1] TRUE TRUE TRUE
ac == ae
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

Operators in R

Using “%in%“ and “which%“ in R

“**%in%**” : Compares an element to all elements in a vector

“**which**”: Returns locations in first named vector of items in second named vector.

Example:

```
stock <- c("car","van")
requests <- c("truck","suv","van","sports","car","waggon","car")
```

```
>which(requests %in% stock)
 [1] 3 5 7
```

```
>requests [which(requests %in% stock)]
 [1] "van" "car" "car"
```


Creating a Script in R

Working with Directories:

- Find the name of your current directory using command: getwd()
- Set your working directory using the command: setwd("c:\\Rscripts")
- Save the path of your working directory with the assignment: mywd <- getwd()
- You can reset your wd later in the program/session with the command: setwd(mywd)
- To see files in a directory use the command: dir(path)
- To view a dataset use the command: view(datasetname)
- To see data sets available to you use data()

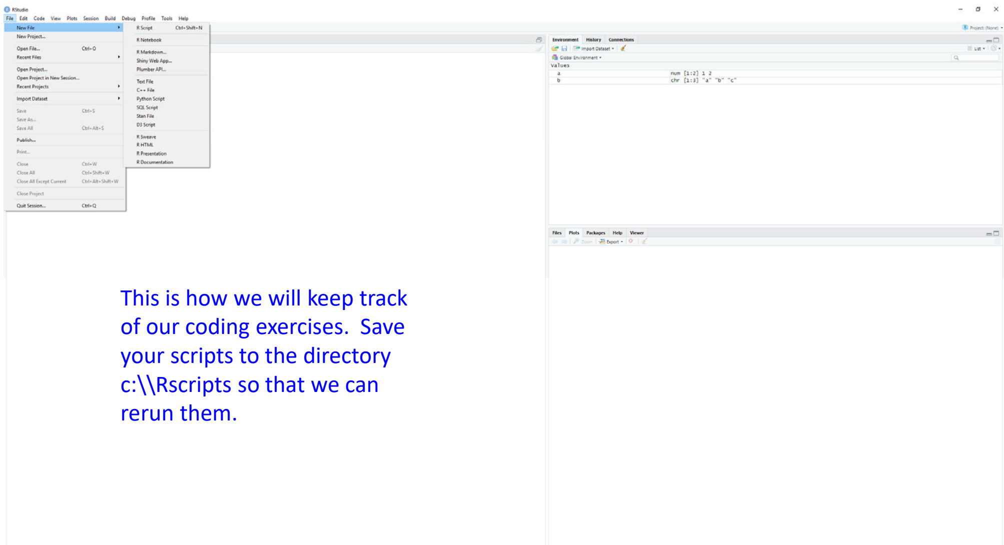
To see the data sets in extra installed packages as well, type:

```
data(package = .packages(all.available = TRUE))
```

You can read the documentation for a particular data set with the usual query:

```
?lynx
```

Creating a Script in R



This is how we will keep track of our coding exercises. Save your scripts to the directory c:\\Rscripts so that we can rerun them.

37

Getting Data Into R

There are several ways to get data into R

1. Use the "<-" operator with the "c" (concatenate function)
2. Read data from an external data file
3. Scanning data from the keyboard
4. Generate data from R code

38

Getting Data Into R

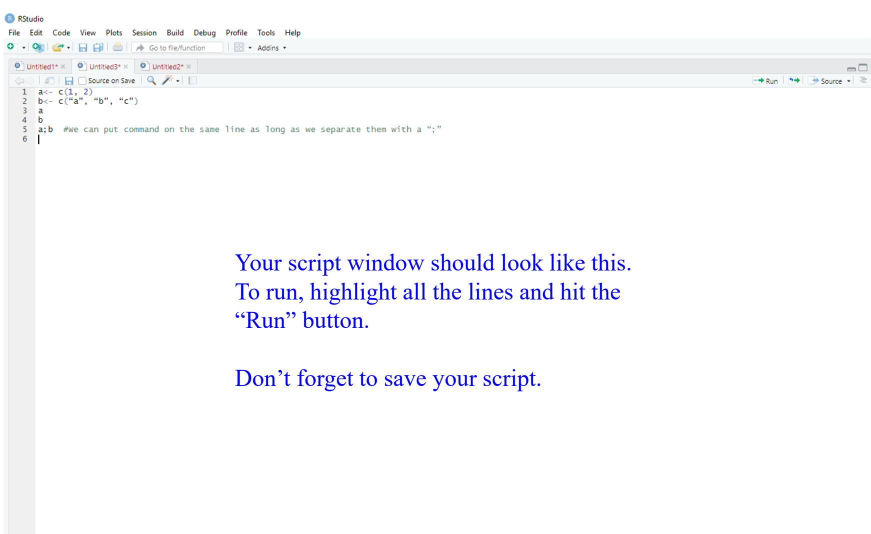
The "**<-**" operator with the "c" (concatenate function)

This operator can assign any object to a variable using the "c()" function or directly.

The c() function is the concatenate function and it work with numeric and character data. Type the following:

```
>a<- c(1, 2)
>b<- c("a", "b", "c")
>a
>b
>a;b  #We can put commands on the same line as long as we separate them with a ";"
```


Getting Data Into R



Your script window should look like this.
To run, highlight all the lines and hit the "Run" button.

Don't forget to save your script.

Getting Data Into R

Example: Write a script to do the following:

1. Create a variable y
2. Store the numbers 1, 2,3,4,5,6,7,8,9,10 in y
3. Calculate the following quantities:

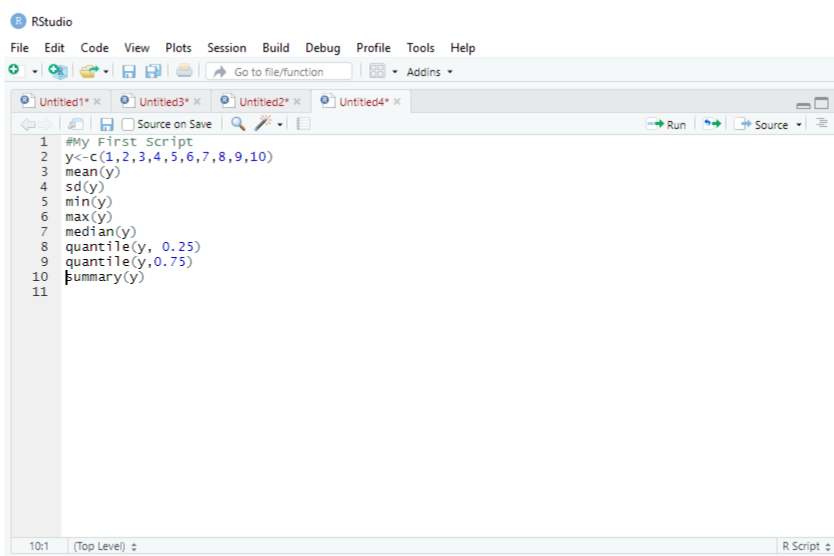
   a) Mean
   b) SD
   c) Min
   d) Max
   e) Median

41

Topic 1: The R Programming Language – Part 1

Getting Data Into R

```
>#My First Script
>y<-c(1,2,3,4,5,6,7,8,9,10)
>mean(y)
>sd(y)
>min(y)
>max(y)
>median(y)
>quantile(y, 0.25)
>quantile(y, 0.75)
>summary(y)
```



42

Homework

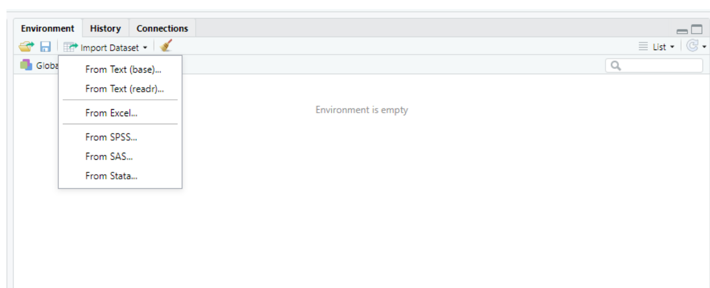
Write a script to do the following:

1. Assign the sequence of numbers 10 through 50 in increments of 5 to the variable named "sequence" and print sequence.

2. Using the variable sequence to perform the following operations on sequence:
   a. square root
   b. Natural logarithm
   c. Base 10 logarithm
   d. Base 2 logarithm

Getting Data Into R

We can manually import a file. Let's import the "worms.txt" file. Find it on your hard drive.



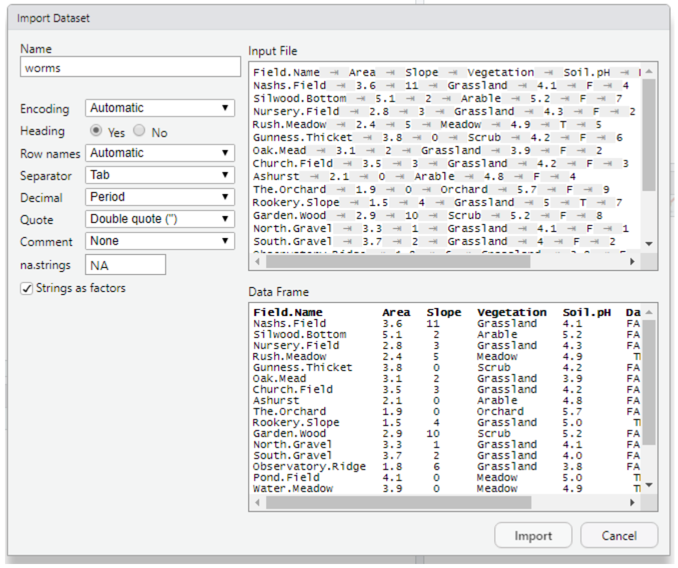
Getting Data Into R

The import window gives you a lot of information about the file.

Make sure you select "Heading" since the fieldnames are in the first row.

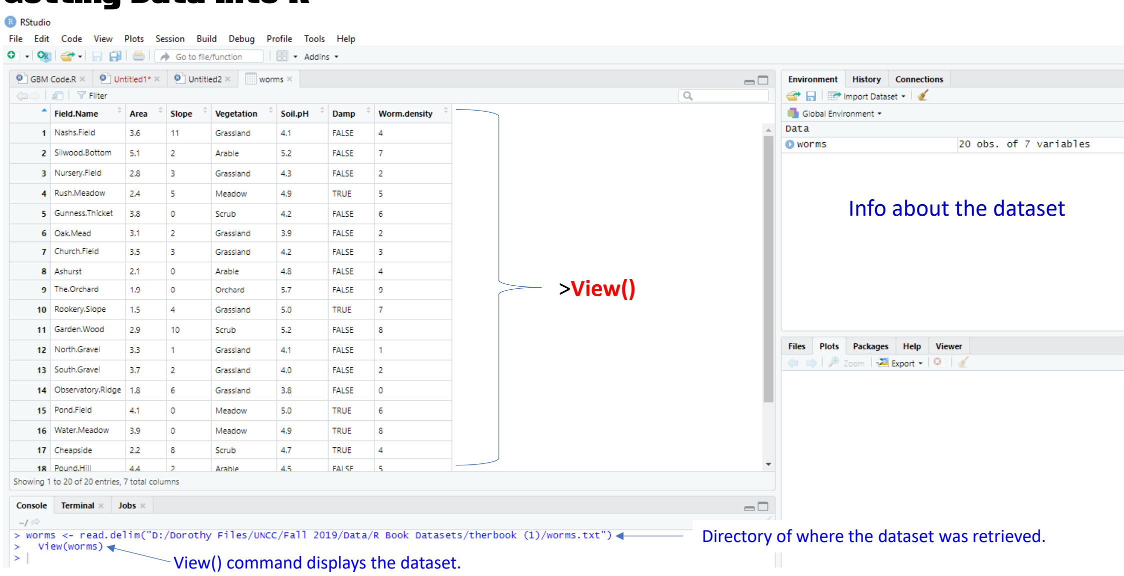
Inspect the data with scroll bars.

Click "Import" to import.



45

Getting Data Into R



Info about the dataset

>View()

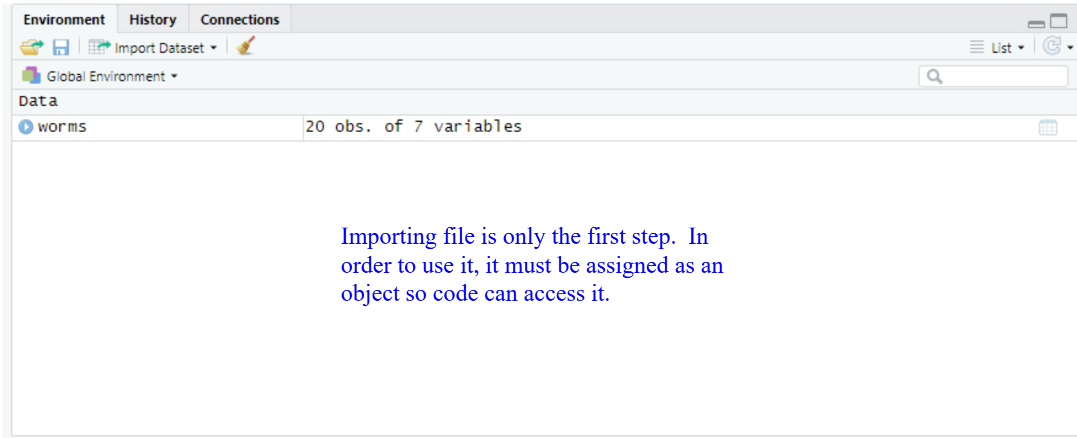
Directory of where the dataset was retrieved.

View() command displays the dataset.

46

Topic 1: The R Programming Language – Part 1

Getting Data Into R



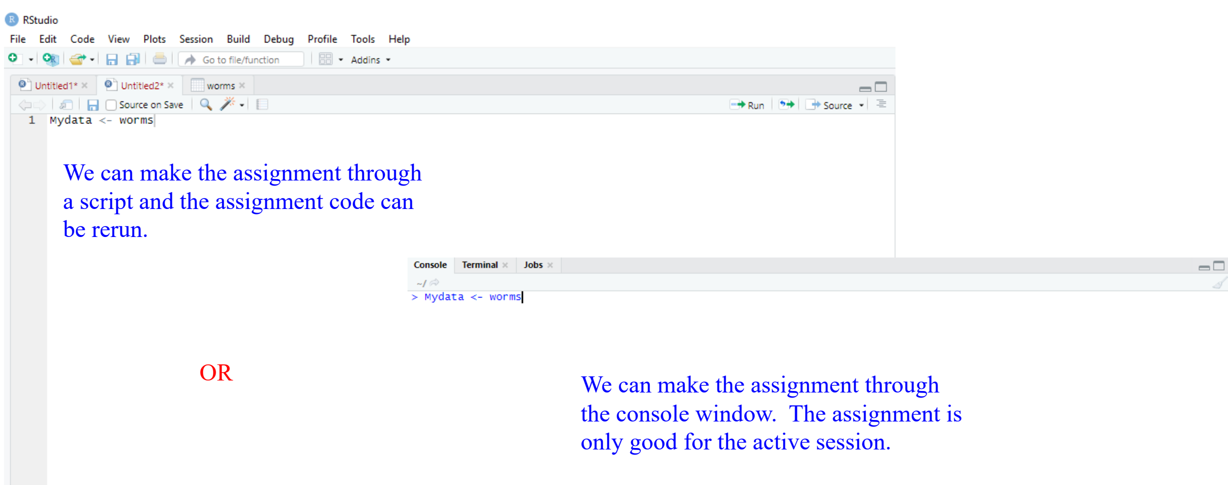
Importing file is only the first step. In order to use it, it must be assigned as an object so code can access it.

UNC CHARLOTTE 47

47

Topic 1: The R Programming Language – Part 1

Getting Data Into R



We can make the assignment through a script and the assignment code can be rerun.

OR

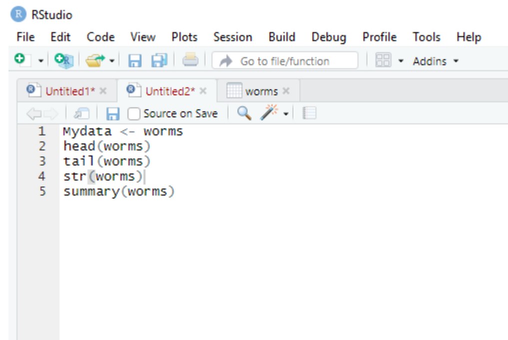
We can make the assignment through the console window. The assignment is only good for the active session.

UNC CHARLOTTE 48

48

Topic 1: The R Programming Language – Part 1

Getting Data Into R

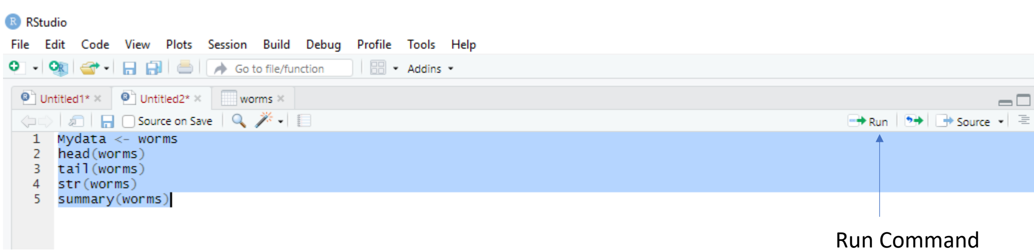


We can run the **head(), tail(), str(), and summary()** commands to gain insight into the structure of the data.

We can also run dim() to get the dimensions of the data.

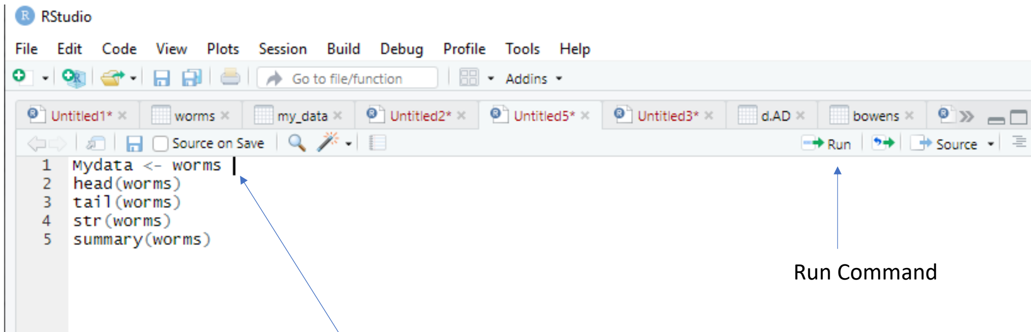
Topic 1: The R Programming Language – Part 1

Getting Data Into R



To run script statements in R, you must first highlight them and them click the "Run" command.

Getting Data Into R



Run Command

To run one line at a time, place cursor any where in the line them click the "Run" command. The cursor will jump to the next line where you can hit "Run" again to run that line.

Getting Data Into R

```
> Mydata <- worms
> head(worms)
       Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
1     Nashs.Field  3.6    11  Grassland     4.1 FALSE            4
2  Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
3   Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
4     Rush.Meadow  2.4     5     Meadow     4.9  TRUE            5
5 Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
6        Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
> tail(worms)
     Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
15   Pond.Field  4.1     0     Meadow     5.0  TRUE            6
16 Water.Meadow  3.9     0     Meadow     4.9  TRUE            8
17    Cheapside  2.2     8      Scrub     4.7  TRUE            4
18   Pound.Hill  4.4     2     Arable     4.5 FALSE            5
19   Gravel.Pit  2.9     1  Grassland     3.5 FALSE            1
20    Farm.Wood  0.8    10      Scrub     5.1  TRUE            3
> str(worms)
'data.frame':   20 obs. of  7 variables:
 $ Field.Name  : Factor w/ 20 levels "Ashurst","Cheapside",..: 8 17 10 16 7 11 3 1 19 15 ...
 $ Area        : num  3.6 5.1 2.8 2.4 3.8 3.1 3.5 2.1 1.9 1.5 ...
 $ Slope       : int  11 2 3 5 0 2 3 0 0 4 ...
 $ Vegetation  : Factor w/ 5 levels "Arable","Grassland",..: 2 1 2 3 5 2 2 1 4 2 ...
 $ Soil.pH     : num  4.1 5.2 4.3 4.9 4.2 3.9 4.2 4.8 5.7 5 ...
 $ Damp        : logi  FALSE FALSE FALSE TRUE FALSE FALSE ...
 $ Worm.density: int  4 7 2 5 6 2 3 4 9 7 ...
> summary(worms)
        Field.Name      Area             Slope             Vegetation      Soil.pH          Damp            Worm.density
 Ashurst      : 1   Min.   :0.800   Min.   : 0.00   Arable    :3   Min.   :3.500   Mode :logical   Min.   :0.00
 Cheapside    : 1   1st Qu.:2.175   1st Qu.: 0.75   Grassland:9   1st Qu.:4.100   FALSE:14        1st Qu.:2.00
 Church.Field: 1   Median :3.000   Median : 2.00   Meadow    :3   Median :4.600   TRUE :6         Median :4.00
 Farm.Wood    : 1   Mean   :2.990   Mean   : 3.50   Orchard   :1   Mean   :4.555                   Mean   :4.35
 Garden.Wood  : 1   3rd Qu.:3.725   3rd Qu.: 5.25   Scrub     :4   3rd Qu.:5.000                   3rd Qu.:6.25
 Gravel.Pit   : 1   Max.   :5.100   Max.   :11.00                  Max.   :5.700                   Max.   :9.00
 (Other)      :14
```

head() gives the 1st six rows.

tail() gives the last six rows.

str() gives structure of data set.

summary() gives quantile measures

Getting Data Into R

Reading data into R

```
read.table(file, header = TRUE, sep = "", quote = "\"'",
        dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
        row.names, col.names, as.is = !stringsAsFactors, na.strings = "NA",
        colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
        fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
        comment.char = "#", allowEscapes = FALSE, flush = FALSE,
        stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
        encoding = "unknown", text, skipNul = FALSE)
```

There are 27 options. The most used options are highlighted. We will normally take defaults on the others.

https://www.rdocumentation.org

Getting Data Into R

Reading data into R examples:

```
>read.table(file, header = TRUE, sep = ",", quote = "\"", dec = ".",
       fill = TRUE, comment.char = "", …)
>murder <- read.table("c:\\temp\\murders.txt",header=T,as.is="region")

>read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".",
       fill = TRUE, comment.char = "", …)

>MyData <- read.csv(file="c:/TheDataIWantToReadIn.csv", header=TRUE,
sep=",")

>read.delim(file, header = TRUE, sep = "\t", quote = "\"", dec = ".",
       fill = TRUE, comment.char = "", …)

>x <- read.delim("readcsv.csv", header=T)
```

read.delim assumes tab delimited by default.

Getting Data Into R

Reading data from websites:

```
my_data <- read.delim("http://www.sthda.com/upload/boxplot_format.txt")
head(my_data)

data <- read.csv("http://apps.fs.fed.us/fiadb-downloads/CSV/LICHEN_SPECIES_SUMMARY.csv")
head(data)

data2 <- read.table ("http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/cancer.txt",
header=T)
head(data2)

PATH <- 'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/mtcars.csv'
df <-read.csv(PATH, header =TRUE, sep = ',', stringsAsFactors =FALSE)
Head(df)
```


Getting Data Into R

Reading Data From Excel Files (xls|xlsx) into R

```
>install.packages("readxl")
># Loading
>library("readxl")
```

This is our 1st package install. To use it you must load it with the library command.

```
># xls files
>my_data <- read_excel("my_file.xls")
># xlsx files
>my_data <- read_excel("my_file.xlsx")
>my_data <- read_excel(file.choose())
># Specify sheet by its name
>my_data <- read_excel("my_file.xlsx", sheet = "data")
># Specify sheet by its index
>my_data <- read_excel("my_file.xlsx", sheet = 2)
```

This command let's you choose the file from a directory.

Getting Data Into R

Dataframes

Goal
Handling data using dataframe functions

The basic structure of a dataframe is that there is one observation per row and each column represents a variable, a measure, feature, or characteristic of that observation.

Learning how to handle your data, how to enter them into the computer, and how to read them into R are among the most important topics you will need to master.

R handles data in objects known as dataframes. A dataframe is an object with rows and columns (a bit like a matrix).

Getting Data Into R

Dataframes

Here is a spreadsheet in the form of a dataframe with seven variables.

The leftmost contains the row names.

The other variables are
- Numeric (Area, Slope, Soil pH and Worm Density)
- Categorical (Field Name and Vegetation)
- Logical (Damp is either true = T or false = F).

Field Name	Area	Slope	Vegetation	Soil pH	Damp	Worm Density
Nash's Field	3.6	11	Grassland	4.1	F	4
Silwood Bottom	5.1	2	Arable	5.2	F	7
Nursery Field	2.8	3	Grassland	4.3	F	2
Rush Meadow	2.4	5	Meadow	4.9	T	5
Gunness' Thicket	3.8	0	Scrub	4.2	F	6
Oak Mead	3.1	2	Grassland	3.9	F	2
Church Field	3.5	3	Grassland	4.2	F	3
Ashurst	2.1	0	Arable	4.8	F	4
The Orchard	1.9	0	Orchard	5.7	F	9
Rookery Slope	1.5	4	Grassland	5	T	7
Garden Wood	2.9	10	Scrub	5.2	F	8
North Gravel	3.3	1	Grassland	4.1	F	1
South Gravel	3.7	2	Grassland	4	F	2
Observatory Ridge	1.8	6	Grassland	3.8	F	0
Pond Field	4.1	0	Meadow	5	T	6
Water Meadow	3.9	0	Meadow	4.9	T	8
Cheapside	2.2	8	Scrub	4.7	T	4

Getting Data Into R

Creating Dataframes

It is easy to create a dataframe by using the data.frame() function.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))
> str(x)    # structure of x
'data.frame':   2 obs. of  3 variables:
$ SN  : int  1 2
$ Age : num  21 15
$ Name: Factor w/ 2 levels "Dora", "John": 2 1
```

Notice above that the third column, Name is of type factor, instead of a character vector. By default, data.frame() function converts character vector into factor. To suppress this behavior, we can pass the argument stringsAsFactors=FALSE.

Getting Data Into R

Dataframes vs. Matrices

A matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns.

```
>matrix(1:9, byrow = TRUE, nrow = 3)

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

Getting Data Into R

Dataframes vs. Matrices

```
> x <- matrix(1:9, nrow = 3, dimnames = list(c("X","Y","Z"), c("A","B","C")))
> x
  A  B  C
X 1  4  7
Y 2  5  8
Z 3  6  9
```

This argument lets you define row and column labels.

Getting Data Into R

Dataframes vs. Matrices

Matrices

All columns in a matrix must have the same mode (numeric, character, etc.) and the same length. The general format is

mymatrix <- **matrix**(*vector*, **nrow**=*r*, **ncol**=*c*, **byrow**=*FALSE*,
dimnames=**list**(*char_vector_rownames*, *char_vector_colnames*))

byrow=TRUE indicates that the matrix should be filled by rows. **byrow=FALSE** indicates that the matrix should be filled by columns (the default). **dimnames** provides optional labels for the columns and rows.

Getting Data Into R

Dataframes vs. Matrices

Data Frames

A dataframe is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
data.frame(..., row.names = NULL, check.rows = FALSE,
        check.names = TRUE, fix.empty.names = TRUE,
        stringsAsFactors = default.stringsAsFactors())

default.stringsAsFactors()]
```

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


Getting Data Into R

Extracting Data From A Data Frame:

1. The subset() function
2. The use of [ ]
3. The $ operators

Getting Data Into R

The **subset( )** function is the easiest way to select variables and observations. In the following example, we select all rows that have a value of age greater than or equal to 20 or age less then 10. We keep the ID and Weight columns.

```
# using subset function
>newdata <- subset(mydata, age >= 20 | age < 10)
>select=c(ID, Weight))

# using subset function
>newdata <- subset(mydata, sex=="m" & age > 25)
>select=(weight:income)
```


Getting Data Into R

```
>#Extensive Subset() Example
>numvec = c(2,5,8,9,0,6,7,8,4,5,7,11)
>charvec = c("David","James","Sara","Tim","Pierre", "Janice","Sara","Priya","Keith","Mark", "Apple","Sara")
>gender = c("M","M","F","M","M","M","F","F","F","M","M","F")
>state = c("CO","KS","CA","IA","MO","FL","CA","CO","FL","CA","WY","AZ")

>subset(numvec, numvec > 7)
>subset(numvec, numvec < 9 & numvec > 4)
>subset(numvec, numvec < 3 |numvec > 9)

>df = data.frame(var1=c(numvec), var2=c(charvec), gender=c(gender), state=c(state))

>a<-subset(df, var1 < 5)
>b<-subset(df, var2 == "Sara")
>c<-subset(df, var1==5, select=c(var2, state))
>d<-subset(df, var2 != "Sara" & gender == "F" & var1 > 5)
```

Let's print a, b, c, and d.

Getting Data Into R

The [ ] Bracket Function

Brackets lets you select, or subset, data from a vector, matrix, array, list or data frame. Exactly how this works and the results you get depend, in part, on the data type. You can use

- [] for subsetting

```
>#Example
>my_df <- data.frame(a = runif(10),
>                    b = rnorm(10, 10),
>                    c = letters[1:10],
>                    d = LETTERS[1:2])
>
>my_df
```

Output

```
##             a         b c d
## 1  0.10709793 11.529173 a A
## 2  0.92978064  9.894295 b B
## 3  0.33239152  8.760886 c A
## 4  0.15785918  9.618424 d B
## 5  0.31492607 10.912449 e A
## 6  0.76551955 10.986973 f B
## 7  0.02208064 11.007789 g A
## 8  0.28574694 11.336040 h B
## 9  0.58989873  9.285833 i A
## 10 0.24728872 11.803678 j B
```

Functions:
1. runif() – Uniform random numbers
2. rnorm() – Normal random numbers
3. letters – Lower case letters
4. Letters – Uppercase letters

Getting Data Into R

The [ ] Bracket Function

#Examples

1. my_df[1:3] (no comma) will subset my_df, returning the first three columns as a data frame.
2. my_df[1:3,] (with comma, numbers to left of the comma) will subset my_df and return the first three rows as a data frame.
3. my_df[, 1:3] (with comma, numbers to right of the comma) will subset my_df and return the first three columns as a data frame, the same as my_df[1:3].
4. my_df[1:3, 1] selects the first three rows, but only from the 1st column
5. index <- c(1, 3, 5, 10)
   my_df[index,] — You can define a variable of indices and subset based on that variable.
6. index <- c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE)
   my_df[index,] — You can define a variable of Boolean operators and subset based on that variable.

Getting Data Into R

The $ operator allows you extract elements by name.

Examples:

```
>x <- list(a=1, b=2, c=3)
>x$b
2
```

```
### import education expenditure data set and assign column names
education <- read.csv("https://vincentarelbundock.github.io/Rdatasets/csv/robustbase/education.csv",
                      stringsAsFactors = FALSE)
colnames(education) < - c("X","State","Region","Urban.Population","Per.Capita.Income",
                      "Minor.Population","Education.Expenditures")
View(education)
```


Homework

1. Create a data frame named "functions" that looks like the one below by making the following variable assignments:

   i. A = -2.7
   ii. B = -0.5
   iii. C = 0.3
   iv. D = 1.5
   v. E. = 2.8

   Use the functions indicated in the data frame. What can you conclude about the differences between floor, ceiling, trunc, and round.

Function	A	B	C	D	E
Floor	-3	-1	0	1	2
Ceiling	-2	0	1	2	3
Trunc	-2	0	0	1	2
Round	-3	-1	0	2	3

Homework

2. Install the following Packages:

   i. readxl
   ii. data.table

   Issue a command in your script to list the datasets in these packages.

   Read the following datasets

   - `data <- read.csv("http://apps.fs.fed.us/fiad-bdownloads/CSV/LICHEN_SPECIES_SUMMARY.csv")`
   - `data2 <- read.table ("http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/cancer.txt", header=T)`
   - `'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/mtcars.csv'`

   Examine the head, tail, and structure of each. Some links maybe broken.

Topic 1: The R Programming Language – Part 1

Homework

3. Use the following code to create the education dataset:

```
>education <- read.csv("https://vincentarelbundock.github.io/Rdatasets/csv/robustbase/education.csv",
        stringsAsFactors = FALSE)
>colnames(education) < - c("X","State","Region","Urban.Population","Per.Capita.Income",
        "Minor.Population","Education.Expenditures")
>View(education)
```

   Calculate the following quantities:

   a. Mean, standard deviation, and range of
      i. Urban.Population
      ii. Per.Capita.Income
      iii. Minor.Population
      iv. Education.Expenditures
   b. Store the values calculated above in vectors of the same name
   c. Print the results by invoking the variables

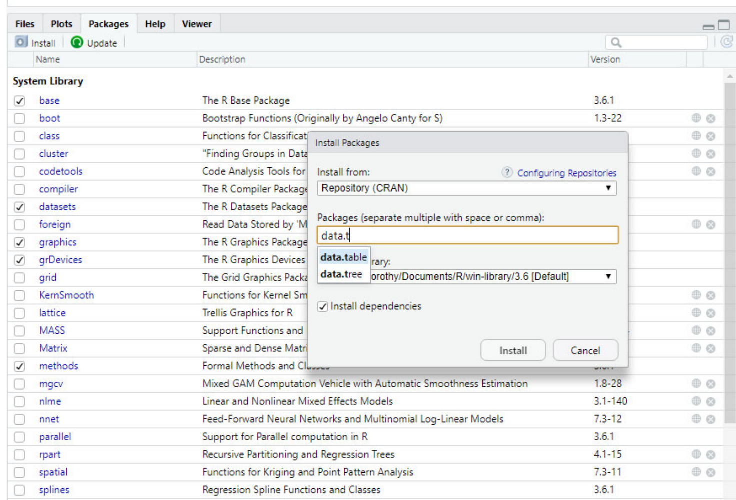
Topic 1: The R Programming Language – Part 1

Getting Data Into

Introduction to data.table

Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, friendly and fast character-separated-value read/write. Offers a natural and flexible syntax, for faster development.

data.table inherits from data.frame. It offers fast and memory efficient: file reader and writer, aggregations, updates, equi, non-equi, rolling, range and interval joins, in a short and flexible syntax, for faster development.



73

Topic 1: The R Programming Language – Part 1

Getting Data Into

Data.Table Functions

Item	Function	Description
1	**:=**	**Assignment by Reference**
2	address	Address in RAM of a variable
3	all.equal	Equality Test Between Two Data Tables
4	as.data.table	Coerce to data.table
5	as.data.table.xts	Equality Test Between Two Data Tables
6	as.matrix	Coerce to data.table
7	as.xts.data.table	Efficient data.table to xts conversion
8	**between**	**Convenience functions for range subsets.**
9	chmatch	Faster match of character vectors
10	copy	Copy an entire object
11	data.table-class	S4 Definition for data.table
12	data.table.optimize	Optimisations in data.table
13	dcast.data.table	Fast dcast for data.table
14	duplicated	Determine Duplicate Rows
15	**first**	**First item of an object**
16	foverlaps	Fast overlap joins
17	frank	Fast rank
18	**fread**	**Fast and friendly file finagler**
19	fsort	Fast parallel sort
20	**fwrite**	**Fast CSV writer**
21	groupingsets	Grouping Set aggregation for data tables
22	IDateTime	Integer based date class
23	J	Creates a Join data table
24	last	Last item of an object
25	like	Convenience function for calling regexpr.
26	melt.data.table	Fast melt for data.table
27	merge	Merge two data.tables
28	na.omit.data.table	Remove rows with missing values on columns specified
29	patterns	Obtain matching indices corresponding to patterns
30	print.data.table	data.table Printing Options
31	rbindlist	Makes one data.table from a list of many
32	rleid	Generate run-length type group id
33	roll	Rolling functions
34	rowid	Generate unique row ids within each group
35	set2key	Deprecated.
36	setattr	Set attributes of objects by reference
37	setcolorder	Fast column reordering of a data.table by reference
38	setDF	Coerce a data.table to data.frame by reference
39	setDT	Coerce lists and data.frames to data.table by reference
40	setDTthreads	Set or get number of threads that data.table should use
41	setkey	Create key on a data.table
42	setNumericRounding	Change or turn off numeric rounding
43	setops	Set operations for data tables
44	setorder	Fast row reordering of a data.table by reference
45	shift	Fast lead/lag for vectors and lists
46	shouldPrint	For use by packages that mimic/divert auto printing e.g. IRkernel and knitr
47	special-symbols	Special symbols
48	split	Split data.table into chunks in a list
49	subset.data.table	Subsetting data.tables
50	tables	Display 'data.table' metadata
51	test.data.table	Runs a set of tests.
52	timetaken	Pretty print of time taken
53	transform.data.table	Data table utilities
54	**transpose**	**Efficient transpose of list**
55	truelength	Over-allocation access
56	tstrsplit	strsplit and transpose the resulting list efficiently
57	update.dev.pkg	Perform update of development version of a package

74

Getting Data Into R

data.table Package

fread(): Similar to read.table but faster and convenient.

```
fread(input, sep="auto", sep2="auto", nrows=-1L, header="auto", na.strings="NA",
stringsAsFactors=FALSE, verbose=FALSE, autostart=30L, skip=-1L, select=NULL,
drop=NULL, colClasses=NULL, integer64=getOption("datatable.integer64"),   # default:
"integer64" showProgress=getOption("datatable.showProgress")   # default: TRUE
)
```

```
# Reads URLs directly :
fread("http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat")
```

```
#Reads Imported CSV files
fread("pre2012_alldatapoints.csv", sep = ",", header= TRUE)
fread('filePath_and_fileName', sep = ',')
```


fread(), fwrire(), write.csv()

“ R does not like the “curly” quote mark. When quotes are copied from a pdf or other source, they are not translated in the “non-curly” quote mark by R.

" R likes the “non-curly” quote mark. If you get an error such as: Error: unexpected input in … “"
Just re-type the quote mark to correct the error.

Note: R also has the same issue with the “^” exponentiation symbol. It does not copy well from pdf or other sources. Just re-type the symbol in R to correct.

fread(), fwrite(), write.csv()

fread(filename**):** For reading large files. Get use to using this file read statement. See complete documentation at https://www.rdocumentation.org/packages/data.table/versions/1.12.6/topics/fread

Examples: Don't forget to issue the command library(data.table) before using fread()!

1. MyData <- fread("C:/RData/pre2012_alldatapoints.csv", sep = ",", header= TRUE)
2. MyData <- fread("C:/RData/pre2012_alldatapoints.csv", sep = ",", header= TRUE, stringsAsFactors=FALSE)
3. MyData <- fread("C:/RData/pre2012_alldatapoints.csv", sep = ",", header= TRUE, stringsAsFactors=FALSE, drop = 2:3) #You can also specify c(2,3) to drop those columns.
4. MyData<- fread("http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat")
5. fwrite(fread("http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat"), "C:/Rdata/MyData.csv", sep = ",")

fread(), fwrite(), write.csv()

fwrite(filename**):** A super fast function to write data sets developed in r to external files. See complete documentation at https://www.rdocumentation.org/packages/data.table/versions/1.12.6/topics/fwrite

Examples: Don't forget to issue the command library(data.table)

1. fwrite(mtcars, "C:/Rdata/Mtcars2.csv")
2. fwrite(mtcars, "C:/Rdata/Mtcars2.csv" ", sep = ",")
3. fwrite(mtcars, "C:/Rdata/Mtcars2.txt", sep = "\t")

sep: The separator between columns. Default is ",".

fread(), fwrite(), write.csv()

write.csv(filename**):** Not as fast as fwrite and can't handle as big files, but it is very simple to use. It only writes csv files. See complete documentation at https://www.rdocumentation.org/packages/utils/versions/3.6.1/topics/write.table

Examples: You don't need to issue the command library(data.table)

1. write.csv(mtcars, file = "C:/Rdata/Mtcars2.csv")
2. write.csv(mtcars, file = "C:/Rdata/Mtcars2.csv" , rownames = FALSE)
3. write.csv(mtcars, file = "C:/Rdata/Mtcars2.csv" , rownames = FALSE, na="")

Getting Data Into R

data.table Package

This is an enhanced version of data.frame!

Description

- data.table inherits from data.frame.
- It offers fast and memory efficient:
  - file reader and writer,
  - aggregations,
  - updates,
  - equi, non-equi, rolling, range and interval joins
- Since a data.table is a data.frame, it is compatible with R functions and packages that accept data.frames.

Go to link below for additional support on data.table:
https://github.com/Rdatatable/data.table/wiki/Support

Getting Data Into R

What Is a Non-Equi Join? (A non equal sign join)

non-equi join operators

Operator	Meaning
“>”	Greater than
“>=”	Greater than or equal to
“<”	Less than
“<=”	Less than or equal to
“!=”	Not equal to
”<>”	Not equal to (ANSI Standard)
BETWEEN ... AND	Values in a range between *x* and *y*

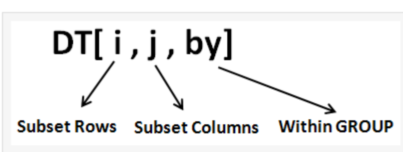
81

Getting Data Into R

data.table Syntax

data.table Syntax

The syntax of data.table is shown in the image :



data.table Syntax

The data.table syntax is NOT RESTRICTED to only 3 parameters.

DT[i , j , by]

1. The first parameter of data.table **i** refers to rows. It implies subsetting rows. It is equivalent to **WHERE** clause in SQL
2. The second parameter of data.table **j** refers to columns. It implies subsetting columns (dropping / keeping). It is equivalent to **SELECT** clause in SQL.
3. The third parameter of data.table **by** refers to adding a group so that all calculations would be done within a group. Equivalent to SQL's **GROUP BY** clause.

82

Getting Data Into R

data.table Example

```
install.packages("data.table")          # Install Package

library(data.table)                     #load required library

#Read Data
mydata = fread("https://github.com/arunsrinivasan/satrdays-workshop/raw/master/flights_2014.csv")

#Describe Data
names(mydata)
head(mydata)
tail(mydata)
```


Getting Data Into R

data.table Examples – Selecting or Keeping Records

```
# returns a vector
dat1 = mydata[ , origin]

# returns a data.table
dat1 = mydata[ , .(origin)]   OR  dat1 = mydata[, c("origin"), with=FALSE]

#return a column based on position
dat2 =mydata[, 2, with=FALSE]

#keeping multiple columns
dat3 = mydata[, .(origin, year, month, hour)]

#keeping columns 2 though 4
dat4 = mydata[, c(2:4), with=FALSE]
```

Note:
Setting *with = FALSE* allows referring to columns by number of a variable that contains column names.

Getting Data Into R

data.table Examples – Selecting or Keeping Records

```
DF = data.frame(x = c(1,1,1,2,2,3,3,3), y = 1:8)
```

```
## (1) normal way
DF[DF$x > 1, ] # data.frame needs that ',' as well
#    x y
# 4 2 4
# 5 2 5
# 6 3 6
# 7 3 7
# 8 3 8
```

```
## (2) using with
DF[with(DF, x > 1), ]
#    x y
# 4 2 4
# 5 2 5
# 6 3 6
# 7 3 7
# 8 3 8
```


85

Getting Data Into R

data.table Examples – Selecting or Keeping Records

```
ans <- flights[, .(.N), by = .(origin)]
ans
#   origin     N
# 1:    JFK 81483
# 2:    LGA 84433
# 3:    EWR 87400

## or equivalently using a character vector in by
ans <- flights[, .(.N), by = "origin"]
```

.(.N): Gives a count by group

86

Getting Data Into R

Examples using data.table Package

Calculate the number of trips for each origin airport for carrier code "AA"
The unique carrier code "AA" corresponds to American Airlines Inc.

```
ans <- flights[carrier == "AA", .N, by = origin]
ans
```

```
#      origin     N
# 1:    JFK 11923
# 2:    LGA 11730
# 3:    EWR  2649
```


Topic 1: The R Programming Language – Part 1

Getting Data Into R

Examples using data.table Package

Get the average arrival and departure delay for each orig, dest pair for each month for carrier code "AA"

```
ans <- flights[carrier == "AA",  .(mean(arr_delay),
                                   mean(dep_delay)),
                                   by = .(origin, dest, month)]
```

```
ans
#        origin dest month         V1         V2
#    1:     JFK  LAX     1   6.590361 14.2289157
#    2:     LGA  PBI     1  -7.758621  0.3103448
#    3:     EWR  LAX     1   1.366667  7.5000000
#    4:     JFK  MIA     1  15.720670 18.7430168
#    5:     JFK  SEA     1  14.357143 30.7500000
#   ---
# 196:     LGA  MIA    10  -6.251799 -1.4208633
# 197:     JFK  MIA    10  -1.880184  6.6774194
# 198:     EWR  PHX    10  -3.032258 -4.2903226
# 199:     JFK  MCO    10 -10.048387 -1.6129032
# 200:     JFK  DCA    10  16.483871 15.5161290
```

Getting Data Into R

HW Using Flights Data

A. Find all flights that:

1. Had an arrival delay of two or more hours
2. Flew to Houston (IAH or HOU)
3. Were operated by United, American, or Delta
4. Arrived more than two hours late, but didn't leave late
5. Were delayed by at least an hour, but made up over 30 minutes in flight
6. Departed between midnight and 6am (inclusive)
7. How many flights have a missing dep_time?

Getting Data Into R

B. Answer the following questions

#1. Given the matrix below, answer the following questions:

```
>A<-matrix(A <- c(8, 1, 4, 5, 6, 3, 9, 10, 12, 23, 44, 32 ), byrow = TRUE, nrow = 3)
>A
```

#2. Provide the output of the following statements based on matrix A

a. A[1,1]
b. A[, 4]
c. A[c(2, 3), c(1, 3)]
d. A[, 3]>4

#3. Provide the output of the following statements

```
>matrix_a <-matrix(1:10, byrow = TRUE, nrow = 5)
>matrix_a
```

Getting Data Into R

B. Answer the following questions

#4. Provide the output of the following statements
```
>matrix_b <-matrix(1:10, byrow = FALSE, nrow = 5)
>matrix_b
```

#5. Provide the output of the following statements
```
>x <- matrix(c(50, 37, 72, 87, 78, 45), ncol=2)
>x
```

#6. Provide the output of the following statements
```
>matrix_d <-matrix(1:12, byrow = FALSE, ncol = 3)
>matrix_d
```


Topic 1: The R Programming Language – Part 1

Getting Data Into R

B. Answer the following questions

#7. Provide the output of the following statements
```
>matrix_a1 <- cbind(matrix_a, c(1:5))
>dim(matrix_a1)
```

#8. Provide the output of the following statements
```
matrix_c <-matrix(1:12, byrow = FALSE, ncol = 3)
add_row <- c(1:3)
matrix_c <- rbind(matrix_b, add_row)
dim(matrix_c)
```

#9. Provide the output of the following statements based on matrix_c

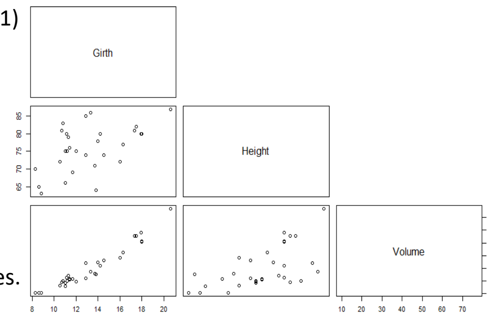
a. matrix_c[1,2]
b. matrix_c[1:3,2:3]
c. matrix_c[,1]
d. matrix_c[1,]

Getting Data Into R

B. Answer the following questions

#10. Given the pairs scatterplot below for the variables Girth, Height, and Volume, rank the correlations from highest (Rank =1) to lowest (Rank = 3). Note: Only the lower diagonal is shown

Correlation Pair	Rank
Height & Volume	______________
Girth & Volume	______________
Girth & Height	______________



Note: The data represents the height, volume, and girth of trees.

93

Basic Plotting in R

R Bar Plot

Bar plots can be created in R using the barplot() function. We can supply a vector or matrix to this function. If we supply a vector, the plot will have bars with their heights equal to the elements in the vector.

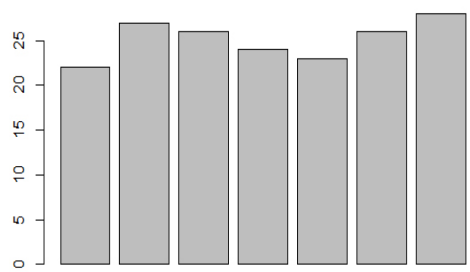
Let us suppose, we have a vector of maximum temperatures (in degree Celsius) for seven days as follows.

```
>max.temp <- c(22, 27, 26, 24, 23, 26, 28)
```

Bar Heights

Now we can make a bar plot out of this data.

```
>barplot(max.temp)
```



This function can take a lot of argument to control the way our data is plotted. You can read about them in the help section ?barplot.

94

Basic Plotting in R

Plotting Categorical Data

Sometimes we have to plot the count of each item as bar plots from categorical data. For example, here is a vector of age of 10 college freshmen.

```
>age <- c(17,18,18,17,18,19,18,16,18,18)
```

Simply doing barplot(age) will not give us the required plot. It will plot 10 bars with height equal to the student's age. But we want to know the number of student in each age category.

This count can be quickly found using the table() function, as shown below.

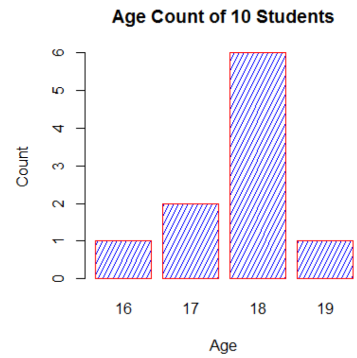
```
> table(age)
age
16 17 18 19
 1  2  6  1
```

The table() function creates a frequency table.

Now plotting this data will give our required bar plot. Note below, that we define the argument density to shade the bars.

R correctly interprets the results of the table.

```
>barplot(table(age), main="Age Count of 10 Students",
        xlab="Age", ylab="Count", border="red",
        col="blue",
        density=10)
```

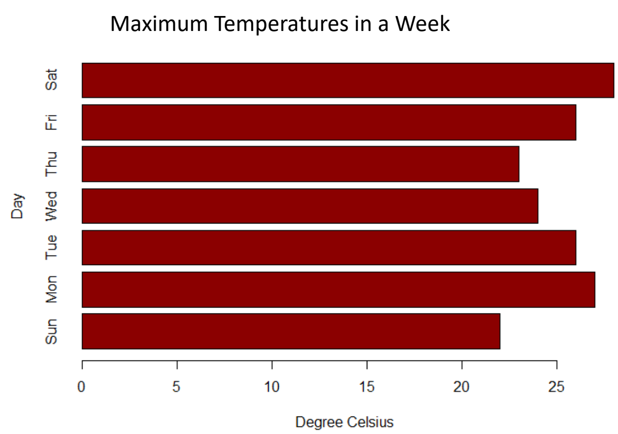


Basic Plotting in R

Some of the frequently used ones are, **main** to give the title, **xlab** and **ylab** to provide labels for the **axes**, **names.arg** for naming each bar, **col** to define color etc.

We can also plot bars horizontally by providing the argument **horiz = TRUE**.

```
# barchart with added parameters
barplot(max.temp,
main = "Maximum Temperatures in a Week",
xlab = "Degree Celsius",
ylab = "Day",
names.arg = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"),
col = "darkred",
horiz = TRUE)
```



Basic Plotting in R

Histogram can be created using the hist() function in R programming language. This function takes in a vector of values for which the histogram is plotted.

Let us use the built-in dataset airquality which has Daily air quality measurements in New York, May to September 1973.-R documentation.

```
> str(airquality)
'data.frame':       153 obs. of  6 variables:
$ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
$ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
$ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
$ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
$ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
$ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

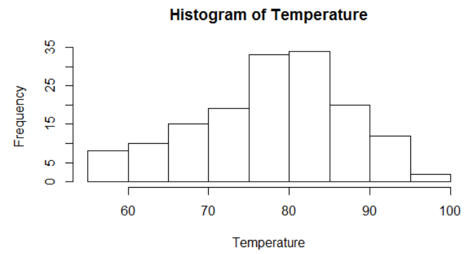
We will use the temperature parameter which has 154 observations in degree Fahrenheit.

str(): Gives the structure of your dataset

Example 1: Simple histogram

```
Temperature <- airquality$Temp
hist(Temperature)
```

datasetname$column_name: assigns a column to a variable



We can see above that there are 9 cells with equally spaced breaks. In this case, the height of a cell is equal to the number of observation falling in that cell.

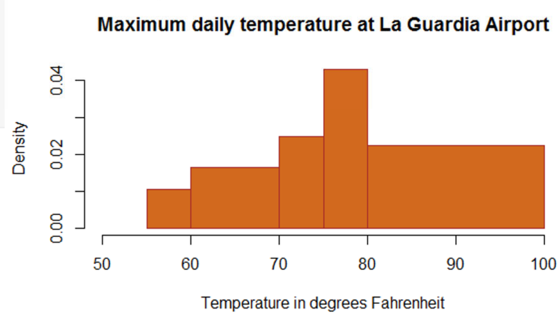
97

Basic Plotting in R

Histogram with non-uniform width

```
hist(Temperature,
main="Maximum daily temperature at La Guardia Airport",
xlab="Temperature in degrees Fahrenheit",
xlim=c(50,100),
col="chocolate",
border="brown",
breaks=c(55,60,70,75,80,100)
)
```

We can manually define the breaks in the data with this option.

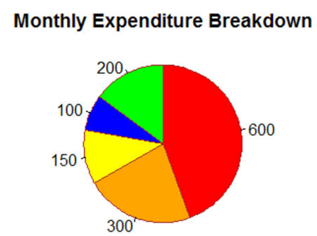


98

Basic Plotting in R

Pie Chart

```
pie(expenditure,
labels=as.character(expenditure),
main="Monthly Expenditure Breakdown",
col=c("red","orange","yellow","blue","green"),
border="brown",
clockwise=TRUE
)
```



As seen in the pie figure, we have used the actual amount as labels. Also, the chart is drawn in clockwise fashion.

Since the human eye is relatively bad at judging angles, other types of charts are more appropriate than pie charts.

This is also stated in the R documentation – **Pie charts are a very bad way of displaying information.**

The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

Basic Plotting in R

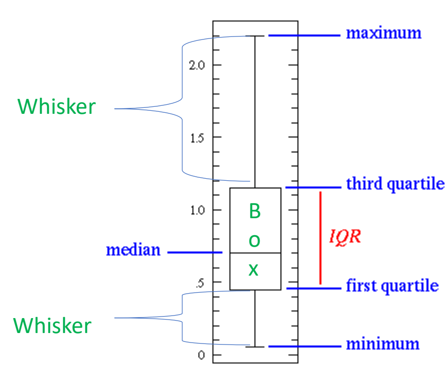
Box Plot – Box & Whisker Plot

The box plot (a.k.a. box and whisker diagram) is a standardized way of displaying the distribution of data based on the five-number summary:

1. Minimum
2. First quartile
3. Median
4. Third quartile
5. Maximum.

Boxplots not only show the location and spread of data but also indicate skewness, which shows up as asymmetry in the sizes of the upper and lower parts of the box.

Tukey's Famous Five Numbers



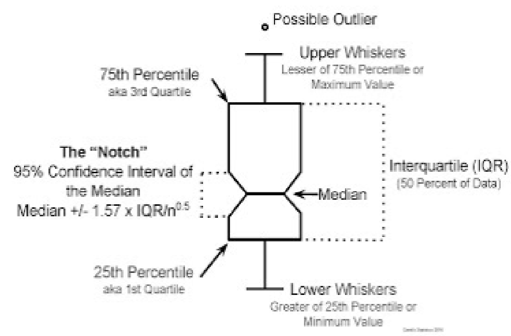
IQR – Interquartile Range

Basic Plotting in R

Boxplots with Notches to Indicate Significant Differences

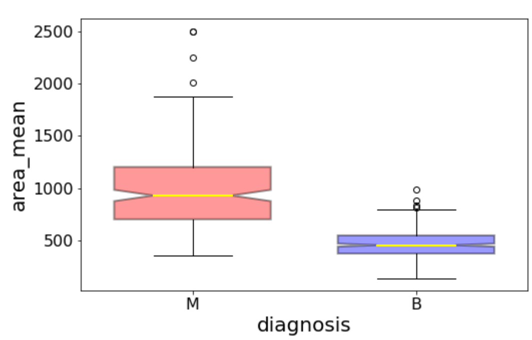
Boxplots are very good at showing the distribution of the data points around the median, but they are not so good at indicating whether or not the median values are significantly different from one another.

Tukey invented **notches** to get the best of both worlds. The notches are drawn as a 'waist' on either side of the median and are intended to give a rough impression of the significance of the differences between two medians.



Basic Plotting in R

Non-Overlapping Boxplots

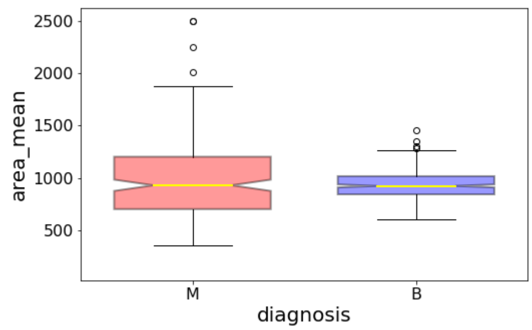


Overlapping mean the notches are overlapping. When the medians do not line up, it means you can conclude that with 95% confidence, that the true medians do differ.

Topic 1: The R Programming Language – Part 1

Basic Plotting in R

Overlapping Boxplots



In this graphic the notches are overlapping. You can conclude that with 95% confidence, that the true medians do NOT differ.

Basic Plotting in R

R Box Plot

In R, boxplot (and whisker plot) is created using the boxplot() function.

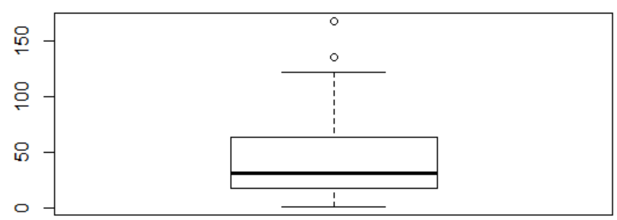
The boxplot() function takes in any number of numeric vectors, drawing a boxplot for each vector.

You can also pass in a list (or dataframe) with numeric vectors as its components. Let us use the built-in dataset airquality which has “Daily air quality measurements in New York, May to September 1973.”-R documentation.

```
> str(airquality)
'data.frame':    153 obs. of  6 variables:
$ Ozone   : int  41 36 12 18 NA 28 23 19 8 NA ...
$ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
$ Wind    : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
$ Temp    : int  67 72 74 62 56 66 65 59 61 69 ...
$ Month   : int  5 5 5 5 5 5 5 5 5 5 ...
$ Day     : int  1 2 3 4 5 6 7 8 9 10 ...
```

Let us make a boxplot for the ozone readings.

```
>boxplot(airquality$Ozone)
```



We can see that data above the median is more dispersed. We can also notice two outliers at the higher extreme.

We can pass in additional parameters to control the way our plot looks. You can read about them in the help section ?boxplot.

Some of the frequently used ones are, main-to give the title, xlab and ylab-to provide labels for the axes, col to define color etc.

Additionally, with the argument horizontal = TRUE we can plot it horizontally and with notch = TRUE we can add a notch to the box.

Basic Plotting in R

R Box Plot

Syntax

The basic syntax to create a boxplot in R is −

boxplot(x, data, notch, varwidth, names, main, xlab, ylab)

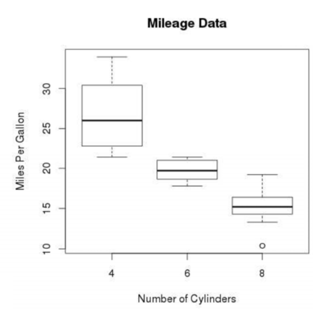
Following is the description of the parameters used:

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.
- **xlab** is x-axis label.
- **ylab** is x-axis label.

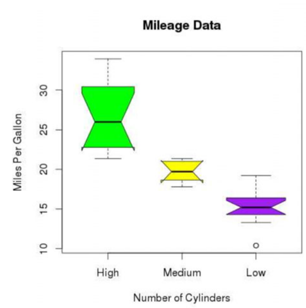
Basic Plotting in R

R Box Plot

```
>input <- mtcars[,c('mpg','cyl')]
>print(head(input))
>#2Plot the chart.
>boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
>  ylab = "Miles Per Gallon", main = "Mileage Data")
```




```
># Plot the chart.
>boxplot(mpg ~ cyl, data = mtcars,
>  xlab = "Number of Cylinders",
>  ylab = "Miles Per Gallon",
>  main = "Mileage Data",
>  notch = TRUE,
>  varwidth = TRUE,
>  col = c("green","yellow","purple"),
>  names = c("High","Medium","Low")
>  )
```



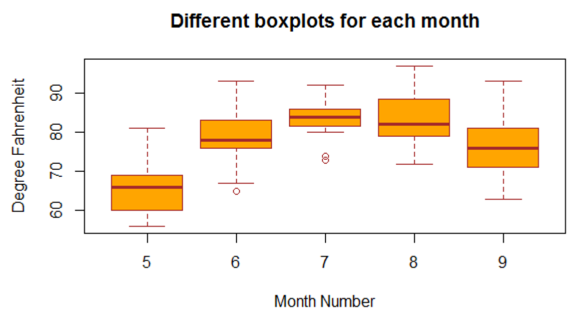
Basic Plotting in R

Boxplot form Formula

The function boxplot() can also take in formulas of the form y~x where, y is a numeric vector which is grouped according to the value of x.

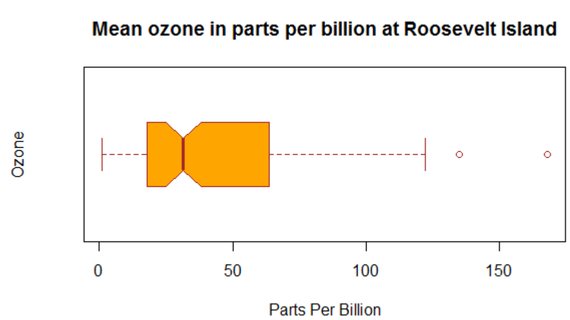
For example, in our dataset airquality, the Temp can be our numeric vector. Month can be our grouping variable, so that we get the boxplot for each month separately. In our dataset, month is in the form of number (1=January, 2-Febuary and so on).

```
boxplot(Temp~Month,
data=airquality,
main="Different boxplots for
each month",
xlab="Month Number",
ylab="Degree Fahrenheit",
col="orange",
border="brown"
)
```



Basic Plotting in R

```
boxplot(airquality$Ozone,
main = "Mean ozone in parts per billion at
Roosevelt Island",
xlab = "Parts Per Billion",
ylab = "Ozone",
col = "orange",
border = "brown",
horizontal = TRUE,
notch = TRUE
)
```



Basic Plotting in R

R Plot Function

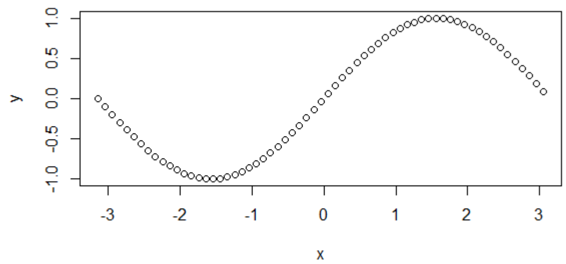
The most used plotting function in R programming is the plot() function. It is a generic function, meaning, it has many methods which are called according to the type of object passed to plot().

In the simplest case, we can pass in a vector and we will get a scatter plot of magnitude vs index. But generally, we pass in two vectors and a scatter plot of these points are plotted.

For example, the command plot(c(1,2),c(3,5)) would plot the points (1,3) and (2,5).

Here is a more concrete example where we plot a sine function form range -pi to pi.

```
x <- seq(-pi,pi,0.1)
plot(x, sin(x))
```



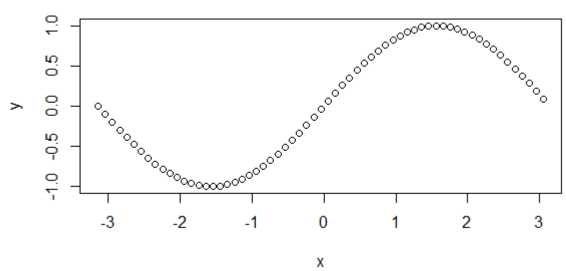
Basic Plotting in R

Scatter Plot vs. Smooth line

```
x <- seq(-pi,pi,0.1)
y<-sin(x)

plot(x, sin(x)) # Plot a scatter

plot(y~x) # Plot a function
```



Basic Plotting in R

Mtcars Dataset

The data was extracted from the 1974 *Motor Trend* US magazine and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Format
A data frame with 32 observations on 11 (numeric) variables.

[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs.)
[, 7]	qsec	1/4 mile time
[, 8]	vs	Engine (0 = V-shaped, 1 = straight)
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

Topic 1: The R Programming Language – Part 1

Basic Plotting in R

Format
A data frame with 32 observations on 11 (numeric) variables.

[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs.)
[, 7]	qsec	1/4 mile time
[, 8]	vs	Engine (0 = V-shaped, 1 = straight)
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.44	18.9	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.07	17.4	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.73	17.6	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.78	18	0	0	3	3
Cadillac Fleetwood	10.4	8	472	205	2.93	5.25	17.98	0	0	3	4
Lincoln Continental	10.4	8	460	215	3	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.2	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.7	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318	150	2.76	3.52	16.87	0	0	3	2
AMC Javelin	15.2	8	304	150	3.15	3.435	17.3	0	0	3	2
Camaro Z28	13.3	8	350	245	3.73	3.84	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79	66	4.08	1.935	18.9	1	1	4	1
Porsche 914-2	26	4	120.3	91	4.43	2.14	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351	264	4.22	3.17	14.5	0	1	5	4
Ferrari Dino	19.7	6	145	175	3.62	2.77	15.5	0	1	5	6
Maserati Bora	15	8	301	335	3.54	3.57	14.6	0	1	5	8
Volvo 142E	21.4	4	121	109	4.11	2.78	18.6	1	1	4	2

Basic Plotting in R

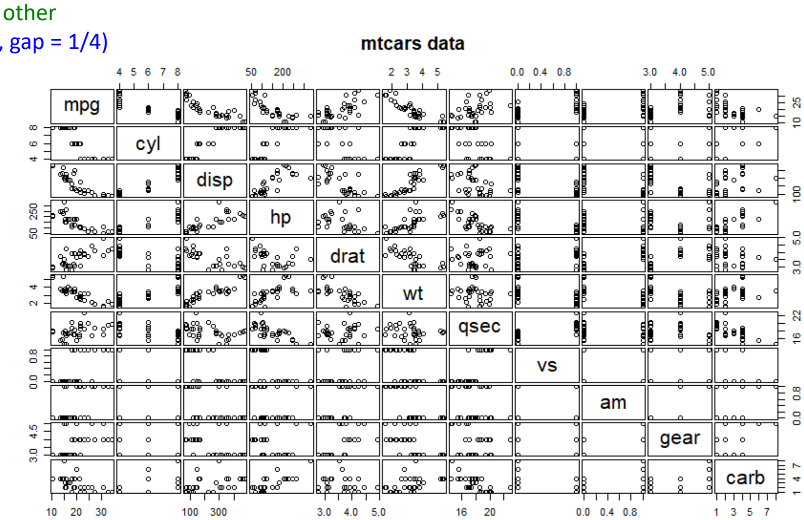
>library(graphics)
>Mtcars<-mtcars

>#Plot all pairs of data against each other
>**pairs**(Mtcars, main= "mtcars data", gap = 1/4)

>#Plot mpg as a function of disp for all levels of cyl
>**coplot**(mpg ~ disp | as.factor(cyl), data = Mtcars, panel =
> panel.smooth, rows=1)

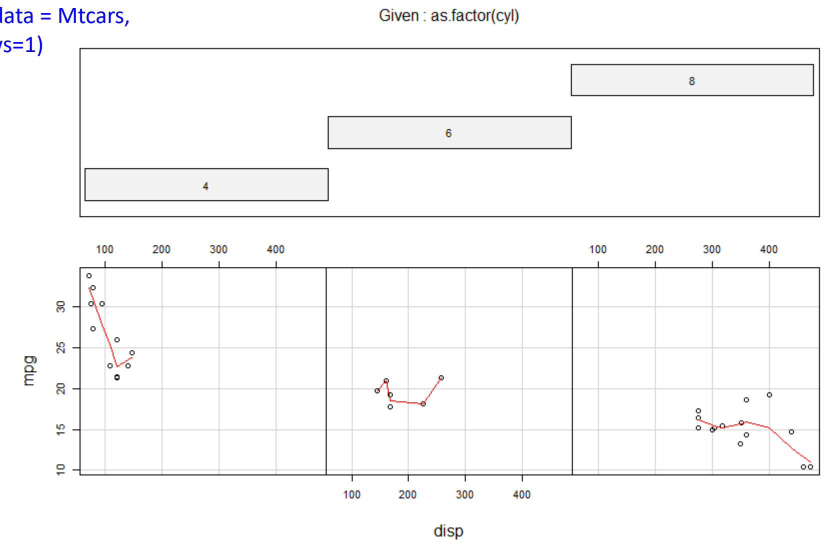
Basic Plotting in R

>#Plot all pairs of data against each other
>pairs(Mtcars, main= "mtcars data", gap = 1/4)



Basic Plotting in R

```
>coplot(mpg ~ disp | as.factor(cyl), data = Mtcars,
>          panel = panel.smooth, rows=1)
```



Basic Plotting in R

```
par(mgp=c(2,1,0), mar=c(3,3,1,1))
```

par() – plotting parameters
mgp – Sets axis label locations
Mar – Sets margins for graph

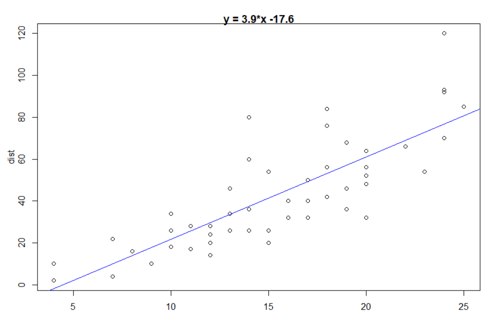
```
# Fit regression line
require(stats)
reg<-lm(dist ~ speed, data = cars)
coeff=coefficients(reg)

# equation of the line :
eq = paste0("y = ", round(coeff[2],1), "*x ", round(coeff[1],1))

# plot
plot(cars, main=eq)
abline(reg, col="blue")
```

Assumes (x, y) in data set

Works like trend line in Excel, but model has to be defined first.



Basic Plotting in R

abline() : Add Straight Lines To A Plot

```
abline(a = NULL, b = NULL, h = NULL, v = NULL,
reg = NULL, coef = NULL, untf = FALSE, ...)
```

- a, b: the intercept and slope, single values.
- untf: logical asking whether to untransform.
- h: the y-value(s) for horizontal line(s).
- v: the x-value(s) for vertical line(s).
- coef: a vector of length two giving the intercept and slope.
- reg: an object with a coef method. See ‘Details’.

We can add any arbitrary lines using this function.

117

Basic Plotting in R

Summary of Plotting Commands:

1. barchart()
2. hist()
3. pie()
4. boxplot()
5. plot()
6. coplot
7. pairs()
8. abline()

118

Basic Plotting in R

Homework

Assignment #1: Use the R dataset mtcars and write an R script to do the following:

a) Assign the data set to the variable mydata
b) Perform the following on the dataset:
    1. head
    2. tail
    3. Summary
c) Create a histogram of mpg(miles per gallon) in mtcars, use hist() (hint: use the '$' sign to access mpg)
d) Create a histogram of mpg in mtcars with more breaks. (hint: see the available options for the hist() function with f1 or ?hist(). hint: 'breaks =').
e) Create a boxplot of mpg in mtcars, use boxplot(). (To make a boxplot for each number of cylinders use: boxplot(mtcars$mpg ~ mtcars$cyl))
f) Create a scatterplot of mpg vs. cyl in mtcars using plot()
g) Plot horsepower (hp) versus mpg using plot (hint: plot(x, y)) and fit a smooth line (hint: abline (lm(y~x)))

Basic Plotting in R

Homework

Assignment #2: Use the R dataset iris for these exercises

a) Perform the following on the dataset:
    1. head
    2. tail
    3. Summary
b) Get all rows of *Species* 'versicolor' in a new data frame. Call this data frame: 'iris.vers'
c) Create a vector called 'sepal.dif' with the difference between 'Sepal.Length' and 'Sepal.Width'of 'versicolor' plants.
d) Update (add) 'iris.vers' with the new column 'sepal.dif'.
e) Filter for all data of *Species* 'virginica' with a 'Sepal.Width' of greater than 3.5. Store the results in a dataframe called 'iris.filtered'
f) Get a new object which contains only the odd values of 'Sepal.Length. Store the results in a dataframe called 'iris.odd'
g) Calculate mean of each of the numeric variables