R: WEEK 1

Gates

INSTRUCTIONS

As you review the basics of R and this Guide..

- Always type in, run, and **practice** with each example.
- Determine if you need more review.
- If you do, use the readings and the Internet to continue to review.

TOPICS IN THIS GUIDE

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
   - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: F-test, z-test, t-test, CI, ANOVA, IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: F-test, z-test, t-test, CI, ANOVA, IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

INSTALLING R

You can find a step by step method for this process here:

http://drgates.georgetown.domains/ANLY500/GetRandRstudio.pdf

You can also use the Internet as a resource if you prefer.

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.

7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

VIEW YOUR CURRENT PACKAGES

```
## View all of the packages you already have installed
AllPackages <- installed.packages()
AllPackages
```

INSTALLING A PACKAGE: METHOD 1

##To install a package, there are options.

##The easiest/fastest is to use the command line in RStudio

and **install.packages("party")**

##**The to include the package you can use**

library(party)

##NOTE: PARTY package in R is used for recursive

##partitioning and this package reflects the continuous

##development of ensemble methods.

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

OPERATORS

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*****	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

DECISIONS: IF/IF ELSE/ELSE

1) **The "if"**

```
if(test condition){
    statement(s)
}
```

2) **The "if/else"**

```
if (test_expression) {
   statement1
} else {
   statement2
}
```

```
if ( test_expression1) {
   statement1
} else if ( test_expression2) {
   statement2
} else if ( test_expression3) {
   statement3
} else
   statement4
```

EXAMPLE CODE

```
## Decision
name="John"
val=34
if(name=="John" & val==34){
  cat(name,val)
}
if(val < 50 | val >100){
  cat(val)
}
```

```
grade = 89

if(grade <59){
  cat("The grade is an F and is", grade)
} else if( grade <= 69){
  cat("The grade is an D and is", grade)
} else if( grade <= 79){
  cat("The grade is an C and is", grade)
} else if (grade <=89){
  cat("The grade is an B and is", grade)
} else {
  cat("The grade is an A and is", grade)
}
```

LOOPS: THE FOR LOOP

```
for (year in c(2010, 2011, 2012, 2013)){
  cat("The year is ", year, "\n")
}
for(i in 1:10){
  cat(i, " ")
}
Mylist<-c("john", "sally", "fred")
for(name in Mylist){
  cat(name," ")
}
```

```
rows=5
cols=4

for (r in 1: rows){
  for(c in 1:cols){
    cat(c)
  }
  cat("\n",r)
}
```

LOOPS:WHILE

```
MyList2 <- c(1,2,3,4,5,6,7,8)
while(length(MyList2 >= 0)){
  cat("The next element is", MyList2[1], "\n")
  ##pop off the first element
  MyList2 <- MyList2[-1]
}
```

```
##while loop
count=0

while(count < 5){
  if(count == 3){
    cat("we are at 3")
  }
  count=count+1
}
```

FUNCTIONS

Functions are created using the **function**() directive and are stored as **R** objects just like anything else.

In particular, they are **R** objects of class "**function**".

The return value of a **function** is the last expression in the **function** body to be evaluated.

```
fun <- function (params){
      stuff….
}
```

RULES ABOUT FUNCTIONS IN R

1) Functions are first class objects – therefore they are treated as all other R objects.

2) Functions CAN be pass as arguments to other function.

3) Functions can be nested (you can define one function inside of another)

4) Function arguments can be missing and can have default values.

5) Function arguments can match be name or position.

6) Suggestion: do not make functions confusing. Keep arguments in order.

EXAMPLES

```
##function definition
fname <- function(param1, p2=7, p3, p4="John"){
  cat(param1, "\n")
  cat(p2, "\n")
  cat(p3, "\n")
  cat(p4, "\n")

}
```

```
##function call1
fname(param1=34,p3=5)
##function call2
fname(1, 2, 3, 4)
```

```
#Simple example
square <- function(x) {
  s <- x * x
  return(s)
}
square(9)
```

FOR DETAILED INFORMATION ON SCOPING

Please review this resource:

http://drgates.georgetown.domains/ANLY500/Readings/functionsandscope.pdf

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): **dataframes**
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

DATA AND R

1) R is magical with data.

2) It can also make you very frustrated.

3) R can read in most types of datasets.

4) We will look at the most common:

https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html

Methods:

scan()

read.table()

read.csv()

read.delim()

SCAN()

scan() is highly flexible and can be used to read: integers, numeric, complex numbers, logical values, characters, lists, or raw data that is separated.

Example syntax:

scan(file="filename" what=<type of data>, n=<max values to be read in>, sep="", dec=".", skip=0, na.strings="NA")

NOTES:

n is the maximum number of data values to be read in. The default is no limit. continued….

MORE ON SCAN()

sep: is the field separator character. If sep=“”, this is one or more white spaces or tabs.

dec: is the character used in the file for decimal points. In the US, this is “.”

skip: this is the number of lines in the file to skip before beginning to read values.

na.strings: This is a character vector. Its elements are to be interpreted as missing (NA) values. Blank fields can also be missing.

CODE EXAMPLES FOR SCAN

```
## Using scan() – scan space del text file
SData<-scan(file="NormalData.txt", sep="", what=numeric(),n=-1,
    dec=".", skip=0, na.strings = "NA")
head(SData)
# scan csv file
SData2<-scan(file="CancerCountyFIPS.csv", sep=",", what = character(),
          n=-1, dec=".", skip=0, na.strings = "NA")
head(SData2)
mode(SData2)  #mode gives the data type
## The result of this is character, not dataframe
SData2[4]  #This gives the 4th char string (index starts at 1) in the data
```

COMMENTS ABOUT SCAN

1) scan() reads the data into a vector or list of what ever mode (data type) was specified using the what= attribute of the function.

2) scan() does not read data into a dataframe.

READ.TABLE()

The **read.table()** function will read a file into **table format** and creates a **dataframe** from it.

syntax:

read.table(file, header=FALSE, sep=“,”, dec=“.”, row.names, col.names)

details:

header: this is a logical value (TRUE or FALSE), indicating whether the file contains the name of the variables as its first line.

READ.TABLE() CONTINUED

sep: the field separator character. Recall, if sep="", this means one or more white space characters (including spaces and tabs).

dec: the character used for decimal points. The default in the US is "."

row.names: a vector of row names

col.names: a vector of column names

READ.TABLE EXAMPLES

```
##read.table()

Ndata<-read.table("NormalData.txt", header=FALSE, sep="", col.names=c("G1", "G2", "G3",
"G4", "G5"))

head(Ndata)

Ndata["G1"] #shows all the data in the G1 col

CData<-read.table("CancerCountyFIPS.csv", header = TRUE, sep=",")

head(CData)

#This accesses CData row 1 and the col called "FIPS"

if(CData[1,"FIPS"]=="12125"){

  cat(CData[1,3])

}
```

READ.CSV() AND READ.CSV2()

read.csv() is used to read delimited files where the delimiter is a comma.

read.csv2() is used to read delimited files in other countries outside of the US where the comma is used as a decimal point and so the file is delimited (usually) with “;”.

We will focus only on read.csv().

However, they both work the same way.

READ.CSV()

```
csvdata<-read.csv("CancerCountyFIPS.csv", header=TRUE)
head(csvdata)

##Here, change the names of the columns
csvdata<-read.csv("CancerCountyFIPS.csv", header = TRUE,
                  col.names = c("county","fips","rate"))
head(csvdata)
```

READ.DELIM()

```
#The read.delim() is for reading tab delimited files.
csvdata<-read.csv("CancerCountyFIPS.csv", header=TRUE)
head(csvdata)
##Here, change the names of the columns
csvdata<-read.csv("CancerCountyFIPS.csv", header = TRUE,
                  col.names = c("county","fips","rate"))
head(csvdata)
##Note that read.delim2() is for non US files where they use , intsteach of .
```

FEW NOTES ABOUT READING IN DATA

1) There is also a package ("xlsx") that will allow you to directly read in Excel files.

- This package is for 3.3.2 or above.

2) Because I always save my Excel files as csv, I do not use it.

CREATING AND WRITING TO FILES IN R

```
#Create a list of column names
stuff<-c("Set1", "Set2", "Set3")
#ncolumns forces the number of columns to create
##append=FALSE will create a new file OR will delete an existing file
write(stuff, file = "NewRFile.csv", sep=",“ , ncolumns = 3, append = FALSE)
#Here we append this data as three columns rowwise
databunch<-c(1,3,5,7,9,11,13,15,17,19,21,23,25,27,29)
write(databunch, file = "NewRFile.csv", sep=",", ncolumns = 3,
    append = TRUE)
```

Delete Files and Directories

Description

`unlink` deletes the file(s) or directories specified by `x`.

Usage

```
unlink(x, recursive = FALSE)
```

Arguments

`x`	a character vector with the names of the file(s) or directories to be deleted. Wildcards (normally '`*`' and '`?`') are allowed.
`recursive`	logical. Should directories be deleted recursively?

Details

If `recursive = FALSE` directories are not deleted, not even empty ones.

file.remove can only remove files, but gives more detailed error information.

Value

The return value of the corresponding system command, `rm -f`, normally `0` for success, `1` for failure. Not deleting a non-existent file is not a failure.

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.

7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

BASIC DATA TYPES IN R

- The next several slides will show the basic data types in R.
- The slides will also show how to determine a data type.
- Finally, slides will show how to cast or convert from one data type to another.

LIST OF THE BASIC TYPES

1. Logical
2. Numeric
3. Integer
4. Complex
5. Character
6. Raw

LOGICAL, NUMERIC, INTEGERS

```
##Data Types
#Logical: TRUE, FALSE
(a<-TRUE)
class(a)
#Numeric: any real number
(a<- -45.7)
class(a)
```

```
#Integer: xL
a<-2
class(a) ##this will give numeric
b<-2L  ## this forces an integer
class(b)
if(b==2){
  cat(b, "\n")
  cat(class(b))
}
c<-34L
class(c)
```

AS.INTEGER AND IS.INTEGER

```
c<-34
#here c is numeric
class(c)
c<-as.integer(c)
res<-is.integer(c)
class(c)
```

```
c<-34.8
class(c)
c<-as.integer(c)
##truncates, does not
round
res<-is.integer(c)
class(c)
(c)
```

```
c<-"john"
class(c)
c<-as.integer(c)
res<-is.integer(c)
class(c)
```

COMPLEX

```
c <- 3 +4i
(c)
class(c)
```

CHARACTER

```
##character uses " " or ' '
a<-"jack"
var2<-"34.6"
class(var2)
name <-'Hello there'
quote <- "'To Be Or Not To Be'"
class(quote)
```

```
##cast var2 to a number
var2<-as.numeric(var2)
(var2)
class(var2)

##Raw
value2<-"Hello"
(charToRaw(value2))
##this produces:
##  48 65 6c 6c 6f
```

NOTES ABOUT DATA TYPES

1) R will assume the data type.

For example:

c <- 34 is assumed to be numeric (not int)

2) Anything in either paired single or paired double quotes is a character type.

3) Data types as by converted to other types.

For example:

as.integer will convert to integer.

4) The type can be checked using class()

5) A specific type can be checked using is.<type> such as is.integer. The result is TRUE or FALSE

DATA STRUCTURES

The following several slides will review many types of data structures in R.

Special attention will be given to dataframes.

Vectors

Lists

Matrices

Arrays

Factors

Data Frames

VECTORS: THE C() FUNCTION

1) Vectors are used throughout R programming and often.

2) They are a basic object that can hold elements of any type.

3) To create a vector of objects or elements, use c()

4) If you create a mixed vector – say strings and numbers – all elements will default to character type.

```r
##Vectors: the c()
(fruit <- c("apple", "banana", "pear"))
class(fruit)  ##returns character

(students <- c("John", 98, "Sally", 96))
class(students)
students[2]  ##returns the string 98
class(students[2])  ##returns char

(sizes <- c(34,67,12,78.9))
class(sizes) #returns numeric

(quant <-c(2L, 7L, 9L, 11L, 3L, 5L))
class(quant)  #returns integer
```

LISTS

1) Lists are more flexible than vectors. Vectors can contain any basic data element. However, lists can contain vectors, functions, and other lists as well.

2) Notice that I can store the sin function in the list and apply it via its list index.

3)To access a primary list element, use [[ ]].

4) To access an element within a list element, use [ ].

```r
##Lists can contain any elements and other lists
## and functions and vectors
List1 <- list(c("a","b"),3.14, "john", sin)
List1
List1[[1]][2]
List1[[2]]
List1[[4]](3.14)
```

247:1 Reading data R Scrip

Console ~/RStudioFolder/

```
[[1]]
[1] "a" "b"

[[2]]
[1] 3.14

[[3]]
[1] "john"

[[4]]
function (x)  .Primitive("sin")

> List1[[1]][2]
[1] "b"
> List1[[2]]
[1] 3.14
> List1[[4]](3.14)
[1] 0.001592653
>
```

MATRICES AND ARRAYS

1) Unlike lists and vectors, matrices and arrays must contain all the same data type.
2) Matrices must be only two dimensions.
3) Arrays can be any dimension.
4) The attribute dim = c(r, c, etc…) will specify.

```r
##Matrices and Arrays
##Must contain all same type

## Array can be any size
(M <- matrix( c('a','a','b','c','b','a'), nrow = 2,
              ncol = 3, byrow = TRUE))
(A <- array(c('green','yellow'),dim = c(2,4)))
```

255:17 Reading data R Script

Console ~/RStudioFolder/

```
[1,] "green"

> (M <- matrix( c('a','a','b','c','b','a'), nrow = 2,
+               ncol = 3, byrow = TRUE))
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
> (A <- array(c('green','yellow'),dim = c(2,4)))
     [,1]     [,2]     [,3]     [,4]
[1,] "green"  "green"  "green"  "green"
[2,] "yellow" "yellow" "yellow" "yellow"
>
```

FACTORS

Factors are the r-objects which are created using a **vector.**

A factor stores the vector AND the distinct values of the elements in the vector as **labels.**

The labels are **always character** irrespective of whether it is numeric or character or Boolean etc. in the input vector.

They are useful in statistical modeling.

Factors are created using the **factor()** function.

The **nlevels** functions gives the count of levels.

Examples…

```r
##Factors
##Two parts - the vector and the labels
(car_colors <- c("blue", "blue", "black", "silver","white", "black"))
(factor_car <- factor(car_colors))
(nlevels(factor_car))
table(factor_car)
```

271:1 Reading data R Script

Console ~/RStudioFolder/

```
> ##Factors
> ##Two parts - the vector and the labels
> (car_colors <- c("blue", "blue", "black", "silver","white", "black"))
[1] "blue"   "blue"   "black"  "silver" "white"  "black"
> (factor_car <- factor(car_colors))
[1] blue   blue   black  silver white  black
Levels: black blue silver white
> (nlevels(factor_car))
[1] 4
> table(factor_car)
factor_car
 black   blue silver  white
     2      2      1      1
>
```

https://www.stat.berkeley.edu/classes/s133/factors.html

DATA FRAMES

1) The next few slides will offer examples of dataframe methods and options.

2) A dataframe, like a matrix, is two-dimensional.

3) However, unlike a matrix, a dataframe does not have to contain all the same type of data. A data frame can have each column (or row) set to contain a different data type.

4) Dataframes are much like tables in a database.

5) A dataframe contains rows and columns.

6) Conventionally, each column can be thought of as a variable, attribute, or field in a dataset, while each row is one entity or object in the database.

Example…

DATAFRAME EXAMPLE

1) In this case, this dataframe has three columns.

2) Each column is a variable or attribute and is named.

3) This dataframe has 6 rows. Each row is one sample of each variable.

4) In row 1, the county is Union County, Florida(6,10), the fips is 12125, and the rate is 982.6.

```
                           county  fips  rate
1        Union County, Florida(6,10) 12125 982.6
2       Harding County, New Mexico(7) 35021 721.0
3    Holmes County, Mississippi(6,10) 28051 697.2
4        Logan County, Nebraska(6,10) 31113 642.5
5 Yalobusha County, Mississippi(6,10) 28161 636.3
6          Dallam County, Texas(6,10) 48111 626.1
>
```

BUILDING A BASIC DATAFRAME

1) Create vectors of data.

2) The name of the vector will be the name of the column

```r

###################
###Dataframes
Name_set=c("Jack","Saly","Jan")
Runs=c(6,7,8)
Score=c(47,34.7,89.1)
df <- data.frame(Name_set, Runs, Score)
df

```

271:1 Reading data

Console ~/RStudioFolder/

```
> Name_set=c("Jack","Saly","Jan")
> Runs=c(6,7,8)
> Score=c(47,34.7,89.1)
> df <- data.frame(Name_set, Runs, Score)
> df
  Name_set Runs Score
1     Jack    6  47.0
2     Saly    7  34.7
3      Jan    8  89.1
> 
```

INDEXING A DATAFRAME

1) You can access a column by name.

2) You can access a column by numerical location (R indexing starts at 1 not 0)

3) You can access a column and then a row using the [[ ]] [] - like with matrices. However, this is not the best way.

More examples…next slide

```
Name_set=c("Jack","Saly","Jan")
Runs=c(6,7,8)
Score=c(47,34.7,89.1)
df <- data.frame(Name_set, Runs, Score)
df["Name_set"]
df[3]
df[[1]][2]
```

277:40 Reading data

Console ~/RStudioFolder/

```
> df["Name_set"]
  Name_set
1     Jack
2     Saly
3      Jan
> df[3]
  Score
1  47.0
2  34.7
3  89.1
> df[[1]][2]
[1] Saly
Levels: Jack Jan Saly
>
```

```r
###Dataframes
Name_set=c("Jack","Saly","Jan", "Ben", "Fred")
Runs=c(6,7,8, 3, 10)
Score=c(47,34.7,89.1, 88.2, 56)
df <- data.frame(Name_set, Runs, Score)
df["Name_set"]
df[3]
df[1,2]
df[3,2:3]
df[2:3,1]
df[-1]
```

Console ~/RStudioFolder/

```
> df["Name_set"]
  Name_set
1     Jack
2     Saly
3      Jan
4      Ben
5     Fred
> df[3]
  Score
1  47.0
2  34.7
3  89.1
4  88.2
5  56.0
> df[1,2]
[1] 6
> df[3,2:3]
  Runs Score
3    8  89.1
> df[2:3,1]
[1] Saly Jan
Levels: Ben Fred Jack Jan Saly
> df[-1]
  Runs Score
1    6  47.0
2    7  34.7
3    8  89.1
4    3  88.2
5   10  56.0
> 
```

The key elements to note are that you can access rows and columns.
df[rows, columns]

COMPARISON WITH BLANK INDEXING

1) Here, you can see that df[,3] is all rows and only column 3.
2) Similarly, df[3,] is row 3 and all columns.

```
df
df[,3]
df[3,]
```

280:8 Reading data

Console ~/RStudioFolder/

```
> df
  Name_set Runs Score
1     Jack    6  47.0
2     Saly    7  34.7
3      Jan    8  89.1
4      Ben    3  88.2
5     Fred   10  56.0
> df[,3]
[1] 47.0 34.7 89.1 88.2 56.0
> df[3,]
  Name_set Runs Score
3      Jan    8  89.1
> 
```

USING R BUILT-IN DATA

1) install datasets
2) To use a package, use: library(package)

3) Here, we have library(datasets)

4) This gives access to R built-in datasets.

5) Now we have many dataframe options.

```
#########
#Using R built-in data
##install.packages("datasets")
##List of R datasets are here:
#https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html
library(datasets)
CarsData <- mtcars
head(CarsData)
```

293:74 Reading data

Console ~/RStudioFolder/

```
> library(datasets)
> CarsData <- mtcars
> head(CarsData)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 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 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> 
```

USING VECTORS TO ACCESS PORTIONS OF DATAFRAMES

1) **This example shows that you can use exact names to access rows or columns..**
2) **You can also use vectors to access portions of dataframes.**
3) **CarsData[c(2,5,7), c(2,3,4,5)] will access rows 2, 5, and 7 and from those rows columns 2, 3, 4, and 5.**

```
297  CarsData["Valiant",]
298  CarsData[c(2,5,7), c(2,3,4,5)]
299
```

295:19   Reading data

Console ~/RStudioFolder/

```
> CarsData["Valiant",]
         mpg cyl disp  hp drat   wt  qsec vs am gear carb
Valiant 18.1   6  225 105 2.76 3.46 20.22  1  0    3    1
> CarsData[c(2,5,7), c(2,3,4,5)]
                  cyl disp  hp drat
Mazda RX4 Wag       6  160 110 3.90
Hornet Sportabout   8  360 175 3.15
Duster 360          8  360 245 3.21
> 
```

METHODS TO CREATE DATAFRAMES

1) You can read in data into a dataframe. This was covered in the data files section above.

2) You can build a dataframe by hand. To do this, create and name vectors and use data.frame to place them together in a dataframe. This was covered earlier.

3) You can use an R built-in dataset which loads as a dataframe. This was covered earlier.

4) You can convert to a dataframe from other structures. (as.data.frame()).

NOTE: Week 2 of the class will begin with dataframes and will review further dataframe methods.

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

BASIC MATH AND STATS IN R

TOPICS

summary stats/IQR

z-test, t-test, pvalues

ANOVA

SUMMARY STATS

NOTE: IQR is Q3 – Q1

```r
####Summary stats
summary(mtcars$mpg)
```

333:18 Reading data

Console ~/RStudioFolder/

```
> summary(mtcars$mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  10.40   15.42   19.20   20.09   22.80   33.90 
> 
```

HYPOTHESIS TESTING IN R: ONE SAMPLE Z TEST IN R

```
###one sample z test….function: a is the data, mu is the pop mean
z.test = function(a, mu, var){
  zeta = (mean(a) - mu) / (sqrt(var / length(a)))
  return(zeta)
}
a = c(65, 78, 88, 55, 48, 95, 66, 57, 79, 81)
#Compare dataset a to mu 75 with var 18
z.test(a, 75, 18)
#The result is z=-2.83  - At alpha =.05 for two-tailed, the rejection region
# is +/- 1.96 so here reject Ho there is a sig diff
```

EXAMPLE 2: Z TEST

```
cdata <- mtcars
head(cdata)
mpgdata <- cdata$mpg

hist(mpgdata) #histogram

#POPULATION PARAMETER CALCULATIONS
(sample_sd <- sd(mpgdata))
#*sqrt((length(mpgdata)-1)/(length(mpgdata)))
(sample_mean <- mean(mpgdata))
(zstat <- (sample_mean - 32) / sample_sd) ##This is the ztest stat
(pval <- pnorm(zstat,0,1))  #This is the p value for the zstat
```

T-TEST EXAMPLE

```r
####The t-test
y <- mtcars$mpg
x <- mtcars$vs  ##can be 0 or 1
t.test(y~x) # where y is numeric and x is a binary factor
```

327:1 Reading data

Console ~/RStudioFolder/

```
        Welch Two Sample t-test

data:  y by x
t = -4.6671, df = 22.716, p-value = 0.0001098
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.462508  -4.418445
sample estimates:
mean in group 0 mean in group 1
       16.61667        24.55714
```

The ANOVA Test and Boxplots

```r
##The ANOVA
# One Way Anova (Completely Randomized Design)
response <- mtcars$mpg
factors <- mtcars$cyl
boxplot(response ~ factors, data=mtcars)
##ANOVA
results=aov(response ~ factors, data=mtcars)
summary(results) ##F is 79.56 a very sig result
```

343:48 Reading data R Script

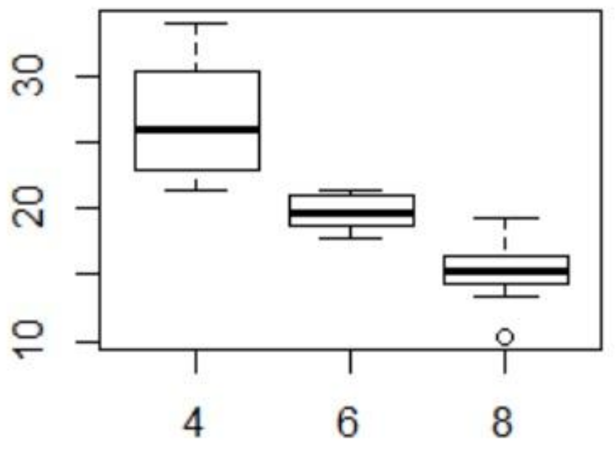
Console ~/RStudioFolder/

```
> response <- mtcars$mpg
> factors <- mtcars$cyl
> boxplot(response ~ factors, data=mtcars)
> ##ANOVA
> results=aov(response ~ factors, data=mtcars)
> summary(results)
            Df Sum Sq Mean Sq F value   Pr(>F)
factors      1  817.7   817.7   79.56 6.11e-10 ***
Residuals   30  308.3    10.3
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

```
response   num [1:32] 21 21 22.8 2...
results    List of 12
```

Files Plots Packages Help Viewer

Zoom Export Publi



BASIC MATH IN R

Operator	Description	Example
x + y	y added to x	2 + 3 = 5
x – y	y subtracted from x	8 – 2 = 6
x * y	x multiplied by y	3 * 2 = 6
x / y	x divided by y	10 / 5 = 2
x ^ y (or x ** y)	x raised to the power y	2 ^ 5 = 32
x %% y	remainder of x divided by y (x mod y)	7 %% 3 = 1
x %/% y	x divided by y but rounded down (integer divide)	7 %/% 3 = 2

NEXT TOPIC

1. Installing R and RStudio and setting up the IDE and coding environment. Complete "Hello World".
2. Installing R libraries.
3. Basic coding operations in R: decisions, loops, functions (User-defined functions, parameters, return structures, and scoping)
4. Working with files (creating, reading and writing) in R.
 - Reading data into R (csv, Excel, txt): dataframes
5. Commonly used R data structures: lists, vectors, matrices, dataframes, logical, strings, and factors.

6. Basic math and stats in R: z-test, t-test, ANOVA, summary/IQR, p values.
7. Basic Plotting and Graphing in R: Bar, Histograms, Boxplots, Scatter.

BASIC PLOTTING TOPICS

Basic Graphing in R:

Bar,

Histograms,

Boxplots,

Scatter,

BAR

```
##Bar
table(mtcars$cyl)
counts <- table(mtcars$cyl)
barplot(counts, main="Car Cylinders",
        xlab="Number of Cylinders")
```

346:16 Reading data R Script

Console ~/RStudioFolder/

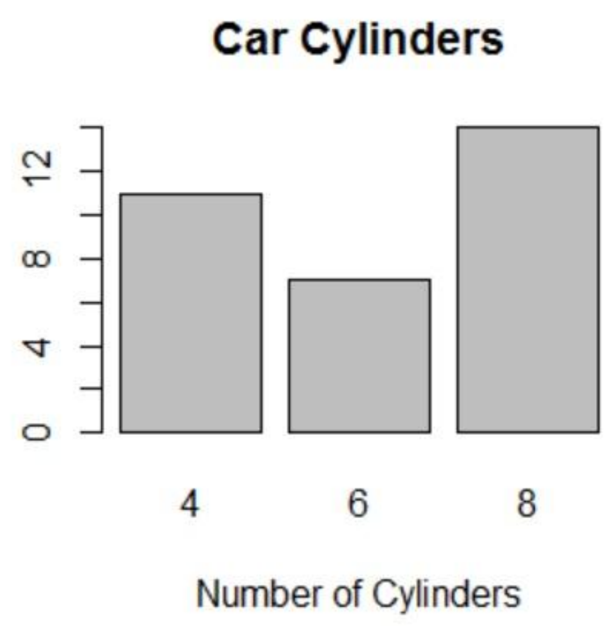
```
> table(mtcars$cyl)

 4  6  8 
11  7 14 
> table(mtcars$cyl)

 4  6  8 
11  7 14 
> counts <- table(mtcars$cyl)
> barplot(counts, main="Car Cylinders",
+         xlab="Number of Cylinders")
> 
```

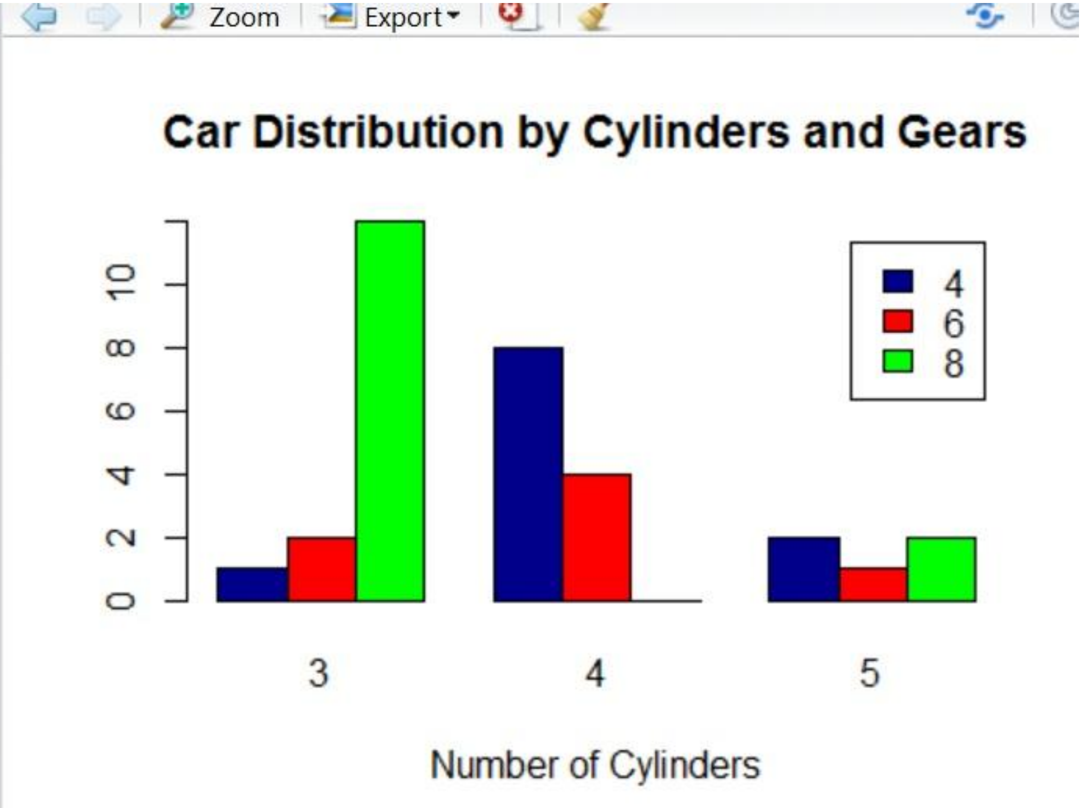
Files Plots Packages Help Viewer

Zoom Export



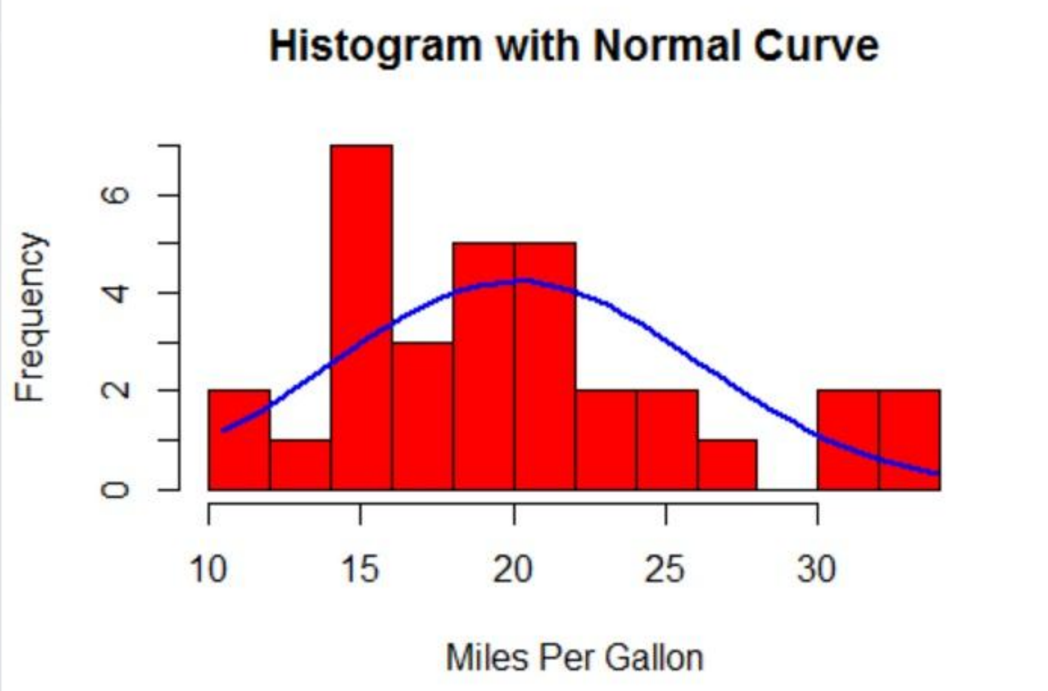
GROUPED BAR

```
##Grouped Bar
head(mtcars)
counts <- table(mtcars$cyl, mtcars$gear)
barplot(counts, main="Car Distribution by Cylinders and Gears",
        xlab="Number of Cylinders", col=c("darkblue","red", "green"
        legend = rownames(counts), beside=TRUE)
```



HISTOGRAM PLUS NORMAL LINE

```r
###Histogram with normal line
x <- mtcars$mpg
h<-hist(x, breaks=10, col="red", xlab="Miles Per Gallon",
        main="Histogram with Normal Curve")
xfit<-seq(min(x),max(x),length=40)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
##RE: https://www.r-bloggers.com/normal-distribution-functions/
```

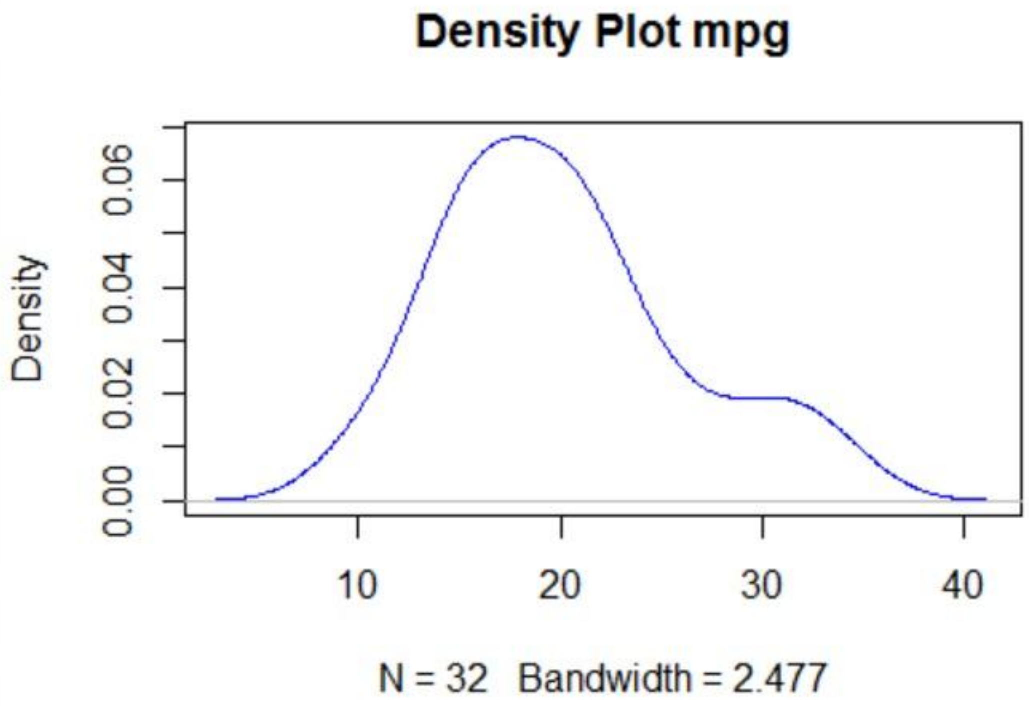


DENSITY PLOT

```r
## Density Plot
d <- density(mtcars$mpg) # returns the density data
plot(d, col="blue", main="Density Plot mpg") # plots the results
```

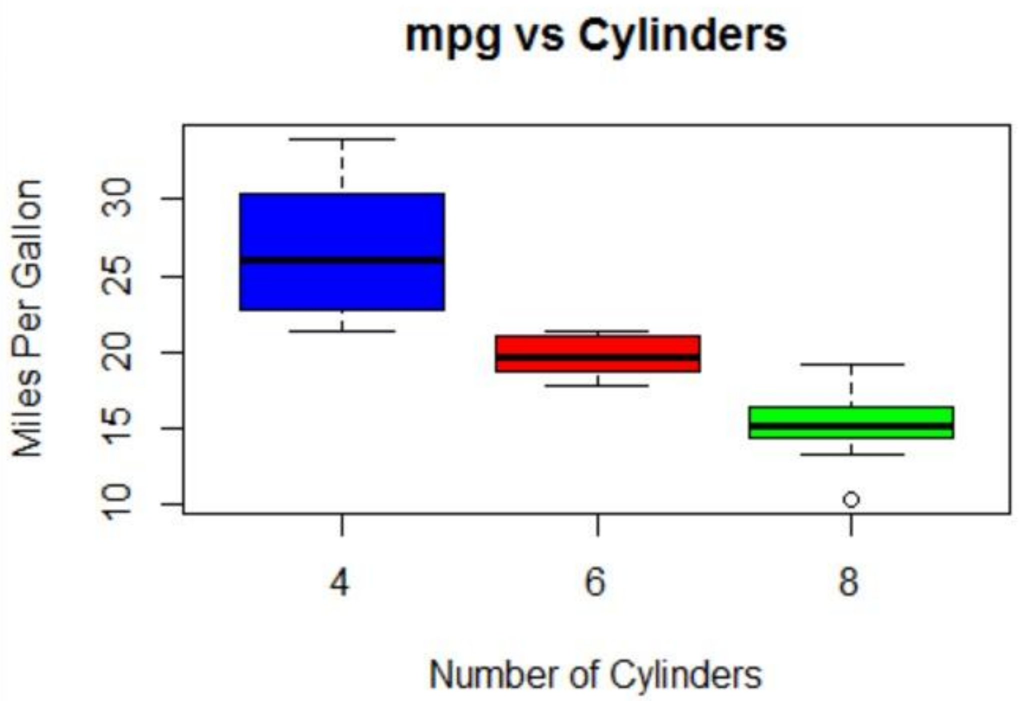
Reading data   R Script

~/RStudioFolder/



BOXPLOTS

```
##Boxplots
boxplot(mpg~cyl,data=mtcars, main="mpg vs Cylinders",
        xlab="Number of Cylinders", ylab="Miles Per Gallon",
        col=c("blue","red", "green"))
```



SCATTERPLOT WITH REGRESSION LINE

##Scatterplots

##Single

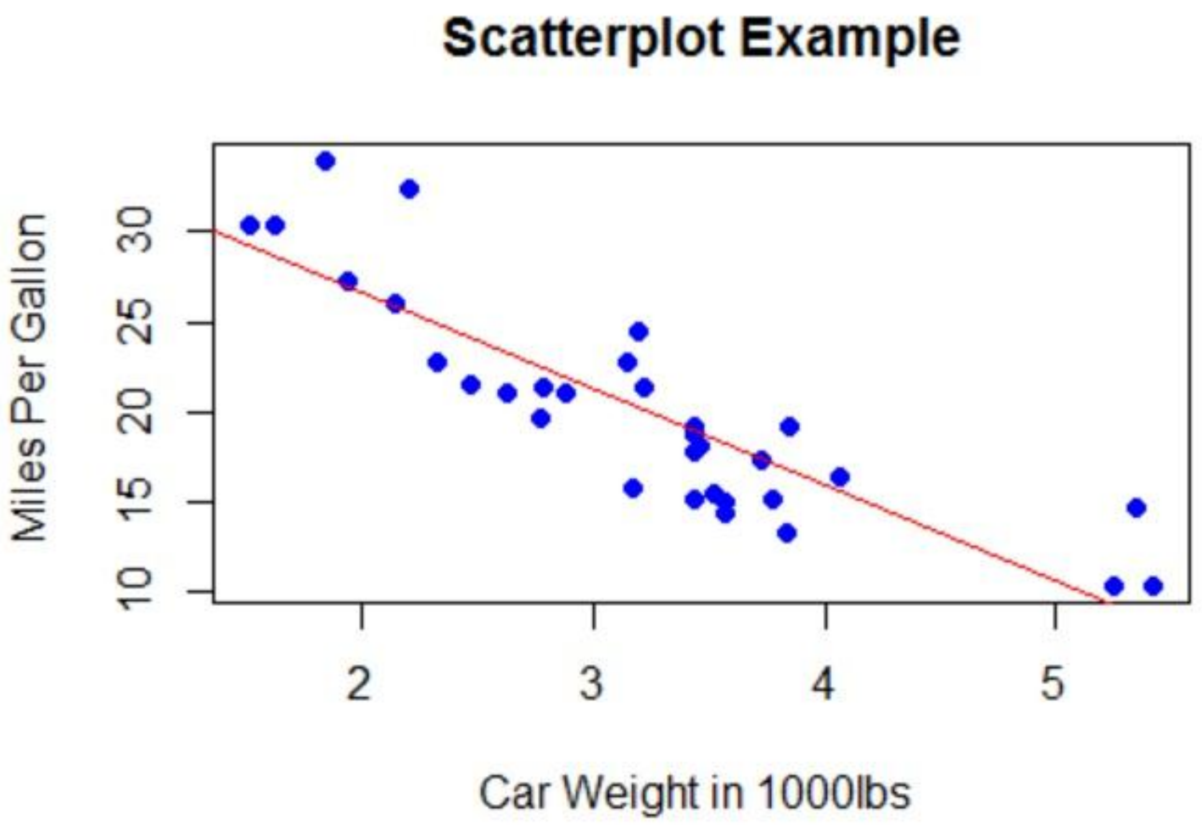
plot(mtcars$wt, mtcars$mpg, main="Scatterplot Example",

xlab="Car Weight in 1000lbs ", ylab="Miles Per Gallon ", pch=19,

col="blue")

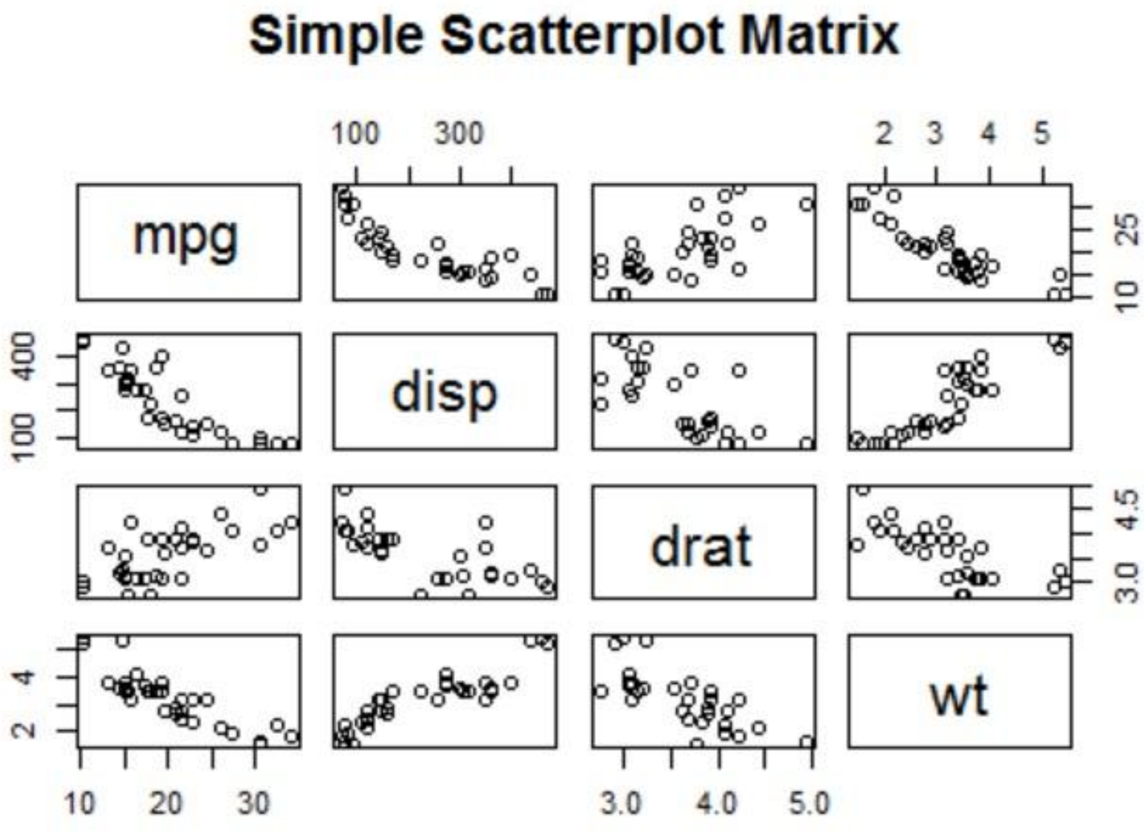
Add fit lines

abline(lm(mtcars$mpg~mtcars$wt), col="red") # regression line (y~x)



MATRIX OF SCATTERS

```
pairs(~mpg+disp+drat+wt,data=mtcars,
    main="Simple Scatterplot Matrix")
```



SUMMARY

This Guide covers the basics of R.

No Guide can cover all R topics.

Use the Internet or any book of your choice to fill in the blanks and to practice.