R Nano Course
Review of R

Anusha Nagari & Tulip Nandu
Green Center Computational Core

Outline

- ✓ ***Basics of R***
- ✓ ***Rstudio***
- ✓ ***Simple commands in R***
- ✓ ***Data Types & Data Structures in R***
- ✓ ***Basic Plotting Parameters***
- ✓ ***Basic Plots***
- ✓ ***Statistical Tests***

Basics of "R"

- Powerful programming language and environment for statistical computing
- Useful for very basic analysis as well as for hard core programming
- Very easy to make publication quality figures
- Run through command line or GUI
- Versions of R exist of Windows, MacOS, Linux and various other Unix flavors

Advantages:

- ✓ Free and open source
- ✓ >2000 packages for handling biological data
- ✓ Widely used, large user community (blogs like Seqanswers, StackOverflow…)
- ✓ Extensive documentation with examples
- ✓ Rstudio, a very user-friendly interface

And it is fun!

R is a tool for…

Data Manipulation

- Connecting to data sources
- Slicing & dicing data

Modeling & Computation

- Statistical Modeling
- Numeric Simulation

Data Visualization

- Visualizing fit of models
- Composing statistical graphics

Learning 'R'

- Check out the course wikisite - lots of good manuals & links
- Read through the CRAN website
- Use http://www.rseek.org/ instead of google
- Know your objects' classes: class(x) or info(x)
- Because R is interactive, errors are your friends!
- ?lm gives you help on lm function. Reading help files can be very… helpful
- MOST IMPORTANT - the more time you spend using R, the more comfortable you become with it. After doing your first real project in R, you won't look back. I promise.

RStudio

RStudio

boxplot.R

Source on Save | Run | Source

```
1  # Go to current working directory:
2  setwd("~/Desktop/R_tutorial")
3
4  ########################################
5  # Boxplot
6  ########################################
7  # Read the data
8  data <- read.table("boxplot.txt",header=T)
9
10 # Display first few lines of the data
11 head(data)
12
13 # Distribution/Statistics of the data
14 summary(data)
15
16 # Draw boxplot
17 boxplot(data$VEH,data$E2)
18 boxplot(log(data$VEH),log(data$E2))
19
```

27:53 (Top Level) R Script

Console ~/Desktop/R_tutorial/

```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> # Go to current working directory:
> setwd("~/Desktop/R_tutorial")
>
> # Read the data
> data <- read.table("boxplot.txt")
>
```

Workspace | History

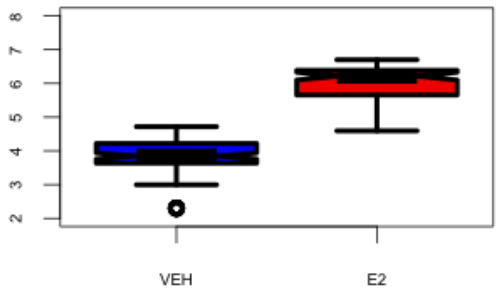
Load | Save | Import Dataset | Clear All

Data

data	100 obs. of 2 variables

Files | Plots | Packages | Help

Zoom | Export | Clear All



Navigating with in the R enviroment

List all the variables:

```
ls()
```

Examining a variable 'x'

```
dim()
head()
tail()
class()
names()
```

Removing variables

```
rm()
rm(list=ls()) # remove everything
```

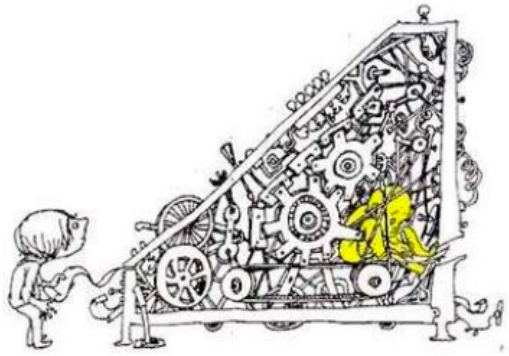
Install packages in R enviroment

```
install.packages()
```

R is an "overgrown calculator"

Simple math:

```
1 + 1 # Simple Arithmetic
[1] 2
2 + 3 * 4 # Operator precedence
[1] 14
3 ^ 2 # Exponentiation
[1] 9
sqrt(10) #Basic mathematical functions are available
[1] 3.162278
```



Storing results in variables:

```
x <- 1 # Can define variables
y <- 3 # using "<-" operator to set values
z <- 4
x * y * z
[1] 12

This.Year <- 2004 # Variable names can include period
This.Year
[1] 2004
```

```
sum(rgamma(rpois(1,lambda=2),shape=49,scale=.2))
```

Data Types & Data Strutures

To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call.

R Data Types:

Example	Type
"a", "swc"	character
2, 15.5	numeric
2 (Must add a `L` at end to denote integer)	integer
`TRUE`, `FALSE`	logical
1+4i	complex

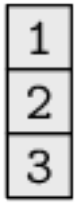
Data Structures:

- Vectors
- List
- Matrix
- Data frame
- Factors

Data Strutures

Vectors:

Collection of values of the same data type




```
# Numeric Vector:
x1 <- c(1, 2, 3)
x1
[1] 1 2 3

length(x1) # Check the length
[1] 3
class(x1)
[1] "numeric"

# Integer Vector:
x2 <- c(1L, 2L, 3L)

# Logical Vector:
x3 <-  c(TRUE, TRUE, FALSE, FALSE)
class(x3)
[1] "logical"
```

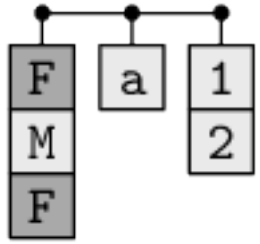
```
# Character Vector:
x4 <- c("Alec", "Dan", "Rob", "Karthik")
class(x4)
[1] "character"
x4
[1] "Alec"    "Dan"     "Rob"     "Karthik"

# Add elements
x4 <- c(x4, "Annette")
x4
[1] "Alec"    "Dan"     "Rob"     "Karthik"
"Annette"
```

Data Strutures

Lists

Collection of data structures

```
y <- list(1, "a", TRUE, 1 + (0 + 4i))
y
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i

length(y)
[1] 4
```



Factors:

A collection of values that all come from a fixed set of possible values

Data Strutures

Matrix:

A two-dimensional collection of values that all have the same type. The values are arranged in rows and columns


```
x <- 1: 3
[1] 1 2 3
y <- 10: 12
[1] 10 11 12

m2 <- cbind(x, y)
m3 <- rbind(x, y)
m2
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12

m3
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
# Add the row and column names:
dimnames(m3) <- list(c("a", "b"), c("c", "d", "e"))
m3
   c  d  e
a  1  2  3
b 10 11 12
```

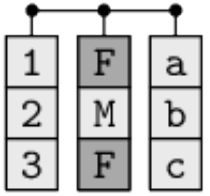
#cbind: Combine R objects by columns
#rbind: Combine R objects by rows

* Arrays are similar to matrices but can have more than two dimensions

Data Strutures

Data frame:

A collection of vectors that all have the same length. This is like a matrix, except that each column can contain a different data type.




```
df <- data.frame(id = letters[1: 10], x = 1: 10, y = 21: 30)

dim(df)
[1] 10  3

head(df)
  id x  y
1  a 1 21
2  b 2 22
3  c 3 23
4  d 4 24
5  e 5 25
6  f 6 26

# Add an additional "name" column to the dataframe
df$name <- paste("id", 1: 10, sep = "")
head(df)
  id x  y name
1  a 1 21  id1
2  b 2 22  id2
3  c 3 23  id3

colnames(df)
[1] "id"   "x"    "y"    "name"

# List current objects in the workspace
ls()
 [1] "df" "m1" "m2" "m3" "x"  "x1" "x2" "x3" "x4" "y"
```

```
# Change the order of the columns:
df <- df[, c(4, 1, 2, 3)]
head(df)
  name id  x  y
1   id1  a  1 21
2   id2  b  2 22
3   id3  c  3 23

#Remove a selected column
df <- subset(df, select = -id)
head(df)
   name  x  y
1   id1  1 21
2   id2  2 22
3   id3  3 23

# Change the column names
colnames(df) <- c("col1", "col2", "col3" )
head(df)
   col1 col2 col3
1  id1    1   21
2  id2    2   22
3  id3    3   23


# Delete an object
rm(df)
```

Basic Plotting

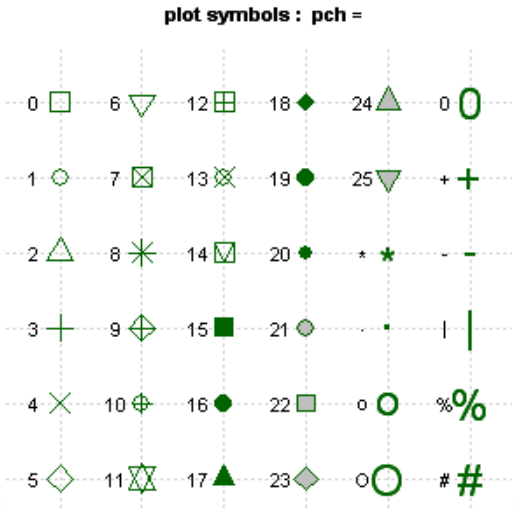
Graphical Parameters

Text and Symbol Size

option	description
cex	number indicating the amount by which plotting text and symbols should be scaled relative to the default. 1=default, 1.5 is 50% larger, 0.5 is 50% smaller, etc.
cex.axis	magnification of axis annotation relative to cex
cex.lab	magnification of x and y labels relative to cex
cex.main	magnification of titles relative to cex
cex.sub	magnification of subtitles relative to cex

Plotting Symbols

Use **pch=** option to specify symbols to use when plotting points.
For symbols 21 through 25, specify border color (col=) and fill color (bg=).

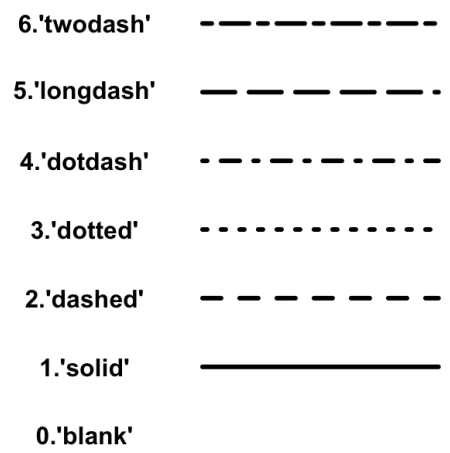


Graphical Parameters

Lines

You can change lines using the following options. This is particularly useful for reference lines, axes, and fit lines.

option	description
lty	line type. see the chart below.
lwd	line width relative to the default (default=1). 2 is twice as wide.



Margins and Graph Size

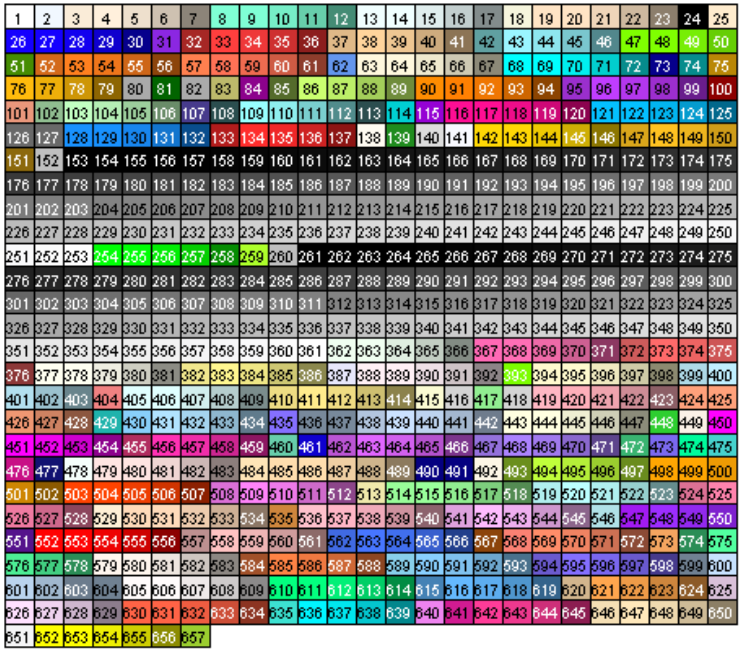
option	description
mar	numerical vector indicating margin size c(bottom, left, top, right) in lines. default = c(5, 4, 4, 2) + 0.1
mai	numerical vector indicating margin size c(bottom, left, top, right) in inches
pin	plot dimensions (width, height) in inches

Graphical Parameters

Colors

option	description
col	Default plotting color. Some functions (e.g. lines) accept a vector of values that are recycled.
col.axis	color for axis annotation
col.lab	color for x and y labels
col.main	color for titles
col.sub	color for subtitles
fg	plot foreground color (axes, boxes - also sets col= to same)
bg	plot background color

For example **col=1**, **col="white"**, & **col="#FFFFFF"** are equivalent



1	white	#FFFFFF	255	255	255
2	aliceblue	#F0F8FF	240	248	255
3	antiquewhite	#FAEBD7	250	235	215
4	antiquewhite1	#FFEFDB	255	239	219
5	antiquewhite2	#EEDFCC	238	223	204
6	antiquewhite3	#CDC0B0	205	192	176
7	antiquewhite4	#8B8378	139	131	120
8	aquamarine	#7FFFD4	127	255	212
9	aquamarine1	#7FFFD4	127	255	212
10	aquamarine2	#76EEC6	118	238	198
11	aquamarine3	#66CDAA	102	205	170
12	aquamarine4	#458B74	69	139	116
13	azure	#F0FFFF	240	255	255
14	azure1	#F0FFFF	240	255	255
15	azure2	#E0EEEE	224	238	238
16	azure3	#C1CDCD	193	205	205
17	azure4	#838B8B	131	139	139
18	beige	#F5F5DC	245	245	220
19	bisque	#FFE4C4	255	228	196
20	bisque1	#FFE4C4	255	228	196
21	bisque2	#EED5B7	238	213	183
22	bisque3	#CDB79E	205	183	158
23	bisque4	#8B7D6B	139	125	107
24	black	#000000	0	0	0
25	blanchedalmond	#FFEBCD	255	235	205
26	blue	#0000FF	0	0	255
27	blue1	#0000FF	0	0	255
28	blue2	#0000EE	0	0	238
29	blue3	#0000CD	0	0	205
30	blue4	#00008B	0	0	139
31	blueviolet	#8A2BE2	138	43	226
32	brown	#A52A2A	165	42	42
33	brown1	#FF4040	255	64	64
34	brown2	#EE3B3B	238	59	59
35	brown3	#CD3333	205	51	51
36	brown4	#8B2323	139	35	35
37	burlywood	#DEB887	222	184	135
38	burlywood1	#FFD39B	255	211	155
39	burlywood2	#EEC591	238	197	145
40	burlywood3	#CDAA7D	205	170	125
41	burlywood4	#8B7355	139	115	85
42	cadetblue	#5F9EA0	95	158	160
43	cadetblue1	#98F5FF	152	245	255
44	cadetblue2	#8EE5EE	142	229	238
45	cadetblue3	#7AC5CD	122	197	205
46	cadetblue4	#53869B	83	134	139
47	chartreuse	#7FFF00	127	255	0
48	chartreuse1	#7FFF00	127	255	0
49	chartreuse2	#76EE00	118	238	0
50	chartreuse3	#66CD00	102	205	0
51	chartreuse4	#458B00	69	139	0
52	chocolate	#D2691E	210	105	30
53	chocolate1	#FF7F24	255	127	36
54	chocolate2	#EE7621	238	118	33
55	chocolate3	#CD661D	205	102	29
56	chocolate4	#8B4513	139	69	19
57	coral	#FF7F50	255	127	80
58	coral1	#FF7256	255	114	86
59	coral2	#EE6A50	238	106	80
60	coral3	#CD5B45	205	91	69
61	coral4	#8B3E2F	139	62	47
62	cornflowerblue	#6495ED	100	149	237
63	cornsilk	#FFF8DC	255	248	220
64	cornsilk1	#FFF8DC	255	248	220
65	cornsilk2	#EEE8CD	238	232	205
66	cornsilk3	#CDC8B1	205	200	177
67	cornsilk4	#8B8878	139	136	120
68	cyan	#00FFFF	0	255	255
69	cyan1	#00FFFF	0	255	255
70	cyan2	#00EEEE	0	238	238
71	cyan3	#00CDCD	0	205	205
72	cyan4	#008B8B	0	139	139
73	darkblue	#00008B	0	0	139
74	darkcyan	#008B8B	0	139	139
75	darkgoldenrod	#B8860B	184	134	11
76	darkgoldenrod1	#FFB90F	255	185	15
77	darkgoldenrod2	#EEAD0E	238	173	14
78	darkgoldenrod3	#CD950C	205	149	12
79	darkgoldenrod4	#8B6508	139	101	8
80	darkgray	#A9A9A9	169	169	169
81	darkgreen	#006400	0	100	0
82	darkgrey	#A9A9A9	169	169	169
83	darkkhaki	#BDB76B	189	183	107
84	darkmagenta	#8B008B	139	0	139
85	darkolivegreen	#556B2F	85	107	47
86	darkolivegreen1	#CAFF70	202	255	112
87	darkolivegreen2	#BCEE68	188	238	104
88	darkolivegreen3	#A2CD5A	162	205	90
89	darkolivegreen4	#6E8B3D	110	139	61
90	darkorange	#FF8C00	255	140	0
91	darkorange1	#FF7F00	255	127	0
92	darkorange2	#EE7600	238	118	0
93	darkorange3	#CD6600	205	102	0
94	darkorange4	#8B4500	139	69	0
95	darkorchid	#9932CC	153	50	204
96	darkorchid1	#BF3EFF	191	62	255
97	darkorchid2	#B23AEE	178	58	238
98	darkorchid3	#9A32CD	154	50	205
99	darkorchid4	#68228B	104	34	139
100	darkred	#8B0000	139	0	0

R Plots Overview

- **Line plot**
- **Box plot**
- **Bar plot with error bars**
- **Violin plot**
- **Correlation plot**
- **Heatmap**

type	description
p	points
l	lines
o	overplotted points and lines
b, c	points (empty if "c") joined by lines
s, S	stair steps
h	histogram-like vertical lines
n	does not produce any points or lines

Getting Data In:

from Files

```
> Insurance <- read.csv("Insurance.csv",header=TRUE)
```

from Databases

```
> con <- dbConnect(driver,user,password,host,dbname)
> Insurance <- dbSendQuery(con, "SELECT * FROM
  claims")
```

from the Web

```
> con <- url('http://labs.dataspora.com/test.txt')
> Insurance <- read.csv(con, header=TRUE)
```

from R objects

```
> load('Insurance.RData')
```

Getting Data Out:

to Files

```
write.csv(Insurance,file="Insurance.csv")
```

to Databases

```
con <- dbConnect(dbdriver,user,password,host,dbname)
dbWriteTable(con, "Insurance", Insurance)
```

to R Objects

```
save(Insurance, file="Insurance.RData")
```

Reading Data From a File

```
# Set a working directory by specifying the path:
setwd("~/Desktop")
data <- read.table("gene_expression.txt", sep = "\t", header = T)

# Check dimensions of the data frame:
dim(data)
[1] 34  4

# Display first 6 rows of the data frame
head(data)
 genesymbol condition1 condition2 condition3
1      P2RY2         12         55        110
2       SOX2         34         33        150
3      OLIG2         13         45        144
4      FOXA1         16         48        138
5       ESR1         16         88        128
6      NANOG         17         49        149

# Display last 6 rows of the data frame
tail(data)

# Calculate average of selected columns in the data frame
data_meaninfo <- apply(data[, 2: 4], 2, mean)
data_meaninfo
condition1 condition2 condition3
  19.17647   63.14706  150.85294
```

?apply
apply(X, MARGIN, FUN, ...)

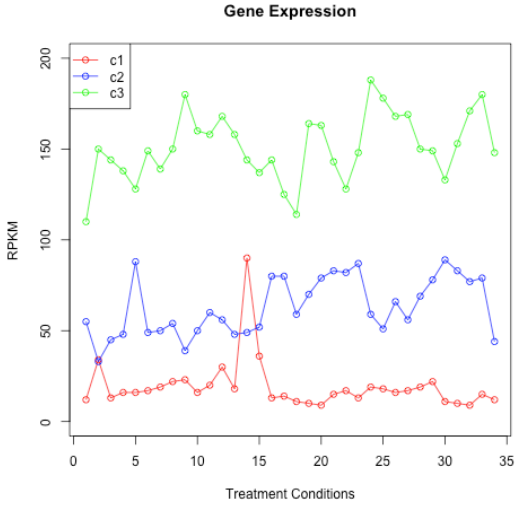
Line Plot

Line Chart:

A line chart is a graph that connects a series of points by drawing line segments between them

```
plot(v,type,col,xlab,ylab)
v     is a vector containing the numeric values.
type takes the value "p" to draw only the points,
                     "l" to draw only the lines and
                     "o" to draw both points and lines.
xlab is the label for x axis.
ylab is the label for y axis.
main is the Title of the chart.
col   is used to give colors to both the points and lines.
```

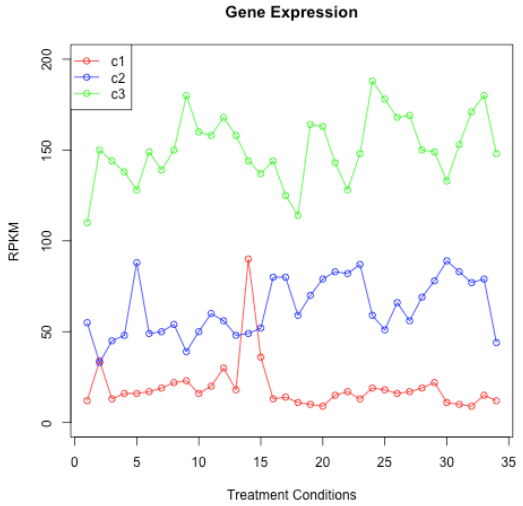
```
# Display first 6 rows of the data frame
head(data)
 genesymbol condition1 condition2 condition3
1      P2RY2         12         55        110
2       SOX2         34         33        150
3      OLIG2         13         45        144
```



Line Plot

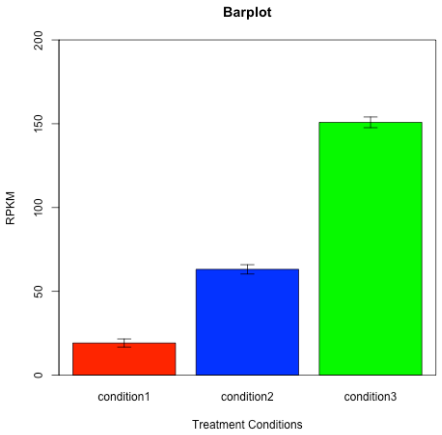
```
# Understand the function using help
?plot # Use q to exit help menu

# Give the chart file a name
png("1_GE_linePlot.png")

# Plot the line chart
plot(data$condition1, type = "o", col = "red", ylim = c(0, 200),
xlab = "Treatment Conditions", ylab = "RPKM", main = "Gene Expression")
lines(data$condition2, type = "o", col = "blue")
lines(data$condition3, type = "o", col = "green")
legend ("topleft",legend=c("c1", "c2", "c3"), col = c("red", "blue", "green"),
lty=1,pch=1)

# Save the file
dev.off()
```

```
### Notes:
#ylim       y-axis limit
#xlab/ylab  x-axis/y-axis label
#main       Main title of the plot
#col        Color of the line
#lty        line type
#pch        symbol type (circle or square or asterisk or triangle…
```



Bar plot With Error Bars

```
# Selecting few columns from the data frame
df <- data[, 2: 4]
  condition1 condition2 condition3
1         12         55        110
2         34         33        150
3         13         45        144

# Read the function to plot error bars into the memory
source("ErrorBar_function.R")

# Calculate means
my_mean <- apply(df, 2, mean)
# Applying mean() function to all the columns in the dataframe 'df'

# Calculate sem (standard error of the mean)
my_sem <- apply(df, 2, sem)
# Applying sem() function to all the columns in the dataframe 'df'
```



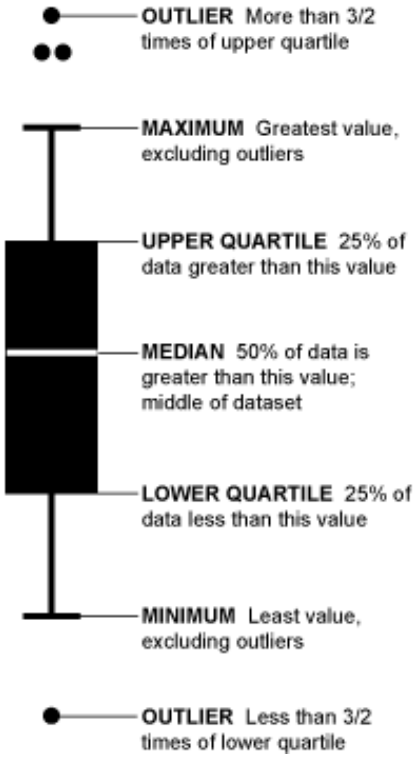

```
# Change colors of the bar chart and save the plot as an image
png("4_GE_barchartWithErrorBars.png")
barx <-
barplot(my_mean, names.arg = names(df),ylim = c(0, 200), xlab = "Treatment Conditions",
ylab = "RPKM", col = c("red", "blue", "green"),main="Barplot")

box()
error.bar(barx, my_mean, my_sem)

dev.off()
```

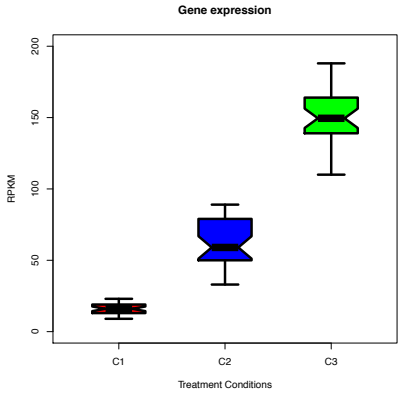
Boxplot

Used to check the distribution of the data. Divides data into 3 quartiles

```
boxplot(data$variable1, notch, names, main)
```

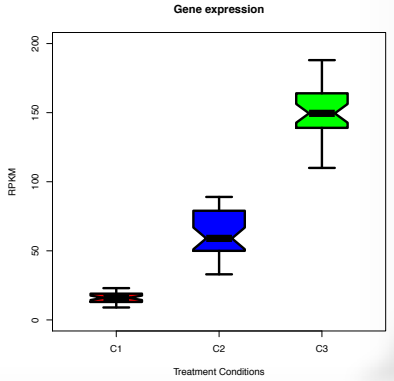
data$variable1	data column in the data frame to be plotted
notch	is a logical value. Set as TRUE to draw a notch.
names	are the group labels which will be printed under each boxplot.
main	is used to give a title to the graph.





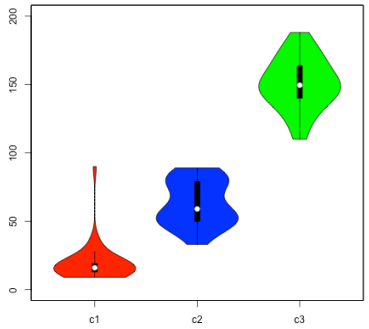
Boxplot

```
# Let's plot the RPKM values of different conditions in the data frame
head(data)
  genesymbol condition1 condition2 condition3
1      P2RY2         12         55        110
2       SOX2         34         33        150
boxplot(data$condition1, data$condition2, data$condition3, ylim=c(0,200))

# Give the plot file a name.
pdf("2_GE_boxplot.pdf")
# Plot the boxplot
boxplot(data$condition1, data$condition2, data$condition3,
col = c("red", "blue", "green"), names = c("C1", "C2", "C3"), pin=c(5,5),
notch = TRUE, outline = FALSE, lwd = 4, lty = 1,
cex.axis = 1, ylim = c(0, 200), boxwex = 0.5,
main = "Gene expression", xlab = "Treatment Conditions", ylab = "RPKM")
# Save the file
dev.off()

### Notes:
#pin  plot dimensions (width, height) in inches
#cex.axis   magnification of axis annotation relative to cex
#outline   logical value whether or not to plot outliers
#lwd       line width of the box
#boxwex    width of the box
#lty  line type
#cex  number by which plotting text and symbols should be scaled
#pch       symbol type (circle or square or asterisk or triangle etc)
```



Violin plot

```
library(vioplot)

# Understand the function using help
#?vioplot # Use shift + q to exit help menu

png("5_GE_violinPlots.png")
vioplot(data$condition1, data$condition2, data$condition3, col = c("red"), horizontal = FALSE)
title("Violin Plots of Miles Per Gallon")
dev.off()


# The easiest thing is to set up your own frame for the plot and then use add=TRUE to add more #
violins to the plot

png("6_GE_violinPlotsColorCoded.png")
# Open a plot first
plot(1, 1, xlim = c(0.5, 3.5), ylim = c(0, 200), type = "n", xlab = "", ylab = "", axes=FALSE)

## Draw the axes with labels and then plot the violin plots one after another
axis(side = 1, at = 1: 3, labels = c("c1", "c2", "c3"))
axis(side = 2)

vioplot(data$condition1, col = "red", at = 1, add = T)
vioplot(data$condition2, col = "blue", at = 2, add = T)
vioplot(data$condition3, col = "green", at = 3, add = T)

dev.off()
```



Understanding mtcars Example:

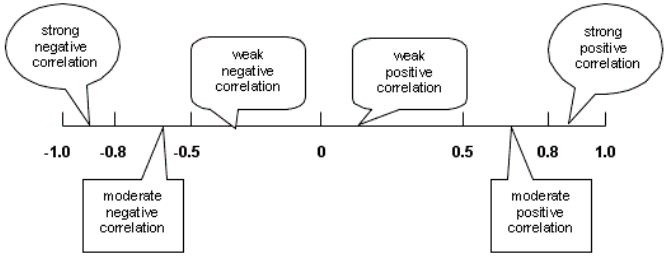
```
mtcars               package:datasets               R Documentation

Motor Trend Car Road Tests

Description:

     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).

Usage:

     mtcars

Format:

     A data frame with 32 observations on 11 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    V/S
       [, 9]  am    Transmission (0 = automatic, 1 = manual)
       [,10]  gear  Number of forward gears
       [,11]  carb  Number of carburetors

Source:

     Henderson and Velleman (1981), Building multiple regression models
     interactively.  _Biometrics_, *37*, 391–411.
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

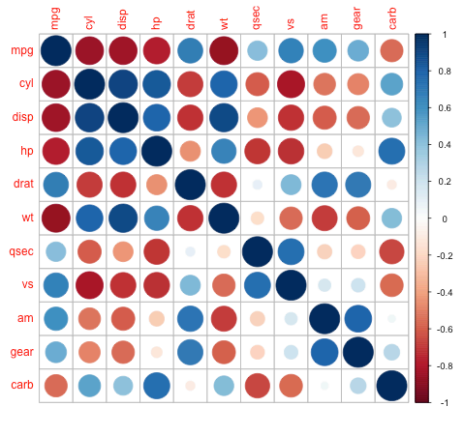
Correlation plot

Correlation measures the strength and direction of a linear relationship.

The correlation coefficient, r, has a specific range of values: $-1 \leq r \leq 1$



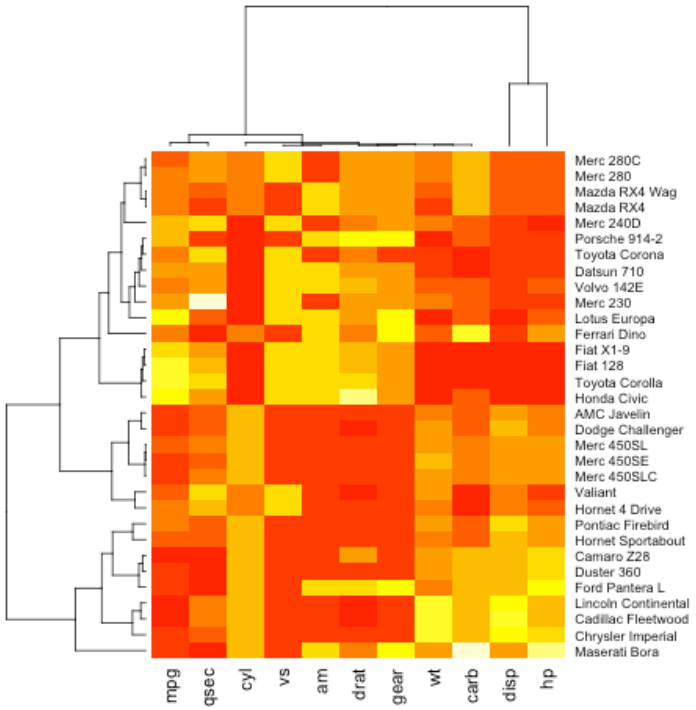

```
library(corrplot)
#?corrplot

df_corr <-cor(mtcars)
df_corr
             mpg        cyl       disp         hp        drat         wt
mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594
cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958
disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799

png("7_GE_CorrelationPlot.png")
corrplot(df_corr, method = "circle")
dev.off()
```



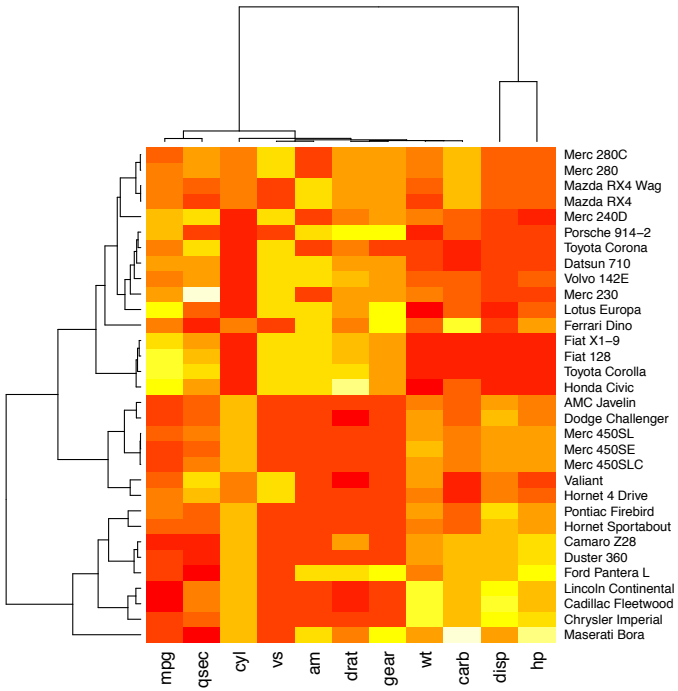

```
corr:       The correlation matrix to visualize
method:     Character, the visualization method of correlation matrix to  be used.
            Currently, it supports seven methods, named '"circle"' (default), '"square"',
            '"ellipse"', '"number"',  '"pie"', '"shade"' and '"color"'
```

Heatmap

A heat map is a false color image with a dendrogram added to the left side and to the top.



.png file



.pdf file

Heatmap

```
# ?heatmap
# The input for heatmap function should be a "matrix"
mat <- as.matrix(mtcars)
mat
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1

heatmap(mat)
```

1 # plots heatmap using default of heatmap function produces false colored image due to wide range of values

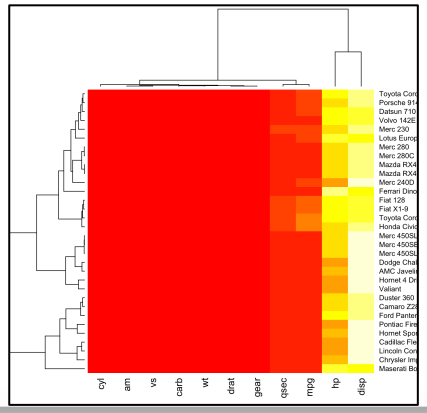
```
# Not clustering the data to understand the problem
```

2 `heatmap(mat, Rowv = NA, Colv = NA, scale = "none")`

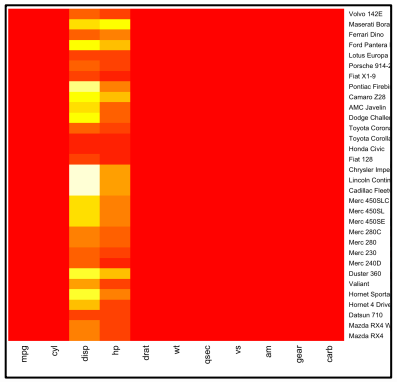
```
heatmap(mat, Rowv = NA, Colv = NA, scale = "column")
```

3 # scale the columns to account for huge variability in the range of values & to see clear structure of the data

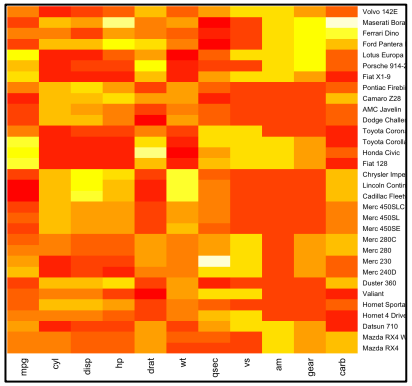
1



2

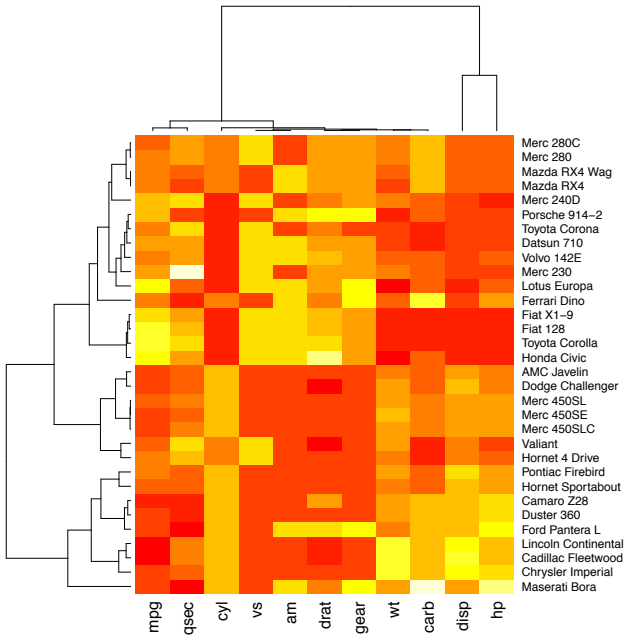


3



Heatmap

```
#Now that the features are at the same scale, we can clearly see that there is structure in the data. We are ready to add clustering to the heatmap; Using default clustering in this example
# Clustering of car attributes(rows)
row_dist <- dist(mat, method = "euclidean")
row_clus <- hclust(row_dist, method = "complete")

# Clustering of cars(columns)
col_dist <- dist(t(mat), method = "euclidean")
col_clus <- hclust(col_dist, method = "complete")

png("8_GE_heatmap.png")
heatmap(mat,

# Clustering
Rowv = as.dendrogram(row_clus),
Colv = as.dendrogram(col_clus),

#scaling
scale = "column", pin=c(10,10),margins=c(5,10))
dev.off()
```




```
#Notes:
dist:       function used to compute the distance (dissimilarity) between both rows and columns.
hclust:     function used to compute the hierarchical clustering when 'Rowv' or 'Colv' are not dendrograms.
Rowv:       determines if and how the _row_ dendrogram should be computed and reordered
Colv:       determines if and how the _column_ dendrogram should be reordered.
```

Statistical Tests

Correlation

- Correlation test is used to evaluate the association between two or more variables There are different methods to perform correlation analysis:
- Pearson correlation (r), which measures a linear dependence between two variables (x and y). It can be used only when x and y are from normal distribution. The plot of y = f(x) is named the linear regression curve.
- Kendall tau and Spearman rho, which are rank-based correlation coefficients (non-parametric)
- Most Commonly used method is Pearson Correlation
- Example Code
  - cor() computes the correlation coefficient
  - cor.test() test for association/correlation between paired samples.
  - It returns both the correlation coefficient and the significance level(or p-value) of the correlation
  - The simplified formats are:
    - cor(x, y, method = c("pearson", "kendall", "spearman"))
    - cor.test(x, y, method=c("pearson", "kendall", "spearman"))

```
        Pearson's product-moment correlation
data:  my_data$wt and my_data$mpg
t = -9.559, df = 30, p-value = 1.294e-10
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9338264 -0.7440872
sample estimates:
       cor
-0.8676594
```

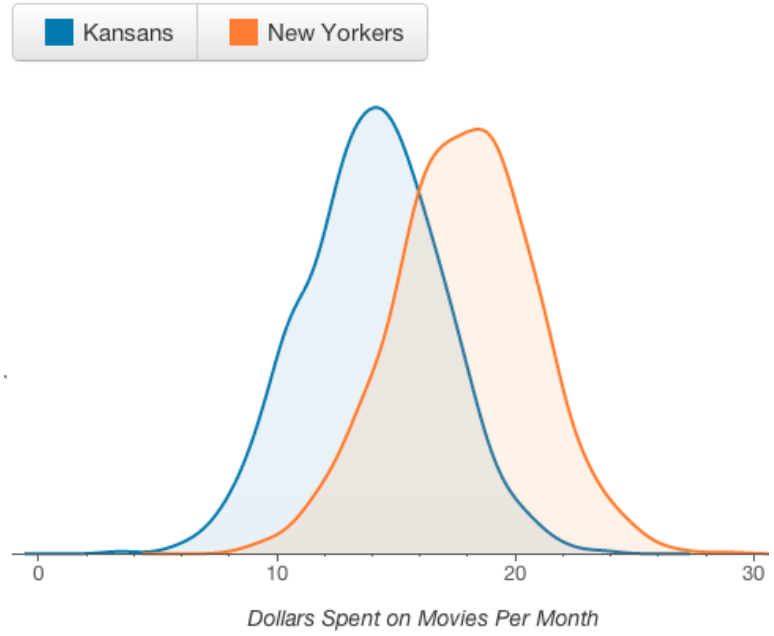
T-Test

- **Parametric test**
- **Determine whether the means of two groups are equal to each other**
- **Assumption for the test is that both groups are sampled from normal distributions with equal variances**
- **Null hypothesis is that the two means are equal, and the Alternative is that they are not**
- **Modification of the t-test - Welch's t-test**
  - **adjusts the number of degrees of freedom when the variances are thought not to be equal to each other**
- **Example**

```
Welch Two Sample t-test data:  x and y
t = 1.4896, df = 15.481, p-value = 0.1564
alternative hypothesis: true difference in means is not equal t
o 0 95 percent confidence interval:  -0.3221869  1.8310421
sample estimates:  mean of x  mean of y  0.1944866 -0.5599410
```

T-Test

The average New Yorker spends more than the average Kansan per month on movies



- A *statistically significant* t-test result is one in which a difference between two groups is unlikely to have occurred because the sample happened to be atypical
- Statistical significance is determined by the size of the difference between the group averages, the sample size, and the standard deviations of the groups

Wilcoxon Signed-Rank Test

- **Two data samples are matched if they come from repeated observations of the same subject**
- **Using the Wilcoxon Signed-Rank Test, we can decide whether the corresponding data population distributions are identical *without* assuming them to follow the normal distribution**
- **Example**

```
wilcox.test(immer$Y1, immer$Y2, paired=TRUE)
      Wilcoxon signed rank test with continuity correction
data:  immer$Y1 and immer$Y2
V = 368.5, p-value = 0.005318
alternative hypothesis: true location shift is not equal to 0
Warning message:
In wilcox.test.default(immer$Y1, immer$Y2, paired = TRUE) :
  cannot compute exact p-value with ties
```

Mann-Whitney-Wilcoxon Test

- **Two data samples are independent if they come from distinct populations and the samples do not affect each other.**
- **Using the Mann-Whitney-Wilcoxon Test, we can decide whether the population distributions are identical *without* assuming them to follow the normal distribution**
- **Example**

```
wilcox.test(mpg ~ am, data=mtcars)
        Wilcoxon rank sum test with continuity correction
data:  mpg by am
W = 42, p-value = 0.001871
alternative hypothesis: true location shift is not equal to 0
Warning message:
In wilcox.test.default(x = c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8,
:
  cannot compute exact p-value with ties
```

Kruskal-Wallis Test

- **A collection of data samples are independent if they come from unrelated populations and the samples do not affect each other.**
- **Using the Kruskal-Wallis Test, we can decide whether the population distributions are identical *without* assuming them to follow the normal distribution**
- **Example**

```
wilcox.test(mpg ~ am, data=mtcars)
        Wilcoxon rank sum test with continuity correction
data:  mpg by am
W = 42, p-value = 0.001871
alternative hypothesis: true location shift is not equal to 0
Warning message:
In wilcox.test.default(x = c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8,
:
  cannot compute exact p-value with ties
```