##
#### *Overview of Data Structures*
##
## *Atomic Vectors:*

```r
# Numeric Vector:
x1 <- c(1, 2, 3)
x1
length(x1)
class(x1)

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

# Logical Vector:
x3 <-  c(TRUE, TRUE, FALSE, FALSE)
class(x3)

# Character Vector:
x4 <- c("Alec", "Dan", "Rob", "Karthik")
class(x4)
x4

# Add elements
x4 <- c(x4, "Annette")
x4
```

## *List:*

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

## *Matrix:*

```r
## Method1:
m1 <- matrix(nrow = 2, ncol = 2)
m1
dim(m1)
m1 <- 1: 10
m1

# Transform the vector into a matrix with 2 rows and 5 columns.
dim(m1) <- c(2, 5)
m1

## Method2:
x <- 1: 3
y <- 10: 12
m2 <- cbind(x, y)
m3 <- rbind(x, y)
m2
m3

# Add the row and column names:
dimnames(m3) <- list(c("a", "b"), c("c", "d", "e"))
m3
```

## *Data frame:*

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

# Add an additional “name” column to the dataframe
df$name <- paste("id", 1: 10, sep = "")
df
colnames(df)


# Change the order of the columns:
df <- df[, c(4, 1, 2, 3)]
df

#Remove a selected column
df <- subset(df, select = -id)
df

# Change the column names
colnames(df) <- c("col1", "col2", "col3" )
df

# List current objects in the workspace
ls()

# Delete an object
rm(df)
```

##
#### Basic Plotting
##

## *Install the following packages:*

```
install.packages("vioplot")
install.packages("e1071")
```

## *Reading data from 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)

# Display first 6 rows of the data frame
head(data)

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

# Calculate average of every column in the data frame
data_meaninfo <- apply(data[, 2: 4], 2, mean)
data_meaninfo
```

## *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:
#type      type of symbol to be used
```

## *Boxplot:*

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

# Let’s plot the RPKM values of different conditions in the data frame
head(data)
boxplot(data$condition1, data$condition2, data$condition3)
boxplot(data$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()


# Add mean to the above boxplot:
png("3_GE_boxplotwithmean.png")

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")

points(data_meaninfo, col="yellow", pch=16, cex=1)

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)
```

## *Barplot with error bars:*

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

# Selecting few columns from the data frame
df <- data[, 2: 4]
head(data)
head(df)

# 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'

# Barplot
barx <- barplot(my_mean, names.arg = names(df), ylim = c(0, 200),
xlab = "Treatment Conditions", ylab = "RPKM")
error.bar(barx, my_mean, my_sem)


# 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()
```

## *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()
```

##
### *Using the data "mtcars" embedded in R*
##

```
# Getting to know your data
?mtcars
# use shift + q to exit the help and return to R terminal

# Load the data into the memory
data(mtcars)

dim(mtcars)
class(mtcars)
```

### *Correlation Plot:*

```
install.packages('corrplot')
library(corrplot)
#?corrplot

df_corr <-cor(mtcars)
df_corr

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

### *Heatmap:*

```
# ?heatmap
mat <- as.matrix(mtcars)
# The input for heatmap function should be a "matrix"

heatmap(mat)
# plots heatmap using default of heatmap function

# Not clustering the data to understand the problem
heatmap(mat, Rowv = NA, Colv = NA, scale = "none")

mtcars

heatmap(mat, Rowv = NA, Colv = NA, scale = "column")
# scale the columns to account for huge variability in the range of values

# 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()
```

##
#### *Statistical Tests*
##
## Covariance

```
mydata = read.csv("YellowstoneGeyserData.csv")
duration = faithful$eruptions # eruption durations
waiting = faithful$waiting # the waiting period
cov(duration, waiting) # apply the cov function
```

##Correlation
#To test the linear relationship of two continuous variables
#The cor.test() function computes the correlation between two continuous variables and test if the y is dependent on the x.
#The null hypothesis is that the true correlation between x and y is zero.

```
duration = faithful$eruptions # eruption durations
waiting = faithful$waiting # the waiting period
cor(duration, waiting) # apply the cor function
```

##Skewness

```
library(e1071) # load e1071
duration = faithful$eruptions # eruption durations
moment(duration, order = 3, center = TRUE)
```

#In particular, the second central moment of a population is its variance. The third central moment of eruption duration is -0.6149.

```
duration = faithful$eruptions # eruption durations
skewness(duration)# apply the skewness function
```

#The skewness of eruption duration is -0.41355. It indicates that the eruption duration distribution is skewed towards the left.

### Commonly Used Tests
##t-test

```
t.test(mpg~am, data = mtcars)
t.test(var1, var2,var.equal = T)
t.test(var1, var2, paired = T)
```

#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.

```
library(MASS) immer = read.csv("ImmerBarleyYeild.csv")
head(immer)
```

#The null hypothesis is that the barley yields of the two sample years are identical populations. To test the hypothesis, we apply the wilcox.test function to compare the matched samples.
#For the paired test, we set the "paired" argument as TRUE.
#As the p-value turns out to be 0.005318, and is less than the .05 significance level, we reject the null hypothesis.

```
wilcox.test(immer$Y1, immer$Y2, paired = TRUE)
```

#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.

```
mtcars = read.csv("mtcars.csv")
mtcars$mpg
mtcars$am
#am,indicates the transmission type of the automobile model(0 = automatic, 1 = manual).
In other words, it is the differentiating factor of the transmission type.
wilcox.test(mpg~am, data = mtcars)
```

###Additional Tests

#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.

```
airquality = read.csv("airquality.csv")
head(airquality)
kruskal.test(Ozone~Month, data = airquality)
```

#Shapiro Test

```
set.seed(100)
normaly_disb <- rnorm(100, mean = 5, sd = 1)
# generate a normal distribution shapiro.test(normaly_disb)
set.seed(100)
not_normaly_disb <- runif(100)# uniform distribution.
shapiro.test(not_normaly_disb)
```

#Kolmogorov And Smirnov Test
#Kolmogorov-Smirnov test is used to check whether 2 samples follow the same distribution.

```
x <- rnorm(50)
y <- runif(50)
ks.test(x, y)
```