R-PROGRAMMING

Table of Contents

Introduction to R .......................................................................................................................... 5
Vector .............................................................................................................................................. 5
Matrix ............................................................................................................................................. 7
Data Frame ..................................................................................................................................... 9
  Common Functions of data frame ................................................................................................. 9
  Creating a data frame ................................................................................................................ 10
Get the Structure of the Data Frame .......................................................................................... 11
Summary of Data in Data Frame ............................................................................................... 11
Extract Data from Data Frame .................................................................................................... 12
Changing column names in data frame ..................................................................................... 14
Changing values in data frame .................................................................................................... 15
Reading Data in R .......................................................................................................................... 16
  Reading data from excel ........................................................................................................... 17
  Importing data from a SPSS ...................................................................................................... 17
  Importing data from SAS .......................................................................................................... 18
  Reading data with scan function ............................................................................................ 18
R-Arrays ........................................................................................................................................ 19
  Naming Columns and Rows of array ......................................................................................... 20
  Accessing Elements of Array ................................................................................................... 21
  Multiplying Elements in Array ................................................................................................ 22
  Calculations across Array Elements ....................................................................................... 24
R-Factors ......................................................................................................................................... 25
  Factors in Data Frame .............................................................................................................. 25
  Changing the Order of Levels .................................................................................................. 26
  Generating Factor Levels .......................................................................................................... 27
R-CSV Files .................................................................................................................................... 27
  Getting and Setting the Working Directory .............................................................................. 28
  Input as CSV File ...................................................................................................................... 28
  Reading a CSV File .................................................................................................................. 29
  Analyzing the CSV File ............................................................................................................. 29
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing into a CSV File</td>
<td>30</td>
</tr>
<tr>
<td>R-XML Files</td>
<td>31</td>
</tr>
<tr>
<td>Input Data</td>
<td>32</td>
</tr>
<tr>
<td>Reading XML File</td>
<td>33</td>
</tr>
<tr>
<td>XML to Data Frame</td>
<td>35</td>
</tr>
<tr>
<td>R-Databases</td>
<td>36</td>
</tr>
<tr>
<td>RMySQL Package</td>
<td>36</td>
</tr>
<tr>
<td>Connecting R to MySql</td>
<td>37</td>
</tr>
<tr>
<td>Querying the Tables</td>
<td>37</td>
</tr>
<tr>
<td>Query with Filter Clause</td>
<td>38</td>
</tr>
<tr>
<td>Updating Rows in the Tables</td>
<td>39</td>
</tr>
<tr>
<td>Inserting Data into the Tables</td>
<td>39</td>
</tr>
<tr>
<td>Creating Tables in MySql</td>
<td>39</td>
</tr>
<tr>
<td>Dropping Tables in MySql</td>
<td>40</td>
</tr>
<tr>
<td>R-Pie Charts</td>
<td>40</td>
</tr>
<tr>
<td>Pie Chart Title and Colors</td>
<td>41</td>
</tr>
<tr>
<td>Slice Percentages and Chart Legend</td>
<td>43</td>
</tr>
<tr>
<td>3D Pie Chart</td>
<td>44</td>
</tr>
<tr>
<td>R-Bar Charts</td>
<td>45</td>
</tr>
<tr>
<td>Bar Chart Labels, Title and Colors</td>
<td>46</td>
</tr>
<tr>
<td>Group Bar Chart and Stacked Bar Chart</td>
<td>47</td>
</tr>
<tr>
<td>R-Boxplots</td>
<td>49</td>
</tr>
<tr>
<td>Creating the Boxplot</td>
<td>50</td>
</tr>
<tr>
<td>Boxplot with Notch</td>
<td>51</td>
</tr>
<tr>
<td>R-Histograms</td>
<td>52</td>
</tr>
<tr>
<td>Range of X and Y values</td>
<td>53</td>
</tr>
<tr>
<td>R-Line Graphs</td>
<td>54</td>
</tr>
<tr>
<td>Line Chart Title, Color and Labels</td>
<td>55</td>
</tr>
<tr>
<td>Multiple Lines in a Line Chart</td>
<td>56</td>
</tr>
<tr>
<td>R-Scatterplots</td>
<td>58</td>
</tr>
<tr>
<td>Creating the Scatterplot</td>
<td>59</td>
</tr>
<tr>
<td>Scatterplot Matrices</td>
<td>60</td>
</tr>
<tr>
<td>R-Mean, Median and Mode</td>
<td>61</td>
</tr>
</tbody>
</table>
Mean ................................................................................................................................. 61
Median ................................................................................................................................. 63
Mode .................................................................................................................................. 63
R-Function ............................................................................................................................ 64
Function Definition ............................................................................................................. 64
Function Components ......................................................................................................... 64
Built-in Function .................................................................................................................. 65
User-defined Function ........................................................................................................ 65
Calling a Function ............................................................................................................... 65
Calling a Function without an Argument ............................................................................ 66
Calling a Function with Argument Values (by position and by name) ................................ 66
Calling a Function with Default Argument ....................................................................... 67
Lazy Evaluation of Function ............................................................................................. 67
R-Linear Regression ........................................................................................................... 68
   Steps to Establish a Regression ....................................................................................... 68
R-Multiple Regression ........................................................................................................ 72
R-Logistic Regression ......................................................................................................... 74
R-Normal Distribution ....................................................................................................... 76
R-Analysis of Covariance .................................................................................................. 81
R-Time Series Analysis ..................................................................................................... 83
R-Decision Tree ................................................................................................................ 87
R-Random Forest .............................................................................................................. 90
R-Survival Analysis ........................................................................................................... 92
R-Chi Square Test .............................................................................................................. 95
Missing Values .................................................................................................................. 97
   Handling and detecting missing values ....................................................................... 99
   Filter functions to handle NA ...................................................................................... 99
Data cleaning or How to clean Data .................................................................................. 100
   Replacing data has to done with extreme care ............................................................ 106
   Saving the data set ....................................................................................................... 106
How to run analyses ........................................................................................................ 107
   Numerical Measures ................................................................................................. 107
   Probability Distributions ............................................................................................ 112
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Estimation</td>
<td>115</td>
</tr>
<tr>
<td>Hypothesis Testing</td>
<td>118</td>
</tr>
<tr>
<td>Type II Error</td>
<td>126</td>
</tr>
<tr>
<td>Type II Error in Lower Tail Test of Population Mean with Known Variance</td>
<td>127</td>
</tr>
<tr>
<td>Type II Error in Upper Tail Test of Population Mean with Known Variance</td>
<td>128</td>
</tr>
<tr>
<td>Type II Error in Two-Tailed Test of Population Mean with Known Variance</td>
<td>129</td>
</tr>
<tr>
<td>Type II Error in Lower Tail Test of Population Mean with Unknown Variance</td>
<td>130</td>
</tr>
<tr>
<td>Type II Error in Upper Tail Test of Population Mean with Unknown Variance</td>
<td>131</td>
</tr>
<tr>
<td>Type II Error in Two-Tailed Test of Population Mean with Unknown Variance</td>
<td>133</td>
</tr>
<tr>
<td>Two Populations</td>
<td>134</td>
</tr>
<tr>
<td>Population Mean Between Two Matched Samples</td>
<td>134</td>
</tr>
<tr>
<td>Population Mean Between Two Independent Samples</td>
<td>135</td>
</tr>
<tr>
<td>Comparison of Two Population Proportions</td>
<td>136</td>
</tr>
<tr>
<td>Goodness of fit</td>
<td>137</td>
</tr>
<tr>
<td>Chi-squared Test of Independence</td>
<td>138</td>
</tr>
<tr>
<td>Analysis of Variance</td>
<td>140</td>
</tr>
</tbody>
</table>
Introduction to R

R is a language and environment for statistical computing and graphics. It provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. R is available as free software runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

Vector

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called components.

Vector can contain numeric values.

```r
> c(1,2,3)  > c(2,4,6,8,10)
[1] 1 2 3  [1] 2 4 6 8 10
```

Vector can contain logical values.

```r
> c(TRUE, FALSE)
[1] TRUE FALSE
```

Vector can contain character strings.

```r
> c("R", "SAS", "SPSS", "ITS","UAlbany")
[1] "R" "SAS" "SPSS" "ITS" "UAlbany"
```

Length function can show the number of members in a vector.

```r
> length(c("R", "SAS", "SPSS", "ITS","UAlbany"))
[1] 5
```

Vectors can be combined via the function c.

```r
> a = c(1,2,3)  > a = c(1,2,3)
> b = c("R")    > b = c(TRUE, FALSE)
> c(a,b)         > c(a,b)
[1] "1" "2" "3" "R" [1] 1 2 3 1 0
```

Vector arithmetic allows vectors to add, subtraction, multiply and division.

```r
> a = c(1,2,3,4)
> b = c(5,6,7,8)
```
If two vectors have different length but their lengths are multiple, then the shorter one will be **recycled** in order to match the longer one.

```r
> a = c(1,2,3)
> b = c(1,2,3,4,5,6,7,8,9)
> a + b
[1] 2 4 6 5 7 9 8 10 12
```

If the lengths are not multiple, then no result will come out.

```r
> a = c(1,2,3,4)
> b = c(1,2,3,4,5,6)
> a + b
[1] 2 4 6 8 6 8
Warning message:
In a + b : longer object length is not a multiple of shorter object length
```

Vector value can be **retrieved** by declaring an **index** inside a single square bracket "[]" operator.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[1]
[1] "R"
```

If the index is **negative** it would remove the member that located at the position with the same absolute value of that negative index.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[-1]
[1] "SAS" "SPSS" "ITS" "UAlbany"
```

If the index is **out-of-range**, the output will be NA.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[6]
[1] NA
```

Two **negative** indexes will not remove two vector values. It only remove the vector value located in the position with the same absolute value of their sum.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[-1 -2]
[1] "R" "SAS" "ITS" "UAlbany"
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[-2 -3]
[1] "R" "SAS" "SPSS" "ITS"
```

To retrieve several vector values, a new vector can be sliced from a given vector with a **numeric index vector**.
Retrieve a member several times gives **duplicate** values.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[c(1,2,3)]
[1] "R" "SAS" "SPSS"
```

Order matters in index vector.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[c(5,4,3)]
[1] "UAlbany" "ITS" "SPSS"
```

`:;` between two indexes can pick a few values within the two indexes.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> a[1:3]
[1] "R" "SAS" "SPSS"
```

A new vector can be sliced from a given vector with a **logical index vector**, which has the same length as the original vector. When the new value is TRUE the original value will be included in the slice.

```r
> a = c("R", "SAS", "SPSS", "ITS","UAlbany")
> L = c(TRUE, TRUE, TRUE, FALSE, FALSE)
> a[L]
[1] "R" "SAS" "SPSS"
```

Vector can be **named**.

```r
> a = c("User First Name", "User Last Name")
> a
[1] "User First Name" "User Last Name"
> names(a) = c("First", "Last")
> a
     First     Last
"User First Name" "User Last Name"
> a["First"]
     First
"User First Name"
```

**Matrix**

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.

A matrix can be written by matrix function. The value should be the same type.
A\[m,n\] can access the element at \(m\)th row and \(n\)th column.

\[
\begin{bmatrix}
[1,] & [2,] & [3,] & [4,] \\
[1,] & 1 & 2 & 3 & 5 \\
[2,] & 3 & 4 & 6 & 7 \\
[3,] & 5 & 6 & 8 & 9 \\
\end{bmatrix}
\]

Enter only half index in the bracket will access either the row or the column.

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 1 & 5 \\
[2,] & 2 & 6 \\
[3,] & 3 & 7 \\
[4,] & 4 & 8 \\
\end{bmatrix}
\]

\[
A[3,2]
\]

Enter only half index in the bracket will access either the row or the column.

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 1 & 5 \\
[2,] & 2 & 6 \\
[3,] & 3 & 7 \\
[4,] & 4 & 8 \\
\end{bmatrix}
\]

\[
A[4,]
\]

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 4 & 8 \\
\end{bmatrix}
\]

\[
A[,1]
\]

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 1 & 2 & 3 \\
\end{bmatrix}
\]

\[
A[c(3,4),]
\]

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 3 & 7 \\
[2,] & 4 & 8 \\
\end{bmatrix}
\]

Names can be assigned on rows and columns in order to access elements by names.
A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Data frames are used for storing data tables or a table like structure where columns contains variables and rows contains case.

**Common Functions of data frame**

- `names(dataframe)`: gives the column variables as output
- `ncol(dataframe)`: returns the number of columns
- `nrow(dataframe)`: returns the number of rows
- `Length(dataframe)`: returns the length of list , similar to ncol()

Example:

```r
> A
 [,1] [,2]
[1,] 1  5
[2,] 2  6
[3,] 3  7
[4,] 4  8
> dimnames(A) = list(
  + c("row1", "row2", "row3", "row4"),
  + c("col1", "col2"))
> A
   col1 col2
row1 1  5
row2 2  6
row3 3  7
row4 4  8
```
Creating a data frame

The function `data.frame()` creates data frames.

```r
> names(x)
[1] "SH"    "Age"   "Name"

> ncol(x)
[1] 3

> nrow(x)
[1] 2

> length(x)  # returns length of the list, same as ncol()
[1] 3
```

The data frame above is stored in `d` where alpha, beta and gamma are the columns and the rows are cases.
Get the Structure of the Data Frame

The structure of the data frame can be seen by using `str()` function.

```r
> # Get the structure of the data frame.
> str(emp.data)

'data.frame':  5 obs. of  4 variables:
$ emp_id     : int 1 2 3 4 5
$ emp_name   : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary     : num 623 515 611 729 843
$ start_date : Date, format: "2012-01-01" "2013-09-23" "2014-11-15" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying `summary()` function.

```r
> # Print the summary.
> print(summary(emp.data))

  emp_id  salary    start_date
Min.   :1  Min. :515.2 Min. :2012-01-01
1st Qu.:2  1st Qu.:611.0 1st Qu.:2013-09-23
Mean   :3  Mean   :664.4 Mean   :2014-01-14
3rd Qu.:4  3rd Qu.:729.0 3rd Qu.:2014-11-15
Max.   :5  Max.   :843.2 Max.   :2015-03-27
```
Extract Data from Data Frame

Extract specific column from a data frame using column name.

```r
> # Extract Specific columns.
> result <- data.frame(emp.data$emp_name, emp.data$salary)
> print(result)
  emp.data.emp_name emp.data.salary
1        Rick         623.30
2        Dan           515.20
3  Michelle         611.00
4        Ryan         729.00
5        Gary         843.25
```

Extract the first two rows and then all columns

```r
> # Extract first two rows
> result <- emp.data[1:2,]
> print(result)
  emp_id emp_name salary start_date
1     1      Rick  623.3 2012-01-01
2     2       Dan  515.2 2013-09-23
```

Extract 3rd and 5th row with 2nd and 4th column

```r
> # Extract 3rd and 5th row with 2nd and 4th column.
> result <- emp.data[c(3,5),c(2,4)]
> print(result)
  emp_name start_date
3  Michelle 2014-11-15
5    Gary 2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```r
> # Add the "dept" column.
> emp.data$dept <- c("IT","Operations","IT","HR","Finance")
> v <- emp.data
> print(v)
  emp_id emp_name salary start_date      dept
1     1      Rick  623.3 2012-01-01      IT
2     2       Dan  515.2 2013-09-23  Operations
3     3  Michelle  611.0 2014-11-15      IT
4     4        Ryan  729.0 2014-05-11      HR
5     5        Gary  843.25 2015-03-27  Finance
```
Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the rbind() function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

> # Create the data frame.
> emp.data <- data.frame(
+   emp_id = c(1:3),
+   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+   salary = c(623.3,515.2,611.0,729.0,843.25),
+   start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11","2015-03-27")),
+   stringsAsFactors=FALSE
+ )
> # Print the data frame.
> print(emp.data)

  emp_id emp_name   salary start_date
1     1     Rick  623.30 2012-01-01
2     2      Dan  515.20 2013-09-23
3     3  Michelle  611.00 2014-11-15
4     4      Ryan  729.00 2014-05-11
5     5      Gary  843.25 2015-03-27

Create the second data frame

> # Create the second data frame
> emp.newdata <- data.frame(
+   emp_id = c(6:8),
+   emp_name = c("Rasmi","Pranab","Tusar"),
+   salary = c(578.0,722.5,632.8),
+   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
+   dept = c("IT","Operations","Finance"),
+   stringsAsFactors=FALSE
+ )
> # Print the new data frame.
> print(emp.newdata)

  emp_id emp_name   salary start_date    dept
1     6     Rasmi  578.00 2013-05-21        IT
2     7     Pranab  722.50 2013-07-30  Operations
3     8     Tusar  632.80 2014-06-17  Finance

Bind the two data frames

> # Bind the two data frames.
> emp.finaldata <- rbind(emp.data,emp.newdata)
> print(emp.finaldata)

  emp_id emp_name   salary start_date    dept
1     1     Rick  623.30 2012-01-01        IT
2     2      Dan  515.20 2013-09-23 Operations
3     3  Michelle  611.00 2014-11-15        IT
4     4      Ryan  729.00 2014-05-11        HR
5     5      Gary  843.25 2015-03-27 Finance
6     6     Rasmi  578.00 2013-05-21        IT
7     7     Pranab  722.50 2013-07-30 Operations
8     8     Tusar  632.80 2014-06-17  Finance
Changing column names in data frame

The column names can be changed using `rename()` from `plyr package`, which is the easiest way to change the column name.

```r
library(plyr)
rename(d, c("beta"="two", "gamma"="three"))
#>   alpha two three
#> 1   1   4   7
#> 2   2   5   8
#> 3   3   6   9
```

In the above case we are changing the column name of “beta” to “two” and the column name of “gamma” to “three” by using `rename()`. In this case `d` is not directly modified, but the result is saved backed to `d`. Here the `rename()` function is used from library.

If we don’t need to rely on `plyr` package then the column names can be changed using below methods.

You can change the column name of “beta” to “two” with

- `names(data.frame name) [names(data.frame name)=="old_column name"] <- "new_column name"`
  
  - Here `data.frame name` is the name of data frame
  - “old_column name” represents the column name that has to be changed and “new_column name” is the new column name which you need in place of existing column name
The column names can be changed by using the index position of column by using

- names(data.frame name) [position of column] <- “new_column name” as shown in above example.

### Changing values in data frame

The values in the data frame can be changed. The simple way is to locate the values to be changed. The values can be accessed by

- Dataframe_name$column_name[row_postion]

**Examples** for accessing the values. “x” represents the dataframe name and “Name” represents column name.

```r
> x["Name"]
Name
1 John
2 Dora

> x$Name
[1] "John" "Dora"

> x[["Name"]]
[1] "John" "Dora"

> x[[3]]
[1] "John" "Dora"
```
The values can be changed by using

- `DataFrame_name$column_name[row_position] <- "new value"

**Example 1:**

```
> alpha two three
> 1 1 4 7
> 2 2 5 8
> 3 3 6 9
```

`d$alpha[1]`  

```r
>> 1
```

## shows the old value

To change the value from “1” to 5

```r
d$alpha[1] <- 5

d$alpha[1]
```

```r
>> 5
```

## The value of alpha[1] is changed to 5.

**Example 2:**

```
> x
SN Age Name
1 1 21 John
2 2 15 Dora
```

```r
> x[1,"Age"] <- 20; x
SN Age Name
1 1 20 John
2 2 15 Dora
```

**Reading Data in R**

Reading data from a file.

`read.table()` command is simplest way to read data from a file.
Reading data from excel

To read an Excel file is to export it to a comma delimited file and import it using the method above. Alternatively you can use the `xlsx` package to access Excel files as shown below. The first row should contain variable/column names.

```r
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on mswindows systems
mydata <- read.table("c:/mydata.csv", header=TRUE, sep="", row.names="id")
```

```r
print(test.csv1)
```

Importing data from a SPSS

```r
read.spss()
```

Read an SPSS data file
Importing data from SAS

Save the SAS data set in transport format

```r
# in R
library(Hmisc)
mydata <- sasxport.get("c:/mydata.xpt")
# character variables are converted to R factors
```

Reading data with scan function

The `scan` function is an extremely flexible tool for importing data. It can be used to read in almost any type of data, numeric, character or complex and it can be used for fixed or free formattted files. Moreover, by using the `scan` function it is possible to input data directly from the console.
The `scan` function reads the fields of data in the file as specified by the `what` option with the default being numeric. If the `what` option is specified to be `what=character()` or `what=""` then all the fields will be read as strings. If the data is a mix of numeric, string or complex data then a list can be used in the `what` option

```
name.x <- scan(), what=""
1: bobby
2: kate dave
4: mia
5:
Read 4 items
name.x
[1] "bobby" "kate" "dave" "mia"
```

In the example below we use LIST in what option for scan function.

```
# inputting a text file and outputting a list
x <- scan("http://stats.idre.ucla.edu/wp-content/uploads/2016/02/scan.txt", what=list(age=0, name=""))
Read 4 records

x
$age
[1] 12 24 35 20

$name
[1] "bobby" "kate" "david" "michael"
```

**R-Arrays**

Arrays are the R data objects which can store data in more than two dimensions.
For example - If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the array() function. It takes vectors as input and uses the values in the dim parameter to create an array.

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```r
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim=c(3,3,2))
> print(result)
```

When we execute the above code, it produces the following result:

```
, , 1
     [,1] [,2] [,3]
[1,] 5  10  13
[2,] 9  11  14
[3,] 3  12  15
, , 2
     [,1] [,2] [,3]
[1,] 5  10  13
[2,] 9  11  14
[3,] 3  12  15
```

**Naming Columns and Rows of array**

We can give names to the rows, columns and matrices in the array by using the dimnames parameter.

```r
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
> column.names
[1] "COL1" "COL2" "COL3"
> row.names
[1] "ROW1" "ROW2" "ROW3"
> matrix.names
[1] "Matrix1" "Matrix2"
```
When we execute the above code, it produces the following result:

```
,, Matrix1

   ROW1 ROW2 ROW3
COL1  5   10  13
COL2  9   11  14
COL3  3   12  15

,, Matrix2

   ROW1 ROW2 ROW3
COL1  5   10  13
COL2  9   11  14
COL3  3   12  15
```

### Accessing Elements of Array

Create two vectors of different lengths.

```
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
```

Take the above vectors as inputs to the array

```
> result <- array(c(vector1,vector2),dim=c(3,3,2),dimnames =
+    list(column.names,row.names,matrix.names))
```
Multiplying Elements in Array

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```r
> # Print the third row of the second matrix of the array.
> print(result[3,,2])
ROW1 ROW2 ROW3
 3  12  15

> # Print the element in the 1st row and 3rd column of the 1st matrix.
> print(result[1,3,1])
[1] 13

> # Print the 2nd Matrix.
> print(result[,,2])
ROW1  ROW2  ROW3
COL1   5   10   13
COL2   9   11   14
COL3   3   12   15
```

```r
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> # Take these vectors as input to the array.
> array1 <- array(c(vector1,vector2),dim=c(3,3,2))
> # Create two vectors of different lengths.
> vector3 <- c(9,1,0)
> vector4 <- c(6,0,11,3,14,1,2,6,9)
> array2 <- array(c(vector1,vector2),dim=c(3,3,2))

> vector1
[1]  5  9  3
> vector2
[1] 10 11 12 13 14 15
> vector3
[1]  9  1  0
> vector4
[1]  6  0 11  3 14  1  2  6  9
```
Create matrices from the above arrays and add the matrices.

```
> array1
   , , 1
     [,1] [,2] [,3]
[1,]   5   10  13
[2,]   9   11  14
[3,]   3   12  15
   , , 2
     [,1] [,2] [,3]
[1,]   5   10  13
[2,]   9   11  14
[3,]   3   12  15

> array2
   , , 1
     [,1] [,2] [,3]
[1,]   5   10  13
[2,]   9   11  14
[3,]   3   12  15
   , , 2
     [,1] [,2] [,3]
[1,]   5   10  13
[2,]   9   11  14
```

Create matrices from the above arrays and add the matrices.

```r
> # create matrices from these arrays.
> matrix1 <- array1[,2]
> matrix2 <- array2[,2]
> # Add the matrices.
> result <- matrix1+matrix2
```

When we execute the above code, it produces the following result:

```
> print(result)
   [,1] [,2] [,3]
[1,]  10  20  26
[2,]  18  22  28
[3,]   6  24  30
```
Calculations across Array Elements

We can do calculations across the elements in an array using the apply() function.

Syntax

apply(x, margin, fun)

Following is the description of the parameters used:

• x is an array.

• margin is the name of the data set used.

• fun is the function to be applied across the elements of the array.

We use the apply() function below to calculate the sum of the elements in the rows of an array across all the matrices.

```r
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> # Take these vectors as input to the array.
> new.array <- array(c(vector1,vector2),dim=c(3,3,2))

> print(new.array)
 , , 1

 [,1] [,2] [,3]
[1,] 5  10  13
[2,] 9  11  14
[3,] 3  12  15

, , 2

 [,1] [,2] [,3]
[1,] 5  10  13
[2,] 9  11  14
[3,] 3  12  15

> # Use apply to calculate the sum of the rows across all the matrices.
> result <- apply(new.array, c(1), sum)
```

When we execute the above code, it produces the following result:
Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male", "Female" and True, False etc. They are useful in data analysis for statistical modeling. Factors are created using the factor () function by taking a vector as input.

```r
> # Create a vector as input.
> data <-
> print(data)
[1] "East"  "West"  "East"  "North"  "North"  "East"  "West"  "West"  "East"  "North"
[10] "East"  "North"
> print(is.factor(data))
[1] FALSE
```

After applying the factor function

```r
> # Apply the factor function.
> factor_data <- factor(data)
> print(factor_data)
[1] East  West  East  North  North  East  West  West  West  East  North
Levels: East North West
> print(is.factor(factor_data))
[1] TRUE
```

Factors in Data Frame
On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```r
> # Create the vectors for data frame.
> height <- c(132,151,162,139,166,147,122)
> weight <- c(48,49,66,53,67,52,40)
> gender <- c("male","male","female","female","male","female","male")
> # Create the data frame.
> input_data <- data.frame(height,weight,gender)

> print(input_data)
   height weight gender
   1    132    48  male
   2    151    49  male
   3    162    66 female
   4    139    53 female
   5    166    67  male
   6    147    52 female
   7    122    40  male

Test which column is a factor.

> # Test if the gender column is a factor.
> print(is.factor(input_data$gender))
[1] TRUE
> print(is.factor(input_data$weight))
[1] FALSE
> print(is.factor(input_data$height))
[1] FALSE
> print(is.factor(input_data$gender))
[1] TRUE

> # Print the gender column so see the levels.
> print(input_data$gender)
[1] male male female female male female male
Levels: female male

Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.
Apply the factor function with required order of the level

```r
> # Create the factors
> factor_data <- factor(data)
> print(factor_data)
[1] East West East North North East West West East North
Levels: East North West
```

Generating Factor Levels

We can generate factor levels by using the `gl()` function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

`gl(n, k, labels)`

Following is the description of the parameters used:

- n is a integer giving the number of levels.
- k is a integer giving the number of replications.
- labels is a vector of labels for the resulting factor levels.

```r
> v <- gl(3, 4, labels = c("Tampa", "Seattle", "Boston"))
> print(v)
[1] Tampa Tampa Tampa Tampa Seattle Seattle Seattle Seattle Boston Boston Boston Boston
Levels: Tampa Seattle Boston
```

R-CSV Files
In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

**Getting and Setting the Working Directory**

You can check which directory the R workspace is pointing to using the `getwd()` function. You can also set a new working directory using `setwd()` function.

```r
> print(getwd())
[1] "C:/Users/TT452622/Documents"

> # Set current working directory.
> setwd("/web/com")

> print(getwd())
[1] "C:/Users/TT452622/Documents"
```

This result depends on your OS and your current directory where you are working.

**Input as CSV File**

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named input.csv.

You can create this file using windows notepad by copying and pasting this data. Save the file as `input.csv` using the save As All files(`*.*`) option in notepad.
Reading a CSV File

Following is a simple example of read.csv() function to read a CSV file available in your current working directory:

```r
> data <- read.csv("input.csv")
> print(data)
   id  name salary start_date      dept
  1  1  Rick  623.30 2012-01-01      IT
  2  2   Dan  515.20 2013-09-23  Operations
  3  3 Michelle  611.00 2014-11-15    IT
  4  4   Ryan  729.00 2014-05-11     HR
  5 NA   Gary  843.25 2015-03-27  Finance
  6  6  Nina  578.00 2013-05-21      IT
  7  7 Simon  632.80 2013-07-30  Operations
  8  8 Guru  722.50 2014-06-17  Finance
```

Analyzing the CSV File

By default the read.csv() function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```r
> data <- read.csv("input.csv")
> print(is.data.frame(data))
[1] TRUE
> print(ncol(data))
[1] 5
> print(nrow(data))
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.
Get the maximum salary:

```r
> # Get the max salary from data frame.
> sal <- max(data$salary)
> print(sal)
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```r
> # Get the person detail having max salary.
> retval <- subset(data, salary == max(salary))
> print(retval)
   id name  salary start_date   dept
   5 NA  843.25  2015-03-27 Finance
```

Get all the people working in IT department

```r
> retval <- subset(data, dept == "IT")
> print(retval)
   id name  salary start_date   dept
   1 1 Rick   623.3  2012-01-01   IT
   3 3 Michelle  611.0  2014-11-15   IT
   6 6 Nina    578.0  2013-05-21   IT
```

Get the persons in IT department whose salary is greater than 600

```r
> info <- subset(data, salary > 600 & dept == "IT")
> print(info)
   id name  salary start_date   dept
   1 1 Rick   623.3  2012-01-01   IT
   3 3 Michelle  611.0  2014-11-15   IT
```

Get the people who joined on or after 2014

```r
> retval <- subset(data, as.Date(start_date) >= as.Date("2014-01-01"))
> print(retval)
   id name  salary start_date   dept
   3 3 Michelle  611.00  2014-11-15   IT
   4 4 Ryan    729.00  2014-05-11   HR
   5 NA  843.25  2015-03-27 Finance
   8 8 Guru  722.50  2014-06-17 Finance
```

Writing into a CSV File

R can create csv file from existing data frame. The write.csv() function is used to create the csv file. This file gets created in the working directory.
Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

```r
> # Write filtered data into a new file.
> write.csv(retval,"output.csv")
> newdata <- read.csv("output.csv")
> print(newdata)
   id  name salary start_date dept
1  3 Michelle 611.00 2014-11-15 IT
2  4     Ryan 729.00 2014-05-11 HR
3  NA   Gary 843.25 2015-03-27 Finance
4  8  Guru    722.50 2014-06-17 Finance
```

R-XML Files

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained in the file.

You can read a xml file in R using the "XML" package. This package can be installed using following command.

```r
> install.packages("XML")
--- Please select a CRAN mirror for use in this session ---
  trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/XML_3.98-1.6.zip'
Content type 'application/zip' length 4305641 bytes [41.1 MB]
downloaded 4.1 MB
package 'XML' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
   C:\Users\TT452622\AppData\Local\Temp\RtmpMliwz3\downloaded_packages
```
Input Data
Create a XML file by copying the below data into a text editor like notepad. Save the file with a .xml extension and choosing the file type as all files(*.*).

```
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>2</ID>
    <NAME>Dan</NAME>
    <SALARY>515.2</SALARY>
    <STARTDATE>9/23/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>3</ID>
    <NAME>Michelle</NAME>
    <SALARY>611</SALARY>
    <STARTDATE>11/15/2014</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>4</ID>
    <NAME>Ryan</NAME>
    <SALARY>729</SALARY>
    <STARTDATE>5/11/2014</STARTDATE>
    <DEPT>HR</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>5</ID>
    <NAME>Gary</NAME>
    <SALARY>843.25</SALARY>
    <STARTDATE>3/27/2015</STARTDATE>
    <DEPT>Finance</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>6</ID>
    <NAME>Nina</NAME>
    <SALARY>578</SALARY>
    <STARTDATE>5/21/2013</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>7</ID>
    <NAME>Simon</NAME>
    <SALARY>632.8</SALARY>
    <STARTDATE>7/30/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>8</ID>
    <NAME>Guru</NAME>
    <SALARY>722.5</SALARY>
    <STARTDATE>6/17/2014</STARTDATE>
    <DEPT>Finance</DEPT>
  </EMPLOYEE>
</RECORDS>
```
Reading XML File

The xml file is read by R using the function xmlParse(). It is stored as a list in R.

```r
> # Load the package required to read XML files.
> library("XML")
Warning message:
package 'XML' was built under R version 3.3.3
> # Also load the other required package.
> library("methods")
>
> # Give the input file name to the function.
> result <- xmlParse(file="input.xml")

When we execute the above code, it produces the following result:

```xml
<?xml version="1.0"?>
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>2</ID>
    <NAME>Dan</NAME>
    <SALARY>515.2</SALARY>
    <STARTDATE>9/23/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>3</ID>
    <NAME>Michelle</NAME>
    <SALARY>611</SALARY>
    <STARTDATE>11/15/2014</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>4</ID>
    <NAME>Ryan</NAME>
    <SALARY>729</SALARY>
    <STARTDATE>5/11/2014</STARTDATE>
    <DEPT>HR</DEPT>
  </EMPLOYEE>
</RECORDS>
```
<EMployee>
  <ID>5</ID>
  <NAME>Gary</NAME>
  <SALARY>843.25</SALARY>
  <STARTDATE>3/27/2015</STARTDATE>
  <DEPT>Finance</DEPT>
</EMployee>

<EMployee>
  <ID>6</ID>
  <NAME>Nina</NAME>
  <SALARY>578</SALARY>
  <STARTDATE>5/21/2013</STARTDATE>
  <DEPT>IT</DEPT>
</EMployee>

<EMployee>
  <ID>7</ID>
  <NAME>Simon</NAME>
  <SALARY>632.8</SALARY>
  <STARTDATE>7/30/2013</STARTDATE>
  <DEPT>Operations</DEPT>
</EMployee>

<EMployee>
  <ID>8</ID>
  <NAME>Guru</NAME>
  <SALARY>722.5</SALARY>
  <STARTDATE>6/17/2014</STARTDATE>
  <DEPT>Finance</DEPT>
</EMployee>
</RECORDS>

Get Number of Nodes Present in XML File

> library("XML")
> library("methods")
> # Give the input file name to the function.
> result <- xmlParse(file="input.xml")
> # Extract the root node form the xml file.
> roottnode <- xmlRoot(result)
> # Find number of nodes in the root.
> rootsize <- xmlSize(roottnode)
> # Print the result.
> print(rootsize)
[1] 8

Details of the First Node

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.
Get Different Elements of a Node

```r
> # Load the packages required to read XML files.
> library("XML")
> library("methods")
> # Give the input file name to the function.
> result <- xmlParse(file="input.xml")
> # Extract the root node from the xml file.
> rootnode <- xmlRoot(result)
> # Print the result.
> print(rootnode[1])

$EMPLOYEE
  <ID>1</ID>
  <NAME>Rick</NAME>
  <SALARY>623.3</SALARY>
  <STARTDATE>1/1/2012</STARTDATE>
  <DEPT>IT</DEPT>
</EMPLOYEE>

attr("class")
[1] "XMLInternalNodeList" "XMLNodeList"
```

To handle the data effectively in large files we read the data in the xml file as a data frame. Then process the data frame for data analysis.
As the data is now available as a dataframe we can use data frame related function to read and manipulate the file.

R-Databases

The data is Relational database systems are stored in a normalized format. So, to carry out statistical computing we will need very advanced and complex Sql queries. But R can connect easily to many relational databases like MySql, Oracle, Sql server etc. and fetch records from them as a data frame. Once the data is available in the R environment, it becomes a normal R data set and can be manipulated or analyzed using all the powerful packages and functions.

RMySQL Package

R has a built-in package named "RMySQL" which provides native connectivity between with MySql database.

You can install this package in the R environment using the following command.

```
> install.packages("RMySQL")
trying URL 'https://cran.r-project.org/bin/windows/contrib/3.3/RMySQL_0.10.11.zip'
Content type 'application/zip' length 2292454 bytes (2.2 MB)
downloaded 2.2 MB
package 'RMySQL' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\TT452622\AppData\Local\Temp\RtmpMliwz3\downloaded_packages
```
**Connecting R to MySql**

Once the package is installed we create a connection object in R to connect to the database. It takes the username, password, database name and host name as input.

```r
# Create a connection Object to MySQL database.
# We will connect to the sample database named "sakila" that comes with MySql installation.
mysqlconnection = dbConnect(MySQL(), user='root', password='',
dbname='sakila', host='localhost')
```

# List the tables available in this database.
dbListTables(mysqlconnection)

When we execute the above code, it produces the following result:

```
[1] "actor"        "actor_info"
[3] "address"     "category"
[5] "city"        "country"
[7] "customer"    "customer_list"
[9] "film"        "film_actor"
[11] "film_category"  "film_list"
[13] "film_text"  "inventory"
[15] "language"   "nicer_but_slower_film_list"
[17] "payment"    "rental"
[19] "sales_by_film_category"  "sales_by_store"
[21] "staff"      "staff_list"
[23] "store"
```

**Querying the Tables**

We can query the database tables in MySql using the function `dbSendQuery()`. The query gets executed in MySql and the result set is returned using the R `fetch()` function. Finally it is stored as a data frame in R.
# Query the "actor" tables to get all the rows.
result = dbSendQuery(mysqlconnection, "select * from actor")

# Store the result in a R data frame object. n=5 is used to fetch first 5 rows.
data.frame = fetch(result, n=5)
print(data.frame)

When we execute the above code, it produces the following result:

<table>
<thead>
<tr>
<th>actor_id</th>
<th>first_name</th>
<th>last_name</th>
<th>last_update</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PENELope</td>
<td>GUINESS</td>
<td>2006-02-15 04:34:33</td>
</tr>
<tr>
<td>2</td>
<td>NICK</td>
<td>WAHLBERG</td>
<td>2006-02-15 04:34:33</td>
</tr>
<tr>
<td>3</td>
<td>ED</td>
<td>CHASE</td>
<td>2006-02-15 04:34:33</td>
</tr>
<tr>
<td>4</td>
<td>JENNIFER</td>
<td>DAVIS</td>
<td>2006-02-15 04:34:33</td>
</tr>
<tr>
<td>5</td>
<td>JOHNNY</td>
<td>LOLLOBRIGIDA</td>
<td>2006-02-15 04:34:33</td>
</tr>
</tbody>
</table>

**Query with Filter Clause**

We can pass any valid select query to get the result.

result = dbSendQuery(mysqlconnection, "select * from actor where last_name='TORN'")

# Fetch all the records(with n = -1) and store it as a data frame.
data.frame = fetch(result, n=-1)
print(data)

When we execute the above code, it produces the following result:

<table>
<thead>
<tr>
<th>actor_id</th>
<th>first_name</th>
<th>last_name</th>
<th>last_update</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>DAN</td>
<td>TORN 2006-02-15 04:34:33</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>KENNETH</td>
<td>TORN 2006-02-15 04:34:33</td>
</tr>
<tr>
<td>3</td>
<td>102</td>
<td>WALTER</td>
<td>TORN 2006-02-15 04:34:33</td>
</tr>
</tbody>
</table>
Updating Rows in the Tables

We can update the rows in a Mysql table by passing the update query to the dbSendQuery() function.

```r
dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")
```

After executing the above code we can see the table updated in the MySql Environment.

Inserting Data into the Tables

```r
dbSendQuery(mysqlconnection,
"insert into mtcars(row_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)
values('New Mazda RX4 Wag', 21, 6, 168.5, 110, 3.9, 2.875, 17.02, 0, 1, 4, 4)"
)
```

After executing the above code we can see the row inserted into the table in the MySql Environment.

Creating Tables in MySql

We can create tables in the MySql using the function dbWriteTable(). It overwrites the table if it already exists and takes a data frame as input.

```r
# Create the connection object to the database where we want to create the table.
mysqlconnection = dbConnect(MySQL(), user='root', password='', dbname='sakila', host='localhost')

# Use the R data frame "mtcars" to create the table in MySql.
# All the rows of mtcars are taken inot MySql.
dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)
```

After executing the above code we can see the table created in the MySql Environment.
Dropping Tables in MySql

We can drop the tables in MySql database passing the drop table statement into the `dbSendQuery()` in the same way we used it for querying data from tables.

```r
dbSendQuery(mysqlconnection, 'drop table if exists mtcars')
```

After executing the above code we can see the table is dropped in the MySql Environment.

R-Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the `pie()` function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is:

```r
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used:

- `x` is a vector containing the numeric values used in the pie chart.
- `labels` is used to give description to the slices.
- `radius` indicates the radius of the circle of the pie chart. (value between -1 and +1).
- `main` indicates the title of the chart.
- `col` indicates the color palette.
- `clockwise` is a logical value indicating if the slices are drawn clockwise or anti clockwise

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.
Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter main to add a title to the chart and another parameter is col which will make use of rainbow color pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).
**Example**

The below script will create and save the pie chart in the current R working directory.

```r
# Create data for the graph.
x <- c(21, 62, 10, 53)
# Give the chart file a name.
png(file = "city_title_colours.png")
# Plot the chart with title and rainbow color pallet.
pie(x, labels, main="City pie chart", col=rainbow(length(x)))
# Save the file.
dev.off()
null device
   1
> pie(x, labels, main="City pie chart", col=rainbow(length(x)))
```

![Pie chart example](image.png)
Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

```r
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London","New York","Singapore","Mumbai")
piepercent<- round(100*x/sum(x), 1)
# Give the chart file a name.
png(file = "city_percentage_legends.jpg")
# Plot the chart.
pie(x, labels=piepercent, main="City pie chart", col=rainbow(length(x)))
legend("topleft", c("London","New York","Singapore","Mumbai"), cex=0.8,
+ fill=rainbow(length(x)))
# Save the file.
dev.off()
null device
1
```

![City pie chart](image.png)

- London: 42.5%
- New York: 14.4%
- Singapore: 6.8%
- Mumbai: 36.3%
3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package plotrix has a function called pie3D() that is used for this.

```r
> # Get the library.
> library(plotrix)
> # Create data for the graph.
> x <- c(21, 62, 10, 53)
> lbl <- c("London","New York","Singapore","Mumbai")
> # Give the chart file a name.
> png(file = "3d_pie_chart.jpg")
> # Plot the chart.
> pie3D(x,labels=lbl,explode=0.1,
> + main="Pie Chart of Countries ")
> # Save the file.
> dev.off()
```

![Pie Chart of Countries](3d_pie_chart.jpg)
R-Bar Charts

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function barplot() to create bar charts. R can draw both vertical and horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is:

```
barplot(H,xlab,ylab,main, names.arg,col)
```

Following is the description of the parameters used:

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

Example

A simple bar chart is created using just the input vector and the name of each bar. The below script will create and save the bar chart in the current R working directory.

```r
> library()
> # Create the data for the chart.
> H <- c(7,12,20,3,41)
> # Give the chart file a name.
> png(file = "barchart.png")
> # Plot the bar chart.
> barplot(H)
> # Save the file.
> dev.off()
null device
1
> barplot(H)
```
When we execute the above code, it produces the following result:

![Bar Chart](image)

**Bar Chart Labels, Title and Colors**

The features of the bar chart can be expanded by adding more parameters. The main parameter is used to add title. The col parameter is used to add colors to the bars. The args.name is a vector having same number of values as the input vector to describe the meaning of each bar.

**Example**

The following script will create and save the bar chart in the current R working directory.

```r
# Create the data for the chart.
H <- c(7, 12, 28, 3, 41)
M <- c("Mar", "Apr", "May", "Jun", "Jul")
# Give the chart file a name.
png(file = "barchart_months_revenue.png")
# Plot the bar chart.
barplot(H, names.arg=M, xlab="Month", ylab="Revenue", col="blue",
+ main="Revenue chart", border="red")
# Save the file.
dev.off()

null device
1
barplot(H, names.arg=M, xlab="Month", ylab="Revenue", col="blue",
+ main="Revenue chart", border="red")
```

When we execute the above code, it produces the following result:

Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

```r
> # Create the input vectors.
> colors <- c("green","orange","brown")
> months <- c("Mar","Apr","May","Jun","Jul")
> regions <- c("East","West","North")
> 
> # Create the matrix of the values.
> Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11),nrow=3,ncol=5,byrow=TRUE)
> # Give the chart file a name.
> png(file = "barchart_stack.png")
```
> # Create the bar chart.
> barplot(Values, main="total revenue", names.arg=months, xlab="month", ylab="revenue", col=colors)
> # Add the legend to the chart.
> legend("topleft", regions, cex=1.3, fill=colors)
> # Save the file.
> dev.off()
null device 1
> barplot(Values, main="total revenue", names.arg=months, xlab="months", ylab="revenue", col=colors)
> legend("topleft", regions, cex=1.3, fill=colors)
R-Boxplots

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the boxplot() function.

Syntax

The basic syntax to create a boxplot in R is:

```
boxplot(x,data,notch,varwidth,names,main)
```

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.

Example

We use the data set "mtcars" available in the R environment to create a basic boxplot. Let's look at the columns "mpg" and "cyl" in mtcars.

```
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
   mpg  cyl
Mazda RX4  21.0  6
Mazda RX4 Wag 21.0  6
Datsun 710  22.8  4
Hornet 4 Drive 21.4  6
Hornet Sportabout 18.7  8
Valiant     18.1  6
```
Creating the Boxplot

The below script will create a boxplot graph for the relation between mpg(miles per gallon) and cyl (number of cylinders).

```r
> # Give the chart file a name.
> png(file = "boxplot.png")
> # Plot the chart.
> boxplot(mpg ~ cyl, data=mtcars,
> + xlab="Number of Cylinders",
> + ylab="Miles Per Gallon",
> + main="Mileage Data")
> # Save the file.
> dev.off()
null device
1
> boxplot(mpg ~ cyl, data=mtcars,
> + xlab="Number of Cylinders",
> + ylab="Miles Per Gallon",
> + main="Mileage Data")
```

![Mileage Data Boxplot](image-url)
Boxplot with Notch

We can draw boxplot with notch to find out how the medians of different data groups match with each other.

The below script will create a boxplot graph with notch for each of the data group.

```r
> # Give the chart file a name.
> png(file = "boxplot_with_notch.png")
> # 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"))
Warning message:
In bxp(list(stats = c(21.4, 22.8, 26, 30.4, 33.9, 17.8, 18.65, 19.7, : 
some notches went outside hinges ("box"): maybe set notch=FALSE
> # Save the file.
> dev.off()
null device
1
> 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"))
Warning message:
In bxp(list(stats = c(21.4, 22.8, 26, 30.4, 33.9, 17.8, 18.65, 19.7, :

```

![Mileage Data Boxplot](boxplot_with_notch.png)

```
R-Histograms

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using hist() function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is:

```
hist(v, main, xlab, xlim, ylim, breaks, col, border)
```

Following is the description of the parameters used:

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example

A simple histogram is created using input vector, label, col and border parameters. The script given below will create and save the histogram in the current R working directory.

```
> # Create data for the graph.
> v <- c(9,13,21,8,36,22,12,41,31,33,19)
> # Give the chart file a name.
> png(file = "histogram.png")
> # Create the histogram.
> hist(v, xlab="Weight", col="yellow", border="blue")
> # Save the file.
> dev.off()
null device
1
> hist(v, xlab="Weight", col="yellow", border="blue")
```
Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

```r
> # Create data for the graph.
> v <- c(9,13,21,8,36,22,12,41,31,33,19)
> # Give the chart file a name.
> png(file = "histogram_lim_breaks.png")
> # Create the histogram.
> hist(v,xlab="Weight",col="green",border="red",xlim = c(0,40), ylim = c(0,5),
+ breaks = 5 )
> # Save the file.
> dev.off()
null device
1
> hist(v,xlab="Weight",col="green",border="red",xlim = c(0,40), ylim = c(0,5),breaks=5)
```
R-Line Graphs

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data. The plot() function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is:

```
plot(v,type,col,xlab,ylab)
```

Following is the description of the parameters used:

- `v` is a vector containing the numeric values.
- `type` takes the value "p" to draw only the points, "i" 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.
Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

```r
> # Create the data for the chart.
> v <- c(7,12,28,3,41)
> # Give the chart file a name.
> png(file = "line_chart.jpg")
> # Plot the bar chart.
> plot(v,type="o")
> # Save the file.
> dev.off()
null device
   1
> plot(v,type="o")
```

![Line Chart Example](image)

Line Chart Title, Color and Labels

The features of the line chart can be expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.
Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the `lines()` function.

After the first line is plotted, the `lines()` function can use an additional vector as input to draw the second line in the chart.
> # Create the data for the chart.
> v <- c(7,12,28,3,41)
> t <- c(14,7,6,19,3)
> # Give the chart file a name.
> png(file = "line_chart_2_lines.jpg")
> # Plot the bar chart.
> plot(v,type="o",col="red",xlab="Month",ylab="Rain fall",main="Rain fall chart")
> lines(t, type="o", col="blue")
> # Save the file.
> dev.off()
null device
1
> plot(v,type="o",col="red",xlab="Month",ylab="Rain fall",main="Rain fall chart")
> lines(t, type="o", col="blue")
**R-Scatterplots**

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis. The simple scatterplot is created using the plot() function.

**Syntax**

The basic syntax for creating scatterplot in R is:

```r
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used:

- `x` is the data set whose values are the horizontal coordinates.
- `y` is the data set whose values are the vertical coordinates.
- `main` is the title of the graph.
- `xlab` is the label in the horizontal axis.
- `ylab` is the label in the vertical axis.
- `xlim` is the limits of the values of `x` used for plotting.
- `ylim` is the limits of the values of `y` used for plotting.
- `axes` indicates whether both axes should be drawn on the plot.

**Example**

We use the data set "mtcars" available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```r
> input <- mtcars[,c('wt','mpg')]
> print(head(input))

          wt  mpg
Mazda RX4  2.620 21.0
Mazda RX4 Wag 2.875 21.0
Datsun 710  2.320 22.8
Hornet 4 Drive 3.215 21.4
Hornet Sportabout 3.440 18.7
Valiant     3.460 18.1
```
Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

```R
> # Get the input values.
> input <- mtcars[,c('wt', 'mpg')]
> # Give the chart file a name.
> png(file = "scatterplot.png")
> # Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30
> plot(x=input$wt, y=input$mpg,
+ xlab="Weight",
+ ylab="Milage",
+ xlim=c(2.5,3.5),
+ ylim=c(15,30),
+ main="Weight vs Milage"
+ )
> # Save the file.
> dev.off()
null device
1
> plot(x=input$wt, y=input$mpg,
+ xlab="Weight",
+ ylab="Milage",
+ xlim=c(2.5,3.5),
+ ylim=c(15,30),
+ main="Weight vs Milage"
+ )
```

![Scatterplot Graph](scatterplot.png)
Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use pairs() function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is:

```
pairs(formula, data)
```

Following is the description of the parameters used:

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
> # Give the chart file a name.
> png(file = "scatterplot_matrices.png")
> # Plot the matrices between 4 variables giving 12 plots.
> # One variable with 3 others and total 4 variables.
> pairs(~wt+mpg+disp+cyl, data=mtcars, 
+   main="Scatterplot Matrix")
> # Save the file.
> dev.off()

null device
  1
> pairs(~wt+mpg+disp+cyl, data=mtcars, main="Scatterplot Matrix")
```
R-Mean, Median and Mode

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

Mean
It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function mean() is used to calculate this in R.

Syntax
The basic syntax for calculating mean in R is:

\[ \text{mean}(x, \text{trim} = 0, \text{na.rm} = \text{FALSE}, ...) \]
Following is the description of the parameters used:

- **x** is the input vector.
- **trim** is used to drop some observations from both end of the sorted vector.
- **na.rm** is used to remove the missing values from the input vector.

```r
> # Create a vector.
> x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
> # Find Mean.
> result.mean <- mean(x)
> print(result.mean)
[1] 8.22
```

### Applying Trim Option

When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.

When trim =0.3, 3 values from each end will be dropped from the calculations to find mean.

In this case the sorted vector is (-21, -5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (-21,-5,2) from left and (12,18,54) from right.

```r
> # Create a vector.
> x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
> # Find Mean.
> result.mean <- mean(x,trim=0.3)
> print(result.mean)
[1] 5.55
```

### Applying NA Option

If there are missing values, then the mean function returns NA.

To drop the missing values from the calculation use na.rm=TRUE, which means remove the NA values.

```r
> # Create a vector.
> x <- c(12,7,3,4.2,18,2,54,-21,8,-5,NA)
> # Find mean.
> result.mean <- mean(x)
> print(result.mean)
[1] NA
> # Find mean dropping NA values.
> result.mean <- mean(x,na.rm=TRUE)
> print(result.mean)
[1] 8.22
```
**Median**
The middle most value in a data series is called the median. The median() function is used in R to calculate this value.

**Syntax**
The basic syntax for calculating median in R is:

```r
median(x, na.rm = FALSE)
```

Following is the description of the parameters used:

- **x** is the input vector.
- **na.rm** is used to remove the missing values from the input vector.

```r
> # Create the vector.
> x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
> # Find the median.
> median.result <- median(x)
> print(median.result)
[1] 5.6
```

**Mode**
The mode is the value that has highest number of occurrences in a set of data. Unlike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

```r
> # Create the function.
> getmode <- function(v) {
+ uniqv <- unique(v)
+ uniqv[which.max(tabulate(match(v, uniqv)))]
+ }
> # Create the vector with numbers.
> v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)
> # Calculate the mode using the user function.
> result <- getmode(v)
> print(result)
[1] 2
> # Create the vector with characters.
> charv <- c("c","it","the","it","it")
> # Calculate the mode using the user function.
> result <- getmode(charv)
> print(result)
[1] "it"
```
**R-Function**

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

**Function Definition**

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows:

```r
function_name <- function(arg_1, arg_2, ...) {
  function body
}
```

**Function Components**

The different parts of a function are:

- **Function Name**: This is the actual name of the function. It is stored in R environment as an object with this name.

- **Arguments**: An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

- **Function Body**: The function body contains a collection of statements that defines what the function does.

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

R has many in-built functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions.
**Built-in Function**

Simple examples of in-built functions are seq(), mean(), max(), sum(x) and paste(...) etc. They are directly called by user written programs.

```r
> # Create a sequence of numbers from 32 to 44.
> print(seq(32,44))
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44
> # Find mean of numbers from 25 to 82.
> print(mean(25:82))
[1] 53.5
> # Find sum of numbers from 41 to 68.
> print(sum(41:68))
[1] 1526
```

**User-defined Function**

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```r
> # Create a function to print squares of numbers in sequence.
> new.function <- function(a) {
+ for(i in 1:a) {
+ b <- i^2
+ print(b)
+ }
+ .}
```

**Calling a Function**

```r
> # Create a function to print squares of numbers in sequence.
> new.function <- function(a) {
+ for(i in 1:a) {
+ b <- i^2
+ print(b)
+ }
+ }
```
Calling a Function without an Argument

> # Create a function without an argument.
> new.function <- function() {
> + for(i in 1:5) {
> + print(i^2)
> + }
> + }
> # Call the function without supplying an argument.
> new.function()
> [1] 1
> [1] 4
> [1] 9
> [1] 16
> [1] 25
> Calling a Function with Argument Values (by position and by name)

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

> # Create a function with arguments.
> new.function <- function(a,b,c) {
> + result <- a*b+c
> + print(result)
> + }
> # Call the function by position of arguments.
> new.function(5,3,11)
> [1] 26
> # Call the function by names of the arguments.
> new.function(a=11,b=5,c=3)
> [1] 58
Calling a Function with Default Argument

We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

> # Create a function with arguments.
> new.function <- function(a = 3, b = 6) {
+ result <- a*b
+ print(result)
+ }
> # Call the function without giving any argument.
> new.function()
[1] 18
> # Call the function with giving new values of the argument.
> new.function(9,5)
[1] 45

Lazy Evaluation of Function

Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

> # Create a function with arguments.
> new.function <- function(a, b) {print(a^2)
+ print(a)
+ print(b)
+ }
> # Evaluate the function without supplying one of the arguments.
> new.function(6)
[1] 36
[1] 6
Error in print(b) : argument "b" is missing, with no default
**R-Linear Regression**

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variables is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is:

\[ y = ax + b \]

Following is the description of the parameters used:

- \( y \) is the response variable.
- \( x \) is the predictor variable.
- \( a \) and \( b \) are constants which are called the coefficients.

**Steps to Establish a Regression**

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is:

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the \texttt{lm()} functions in R.
- Find the coefficients from the model created and create the mathematical equation using these.
- Get a summary of the relationship model to know the average error in prediction. Also called residuals.
- To predict the weight of new persons, use the \texttt{predict()} function in R.
**Input Data**

Below is the sample data representing the observations:

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

**lm() Function**

This function creates the relationship model between the predictor and the response variable.

Syntax The basic syntax for `lm()` function in linear regression is:

```
lm(formula, data)
```

Following is the description of the parameters used:

- `formula` is a symbol presenting the relation between x and y.
- `data` is the vector on which the formula will be applied.

```
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> # Apply the lm() function.
> relation <- lm(y~x)
> print(relation)
```

**Call:**

```
lm(formula = y ~ x)
```

**Coefficients:**

```
(Intercept) x
-38.4551 0.6746
```

---

**Get the Summary of the Relationship**
predict() Function

Syntax

The basic syntax for predict() in linear regression is:

predict(object, newdata)

Following is the description of the parameters used:

- object is the formula which is already created using the lm() function.
- newdata is the vector containing the new value for predictor variable.

```r
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> # Apply the lm() function.
> relation <- lm(y~x)
> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max
-6.3002 -1.6629  0.0412  1.8944  3.9775

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,   Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.16e-06
```

```r
> # The predictor vector.
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> # The response vector.
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> # Apply the lm() function.
> relation <- lm(y~x)
> # Find weight of a person with height 170.
> a <- data.frame(x=170)
> result <- predict(relation,a)
> print(result)
    1
76.22869
```
Visualize the Regression Graphically

```r
> # Create the predictor and response variable.
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> relation <- lm(y~x)
> # Give the chart file a name.
> png(file = "linearregression.png")
> # Plot the chart.
> plot(y,x,col="blue",main="Height & Weight Regression",
+ abline(lm(x~y)),cex = 1.3,pch=16,xlab="Weight in Kg",ylab="Height in cm")
> # Save the file.
> dev.off()
null device

> plot(y,x,col="blue",main="Height & Weight Regression",
+ abline(lm(x~y)),cex = 1.3,pch=16,xlab="Weight in Kg",ylab="Height in cm")
```

![Height & Weight Regression](linearregression.png)
R-Multiple Regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is:

\[ y = a + b_1x_1 + b_2x_2 + ... + b_nx_n \]

Following is the description of the parameters used:

- \( y \) is the response variable.
- \( a, b_1, b_2...b_n \) are the coefficients.
- \( x_1, x_2, ...x_n \) are the predictor variables.

We create the regression model using the \texttt{lm()} function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients.

\textbf{lm()} Function

This function creates the relationship model between the predictor and the response variable.

\textbf{Syntax}

The basic syntax for \texttt{lm()} function in multiple regression is:

\texttt{lm(y ~ x1+x2+x3...,data)}

Following is the description of the parameters used:

- \texttt{formula} is a symbol presenting the relation between the response variable and predictor variables.
- \texttt{data} is the vector on which the formula will be applied.

\textbf{Example}

\textbf{Input Data}

Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.
The goal of the model is to establish the relationship between "mpg" as a response variable with "disp", "hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

```r
> input <- mtcars[, c("mpg", "disp", "hp", "wt")]
> print(head(input))

     mpg disp  hp  wt
Mazda RX4   21.0 160 110 2.620
Mazda RX4 Wag 21.0 150 110 3.150
Datsun 710  22.8 108  93 2.875
Hornet 4 Drive 21.4 258 110 3.215
Hornet Sportabout 18.7 360 175 3.440
Valiant     18.1 225 105 3.460
```

### Create Relationship Model & get the Coefficients

```r
> input <- mtcars[, c("mpg", "disp", "hp", "wt")]
> # Create the relationship model.
> model <- lm(mpg ~ disp + hp + wt, data = input)
> # Show the model.
> print(model)

Call:
  lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:
(Intercept)      disp         hp         wt
  37.105505 -0.0009370 -0.0311555 -3.800891

> # Get the Intercept and coefficients as vector elements.
> cat("# # The Coefficient Values ", "\n")
# # The Coefficient Values
> a <- coef(model)[1]
> print(a)
(Intercept)
  37.10551

> Xdisp <- coef(model)[2]
> Xhp <- coef(model)[3]
> Xwt <- coef(model)[4]
> print(Xdisp)
  disp
-0.0009370091
> print(Xhp)
  hp
-0.0311555
> print(Xwt)
  wt
-3.800891
```
Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

\[ Y = a + \text{disp} \cdot x_1 + \text{hp} \cdot x_2 + \text{wt} \cdot x_3 \]

or

\[ Y = 37.15 + (-0.000937) \cdot x_1 + (-0.0311) \cdot x_2 + (-3.8008) \cdot x_3 \]

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is:

\[ Y = 37.15 + (-0.000937) \cdot 221 + (-0.0311) \cdot 102 + (-3.8008) \cdot 2.91 = 22.7104 \]

R-Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is:

\[ y = \frac{1}{1+e^{-(a+b_1x_1+b_2x_2+b_3x_3+\ldots)}} \]

Following is the description of the parameters used:

- \( y \) is the response variable.
- \( x \) is the predictor variable.
- \( a \) and \( b \) are the coefficients which are numeric constants.

The function used to create the regression model is the `glm()` function.

Syntax

The basic syntax for `glm()` function in logistic regression is:

```
glm(formula, data, family)```

Following is the description of the parameters used:

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. Its value is binomial for logistic regression.

**Example**

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column *am* which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

Create Regression Model

We use the `glm()` function to create the regression model and get its summary for analysis.

```r
> input <- mtcars[,c("am","cyl","hp","wt")]
> print(head(input))
  am  cyl  hp  wt
Mazda RX4      1 6 110 2.620
Mazda RX4 Wag  1 6 110 2.875
Datsun 710     1 4  93 3.220
Hornet 4 Drive 0 6 110 3.690
Hornet Sportabout 0 8 175 3.150
Valiant        0 6 105 3.620

> am.data = glm(formula=am ~ cyl + hp + wt, + data=input, + family=binomial)
> print(summary(am.data))
```

```r
call: glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:
    Min      1Q  Median      3Q     Max
  -2.17272 -0.14907 -0.01464  0.14116  1.27641

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)  19.70288     8.11637   2.428  0.01526 *
cyl          0.49760     1.07162   0.465  0.64913
hp           0.03259     0.01886   1.728  0.08406 .
wt          -9.14947     4.15332  -2.203  0.02763 *

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.2297 on 31 degrees of freedom
  Residual deviance: 9.8415 on 28 degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```
In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

**R-Normal Distribution**

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.

R has four in built functions to generate normal distribution.

```r
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

Following is the description of the parameters used in above functions:

- x is a vector of numbers.
- p is a vector of probabilities.
- n is number of observations(sample size).
- mean is the mean value of the sample data. It's default value is zero.
- sd is the standard deviation. It's default value is 1.

**dnorm()**

This function gives height of the probability distribution at each point for a given mean and standard deviation.
> # Create a sequence of numbers between -10 and 10 incrementing by 0.1.
> x <- seq(-10, 10, by = .1)
> # Choose the mean as 2.5 and standard deviation as 0.5.
> y <- dnorm(x, mean = 2.5, sd = 0.5)
> # Give the chart file a name.
> png(file = "dnorm.png")
> plot(x, y)
> # Save the file.
> dev.off()
null device
1
> plot(x, y)
**pnorm()**

This function gives the probability of a normally distributed random number to be less than the value of a given number. It is also called "Cumulative Distribution Function".

```r
> # Create a sequence of numbers between -10 and 10 incrementing by 0.2.
> x <- seq(-10, 10, by=.2)
> # Choose the mean as 2.5 and standard deviation as 2.
> y <- pnorm(x, mean=2.5, sd = 2)
> # Give the chart file a name.
> png(file = "pnorm.png")
> # Plot the graph.
> plot(x,y)
> # Save the file.
> dev.off()
```

qnorm()
**qnorm()**

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```r
> # Create a sequence of probability values incrementing by 0.02.
> x <- seq(0,1,by=0.02)
> # Choose the mean as 2 and standard deviation as 3.
> y <- qnorm(x,mean=2,sd=1)
> # Give the chart file a name.
> png(file = "qnorm.png")
> # Plot the graph.
> plot(x,y)
> # Save the file.
> dev.off()
null device
  1
> plot(x,y)
```

![R Graphics: Device 2 (ACTIVE)](image-url)
rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

```r
> # Create a sample of 50 numbers which are normally distributed.
> y <- rnorm(50)
> # Give the chart file a name.
> png(file = "rnorm.png")
> # Plot the histogram for this sample.
> hist(y, main = "Normal Distribution")
> # Save the file.
> dev.off()
null device
1
> hist(y, main = "Normal Distribution")
```

![Histogram of normally distributed numbers](Image.png)
R-Analysis of Covariance

We use Regression analysis to create models which describe the effect of variation in predictor variables on the response variable. Sometimes, if we have a categorical variable with values like Yes/No or Male/Female etc. The simple regression analysis gives multiple results for each value of the categorical variable. In such scenario, we can study the effect of the categorical variable by using it along with the predictor variable and comparing the regression lines for each level of the categorical variable. Such an analysis is termed as Analysis of Covariance also called as ANCOVA.

Example

Consider the R built in data set mtcars. In it we observer that the field "am" represents the type of transmission (auto or manual). It is a categorical variable with values 0 and 1. The miles per gallon value(mpg) of a car can also depend on it besides the value of horse power("hp").

We study the effect of the value of "am" on the regression between "mpg" and "hp". It is done by using the aov() function followed by the anova() function to compare the multiple regressions.

Input Data

Create a data frame containing the fields "mpg", "hp" and "am" from the data set mtcars. Here we take "mpg" as the response variable, "hp" as the predictor variable and "am" as the categorical variable.

```r
> input <- mtcars[,c("am","mpg","hp")]
> print(head(input))
  am   mpg   hp
Mazda RX4 1 21.0 110
Mazda RX4 Wag 1 21.0 110
Datsun 710 1 22.8  93
Hornet 4 Drive 0 21.4 110
Hornet Sportabout 0 18.7 175
Valiant 0 16.4 120
```

ANCOVA Analysis

We create a regression model taking "hp" as the predictor variable and "mpg" as the response variable taking into account the interaction between "am" and "hp".
Model with interaction between categorical variable and predictor variable

```r
> # Get the dataset.
> input <- mtcars
> # Create the regression model.
> result <- aov(mpg~hp*am,data=input)
> print(summary(result))

             Df Sum Sq Mean Sq F value    Pr(>F)
hp           1  678.4  678.44  77.391 1.50e-09 ***
am           1  202.2  202.22  23.072 4.75e-05 ***
hp:am        1    0.0    0.00  0.0010 0.9811
Residuals    28 245.41  8.76
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05. But the interaction between these two variables is not significant as the p-value is more than 0.05.

Model without interaction between categorical variable and predictor variable

```r
> # Get the dataset.
> input <- mtcars
> # Create the regression model.
> result <- aov(mpg~hp+am,data=input)
> print(summary(result))

             Df Sum Sq Mean Sq  F value    Pr(>F)
hp            1  678.4  678.44  70.1500 7.63e-10 ***
am            1  202.2  202.22  21.8900 3.46e-05 ***
Residuals     29 245.41  8.43
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05.

Comparing Two Models

Now we can compare the two models to conclude if the interaction of the variables is truly insignificant. For this we use the anova() function.
As the p-value is greater than 0.05 we conclude that the interaction between horse power and transmission type is not significant. So the mileage per gallon will depend in a similar manner on the horse power of the car in both auto and manual transmission mode.

### R-Time Series Analysis

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

The time series object is created by using the `ts()` function.

**Syntax**

The basic syntax for `ts()` function in time series analysis is:

```r
timeseries.object.name <- ts(data, start, end, frequency)
```

Following is the description of the parameters used:

- **data** is a vector or matrix containing the values used in the time series.
- **start** specifies the start time for the first observation in time series.
- **end** specifies the end time for the last observation in time series.
- **frequency** specifies the number of observations per unit time. Except the parameter "data" all other parameters are optional.
Example

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

```r
> # Get the data points in form of a R vector.
> rainfall <-
+ c(799.1, 174.8, 865.1, 1334.6, 635.4, 918.5, 685.5, 998.6, 784.2, 985.0, 882.8, 1071.0)
> # Convert it to a time series object.
> rainfall.timeseries <- ts(rainfall,start=c(2012,1),frequency=12)
> # Print the timeseries data.
> print(rainfall.timeseries)

Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
2012 799.0 1174.8 865.1 1334.6 635.4 918.5 685.5 998.6 784.2 985.0 882.8 1071.0
```

> # Give the chart file a name.
> png(file = "rainfall.png")
> # Plot a graph of the time series.
> plot(rainfall.timeseries)
> # Save the file.
> dev.off()
null device
1
> plot(rainfall.timeseries)

![Rainfall Time Series Plot](image-url)
**Different Time Intervals**

The value of the frequency parameter in the `ts()` function decides the time intervals at which the data points are measured. A value of 12 indicates that the time series is for 12 months. Other values and their meaning are as follows:

- frequency = 12 pegs the data points for every month of a year.
- frequency = 4 pegs the data points for every quarter of a year.
- frequency = 6 pegs the data points for every 10 minutes of an hour.
- frequency = 24*6 pegs the data points for every 10 minutes of a day.

**Multiple Time Series**

We can plot multiple time series in one chart by combining both the series into a matrix.

```r
> # Get the data points in form of a R vector.
> rainfall1 <-
+ c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985.8,882.8,1071)
> rainfall2 <-
+ c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)
> # Convert them to a matrix.
> combined.rainfall <- matrix(c(rainfall1, rainfall2), nrow=12)
> # Convert it to a time series object.
> rainfall.timeseries <- ts(combined.rainfall, start=c(2012,1), frequency=12)
> # Print the timeseries data.
> print(rainfall.timeseries)
```

<table>
<thead>
<tr>
<th></th>
<th>Series 1</th>
<th>Series 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>799.0</td>
<td>655.0</td>
</tr>
<tr>
<td>Feb</td>
<td>1174.8</td>
<td>1306.9</td>
</tr>
<tr>
<td>Mar</td>
<td>865.1</td>
<td>1323.4</td>
</tr>
<tr>
<td>Apr</td>
<td>1334.6</td>
<td>1172.2</td>
</tr>
<tr>
<td>May</td>
<td>635.4</td>
<td>562.2</td>
</tr>
<tr>
<td>Jun</td>
<td>918.5</td>
<td>824.0</td>
</tr>
<tr>
<td>Jul</td>
<td>685.5</td>
<td>822.4</td>
</tr>
<tr>
<td>Aug</td>
<td>998.6</td>
<td>1265.5</td>
</tr>
<tr>
<td>Sep</td>
<td>784.2</td>
<td>799.6</td>
</tr>
<tr>
<td>Oct</td>
<td>985.0</td>
<td>1105.6</td>
</tr>
<tr>
<td>Nov</td>
<td>882.8</td>
<td>1106.7</td>
</tr>
<tr>
<td>Dec</td>
<td>1071.0</td>
<td>1337.8</td>
</tr>
</tbody>
</table>
> # Give the chart file a name.
> png(file = "rainfall_combined.png")
> # Plot a graph of the time series.
> plot(rainfall.timeseries, main = "Multiple Time Series")
> # Save the file.
> dev.off()
> null device
> 1
> plot(rainfall.timeseries, main = "Multiple Time Series")
R-Decision Tree

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. It is mostly used in Machine Learning and Data Mining applications using R.

Examples of use of decision trees is - predicting an email as spam or not spam, predicting of a tumor is cancerous or predicting a loan as a good or bad credit risk based on the factors in each of these. Generally, a model is created with observed data also called training data. Then a set of validation data is used to verify and improve the model. R has packages which are used to create and visualize decision trees. For new set of predictor variable, we use this model to arrive at a decision on the category (yes/No, spam/not spam) of the data.

The R package "party" is used to create decision trees.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

```r
> install.packages("party")
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.cnr.berkeley.edu/bin/windows/contrib/3.3/party_1.2-3.zip'
Content type 'application/zip' length 715135 bytes (698 KB)
downloaded 698 KB

package 'party' successfully unpacked and MD5 sums checked
```

The downloaded binary packages are in

```
C:\Users\Tf452622\AppData\Local\Temp\RtmpTkzhQ\downloaded_packages
```

The package "party" has the function ctree() which is used to create and analyze decision tree.

Syntax

The basic syntax for creating a decision tree in R is:

```r
ctree(formula, data)
```

Following is the description of the parameters used:

- formula is a formula describing the predictor and response variables.
- data is the name of the data set used.
Input Data

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoeSize","score" and whether the person is a native speaker or not.

Here is the sample data.

```r
> # Load the party package. It will automatically load other dependent packages.
> library(party)
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

   as.Date, as.Date.numeric

Loading required package: sandwich
Warning messages:
1: package 'party' was built under R version 3.3.3
2: package 'strucchange' was built under R version 3.3.3
3: package 'zoo' was built under R version 3.3.3
4: package 'sandwich' was built under R version 3.3.3
> # Print some records from data set readingSkills.
> print(head(readingSkills))
nativeSpeaker age shoeSize score
1   yes     5 24.83189   32.29385
2   yes     6 25.95238   36.63105
3   no      11 30.42170   49.60593
4   yes     7 28.66450   40.28456
5   yes     11 31.88207   55.46085
6   yes     10 30.07843   52.83124
```

Example

We will use the ctree() function to create the decision tree and see its graph.
> # Load the party package. It will automatically load other dependent packages.
> library(party)
> # Create the input data frame.
> input.dat <- readingSkills[c(1:105),]
> # Give the chart file a name.
> png(file = "decision_tree.png")
> # Create the tree.
> output.tree <- ctree(
+ nativeSpeaker ~ age + shoeSize + score,
+ data = input.dat)
> # Plot the tree.
> plot(output.tree)
> # Save the file.
> dev.off()
null device
1
> plot(output.tree)
**R-Random Forest**

In the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an OOB (Out-of-bag) error estimate which is mentioned as a percentage.

The R package "randomForest" is used to create random forests.

**Install R Package**

Use the below command in R console to install the package. You also have to install the dependent packages if any.

```r
> install.packages("randomForest")
```

Content type 'application/zip' length 179017 bytes (174 KB)
downloaded 174 KB

The downloaded binary packages are in

C:\Users\IT452622\AppData\Local\Temp\RtmpcTkzhQ\downloaded_packages

The package "randomForest" has the function randomForest() which is used to create and analyze random forests.

**Syntax**

The basic syntax for creating a random forest in R is:

```r
randomForest(formula, data)
```

Following is the description of the parameters used:

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

**Input Data**

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoeseize","score" and whether the person is a native speaker.
Here is the sample data.

```r
> # Load the party package. It will automatically load other required packages.
> library(party)
> # Print some records from data set readingSkills.
> print(head(readingSkills))

nativeSpeaker  age  shoeSize  score
1     yes   5  24.83189  32.29385
2     yes   6  25.95238  36.53015
3       no  11  30.42170  49.60593
4     yes   7  28.66450  40.28456
5     yes  11  31.88207  55.46085
6     yes  10  30.07843  52.83124
```

**Example**

We will use the randomForest() function to create the decision tree and see it's graph.

```r
> # Load the party package. It will automatically load other required packages.
> library(party)
> library(randomForest)
> # Create the forest.
> output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,
+ data=readingSkills)
> # View the forest results.
> print(output.forest)

Call:
  randomForest(formula = nativeSpeaker ~ age + shoeSize + score, data = readingSkills)  
Type of random forest: classification  
Number of trees: 500  
No. of variables tried at each split: 1

OOB estimate of  error rate: 1%

Confusion matrix:
  no yes class.error
no 99  1     0.01
yes 1 99    0.01
```

From the random forest shown above we can conclude that the shoesize and score are the important factors deciding if someone is a native speaker or not. Also the model has only 1% error which means we can predict with 99% accuracy.
R-Survival Analysis

Survival analysis deals with predicting the time when a specific event is going to occur. It is also known as failure time analysis or analysis of time to death. For example predicting the number of days a person with cancer will survive or predicting the time when a mechanical system is going to fail.

The R package named survival is used to carry out survival analysis. This package contains the function Surv() which takes the input data as a R formula and creates a survival object among the chosen variables for analysis. Then we use the function survfit() to create a plot for the analysis.

Install Package

> install.packages("survival")
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.r-project.org/bin/windows/contrib/3.3/survival_2.41-3.zip'
Content type 'application/zip' length 5250851 bytes (5.0 MB)
downloaded 5.0 MB

package 'survival' successfully unpacked and MD5 sums checked
Warning: cannot remove prior installation of package 'survival'

The downloaded binary packages are in
  C:\Users\TT452622\AppData\Local\Temp\RtmpKCEZ99\downloaded_packages

Syntax

The basic syntax for creating survival analysis in R is:

Surv(time,event)

survfit(formula)

Following is the description of the parameters used:

- **time** is the follow up time until the event occurs.
- **event** indicates the status of occurrence of the expected event.
- **formula** is the relationship between the predictor variables.
Example

We will consider the data set named "pbc" present in the survival packages installed above. It describes the survival data points about people affected with primary biliary cirrhosis (PBC) of the liver. Among the many columns present in the data set we are primarily concerned with the fields "time" and "status". Time represents the number of days between registration of the patient and earlier of the event between the patient receiving a liver transplant or death of the patient.

# Load the library.
library("survival")

# Print first few rows.
print(head(pbc))

     id time status trt albumin copper alk.phos age sex ascites hepatospiders edema  bili chol
1 1     400     2     f 58.76523      2.66 156 1718.0 137.951 0 1 1 1 0.0 14.5 261
2 2    4500     0     f 56.44627      4.14  54  7394.8 113.520 0 1 1 0 1.1 302
3 3   1012     2     m 70.07255      3.48  210  516.0  96.100 0 0 0 0.5 1.4 176
4 4   1925     2     f 54.74059      2.54  64  6121.8  60.630 0 1 1 0.5 1.8 244
5 5   1584     1     f 38.10541      3.53  143  671.0 113.150 0 0 1 0.0 3.4 279
6 6  2503     2     f 66.25873      3.98  50  944.0  93.000 0 1 0 0.0 0.8 248

    trig platelet prot time stage
1   172    190    12.2    4
2    88    221    10.6    3
3    55    151    12.0    4
4    92    183    10.3    4
5    72    136    10.9    3
6    63     NA     11.0    3
Applying `Surv()` and `survfit()` Function

```r
# Load the library.
library("survival")

# Create the survival object.
survfit(Surv(pbc$time, pbc$status == 2) ~ 1)

# Give the chart file a name.
png(file = "survival.png")

# Plot the graph.
plot(survfit(Surv(pbc$time, pbc$status == 2) ~ 1))

# Save the file.
dev.off()

Call: survfit(formula = Surv(pbc$time, pbc$status == 2) ~ 1)

n  events median 0.95LCL 0.95UCL
418    161     3395    3090  3853
```
R-Chi Square Test

Chi-Square test is a statistical method to determine if two categorical variables have a significant correlation between them. Both those variables should be from same population and they should be categorical like - Yes/No, Male/Female, Red/Green etc.

For example, we can build a data set with observations on people's ice-cream buying pattern and try to correlate the gender of a person with the flavor of the ice-cream they prefer. If a correlation is found we can plan for appropriate stock of flavors by knowing the number of gender of people visiting.

Syntax

The function used for performing chi-Square test is `chisq.test()`. The basic syntax for creating a chi-square test in R is:

```
chisq.test(data)
```

Following is the description of the parameters used:

- **data** is the data in form of a table containing the count value of the variables in the observation

Example

We will take the Cars93 data in the "MASS" library which represents the sales of different models of car in the year 1993.
The above result shows the dataset has many Factor variables which can be considered as categorical variables. For our model we will consider the variables "Airbags" and "Type". Here we aim to find out any significant correlation between the types of car sold and the type of air bags. If correlation is observed we can estimate which types of cars can sell better with what types of air bags.

> # Load the library.
> library("MASS")
> # Create a data frame from the main data set.
> car.data <- data.frame(Cars93$AirBag, Cars93$Type)
> # Create a table with the needed variables.
> print(car.data)
The result shows the p-value of less than 0.05 which indicates a strong correlation.

**Missing Values**

- The missing values in R are represented by symbol `NA` (not available).
- The impossible values in R are represented by `NaN` (not a number). Example for impossible value could be value dividing by zero.
  - R language uses same symbol for both numeric and character. A missing value is one whose value is unknown.
  - `NA` is a special value whose properties are different from other values. `NA` is one of the very few reserved words in R: you cannot give anything this name. (Because R is case-sensitive, `na` and `Na` are okay to use, although it is not recommended.)
  - **Missing values are often legitimate:** values really are missing in real life. NAs can arise when you read in an Excel spreadsheet with empty cells, for example.
  - You will also see `NA` when you try certain operations that are illegal or don't make sense. Here are some examples of operations that produce NA's.
One of the most common operations which produce NA is illegal conversion of numeric or character and out of range where we try to access out of given bounds.

```
> var (8)              # Variance of one number
[1] NA
> as.numeric (c("1", "2", "three", "4"))  # Illegal conversion
[1] 1 2 NA 4
Warning message:
NAs introduced by coercion
> c(1, 2, 3)[4]        # Vector subscript out of range
[1] NA
> NA - 1               # Most operations on NAs produce NAs
[1] NA
```

Note that if you specify a row number out of range for a data frame, that's not an error. You just get a row full of NAs. Interestingly, if you specify a row index of a matrix that's too big, you get a different response altogether. That is an error. It's also an error to specify too big an index for a column of either a matrix or a data frame.

Most of operations done or involving NA produces NA.

Example
```
> a <- data.frame (a = 1:3, b = 2:4)  # Data frame row subscript out of range
> a[4,]                              # The first NA there is the row number
   a b
  NA NA NA
> a[4,]                              # Specifying a non-existent column just produces an error
Error in `[.data.frame'(a, , 4) : undefined columns selected
# Here's one that's particularly irksome
#
> a[1,2] <- NA                       # Suppose you have an NA in your dataframe...
> a[a$b < 4,]                         # ...and you try to index on that column
   a b
  NA NA NA
  2 2 3
```

Here the length() would return only the count rather than checking for the type of data.
Handling and detecting missing values

We can not find missing values by looking at \( x == \text{NA} \). Like most other functions, the == operator returns NA when either argument is NA. The is.na() function will find missing values for you: this function returns a logical vector the same length as its argument, with T for missing values and F for non-missing. It's fairly common to want to know the index of the missing values, and the which() function will help do this for you.

Consider the example below

```
> x # Here's my vector
[1]  1  2 NA  4
> is.na(x) # Is it NA?
[1] F  F  T  F # Answer: no, no, yes, no.
> which(is.na(x)) # Which one is NA?
[1] 3 # Answer: the third one
```

```
The code which can be used to find all the rows in a data frame with at least one NA

> unique (unlist (lapply (your.data.frame, function (x) which (is.na (x)))))
```

The function is.finite() determines which values are “regular” values.

```
is.finite(c(1, Inf, NaN, NA))
## [1]  TRUE FALSE FALSE FALSE
```

Filter functions to handle NA

Math functions generally have a way to exclude missing values in their calculations. mean(), median(), colSums(), var(), sd(), min() and max() all take the na.rm argument.

When this is TRUE, missing values are omitted. The default is FALSE, meaning that each of these functions returns NA if any input number is NA.

R's modeling functions accept an na.action argument that tells the function what to do when it encounters an NA.

This causes the modeling function to call one of the missing value filter functions. These functions replace the original data set by a new data set in which the NAs have been altered. The default setting is na.omit, which excludes all rows with any missing values.

An alternative is na.action=na.fail, which just stops when it encounters any missing values. This is useful if you didn't know you had any.

The filter functions:
• **na.fail**: Stop if any missing values are encountered
• **na.omit**: Drop out any rows with missing values anywhere in them and is discarded forever.
• **na.exclude**: drops out rows with missing values
• **na.pass**: take no action

---

**Data cleaning or How to clean Data**

Data cleaning, or data preparation, is an essential part of statistical analysis. In fact, in practice it is often more time-consuming than the statistical analysis itself. Data cleaning may profoundly influence the statistical statements based on the data. Typical actions like imputation or outlier handling obviously influence the results of a statistical analyses.

For this reason, data cleaning should be considered a statistical operation, to be performed in a reproducible manner. The R statistical environment provides a good environment for reproducible data cleaning since all cleaning actions can be scripted and therefore reproduced.

---

Each box represents data in a certain state while each arrow represents the activities needed to get from one state to the other.

---

• **Raw Data**: The data “as is” may lack headers, contain wrong data types (e.g., numbers stored as strings), wrong category labels, unknown or unexpected character encoding and
so on. Reading such files into an R data.frame directly is either difficult or impossible without some sort of preprocessing.

- **Technically correct data**: The data can be read into an R data.frame, with correct names, types and labels, without further trouble. However, that does not mean that the values are error-free or complete.

- **Consistent data**: The data is ready for statistical inference. It is the data that most statistical theories use as a starting point. Ideally, such theories can still be applied without taking previous data cleaning steps into account. In practice however, data cleaning methods like imputation of missing values will influence statistical results and so must be accounted for in the following analyses or interpretation thereof.

- **Statistical results**: The results of the analysis have been produced and can be stored for reuse.

- **Formatted output**: The results in tables and figures ready to include in statistical reports or publications.

- It is always recommended to store input data for each stage separately for reuse. Each step between the stages may be performed by a separate R script for reproducibility.

**Consider an example to remove outliers and duplicates**

➢ **Na.string**

Note: formhub.R makes is easy to download and work with datasets on formhub

```
library(formhub)
my_data <- formhubRead("~/Downloads/MyDataSet.csv", "~/Downloads/MyForm.json")
```

The above code is used to read data set.

Next selecting a variable that you want to do outlier work on. Let's look at the total amount of female pupils per school for this particular data set, labeled as num_students_total_gender.num_students_female.

```
library(ggplot2) #install.packages('ggplot2') may need to be run if you don't have the package
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools")
```

Let us consider the graph
Plot 1

A quick look at the plot 1 tells us that there are a couple of female student outliers that are quite high - as indicated by the extension of x-axis to 5000. Zooming in our plot may help look at the distribution better:

```r
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools") + xlim(c(0, 1500))
```

The above code will further zoom into the table considering x axis values from 0 to 1500.

Plot 2
In plot 2 you can observe that there is a spike at 1000. We can observe the values more closely with the below code.

```r
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools") + xlim(c(990, 1100))
```

Plot 3

```r
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools") + xlim(c(990, 1010))
```

Plot 4
With the help of plot 4 you can observe that the spike is of the value “999”, which (in its negative version) is often used as a “Do Not Know” type of value in surveys.

R lets us deal with individual values like this by specifying a na.strings parameter when reading in csvs. We can get rid of this value by re-reading our dataset while providing the na.strings parameter by using `na.string=c("999")`.

The plot below does not contain the spike near 1000.

![Plot](image)

It is always recommended when you use 999 or -999 or other values for “NA” values. You need to include those values in na.string.

- **Outlier cut-offs**

Super-high outlier is still present at the dataset. At zoom level, we that the vast majority of schools have less than 500 female pupils. For the sake of crudely setting our outlier parameters, let's say that any facility reporting to have over 1000 female pupils will be counted as an outlier.

**Note:** You can use the below function code in your R scripts to do outlier replacements, but do note that the `outlierReplace` function will replace data in your dataframe directly.

```R
# install.packages('data.table') may need to be run if you don't have the # package
library(data.table)
outlierReplace = function(dataframe, cols, rows, newValue = NA) {
  if (any(rows)) {
    set(dataframe, rows, cols, newValue)
  }
}
```
The function **outlierReplace** is used in the example dataset, where we'll replace all values in the column `num_students_total_gender.num_students_female`, for all rows in which the value is > 1000, with NA. Afterwards, we'll plot the graph without adjusting the x-axis, and see that the extreme value has been removed.

```r
outlierReplace(my_data, "num_students_total_gender.num_students_female",
which(my_data$num_students_total_gender.num_students_female >
1000), NA)
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools")
```

It is also possible to use the **outlierReplace** function to change the value of more than one data point. Using the same outlier limit of 1000 for instance, we can change both the number of female pupils and the total number of pupils to NA like so:

```r
outlierReplace(my_data, c("num_students_total_gender.num_students_female",
"num_students_total_gender.num_students_total"),
which(my_data$num_students_total_gender.num_students_female > 1000), NA)
```

Finally, instead of changing outliers to NA, we could make them equal to a maximal number. To do this, and show you a clear results, we'll take all observations with more than 500 female students, and cap them at 500.

```r
outlierReplace(my_data, "num_students_total_gender.num_students_female",
which(my_data$num_students_total_gender.num_students_female >
500), 500)
qplot(data = my_data, x = num_students_total_gender.num_students_female) + ylab("Number of Schools")
```
Replacing data has to done with extreme care

Replacing data is hard to undo easily, so be careful with functions like outlierReplace. In order to undo, we will have to re-read our dataset, and re-perform all the actions before the replace. Thankfully, however, we haven't saved our data, and there is only one thing we did before the replace, which is easy to re-create:

```r
my_data <- formhubRead("~/Downloads/MyDataSet.csv", "~/Downloads/MyForm.json")
outlierReplace(my_data, c("num_students_total_gender.num_students_female",
"num_students_total_gender.num_students_total"),
which(my_data$num_students_total_gender.num_students_female > 1000, NA))
```

Saving the data set

It is important to save data set after performing the above operations.
There are two ways to do the save.

If you want all the form information preserved (and maybe the ability to run functions like replaceHeaderNamesWithLabels in the future, you can save the Data object as is, in an .rds file. Only R will be able to read this file.

```r
saveRDS(my_data, file = "~/Desktop/MyFormhubData.rds")
# later, you can do my_data <- loadRDS('~/Desktop/MyFormhubData.rds') to
# re-load this same object
```

If you need a widely usable file, then use data.frame, and save the data frame, for example as a csv.

```r
write.csv(data.frame(my_data), file = "~/Desktop/MyFormhubData.csv")
```

### How to run analyses

#### Numerical Measures

As a statistics analysis software, R has powerful function. From doing basic math operations to all kinds of analysis by using different models and analysis packages.

To start with, R is a calculator.

```
> 2 + 2
[1] 4
> 12 + (2 + 2) - 8
[1] 8
> 100 * 2017
[1] 2017000
> sqrt(2017)
[1] 44.91102
```

R can be used to calculate interest and annual problems. With 0.75% interest rate, $100 earns $7.76 interest.

```
> 100 * (1 + 0.0075) ^ 10 - 100
[1] 7.758255
```

R can read pi. R also can read trigonometric function

```
> pi
> sin(pi/2)
> pi * 5 ^ 2 > tan(pi/2)
```

To analysis data, R needs data to be load in. R can calculate average, standard deviation, variance and all these similar values by the function such as mean, sd, var, min, max, median, range, and quantile.
There is a data frame called faithful in the R system. To show this data frame, type `head()` function with the data frame name inside of the brackets.

```r
> head(faithful)
  eruptions waiting
1   3.600    79
2   1.800    54
3   3.333    74
4   2.283    62
5   4.533    85
6   2.883    55
```

To find the mean eruption duration in the data set faithful, set eruptions equal to duration and apply mean function to the set.

```r
> head(faithful)
  eruptions waiting
1   3.600    79
2   1.800    54
3   3.333    74
4   2.283    62
5   4.533    85
6   2.883    55
> duration <- faithful$eruptions
> mean(duration)
[1] 3.487783
```

With similar method by applying different functions like `max`, `min`, `median`, maximum value, minimum value, median etc can be computed. The difference between maximum and minimum is the range. The two function can be subtracted.

```r
> head(faithful)
  eruptions waiting
1   3.600    79
2   1.800    54
3   3.333    74
4   2.283    62
5   4.533    85
6   2.883    55
> duration <- faithful$eruptions
> mean(duration)
[1] 3.487783
> min(duration)
[1] 1.6
> max (duration)
[1] 5.1
> median (duration)
[1] 4
> max (duration) - min (duration)
[1] 3.5
```
In statistics, the first **quartile**, or lower quartile, means the value that cuts off the first 25% of the data when it is sorted in ascending order. The second **quartile**, or median, is the value that cuts off the first 50%. The third **quartile**, or upper quartile, is the value that cuts off the first 75%.

```
> quantile (duration)
0%   25%   50%   75%   100%
1.60000 2.16275 4.00000 4.45425 5.10000
```

The n\(^{th}\) **percentile** of an observation variable is the value that cuts off the first n percent of the data values when it is sorted in ascending order. **Quartile** function is using for n\(^{th}\) percentile too. c(.31, .48, .99) stands for the 31\(^{st}\), 48\(^{th}\) and 99\(^{th}\) percentiles of the eruption durations in faithful.

```
> quantile(duration, c(.31, .48, .99))
  31%  48%  99%
2.35017 3.95000 5.00957
```

The **interquartile range** of an observation variable is the difference of its upper and lower quartiles. Written as IQR function.

```
> IQR (duration)
[1] 2.2915
```

The **box plot** of an observation variable is a graphical representation to show the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum.

```
> boxplot(duration, horizontal=TRUE)
```
> boxplot(duration, horizontal=FALSE)
The **variance** is a numerical measure of how the data values is dispersed around the mean. Variance can be calculated by the following formula.

\[
s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

> var(duration)
[1] 1.302728

**Standard deviation** is the square root of its variance.

> sd(duration)
[1] 1.141371

The **covariance** of two variables \( x \) and \( y \) in a data set measures how the two are linearly related. The linear relationship is positive if covariance is positive. The linear relationship is negative if covariance is negative. Similar with variance, covariance can be calculated by

\[
s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})
\]

> duration = faithful$eruptions
> waiting = faithful$waiting
> cov(duration, waiting)
[1] 13.97781

**Central moment** the moment is the probability distribution composed by the mean of a group of random variables. The formula below is the way to calculate central moment.

\[
\mu_k^2 = \frac{1}{N} \sum_{i=1}^{n} (x_i - \mu)^k
\]

> library(e1071)  # load e1071
> duration = faithful$eruptions  # eruption durations
> moment(duration, order=3, center=TRUE)
[1] -0.6149

**Skewness** is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. A negative skewness or called left-skewed distribution has longer left tail and the mass of the distribution is concentrated on the right of the figure. It stands for the mean of the data values is less than the median. A positive skewness or called right-skewed distribution has longer right tail and the mass of the distribution is concentrated on the left of the figure. It stands for the mean of the data values is larger than the median. Skewness is defined by

\[
\gamma_1 = \frac{\mu_3}{\mu_3^2}
\]
Probability Distributions

The binomial distribution is the discrete probability distribution to describe results of \( n \) independent trials in an experiment. Each trial can have only two outcomes which is success or failure. If the probability of a successful trial is \( p \), the probability of a fair trial is \( 1 - p \), then the probability of having \( x \) successful outcomes in an experiment of \( n \) independent trials is as follows

\[
f(x) = \binom{n}{x} p^x (1 - p)^{n-x}
\]

Example

Suppose there are 20 multiple choice questions in an exam and each question has five choice of answers but only one is correct. If all the answers are picking by random, and each answer is independent. What is the probability of having 2 or less incorrect answers?

The possibility for get one answer wrong is 0.8. And there are 20 questions. So the probability that there are exactly 2 answers are wrong is

\[
\text{> dbinom}(2, \text{size}=20, \text{prob}=0.8)
\]

[1] 3.18767e-11

The probability that there are 2 answers are wrong, 1 answer is wrong and 0 answer is wrong can be compute as either of the following formulas.

\[
\text{> dbinom}(2, \text{size}=20, \text{prob}=0.8)+
\text{ dbinom}(1, \text{size}=20, \text{prob}=0.8)+
\text{ dbinom}(0, \text{size}=20, \text{prob}=0.8)
\]

[1] 3.272606e-11

\[
\text{> pbinom}(2, \text{size}=20, \text{prob}=0.8)
\]

[1] 3.272606e-11

The Poisson distribution is the probability distribution of independent event occurrences in an interval. If \( \lambda \) is the mean occurrence per interval, then the probability of having \( x \) occurrences within a given interval is:

\[
f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x = 0, 1, 2, 3, 4, \ldots
\]

Example

If there are four students entering ITS per minute on average, find the probability of having seven or more students entering ITS in a particular minute.

The probability of six or less students entering ITS can be calculated by \texttt{ppois}.
It is counting the low tail.

The probability of seven or more students entering ITS can be either computed by 1 minus the probability of six or less students entering ITS or by \texttt{ppois}.

\begin{verbatim}
> 1 - 0.889326 > ppois(6, lambda=4, lower=FALSE)
[1] 0.110674  [1] 0.110674
\end{verbatim}

The \textbf{continuous uniform distribution} is the probability distribution of random number selection from the continuous interval between a and b. Its density function is defined by the following.

\[
f(x) = \begin{cases} 
\frac{1}{b-a} & \text{for } x \in [a, b] \\
0 & \text{otherwise}
\end{cases}
\]

\textbf{Example}

Select ten random numbers between one and three.

\texttt{Runif} command picks any ten random number between one to three.

\begin{verbatim}
> runif(10, min=1, max=3) 
[1]  1.363119  1.055976  2.407588  2.044956  1.543327  2.300288  1.374274  2.492826  1.793663  2.446566
> runif(10, min=1, max=3) 
> runif(10, min=1, max=3) 
[1]  1.272866  2.311514  2.113181  1.862519  2.763715  1.416313  1.899432  1.382788  1.108341  1.303810
\end{verbatim}

Since the numbers are picked randomly, any two results are different.

The \textbf{exponential distribution} describes the arrival time of a randomly recurring independent event sequence. If \(\mu\) is the mean waiting time for the next event recurrence, its probability density function is:

\[
f(x) = \begin{cases} 
\frac{1}{\mu} e^{\frac{-x}{\mu}} & \text{for } x \geq 0 \\
x & \text{for } x < 0
\end{cases}
\]

\textbf{Example}

Suppose the mean assistant time of ITS front desk is five minutes. Find the probability of a student being helped being completed by ITS front desk in less than three minutes.

The assistant processing rate is equals to one divided by the mean assistant completion time. Hence the processing rate is 1/5 assistants per minute. The function \texttt{pexp} stands for the exponential distribution. The rate here equal to 1/5.
The normal distribution one of the most common distribution in many different fields. It can be used to represent real-valued random variables whose distributions are not known. Normal distribution is defined by the following probability density function, where \( \mu \) is the population mean and \( \sigma^2 \) is the variance.

\[
f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

The normal distribution with \( \mu = 0 \) and \( \sigma = 1 \) is called the standard normal distribution. The function \texttt{pnorm} stands for normal distribution in R.

Example

The test scores of an exam fits a normal distribution. The mean test score is 60, and the standard deviation is 10. What is the percentage of students scoring 80 or more in the exam?

\[
> \texttt{pnorm(80, mean=60, sd=10, lower.tail=FALSE)}
\]

\[\text{[1]} 0.02275013\]

If \( X_1, X_2, \ldots, X_m \) are \( m \) independent random variables having the standard normal distribution, then the following quantity follows a Chi-Squared distribution with \( m \) degrees of freedom. Its mean is \( m \), and its variance is \( 2m \).

\[V = X_1^2 + X_2^2 + \cdots + X_m^2 \sim \chi^2_m\]

Example

Find the 95th percentile of the Chi-Squared distribution with 6 degrees of freedom.

\[
> \texttt{qchisq(.95, df=6)}
\]

\[\text{[1]} 12.59159\]

Assume that a random variable \( Z \) has the standard normal distribution, and another random variable \( V \) has the Chi-Squared distribution with \( m \) degrees of freedom. Assume further that \( Z \) and \( V \) are independent, then the following quantity follows a Student t distribution with \( m \) degrees of freedom.

Example

Find the 2.5th and 97.5th percentiles of the Student t distribution with 5 degrees of freedom.

\[
> \texttt{qt(c(.025, .975), df=5)}
\]

\[\text{[1]} -2.570582 \quad 2.570582\]

If \( V_1 \) and \( V_2 \) are two independent random variables having the Chi-Squared distribution with \( m_1 \) and \( m_2 \) degrees of freedom respectively, then the following quantity follows an F distribution with \( m_1 \) numerator degrees of freedom and \( m_1 \) denominator degrees of freedom, i.e., \((m_1, m_2)\) degrees of freedom.
Example

Find the 95th percentile of the F distribution with (5, 2) degrees of freedom.

> qf(.95, df1=5, df2=2)
[1] 19.29641

**Interval Estimation**

The sample mean for any particular random sample is always computable. Although most often it is not the actual population mean, it does serve as a good **point estimate**. For example, in the data set survey, the survey is performed on a sample of the student population. We can compute the sample mean and use it as an estimate of the corresponding population parameter.

**Example**

Find a point estimate of mean university student height with the sample data from survey.

The idea of doing this case is loading data into R first and saving the survey data of student heights in a variable called height.survey. And then drop all the missing value in the dataset.

> library(MASS)
> height.survey = survey$Height
> mean(height.survey, na.rm=TRUE)
[1] 172.3809

After we found a point estimate of the population mean, we would need a way to quantify its accuracy. Here, we discuss the case where the population variance $\sigma^2$ is assumed known.

Let us denote the $100(1-\alpha/2)$ percentile of the standard normal distribution as $z_{\alpha/2}$. For random sample of sufficiently large size, the end points of the **interval estimate** at $(1 - \alpha)$ confidence level is given as follows:

$$
\bar{x} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}}
$$

**Example**

Assume the population standard deviation $\sigma$ of the student height in survey is 9.48. Find the margin of error and interval estimate at 95% confidence level.

Like last example. Inputting data, cleaning data and calculating data.

> library(MASS)
> height.response = na.omit(survey$Height)

Then we compute the standard error of the mean. 9.48 is population standard deviation, $sem = \sigma / \sqrt{n}$ is standard error of the mean.
Since there are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail. Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`. We multiply it with the standard error of the mean `sem` and get the margin of error. We then add it up with the sample mean, and find the confidence interval as told.

```r
> E = qnorm(.975) * sem; E
[1] 1.285237
> xbar = mean(height.response)
> xbar + c(-E, E)
[1] 171.0956 173.6661
```

The quality of a sample survey can be improved by increasing the sample size. The formula below provide the sample size needed under the requirement of population mean interval estimate at $(1 - \alpha)$ confidence level, margin of error $E$, and population variance $\sigma^2$. Here, $z_{\alpha/2}$ is the 100$(1 - \alpha/2)$ percentile of the standard normal distribution.

$$ n = \frac{Z\alpha/2^2 \sigma^2}{E^2} $$

**Example**

Assume the population standard deviation $\sigma$ of the student height in survey is 9.48. Find the sample size needed to achieve a 1.2 centimeters margin of error at 95% confidence level.

There are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail. Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`.  

```r
> zstar = qnorm(.975)
> sigma = 9.48
> E = 1.2
> zstar^2 * sigma^2 / E^2
[1] 239.7454
```

Multiple choice questionnaires in a survey are often used to determine the proportion of a population with certain characteristic. For example, we can estimate the proportion of female students in the university based on the result in the sample data set survey.

**Example**

Find a point estimate of the female student proportion from survey.

We first filter out missing values in `survey$Sex` with the `na.omit` function, and save it in `gender.response`.  

```r
> n = length(gender.response)
> sigma = 9.48
> sem = sigma/sqrt(n); sem
[1] 0.6557453
```
To find out the number of female students, we compare gender.response with the factor 'Female', and compute the sum. Dividing it by n gives the female student proportion in the sample survey.

```r
> k = sum(gender.response == "Female")
> pbar = k/n; pbar
[1] 0.5
```

After we found a point sample estimate of the population proportion, we would need to estimate its confidence interval.

Let us denote the 100(1 −α/2) percentile of the standard normal distribution as $z_{alpha/2}$. If the samples size $n$ and population proportion $p$ satisfy the condition that $np ≥ 5$ and $n(1 - p) ≥ 5$, than the end points of the interval estimate at $(1 - alpha)$ confidence level is defined in terms of the sample proportion as follows.

$$\bar{p} ± z_{alpha/2} \sqrt{\frac{\bar{p}(a - \bar{p})}{n}}$$

**Example**

Compute the margin of error and estimate interval for the female students proportion in survey at 95% confidence level.

We first determine the proportion point estimate.

```r
> library(MASS)
> gender.response = na.omit(survey$Sex)
> n = length(gender.response)
> k = sum(gender.response == "Female")
> pbar = k/n; pbar
[1] 0.5
```

Then we estimate the standard error.

```r
> SE = sqrt(pbar*(1-pbar)/n); SE
[1] 0.03254723
```

Since there are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail. Therefore, $z_{alpha/2}$ is given by qnorm(.975). Hence we multiply it with the standard error estimate $SE$ and compute the margin of error.

```r
> E = qnorm(.975)*SE; E
[1] 0.06379139
```

Combining it with the sample proportion, we obtain the confidence interval.
The quality of a sample survey can be improved by increasing the sample size. The formula below provide the sample size needed under the requirement of population proportion interval estimate at \((1 - \alpha)\) confidence level, margin of error \(E\), and planned proportion estimate \(p\). Here, \(z_{\alpha/2}\) is the \(100(1 - \alpha/2)\) percentile of the standard normal distribution.

\[
n = \frac{z^2 \alpha^2 \, p(1 - p)}{E^2}
\]

Using a 50% planned proportion estimate, find the sample size needed to achieve 5% margin of error for the female student survey at 95% confidence level.

\[
> \text{zstar} = \text{qnorm}(.975)
> p = 0.5
> E = 0.05
> \text{zstar}^2 * p * (1-p) / E^2
\]

\[1\] 384.1459

**Hypothesis Testing**

In statistics, **null hypothesis** is a general statement that have relationship between two measured phenomena. We usually expect to reject or disprove the null hypothesis to conclude that there is a relationship between two phenomena. Null hypothesis is usually assumed to be true until evidence shows it is false. It is denoted as \(H_0\) and read as H-null or H-zero.

The null hypothesis of the **lower tail test of the population mean** can be expressed as follows:

\[
\mu \geq \mu_0
\]

where \(\mu_0\) is a hypothesized lower bound of the true population mean \(\mu\).

If the test statistic \(z\) defined in terms of the sample mean \(\bar{x}\), the sample size and the population standard deviation \(\sigma\), then it can be calculated by

\[
z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}
\]

Then the null hypothesis of the lower tail test is to be rejected if \(z \leq -z_\alpha\), where \(z_\alpha\) is the \(100(1 - \alpha)\) percentile of the standard normal distribution.

**Example**

Suppose the mean score of GRE exam is more than 300. In a 32 takers GRE exam center, it was found that the average was 290. Assume the population standard deviation is 18. At 0.05 significance level, can we reject the claim by the score?
The null hypothesis is that $\mu \geq 300$. With sample mean equal to 290, hypothesized value equal to 300, population standard deviation is 18, sample size is the 32 takers, and we begin with computing the test statistic.

```
> xbar = 290
> mu0 = 300
> sigma = 18
> n = 32
> z = (xbar-mu0)/(sigma/sqrt(n))
> z
[1] -3.142697
```

We the critical value at .05 significance level can be computed. The critical value is shown in function -z.alpha.

```
> alpha = .05
> z.alpha = qnorm(1-alpha)
> -z.alpha
[1] -1.644854
```

The test statistic -3.142697 is less than the critical value of -1.644854. Hence, at .05 significance level, we reject the claim that mean score of GRE is above 300.

The null hypothesis of the upper tail test of the population mean can be expressed as follows:

$$
\mu \leq \mu_0
$$

where $\mu_0$ is a hypothesized lower bound of the true population mean $\mu$.

If the test statistic $z$ defined in terms of the sample mean $\bar{x}$, the sample size and the population standard deviation $\sigma$, then it can be calculated by

$$
z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}
$$

Then the null hypothesis of the lower tail test is to be rejected if $z \geq -z_\alpha$, where $z_\alpha$ is the 100($1 - \alpha$) percentile of the standard normal distribution.

**Example**

Assume an insurance company states their waiting time in phone service is at most 10 minutes. In a sample of 200 phone services, it is found that the mean waiting time per service is 12 minutes. Assume that the population standard deviation is 1.8 minutes. At .05 significance level, can we reject the company’s statement?

The null hypothesis is that $\mu \leq 10$. With sample mean equal to 12, hypothesized value equal to 10, population standard deviation is 1.8, sample size is the 200 phone services, and we begin with computing the test statistic.
We the critical value at .05 significance level can be computed. The critical value is shown in function \( -z_{\alpha} \).

\[
> \text{xbar} = 12 \\
> \text{mu0} = 10 \\
> \text{sigma} = 1.8 \\
> n = 200 \\
> z = (\text{xbar}-\text{mu0})/(\text{sigma}/\text{sqrt}(n)) \\
> z \\
[1] 15.71348
\]

The test statistic 15.71348 is much greater than the critical value of 1.1644854. Hence, at .05 significance level, we reject the claim that the waiting time in phone service is at most 10 minutes.

The null hypothesis of the lower tail test of the population mean can be expressed as follows:

\[
\mu = \mu_0
\]

where \( \mu_0 \) is a hypothesized lower bound of the true population mean \( \mu \).

If the test statistic \( z \) defined in terms of the sample mean \( \bar{x} \), the sample size and the population standard deviation \( \sigma \), then it can be calculated by

\[
z = \frac{\bar{x} - \mu_0}{\sigma} \frac{1}{\sqrt{n}}
\]

Then the null hypothesis of the two-tailed test is to be rejected if \( z \leq -z_\alpha \) or \( z \geq -z_\alpha \), where \( \frac{z_\alpha}{z} \) is the 100\( (1 - \alpha/2) \) percentile of the standard normal distribution.

**Example**

Suppose the mean E-mail reply time for ITS Help Desk was 15.4 minutes last year. In a sample of 42 E-mail replies this year, the mean reply time is 15.64 minutes. Assume the population standard deviation is 0.9 minutes. At .05 significance level, can we reject the null hypothesis that the mean E-mail reply time for ITS Help Desk does not differ from last year?

The null hypothesis is that \( \mu = 15.4 \). With sample mean equal to 15.64, hypothesized value equal to 15.4, population standard deviation 0.9, sample size is the 42 E-mail replies, and we begin with computing the test statistic.

\[
> \text{xbar} = 15.64 \\
> \text{mu0} = 15.4 \\
> \text{sigma} = 0.9 \\
> n = 42 \\
> z = (\text{xbar}-\text{mu0})/(\text{sigma}/\text{sqrt}(n)) \\
> z \\
[1] 1.726198
\]
We then compute the critical values at .05 significance level. By using function `c(-z-half.alpha, z-half.alpha)`, we get a range of critical values.

```r
> alpha = .05
> z-half.alpha = qnorm(1-alpha/2)
> c(-z-half.alpha, z-half.alpha)
[1] -1.959964  1.959964
```

The test statistic 1.728198 lies between the critical values -1.959964 and 1.959964. Hence, at .05 significance level, we do not reject the null hypothesis that the mean E-mail reply time for ITS Help Desk does not differ from last year.

The null hypothesis of the **lower tail test of the population mean** can be expressed as follows:

\[\mu \geq \mu_0\]

where \(\mu_0\) is a hypothesized lower bound of the true population mean \(\mu\).

If the test statistic \(z\) defined in terms of the **sample mean** \(\bar{x}\), the **sample size** and the **sample standard deviation** \(s\), then it can be calculated by

\[t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}\]

Then the null hypothesis of the lower tail test is to be rejected if \(t \leq -t_\alpha\), where \(t_\alpha\) is the 100(1 – \(\alpha\)) percentile of the Student t distribution with \(n - 1\) degrees of freedom.

**Example**

Suppose the mean score of GRE exam is more than 300. In a 32 takers GRE exam center, it was found that the average was 290. Assume the sample standard deviation is 20. At 0.05 significance level, can we reject the claim by the score?

The null hypothesis is that \(\mu \geq 300\). With sample mean equal to 290, hypothesized value equal to 300, sample standard deviation is 20, sample size is the 32 takers, and we begin with computing the test statistic.

```r
> xbar = 290
> mu0 = 300
> s = 20
> n = 32
> t = (xbar-mu0)/(s/sqrt(n))
> t
[1] -2.828427
```

We then compute the critical value at .05 significance level. The critical value is shown in function `-t.alpha`.

```r
> alpha = .05
> t.alpha = qt(1-alpha, df=n-1)
> -t.alpha
[1] -1.695519
```
The test statistic -2.828427 is less than the critical value of -1.695519. Hence, at .05 significance level, we can reject the claim that mean score of GRE is higher than 300.

The null hypothesis of the **upper tail test of the population mean** can be expressed as follows:

\[ \mu \leq \mu_0 \]

where \( \mu_0 \) is a hypothesized lower bound of the true population mean \( \mu \).

If the test statistic \( z \) defined in terms of the sample mean \( \bar{x} \), the sample size and the sample standard deviation \( s \), then it can be calculated by

\[ t = \frac{\bar{x} - \mu_0}{s \sqrt{n}} \]

Then the null hypothesis of the lower tail test is to be rejected if \( t \geq -t_\alpha \), where \( t_\alpha \) is the 100(1 - \( \alpha \)) percentile of the Student t distribution with \( n - 1 \) degrees of freedom.

**Example**

Assume an insurance company states their waiting time in phone service is at most 10 minutes. In a sample of 200 phone services, it is found that the mean waiting time per service is 12 minutes. Assume that the sample standard deviation is 2.1 minutes. At .05 significance level, can we reject the company’s statement?

The null hypothesis is that \( \mu \leq 10 \). With sample mean equal to 12, hypothesized value equal to 10, sample standard deviation is 2.1, sample size is the 200 phone services, and we begin with computing the test statistic.

```r
> xbar = 12
> mu0 = 10
> s = 2.1
> n = 200
> t = (xbar-mu0)/(s/sqrt(n))
> t

[1] 13.4687
```

We then compute the critical value at .05 significance level. The critical value is shown in function -t.alpha.

```r
> alpha = .05
> t.alpha = qt(1-alpha, df=n-1)
> t.alpha

[1] 1.652547
```

The test statistic 13.4687 is much greater than the critical value of 1.652547. Hence, at .05 significance level, we can reject the claim that the waiting time in phone service is at most 10 minutes.

The null hypothesis of the **two-tailed test of the population mean** can be expressed as follows:
\[ \mu = \mu_0 \]

where \( \mu_0 \) is a hypothesized lower bound of the true population mean \( \mu \).

If the test statistic \( t \) defined in terms of the sample mean \( \bar{x} \), the sample size and the population standard deviation \( s \), then it can be calculated by

\[ t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \]

Then the null hypothesis of the two-tailed test is to be rejected if \( t \leq -t_\alpha \) or \( t \geq -t_\alpha \), where \( \frac{t_\alpha}{2} \) is the 100(1 - \( \alpha \)) percentile of the Student t distribution with \( n - 1 \) degrees of freedom.

**Example**

Suppose the mean E-mail reply time for ITS Help Desk was 15.4 minutes last year. In a sample of 42 E-mail replies this year, the mean reply time is 15.64 minutes. Assume the sample standard deviation is 0.9 minutes. At .05 significance level, can we reject the null hypothesis that the mean E-mail reply time for ITS Help Desk does not differ from last year?

The null hypothesis is that \( \mu = 15.4 \). With sample mean equal to 15.64, hypothesized value equal to 15.4, sample standard deviation is 0.9, sample size is the 42 E-mail replies, and we begin with computing the test statistic.

\[
> \text{xbar} = 15.64 \\
> \mu_0 = 15.4 \\
> s = 0.9 \\
> n = 42 \\
> t = (\text{xbar}-\mu_0)/(s/\text{sqrt}(n)) \\
> t \\
[1] 1.728198
\]

We then compute the critical values at .05 significance level. By using function \( c(-\text{t.half.alpha}, \text{t.half.alpha}) \), we get a range of critical values.

\[
> \text{alpha} = .05 \\
> \text{t.half.alpha} = \text{qt}(1-\text{alpha}/2, \text{df}=n-1) \\
> c(-\text{t.half.alpha}, \text{t.half.alpha}) \\
[1] -2.019541 2.019541
\]

The test statistic 1.728198 lies between the critical values -2.019541 and 2.019541. Hence, at .05 significance level, we do not reject the null hypothesis that the mean E-mail reply time for ITS Help Desk does not differ from last year.

The null hypothesis of the lower tail test about population proportion can be expressed as follows:

\[ p \geq p_0 \]

where \( p_0 \) is a hypothesized lower bound of the true population proportion \( p \).
The test statistic \( z \) is defined in terms of the **sample proportion** \( p \) and the **sample size** \( n \):

\[
z = \frac{\bar{p} - p_0}{\sqrt{\frac{p_0(1 - p_0)}{n}}}
\]

Then the null hypothesis of the lower tail test is to be rejected if \( z \leq -z_{\alpha} \), where \( z_{\alpha} \) is the 100\( (1 - \alpha) \) percentile of the standard normal distribution.

**Example**

Suppose 60\% of citizens voted in last election. 85 out of 148 people in a telephone survey said that they voted in current election. At 0.5 significance level, can we reject the null hypothesis that the proportion of voters in the population is above 60\% this year?

The null hypothesis is that \( p \geq 0.6 \). \( \frac{85}{148} \) is sample proportion. Hypothesized value equal to 0.6 and sample size is 148. The test statistic can be computed as

```r
> pbar = 85/148
> p0 = .6
> n = 148
> z = (pbar-p0)/sqrt(p0*(1-p0)/n)
> z
[1] -0.6375983
```

We then compute the critical value at .05 significance level. The critical value is shown in function \(-z._alpha\).

```r
> alpha = .05
> z.alpha = qnorm(1-alpha)
> -z.alpha
[1] -1.644854
```

The test statistic \(-0.6375983\) is not less than the critical value of \(-1.644854\). Hence, at .05 significance level, we do not reject the null hypothesis that the proportion of voters in the population is above 60\% this year.

The null hypothesis of the **upper tail test about population proportion** can be expressed as follows:

\[ p \leq p_0 \]

where \( p_0 \) is a hypothesized lower bound of the true population proportion \( p \).
Then the null hypothesis of the lower tail test is to be rejected if \( z \geq -z_\alpha \), where \( z_\alpha \) is the 100(1 – \( \alpha \)) percentile of the standard normal distribution.

**Example**

Suppose that 12% of apples harvested in an orchard last year was rotten. 30 out of 214 apples in a harvest sample this year turns out to be rotten. At .05 significance level, can we reject the null hypothesis that the proportion of rotten apples in harvest stays below 12% this year?

The null hypothesis is that \( p \leq 0.12 \). \( \frac{30}{214} \) is sample proportion. Hypothesized value equal to 0.12. Sample size is 214. The test statistic can be computed as

\[
> \text{pbar} = 30/214 \\
> \text{p0} = .12 \\
> \text{n} = 214 \\
> z = (\text{pbar}-\text{p0})/\text{sqrt}(\text{p0}*(1-\text{p0})/\text{n}) \\
> z \\
[1] 0.908751
\]

We then compute the critical value at .05 significance level. The critical value is shown in function \(-z\alpha\).

\[
> \text{alpha} = .05 \\
> \text{z.alpha} = \text{qnorm}(1-\text{alpha}) \\
> \text{z.alpha} \\
[1] 1.644854
\]

The test statistic 0.908751 is not greater than the critical value of 1.644854. Hence, at .05 significance level, we do not reject the null hypothesis that the proportion of rotten apples in harvest stays below 12% this year.

The test statistic of the **two-tailed test about population proportion** can be expressed as follows:

\[
p = p_0
\]

where \( p_0 \) is a hypothesized value of the true population proportion \( p \).

The test statistic \( z \) is defined in terms of the **sample proportion** \( p \) and the **sample size** \( n \):

\[
z = \frac{\bar{p} - p_0}{\sqrt{\frac{p_0(1 - p_0)}{n}}}
\]
Then the null hypothesis of the two-tailed test is to be rejected if \( z \geq -z_{\alpha/2} \) or \( z \leq -z_{\alpha/2} \), where \( z_{\alpha/2} \) is the 100(1 − \( \alpha \)) percentile of the standard normal distribution.

**Example**

Suppose a coin toss turns up 6415 heads out of 9999 trials. At .05 significance level, can one reject the null hypothesis that the coin toss is fair?

The null hypothesis is that \( p = 0.5 \). \( \frac{6415}{9999} \) is sample proportion. Hypothesized value equal to 0.5. Sample size is 9999 trials. The test statistic can be computed as

```r
> pbar = 6415/9999
> p0 = .5
> n = 9999
> z = (pbar-p0)/sqrt(p0*(1-p0)/n)
> z
[1] 28.31142
```

We then compute the critical value at .05 significance level. The critical value is shown in function `c(-z.half.alpha, z.half.alpha)`.

```r
> alpha = .05
> z.half.alpha = qnorm(1-alpha/2)
> c(-z.half.alpha, z.half.alpha)
[1] -1.959964  1.959964
```

The test statistic 28.31142 doesn’t lie between the critical values -1.959964 and 1.959964. Hence, at .05 significance level, we reject the null hypothesis that the coin toss is fair.

**Type II Error**

When the null hypothesis is true but was rejected, the judgement made a type I error. The probability of making a type I error is \( \alpha \), which is the level of significance set for hypothesis test. An \( \alpha \) of 0.05 indicates that you are willing to accept a 5% chance that you are wrong when you reject the null hypothesis. In order to reduce the risk a lower value of \( \alpha \) is necessary. But a lower \( \alpha \) will less likely to detect a true difference if it is really exist.

When the null hypothesis is false but was not rejected, the judgement made a type II error. The probability of making a type II error is \( \beta \), which depends on the power of the test. The risk of committing a type II error can be reduced by ensuring the test has enough power. By ensuring a large enough sample size, the practical difference can be detected when one truly exists.

Here is an example to help understanding the relationship between type I and type II error and determining which error has more severe consequences.
A medical researcher wants to compare the effectiveness of two medications. The null and alternative hypotheses are:

- **Null hypothesis** (H₀): \( \mu_1 = \mu_2 \)
  The two medications are equally effective.

- **Alternative hypothesis** (H₁): \( \mu_1 \neq \mu_2 \)
  The two medications are not equally effective.

A type I error occurs if the researcher rejects the null hypothesis and concludes that the two medications are different but they are the same in fact. In this case medications work in the same way but researchers rejected. Medications users will not have major problem regarding the medications themselves or the decision made by researchers.

A type II error occurs if the research fails to reject the null hypothesis and concludes that the two medications are the same but they are not actually. This error has potential to cause severe life-threatening since the real effective one might be missed to sell to patients.

As a result, the risks of type I and type II errors need to be considered when making hypothesis. If the consequences between two types of error are severe, then choose a level of significance and a power for the test that will reflect the relative severity of those consequences.

**Type II Error in Lower Tail Test of Population Mean with Known Variance**

In a lower tail test of the population mean, the null hypothesis claims that the true population mean \( \mu \) is greater than a given hypothetical value \( \mu_0 \).

\[
\mu \geq \mu_0
\]

A type II error occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean \( \mu \) is in fact less than \( \mu_0 \).

Assume that the population has a known variance \( \sigma^2 \). By the Central Limit Theorem, the population of all possible means of samples of sufficiently large size \( n \) approximately follows the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will not be rejected, and then obtain an estimate of the probability of type II error.

**Example**
Suppose the mean score of GRE exam is more than 300. Assume actual mean score is 294. And the population standard deviation is 18. At .05 significance level, what is the probability of having type II error for a sample size of 32 exam takers?

The sample size is the 32 takers with population standard deviation equal to 18. And the function

\[
\text{sem} = \frac{\text{sigma}}{\sqrt{n}}; \text{sem}
\]

stands for standard error

\[
\begin{align*}
> n &= 32 \\
> \text{sigma} &= 18 \\
> \text{sem} &= \frac{\text{sigma}}{\sqrt{n}}; \text{sem} \\
\end{align*}
\]

[1] 3.181981

We next compute the lower bound of sample means for which the null hypothesis \( \mu \geq 300 \) would not be rejected. The significance level is 0.05 and hypothetical lower bound equal to 300.

\[
\begin{align*}
> \alpha &= .05 \\
> \mu_0 &= 300 \\
> q &= \text{qnorm}(\alpha, \text{mean}=\mu_0, \text{sd}=\text{sem}); q \\
\end{align*}
\]

[1] 294.7661

Therefore, when the sample mean is greater than 294.7661 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 294. The probability of the sample mean above 294, and thus found the probability of type II error.

\[
\begin{align*}
> \mu &= 294 \\
> \text{pnorm}(q, \text{mean}=\mu, \text{sd}=\text{sem}, \text{lower.tail}=\text{FALSE}) \\
\end{align*}
\]

[1] 0.4048683

So, if the exam takers sample size is 32, the actual mean exam score is 294 and the population standard deviation is 18, then the probability of type II error for testing the null hypothesis \( \mu \geq 300 \) at .05 significance level is 40.49%, and the power of the hypothesis test is 59.51%.

**Type II Error in Upper Tail Test of Population Mean with Known Variance**

In a upper tail test of the population mean, the null hypothesis claims that the true population mean \( \mu \) is less than a given hypothetical value \( \mu_0 \).

\[
\mu \leq \mu_0
\]

A type II error occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean \( \mu \) is in fact greater than \( \mu_0 \).

Assume that the population has a known variance \( \sigma^2 \). By the Central Limit Theorem, the population of all possible means of samples of sufficiently large size \( n \) approximately follows the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will not be rejected, and then obtain an estimate of the probability of type II error.

**Example**
Assume an insurance company states their waiting time in phone service is at most 10 minutes. Assume the actual mean amount of waiting time per service is 10.8 minutes and the population standard deviation is 1.8 minutes. At .05 significance level, what is the probability of having type II error for a sample size of 200 calls?

The sample size is the 200 phone calls with population standard deviation equal to 1.8 minutes. And the function \( \text{sem} = \frac{\text{sigma}}{\sqrt{n}} \); \( \text{sem} \) stands for standard error.

```r
> n = 200
> sigma = 1.8
> sem = sigma/sqrt(n); sem
[1] 0.1272792
```

We next compute the lower bound of sample means for which the null hypothesis \( \mu \leq 10 \) would not be rejected. The significance level is 0.05 and hypothetical lower bound equal to 10.

```r
> alpha = .05
> mu0 = 10
> q = qnorm(alpha, mean=mu0, sd=sem, lower.tail=FALSE); q
[1] 10.20936
```

Therefore, when the sample mean is less than 10.20936 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 10.8. The probability of the sample mean below 10.20936, and thus found the probability of type II error.

```r
> mu = 10.8
> pnorm(q, mean=mu, sd=sem)
[1] 1.7375e-06
```

So, if the phone calls sample size is 200, the actual mean phone call waiting time is 10.8 minutes and the population standard deviation is 1.8 minutes, then the probability of type II error for testing the null hypothesis \( \mu \leq 10 \) at .05 significance level is 1.7375e-06, and the power of the hypothesis test is 99.99982625%.

**Type II Error in Two-Tailed Test of Population Mean with Known Variance**

In a **two-tailed test of the population mean**, the null hypothesis claims that the true population mean \( \mu \) is equal to a given hypothetical value \( \mu_0 \).

\[
\mu = \mu_0
\]

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean \( \mu \) is in fact different from \( \mu_0 \).

Assume that the population has a known variance \( \sigma^2 \). By the Central Limit Theorem, the population of all possible means of samples of sufficiently large size \( n \) approximately follows the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will not be rejected, and then obtain an estimate of the probability of type II error.

**Example**
Suppose the mean E-mail reply time for ITS Help Desk was 15.4 minutes last year. Assume the actual mean population reply time is 16 minutes, and the population standard deviation is 0.9 minutes. At .05 significance level, what is the probability of having type II error for a sample size of 42 E-mail replies?

The sample size is the 42 E-mail replies with population standard deviation equal to 0.9 minutes. And the function \( \text{sem} = \frac{\sigma}{\sqrt{n}} \); \( \text{sem} \) stands for standard error.

\[
\begin{align*}
&> n = 42 \\
&> \sigma = 0.9 \\
&> \text{sem} = \frac{\sigma}{\sqrt{n}}; \text{sem} \\
&\quad \text{[1]} \ 0.138873
\end{align*}
\]

We next compute the lower and upper bounds of sample means for which the null hypothesis \( \mu = 15.4 \) would not be rejected. Significance level is 0.05. And hypothetical mean is 15.4.

\[
\begin{align*}
&> \alpha = .05 \\
&> \mu_0 = 15.4 \\
&> I = c(\alpha/2, 1-\alpha/2) \\
&> q = \text{qnorm}(I, \text{mean}=\mu_0, \text{sd}=\text{sem}); \ q \\
&\quad \text{[1]} \ 15.12781 \ 15.67219
\end{align*}
\]

Therefore, as long as the sample mean is between 15.12781 and 15.67219 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 16, we can compute the lower tail probabilities of both end points.

\[
\begin{align*}
&> \mu = 16 \\
&> p = \text{pnorm}(q, \text{mean}=\mu, \text{sd}=\text{sem}); \ p \\
&\quad \text{[1]} \ 1.687888e-10 \ 9.124426e-03
\end{align*}
\]

Finally, the probability of type II error is the probability between the two end points.

\[
\begin{align*}
&> \text{diff}(p) \\
&\quad \text{[1]} \ 0.009124426
\end{align*}
\]

If the Email sample size is 35, the actual mean population waiting time is 16 minutes and the population standard deviation is 0.9 minutes, then the probability of type II error for testing the null hypothesis \( \mu = 15.4 \) at .05 significance level is 0.912%, and the power of the hypothesis test is 99.088%.

**Type II Error in Lower Tail Test of Population Mean with Unknown Variance**

In a lower tail test of the population mean, the null hypothesis claims that the true population mean \( \mu \) is greater than a given hypothetical value \( \mu_0 \).

\[
\mu \geq \mu_0
\]

A type II error occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean \( \mu \) is in fact less than \( \mu_0 \).
Let $s^2$ be the sample variance. For sufficiently large $n$, the population of the following statistics of all possible samples of size $n$ is approximately a Student t distribution with $n - 1$ degrees of freedom.

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

This allows us to compute the range of sample means for which the null hypothesis will not be rejected, and to obtain the probability of type II error.

**Example**

Suppose the mean score of GRE exam is more than 300. In a 32 takers GRE exam center, the standard deviation of the score is 20. If actual mean GRE score is 294, what is the probability of type II error for a hypothesis test at .05 significance level?

Sample size $n$ is 32. Sample standard deviation is 20. The standard error estimate, $SE$, can be get.

```r
> n = 32
> s = 20
> SE = s/sqrt(n); SE
[1] 3.535534
```

We next compute the lower bound of sample means for which the null hypothesis $\mu \geq 300$ would not be rejected. Significance level is 0.05. And hypothetical lower bound is 300.

```r
> alpha = .05
> mu0 = 300
> q = mu0 + qt(alpha, df=n-1) * SE; q
[1] 294.0054
```

Therefore, so long as the sample mean is greater than 294.0054 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 294, we can compute the probability of the sample mean above 294.0054, and thus found the probability of type II error.

```r
> mu = 294
> pt((q - mu)/SE, df=n-1, lower.tail=FALSE)
[1] 0.4993916
```

If the exam takers sample size is 32, the sample standard variance is 20 and the actual mean score is 294, then the probability of type II error for testing the null hypothesis $\mu \geq 300$ at .05 significance level is 49.93916%, and the power of the hypothesis test is 50.06084%.

**Type II Error in Upper Tail Test of Population Mean with Unknown Variance**

In a upper tail test of the population mean, the null hypothesis claims that the true population mean $\mu$ is less than a given hypothetical value $\mu_0$.

$$\mu \leq \mu_0$$
A type II error occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean \( \mu \) is in fact greater than \( \mu_0 \).

Let \( s^2 \) be the sample variance. For sufficiently large \( n \), the population of the following statistics of all possible samples of size \( n \) is approximately a Student t distribution with \( n - 1 \) degrees of freedom.

\[
\frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}
\]

This allows us to compute the range of sample means for which the null hypothesis will not be rejected, and to obtain the probability of type II error.

**Example**

Suppose an insurance company states their waiting time in phone service is at most 10 minutes. Assume in a random sample of 200 phone services, the standard deviation of waiting time is 2.1 minutes. If actual mean amount of waiting time per service is 12 minutes, what is the probability of type II error for a hypothesis test at .05 significance level?

Sample size \( n \) is 200. Sample standard deviation is 2.1. The standard error estimate, SE, can be get.

\[
> n = 200 \\
> s = 2.1 \\
> SE = s/sqrt(n); SE \\
[1] 0.1484924
\]

We next compute the lower bound of sample means for which the null hypothesis \( \mu \leq 10 \) would not be rejected. Significance level is 0.05. And hypothetical lower bound is 10.

\[
> alpha = .05 \\
> mu0 = 10 \\
> q = mu0 + qt(alpha, df=n-1, lower.tail=FALSE) * SE; q \\
[1] 10.24539
\]

Therefore, so long as the sample mean is less than 10.24539 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 12, we can compute the probability of the sample mean below 10.24539, and thus found the probability of type II error.

\[
> mu = 12 \\
> pt((q - mu)/SE, df=n-1) \\
[1] 4.672811e-25
\]

If the phone services size is 200, the sample standard deviation of waiting time per service is 10 minutes and the actual mean amount of waiting time per service is 12 minutes, then the probability of type II error for testing the null hypothesis \( \mu \leq 10 \) at .05 significance level is 4.672811e-25 and the power of the hypothesis test is 1 - 4.672811e-25.
Type II Error in Two-Tailed Test of Population Mean with Unknown Variance

In a two-tailed test of the population mean, the null hypothesis claims that the true population mean $\mu$ is equal to a given hypothetical value $\mu_0$.

$$\mu = \mu_0$$

A type II error occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean $\mu$ is in fact different from $\mu_0$.

Let $s^2$ be the sample variance. For sufficiently large $n$, the population of the following statistics of all possible samples of size $n$ is approximately a Student t distribution with $n-1$ degrees of freedom.

$$\frac{\bar{x} - \mu}{s} \sqrt{n}$$

This allows us to compute the range of sample means for which the null hypothesis will not be rejected, and to obtain the probability of type II error.

Example

Suppose the mean E-mail reply time for ITS Help Desk was 15.4 minutes last year. Assume in a random sample 42 Email replies, the standard deviation of time is 0.9 minutes. If actual mean reply time is 15.6 minutes, what is the probability of type II error for a hypothesis test at .05 significance level?

Sample size $n$ is 42. Sample standard deviation is 0.9. The standard error estimate, SE, can be get.

```r
> n = 42
> s = 0.9
> SE = s/sqrt(n); SE  # standard error estimate
[1] 0.138873
```

We next compute the lower and upper bounds of sample means for which the null hypothesis $\mu = 15.4$ would not be rejected.

```r
> alpha = .05
> mu0 = 15.4
> I = q(alpha/2, 1-alpha/2)
> q = mu0 + qt(I, df=n-1) * SE; q
[1] 15.11954 15.68046
```

Therefore, so long as the sample mean is between 15.11954 and 15.68046 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 15.6, we can compute the lower tail probabilities of both end points.

```r
> mu = 15.6
> p = pt((q - mu)/SE, df=n-1); p
[1] 0.0006378329 0.7172473587
```
Finally, the probability of type II error is the probability between the two end points.

```r
> diff(p)
[1] 0.7166095
```

If the Email sample size is 42, the sample standard deviation of reply time is 0.9 minutes and the actual mean population reply time is 15.6 minutes, then the probability of type II error for testing the null hypothesis \( \mu = 15.4 \) at .05 significance level is 71.66095%, and the power of the hypothesis test is 28.33905%.

**Two Populations**

The analysis above are all about measurements taken in one variable for each sampling unit. This is referred to as univariate data. This part will focus on measurements taken in two variables for each sampling unit. This is referred to as bivariate data. For example, if the measurements are **categorical**, such as Yes or No question, and taken from two distinct groups such as different genders or ages, the analysis will involve comparing two independent proportions; if the measurements are **quantitative** and taken from two distinct groups the analysis will involve comparing two independent means; or if the measurements are **quantitative** and taken twice from each subject, like an experiment before and after a medicine, the analysis will involve comparing two dependent means.

**Population Mean Between Two Matched Samples**

Two data samples are matched if they come from repeated observations of the same subject. Here, we assume that the data populations follow the normal distribution. Using the **paired t-test**, we can obtain an interval estimate of the difference of the population means.

**Example**

In the built-in data set named immer, the barley yield in years 1931 and 1932 of the same field are recorded. The yield data are presented in the data frame columns Y1 and Y2. Assuming that the data in immer follows the normal distribution, find the 95% confidence interval estimate of the difference between the mean barley yields between years 1931 and 1932.
Between years 1931 and 1932 in the data set immer, the 95% confidence interval of the difference in means of the barley yields is the interval between 6.122 and 25.705.

**Population Mean Between Two Independent Samples**

Two data samples are independent if they come from unrelated populations and the samples do not affect each other. Here, we assume that the data populations follow the normal distribution. Using the **unpaired t-test**, we can obtain an interval estimate of the difference between two population means.

**Example**

In the data frame column mpg of the data set mtcars, there are gas mileage data of various 1974 U.S. automobiles.

```r
> mtcars$mpg
[1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

Another data column in mtcars, named am, indicates the transmission type of the automobile model (0 = automatic, 1 = manual).

```r
> mtcars$am
[1] 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1
```

In particular, the gas mileage for manual and automatic transmissions are two independent data populations. Assuming that the data in mtcars follows the normal distribution, find the 95% confidence interval estimate of the difference between the mean gas mileage of manual and automatic transmissions.
> L = mtcars$am == 0
> mpg.auto = mtcars[L,]$mpg
> mpg.auto
   [1] 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7 21.5
   [16] 15.5 15.2 13.3 19.2

Using the negation of L to find the gas mileage for manual transmission.

> mpg.manual = mtcars[!L,]$mpg
> mpg.manual
   [1] 21.0 21.0 22.8 32.4 30.4 33.9 27.3 26.0 30.4 15.8 19.7 15.0 21.4

Then apply the t.test function to compute the difference in means of the two sample data.

> t.test(mpg.auto, mpg.manual)

    Welch Two Sample t-test

data:  mpg.auto and mpg.manual
    t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-11.280194  -3.209684
sample estimates:
mean of x mean of y
    17.14737   24.39231

In mtcars, the mean mileage of automatic transmission is 17.147 mpg and the manual transmission is 24.392 mpg. The 95% confidence interval of the difference in mean gas mileage is between 3.2097 and 11.2802 mpg.

**Comparison of Two Population Proportions**

A survey conducted in two distinct populations will produce different results. It is often necessary to compare the survey response proportion between the two populations. Here, we assume that the data populations follow the normal distribution.

**Example**

In the built-in data set named quine, children from an Australian town is classified by ethnic background, gender, age, learning status and the number of days absent from school.

> library(MASS)
> head(quine)

    Eth Sex Age Lrn Days
    1   A  M  F0  SL  2
    2   A  M  F0  SL 11
    3   A  M  F0  SL 14
    4   A  M  F0  AL  5
    5   A  M  F0  AL  5
    6   A  M  F0  AL 13

In effect, the data frame column Eth indicates whether the student is Aboriginal or Not ("A" or "N"), and the column Sex indicates Male or Female ("M" or "F").
In R, we can tally the student ethnicity against the gender with the table function. As the result shows, within the Aboriginal student population, 38 students are female. Whereas within the Non-Aboriginal student population, 42 are female.

```r
> table(quine$Eth, quine$Sex)

F  M
A 38 31
N 42 35
```

Assuming that the data in quine follows the normal distribution, find the 95% confidence interval estimate of the difference between the female proportion of Aboriginal students and the female proportion of Non-Aboriginal students, each within their own ethnic group.

We apply the `prop.test` function to compute the difference in female proportions. The Yates’s continuity correction is disabled for pedagogical reasons.

```r
> prop.test(table(quine$Eth, quine$Sex), correct=FALSE)

2-sample test for equality of proportions without continuity correction

data:  table(quine$Eth, quine$Sex)
X-squared = 0.0040803, df = 1, p-value = 0.9491
alternative hypothesis: two.sided
95 percent confidence interval:
-0.1564218  0.1669620
sample estimates:
  prop 1  prop 2
0.5507246 0.5454545
```

So the 95% confidence interval estimate of the difference between the female proportion of Aboriginal students and the female proportion of Non-Aboriginal students is between -15.6% and 16.7%.

**Goodness of fit**

The goodness of fit used in statistics and statistical modelling to compare how far apart the expected values from the actual values. Measures of goodness of fit which can be used in statistical hypothesis testing such as testing for normality of residuals indicates the difference between observed values and expected values under the model. In the analysis of variance, one of the components into which the variance is partitioned may be a lack-of-fit sum of squares.

A chi-squared test is the statistical hypothesis test whether the sampling distribution of the test statistic is a chi-squared distribution when the null hypothesis is true.

In order to calculate a chi-square goodness-of-fit, state the null hypothesis and the alternative hypothesis at first, then choose a significance level, and determine the critical value.
A population is called **multinomial** if its data is categorical and belongs to a collection of discrete non-overlapping classes.

The null hypothesis for **goodness of fit test for multinomial distribution** is that the observed frequency $f_i$ is equal to an expected count $e_i$ in each category. It is to be rejected if the p-value of the following **Chi-squared test statistics** is less than a given significance level $\alpha$.

$$X^2 = \sum \frac{(f_i - e_i)^2}{e_i}$$

**Example**

In the built-in data set survey, the Smoke column records the survey response about the student’s smoking habit. As there are exactly four proper response in the survey: "Heavy", "Regul" (regularly), "Occas" (occasionally) and "Never", the Smoke data is multinomial. It can be confirmed with the levels function in R.

```r
> library(MASS)
> levels(survey$Smoke)
```

[1] "Heavy" "Never" "Occas" "Regul"

By using the table function we can find frequency distribution.

```r
> smoke.freq = table(survey$Smoke)
> smoke.freq

<table>
<thead>
<tr>
<th>Heavy</th>
<th>Never</th>
<th>Occas</th>
<th>Regul</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>189</td>
<td>19</td>
<td>17</td>
</tr>
</tbody>
</table>
```

Suppose the campus smoking statistics is as below. Determine whether the sample data in survey supports it at .05 significance level.

We save the campus smoking statistics in a variable named smoke.prob. Then we apply the **chisq.test** function and perform the Chi-Squared test.

```r
> smoke.prob = c(.045, .795, .085, .075)
> chisq.test(smoke.freq, p=smoke.prob)

Chi-squared test for given probabilities

data:  smoke.freq
X-squared = 0.10744, df = 3, p-value = 0.9909
```

Since the p-value 0.991 is greater than the .05 significance level, we do not reject the null hypothesis that the sample data in survey supports the campus-wide smoking statistics.

**Chi-squared Test of Independence**

The Chi-Square test of Independence is used to determine if there is a significant relationship between two nominal variables. Two random variables $x$ and $y$ are called independent if the probability distribution of one variable is not affected by the presence of another.
Assume \( f_{ij} \) is the observed frequency count of events belonging to both \( i \)-th category of \( x \) and \( j \)-th category of \( y \). Also assume \( e_{ij} \) to be the corresponding expected count if \( x \) and \( y \) are independent. The null hypothesis of the independence assumption is to be rejected if the p-value of the following Chi-squared test statistics is less than a given significance level \( \alpha \).

\[
X^2 = \sum_{i,j} \frac{(f_{ij} - e_{ij})^2}{e_{ij}}
\]

**Example**

In the built-in data set survey, the Smoke column records the students smoking habit, while the Exer column records their exercise level. The allowed values in Smoke are "Heavy", "Regul" (regularly), "Occas" (occasionally) and "Never". As for Exer, they are "Freq" (frequently), "Some" and "None".

We can tally the students smoking habit against the exercise level with the table function in R. The result is called the contingency table of the two variables.

```r
> library(MASS)
> tbl = table(survey$Smoke, survey$Exer)
> tbl

Freq  None  Some
Heavy  7   1   3
Never  87  18  84
Occas 12   3   4
Regul  9   1   7
```

Test the hypothesis whether the students smoking habit is independent of their exercise level at .05 significance level.

We apply the `chisq.test` function to the contingency table `tbl`, and found the p-value to be 0.4828.

```r
> chisq.test(tbl)

Pearson's Chi-squared test

data:  tbl
X-squared = 5.4885, df = 6, p-value = 0.4828

Warning message:
In chisq.test(tbl) : Chi-squared approximation may be incorrect
```

Since the p-value 0.4828 is greater than the .05 significance level, we do not reject the null hypothesis that the smoking habit is independent of the exercise level of the students.
Analysis of Variance

Analysis of Variance (ANOVA) is a statistical method used to test differences between two or more means. ANOVA evaluates the importance of one or more factors by comparing the response variable means at the different factor levels. The null hypothesis assumes that all population means are equal but the alternative hypothesis assumes that at least one of the mean is different with others.

In a **completely randomized design**, there is only one primary factor under consideration in the experiment. The test subjects are assigned to treatment levels of the primary factor at random.

**Example**

A test is concerning about students’ understanding after ITS Help Desk SAS, R and SPSS teaching sessions. There were 6 randomly picked students for each session. The picked students will do a test paper with full score 100. Their performance indicates how the teaching session improve their understanding the software. Suppose the following table represents the test score of the 3 session after tests. At .05 level of significance, test whether the student understanding to the 3 sessions are all equal.

<table>
<thead>
<tr>
<th></th>
<th>SAS</th>
<th>R</th>
<th>SPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>52</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>33</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>8</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>47</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>43</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>32</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>

df1 = read.table("score.txt", header=TRUE); df1