uRProgramming

By

Zekai Otles

Created on Wed Nov 29 22:44:42 2017

Table of Contents

Acknowledgements

I would love thank my wife Sevtap Otles putting up with me while I was working on this book. I would not meet with R programming, if I was not hired by Dr. Paul Carbone who has shown how to be dedicated and sincere for his colleagues.

It would have been impossible to reach wherever I am at without my mom's trust and support. She (Zulfinaz Otles) influenced me and others to reach our education goals to be successful in life. She constantly worries the lack of educational opportunities for the people and children of Turkey, she is so concerned about education cost which is financially stressing the families.

My grand father (Ethem Demirkol) has taught me to work hard, be honest and treat all the people equal.

Finally, I would love thank to both of my sons (Erkin and Arel) for their sincerity, honesty, life choices and support to my wife and myself.

Thanks to all supporters, readers of this book and contributors to society.

1 Introduction

The idea is to create simple book, it will flow well and also easy to understand. Many people don't have much time to start a R programming and don't have experience other software engineering tools. Here in this book my aim is to start simple and built on using some software technologies.

2 Environment Setup for R programming Developers

2.1 *Environment setup*

	Package	**Version**
1	uRProgramming	1.1.1
2	tcltk	3.2.3
3	readr	1.1.1
4	log4r	0.2
5	odfWeave	0.8.4
6	XML	3.98-1.9
7	lattice	0.20-33
8	chron	2.3-51
9	RMySQL	0.10.13
10	DBI	0.7
11	stats	3.2.3
12	graphics	3.2.3
13	grDevices	3.2.3
14	utils	3.2.3
15	datasets	3.2.3

	Package	Version
16	methods	3.2.3

	Package	Version
17	base	3.2.3

```
> environment("ls")
```

```
usage of ls()
functNameusage of ls(all=TRUE)
functName
> cat("Current Working Dir \n", getwd())
Current Working Dir
```

```
/home/zekai/R_workspace/odfWeaveTmp
```

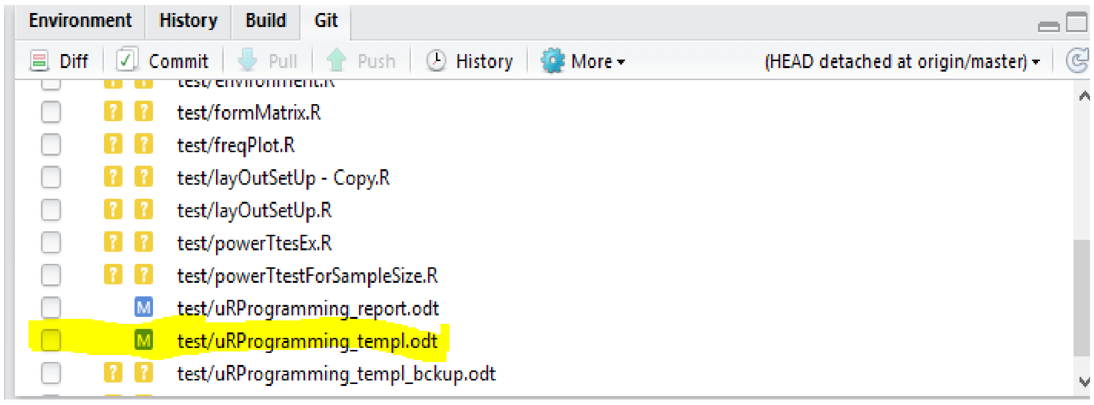
2.2 *Using Version Control System*

Talk about the version control, and why it is so important.

Explain why you decide to use git and demonstrate the git.

Talk about git, demonstrate add, commit checkout the difference in the files.
Binary files in the git,

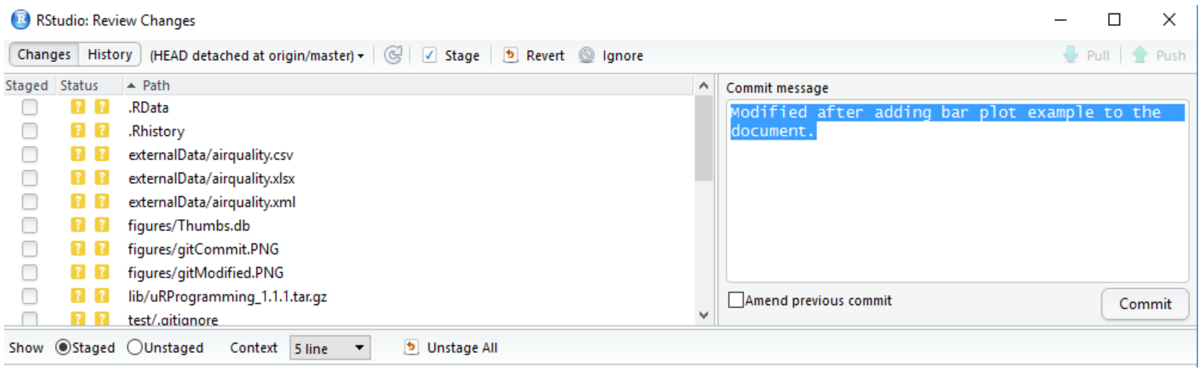
how to branch.



test/environment.R
test/formMatrix.R
test/freqPlot.R
test/layOutSetUp - Copy.R
test/layOutSetUp.R
test/powerTtesEx.R
test/powerTtestForSampleSize.R
test/uRProgramming_report.odt
test/uRProgramming_templ.odt
test/uRProgramming_templ_bckup.odt

Figure 1 : Git from Rstudio shows which files are modified recently

The git commit example from Rstudio, after committed with following message “Modified after adding bar plot example to the document.” we also push our code to the repository.



Staged	Status	Path
☐	? ?	.RData
☐	? ?	.Rhistory
☐	? ?	externalData/airquality.csv
☐	? ?	externalData/airquality.xlsx
☐	? ?	externalData/airquality.xml
☐	? ?	figures/Thumbs.db
☐	? ?	figures/gitCommit.PNG
☐	? ?	figures/gitModified.PNG
☐	? ?	lib/uRProgramming_1.1.1.tar.gz
☐	? ?	test/.gitignore

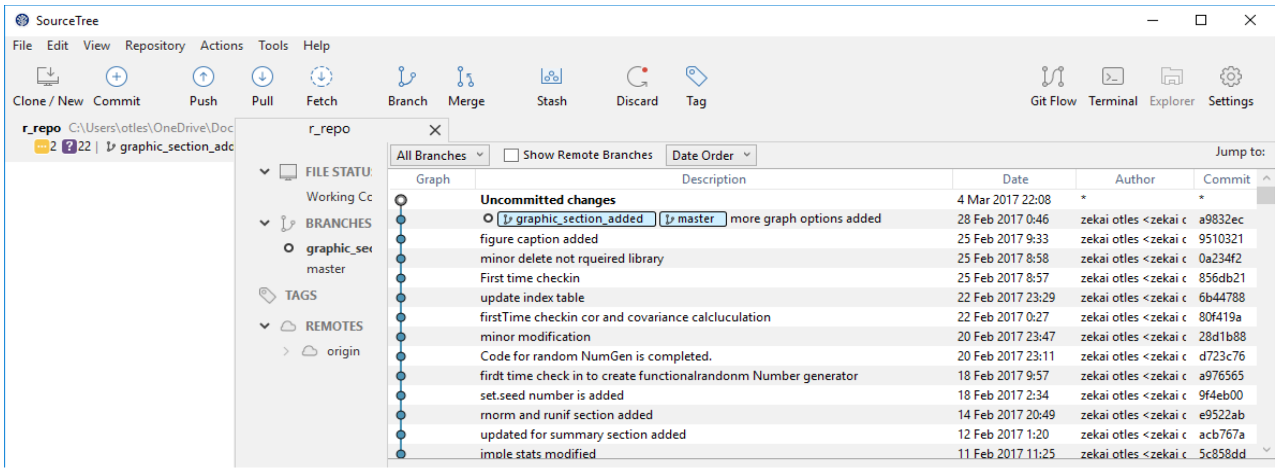
Commit message

```
Modified after adding bar plot example to the
document.
```

☐ Amend previous commit

Show [✓ Staged] Unstaged

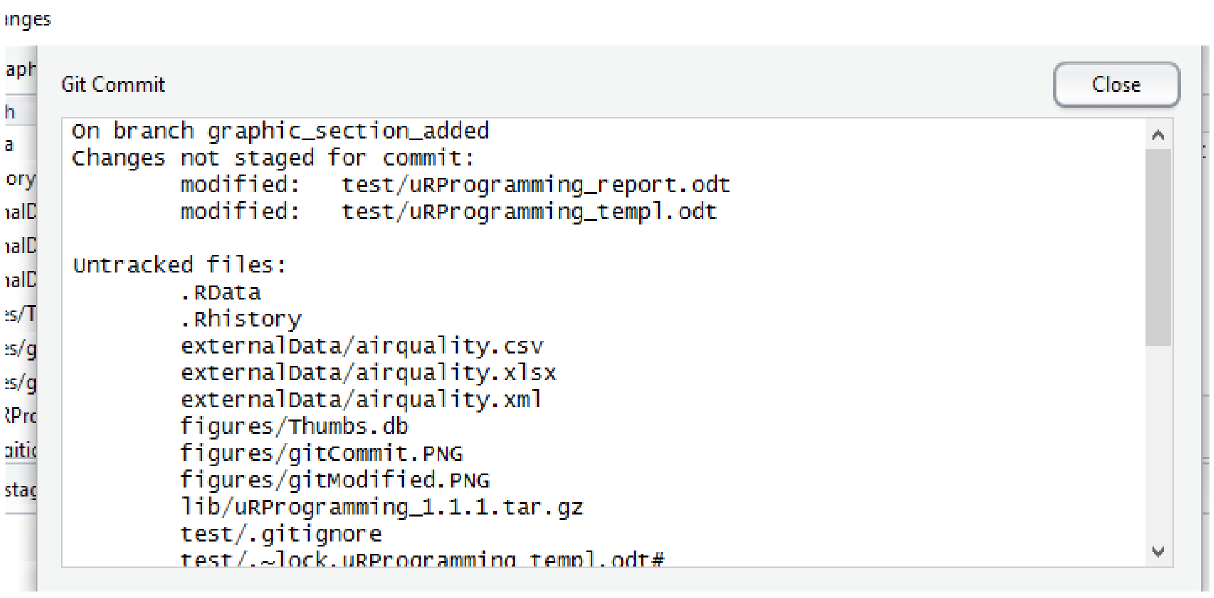
The source code branching with Git is created either using command line or git GUI like source tree can be used for creating branch and other git functionality.



r_repo C:\Users\otles\OneDrive\Doc
2 22 | graphic_section_adc

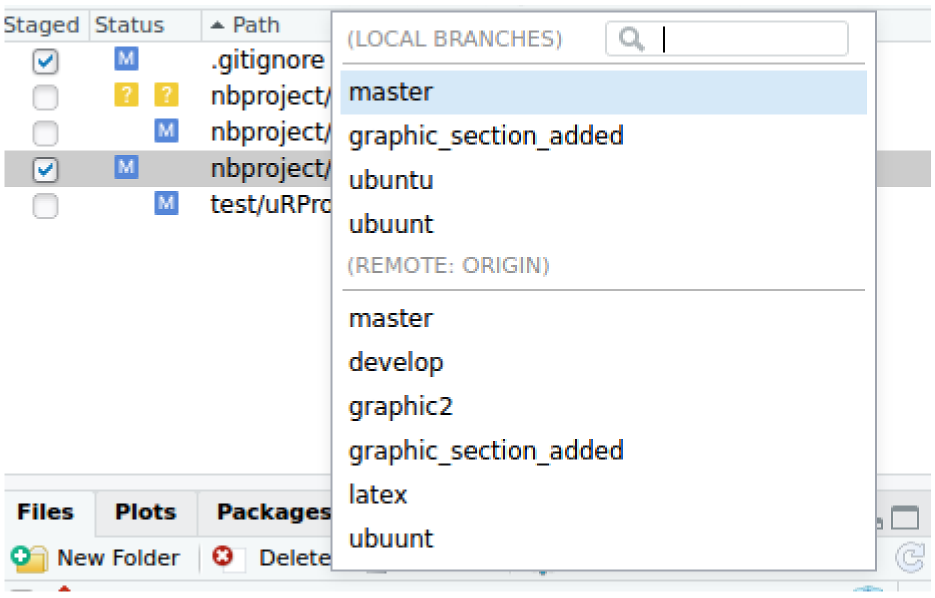
Graph	Description	Date	Author	Commit
	Uncommitted changes	4 Mar 2017 22:08	*	*
	graphic_section_added master more graph options added	28 Feb 2017 0:46	zekai otles <zekai c	a9832ec
	figure caption added	25 Feb 2017 9:33	zekai otles <zekai c	9510321
	minor delete not rqueired library	25 Feb 2017 8:58	zekai otles <zekai c	0a234f2
	First time checkin	25 Feb 2017 8:57	zekai otles <zekai c	856db21
	update index table	22 Feb 2017 23:29	zekai otles <zekai c	6b44788
	firstTime checkin cor and covariance calcluculation	22 Feb 2017 0:27	zekai otles <zekai c	80f419a
	minor modification	20 Feb 2017 23:47	zekai otles <zekai c	28d1b88
	Code for random NumGen is completed.	20 Feb 2017 23:11	zekai otles <zekai c	d723c76
	firdt time check in to create functionalrandonm Number generator	18 Feb 2017 9:57	zekai otles <zekai c	a976565
	set.seed number is added	18 Feb 2017 2:34	zekai otles <zekai c	9f4eb00
	rnorm and runif section added	14 Feb 2017 20:49	zekai otles <zekai c	e9522ab
	updated for summary section added	12 Feb 2017 1:20	zekai otles <zekai c	acb767a
	imple stats modified	11 Feb 2017 11:25	zekai otles <zekai c	5c858dd

When another branch checked out from R studio, it shows what files are checkout from the branch.



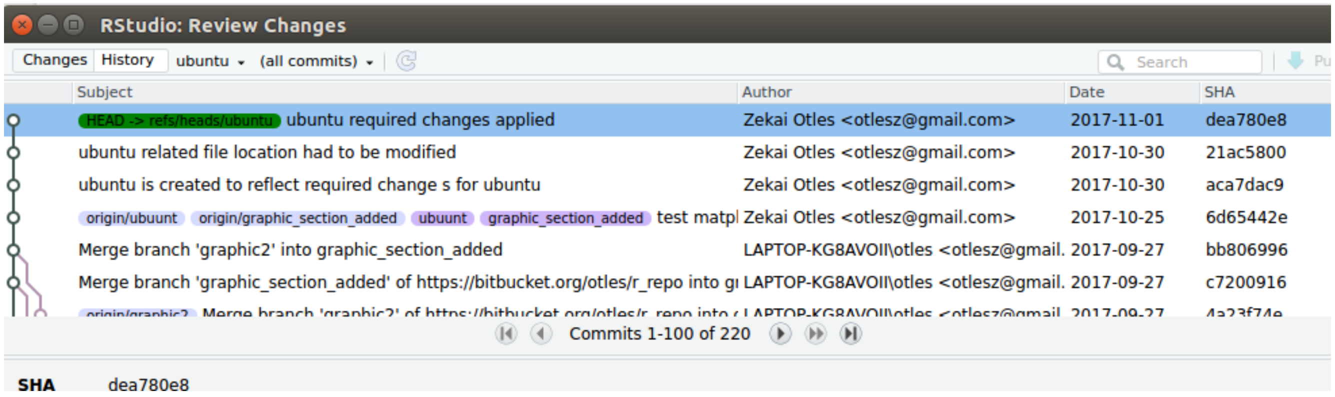
```
On branch graphic_section_added
Changes not staged for commit:
        modified:   test/uRProgramming_report.odt
        modified:   test/uRProgramming_templ.odt

Untracked files:
        .RData
        .Rhistory
        externalData/airquality.csv
        externalData/airquality.xlsx
        externalData/airquality.xml
        figures/Thumbs.db
        figures/gitCommit.PNG
        figures/gitModified.PNG
        lib/uRProgramming_1.1.1.tar.gz
        test/.gitignore
        test/.~lock.uRProgramming_templ.odt#
```

We can checkout a branch an worked on the branch independently. For example, we checkout the codes on ubuntu machine and name that branch as ubuntu. Let's assume, we are working on the codes to make them compatible with ubuntu . The “graphic_section_added” branch pulled from remote repository and created new branch called as “ubuntu” from that that branch, After modification made in the repository, the “ubuntu” branch is also *pushed* to the remote branch with the same name. Once the local branches pushed into the remote repository, they could be used by other people but also can be checkout from another system.



Staged	Status	Path
✓	M	.gitignore
	? ?	nbproject/
	M	nbproject/
✓	M	nbproject/
	M	test/uRPro

(LOCAL BRANCHES)
master
graphic_section_added
ubuntu
ubuunt
(REMOTE: ORIGIN)
master
develop
graphic2
graphic_section_added
latex
ubuunt



RStudio: Review Changes

Subject	Author	Date	SHA
HEAD -> refs/heads/ubuntu ubuntu required changes applied	Zekai Otles <otlesz@gmail.com>	2017-11-01	dea780e8
ubuntu related file location had to be modified	Zekai Otles <otlesz@gmail.com>	2017-10-30	21ac5800
ubuntu is created to reflect required change s for ubuntu	Zekai Otles <otlesz@gmail.com>	2017-10-30	aca7dac9
origin/ubuunt origin/graphic_section_added ubuunt graphic_section_added test matpl	Zekai Otles <otlesz@gmail.com>	2017-10-25	6d65442e
Merge branch 'graphic2' into graphic_section_added	LAPTOP-KG8AVOII\otles <otlesz@gmail.	2017-09-27	bb806996
Merge branch 'graphic_section_added' of https://bitbucket.org/otles/r_repo into gı	LAPTOP-KG8AVOII\otles <otlesz@gmail.	2017-09-27	c7200916
origin/graphic2 Merge branch 'graphic2' of https://bitbucket.org/otles/r_repo into	LAPTOP-KG8AVOII\otles <otlesz@gmail.	2017-09-27	4a23f74e

Commits 1-100 of 220

SHA dea780e8

Git also allows us to merge different branches within a local or remote repositories. Merging allow us to combine different branches, permit developer freely develop codes in a separate branches until they are ready to finalize the code.

2.3 *Using IDE to develop R scripts*

R studio or other tools to develop application,

2.4 *Writing functions*

Simple example

2.5 *Create R package System*

Build R package Rtools needs to be installed (Refence) for windows and Linux.

Windows insallation and linux instalation. Windows installation is requires downloading Rtools.exe for the R version. Version of R can be found by *version* to the R shell window.

R CMD build uRProgramming

Show them how to build pacjkage from GUI

R CMD BATCH odfWeave.R

3 Simple Data Types and Objects

R language has following defined data types, Double, Character, Logical and Complex. Numeric variable as a default defined as double,simple text string defined as Character, single character [T and F] is defined as Logical,and complex numbers are defined as Complex.

Integer objects are created with integer, as.integer keywords. Similarly numeric objects are created with numeric and as.numeric keywords. Boolean, Double, Character, Logical, Complex objects are created by their types [boolean, double,character,..] and as.types [boolean.double,..], The variables or vector also can be casted from one type to another type using similar keywords.

```
library(uRProgramming)
dataTypes(24)
dataTypes('ali')
dataTypes(T)
a<-as.integer(24)
dataTypes(a)
standard output from above lines
[1] "double"
[1] "character"
[1] "logical"
[1] "integer"
```

In the following examples show how casting works. Following snippet it, can be copied and saved as testDataTypes.R. This R script should be sourced in the R shell in order to have function testDataTypes(), and run the program. The x vector created as sequence of variables between 1 and 100 increment by 10.011. The x is created as default double variable. The x vector is casted to integer y vector. The later part of the example demonstrates, casting x vector into z character vector.

```
testDataTypes<-function(){
```

```
#create logical variable
iTest<-T
is.logical(iTest)
if(iTest ==  is.logical(iTest)) cat('Logical ')
#create simple double vector
x<-seq(1,100,by=10.011)
is.double(x)
if(iTest ==  is.double(x)) cat('x is double Vector  ')
#cast double vector to integer
y<-as.integer(x)
is.integer(y)
if(iTest ==  is.integer(y)) cat('y is integer Vector ')
#cast to character vector
z<-as.character(x)
is.character(z)
if(iTest ==  is.character(z)) cat('z is character Vector ')
}
```

Source this above program, and run the program (testDataTypes())

```
TRUE
Logical
1 11.011 21.022 31.033 41.044 51.055 61.066 71.077 81.088 91.099
x is double Vector
1 11 21 31 41 51 61 71 81 91
y is integer Vector
1 11.011 21.022 31.033 41.044 51.055 61.066 71.077 81.088 91.099
z is character Vector
```

4 Control Structures

Control structures are used to change a flow or follow a logic. The control structures are easily could make any program more followable for maintenance and understanding, they could also turn the most unfollowable code. Control structures are very straight forward when used in a modular way in a less number of statements or calling other reusable functions.oo

In the following sections, the usage of *if* block will be explained with examples. The samples should help reader to solidify the understandif of te control structures.

4.1 *If statements*

Let's use simple vector which contains only numbers, check if numbers is equal to specific value assign iFound as True.

If (Statements) { Assignment}

```
x<-c(0,3,6,9,12,15,18,21,24)
if(x[which(x==6)]==6){iFound<-TRUE}
print(iFound)
[1] TRUE
```

In the above example, which finds the index number for the value. The usage of *which* in vector is pretty powerful as shown in the example. The *which* statement is pretty handy determining the index of vector or array. When the vector with right index with exact value found, iFound assigned to be *true*.

Logical operators in R are

Operator	**Description**
==	Equal to
!=	Not equal to
<	Less than
<=	Less or equal to
>	Greater than
>=	Greater or equal to
&	And
|	Or

Following code snippet is simple example of how to use logical operators. The first example is the exacty *equal* to operator. The second example demonstrates *less than or equal* to operator and assigning the value based on the result. The third example shows *greater than* comparison.The last example is combination *greater than, and, less then or equal* to.

```
Ex. 1
if(demDat[indx,1] ==1 ) {demDat[indx,1]<-'M'}

Ex.2
if(demDat[indx,2] <=25 )demDat[indx,2]<-'Y'

Ex.3
if(demDat[indx,2] > 50 ) {demDat[indx,2]<-'O'}

Ex.4
if(demDat[indx,2] > 25 & demDat[indx,2] <=50 )demDat[indx,2]<-'M'
```

4.2 *If then Else statements*

If then else block is used when comparison expected to yield result either true or false. It gives the programmer, the power to control logical flow. In R programming, if then else blocks are in the following format

If (Statements) { Assignment}

else (statements) { Assignment}

4.3 Switch statements

One can use switch statement to modular conditional statements. The switch statement evaluates according the to the first argument.

switch(Expression,List)

The first argument determines how the results will be evaluated. The first line shows a number in the first argument will select the corresponding list element, in the second line of code snippet shows the the corresponding character strings of list element.

```
> switch(3, 1.5, 2, 300, 4)
[1] 300
> switch("iki", bir = "bir", iki = "Two", uc = "uc")
[1] "Two"
```

Following function demonstrates how simple if then else block could be replaced by modular switch statements, this type of example can be extended to more modular functionality.

```
#Simple example for data types casting using swith statement
usingSwitch<-function(x,dataTypes){

  switch (dataTypes,
    integer = {x<-as.integer(x)},
    double = {x<-as.double(x)},
    character = {x<-as.character(x)}

  )

}
> x <- seq(1, 100, by = 10.011)
> x
[1]  1.000 11.011 21.022 31.033 41.044 51.055 61.066 71.077 81.088 91.099
> xC <- usingSwitch(x, "character")
> xC
[1] "1"      "11.011" "21.022" "31.033" "41.044" "51.055" "61.066" "71.077"
"81.088" "91.099"
> xI <- usingSwitch(x, "integer")
> xI
[1]  1 11 21 31 41 51 61 71 81 91
```

5 Loop Structures

The loop structures are needed to access elements of vectors, matrix, list and data frame. The access of the data is important to reach or manipulate the elements. The *for* and *while* loop are used to iterate elements.

5.1 *For loop*

The for loop is used for sequentially access the data elements. Each element of data stuctures can be reached, used or manipulated within the *for* loop. The *for* loop is in the following format.

```
for (values in sequence) {
        statement
}
```

The following code snippets how *for* loop is used. In the following example, 100 number with specified seed assign to vector. This vector has only integers for demonstration pupose.

```
set.seed(312)
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
for(indx in 0:length(xInt) ){
if(indx %%10 ==0){print(xInt[indx])}
}
numeric(0)
[1] 450
[1] 261
[1] 405
[1] 295
[1] 763
[1] 356
[1] 833
[1] 329
[1] 419
[1] 948
```

5.2 *While loop*

The access of certain elements could be controlled by using while loop, the expression determines executing statements or not.

```
while(expression){
        statement
}
```

In the following example, the elements between two indices are assigned as either *even* or *odd*. Execution will stop when expression is false, otherwise statements within the block is executed.

```
set.seed(312)
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
index<-80
while(index<=length(xInt)){
xInt[index]<-ifelse(as.integer(xInt[index]) %%2==0,"Even","Odd")
if(index %% 4 == 0 ){print(xInt[index])}
index<-index+1
}
[1] "Odd"
[1] "Even"
[1] "Odd"
[1] "Odd"
[1] "Odd"
[1] "Even"
```

6 Data Structures

In my experience as software engineer and data analyst, introduction to database and utilizing with the data structures would be valuable experience in terms of programming, Specifically storing and retrieving data from database is a great experience for the structured programming. Thus, we introduce the database in this section. The MySQL was chosen because of of simplicity to installation and free to use. The community version is freely avaliable to use, it can be downloaded from oracle. The RMYSQL package allows us utilize some of database functionality within R scripts. Avaliable datasets from basic R installation will be loaded to the urprogramming schema into MySql database. Later those dataset tables will be used to demonstrate to basic data structues like vector, list and matrix.

Insert air quality data from R datasets into airquality table into the schema. Later on selected data from mysql table will be converted into vector and list, etc.

```
#insert data into "airquality" table
insertData<-function(){
library("RMySQL")
#if(con)dbDisconnect(con)


numR<-dim(airquality)[1]
#con <- dbConnect(RMySQL::MySQL(), dbname =
"urprogramming",password="urprogramming")

con <- dbConnect(RMySQL::MySQL(),  user = 'rprogrammer', password =
'uRProgramming', host = '127.0.0.1' , dbname = 'urprogramming')

results<-dbSendQuery(con, "SELECT * FROM airquality")
dataFrm<-dbFetch(results)

#dbFetch(results)
print(dim(dataFrm)[1])
#print(dataFrm[1:5,])
iSkip<-FALSE
if(dim(dataFrm)[[1]] > 1){iSkip<-FALSE
}else {
   iSkip<-TRUE
}

if(iSkip){
  for (iRow in 1:numR) {

# print(iRow)
  data<-airquality[iRow,]

  data[is.na(data)]<-999999
  statement<-paste("INSERT INTO ","airquality")

#  print(statement)
  insertQuery <- paste(paste(statement,"VALUES("), paste(data, collapse = ", "),
")")
#
print(insertQuery)
dbGetQuery(con,insertQuery)
  }
}#end of iSkip
dbDisconnect(con)

}
```

In order to load airquality data to the table in MySql database, connection to database will open with following statement.

con <- dbConnect(RMySQL::MySQL(), user = 'rprogrammer', password = 'uRProgramming', host = '127.0.0.1' , dbname = 'urprogramming')

The dbConnect statement in RMySQL library requires name of database, schema, username and password. Once the connection is established, the connection object is used to populate the rows in table using the statements starting with "INSERT INTO airquality VALUES(,,,,)".

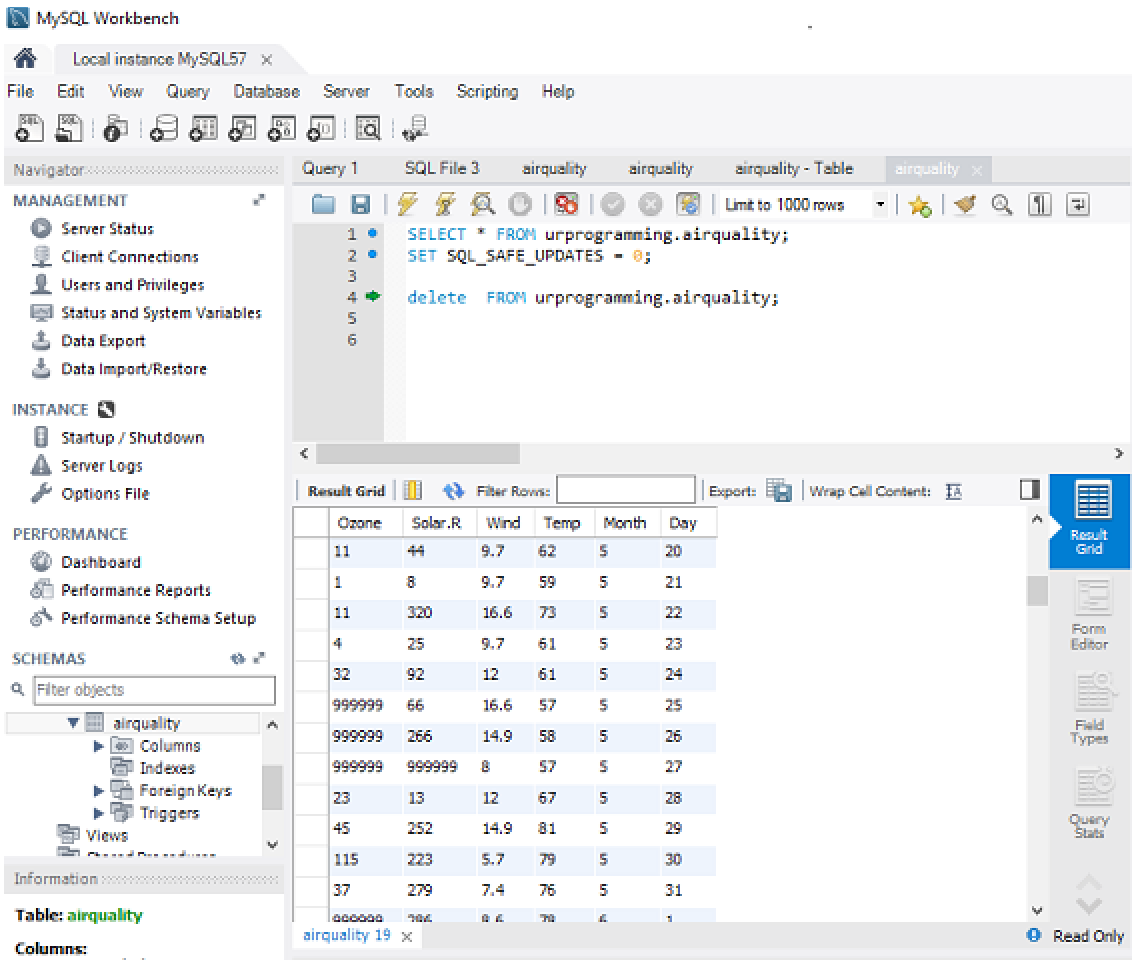
```
> insertData()
[1] 153
[1] TRUE
> results <- dbSendQuery(con, "SELECT * FROM airquality")
> dataFrm <- dbFetch(results, )
> print(dim(dataFrm))
[1] 153   6
> print(dataFrm[1:5, ])
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
2     36     118  8.0   72     5   2
3     12     149 12.6   74     5   3
4     18     313 11.5   62     5   4
5 999999  999999 14.3   56     5   5
```

The following statement will select each row and form a result set.

results<-dbSendQuery(con, "SELECT * FROM airquality")

The result set will be loaded to dataframe using *dbFetch(results,)* function from the RMySQL library. The i*nsertData* function is configured so the records will only inserted when there is no data in the table. Here, we introduced the basic functionality of database in R. Similar to *insert and select* , *update*, *truncate* could be used to manipulate data in tables. There are many options for SQL statements to create complex joins or selecting the subsets of the data . This kind of SQL statements can be found in MySQL documentations, that's why we did not want to go beyond the basic of SQL statements.

The following figure shows the instance from MySQL Workbench. The tool can be used to execute SQL statements.



```
SELECT * FROM urprogramming.airquality;
SET SQL_SAFE_UPDATES = 0;

delete  FROM urprogramming.airquality;
```

Ozone	Solar.R	Wind	Temp	Month	Day
11	44	9.7	62	5	20
1	8	9.7	59	5	21
11	320	16.6	73	5	22
4	25	9.7	61	5	23
32	92	12	61	5	24
999999	66	16.6	57	5	25
999999	266	14.9	58	5	26
999999	999999	8	57	5	27
23	13	12	67	5	28
45	252	14.9	81	5	29
115	223	5.7	79	5	30
37	279	7.4	76	5	31
999999	286	8.6	78	6	1

Similar data stored in in data bases, one could easily have data stored so called as flat data asci files. Those are created from either either simple editor or spread sheet programs. R has very easy interfaces to read flat files or spreadsheet data. The flat file stored in "/home/zekai/R_workspace/r_repo/externalData/airquality.csv" can be read with following statement

read.csv(file="/home/zekai/R_workspace/r_repo/externalData/airquality.csv",header=TRUE,sep=",")

If data is in spreadsheet, it can be saved as flat file from spreadsheet then read data from flat file using similar staement. The data file could be read using read.csv funtion from R util pacakage. There are other packages avaliable for reading spreadsheet data to R.

6.1 Data Frame

Data frame is one of the available data structures. The data frame is similar to the spreadsheet data, one can easily see the structure of the data. Once the data saved in data frame structure, the structure of data, dimensions and contents of data is readily available in R. The structure of data is determined from **str(name of data frame object)** as shown in the the following example

str(dataFrm).

The statement gives the structure of data, name of data variables and also type of variable stored. As seen in the example, air quality data store Ozone, Solar.R, Temp, Month, Day variables as **int** and wind as **num**. These variables can be changed using casting as we shown earlier in Simple Data Type section. We can figure out the data stored in a data frame using existing function **dim** in R base functions, **colnames** (again from R base functions) refers the variable names stored in data structures. The **View** function R util functions can display the data frame contents so we can have some idea about the data. The following section

View(dataFrm[1:4,])

is displaying the data from the first through fourth row in dataFrm.

```
results<-dbSendQuery(con, "SELECT * FROM airquality")
dataFrm<-dbFetch(results)
the results from select assigned to dataFrm. The dataFrm has following structure
str(dataFrm)
'data.frame': 153 obs. of  6 variables:
$ Ozone  : int  41 36 12 18 999999 28 23 19 8 999999 ...
$ Solar.R: int  190 118 149 313 999999 999999 299 99 19 194 ...
$ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
$ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
$ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
$ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
dim(dataFrm)
[1] 153   6
colnames(dataFrm)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
View(dataFrm[1:4,])
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
```

6.2 *Vector*

Vector is used for atomic data types (integer,numeric, complex, and character) and list storage. It is a powerful way to store data and it usage in the computations is extremely simplifies the programming. The examples will demonstrate the power of vector calculations. Temperature from Fahrenheit is easily converted to Celsius with following statement.

tempAsCels<-5*(dataFrm$Temp-32)/9.0

```
> tempAsCels <- 5 * (dataFrm$Temp - 32)/9
> cat("is.vector(tempAsCels)")
is.vector(tempAsCels)
> print(is.vector(tempAsCels))
[1] TRUE
> cat("View(tempAsCels(1:5)")
View(tempAsCels(1:5)
> print(tempAsCels[1:5])
[1] 19.44444 22.22222 23.33333 16.66667 13.33333
```

It is also easy to dissect and examine data stored in vector format. The following simple snippet will demonstate how to retrive the month data from the dataFrm object. We have determined the months which days are 80 degree Fahrenheit . It also shown number of 80 degree days for our data.

```
dataFrm$Month[dataFrm$Temp==80]
length(dataFrm$Month[dataFrm$Temp==80])
[1] 6 6 7 8 9
[1] 5
```

The **which,** function is available from the R base package, is used to determine the indices of the vector when the condition meets up the criteria. The following snippets show which element of the temperature vector was exactly 80 degrees Fahrenheit.

```
which(dataFrm$Temp==80)
[1]  45  59  76 106 130
```

6.3 *Matrix*

The matrix is available in R to store two dimensional **array** in a matrix format. The discussion of matrix without **array** will not be complete to understand data structures in **R** programming. The array can store vector data in one, two or dimensional format. The vector data is stored without array bound information, but **array** and **matrix** needs complete information about dimensions.

The **array** needs the information about data types and dimensions of the array. The **array** stores one to many dimensional data. In the following example, row_names stored in a one dimensional **array** .

row_Names<-array("",dim=length(unique(dataFrm$Month)))

Structure of **array** can be found using **str** from util package in R. The **str** function will give data types and dimension of the **array**. The following demonstrates how to use **str** function with **array**.

strr(row_Names)

Another useful function to check whether data stored in array or not. If the data stored in an array, *is.array(row_Names*) will return *TRUE*.

The **matrix** is a special case of **array** which is a two dimensional data structure. The **matrix** need data types, row and column dimensions. The tempAsMatrix is defined as **matrix** in the following example.

tempAsMatrix<-matrix("",nrow=length(unique(dataFrm$Month)),ncol=30)

```
#form a Matrix from dataFrm data
#the purpose of this function to ceate Temp matrix
formTempMatrix<-function(dataFrm)
{
  monthAsNum<-unique(dataFrm$Month)
  row_Names<-array("",dim=length(unique(dataFrm$Month)))
  tempAsMatrix<-matrix("",nrow=length(unique(dataFrm$Month)),ncol=30)


  for(mIndx in 1:length(monthAsNum)){
    row_Names[mIndx]<-
switch(as.character(monthAsNum[mIndx]),"5"="May","6"="June","7"="July","8"="Aug","
9"="Sep")
  }
  rownames(tempAsMatrix)<-row_Names

  length(unique(dataFrm$Month))
  rowIndex<-unique(dataFrm$Month)
  for (index in 1:length(rowIndex)){



    x<-dataFrm$Temp[dataFrm$Month==rowIndex[index]]
    tempAsMatrix[index,]<-x[1:30]
  }

  return(tempAsMatrix)
}
```

The **str** function for matrix type will return data types, and bounds of the each dimensions. The base package has a function to determine whether data structure is matrix or not. In the following example,

is.matrix(tempAsMatrix)

the return value will be TRUE.

6.4 List

The list is very powerful data structure, heterogeneous data can be stored in a list. The list can contain list, data frame, array and matrix in a structure. The *dataFrm*, *row_Names* and *tempAsMatrix* data structures are all stored in a **list** called aList. The data structure can be retrieved using the list$name structure, such as *aList$dataFrm*.

```
#demonstate the list
useList<-function(dataFrm,row_Names,tempAsMatrix){
  aList<-list(dataFrm=dataFrm,rowNames=row_Names,matrix=tempAsMatrix)
  str(aList)
  return(aList)

}


str(aList)
List of 3
$ dataFrm :'data.frame': 153 obs. of  6 variables:
  ..$ Ozone  : int [1:153] 41 36 12 18 999999 28 23 19 8 999999 ...
  ..$ Solar.R: int [1:153] 190 118 149 313 999999 999999 299 99 19 194 ...
  ..$ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:153] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:153] 1 2 3 4 5 6 7 8 9 10 ...
$ rowNames: chr [1:5(1d)] "" "" "" "" ...
$ matrix  : chr [1:5, 1:30] "" "" "" "" ...
```

As seen above, the list data structure is very convenient way to store multiple data structures into a data structure.

7 Simple Statistical Analysis

This section is simple introduction to statistical analysis. There will be brief explanation about creating normally distributed variables used statistical analysis, creating simple summary mean, standard deviation. The simple statistical functions are available stats, base, time functional packages. The functional examples will show how to use those simple statistics using already defined data structures. Introduction how to use simple statistics against their data will give foundation to extend the readers ability to do further complex statistical analysis. This will also help utilizing the other packages in available by R community.

```
#demonstrate simple distribution and some statistics
simpleStats<-function(n,mean,sd){
x<-rnorm(n,mean=mean,sd)
#mean of variable
mean=mean(x)
ave=ave(x)
max=max(x)
min=min(x)
median=median(x)
sd=sd(x)
sum=sum(x)
summary=summary(x)
return(list(mean=mean,ave=ave,max=max,min=min,median=mean,sd=sd,summary=summary,su
m=sum))
}
```

In the example, normally distributed random variable created using ***rnorm*** function from ***stats*** package. Using some of the available functions such as ***mean***, ***summary*** and ***sd*** from ***base*** and ***stats*** packages. The ***mean*** from ***base*** package gives the arithmetic mean of the values. The ***min*** and ***max*** returns minimum and maximum values for the input values. The ***sd*** is standard deviation of the input values. The summary function from base package returns ***min*** and ***max*** values , the first and third quartiles, ***median*** and ***mean*** of input values.

```
xStats<-simpleStats(100,273.16,15)
str(xStats)
List of 8
$ mean   : num 273
$ ave    : num [1:100] 273 273 273 273 273 ...
$ max    : num 304
$ min    : num 232
$ median : num 273
$ sd     : num 14.4
$ summary:Classes 'summaryDefault', 'table'  Named num [1:6] 232 264 273 273
283 ...
  .. ..- attr(*, "names")= chr [1:6] "Min." "1st Qu." "Median" "Mean" ...
$ sum    : num 27295
```

Normally distributed random numbers is generated by using ***rnorm*** function from stats package. The **runif** generates uniformly distributed random numbers. The random number generators ran four times in the example, and selected the fifth element of the storage vectors (**xTempNorm** and **xTempUni**). Since the script is not using fixed seed number for the random number generators, the storage vectors will consist of different elements as we shown here with the fifth elements of the vectors.

```
#Create randomly number generated vectors
#simple statistics
randomNumGen<-function(seed=FALSE){
print("Normally distributed Random Number")
  for(indx in 0:3){
    print(paste("Iteration ",indx,sep= ""))
    if(seed) set.seed(6543)
    xTempNorm<-rnorm(100,273.16,15)

    print(paste("xTempNorm[5]=",xTempNorm[5]))
    cat('summary(xTempNorm)\n')
    print(summary(xTempNorm))
  }

  print("Uniformly distributed Random Number")
  for(indx in 0:3){
    if(seed)set.seed(6543)
    xTempUni<-runif(100,240,320)
    print(paste("Iteration ",indx,sep= ""))
    print(paste("xTempUni[5]=",xTempUni[5]))
    cat('summary(xTempUni)\n')
    #

    print(summary(xTempUni))

  }

return(list(xTempNorm=xTempNorm,xTempUni=xTempUni))
}
[1] "Normally distributed Random Number"
[1] "Iteration 0"
[1] "xTempNorm[5]= 282.506660883275"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  241.3   261.6   274.4   273.5   284.2   321.4
[1] "Iteration 1"
[1] "xTempNorm[5]= 266.941109367306"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  236.3   260.4   274.3   272.8   284.6   314.1
[1] "Iteration 2"
[1] "xTempNorm[5]= 261.022663275379"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  235.5   261.8   271.0   271.1   280.9   305.0
[1] "Iteration 3"
[1] "xTempNorm[5]= 279.732265995178"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  235.7   262.1   274.4   273.2   282.8   304.0
[1] "Uniformly distributed Random Number"
[1] "Iteration 0"
[1] "xTempUni[5]= 245.185738392174"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  240.4   256.4   274.5   277.0   299.3   319.5
[1] "Iteration 1"
[1] "xTempUni[5]= 302.323486860842"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  240.6   255.8   271.9   277.4   299.8   319.9
[1] "Iteration 2"
[1] "xTempUni[5]= 266.91300611943"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  240.3   258.8   278.9   279.3   297.9   319.0
[1] "Iteration 3"
[1] "xTempUni[5]= 242.070628292859"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  240.4   256.3   277.7   277.9   301.1   319.9
```

The same example will be repeated for fixed seed numbers for the random number generation. When we set the seed the numbers the produced random number vectors using **rnorm** and **runif** will always contains same elements. The fifth elements for the **xTempNorm** and **xTempUni** vectors is always same for the given seed number 6543. The **summary** of the storage **xTempNorm** and **xTempUni** vector will yield same results because of the same seed number.

```
[1] "Normally distributed Random Number"
[1] "Iteration 0"
[1] "xTempNorm[5]= 288.495445822198"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  221.5   259.9   273.3   270.8   281.0   304.0 
[1] "Iteration 1"
[1] "xTempNorm[5]= 288.495445822198"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  221.5   259.9   273.3   270.8   281.0   304.0 
[1] "Iteration 2"
[1] "xTempNorm[5]= 288.495445822198"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  221.5   259.9   273.3   270.8   281.0   304.0 
[1] "Iteration 3"
[1] "xTempNorm[5]= 288.495445822198"
summary(xTempNorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  221.5   259.9   273.3   270.8   281.0   304.0 
[1] "Uniformly distributed Random Number"
[1] "Iteration 0"
[1] "xTempUni[5]= 272.028888594359"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  240.0   262.8   281.6   280.5   297.3   318.4 
[1] "Iteration 1"
[1] "xTempUni[5]= 272.028888594359"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  240.0   262.8   281.6   280.5   297.3   318.4 
[1] "Iteration 2"
[1] "xTempUni[5]= 272.028888594359"
summary(xTempUni)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  240.0   262.8   281.6   280.5   297.3   318.4 
[1] "Iteration 3"
[1] "xTempUni[5]= 272.028888594359"
summary(xTempUni)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  240.0   262.8   281.6   280.5   297.3   318.4 
```

The xRandGenT list is used to sample two vector datasets. The sample method from base R package used to select 15 random elements from vector datasets . Once the samples assigned the correlations values computed by using different methods avaliable from **stats** package. The covariance between **xTempNorm** and **xTempUni** vector datasets computed using **cov** function from **stats** package. The standard deviations calculated using **sd** function from he **stats** package.

```
corTest<-function(xVals=xVals,yVals=yVals){
x<-sample(xVals,size=15)
y<-sample(yVals,size=15)
#cor(x,y,use="everything",method=c("pearson","kendall","spearman"))
cor.test(x,y,method="pearson")
print(cor.test(x,y,method="pearson"))
sdXsdY=sd(x)*sd(y)
pearsonCor=cov(x,y,method="pearson")/sdXsdY
print(paste("Pearson=",pearsonCor))

cor.test(x,y,method="spearman")
print(cor.test(x,y,method="spearman"))

cor.test(x,y,method="kendall")
print(cor.test(x,y,method="kendall"))


}
corTest(xRandGenT$xTempNorm,xRandGenT$xTempUni)

Pearson's product-moment correlation

data:  x and y
t = -0.7144, df = 13, p-value = 0.4876
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.6426395  0.3530540
sample estimates:
       cor 
-0.1943601 

[1] "Pearson= -0.194360080385567"

Spearman's rank correlation rho

data:  x and y
S = 766, p-value = 0.1779
alternative hypothesis: true rho is not equal to 0
sample estimates:
       rho 
-0.3678571 
```

```
Kendall's rank correlation tau

data:  x and y
T = 40, p-value = 0.2395
alternative hypothesis: true tau is not equal to 0
sample estimates:
       tau
-0.2380952
```

8 Graphic Outputs

This section is designed to be how to represent data output in a graphical formats.

The visual presentation of data will help audience form an opinion much faster in human's brain. This might be very well represented by the common known phrase “A picture is worth a thousand words” (http://www.phrases.org.uk/meanings/a-picture-is-worth-a-thousand-words.html). This is mind, we will show examples of histogram, bar plot, plot, matplot is readily avaliable R **graphics** pckage.

```
#The purpose of this test to see effect of number of Normal distribution
#Normal distribution is back bone statistical theory
#to run this
#testNormal(numSet=100,mean=c(5,15,50),sd=c(10,5,3,1))
#num number of random observations
normalDist<-function(numSet=numSet,meanSet=meanSet,sdSet=sdSet){

   #opar<-par(ask=interactive())
   opar<-par(font.main=1,ps=9)
   layout(matrix(c(1,2,3,4),2,2,byrow=TRUE))
  for(numIndx in 1:length(numSet)){
   num<-numSet[numIndx]
   for(pIndx in 1:length(meanSet)){
      p=meanSet[pIndx]
      for(i in 1:length(sdSet)){

        x<-rnorm(num,mean=p,sd=sdSet[i])
        iqr<-(quantile(x)[4]-quantile(x)[2])
        psd<-iqr/1.35

        if(sd(x)<psd){
            tail="light"
        }else{
            tail="heavy"
        }
        titleOfPlot<-paste(num,"obs ",sep=" ")
        titleOfPlot<-
paste(titleOfPlot,paste("mean",sprintf("%4.2f",p),sep="="),sep="   ")
        titleOfPlot<-
paste(titleOfPlot,paste("sd",sprintf("%4.2f",sdSet[i]),sep="="),sep="   ")
        titleOfPlot<-paste(titleOfPlot,paste("tail",tail,sep="="),sep="   ")
        hist(x,probability=T,main=titleOfPlot)
        #xVals<-0:nset[i]
        #points(xVals,dbinom(xVals,nset[i],p),type="h",lwd="3")
        #points(xVals,dbinom(xVals,nset[i],p),type="p",lwd="3")


        curve(dnorm(x,mean=p,sd=sdSet[i]),add=T)
        points(quantile(x)[3],dnorm(quantile(x)
[3],mean=p,sd=sdSet[i]),type="h",lwd="3")
        points(quantile(x)[2],dnorm(quantile(x)
[2],mean=p,sd=sdSet[i]),type="h",lwd="2")
        points(quantile(x)[4],dnorm(quantile(x)
[4],mean=p,sd=sdSet[i]),type="h",lwd="2")
      }
    }
   #frame()

  }
  par(opar)


   #print(x)
  return(as.numeric(x))
}
```

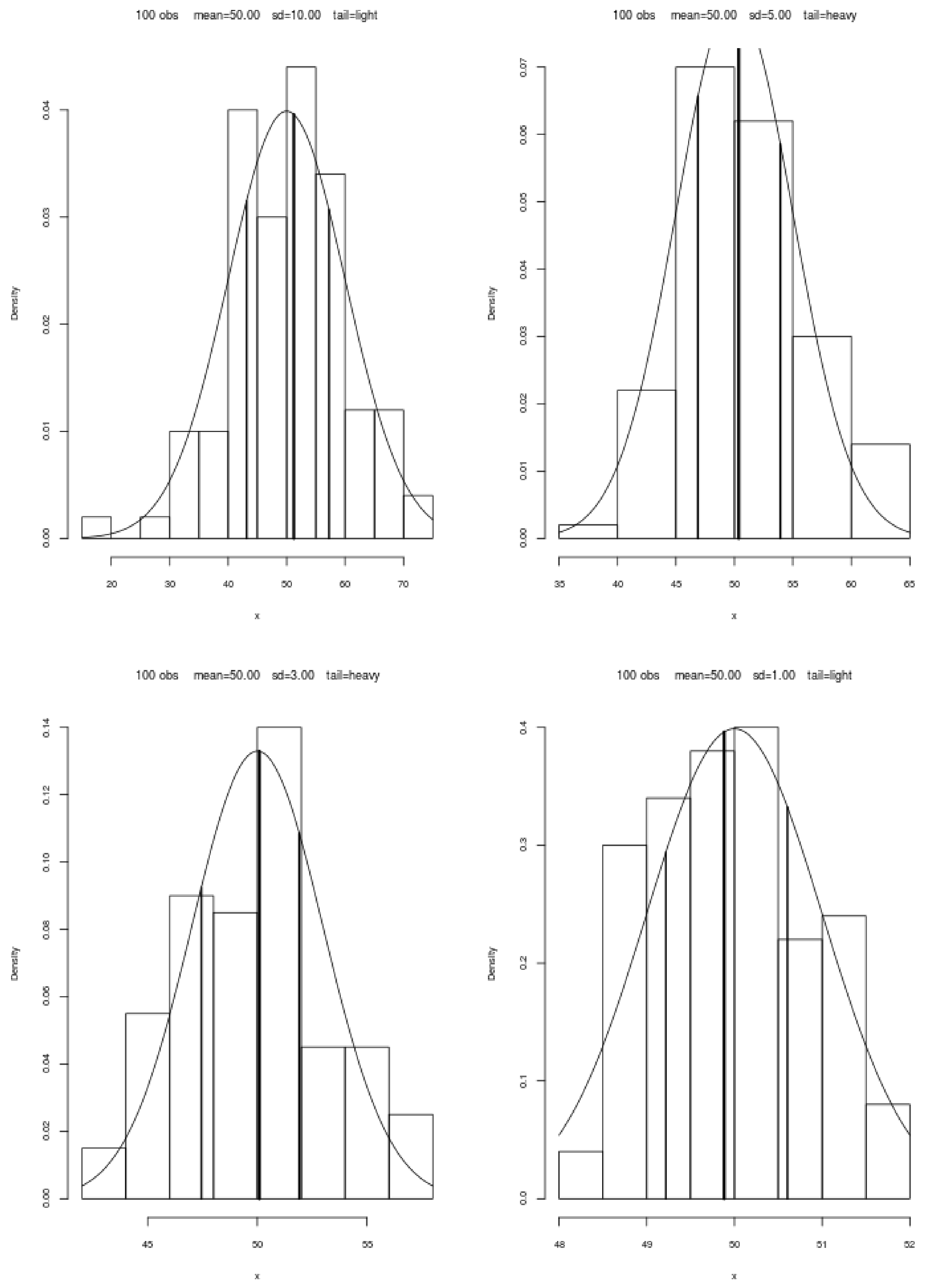


Figure 2 : Example Histogram

```
######################################
# Function freqPlot
#######################################
#The purpose of these function to teach some statistics and demonstrate R
#functionality
#This function plots  barplots for for given matrix data,
#with bar stacked on each other or bars side of each other
#different pattern style filling is also an option
#This function written by Zekai Otles <otlesz@gmail.com>

freqPlot<-
function(mat=freqData,xlabels=NULL,ylabels=NULL,onePlot=FALSE,besideOrNot=TRUE,isC
ol=TRUE,titleOfPlot=NULL){

#num of Data Columns besides the first column with identifies common data variable
#in this case is plate number
numRows<-dim(mat)[1]
#layout for the plots
layOutSetUp(1)
if(!onePlot)
{
  layOutSetUp(numRows-1)}
  else{
    layOutSetUp(2)
  }

colArray<-array("",dim=numRows)
if(isCol & onePlot){
  colArray<-c("#00007F", "blue", "#007FFF", "cyan","#7FFF7F", "yellow", "#FF7F00",
"red", "#7F0000")
}else{
   colArray="gray"
}
titleOfP<-paste(titleOfPlot, "Months")
if(!onePlot){

  for(index in 2:numRows){

    titleOfP<-paste(titleOfPlot,rownames(mat)[index])
barplot(as.numeric(mat[index,]),xlab=xlabels,ylab=ylabels,col=colArray)
    title(titleOfP)
        box()
  }#end of for loop for index
}else {

#barplot(mat,xlab=xlabels,ylab=ylabels,beside=besideOrNot,col=colArray[1:numRows])
  barplot(mat,beside=besideOrNot,xlab=xlabels,ylab=ylabels,col=colArray)
   title(titleOfP)

    box()
}
}#end of function
     55-60 60-65 65-70 70-75 75-80 80-85 85-90 90-95 95-100 100-105
May      8     7     9     4     1     1     0     0      0       0
June     0     1     1     5    13     5     3     2      0       0
July     0     0     0     2     1    17     7     3      0       0
Aug      0     0     0     2     9     6     9     2      2       0
```

```
Sep      0    2    5    6    8    4    1    4     0      0
```

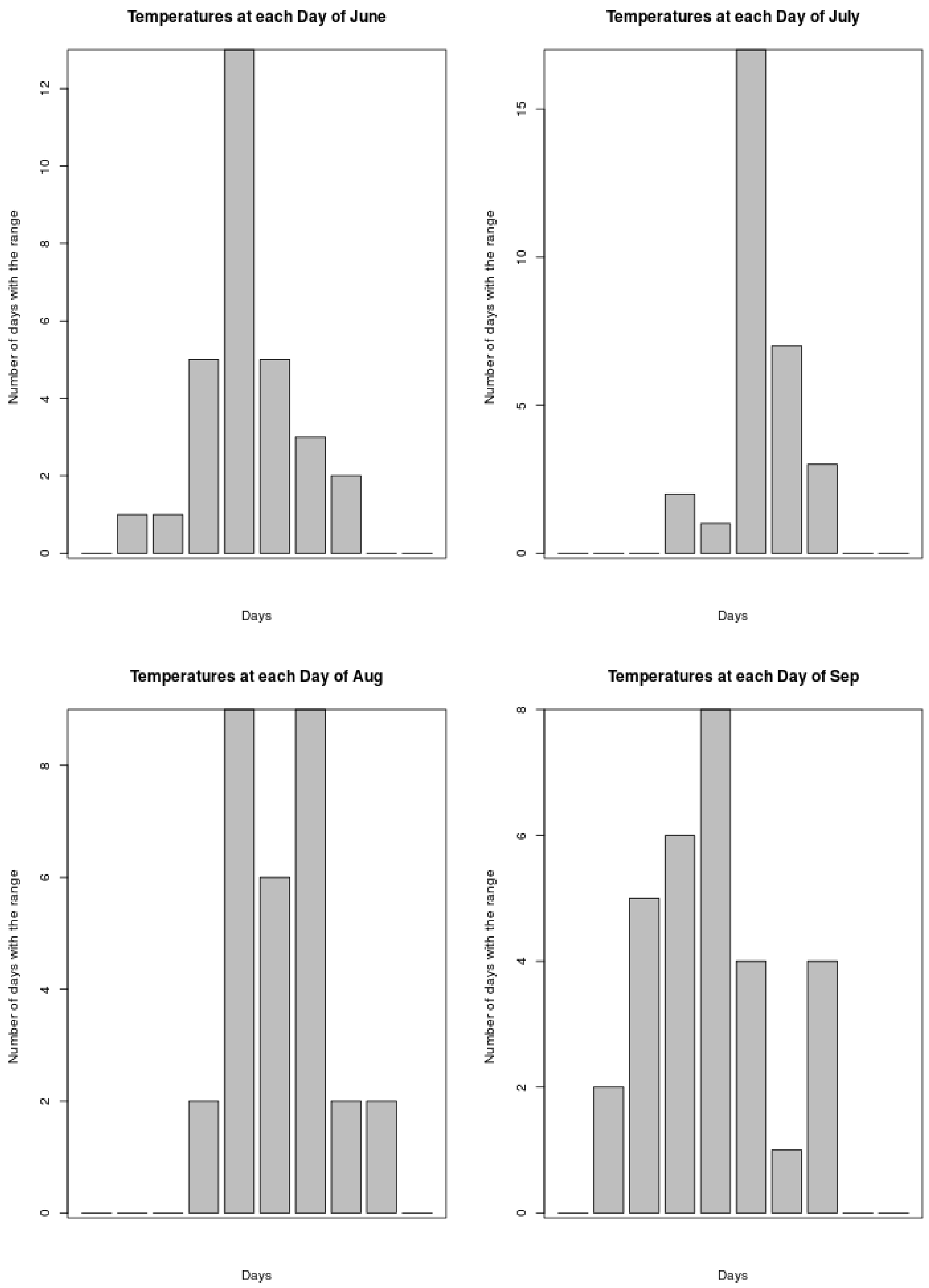


Figure 3 : Example Bar Plot

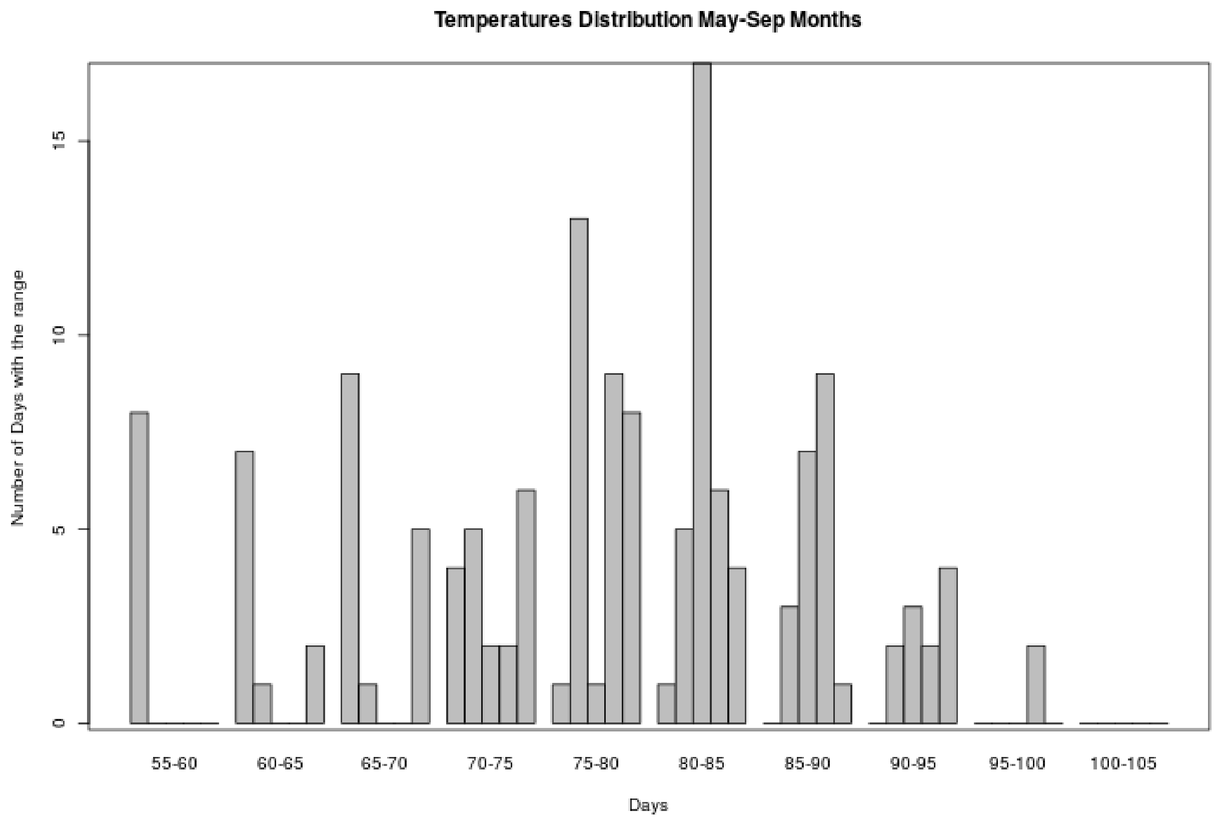


Figure 4 : Example Bar Plot with combined data

stackbar plot

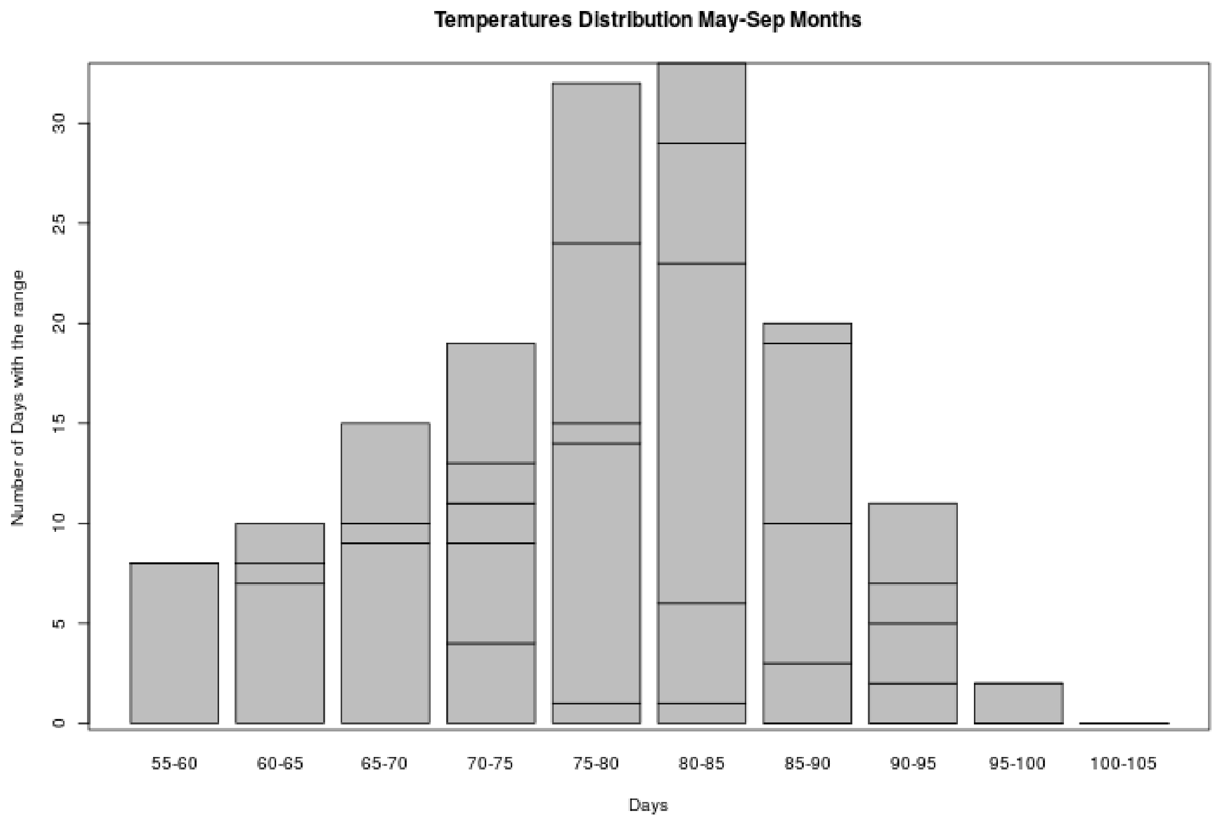


Figure 5 : Example Bar Plot with combined data

Multi line plots

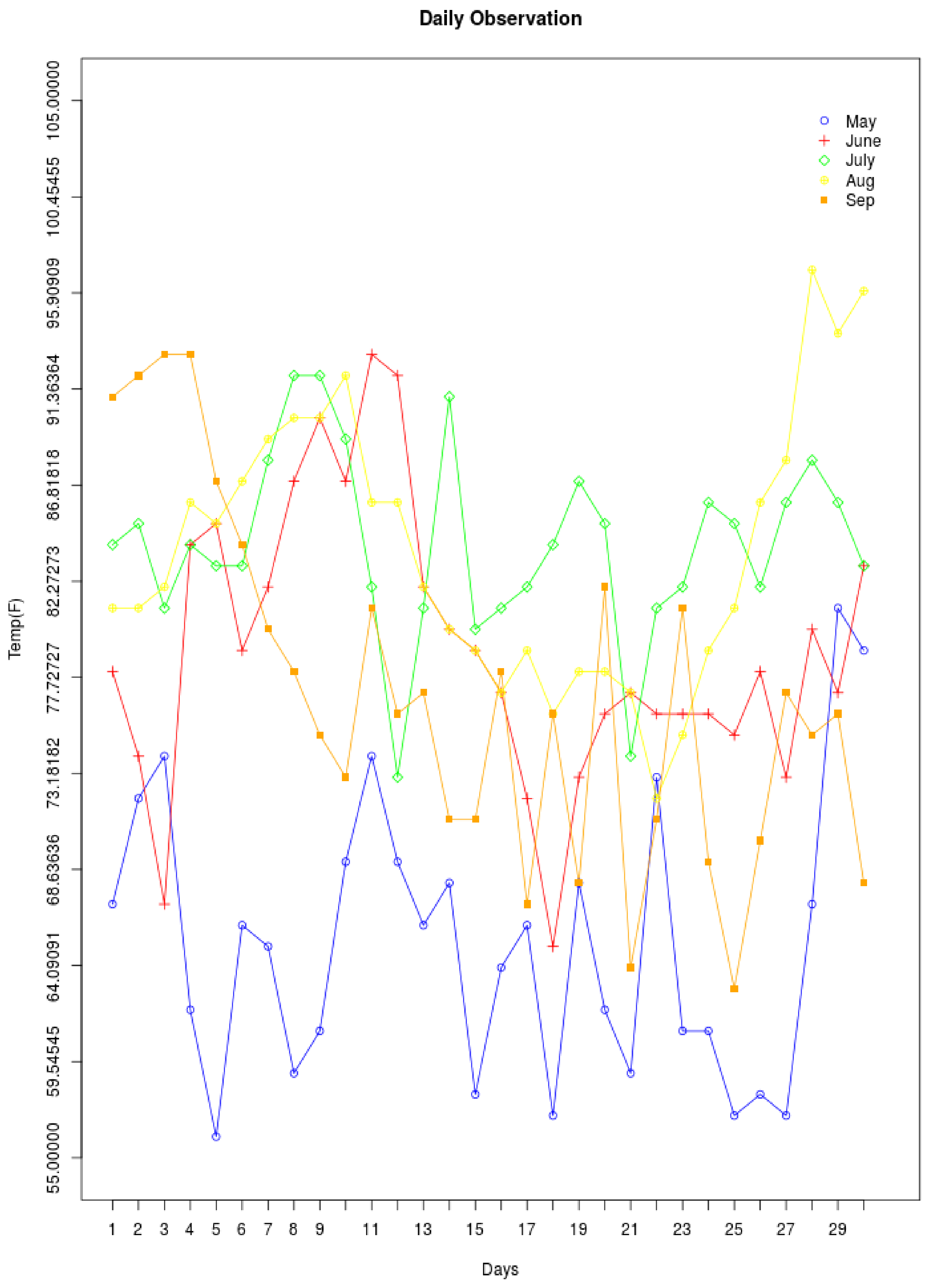


Figure 6 : Observation

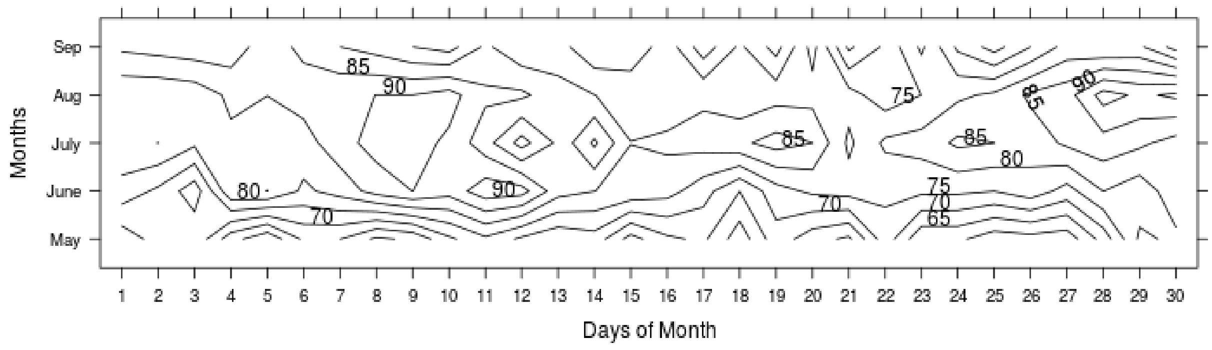
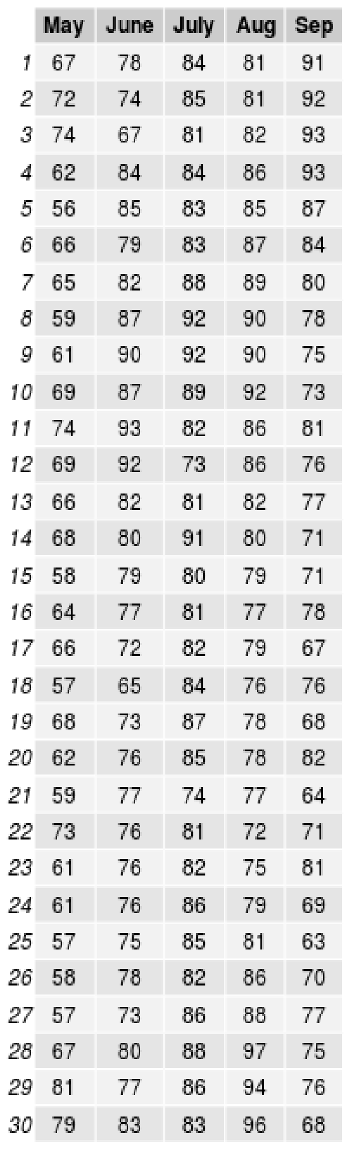


Figure 7 : Daily Temperature Values Plotted as Contours

9 Simple Table Outputs

```
May June July Aug Sep
```



	May	June	July	Aug	Sep
1	67	78	84	81	91
2	72	74	85	81	92
3	74	67	81	82	93
4	62	84	84	86	93
5	56	85	83	85	87
6	66	79	83	87	84
7	65	82	88	89	80
8	59	87	92	90	78
9	61	90	92	90	75
10	69	87	89	92	73
11	74	93	82	86	81
12	69	92	73	86	76
13	66	82	81	82	77
14	68	80	91	80	71
15	58	79	80	79	71
16	64	77	81	77	78
17	66	72	82	79	67
18	57	65	84	76	76
19	68	73	87	78	68
20	62	76	85	78	82
21	59	77	74	77	64
22	73	76	81	72	71
23	61	76	82	75	81
24	61	76	86	79	69
25	57	75	85	81	63
26	58	78	82	86	70
27	57	73	86	88	77
28	67	80	88	97	75
29	81	77	86	94	76
30	79	83	83	96	68

```
pValues<-array("",c(5,6))

colnames(xMatr)<-colNames
for (colIndx in 1:length(colNames)){
  x<-
chickwts[which(as.character(chickwts$feed)==as.character(colNames[colIndx])),]
  x1<-x$weight
  #print("x",x1)
  for (rowIndx in 1:length(x1)){

    xMatr[rowIndx,colIndx]<-x1[rowIndx]
  }

}

maxLen<-dim(xMatr)[2]-1

for (rrowIndx in 1:maxLen){
  iStart<-rrowIndx+1
  for (ccolIndx in iStart:dim(xMatr)[2]){

    pvals<-t.test(xMatr[,rrowIndx],xMatr[,ccolIndx],paired=TRUE,
conf.level=0.95)$p.value

    pValues[rrowIndx,ccolIndx]<-format(pvals,digits = 1 ,nsmall = 5, justify =
"left")

    print (pValues[rrowIndx,ccolIndx])
  }
}

return(list(xMatr=xMatr,pValues=pValues))
}
[1] "9e-05"
[1] "8e-04"
[1] "0.09916"
[1] "0.02294"
[1] "0.79590"
[1] "0.02223"
[1] "2e-04"
[1] "4e-06"
[1] "9e-06"
[1] "0.01167"
[1] "0.16020"
[1] "6e-05"
[1] "0.46506"
[1] "0.00584"
[1] "0.00252"
```

source paired_test and modify heightData input it will be read

	casein	**horsebean**	**linseed**	**meatmeal**	**soybean**	**sunflower**
1	368	179	309	325	243	423
2	390	160	229	257	230	340

	casein	horsebean	linseed	meatmeal	soybean	sunflower
3	379	136	181	303	248	392
4	260	227	141	315	327	339
5	404	217	260	380	329	341
6	318	168	203	153	250	226
7	352	108	148	263	193	320
8	359	124	169	242	271	295
9	216	143	213	206	316	334
10	222	140	257	344	267	322
11	283	NA	244	258	199	297
12	332	NA	271	NA	171	318
13	NA	NA	NA	NA	158	NA
14	NA	NA	NA	NA	248	NA

t -test p-values compare to each other

	casein	horsebean	linseed	meatmeal	soybean	sunflower
casein		9e-05	8e-04	0.09916	0.02294	0.79590
horsebean			0.02223	2e-04	4e-06	9e-06
linseed				0.01167	0.16020	6e-05
meatmeal					0.46506	0.00584
soybean						0.00252

Please start here

10 Reference

-MySQL reference

-RMySQL