Chapter 2: The Use of R Programming

Prepared by Jixiang Wu

Associate Professor of Quantitative Genetics/Biostatistics

Plant Science Department, South Dakota State University

Brookings, SD 57007, USA

Email: Jixiang.wu@sdstate.edu

Phone: 668-5947

Introduction

This is a brief user instruction for using R platform and R packages. I will show you some basic knowledge of R programming that may be needed by this course.

Installation of R Platform

If you haven't install an R platform in your PC, you can go to any Go to any CRAN site (see http://cran.r-project.org/mirrors.html for a list), navigate to the bin/windows/base directory and collect the file(s) you need. The current release is distributed as an installer 'R-3.1.1-win.exe' of about 54Mb.

To install 'R-3.1.1-win.exe'. Just double-click on the icon and follow the instructions. If you have an account with Administrator privileges you will be able to install R in the Program Files area and to set all the optional registry entries; otherwise you will only be able to install R in your own file area. You may need to confirm that you want to proceed with installing a program from an unknown' orunidentified' publisher.

After installation you should choose a working directory for R. You will have a shortcut to Rgui.exe on your desktop and/or somewhere on the Start menu file tree, and perhaps also in the Quick Launch part of the taskbar (Vista and earlier). Right-click each shortcut, select Properties... and change the `Start in' field to your working directory.

On some systems you will have two shortcuts, one for 32-bit with a label starting R i386 and one for 64-bit starting R x64.

How to Run R

Click R icon if it is on your desktop screen. If not, go to All Programs and find it and click it. Then you should see the R Console for you to use.

You can type a code in the window R Console after > then hit return key

The following two examples show how to assign 10 to x and 2 to y.

```
x=10
x

## [1] 10

y=2
y

## [1] 2
```

You may also do some math with these two numbers. For example:

```
x+y

## [1] 12

x/y

## [1] 5

x*y

## [1] 20

x^y

## [1] 100
```

However, any R codes/scripts could disaappear if you use R console to run your programs. Therefore we often open a new script window to develop scripts and keep them into a file via save as or save function. You may also open one or more currently available R script files to run data analyses or modify some functions.

Installation of Specific R Packages

Many commonly used R functions are built-in with the R platform. For example, functions like lm for linear model analysis and cor for correlation analysis are built-in. However, sometimes some functions are only suitable for some specific purposes. For example, some

R functions used for spatial statistical analysis are only available in some specific R packages. Before you can use these functions, you will need install and load these packages.

For example, you can use the function install.packages() to install the package agridat, which collects many published experimental data sets. After you use the R script install.packages("agridat"), you can use require to load this R package. For example

```
#install.packages("agridat")
require(agridat)
```

```
## Loading required package: agridat
```

```
## Warning: package 'agridat' was built under R version 3.0.2
```

```
## Loading required package: grid
## Loading required package: lattice
## Loading required package: reshape2
```

```
## Warning: package 'reshape2' was built under R version 3.0.2
```

Some examples

Now we will look at some examples to demonstrate the use of R codes for data analysis. Before we can run some data analyses, we need load some R packages and obtain some data sets. Now we want to use a data set called cot in the minque R package, which is a cotton data set. Suppose you already installed this package "install.packages("minque")", you can load this R package.

```
require (minque)
```

```
## Loading required package: minque
```

```
## Warning: package 'minque' was built under R version 3.0.3
```

```
## Loading required package: klaR
```

```
## Warning: package 'klaR' was built under R version 3.0.2
```

```
## Loading required package: MASS
##
## Attaching package: 'MASS'
##
## The following object is masked _by_ '.GlobalEnv':
##
##     ginv
##
## Loading required package: Matrix
```

```
## Warning: package 'Matrix' was built under R version 3.0.2
```

After you load the minque package, you can use the following R code to obtain the data cot.

```
data(cot)
dim(cot)
```

```
## [1] 288   7
```

```
names(cot)
```

```
## [1] "LOC"  "Geno" "REP"  "BN"   "BS"   "LP"   "LY"
```

This cot data set has 288 obervations for each of four cotton traits. It will use too much space to print out the whole data set. However, you may look at the first six observarions. By doing so, you can use the head function.

```
head(cot)
```

LOC	Geno	REP	BN	BS	LP	LY
1	1	1	330.2	4.31	31.25	447
1	2	1	295.2	4.75	30.11	459
1	3	1	511.2	4.83	33.28	836
1	4	1	363.9	4.16	30.82	469
1	5	1	224.3	4.98	34.27	391
1	6	1	335.9	4.82	31.97	579

The R script head(cot) is equivalent to cot[1:6,].

```
cot[1:6,]
```

LOC	Geno	REP	BN	BS	LP	LY
1	1	1	330.2	4.31	31.25	447
1	2	1	295.2	4.75	30.11	459
1	3	1	511.2	4.83	33.28	836
1	4	1	363.9	4.16	30.82	469
1	5	1	224.3	4.98	34.27	391
1	6	1	335.9	4.82	31.97	579

Now we try to extract the trait BN (cotton boll number) for a few data analyses.

```
y=cot$BN
y
```

```
##   [1] 330.2 295.2 511.2 363.9 224.3 335.9 444.0 377.9 599.8 229.4 486.4
##  [12] 452.7 481.7 338.0 281.3 467.8 309.8 417.8 385.5 327.7 632.5 460.7
##  [23] 429.8 487.8 552.5 522.2 465.6 489.5 447.0 444.1 519.9 463.3 648.8
##  [34] 539.0 416.1 352.4 232.8  34.6 378.1 354.8 164.0 466.6 495.9 441.9
##  [45] 373.0 373.2 561.9 397.2 430.9 369.1 331.3 474.9 307.8 387.2 406.3
##  [56] 464.1 493.3 404.1 486.4 561.8 507.6 361.9 484.1 477.5 414.5 296.5
##  [67] 437.8 427.9 396.1 275.3 478.7 446.3 248.5 367.7 564.1 320.3 277.1
##  [78] 283.2 398.2 377.5 456.2 339.3 237.4 170.7 275.8 240.0 411.4 433.7
```

```
##  [89] 251.5 380.1 409.7 363.3 492.1 386.8 648.4 495.3 479.5 514.7 519.8
## [100] 439.2 420.1 535.4 539.4 673.7 468.7 525.6 479.0 195.7 438.8 407.7
## [111] 299.8 336.2 304.6 351.9 421.7 483.2 435.7 292.4 530.7 555.9 548.6
## [122] 441.2 407.4 375.9 423.6 329.8 352.3 471.0 484.4 320.4 578.1 473.1
## [133] 540.7 318.7 585.5 607.8 406.9 435.3 484.7 526.7 485.4 201.7 311.3
## [144] 544.1 284.7 306.5 464.6 257.3 349.8 484.9 554.7 460.8 533.8 461.7
## [155] 523.7 539.0 548.1 412.1 317.6 438.9 388.2 474.5 476.8 444.5 588.9
## [166] 620.9 654.4 576.7 481.1 534.1 505.8 404.4 569.0 717.0 532.9 612.6
## [177] 626.3 608.2 589.9 607.7 428.9 273.3 461.6 231.8 341.7 476.4 470.1
## [188] 367.5 457.0 348.1 668.0 528.1 487.0 360.8 311.3 670.5 287.9 486.9
## [199] 377.9 459.0 590.5 596.6 763.0 483.3 575.5 615.0 621.1 572.4 577.8
## [210] 562.4 599.6 539.7 366.3 569.0 526.0 553.8 254.2 265.0 458.8 316.0
## [221] 223.8 226.4 534.5 470.1 225.7 302.1 423.8 468.7 431.5 389.6 319.1
## [232] 531.7 286.5 473.3 404.7 415.5 592.8 610.1 514.6 605.3 571.7 497.3
## [243] 648.5 474.1 532.8 525.9 516.0 614.8 486.1 566.9 479.3 494.0 424.2
## [254] 353.9 451.1 208.7 248.9 405.5 403.9 411.6 419.8 351.8 484.3 430.3
## [265] 409.1 243.2 322.7 537.3 323.1 378.3 384.7 440.1 510.6 701.0 513.4
## [276] 458.6 463.0 528.5 498.5 567.3 416.0 425.9 502.4 481.7 379.7 535.9
## [287] 466.7 397.5
```

Population mean, variance, and standard deviation

Suppose this y represents a whole population for boll number. We can obtain the population mean.

```
mu=mean(y)
mu
```

```
## [1] 441.8
```

The population mean is 441.8.

Now we can obtain the population variance and standard deviation.

```
v=var(y)
v
```

```
## [1] 13146
```

```
sd=sqrt(v)
sd
```

```
## [1] 114.7
```

The poulation variance is 13146 and standard deviation is 114.7 for boll number.

Sample mean, variance, and standard error

Now we can try to get a sample from this population with sample size of 50 and then calculate the sample mean, sample variance, and standard error.

```
r=50
y1=sample(y,r)
y1
```

```
##  [1] 366.3 277.1 428.9  34.6 373.0 361.9 533.8 575.5 423.8 483.3 414.5
## [12] 507.6 564.1 327.7 437.8 648.5 226.4 423.6 170.7 479.3 478.7 510.6
## [23] 439.2 470.1 648.4 412.1 544.1 763.0 484.1 461.6 567.3 607.7 435.7
## [34] 513.4 311.3 537.3 444.0 431.5 338.0 605.3 435.3 484.7 284.7 530.7
## [45] 351.9 502.4 304.6 229.4 444.5 378.3
```

```
mean(y1)
```

```
## [1] 440.2
```

```
var(y1)
```

```
## [1] 16905
```

```
sqrt(var(y1))
```

```
## [1] 130
```

The sample you obtain will be different from time to time. Thus, sample mean, sample variance, and standard error will be different from time to time as well. For example, if we repeat a second time, we will have slightly different results.

```
r=50
y1=sample(y,r)
y1
```

```
##  [1] 231.8 353.9 484.4 476.4 275.3 361.9 430.9 572.4 608.2 277.1 226.4
## [12] 525.6 481.7 311.3 387.2 348.1 306.5 484.1 566.9 519.8 339.3 561.9
## [23] 286.5 589.9 701.0 474.9 416.0 592.8 485.4 588.9 409.1 605.3 502.4
## [34] 473.3 335.9 316.0 201.7 404.4 539.0 406.3 492.1 461.7 514.6 648.8
## [45] 458.8 296.5 329.8 451.1 530.7 484.7
```

```
mean(y1)
```

```
## [1] 442.6
```

```
var(y1)
```

```
## [1] 14223
```

```
sqrt(var(y1))
```

```
## [1] 119.3
```

Actually, we can repeat this process as many as we like. For doing so, we can use a loop to repeat this process.

```
rep=500
m=numeric(rep)
v=numeric(rep)
se=numeric(rep)
```

```
r=50
for(i in 1:rep){
  y1=sample(y,r)
  m[i]=mean(y1)
  v[i]=var(y1)
  se[i]=sqrt(var(y1))
}
data.frame(m,v,se)[1:20,]
```

m	v	se
437.7	14082	118.67
460.4	14338	119.74
427.6	12977	113.92
434.4	15180	123.21
429.0	14177	119.07
442.2	14051	118.54
440.4	11936	109.25
421.5	8272	90.95
445.0	13707	117.07
427.6	12658	112.51
436.1	13290	115.28
448.6	11611	107.75
433.9	15052	122.69
451.1	16610	128.88
423.9	13727	117.16
460.0	14672	121.13
442.4	13893	117.87
432.5	12656	112.50
453.0	10780	103.82
444.2	10970	104.74

Now we can look at the mean, variance, and standard error for these 50 sample means.

```
mean(m)
```

```
## [1] 442
```

```
var(m)
```

```
## [1] 218.4
```

```
sqrt(var(m))
```

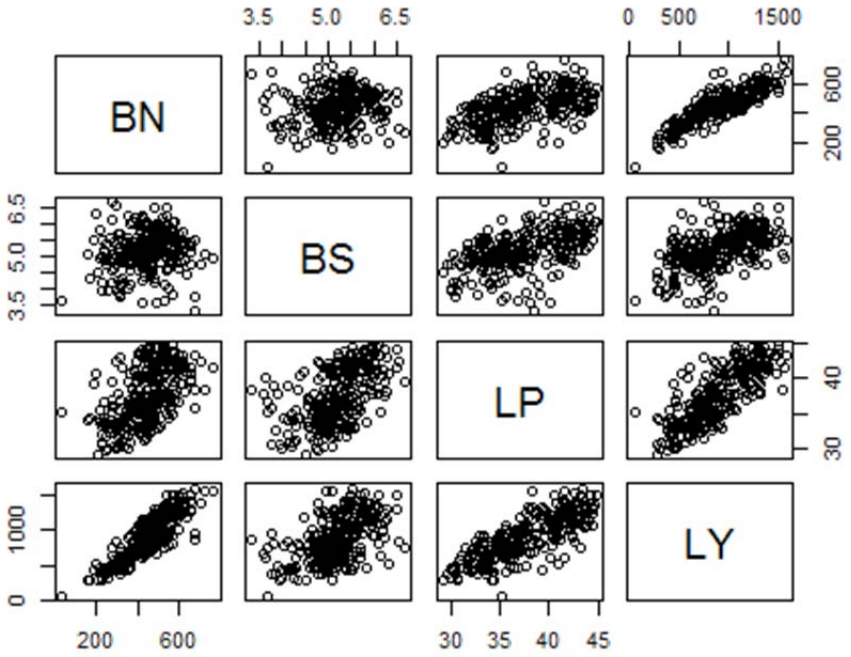
```
## [1] 14.78
```

The same procedure can be used for the sample variances and standard errors.

```
mean(v)

## [1] 13092

var(v)

## [1] 5825909

sqrt(var(v))

## [1] 2414

mean(se)

## [1] 113.9

var(se)

## [1] 111.9

sqrt(var(se))

## [1] 10.58
```

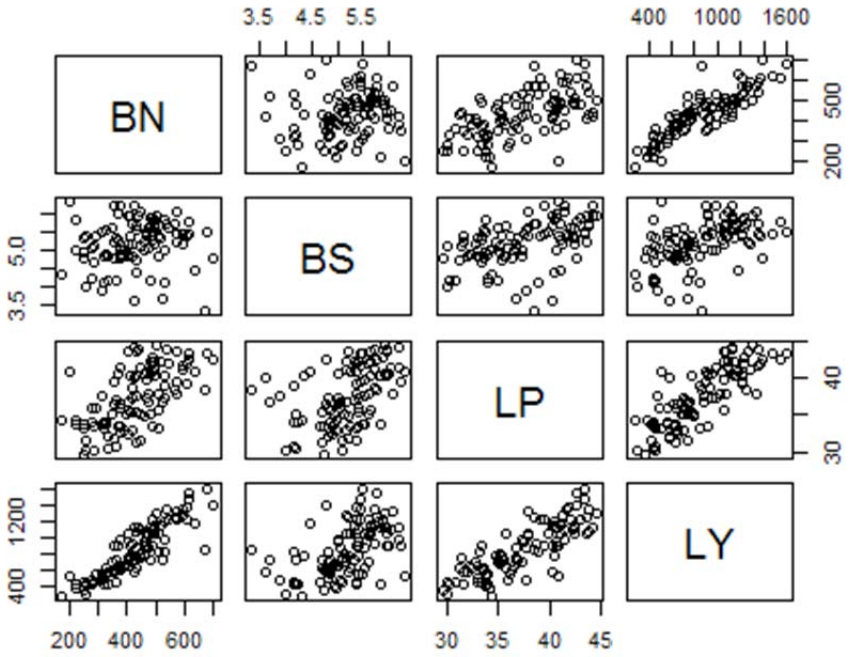
Population Covariance and correlation

```
BN=cot$BN
LY=cot$LY
cov(BN,LY) ## covariance between BN and LY

## [1] 31851

cor(BN,LY) ## correlation between BN and LY

## [1] 0.8768

Y=cot[,-(1:3)]
cor(Y)

##        BN     BS     LP     LY
## BN 1.0000 0.1502 0.5560 0.8768
## BS 0.1502 1.0000 0.5358 0.5420
## LP 0.5560 0.5358 1.0000 0.8009
## LY 0.8768 0.5420 0.8009 1.0000

pairs(Y)
```



Sample covariance and correlation

```
r=100
id=sample(length(BN),r)
BN1=BN[id]
LY1=LY[id]
cov(BN1,LY1) ## covariance between BN and LY

## [1] 30954

cor(BN1,LY1) ## correlation between BN and LY

## [1] 0.8796

Y1=Y[id,]
cor(Y1)

##         BN     BS     LP     LY
## BN 1.0000 0.1165 0.6137 0.8796
## BS 0.1165 1.0000 0.4645 0.5135
## LP 0.6137 0.4645 1.0000 0.8203
## LY 0.8796 0.5135 0.8203 1.0000

pairs(Y1)
```



Simple linear regression

You can use the lm function to run simple linear regression analysis. The R script is very simple. Since two variables LY and BN are global variables now, you can use the following codes to run the analysis.

```
lm1=lm(LY~BN)
lm1

##
## Call:
## lm(formula = LY ~ BN)
##
## Coefficients:
## (Intercept)           BN
##     -188.76         2.42

summary(lm1)

##
## Call:
## lm(formula = LY ~ BN)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -571.7  -95.4  -16.7  101.2  407.6
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -188.7609    35.8694   -5.26  2.8e-07 ***
## BN             2.4230     0.0786   30.83  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 153 on 286 degrees of freedom
## Multiple R-squared:  0.769,  Adjusted R-squared:  0.768
## F-statistic:  951 on 1 and 286 DF,  p-value: <2e-16
```

These two variables are also contained in the datafile cot. You may also use the following code to run the simple regression analysis, which will generate the exactly same results as above.

```
lm2=lm(LY~BN,data=cot)
lm2

##
## Call:
## lm(formula = LY ~ BN, data = cot)
##
## Coefficients:
## (Intercept)           BN
##     -188.76         2.42

summary(lm2)

##
## Call:
## lm(formula = LY ~ BN, data = cot)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -571.7  -95.4  -16.7  101.2  407.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -188.7609    35.8694   -5.26  2.8e-07 ***
## BN             2.4230     0.0786   30.83  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 153 on 286 degrees of freedom
## Multiple R-squared:  0.769,  Adjusted R-squared:  0.768
## F-statistic:  951 on 1 and 286 DF,  p-value: <2e-16
```

The above analyses suggest that the same data analyses can be conducted in different ways, indicating the great flexibility of the use of R.

Simple t-test based on sample

If we consider all boll number (BN) observations as a population that we are interested in, now we can do some t-test based on sampling. The population mean for this trait is 441.8. Now we want to sample 50 individuals for such a test

```
require(coursedata)
```

```
## Loading required package: coursedata
```

```
data(cot)
y=cot$BN
mu=mean(y)
y1=sample(y,50)
a=t.test(y1,mu=mu)
a$p.value
```

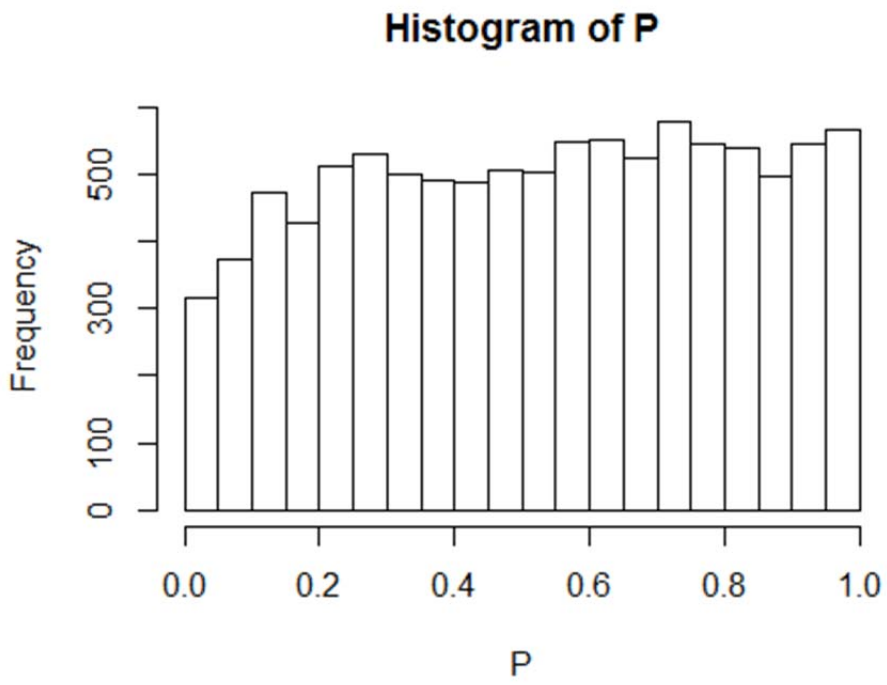
```
## [1] 0.9851
```

You might see that the above p value from this test was greater than 0.05, indicating we can not reject the null hypothesis, H0:mu=441.8. You can repeatedly sample 10000 times and check these p values. By doing so, you can follow the following R codes.

```
rep=10000
require(coursedata)
data(cot)
y=cot$BN
mu=mean(y)
P=numeric(rep)
for(i in 1:rep){
  y1=sample(y,50)
  a=t.test(y1,mu=mu)
  P[i]=a$p.value
}
summary(P)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0002  0.2850  0.5400  0.5270  0.7670  1.0000
```

```
hist(P)
```




```
id=which(P<=0.05)
length(id)
```

```
## [1] 317
```

```
length(id)/rep
```

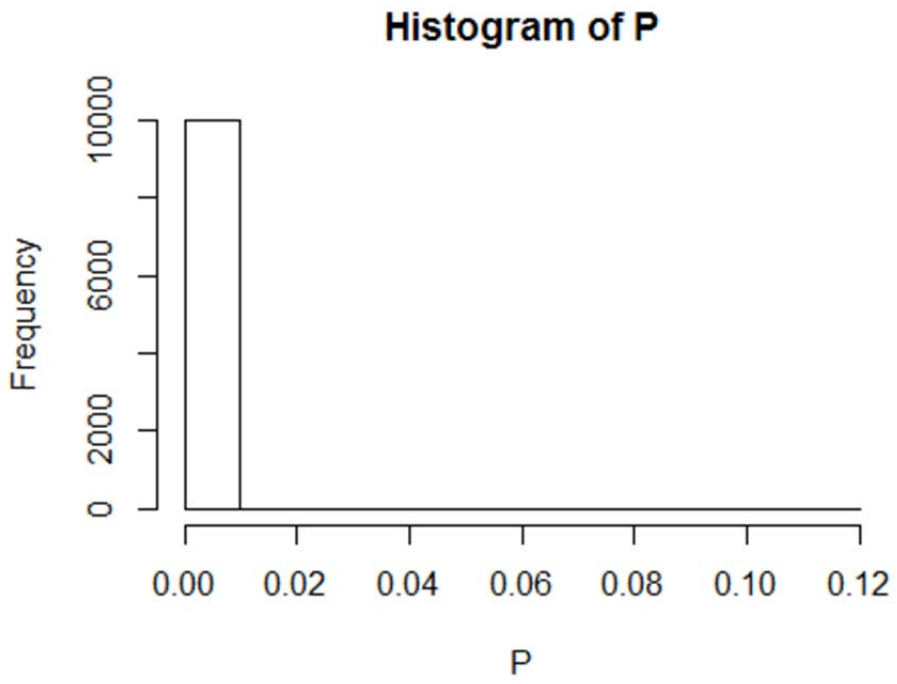
```
## [1] 0.0317
```

If you change the null hypothesis as H0: mu=350, we can run the above t-test again using the following R codes.

```
rep=10000
require(coursedata)
data(cot)
y=cot$BN
mu=350
P=numeric(rep)
for(i in 1:rep){
  y1=sample(y,50)
  a=t.test(y1,mu=mu)
  P[i]=a$p.value
}
summary(P)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.00010 0.00001 0.11500
```

```
hist(P)
```




```
id=which(P<=0.05)
length(id)
```

```
## [1] 9999
```

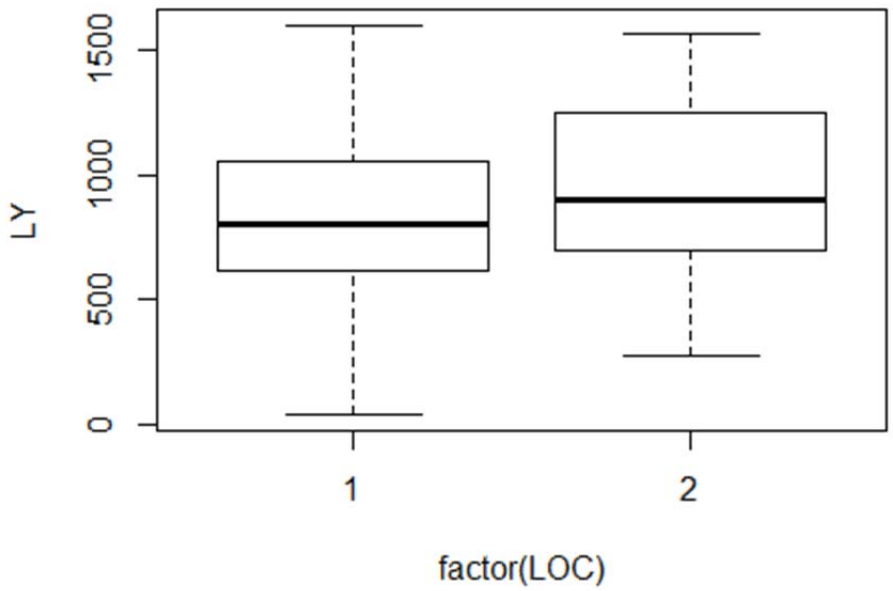
```
length(id)/rep
```

```
## [1] 0.9999
```

Mean comparisons between two environments

Now we will look at a simple t-test by using this data set. The data were collected from two environments. For the purpose of demonstration, we can use a t-test to compare the means between teo environments (locations).

```
plot(LY ~ factor(LOC), data = cot)
```




```
id=which(cot$LOC==1)
y1=LY[id]  ## extract the observations for location 1
y2=LY[-id] ## extract the observations for location 2

t.test(y1,y2)

##
##  Welch Two Sample t-test
##
## data:  y1 and y2
## t = -2.83, df = 279.9, p-value = 0.004997
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -177.02  -31.77
## sample estimates:
## mean of x mean of y
##     829.6     934.0
```

There are many options in the t.test fucntion. You can look at the function by just running the code ??t.test.

The following R script shows another way to run a t test for comapring these two location means for lint yield (LY).

```
## Formula interface
t.test(LY ~ LOC, data = cot)

##
##  Welch Two Sample t-test
##
## data:  LY by LOC
## t = -2.83, df = 279.9, p-value = 0.004997
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -177.02  -31.77
## sample estimates:
## mean in group 1 mean in group 2
##           829.6           934.0
```

The above t.test runs the default option: two-sided t test. You can add more alternatives for the t.tests.

```
t.test(y1,y2,alternative = "less")

##
##  Welch Two Sample t-test
##
## data:  y1 and y2
## t = -2.83, df = 279.9, p-value = 0.002499
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##    -Inf -43.51
## sample estimates:
## mean of x mean of y
##     829.6     934.0
```

You might see the probability changed.

```
t.test(y1,y2,alternative = "greater")

##
##  Welch Two Sample t-test
##
## data:  y1 and y2
## t = -2.83, df = 279.9, p-value = 0.9975
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -165.3    Inf
## sample estimates:
## mean of x mean of y
##     829.6     934.0

t.test(y1,y2,paired = FALSE, var.equal = TRUE,conf.level = 0.99)

##
##  Two Sample t-test
```

```
##
## data:  y1 and y2
## t = -2.83, df = 286, p-value = 0.00499
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  -200.066   -8.726
## sample estimates:
## mean of x mean of y
##     829.6     934.0
```

Barley Heading Date

Now we can look at another example, barley heading date. This data set contains a SNP marker with two alleles and heading date. The data set is available now in the package coursedata. We will use this data set to demonstrate some classical statistical tests. Some R scripts are available in my PPT slides. First we need install this package (local installation) and load the data.

```
require(coursedata)

## Loading required package: coursedata

data(snp)
snp
```

SNP	Head
AA	60
AA	57
AA	58
AA	56
AA	58
AA	59
AA	59
AA	58
BB	62
BB	63
BB	62
BB	59
BB	60
BB	60
BB	60
BB	59
BB	63

BB	63
BB	62
BB	61
BB	60
BB	60
BB	60
BB	59
BB	59
BB	58
BB	57
BB	57
BB	59
BB	64
BB	65
BB	59
BB	64
BB	64
BB	63
BB	64
BB	63
BB	62
BB	58
BB	63
BB	59
BB	65
BB	63
BB	63
BB	60
BB	63
BB	64
BB	63
BB	58
BB	59
BB	65
BB	59
BB	58

BB	62
BB	56
BB	57
BB	64
BB	64
BB	58
BB	56
BB	56
BB	59
BB	60
BB	61
BB	60
BB	60
BB	61
BB	60
BB	65
BB	55
BB	59
BB	59
BB	60
BB	60
BB	59
BB	60
BB	59
BB	60
BB	60
BB	61
BB	61
BB	65
BB	61
BB	65
BB	58
BB	59
BB	59
BB	58
BB	59

BB	60
BB	61
BB	60
BB	65
BB	64
BB	60
BB	60

Simple t test

The following R scripts show how to run a basic t test without a built-in function.

```
attach(snp)
n<-length(Head)
A1<-Head[SNP=="AA"]
A2<-Head[SNP=="BB"]
m1<-mean(A1)
m2<-mean(A2)
n1<-length(A1)
n2<-length(A2)
v1<-var(A1)
v2<-var(A2)
vp<-(v1*(n1-1)+v2*(n2-1))/(n-2)
t<-(m1-m2)/sqrt(vp*(1/n1+1/n2))
pvalue<-(1-pt(abs(t),n-2))  ## two-tailed test

data.frame(n,m1,m2,n1,n2,v1,v2,vp,t,pvalue)
```

n	m1	m2	n1	n2	v1	v2	vp	t	pvalue
96	58.12	60.65	8	88	1.554	6.07	5.734	-2.853	0.0027

The pvalue is 0.005 indicating that this snp marker is highly associated with heading date in this spring barley population.

Built-in t test

We can directly use the built-in function to run t test for this SNP data set. It is very easy to do it as shown here.

```
t.test(A1,A2)

##
##  Welch Two Sample t-test
##
## data:  A1 and A2
## t = -4.918, df = 12.73, p-value = 0.0002992
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  -3.633 -1.412
## sample estimates:
## mean of x mean of y
##     58.12     60.65
```

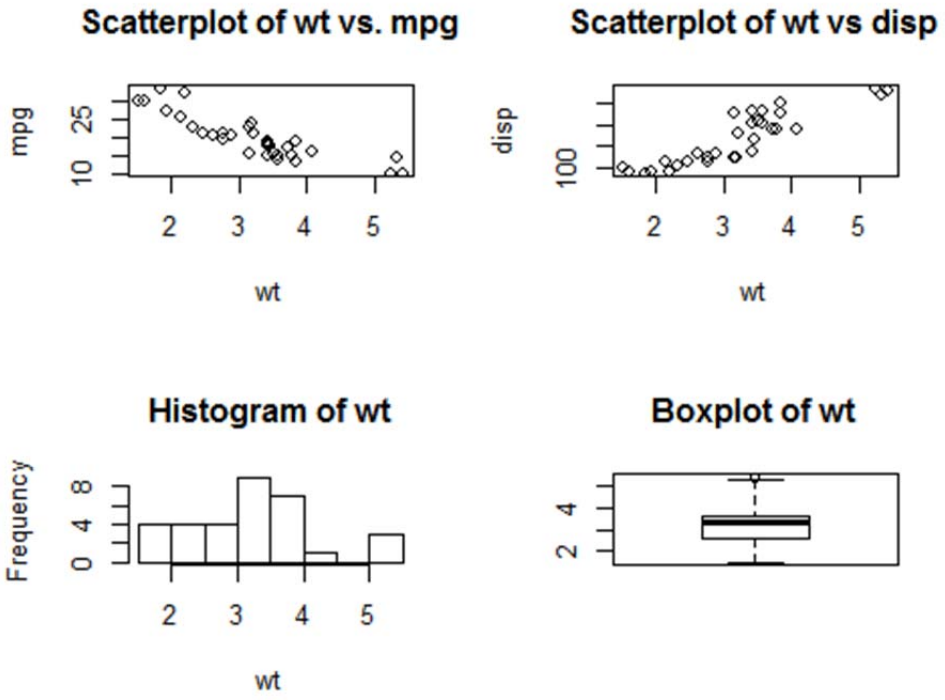
```
t.test(Head~SNP)
```

```
##
##  Welch Two Sample t-test
##
## data:  Head by SNP
## t = -4.918, df = 12.73, p-value = 0.0002992
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.633 -1.412
## sample estimates:
## mean in group AA mean in group BB
##            58.12            60.65
```

Again, all these should generate the same results.

Four figures arranged in 2 rows and 2 columns

```
attach(mtcars)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```



Conclusions

There are much more ways than we can explore in this class. You spend a lot time to run various data analyses or write your own R scripts if you want to be familiar with R programming or even if you want to be an expert in R programming.

The some data sets will be repeatedly used in this class.