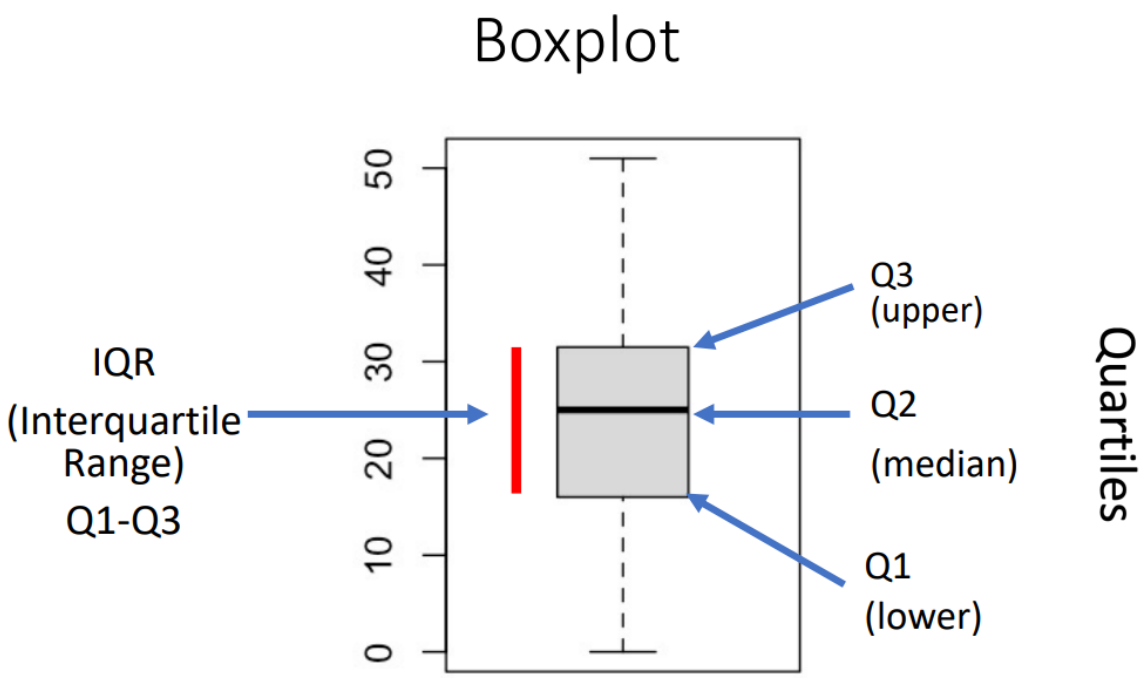
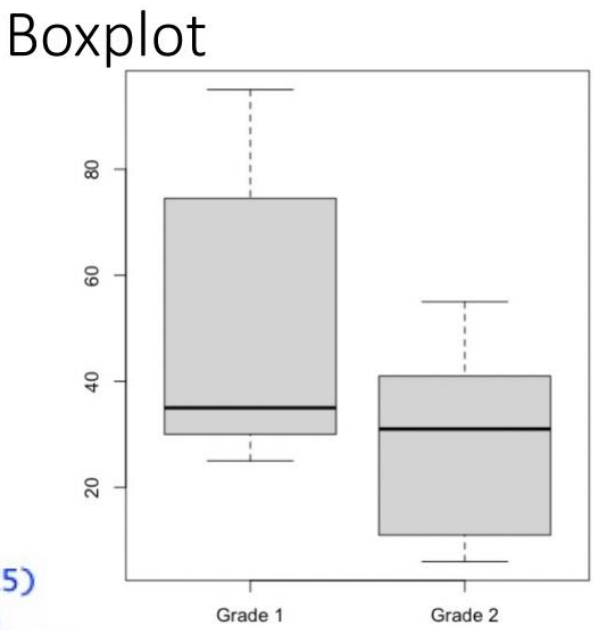
Boxplot Explanation:



Boxplot Example:




```
> grade1 <- c(35,29,31,89,60,95,25)
> grade2 <- c(31,35,47,55,8,14,6)
> boxplot(grade1, grade2, names = c('Grade 1', 'Grade 2'))
>
```

From 'results.txt'

Import the data set

Then attach it:

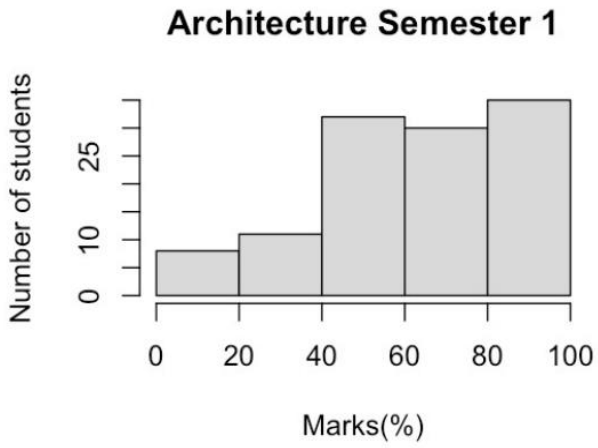
'attach(results)'

Histogram Example:

Histograms

```
> arch1
  [1]  99  NA  97  99  89  91 100  86  89  85  50  96
 [13]  98  96  73  67  80  91  89  77  71  84  95   3
 [25]  95  NA  59  95  80  97  81  77  69  82  85  87
 [37]  88  83  51  76  88  61  83  90  40  92  76  72
 [49]  77  58  63  48  40  40  75  49  54  56  75  64
 [61]  88  82  73  59  74  45  70  74  43  49  45  74
 [73]  46  56  16  21  47  77  27  74  16  14  23  83
 [85]  NA  45  40  48  91  50  77  49  96  21  61  50
 [97]  68  50  69  60  43  43  47  60  40  45  45  31
[109]  49  87  40   8  62  14   7  16  73  56  46
```

```
hist(arch1, breaks=5)
```



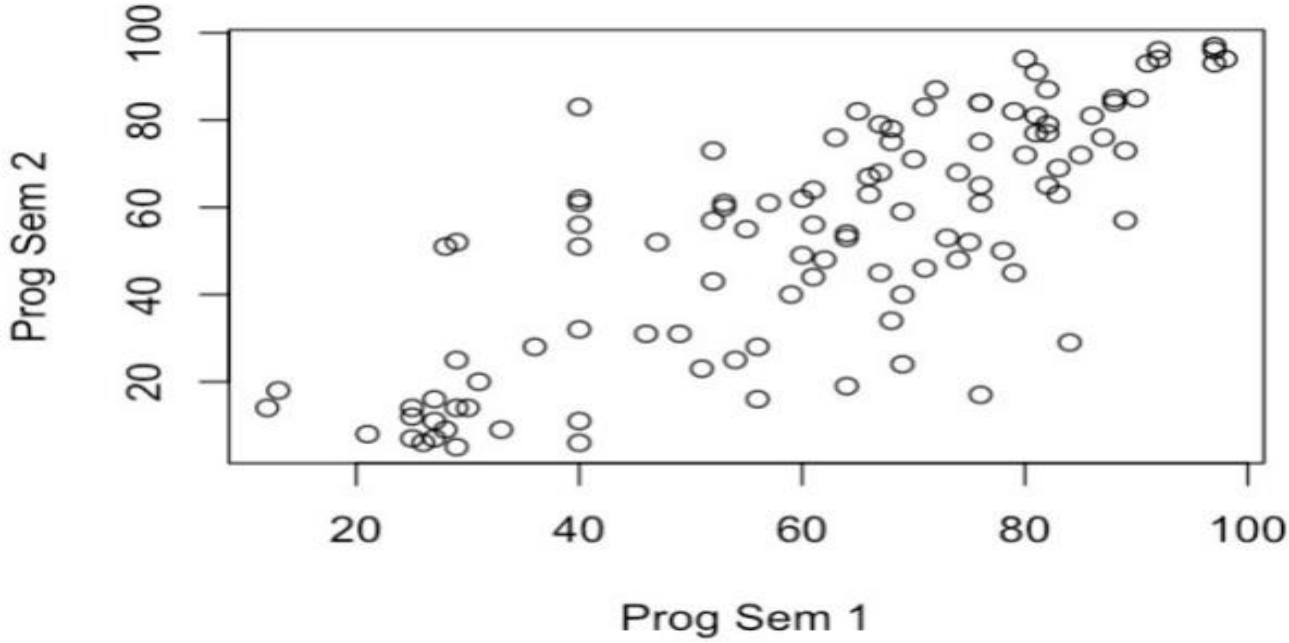
From 'results.txt'

Import the data set

Then attach it:

'attach(results)'

Plot Example:




```
> plot(prog1, prog2,
       xlab = "Prog Sem 1",
       ylab = "Prog Sem 2")
```

Help Options: (To see documentation)

- ***Place item of question into the brackets [e.g help('boxplot')]***

- click the *Help* button on the toolbar.
- `help()`
- `help.start()`
- `demo()`
- `?read.table`
- `help.search ("data.entry")`
- `apropos ("boxplot")`

'rm(list = ls())' to remove all datasets, variables and vectors from the environment

'rm()' to a single specific dataset, vector or variable

From 'results.txt'

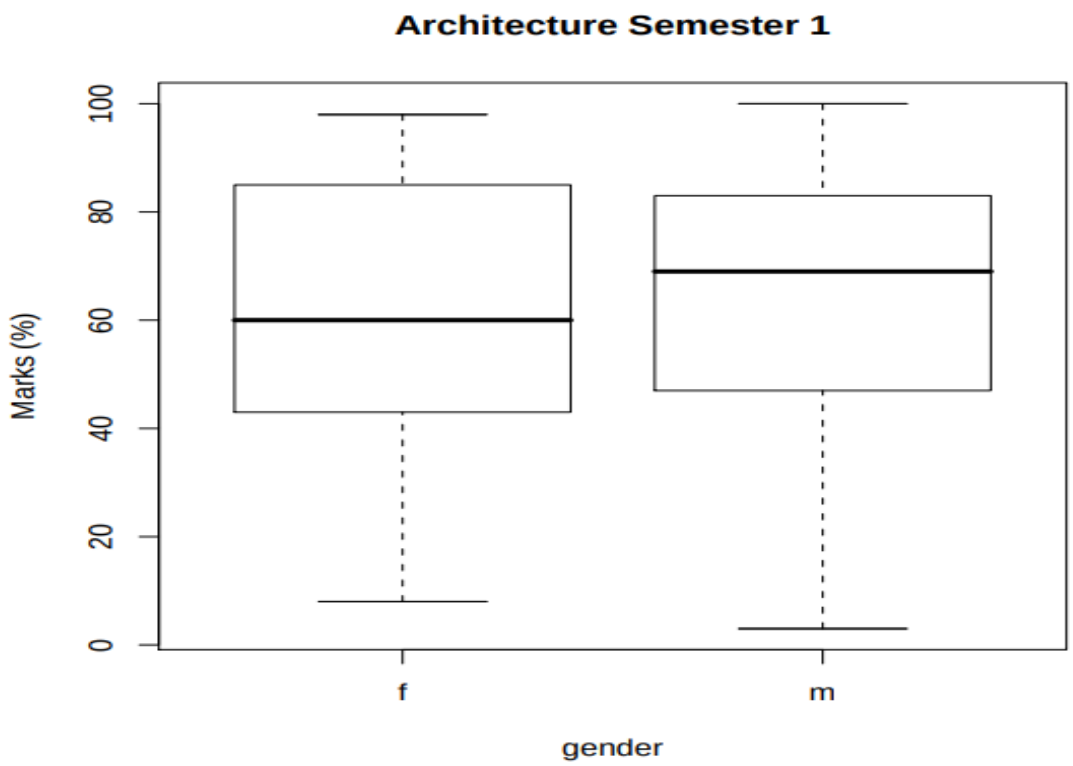
Import the data set

Then attach it:

'attach(results)'

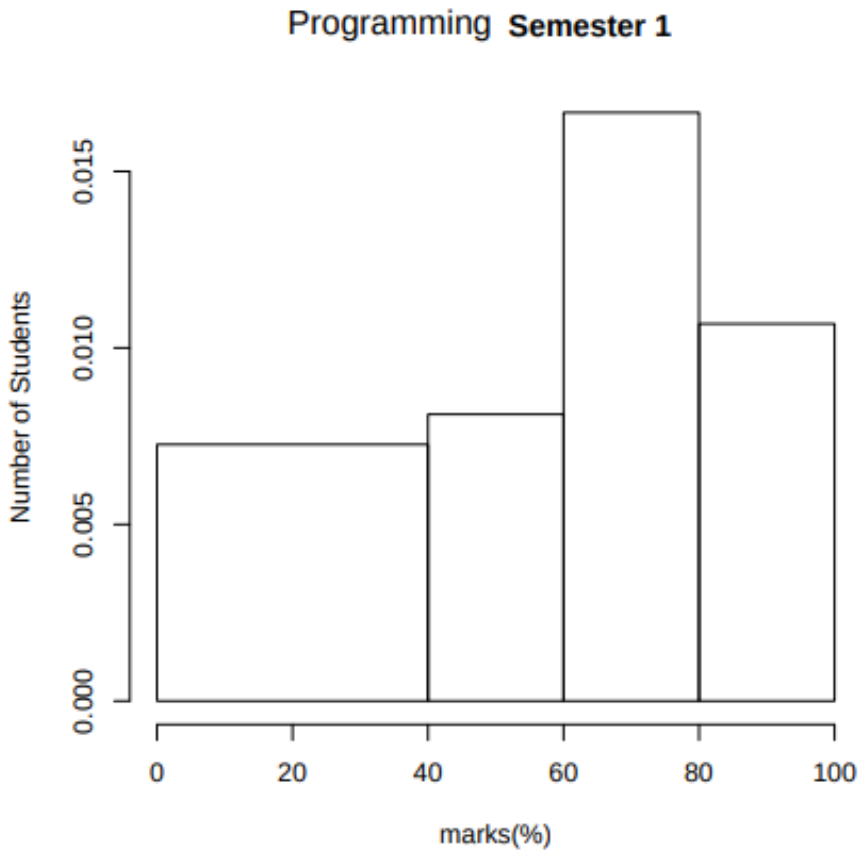
Graphical Displays: Multiple Boxplots

```
boxplot(arch1~gender,
    xlab = "gender",
    ylab = "Marks (%)",
    main = "Architecture Semester 1")
```



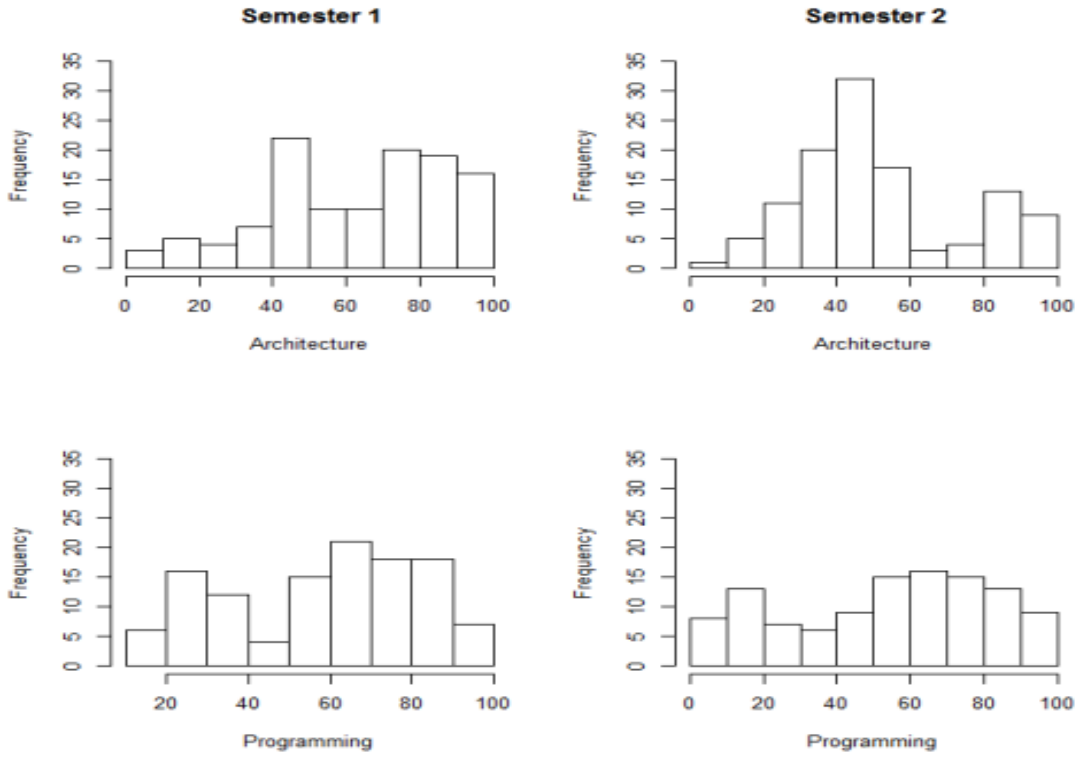
Another Histogram Example:

```
bins <- c(0, 40, 60, 80, 100)
hist(prog1,
     xlab ="Marks(%)",  breaks = bins,
     ylab = "Number of students",
     main = " Programming Semester 1")
```



Multiple Histograms:

```
par (mfrow = c(2,2))
hist(arch1, xlab = "Architecture",
   main = " Semester 1", ylim = c(0, 35))

hist(arch2, xlab = "Architecture",
   main = " Semester 2", ylim = c(0, 35))

hist(prog1, xlab = "Programming",
  main = " ", ylim = c(0, 35))

hist(prog2, xlab = "Programming",
  main = " ", ylim = c(0, 35))
```



Another Plot Example:

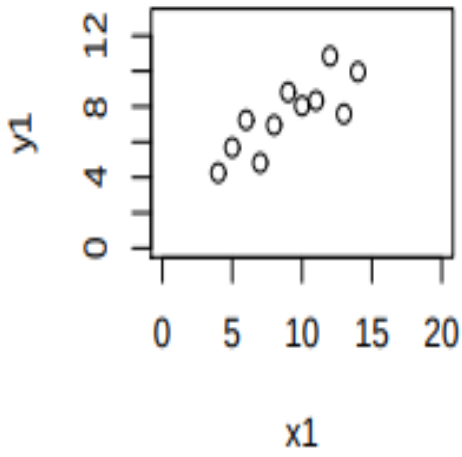
First read the data into separate vectors:

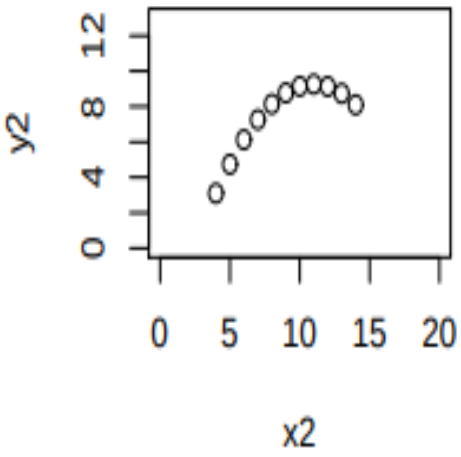
```
x1<-c(10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5)
y1<-c(8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68)

x2 <- c(10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5)
y2 <-c(9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74)

x3<- c(10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5)
y3 <- c(7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73)

x4<- c(8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8)
y4 <- c(6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89)
```

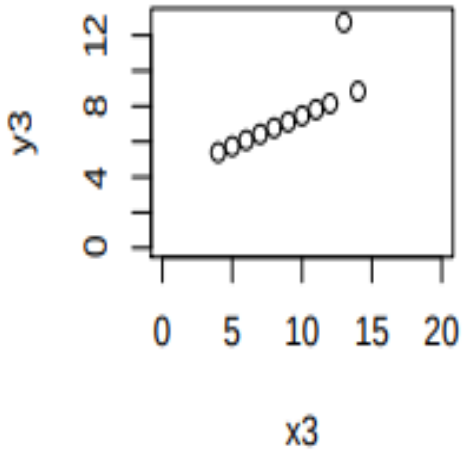
```
par(mfrow = c(2, 2))
plot(x1,y1, xlim=c(0, 20), ylim =c(0, 13))
plot(x2,y2, xlim=c(0, 20), ylim =c(0, 13))
plot(x3,y3, xlim=c(0, 20), ylim =c(0, 13))
plot(x4,y4, xlim=c(0, 20), ylim =c(0, 13))
```

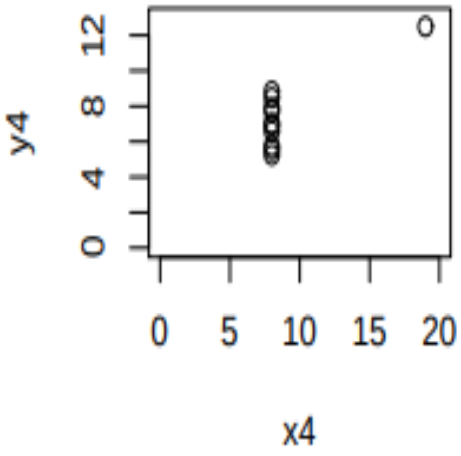
Note that xlim =c() and ylim =c() are for choosing the numbers in the x axis and y axis

Following plots are created:









Useful Functions:

- ***mean()***
- ***sd()***
- ***attach ()***
- ***median()***
- ***summary()***
- ***table()***
- ***sample()***
- ***choose()***
- ***factorial()***
- ***rev()***
- ***identical()***
- ***rep()***
- ***pnorm()***
- ***dnorm()***
- ***qnorm()***

Main="" for the title of the representation model [e.g., plots, histograms]

xlab="" for the x axis of a representation model [e.g., plots, histograms]

ylab="" for the y axis of a representation model [e.g., plots, histograms]

type="" (look at documentation it's hard to explain)

Vector Example:

x <-c(1,2,3)

x

= 1 2 3

Simulation Example:

A Password is 5 characters, one letter and 4 numbers. The password only uses lowercase letters for simplicity. The numbers are from 0 to 9 inclusive.

(e) if a password was selected at random, what is the probability the password would read the same backwards as forwards?

(f) solve (e) by making a simulation in R i.e. write some code to generate lots of passwords using the sample function, and count how many time you get a palindromic password. You will want to get a large number of samples here for to generate an accurate estimate (i.e. 100,000).

Code:

```
> letters = c('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'r
 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z')
> letters_and_numbers = c(letters, 0:9)
>
> cnt <-0
> n_iters <- 100000
>
>
> for (i in 1:n_iters)
+ {
+     password_p1 <- sample(letters, 1, replace=T)
+     password_p2 <- sample(letters_and_numbers, 4, replace=T)
+
+     # combinng part 1 (first letter) and part 2 (the other 4 letters/numbe
+     password <- c(password_p1, password_p2)
+
+     if(identical(password, rev(password)))
+     {
+         cnt <- cnt+1
+     }
+
+ }
>
>
> cnt/n_iters
[1] 0.00071
```

Breakdown:

1. ***Created a vector the contains all the letters of the alphabet called letters***
2. ***Created a vector that contains all the letters of the alphabet and the numbers 0 to 9 inclusive called letters and numbers***
3. ***Created a count variable and assigned it the value 0***
4. ***Created a number of iterations variable and assigned it the value 100k***
5. ***Created the for loop condition, being to iterate 100k times***
6. ***Through each iteration a single password is created, first one letter is chosen at random from the alphabet with the sample function. It's then assigned to a variable***
7. ***Then four letters are choosen at random from the numbers 0 to 9 inclusive and assigned to a variable***
8. ***The single password is then created when both of the two previously created variables are combined to make up the password containing one letter and 4 numbers***

Variance in R:

Summarising Random Variables (Variances)

Worked example:

No. that compiles	0	1	2	3	4	5
Probability	.237	.396	.264	.088	.014	.001

Calculate the expected number of programs that will compile per day:

E(X) = 0×.237+1×.396+2×.264+3×.088+4×.014+5×.001 = 1.25

(or in R):

```
x<- 0:5
prob <- c( .237, .396, .264, .088, .014, .001)
expectation <-sum(x*prob)
expectation
[1] 1.249
```

$$E(X) = \sum_x xp(x)$$

Binomial Distribution in R:

`

R Functions for the Binomial Distribution

- **dbinom** e.g. P(X = 4) with n = 20 and p = .2
  - dbinom(x = 4, size = 20, prob = .2) …. or…..
  - dbinom(4, 20, .2)

- **pbinom** e.g. P(X ≤ 4) with n = 20 and p = .2
  - pbinom(x = 4, size = 20, prob = .2) … or …
  - pbinom(4, 20, .2)

- **qbinom** e.g. Choose k so that P(X ≤ k) ≥ .95
  - qbinom(.95, size = 20, prob = .2) … or …
  - qbinom(.95, 20, .2)

Hypergeometric Distribution in R:

In *R* :

- **Example 1:** Five cards from a deck

```
> x<-0:5
> hyperprob<-dhyper(x, 13, 39, 5)
> round(hyperprob, 4)
[1] 0.2215 0.4114 0.2743 0.0815 0.0107 0.0005
```

- **Example 2:** Three transisters from 6.

```
x<-0:3
dhyper(x, 3, 3, 3)
[1] 0.05 0.45 0.45 0.05
```

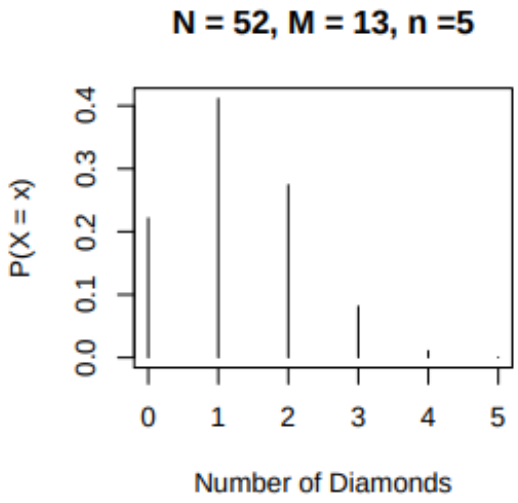
- **Example 3:** 10 IC chips from 20

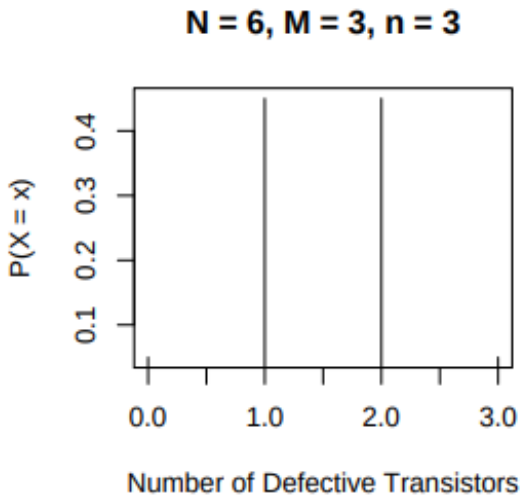
```
x<-0:4
dhyper(x, 4, 16, 10)
[1] 0.04334365 0.24767802 0.41795666 0.24767802 0.04334365
```

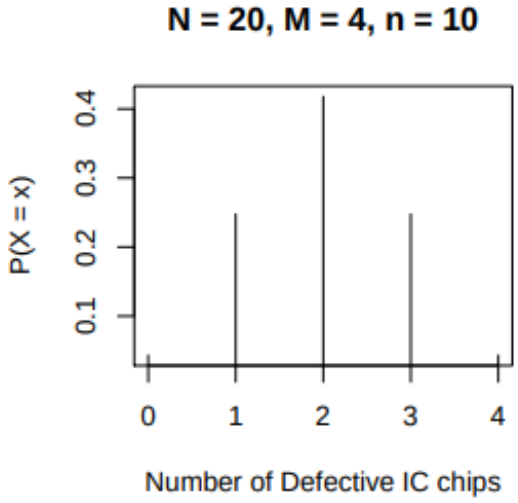
- **Example 4:** 20 printed circuit cards from 100

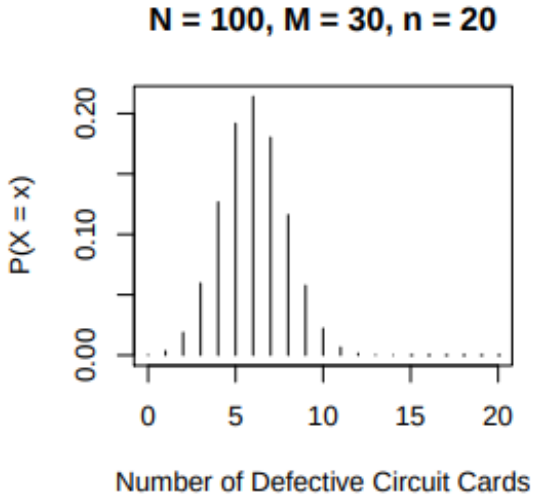
```
x<- 0:10
dhyper(x, 30,70, 20)
```

Results:







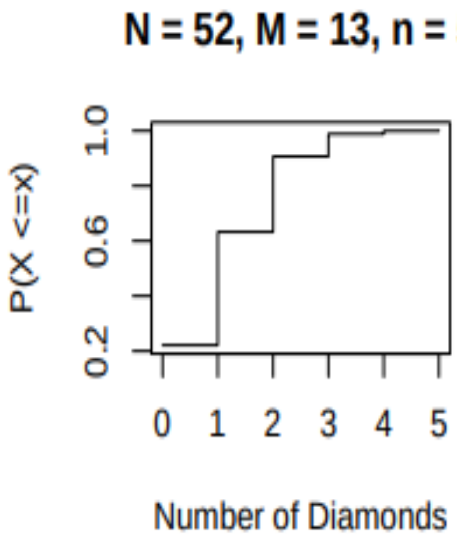


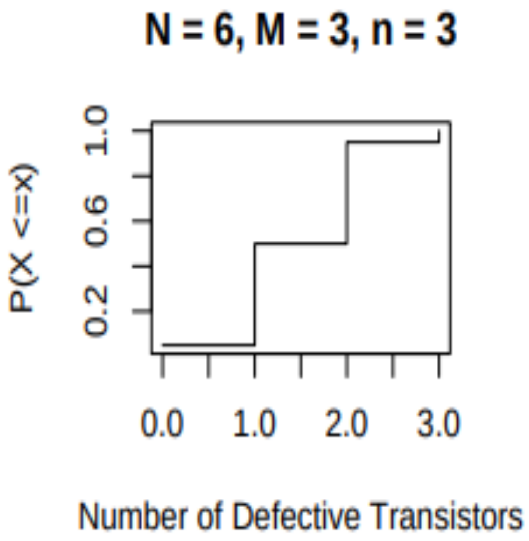
Alternatively:

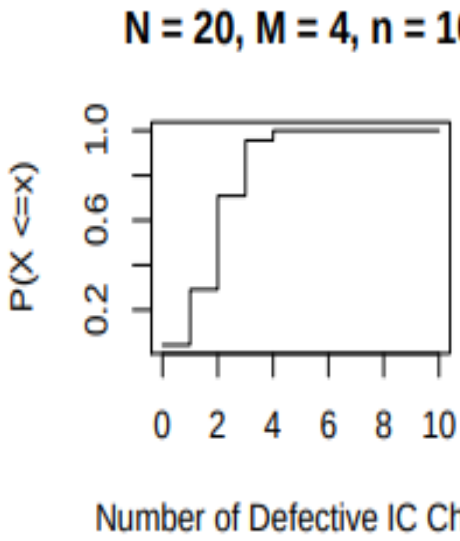
R Code:

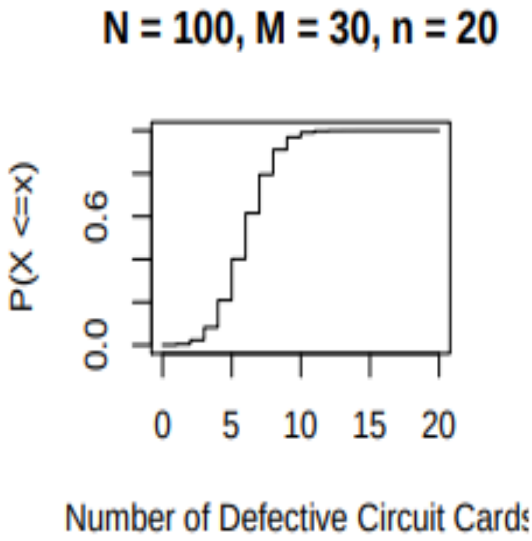
```
 par(mfrow = c(2,2))
x<- 0:5 #Example 1
plot(x, phyper(x, 13, 39, 5),
 xlab = "Number of Diamonds",
 type = "s", ylab = "P(X <=x)",
 main = "N = 52, M = 13, n = 5")

x<- 0:3 #Example 2
plot(x, phyper(x, 3, 3, 3),
 xlab = "Number of Defective Transistors",
 type = "s", ylab = "P(X <=x)",
 main = "N = 6, M = 3, n = 3")

x<- 0:10 #Example 12.3
plot(x, phyper(x, 4, 16, 10),
  xlab = "Number of Defective IC Chips",
   type = "s", ylab = "P(X <=x)",
   main = "N = 20, M = 4, n = 10")

x<- 0:20 #Example 12.4
plot(x, phyper(x, 30, 70, 20),
   xlab = "Number of Defective Circuit Cards",
   type = "s", ylab = "P(X <=x)",
   main = "N = 100, M = 30, n = 20")
```

Alternative Results:


N = 52, M = 13, n = 5
P(X <=x)
0.2
0.6
1.0
0 1 2 3 4 5
Number of Diamonds


N = 6, M = 3, n = 3
P(X <=x)
0.2
0.6
1.0
0.0 1.0 2.0 3.0
Number of Defective Transistors


N = 20, M = 4, n = 10
P(X <=x)
0.2
0.6
1.0
0 2 4 6 8 10
Number of Defective IC Chips


N = 100, M = 30, n = 20
P(X <=x)
0.0
0.6
0 5 10 15 20
Number of Defective Circuit Cards

More Useful Functions:

1. Cheat sheet for lab exam 2

1.1. R functions of use

```
sum(), prod() -- sum and product of items in a collection
choose() -- combinations
factorial()
par() -- plot multiple, e.g par(mfrow=c(1, 2))
plot()
boxplot()
stem()
tapply() -- apply function to each element in a table/collection
c() -- combine collections together ( flattens fields too )
round()
identical() -- compare collections
unique() -- returns unique items of collection
runif() -- e.g runif(1) random floating point in range 0, 1
length()
colnames()
nrow(), ncol() -- number of rows and columns
seq() -- e.g seq(1, 5, .5) -> 1.0 1.5 2.0 ... 5.0
sample() -- take n samples from a collection
table() -- get a summary of frequencies
points() -- overlay plots
read.csv() -- e.g windows weirdness read.csv("C:\\my\\folder\\file.csv", header=T)
attach() -- attach reference to dataframe, no need to reference via dataframe variable
mean() -- get mean of collection ( note add na.rm = T to ignore NA )
sd() -- get standard deviation of collection ( note add na.rm = T to ignore NA )
var() -- get variance of collection ( note add na.rm = T to ignore NA )
abline() -- line of best fit ( refer to tutorial week 10, _near the end_ )
which.max(), which.min() -- get indices of max or min values
max(), min() -- get max or min values
```

1.1.1. Distributions

```
dbinom(), pbinom(), qbinom() -- binomial pdf, cdf
dgeom(), pgeom(), qgeom() -- geometric pdf, cdf
dpois(), ppois(), qpois() -- poisson pdf, cdf
dhyper(), phyper(), qhyper() -- hypergeometric pdf, cdf
```

1.2. Misc

i:j – collection in range i, j

In plots and other functions there is a special syntax for putting one collection against another

```
collection1 ~ collection2
```

1.2.1. Powerful indexing

```
data[field1 == 2] -> subset of data in which field1 values are 2
```