You are not using Markdown properly. Use text when appropriate instead of commenting in your code.

STAT4600 Midterm1

Xinhao Wang

2018-03-05

32.5 / 40

```r
# package
library(lattice)
```

```r
# Question 1

#(A)
#define 586 cards given name 1 to 586
cards <- c(1:586)

#buy 100 packs of cards
# replicate(100,sample(cards,10,replace = FALSE))
#number of different unique cards after bought 100 packs.
# length(unique(as.vector(packs100)))

#if we buy 2000 times of 100 packs cards.
diff.cards <- mean(replicate(2000,length(unique(as.vector(replicate(100,sample(cards,10,replace = FALSE)
paste('there is about',round(diff.cards), 'different cards')
```

It was asked to provide some appropriate output! conf. interval?

```
## [1] "there is about 481 different cards"
```

no rounding!

```r
#(B)
#buy 300 packs of cards
# replicate(300,sample(cards,10,replace = FALSE))
#number of different unique cards after bought 300 packs.
# length(unique(as.vector(replicate(300,sample(cards,10,replace = FALSE)))))

#if we buy 2000 times of 300 packs cards.
diff.cards <- replicate(2000,length(unique(as.vector(replicate(300,sample(cards,10,replace = FALSE)))))
#the probability that the boy will own the complete set of cards after purchasing 300 packs of cards.
p <- length(which(diff.cards == 586))/2000
```

easier: mean(diff.cards==586)

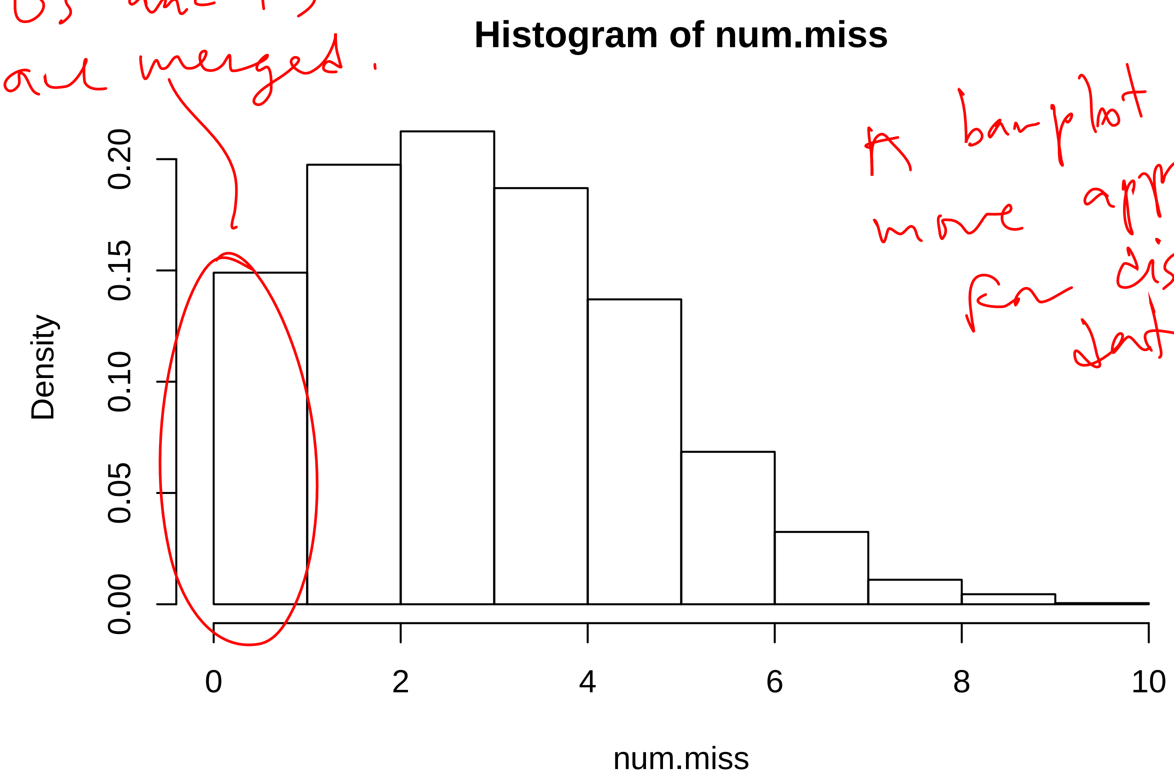
what is p?

```r
#(C)
diff.cards <- replicate(2000,length(unique(as.vector(replicate(300,sample(cards,10,replace = FALSE)))))
# the number of missing cards after purchasing 300 packs of cards.
num.miss <- 586 - diff.cards

hist(num.miss,freq=F)
```

had to modify your code to find out. that shouldn't have to be the case.

Q1 A-B-C 9/10

0's and 1's are merged.



A barplot is more appropriate for discrete data.

```
#(D)
expected.cost <- function(n.packs, n.simul){
  packs.cost <- n.packs * 0.5
  diff.cards <- replicate(n.simul,length(unique(as.vector(replicate(n.packs,sample(cards,10,replace = F
  num.miss <- 586 - diff.cards
  dealer.cost <- num.miss * 0.25
  return(mean(packs.cost+dealer.cost))
}
expected.cost(75,1000)
```

```
## [1] 77.844
```

```
expected.cost(100,1000)
```

```
## [1] 76.20175
```

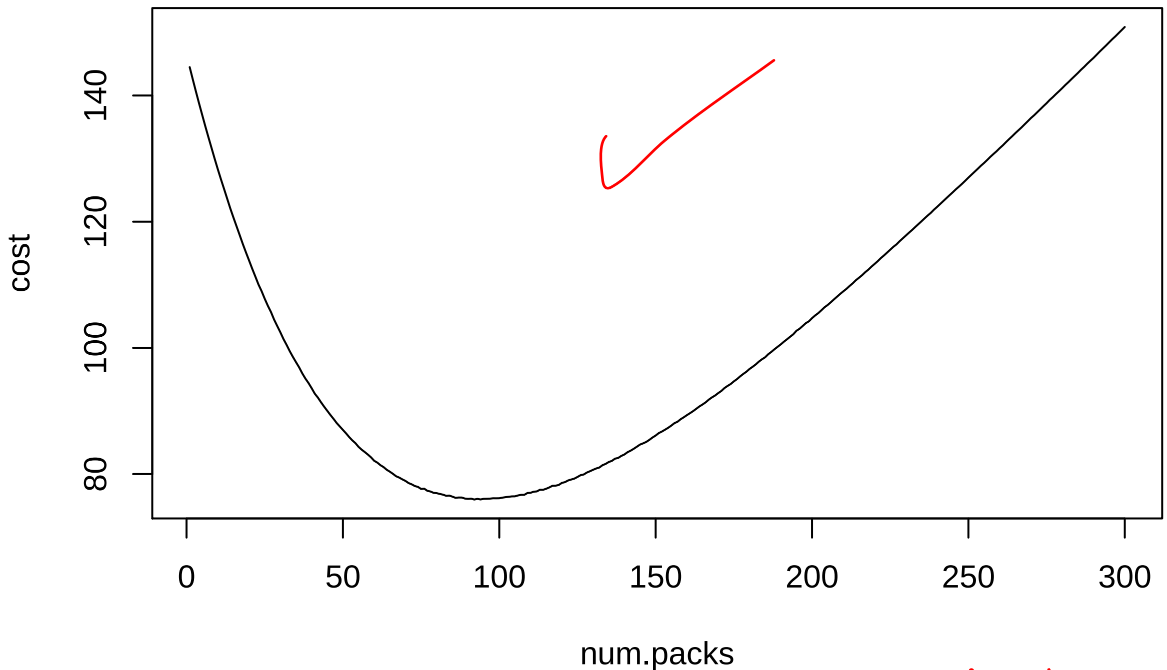
```
expected.cost(125,1000)
```

```
## [1] 79.58375
```

```
#(E)
# define a function cost.graph to make a graph of packs number and the cost.
# the maximum packs is the variable N.packs in function cost.graph.


# cost.graph <- function(N.packs){
#   num.packs <- c(1:N.packs)
#   cost <- c(rep(0,N.packs))
#   for(i in 1:N.packs){
#     cost[i] <- expected.cost(i,1000)
#   }
#   return(plot(num.packs,cost,"l"))
```

Q1
D-E-F
10/10

```
# }
# cost.graph(300)


#################
num.packs <- c(1:300)
cost <- c(rep(0,300))
for(i in 1:300){
  cost[i] <- expected.cost(i,1000)
}
plot(num.packs,cost,"l")
```




```
indx <- which(cost == min(cost))
min.cost <- c(num.packs[indx],cost[indx])

######################
#   conclusion: From the graph, we can conclude when we buy 97 packs of cards the
#               overall cost is minimize with about 76 dollars.
######################
```

Adapt these to the actual simulation output.

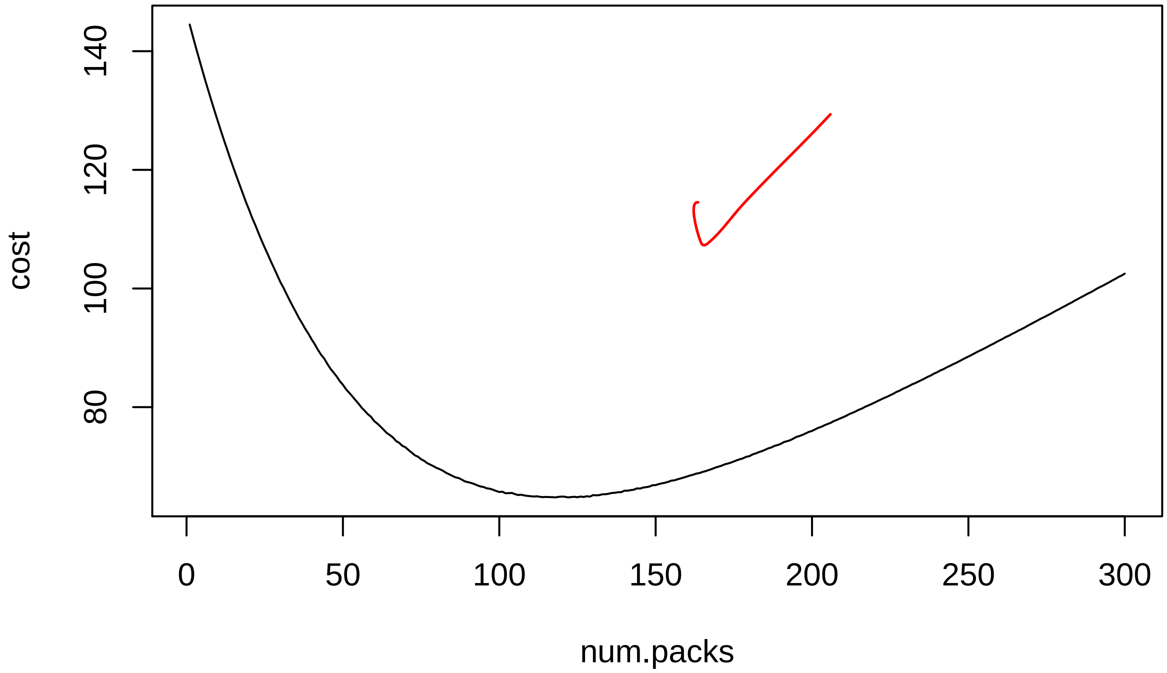
This is one of the important features of Mark down.

```
#(F)
expected.cost1 <- function(n.packs, n.simul){
  packs.cost <- n.packs * 0.5
  diff.cards <- replicate(n.simul,length(unique(as.vector(replicate(n.packs,sample(cards,10,replace = F
  #numer of duplicates cards
  dup.cards <- n.packs*10 - diff.cards
  buy.back <- dup.cards*0.02
  num.miss <- 586 - diff.cards
  dealer.cost <- num.miss * 0.25

  return(mean(packs.cost+dealer.cost-buy.back))
}
```

```
expected.cost1(100,1000)
```

```
## [1] 65.86563
```

```
num.packs <- c(1:300)
cost <- c(rep(0,300))
for(i in 1:300){
  cost[i] <- expected.cost1(i,1000)
}
plot(num.packs,cost,"l")
```

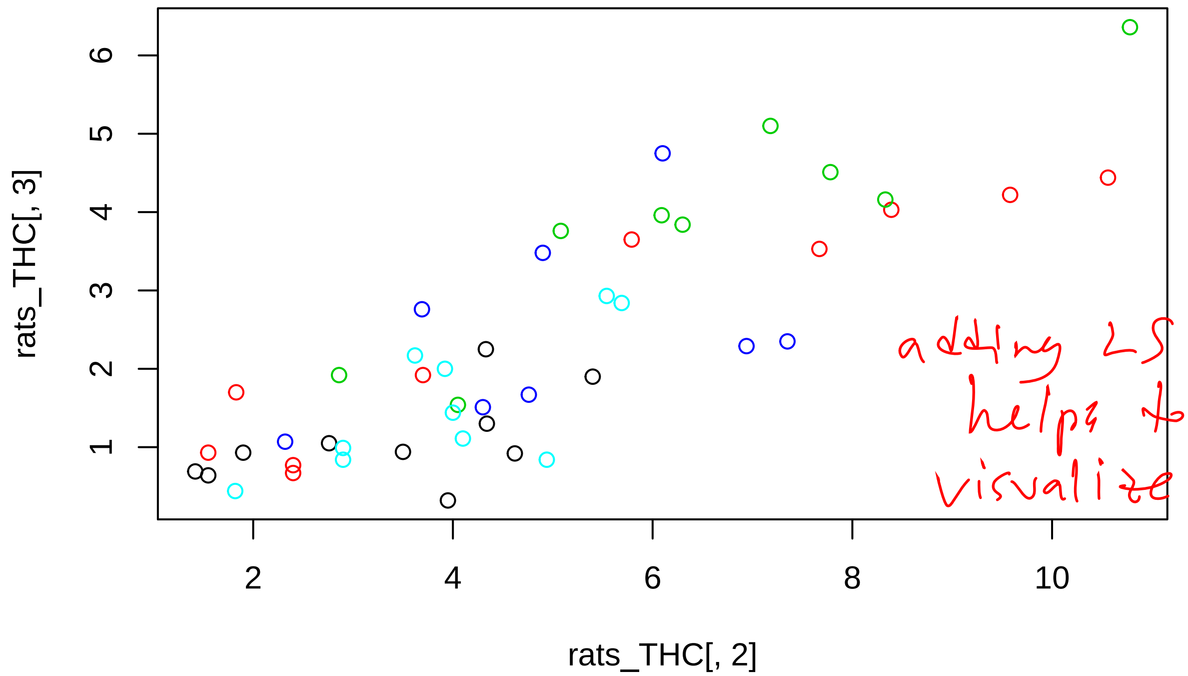


again.

```
indx <- which(cost == min(cost))
min.cost <- c(num.packs[indx],cost[indx])

######################
#   conclusion: From the graph, we can conclude when we buy 118 packs of cards the
#               overall cost is minimize with about 65 dollars.
######################
```

```
#Question2

# part(A)
# The permutation test based on the test statistics D by take samples without replacement from data rat
# define a function 'perm.D' based on the test statistics D. and we take N times replicate to obtain th
# permutation test.If those test statistics D close to 0, then we fial to reject H0.
```

```
# part(B)
rats_THC <- read.csv("rats_THC.csv")
with(rats_THC, plot(rats_THC[,2],rats_THC[,3], col = rats_THC[,1]))
```



adding LS lines helps to visualize.

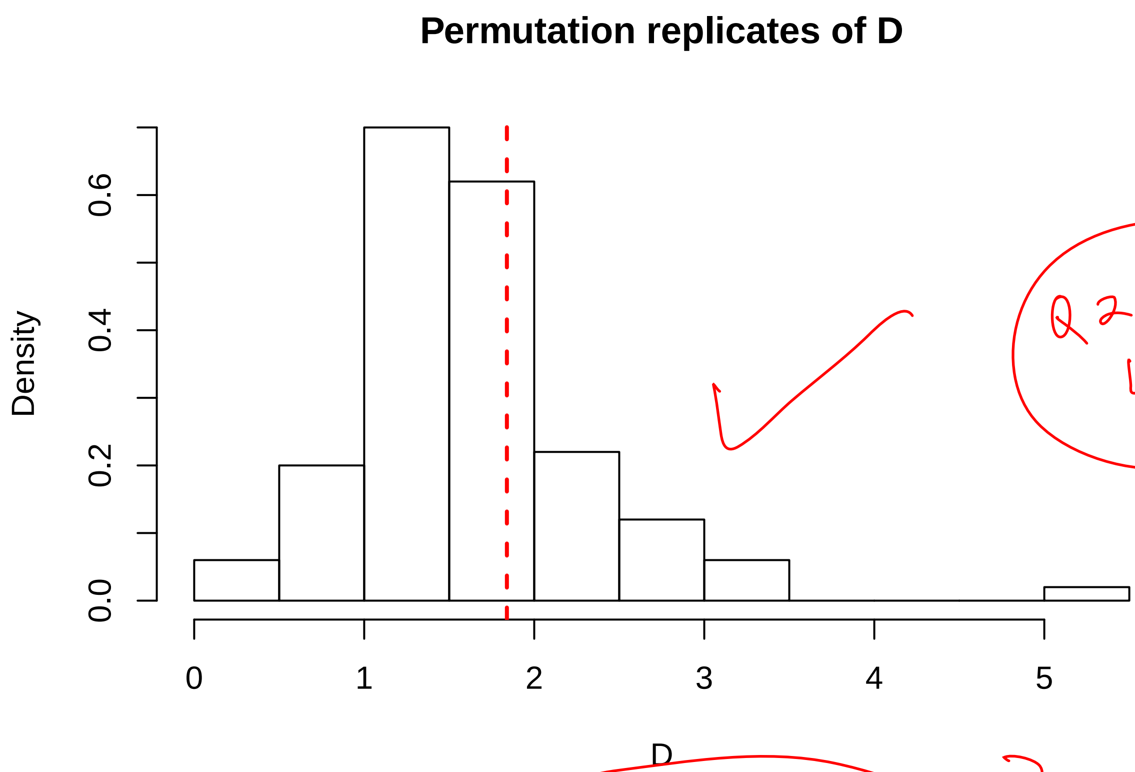
```
group.0ug <- rats_THC[1:10,2:3]
group.0.1ug <- rats_THC[11:20,2:3]
group.0.5ug <- rats_THC[21:29,2:3]
group.1ug <- rats_THC[30:37,2:3]
group.2ug <- rats_THC[38:47,2:3]

beta1 <- lm(group.0ug$post~group.0ug$pre)$coeff
beta2 <- lm(group.0.1ug$post~group.0.1ug$pre)$coeff
beta3 <- lm(group.0.5ug$post~group.0.5ug$pre)$coeff
beta4 <- lm(group.1ug$post~group.1ug$pre)$coeff
beta5 <- lm(group.2ug$post~group.2ug$pre)$coeff

# The number of permutation replicates of D (It is too large)
choose(47,10)*choose(37,10)*choose(27,9)*choose(18,8)*choose(10,10)
```

yes.

```
## [1] 3.69909e+29
```

```
#
#
# Generating one permutation replicate of D
#
perm.D <- function(group.1,group.2,group.3,group.4,group.5){
  n.g1 <- nrow(group.1)
  n.g2 <- nrow(group.2)
  n.g3 <- nrow(group.3)
  n.g4 <- nrow(group.4)
  n.g5 <- nrow(group.5)
  n.tot <- nrow(rbind(group.1,group.2,group.3,group.4,group.5))

  indices <- c(1:n.tot)
  indices.g1 <- sample(indices, size = n.g1, replace = FALSE)
  indices.g2 <- sample(indices[-indices.g1], size = n.g2, replace = FALSE)
  indices.g3 <- sample(indices[-c(indices.g1,indices.g2)], size = n.g3, replace = FALSE)
```

```r
  indices.g4 <- sample(indices[-c(indices.g1,indices.g2,indices.g3)], size = n.g4, replace = FALSE)
  indices.g5 <- indices[-c(indices.g1,indices.g2,indices.g3,indices.g4)]

  selected.g1 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g1,]
  selected.g2 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g2,]
  selected.g3 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g3,]
  selected.g4 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g4,]
  selected.g5 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g5,]

  beta1 <- lm(selected.g1$post~selected.g1$pre)$coeff
  beta2 <- lm(selected.g2$post~selected.g2$pre)$coeff
  beta3 <- lm(selected.g3$post~selected.g3$pre)$coeff
  beta4 <- lm(selected.g4$post~selected.g4$pre)$coeff
  beta5 <- lm(selected.g5$post~selected.g5$pre)$coeff

  D <- max(abs(c((beta1[1]-beta2[1]),(beta1[1]-beta3[1]),(beta1[1]-beta4[1]),(beta1[1]-beta5[1]),
             (beta2[1]-beta3[1]),(beta2[1]-beta4[1]),(beta2[1]-beta5[1]),
             (beta3[1]-beta4[1]),(beta3[1]-beta5[1]),
             (beta4[1]-beta5[1]))))+
           max(abs(c((beta1[2]-beta2[2]),(beta1[2]-beta3[2]),(beta1[2]-beta4[2]),(beta1[2]-beta5[2]),
                   (beta2[2]-beta3[2]),(beta2[2]-beta4[2]),(beta2[2]-beta5[2]),
                   (beta3[2]-beta4[2]),(beta3[2]-beta5[2]),
                   (beta4[2]-beta5[2]))))

  return(D)
}
#
# The replicates
#
replicates.D <- replicate(100,perm.D(group.0ug,group.0.1ug,group.0.5ug,group.1ug,group.2ug))
#
# the approximate permutation distribution of r and ASL
#
hist(replicates.D, xlab='D', freq=FALSE, main="Permutation replicates of D")

D.obs <- max(abs(c((beta1[1]-beta2[1]),(beta1[1]-beta3[1]),(beta1[1]-beta4[1]),(beta1[1]-beta5[1]),
                   (beta2[1]-beta3[1]),(beta2[1]-beta4[1]),(beta2[1]-beta5[1]),
                   (beta3[1]-beta4[1]),(beta3[1]-beta5[1]),
                   (beta4[1]-beta5[1]))))+
  max(abs(c((beta1[2]-beta2[2]),(beta1[2]-beta3[2]),(beta1[2]-beta4[2]),(beta1[2]-beta5[2]),
          (beta2[2]-beta3[2]),(beta2[2]-beta4[2]),(beta2[2]-beta5[2]),
          (beta3[2]-beta4[2]),(beta3[2]-beta5[2]),
          (beta4[2]-beta5[2]))))
abline(v=D.obs, lty=2, lwd=2, col='red')
text(D.obs,20,pos=2,label=expression(paste(beta[1]," = 1.84")))
```



Q2 A-B
4/5

```
#
asl <- mean(replicates.D >= D.obs) + mean(replicates.D <= -D.obs)
#


# conclusion: we reject H0. which means there is i,j such that beta[0,i] not equal to
#              beta[0,j] and/or beta[1,i] not equal to beta[1,j].
############################
```

?

why?

asl is like a p value and here ≈ 0.27

```
# part(C)
# the difference between permutation test and bootstrapping is that permutation test takes the sample
# without replacement, but bootstrapping with replacement.
```

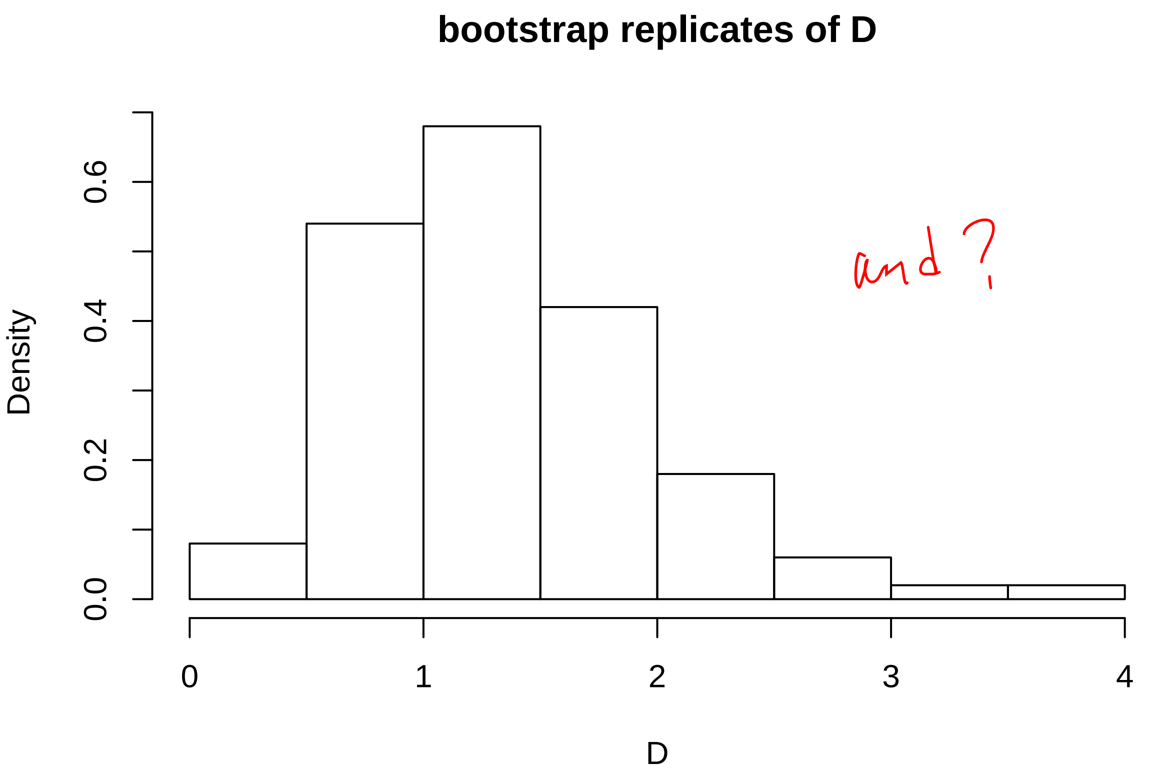
```
# part(D)
bootstrap.D <- function(group.1,group.2,group.3,group.4,group.5){
  n.g1 <- nrow(group.1)
  n.g2 <- nrow(group.2)
  n.g3 <- nrow(group.3)
  n.g4 <- nrow(group.4)
  n.g5 <- nrow(group.5)
  n.tot <- nrow(rbind(group.1,group.2,group.3,group.4,group.5))

  indices.g1 <- sample(1:n.tot, size = n.g1, replace = FALSE)
  indices.g2 <- sample(1:n.tot,n.g2,replace = FALSE)
  indices.g3 <- sample(1:n.tot,n.g3,replace = FALSE)
  indices.g4 <- sample(1:n.tot,n.g4,replace = FALSE)
  indices.g5 <- sample(1:n.tot,n.g5,replace = FALSE)

  selected.g1 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g1,]
  selected.g2 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g2,]
```

bootstrap is supposed to retain the group structure,

So, sample with replacement within each group.

```r
  selected.g3 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g3,]
  selected.g4 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g4,]
  selected.g5 <- rbind(group.1,group.2,group.3,group.4,group.5)[indices.g5,]

  beta1 <- lm(selected.g1$post~selected.g1$pre)$coeff
  beta2 <- lm(selected.g2$post~selected.g2$pre)$coeff
  beta3 <- lm(selected.g3$post~selected.g3$pre)$coeff
  beta4 <- lm(selected.g4$post~selected.g4$pre)$coeff
  beta5 <- lm(selected.g5$post~selected.g5$pre)$coeff

  D <- max(abs(c((beta1[1]-beta2[1]),(beta1[1]-beta3[1]),(beta1[1]-beta4[1]),(beta1[1]-beta5[1]),
               (beta2[1]-beta3[1]),(beta2[1]-beta4[1]),(beta2[1]-beta5[1]),
               (beta3[1]-beta4[1]),(beta3[1]-beta5[1]),
               (beta4[1]-beta5[1]))))+
    max(abs(c((beta1[2]-beta2[2]),(beta1[2]-beta3[2]),(beta1[2]-beta4[2]),(beta1[2]-beta5[2]),
            (beta2[2]-beta3[2]),(beta2[2]-beta4[2]),(beta2[2]-beta5[2]),
            (beta3[2]-beta4[2]),(beta3[2]-beta5[2]),
            (beta4[2]-beta5[2]))))
  return(D)
}

replicatess.D <- replicate(100,bootstrap.D(group.0ug,group.0.1ug,group.0.5ug,group.1ug,group.2ug))
hist(replicatess.D, xlab='D', freq=FALSE, main="bootstrap replicates of D")
```

Q2 GD
0/5 + 2.5 Bonus




```r
library(lattice)
# Question 3

# part(A)
# 'G' is the number of groups.
```

Q3 7/10

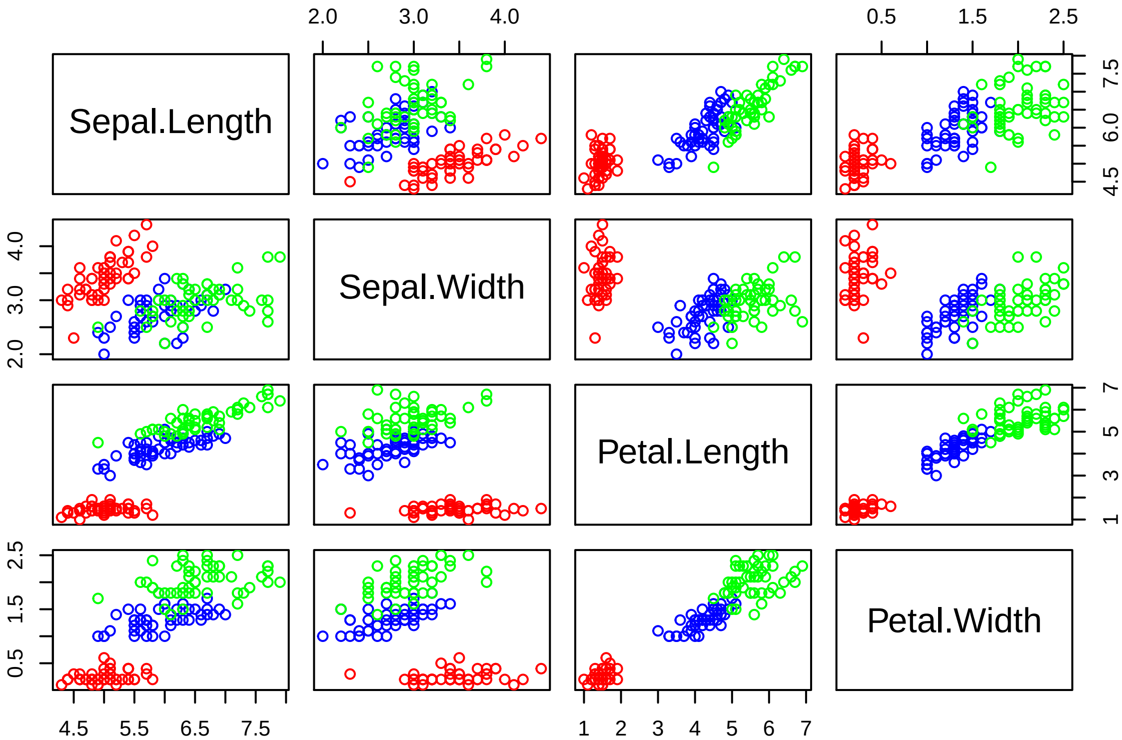
```r
Clustering <- function(matrix.data, G){
  n.obs <- nrow(matrix.data)
  #
  # first step: randomly assigning a group label from 1 to G to each observation;
  #
  random.label <- sample(c(1:G),n.obs,replace = TRUE)
  group.label <- cbind(matrix.data,random.label)
  #
  # second step: iterating
  #
  re.label <- rep(0,n.obs)
  nb.iter <- 0
  #
  # using repeat loop to iterating until the assignment of observations to groups stops changing.
  #
  repeat{
    # calculate group mean (centres)
    centres <- aggregate(group.label[,1:4],by=list(label=group.label[,ncol(group.label)]),FUN=mean)
    for(i in 1:n.obs){
      d <- c(1:G)
      for(j in 1:G){
        d[j] <- apply(matrix.data[i,]-centres[j,2:5],1,function(x) sum(x^2))
      }
      re.label[i] <- which(d==min(d))
    }
    group.label <- cbind(group.label,re.label)
    nb.iter <- nb.iter+1
    #
    # we find some vector of group label here, once we find two close vector are equal, then break.
    #
    if(all(group.label[,ncol(group.label)]==group.label[,ncol(group.label)-1])){
      break
    }
  }
  #
  # output the final group data with labels.
  #
  final.group <- group.label[,c(1:4,ncol(group.label))]

  labels <- final.group[,5]
  centres <- aggregate(final.group[,1:4],by=list(label=final.group[,5]),FUN=mean)
  total.D <- c(0,0,0)
  for(j in 1:G){
    total.D[j] <- sum(apply(final.group[which(final.group$re.label==j),][,1:4] -
                    as.matrix(centres[j,2:5]),1,function(x) sum(x^2)))
  }
  total.D <- sum(total.D)
  final.list <- list(labels,centres,nb.iter,total.D)
  names(final.list) <- c('labels','centres','nb.iter','total.D')
  return(final.list)
}
```

this is wrong. but the assignment to groups based on distances from the centres is correct.

```r
#
# Part(B)
```

?

```
#
iris <- iris
pairs(iris[,1:4],col=c(rep('red',50),rep('blue',50),rep('green',50)))
```

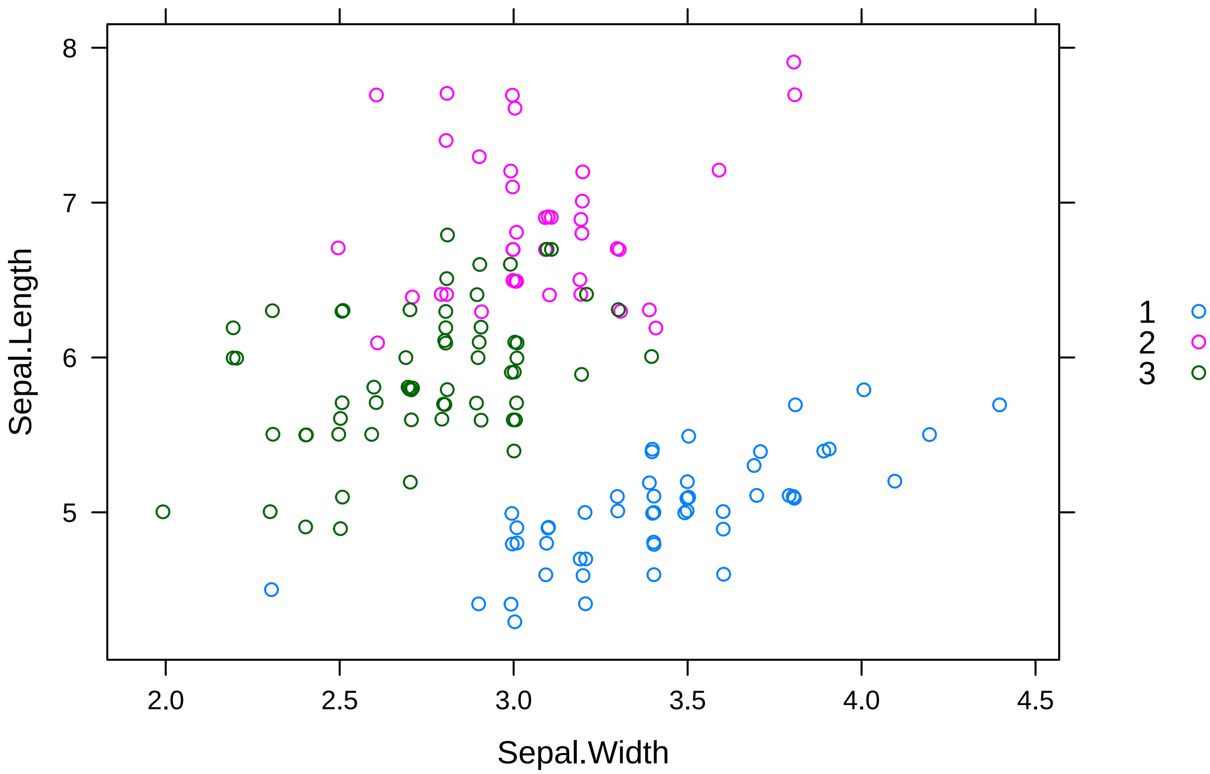



```
#
new.iris <- Clustering(iris[,1:4],G = 3)
#
# iris111 is the iris data with new labels.
#
iris111 <- cbind(iris[,1:4],new.iris$labels)

# the are 6 pairs of measurements.
xyplot(Sepal.Length ~ Sepal.Width, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```

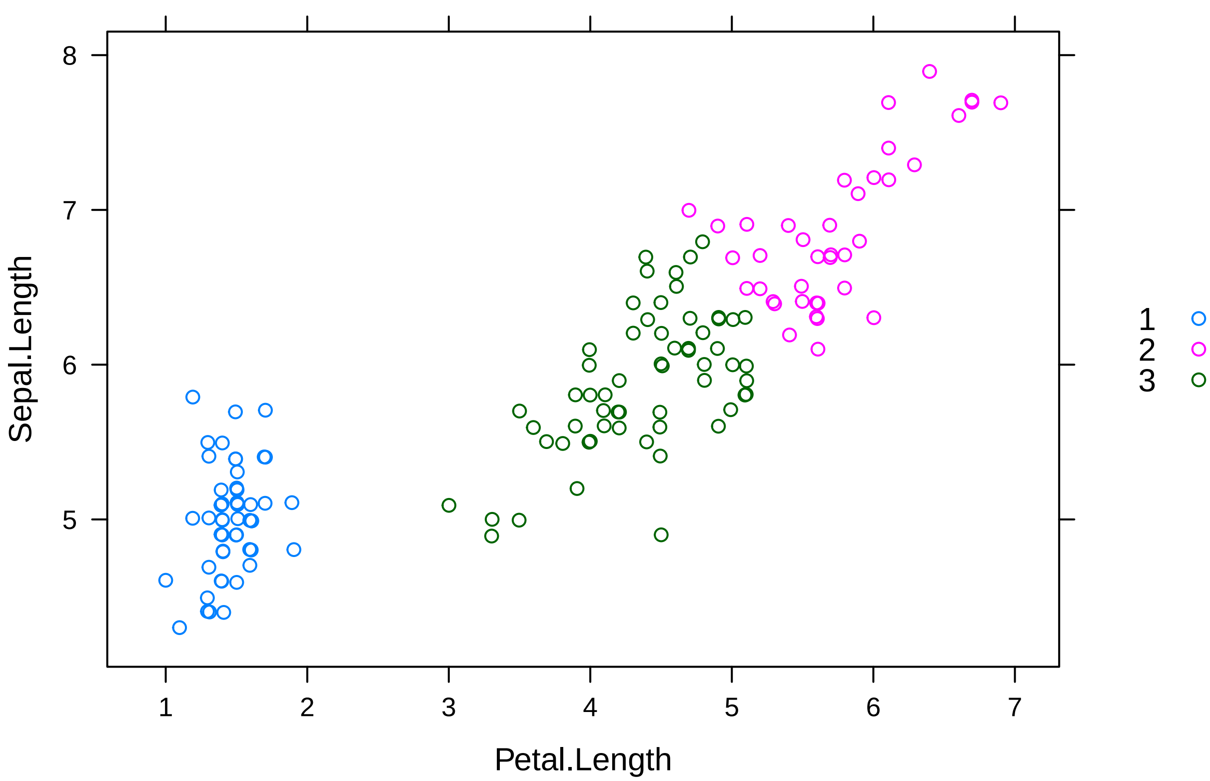
Again, I need to see what this returns.

This is important as you didn't answer the last part of the question!

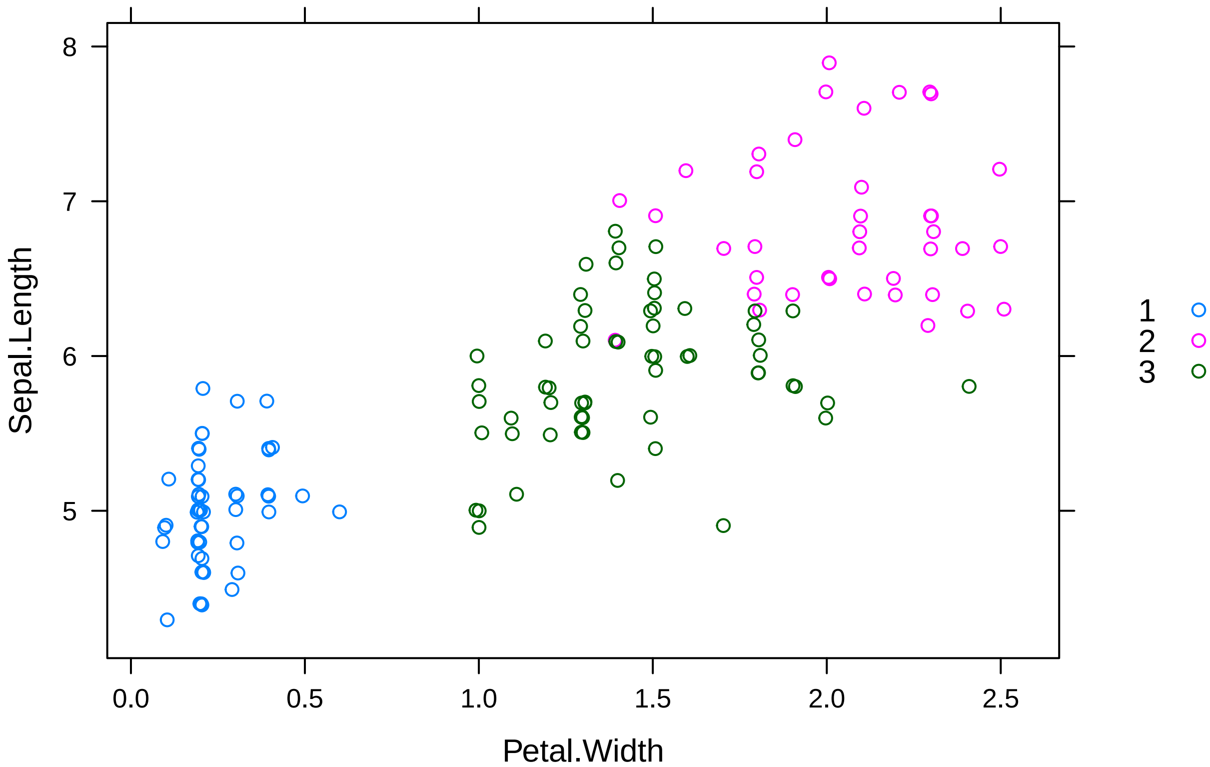
What is the proportion of missclassified points?



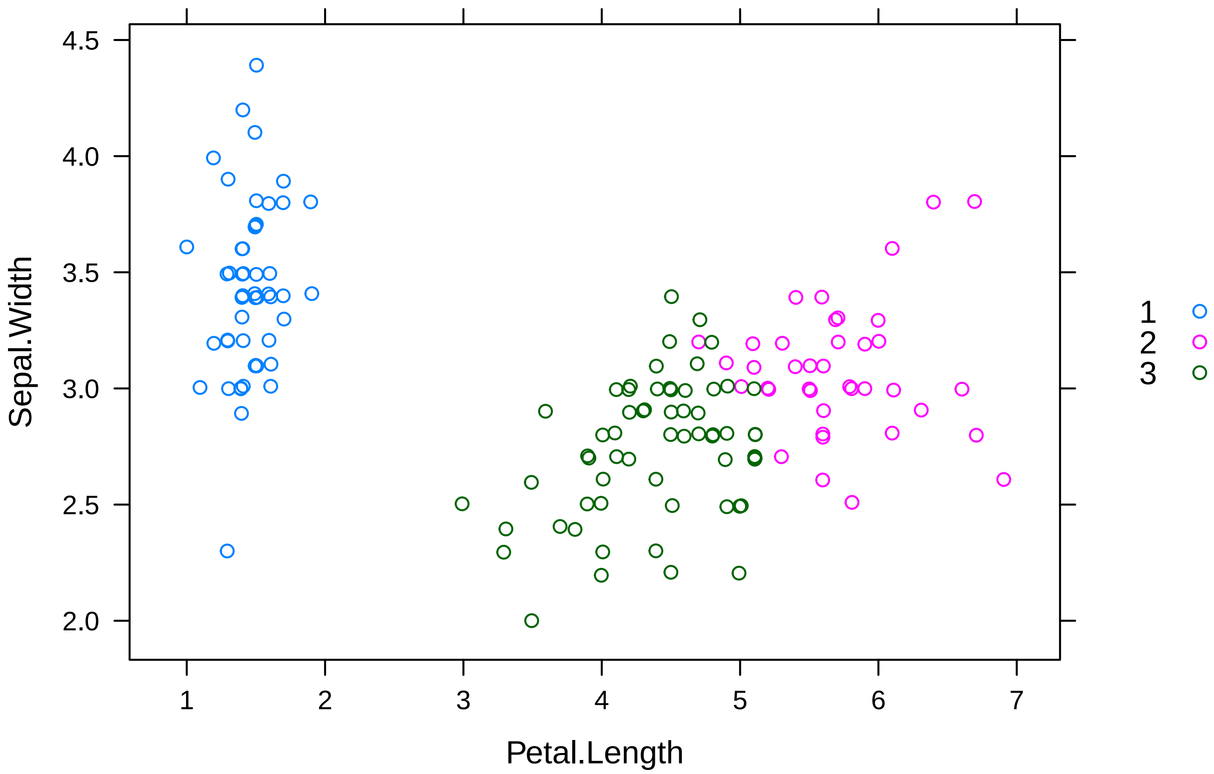

```
xyplot(Sepal.Length ~ Petal.Length, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```



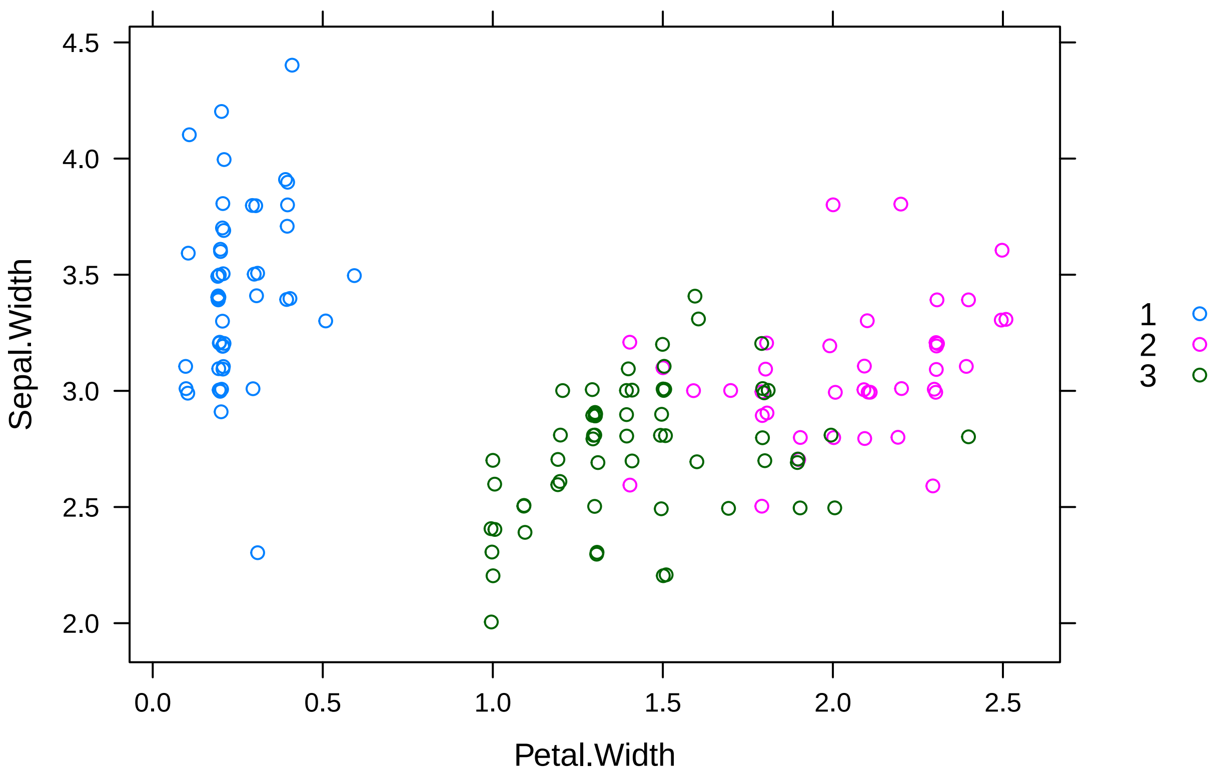
```r
xyplot(Sepal.Length ~ Petal.Width, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```




```r
xyplot(Sepal.Width ~ Petal.Length, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```




```r
xyplot(Sepal.Width ~ Petal.Width, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```



```r
xyplot(Petal.Length ~ Petal.Width, group=iris111[,5], data=iris111,
       auto.key=list(space="right"),
       jitter.x=TRUE, jitter.y=TRUE)
```