FINAl

Zijing Gao

2019/12/10

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Question 1

(a)

$$
\begin{aligned}
P(X) &= P(A,S,T,C,B,E,R,D) \\
&= P(D|E,B)P(A,S,T,C,B,E,R) \\
&= P(D|E,B)P(R|E)P(A,S,T,C,B,E) \\
&= P(D|E,B)P(R|E)P(E|T,C)P(T|A)P(A)P(C|S)P(B|S)P(S)
\end{aligned}
$$

So, $P(X)$ is completely determined for all X in the sample space of X.

the simplier form is

$$\Phi(A)\Phi(T,A)\Phi(R,E)\Phi(E,T,C)\Phi(D,E,B)\Phi(C,S)\Phi(B,S)\Phi(S)$$

For example, $\Phi(A) = P(A), \Phi(T,A) = P(T|A)$.

(b)

i

Firstly, we plug in the corresponding probabilities from the table into the stationary distribution.

```
# MH sampler

stationary_dist = function(X){
  a = X[1];s = X[2];d = X[3];r = X[4];b = X[5];c = X[7];t = X[8];e = min(1,c+
```

```
t)
  p.a = ifelse(a == 1,0.01,0.99)
  p.ta <- if(a==1){
    ifelse(t==1, 0.05, 0.95)
  }else{
    ifelse(t==1, 0.01, 0.99)
  }
  p.s = 0.5
  p.cs <- if(s==1){
    ifelse(c==1, 0.1, 0.9)
  }else{
    ifelse(c==1, 0.01, 0.99)
  }

  p.bs <- if(c==1){
    ifelse(b==1, 0.6, 0.4)
  }else{
    ifelse(b==1, 0.3, 0.7)
  }

  p.re <- if(e==1){
    ifelse(r==1, 0.98, 0.02)
  }else{
    ifelse(r==1, 0.05, 0.95)
  }

  p.deb <- if(b==1 & e==1){
    ifelse(d==1, 0.9, 0.1)
  } else if(b==0 & e==1){
    ifelse(d==1, 0.7, 0.3)
  } else if(b==1 & e==0){
    ifelse(d==1, 0.8, 0.2)
  } else{
    ifelse(d==1, 0.1, 0.9)
  }
  res = prod(p.a, p.ta, p.s, p.cs, p.bs, p.re, p.deb)
  return(res)
}

MH_sampler = function(start_X,time_steps){

  X = start_X
  X[6] = min(1,X[7]+X[8]) # e is determined by c and T from the table
  n = length(X)
  path = matrix(0, nrow = time_steps, ncol = n)

  for(i in 1:time_steps){
    path[i,] = X
```

```
    # proposal
    flip = sample.int(n,1)
    p_x = X
    p_x[flip] = ifelse(p_x[flip] == 1,0,1)

    # MH ratio
    mh_ratio = exp(log(stationary_dist(p_x)) - log(stationary_dist(X)))

    if(runif(1)<mh_ratio)
      X = p_x
  }

  colnames(path) = c("A","S","D","R","B","E","C","T")

  return(path)
}

# test
start_X = sample(c(1,0),8, replace = TRUE)

mysampler = MH_sampler(start_X,10)
mysampler

##       A S D R B E C T
##  [1,] 0 0 0 0 0 1 0 1
##  [2,] 0 0 0 0 0 1 0 1
##  [3,] 0 0 1 0 0 1 0 1
##  [4,] 0 0 1 0 0 1 0 1
##  [5,] 0 0 1 1 0 1 0 1
##  [6,] 0 0 1 1 0 1 0 1
##  [7,] 0 0 1 1 0 1 0 1
##  [8,] 0 0 1 1 0 1 0 1
##  [9,] 0 0 1 1 0 1 0 1
## [10,] 0 1 1 1 0 1 0 1
```

I use it to compute $P(R = 1|A, S, D)$

```
P.rasd = function(X_rest,ASD){
  start_X = c(ASD,X_rest)
  mh = MH_sampler(start_X, time_steps = 500000)
  mh = as.data.frame(mh[300000:500000,]) # burn-in
  m_ASD = mh %>% select(A,S,D,R) %>% filter(A == ASD[1] & S == ASD[2] & D ==
ASD[3])

  m_R = m_ASD %>% filter(R == 1)

  prob = nrow(m_R)/nrow(m_ASD)
  return(prob)
}
```

```
X_rest = sample(c(1,0),5,replace = TRUE)
ASD = c(1,0,1)
estimate_P = P.rasd(X_rest,ASD)
estimate_P

## [1] 0.1765517
```

(b)

(ii)

Using the relation, we have

$$P(R=1|A=1,S=0,D=1)=\frac{P(R=1,A=1,S=0,D=1)}{P(A=1,S=0,D=1)}$$

The numeriator is

$$P(R=1,A=1,S=0,D=1)=\sum_{T=0}^{1}\sum_{C=0}^{1}\sum_{B=0}^{1}\sum_{E=0}^{1}P(X)$$

The denominator is

$$P(A=1,S=0,D=1)=\sum_{R=0}^{1}\sum_{T=0}^{1}\sum_{C=0}^{1}\sum_{B=0}^{1}\sum_{E=0}^{1}P(X)$$

```
m_numerator = cbind(A = 1,S = 0,D = 1,R = 1,expand.grid(B=0:1,E=0:1,C=0:1,T=0:
1))

m_denominator = cbind(A = 1,S = 0,D = 1,expand.grid(R=0:1,B=0:1,E=0:1,C=0:1,T
=0:1))

numerator = apply(m_numerator,1,stationary_dist) %>% sum
denominator = apply(m_denominator,1,stationary_dist) %>% sum

actual_p = numerator/denominator
actual_p

## [1] 0.1763063
```

Question 2

(a)

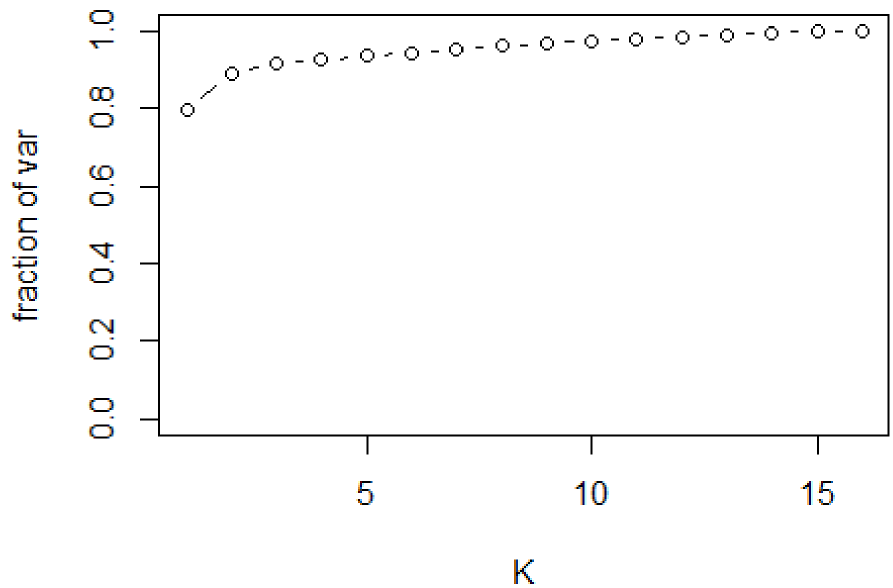
i

```
dat = read.csv("grades.csv")
```

```
myPCA = prcomp(dat)
# center and scale refers to respective mean and standard deviation of the va
riables that are used for normalization prior to implementing PCA
```

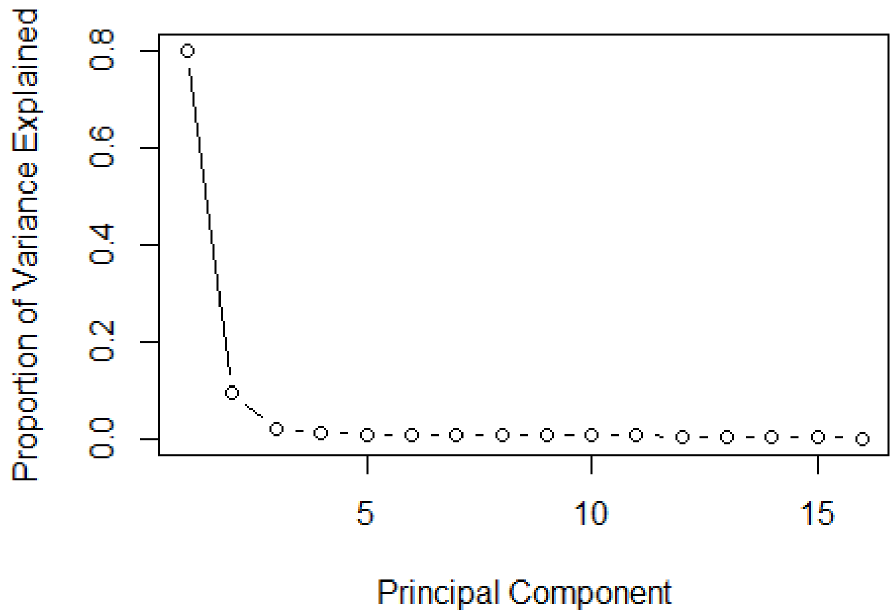
Here is PCA function

```
mypca <- function(X) {


  N <- nrow(X)
  mu <- colMeans(X);
  tildeX <- apply(X, 1, function(xi) xi - mu) %>% t

  Sigma <- 1/N*t(tildeX) %*% tildeX
  eigen.out <- eigen(Sigma)

  # get first two eigenvectors for part (ii)
  Q <- eigen.out$vectors[,1:2]
  # get all eigenvalues for part (i)
  ev <- eigen(Sigma)$values
  print(ev)

  plot(cumsum(ev)/sum(ev), xlab="K", ylab="fraction of var", ylim=c(0,1),type
 = "b")

  c <- tildeX %*% Q

  # part (iii)
  cov_c <- cov(c)
  cat("mean of c1 and c2", mean(c[,1]), mean(c[,2]), "\n")
  cat("covariance of c1 and c2", cov_c[1,2], "\n")
  cat("variance of c1 and c2", mean(c[,1]^2), mean(c[,2]^2), "\n")
  cat("first two eigenvalues", ev[1:2], "\n")

}

mypca(dat)

##  [1]  2.444963e+03  2.873013e+02  6.730160e+01  3.220885e+01  3.028332e+01
##  [6]  2.852416e+01  2.742656e+01  2.508522e+01  2.225354e+01  1.943409e+01
## [11]  1.911465e+01  1.623206e+01  1.465519e+01  1.417747e+01  1.303357e+01
## [16] -1.102239e-13
```

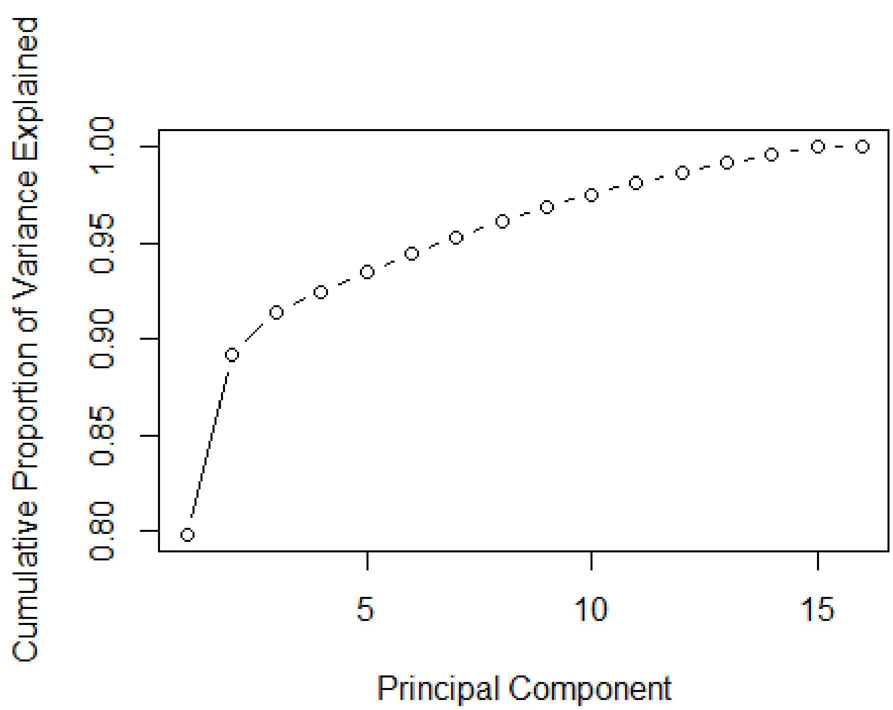



```
## mean of c1 and c2 -2.355837e-15 6.808613e-15
## covariance of c1 and c2 -1.406483e-12
## variance of c1 and c2 2444.963 287.3013
## first two eigenvalues 2444.963 287.3013
```

```r
dev = myPCA$sdev

var = dev^2

prop_var = var/sum(var)

plot(prop_var, xlab = "Principal Component",
             ylab = "Proportion of Variance Explained",
             type = "b")
```



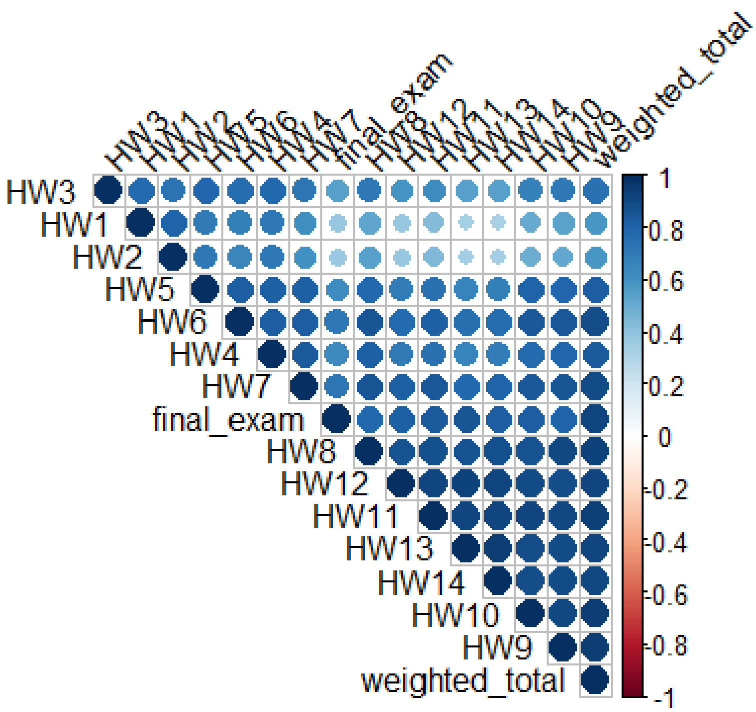

```r
plot(cumsum(prop_var), xlab = "Principal Component",
              ylab = "Cumulative Proportion of Variance Explained",
              type = "b")
```



From the plot, we can see that almost 90% variance in the data set can be explained by 2 components.

ii

```
library(corrplot)

## corrplot 0.84 loaded

corr = cor(dat)
corrplot(corr, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```




```
corr

##                      HW1       HW2       HW3       HW4       HW5       HW6
## HW1            1.0000000 0.8004422 0.7712536 0.7176956 0.7029132 0.6854362
## HW2            0.8004422 1.0000000 0.7378522 0.7190658 0.7123571 0.6595701
## HW3            0.7712536 0.7378522 1.0000000 0.7892537 0.7985741 0.7644114
## HW4            0.7176956 0.7190658 0.7892537 1.0000000 0.8183827 0.8233490
## HW5            0.7029132 0.7123571 0.7985741 0.8183827 1.0000000 0.8223197
## HW6            0.6854362 0.6595701 0.7644114 0.8233490 0.8223197 1.0000000
## HW7            0.6151078 0.6067787 0.7236082 0.8360619 0.8150986 0.8232179
## HW8            0.5291789 0.5434235 0.7176091 0.8119957 0.7846286 0.8524682
## HW9            0.5322441 0.5131162 0.7179150 0.7989813 0.7958793 0.8416975
## HW10           0.5033430 0.4963965 0.6779905 0.7702390 0.8062538 0.8448787
## HW11           0.4328231 0.4404652 0.6269860 0.7437481 0.7417378 0.8137535
## HW12           0.3878362 0.3824429 0.5944955 0.7076916 0.6948544 0.7796751
```

```
## HW13           0.3343705 0.3419050 0.5421543 0.6690324 0.6682266 0.7558206
## HW14           0.3257472 0.3401632 0.5450630 0.6993907 0.6861845 0.7649038
## final_exam     0.3804571 0.3733972 0.5486601 0.6321795 0.6248178 0.7169774
## weighted_total 0.5852074 0.5807193 0.7479931 0.8353414 0.8299378 0.8885455
##                      HW7       HW8       HW9      HW10      HW11      HW12
## HW1            0.6151078 0.5291789 0.5322441 0.5033430 0.4328231 0.3878362
## HW2            0.6067787 0.5434235 0.5131162 0.4963965 0.4404652 0.3824429
## HW3            0.7236082 0.7176091 0.7179150 0.6779905 0.6269860 0.5944955
## HW4            0.8360619 0.8119957 0.7989813 0.7702390 0.7437481 0.7076916
## HW5            0.8150986 0.7846286 0.7958793 0.8062538 0.7417378 0.6948544
## HW6            0.8232179 0.8524682 0.8416975 0.8448787 0.8137535 0.7796751
## HW7            1.0000000 0.8563578 0.8529312 0.8420689 0.8451175 0.8162409
## HW8            0.8563578 1.0000000 0.9010040 0.8699171 0.8795309 0.8684842
## HW9            0.8529312 0.9010040 1.0000000 0.9094128 0.9072048 0.8886536
## HW10           0.8420689 0.8699171 0.9094128 1.0000000 0.9120860 0.8932285
## HW11           0.8451175 0.8795309 0.9072048 0.9120860 1.0000000 0.9121485
## HW12           0.8162409 0.8684842 0.8886536 0.8932285 0.9121485 1.0000000
## HW13           0.7811846 0.8602647 0.8928006 0.8834410 0.9167467 0.9262406
## HW14           0.8023462 0.8762204 0.8930592 0.8842400 0.9183786 0.9154517
## final_exam     0.7277017 0.7865612 0.8078320 0.8263148 0.8360160 0.8185176
## weighted_total 0.8948895 0.9284772 0.9429228 0.9415268 0.9373424 0.9157970
##                     HW13      HW14 final_exam weighted_total
## HW1            0.3343705 0.3257472  0.3804571      0.5852074
## HW2            0.3419050 0.3401632  0.3733972      0.5807193
## HW3            0.5421543 0.5450630  0.5486601      0.7479931
## HW4            0.6690324 0.6993907  0.6321795      0.8353414
## HW5            0.6682266 0.6861845  0.6248178      0.8299378
## HW6            0.7558206 0.7649038  0.7169774      0.8885455
## HW7            0.7811846 0.8023462  0.7277017      0.8948895
## HW8            0.8602647 0.8762204  0.7865612      0.9284772
## HW9            0.8928006 0.8930592  0.8078320      0.9429228
## HW10           0.8834410 0.8842400  0.8263148      0.9415268
## HW11           0.9167467 0.9183786  0.8360160      0.9373424
## HW12           0.9262406 0.9154517  0.8185176      0.9157970
## HW13           1.0000000 0.9422003  0.8541608      0.9160633
## HW14           0.9422003 1.0000000  0.8226851      0.9098346
## final_exam     0.8541608 0.8226851  1.0000000      0.9132108
## weighted_total 0.9160633 0.9098346  0.9132108      1.0000000
```

Here is my own correlation matrix function.

We know that

$$r_{xy} = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}}$$

```r
# correlation matrix
mycorr = function(X){
  corr_m = matrix(0,nrow = ncol(X), ncol = ncol(X))
  for(i in 1:ncol(X)){
```

```
    for(j in 1:ncol(X)){
      numerator = sum((X[,i]-mean(X[,i]))*(X[,j]-mean(X[,j])))
      denominator = (sum((X[,i]-mean(X[,i]))^2) * sum((X[,j]-mean(X[,j]))^2))
 %>% sqrt
      corr_m[i,j] = numerator/denominator
    }
  }
  colnames(corr_m) = colnames(X)
  row.names(corr_m) = colnames(X)
  return(corr_m)
}

mycorr(dat)

##                      HW1       HW2       HW3       HW4       HW5       HW6
## HW1            1.0000000 0.8004422 0.7712536 0.7176956 0.7029132 0.6854362
## HW2            0.8004422 1.0000000 0.7378522 0.7190658 0.7123571 0.6595701
## HW3            0.7712536 0.7378522 1.0000000 0.7892537 0.7985741 0.7644114
## HW4            0.7176956 0.7190658 0.7892537 1.0000000 0.8183827 0.8233490
## HW5            0.7029132 0.7123571 0.7985741 0.8183827 1.0000000 0.8223197
## HW6            0.6854362 0.6595701 0.7644114 0.8233490 0.8223197 1.0000000
## HW7            0.6151078 0.6067787 0.7236082 0.8360619 0.8150986 0.8232179
## HW8            0.5291789 0.5434235 0.7176091 0.8119957 0.7846286 0.8524682
## HW9            0.5322441 0.5131162 0.7179150 0.7989813 0.7958793 0.8416975
## HW10           0.5033430 0.4963965 0.6779905 0.7702390 0.8062538 0.8448787
## HW11           0.4328231 0.4404652 0.6269860 0.7437481 0.7417378 0.8137535
## HW12           0.3878362 0.3824429 0.5944955 0.7076916 0.6948544 0.7796751
## HW13           0.3343705 0.3419050 0.5421543 0.6690324 0.6682266 0.7558206
## HW14           0.3257472 0.3401632 0.5450630 0.6993907 0.6861845 0.7649038
## final_exam     0.3804571 0.3733972 0.5486601 0.6321795 0.6248178 0.7169774
## weighted_total 0.5852074 0.5807193 0.7479931 0.8353414 0.8299378 0.8885455
##                      HW7       HW8       HW9      HW10      HW11      HW12
## HW1            0.6151078 0.5291789 0.5322441 0.5033430 0.4328231 0.3878362
## HW2            0.6067787 0.5434235 0.5131162 0.4963965 0.4404652 0.3824429
## HW3            0.7236082 0.7176091 0.7179150 0.6779905 0.6269860 0.5944955
## HW4            0.8360619 0.8119957 0.7989813 0.7702390 0.7437481 0.7076916
## HW5            0.8150986 0.7846286 0.7958793 0.8062538 0.7417378 0.6948544
## HW6            0.8232179 0.8524682 0.8416975 0.8448787 0.8137535 0.7796751
## HW7            1.0000000 0.8563578 0.8529312 0.8420689 0.8451175 0.8162409
## HW8            0.8563578 1.0000000 0.9010040 0.8699171 0.8795309 0.8684842
## HW9            0.8529312 0.9010040 1.0000000 0.9094128 0.9072048 0.8886536
## HW10           0.8420689 0.8699171 0.9094128 1.0000000 0.9120860 0.8932285
## HW11           0.8451175 0.8795309 0.9072048 0.9120860 1.0000000 0.9121485
## HW12           0.8162409 0.8684842 0.8886536 0.8932285 0.9121485 1.0000000
## HW13           0.7811846 0.8602647 0.8928006 0.8834410 0.9167467 0.9262406
## HW14           0.8023462 0.8762204 0.8930592 0.8842400 0.9183786 0.9154517
## final_exam     0.7277017 0.7865612 0.8078320 0.8263148 0.8360160 0.8185176
## weighted_total 0.8948895 0.9284772 0.9429228 0.9415268 0.9373424 0.9157970
##                     HW13      HW14 final_exam weighted_total
## HW1            0.3343705 0.3257472  0.3804571      0.5852074
```

```
## HW2            0.3419050 0.3401632  0.3733972      0.5807193
## HW3            0.5421543 0.5450630  0.5486601      0.7479931
## HW4            0.6690324 0.6993907  0.6321795      0.8353414
## HW5            0.6682266 0.6861845  0.6248178      0.8299378
## HW6            0.7558206 0.7649038  0.7169774      0.8885455
## HW7            0.7811846 0.8023462  0.7277017      0.8948895
## HW8            0.8602647 0.8762204  0.7865612      0.9284772
## HW9            0.8928006 0.8930592  0.8078320      0.9429228
## HW10           0.8834410 0.8842400  0.8263148      0.9415268
## HW11           0.9167467 0.9183786  0.8360160      0.9373424
## HW12           0.9262406 0.9154517  0.8185176      0.9157970
## HW13           1.0000000 0.9422003  0.8541608      0.9160633
## HW14           0.9422003 1.0000000  0.8226851      0.9098346
## final_exam     0.8541608 0.8226851  1.0000000      0.9132108
## weighted_total 0.9160633 0.9098346  0.9132108      1.0000000
```

iii

```
# step 1 bin the data

data_bin = function(X){
  X = X[,1:16-1]
  # Since the weighted total are not integers, I assume I could not find the
joint probability.
  # It is meaningless to compute the MI of it because they are all different.
  n = ncol(X)
  for (i in 1:n) {
    c <- quantile(X[,i], seq(0,1,0.05))
    gbin <- as.numeric(cut(X[,i], breaks = unique(c),
    include.lowest = TRUE))
    X[,i] <- gbin
  }
  return(X)
}

# step 2 compute entropy of each pair

MyEntropy = function(X){
  p = table(X)/length(X)
  res = -sum(p*log(p))
  return(res)
}

# step 3 compute the mutual information of each pair

MI = function(X,Y,method = "standard"){
  res = 0
  for(x in unique(X)){
    for(y in unique(Y)){
      pxy <- mean(X==x & Y==y)
      if(pxy!=0){
```

```
        px <- mean(X==x)
        py <- mean(Y==y)
        res <- res + pxy * log(pxy/px/py)
      }
    }
  }

  if(method == "normalized")
    res = 2*res/(MyEntropy(X)+MyEntropy(Y)) # normalize the MI if needed
  return(res)
}

# step 4 compute the mutual information matrix

MIM = function(X,method = "standard"){
  X = data_bin(X)
  n = ncol(X)
  M = matrix(0,nrow = n, ncol = n)
  for(i in 1:n){
    for(j in 1:n){
      M[i,j] = MI(X[,i],X[,j],method = method)
    }
  }
  row.names(M) = colnames(X)
  colnames(M) = colnames(X)

  return(M)
}

mutual_info_matrix = MIM(dat,method = "normalized")
mutual_info_matrix
```

```
##                  HW1       HW2       HW3       HW4       HW5       HW6
## HW1        1.0000000 0.4599706 0.4359510 0.4069212 0.4088218 0.4123418
## HW2        0.4599706 1.0000000 0.4565146 0.4044556 0.4261480 0.4117339
## HW3        0.4359510 0.4565146 1.0000000 0.4496807 0.4557124 0.4326624
## HW4        0.4069212 0.4044556 0.4496807 1.0000000 0.4374475 0.4339905
## HW5        0.4088218 0.4261480 0.4557124 0.4374475 1.0000000 0.4263413
## HW6        0.4123418 0.4117339 0.4326624 0.4339905 0.4263413 1.0000000
## HW7        0.4140061 0.4101959 0.4196413 0.4414670 0.4448932 0.4367213
## HW8        0.4145062 0.4327756 0.4436892 0.4368562 0.4293055 0.4633965
## HW9        0.3938463 0.3618583 0.4356817 0.3964566 0.4428296 0.4552380
## HW10       0.4006207 0.3763764 0.4605773 0.4248603 0.4638262 0.4640461
## HW11       0.3679483 0.3765498 0.3905254 0.3944775 0.4251898 0.4058119
## HW12       0.3618739 0.3529097 0.3797422 0.3667018 0.3740056 0.3940596
## HW13       0.3487601 0.3331339 0.3950683 0.3826270 0.3972765 0.4012629
## HW14       0.3356840 0.3263938 0.3520960 0.3709962 0.4046729 0.4018766
## final_exam 0.3330294 0.3393042 0.3650782 0.3505631 0.3779911 0.3877533
##                  HW7       HW8       HW9      HW10      HW11      HW12
## HW1        0.4140061 0.4145062 0.3938463 0.4006207 0.3679483 0.3618739
```

```
## HW2        0.4101959 0.4327756 0.3618583 0.3763764 0.3765498 0.3529097
## HW3        0.4196413 0.4436892 0.4356817 0.4605773 0.3905254 0.3797422
## HW4        0.4414670 0.4368562 0.3964566 0.4248603 0.3944775 0.3667018
## HW5        0.4448932 0.4293055 0.4428296 0.4638262 0.4251898 0.3740056
## HW6        0.4367213 0.4633965 0.4552380 0.4640461 0.4058119 0.3940596
## HW7        1.0000000 0.4874755 0.4318626 0.4472763 0.4382257 0.4368328
## HW8        0.4874755 1.0000000 0.4884062 0.4988844 0.4223830 0.4600006
## HW9        0.4318626 0.4884062 1.0000000 0.4903794 0.4761482 0.4559159
## HW10       0.4472763 0.4988844 0.4903794 1.0000000 0.4581082 0.4613796
## HW11       0.4382257 0.4223830 0.4761482 0.4581082 1.0000000 0.4623484
## HW12       0.4368328 0.4600006 0.4559159 0.4613796 0.4623484 1.0000000
## HW13       0.4255080 0.4297325 0.4742205 0.4739852 0.5019769 0.4928483
## HW14       0.4337992 0.4663587 0.4586551 0.4650992 0.4982486 0.4908181
## final_exam 0.4154834 0.3904348 0.4070262 0.4135650 0.4267065 0.4138646
##                 HW13      HW14 final_exam
## HW1        0.3487601 0.3356840  0.3330294
## HW2        0.3331339 0.3263938  0.3393042
## HW3        0.3950683 0.3520960  0.3650782
## HW4        0.3826270 0.3709962  0.3505631
## HW5        0.3972765 0.4046729  0.3779911
## HW6        0.4012629 0.4018766  0.3877533
## HW7        0.4255080 0.4337992  0.4154834
## HW8        0.4297325 0.4663587  0.3904348
## HW9        0.4742205 0.4586551  0.4070262
## HW10       0.4739852 0.4650992  0.4135650
## HW11       0.5019769 0.4982486  0.4267065
## HW12       0.4928483 0.4908181  0.4138646
## HW13       1.0000000 0.5020475  0.4263429
## HW14       0.5020475 1.0000000  0.3886404
## final_exam 0.4263429 0.3886404  1.0000000
```

(b)

we want to compute the distribution of the weighted total conditioned on the first six homework grades.

$$P(weighted\ total|X_{16})$$

where X_{16} is $X_1, X_2, X_3, X_4, X_5, X_6$

In this case, we have

$$Weighted\ Total = 0.7\ [X_1, \dots, X_{14}] + 0.3\ final$$

And we try to use normal equation and lm() to analyse the weighted total by using $X_1, X_2, X_3, X_4, X_5, X_6$.

```
# MLE estimate

dat$average_hw = rowMeans(dat[,1:14])
X_16 = dat[,1:6]
```

```
mu_16 = X_16
mu_16[,1:6] = rowMeans(X_16)

# final ~ HW1-HW6

y1 = dat$final_exam
B = cbind(rep(1,length(y1)), dat[,1:6] %>% as.matrix)

a1 = solve(t(B) %*% B, t(B) %*% y1) %>% as.numeric
sigma2 = mean((y1 - B %*% a1)^2)

MLE_1 = c(a1,sigma2) %>% setNames(c(paste0('a',0:6),"sigma2"))
MLE_1

##          a0          a1          a2          a3          a4          a5
##   8.7997859  -0.3451830  -0.2842515   0.1425440   0.3314955   0.2226979
##          a6      sigma2
##   0.8321140 113.8703004

# average ~ HW1-HW6

y2 = dat$average_hw
a2 = solve(t(B) %*% B, t(B) %*% y2) %>% as.numeric
sigma2 = mean((y2 - B %*% a2)^2)

MLE_2 = c(a2,sigma2) %>% setNames(c(paste0('a',0:6),"sigma2"))
MLE_2

##          a0          a1          a2          a3          a4          a5
##  1.35532364 -0.14062798 -0.09040413  0.13092890  0.33476758  0.28003384
##          a6      sigma2
##  0.47154057 11.19757835

# weighted total ~ HW1-HW6

y3 = dat$weighted_total
a3 = solve(t(B) %*% B, t(B) %*% y3) %>% as.numeric
sigma2 = mean((y3 - B %*% a3)^2)

MLE_3 = c(a3,sigma2) %>% setNames(c(paste0('a',0:6),"sigma2"))
MLE_3

##         a0         a1         a2         a3         a4         a5
##  3.2164392 -0.1917667 -0.1388660  0.1338327  0.3339496  0.2656999
##         a6     sigma2
##  0.5616839 21.9332355
```

Here, I try to take Bayesian approach to esitimate the paramters.

```
# compute log posterior probability
log_posterior <- function(theta, y, B)
```

```
{
  a <- theta[1:7]
  s2 <- theta[8]

  # get log priors
  fa <- dnorm(a, mean=0, sd=10, log=T) %>% sum
  fs2 <- -s2

  likeli <- dnorm(y - B %*% a, mean=0, sd=sqrt(s2), log=T) %>% sum

  return (fa + fs2 + likeli)
}

MH <- function(theta, y, B, iter=1E4)
{
  theta_m <- matrix(0, nrow=iter, ncol=8)

  for (i in 1:iter) {
    p_theta <- theta + rnorm(8, mean=0, sd=0.5)
    p_theta[8] <- abs(p_theta[8])

    # proposal is symmetric, so the ratio R_ps/R_sp = 1
    MH_ratio <- exp(log_posterior(p_theta,y,B) - log_posterior(theta,y,B))
    if (runif(1) < MH_ratio)
      theta <- p_theta

    theta_m[i,] <- theta

  }

  colnames(theta_m) <- c(paste0('a',0:6),"sigma2")
  return (theta_m)
}

# test
theta <- sample(1:10,8)
y = y3
MH(theta, y, B, iter=10)

##              a0       a1       a2       a3       a4       a5       a6
##  [1,] 10.000000 5.000000 8.000000 3.000000 4.000000 7.000000 6.000000
##  [2,] 10.000000 5.000000 8.000000 3.000000 4.000000 7.000000 6.000000
##  [3,] 10.227124 4.885092 7.566383 2.634680 4.708715 6.673049 6.283047
##  [4,] 10.258722 4.371114 7.634504 2.513882 4.886061 6.595877 6.281945
##  [5,]  9.073756 3.935217 6.899899 2.859663 4.676052 6.434837 6.994150
##  [6,]  9.301783 3.556208 6.954942 2.250546 4.492589 5.739745 7.115569
##  [7,]  9.301783 3.556208 6.954942 2.250546 4.492589 5.739745 7.115569
##  [8,]  9.502434 4.754504 6.660602 1.644092 4.597796 6.026393 7.254331
##  [9,]  9.502434 4.754504 6.660602 1.644092 4.597796 6.026393 7.254331
```

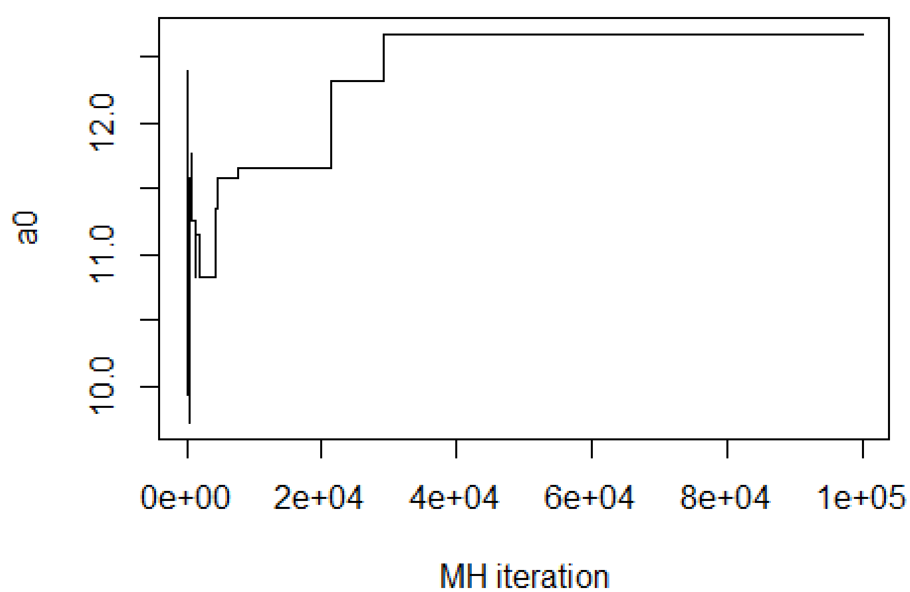
```
## [10,]  9.502434 4.754504 6.660602 1.644092 4.597796 6.026393 7.254331
##          sigma2
##   [1,] 1.000000
##   [2,] 1.000000
##   [3,] 1.591991
##   [4,] 2.161446
##   [5,] 2.542795
##   [6,] 3.471225
##   [7,] 3.471225
##   [8,] 3.808215
##   [9,] 3.808215
## [10,] 3.808215
```

Looks reasonable, let's run for a while...

```
m <- MH(theta, y, B, iter=1E5)
```

Just to check for convergence, let's plot the a_0 values

```
plot(1:nrow(m), m[,"a0"], xlab="MH iteration", ylab="a0", type="l")
```



It does not seem to comply with the value of a_0 I just computed.

But, we can model the expected value of y as a linear function of 6 explanatory variables like this.

$$E[weighted\ total|X_1,\ldots,X_6] = a_0 + a_1X_1 + \ldots + a_6X_6$$

```
MLE_3.matrix = matrix(0,ncol = 7,nrow = 147)
for(i in 1:147)
  MLE_3.matrix[i,] = MLE_3[-8] # keep out sigma2

wt.pred = rowSums(MLE_3.matrix * B)

# here I write bin function again
mybin = function(X){
  c <- quantile(wt.pred, seq(0,1,0.05)) # bin the data
  bin <- as.numeric(cut(wt.pred, breaks = unique(c),
    include.lowest = TRUE))
  return(bin)
}

# so, I use table to compute the frequency of weighted total given HW1-6

bin = mybin(wt.pred)
freq = table(bin)/sum(table(bin)) %>% as.vector
freq

## bin
##          1          2          3          4          5          6
## 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905 0.04761905
##          7          8          9         10         11         12
## 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905 0.04761905
##         13         14         15         16         17         18
## 0.04761905 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905
##         19         20
## 0.04761905 0.05442177
```

I also use linear regression model to fit the data.

```
lm1 = lm(average_hw~HW1+HW2+HW3+HW4+HW5+HW6, data = dat)
lm2 = lm(final_exam~HW1+HW2+HW3+HW4+HW5+HW6, data = dat)
lm3 = lm(weighted_total~HW1+HW2+HW3+HW4+HW5+HW6, data = dat)

WeightedTotalDistribution = function(X,method = "bin_prob"){
    res = 0.7 * predict(lm1,X) + 0.3 * predict(lm2, X)
    bin = mybin(res)

    freq = table(bin)/sum(table(bin)) %>% as.vector

    if(method == "pred_value")
      return(res)

    return(freq)
}
WeightedTotalDistribution(mu_16)

## bin
##          1          2          3          4          5          6
```

```
## 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905 0.04761905
##          7          8          9         10         11         12
## 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905 0.04761905
##         13         14         15         16         17         18
## 0.04761905 0.05442177 0.04761905 0.04761905 0.05442177 0.04761905
##         19         20
## 0.04761905 0.05442177
```

(c)

i

Now, I compute $E[weighted\ total|X_1,...,X_6]$

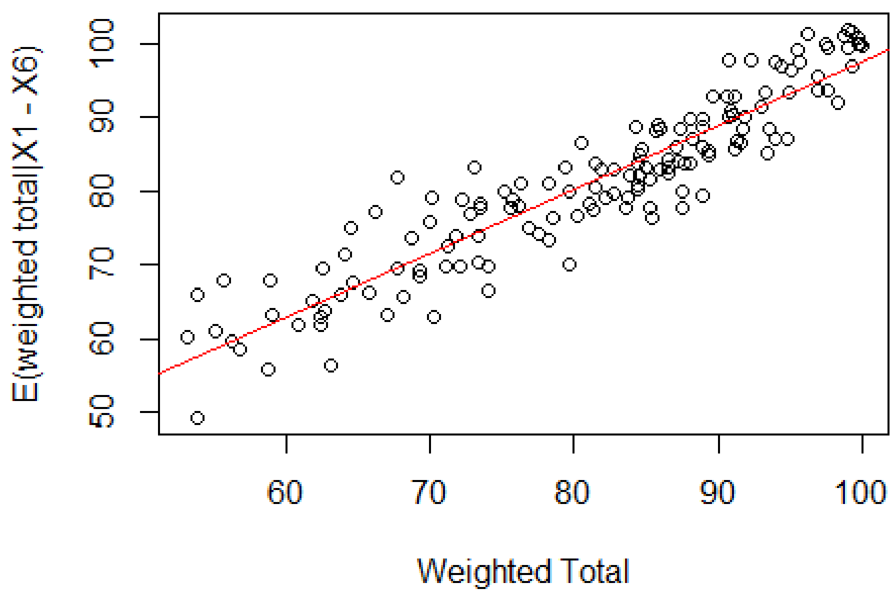
As is mentioned above, we can use the linear regression to model the expected value of weighted total given $X_1,...,X_6$.

```
E.X16 <- WeightedTotalDistribution(X_16,method = "pred_value")
E.X16
```

```
##         1         2         3         4         5         6         7
##  99.32848  83.67835  83.14987  88.44306  77.82948  84.84874  78.93776
##         8         9        10        11        12        13        14
##  70.42895  67.75814  84.10731 100.81985  55.68139  97.81353  80.90754
##        15        16        17        18        19        20        21
##  49.25989  91.98860 101.57439  69.88605  85.59818 101.06982  69.54936
##        22        23        24        25        26        27        28
##  88.01729  81.95191  71.51578  77.05995  63.29893  73.32525  88.33984
##        29        30        31        32        33        34        35
##  82.16650  77.01700  83.18500  79.23230  85.85989  61.78825  93.46940
##        36        37        38        39        40        41        42
##  92.79069 101.40748  88.48752  79.99715  76.46262  86.06693  82.27258
##        43        44        45        46        47        48        49
##  65.24735  84.72818  97.54334  58.47496  87.04789  69.68232  97.69144
##        50        51        52        53        54        55        56
##  65.74081  78.40628 100.05356  65.92828  99.60788  99.60788  75.01405
##        57        58        59        60        61        62        63
##  66.05994  63.00518  90.25989  92.80909  79.71394  80.68491  92.93052
##        64        65        66        67        68        69        70
##  70.04725  95.63636  86.43407  61.96837  86.98764  56.31151  83.10182
##        71        72        73        74        75        76        77
##  66.25659  87.16088  80.01006  77.61465  97.50030  77.67807  79.41776
##        78        79        80        81        82        83        84
##  63.15197  82.81730  84.22060  99.20482  89.09162  77.80900 101.87151
##        85        86        87        88        89        90        91
##  89.75501  96.79793  74.88048  72.51849  59.52269  60.17350  83.77876
##        92        93        94        95        96        97        98
##  76.61155  61.13388  86.48738  83.66336  78.87050  88.82144  73.67704
##        99       100       101       102       103       104       105
##  77.87603  81.01316  85.81042  88.78087  76.24187  81.66135  80.17427
```

```
##       106       107       108       109       110       111       112 
## 99.93263 79.02937 68.49966 75.77014 74.25355 67.81939 93.32234 
##       113       114       115       116       117       118       119 
## 85.08504 73.89275 83.34322 82.22279 79.22194 82.81890 90.06906 
##       120       121       122       123       124       125       126 
## 86.86588 63.82787 80.49902 90.98495 89.82655 96.96555 69.20194 
##       127       128       129       130       131       132       133 
## 88.43182 66.42589 77.82302 77.35819 83.67549 79.98272 82.98535 
##       134       135       136       137       138       139       140 
## 93.73372 67.58667 73.88745 96.44687 99.37836 85.41945 69.83807 
##       141       142       143       144       145       146       147 
## 99.86833 91.48105 63.01445 90.11345 69.54125 93.76001 78.39499
```

```
plot(y = E.X16, x = dat$weighted_total, xlab = "Weighted Total", ylab = "E(we
ighted total|X1 - X6)")
abline(fit1 <-lm(E.X16~dat$weighted_total),col=2)
```

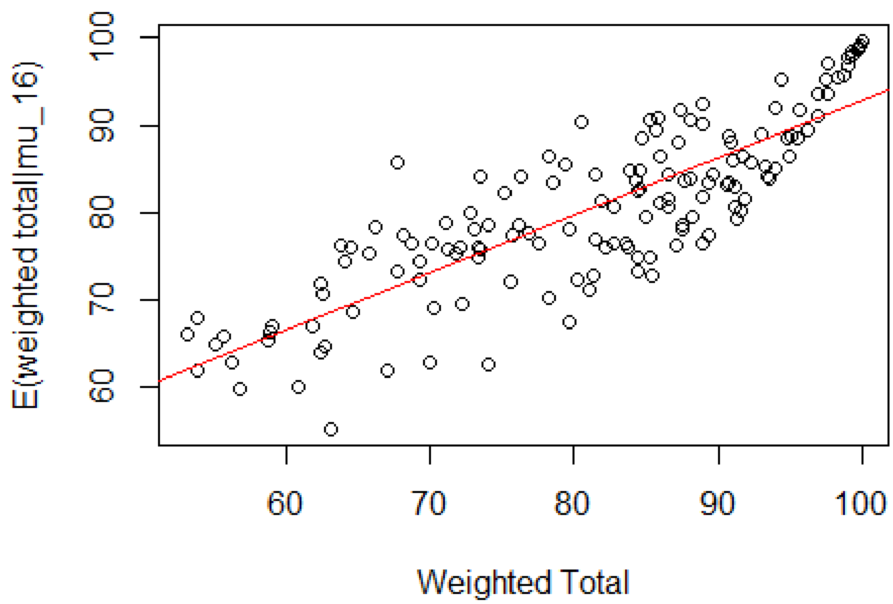



```
E.mu16 <- WeightedTotalDistribution(mu_16,method = "pred_value")
E.mu16
```

```
##        1        2        3        4        5        6        7        8 
## 93.52667 83.92475 78.00356 91.76632 75.76311 82.80452 77.52346 76.08318 
##        9       10       11       12       13       14       15       16 
## 66.32122 88.08558 98.64769 65.52106 88.72571 86.32523 61.84032 95.44705 
##       17       18       19       20       21       22       23       24 
## 98.00756 75.92314 88.40564 95.76712 73.20260 89.36583 85.84513 74.48285 
##       25       26       27       28       29       30       31       32
```

```
## 78.32362 61.84032 70.16199 83.92475 84.72491 79.92394 85.52507 75.92314
##       33       34       35       36       37       38       39       40
## 76.24321 64.08077 86.48526 83.12458 89.52587 86.48526 78.00356 72.72250
##       41       42       43       44       45       46       47       48
## 81.84433 81.52426 66.96135 83.44465 91.76632 59.91994 88.40564 78.80372
##       49       50       51       52       53       54       55       56
## 85.68510 77.36343 84.08478 99.12779 76.24321 99.60788 99.60788 77.68350
##       57       58       59       60       61       62       63       64
## 67.92154 69.20180 86.00516 82.96455 80.56407 82.48446 84.40484 67.60148
##       65       66       67       68       69       70       71       72
## 93.52667 90.32603 60.07997 85.04497 55.27901 79.60388 75.44305 79.60388
##       73       74       75       76       77       78       79       80
## 82.16439 78.16359 91.92635 74.96295 76.56327 67.12138 86.48526 84.40484
##       81       82       83       84       85       86       87       88
## 88.56567 90.80612 72.08237 97.68750 90.64609 98.32763 75.92314 75.76311
##       89       90       91       92       93       94       95       96
## 62.80052 66.16119 84.40484 72.24241 65.04096 80.24401 73.20260 69.52186
##       97       98       99      100      101      102      103      104
## 83.60468 76.56327 78.64369 84.08478 80.56407 90.00596 83.44465 90.48606
##      105      106      107      108      109      110      111      112
## 74.80292 95.12699 76.40324 74.48285 62.80052 76.40324 65.84112 85.20500
##      113      114      115      116      117      118      119      120
## 84.08478 74.96295 80.72410 84.88494 75.92314 76.56327 83.44465 79.28382
##      121      122      123      124      125      126      127      128
## 64.72090 76.88334 87.92555 92.40644 95.12699 72.24241 81.20420 62.64048
##      129      130      131      132      133      134      135      136
## 76.40324 72.72250 83.76471 78.48366 81.36423 97.04737 68.72170 75.28302
##      137      138      139      140      141      142      143      144
## 88.72571 96.88734 77.52346 78.48366 98.80772 88.88574 71.92234 81.68430
##      145      146      147
## 70.80212 90.96615 71.12218
```

```
plot(y = E.mu16,x = dat$weighted_total,xlab = "Weighted Total", ylab = "E(wei
ghted total|mu_16)")
abline(fit2 <-lm(E.mu16~dat$weighted_total),col=2)
```




```r
# X1,...X6
summary(fit1)

##
## Call:
## lm(formula = E.X16 ~ dat$weighted_total)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8705 -3.1491 -0.1103  3.1633 12.3762
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        11.27850    2.38820   4.723 5.44e-06 ***
## dat$weighted_total  0.86184    0.02905  29.664  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.388 on 145 degrees of freedom
## Multiple R-squared:  0.8585, Adjusted R-squared:  0.8576
## F-statistic:   880 on 1 and 145 DF,  p-value: < 2.2e-16
```

From the summary, I find that Multiple R-squared is 0.8585 and Adjusted R-squared is 0.8576 and the std error for each estimator are 2.38820 and 0.02905

```r
# mu_16
summary(fit2)
```

```
##
## Call:
## lm(formula = E.mu16 ~ dat$weighted_total)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2586  -3.7334   0.4084   4.2571  14.3024
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          27.24899    2.90906   9.367   <2e-16 ***
## dat$weighted_total    0.65482    0.03539  18.503   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.346 on 145 degrees of freedom
## Multiple R-squared:  0.7025, Adjusted R-squared:  0.7004
## F-statistic: 342.4 on 1 and 145 DF,  p-value: < 2.2e-16
```

From the summary, I find that Multiple R-squared is 0.7025 and Adjusted R-squared is 7004. They are both smaller than values in the previous model, suggesting that the model using $X_1,\ldots,X_6$ fits better with greater accuracy.

Also, the std error for each estimator are 2.90906 and 0.03539, which are both bigger than the values in the previous model.

So, I think it is better to use $X_1,\ldots,X_6$ to predict the weighted total than to use the student's average over the first 6 homework.

multivariate normal

I assume that WT | X_16 is multi-normally distributed.

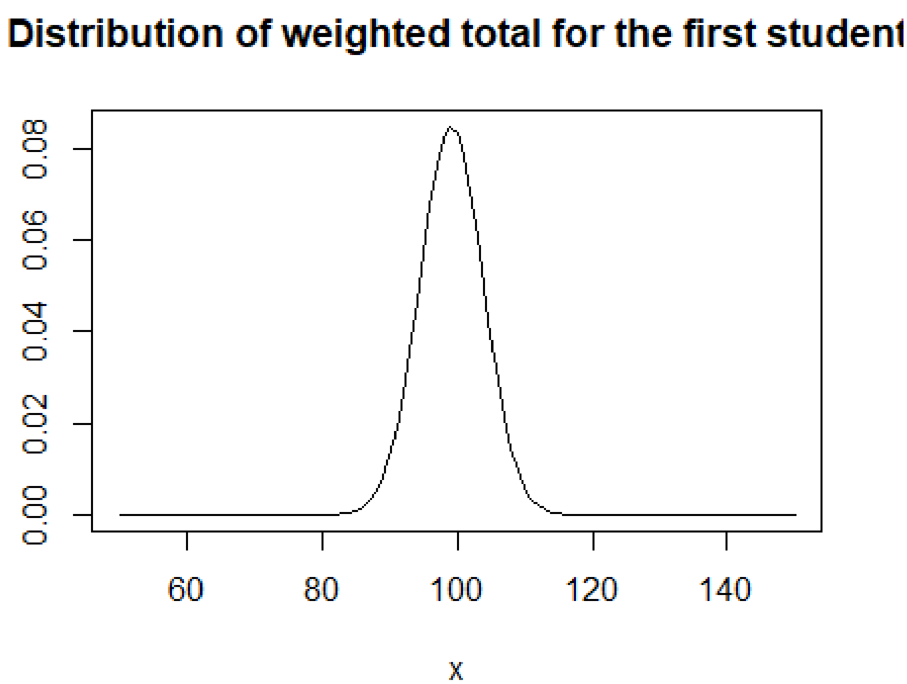
$$X \sim \text{MVN}(\mu, \Sigma)$$

$$\text{Weighted Total}|X_{16} \sim N\left(\mu_w + \Sigma_{12}\Sigma_{22}^{-1}(X_{16} - \mu_{16}), \Sigma_{11} - \Sigma_{12}^T\Sigma_{22}^{-1}\Sigma_{12}\right)$$

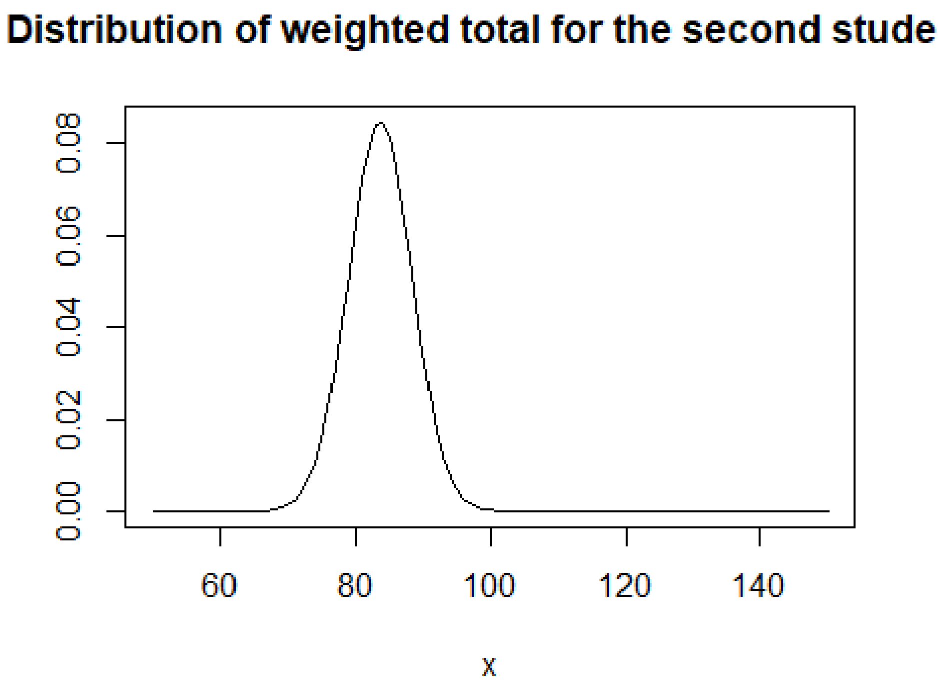
where μ_w is the mean of Weighted Total, μ_{16} is the mean vector of X_1-X_6, $Sigma_{11}$ is the variance of Weighted total,Σ_{12} is the covariance between Weighted Total and X_1-X_6, and Σ_{22} is the covariance matrix of X_1-X_6

```r
mu_hat <- colMeans(dat)
Sigma_hat <- cov(dat)
# fit the model and estimate the parameters
mu_w <- mu_hat[16]
mu_16 <- mu_hat[1:6]
Sigma11 <- Sigma_hat[16,16]
Sigma12 <- Sigma_hat[16, 1:6]
Sigma22 <- Sigma_hat[1:6,1:6]
```

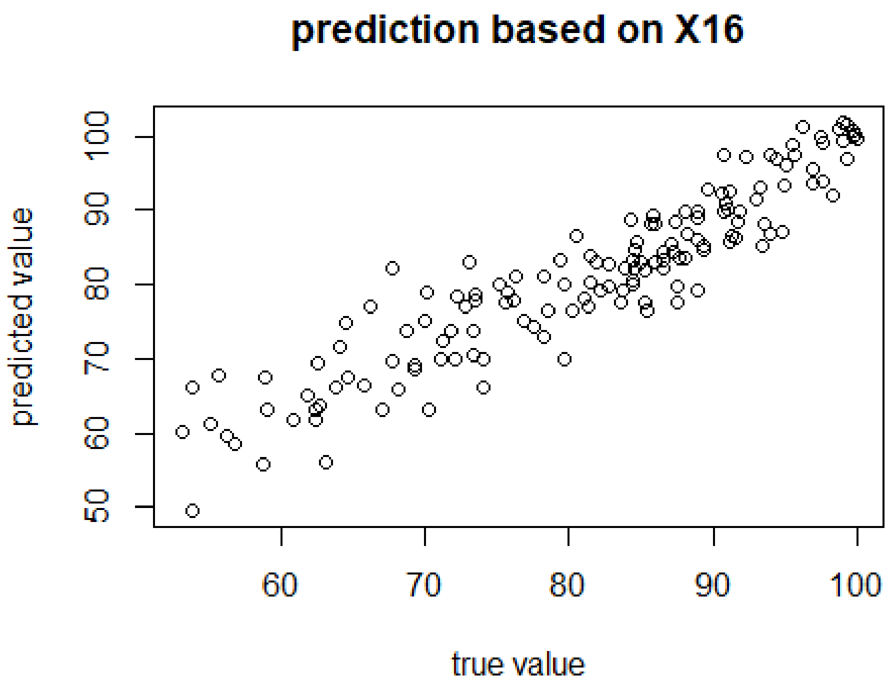
```r
# get the distribution
WeightedTotalDistribution <- function(X16){
    mu <- mu_w + Sigma12 %*% solve(Sigma22) %*% t(X16-mu_16)
    sigma2 <- Sigma11 - t(Sigma12) %*% solve(Sigma22) %*% Sigma12
    return(function(x) {dnorm(x,mu, sqrt(sigma2))})
}


curve(WeightedTotalDistribution(dat[1,1:6])(x),
      xlim=c(50,150),
      main="Distribution of weighted total for the first student",
      ylab = "")
```



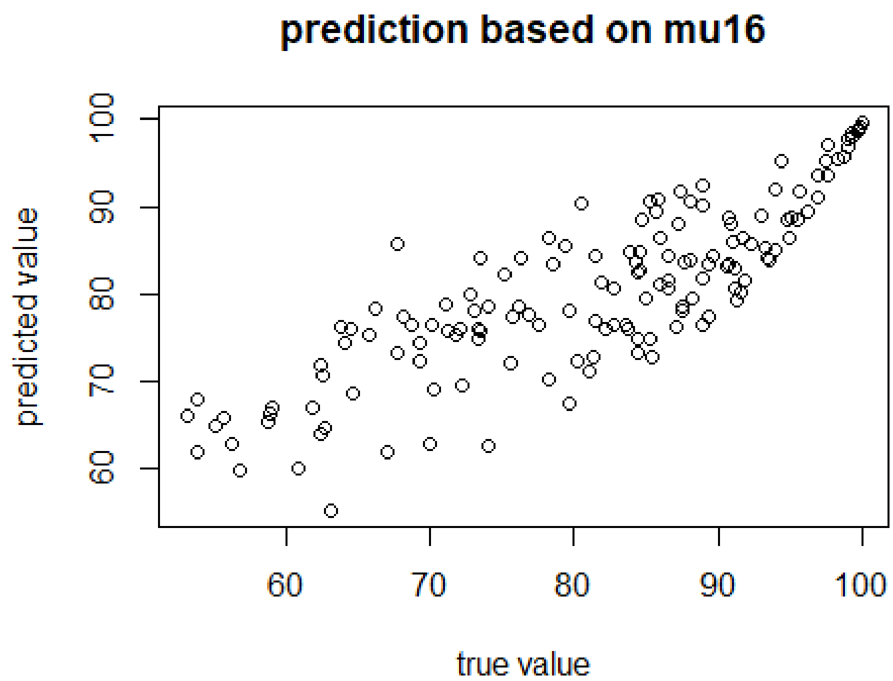

```r
curve(WeightedTotalDistribution(dat[2,1:6])(x),
    xlim=c(50,150),
    main="Distribution of weighted total for the second student",
    ylab = "")
```




```r
predict_w <- function(X16){
    return(mu_w + Sigma12 %*% solve(Sigma22) %*% (X16-mu_16))
}
pred_X16 <- apply(dat[,1:6],1,predict_w)
plot(dat$weighted_total, pred_X16,
    main="prediction based on X16",
    xlab="true value", ylab="predicted value")
```




```
MSE1 <- mean((dat$weighted_total-pred_X16)^2)
MSE1

## [1] 21.93324

mu16 <- dat[,1:6]
mu16[,1:6] <- rowMeans(mu16)
pred_mu16 <- apply(mu16,1,predict_w)
plot(dat$weighted_total, pred_mu16,
    main="prediction based on mu16",
    xlab="true value", ylab="predicted value")
```




```r
MSE2 <- mean((dat$weighted_total-pred_mu16)^2)
MSE2

## [1] 47.29528
```

By comparing the MSE of each model, I think $E(WT|X_{16})$ is better since its MSE is samller.