

Homework 5

Steve Harms

October 8, 2017

Exercise 1

a)

```
#read in the dataset
genes <- data.frame(read.table(file = "http://maitra.public.iastate.edu/stat579/datasets/diurnaldata.csv", sep = ",", header = TRUE))

dim(genes)
```

```
## [1] 22810 23
```

```
names(genes) <- c("Probe", "0h", "1h", "2h", "4h", "8h", "12h", "13h", "14h", "16h",
, "20h", "24h",
, "0h.", "1h.", "2h.", "4h.", "8h.", "12h.", "13h.", "14h.", "16h.",
, "20h.", "24h.")
```

b)

```
#Create an array
trial1 <- as.matrix(cbind(genes[,2:12]), nrow = 22810, ncol = 11, byrow = T)
trial2 <- as.matrix(cbind(genes[,13:23]), nrow = 22810, ncol = 11, byrow = T)
genearray <- array( data = c(trial1, trial2), dim = c(22810, 11, 2))

#means for each probe across the 2 measurements
means <- as.matrix(x = apply(genearray, MARGIN = c(1,2), FUN = mean), nrow = 22810
)
dim(means)
```

```
## [1] 22810 11
```

c)

i)

```
#calculate mean of means
means1 <- apply(means, MARGIN = 1, FUN = mean)
totals <- data.frame(genes$Probe, as.numeric(means1)); names(totals) <- c("gene",
"average")
#check to see if we calculated correctly
head(totals)
```

```
##           gene    average
## 1  AFFX-r2-P1-cre-5_at 3069.8545
## 2  AFFX-r2-P1-cre-3_at 3645.4227
## 3 AFFX-r2-Ec-bioD-5_at  723.3045
## 4 AFFX-r2-Ec-bioD-3_at  829.3182
## 5 AFFX-r2-Ec-bioC-5_at  184.3591
## 6 AFFX-r2-Ec-bioC-3_at  223.9227
```

ii)

```
#replicated matrix of means of means
repped <- matrix(data = means1, nrow = 22810, ncol = 11)
#remove mean effect from averages calculated in part (b)
meaneffect <- means - repped
```

iii)

```
##iii
#standard deviation of each row
stdevs <- apply(X = means, MARGIN = 1, FUN = sd)
#scaled measurements are the standardized genes
ssss <- matrix(data = stdevs, nrow = 22810, ncol = 11)
scaled <- meaneffect/ ssss
```

d)

```
#read in data
measures <- read.table(file = "http://maitra.public.iastate.edu/stat579/datasets/mi
cromeans.dat", sep = "", header = F)
#standardize as above
meansm <- apply(measures, MARGIN = 1, FUN = mean)
#replicated matrix of means of means
reppedm <- matrix(data = meansm, nrow = 20, ncol = 11)
#remove mean effect from averages calculated in part (b)
meaneffectm <- measures - reppedm
#standard deviation of each row
stdevsm <- apply(X = measures, MARGIN = 1, FUN = sd)
#scaled measurements are the standardized genes
scaledm <- as.matrix(meaneffectm/ matrix(data = stdevsm, nrow = 20, ncol = 11), nr
ow = 20, ncol = 11)
```

e)

```
#set up arrays of replicated data sets, check dimensions
arrayg <- array(data = scaled, dim = c(22810, 11, 20))
arraym <- aperm(array(data = c(scaledm), dim = c(20, 11, 22810)), perm = c(3, 2, 1))
dim(arraym); dim(arrayg)
```

```
## [1] 22810    11    20
```

```
## [1] 22810    11    20
```

```
#combine them into a 4-D array
arrayc <- array(data = c(arrayg, arraym), dim = c(22810, 11, 20, 2))
#calculate euclidean distance for each point
#first, a function since dist() is not good
euclid <- function(x) {
  eu <- (x[,1]-x[,2])^2
  sum <- sum(eu)
  return(sqrt(sum))
}

#apply the function to each row/3rd dimension to get the appropriate result
edist <- apply(arrayc, MARGIN = c(1, 3), FUN = euclid)
dim(edist)
```

```
## [1] 22810    20
```

```
head(edist)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 5.417885 4.261081 5.207925 3.125414 4.914939 6.096483 4.562500
## [2,] 5.516634 4.434301 5.165336 3.071993 5.058892 6.162220 4.510031
## [3,] 5.792417 4.489249 5.056531 2.773029 5.173668 6.135268 4.342259
## [4,] 5.421806 4.921538 4.593702 2.728476 5.642030 6.192825 3.824113
## [5,] 5.125586 4.153161 5.022716 2.747196 5.116426 6.081829 4.196495
## [6,] 5.490807 4.494354 4.921923 2.645045 5.374061 6.174676 4.142269
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 5.011322 5.401339 3.423868 2.711460 2.586746 3.553620 4.287767
## [2,] 5.124380 5.389201 3.452634 2.376768 2.455553 3.376037 4.145981
## [3,] 5.223888 5.332790 3.712574 1.900702 2.409075 3.417172 4.349113
## [4,] 5.063302 5.601996 4.238931 2.104583 3.089493 3.674684 4.364259
## [5,] 4.637203 5.614488 3.804447 2.991776 3.055701 3.972082 4.680032
## [6,] 4.985243 5.572431 3.840632 2.122786 2.716344 3.600823 4.331514
##           [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## [1,] 3.455173 4.443603 3.807057 4.268661 5.032314 5.147629
## [2,] 3.272239 4.622061 3.531036 4.082220 5.080642 5.343634
## [3,] 3.370305 5.114404 3.007830 3.613390 5.160581 5.622553
## [4,] 2.427275 5.268009 2.870164 3.203122 4.676630 5.686295
## [5,] 3.187997 4.404006 4.082567 4.212892 4.607245 4.952873
## [6,] 2.795264 4.944761 3.358742 3.655613 4.871261 5.486299
```

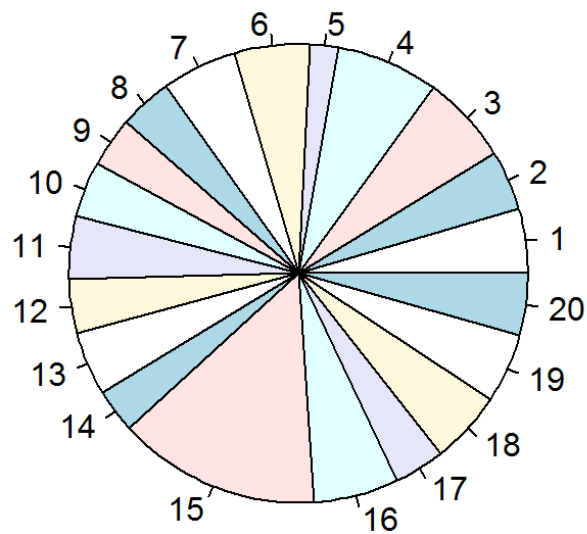
```
#find the index of the minimum distance in each row
min.idx <- apply(edist, MARGIN = 1, FUN=which.min)
head(min.idx)
```

```
## [1] 12 11 11 11 4 11
```

```
#frequency table
table(min.idx)
```

```
## min.idx
##      1      2      3      4      5      6      7      8      9     10     11     12     13     14     15
## 1043   959 1424 1641   449 1213 1221   838   807   891 1008   868 1033   709 3246
##      16      17      18      19      20
## 1361   823 1168 1109   999
```

```
#pie chart of the frequencies
pie(table(min.idx), labels = c(as.character(1:20)))
```



Exercise 2

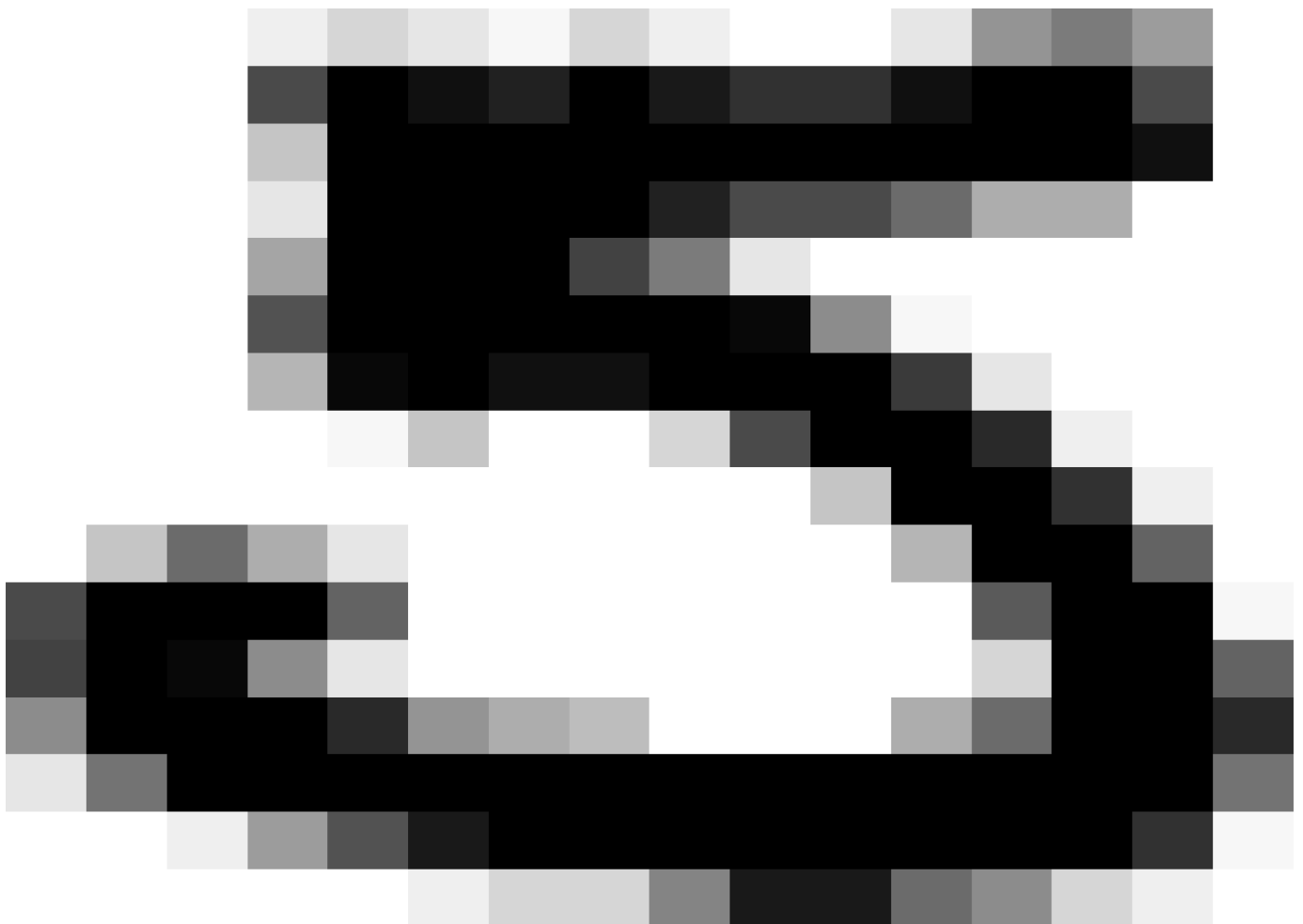
a)

```
#read in the data
ziptrain1 <- as.matrix(read.table(file = "ziptrain.dat", header = F, sep = ""))
#store as an appropriate array
ziptrain2 <- array(data = t(ziptrain1), dim = c(16,16,2000))
```

b)

i)

```
#resize plot area
par(mar = rep(0.05, 4))
#plot the image
image(z=ziptrain2[,16:1,2], col = rev(gray(0:31/31)), axes = F )
```



ii)


```
##c#
#Read in the digits index file
digits <- read.table(file = "zipdigit.dat", header = F, sep = "")

#create index matrix of the 10 digits, I made a function to do it with a loop (probably not the easiest)
dig <- 0:9
indexing <- function(v) {
  indexmat <- matrix(ncol = 10, nrow= max(table(v)))
  for (y in 0:9) {
    indexmat[,y+1] <- c(which(v == y), rep(NA, times = max(table(v)) - length(which(v==y))))
  }
  return(indexmat)
}
#apply the function to our digits data to get indices
indexes <- indexing(digits)

#Create looping functions for means and standard deviations (again, probably not smart to use loops)
means <- function(input, index){
  meanarray <- array(dim = c(16,16,10))
  for (j in 0:9){
    meanarray[,j+1] <- apply(input[,c(index[,j+1])], MARGIN = c(1,2), FUN = mean, na.rm = TRUE)
  }
  return(meanarray)
}

stds <- function(input, index){
  stdarray <- array(dim = c(16,16,10))
  for (j in 0:9){
    stdarray[,j+1] <- apply(input[,c(index[,j+1])], MARGIN = c(1,2), FUN = sd, na.rm = TRUE)
  }
  return(stdarray)
}

#apply the function to the digits
digmeans <- means(ziptrain2, indexes)
stdarray <- stds(ziptrain2, indexes)

#plot the stds images (should be 1 for each digit). Problem did not ask for mean images so I didn't plot them
par(mfrow = c(3,4))
apply(stdarray, MARGIN = 3, FUN = pics)
```

```
## NULL
```



d) I used a lot of functions for this one.


```

#First, a function to remove mean effect from each digit based on index
rm.means <- function(input, index, means){
  newarray <- array(dim = c(16,16,2000))
  for (j in 1:2000){
    digindex <- index[j,1]
    newarray[,j] <- input[,j] - means[,digindex+1]
  }
  return(newarray)
}

#apply this function to our image matrices
mean.removed <- rm.means(ziptrain2, digits, digmeans)
records <- matrix(mean.removed, nrow = 2000, ncol = 256, byrow = T)

#Now a function for the svd and diagonalization for k eigenvalues
decompose <- function(input, k){
  decomp <- svd(input)
  Dk <- diag(c(c(decomp$d[1:k]), rep(0, times = 256-k)))
  Y <- decomp$u%*%Dk%*%t(decomp$v)
  return(Y)
}

#Then, a function to add back the mean effect
addback <- function(original, index, means){
  newarray1 <- array(dim = c(16,16,2000))
  z<- array(data = t(original), dim = c(16,16,2000))
  for (j in 1:2000){
    digindex <- index[j,1]
    newarray1[,j] <- z[,j] + means[,digindex+1]
  }
  return(newarray1)
}

#Now we see what the results look like
#First with k=25 eigenvalues

new25 <- decompose(records, k= 25)
image25 <-addback(new25,digits, digmeans)

par(mfrow = c(40,50), pin=c(6.5,5.2))
apply(image25, MARGIN = 3, FUN = pics)

```

65473031017011774801427487374136741377454274137748
63208664087820982008120833082208144898467019708046
80030809038010290665920919127109080791384435128544
68448640239848985680226841027102471092704808727132
73221710228542279702710260276082711017764462911903
11810316117553650688000199570001601986019868009685
449408643121212680213021313820279000068977999129453
8571801403532010666090260559617376132060521613666
0902602560902605216054664736181609026043860561820
609020180160622617616047313860902601701606220921609
02618216191360902617004626180146992971028070297102
8234282122865833737282172827200938365830328601281
0282652826284328332811282221222116270224012346230
60930230088300833024030215302743080930030230247811
11213721088037130851685708911980003200112003759402
97418213002213021090914601720017060213801970018410
13640260101830910498072006509006300090700934406900
70806600918654060970075808330066009070093300912006
97075423567188600180200048280600139971040185023853
45891073815296306010746913232048338666149700677405
98009109848449550482322698480810720850065376693576
5919646854468009688036811485027857188523711893190
11997719714191197141972600066322463164631466547390
01900957006270801700940296408054101724079413934254
19170141531409144301431014311414724194250192501425
01411391470321232120221268214621342140921701121228
30366303268035608493030263034430327199661960319713
19519061980419713672172798670627341725037251272203
72032374063814138134370753813838117372029607370838
05019899199681972619880403197111971119717404176011
70131757317011175731736017532175421706617609360177
63757316915734011705617311757317306170217268174071
73331701117055197201972600119804197031971198501970
79700198039851989101961906702277110019726001019805
19897989914201802171401972600100540105345064010590
10564108870288203770062260603806790087660912306103
06087060870409604963492604751536314014106049049301
42009700197207971029721018642193301933518649181011
59189011870419101191321910119101192551910417124191
41190131400314130141011903110110319010103190131901

```
## NULL
```

```
#Then with k=50 eigenvalues
new50 <- decompose(records, k=50)
image50 <- addback(new50,digits, digmeans)

par(mfrow = c(40,50), pin=c(6.5,5.2))
apply(image50, MARGIN = 3, FUN = pics)
```

65473031017011774801427487374136741377434274137748
63208662087820902208120833082208144898967619708046
80030809038012290665920919117109080791304435128544
68448640239868935680226841027102271092704808727132
73222710228542278702710260276082711017764462911903
1181031611755365068800001495700001601986019868009685
4494086431212126802130233380279000068917949129453
8571801463532010666090260559617776182060321617666
090260256090260521605466473618-1609026043560561820
609026018016052261761604731826090261701606220927609
02618216191360902619004626180146992971028070297102
82342821228658287378221728272009288365830328601281
02826528262843282328112822621222116270224012345230
60930230088300833024030815302743080930030230247811
11213721038037130851485708911980005200112003759402
97818213002211021090914601720019060213801970018410
136402601018309104980728006509006300090700934406900
7086660091865406060700758002300066009070093300912006
970754235671886001802020482806021399711040185023853
45891073815246266010746912232048538666149700697405
9800290914884955048232698480810720850065370692572
5919646854468009688036811485027857188522711893190
11997719714191197141972602066222463164631466547390
01900457006270801700940296408054101724079412284254
14170141531409144301421014311414724194230142501425
01411391470321232120221268212021242140421701121228
30366303269035603493030263034430327199661980319713
19519861980419713672177798670627241725037251272203
720323740388138134380753813838117372029607370838
05019899199681972019880403197111971119717404176011
701319737317011175731736017837175421706617604360177
63737316915734011705617311737317306170217268174071
73331701117055197201972600119804197031971198501970
79700198039851989101961906702277110019726001019805
19897989914201802171101972600100540105345054010590
10564108870288203770062260603866790087060913506103
06087060870409604963492604751536314014106049049301
42009720197207971029721018642143301933518049181011
59189011870419101191321910119101192551910417124191
41190131400319130191011903110110319010109190151901

NULL

```
#And finally with 75 eigenvalues
new75 <- decompose(records, k=75)
image75 <- addback(new75,digits, digmeans)
par(mfrow = c(40,50), pin=c(6.5,5.2))
apply(image75, MARGIN = 3, FUN = pics)
```



```

65473031017011774801487487374136741377434274137748
63208666087820902208120833282208144898967619708046
80030809038012290665920919117109080791204435168544
68448640239868935680226841027102271092704808727132
73222710228541278702710260276082711017764462911903
1181031611755365068800001495700001601986019868009685
4494086481212126802130233380279000068917949129453
8571801463532010666090260559617776182060521618666
0902602560902605216054664736181609026043560561820
60902618016052261761604731826090261701606220927609
02618216191360902619004626180146992971028070297102
82342821228658287372821728272009288362830328601281
0282652826284328232811282222116270224012345230
609302300888300833024030815302743080930030230247811
112137121038037130851185708911980005200112003759402
97418213002211021090214601720019060213801970018410
136402601018309104980720065090063000907009304406900
708666091865460607007580023006600907009300912006
970754235571886001802020482806021329711040185023853
45891073815246206010746912232048538666149700697405
98002909148849550482322698480810720850065370692572
59196546854468009688036811485027857188522711893190
11997719714191197141972602066222463164631466547390
01900457006270809700940296405054101724079412284254
14170141531409144301421014311414724194230142501425
0141139147032123212321268212621262140421701121228
30366303263035603493030263034430327199661980319713
19519861980419713672177798670627241725037251272203
72032274033815138134370753813838117372029607370838
050198991996819720198804031971119711179717404176011
70131737317011175731736017837175421706617604360177
63737316915734011705617311737317306170217268174071
73331701117055197201972600119804197031971198501970
79700198039851989101961906702277110019726001019805
19897989914201802171101972600100540105345054010590
10564108870288203770062260603806790087060913506103
06087060870409604963492004751536314014106049049301
42009720197207971029721018642143301933518049181011
59189011870419101191321910119101192551910419124191
41190131400319130141011903110110319010109190151901

```

```
## NULL
```

For lower values of k , we remove more of the noise (but also possibly more information), which results in less clear images. This is because only the pixels with the smallest deviations are magnified by the singular value multiplications. As k gets closer to 256, the images get closer to the original. We are basically compressing the images into smaller data storage by choosing small k . Most of the images don't have much deviation so the differences aren't as visible as we would hope. To see the real difference, try $k < 10$ vs $k > 200$.

Exercise 3

a)

```
states <- data.frame(state.x77, state.region)
attach(states)
```

```
## The following object is masked from package:datasets:
##
## state.region
```

```
#a
#using tapply
percapinc.t <- tapply(Income, INDEX = state.region, FUN = mean)
percapinc.t
```

```
##      Northeast      South North Central      West
##      4570.222      4011.938      4611.083      4702.615
```

```
#using aggregate
percapinc.a <- aggregate(Income, by = list(state.region), FUN = mean)
percapinc.a
```

```
##      Group.1      x
## 1      Northeast 4570.222
## 2      South    4011.938
## 3 North Central 4611.083
## 4      West     4702.615
```

b)

```
#b
#using tapply
maxilliterate.t <- tapply(Illiteracy, INDEX = state.division, FUN=max )
maxilliterate.t
```

```
##      New England      Middle Atlantic      South Atlantic
##      1.3      1.4      2.3
## East South Central West South Central East North Central
##      2.4      2.8      0.9
## West North Central      Mountain      Pacific
##      0.8      2.2      1.9
```

```
#using aggregate
maxilliterate.a <- aggregate(Illiteracy, by = list(state.division), FUN=max)
maxilliterate.a
```

```
##      Group.1      x
## 1      New England 1.3
## 2      Middle Atlantic 1.4
## 3      South Atlantic 2.3
## 4 East South Central 2.4
## 5 West South Central 2.8
## 6 East North Central 0.9
## 7 West North Central 0.8
## 8      Mountain 2.2
## 9      Pacific 1.9
```

c)

```
#c
#The easiest way would be to just use a frequency table
table(state.region)
```

```
## state.region
##      Northeast      South North Central      West
##           9          16          12          13
```

```
#using tapply
regioncount.t <- tapply(state.name, INDEX = state.region, FUN= length )
regioncount.t
```

```
##      Northeast      South North Central      West
##           9          16          12          13
```

```
#using aggregate
regioncount.a <- aggregate(state.name, by = list(state.region), FUN = length)
regioncount.a
```

```
##      Group.1  x
## 1 Northeast  9
## 2 South     16
## 3 North Central 12
## 4 West      13
```

d)

```
#d
#using tapply
names.t <- tapply(state.name, INDEX = state.division, FUN=list)
names.t
```

```
## $`New England`
## [1] "Connecticut" "Maine" "Massachusetts" "New Hampshire"
## [5] "Rhode Island" "Vermont"
##
## $`Middle Atlantic`
## [1] "New Jersey" "New York" "Pennsylvania"
##
## $`South Atlantic`
## [1] "Delaware" "Florida" "Georgia" "Maryland"
## [5] "North Carolina" "South Carolina" "Virginia" "West Virginia"
##
## $`East South Central`
## [1] "Alabama" "Kentucky" "Mississippi" "Tennessee"
##
## $`West South Central`
## [1] "Arkansas" "Louisiana" "Oklahoma" "Texas"
##
## $`East North Central`
## [1] "Illinois" "Indiana" "Michigan" "Ohio" "Wisconsin"
##
## $`West North Central`
## [1] "Iowa" "Kansas" "Minnesota" "Missouri"
## [5] "Nebraska" "North Dakota" "South Dakota"
##
## $Mountain
## [1] "Arizona" "Colorado" "Idaho" "Montana" "Nevada"
## [6] "New Mexico" "Utah" "Wyoming"
##
## $Pacific
## [1] "Alaska" "California" "Hawaii" "Oregon" "Washington"
```

```
#using aggregate
names.a <- aggregate(state.name, by=list(state.division), FUN = list)
names.a <- aggregate(formula = state.name~state.division, data=states, FUN = list)
names.a
```

```
##      state.division
## 1      New England
## 2      Middle Atlantic
## 3      South Atlantic
## 4 East South Central
## 5 West South Central
## 6 East North Central
## 7 West North Central
## 8      Mountain
## 9      Pacific
##
state.name
## 1      Connecticut, Maine, Massachusetts, New Hampshire, Rhode
Island, Vermont
## 2      New Jersey, New Y
ork, Pennsylvania
## 3 Delaware, Florida, Georgia, Maryland, North Carolina, South Carolina, Virginia
, West Virginia
## 4      Alabama, Kentucky, Missi
ssippi, Tennessee
## 5      Arkansas, Louisiana
, Oklahoma, Texas
## 6      Illinois, Indiana, Michigan,
Ohio, Wisconsin
## 7      Iowa, Kansas, Minnesota, Missouri, Nebraska, North Dako
ta, South Dakota
## 8      Arizona, Colorado, Idaho, Montana, Nevada, New Mexic
o, Utah, Wyoming
## 9      Alaska, California, Hawaii, Or
egon, Washington
```

e)

```
#e
#first create our variable
state.size <- cut(x = state.x77[, "Population"], breaks = c(0, 2000, 10000, Inf),
                 labels = c("Small", "Medium", "Large"))
#using tapply
pops.t <- tapply(Population, INDEX = list(state.size, state.region), FUN=median)
pops.t
```

```
##      Northeast South North Central      West
## Small      871.5  1189      681    779.5
## Medium     5814.0  3806      4589   2412.5
## Large      14968.0 12237      10966 21198.0
```

```
#using aggregate
pops.a <- aggregate(Population, by=list(state.size, state.region), FUN = median)
pops.a
```



```
##      Group.1      Group.2      x
## 1      Small      Northeast  871.5
## 2      Medium      Northeast 5814.0
## 3      Large      Northeast 14968.0
## 4      Small              South 1189.0
## 5      Medium              South 3806.0
## 6      Large              South 12237.0
## 7      Small North Central   681.0
## 8      Medium North Central 4589.0
## 9      Large North Central 10966.0
## 10     Small              West   779.5
## 11     Medium              West 2412.5
## 12     Large              West 21198.0
```

Exercise 4

a)

```
cars <- data.frame(mtcars)
##a
#using aggregate
mada <- apply(cars, MARGIN = 2, FUN = mad)
mada
```

```
##      mpg      cyl      disp      hp      drat      wt
## 5.4114900 2.9652000 140.4763500 77.0952000 0.7042350 0.7672455
##      qsec      vs      am      gear      carb
## 1.4158830 0.0000000 0.0000000 1.4826000 1.4826000
```

b)

```
##b
#using apply twice and a sweep function to clear it out
#we get the same result as above
meds <- apply(cars, 2, median)
swept <- sweep(cars, 2, meds)
mad2 <- 1.4826 * apply(abs(swept), 2, median)
mad2
```

```
##      mpg      cyl      disp      hp      drat      wt
## 5.4114900 2.9652000 140.4763500 77.0952000 0.7042350 0.7672455
##      qsec      vs      am      gear      carb
## 1.4158830 0.0000000 0.0000000 1.4826000 1.4826000
```