Assignment 9

edelsonc

10/3/2016

This exercise will combine the skill learned throughout the previous 6 weeks in order to look at the overall quality and structure of a hepatitis data set obtained from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Hepatitis)

Downloading and Reading Into R

The data is downloaded off of the web with the `bash`'s `wget` command and saved locally. Both the data set (saved as `hepatitis.data`) and the annotations (`hepatitis.names`) were saved locally.

`hepatitis.data` is a csv, and so can be quickly read into R using the `read.csv` utility

```
# No header on the data, so header set to false
df_hep <- read.csv("hepatitis.data", header=FALSE)
str(df_hep)
```

```
## 'data.frame':    155 obs. of  20 variables:
##  $ V1 : int  2 2 2 2 2 2 2 1 2 2 2 ...
##  $ V2 : int  30 50 78 31 34 34 51 23 39 30 ...
##  $ V3 : int  2 1 1 1 1 1 1 1 1 1 ...
##  $ V4 : Factor w/ 3 levels "?","1","2": 2 2 3 1 3 3 2 3 3 3 ...
##  $ V5 : int  2 2 2 1 2 2 2 2 2 2 ...
##  $ V6 : Factor w/ 3 levels "?","1","2": 3 2 2 3 3 3 2 3 2 3 ...
##  $ V7 : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ V8 : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ V9 : Factor w/ 3 levels "?","1","2": 2 2 3 3 3 3 3 3 3 3 ...
##  $ V10: Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 2 3 ...
##  $ V11: Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ V12: Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ V13: Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ V14: Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ V15: Factor w/ 35 levels "?","0.30","0.40",..: 9 8 6 6 9 8 1 9 6 9 ...
##  $ V16: Factor w/ 84 levels "?","100","102",..: 78 19 84 51 1 83 1 1 1 1 ...
##  $ V17: Factor w/ 85 levels "?","100","101",..: 26 54 48 62 31 44 1 1 60 8 ...
##  $ V18: Factor w/ 30 levels "?","2.1","2.2",..: 18 13 18 18 18 18 1 1 22 17 ...
##  $ V19: Factor w/ 45 levels "?","0","100",..: 1 1 1 42 1 38 1 1 1 1 ...
##  $ V20: int  1 1 1 1 1 1 1 1 1 1 ...
```

We now have a dataframe. However, there are no names for the columns. Those are located in the other file, `hepatitis.names`

Extracting Names

Looking throught the `hepatitis.names` we see that all of the column names are perceded by at least four spaces, and then either a one or two digit number followed by a period. Since we can quicky see this, regex is a good tool to extract this information. Combining `egrep` with `awk` can ten achieve the outcome we desire.

```
header=$(egrep '^ {4}( |[0-9])[0-9]+\.' hepatitis.names | awk '{print $2}' | sed 's/:$//')
echo "$header" > hepatitis.headers
```

We can check to ensure this worked:

```
cat hepatitis.headers
```

```
## Class
## AGE
## SEX
## STEROID
## ANTIVIRALS
## FATIGUE
## MALAISE
## ANOREXIA
## LIVER
## LIVER
## SPLEEN
## SPIDERS
## ASCITES
## VARICES
## BILIRUBIN
## ALK
## SGOT
## ALBUMIN
## PROTIME
## HISTOLOGY
```

Now `hepatitis.headers` can be read into r and set as the names for the df_hep

```
headers <- read.csv("hepatitis.headers", header=FALSE)
colnames(df_hep) <- headers$V1
str(df_hep)
```

```
## 'data.frame':    155 obs. of  20 variables:
##  $ Class     : int  2 2 2 2 2 2 1 2 2 2 ...
##  $ AGE       : int  30 50 78 31 34 34 51 23 39 30 ...
##  $ SEX       : int  2 1 1 1 1 1 1 1 1 1 ...
##  $ STEROID   : Factor w/ 3 levels "?","1","2": 2 2 3 1 3 3 2 3 3 3 ...
##  $ ANTIVIRALS: int  2 2 2 1 2 2 2 2 2 2 ...
##  $ FATIGUE   : Factor w/ 3 levels "?","1","2": 3 2 2 3 3 3 2 3 2 3 ...
##  $ MALAISE   : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ ANOREXIA  : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ LIVER     : Factor w/ 3 levels "?","1","2": 2 2 3 3 3 3 3 3 3 3 ...
##  $ LIVER     : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 2 3 ...
##  $ SPLEEN    : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ SPIDERS   : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 2 3 3 3 ...
##  $ ASCITES   : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ VARICES   : Factor w/ 3 levels "?","1","2": 3 3 3 3 3 3 3 3 3 3 ...
##  $ BILIRUBIN : Factor w/ 35 levels "?","0.30","0.40",..: 9 8 6 6 9 8 1 9 6 9 ...
##  $ ALK       : Factor w/ 84 levels "?","100","102",..: 78 19 84 51 1 83 1 1 1 1 ...
##  $ SGOT      : Factor w/ 85 levels "?","100","101",..: 26 54 48 62 31 44 1 1 60 8 ...
##  $ ALBUMIN   : Factor w/ 30 levels "?","2.1","2.2",..: 18 13 18 18 18 18 1 1 22 17 ...
##  $ PROTIME   : Factor w/ 45 levels "?","0","100",..: 1 1 1 42 1 38 1 1 1 1 ...
##  $ HISTOLOGY : int  1 1 1 1 1 1 1 1 1 1 ...
```

There, all pretty! Now we may want to write this to a new csv, that way we don't have to go throught the previous steps again. This is a simple command with `R`, `write.csv`

```
write.csv(df_hep, "hepatitis.data.headers")
```

We can check that it wrote properly using `bash` again

```
head hepatitis.data.headers
```

```
## "","Class","AGE","SEX","STEROID","ANTIVIRALS","FATIGUE","MALAISE","ANOREXIA","LIVER","LIVER","SPLEEN
## "1",2,30,2,"1",2,"2","2","2","1","2","2","2","2","2","1.00","85","18","4.0","?",1
## "2",2,50,1,"1",2,"1","2","2","1","2","2","2","2","2","0.90","135","42","3.5","?",1
## "3",2,78,1,"2",2,"1","2","2","2","2","2","2","2","2","0.70","96","32","4.0","?",1
## "4",2,31,1,"?",1,"2","2","2","2","2","2","2","2","2","0.70","46","52","4.0","80",1
## "5",2,34,1,"2",2,"2","2","2","2","2","2","2","2","2","1.00","?","200","4.0","?",1
## "6",2,34,1,"2",2,"2","2","2","2","2","2","2","2","2","0.90","95","28","4.0","75",1
## "7",1,51,1,"1",2,"1","2","1","2","2","1","1","2","2","?","?","?","?","?",1
## "8",2,23,1,"2",2,"2","2","2","2","2","2","2","2","2","1.00","?","?","?","?",1
## "9",2,39,1,"2",2,"1","2","2","2","1","2","2","2","2","0.70","?","48","4.4","?",1
```

Perfect! Now we can move forward knowing that we have a safe copy of the data.

Complete Rows

We can quickly check if there are any incomplete rows using the `complete.cases` command,

```
# number of rows
nrow(df_hep)
```

```
## [1] 155
```

```
# complete cases
sum(complete.cases(df_hep))
```

```
## [1] 155
```

Great, there are no missing values!…or at least no default NA's. In reality, the dataset uses the "?" as the missing value indicator. So, we just need to swap these for NA, and we'll be on our way!

```
df_hep[df_hep == "?"] <- NA
sum(complete.cases(df_hep))
```

```
## [1] 80
```

Almost half of the rows are imcomplete.

Subsetting on Columns

We wish to subset a few of the columns, converting known numeric values

```
# subset on the appropriate names
df_sub1 <- df_hep[, c("AGE", "SEX", "BILIRUBIN", "ALK", "SGOT", "ALBUMIN")]

# change the last four factors to numeric. Apply is used to speed up
df_sub1[c(3:6)] <- apply(df_sub1[c(3:6)], 2, as.numeric)
str(df_sub1)
```

```
## 'data.frame':    155 obs. of  6 variables:
##  $ AGE      : int  30 50 78 31 34 34 51 23 39 30 ...
##  $ SEX      : int  2 1 1 1 1 1 1 1 1 1 ...
##  $ BILIRUBIN: num  1 0.9 0.7 0.7 1 0.9 NA 1 0.7 1 ...
##  $ ALK      : num  85 135 96 46 NA 95 NA NA NA NA ...
##  $ SGOT     : num  18 42 32 52 200 28 NA NA 48 120 ...
##  $ ALBUMIN  : num  4 3.5 4 4 4 4 NA NA 4.4 3.9 ...
```

Now we can investigate the nunber of complete rows for this new dataframe

```
sum(complete.cases(df_sub1))
```

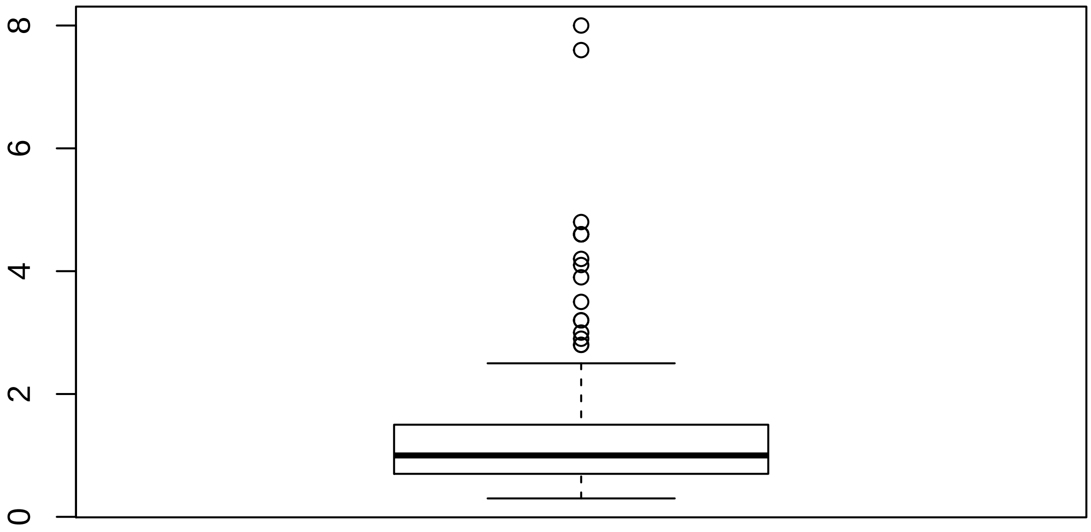
```
## [1] 120
```

Much closer to the total number of rows.

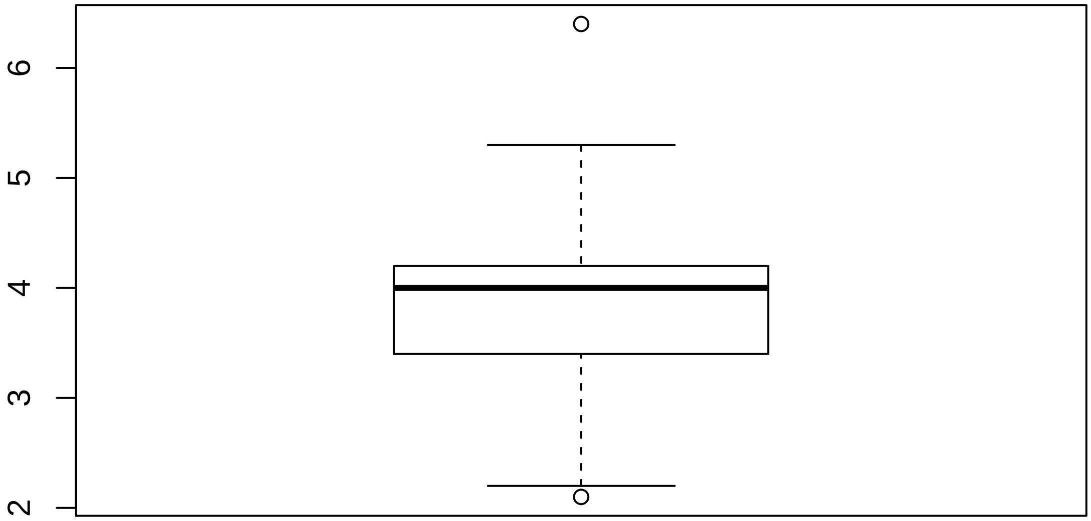
Outliers

The quickest way to vet data is by using a plotting tool. The human eye is extremely good at discriminiating between visual differences, so a tool like a boxplot is perfect.

```
boxplot(df_sub1$BILIRUBIN)
```



```
boxplot(df_sub1$ALBUMIN)
```



Bilirubin seems to have quite a number of outliers, while albumin only has two of so.

Binning

The data for age is binned nicely into decades

```
binned_age <- cut(df_sub1$AGE, breaks=seq(0,90,10), include.lowest = TRUE)
str(binned_age)
```

```
##  Factor w/ 9 levels "[0,10]","(10,20]",..: 3 5 8 4 4 4 6 3 4 3 ...
```

```
# add onto dataframe for use later
df_sub1 <- cbind(df_sub1, binned_age)
```

Aggregate

Here the data will be aggragated as a function of bined age and sex

```
aggregate(df_sub1, by=list(df_sub1$binned_age, df_sub1$SEX), mean, na.rm = TRUE)
```

```
##    Group.1 Group.2      AGE SEX BILIRUBIN      ALK      SGOT  ALBUMIN
## 1   [0,10]       1  7.00000   1 0.7000000 256.0000  25.00000 4.200000
## 2  (10,20]       1 20.00000   1 0.9500000 124.5000 135.00000 3.450000
## 3  (20,30]       1 26.75000   1 1.2458333 105.1875  77.73913 4.218182
## 4  (30,40]       1 35.83333   1 1.2086957 100.1000  77.48936 3.847727
## 5  (40,50]       1 45.68750   1 1.8580645 102.2593  97.78125 3.578571
## 6  (50,60]       1 54.13636   1 1.9150000 102.3750  93.04762 3.700000
## 7  (60,70]       1 63.75000   1 1.0428571 104.0000 111.00000 3.700000
## 8  (70,80]       1 75.00000   1 0.8500000 105.5000  42.00000 3.700000
## 9  (10,20]       2 20.00000   2 2.3000000 150.0000  68.00000 3.900000
## 10 (20,30]       2 26.20000   2 0.9200000 100.8000 101.00000 3.920000
## 11 (30,40]       2 34.00000   2 0.6500000  50.0000  24.00000 4.050000
## 12 (40,50]       2 45.33333   2 0.8666667 132.0000  81.66667 4.200000
## 13 (50,60]       2 55.50000   2 1.4500000 128.0000  37.00000 3.400000
## 14 (60,70]       2 66.00000   2 2.0000000 146.3333 120.33333 3.400000
##    binned_age
## 1          NA
## 2          NA
## 3          NA
## 4          NA
## 5          NA
## 6          NA
## 7          NA
## 8          NA
## 9          NA
## 10         NA
## 11         NA
## 12         NA
## 13         NA
## 14         NA
```

R made this simple with the use of the builtin aggregate function. The only thing to tell it was the data, which factors to aggregate by, and what to do with NA values.

Ordering the data

R has a nice order function. This function takes a vector or list as an input, and returns the index a given item should be self to as an output.

```
df_sub1 <- df_sub1[order(df_sub1$BILIRUBIN),]
```

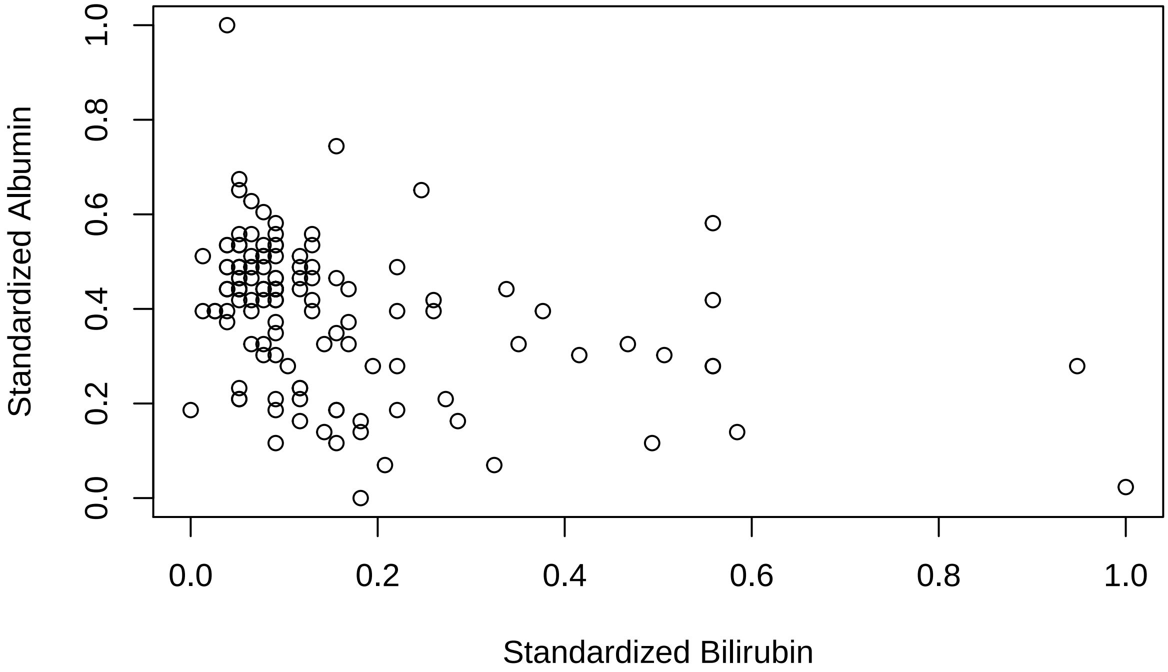
Once again, easy peasy.

Standardizing

It is fairly easy to standardize data as well.

```
standardize <- function(x){
  # standardizes a single vector x
  x_min <- min(x, na.rm = TRUE)
  x_max <- max(x, na.rm = TRUE)
  xrange <- x_max - x_min
  x_stand <- (x - x_min) / xrange
  return(x_stand)
}

plot(standardize(df_sub1$BILIRUBIN), standardize(df_sub1$ALBUMIN),
      xlab = "Standardized Bilirubin",ylab = "Standardized Albumin")
```



PCA

Even principle compnent analysis (PCA) is a simple matter with R

```r
# subset on last four numeric columns and remove non complete cases
hep_numeric <- df_sub1[c(3:6)]
hep_numeric <- hep_numeric[complete.cases(hep_numeric),]

# perform pca
princomp(hep_numeric)
```

```
## Call:
## princomp(x = hep_numeric)
##
## Standard deviations:
##     Comp.1     Comp.2     Comp.3     Comp.4
## 81.8700900 49.9311803  0.9133541  0.5196922
##
##  4  variables and  120 observations.
```

As we can see 61.45% of the varience is accounted for by the first principle component.

Joining Data

The merge function allows the joining of multiple tables

```r
df_sub2 <- df_hep[,c("AGE", "SEX", "STEROID", "ANTIVIRALS")]
df_sub3 <- df_hep[,c("AGE", "SEX", "BILIRUBIN", "ALK", "SGOT", "ALBUMIN")]
```

```r
# get only complete cases of df_sub3 for joining on
df_sub3 <- df_sub3[complete.cases(df_sub3),]

# merge the two dataframes
df_merge <- merge(df_sub2, df_sub3, sort=FALSE)
str(df_merge)
```

```
## 'data.frame':    481 obs. of  8 variables:
##  $ AGE       : int  30 50 50 50 50 50 50 50 50 50 ...
##  $ SEX       : int  2 1 1 1 1 1 1 1 1 1 ...
##  $ STEROID   : Factor w/ 3 levels "?","1","2": 2 3 3 3 3 3 2 2 2 2 ...
##  $ ANTIVIRALS: int  2 2 2 2 2 2 2 2 2 2 ...
##  $ BILIRUBIN : Factor w/ 35 levels "?","0.30","0.40",..: 9 8 9 9 24 14 8 9 9 24 ...
##  $ ALK       : Factor w/ 84 levels "?","100","102",..: 78 19 78 21 25 2 19 78 21 25 ...
##  $ SGOT      : Factor w/ 85 levels "?","100","101",..: 26 54 77 80 77 2 54 77 80 77 ...
##  $ ALBUMIN   : Factor w/ 30 levels "?","2.1","2.2",..: 18 13 18 17 4 29 13 18 17 4 ...
```

```r
dim(df_merge)
```

```
## [1] 481   8
```

This has joined on all matching columns and includes the ones that have matches. Many different joins are possible using R's merge, but this is the simplest.