Lab 2 Multivariate Data

# Set up R session

## Download packages

Install and load packages

install.packages("MVA")  
install.packages("psych")  
install.packages("Hmisc")  
install.packages("vegan")  
install.packages("StatMatch")  
install.packages("MASS")  
install.packages("raster")  
install.packages("cluster")

library(MVA)  
library(psych)  
library(Hmisc)  
library(vegan)  
library(StatMatch)  
library(MASS)  
library(raster)  
library(cluster)

## Importing Data

We will be using the USairpollution and the usAir\_mod.csv data sets again.

The MVA US air pollution data set:

usAir <- USairpollution

The modified USairpollution data set from your working directory is a csv file:

usAir\_mod <- read.csv("Data/lab\_2/usAir\_mod.csv", row = 1, header = TRUE)

# Data screening

Your first move when conducting a multivariate analysis (or any analysis) is to screen the data. You are looking for data errors, missing data, and outliers that may influence your analysis.

## Data errors

One way to check for data errors is to examine the summary statistics for your data set.

First look at the summary statistics for usAir:

describeBy(usAir)

Do you see any unrealistic values?

Now look at the summary statistics for usAir\_mod:

describeBy(usAir\_mod)

Look at the max for temperature. It will be easier to look for data errors when it is your own data.

## Missing Data

When you have missing entries in your data sheet, R replaces them with “NA”. You can check if you have any missing variables in *usAir\_mod*:

describe(usAir\_mod)

Describe provides some of the same information as describeBy, but importantly shows you which variables have missing values.

We talked about three methods for dealing with missing values in lecture; **Complete Case, Available Case, Imputation**. We will look at **complete case and imputation** for now.

**Complete Case** involves the removal of samples (in this case cities) with missing data:

usAir\_mod[complete.cases(usAir\_mod), ]

**Imputation** involves filling in missing values with plausible data. Let’s replace NAs with the mean of the variable.

# First, let’s calculate the mean of each variable (column) with the NA  
# removed:  
  
meanz <- colMeans(usAir\_mod, na.rm = T)  
  
# `na.rm=T`, means that you want to remove NAs  
  
# To replace your NAs with the means you just calculated you will use the  
# following function:  
  
naFunc <- function(column) {  
 column[is.na(column)] = round(mean(column, na.rm = TRUE), 2)  
 return(column)  
}  
  
# and “apply” it to the usair\_mod data set  
  
Impute <- apply(usAir\_mod, 2, naFunc)

Check it out and make sure that the NA’s have been replaced.

We will not go into this advanced function too much. However, know that apply allows us to apply a function to all the rows and/or columns in a data frame of matrix. As we spoke about in lecture, there are many types of imputation methods. We can explore further methods for your specific missing data.

# Data transformation

The next step is preparing your data for analysis is transforming the data. Today we will look at the log square root, and arcsine square root transformations.

## Log transformation:

Several common transformation have built-in functions in R. While you can build transformation functions on your own, we will use the ones R has developed today. First, let’s look at a histogram of our first variable, SO2, to determine if transformation is necessary:

Remember, to extract the SO2 column:

usAir$SO2  
  
# or  
  
usAir[, 1]  
  
  
# Next you can simply wrap either of those commands in the histogram  
# function:  
  
hist(usAir$SO2)  
  
# or  
  
hist(usAir[, 1])

To log transform each value in our data frame:

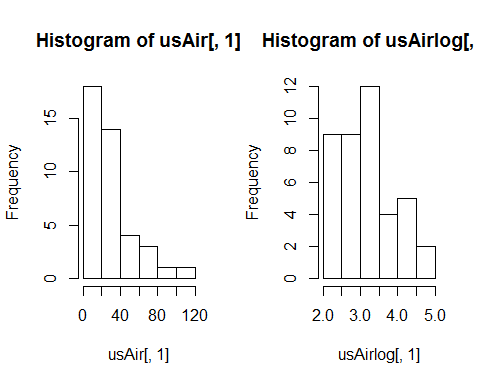
usAirlog <- log1p(usAir)

and the histogram:

hist(usAirlog$SO2)  
  
# or  
  
hist(usAirlog[, 1])

You can compare the histograms side by side using the *par* function followed by *hist*:

par(mfrow = c(1, 2))  
  
hist(usAir[, 1])  
hist(usAirlog[, 1])



Placing 1, 2 in parentheses after the c (which stands for concatenate) in the par function indicates that you want you plots arranged in 1 row and two columns.

Compare histograms for the raw data and the log transformed data for each variable.

Which variable might not need to be log transformed?

## Square root transformation:

To square root transform each value in our data frame:

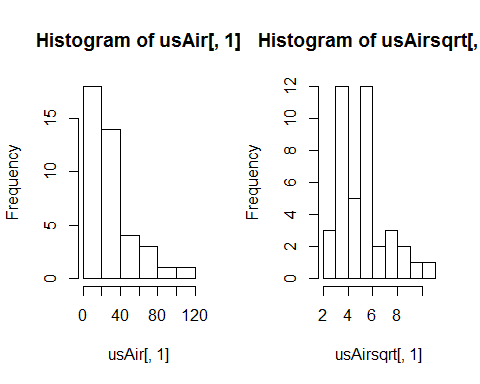
usAirsqrt <- sqrt(usAir)

and the histogram:

hist(usAirsqrt$SO2)  
  
# or  
  
hist(usAirsqrt[, 1])

Compare the histograms side by side using the *par* function followed by *hist*:

par(mfrow = c(1, 2))  
  
hist(usAir[, 1])  
hist(usAirsqrt[, 1])



Compare histograms for the raw data and the log transformed data for each variable…

**Remember that square root transformations are best used on count data.**

## Arcsine square root transformation: = arcsine

If you remember arcsine square root transformations are for percentage data. So, you values for your variable must lie between 0 and 1. None of the variable in usAir are appropriate for this transformation. Let’s draw some random numbers between 0 and 1 so we can use the arcsine square root transformation.

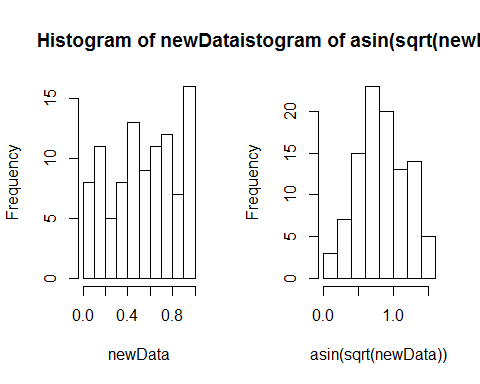
newData <- runif(100, 0, 1)

You just chose 100 random values between 0 and 1. Now let’s transform:

asin(sqrt(newData))

and compare histograms:

par(mfrow = c(1, 2))  
  
hist(newData)  
hist(asin(sqrt(newData)))



# Data standardization

Column standardization adjusts for differences among variables. Focus is on the profile across a sample unit. Row standardization adjusts for differences among sample units. Focus is on the profile within a sample unit (good when variables are measured in the same units, e.g. species). You will more often than not, be using column standardization.

## Coefficient of Variation (cv)

Let’s first see if the air pollution data set needs standardization by calculating the *coefficient of variation* **(cv)** for columns totals. Remember, the **cv** is the ratio of the standard deviation to the mean (σ/μ):

First calculate the column **sums**:

cSums <- colSums(usAir)

Then calculate the **standard deviation** and **mean** for the column sums:

Sdev <- sd(cSums)  
M <- mean(cSums)

Finally, calculate the **cv**:

Cv <- Sdev/M \* 100

You can also use the function cv in the R package *raster*. Our rule of thumb for cv is that if **cv> 50**, data standardization is necessary. What do you think for the USairpollution data?

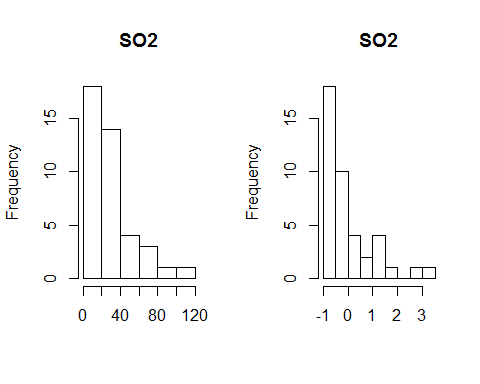
## Z- standardization = (

Your goal here is to equalize the variance for variables measured on different scales. There is a built-in function scale that will do this for you:

scaledData <- scale(usAir)

Let’s look at histograms for the scaled and unscaled data for the first variable, SO2:

par(mfrow = c(1, 2))  
  
hist(usAir[, 1], main = colnames(usAir)[1], xlab = " ")  
hist(scaledData[, 1], main = colnames(usAir)[1], xlab = " ")



Compare the raw and standardized histograms for all of the variables. Are you convinced that the variances are equalized? Just to check, calculate the mean and variance for each of the standardized variables.

**Z standardization is very common in life sciences.**

# Detecting Outliers

Outliers are recorder values of measurements or observations that are outside the range of the bulk of the data. Outliers can inflate variance and lead to erroneous conclusions.

## Univariate outliers

One way to deal with outliers in multivariate data is to examine each variable separately. You will standardize your data into standard deviation units (z –standardization) and look for values that fall outside of three standard deviations.

First the z-standardization:

scaledData <- scale(usAir)

Next we will create histograms to look for values > than 3 sd. However, this time we will use the par function to look at all seven histograms at once.

par(mfrow = c(2, 4))  
hist(scaledData[, 1], main = colnames(usAir)[1], xlab = " ")  
hist(scaledData[, 2], main = colnames(usAir)[2], xlab = " ")  
hist(scaledData[, 3], main = colnames(usAir)[3], xlab = " ")  
hist(scaledData[, 4], main = colnames(usAir)[4], xlab = " ")  
hist(scaledData[, 5], main = colnames(usAir)[5], xlab = " ")  
hist(scaledData[, 6], main = colnames(usAir)[6], xlab = " ")  
hist(scaledData[, 7], main = colnames(usAir)[7], xlab = " ")

Finally, you can identify the outlier(s) for each variable:

scaledData[, 1][scaledData[, 1] > 3]  
scaledData[, 2][scaledData[, 2] > 3]  
scaledData[, 3][scaledData[, 3] > 3]  
scaledData[, 4][scaledData[, 4] > 3]  
scaledData[, 5][scaledData[, 5] > 3]  
scaledData[, 6][scaledData[, 6] > 3]  
scaledData[, 7][scaledData[, 7] > 3]

Alternatively, you could use the *mapply* function, less typing!

For the histogram function (hist):

par(mfrow = c(2, 4))  
mapply(hist, as.data.frame(usAir), main = colnames(usAir), xlab = " ")

Here is a new function for detecting outliers called out.

out <- function(x) {  
 lier <- x[abs(x) > 3]  
 return(lier)  
}

Let’s apply that function:

apply(scaledData, 2, out)

## $SO2  
## Chicago   
## 3.406199   
##   
## $temp  
## named numeric(0)  
##   
## $manu  
## Chicago   
## 5.112752   
##   
## $popul  
## Chicago   
## 4.766583   
##   
## $wind  
## named numeric(0)  
##   
## $precip  
## named numeric(0)  
##   
## $predays  
## named numeric(0)

Do you detect any outliers? For which variables?

## Multivariate outliers

**We will come back to this…**

# Distance and Dissimilarity

As we know from lecture, multivariate data with *p* variables are visually represented by a collection of points forming a data cloud in *p*-dimensional space. The shape, clumping, and dispersion of the data cloud contains information we seek to describe. Several distance and dissimilarity measures are used to calculate the distance between data points.

## Euclidean Distance:

**Euclidean** distance is one of the most used distance measures. It is normally preceded by column standardization (e.g. z standardization). Let calculate Euclidean distance for the US air pollution data set. You will use the function vegdist from the *vegan* (vegetation analysis) package. Look up vegdist to see the different indices available in this package.

`?`(vegdist)

First, z standardization:

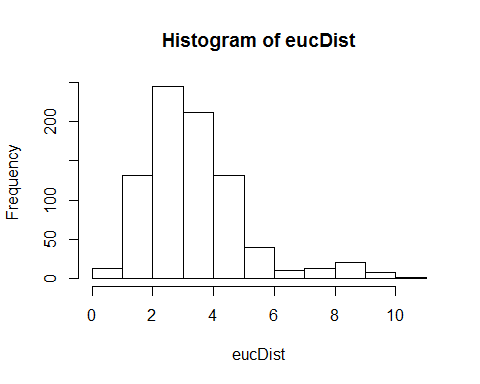
scaledData <- scale(usAir)

Then calculate distance:

eucDist <- vegdist(scaledData, "euclidean")

Let’s look at a histogram of distances:

hist(eucDist)



What does this frequency distribution tell you about pollution conditions across these 41 cities?

**Euclidean Distance can be weird**. Let look at the data matrix below:

We want to determine how similar these farms are in there production of Strawberries, Peaches, and Raspberries.

Fruit <- rbind(c(1, 0, 1, 1), c(2, 1, 0, 0), c(3, 0, 4, 4))  
colnames(Fruit) <- c("Farm", "Strawaberry", "Peach", "Rasberry")  
Fruit

## Farm Strawaberry Peach Rasberry  
## [1,] 1 0 1 1  
## [2,] 2 1 0 0  
## [3,] 3 0 4 4

Calculating Euclidean distance on these data:

eucDist<- vegdist(Fruit[,-1], “euclidean”)

*eucDist* gives us this distance matrix (R gives you the triangular matrix without the diagonal):

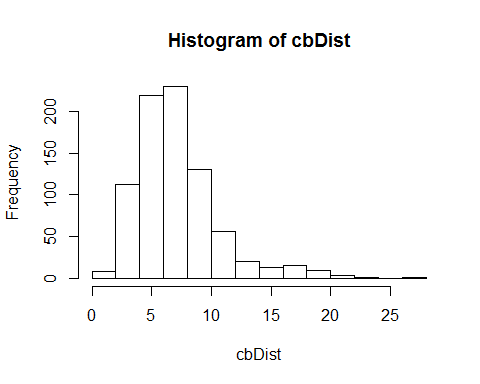
The distance between farms 1 and 2, which grow none of the same fruits:

Is **less** (i.e., these farms are more similar in there fruit production) than farms 1 and 3, which grow the same fruit:

Euclidean distance is not a jack-of-all-trades and is not appropriate for all data sets. Our next distance metric, Manhattan distance would also rank Farms 1 and 2 more similar than 1 and 3.

## City-block (Manhattan) distance

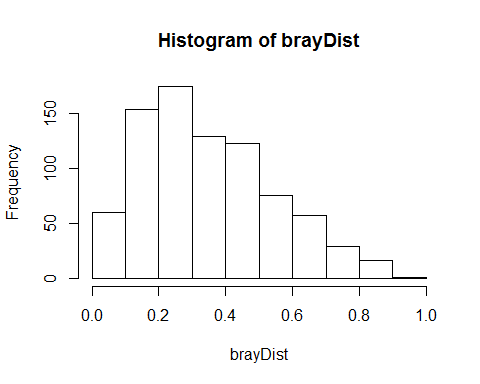
cbDist <- vegdist(scaledData, "manhattan")  
  
# Let’s look at a histogram of distances:  
  
hist(cbDist)



How does this distribution compare to Euclidean distance?

## Bray-Curtis dissimilarity

brayDist <- vegdist(usAir, "bray")  
  
# Histogram:  
  
hist(brayDist)



**Let’s quickly look at our fruit farm data with bray-curtis**:

brayFruit <- vegdist(Fruit[, -1], "bray")  
brayFruit

## 1 2  
## 2 1.0   
## 3 0.6 1.0

That makes more sense! Farms 1 and 2 (and 2 and 3) are at maximum dissimilarity and farms 1, 3 are more similar.

## Gower dissimilarity

The **Gower** dissimilarity is very useful because it allows for mixed variables and missing values. Gower is available in the *raster* package using function daisy. Use it on the *usAir\_mod* data set because it has missing values.

daisy(usAir\_mod, metric = "gower")

**Back to multivariate outliers!**

Your goal here is to examine deviations of the sample average distances to other samples. We will use **bray-curtis** distance:

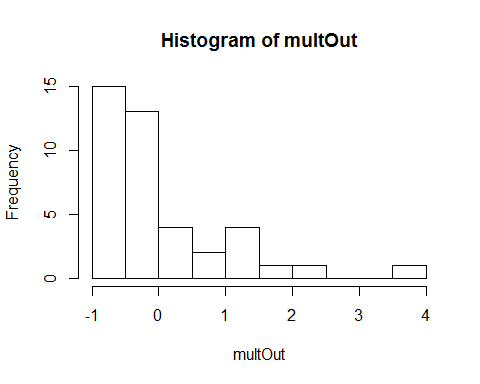
brayDist <- vegdist(usAir, "bray")

Next, calculate column means. These column means represent the average dissimilarity of each city to all other cities. You want to know if any cities are on average more than 3 standard deviation units (z scores). To achieve this, z-transform the averages:

multOut <- scale(colMeans(as.matrix(brayDist)))

Plot a histogram and look for observations >3 sd units:

hist(multOut)



You can find the cities that are outliers with:

multOut[multOut > 3, ]

## Chicago   
## 3.678029

Another possibility is to determine which observation are > 3 standard deviations from the mean. Using bray-curtis distance again:

Calculate column means:

colBray <- colMeans(as.matrix(brayDist))

Calculate the mean of the column means:

mBray <- mean(colBray)

Calculate the standard deviation:

stdBray <- sd(colBray)

… 3 standard deviations

threeSD <- stdBray \* 3 + mBray

plot a histogram and look for observations >3 sd:

hist(colBray)

Find the outliers:

colBray[colBray > threeSD]

## Chicago   
## 0.7113063

**NOW, RUN THROUGH THE ABOVE EXERCISES WITH YOUR OWN DATA!**