Lab 2 Multivariate Data Screening

# Set up R session

## Install and load packages.

Note that for a RMarkdown document to knit, you need to comment out these ‘install.packages’ lines of code before knitting. This is because you cannot knit when your document needs to connect to an external web server. Right now these lines of code are commented out.

# install.packages('mvnormtest') install.packages('MVN')  
# 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(mvnormtest)  
library(MVN)  
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 for these examples.

The MVA US air pollution data set:

usAir <- USairpollution

The modified USairpollution data set from your working directory is a csv file. Note you will need to modify you working directory if it is different.

usAir\_mod <- read.csv("G:/Shared drives/MultivariateStatistics/Data/LabData/Lab1/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)

***Question 1: Do you see any unrealistic values? (5 pts) Note please answer all questions with points related to them.***

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)

The *describe* function provides some of the same information as *describeBy*, but importantly shows you which variables have missing values.

We talked about two methods for dealing with missing values in lecture; **Complete Case and 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 out the new Impute data object 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 perform a function on 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.

# Multivariate Normal Distribution

Many of the analyses we will do in this course have an assumption of multivariate normality. While there are many tests of multivariate normality, they tend to be overly conservative. If we strictly followed these tests, we may never run a multivariate analysis with ecological or agricultural data. Here we will look at two multivariate tests of normality.

## Shapiro-Wilks test

Shapiro-Wilks tests if the distribution of the observed data differs from multivariate normal distribution. So, we are looking for p-values > 0.05.

mshapiro.test(t(usAir))

##   
## Shapiro-Wilk normality test  
##   
## data: Z  
## W = 0.59549, p-value = 2.025e-09

## Mardia test

Mardia’s test looks at multivariate extensions of Skewness and Kurtosis. In both cases, we are looking for p-values > 0.05 to show that our data do not deviate from the expectations of multivariate normal Skewness and Kurtosis. For the observed data to be considered multivariate normal, p-values from both the Skewness and Kurtosis statistics must be > 0.05. This function also tests for univariate normality of residuals using the Shapiro-Wilk statistic.

mvn(usAir, mvnTest = "mardia")

## $multivariateNormality  
## Test Statistic p value Result  
## 1 Mardia Skewness 226.612731693166 4.82491336101954e-15 NO  
## 2 Mardia Kurtosis 3.97754689564216 6.96298933924311e-05 NO  
## 3 MVN <NA> <NA> NO  
##   
## $univariateNormality  
## Test Variable Statistic p value Normality  
## 1 Anderson-Darling SO2 2.3841 <0.001 NO   
## 2 Anderson-Darling temp 0.9633 0.0136 NO   
## 3 Anderson-Darling manu 4.2925 <0.001 NO   
## 4 Anderson-Darling popul 3.4292 <0.001 NO   
## 5 Anderson-Darling wind 0.3784 0.3911 YES   
## 6 Anderson-Darling precip 0.8742 0.0228 NO   
## 7 Anderson-Darling predays 0.5175 0.1783 YES   
##   
## $Descriptives  
## n Mean Std.Dev Median Min Max 25th 75th Skew  
## SO2 41 30.048780 23.472272 26.00 8.00 110.0 13.00 35.00 1.584112608  
## temp 41 55.763415 7.227716 54.60 43.50 75.5 50.60 59.30 0.822975684  
## manu 41 463.097561 563.473948 347.00 35.00 3344.0 181.00 462.00 3.484603302  
## popul 41 608.609756 579.113023 515.00 71.00 3369.0 299.00 717.00 2.941257977  
## wind 41 9.443902 1.428644 9.30 6.00 12.7 8.70 10.60 0.002675131  
## precip 41 36.769024 11.771550 38.74 7.05 59.8 30.96 43.11 -0.692518149  
## predays 41 113.902439 26.506419 115.00 36.00 166.0 103.00 128.00 -0.550092270  
## Kurtosis  
## SO2 2.25541093  
## temp 0.09066032  
## manu 14.33200058  
## popul 10.57605759  
## wind 0.06015407  
## precip 0.49578021  
## predays 0.72033969

# 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 transformations 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:

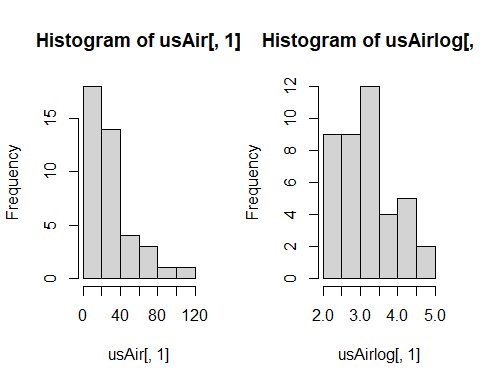
usAirlog <- log1p(usAir)

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. Note this plotting is done in base R as opposed to using the ggplot functions of Tidyverse. It is helpful to know base R and Tidyverse to be able to read and trouble shoot code with a wide range of collaborators. In ggplot this code would be similar to what the *facet* function does.

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

**Question 2: Which variable might not need to be log transformed? (5 pts)**

## Square root transformation:

To square root transform each value in our data frame:

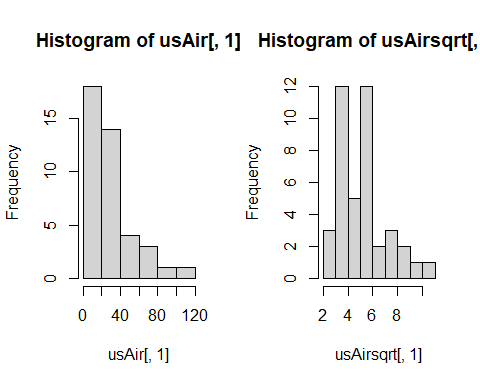
usAirsqrt <- sqrt(usAir)

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, the values for your variable must range between 0 and 1. None of the variables 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.

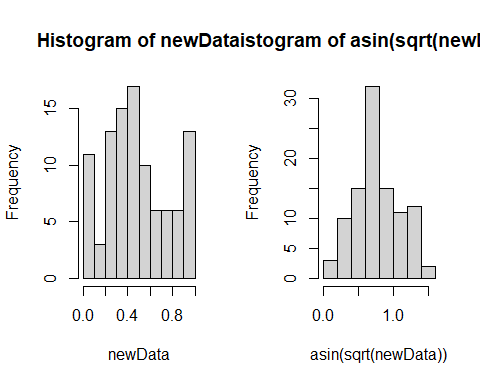
newData <- runif(100, 0, 1)

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. The focus is on the profile across a sample unit. Row standardization adjusts for differences among sample units, wherein the focus is on the profile within a sample unit. Row standardization is 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 column 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

Our rule of thumb for cv is that if **cv> 50**, data standardization is necessary.

***Questin 3: Is standardization necessary for the USairpollution data? (5 pts)***

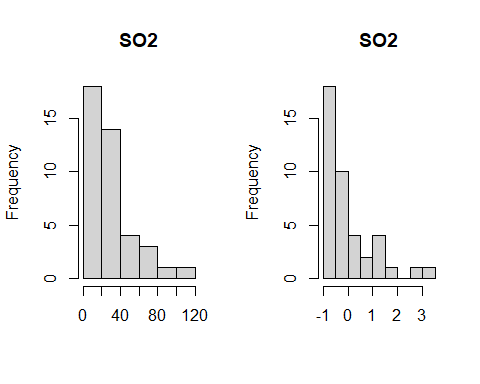
## Z- standardization = (

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.

***Question 4: Are you convinced that the variances are equalized? Just to check, calculate the mean and variance for each of the standardized variables. (10 pts)***

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

# Detecting Outliers

Outliers are recorded 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 fu*nction 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 apply 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)

**Question 5: Do you detect any outliers? For which variables? (5 pts)**

## 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 commonly used distance measures. It is normally preceded by column standardization (e.g. z standardization). Let’s 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:

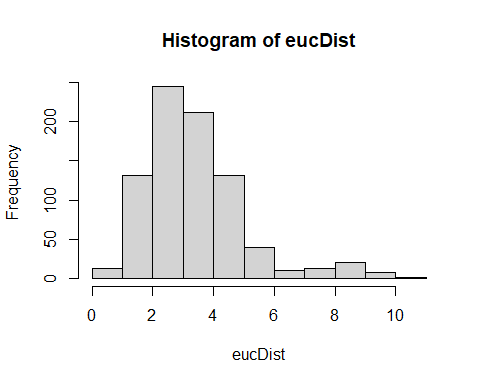
scaledData <- scale(usAir)

Then calculate distance:

eucDist <- vegdist(scaledData, "euclidean")

Let’s look at a histogram of distances:

hist(eucDist)



**Question 6: What does this frequency distribution tell you about pollution conditions across these 41 cities? (5 pts)**

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

We want to determine how similar these farms are in theit 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", "Strawberry", "Peach", "Rasberry")  
Fruit

## Farm Strawberry 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")

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

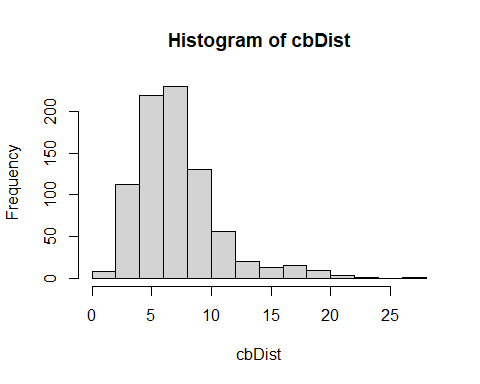
The distance between farms 1 and 2, which grow none of the same fruits:

Is **less** (i.e., these farms are more similar in their 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

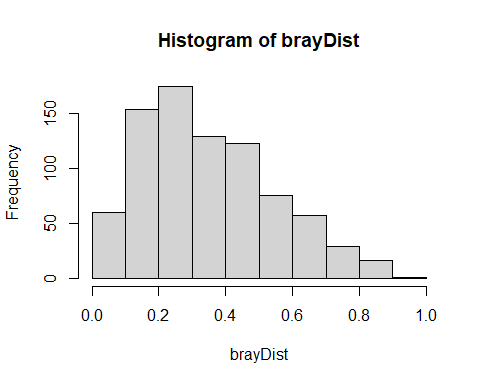
cbDist <- vegdist(scaledData, "manhattan")  
  
# Let’s look at a histogram of distances:  
  
hist(cbDist)



**Question 7: How does this distribution compare to Euclidean distance? (5 pts)**

## 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.

**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:

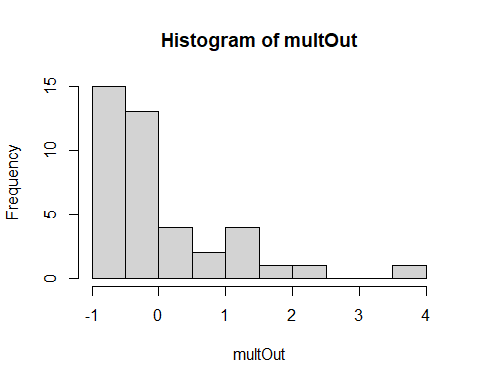
brayDist <- vegdist(usAir, "bray")

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:

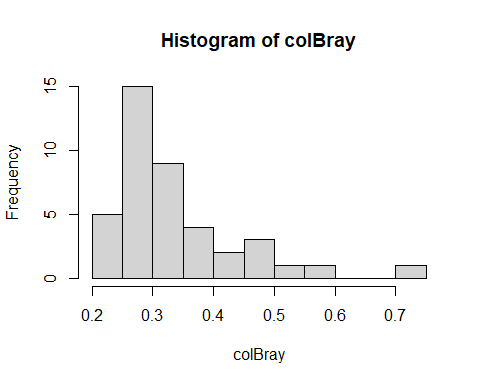
stdBray <- sd(colBray)

… 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

**Question 8: NOW, RUN THROUGH THE ABOVE EXERCISES WITH YOUR OWN DATA! (55 pts)**