**Example Dataset:**

This dataset was collected as part of a student thesis project – and is one of the messiest datasets our lab has worked with ☺. We asked students about their advisors and their stress levels by using lots of scales. I have included the following variables:

* Gender – male / female gender variable, should import as a factor.
* Level – factor variable indicating the level of the student (freshman – Ph.D. student)
* Advising – a yes / no variable that indicated if they had an assigned advisor or not, should import as an integer.
* Q1-Q23 – one of the scales we used to assess advising, which was coded 1-7.

**Helpful hint:**

* At some point, you will mess up the dataset. Therefore, I am going to suggest you create new datasets for each step, which means that if you mess up, you can just back up in your code to the last save point. Think about it like a video game with multiple save spots.

**Accuracy**:

* Create a new dataset called notypos.
  + notypos = *whatever\_you\_imported\_the\_data\_as*
* Use the summary() function to look at factor coding, minimum, and maximum values for each column.
  + summary(*data set name*)
* Fix unfactored (or incorrectly factored) columns using the factor() command.
  + In the LEVELS part – you must put in the numbers or labels that are already present in the dataset, otherwise you will wipe out the column. Therefore, if the column has 1, 2, 3, 4 type c(1,2,3,4).
  + In the LABELS part – you will put in the labels you want to use for those levels. You must have the same number of items in levels and labels.
  + *dataset$column =* factor(*dataset$column,*

levels = c(*stuff*),

labels = c(*stuff*))

* Fix any out of range values.
  + *dataset$column*[ *dataset$column* > *max value* ] = NA
  + *dataset$column*[ *dataset$column* < *min value* ] = NA
  + This code will set the values to missing, but you could also set them to a specific number.
    - For this example ONLY, set the values above 7 to NA and below 1 to be 1.

**Missing data:**

* We are going to create a couple new datasets here by partitioning off participants based on specific rules.
* Calculate the missing values for participants and variables.
  + First, run this function exactly:
    - percentmiss = function(x){ sum(is.na(x)/length(x)) \* 100}
  + Participants:
    - missing = apply(*dataset*, 1, percentmiss).
    - This line will save a variable that calculates the percent missing for each participant.
    - Often you will have participant who just quit in an experiment – while technically that is MNAR data, you are actually dealing with non-completers. We should exclude those people.
    - Additionally, you would want to exclude participants who are missing more than 5% of their data because we can’t replace that data anyway.
    - Create two datasets:
      * replacepeople = subset(*dataset name*, missing <= 5)
      * nopeople = subset(*dataset name*, missing > 5)
    - Sometimes, you will never use the nopeople again, because they do not have enough data. However, if you have one analysis you can use them in but not another, you can hang on to their data to use in the one analysis and exclude them in the second one where they are missing the data (this procedure is called pairwise elimination).
  + Columns:
    - apply(replacepeople, 2, percentmiss)
      * You will get a list of percentages of missing data for each column.
      * Take note of which ones were problems.
      * Take note of the columns you do not want to fill in because they are categorical or demographics.
      * Notice that I used replace people here … I do not want those noncompleters to count against my 5% rule for columns.
    - Create two more mini datasets:
      * replaceall = replacepeople[ , c(*columns to keep*) ]
      * nocolumn = replacepeople[ , c(*columns to exclude*) ]
      * In the c() put the numbers of the columns like this:
        + Just an example:
        + replaceall = replacepeople[ , c(4, 5, 6, 13, 47) ]
      * Don’t forget you can use the head(*dataset name*) to make sure you picked the right columns here.
  + Install the mice package and load the library.
    - Mice is a package that will import data for you based on multiple imputation and expected maximization.
    - Run these two lines to estimate:
      * tempnomiss = mice(replaceall)
      * replaced = complete(tempnomiss, 1)
      * The first line does the estimation, the second line combines the estimation with the original dataset.
  + Combine everything back together as needed.
    - allcolumns = cbind(replaced, nocolumn)
      * Cbind = column bind puts together columns into one dataset.
    - allrows = rbind(allcolumns, nopeople)
      * Rbind = row bind puts together all rows into one dataset.
      * You may not use this half depending on your goals for the analysis.
      * For this example, use allcolumns for the next steps.

**Outliers**

* Create the Mahalanobis scores:
  + Remember that dataset here will be allrows or allcolumns.
  + mahal = mahalanobis(*dataset*[ , -c(*columns that are categorical*)],

colMeans(*dataset*[ , -c(*columns that are categorical*)], na.rm = T),

cov(*dataset*[ , -c(*columns that are categorical*)], use = “pairwise.complete.obs”))

* + That looks really gross, but here’s what’s going on:
    - Mahalanobis(dataset, colMeans(dataset), cov(dataset))
    - Mahal values are calculated using the mean of each column and the covariance matrix (like correlations but unstandardized).
    - So, be sure to fill in the dataset name and excluded the categorical variables.
* Figure out the cut off score for Mahalanobis.
  + We will use the chi-square distribution function.
  + Why chi-square? Because Mahalanobis is distributed with a chi-square distribution (just like how we use the normal distribution to test z-scores because they tend to be normal).
  + cutoff = qchisq(1-.001, ncol(*dataset*[ , -c(*columns that are categorical*)]))
    - qchisq = gives you the cut off score for a chi-square distribution.
    - The arguments are p-value (1 – rule, so p < .001), and *df* (ncol counts the number of columns we are using to calculate Mahalanobis) = number of variables.
  + Remember, if you are asked for *df* on the homework, you should run:
    - ncol(*dataset*[ , -c(*columns that are categorical*)])
  + If you want to see what the cut off score is run:
    - cutoff
  + If you want to see how many outliers there are:
    - summary(mahal < cutoff)
    - True = NOT outlier, False = YES outlier
* Exclude those outliers and create a noout dataset.
  + noout = subset(*dataset name*, mahal < cutoff)

**Additivity:**

* Remember the new dataset we are working with will be noout.
* Save a correlation table.
  + correl = cor(*dataset*[ , -c(*columns that are categorical*)], use = “pairwise.complete.obs”)
  + You can view the correlation table by typing correl.
  + Eeww gross.
* Use the symnum function to see the bad correlations.
  + symnum(correl).
  + The picture you get back will star/mark values based on the size of r (NOT p).
  + So the dot (.) indicates that one of the values is over .3 – it will only show you the bottom half of the triangle, remember that the top half is a repeat.
  + \* and B values are bad!
* Drop or average any columns that are too highly correlated.
* Create a final dataset called final.

**The rest of the assumptions set up:**

* For ANOVA, t-tests, correlation: you will use a *fake* regression analyses – it’s considered fake because it’s not the real analysis, just a way to get the information you need to do data screening.
* For regression based tests: you can run the *real* regression analysis to get the same information. The rules are altered slightly, so make sure you make notes in the regression section on what’s different.
* Create a set of random numbers to compare against.
  + random = rchisq(nrow(no\_out\_final), 7)
  + Rchisq is a function that generates random numbers from the chi square distribution.
  + The arguments are: number of randoms to make, *df*
    - Why 7? Because it seems to work well … you can pick anything above 2 and should get similar results.
  + Nrow tells you the number of rows in your data frame – remember that length of a dataframe tells you how many columns there are, not the length of the columns.
* Run a fake regression.
  + fake = lm(random ~ . , data = final)
  + This function uses the same format as t-tests: DV ~ IV, but we can use the dot (.) to screen all columns (the entire dataset).
  + That will save all the information for the regression for you to use later. You don’t really want to see a summary of this function because it’s fake – you just want to use different parts of it later.
* Create the standardized residuals and scaled fitted values.
  + standardized = rstudent(fake)
  + fitted = scale(fake$fitted.values)

**Normality:**

* You can check individual columns using the moments library with skewness() and kurtosis().
  + skewness(*column name,* na.rm = T), kurtosis(*column name,* na.rm = T)
  + What are these numbers?
    - These values are Z scores, so the cut off score is 3 (which is *p* < .001 for Z scores).
    - You do NOT want scores larger than |3|, which would indicate very extreme skew/kurtosis.
    - Larger sample sizes (*N* > 30) tend to be more normal, and you do not have to worry about skew/kurtosis as much.
* Check out the multivariate histogram.
  + hist(standardized)
  + Look to see that most of the bars are centered over zero between the same numbers on each side (i.e. -1 to 1 or -2 to 2).

**Linearity:**

* Check out a QQ plot.
  + qqnorm(standardized)
  + abline(0,1)
  + See if most of the dots line up on the line.

**Homogeneity/Homoscedasticity:**

* Check out the scatterplot of the residuals.
  + plot(fitted, standardized)
  + abline(0,0)
  + abline(v= 0)
* For homogeneity:
  + Are the dots roughly centered around zero horizontally and vertically? Use the lines on the graph to help you tell.
  + Don’t go too nuts – one or two random dots does not constitute a problem.
* For homoscedasticity:
  + Are the dots roughly blob shaped? Imagine a line drawn around the dots – it should make blob shape and not a megaphone or UFO.

**Results**

Prior to analysis, holistic, content, structure, stance, sentence fluency, diction and conventions within English papers were examined through various SPSS programs for accuracy of data entry, missing values, and fit between their distributions and the assumptions of multivariate analysis. The variables were examined for the 322 participants in the study. The descriptive statistics showed that the means and standard deviations were relatively normal and that the maximum pre-development values for stance, sentence fluency, diction, and convention, and post-development stance, sentence fluency, and diction were incorrect. These values were found to be outside of the used scale and were replaced.

There were seven variables each with one missing value. These were pre-development workshop values for structure, sentence fluency, diction, and convention, and post-development workshop values for structure, stance, sentence fluency, and convention. These missing values were replaced with *mice*. Two multivariate outliers were found using Mahalanobis distance with p < .001. These outliers were deleted, leaving 320 cases. I ran a bivariate correlation to check for multicollinearity and singularity for the pre-workshop and post-workshop holistic variable. I made no change to this variable because it was not too highly correlated. I checked skewness and kurtosis of the variables and found all of them to be normal, so no changes were made. The multivariate normality plot showed that results were normal, but slightly skewed to the right. The normal QQ Plot of regression standardized residuals shows that the variables were linear. The Standardized Regression Scatter plot shows that the results were homogeneic, but were not homoscedastic.