CHAPTER 2

# ADVANCED DATA SCREENING, OUTLIERS, AND MISSING VALUES

Data management and screening are essential steps in analyzing data. Violating assumptions can increase the risk of Type I errors (for discussion see Warner, 2020b). Consequently, data should be scrutinized and violations addressed. This chapter discusses techniques to evaluate variables and identify outliers and missing values. Further discussion of univariate and bivariate data screening is included in Rasco (2020a).

## DATA MANAGEMENT

One of the first steps in managing data is saving and archiving a copy of the original data file. Increasingly, journals and other interested parties require researchers to provide the original data so screening and analytic processes can be repeated by an independent analyst to see if results are reproducible (i.e., consistent with those reported). Consequently, the original data must be maintained throughout the data screening and cleaning process, and researchers should save modified data as new files. For example, a researcher may save several versions of the data using descriptive file names: eligible participants (i.e., individuals who met predetermined inclusion criteria), missing removed, or outliers removed. These designations (bolded) can be added when the modified data are written to a file for storage: `write.csv(DataFrame, "StudyName_`**`MissingRemoved`**`.csv", row.names=FALSE)`. The exact labels depend on the issues the researcher encounters, and date and time labels could be used as an alternative or additional method of tracking the modifications.

If the data do not initially contain participant identifiers, you often need to add an identification variable. The `row.names` function can be used to create an identifier.

```
DataFrame$PartID <- row.names(DataFrame)
```

You may need to transform or recode variables. Typically, it is best to create a new variable and retain the original in case there is an issue with the code to transform or recode the variable. That is to say, if you make a mistake in creating the modified variable and replace the original variable, you cannot easily undo your mistake. Examples of how to recode a continuous variable to a grouping variable (i.e., factor) and how to transform a variable (e.g., `log10`) are provided in Rasco (2020a).

As you create updated variables, it is important to use descriptive variable labels. For example, if you are using a log transformation for response time, *logTime* could be a short and informative variable label that aids in later recalling the transform used and helps other individuals understand the data if you are working with a research team. Similarly, if you have reverse-coded items, you will likely want to label the original item (e.g., *item3_rev*), and you can create a new item with the reversed scores: *item3*. This transformation is discussed more in Chapter 13.

## CODING MISSING VALUES

You can identify missing values in R when you import the data. The default is to use "NA" to represent missing data. The original data may present missing values as blanks, though (i.e., cells with no value [""] or a space [" "]). Functions like *read.csv* will allow you to note missing values explicitly as one of the function arguments: *na.strings=c("NA", " ", "")*. This argument tells the function to treat the three types of cases provided ("NA", " ", "") as missing data, and we could include other values too. For example, occasionally researchers use *-99* or another unobtainable value to note an item did not pertain to an individual. We could list this value as an *NA* value: *na.strings=c("NA", " ", "", "-99")*.

You can assess the number of missing values for a variable using the *sum* and *is.na* functions. You also can obtain a sum for each variable in a data frame with the *colSum* and *is.na* functions. The bolded sections of the code can be replaced with the appropriate data frame and variable as needed.

```
sum(is.na(DataFrame$variable))
colSum(is.na(DataFrame))
```

After you evaluate the presence of missing data, missing values can be addressed in a few ways. First, the missing values can be removed from the data frame using the *na.omit* function. This function removes all rows with an *NA* value. Second, the missing values can be ignored by adding an *na.rm* argument to ignore *NA* values in a specific calculation: *na.rm=TRUE*. Third, missing values can be replaced, which will be discussed later in the chapter.

## SCREENING DATA

An initial step in data screening is often ensuring the values for each variable coincide with possible scores given how the variable is measured or coded. Relevant functions include *min, max,* and *table*. The *min* and *max* functions allow you to quickly identify the range of scores on a variable. For continuous variables, these values can be informative, and if a value exceeds the possible scores, you can revisit the original data (if possible), clear the cell, or delete the case to avoid erroneous data adding error to later analyses. The *table* function is useful for most types of data, and it is particularly helpful with categorical data (i.e., factor variables) to ensure all of the groups (i.e., factor levels) are properly identified.

Checking for missing values and potentially addressing missing cases is another important step in screening the data. If a relatively small number of values are missing, it may be possible to ignore cases with missing values when you use certain functions by adding the `na.rm` argument, or you can potentially exclude cases with too many missing values. The functions discussed earlier in the chapter (e.g., `is.na, colSum, na.omit`) can be useful in screening the data at this step. You can also assess if there is a pattern to the missing data using the `md.pattern` function from the `mice` package (van Buuren & Groothuis-Oudshoorn, 2011). Then, if the data are missing at random, missing values can be imputed using the `mice` function.

The distributions for continuous scores also should be evaluated. These distributions can be visually assessed using histograms (`ggplot2:: geom_histogram`) and Q-Q plots (`qqnorm`), and they can be quantitatively assessed using skewness (`semTools:: skew;` Jorgensen, Pornprasertmanit, Schoemann, & Rosseel, 2018; Weisstein, n.d.-b), kurtosis (`semTools:: kurtosis;` Weisstein, n.d.-a), the Kolmogorov-Smirnov test with a Lilliefors correct (`nortest:: lillie.test;` Gross & Ligges, 2015), and the Shapiro-Wilk test (`shapiro.test`), which are introduced in Chapter 6 of the first volume (Rasco, 2020a).

> Note: If the function is not included in the built-in packages, the package can be noted before the double colon. For example, the `geom_histogram` function comes from the `ggplot2` package (Wickham, 2016): `ggplot2:: geom_histogram`. This format allows us to use a function without attaching the package.

## Data Screening for Comparative Tests

Tests that involve comparing the means across groups (e.g., analysis of variance) require a certain number of cases per group. You can ascertain the number of cases per group using the `table` function to ensure a sufficient number are present (see Warner, 2020b, for further discussion).

The continuous variable for the test must be evaluated by group to ensure the variable is normally distributed and free from outliers. If you want to visually evaluate these distributions by group, you can check the section on Visualizing and Accounting for a Second Variable in Chapter 5 of the first volume (Rasco, 2020a). This section discusses how to create histograms (`geom_histogram`), density plots (`geom_density`), and boxplots (`geom_boxplot`) and account for a grouping variable with `fill` or `x` arguments using the `ggplot2` package.

## Data Screening for Associative Tests

Correlations and regressions require that variables be linearly associated and free from bivariate or multivariate outliers. These assumptions can be checked using scatterplots (`ggplot:: geom_point`) and Mahalanobis distance. If the association is not linear, you occasionally can transform a variable to create a linear association. Common transformations include the natural log (`log`), base 10 log (`log10`), and power functions (e.g., `x^2;` see Warner, 2020b, for discussion on when to use these functions).

The distributions for the variables involved in the correlation or regression need to be similar in shape. This similarity can be evaluated using histograms (`ggplot2::geom_histogram`) for the variables involved.

The residuals from a regression should be normally distributed and not correlated with values of `y` or `y'`. The residuals can be obtained from the regression model object: `DataFrame$RegResid <- residuals(RegModel)`. Additionally, predicted values can be obtained: `DataFrame$RegPred <- predict(RegModel)`. A histogram of the residual values and scatterplots of the residual values with `y` or `y'` can be obtained to evaluate these assumptions (see Chapter 11 in the first volume; Rasco, 2020a).

**Chapter 2 Summary of Key Functions (AKA: Function Cheat Sheet)**

| Function Call | Package | Description |
|---|---|---|
| write.csv | Included in utils | Saves CSV file with object in project folder |
| row.names | Included in base | Returns or changes data frame row names |
| sum | Included in base | Calculates total for argument passed to function |
| is.na | Included in base | Returns if element is NA or not; sets element to NA |
| colSum | Included in base | Calculates totals by column |
| na.omit | Included in stats | Omits rows containing missing data (i.e., NAs) |
| min | Included in base | Returns minimum value |
| max | Included in base | Returns maximum value |
| table | Included in base | Builds contingency table for factor variables |
| md.pattern | Mice | Displays pattern of missing data |
| mice | Mice | Generates replacement values for missing data |
| geom_histogram | ggplot2 | Creates histogram |
| qqnorm | Included in stats | Produces Q-Q plot with comparison to normal distribution |
| skew | semTools | Calculates skewness |
| kurtosis | semTools | Calculates kurtosis |
| lillie.test | Nortest | Performs Kolmogorov-Smirnov test with Lilliefors correction |
| shapiro.test | Included in stats | Performs Shapiro-Wilk test |
| geom_density | ggplot2 | Creates density plot |

| Function Call | Package | Description |
|---|---|---|
| geom_boxplot | ggplot2 | Generates boxplot |
| geom_point | ggplot2 | Produces scatterplot or adds point to existing plot |
| log | Included in base | Calculates natural log |
| log10 | Included in base | Computes base 10 log |
| ^ | Included in stats | Raises base to power (exponent) indicated |
| residuals | Included in stats | Extracts residuals from objects returned by model functions |
| predict | Included in stats | Calculates predicted values for model |