Stats 10 Lab 2: Data Cleaning/Preparation

Objectives

1. Understand logical statements and subsetting
2. Learn data cleaning and preparation techniques in R
3. Visualize prepared data using appropriate plots

Section 1 - Logical Statements/Relational Operators

Logical Expressions: Type **?Comparison** to see the R documentation on the list of all relational operators you can apply. Many logical expressions in R use these relational operators.

Try running the lines of code below that use the relational operators >, >=, <=, ==, !=:

```
4 > 3 # Is 4 greater than 3?
c(3, 8) >= 3 # Is 3 or 8 greater than or equal to 3?
c(3, 8) <= 3 # Is 3 or 8 less than or equal to 3?
c(1, 4, 9) == 9 # Is 1, 4, or 9 exactly equal to 9?
c(1, 4, 9) != 9 # Is 1, 4, or 9 not (exactly) equal to 9?
```

Notice that the output is a logical vector (i.e., uses TRUE and FALSE) that has the length of the vector on the left of the relational statement.

Applications of logical statements: calculations

We can perform certain calculations on logical vectors because R reads TRUE as 1 and FALSE as 0. Create the NCbirths object from last lab and try these examples:

```
sum(NCbirths$weight > 100) #the number of babies that weighed more than 100 ounces
mean(NCbirths$weight > 100) #the proportion of babies that weighed more than 100 ounces
mean(NCbirths$gender == "Female") #the proportion of female babies
```

mean(NCbirths$gender != "Male") #gives the proportion of babies not assigned male

Applications of logical statements: subsets

We can combine logical statements with square brackets to subset data based on conditions. Examples with NCbirths:

```
fem_weights <- NCbirths$weight[NCbirths$gender == "Female"]
```

With the line above we created a vector called fem_weights that contains the weights of all the female babies. We can combine multiple conditions using &&, and |, but these will be discussed in future labs.

Good coding practices

Please consider implementing the following in your code:

1. Use the pound symbol (#) often to comment on different code sections. Consider using them to label your exercise numbers and question parts, and to help describe what your code does.
2. Use good spacing. Adding a space between arguments and inside of functions makes your code easier to read. You can also skip lines for clarity.
3. Create as many objects as you like to make it easier to follow. For example, consider my line above creating the fem_weights object. An alternative way to code this using best practices is below:

```
## Create an object with the baby weights from NCbirths
baby_weight <- NCbirths$weight

## Create an object with the baby genders from NCbirths
baby_gender <- NCbirths$gender

## Create a logical vector to describe if the gender is female
is_female <- baby_gender == "Female"

## Create the vector of weights containing only females
fem_weights <- baby_weight[is_female]
```

Section 2 - Data Cleaning/Preparation and Visualization in R

In this section, you will learn how to clean and prepare data in R for visualization. We will use the grades data frame as an example.

First, let's create the grades dataset into R using the data.frame() function:

```
grades <- data.frame(
        student_id = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
        subject = c("Math", "Science", "English", "History", "Art",
                        "Physical Education", "Social Studies", "Music", "Geography",
                        "Computer Science"),
        grade = c(80, 90, 75, NA, 95, 70, 78, 92, NA, 82),
        age = c(17, 16, NA, 16, 17, 16, 18, NA, 17, 18),
        gender = c("M", "F", "M", "F", NA, "M", "F", "M", "F", "M"))
```

Before cleaning and preparing the data, it is important to explore it to understand its structure and identify any potential issues.
Use the following code to view the first few rows of the grades data frame:

```
head(grades)
```

From this output, we can see that there are missing values (NA) in the grade, age, and gender columns. We will need to handle these missing values during the data cleaning process.

Clean and Prepare the Data

- Checking for Missing Data

Missing data is a common issue in datasets and can cause problems when analyzing the data. In R, missing data is represented by the value NA.
To check for missing values in a dataset, we can use the is.na() function, which returns a logical vector indicating whether each element in a vector or data frame is NA. For example, to check for missing values in the grades dataset, we can use:

```
is.na(grades)
```

This will return a data frame with the same dimensions as grades, where each element is TRUE if it is NA and FALSE otherwise.
We can see that there are two missing values in the grade variable and two missing values in the age variable.
To check how many values are missing in total, use:

```
sum(is.na(grades))
```

- Handling Missing Data

One way to handle missing data in R is to use the na.omit() function to remove any rows with missing values.

```
grades_cleaned <- na.omit(grades)
```

Note that this will remove any rows with missing values from the grades data frame and return a new data frame, grades_cleaned. Note that this method may result in a loss of information, as any row with missing data will be removed. If you want to impute missing values instead of removing them, you can use the na.fill() function to replace missing values with a specific value.

- Converting Data Types

Sometimes it may be necessary to convert data types during the data cleaning process. You can use the class() function in R to check the data type and class of a column. For example, to check the data type and class of the grade column, use the following code:

```
class(grades_cleaned$grade)
```

This will return "numeric", indicating that the grade column is of type numeric. Similarly, to check the data type and class of the subject column, use the following code:

```
class(grades_cleaned$subject)
```

This will return "character", indicating that the subject column is of type character. Check whether the other variables in the dataset have the correct types.

If variables are not in the correct format for the analysis or visualization you want to perform, you can convert a column to a different data type, For example, you may need to convert a column of strings to dates or numeric values. This can be done using functions such as as.character(), as.numeric(), etc.

You may have noticed that student_id was imported as a numeric type instead of a character. The student_id variable should be converted to a character variable since it is an identifier and not a numerical value. You can use the as.character() function to convert it:

```
grades_cleaned $student_id <- as.character(grades_cleaned $student_id)
```

You can confirm the change by checking the data type and class of the column using the class() function.