Lab 3–Data Cleaning and Manipulation

Overview

The data we receive are often messy. A lot of preparation and care needs to be taken with data before you even begin analysis. Even if the data are "clean", some manipulation is usually needed in order to create variables of interest. In this lab, we learn how to identify and correct possible data entry errors and create new variables. We also verify that coding worked as intended - it is very important to check your work. Note that as we go through the various data cleaning steps, the R code is heavily commented - this is good practice.

The Data

The U.S. Census Bureau is best known for conducting the census every 10 years, which attempts to count every resident of the United States. Although population size data are valuable, very limited data are collected, and it is not collected frequently. The U.S. Census Bureau initiated the American Community Survey (ACS) in 2005 to collect more detailed information about the population, such as income, benefits, health insurance, education, employment, etc. Approximately 3.5 million households are surveyed annually, from which the data are used to allocate more than $400 billion in state and federal funds each year. Although the data are publicly available from the census website, it is quite large and contains many variables. Here, we provide a random sample of 1000 observations and 10 variables from the American Community Survey in the data set `acs.csv`.

Variable	Description
`Sex`	gender
`Age`	age in years
`MarStat`	marital status
`Income`	annual income (in $1,000s)
`HoursWk`	hours of work per week
`Race`	Asian, Black, White, or Other
`US Citizen`	citizen versus non-citizen
`HealthInsurance`	yes=have health insurance, no = no health insurance
`Language`	native English speaker versus other

Explore and Clean the Data

Import the acs.csv data set into RStudio and get to know the data by idenifying the data types, examining the data values, and looking at figures of the data.

```
# Set working directory
# use getwd() to see what your current working directory is.
setwd("~/Your/Working/Directory/") #the exact file path will differ individually

# Read in data
acs<-read.csv("acs.csv",header=T)
```

```r
# Examine structure of ACS
str(acs)

# Examine ACS variables
summary(acs)
```

Create new variables

1. Re-code values to missing

First, let's examine the distribution of `Age`. It is helpful to do this graphically, and by examining the different age values. A (table) of the `Age` variable shows all values of age, and how many observations took on those values.

```r
# Table summarizing age
table(acs$Age)

# Histogram of age
hist(acs$Age)
```

Although a value of 0 for age may initially seem implausible, there are 11 observations with a value of 0, and many more young children present in the data set, so it is likely that parents report information for their children when surveyed. The histogram also shows that younger ages are conisistent with the distribution of age. However, the value of 130 for age is clearly implausible and needs to be re-coded to missing. First, we create a new Age variable called Age2 which represents the cleaned version of the `Age` variable.

```r
#create new variable which will be the new cleaned version of age
acs$Age2<-acs$Age
```

Now the variable `Age2` is in the `acs` data set, and has the same values as `Age`. We need to replace the age value of 130 with `NA` to indicate that it is missing. First, we need to identify which observations have a value of 130. Since we are only interested in re-coding one value (130), we can see where `Age` is equal to that value. We could also examine a range of values (like all ages greater than 100).

```r
#identify implausible observations in age
acs$Age==130
acs$Age>100
```

Using either command, we can see that the 157th data entry contains the problematic age, because the expression takes on a value of TRUE. The 157th data entry of Age is implausible, so we need to re-code the 157th data entry of Age2. Now we use square brackets to access (also called indexing) this observation, and re-code it to missing. You can think of brackets as the equivalent of where. For example, we can view the 157th data entry

```r
#view position 157 in age
acs$Age2[157]
```

Age2 is a vector, or a single string of numbers. The values in this vector can be identifed by the position in the vector - each value indexed by a single number which represents the position. We can assign this observation to have a value of NA, which represents missing data for either character or numeric values.

```
#re-code 157th entry of Age2 to NA
acs$Age2[157]<-NA
```

This was simple because there was only one value to re-code. What if there were many ages over 100 that we wanted to assign to missing? An efficient way is to assign Age2 a value of NA where Age is greater than 100. This will produce an equivalent result to the previous code. Whenever possible, you should utilize this method.

```
#re-code all entries of Age2 where Age is greater than 100 to NA
acs$Age2[acs$Age>100]<-NA
```

To verify that the code worked correctly, we view a summary of `Age2` and verify that 130 has been replaced with `NA`.

```
#verify re-coding of Age2
summary(acs$Age2)
```

Now you should see that the maximum value of age is 94 (previously it was 130), and there is now one `NA` (previously there were none). You could also examine the rows of the data set that were recoded. Data frames are *indexed* by two positions: the row and the column. For example, we can view the implausible age entry in the 157th row and 2nd column of the data frame.

```
#print entry in 157th row and second column
acs[157,2]
```

We can view the entire 157th row by leaving the column index blank, accessing all columns.

```
#print all columns in 157th row
acs[157,]
```

Here, we can see that the original variable `Age` has a value of 130, but the cleaned version of the variable (`Age2`) has a value of `NA`, further confirming that the re-code worked correctly.
Equivalently, instead of accessing a specific row we could also access all rows that satisfy a certain condition.

```
#print all columns where age is greater than 100
acs[acs$Age>100,]
```

When data entries take on a value of missing, you may need to use an extra argument for some R Commands.For example, a summary Age2 works and shows that Age2 has a mean of 40.04 and has one missing value. However, when we try just to get the mean of age we get a result of `NA`.

```
mean(acs$Age2)
```

This is because of the missing value. In order to compute the mean using this function, you would need to specify the argument na.rm = TRUE in order for R to calculate the mean ignoring the missing values.

```
mean(acs$Age2,na.rm=T)
```

2. Numerical to Categorical (Factor)

Age is a numerical variable, but suppose we would like to classify individuals into age groups. Consider 0-18 as children, 19-55 adults, and those over 55 as senior citizens. Since we want to create a completely new variable (we do not want to retain any of the original numeric values of the age variable), we start by creating a new variable in our data set that represents an empty shell to fill in. A common method is to define this new variable to be a factor with all missing values, which we will then over-write. Simultaneously, we also define the `levels` of the factor variable. The function `c()` combines values for the different levels.

```
#Create new variable for age category
acs$AgeCategory<-factor(NA,levels=c("child","adult","senior citizen"))
```

Even though all values in `AgeCategory` are `NA`, this variable can take on the values specified by `levels`. Moreover, the ordering of the `levels` is as presented (otherwise the default is alphabetical ordering). Now we re-assign specific entries of age category based on our cleaned version of age.

```
#Asssign values of age category
acs$AgeCategory[acs$Age2<=18]<-"child"
acs$AgeCategory[acs$Age2>18 & acs$Age2<=55]<-"adult"
acs$AgeCategory[acs$Age2>55]<-"senior citizen"
```

In the second command we use & to specify when two conditions are satisfied simultaneously. This command assigns a value of "adult" for all ages greater than 18 and less than or equal to 55. When doing these assignments, be very careful using less/greater than and less/greater than or equal to correctly specify cut off points. If you need to define a variable as a factor, or change the ordering of factor levels, you can use the following general syntax where you would substitute in the variable name and level values for your specific variable.

```
#create a factor variable and order the levels
variable<-factor(variable, levels=c("level1","level2","level3"))
```

To verify that this coding worked correctly, view a contingency table of the two variables. The first variable listed should be the original numerical variable, and the second variable should be the new categorical variable.

```
#check coding of age category
table(acs$Age2,acs$AgeCategory)
```

From this table, we can see that ages 0 to 18 were correctly assigned as "child", 19 to 55 as "adult", and 56 to 94 as "senior citizen".

3. Categorical to categorical: re-assigning the levels

Race has four categories, but suppose we would like to classify individuals as "white" or "non-white". Because we want to create a completely new variable (we do not want to retain any of the original values of the race variable), we start by creating a new variable in our data set with all missing values.

```
#Create new variable for race category
acs$RaceNew<-factor(NA,levels=c("white","non-white"))
```

Now we re-assign specific entries of race. Where the original variable is "white", the new variable should also be "white". Where the original variable is "asian" or "black" or "other" the new variable should be "non-white". A vertical line (|) represents or in R syntax.

```r
#re-assign values of the new race variable
acs$RaceNew[acs$Race=="white"]<-"white"
acs$RaceNew[acs$Race=="asian" | acs$Race=="black" | acs$Race=="other"]<-"non-white"
```

Lastly, to verify that the re-coding worked correctly view a contigency table of the original and new variable. Specify the original variable first, and the new variable second.

```r
#Check re-coding of RaceNew
table(acs$Race,acs$RaceNew)
```

In the contingency table, it is clear that “asian”, “black”, and “other” were all correctly assigned a value of “non-white”.