MSIS 5223: Tutorial 11 – Regression with Categorical Data in R

Instructions

The purpose of this assignment is to help you become familiar with using R. In this assignment you will become familiar with performing regression in R using categorical data.

1. Categorical Data

Categorical data comes in many different forms. For example, gender is typically a binary value and is considered categorical. Another example is income range, which typically assesses income of individuals by providing discrete ranges of values (see table below).

Income Ranges
$0 – 19,999
$20,000 – 39,999
$40,000 – 59,999
$60,000 – 79,999
$80,000 – 99,999
$100,000 – 119,999
$120,000 – 139,999

Table 1 Income Ranges

In the last tutorial you learned how to create regression models using continuous independent variables. Recall that the regression equation is

$$y = \beta_0 + \beta_1 x_1$$

where *y* is the dependent variable, *x* is the independent variable, and β_1 and β_0 are parameters, or constants. The parameter β_1 is the slope of the line and indicates to extent to which *y* changes with *x*. When interpreting the regression output, the slope is multiplied by *x*, or the actual value from your data.

Imagine having a variable that is categorical within your regression equation. How would you multiply the value of "$100,000 – 119,999" by a slope? As you can see, this is not possible and presents a problem. Categorical data is not ready for use in regression because it is not numerical.

Within statistics, two main families of variable derivation exist: 1) Combining Variables and 2) Single-Variable Transformations. Combining Variables includes taking

two or more variables and merging them together. Many well-known transformations are a result of combining data:

- Price-Earnings Ratio
- Body Mass Index
- Wind Chill Index

Combining Variables doesn't help with categorical data in regression. This is where Single-Variable Transformation is useful. Simply put, Single Variable Transformation involves converting a single variable into a different format. An example of this is mean-centering, or standardizing. Many variables have a different scale, such as income and age, and in order to compare them on equivalent scales both need to undergo standardization. Another example of a type of transformation is converting counts into rates or into percentages.

Categorical data requires careful consideration when converting into numerical values. The first impulse is to arbitrarily assign numbers to the differing levels. Look at the examples in Table 2 and Table 3 below. For *income ranges*, assigning numeric values increasing from the lowest amount of "$0 – 19,999" up to "$120,000 – 139,999" makes some sense. Regression will assume that a value of 2 is one greater than a value of 1.

Income Ranges	Values
$0 – 19,999	1
$20,000 – 39,999	2
$40,000 – 59,999	3
$60,000 – 79,999	4
$80,000 – 99,999	5
$100,000 – 119,999	6
$120,000 – 139,999	7

Table 2 Income Ranges with Assigned Values

Family Structure	Values
Single, no children	1
Single, with children	2
Married, no children	3
Married, with children	4
Life partner, no children	5
Life partner, with children	6

Table 3 Family Structure with Assigned Values

The same does not hold true for *family structure*. In Table 3 numerical values are assigned to *family structure* in a similar way to what was done for *income range*. Again, regression will interpret "Married, with children" as being one greater than "Married, no

children". In reality, one of these is not quantifiably better than the other on this scale. That is, "Life partner, with children" is not 5 times more than "Single, no children". Obviously, many other categorical variables behave similarly to *family structure*, so a different solution is needed.

2. Indicator Variables

The solution to the problem presented in the previous section is the use of Indicator Variables, or Dummy Variables as they are sometimes called. This is an alternative method to replacement with numerical values. The process is very simple: create n-1 variables where n is the number of categories or the number of levels. A data value of "1" represents a record in the data has that category while a value of "0" represents that record not having that category.

For example, gender includes three options: male, female, N/A. Create n-1, or 3-1, variables: two columns, one that is labeled *male* and the other labeled *female*. Those who are male receive a "1" for male and "0" for female; those who selected female receive a "1" for *female* and "0" for *male*. Those who did not select either one receive a "0" within both columns. That is, a "0" in both columns represents the third option of "N/A."

Return to the example of *family structure*. The variable has six levels, resulting in *n*-1 or 5 new columns. As shown in Table 4 below, this is what the data would look like. This results in a great many zeroes within the newly created columns. This is alright and an intended consequence of the process. This data is straightforward. Each individual has a 1 in the column that represents their category. For example, Greg is single with no children; Barbara Sue is married, with children. The exception is Kris, who does not have a value of 1 within any column. This means she has the category that is not provided as a column: life partner, with children.

Name	Age	Sing-NC	Sing-WC	Mar-NC	Mar-WC	LP-NC
Greg	23	1	0	0	0	0
Barbara Sue	37	0	0	0	1	0
Jolinda	49	0	0	1	0	0
Sankara	46	0	0	0	0	1
Parker	32	0	1	0	0	0
Kris	26	0	0	0	0	0

Table 4 Indicator Variable Example: Family Structure

This process can become overwhelming if the number of categories is extremely large. For example, converting all 50 US states into *n*-1 categories. Another example is that of actual, specific income values (not income ranges). With a sample of 2,000 individuals, each providing their income from last year, it would be next to impossible to create *n*-1 indicator variables. This is where binning comes to the rescue.

This is a process of converting numeric values into categorical levels. Binning comes in three different types:

- Equal width binning – Create a set number of bins giving each one equal ranges; may lead to unbalanced bins.
- Equal weight binning – Creates bins based on percentage; so, you could have 5 bins each with 20%; 4 bins each with 25%
- Supervised binning – Using information about the target variable to influence the input variable (classification tree); be warned, you can't always generalize to other datasets or your population

With income, simply find the distribution or frequency to help determine the number of bins and their respective sizes. Binning is a subjective process and requires a few iterations in order to find an optimal range. The example below will walk you through this process.

3. Binning a Variable

The example presented here will use the data file reduction_data_new.txt. Only a subset of this data will be used including the following columns: *peruse01*, *intent01*, *gender*, *educ_level*, and *age*. The columns are renamed to improve readability. Additionally, missing values are removed from the dataframe.

```
> red_data = reduction_data[, c('peruse01', 'intent01', 'gender', 'educ_level', 'age')]
> colnames(red_data) = c('usefulness', 'intent', 'gender', 'education', 'age')
> red_data = na.omit(red_data)
```

Figure 1 Preparing the Data for Analysis

The basic structure of the dataframe reveals three numeric columns (*usefulness*, *intent*, and *age*) and two categorical columns (*gender* and *education*) (see Figure 2 below). While the temptation to use *age* as a continuous variable is strong, it is not usable

in its current format. Look at Figure 2 below. The two lines of code at the bottom reveal the ranges for *intent* and *age*. *Intent* has a range of 1 to 7 while *age* spans 18 to 48. As these are not on the same scale, they are not directly comparable. One solution would be to standardize both variables; the other is to convert *age* into a categorical variable.

```
> str(red_data)
'data.frame':   165 obs. of  5 variables:
 $ usefulness: int  7 6 5 5 7 6 6 6 5 6 ...
 $ intent    : int  7 7 5 5 6 5 6 6 5 5 ...
 $ gender    : Factor w/ 2 levels "1","2": 1 2 1 1 2 2 1 1 2 1 ..
 $ education : Factor w/ 5 levels "2","3","4","5",..: 3 3 1 2 2 2
 $ age       : int  22 32 19 21 21 21 21 21 23 23 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:3] 164 165 166
  .. ..- attr(*, "names")= chr [1:3] "164" "165" "166"
> head(red_data)
  usefulness intent gender education age
1          7      7      1         4  22
2          6      7      2         4  32
3          5      5      1         2  19
4          5      5      1         3  21
5          7      6      2         3  21
6          6      5      2         3  21
> range(red_data$intent)
[1] 1 7
> range(red_data$age)
[1] 18 48
```

Figure 2 Basic Structure of the Dataframe

The purpose of binning is to impose "levels" or "categories" onto a numerical variable. Think back to the example of income. It is possible to have discrete values of income, but often it is easier to work with income brackets. The same principal applies to age.

Binning is quite simple using the library car. This library was used in Tutorial 9 to calculate the Durbin-Watson score and determine variance inflation factors. The process of binning requires you to select a size for each bin. That is, how many units will each bin hold? In this case, one unit of age is one year. This data contains a range of 30 years because the maximum value is 48 and the minimum is 18. It is always good to start off with a conservative number of bins; anywhere from 5 to 8. Let's start off with 6 bins. Divide 30 by 6 and the result is 5; that is, each bin will contain 5 years.

```
> bin_interval = seq(18, 48, by = 5)
> bin_interval
[1] 18 23 28 33 38 43 48
> table(cut(red_data$age, bin_interval, right = FALSE))

[18,23) [23,28) [28,33) [33,38) [38,43) [43,48)
    116      32      10       2       3       1
```

Figure 3 Binning by 5-Year Increments

Figure 3 presents the result of creating 5-year incremental bins. The first line creates a sequence of numbers with the second line of code revealing what those numbers are. Each of these numbers represents the start of each bin. For example, 18 is the start of the range 18 through 22; 23 is the start of 23 through 27.

The third line of code provides a table that shows the sample size within each bin. Notice the three bins on the left contain the most data: 158 records out of 165. The figure below presents a histogram of *age*. Not surprisingly, the data is skewed to the right with the majority on the left side. Perhaps the bin sizes are too great. Let's use a bin size of 2 instead of 5.

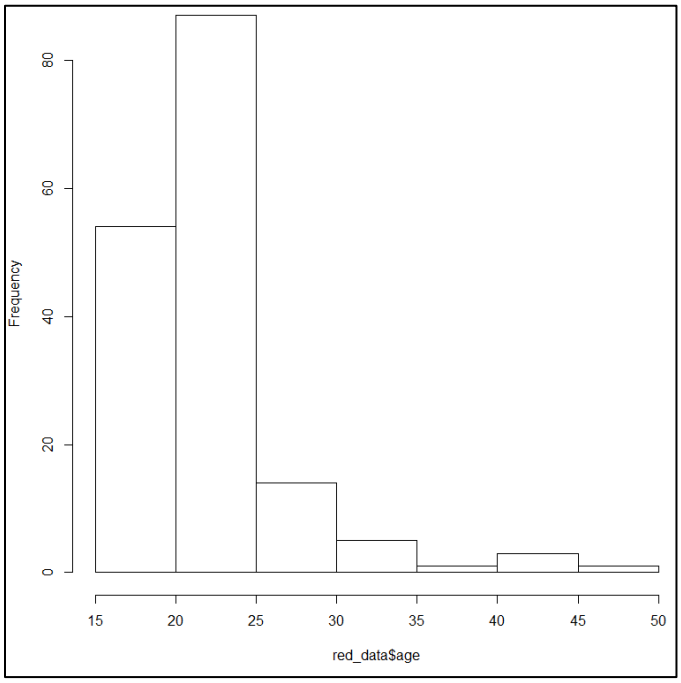

Figure 4 Histogram of Age

```
> bin_interval = seq(18, 48, by = 2)
> table(cut(red_data$age, bin_interval, right = TRUE))

(18,20] (20,22] (22,24] (24,26] (26,28] (28,30] (30,32] (32,34] (34,36] (36,38] (38,40]
     53      62      17      13       5       4       3       1       1       0       1
(40,42] (42,44] (44,46] (46,48]
      2       1       0       1
```

Figure 5 Binning with 2-Year Increments

Binning with 2-year increments has created a more even distribution of the sample within each bin, as shown in Figure 5. While the majority of the data is retained in the first few bins, it isn't an overwhelming amount.

At this point, an important warning should be given about the function cut() which is used within the table() function. Notice it has an argument right=TRUE. Prior to explaining what it is, run the code with that set to FALSE and see what happens. Figure 6 shows the code with the argument set to FALSE and set to TRUE. Look at the last bin, ages 46 through 48; now look at the first bin, ages 18 through 20. When the argument right=TRUE is used, 18-20 includes three years, not two; when right=FALSE is used, 46-48 is completely empty!

```
> table(cut(red_data$age, bin_interval, right = FALSE))

[18,20) [20,22) [22,24) [24,26) [26,28) [28,30) [30,32) [32,34) [34,36)
     21      69      36      15       7       7       2       2       1
[36,38) [38,40) [40,42) [42,44) [44,46) [46,48)
      0       1       1       2       0       0
> table(cut(red_data$age, bin_interval, right = TRUE))

(18,20] (20,22] (22,24] (24,26] (26,28] (28,30] (30,32] (32,34] (34,36]
     53      62      17      13       5       4       3       1       1
(36,38] (38,40] (40,42] (42,44] (44,46] (46,48]
      0       1       2       1       0       1
```

Figure 6 Dissecting Argument right=TRUE

The maximum value of *age* is 48, so something appears wrong with the first line of code. When right=FALSE is used, the right most value within a bin is not inclusive; when it is TRUE, it is inclusive. The problem presented here is that in order for bin 46-48 to have a value, then 18-20 has three years of data. Remember, each bin is only supposed to have two years' worth of data. The way around this problem is to stretch the maximum value passed the value of 48; i.e. extend it to the next bin size, which is 50 because we are using 2-year increments. Figure 7 presents this change.

```
> table(cut(red_data$age, bin_interval, right = FALSE))

[18,20) [20,22) [22,24) [24,26) [26,28) [28,30) [30,32) [32,34)
     21      69      36      15       7       7       2       2
[34,36) [36,38) [38,40) [40,42) [42,44) [44,46) [46,48) [48,50)
      1       0       1       1       2       0       0       1
```

Figure 7 Insert a New Bin

This is much better, but does not have equally distributed bins. The first four bins still contain the majority of the data. In fact, all the other combined contain 24 data points. Perhaps forcing the other bins into one single bin would make sense in order to keep a fairly consistent distribution.

```
> bin_interval = c(18, 20, 22, 24, 26, 50)
> table(cut(red_data$age, bin_interval, right = FALSE))

[18,20) [20,22) [22,24) [24,26) [26,50)
     21      69      36      15      24
```

Figure 8 Consolidating Bins

Figure 8 presents the adjustment to the bins. Notice the last bin contains data with ages between 26 and 50. This has a total of 24 records, which is a good amount. The first bin contains 21, so this is close enough to make the distribution of all these bins fairly normal. Think of this process as similar to applying a logarithmic transformation.

Now that the bin values have been selected, the *age* column must be converted into these bins and added back into the dataframe. Figure 9 below presents the code that follows these steps. The function recode comes from the library car. Essentially, it finds values within a target—the variable *age* in this case—and replaces them based on criteria provided. For example, if the age is 18 through 19, then the new value is a character "18-19"; if the age is 26 through 48, then the new value is "26-48".

```
> age_vector = recode(red_data$age, "18:19='18-19'; 20:21='20-21';
+                     22:23='22-23'; 24:25='24-25'; 26:48='26-48'")
+
> red_data$age_categ = age_vector
> red_data[, c('age', 'age_categ')]
   age age_categ
1   22     22-23
2   32     26-48
3   19     18-19
4   21     20-21
5   21     20-21
6   21     20-21
```

Figure 9 Recoding Age Into New Values

The second line of code adds this newly created data back into the dataframe as a new column called *age_categ*. The third line of code merely provides a way to compare the original column *age* with the new column *age_categ*. Notice that the values correspond perfectly between the two.

At this point, no indicator variables have been created. This process has only led to the creation of a new categorical variable based on *age*. To create a dummy variable, use the library dummies. Ensure it is installed prior to continuing. The process is straight forward because the library does all the heavy lifting. The figure below presents the results of using the function dummy(). The argument sep = '_' is used for naming the new columns. Each new column contains the name of the original column. Since the original

column is named *age_categ* each new dummy variable will begin with that. As R creates each new column, the category level is used to denote the increment of each new dummy variable. The underscore is then used to separate the original name of the variable from the categorical label.

```
> red_dummy1 = dummy(red_data$age_categ, sep = '_')
> colnames(red_dummy1)
[1] "age_categ_18-19" "age_categ_20-21" "age_categ_22-23" "age_categ_24-25" "age_categ_26-48"
> colnames(red_dummy1) = c('age_18_19', 'age_20_21', 'age_22_23', 'age_24_25', 'age_26_48')
> red_dummy2 = as.data.frame(red_dummy1)
> red_data = data.frame(red_data, red_dummy1)
```

Figure 10 Creating Indicator Variables from Age_Categ

Look at line two which provides the column names of the newly created dummy variables. The naming convention is apparent. These are not the most user-friendly, so these columns are renamed. Finally, the newly created dummies are converted into a dataframe object and then placed back into the dataframe. Within the tutorial script the variable *education* is also converted into indicator variables; however, because it is repetitive, the process will not be covered in this tutorial document.

One final note should be mentioned here. Recall, when creating indicator variables, the process is to create *n*-1 new columns. The function dummy() within R does not provide such a fine distinction. It creates *n* new columns; so, one more column than needed is created. When creating a regression model, merely leave off one of the dummy variables from the regression equation. Personally, I just leave off the last of the columns. What happens if you include it within your regression equation? The regression model returns “N/A” in place of the statistical values for that variable.

4. Regression with Categorical Data

Now that the conversion process is complete, a regression model may be created. This process is very similar to that of multiple regression, except the assumptions of linear regression are not assessed. Since these columns are binary, a linear relationship is not possible. Keep in mind when using binary data, the result is just like ANOVA; therefore, the assumptions of linear regression are not required.

The first regression equation will use gender and the age dummy variables. When the indicator variables for age were created, five new columns were added to the dataframe. Recall, the function dummy() creates one too many indicator variables. Leave

the last indicator variable out of the equation. Figure 11 provides the results of this assessment.

```
lm(formula = red_data$intent ~ red_data$gender + red_data$age_18_19 +
    red_data$age_20_21 + red_data$age_22_23 + red_data$age_24_25)

Residuals:
    Min      1Q  Median      3Q     Max
-4.9030 -0.5359  0.7159  0.9515  1.9268

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)         6.4296     0.3541  18.156  < 2e-16 ***
red_data$gender2   -0.1455     0.2645  -0.550  0.58305
red_data$age_18_19 -0.7482     0.4979  -1.503  0.13489
red_data$age_20_21 -1.2108     0.3962  -3.056  0.00263 **
red_data$age_22_23 -0.6270     0.4389  -1.429  0.15508
red_data$age_24_25 -0.3811     0.5483  -0.695  0.48807
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 1.665 on 159 degrees of freedom
Multiple R-squared:  0.06913,   Adjusted R-squared:  0.03986
F-statistic: 2.362 on 5 and 159 DF,  p-value: 0.04237
```

Figure 11 Gender and Age on Intent

The age range of 20 to 21 is found to be significant when predicting the usage of mobile technology. The interpretation of this is no different from multiple regression. Using the significant variables, create a regression equation using the coefficients and the intercept. Insert the actual values from the dataset and find the value for the dependent variable.

```
lm(formula = red_data$intent ~ red_data$gender + red_data$age_18_19 +
    red_data$age_20_21 + red_data$age_22_23 + red_data$age_24_25 +
    red_data$educ_2 + red_data$educ_3 + red_data$educ_4 + red_data$educ_5)

Residuals:
    Min      1Q  Median      3Q     Max
-4.8674 -0.3415  0.5997  0.9587  1.9341

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        6.40034    0.61736  10.367  < 2e-16 ***
red_data$gender2  -0.09834    0.26569  -0.370 0.711790
red_data$age_18_19 -0.99935    0.54648  -1.829 0.069365 .
red_data$age_20_21 -1.52522    0.44079  -3.460 0.000698 ***
red_data$age_22_23 -0.82542    0.46013  -1.794 0.074779 .
red_data$age_24_25 -0.45870    0.55335  -0.829 0.408410
red_data$educ_2   -0.23672    0.78681  -0.301 0.763921
red_data$educ_3    0.46640    0.65531   0.712 0.477701
red_data$educ_4   -0.07858    0.75470  -0.104 0.917212
red_data$educ_5   -0.34811    0.68949  -0.505 0.614357
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 1.655 on 155 degrees of freedom
Multiple R-squared:  0.1041,    Adjusted R-squared:  0.05203
F-statistic:     2 on 9 and 155 DF,  p-value: 0.04268
```

Figure 12 Gender, Age, and Education

Figure 12 presents another regression equation using the education dummy variables. The addition of these variables does not improve the regression, resulting in no significant relationships.

```
lm(formula = red_data$intent ~ red_data$gender + red_data$age_18_19 +
    red_data$age_20_21 + red_data$age_22_23 + red_data$age_24_25 +
    red_data$usefulness)

Residuals:
    Min      1Q  Median      3Q     Max
-5.0340 -0.3737  0.3972  0.8566  2.6939

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)          3.9348     0.7520   5.232 5.25e-07 ***
red_data$gender2    -0.1787     0.2546  -0.702 0.483759
red_data$age_18_19  -0.7765     0.4789  -1.621 0.106976
red_data$age_20_21  -1.3687     0.3835  -3.569 0.000475 ***
red_data$age_22_23  -0.8253     0.4255  -1.939 0.054240 .
red_data$age_24_25  -0.4781     0.5280  -0.906 0.366578
red_data$usefulness  0.4593     0.1234   3.721 0.000275 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 1.602 on 158 degrees of freedom
Multiple R-squared:  0.1441,    Adjusted R-squared:  0.1116
F-statistic: 4.435 on 6 and 158 DF,  p-value: 0.0003566
```

Figure 13 Gender, Age, and Usefulness

The last regression model represents an ANCOVA model. Education has been removed from the model and replaced with *usefulness*. Recall, this variable is continuous and not a categorical variable. Not surprisingly, this is a significant variable in the model.