second R-02

Paul E. Johnson\textsuperscript{1} \textsuperscript{2}

\textsuperscript{1}Department of Political Science

\textsuperscript{2}Center for Research Methods and Data Analysis, University of Kansas

2015
Outline

1. Overview

2. Numerics are Easy

3. What about Factors?
   - What’s a Factor?
   - How are Factors Created?
   - Benefits
   - Recoding Factors

4. NES
   - Numeric Variables
   - Recoding Factor Variables
   - Functions Respond Differently to Factors
   - Regression Responds differently
Outline

1. Overview

2. Numerics are Easy

3. What about Factors?
   - What’s a Factor?
   - How are Factors Created?
   - Benefits
   - Recoding Factors

4. NES
   - Numeric Variables
   - Recoding Factor Variables
   - Functions Respond Differently to Factors
   - Regression Responds differently
Recoding Data

- Numeric recoding is pretty easy
- Factors can be challenging
Recoding Numeric Variables

- Numeric variables are easy because they have not much internal structure that is hidden from view.
- Example: a numeric vector has no “attributes”

```r
x <- c(2.2, 1.1, 5.2, 10.1)
class(x)
```

```
[1] "numeric"
```

```r
attributes(x)
```

```
NULL
```
Recoding is easy!

- R has math functions like +, -, /, log(), sqrt(), exp(), and so forth.
- Run "help("+")", or "?log", etc.
- It is as simple as column in, column out

```
exp(x)
```

```
```

```
log(x)
```

```
[1] 0.78845736 0.09531018 1.64865863 2.31253542
```

```
sqrt(x)
```

```
[1] 1.483240 1.048809 2.280351 3.178050
```
Use cut to create categorical ranges

- If you want to divide a numeric range into groupings, use the cut function

```r
set.seed(234234)
x2 <- rnorm(100)
x2cut <- cut(x2, breaks = c(-50, -1, 2, 100), labels = c("Minimal", "Medium", "Huge"))
table(x2cut, exclude = NULL)
```

<table>
<thead>
<tr>
<th>x2cut</th>
<th>Minimal</th>
<th>Medium</th>
<th>Huge</th>
<th>&lt;NA&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>86</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Conditional Recodes are Easy As Well

- in second-R-01, I had an example using a TRUE/FALSE index variable.
- The function ifelse() is convenient, perhaps easier arguments are a “logical condition”, and a value if the condition is true, and one if it is false.

```r
y <- c(31, 33, 41, 61)
ifelse(x < 3, y, x)
```

```
[1] 31.0 33.0 5.2 10.1
```

- Can also manufacture a string variable

```r
ifelse(x < 3, "Paul is Great", "Paul is Excellent")
```

```
[1] "Paul is Great"  "Paul is Great"  "Paul is Excellent"  "Paul is Excellent"
```
The only dangers are

- You don’t understand the function you apply, such as

  ```r
  z <- c(-2, -0.4, 0, 1, 2, 3)
  log(z)
  ```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
<td>-Inf</td>
<td>0.6931472</td>
<td>1.0986123</td>
</tr>
</tbody>
</table>

- Or you forget that computer calculations are numerically approximate.
  - Comparison of any particular numeric value against a numeric variable is hazardous.
  - You wouldn’t draw from a normal distribution and ask if a case is exactly equal to 1.96, would you?
  - Until yesterday, I thought ≥ and ≤ were safe, but look this example I found on StackExchange.com

  ```r
  a <- 0.58 ; b <- 0.08 ; (a-b) >= 0.5
  ```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>FALSE</td>
</tr>
</tbody>
</table>
The only dangers are ...

- I had thought that \(<\) or \(>\) were safe, but now I wonder.
- So far as I know, a fastidious person would need to build a complicated comparison logic into any if-then statement:

  ```r
  tol <- 1e-5  ## tolerance is 0.00001
  (a-b) >= (0.05-tol)
  ```

  [1] TRUE

- My conclusion is that, if I were doing fine-grained numeric calculations, I’d need a good book or two and some buddies in the Internet.

- The function “zapsmall()” will return 0.0 if a value is numerically close enough to 0. That doesn’t always to the way you plan.

  ```r
  a - b - 0.5
  ```

  [1] -5.551115e-17

  ```r
  zapsmall(a - b - 0.5)
  ```
The only dangers are ...

[1] $-5.551115e-17$
Outline

1. Overview

2. Numerics are Easy

3. What about Factors?
   - What’s a Factor?
   - How are Factors Created?
   - Benefits
   - Recoding Factors

4. NES
   - Numeric Variables
   - Recoding Factor Variables
   - Functions Respond Differently to Factors
   - Regression Responds differently
Factor Variables = Categorical Variables

- A factor is a categorical variable, discrete values like \{Catholic, Protestant\} or \{left, right, middle\}
- A factor that is subjectively ordered is called an “ordered factor”, or simply an “ordered” variable.
- Factors have attributes (the “levels”).
- Functions notice that and customize calculations and beautify displays
Creating Factors

- We’ve already seen that the `cut()` function can create a factor variable.
- Direct access to the factor machine: `factor()` or `ordered()`.
- `factor()` has 3 key arguments,
  - a variable: something with values that need to be converted to a factor
  - levels: values of existing variable that are to be used, in order we want them to appear
  - labels: character strings. If omitted, R runs “as.character(levels)” to manufacture labels.
Creating Factors ...

*factor() example: Convert a character string to a factor*

```r
y <- c("KU", "KSU", "KU", "KU", "KU", "WSU", "KU", "USC", "KSU", "KU", "KU")
yf <- factor(y)
```

R guesses levels and labels. It guesses levels *in alphabetical order*

- See what you got

  ```r
  levels(yf)
  ```

  ```r
  [1] "KSU"  "KU"  "USC"  "WSU"
  ```

  ```r
  table(yf)
  ```

  ```r
  yf  
  KSU KU USC WSU 
  2  7  1  1 
  ```
Creating Factors ...

<table>
<thead>
<tr>
<th>character string</th>
<th>R assigned internal value</th>
<th>label for the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>“KSU”</td>
<td>1</td>
<td>“KSU”</td>
</tr>
<tr>
<td>“KU”</td>
<td>2</td>
<td>“KU”</td>
</tr>
<tr>
<td>“USC”</td>
<td>3</td>
<td>“USC”</td>
</tr>
<tr>
<td>“WSU”</td>
<td>4</td>
<td>“WSU”</td>
</tr>
</tbody>
</table>

Attribute

```r
attributes(yf)
```

```r
$levels
[1] "KSU"  "KU"  "USC"  "WSU"
```

```r
$class
[1] "factor"
```
Creating Factors ...

Specify the particular levels you want, in which order

\[
yf \leftarrow \text{factor}(y, \text{levels} = \text{c("WSU", "KU", "KSU"))}
\]

The levels argument specifies

- the CURRENT values that are to be used in the new variable “yf”
  - the ORDER in which the values are kept
  - Any existing values that are not named are converted to missing, which appears as \(<\text{NA}>\)

- Check what you got

\[
\text{table}(yf)
\]

\[
\begin{array}{ccc}
\text{yf} \\
\text{WSU} & \text{KU} & \text{KSU} \\
1 & 7 & 2
\end{array}
\]
Creating Factors ...

Here’s what I see

<table>
<thead>
<tr>
<th>original value</th>
<th>R assigned internal value</th>
<th>label for the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>character string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“WSU”</td>
<td>1</td>
<td>“WSU”</td>
</tr>
<tr>
<td>“KU”</td>
<td>2</td>
<td>“KU”</td>
</tr>
<tr>
<td>“KSU”</td>
<td>3</td>
<td>“KSU”</td>
</tr>
</tbody>
</table>

Everything is OK, as long as you actually want those levels in that order
Creating Factors ...

Specify the particular levels you want, and add new labels

```r
yf <- factor(y, levels = c("WSU", "KU", "KSU"),
             labels = c("southern", "good", "western"))
levels(yf)

[1] "southern" "good"   "western"

yf

[1] good    western  good  good  good  southern  good  <NA>  western  good
Levels: southern good western
```

Must keep labels in order with levels!
Creating Factors ...

factor() example: Convert an integer variable to a labeled factor

```r
x <- c(1, 2, 1, 2, 2, 7)
x_f <- factor(x, levels = c(7, 2, 1), labels = c("any_for_7", "level_two", "iwant_1"))
levels(x_f)
```

```
[1] "any_for_7" "level_two" "iwant_1"
```

**levels** tells R “in which order should the values of x come in?” I jumbled the order to make a point.

<table>
<thead>
<tr>
<th>original value</th>
<th>R assigned internal value</th>
<th>label for the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>“any_for_7”</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>“level_two”</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>“iwant1”</td>
</tr>
</tbody>
</table>

- Suppose we forget the labels argument in the factor function. What do we get? R guesses the labels should be the as.character(x)
What about Factors?

How are Factors Created?

Creating Factors …

```r
xf2 <- factor(x, levels = c(7, 2, 1))
levels(xf2)
```

```
[1] "7" "2" "1"
```

And the part that can drive you crazy is that the internal numbers are 1, 2, 3, but the named labels are “7”, “2”, “1”.

<table>
<thead>
<tr>
<th>original value</th>
<th>R assigned internal value</th>
<th>label for the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>“7”</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>“2”</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>“1”</td>
</tr>
</tbody>
</table>

```r
xf2
```

```
[1] 1 2 1 2 2 7
Levels: 7 2 1
```
Creating Factors …

- Converting back to a numbered variable is not conceptually simple. We currently have a “factor with levels that look like numbers, but are not”!

```r
as.numeric(xf2)
```

[1] 3 2 3 2 2 1

- I once wrote a whole section in a book manuscript about recovering a numeric value from an accidentally created factor of that type

```r
xnew <- as.numeric(levels(xf2))[xf2]
xnew
```

[1] 1 2 1 2 2 7

```r
table(xnew, x)
```

<table>
<thead>
<tr>
<th></th>
<th>xnew</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 7</td>
</tr>
<tr>
<td>2</td>
<td>1 2 0 0</td>
</tr>
<tr>
<td>7</td>
<td>2 0 3 0</td>
</tr>
<tr>
<td>0</td>
<td>7 0 0 1</td>
</tr>
</tbody>
</table>
Benefits of Using Factors

- Using factors reduces coding errors. If you code “Female” as 1 and “Male” as 2, there’s a good chance later you’ll make a mistake when looking at values “1” and “2”

- Procedures notice the levels. Remember xf?

```r
xf

[1] iwant_1 level_two iwant_1 level_two level_two any_for_7
Levels: any_for_7 level_two iwant_1

xf=="level_two"

[1] FALSE  TRUE FALSE  TRUE  TRUE FALSE

table(xf)

xf
any_for_7 level_two iwant_1
1     3    2"
Benefits of Using Factors …

- If you give the factor variable to a plotting function, that function should notice it is not a numeric variable and act accordingly.

```r
plot(xf, main = "plot noticed its a factor!", xlab = "xf: a factor", ylab = "Count")
text(0.2, 2, "R noticed xf \n is a factor. \n So R ran table \n and sent output \n to barplot", pos=4)
```

![plot noticed its a factor!]

- R noticed `xf` is a factor.
- So R ran table and sent output to barplot.
Examples of Needed Recodes

- Most often, I have a factor with labels like `c("Male", "Man", "Female", "Woman")` and I want to boil those down to `{Male, Female}`

- In the rockchalk package, I created a function `combineLevels()` that can work on a problem like that, and it does some error checking to make sure it works correctly.

- However, many of the recodes we want can’t be done that easily, and thus we have to learn some of the R functions that manipulate factors.
Working with factors

1. Run `str`, or ask a particular variable for its class
   ```r
class(xf)
```
   or run
   ```r
is.factor(xf)
```

2. Run table to review the values.
   ```r
table(xf, exclude = NULL)
```
   The “exclude = NULL” argument forces R to print out all of the levels of `xf`, even ones for which we currently have no observations

3. Levels. Displays the character strings used for levels.
   ```r
levels(xf)
```
   ```r
xl <- levels(xf)
```
   Now `xl` is a vector, “`xl[1]`” is quick. That’s nicer that rewriting “`levels(xf)[1]`”. 
Reduce/simplify the number of Labels

1. **Relabel**: To assign more succinct/different labels to the levels.
   - R has quite a few functions that can be used to both get and set values.
   - Replace all the levels with a new set of labels.
     ```r
ingo levels(aFactor) <- c("new1", "new2", "new3", "new4", "new5")
```
   - **Vital**: Must provide names for all levels
   - Duplicated level names have been discouraged for a long time, but this works now without error
     ```r
     levels(aFactor) <- c("new1", "new1", "new3", "new4", "new1")
     ```
   - The previously used levels 1, 2, and 5 will now be “new1”.
   - **rockchalk::combineLevels** has some handy sanity-preserving features :) 
     ```r
     library(rockchalk)
     combineLevels(aFactor, levs = c("new2", "new5"), newLabel = c("new1"))
     ```
What about Factors?

Recoding Factors

Let’s suppose we don’t want level 3 any more. If we knew for sure its label was “California”, we could hit that specifically

```r
xf[xf %in% "California"] <- NA
```

I’m usually worried I’ve spelled “California” incorrectly, so I will instead refer to level 3’s label programatically

```r
xf[xf %in% levels(xf)[3]] <- NA
```

That uses the “indexing” TRUE/FALSE property of R vectors.
Alter values by pattern

1. This works, if “SoCal” is currently a valid level
   \[ xf[xf \text{ in} \text{ levels}(xf)[3]] \leftarrow \text{"SoCal"} \]

2. This works, if “SoCal” is level 1
   \[ xf[xf \text{ in} \text{ levels}(xf)[3]] \leftarrow \text{levels}(xf)[1] \]

3. Want to set \( xf[c(1,3,10)] \) to a not-yet existing value? You’ve got trouble coming
   \[ \text{table}(xf, \text{exclude} = \text{NULL}) \]
   \[ xf[xf \text{ in} \text{ levels}(xf)[3]] \leftarrow \text{"not currently a valid level"} \]

   That returns an error. In order to assign that weird level, you have to add it to the levels first
   \[ \text{levels}(xf) \leftarrow \text{c(levels}(xf), \text{"not currently a valid level")} \]
   \[ \text{levels}(xf) \]
What about Factors?

Recoding Factors

Alter values by pattern ...

```r
[1] "any_for_7"  "level_two"
[4] "iwant_1"
[4] "not_currently_a_valid_level"
```

The Unused Levels problem.

- Suppose there are possible levels `c("Catholic", "Protestant", "Jewish", "Muslim", "Hindu", "Atheist")`
- But the survey is conducted in the center of Elf’s Breath, Montana, where every single person is an Atheist.
- The variable will have **unused levels** “Catholic” “Protestant” “Jewish” “Muslim” and “Hindu”. Unused level is slang for “values not present in a sample.”
- Sometimes we want to carry those along because we expect to add more cases
- But, keeping them makes noise in some plots and tables
- Ways to purge the unused levels if you don’t want them.
  - Run them through the factor function (I find this easiest)
What about Factors?

Recoding Factors

Alter values by pattern ...

```
xf <- factor(xf)
```

R Documentation suggests this is more meaningful (?)

```
xf <- xf[, drop = FALSE]
```
Outline

1. Overview

2. Numerics are Easy

3. What about Factors?
   - What’s a Factor?
   - How are Factors Created?
   - Benefits
   - Recoding Factors

4. NES
   - Numeric Variables
   - Recoding Factor Variables
   - Functions Respond Differently to Factors
   - Regression Responds differently
Get Some Data from Nat. Election Study 2004

- If you already have a “../workingdata” folder, and it has a file “04245-0001-Data.dta”, let's use that.
- Otherwise, go download a copy, so you’ll have one in the future

```r
fn <- "../workingdata/04245-0001-Data.dta"
if (!file.exists(fn)){
  if (!file.exists("../workingdata")) dir.create("../workingdata/", showWarnings = FALSE)
  download.file("http://pj.freefaculty.org/guides/Rcourse/DataSets/04245-0001-Data.dta", destfile = "../workingdata/04245-0001-Data.dta")
}
library(foreign)
mydt1 <- read.dta(fn)
```
That’s a handy idiom: `!file.exists()`

- In `!file.exists...` the exclamation mark `!` means NOT.
- It saves copy for next time, so it doesn’t download again
A Codebook Lists the Variables

■ Some things we can treat as numeric

```markdown
## V043039  B1b. Feeling Thermometer: John Kerry
## V043250  Y1x. Summary: Respondent age
```

■ Some are clearly categorical

```markdown
## V043210  R1. R position on gay marriage
## V043213  S3. National economy better/worse since GW Bush took ofc
## V045145X H5x. Summary: Pre-Post US flag makes R feel
## V043116  J1x. Summary: R party ID
```

■ Some are treated as numeric by some people

```markdown
## V045117  G4a. Liberal/conservative 7-point scale: self-placement
## V043116  J1x. Summary: R party ID
```

but they are treated as categorical by others!
Numeric Recodes

Create New Variable: The difference in thermometer between Bush and Kerry

```r
mydta1$th bk <- mydta1$V043038 - mydta1$V043039
```

![Bush - Kerry](image-url)
Numeric Recodes ...

```r
hist(mydta1$th.bk, breaks = 40, main = "Bush − Kerry", xlab = "Thermometer Difference")
```

- Check out the age variable

```r
hist(mydta1$V043250, breaks = 40, main = "", xlab = "Age")
```
Use cut to create a “dummy variable” for old people

```r
mydta1$aged <- cut(mydta1$V043250, breaks = c(-1, 56, 200), labels = c("young", "old"))
```

56 is this year’s definition of old, in case you wondered.

```r
plot(mydta1$aged, xlab = "Age dichotomized")
```
Forgot to check histogram the Bush Thermometer!

```r
hist(mydta1$V043038, breaks = 50, xlim = c(-1, 101))
```
That funny thing about thermometer scores

```r
options(width=60)
table(mydtal$V043038)
```

<table>
<thead>
<tr>
<th>0</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>49</th>
<th>50</th>
<th>55</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>95</td>
<td>3</td>
<td>1</td>
<td>81</td>
<td>1</td>
<td>90</td>
<td>2</td>
<td>1</td>
<td>97</td>
<td>1</td>
<td>104</td>
</tr>
<tr>
<td>65</td>
<td>70</td>
<td>75</td>
<td>80</td>
<td>85</td>
<td>90</td>
<td>95</td>
<td>98</td>
<td>100</td>
<td>2</td>
<td>155</td>
<td>4</td>
<td>6</td>
<td>194</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```r
options(width=100)
```
Clean up a bunch of variables and value labels

**Party Identification**

```r
## Party
## drop unused levels
mydta1$V043116 <- mydta1$V043116[, drop = TRUE]
table(mydta1$V043116, exclude = NULL)
```

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Strong Democrat (2/1/. -8-9/.)</td>
<td>203</td>
</tr>
<tr>
<td>1. Weak Democrat (2/5 -8-9/.)</td>
<td>179</td>
</tr>
<tr>
<td>2. Independent-Democrat (3-4-5/.5)</td>
<td>210</td>
</tr>
<tr>
<td>3. Independent-Independent</td>
<td>118</td>
</tr>
<tr>
<td>4. Independent-Republican (3-4-5/.1</td>
<td>138</td>
</tr>
<tr>
<td>5. Weak Republican (1/5 -8-9/.)</td>
<td>154</td>
</tr>
</tbody>
</table>
Clean up a bunch of variables and value labels ...

<table>
<thead>
<tr>
<th>6. Strong Republican (1/1/. )</th>
<th>7. Other; minor party; refuses to say</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

## abbreviate party attachment labels
```r
levels(mydta1$V043116) <- c("SD", "WD", "ID", "I", "IR", "WR", "SR", "O")
table(mydta1$V043116, exclude = NULL)
```

```
SD  WD  ID  I  IR  WR  SR  O  <NA>
203 179 210 118 138 154 193  5  12
```

## This is America. We hate level 8. Lets make them MISSING
```r
mydta1$V043116[ mydta1$V043116 %in% levels(mydta1$V043116)[8] ] <- NA
mydta1$V043116 <- mydta1$V043116[, drop = TRUE]
table(mydta1$V043116, exclude = NULL)
```
Clean up a bunch of variables and value labels ...

<table>
<thead>
<tr>
<th>SD</th>
<th>WD</th>
<th>ID</th>
<th>I</th>
<th>IR</th>
<th>WR</th>
<th>SR</th>
<th>&lt;NA&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>203</td>
<td>179</td>
<td>210</td>
<td>118</td>
<td>138</td>
<td>154</td>
<td>193</td>
<td>17</td>
</tr>
</tbody>
</table>

- **Ideology**

```r
# IDEO
table(mydata1$V045117, exclude = NULL)
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Extremely liberal</td>
<td>20</td>
</tr>
<tr>
<td>02</td>
<td>Liberal</td>
<td>103</td>
</tr>
<tr>
<td>03</td>
<td>Slightly liberal of the road</td>
<td>125</td>
</tr>
<tr>
<td>04</td>
<td>Moderate; middle of the road</td>
<td>279</td>
</tr>
<tr>
<td>05</td>
<td>Slightly conservative</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Conservative</td>
<td></td>
</tr>
</tbody>
</table>

...
Clean up a bunch of variables and value labels ...

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>Extremely conservative</td>
<td>143</td>
</tr>
<tr>
<td>166</td>
<td>Haven't thought much {DO NOT PROBE}</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>Don't know</td>
<td>0</td>
</tr>
<tr>
<td>89</td>
<td>Refused</td>
<td>0</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>345</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
mydta1$V045117 <- mydta1$V045117[, drop = TRUE]
levels(mydta1$V045117)
```
Clean up a bunch of variables and value labels ...

```r
[1] "01. Extremely liberal"  #02. Liberal
[3] "03. Slightly liberal"    #04. Moderate; middle of the road
[5] "05. Slightly conservative"  #06. Conservative
[7] "07. Extremely conservative"

levels(mydta1$V045117) <- c("EL","L","SL","M","SC","C","EC")
table(mydta1$V045117)

EL L SL M SC C EC
20 103 125 279 143 166 31
```

### Gender

```r
# Gender
table(mydta1$V041109A, exclude = NULL)

1. Male 2. Female <NA>
566 646 0
```
Clean up a bunch of variables and value labels ...

```r
levels(mydta1$V041109A) <- c("M","F")
```

- Gay Marriage: Note the interesting mismatch between the “value labels” from the original data format and the levels as we see them in R

```
## Gay Marriage
table(mydta1$V043210, exclude = NULL)
```

<table>
<thead>
<tr>
<th>1. Should be allowed</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Should not be allowed</td>
<td>705</td>
</tr>
<tr>
<td>5. Should not be allowed to marry but should be allowed</td>
<td>41</td>
</tr>
<tr>
<td>VOL</td>
<td>30</td>
</tr>
<tr>
<td>8. Don't know</td>
<td>0</td>
</tr>
<tr>
<td>9. Refused</td>
<td>0</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>36</td>
</tr>
</tbody>
</table>
Recoding Factor Variables

Clean up a bunch of variables and value labels ... 

```r
levels(mydata1$V043210)
```

```
[1] "1. Should be allowed"
[3] "5. Should not be allowed to marry but should be allowed"
[4] "VOL"
[5] "8. Don't know"
```

```r
mydata1$V043210[ mydata1$V043210 %in% levels(mydata1$V043210)[4:7] ]
<-NA
mydata1$V043210 <- mydata1$V043210[, drop = TRUE]
levels(mydata1$V043210) <- c("Allow","No","Med")
table(mydata1$V043210, exclude = NULL)
```

<table>
<thead>
<tr>
<th></th>
<th>Allow</th>
<th>No</th>
<th>Med</th>
<th>&lt;NA&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>400</td>
<td>705</td>
<td>41</td>
<td>66</td>
</tr>
</tbody>
</table>

- Expect the economy to get better?
Clean up a bunch of variables and value labels ...

```r
## Economy
mydta1$V043213 <- mydta1$V043213[, drop = TRUE]
table(mydta1$V043213, exclude = NULL)

1. Better 3. Worse 5. The same <NA>
   190   668   343    11

lvl <- levels(mydta1$V043213)
econnew <- factor(mydta1$V043213, levels = lvl[2:3:1],
                   labels = c("Worse", "Same", "Better"))
table(mydta1$V043213, econnew)

econnew

<table>
<thead>
<tr>
<th></th>
<th>Worse</th>
<th>Same</th>
<th>Better</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Better</td>
<td>0</td>
<td>0</td>
<td>190</td>
</tr>
<tr>
<td>3. Worse</td>
<td>668</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5. The same</td>
<td>0</td>
<td>343</td>
<td>0</td>
</tr>
</tbody>
</table>

mydta1$V043213 <- econnew
rm(econnew)
```

How does it make you feel to see the flag?
Clean up a bunch of variables and value labels ...

```r
## Flag
mydta1$V045145X <- mydta1$V045145X[, drop = TRUE]
table(mydta1$V045145X)

1. Extremely good          2. Very good
   3. Somewhat good           338
   570

4. Not very good          7. Don't feel anything {VOL}
   38
   18
```
Plot two numeric variables

```r
plot(V043038 ~ V043039, ylab = "Bush Thermometer", xlab = "Kerry Thermometer", data = mydata1)
```
Plot one numeric and one factor variable

```r
plot(V043038 ~ V043213, ylab = "Bush Thermometer", xlab = "Economic Expectations", data = mydtal)
```
How about the Age effect?

plot(jitter(V043038) ~ V043250, ylab = "Bush Thermometer", xlab = "Age", data = mydata1)

jitter “scatters” points so they don’t pile up
Numeric Predictor

- Predict the Bush-Kerry Difference from respondent Age

```r
mod1 <- lm(th.bk ~ V043250, data = mydta1)
summary(mod1)
```

Call:
`lm(formula = th.bk ~ V043250, data = mydta1)`

Residuals:
```
          Min       1Q   Median       3Q      Max
-108.821  -42.753  -2.003  42.905  103.340
```

Coefficients:
```
            Estimate Std. Error  t value  Pr(>|t|)
(Intercept)  -6.84133    4.59553  -1.4890   0.13724
V043250      0.18426     0.09181   2.0069   0.04502 *
```

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 53.89 on 1189 degrees of freedom
(21 observations deleted due to missingness)
Multiple R^2: 0.003376, Adjusted R^2: 0.002538
F-statistic: 4.028 on 1 and 1189 DF,  p-value: 0.04498
Add A Factor as a Predictor

```r
mod2 <- lm(th.bk ~ V043250 + V041109A, data = mydta1)
summary(mod2)
```

```
Call:
  lm(formula = th.bk ~ V043250 + V041109A, data = mydta1)

Residuals:
  Min      1Q  Median      3Q     Max
-113.174 -42.222  -2.782  42.478 107.164

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.0850    4.83136 -0.6390  0.5232
V043250     0.1912    0.09166  2.0870  0.0371 *
V041109AF  -7.7134    3.12318 -2.4700  0.0137 *

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 53.77 on 1188 degrees of freedom
  (21 observations deleted due to missingness)
Multiple R^2:  0.008467 , Adjusted R^2:  0.006798
F-statistic: 5.072 on 2 and 1188 DF,  p-value: 0.006404
```
Now Look Back at What R did with the Gender Predictor

- R creates the “design matrix”, the purely numerical representation of the variables. Notice it creates the dummy variable for Gender.

```r
mod2mm <- model.matrix(mod2)
head(mod2mm)
```

<table>
<thead>
<tr>
<th>(Intercept)</th>
<th>V043250</th>
<th>V041109AF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>53</td>
</tr>
</tbody>
</table>
Add Party ID as a Predictor

```r
mod3 <- lm(th.bk ~ V043250 + V041109A + V043116, data = mydta1)
summary(mod3)
```

```r
call: lm(formula = th.bk ~ V043250 + V041109A + V043116, data = mydta1)

Residuals:
  Min     1Q  Median     3Q    Max
-132.23 -22.22    0.67  21.24 152.95

Coefficients:                Estimate Std. Error t value Pr(>|t|)
(Intercept)          -61.90433   4.17721  -14.820  < 2e-16 ***
V043250               0.15178    0.05996   2.531   0.011  *
V041109AF             -0.75965    2.02758  -0.375   0.708
V043116WD             32.54200    3.57568   9.101  < 2e-16 ***
V043116ID             27.33216    3.46369   7.891  6.87e-15 ***
V043116I              54.11372    4.12505  13.118  < 2e-16 ***
V043116IR             88.46875    3.84445  23.012  < 2e-16 ***
V043116SR             93.24480    3.70477  25.169  < 2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 34.27 on 1165 degrees of freedom
(38 observations deleted due to missingness)
Multiple R^2:  0.6005, Adjusted R^2:  0.5977
F-statistic: 218.9 on 8 and 1165 DF,  p-value: < 2.2e-16
```
Check the model matrix now

```r
mod3mm <- model.matrix(mod3)
head(mod3mm, 10)
```

<table>
<thead>
<tr>
<th></th>
<th>(Intercept)</th>
<th>V043250</th>
<th>V041109AF</th>
<th>V043116WD</th>
<th>V043116ID</th>
<th>V043116I</th>
<th>V043116IR</th>
<th>V043116WR</th>
<th>V043116SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>37</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>71</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>62</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>49</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>56</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>47</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- In the olden days (or now if you use some software), the user has to create all those “dummy” columns to represent the levels. In R, we avoid it.
Let's stash a copy of this working data frame

today <- format(Sys.time(), "%Y%m%d")
saveRDS(mydta1, file = paste0("../workingdata/nes2004-", today, ".rds"))
save(mod1, mod2, mod3, file = paste0("../workingdata/nes2004-objects-", today, ".RData"))