Importation II

Foreign

Paul E. Johnson[1,2]

[1]Department of Political Science
University of Kansas

[2]Center for Research Methods and Data Analysis
University of Kansas

2013

Outline

Here's The Problem

- Proprietary/secret data storage formats are used.
- Export/Import lead to frustration.
- Many little wrinkles with "text encoding" and communication across formats.

The Usual: Use One Rectangular Data Set

- Following the introduction of SPSS in 1968, social scientists became accustomed to the idea of using a "rectangular data set."

*Var*1	*Var*2	*Var*3	*Var*4	*Var*5
1	5	6	4	31
2	2	3	5	29
3	4	5	5	53
4	2	2	3	22

(1)

- The first columns are usually "respondent identifier"
- Some programs (first SPSS, later Stata) insist the user may access only one data collection at a time.
- SAS (later R), diverged, allowing users to have many data collections open at one time.

Variable "name" "value" and "value label"

SPSS also established the terms variable name, value, and value label.

Value The values of the variables are kept in a numeric format, say 1,2,3.

Value Label Substantive labels, one for each value.

Var2		
Value	Label	Count
1	Never	145
2	Sometimes	424
3	Often	321
4	Always	40
8	Don't Know	188
9	Other Missing	200

Codebook A listing of variables, each with values and labels, often including frequency distributions.

Transition from "data set" to "database"

Database A collection of separate rectangular tables that are linked "relationally". Storage, insertion, and retrieval of particular bits is optimized for speed and efficiency of memory usage.

Why Database? Less "data copying". If you have "Age" data on 2000 respondents, but "Calcium" data on only 10, keep 2 tables.

ID	Age
1	24
2	44
...	*2000 rows*
2000	66

ID	Calcium
1	
193	
...	*200 rows*
1932	

Prediction: more database storage in future

- The problem we see now is that data sets are "too big" to be opened "all at once"
- Will be more efficient to leave most of the data "in the box" and just take out columns when needed
- Will require some new skills (SQL language, perhaps)
- Prediction: large-ish collections of data are increasingly likely to be stored in SQL-accessible databases, rather than "big rectangular data sets".
- Tools exist for R to allow one to draw data from a data base when needed, although this is still a new thing, an area of active development.

Many Competing Storage Formats Currently Exist

- SPSS, SAS, STATA, WinRATS, all maintain their own proprietary "mini-database" routines to keep data and make it available.
- "Inside" their guts, they aren't necessarily just storing the data as a "big rectangle" of numbers.
- They have unique/special/secret code to remove numbers from their storage and make them available to users.

R's Data Frame Roughly Equivalent to SPSS Data Set

Data Frame R calls a "rectangular data thing" a data frame.

Definition A list of Columns, all of which have the same number of elements

Important Note Many R functions allow the "formula interface" which will require a data frame:

```
plot(y ~ x, data = mydf)
```

or

```
lm(y ~ x, data = mydf)
```

What's Different about R

Open Many Frames R will allow the user to have many data frames "open" at the same time

Data Management R has many tools to merge, sort, and re-shape "rectangular data things" to eventually create new data frames that will be used in analysis.

Raw Text Transfer: The worst case scenario

- Use "whatever" program to write a text file (say, comma delimited or such).

Caution : Putting a data set into a "flat" text file may lose some information (e.g., value labels)

- In R, use read.table() (or similar) to access the file.

The Best Case Scenario

- A program's data storage format is
  - well documented, and
  - no proprietary tools are used to compress or encrypt the file.
- There is an R volunteer (or R Core Development member) who wants to import that data format and does the work required to translate that data into R
- R's foreign package, which was put into the R base collection in 1999, is a focal point for us.
- In my experience, Stata Data format is the only member of this group that almost always works. `read.dta()` is very well done (Thanks to Thomas Lumley) and `write.dta()` is the only R function I know of that, without fail, creates files that are readable as native by another program.

The Middle Case Scenario #1 (Neutral Formats)

- There is no export from program A into R. (Presumably, A's format is either 1) secret or 2) badly designed or 3) uninteresting to all R users on earth.)
- Format A's developers (or users) offer an export tool that convert format A to some other format in which R developers have had interest.
- Examples of such interchange formats
  - Science-oriented formats: HDF5 (from NCSA) and netCDF (from UCAR)
  - SQL database text. A more-or-less standardized language for exchange of tables via carefully formatted text files.

The Middle Case Scenario #2 (Excel)

- Some other program exists which is able to open format A and write a format that R can understand.
- Example: Microsoft Excel
  - Since Excel has a proprietary, poorly documented storage format, opening and sorting those files requires hard work by somebody who *really needs* to access data in that format.
  - The program Perl is used worldwide by many people who *really need* to read/write Excel files.
  - Greg Warnes, author of the R package "gdata", wrote connective tissue between Perl and R called `read.xls()`. This is by far the most reliable Excel importer.
- The Good News is that Excel is not a highly nuanced storage format–it does not have "factor variables" to worry about.

The Middle Case Scenario #2 (SPSS)

- SPSS has had many formats for file storage, documentation not profusely available.
- The PSPP is a open source volunteer effort made up of people who *really* want to import SPSS files for statistical analysis.
- The PSPP project's code is written in C and distributed under GPL. It is used in
  - R Core's foreign package as the basis of `read.spss()`
  - Martin Elff's memisc package (which offers several very SPSS-like data interaction functions).
- My experience is that `read.spss()` succeeds about 2/3 of the time. (More success with small, single-developer datasets).
- Many peculiar problems arise with character encoding. I've asked in r-help, they say SPSS GUI does not "defend" itself against users who enter characters that are from unexpected encodings.

The Middle Case Scenario #3 (I/E Programs)

- I often find that hard-to-open datasets can be opened by one of these programs.
  - GRETL. An open source software project Gretl (GNU regression and econometrics library) can open E-view, SPSS (again, relying on PSPP code), and other formats.
  - psppire. The PSPP project's user interface program, not entirely unfamiliar to social scientists.
  - StatTransfer. A commercial product that purports to be able to open data in many formats and convert between them. directly competes with R as a modeling tool for social science researchers.

Importing is just the start, however

Suppose we've opened the data in R.
We still need to

- Verify that it is imported correctly
- Check missing value codes
- Recode variables to suit our purposes

In the following, I offer some detailed case studies of data importation and management.

Stata Data is Most Reliably Imported

- I've tested these over the years, nothing lately has changed my opinion:
- `read.dta()` and `write.dta()` are most stable, reliable data exchange options
- If you have a choice of formats from which to import, *pick Stata*.
- As with other data formats, the primary problem is usually the handling of factors

Example: The American National Election Study, 2004

- If your institution is a member of the ICPSR:
  `http://www.icpsr.umich.edu/cgi-bin/bob/newark?study=4245`
- Choose the "Take All Files in One Archive" option.
- Unzip that in the R current working directory. It is probably easier to use some program like 7-zip in your operating system, but you can unzip without leaving R if you want to:

```
system("unzip 10243963.zip")
```

- Snoop around in there, you find codebooks and various scripts and data files. I decided to concentrate on the file "04245-0001-Data.dta" which is in a subdirectory called "10243963/ICPSR_04245/DS0001.
- I have a copy of that Stata data file in the examples subdirectory

Exercise: Try This For Yourself

```
require(foreign)
mydta1 <- read.dta("examples/04245-0001-Data.dta")
str(mydta1)
colnames(mydta1)
attributes(mydta1)
View(mydta1)
```

Look at the Input Data One Column at a Time

- As with other data formats, the primary problem is usually the handling of factors
- The default value for convert.factors = TRUE.
- If read.dta finds a column with

only numbers	it creates a numeric variable
characters	it creates a factor
value labels	it creates a factor using the value labels
both numbers and letters	it creates a factor

There is a Codebook Somewhere

- Review the file "DS0001/04245-0001-Codebook.pdf"
- Example

```
-------------------------------------------------------------------------------
V043116  J1x.  Summary:  R party ID
-------------------------------------------------------------------------------
PRE-ELECTION SURVEY:
QUESTION:
----------
Generally speaking, do you usually think of yourself as a
REPUBLICAN, a DEMOCRAT, an INDEPENDENT, or what?
Would you call yourself a STRONG [Democrat/Republican] or
a NOT VERY STRONG [Democrat/Republican]?
Do you think of yourself as CLOSER to the Republican
Party or to the Democratic party?
VALID CODES:
------------
0. Strong Democrat (2/1/.)
1. Weak Democrat (2/5-8-9/.)
2. Independent-Democrat (3-4-5/./5)
3. Independent-Independent
(3/./3-8-9 ; 5/./3-8-9 if not apolitical)
4. Independent-Republican (3-4-5/./1)
```

There is a Codebook Somewhere ...

```
5. Weak Republican (1/5–8–9/.)
6. Strong Republican (1/1/.)
7. Other; minor party; refuses to say (9/./. ; 4/./3–8–9)
MISSING CODES:
-----------------
8. Apolitical (5/./3–8–9 if apolitical)
9. DK
```

The Most Famous Variable in Political Science

- Observe levels of the party variable as it stands now:

```
levels(mydta1$V043116)
```

I see:

```
 [1] "0. Strong Democrat (2/1/.)"
 [2] "1. Weak Democrat (2/5-8-9/.)"
 [3] "2. Independent-Democrat (3-4-5/./5)"
 [4] "3. Independent-Independent"
 [5] "4. Independent-Republican (3-4-5/./1)"
 [6] "5. Weak Republican (1/5-8-9/.)"
 [7] "6. Strong Republican (1/1/.)"
 [8] "7. Other;minor party;refuses to say"
 [9] "8. Apolitical (5/./3-8-9 if apolitical)"
[10] "9. DK (8/./.)"
```

Yes, Virginia. The Factor's Levels Really Are Ugly

- read.dta tries to convert variables that have labels into factors
- this factor has troublesome labels for the levels, as in:

```
"4. Independent-Republican (3-4-5/./1)"
```

Look at This Ugly Table

```
table(mydta1$V043116)
```

```
                                0. Strong Democrat (2/1/.)
                                                        203
                                                1. Weak Democrat
                                                        179
                                2. Independent-Democrat (3-4-5/./5)
                                                        210
                                                  3. Independent
                                                        118
                                4. Independent-Republican (3-4-5/./1)
                                                        138
                                                5. Weak Republican
                                                        154
                                6. Strong Republican (1/1/.)
                                                        193
                                7. Other;minor party; refuses to say
                                                          5
8. Apolitical (5/./3-8-9 if apolitical) DK (8/./.)                9.
                                                          0
```

Some More Succinct Labels Would help

- Let's go terse!
- First, lets make a copy of the party ID variable and re-assign labels for the levels (careful! keep ordering)

```
partyid <- mydta1$V043116
levels(partyid) <- c("SD","WD","ILD","II",
                     "ILR","WR","SR","O","APol","DK"
                       )
```

- Second, get rid of unused levels ("APol" and "DK")

```
partyid <- partyid[ , drop=TRUE]
# same as partyid <- factor(partyid)
table(partyid)
```

```
partyid
 SD  WD ILD  II ILR  WR  SR   O
203 179 210 118 138 154 193   5
```

The 5 Os Bug Me

- Notice the use of `%in%` and `levels()` here:

```
partyid[partyid %in% levels(partyid)[8]] <- NA
```

levels(partyid) is an array of level names for partyid

levels(partyid)[8] is the 8'th level.

%in% selects rows where partyid is set at 8th level.

<- NA replaces those as NA, making O an "unused level"

```
table(partyid, exclude = NULL)
```

```
partyid
 SD  WD ILD   II  ILR  WR  SR   O
203 179 210 118 138 154 193   0
```

Can eliminate a range of levels if you want:

```
partyid[partyid %in% levels(partyid)[5:8] ] <- NA
```

Factors. Can't Live With 'Em. Can't Live ... ?

- It is possible to avoid factors altogether

```
mydtanum1 <- read.dta("examples/04245
    -0001-Data.dta",
                      convert.factors=F)
```

- That "throws away" the information, but if you just want the numerical variables, who cares?
- With the "numbers only" version, no "factor hassle"
- Then "make due" (like in the *olden days*) by
  - Comparing the Codebook against the observed numbers
  - Creating new factor variables
- I often import data both ways and compare, e.g.

```
table(mydtanum1$V1234, mydat1$V1234)
```

I've Heard there is a Program called Excel, by Microsoft

But I've not seen it :)

- Excel uses a proprietary data storage format
- Despite repeated efforts by the manufacturer to obscure the storage format, intrepid volunteers have found ways to rescue data.

The "Boy Scout" Approach

- Open the Excel file in a Spread Sheet program
- Make sure
  - Make the sheet is rectangular
  - Each column is of a "homogeneous" type
  - All variables are numeric or character (no "generic", no "percent").
  - File-> SaveAs (CSV) or (TXT).
  - If you see buttons for customization, take them!
- Review the text file in a plain text editor (e.g., Emacs)
- Use "read.table" to bring that file into R.

Use A Special Package To Access the Excel Spreadsheet

- The "gdata" package (by Gregory Warnes) has a long-standing track record
- It requires Perl (For which free versions are available)
- `read.xls()` has syntax similar to `read.table()`

```
myfn <- "practiceData.xls"
dat <- read.xls(myfn, sheet=1, header=T)
```

Use An Intermediate Program To Transfer Data

- gretl
  - "gretl" is the GNU Econometrics and Time Series program.
  - Gretl can import Excel, Eviews, SPSS, and other formats
  - Gretl can export to text, R, Octave
- Stat Transfer is a commercial product for Windows that can exchange data between formats.

You Want to Write R Data To Excel?

- R can create "comma delimited text" files that Spreadsheets can import
- Several package writers are endeavoring to create a workable write.xls function. When I last tested, the one that worked best was a commerical product (that I was unwilling to purchase).

Some SPSS Files Are Imported Easily. Some Not

- SPSS "sav" files often work
- SPSS "por" (portable) files seem less portable than "sav" files.
- Two major problems with SPSS files are
  - Character Encoding Problems
  - Accidental Factor Variables

Check ?read.spss

```
Read an SPSS Data File

Description:

     'read.spss' reads a file stored by the SPSS
     'save' or 'export' commands.

Usage:

     read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE,
               max.value.labels = Inf, trim.factor.names = FALSE,
               trim_values = TRUE, reencode = NA,
               use.missings = to.data.frame)
```

Highlights in read.spss

- Notice "to.data.frame": Otherwise, it creates a list.
- use.value.labels: Do you want to try to create R factors using the SPSS value labels
- reencode: R may be able to spot characters from foreign, unsupported character sets and convert them to current locale. Every time I open an SPSS data set, a message appears "reencoding from latin1" or such.

SPSS and the factor problem

- By default, read.spss will scan for variables that have one or more labeled values. Those variables will be interpreted as factors.
- For "gender" or "race", we want that.
- But some SPSS variables are numeric, but they also have labels!
  Ex: "age" is in years, but the value 998 is labeled "Unknown".
- R becomes confused, it wants to turn "age" into a factor, with levels "1","2","3",...,"99","100","Unknown"
- To avoid that one case, read.spss allows the option

  ```
  max.value.labels = 8
  ```

  A variable with more than 8 values will not be converted to a factor.

Bizarre, Interesting Things Happen

- My experience: read.spss() is a "fragile" function
- Sometimes the import will fail entirely
- Sometimes it will succeed with frightening warnings
- Has been getting better
- Never "damages" data or recovers it incorrectly

The General Social Survey

- NORC (Nat'l Opinion Research Center)
- Administered Annually or Bi-annually
  - Permanent Questions
  - Question Modules (addressed to subsets of respondents)
  - Some Questions only asked once or twice
- Davis, James A., Tom W. Smith, and Peter V. Marsden. *General Social Surveys, 1972-2006 [Cumulative File]* Storrs, CT: Roper Center for Public Opinion Research, [Computer file]. ICPSR04697-v4 University of Connecticut/Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributors], 2009-12-04.

Outline

General Social Survey

- The ICPSR (U. of Michigan) is a canonical source
  - `http://www.icpsr.umich.edu/icpsrweb/ICPSR/`
  - `GeneralSocialSurveys,1972-2006[CumulativeFile]`
    - "Download All" button grabs a file called "10805932.zip"

What's in that Zip File?

- Unzip that to reveal the contents:
  - TermsOfUse.html
  - ICPSR_04697: a folder
  - Inside ICPSR_04697

04697-descriptioncitation.pdf	how to cite
04697-manifest.txt	list of files
04697-related_literature.txt	citations
series-28-related_literature.txt	citations
DS0001	Another Folder

Its Like Peeling an Onion: Inside DS0001

04697-0001-Codebook.pdf	List of Variables & Summary Info
04697-0001-Data.dta	Stata Format
04697-0001-Data.sav	SPSS save Format
04697-0001-Data.stc	Terrasoft SAS data file
04697-0001-Data.tsv	tab-separated text
04697-0001-Data.txt	"fixed field" text
04697-0001-Setup.dct	Stata "data dictionary"
04697-0001-Setup.do	Example Stata code uses txt*
04697-0001-Setup.sas	Example SAS code uses txt*
04697-0001-Setup.sps	Example SPSS code uses tsv*
04697-0001-Supplemental_syntax.do	Cleans up missing values
04697-0001-Supplemental_syntax.sas	Cleans up missing values

Getting GSS into R: read.spss

```
datraw <- read.spss("04697-0001-Data.sav",
    to.data.frame=T,
                    trim.factor.names=T)
```

- Before 2010-02-26, read.spss() would fail entirely.
- As of 2010-02-26, it succeeds, but gives warnings:

```
re-encoding from CP1252
There were 24 warnings (use warnings() to see them)
> warnings()
Warning messages:
1: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable TVRELIG
```

Getting GSS into R: read.spss ...

```
2: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable SEI
3: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable FIRSTSEI
4: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable PASEI
5: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable MASEI
```

Getting GSS into R: read.spss ...

```
6: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 99.9 for variable SPSEI
7: In read.spss("04697-0001-Data.sav",
    to.data.frame = T, trim.factor.names = T) :
  04697-0001-Data.sav: File contains duplicate
     label for value 0.75 for variable YEARSJOB
8: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]] :
  longer object length is not a multiple of shorter
      object length
9: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]] :
  longer object length is not a multiple of shorter
      object length
10: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
```

Getting GSS into R: read.spss ...

```
  longer object length is not a multiple of shorter
      object length
11: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
12: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
13: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
14: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
```

Getting GSS into R: read.spss ...

```
  longer object length is not a multiple of shorter
      object length
15: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
16: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
17: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
18: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
```

Getting GSS into R: read.spss ...

```
  longer object length is not a multiple of shorter
      object length
19: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
20: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
21: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
  longer object length is not a multiple of shorter
      object length
22: In xi >= z[1L] | xi <= z[2L] | xi[xi == z[3L]]
    :
```

Getting GSS into R: read.spss ...

```
  longer object length is not a multiple of shorter
      object length
23: In `levels<-`(`*tmp*`, value = c("NA", "DK", "
   NA")) :
  duplicated levels will not be allowed in factors
     anymore
24: In `levels<-`(`*tmp*`, value = c("1ADLT,0KIDS",
    "1ADLT,1+KIDS",  ... :
  duplicated levels will not be allowed in factors
     anymore
```

What Do I Do If I See That?

- Snoop in the data, compare to Codebook
- Create a number-only version of the data set by turning off the labels to factors converter

```
datnum <- read.spss("04697-0001-Data.sav",
    use.value.labels = FALSE,
                    to.data.frame=T,
                        trim.factor.names=T)
```

- Compare values:

```
table(datraw$SEX, datnum$SEX)
```

- Usually, you find the information is read in "correctly", in the sense that you can recode to get what you need.

Got a Bull By the Horns? Or Does It Have You?

This data set includes responses from several thousand people in each of 26 surveys
In each survey, there will be 1500-4300 respondents.
Laptop can't handle this file (even with 2GB memory)

Imagine a spreadsheet with 51020 rows and 5137 columns

year	id	Q1	Q2	Q3	Q4	Q5	···	Q5135
1972	1	3	1	NAP	NAP	NAP		3
1972	2	2	2	NAP	NA	NAP		2
⋮								
1974	3455	3	3	5	NAP	NAP		NA
1974	3456	1	2	6	NAP	NAP		NA
⋮								
2006	44555	NAP	4	NAP	1	4		3
2006	44556	NAP	5	NAP	2	3		1

NAP: question not included in survey for that year or for that respondent

NAs: ordinary missings are also observed

Obviously, One Must Select Some Information

- Select some variables, and compare over time
- Select one year, study some variables

"SEXFREQ" Sounds More Interesting Than It Really Is

- SEXFREQ : About how often did you have sex during the last 12 months?

Val	Label	1989	1990	⋮	1994	⋮	2006
0	not at all	298	110		563		595
1	once or twice	99	39		188		205
2	once a month	114	57		290		265
3	2-3 times a month	221	91		416		361
4	weekly	258	110		483		343
5	2-3 per week	307	108		538		430
6	4+ per week	64	37		155		134
8	don't know	0	0		3		6
-1/9	NAP/NA	136/40	199/621		201/155		2096/75
	Valid N	1361	552		2533		2333

- Never included in surveys before 1989
- Asked of *some* respondents in other years

Consider 2006

- Drop all of those other years

```
dat2 <- datraw[datraw$YEAR==2006, ]
```

There's a Whole Lot of Nothin' In There

- The GSS is a cumulative project, accumulating questions.
- There are many "unasked questions" in any particular year

Get Rid of Variables That are Mostly "NAP"

```
### sapply a function that counts NAP in each
    variable
### result sout is a vector with an NAP count for
    each column
sout <- sapply( dat2, function(x) length(x[x=="
    NAP"]))
### which variables have less than 4000 cases NAP
wsout <- which(sout < 4000)
### choose those variables
datnew <- dat2[, wsout]
### Save to R data frame
save(datnew, file="gss-subset1.Rda")
```

- examples folder has copies of large and small gss extracts along with codebook information

Voter Participation in 2006

Cell Contents		
N N / Table Total		
VOTED	DID NOT VOTE	INELIGIBLE
1826 0.623	715 0.244	389 0.133

Men and Women Really Are The Same!

Voter Participation by Sex in 2006

	MALE	FEMALE
VOTED	61%	63%
DID NOT VOTE	25	24
INELIGIBLE	14	13
REFUSED TO ANSWER	0	0
N	1273	1657

Note: This is LaTeX table output from the memisc function `toLatex()`.

Outline

The European Social Survey

- ESS is a large, multinational project
- `http://ess.nsd.uib.no/ess`
- Free download allowed after registration
- Students asked for help with these Files:

```
ESS3e03_2.spss.zip

ESS2e03_1.spss.zip
```

Try the Standard Approach

```
d2 <- read.spss("ESS3e03_2.por", to.data.frame =
    TRUE)
```

```
### BIG DISASTER, all variables are missing.
    Observe

#> table(d2$UEMPLAP)
#
#Not marked      Marked
#          0           0
```

Huh? Try Again, This Time More Slowly

```
d2 <- read.spss("ESS3e03_2.por")
```

Inspect, then coerce into a data frame!

```
d2 <- as.data.frame(d2)
```

```
> table(d2$UEMPLAP)

Not marked     Marked
     42336        664
```

Is Something Rotten in the State of Denmark? (Probably Not

- re-read data as numeric, compare 2 data frames
- Import data into PSPP and compare.

Students want to compare data from 2 years of ESS

- Create a marker for the first survey

```
d2$whichSurvey <- 2
```

- Get the other one

```
d3 <- read.spss("ESS2e03_1.por")
d3 <- as.data.frame(d3)
d3$whichSurvey <- 3
```

Putting 2 Years of ESS Together

- We want to "stack those surveys together" into one data frame.
- But they don't have all of the same variables
- Couldn't figure how to make merge() do this, so take the brute force method:
  "reorganize" the columns by name and then use rbind.

```
namesd2 <- names(d2)
namesd3 <- names(d3)
commonNames <- intersect( namesd3, namesd2)
combod23 <- rbind(d2[ , commonNames], d3[,
    commonNames])
save(combod23, file = "combod23.Rda")
```

Funny Factor Problem (again)

- The question HAPPY is scored on a 0-10 point scale.
- The SPSS data has value labels for some values

```
0: Extremely unhappy
10: Extremely happy
11: Refusal to answer
12: Don't know
13: No answer
```

Wow! Here's a Problem

```
levels(combod23$HAPPY)
```

```
 [1] "Extremely unhappy" "1"                    "2"
 [4] "3"                 "4"                    "5"
 [7] "6"                 "7"                    "8"
[10] "9"                  "Extremely happy"      "
    Refusal"
[13] "Don't know"         "No answer"
```

That's wrong on several levels

- It is not a numeric variable
- Off by one: R gives internal value 1 to the outcome that SPSS called 0, and so forth.

How To Fix HAPPY?

- Create a new variable to play with

```
combod23$HAPPY2 <- combod23$HAPPY
```

- Change Extremely Unhappy to text "0"

```
levels(combod23$HAPPY2)[1] <- "0"
```

- Rename level Extremely Happy to "10"

```
levels(combod23$HAPPY2)[11] <- "10"
```

- Get a copy of the levels from HAPPY2

```
happyLevels <- levels(combod23$HAPPY2)
```

How To Fix HAPPY? ...

```
> happyLevels

 [1] "0"    "1"    "2"    "3"    "4"
 [6] "5"    "6"    "7"    "8"    "9"
[11] "10"          "Refusal"    "Don't know" "No
     answer"
```

- Set NA on levels 12, 13, and 14

```
combod23$HAPPY2[combod23$HAPPY2 %in%
    happyLevels[12:14] ] <- NA
## Check result
table(combod23$HAPPY, combod23$HAPPY2)
```

- Eliminate the unused levels from HAPPY2

```
combod23$HAPPY2 <- factor(combod23$HAPPY2)
```

How To Fix HAPPY? ...

- Use this "factor trick". It will convert a factor that has numeric-looking levels back to a numeric variable:

```
combod23$HAPPYN <- as.numeric(happyLevels)[
    combod23$HAPPY2]
```

- I'm delighted to report there are many happy Europeans

```
table(combod23$HAPPYN)
```

```
   0    1    2    3    4    5
 756  731 1424 2457 2974 10336

   6    7    8    9   10
8119 15923 23860 13838 9447
```

Outline

Professor Putnam is not "Bowling Alone" Anymore

- Robert Putnam's famous Social Capital Benchmark Survey
- `http://www.ropercenter.uconn.edu/data_access/data/datasets/social_capital_community_survey.html`
- Free for download after registration
- An SPSS "portable" file is offered: usmisc2000-soccap.por

I Have Never Succeeded with read.spss and SCBS

- The "character encoding" of some variable names is unacceptable to my operating system
- Error looks like this:

```
Error in read.spss("usmisc2000-soccap.por",
    to.data.frame = T) :
  error reading portable-file dictionary
In addition: Warning message:
In read.spss("usmisc2000-soccap.por", to.data.frame
    = T) :
  Duplicate label for value 0.5 for variable
     WWWTIME
```

There Are Several Workarounds

- PSPP version 0.6 or higher can import and then re-save to SPSS "sav" format, which R can open.
- GRETL also can open the SPSS por file and save as an R data object file.
- The "memisc" package (by Martin Elff) has its own SPSS importing routines (author has incorporated fixes to allow importation of troublesome encodings like this one)

memisc has a nice "codebook" feature

```
library(memisc)
scbs <- spss.portable.file("usmisc2000-soccap.por")
## shows var names & descriptions
description(scbs)
## Generates a codebook
codebook(scbs)
## Causes the data to actually be read
scbsdat <- as.data.frame(scbs)
## If only need a few variables, do this instead
##scbsdat <- subset(scbs, select = c(tr2nei,tr2cop, effcom, polknow
    ))
```

The memisc description output looks like this

```
belrel    '5D. Your place of worship gives you a sense of community'
belwrk    '5E. The people you work with or go to school with give
    you a sense of community'
beleth    '5F. People who share your ethnic background give you a
    sense of community'
belcom    '5G. People you have met online give you a sense of
    community'
trust     '6. Whether most people can be trusted or...'
```

The memisc Codebook Output Is Excellent

```
trust '6. Whether most people can be trusted or...'
```

storage.mode double
measurement nominal

		Values and labels	N	Percent	
-9		'Other (specify)'	0	0.0	0.0
-7		'Don't know'	0	0.0	0.0
-6		'Refused'	0	0.0	0.0
-5		'No Answer'	0	0.0	0.0
-4		'Blank'	0	0.0	0.0
1		'People can be trusted'	14502	50.0	49.6
2		'You can't be too careful'	12612	43.5	43.1
3		'(VOLUNTEERED) Depends'	1902	6.6	6.5
8	M	'Don't know'	174		0.6
9	M	'Refused'	43		0.1

Outline

Arab Barometer

- `http://www.arabbarometer.org/survey/survey.html`
- Includes Countries (with Frequencies):

Country	freq
Jordan	1143
Palestine	1270
Algeria	1300
Morocco	1277
Kuwait	750
Lebanon	1200
Yemen	1182

Data provided in SPSS format

- File name: "AB_10-23-2009_Eng_for_deposit.sav"
- Rename to eliminate spaces (spaces are EVIL)
- To my shock and amazement, R-2.15 gracefully imports this data:

```
library(foreign)
dat <- read.spss("examples/
    AB_10-23-2009_Eng_for_deposit.sav",
    to.data.frame = TRUE)
```

There is a message like this, but otherwise no drama

```
re-encoding from latin1
```

And it worked!

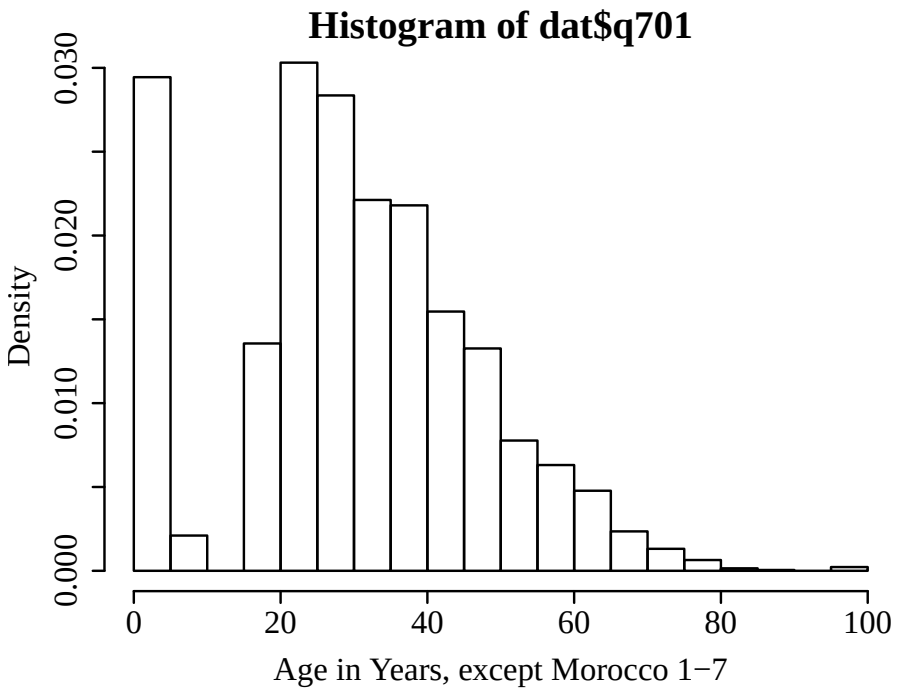
Recoding Problems: Country Subsets

- The survey in Morocco was administered differently, resulting in a complicated problem of different measurements for data reported in the same variables.
- For example, age is recorded in years in all of the countries except Morocco.
- In Morocco, a categorical value was assigned for age. The categories were assigned thusly:

Value	range	freq
1	18-24	269
2	25-34	361
3	35-44	261
4	45-54	182
5	55-64	117
6	65-74	62
7	75+	23

The strange histogram for age

```
hist(dat$q701, breaks = 20, prob = TRUE, xlab = "
    Age in Years, except Morocco 1–7")
```



Create Country Subsets

- It is easy to "grab" separate chunks for the different countries.
- datm is the data for respondents in the country Morocco

```
datm <- dat[dat$country=="Morocco", ]
```

- dato is the data for respondents in all of the other countries

```
dato <- dat[! dat$country=="Morocco", ]
```

- We want to re-work the age variable so that it is the same in both data frames, then we will stack them back together.

Morocco is the easy one

- All we need to do is turn variable datm$q701 into a factor with the proper levels.

```
datm$q701f <- factor(datm$q701, levels=1:7,
    labels=c("18-24", "25-34", "35-44","45-54",
    "55-64","65-74","75+"))
table(datm$q701f, datm$q701)
```

	1	2	3	4	5	6	7
18-24	269	0	0	0	0	0	0
25-34	0	361	0	0	0	0	0
35-44	0	0	261	0	0	0	0
45-54	0	0	0	182	0	0	0
55-64	0	0	0	0	117	0	0
65-74	0	0	0	0	0	62	0
75+	0	0	0	0	0	0	23

How to match age for the other countries to that?

- R's cut function can manufacture a categorical variable from a numeric variable by "collapsing" ranges.

```
dato$q701f <- cut(dato$q701, breaks <- c(17,
    24, 34, 44, 54, 64, 74, 110),
 labels=c("18-24", "25-34", "35-44", "45-54",
     "55-64", "65-74", "75+"))
table(dato$q701f)
```

```
18-24 25-34 35-44 45-54 55-64 65-74   75+
 1508  2066  1578   946   445   210    55
```

Finish Up

- Then row bind datm and dato back together

```
dat2 <- rbind(dato, datm)
table(dat2$q701f, dat2$country)
```

	Jordan	Palestine	Algeria	Morocco	Kuwait	Lebanon	Yemen
18−24	244	258	355	269	205	200	246
25−34	326	337	458	361	182	292	471
35−44	275	319	228	261	151	317	288
45−54	152	192	134	182	110	239	119
55−64	88	95	73	117	71	86	32
65−74	47	46	28	62	31	57	1
75+	10	15	19	23	0	9	2

Merge puts together data frames

- SCBS has data on individuals in communities
- We can separately gather data on the communities
- Use R's merge function to put them together
- Many "hierarchical" or "multi-level" models will be managed in this way

Wide Versus Long Data

- Wide Format: one column for observations on each unit

year	canada	usa	denmark	belgium
2000	44	43	55	22
2001	23	33	33	33
2002	11	22	42	44
2003	19	11	27	55

Some Stat Tools Need the Long Format Instead

year	country	depvar
2000	canada	44
2001	canada	23
2002	canada	11
2003	canada	19
2000	usa	43
2001	usa	33
2002	usa	22
2003	usa	11

Why The Difference?

- Some R tools work with the "wide" format (e.g., matplot)
- Most statistical models expect data in the "long" format

Going Between the Wide and the Long Formats

- See ?reshape
- See ?stack

Here's An Example Recently Encountered

- Given 3 columns, "lat", "lon" and "val"

  lat latitude
  lon longitude
  val CO_2 Observations at position

lat	lon	val
0	0	44
0	1	23
0	2	11
1	0	19
1	1	43
1	2	33
2	0	22
2	1	11
2	2	18

We Need a Matrix like so:

		lat		
		0	1	2
lon	0	44	23	11
	1	19	43	33
	2	22	11	18

The "reshape" package to the Rescue

- As it so often happens, Hadley Wickham has encountered this problem and he's written an R package especially designed to help with it.

```
> library(reshape)
> nd
  lat lon         val
1   1   1  0.88372864
2   1   2  1.46431589
3   1   3 -0.68027313
4   1   4  0.07207392
5   1   5  0.70322429
6   2   1 -0.35695088
7   2   2  0.16997932
8   2   3  0.43363888
9   2   4  1.74066891
```

The "reshape" package to the Rescue ...

```
10    2    5 -1.66335276

> mnd <- melt(nd, id=c("lat","lon"))
> weWant <- cast(mnd, lat ~ lon ~ variable)
> weWant
, , variable = val

    lon
lat           1          2          3          4
              5
  1  0.8837286 1.4643159 -0.6802731 0.07207392  0
     .7032243
  2 -0.3569509 0.1699793  0.4336389 1.74066891
     -1.6633528
```

The help page on ?cast has a ton of examples,