Import Data

Paul E. Johnson[1] [2]

[1]Department of Political Science

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

2018

KU

Outline

KU

Outline

KU

Data Input Formats

- Base R(R Core Team, 2017) includes importers for input data files that are saved in 2 formats
  1. R Serialization Data structures ("rds" files)
  2. text files ("csv" comma separated, but also "tab" or other separators).
- R packages can be loaded to import files from
  - SPSS, Stata, SAS
  - Excel
  - SQL
  - Others

What is the difficult part?

- Every program uses a specialized format
- Commercial software often does not publish guidelines (or follow their own published guidelines)
- There is tension within the R community on the question of how hard R programmers should work in order to cooperate with other storage formats.

KU

Outline

KU

In the R folder for this workshop

- A file named "import.R" is saved in the R directory.
- Open that file with Rstudio, Emacs, Notepad++, or R.app
- Make sure of the current working directory is correct and that the data is where we expect

```
getwd()
```

- I'm writing this presentation in the top level folder "summeR-2-2-import", but you should be running inside R because that's where import.R is located.
- From import.R, inside the R directory, the location of data is

```
ddir <- "../data"
```

```
list.files(ddir)
```

Quick test 1: RDS

- I saved a copy of a data frame named "swiss" in a file in the data folder. Lets load it.

```
swiss <- readRDS( file.path(ddir, "swiss.rds") )
```

```
head(swiss)
```

```
             Fertility Agriculture Examination Education
Courtelary        80.2        17.0          15        12
Delemont          83.1        45.1           6         9
Franches-Mnt      92.5        39.7           5         5
Moutier           85.8        36.5          12         7
Neuveville        76.9        43.5          17        15
Porrentruy        76.1        35.3           9         7
             Catholic Infant.Mortality
Courtelary       9.96             22.2
Delemont        84.84             22.2
Franches-Mnt    93.40             20.2
Moutier         33.77             20.3
Neuveville       5.16             20.6
Porrentruy      90.57             26.6
```

Quick test 1: RDS ...

As we will see, this is a "data frame", but RDS is a general purpose storage format (R Data Serialization) which can save one object of any type.

Quick test 2: a text file

1. `read.table()` can import text files (*.txt, *.csv).
2. Example usage

```
fn <- file.path(ddir, "oregon.csv")
ore1 <- read.table(fn, sep = ",", header =
    TRUE)
```

Output named "`ore1`" (NB: use brief names for frequently used objects).

KEY ARGUMENTS

1. the file name, an un-named first argument
2. **sep**: the separator's symbol
3. **header**: is the first row to be used as column names?

3. See `?read.table` for additional arguments, especially `stringsAsFactors` (will discuss below)
4. The returned thing is a "`data.frame`" object.
5. There are specialized "wrapper" versions of same, like `read.csv`, that have default settings which may help.

Outline

Data Frame

- R has many different kinds of data structures.
  - matrix
  - list
  - vector
  - data frame
- For importing data, we often focus on data frames because that format is closest to format used by other programs.

What is a Data Frame?

- Definition: elements have same number of rows.
- Typically
  - Columns are "Variables"
  - Rows are "Observations"

Example: The General Social Survey: 51020 rows 5137 columns

id	year	Q1	Q2	Q3	Q4	Q5	$\cdots$	Q5135
1	1972	3	1	NAP	NAP	NAP		3
2	1972	2	2	NAP	NA	NAP		2
	$\vdots$							
3455	1974	3	3	5	NAP	NAP		NA
3456	1974	1	2	6	NAP	NAP		NA
	$\vdots$							
44555	2006	NAP	4	NAP	1	4		3
44556	2006	NAP	5	NAP	2	3		1

http://pj.freefaculty.org/guides/stat/DataSets/GSS-Overview/GSS-1.pdf

R also has matrices

- A matrix is a 2 dimensional array.
- Lets create a matrix with the matrix() function

```
X <- matrix(1:24, nrow = 6, ncol = 4, dimnames =
    list(NULL, c("Den", "KC", "Oak", "SD")))
X
```

```
     Den KC Oak SD
[1,]   1  7  13 19
[2,]   2  8  14 20
[3,]   3  9  15 21
[4,]   4 10  16 22
[5,]   5 11  17 23
[6,]   6 12  18 24
```

- A "cell" in that matrix can be extracted as

```
X[5, 2]
```

R also has matrices ...

```
KC
11
```

- The 2nd row can be extracted as

```
X[2, ]
```

```
Den  KC Oak  SD
  2   8  14  20
```

- The 3rd column:

```
X[ , 3]
```

```
[1] 13 14 15 16 17 18
```

- R includes a raft of specialized matrix algebra functions (a topic for summeR-4).

Compare/Contrast matrix & data frame?

Similarities

- Both allow row, column index access to values. It is unusual to do this with a data frame, but the values can be accessed by index number, just like matrices.

```
swiss[3, ]
```

```
             Fertility Agriculture Examination Education Catholic
                 Infant.Mortality
Franches-Mnt        92.5        39.7           5         5     93.4
                  20.2
```

- Both make it possible to name the columns (or rows) by characters, which can be used to retrieve that column.

```
X[1:5 , "Oak"]
```

```
[1] 13 14 15 16 17
```

Compare/Contrast matrix & data frame? ...

or a pair of columns

```
X[1:5 , c("Den", "SD")]
```

```
     Den SD
[1,]   1 19
[2,]   2 20
[3,]   3 21
[4,]   4 22
[5,]   5 23
```

- Both allow a special symbolic value **NA** to represent a missing score

Differences

- A matrix must be filled with values on one single type
  - floating point numbers
  - characters:
  - logical values:
- A data frame allows columns of different types

Compare/Contrast matrix & data frame? ...

- Matrices are used for serious computations, generally faster than data frames
- **But** the fundamental limitation of a matrix: values are only of one type.

Data frame advantages

Data Frames are slower. Why bother?

Answer: Convenience.

1. Diverse column types (characters, integers, real numbers, factors, dates, etc.)
2. Shortcut "$" for access to individual columns
   1. A column within a matrix (X) OR data frame (DF) can be accessed by name as

```
X[ , "column_name_here"]
DF[ , "column_name_here"]
```

   2. But a data frame allows shorter syntax like this

```
DF$column_name_here
```

3. Data interchange. Excel, SPSS, Stata allow "data frame" style diversity.

KU

swiss, for example

- Make sure that thing is a data.frame

```
is.data.frame(swiss)
```

```
[1] TRUE
```

- Its a small data frame

```
print(swiss)
```

```
                Fertility Agriculture Examination Education Catholic
                    Infant.Mortality
Courtelary           80.2        17.0          15        12     9.96
                   22.2
Delemont             83.1        45.1           6         9    84.84
                   22.2
Franches-Mnt         92.5        39.7           5         5    93.40
                   20.2
Moutier              85.8        36.5          12         7    33.77
                   20.3
Neuveville           76.9        43.5          17        15     5.16
                   20.6
```

swiss, for example ...

```
             Porrentruy     76.1 35.3  9  7 90.57 26.6
                 Broye      83.8 70.2 16  7 92.85 23.6
                 Glane      92.4 67.8 14  8 97.16 24.9
10           Gruyere      82.4 53.3 12  7 97.67 21.0
                 Sarine     82.9 45.2 16 13 91.38 24.4
                 Veveyse    87.1 64.5 14  6 98.61 24.5
                 Aigle      64.1 62.0 21 12  8.52 16.5
                 Aubonne    66.9 67.5 14  7  2.27 19.1
15           Avenches     68.9 60.7 19 12  4.43 22.7
                 Cossonay   61.7 69.3 22  5  2.82 18.7
                 Echallens  68.3 72.6 18  2 24.20 21.2
                 Grandson   71.7 34.0 17  8  3.30 20.0
                 Lausanne   55.7 19.4 26 28 12.11 20.2
```

swiss, for example ...

```
20 La Vallee              54.3       15.2       31       20        2.15
                          10.8
   Lavaux                 65.1       73.0       19        9        2.84
                          20.0
   Morges                 65.5       59.8       22       10        5.23
                          18.0
   Moudon                 65.0       55.1       14        3        4.52
                          22.4
   Nyone                  56.6       50.9       22       12       15.14
                          16.7
25 Orbe                   57.4       54.1       20        6        4.20
                          15.3
   Oron                   72.5       71.2       12        1        2.40
                          21.0
   Payerne                74.2       58.1       14        8        5.23
                          23.8
   Paysd'enhaut           72.0       63.5        6        3        2.56
                          18.0
   Rolle                  60.5       60.8       16       10        7.72
                          16.3
30 Vevey                  58.3       26.8       25       19       18.46
                          20.9
   Yverdon                65.4       49.5       15        8        6.10
                          22.5
   Conthey                75.5       85.9        3        2       99.71
                          15.1
```

swiss, for example ...

```
                Entremont 69.3 84.9  7  6 99.68 19.8
                Herens    77.3 89.7  5  2 100.00 18.3
35              Martigwy  70.5 78.2 12  6 98.96 19.4
                Monthey   79.4 64.9  7  3 98.22 20.2
                St Maurice 65.0 75.9  9  9 99.06 17.8
                Sierre    92.2 84.6  3  3 99.46 16.3
                Sion      79.3 63.1 13 13 96.83 18.1
40              Boudry    70.4 38.4 26 12  5.62 20.3
                La Chauxdfnd 65.7  7.7 29 11 13.79 20.5
                Le Locle  72.7 16.7 22 13 11.22 18.9
                Neuchatel 64.4 17.6 35 32 16.92 23.0
                Val de Ruz 77.6 37.6 15  7  4.97 20.0
45              ValdeTravers 67.6 18.7 25  7  8.65 19.5
```

swiss, for example ...

```
V. De Geneve        35.0          1.2          37          53       42.34
                    18.0
Rive Droite         44.7         46.6          16          29       50.43
                    18.2
Rive Gauche         42.8         27.7          22          29       58.33
                    19.3
```

Where to find out more?

In Fall, 2016 I put together some new notes that are available in

http://pj.freefaculty.org/guides/Rcourse/data_structures

- vectors
- matrices
- data.frames
- lists

Functions to memorize

Functions to quickly overview the imported material.

head()	See the first few lines (Similar **tail()**)
summary()	A generic R function that **usually** tells you something informative.
str()	Structure summary
View()	Graphic
colnames()	All df will have colnames, perhaps bland like "V1" "V2"
rownames()	If creator does not assign rownames, R will use the integer row numbers in character format.
dim()	quick way to find out how many rows and columns there are
attributes()	R uses "attributes" in programming, sometimes this is vital in troubleshooting

1. `head()` shows the first 5 observations of each variable (could ask for more)

Functions to memorize ...

```
head(swiss)
```

```
             Fertility Agriculture Examination Education Catholic
                  Infant.Mortality
Courtelary        80.2        17.0          15        12     9.96
                 22.2
Delemont          83.1        45.1           6         9    84.84
                 22.2
Franches-Mnt      92.5        39.7           5         5    93.40
                 20.2
Moutier           85.8        36.5          12         7    33.77
                 20.3
Neuveville        76.9        43.5          17        15     5.16
                 20.6
Porrentruy        76.1        35.3           9         7    90.57
                 26.6
```

2. `str()` is a structural overview

Column 1 will be variable name

Column 2 is variable type

Functions to memorize ...

```
str(swiss)
```

```
'data.frame': 47 obs. of  6 variables:
 $ Fertility        : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4
     82.4 82.9 ...
 $ Agriculture      : num  17 45.1 39.7 36.5 43.5 35.3 70.2 67.8
     53.3 45.2 ...
 $ Examination      : int  15 6 5 12 17 9 16 14 12 16 ...
 $ Education        : int  12 9 5 7 15 7 7 8 7 13 ...
 $ Catholic         : num  9.96 84.84 93.4 33.77 5.16 ...
 $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6 24.9
     21 24.4 ...
```

- This example has: some "`integer`" variables and some "`numeric`".
  - No `factor`, `character`, `logical`, or `date` variables.
- Add a logical (TRUE/FALSE) variable, then test `str`

```
swiss$Edgt10 <- ifelse(swiss$Education >
    10, TRUE, FALSE)
str(swiss)
```

Functions to memorize ...

```
'data.frame': 47 obs. of  7 variables:
 $ Fertility        : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8
     92.4 82.4 82.9 ...
 $ Agriculture      : num  17 45.1 39.7 36.5 43.5 35.3 70.2
     67.8 53.3 45.2 ...
 $ Examination      : int  15 6 5 12 17 9 16 14 12 16 ...
 $ Education        : int  12 9 5 7 15 7 7 8 7 13 ...
 $ Catholic         : num  9.96 84.84 93.4 33.77 5.16 ...
 $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6
     24.9 21 24.4 ...
 $ Edgt10           : logi  TRUE FALSE FALSE FALSE TRUE FALSE
     ...
```

3. If you are a graphic-sort-of-person, try View().

```
View(swiss)
```

Functions to memorize ...

Data: swiss

	row.names	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
1	Courtelary	80.2	17.0	15	12	9.96	22.2
2	Delemont	83.1	45.1	6	9	84.84	22.2
3	Franches-Mnt	92.5	39.7	5	5	93.40	20.2
4	Moutier	85.8	36.5	12	7	33.77	20.3
5	Neuveville	76.9	43.5	17	15	5.16	20.6
6	Porrentruy	76.1	35.3	9	7	90.57	26.6
7	Broye	83.8	70.2	16	7	92.85	23.6
8	Glane	92.4	67.8	14	8	97.16	24.9
9	Gruyere	82.4	53.3	12	7	97.67	21.0
10	Sarine	82.9	45.2	16	13	91.38	24.4
11	Veveyse	87.1	64.5	14	6	98.61	24.5
12	Aigle	64.1	62.0	21	12	8.52	16.5
13	Aubonne	66.9	67.5	14	7	2.27	19.1
14	Avenches	68.9	60.7	19	12	4.43	22.7
15	Cossonay	61.7	69.3	22	5	2.82	18.7
16	Echallens	68.3	72.6	18	2	24.20	21.2

4. `summary()`. The builtin summary function for data frames

```
summary(swiss)
```

Functions to memorize ...

```
  Fertility        Agriculture       Examination       Education
               Catholic          Infant.Mortality
Min.   :35.00   Min.   : 1.20   Min.   : 3.00   Min.   : 1.00
     Min.   :  2.150   Min.   :10.80
1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00   1st Qu.: 6.00
     1st Qu.:  5.195   1st Qu.:18.15
Median :70.40   Median :54.10   Median :16.00   Median : 8.00
     Median : 15.140   Median :20.00
Mean   :70.14   Mean   :50.66   Mean   :16.49   Mean   :10.98
     Mean   : 41.144   Mean   :19.94
3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00   3rd Qu.:12.00
     3rd Qu.: 93.125   3rd Qu.:21.70
Max.   :92.50   Max.   :89.70   Max.   :37.00   Max.   :53.00
     Max.   :100.000   Max.   :26.60
  Edgt10
Mode :logical
FALSE:30
TRUE :17
```

5. Other functions to try

Functions to memorize ...

```
rownames(swiss)
colnames (swiss)
attributes(swiss)
```

Various Column Access Methods

- Different ways to choose the fertility column

```
# The $ accessor
f1 <- swiss$Fertility
# matrix style, choose column 1
f2 <- swiss[ , 1]
# By column name (my favorite)
f3 <- swiss[ , "Fertility"]
all.equal(f1, f2, f3)
```

```
[1] TRUE
```

- The column number method is NOT widely used (danger of counting incorrectly).

Various Column Access Methods ...

- Sidenote: When you choose one column, the resulting selection is "demoted" from "data frame" to "vector". This is an important "feature" in R, also a source of my frequent frustration in writing R functions (see blog post "R's drop 'gotcha' and the diag 'curse'" http://pj.freefaculty.org/blog/?p=274)
- Choose several columns

```
# matrix style, cols 1, 3, 5
f2 <- swiss[ , c(1, 3, 5)]
# By column name (my favorite)
f3 <- swiss[ , c("Fertility", "Examination",
    "Catholic")]
all.equal(f2, f3)
```

```
[1] TRUE
```

```
head(f2)
```

Various Column Access Methods ...

```
             Fertility Examination Catholic
Courtelary        80.2          15     9.96
Delemont          83.1           6    84.84
Franches-Mnt      92.5           5    93.40
Moutier           85.8          12    33.77
Neuveville        76.9          17     5.16
Porrentruy        76.1           9    90.57
```

The $ notation is not helpful when you need a few columns

- In larger data sets, when it is tedious to type lots of names, we would use R character string functions to scan values of `colnames(swiss)`. That's a bigger topic.

See what we got from Oregon!

1. `head(ore1, 8)` shows the first 8 observations

```
head(ore1, 8)
```

```
  station latitude longitude elevation tann
1     ANT   44.917  -120.717       846  9.6
2     ARL   45.717  -120.200        96 12.5
3     ASH   42.217  -122.717       543 11.1
4     AST   46.150  -123.883         2 10.3
5     BKA   44.833  -117.817      1027  7.6
6     BKK   44.783  -117.833      1050  8.4
7     BAN   43.117  -124.383        24 10.8
8     BND   44.067  -121.317      1097  7.7
```

2. `str` is a structural overview

```
str(ore1)
```

See what we got from Oregon! ...

```
'data.frame': 92 obs. of  5 variables:
 $ station  : Factor w/ 92 levels "ANT","ARL","ASH",..: 1 2 3 4 7
     8 5 9 6 10 ...
 $ latitude : num  44.9 45.7 42.2 46.1 44.8 ...
 $ longitude: num  -121 -120 -123 -124 -118 ...
 $ elevation: int  846 96 543 2 1027 1050 24 1097 999 18 ...
 $ tann     : num  9.6 12.5 11.1 10.3 7.6 8.4 10.8 7.7 9.5 11.2 ...
```

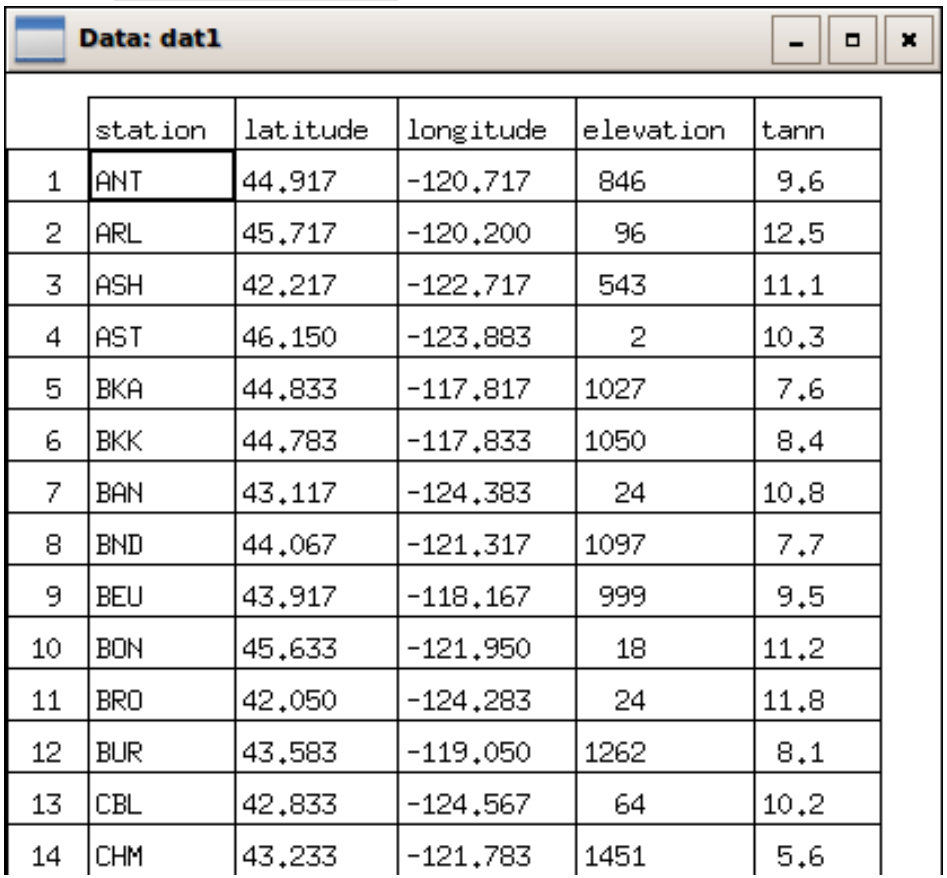
Notice there are different types of variables?

- I'm not happy to see the station name was automatically converted to a factor variable.
  I wish I had run this instead, so they would just be character strings.
  Will explain later.

```
ore1 <- read.table("data/oregon.csv", sep
    = ",",
                header = TRUE,
                   stringsAsFactors = FALSE)
```

See what we got!

3. Run `View(ore1)`



Data: dat1

	station	latitude	longitude	elevation	tann
1	ANT	44.917	-120.717	846	9.6
2	ARL	45.717	-120.200	96	12.5
3	ASH	42.217	-122.717	543	11.1
4	AST	46.150	-123.883	2	10.3
5	BKA	44.833	-117.817	1027	7.6
6	BKK	44.783	-117.833	1050	8.4
7	BAN	43.117	-124.383	24	10.8
8	BND	44.067	-121.317	1097	7.7
9	BEU	43.917	-118.167	999	9.5
10	BON	45.633	-121.950	18	11.2
11	BRO	42.050	-124.283	24	11.8
12	BUR	43.583	-119.050	1262	8.1
13	CBL	42.833	-124.567	64	10.2
14	CHM	43.233	-121.783	1451	5.6

See what we got!

4. summary()

```
summary(ore1)
```

```
   station       latitude       longitude        elevation
                    tann
ANT     : 1   Min.   :42.05   Min.   :-124.6   Min.   :   2.00
    Min.   : 3.200
ARL     : 1   1st Qu.:43.60   1st Qu.:-123.2   1st Qu.:  94.25
    1st Qu.: 8.700
ASH     : 1   Median :44.46   Median :-121.7   Median : 462.00
    Median :10.400
AST     : 1   Mean   :44.32   Mean   :-121.3   Mean   : 556.99
    Mean   : 9.885
BAN     : 1   3rd Qu.:45.28   3rd Qu.:-119.6   3rd Qu.: 863.75
    3rd Qu.:11.200
BEU     : 1   Max.   :46.15   Max.   :-117.0   Max.   :1974.00
    Max.   :12.600
(Other):86
```

5. rockchalk::summarize() has nicer summary, I think

See what we got! ...

```
library(rockchalk)
summarize(ore1)
```

```
Numeric variables
           latitude   longitude   elevation      tann
min          42.050    -124.567       2          3.200
med          44.459    -121.733     462         10.400
max          46.150    -116.967    1974         12.600
mean         44.321    -121.283     556.989      9.885
sd            1.116       2.278     498.276      1.874
skewness     -0.372       0.474       0.634     -1.090
kurtosis     -0.997      -1.040      -0.631      1.062
nobs         92          92          92         92
nmissing      0           0           0          0

Nonnumeric variables
                station
 ANT           : 1
 ARL           : 1
 ASH           : 1
 AST           : 1
 (All Others): 88
 nobs           : 92.00
```

See what we got! ...

```
nmiss          :   0.00
entropy        :   6.52
normedEntropy:   1.00
```

First, check your input file

- Look inside the file to find out what is really in there
- Don't trust the extension (*.txt, *.csv, *.dat).
- File can be named anything, R only cares about content inside.

Comma Separated variables with column header

```
name,id,test1,test2
paul,001,188,99
jane,002,9,33
rick,003,101,13
```

Space Separated variables with column header

```
name id test1 test2
paul 001 188 99
jane 002 9 33
rick 003 101 13
```

- Vital to check for
  - Presence of column names in row 1
  - Column separator: Is it "tab", "comma", "space", or some exotic symbol like "|"

read.table arguments

- Example

```
mydata <- read.table("../data/filename.csv",
    header = TRUE, sep = ",", stringsAsFactors
    = FALSE)
```

- read Arguments in ?read.table

1. file (unnamed argument in my example).
   1. If I use "filename.csv", R looks in current working directory.
   2. If the file is not found, a not-so-vague error message results

```
> read.table('blah.txt')
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'blah.txt': No such file or directory
```

If it's not there, you'll either have to move it in there, or you need to tell R where it is.

read.table arguments ...

2. header.
   1. Column names in row 1.
   2. R sanitizes variable names. Valid names must not
      1. have spaces
      2. begin with numbers
      3. have characters except letters, numbers, or "_" and "."
   3. Wonder how R does that? See "`?make.names`"
3. sep is the "separator".
   1. The default separator in R is the blank space.
   2. Here, I use comma, ","
4. stringsAsFactors.
   1. Factors are important in R stats and graphics, but inconvenient in recoding.
   2. The `read.table` function assumes that you want your character string variables to be converted into R factor variables. (A column of street addresses will get mangled).
   3. Additional read.table argument "stringsAsFactors = FALSE" solves that.

Outline

Access Values in a Data Frame

- Make a little toy data frame

```
set.seed(234)
fd <- data.frame(v3 = rnorm(5),
                 v4 = rnorm(5),
                 v5 = LETTERS[1:5],
                    stringsAsFactors = FALSE)
```

That creates named variables "v3", "v4", and "v5".

- "fd" stands for fun data. (*Subconscious message: statistics with R is fun*!)

```
head(fd)
```

Access Values in a Data Frame ...

```
           v3          v4 v5
1   0.6607697   0.1401390  A
2  -2.0529830   0.2091844  B
3  -1.4992061  -3.0360898  C
4   1.4712331  -0.4869341  D
5   1.4591385  -1.0878673  E
```

- Different ways to choose the second column

```
fd[ , "v4"]
```

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

```
fd$v4
```

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

```
fd[ , 2]
```

Access Values in a Data Frame ...

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

```
fd[ , c(FALSE, TRUE, FALSE)]
```

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

The last one looks dumb now, but it turns out to be useful when choosing rows or columns algorithmically

Col/Row names & Numbers

- Access by column numbers succeeds

```
fd[ , 2]
```

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

- But that's dangerous, name for column probably smarter

```
fd[ , "v4"]
```

```
[1]  0.1401390  0.2091844 -3.0360898 -0.4869341 -1.0878673
```

  - That appears like a row, but it is really one column from the data.frame, pressed flat for printing.
  - I like to leave an empty space before the comma, for readability.
- Remember there's a difference between a row name "1" and a row number 1.

```
fd["1", "v4"]
```

Col/Row names & Numbers ...

```
[1] 0.140139
```

A quoted "1" is the character "1":

```
rownames(fd)
```

```
[1] "1" "2" "3" "4" "5"
```

- Can use non-numeric row names:

```
rownames(fd) <- c("Bill", "Joe", "Frosty",
    "Mickey", "Henry")
```

- After that, the rowname "1" no longer exists

```
fd[1, ] # position 1
```

```
            v3       v4 v5
Bill 0.6607697 0.140139  A
```

Col/Row names & Numbers ...

```
 fd["Bill", ] # rowname "Bill"
```

```
             v3       v4 v5
Bill 0.6607697 0.140139  A
```

but you'll get a error from running: fd[“1”,]

Create new data frames by Extracting Columns

If you choose 2 or more columns, the returned structure is a new data frame

```
fd13 <- fd[ , c("v3", "v5")]
str(fd13)
```

```
'data.frame': 5 obs. of  2 variables:
 $ v3: num  0.661 -2.053 -1.499 1.471 1.459
 $ v5: chr  "A" "B" "C" "D" ...
```

```
head(fd13)
```

```
               v3 v5
Bill    0.6607697  A
Joe    -2.0529830  B
Frosty -1.4992061  C
Mickey  1.4712331  D
Henry   1.4591385  E
```

Default "demotion" of 1 column data.frames

There are some wrinkles when returned object structures are changed. There's one I call the "drop gotcha." See my blog http://pj.freefaculty.org/blog/?p=274.

- Did you notice that this is a vector, not a column?

```
z <- fd[ , "v4"]
str(z)
```

```
num [1:5] 0.14 0.209 -3.036 -0.487 -1.088
```

```
w2 <- fd["v4"]
w2
```

```
                v4
Bill     0.1401390
Joe      0.2091844
Frosty  -3.0360898
Mickey  -0.4869341
Henry   -1.0878673
```

Default "demotion" of 1 column data.frames ...

```
str(w2)
```

```
'data.frame': 5 obs. of  1 variable:
 $ v4: num  0.14 0.209 -3.036 -0.487 -1.088
```

- I know of 2 ways to preventing "demotion" from one-column data frame to vector.

1. Here is the syntax I'm accustomed to using

```
w1 <- fd[ , "v4", drop = FALSE]
w1
```

```
               v4
Bill    0.1401390
Joe     0.2091844
Frosty -3.0360898
Mickey -0.4869341
Henry  -1.0878673
```

Default "demotion" of 1 column data.frames ...

```
str(w1)
```

```
'data.frame': 5 obs. of  1 variable:
 $ v4: num  0.14 0.209 -3.036 -0.487 -1.088
```

If you were thinking "[]" is matrix notation, you are shocked that a named argument `drop` can be inserted after 2 coordinates. I still find that shocking, *after 20 years*

2. Until recently, I was uncomfortable with this simpler approach

```
z <- fd[ , "v4"]
str(z)
```

```
num [1:5] 0.14 0.209 -3.036 -0.487 -1.088
```

```
w2 <- fd["v4"]
w2
```

Default "demotion" of 1 column data.frames ...

```
                v4
Bill     0.1401390
Joe      0.2091844
Frosty  -3.0360898
Mickey  -0.4869341
Henry   -1.0878673
```

```
str(w2)
```

```
'data.frame': 5 obs. of  1 variable:
 $ v4: num  0.14 0.209 -3.036 -0.487 -1.088
```

This treats the data.frame like an R list, which we have not discussed yet.

Deleting a column

- Get rid of a column, permanently
  - Assign it as NULL

```
fd13$v3 <- NULL
```

- Create a new data frame from fd by excluding the unwanted ones
  - the minus sign of a column number is interpreted as "remove this one"

```
fdnew <- fd[ , -c(2)]
head(fdnew)
```

```
                v3 v5
Bill     0.6607697  A
Joe     -2.0529830  B
Frosty -1.4992061  C
Mickey   1.4712331  D
Henry    1.4591385  E
```

  - This works, but is dangerous because it requires good counting. Usually I'd use some text string manipulation to indicate which columns need to be kept.

Outline

What are we Looking For?

- **NO!** publication quality graphics
- **YES!** quick look at each variable, find out what we have
- Check for
  - Illegal scores, measurement noise
  - Missings masquerading as 9999 (or such)
  - Unexpected data distributions

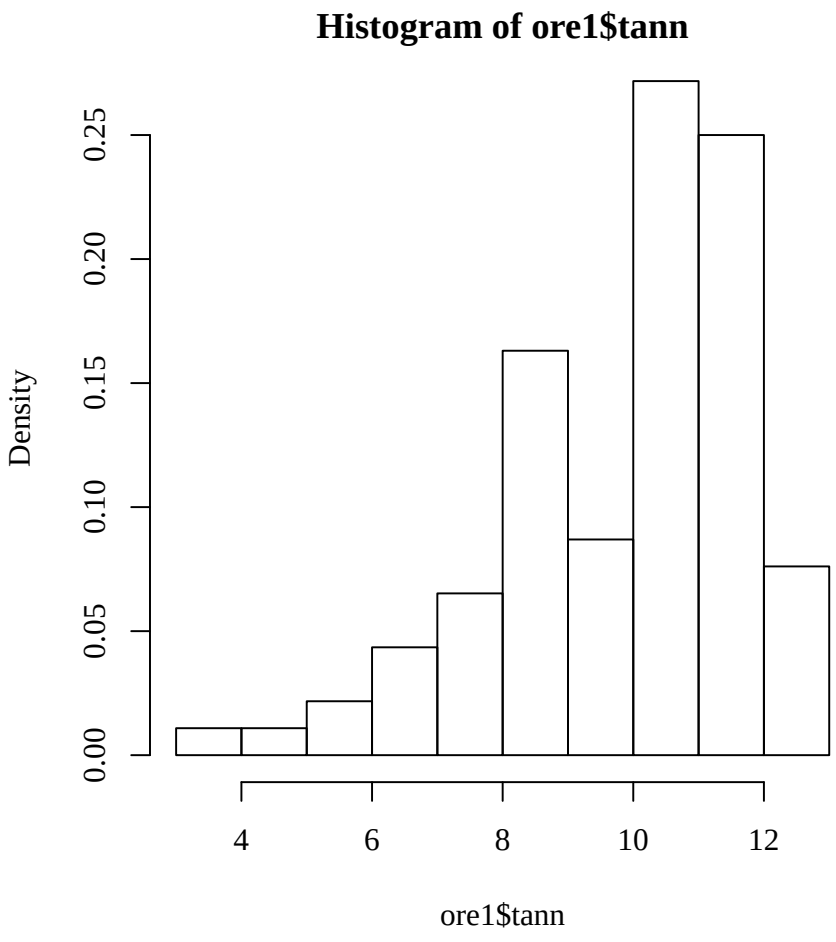
Check Histograms for numeric variables

Can do this one variable at a time (tedious!)

- `hist()` is the base distribution.

```
hist(ore1$tann, prob = TRUE)
```

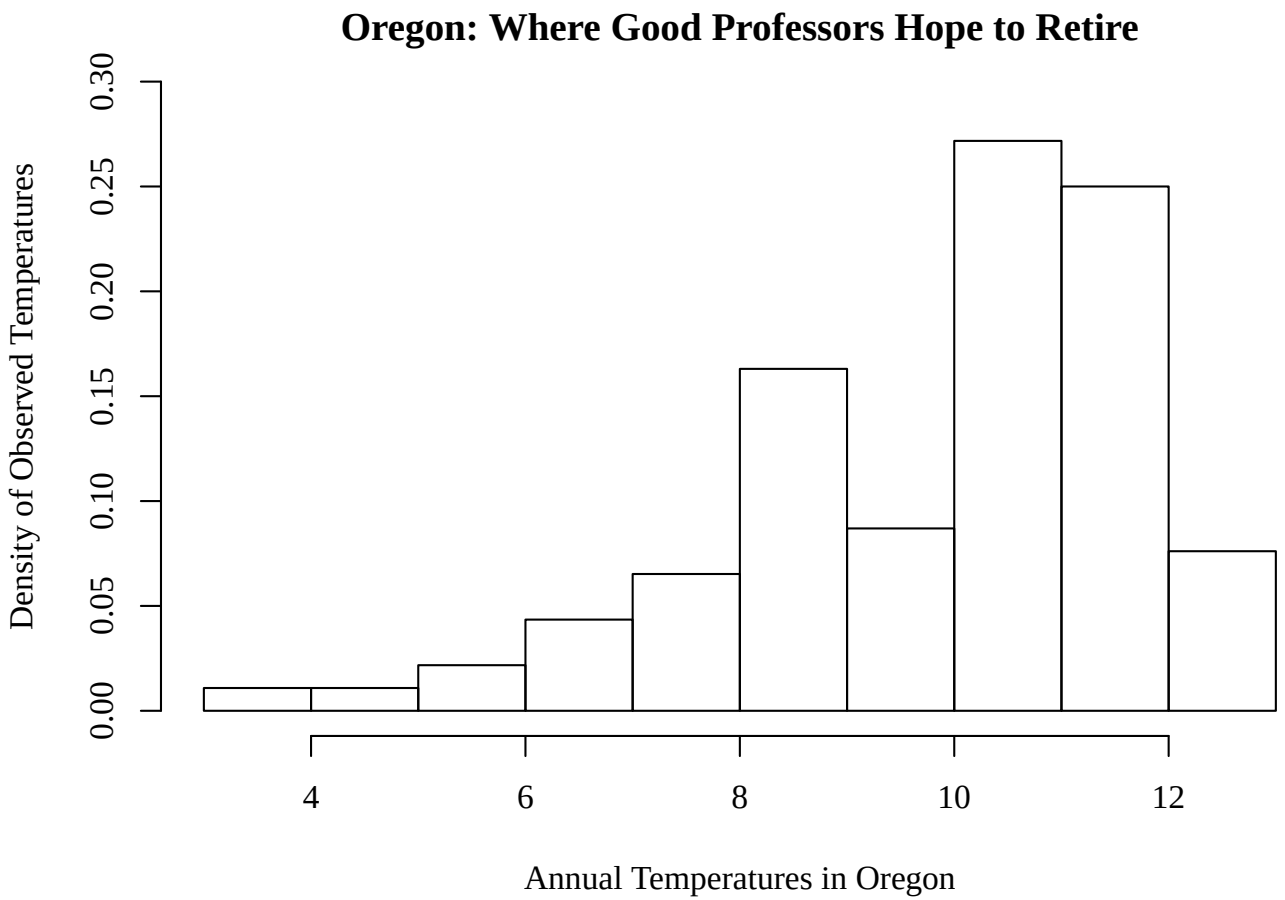
Check Histograms for numeric variables ...



Can beautify labels if you want to show somebody else

```
hist(ore1$tann, prob = TRUE,  xlab = "Annual
    Temperatures in Oregon", ylab = "Density of
    Observed Temperatures", main = "Oregon: Where
    Good Professors Hope to Retire", ylim = c(0,
    0.30))
```

Can beautify labels if you want to show somebody else ...



kutils: peek()

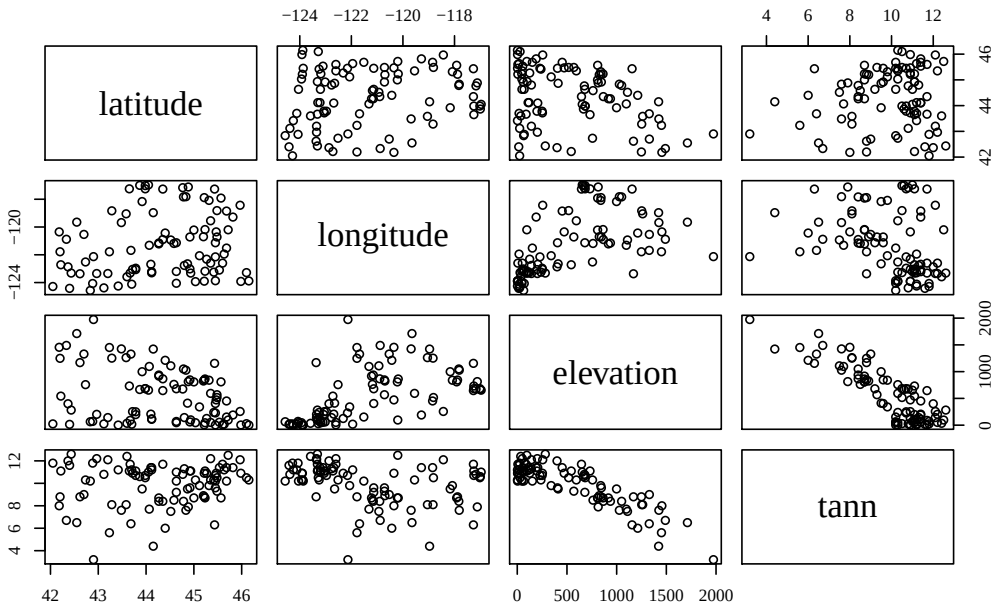
- In the 2016 summer program, we wondered if we could make a quick, automatic, variable-by-variable data display.
- The result: `kutils::peek()` ! Will scan, showing histograms for numeric variables and sideways barplots for discrete variables

```
library(kutils)
peek(ore1)
```

Scatterplot Matrix

- The function `pairs()` is handy with a small data set like this.

```
pairs(ore1[ , -1])
```



Column 1 was the station name, and I exclude that with the "-1"

Outline

Excel data requires more work

- Excel import is not supported by R Core.
- Various packages have been offered by volunteers
  1. Since 2016, our most dependable R package for this is **openxlsx.** Uses routines written in C++, requires Rcpp.
  2. From 2000 until 2015, **gdata** was my favorite.
     1. Still needed for older xls files (openxlsx is only for xlsx)
     2. On huge, simple Excel files, Perl-based routine `gdata` is still faster
  3. Other Excel reading packages exist. Suggestion: avoid packages based on Java

The openxlsx package

- Example

```
library(openxlsx)
fn <- file.path(ddir, "gradebook.xlsx")
grades <- read.xlsx(fn, colNames = TRUE,
    check.names = TRUE)
head(grades)
```

```
  namelast namefirst     id hw1 hw2 hw3 mterm fin grade
1   Potter    Harry 453345   8   7   P    84  77     B
2  Granger  Hermoine 942832   7  NA   P   101 999    A+
3  Weasley      Ron 342234   9   8   F    78  88    B-
4   Malfoy    Draco 666666   8   4   P    66  85    C+
5    Brown  Lavender 234252   6   8   P    87  77     B
6 Longbottom  Neville 789897   5   9   F    81  79    B-
```

The openxlsx package ...

```
    namelast namefirst     id hw1 hw2 hw3 mterm fin grade
1     Potter     Harry 453345   8   7   P    84  77     B
2    Granger  Hermoine 942832   7  NA   P   101 999    A+
3    Weasley       Ron 342234   9   8   F    78  88    B-
4     Malfoy     Draco 666666   8   4   P    66  85    C+
5      Brown  Lavender 234252   6   8   P    87  77     B
6 Longbottom    Neville 789897   5   9   F    81  79    B-
```

```
str(grades)
```

```
'data.frame': 7 obs. of  9 variables:
 $ namelast : chr  "Potter" "Granger" "Weasley" "Malfoy" ...
 $ namefirst: chr  "Harry" "Hermoine" "Ron" "Draco" ...
 $ id       : num  453345 942832 342234 666666 234252 ...
 $ hw1      : num  8 7 9 8 6 5 8
 $ hw2      : num  7 NA 8 4 8 9 7
 $ hw3      : chr  "P" "P" "F" "P" ...
 $ mterm    : num  84 101 78 66 87 81 84
 $ fin      : num  77 999 88 85 77 79 66
 $ grade    : chr  "B" "A+" "B-" "C+" ...
```

The openxlsx package ...

CAUTION: read.xlsx has unusual argument names

- colNames : TRUE or FALSE for 1st row is a header
- startRow = n. Row number where material to be imported begins.

Does not "check.names" by default

The openxlsx package ...

- Recall: R's `read.table()` checked column names and converted them to legal variable names
- `open.xlsx()` does not "check.names" by default.
- Allows column names that are *illegal*
- Can Fix them yourself
  - edit `colnames` manually, or
  - run `make.names()` on `colnames(your_data_frame)`
- Newer `openxlsx` introduced argument

```
check.names = TRUE
```

- Note: `read.xlsx()` does not automatically convert character variables to R factors.

Outline

R Foreign package

- The R standard distribution includes the foreign package.
- foreign includes
  - import for Stata through Stata version 12
  - SPSS (with limitations)
- Load the foreign package

```
library(foreign)
## as always, review
help(package = "foreign")
```

The only ones I've used are for SPSS, Stata, SAS

- `read.dta()` : For Stata through Stata Version 12. Does not require the user's system to have a copy of Stata
- `read.spss()` : Works fairly well with SPSS through version 20, but there are problems with foreign character sets that have existed for a long time.

KU

R Foreign package ...

- `read.xport()` : Requires the user's system to have a copy of SAS. It has worked for me since 2013, but the requirement to have SAS is a big hurdle.
- Recently, we had fantastic luck with the SAS file importer in package `SAScii` . Is able to read SAS code files and associated CSV data files.

Stata (through V12) Historically Most Workable

- Syntax is simple

```
library(foreign)
toe1f <- read.dta("data/toenail.dta")
```

- Stop there! practice your R data.frame exploration skills on toe1f!
- R will construe value labels as factor levels, unless you ask it not to

```
fn <- file.path(ddir, "toenail.dta")
toe1n <- read.dta(fn, convert.factors = FALSE)
```

  - Confusion between "values" and "labels" begins here.

Compare

```
table(toe1f$treatment, toe1n$treatment,
    exclude = NULL, dnn = list("factor",
    "numeric"))
```

Stata (through V12) Historically Most Workable ...

```
                numeric
factor             0   1
  Itraconazole   937   0
  Terbinafine      0 971
```

- Generally, I DO want to import the labels.
- However, as above, I import 2 versions
- R can write Stata files for version 12

```
write.dta(dat, file = "whatever.dta")
```

- And SPSS can import those Stata files we write from R's write.dta() (same NOT true of dta-writers in user-contributed packages)
- Within Stata, use "`saveold`" for version 12 to maintain compatability.

Usually works to import SPSS data

- Syntax is slightly more complicated, because SPSS internal structure is slightly more complicated.
- Creates an R list, not a data frame. Which we can inspect and convert

```
dat <- read.spss("whatever.sav")
dat <- as.data.frame(dat)
```

- Ask R to try to create a data frame for you, skip the list entirely

```
dat <- read.spss("whatever.sav",
    to.data.frame = TRUE)
```

- Whenever you try to import an SPSS data set, you will generally see either errors or warnings. It is difficult to know for sure who to blame about this, but it is a fact of life.

Usually works to import SPSS data ...

- In 2014, we started to see errors because SPSS created text columns that were 256 characters wide, even if users just entered values like "Tim" and "Jane". R seems to panic and it imports the character string, but then it creates a block of empty columns.
- Hmisc package function `spss.get()` is a "wrapper" for `foreign:read.spss()` which organizes the SPSS labels somewhat differently
- When in trouble, I try the SPSS importers offered by other packages
  1. memisc
  2. Haven
- When those fail, I become desperate and get an SPSS user to open the file and look to see what's so funny about the data, and possibly write a new one or export to Stata format

Outline

The variable key is a way we created to manage projects

- When the GRAs import data, they are generally inclined to start making a lot of changes.
- From a management point of view, we'd like the managers and principal investigators to guide the GRA decisions.
- Worst case scenario: GRAs cobble together 1000s of lines of changes that we don't understand.

KU

Example

```
library(car)

## Read in data
dat<-read.csv(file="fulldata.csv",header=T,
    na.string=c("-980","-981","-982","-983","-984","-985",
"-986","-987","-988","-989","-990","-991","-992","-993",
    "-994","-995","-996","-997","-998","-999"))
##family predictors
dat$Rnp1G1e<-recode(dat$np1G1e, "0=2;1=1;2=0")
dat$Rnp1G1h<-recode(dat$np1G1h, "0=2;1=1;2=0")
dat$Rnp1G5a<-recode(dat$np1G5a, "4=1;3=2;2=3;1=4")
dat$Rnp1F1d<-recode(dat$np1F1d, "1=6;2=5;3=4;4=3;5=2;6=1")
##[SNIP 100s more lines like this]
```

- I'd debate the 1) choice of new variable names, 2) the method of re-assigning values

Example

- Note in following, if mnames and w6vars are not perfectly aligned, we have FATAL error

```
w6 <- read.table("w6.dat", header = TRUE)
## Create variable names for merged dataset
mnames <- c("distid", "wave", "region", "province",
    "district","sec1", "sec2", "sec3", "sec4", "sec5", "trn1",
    "trn2", "trn3", "trn4", "crp1", "crp2", "crp3",
    "crp4","crp5", "pgv1", "pgv2", "pgv3", "pgv4", "pgv5",
    "dgv1", "dgv2", "dgv3", "dgv4", "dgv5", "dgv6", "rec1",
    "rec2")
## Wave 6 data management
w6$wave <- 6
w6$fill <- NA
w6vars <- c("dist", "wave", "m4b", "m7", "m5", "q1", "q2",
    "q3", "q9", "q30", "q12", "q14", "q20", "q26", "q13",
    "q21", "q35e", "q36e","q63", "q35a","q41a", "q41b",
    "q41c", "q41d", "q36a", "q43a", "q43b", "q43c", "q43d",
    "q47c", "q10", "q73")
w6 <- w6[ , w6vars]
colnames(w6) <- mnames
```

Example ...

```
## ...SNIP.. a few 100 lines
## "sec1"
mdata$sec1[mdata$sec1 == 100] <- 1
mdata$sec1[mdata$sec1 == 101] <- 2
mdata$sec1[mdata$sec1 == 102] <- 3
mdata$sec1[mdata$sec1 > 3] <- NA

## "sec2"
mdata$sec2[mdata$sec2 == 100] <- 1
mdata$sec2[mdata$sec2 == 101] <- 2
mdata$sec2[mdata$sec2 == 102] <- 3
mdata$sec2[mdata$sec2 > 3] <- NA

## "trn1" Recoded
mdata$trn1[mdata$trn1 == 100] <- 6
mdata$trn1[mdata$trn1 == 101] <- 5
mdata$trn1[mdata$trn1 == 102] <- 4
mdata$trn1[mdata$trn1 == 103] <- 3
mdata$trn1[mdata$trn1 == 104] <- 2
```

The variable key is a way we created to manage projects

- The variable key is a display of the variables in a spreadsheet format which we use to figure out "what we have" and "what do we wish it would be".
- A vignette with the kutils package explains many of the details
- The wide key format. This creates a table we can revise

```
library(kutils)
keyw <- keyTemplate(grades, file =
    "key-wide.csv")
```

- The column names in the key will be
  1. name_old
  2. name_new
  3. class_old
  4. class_new
  5. value_old
  6. value_new
  7. missings
  8. recodes

The Wide Variable key

- Not unexpectedly, the wide key is too long to display on a page

name_old	name_new	class_old	class_new	value_old	value_new
namelast	namelast	character	character	Brown|Granger|Longbottom|Malfoy|Parkinson|Potter|Weasley	Brown|Granger|Longbotto
namefirst	namefirst	character	character	Draco|Harry|Hermoine|Lavender|Neville|Pansy|Ron	Draco|Harry|Hermoine|La
id	id	integer	integer	234252|320720|342234|453345|666666|789897|942832	234252|320720|342234|45
hw1	hw1	integer	integer	5|6|7|8|9	5|6|7|8|9
hw2	hw2	integer	integer	4|7|8|9	4|7|8|9
hw3	hw3	character	character	F|P	F|P
mterm	mterm	integer	integer	66|78|81|84|87|101	66|78|81|84|87|101
fin	fin	integer	integer	66|77|79|85|88|999	66|77|79|85|88|999
grade	grade	character	character	A+|B|B-|C-|C+	A+|B|B-|C-|C+

- The Wide key has one variable per line
- If `length(value_old)` becomes greater than a threshold, then no values are printed
- Challenge: `value_old` and `value_new` must match, one for one
- Want to delete a variable?
  - Omit the row, or
  - make empty the `name_new` value
- Want a value to become missing? We suggest putting "." in value_new
- How to use recodes and missings? You better read the manual

The Long Variable key

- The Long key has one value per line (hence, many more rows)

```
library(kutils)
keyl <- keyTemplate(grades, long=TRUE, file =
    "key-long.csv")
```

name_old	name_new	class_old	class_new	value_old	value_new	missings	rec
namelast	namelast	character	character	Brown	Brown		
namelast	namelast	character	character	Granger	Granger		
namelast	namelast	character	character	Longbottom	Longbottom		
namelast	namelast	character	character	Malfoy	Malfoy		
namelast	namelast	character	character	Parkinson	Parkinson		
namelast	namelast	character	character	Potter	Potter		
namelast	namelast	character	character	Weasley	Weasley		
namefirst	namefirst	character	character	Draco	Draco		
namefirst	namefirst	character	character	Harry	Harry		
namefirst	namefirst	character	character	Hermoine	Hermoine		
namefirst	namefirst	character	character	Lavender	Lavender		
namefirst	namefirst	character	character	Neville	Neville		

The Long Variable key ...

namefirst	namefirst	character	character	Pansy	Pansy
namefirst	namefirst	character	character	Ron	Ron
id	id	integer	integer	234252	234252
id	id	integer	integer	320720	320720
id	id	integer	integer	342234	342234
id	id	integer	integer	453345	453345
id	id	integer	integer	666666	666666
id	id	integer	integer	789897	789897
id	id	integer	integer	942832	942832
hw1	hw1	integer	integer	5	5
hw1	hw1	integer	integer	6	6
hw1	hw1	integer	integer	7	7
hw1	hw1	integer	integer	8	8
hw1	hw1	integer	integer	9	9
hw2	hw2	integer	integer	4	4
hw2	hw2	integer	integer	7	7
hw2	hw2	integer	integer	8	8
hw2	hw2	integer	integer	9	9
hw3	hw3	character	character	F	F
hw3	hw3	character	character	P	P
mterm	mterm	integer	integer	66	66
mterm	mterm	integer	integer	78	78

The Long Variable key ...

mterm	mterm	integer	integer	81	81
mterm	mterm	integer	integer	84	84
mterm	mterm	integer	integer	87	87
mterm	mterm	integer	integer	101	101
fin	fin	integer	integer	66	66
fin	fin	integer	integer	77	77
fin	fin	integer	integer	79	79
fin	fin	integer	integer	85	85
fin	fin	integer	integer	88	88
fin	fin	integer	integer	999	999
grade	grade	character	character	A+	A+
grade	grade	character	character	B	B
grade	grade	character	character	B-	B-
grade	grade	character	character	C-	C-
grade	grade	character	character	C+	C+

The Big Key Idea

- PI, Manager, and GRA revise the key file (wide or long)
- The `keyImport()` function: revised key file is read in
- The `keyApply()` function: The data frame and the key are applied to create a new, recoded data frame.
- Some problems arise because various spreadsheet programs use different methods to
  - store character variables
  - save CSV files.

Outline

Keep a clean work environment

- R allows you to have many data frames open at once.
- It also allows you to have "free floating", disconnected vectors.
- Look at the headache I've got after this

```
V1 <- seq(1, 10, by = 0.01)
V2 <- c(3)
V3 <- c(1, 2, 3, 4, 5)
head(fd)
```

```
                v3         v4 v5
Bill     0.6607697  0.1401390  A
Joe     -2.0529830  0.2091844  B
Frosty  -1.4992061 -3.0360898  C
Mickey   1.4712331 -0.4869341  D
Henry    1.4591385 -1.0878673  E
```

- ls() lists objects

```
ls()
```

Keep a clean work environment ...

```
 [1] "ddir"          "f1"            "f2"
 [4] "f3"            "fd"            "fd13"
 [7] "fdnew"         "fn"            "grades"
[10] "keyl"          "keyw"          "options.orig"
[13] "opts.orig"     "ore1"          "par.orig"
[16] "pjmar"         "swiss"         "tdir"
[19] "toe1f"         "toe1n"         "V1"
[22] "V2"            "V3"            "w1"
[25] "w2"            "X"             "xt1"
[28] "xt2"           "z"
```

This displays the R "Global Environment": objects that can be accessed within smaller environments created by packages and functions.

- Use rm to destroy an unwanted thing.

```
rm(V1)
```

- And if you want to remove several things at once:

```
rm(list = c("V1", "V2"))
```

Keep a clean work environment ...

Syntax here confuses me a bit, don't know why we say **list =** and then supply a vector of character strings.

- The nuclear option: remove everything in the workspace

```
rm(list = ls())
```

Takes the output from `ls()`, and gives it as the list of things to be removed.

Outline

For practice,

- We will have some data files...
- We can also try to import any text data files you happen to drag up.

References

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 19.04

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
```

Session ...

```
[1] xtable_1.8-4       foreign_0.8-71     openxlsx_4.1.0
    rockchalk_1.8.144

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.1        lattice_0.20-38 MASS_7.3-51.4   grid_3.6.0
     plyr_1.8.4        nlme_3.1-140
 [7] stats4_3.6.0      zip_2.0.2       carData_3.0-2   minqa_1.2.4
     nloptr_1.2.1      Matrix_1.2-17
[13] pbivnorm_0.6.0    boot_1.3-22     splines_3.6.0   lme4_1.1-21
    tools_3.6.0       kutils_1.69
[19] compiler_3.6.0    mnormt_1.5-5    lavaan_0.6-3
```