How to manage data

Matthew Suderman

Lecturer in Epigenetic Epidemiology

MRC Integrative Epidemiology Unit

University of BRISTOL

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

  e.g. convert a continuous variable to binary

  e.g. combine categories in categorical variable

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

  e.g. convert a continuous variable to binary

  e.g. combine categories in categorical variable
- Analyse **multiple** datasets on the same individuals

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

  e.g. convert a continuous variable to binary

  e.g. combine categories in categorical variable
- Analyse **multiple** datasets on the same individuals

  e.g. Test associations between 30K genes and a phenotype while adjusting for potential confounders.

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

  e.g. convert a continuous variable to binary

  e.g. combine categories in categorical variable
- Analyse **multiple** datasets on the same individuals

  e.g. Test associations between 30K genes and a phenotype while adjusting for potential confounders.
- **Summarize** a variable by group

Typical tasks in (molecular) epidemiology

- **Load** a massive file into memory and run some simple tests
- Normalize/clean a dataset and **save** the result to a file
- **Transform** a variable

  e.g. convert a continuous variable to binary

  e.g. combine categories in categorical variable
- Analyse **multiple** datasets on the same individuals

  e.g. Test associations between 30K genes and a phenotype while adjusting for potential confounders.
- **Summarize** a variable by group

  e.g. Given data for 100,000 individuals, calculate mean income by county.

Load a massive file into memory

Loading large spreadsheets using `read.csv`/`read.table` can take a long time.

Load a massive file into memory

Loading large spreadsheets using `read.csv/read.table` can take a long time.

Here is one way to speed it up.

Step 1. Peek at the first few rows to determine the data class of each column:

```
dat <- read.table("big-massive-file.txt",
                  nrows=3, header=T, sep=" ")
dat
##   CHR    POS            SNP
## 1   1 721290 rs12565286:G:C
## 2   1 723891  rs2977670:G:C
## 3   1 752566  rs3094315:G:A
```

Load a massive file into memory

Loading large spreadsheets using `read.csv`/`read.table` can take a long time.

Here is one way to speed it up.

Step 1. Peek at the first few rows to determine the data class of each column:

```
dat <- read.table("big-massive-file.txt",
                  nrows=3, header=T, sep=" ")
dat
##   CHR    POS              SNP
## 1   1 721290 rs12565286:G:C
## 2   1 723891  rs2977670:G:C
## 3   1 752566  rs3094315:G:A
```

Step 2. Use `fread()` from the `data.table` library to load the file:

```
library(data.table)
classes <- c(chr="character", pos="integer", snp="character")
dat <- fread("big-massive-file.txt",
             header=T, sep=" ",
             colClasses=classes)
```

Load a massive file, cont

Some spreadsheets are just too large to load, e.g. files larger than 1/2 Gb.

Load a massive file, cont

Some spreadsheets are just too large to load, e.g. files larger than 1/2 Gb.

One solution is to load parts of the table *as you need them*.

```r
library(ff)
dat <- read.table.ffdf("big-massive-file.txt", header=T, sep=" ")
```

You can now access `dat` just like a data frame.

```r
dat[1:3,]   ## retrieve the first three rows
dat$CHR     ## retreive the column named "CHR"
```

However, you might notice that retrievals are *much* slower than a normal data frame.

Load a massive file, cont

R can handle many other file formats.

The following functions all read the file and return a data frame:

file format	**R package**	**function(s)**
STATA (.dat)	foreign	`read.dta()`
	readstata13	`read.dta13()`
	haven	`read_dta()`
SPSS (.sav)	foreign	`read.spss()`
	haven	`read_spss()`
Excel	readxl	`read_xls()` and `read_xlsx()`

Load a massive file, cont

R can handle many other file formats.

The following functions all read the file and return a data frame:

file format	**R package**	**function(s)**
STATA (.dat)	foreign	`read.dta()`
	readstata13	`read.dta13()`
	haven	`read_dta()`
SPSS (.sav)	foreign	`read.spss()`
	haven	`read_spss()`
Excel	readxl	`read_xls()` and `read_xlsx()`

Note

- This is not a complete list. See the `foreign` and `haven` packages for other formats.
- `read.dta13()` is specifically for STATA v13 files.
- `read_dta()` loads variable comments, `read.dta13()` does not

Save data to a file: csv

The most common way in R to save a data frame is to use `write.csv()` or `write.table()`.

```
write.csv(dat, row.names=F, quote=F)
write.table(dat, row.names=F, quote=F, sep=",")
```

Save data to a file: csv

The most common way in R to save a data frame is to use `write.csv()` or `write.table()`.

```r
write.csv(dat, row.names=F, quote=F)
write.table(dat, row.names=F, quote=F, sep=",")
```

The newer function `fwrite()` is 30-60 times faster but otherwise identical.

```r
library(data.table)
fwrite(dat, row.names=F, quote=F, sep=",")
```

Save data to a file: RData and RDS

If a dataset will only be used by R, then it is most efficient to use the RData/RDS format.

Save data to a file: RData and RDS

If a dataset will only be used by R, then it is most efficient to use the RData/RDS format.

RData

The `save()` function saves variables to to an `RData` file.

```
save(dat, dat.norm, ret, file="datase
```

The `load()` function loads these variables into R.

```
load("datasets.rda", verbose=T)
## Loading objects:
##   dat
##   data.norm
##   ret
```

Save data to a file: RData and RDS

If a dataset will only be used by R, then it is most efficient to use the RData/RDS format.

RData

The `save()` function saves variables to to an `RData` file.

```
save(dat, dat.norm, ret, file="datase
```

The `load()` function loads these variables into R.

```
load("datasets.rda", verbose=T)
## Loading objects:
##   dat
##   data.norm
##   ret
```

RDS

`RDS` files each save only one object.

```
saveRDS(dat, file="dataset.rds")
```

That single object can be loaded into R with any given variable name.

```
mydat <- readRDS("dataset.rds")
```

Transform a variable

Suppose we have a numerical variable x.

We can convert it to 3 categories corresponding to three ranges of values.

```
x.3 <- rep('null', length(x))
x.3[x < -0.5] <- 'neg'
x.3[x > 0.5] <- 'pos'
```

Transform a variable

Suppose we have a numerical variable x.

We can convert it to 3 categories corresponding to three ranges of values.

```
x.3 <- rep('null', length(x))
x.3[x < -0.5] <- 'neg'
x.3[x > 0.5] <- 'pos'
```

Factors are a more efficient way to store categorical variables in the computer as well as a way to give an ordering to the categories.

```
x.3 <- factor(x.3, levels=c("neg", "null", "pos"), ordered=T)
```

Transform a variable

Suppose we have a numerical variable x.

We can convert it to 3 categories corresponding to three ranges of values.

```
x.3 <- rep('null', length(x))
x.3[x < -0.5] <- 'neg'
x.3[x > 0.5] <- 'pos'
```

Factors are a more efficient way to store categorical variables in the computer as well as a way to give an ordering to the categories.

```
x.3 <- factor(x.3, levels=c("neg", "null", "pos"), ordered=T)
```

We could combine the 'null' and 'pos' categories and replace call the resulting category 'nonneg'.

```
x.2 <- as.character(x.3)
x.2[x.3=="null" | x.3=="pos"] <- "nonneg"
x.2 <- factor(x.2, levels=c("neg", "nonneg"), ordered=T)
```

Align datasets

For a population sample, we have gene expression levels and phenotype information.

- `gene.dat` is a matrix rows=genes and columns=samples
- `pheno.dat` is a data frame with rows=samples and columns=variables

```
> gene.dat[1:3,1:3]
      s234 s199 s397
NR3C1   83  130  171
IGF2    87   99  126
MYC    200  160   32
```

```
> pheno.dat[1:3,]
    id sex age smoker bmi
1 s199   M  23  FALSE  24
2 s349   F  50  FALSE  23
3 s456   F  19  FALSE  29
```

Align datasets

For a population sample, we have gene expression levels and phenotype information.

- `gene.dat` is a matrix rows=genes and columns=samples
- `pheno.dat` is a data frame with rows=samples and columns=variables

```
> gene.dat[1:3,1:3]
      s234 s199 s397
NR3C1   83  130  171
IGF2    87   99  126
MYC    200  160   32
```

```
> pheno.dat[1:3,]
    id sex age smoker bmi
1 s199   M  23  FALSE  24
2 s349   F  50  FALSE  23
3 s456   F  19  FALSE  29
```

To test associations between gene expression and phenotypes, we need to make sure that the samples are in the same order.

```
idx <- match(pheno.dat$id, colnames(gene.dat))
gene.dat <- gene.dat[,idx]
```

Now we can test associations.

```
fit <- lm(gene.dat["MYC",] ~ ., data=pheno.dat)
```

Summarize a variable by group

Suppose we have data from an experiment in which guinea pigs were each given one of two supplements at one of 3 different doses. We want to know which supplement/dose maximizes tooth growth.

```
> head(ToothGrowth)
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
```

We calculate the mean growth by supplement/dose using `aggregate()`.

```
> with(ToothGrowth, aggregate(len, by=list(supp, dose), mean))
  Group.1 Group.2     x
1      OJ     0.5 13.23
2      VC     0.5  7.98
3      OJ     1.0 22.70
4      VC     1.0 16.77
5      OJ     2.0 26.06
6      VC     2.0 26.14
```

Tidyverse (https://www.tidyverse.org/)

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures."

Some of the more popular packages are:

- **dplyr** (https://dplyr.tidyverse.org/) is for manipulating data.
- **ggplot2** (https://ggplot2.tidyverse.org/) is for creating plots.

ALSPAC R package

The `alspac` R package provides
a convenient interface in R to the ALSPAC dataset.

- **Installation**

```
library(devtools)
install_github("explodecomputer/alspac")
```

- **Example use**

```
library(alspac)
setDataDir("/path/to/R drive/data/")
vars <- findVars(c("kz021","c645a","b032",
                   "b670","c804"))
results <- extractVars(vars)
```

The requested data is in data frame `results`.