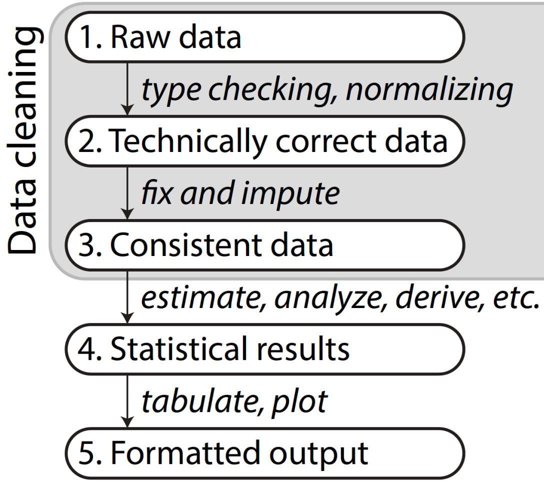
Data cleaning

For applied statisticians, data cleaning can be the most time-consuming part of the job. This involves extracting variables from data bases, reorganizing data, merging files that are sometimes in different formats, analyzing subsets of data, understanding missing data in your data set, and so forth.

Often, raw data cannot be analyzed as is, but must be processed considerably first. In many cases, also, a data set is not labeled to tell you which variables are the response and which variables are the predictors. Often this must be determined by talking withoever collected the data.

Data cleaning



Data cleaning

1. Raw data might have some variables labeled or some not, might have some entries coded in a way that turns a numerical column into strings, have inconsistent date formats (such as 12/1/17, 12/1/2017 and 1-Dec-17 all in the same column), have missing data which might be coded inconsistently, or have comments in the data. Often this data can't even be read in directly as a `data.frame` in R because it is not rectangular (same number of rows for each column).
2. Technically correct data can be read into R as a `data.frame` object but might still have data values that are not possible or are incorrect, such as negative numbers for heights or counts, or dates that occur in the future.
3. Consistent data means data that is cleaned and ready for analysis and statistical procedures. Outliers might still be present in the data

Data cleaning

4. Statistical results. Results are created in the computer and can be reused if necessary, but hasn't necessarily been formatted for a report, paper, or presentation.
5. Formatted output. Results are presented in a form suitable for a report or presentation, such as in tables or plots.

Note: it is best to store data in different stages, including raw, technically correct, and cleaned data so that if there is a question about how the data has been cleaned, the raw data can be used to clean the data again using a slightly different method. Also, variables in the raw data might not have been retained in the cleaned data, so it is always good to have original raw data available instead of modifying this file. In some cases, such as in the pharmaceutical industry, data sets are "frozen" so that the data analyst does not have permissions in the computer to modify an original data file. If data is in an Excel `.csv` file, a staistician might modify the `data.frame` object in R instead of modigying the `.csv` file.

Data cleaning

There are several data types in R. These include

- numeric (includes decimals to approximate real numbers up to some level of precision)
- integer (whole numbers for counting)
- factor (categorical–often strings are treated as factor)
- character (used for strings that cannot be numeric. Although apparent numbers might be treated as character strings or factors, especially when there are errors in the data)
- ordered (ordered categories, such as small, medium, large)
- raw (binary data, which is rarely used)
- matrix (similar to numeric vectors, but arranged in a rectangle)

In addition, certain functions or packages might define their own data types, such as linear model objects, evolutionary trees, etc.

Data cleaning

Note that vectors and matrices have objects of one type.

```
> x <- c(1,2,"3")
> class(x)
[1] "character"
> x
[1] "1" "2" "3"
> mean(x)
[1] NA
Warning message:
In mean.default(x) : argument is not numeric or logical:
returning NA
```

Here all objects were converted to character. If you have a column in a data set that is supposed to be numeric, but someone accidentally typed a character (like O instead of 0) somewhere, it can convert the entire column to character, which means that you can't take an average or do other mathematical operations.

Data cleaning

`list` objects can be very hetrogeneous and can store an array of objects of different types

```
> a <- matrix(1:4,ncol=2)
> b <- list(1,a,"a","1")
> b
[[1]]
[1] 1
[[2]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[[3]]
[1] "a"
[[4]]
[1] "1"
```

Data cleaning

Something to be careful of is how R multiplies two vectors together. Consider the following

```
> x <- 1:4
> y <- c(-1,1)
> x*y
[1] -1  2 -3  4
> z <- 1:3
> z*y
[1] -1  2 -3
Warning message:
In z * y : longer object length is not a multiple of shorter
object length
```

When two vectors get multiplied, R uses **recycling** of the shorter object to keep going to reuse values if the lengths of the vectors don't match up. This doesn't generate a warning as long as the length of the longer object is a multiple of the length of the shorter object.

Data cleaning

There are a few values in R that are treated in a special way, such as `NA` which means “not available”, and is used for missing values.

```
> a <- c(1,2,NA,4)
> b <- c(NA,5,6,7)
> a+b
[1] NA  7 NA 11
> mean(a,rm.na=T)
[1] NA
> mean(a,na.rm=T)
[1] 2.333333
```

Data cleaning

```
> a <- c(1,2,NA,4,5,6)
> b <- c(NA,7,8,9,1,2)
> cor.test(a,b,na.rm=T)

Pearson's product-moment correlation

data:  a and b
t = -1.3665, df = 2, p-value = 0.3051
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9928808  0.8014513
sample estimates:
       cor
-0.6948677
```

Note the degrees of freedom. How many observations were used?

Data cleaning

To get a better understanding of how NA is used, try the following:

```
NA + 1
sum(c(NA, 1, 2))
median(c(NA, 1, 2, 3), na.rm = TRUE)
length(c(NA, 2, 3, 4))
3 == NA
NA == NA
TRUE | NA
TRUE & NA
# use is.na() to detect NAs
is.na(c(1, NA, 3))
```

Data cleaning

There is also a distinction between missing numeric values and missing string values. Consider the following data set, called “test1.txt” (I don’t have a link; you just need to make this file yourself if you want to try it):

```
3 tree
4 house
NA dog
5 NA
```

Data cleaning

```
> x <- read.table("test1.txt")
> x
  V1    V2
1  3  tree
2  4 house
3 NA   dog
4  5  <NA>
> table(x$V1)

3 4 5
1 1 1
> table(x$V2)

  dog house  tree
    1     1     1
```

Data cleaning

Another special value in R is NULL which can be used to initialize a variable without putting anything in it.

```
length(c(1, 2, NULL, 4))
sum(c(1, 2, NULL, 4))
x <- NULL
length(x)
c(x, 2)
# use is.null() to detect NULL variables
is.null(x)
```

Data cleaning

I sometimes use NULL to create a vector when I don't know how many elements it will have. For example, I roll two dice, and I want to save outcomes where the sum is greater than or equal to 4. I could use this code

```
success <- NULL
for(i in 1:10000) {
x <- sample(1:6,2,replace=T)
if(sum(x) >= 4) success <- c(success,x)
}
```

This code creates a vector called `success` which records the value every time the sum was greater than or equal to 4, and does so by concatenating to the end of previous results. I don't know in advance how many successes I will have.

Data cleaning

Inf Stands for ∞ and only applies to vectors of class numeric (not integer). Technically, Inf is a valid numeric that results from calculations like division of a number by zero, or numbers that are just too big for R to represent.

```
> 1/0
[1] Inf
> 1/-0
[1] -Inf
> 10^1000
[1] Inf
> Inf+1
[1] Inf
> Inf+1 > Inf
[1] FALSE
> 100 < Inf
[1] TRUE
```

Data cleaning

As a final special value, NaN stands for not a number, which is different from missing values handled with NA. Here are some weird examples:

```
> Inf+Inf
[1] Inf
> Inf-Inf
[1] NaN
> Inf-Inf
[1] NaN
> Inf-NaN
[1] NaN
> NA-NaN
[1] NA
> NaN-NA
[1] NaN
```

Data cleaning

Special attention is needed when dealing with dates and times as data.

The base R installation has three types of objects to store a time instance: `Date`, `POSIXlt`, and `POSIXct`. This is a little bit technical. There is also the `lubridate` package which has extra functions for dealing with dates.

Data cleaning

The function, as.Date() converts character strings to date objects, which is a built in class, like numeric and character. Objects of class date can be manipulated like numbers and other data.

As an example, let's say that you want to generate a data set with all dates in 2018, with one row per date.

```
date <- as.Date(``01/01/2018'',format=``%m/%d/%Y'')
tempdate <- date
for(i in 1:365) {
  tempdate2 <- format(as.Date(tempdate), ``%m/%d/%Y'')
  string1 <- as.character(tempdate2)
  string <- paste(i,string1)
  write(string,file=''calendar.txt'',append=T)
  tempdate <- format(as.Date(tempdate),``%m/%d/%Y'')
  tempdate <- tempdate+1
}
```

Data cleaning

Note that formatting a `date` object creates a character string. Consequently, in the above code, `tempdate` is a `date` object, while `tempdate2` is a character string.

```
> tempdate <- as.Date("01/01/2018",format="%m/%d/%Y")
> tempdate2 <- format(tempdate,format="%m/%d/%Y")
> tempdate
[1] "2018-01-01"
> tempdate2
[1] "01/01/2018"
> class(tempdate)
[1] "Date"
> class(tempdate2)
[1] "character"
```

Data cleaning

The code above was something I wanted for a project, where I wanted one row per day and wanted to generate all the days in a year. I was able to add 1 to each temporary day to generate the following day. R is smart enough to know that 2018 is not a leap year for example. If you ran the same code starting on 1/1/2020, and ran the loop 366 days, then it would know that 2020 was a leap year, and generate the dates correctly.

In the code, the date was read in using month-day-year format (although with Jan 1, I could have interpreted it as day-month-year instead), and adding 1 to each date added to the day rather than the year or month.

Data cleaning

A similar program could be used to convert each day to a different format. For example, consider the following code for a fake file called hospital_data.csv.

```
x <- read.csv(‘‘hospital_data.csv’’)
x$date <- as.Date(x$date,format=‘‘%m/%d/%Y’’)
x$date <- as.Date(x$date,format=‘‘%Y/%m/%d’’)
write.csv(x,file=’’hospital_data_newdate.csv’’)
```

Data cleaning

This type of operation can be important when you have data from different files that needs to be combined and dates are coded inconsistently.

Data cleaning

If you want to extract a month or a day of a week from a date, if all the dates are in the same format, then the there will be exactly the same number of characters in each date. So you could get the numerical month or day. Another possibility is to use the functions `weekdays()` or `months`.

```
> tempdate <- as.Date("01/29/2018",format="%m/%d/%Y")
> tempdate
[1] "2018-01-29"
> month <- substr(tempdate,6,7)
> month
[1] "01"
> months(tempdate)
[1] "January"
> weekdays(tempdate)
[1] "Monday"
```

Data cleaning

Another useful operation on dates is subtracting them. For example, if you record times that a patient visits a hospital, you might want to create a new variable for the time between visits. Or if you have the birthdate and the time of a visit, you might want to calculate the age at the time of the visit, which might not be explicit in the data.

If you have two objects of type date. Here is an example data set in `hospital_data.csv`:

```
subject,birth,visit1
1,10/05/2001,2017-05-31
2,11/06/2002,2016-06-29
3,08/06/1995,2016-06-16
```

Data cleaning

Suppose we want to compute the age of each patient at the time of `visit1`.

```
> x <- read.csv(‘‘hospital_data.csv’’)
> birth <- as.Date(x$birth,format="%m/%d/%Y")
> birth
[1] "2001-10-05" "2002-11-06" "1995-08-06"
> visit1 <- as.Date(x$visit1,format="%Y-%m-%d")
> visit1
[1] "2017-05-31" "2016-06-29" "2016-06-16"
> visit1-birth
Time differences in days
[1] 5717 4984 7620
> (visit1-birth)/365
Time differences in days
[1] 15.66301 13.65479 20.87671
```

Data cleaning

Speaking of `.csv` files...One reason they are useful is that they can accommodate data where there might be spaces in the data itself. For example, data with street addresses might look something like this:

```
1101 Mesa Verde Ave NE, Albuquerque, NM, 87110
123459 Fork in the Road Drive, Rio Rancho, NM, 87124
etc
```

Names also have a variable number of words (0, 1, or 2 middle names, for example). Occasionally you might want commas to be in the name, for example in a database of song or movie titles. In this case you can use a different delimiter instead of a comma, such as a semicolon, which is less likely to be actual character data.

Data cleaning

Sometimes, in addition to dates to work with, you have times in minutes and seconds (or fractions of seconds). This often occurs, for example, in automated tracking data from GPS or other locations. Consider the following (the data is available in that file name at math.unm.edu/$\sim$james but with no link) on earthquakes in New Zealand.

Data cleaning

```
> x <- read.csv("quakes1Jan11_31Mar11.csv")
> head(x)
  publicid  eventtype               origintime        modificati
1  3489855 earthquake 2011-03-31T22:38:41.882Z 2013-08-21T16:47:0
2  3489852 earthquake 2011-03-31T22:28:04.740Z 2013-08-21T16:47:0
3  3489806 earthquake 2011-03-31T20:09:27.623Z 2013-08-21T16:47:0
4  3489774 earthquake 2011-03-31T18:41:13.638Z 2013-08-21T16:47:0
5  3489769 earthquake 2011-03-31T18:27:31.492Z 2013-08-21T16:47:0
6  3489764 earthquake 2011-03-31T18:10:52.191Z 2013-08-21T16:47:0
  longitude  latitude magnitude   depth magnitudetype        dept
1  166.7705 -45.47349     3.016 12.0000            ML operator as
2  172.7105 -43.58500     2.937  6.9356            ML
3  172.7047 -43.48363     2.338  5.0000            ML operator as
4  172.4061 -43.40538     2.744  5.0000            ML operator as
5  172.0999 -43.50152     3.025  8.2128            ML
6  172.7004 -43.57466     3.053  5.6451            ML
```

Data cleaning

The data set includes lots of variables, only some of which might be of interest for a statistical analysis. Each earthquake (these include many aftershocks) gets a time stamp based on the time of origin as well as the time at which the data was reviewed and possibly modified. Often the magnitude of a quake is initially estimated and then later adjusted. Here is a list of the variables in the data set:

```
> names(x)
 [1] "publicid"             "eventtype"             "origintime"
 [4] "modificationtime"     "longitude"             "latitude"
 [7] "magnitude"            "depth"                 "magnitudety
[10] "depthtype"            "evaluationmethod"      "evaluations
[13] "evaluationmode"       "earthmodel"            "usedphaseco
[16] "usedstationcount"     "magnitudestationcount" "minimumdist
[19] "azimuthalgap"         "originerror"           "magnitudeun
```

Data cleaning

Suppose you are interested in only the `origintime` variable. The variable includes dates and times in a single string. The dates are in the format YYYY-mm-dd followed by the letter T and the time in 24-hr (military) time with hours:minutes:seconds and seconds are measured to the nearest thousandth. The time is then followed by a Z with no space. What to do with data like this?

If you were only interested in the dates, then you could replace the T with a space and eliminate the Zs. Then you would need to create two variable names instead of just the one `origintime`, such as `origindate` and `originSeconds`. This might be tricky to do in the original file because you might not be able to replace every instance of T or Z without corrupting other data in the file. Consequently, one possibility would be to copy the `origintime` column to a new spreadsheet, do the replace functions there, then copy back to the original file. Another possibility is to not modify any spreadsheet and do the manipulations in R.

Data cleaning

In R, you have x$origintime as a variable. We could create a two new variables with just the date and just the hour-minutes-seconds. The following code would work:

```
mydate <- x$origintime
mydate <- substr(mydate,1,10)
mytime <- x$origintime
mytime <- substr(mytime,12,23)
```

This code works using the substr() function which extracts only certain characters from a string, such as characters 1 through 10, or 12 through 23.

Data cleaning

Another solution to the problem is to replace T with nothing, or replace it with a space, and similarly with the Z.

```
> mydate <- x$origintime
> mydate <- gsub("T"," ",mydate)
> mydate <- gsub("Z","",mydate)
> head(mydate)
[1] "2011-03-31 22:38:41.882" "2011-03-31 22:28:04.740"
[3] "2011-03-31 20:09:27.623" "2011-03-31 18:41:13.638"
[5] "2011-03-31 18:27:31.492" "2011-03-31 18:10:52.191"
```

In this case, I've have kept the date and time as a single variable string but with a space separating the date and time. The `gsub()` function replaces the first string with the second string in the variable listed in the third argument.

Data cleaning

The `lubridate` package is useful for a wide range of date-time functions, especially for reading in seconds. It uses a different data type from the base R time/date functions:

```
> mydate2 <- ymd_hms(mydate,tz="Pacific/Auckland")
> mydate2[1]
[1] "2011-03-31 22:38:41 NZDT"
> class(mydate2)
[1] "POSIXct" "POSIXt"
> mydate2[2]-mydate2[1]
Time difference of -10.61903 mins
```

Data cleaning

A final topic to mention that comes up in data cleaning is merging files. Let's say you have `file1` which has patient ID and insurance status (Presbyterian, BCBS, or none), and `file2` has patient ID and number of visits to a clinic. Your job is to determine how type of insurance is related to the number of visits to the number of visits. To make things more complicated, not every patient is listed in both files.

Here is what the data might look like

```
####merge1.txt
ID insurance age
11047 Pres 47
11098 BCBS 48
12011 BCBS 41
12221 none 37
12241 BCBS 51

####merge2.txt
ID nvisits
11047 3
11066 1
12011 1
12221 2
12241 5
13131 1
```

```
> d1 <- read.table("merge1.txt",header=T)
> d2 <- read.table("merge2.txt",header=T)
> merged.data <- merge(d1, d2, by="ID")
> merged.data # only includes cases with no missing data
     ID insurance age nvisits
1 11047      Pres  47       3
2 12011      BCBS  41       1
3 12221      none  37       2
4 12241      BCBS  51       5
> help(merge)
> merged.data <- merge(d1, d2, by="ID",all=TRUE)
> merged.data # includes all patient IDs in either data set
     ID insurance age nvisits
1 11047      Pres  47       3
2 11066      <NA>  NA       1
3 11098      BCBS  48      NA
4 12011      BCBS  41       1
5 12221      none  37       2
6 12241      BCBS  51       5
7 13131      <NA>  NA       1
```

It is also possible to merge by more than one variable. For example if you don't have unique IDs, then you might have to merge by name (which might not be unique) in combination with other variables, such as address or phone number. Things can get messy, however. In one data set, names might have middle initials while they don't occur in the other data set, address info might be slightly different due to abbreviations, etc.