Flat data is data that is arranged with one case per row, with one column per variable– more or less. It's stored in a variety of formats, with different conventions. Our goal is to get it into the most useful format for analysis: what's known as tidy data.

```
library(gdata)  #gdata is finicky ... you may need to install it, but not load it.
```

```
## gdata: Unable to locate valid perl interpreter
## gdata:
## gdata: read.xls() will be unable to read Excel XLS and XLSX files
## gdata: unless the 'perl=' argument is used to specify the location
## gdata: of a valid perl intrpreter.
## gdata:
## gdata: (To avoid display of this message in the future, please
## gdata: ensure perl is installed and available on the executable
## gdata: search path.)
```

```
## gdata: Unable to load perl libaries needed by read.xls()
## gdata: to support 'XLX' (Excel 97-2004) files.
```

```
##
```

```
## gdata: Unable to load perl libaries needed by read.xls()
## gdata: to support 'XLSX' (Excel 2007+) files.
```

```
##
```

```
## gdata: Run the function 'installXLSXsupport()'
## gdata: to automatically download and install the perl
## gdata: libaries needed to support Excel XLS and XLSX formats.
```

```
##
## Attaching package: 'gdata'
```

```
## The following object is masked from 'package:stats':
##
##     nobs
```

```
## The following object is masked from 'package:utils':
##
##     object.size
```

```
## The following object is masked from 'package:base':
##
##     startsWith
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.3
```

```
## -- Attaching packages ---------------------------------------------------------------------- tidyv
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
## Warning: package 'tibble' was built under R version 3.5.3
```

```
## Warning: package 'tidyr' was built under R version 3.5.3
```

```
## Warning: package 'purrr' was built under R version 3.5.3
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
## Warning: package 'stringr' was built under R version 3.5.3
```

```
## Warning: package 'forcats' was built under R version 3.5.3
```

```
## -- Conflicts ------------------------------------------------------------------------ tidyverse_c
## x dplyr::combine() masks gdata::combine()
## x dplyr::filter()  masks stats::filter()
## x dplyr::first()   masks gdata::first()
## x purrr::keep()    masks gdata::keep()
## x dplyr::lag()     masks stats::lag()
## x dplyr::last()    masks gdata::last()
```

```
library(haven)
```

```
## Warning: package 'haven' was built under R version 3.5.3
```

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.5.3
```

CSV or other delimited files

We'll start with a csv file which is among the most common formats for datasets. CSV stands for *C* omma *S* eparated *V* alue, meaning that each row is divided into cells by commas. An end of line completes the row.

```
#Delimited files

#Load in the HSB dataset from the UCLA statistical computing site

hsb<-read_csv(file="https://stats.idre.ucla.edu/wp-content/uploads/2016/02/hsb2-2.csv")
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   female = col_double(),
##   race = col_double(),
##   ses = col_double(),
##   schtyp = col_double(),
##   prog = col_double(),
##   read = col_double(),
##   write = col_double(),
##   math = col_double(),
##   science = col_double(),
##   socst = col_double()
## )
```

```
write_csv(hsb,path="hsb.csv")

hsb%>%mutate(ses=as.character(ses))->hsb
hsb%>%mutate(ses=as.numeric(ses))->hsb


#Check it out
head(hsb)
```

```
## # A tibble: 6 x 11
```

```
##      id female  race   ses schtyp  prog  read write  math science socst
##   <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
## 1    70      0     4     1      1     1    57    52    41      47    57
## 2   121      1     4     2      1     3    68    59    53      63    61
## 3    86      0     4     3      1     1    44    33    54      58    31
## 4   141      0     4     3      1     3    63    44    47      53    56
## 5   172      0     4     2      1     2    47    52    57      53    61
## 6   113      0     4     2      1     2    44    52    51      63    61
```

```
##Need these for later
my.names<-names(hsb)

#Write this in a variety of formats to be used later
write_delim(hsb, path="hsb.txt",delim="\t")

write_delim(hsb, path="hsb_semicolon.txt",delim=";")

gdata::write.fwf(data.frame(hsb),file="hsb.dat",sep="",colnames=FALSE)
```

Quick exercise: write out the HSB file with a semicolon delimiter

Fixed width files

Fixed width files are an older file format that you don't see as much of any more. To read these in, you need a file that tells you the locations of the different variables, known as column positions or locations. You need to get the "widths" somewhere, usually a data dictionary.

```
#Fixed width files

my.widths=c(3,#id
            1, #female
            1, #race
            1, #ses
            1, #schtyp
            1, #prog
            2, #read
            2, #write
            2, #math
            2, #science
            2 #socst
            )

my_positions<-fwf_widths(my.widths)

hsb3<-read_fwf("hsb.dat",
         col_positions=my_positions)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
```

```
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double(),
##   X11 = col_double()
## )
```

```
head(hsb3)
```

```
## # A tibble: 6 x 11
##      X1    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1    70     0     4     1     1     1    57    52    41    47    57
## 2   121     1     4     2     1     3    68    59    53    63    61
## 3    86     0     4     3     1     1    44    33    54    58    31
## 4   141     0     4     3     1     3    63    44    47    53    56
## 5   172     0     4     2     1     2    47    52    57    53    61
## 6   113     0     4     2     1     2    44    52    51    63    61
```

```
names(hsb3)<-my.names

head(hsb3)
```

```
## # A tibble: 6 x 11
##      id female  race   ses schtyp  prog  read write  math science socst
##   <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
## 1    70      0     4     1      1     1    57    52    41      47    57
## 2   121      1     4     2      1     3    68    59    53      63    61
## 3    86      0     4     3      1     1    44    33    54      58    31
## 4   141      0     4     3      1     3    63    44    47      53    56
## 5   172      0     4     2      1     2    47    52    57      53    61
## 6   113      0     4     2      1     2    44    52    51      63    61
```

Other programming languages

Other statistical programs have their own file formats. These are easy for these programs to read in. R can understand all of them, if the `haven` packages is used.

Stata

Stata is a very popular statistical programming language among economists and social scientists. Stata files are stored in `dta` format.

```
# Stata

hsb_stata<-read_dta("https://stats.idre.ucla.edu/stat/stata/notes/hsb2.dta")

head(hsb_stata)
```

```
## # A tibble: 6 x 11
##      id  female    race     ses  schtyp    prog  read write  math science
##   <dbl> <dbl+l> <dbl+l> <dbl+l> <dbl+l> <dbl+l> <dbl> <dbl> <dbl>   <dbl>
## 1    70 0 [mal~ 4 [whi~ 1 [low] 1 [pub~ 1 [gen~    57    52    41      47
## 2   121 1 [fem~ 4 [whi~ 2 [mid~ 1 [pub~ 3 [voc~    68    59    53      63
## 3    86 0 [mal~ 4 [whi~ 3 [hig~ 1 [pub~ 1 [gen~    44    33    54      58
```

```
## 4   141 0 [mal~ 4 [whi~ 3 [hig~ 1 [pub~ 3 [voc~    63    44    47      53
## 5   172 0 [mal~ 4 [whi~ 2 [mid~ 1 [pub~ 2 [aca~    47    52    57      53
## 6   113 0 [mal~ 4 [whi~ 2 [mid~ 1 [pub~ 2 [aca~    44    52    51      63
## # ... with 1 more variable: socst <dbl>
```

SPSS

SPSS is one of the oldest statistical programming languages out there. SPSS data files are stored in `.sav` format.

```
#SPSS
example_spss<-read_spss("https://stats.idre.ucla.edu/stat/data/binary.sav")

head(example_spss)
```

```
## # A tibble: 6 x 4
##   admit   gre   gpa  rank
##   <dbl> <dbl> <dbl> <dbl>
## 1     0   380  3.61     3
## 2     1   660  3.67     3
## 3     1   800  4        1
## 4     1   640  3.19     4
## 5     0   520  2.93     4
## 6     1   760  3        2
```

SAS

Lots of large organizations and state agencies use SAS for statistical programming. SAS output can be really funky. It's usually (but not always) stored as `.sas7bdat`.

```
#SAS
hsb_sas<-read_sas("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/hsb2.sas7bdat")

head(hsb_sas)
```

```
## # A tibble: 6 x 11
##      id female  race   ses schtyp  prog  read write  math science socst
##   <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
## 1     3      0     1     1      1     2    63    65    48      63    56
## 2     5      0     1     1      1     2    47    40    43      45    31
## 3    16      0     1     1      1     3    47    31    44      36    36
## 4    35      1     1     1      2     1    60    54    50      50    51
## 5     8      1     1     1      1     2    39    44    52      44    48
## 6    19      1     1     1      1     1    28    46    43      44    51
```

Excel (sigh)

In most work settings you'll work with excel files. To get these into shape you'll have to do some wrangling. Below I show how this is done with data in a common reporting format.

```
## Web page:
##http://nces.ed.gov/programs/digest/d14/tables/dt14_204.10.asp

if(file.exists("free.xls")==FALSE){
```

```
  download.file("http://nces.ed.gov/programs/digest/d14/tables/xls/tabn204.10.xls",destfile="free.xls")
free<-read_excel("free.xls",skip=4,col_names=FALSE)
}else{
  free<-read_excel("free.xls",skip=4,col_names=FALSE)
}
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ... and 14 more problems
```

```
head(free)
```

```
## # A tibble: 6 x 19
##   ...1    ...2 ...3    ...4   ...5 ...6    ...7   ...8 ...9   ...10  ...11
##   <chr>  <dbl> <chr>  <dbl>  <dbl> <chr>  <dbl>  <dbl> <chr>  <dbl>  <dbl>
## 1 Unit~ 4.66e7 "\\1~ 4.89e7 4.90e7 "\\1~ 4.91e7 1.78e7 "\\1~ 2.35e7 2.43e7
## 2 Alab~ 7.28e5 <NA>  7.30e5 7.32e5 <NA>  7.40e5 3.35e5 <NA>  4.02e5 4.20e5
## 3 Alas~ 1.05e5 <NA>  1.32e5 1.31e5 <NA>  1.31e5 3.25e4 <NA>  5.07e4 5.32e4
## 4 Ariz~ 8.78e5 "\\2~ 1.07e6 1.02e6 <NA>  9.90e5 2.74e5 "\\2~ 4.82e5 5.12e5
## 5 Arka~ 4.50e5 <NA>  4.82e5 4.83e5 <NA>  4.86e5 2.05e5 <NA>  2.92e5 2.94e5
## 6 Cali~ 6.05e6 <NA>  6.17e6 6.20e6 "\\2~ 6.18e6 2.82e6 <NA>  3.34e6 3.35e6
## # ... with 8 more variables: ...12 <chr>, ...13 <dbl>, ...14 <dbl>,
## #   ...15 <chr>, ...16 <dbl>, ...17 <dbl>, ...18 <chr>, ...19 <dbl>
```

The resulting dataset is a mess. We need to get rid of white space, get columns named appropriately, and get rid of any unwanted columns.

Side Note: Indexes in R

One of R's big advantages is that can take index arguments. Every R dataset is indexed. Using square brackets after the dataset name allows the user to select any part of the dataset, or remove any part of the dataset. The format is always:

`dataset_name[row_number, column_number]`

To select the 5th row and the 10th column, we would type:

`dataset_name[5,10]`

To select the 5th and 7th rows and the 10th through the 15th column, we would type:

`dataset_name[c(5,7),c(10:15)]`

To NOT select the 7th row, we would type:

`dataset_name[-7,]`

Notice that no column is selected, which R interprets as using ALL columns.

Tidying a Dataset

First, let's git rid of unwanted columns by selecting a list of things we don't want, using the `-(c(element1,element2,...))` setup.

```
# Now need to clean up
#Get rid of unwanted columns

free2<-free[ ,-(c(3,6,9,12,15,18))]


#Get rid of unwanted rows
free2<-free2%>%filter(is.na(...1)==FALSE)
```

Now we're going to select some things we DO want, by using the `number_1:number_x` setup.

```
##50 states plus dc only
free2<-free2[2:52,]

head(free2)
```

```
## # A tibble: 6 x 13
##   ...1    ...2   ...4   ...5   ...7   ...8  ...10  ...11  ...13 ...14 ...16
##   <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
## 1 Alab~ 7.28e5 7.30e5 7.32e5 7.40e5 3.35e5 4.02e5 4.20e5 4.30e5  46.0  55.1
## 2 Alas~ 1.05e5 1.32e5 1.31e5 1.31e5 3.25e4 5.07e4 5.32e4 5.31e4  30.8  38.4
## 3 Ariz~ 8.78e5 1.07e6 1.02e6 9.90e5 2.74e5 4.82e5 5.12e5 5.14e5  31.2  45.2
## 4 Arka~ 4.50e5 4.82e5 4.83e5 4.86e5 2.05e5 2.92e5 2.94e5 2.99e5  45.6  60.5
## 5 Cali~ 6.05e6 6.17e6 6.20e6 6.18e6 2.82e6 3.34e6 3.35e6 3.48e6  46.6  54.1
## 6 Colo~ 7.24e5 8.43e5 8.54e5 8.63e5 1.95e5 3.36e5 3.49e5 3.59e5  26.9  39.9
## # ... with 2 more variables: ...17 <dbl>, ...19 <dbl>
```

```
tail(free2)
```

```
## # A tibble: 6 x 13
##   ...1    ...2   ...4   ...5   ...7   ...8  ...10  ...11  ...13 ...14 ...16
##   <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
## 1 Verm~ 1.02e5 8.51e4 8.35e4 8.36e4  23986  31339  32748  32581  23.5  36.8
## 2 Virg~ 1.07e6 1.25e6 1.23e6 1.24e6 320233 458879 480821 487463  30.0  36.7
## 3 Wash~ 1.00e6 1.04e6 1.04e6 1.05e6 326295 418065 463246 474940  32.5  40.1
## 4 West~ 2.86e5 2.83e5 2.83e5 2.83e5 143446 145605 149407 148493  50.1  51.5
## 5 Wisc~ 8.59e5 8.72e5 8.70e5 8.71e5 219276 342660 354527 360803  25.5  39.3
## 6 Wyom~ 8.99e4 8.88e4 8.94e4 9.15e4  43483  32968  33145  34617  48.4  37.1
## # ... with 2 more variables: ...17 <dbl>, ...19 <dbl>
```

Okay, this is looking better. Let's add some nice column names.

```
names(free2)<-c("state",
                "total_2000",
                "total_2010",
                "total_2011",
                "total_2012",
                "frl_2000",
                "frl_2010",
                "frl_2011",
                "frl_2012",
                "pc_frl_2000",
                "pc_frl_2010",
                "pc_frl_2011",
                "pc_frl_2012")
```

Quick Exercise: Read in this file: http://nces.ed.gov/programs/digest/d14/tables/xls/tabn302.10.xls

Tidy data

Tidy data follows two key principles: each column is one variable and one variable only, while each row is a case. Below, I show how to make the data from the above spreadsheet tidy, and why we would do this.

In the first step, I select the state name and the "total" columns.

```
free_total<-free2%>%select(state,
                           total_2000,
                           total_2010,
                           total_2011,
                           total_2012)

names(free_total)<-c("state","2000","2010","2011","2012")
```

In the second step, I use the `gather` command to place all of the "total" variables into a single variable, with the key being set to year, and the value being set tot total number of students.

```
free_total<-free_total%>%
  gather(`2000`,`2010`,`2011`,`2012`,key=year,value=total_students)
```

I then repeat that process with the number of free and reduced price lunch students.

```
frl_total<-free2%>%select(state,
                           frl_2000,
                           frl_2010,
                           frl_2011,
                           frl_2012)

names(frl_total)<-c("state","2000","2010","2011","2012")

frl_total<-frl_total%>%gather(`2000`,`2010`,`2011`,`2012`,key=year,value=frl_students)
```

Now I join these two datasets, using what's called a "left" join. This means that the merge will keep all of the data in the first dataset, and only those rows in the second dataset that match based on state and year, which are specified in the command.

```
free_tidy<-left_join(free_total,frl_total,by=c("state","year"))

free_tidy
```

```
## # A tibble: 204 x 4
##    state                year  total_students frl_students
##    <chr>                <chr>          <dbl>        <dbl>
##  1 Alabama ..........   2000          728351       335143
##  2 Alaska ...........   2000          105333        32468
##  3 Arizona ..........   2000          877696       274277
##  4 Arkansas .........   2000          449959       205058
##  5 California .......   2000         6050753      2820611
##  6 Colorado .........   2000          724349       195148
##  7 Connecticut ......   2000          562179       143030
##  8 Delaware .........   2000          114676        37766
##  9 District of Columbia 2000           68380        47839
## 10 Florida ..........   2000         2434755      1079009
## # ... with 194 more rows
```

This "tidy" format for datasets is MUCH easier to work with. Wickham has an extensive section on this, but just try to get a feel for it and make sure that your data follows the basic principles laid out here. It will

make your life much easier.

The principles are:

1. Each variable forms a column.
2. Each observation forms a row
3. Each type of observational unit forms a table.

To be formal, in relational databases, this is known as the 3rd normal form, but that's not important. The key is that if it's one variable (say total students) it goes in one column. Each row is formed by the unique set of characteristics that define an observation. In this case it's state and year that define an observation.

Using this tidy dataset, we can easily ask R to give us the total number of free and reduced price lunch students from each year.

```
## Total by year
free_tidy%>%group_by(year)%>%summarize(total_students=sum(total_students),
                                       total_frl_student=sum(frl_students))%>%
  mutate(pct_frl=total_frl_student/total_students)
```

```
## # A tibble: 4 x 4
##   year  total_students total_frl_student pct_frl
##   <chr>          <dbl>             <dbl>   <dbl>
## 1 2000        46579068          17839867   0.383
## 2 2010        48941267          23544479   0.481
## 3 2011        48995812          24291646.  0.496
## 4 2012        49084316          25188294   0.513
```

Quick Exericse: now add in percent of students eligible by state

Output

Most of the time, you should store your data as a csv file. This will ensure that pretty much anyone can take a look at it. If you're sure that the only users will be other R users (why would you be sure of this?), then feel free to save it as an `.Rdata` file.

```
#Saving as an R file
# save(free2,file="frl.Rdata")
```

Otherwise, go with csv, like so:

```
#Outputting delimited
write_csv(free2,"frl.csv")
```

Databses and flat files

As we'll discuss later, databases are linked collections of flat files, usually stored in `.csv` format. If you're working with a database technician and would like a flat file for analysis, the way to ask for it is to request a flat file in csv format. Most database analysts will know what you mean by this. We'll cover databases in much more detail later.