

Class 2. Handling data in R

Creating, editing, reading, & exporting data frames; sorting, subsetting, combining

Goals:

- (1) Creating matrices and dataframes: `cbind` and `as.data.frame`
- (2) Editing data: `edit(data)`
- (3) Problems mixing numerical and character data
- (4) Importing or exporting data from/to a file: `read.table` and `write.table`
- (5) Sorting: `sort` and `order`
- (6) Subsetting: `subset` function
- (7) Combining data: `cbind`, `rbind`, `merge`
- (8) Handling dates in R

Brief cheat sheet of major functions covered here.

| | |
|---|--|
| Create a vector | <code>shoe<-c(8,7,8.5,6,10.5,11,7,6,12,10)</code> |
| bind vectors together side-by-side | <code>a<-cbind(shoe,hate,belt)</code> |
| show part of a matrix | <code>a[1:5,2:3] #rows 1-5, cols 2-3</code> |
| mean of a column of a matrix | <code>mean(a[,1]) #mean of column 1</code> |
| mean of a row of a matrix | <code>mean(a[2,]) #mean of row 2</code> |
| convert matrix to data frame | <code>adf<-as.data.frame(a) #matrix a to data frame adf</code> |
| open Data Editor | <code>adf<-edit(adf) #edit data frame adf</code> |
| change a variable mode | <code>> adf[,4]<-as.numeric(adf[,4]) #make 4th column of adf numeric</code> |
| read a comma-delimited file with chooser | <code>data1<-read.table(file.choose(),sep=",",header=TRUE)</code> |
| read a comma-delimited file with headers | <code>data1<-read.table(file="mydata.csv",sep=",",header=TRUE)</code> |
| read a comma-delimited file with headers | <code>data1<-read.csv(file="mydata.csv",header=TRUE)</code> |
| read a tab-delimited file with headers | <code>data1<-read.table(file="mydata.txt",sep="\t",header=TRUE)</code> |
| write a comma-delimited file with headers | <code>write.table(statout,"statout.csv",sep=",",quote=FALSE, col.names=NA,row.names=TRUE)</code> |
| sort a vector | <code>shoe<-sort(shoe)</code> |
| order a data frame | <code>adf[order(adf\$sex),] #order adf by variable sex</code> |
| order a data frame by multiple columns | <code>adf[order(adf\$sex,adf\$shoe),]</code> |
| order a data frame in descending order | <code>adf[order(adf\$sex,-adf\$shoe),]</code> |
| subsetting data by values in a variable | <code>males<-adf[adf\$sex=="m",]</code> |
| subsetting data by values in a variable | <code>males<-subset(adf,subset=adf\$sex=="m")</code> |
| subsetting on multiple factors | <code>bigmalefeet<-adf[adf\$shoe>=10 & adf\$sex=="m",]</code> |
| subsetting particular columns | <code>justsexshoes<-adf[,c("student","sex","shoe")]</code> |
| stack columns of two files vertically | <code>c<-rbind(a,b)</code> |
| merge data frames on a common variable | <code>mergedData<-merge(shoes,moredata,by="student")</code> |

Normally you will set up your data in a spreadsheet or database somewhere, and then import the data into R. However, because it is really useful and sometimes necessary to rearrange data inside R, let's start by creating matrices and data frames the tedious way.

(1) MATRICES AND DATAFRAMES (tedious illustrative version)

Creating dataframes using cbind and as.data.frame

First create three separate **vectors** of the same length

```
shoe<-c(8,7,8.5,6,10.5,11,7,6,12,10)
hat<-c(7.5,7,7.25,6.75,8,7.75,7,7.25,7.5,8)
belt<-c(30,26,34,29,31,25,32,38,33,31)
shoe #look at the vector shoe
```

```
a<-cbind(shoe,hat,belt) #bind them together into a matrix
a #look at the matrix
```

```
      shoe hat belt
[1,]    8 7.50  30
[2,]    7 7.00  26
[3,]    8 7.25  34
[4,]    6 6.75  29
[5,]   10 8.00  31
[6,]   11 7.75  25
[7,]    7 7.00  32
[8,]    6 7.25  38
[9,]   12 7.50  33
[10,]  10 8.00  31
```

look at and use parts of the matrix using the matrixName[rows,columns] convention

```
a[1:5,2:3] #show part of the matrix
```

```
      hat belt
[1,] 7.50  30
[2,] 7.00  26
[3,] 7.25  34
[4,] 6.75  29
[5,] 8.00  31
```

```
mean(a[,1]) #calculate the mean of values in column 1 of the matrix
[1] 8.6
```

Note: you can't use the \$ format to call column names of a matrix

```
a$shoe
```

Error in a\$shoe : \$ operator is invalid for atomic vectors

```
a[,"shoe"] # but this format works for a matrix
```

```
[1] 8.0 7.0 8.5 6.0 10.5 11.0 7.0 6.0 12.0 10.0
```

#Most often, you'll want to have your data in a data frame

```
adf<-as.data.frame(a) # convert the matrix to a data frame
```

```
mean(adf$shoe) #calculate the mean of shoe sizes in the data frame
```

```
[1] 8.6
```

#note: could have created the data frame in one step

```
adf<-as.data.frame(cbind(shoe,hat,belt))
```

Compare the structure of the matrix "a" and the data frame "adf"

```
str(a)
num [1:10, 1:3] 8 7 8 6 10 11 7 6 12 10 ...
- attr(*, "dimnames")=List of 2
..$ : NULL
..$ : chr [1:3] "shoe" "hat" "belt"
```

```
str(adf)
'data.frame':    10 obs. of  3 variables:
 $ shoe : num  8 7 8 6 10 11 7 6 12 10
 $ hat  : num  7.5 7 7.25 6.75 8 7.75 7 7.25 7.5 8
 $ belt : num  30 26 34 29 31 25 32 38 33 31
```

#now you can refer to specific columns by name

```
mean(adf$belt) #adf refers to the data frame, and then columns after $
[1] 30.9
```

Let's say you also have data on individual students and their sex:

#Read in these vectors

```
student<-
c("biao","blair","cath","cyn","dave","jae","jenn","amy","joe","jorge")
sex<-c("m","f","f","f","m","m","f","f","f","m")
```

As long as they are in the same order, you can add students and sex to the "adf" dataframe

```
adf<-cbind(adf,student,sex)
head(adf,3)
  shoe hat belt student sex
1    8 7.50  30     biao  m
2    7 7.00  26    blair  f
3    8 7.25  34 catherine f
```

str(adf) #look at the structure

#note that R automatically recognized student and sex as factors

```
'data.frame':    10 obs. of  5 variables:
 $ shoe : num  8 7 8.5 6 10.5 11 7 6 12 10
 $ hat  : num  7.5 7 7.25 6.75 8 7.75 7 7.25 7.5 8
 $ belt : num  30 26 34 29 31 25 32 38 33 31
 $ student: Factor w/ 10 levels "amy","biao","blair",...: 2 3 4 5 6 7 8 1 9 10
 $ sex   : Factor w/ 2 levels "f","m": 2 1 1 1 2 2 1 1 1 2
```

#Re-order the columns to make the data frame prettier

#by assigning the preferred order of columns in a vector

```
adf<-adf[,c('student','sex','shoe','hat','belt')]
head(adf,2)
```

```
  student sex shoe hat belt
1     biao  m    8  7.5  30
2     blair  f    7    7  26
```

2. EDITING DATA

We now have the following data frame called "adf"

```
adf
  student sex shoe hat belt
1     biao  m   8 7.50  30
2     blair f   7 7.00  26
3 catherine f   8 7.25  34
4   cynthia f   6 6.75  29
5       dave m  10 8.00  31
6        jae m  11 7.75  25
7       jenn f   7 7.00  32
8   jennie  f   6 7.25  38
9        joe f  12 7.50  33
10    jorge  m  10 8.00  31
```

Notice that the sex of joe is incorrect (presumably).

Using command lines to edit data

You can change this single value in the current data set in a couple different ways

```
adf[9,2]<-"m"
```

or

```
adf[9,"sex"]="m"
```

or

```
rownames(adf)<-adf$student; adf["joe","sex"]<-"m"
```

#note that this change is only retained in your current session

Using R Data Editor to edit data

```
adf<-edit(adf) # opens your data in the Data Editor, like this
```

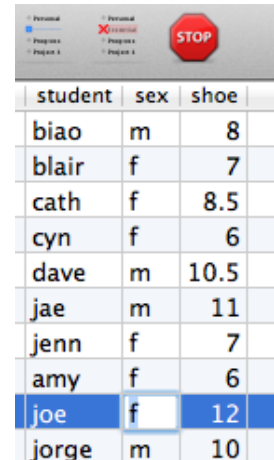
Double click on the cell you want to change,
fix the data,
and when ready, simply close the window

#note 1: this change is only retained in your current session

#and not in any external data file you have

#note 2: using only edit(adf) does not save any changes you make

```
adf
  student sex shoe hat belt
1     biao  m   8 7.50  30
2     blair f   7 7.00  26
3 catherine f   8 7.25  34
4   cynthia f   6 6.75  29
5       dave m  10 8.00  31
6        jae m  11 7.75  25
7       jenn f   7 7.00  32
8   jennie  f   6 7.25  38
9        joe m  12 7.50  33
10    jorge  m  10 8.00  31
```



| student | sex | shoe |
|---------|-----|------|
| biao | m | 8 |
| blair | f | 7 |
| cath | f | 8.5 |
| cyn | f | 6 |
| dave | m | 10.5 |
| jae | m | 11 |
| jenn | f | 7 |
| amy | f | 6 |
| joe | f | 12 |
| jorge | m | 10 |

3. PROBLEMS MIXING NUMERICAL AND CHARACTER DATA

So didn't we just import all five columns at once to create adf? It should be easy to do this:

```
shoe<-c(8,7,8.5,6,10.5,11,7,6,12,10)
hat<-c(7.5,7,7.25,6.75,8,7.75,7,7.25,7.5,8)
belt<-c(30,26,34,29,31,25,32,38,33,31)
student<-
c("biao","blair","cath","cyn","dave","jae","jenn","amy","joe","jorge")
sex<-c("m","f","f","f","m","m","f","f","m","m")
adf<-as.data.frame(cbind(student,sex,shoe,hat,belt))
adf
```

```
  student sex shoe  hat belt
1  biao   m    8  7.5   30
2  blair  f    7    7   26
3  cath  f   8.5  7.25  34
4   cyn  f    6  6.75  29
5  dave  m  10.5    8   31
6   jae  m   11  7.75  25
7  jenn  f    7    7   32
8   amy  f    6  7.25  38
9   joe  m   12  7.5   33
10 jorge  m   10    8   31
```

Looks great, but:

```
mean(adf$shoe)
[1] NA
Warning message:
In mean.default(adf$shoe) :
  argument is not numeric or logical: returning NA
```

This is because unless you already have a data frame created, when you mix character and numerical data R treats everything as if it were character data in cbind (and character data automatically become Factors in a data frame) :

```
str(adf)
'data.frame':   10 obs. of  5 variables:
 $ student: Factor w/ 10 levels "amy","biao","blair",...: 2 3 4 5 6 7 8 1 9 10
 $ sex    : Factor w/ 2 levels "f","m": 2 1 1 1 2 2 1 1 2 2
 $ shoe   : Factor w/ 8 levels "10","10.5","11",...: 7 6 8 5 2 3 6 5 4 1
 $ hat    : Factor w/ 6 levels "6.75","7","7.25",...: 4 2 3 1 6 5 2 3 4 6
 $ belt   : Factor w/ 9 levels "25","26","29",...: 4 2 8 3 5 1 6 9 7 5
```

You can fix these one at a time:

```
adf$shoe<-as.numeric(adf$shoe)
adf$hat<-as.numeric(adf$hat)
adf$belt<-as.numeric(adf$belt)
str(adf)
'data.frame':   10 obs. of  5 variables:
 $ student: Factor w/ 10 levels "amy","biao","blair",...: 2 3 4 5 6 7 8 1 9 10
 $ sex    : Factor w/ 2 levels "f","m": 2 1 1 1 2 2 1 1 2 2
 $ shoe   : num  7 6 8 5 2 3 6 5 4 1
 $ hat    : num  4 2 3 1 6 5 2 3 4 6
 $ belt   : num  4 2 8 3 5 1 6 9 7 5
```

You could instead go straight to a data frame
`adf<-data.frame(student,sex,shoe,hate,belt)`

`str(adf)`

```
'data.frame':      10 obs. of  5 variables:
 $ student: Factor w/ 10 levels "amy","biao","blair",...: 2 3 4 5 6 7 8 1 9 10
 $ sex    : Factor w/ 2 levels "f","m": 2 1 1 1 2 2 1 1 2 2
 $ shoe   : num  8 7 8.5 6 10.5 11 7 6 12 10
 $ hat    : num  7.5 7 7.25 6.75 8 7.75 7 7.25 7.5 8
 $ belt   : num  30 26 34 29 31 25 32 38 33 31
```

4. IMPORTING AND WRITING DATA FROM/TO A FILE

Generally, prepare your data in a spreadsheet or database like Excel, Google docs, Filemaker Pro, Access, or MySQL, and then import the data into R. Protocols to import data directly from .xls and .fmp files exist (e.g., read.xls in the package gdata) – but can be messy. Much easier is to import a flat file of data saved as a tab-delimited (.txt) or comma-delimited (.csv) file. I prefer .csv files because of occasional glitches with tab-delimited exports, but generally either works. See the short-form version in *Class1 Getting started*.

Organizing your data

Some general guidelines to organize your data cleanly and make it easy to use in R.

- (1) The first row should contain variable names. Use short names without spaces.
- (2) Avoid special characters or punctuation (e.g., !@#\$%^&*()-+{}[];":'<>/?~`) and it is best to replace spaces with an underscore (e.g., Species_Num). Avoid commas.
- (3) If you have an identifying number for the record in each row, put it in the first column.

Exporting data from Excel

From the File menu in Excel, choose Save As. From the Format box, choose CSV (Comma delimited) or Text (Tab delimited) (I prefer CSV, but either will work. I'll do everything with CSV format from here out). Save it to your working directory

Create a data frame from the exported data

Assuming you have already set the working directory (see setwd) and your .csv data file is in that directory, you can import the data using the read.table function as follows.

```
mock<-read.table(file="mockdata.csv",header=TRUE,sep=",")
```

or, if you prefer, use the chooser to navigate to the file as

```
mock<-read.table(file.choose(),header=TRUE,sep=",")
```

Note1: for tab delimited files, use sep="\t"; for space delimited, use sep=" "

Note2: if the file you want to read isn't in your working directory, use file.choose() or simply include the full path to the file, wherever it is. e.g.,

```
mock<-
```

```
read.table(file="/Users/ggilbert/Desktop/mockdata.csv",header=TRUE,sep=",")
```

Note 3: By default, read.table converts any character string to a factor – a categorical variable that allows you to use it as a group for analyses. Most often that is probably what you want. If you instead want it as a character string, you can either declare the variable type for each column individually as a vector in the colClasses argument read.table statement, or, after reading in as shown above, convert the factor to character string state. E.g., for the variable mock\$species, which originally was imported as a factor, convert to a string with:

```
mock$species<-as.character(mock$species).
```

Note 4: If you want import your all-numerical data as a matrix, use the scan function instead.

Looking at the imported data frame

```
str(mock) # Check the type of all the variables in the data frame called mock
```

```
head(mock) # prints out the first 5 lines of the data frame called mock
```

An assortment of ways to read files into a data frame

#read a csv file, with headers, into a data frame

```
data1<-read.table(file="mydata.csv", header=TRUE, sep=",")
```

#read a csv file, with headers, into a data frame using read.csv

```
data1<-read.csv(file="mydata.csv", header=TRUE)
```

#read a tab-delimited file, with headers, into a data frame

```
data1<-read.table(file="mydata.txt", header=TRUE, sep="\t")
```

#read a space-delimited file, with headers, into a data frame

```
data1<-read.table(file="mydata.txt", header=TRUE, sep=" ")
```

#read a csv file, without headers, into a dataframe

#and then add in column names

```
data1<-read.table(file="mydata.csv", header=FALSE, sep=",")
```

```
names(data1)<-c("treatment","mass","height")
```

#read in a tab-delimited file, skipping 3 lines of extraneous information,

#at the top, and with headers in row 4, and data starting in row 5

```
data1<-read.table(file="mydata.csv", skip=3, header=TRUE, sep="\t")
```

#read in only the first 150 lines of a tab-delimited file

```
data1<-read.table(file="mydata.csv", header=TRUE, sep="\t", nrows=150)
```

#read in a csv file, with the first column of data included as row names.

```
data1<-read.table(file="mydata.csv", header=TRUE, sep=",",row.names=1)
```

Examples of how to write a data frame to a file to use other programs or back into R

#Create a tab-delimited text file, including headers,

#replacing the file each time it is written

#note that col.names=NA helps line up the headers with the data correctly

```
write.table(data1,file="data1out.txt", col.names=NA, sep="\t",append=FALSE)
```

#Create a tab-delimited text file, appending the data to the end of

the file of that name

```
write.table(data1,file="data1out.txt", sep="\t",append=TRUE,col.names=FALSE)
```

#Create a comma-delimited csv file, including headers,

#replacing the file each time it is written

#note that col.names=NA helps line up the headers with the data correctly

```
write.table(data1,file="data1out.csv", col.names=NA, sep=",",append=FALSE)
```

#Create a comma-delimited csv file, including headers,

#replacing the file each time it is written

```
write.csv(data1,file="data1out.csv")
```


Aside: creating TOY MATRICES AND DATAFRAMES

Toy Matrices: a matrix with 6 rows and 4 columns of random data, representing soil moisture on a physical grid with 10-m spacing between points, starting at location 0,0.

```
vals<-round(runif(24,0,100),0) #create 24 random numbers from 0-100
MyMat<-matrix(vals,nrow=6,ncol=4) #put those values into a 6x4 matrix
#note that the matrix is filled column by column from vals
colnames(MyMat)<-c(seq(from=0,to=30,by=10)) #assign column names
rownames(MyMat)<-c(seq(from=0,to=50,by=10)) #assign row names
MyMat #take a look
  0 10 20 30
0 36 0 87 17
10 1 29 29 60
20 58 62 4 48
30 6 85 47 22
40 52 22 34 30
50 65 99 34 56
```

Toy data frames:

First create a placeholder data frame with space for 20 records, specifying the type /mode for each of 4 columns.

```
data2<-data.frame(group=character(20),
mass=numeric(20),length=numeric(20),status=character(20))
str(data2)
'data.frame':   20 obs. of  4 variables:
 $ group : Factor w/ 1 level "": 1 1 1 1 1 1 1 1 1 1 ...
 $ mass  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ length: num  0 0 0 0 0 0 0 0 0 0 ...
 $ status: Factor w/ 1 level "": 1 1 1 1 1 1 1 1 1 1 ...
```

#Populate the data frame with random data.

```
data2$group<-c(rep("A",10),rep("B",10))
data2$mass<-c(runif(10,0,20),runif(10,20,40))
data2$length<-c(runif(10,10,30),runif(10,15,35))
data2$status<-c(rep(c("m","f","f","m"),5))
data2$density<-data2$mass/data2$length
data2[c(1:5,9:13),] #look at part of the data frame
  group    mass  length status  density
1     A 14.141854 12.07597      m 1.1710742
2     A 11.101069 13.96204      f 0.7950894
3     A  3.303160 21.47031      f 0.1538478
4     A  7.202421 25.16997      m 0.2861514
5     A  9.880299 10.80644      m 0.9142974
9     A 16.763635 27.93993      m 0.5999884
10    A  5.696507 22.38397      f 0.2544904
11    B 22.498757 33.66048      f 0.6684028
12    B 24.835708 18.72881      m 1.3260699
13    B 32.157256 22.84468      m 1.4076473
```

5. SORTING DATA

Sorting vectors

```
shoe<-c(8,7,8,6,10,11,7,6,12,10)
shoe
[1] 8 7 8 6 10 11 7 6 12 10
sort(shoe)
[1] 6 6 7 7 8 8 10 10 11 12
```

```
#To keep it in that order, assign the vector to itself, but be careful
#this means that any cbind functions you do with have this vector
#in a different order from other vectors
shoe<-sort(shoe)
shoe
[1] 6 6 7 7 8 8 10 10 11 12
```

Sorting Data frames

Sort doesn't work on data frames, unless you only want to sort one column but leave all the others as they were (can be useful for randomization tests. To sort a whole data frame, use `order` .

Start with data frame "adf"

```
adf
  student sex shoe hat belt
1   biao   m  8.0 7.50  30
2  blair   f  7.0 7.00  26
3   cath   f  8.5 7.25  34
4    cyn   f  6.0 6.75  29
5   dave   m 10.5 8.00  31
6    jae   m 11.0 7.75  25
7   jenn   f  7.0 7.00  32
8    amy   f  6.0 7.25  38
9    joe   m 12.0 7.50  33
10 jorge   m 10.0 8.00  31
```

To order adf by sex:

```
adf<-adf[order(adf$sex),]
adf
#sort data frame adf on column sex
  student sex shoe hat belt
2  blair   f  7.0 7.00  26
3   cath   f  8.5 7.25  34
4    cyn   f  6.0 6.75  29
7   jenn   f  7.0 7.00  32
8    amy   f  6.0 7.25  38
1  biao   m  8.0 7.50  30
5   dave   m 10.5 8.00  31
6    jae   m 11.0 7.75  25
9    joe   m 12.0 7.50  33
10 jorge   m 10.0 8.00  31
```

To order by multiple columns:

```
adf<-adf[order(adf$sex,adf$shoe),]
adf
  student sex shoe hat belt
4    cyn   f  6.0 6.75  29
8    amy   f  6.0 7.25  38
2  blair   f  7.0 7.00  26
7   jenn   f  7.0 7.00  32
3   cath   f  8.5 7.25  34
1   biao   m  8.0 7.50  30
10 jorge   m 10.0 8.00  31
5   dave   m 10.5 8.00  31
6    jae   m 11.0 7.75  25
9    joe   m 12.0 7.50  33
```

To order in descending order, add a "-"

```
adf<-adf[order(adf$sex,-adf$shoe),]
adf
  student sex shoe hat belt
3   cath   f  8.5 7.25  34
2  blair   f  7.0 7.00  26
7   jenn   f  7.0 7.00  32
4    cyn   f  6.0 6.75  29
8    amy   f  6.0 7.25  38
9    joe   m 12.0 7.50  33
6    jae   m 11.0 7.75  25
5   dave   m 10.5 8.00  31
10 jorge   m 10.0 8.00  31
1   biao   m  8.0 7.50  30
```

6. SUBSETTING DATA

Selecting records with particular values

Start with the dataframe "adf"

```
adf
  student sex shoe hat belt
3  cath   f  8.5 7.25  34
2  blair  f  7.0 7.00  26
7  jenn   f  7.0 7.00  32
4  cyn    f  6.0 6.75  29
8  amy    f  6.0 7.25  38
9  joe    m 12.0 7.50  33
6  jae    m 11.0 7.75  25
5  dave   m 10.5 8.00  31
10 jorge  m 10.0 8.00  31
1  biao   m  8.0 7.50  30
```

Start with the dataframe "adf"

We want to make a different data frame called "males" that includes only sex="m"

This uses the general format:

```
NewDF <- OldDF[rows where sex = "m", all columns]
```

note1: remember that "==" means is equal to

```
males<-adf[adf$sex=="m",]
males
```

```
  student sex shoe hat belt
9  joe    m 12.0 7.50  33
6  jae    m 11.0 7.75  25
5  dave   m 10.5 8.00  31
10 jorge  m 10.0 8.00  31
1  biao   m  8.0 7.50  30
```

```
#note: you get the same thing with males<-adf[adf$sex!="f",]
```

You can also do exactly the same thing with the subset function, which takes the form:

```
newDF<-subset(Dataframe,subset=SelectionVariable==VariableValue)
```

```
males<-subset(adf,subset=adf$sex=="m") #gives same subset as above
```

Select on two variables together using & (and) or | (or) to define sets

```
malebigfoot<-adf[adf$shoe>=10 & adf$sex=="m",]
```

```
malebigfoot
```

```
  student sex shoe hat belt
9  joe    m 12.0 7.50  33
6  jae    m 11.0 7.75  25
5  dave   m 10.5 8.00  31
10 jorge  m 10.0 8.00  31
```

A complex example (all the females and the males with shoes < size 10)

```
allfem_smallmale<-adf[(adf$shoe<10 & adf$sex=="m")|adf$sex=="f",]
```

```
allfem_smallmale
```

```
  student sex shoe hat belt
3  cath   f  8.5 7.25  34
2  blair  f  7.0 7.00  26
7  jenn   f  7.0 7.00  32
4  cyn    f  6.0 6.75  29
8  amy    f  6.0 7.25  38
1  biao   m  8.0 7.50  30
```

Selecting particular columns for new data frame

Sometimes you want to create a separate data frame with only some of the columns from a larger data frame.

```
justsexshoes<-adf[,c("student", "sex", "shoe")]
```

```
justsexshoes
  student sex shoe
3   cath   f  8.5
2  blair   f  7.0
7   jenn   f  7.0
4    cyn   f  6.0
8    amy   f  6.0
9    joe   m 12.0
6    jae   m 11.0
5   dave   m 10.5
10 jorge   m 10.0
1   biao   m  8.0
```

#these will give you the same thing:

```
justsexshoes<-adf[,c(1:3)]
```

#or using the subset function with the select option:

```
justsexshoes<-subset(adf,select=c("student", "sex", "shoe"))
```

or

```
justsexshoes<-subset(adf,select=c(1:3))
```

The subset function uses "select" to choose particular columns and "subset" to select rows that have a particular value in one variable. Subset forces you to be very explicit in what you are doing, but you can do all the subsetting with creative use of `x2<-x1[row,col]` statements.

7. COMBINING DATA: three useful joiners -- cbind, rbind, merge

cbind: joins vectors side-by-side; rows (records) must be in the same order

rbind: stacks data vertically; must have the same number of columns in the same order

merge: flexible way to merge linked data sets side-by-side or top-to-bottom

cbind

If you have multiple vectors (or matrices, or data frames) *with all the records in the same order*, you can combine them into one with cbind.

```

person<-c(1,2,3,4,5,6,7,8)
shoe<-c(8,7,8.5,6,10.5,11,7,6)
hat<-c(7.5,7,7.25,6.75,8,7.75,7,7.25)
belt<-c(30,26,34,29,31,25,32,38)
a<-cbind(person,shoe,hat,belt)
a
      person shoe  hat belt
[1,]      1  8.0  7.50  30
[2,]      2  7.0  7.00  26
[3,]      3  8.5  7.25  34
[4,]      4  6.0  6.75  29
[5,]      5 10.5  8.00  31
[6,]      6 11.0  7.75  25
[7,]      7  7.0  7.00  32
[8,]      8  6.0  7.25  38

```

rbind

rbind stacks data of the same number of columns vertically, taking column names from the first data set.

Let's collect more data on shoes, hats, and belts

```

person<-c(11,12,13,14)
shoe<-c(4,6,9,10)
hat<-c(6,7,6.5,7.5)
belt<-c(22,25,30,29)
b<-cbind(person,shoe,hat,belt)
b
      person shoe hat belt
[1,]     11   4 6.0  22
[2,]     12   6 7.0  25
[3,]     13   9 6.5  30
[4,]     14  10 7.5  29

ab<-rbind(a,b)
#stack the new data "b" on top of "a"
ab
      person shoe  hat belt
[1,]      1  8.0  7.50  30
[2,]      2  7.0  7.00  26
[3,]      3  8.5  7.25  34
[4,]      4  6.0  6.75  29
[5,]      5 10.5  8.00  31
[6,]      6 11.0  7.75  25
[7,]      7  7.0  7.00  32
[8,]      8  6.0  7.25  38
[9,]     11  4.0  6.00  22
[10,]     12  6.0  7.00  25
[11,]     13  9.0  6.50  30
[12,]     14 10.0  7.50  29

```

!OjO! Be careful!

Note that for matrices, rbind doesn't pay attention to matching column names.

rbind will happily stack the following onto a, mixing belt and ring into one

consider data frame d

```
cat<-c(20,21,22,23); tail<-c(4,6,9,10);
```

```
whiskers<-c(6.0,7.0,6.5,7.5)
```

```
claws<-c(10,11,8,9);
```

```
d<-cbind(cat,tail,whiskers,claws); d
```

```

      cat tail whiskers claws
[1,]  20   4     6.0    10
[2,]  21   6     7.0    11
[3,]  22   9     6.5     8
[4,]  23  10     7.5     9

      person shoe hat belt
[1,]     11   4 6.0  22
[2,]     12   6 7.0  25
[3,]     13   9 6.5  30
[4,]     14  10 7.5  29
[5,]     20   4 6.0  10
[6,]     21   6 7.0  11
[7,]     22   9 6.5   8
[8,]     23  10 7.5   9

```

rbind for data frames

You can use `rbind` to stack data frames as well, but for only if all the column names are the same.

Let's convert `a`, `b`, and `d` into data frames

```
df_a<-as.data.frame(a)
df_b<-as.data.frame(b)
df_d<-as.data.frame(d)
```

You can easily rbind the two people data frames with identical columns

```
df_ab<-rbind(df_a,df_b)
```

```
df_ab
```

| | person | shoe | hat | belt |
|----|--------|------|------|------|
| 1 | 1 | 8.0 | 7.50 | 30 |
| 2 | 2 | 7.0 | 7.00 | 26 |
| 3 | 3 | 8.5 | 7.25 | 34 |
| 4 | 4 | 6.0 | 6.75 | 29 |
| 5 | 5 | 10.5 | 8.00 | 31 |
| 6 | 6 | 11.0 | 7.75 | 25 |
| 7 | 7 | 7.0 | 7.00 | 32 |
| 8 | 8 | 6.0 | 7.25 | 38 |
| 9 | 11 | 4.0 | 6.00 | 22 |
| 10 | 12 | 6.0 | 7.00 | 25 |
| 11 | 13 | 9.0 | 6.50 | 30 |
| 12 | 14 | 10.0 | 7.50 | 29 |

but you can't rbind people and cats, because the column names don't match

```
df_bd<-rbind(df_b,df_d)
```

```
Error in match.names(clabs, names(xi)) :
student do not match previous names
```

merge

Merge is a flexible way to merge data frames by columns or rows.

Let's start with two data frames, each with different kinds of data but from the same 10 subjects.

Create the data frames "shoes" and "moredata" with the following code

```
student<-c("biao","blair","catherine","cynthia","dave","jae","jenn")
shoe<-c(8,7,8.5,6,10.5,11,7)
sex<-c("m","f","f","f","m","m","f")
hat<-c(7.5,7,7.25,6.75,8,7.75,7)
belt<-c(30,26,34,29,31,25,32)
shoes<-data.frame(student,shoe)
shoes[,2]<-as.numeric(shoes[,2])
moredata<-data.frame(student,sex,hat,belt)
moredata[,3]<-as.numeric(moredata[,3])
moredata[,4]<-as.numeric(moredata[,4])
```

shoes

| | student | shoe |
|---|-----------|------|
| 1 | biao | 8.0 |
| 2 | blair | 7.0 |
| 3 | catherine | 8.5 |
| 4 | cynthia | 6.0 |
| 5 | dave | 10.5 |
| 6 | jae | 11.0 |
| 7 | jenn | 7.0 |

moredata

| | student | sex | hat | belt |
|---|-----------|-----|------|------|
| 1 | biao | m | 7.50 | 30 |
| 2 | blair | f | 7.00 | 26 |
| 3 | catherine | f | 7.25 | 34 |
| 4 | cynthia | f | 6.75 | 29 |
| 5 | dave | m | 8.00 | 31 |
| 6 | jae | m | 7.75 | 25 |
| 7 | jenn | f | 7.00 | 32 |

Merging rows that match in one attribute.

Merge shoe and moredata, so that the data from each student line up

```
mergedData<-merge(shoes,moredata,by="student")
```

mergedData

| | student | shoe | sex | hat | belt |
|---|-----------|------|-----|------|------|
| 1 | biao | 8.0 | m | 7.50 | 30 |
| 2 | blair | 7.0 | f | 7.00 | 26 |
| 3 | catherine | 8.5 | f | 7.25 | 34 |
| 4 | cynthia | 6.0 | f | 6.75 | 29 |
| 5 | dave | 10.5 | m | 8.00 | 31 |
| 6 | jae | 11.0 | m | 7.75 | 25 |
| 7 | jenn | 7.0 | f | 7.00 | 32 |

Notes on merge:

(1) In R, you don't have to sort the data before merging, like you do in SAS

(2) If the column that needs to match has a different name in the two files (e.g., if shoes had "student" and moredata had "name", you can specify the columns to match

```
mergedData2<-merge(shoes,moredata,by.x="student",by.y="name")
```

Merging by matching rows when the two data frames are unbalanced.

By default, merge only keeps those rows for which this matching value exists in BOTH files are included (e.g., if there was no shoe size for dave, then none of his data would appear in the merged file. You can force keeping all the data from one or both of the files.

#First, create a new shoe data frame, but without the record for dave

```
shoesNoDave<-shoes[student!='dave',]
```

```
shoesNoDave
  student shoe
1    biao  8.0
2    blair  7.0
3 catherine 8.5
4   cynthia 6.0
6      jae 11.0
7     jenn  7.0
```

#merge shoesNoDave and moredata, inserting "NA" for the missing dave shoe data

```
mergedData3<-merge(shoesNoDave,moredata, all=TRUE)
```

```
mergedData3
  student shoe sex  hat belt
1    biao  8.0  m  7.50  30
2    blair  7.0  f  7.00  26
3 catherine 8.5  f  7.25  34
4   cynthia 6.0  f  6.75  29
5      dave  NA  m  8.00  31
6      jae 11.0  m  7.75  25
7     jenn  7.0  f  7.00  32
#note that the NA in the shoe column
#will affect calculations on that column
mean(mergedData3$shoe)
[1] NA
#need to specify to remove the NAs first
mean(mergedData3$shoe,na.rm=TRUE)
[1] 7.916667
```

Similarly, you can specify to include all the records from the first file (all.x=TRUE) or all records from the second file (all.y=TRUE), as is most appropriate

```
mergedData4<-merge(shoesNoDave,moredata, all.x=TRUE,all.y=FALSE)
```

Why use merge? Merge is really useful for space saving if you have your data in relational databases, or even several related flat files. If you have, for instance, one table of species codes, species names, and taxonomic information, and another larger data frame with species codes and various measurements for many individuals, you can easily merge and output data with species names and measurements on the fly, without having to have very large data files that repeat the taxonomic information over and over.

8. Handling Dates in R

Importing dates and time from Excel into R requires a bit of care. If you just import a .csv file that has dates (e.g., 12-May-96 or 3/6/2001) into an R data frame, it is read as character strings and is automatically converted to a Factor.

The easiest approach (I think) is to format your dates in Excel using one of several standard formats (avoid using a simple number because of the 1900 vs 1904 start date issues).

Set formats in Excel, using Format:Cells:Date to choose one of these standard types:

03/15/05 15-Mar-05 15-Mar-2005 3/15/05 1:30 PM 3/15/05 13:30

Read your data in as usual using read.csv(). Given below is an example file on the website.

```
d<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/AsstDates.csv")
original<-d #make a copy of this so you can later compare start to finish
d #take a quick look at the data frame

str(d) #note that the dates are treated as factors.

#you need to do a double formatting now. First convert the date data from
#factors to characters, and then convert that to date format. You will need
#to specify exactly what the format of the date is, however.
#as.character() converts from factor to character
#as.POSIXct, with a specific format statement, convert characters to dates.

d$dates1<-as.POSIXct(as.character(d$dates1),format="%d-%b-%y") #25-Aug-96
d$dates2<-as.POSIXct(as.character(d$dates2),format="%m/%d/%y") #8/25/96
d$dates3<-as.POSIXct(as.character(d$dates3),format="%d-%b-%Y") #25-Aug-1996
d$dates4<-as.POSIXct(as.character(d$dates4),format="%m/%d/%y %H:%M") #8/25/96 19:25
d$dates5<-as.POSIXct(as.character(d$dates5),format="%m/%d/%y %I:%M %p") #8/25/96 3:15 AM

d #look at the data frame.
# R date format is YEAR-MONTH-DAY HOUR:MIN:SEC 2010-03-04 03:35:00
str(d) #note that they have now been converted to type dates

d$d4to5<-d$dates5-d$dates4 #create new column of the difference between date5 and date 4
original #show the original data frame
d #shows the final data frame including the difference in hours
d<-d[order(d$dates5),] #sort in order of dates 5
d #show the sorted data
#You can see all the codes for time and date formatting at help(strptime)
```

Here are the main codes. Enclose them in quotes, with - / or space between as appropriate.

| | | |
|--|--|--|
| %Y year with century | %I hours as decimal (01-12) | %A Full weekday name |
| %y year without century (00-68 given 20 and 69-99 given 19) | %p AM/PM indicator to use with %I | %w weekday as decimal (0-6, 0=Sunday) |
| %b Abbr. month name | %M minute as decimal (00-59) | |
| %B Full month name | %S seconds as decimal (00-60) | |
| %m month as decimal (01-12) | %X time as "%H:%M:%S" | |
| %d day of month (01-31) | | |
| %j day of year (001-366) | %U week of year as decimal (00-53, Sunday as 1 st day) | |
| %x date as "%y/%m/%d" | | |
| %H hours as decimal (00-23) | %a Abbr. weekday name | |