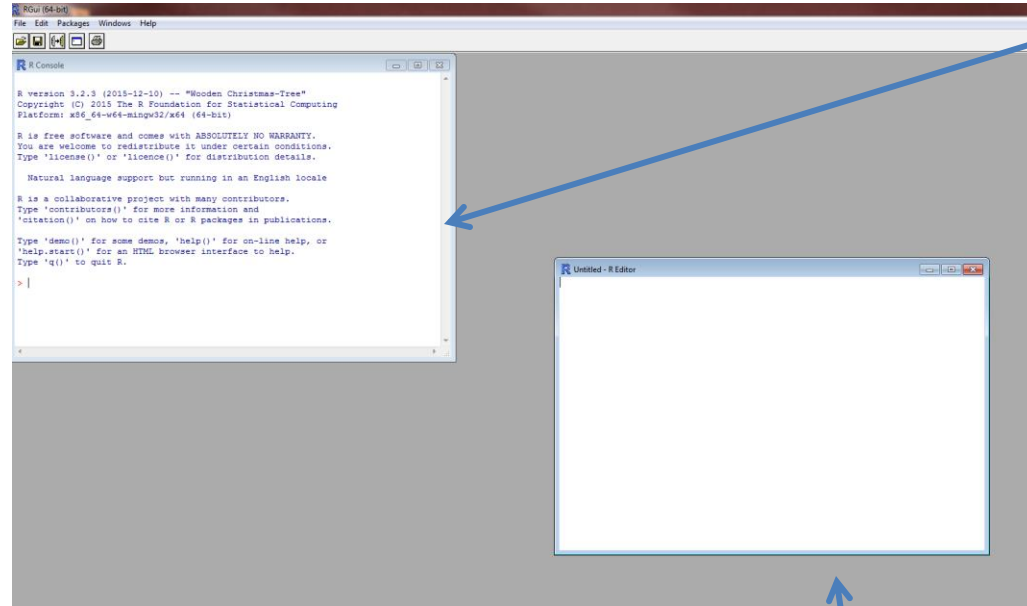


R

Intro to R

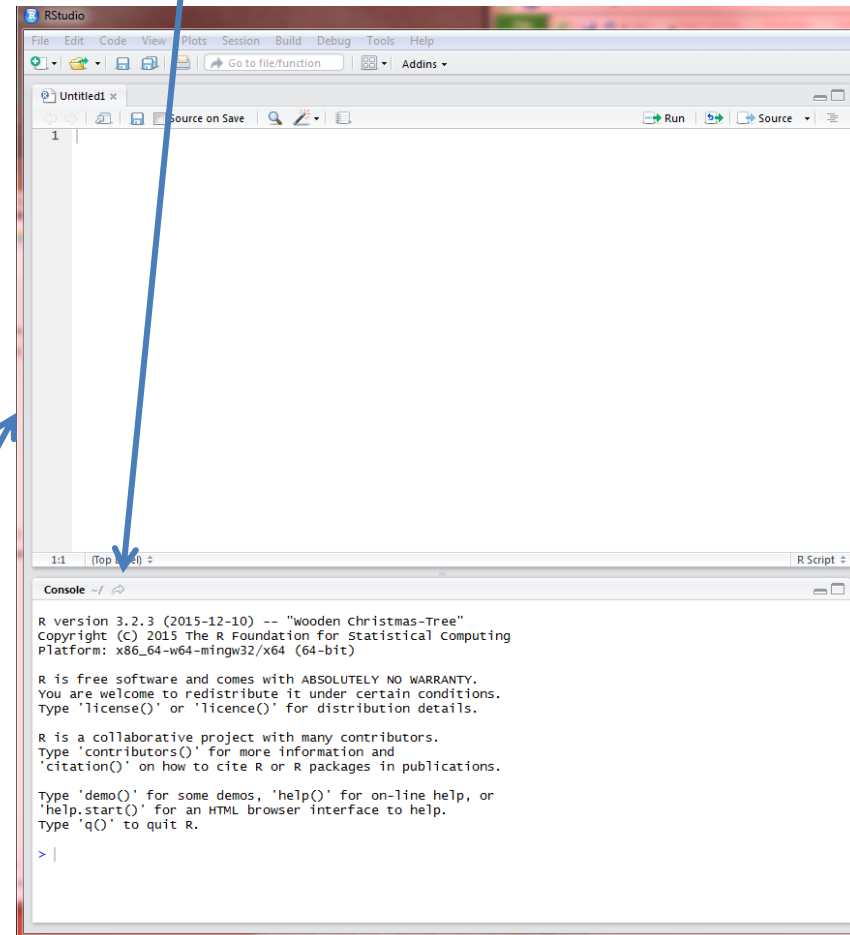
Console

R studio



There is a large number of applications for interfacing with R. Here is an example of two ways to work with R.

Blank script



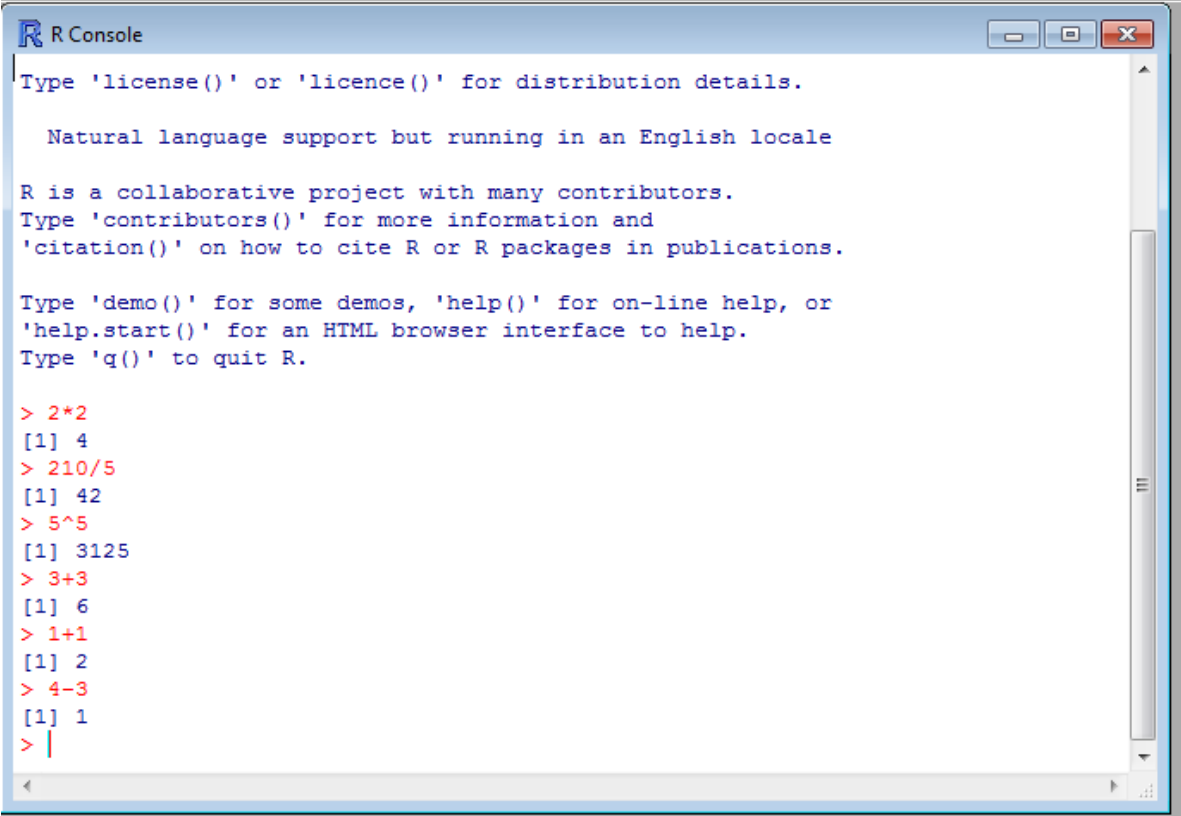
Using R

- The **console** is the main component of R. It is where you run all of your commands and functions.
- The **script** is where you write all of your commands to run in R. This file is saved (usually .R format) and a good script can be run at any time.

Working with numbers in R

- R acts like a calculator:

Red=inputs from your script
Blue= output from R



```
R Console
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

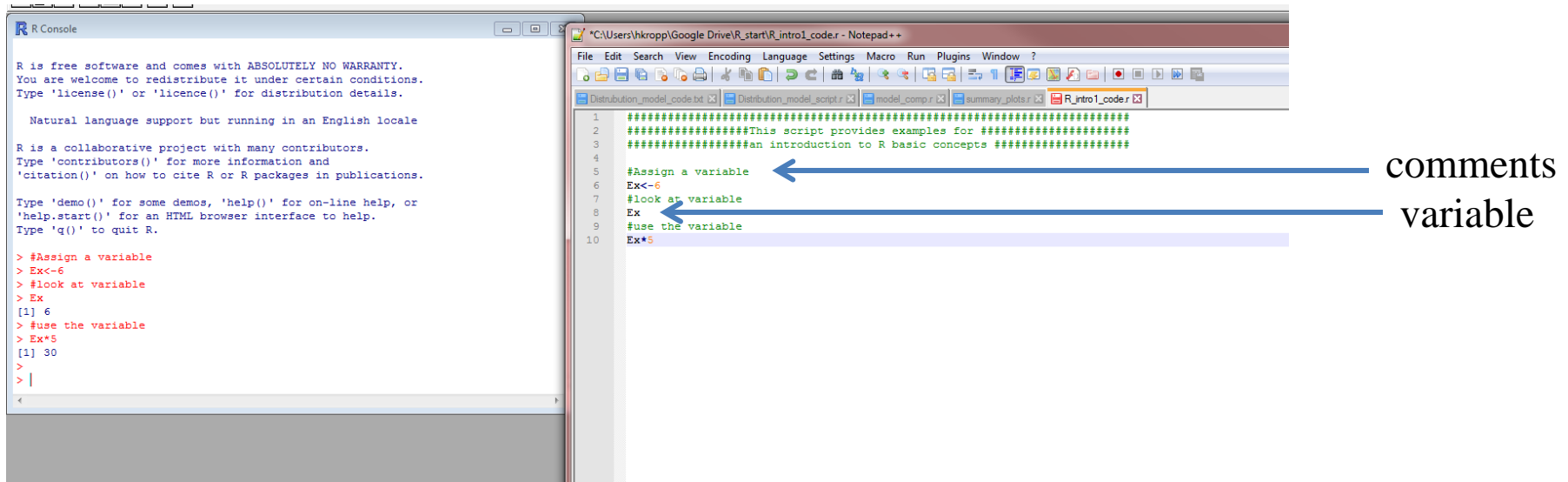
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 2*2
[1] 4
> 210/5
[1] 42
> 5^5
[1] 3125
> 3+3
[1] 6
> 1+1
[1] 2
> 4-3
[1] 1
> |
```

Assigning variables in R

- Give an item a name using the <- followed by the item
 - This allows you to refer back to items without having to remember them or write huge amounts of code
- Comment your code using # in front of the line
 - Commenting allows you to keep track of what you are doing and provide reminders for later



The image shows two windows side-by-side. The left window is the R Console, and the right window is Notepad++.

R Console:

```
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> #Assign a variable  
> Ex<-6  
> #look at variable  
> Ex  
[1] 6  
> #use the variable  
> Ex*5  
[1] 30  
> |  
>
```

Notepad++:

```
1 #####  
2 #####This script provides examples for #####  
3 #####an introduction to R basic concepts #####  
4  
5 #Assign a variable  
6 Ex<-6  
7 #look at variable  
8 Ex  
9 #use the variable  
10 Ex*5
```

Annotations on the right side of the Notepad++ window:

- A blue arrow points from the text "comments" to line 5: `#Assign a variable`.
- A blue arrow points from the text "variable" to line 6: `Ex<-6`.

Vectors and matrix in R

- R automatically treats inputs like they are vectors.
- Create a vector using `c()`

```
#set up a vector  
Vec<-c(3,4,5,2,10,4,6)
```

- Set up a matrix using `matrix()`

```
>  
> Vec<-c(3,4,5,2,10,4,6)  
> Mat<-matrix(c(1,2,3,4,5,6), ncol=2, byrow=TRUE)  
> Mat  
      [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
> #set up a matrix that fills in by columns  
> Mat.bycol<-matrix(c(1,2,3,4,5,6), ncol=2, byrow=FALSE)  
> Mat.bycol  
      [,1] [,2]  
[1,]    1    4  
[2,]    2    5  
[3,]    3    6  
> |
```

```
10 Ex*5  
11  
12 #set up a vector  
13 Vec<-c(3,4,5,2,10,4,6)  
14  
15 #set up a matrix  
16 Mat<-matrix(c(1,2,3,4,5,6), ncol=2, byrow=TRUE)  
17 Mat  
18 #set up a matrix that fills in by columns  
19 Mat.bycol<-matrix(c(1,2,3,4,5,6), ncol=2, byrow=FALSE)  
20 Mat.bycol  
21  
22  
23
```

Functions in R

- Using `matrix()` is an example of a function in R
- There are a lot of “built in” functions in R that make it easier to work with data or statistics
- For example, calculating an average using `mean()`:

```
> mean(Vec)  
[1] 4.857143  
> |
```

```
21  
22 #calculate the average value of Vec  
23 mean(Vec)  
24
```

Functions in R

- A **function** typically gives you an **output** based on the inputs you give it
- The inputs needed for a function are often called **arguments**
- R will have a description of the arguments and output for its function
 - google the *function.name*
 - type `help(function.name)` in the console

mean {base}

function.name

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

x

An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.

trim

the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

na.rm

a logical value indicating whether NA values should be stripped before the computation proceeds.

...

further arguments passed to or from other methods.

Way that the function gets used and the arguments that have some default assumptions

Detailed description

- Here *x*, *trim*, and *na.rm* are the names of arguments
- The order here is important because if the names of the arguments aren't used, then they are assumed to be in this default order.
- If the default arguments are sufficient there is no need to include them in the function
- If you don't want to use the default order than you can specify the order using the argument names:

```
[1] 4.857143  
> mean(na.rm=TRUE, x=Vec)  
[1] 4.857143
```

```
24 #specify arguments by name and not order  
25 mean(na.rm=TRUE, x=Vec)
```


Working with vectors and matrices:

- Keep in mind that R automatically does vector/matrix math:

```
[3,] 15 18
> #multiply vector by 5
> Vec*5
[1] 15 20 25 10 50 20 30
> |
```

```
30
31 #multiply vector by 5
32 Vec*5
```

```
> #set multiply matrix by vector
> Mat.scale<-matrix(c(2,2,2,3,3,3), ncol=2,byrow=TRUE)
> Mat*Mat.scale
  [,1] [,2]
[1,]  2  4
[2,]  6 12
[3,] 15 18
> |
```

```
31 # multiply matrix by vector
32 Mat.scale<-matrix(c(2,2,2,3,3,3), ncol=2,byrow=TRUE)
33 Mat*Mat.scale
```

Packages in R

- People have created thousands of **packages** to add more functions to R
- Packages allow you to download only the ones you want to use (it would take up a lot of space)
- Some functions may have the same name in different packages so be sure to note potential overlap in packages
 - only load the ones you are going to use

Data Types

- Numeric: number can have any number of decimals
- Character: text
- Factor: text but a short identifying category name

```
> N.ex
[1] 0.9333333
> C.ex<-c("December")
> C.ex
[1] "December"
> F.ex<-as.factor("a")
> F.ex<-as.factor("a")
> F.ex
[1] a
Levels: a
```

```
33
34 #data types
35 #numeric
36 N.ex<-14/15
37 N.ex
38 #character
39 C.ex<-c("December")
40 C.ex
41 #Factor
42 F.ex<-as.factor("a")
43 F.ex
```

Reading in Data

- The easiest and most consistent way to read in data in R is through a comma separated text file (.csv)
- You need to tell R where to find the data
 - Set a working directory to always get files from one folder
 - Or specify the file path with the csv name
- File/File paths always need to be in quotes and file paths always have \\ between folders

```

[1] C:/Users/hkropp/Google Drive/data_to_do/Example_dataR
> #set working directory
> setwd("c:\\Users\\hkropp\\Google Drive\\data_to_do\\Example_dataR")
> #check what your working directory is set at
> getwd()
[1] "c:/Users/hkropp/Google Drive/data_to_do/Example_dataR"
> |

```

```

44
45 #set working directory
46 setwd("c:\\Users\\hkropp\\Google Drive\\data_to_do\\Example_dataR")
47 #check what your working directory is set at
48 getwd()

```

R programming language

length : 1050 lines : 48

```

#read in data file
datM<-read.csv("mountain_data.csv")
#check out data
datM

```

- Always think about your names
 - Length
 - Clarity
- Capitalization matters!

```

> datM

```

	Rank	Name	Region	Elev.m	Prom.m	Elev.ft	Prom.ft
1	1	Mt Everest	Nepal Tibet	8848	8848	29028	29028
2	2	Aconcagua	Argentina	6962	6962	22841	22841
3	3	Mt McKinley Denali	US	6194	6138	20320	20138
4	4	Kilimanjaro	Tanzania	5895	5885	19340	19308
5	5	Cristobal Colon	Colombia	5700	5509	18701	18074
6	6	Mt Logan	Canada	5959	5250	19550	17224
7	7	Pico de Orizaba Citlaltepetl	Mexico	5636	4922	18491	16148
8	8	Vinson Massif	Antarctica	4892	4892	16050	16050
9	9	Puncak Jaya	Indonesia	4884	4884	16023	16023
10	10	Gora Elbrus	Russia	5642	4741	18510	15554
11	11	Mont Blanc	France Italy	4809	4696	15777	15406
12	12	Damavand	Iran	5610	4667	18405	15311
13	13	Klyuchevskaya Volcano	Russia	4750	4649	15584	15252
14	14	Nanga Parbat	Pakistan	8125	4608	26657	15118
15	15	Mauna Kea	US	4205	4205	13796	13796
16	16	Jengish Chokusu ex Pik Pobedy	Kyrgyzstan China	7439	4148	24406	13609
17	17	Chimborazo	Ecuador	6267	4122	20561	13523
18	18	Bogda Shan	China	5445	4122	17864	13523
19	19	Namcha Barwa	China	7782	4106	25531	13471
20	20	Kinabalu	Malaysia	4095	4095	13435	13435
21	21	Mt Rainier	US	4393	4023	14411	13196
22	22	K2	Pakistan China	8611	4017	28251	13179
23	23	Ras Dejen	Ethiopia	4533	3980	15092	13090
24	24	Volcan Tajumulco	Guatemala	4220	3980	13845	13058
25	25	Pico Bolivar	Venezuela	4981	3957	16341	12982

```

> |

```

Properties of a data frame

- Typically the columns have names
- All columns are the same length
- There can be different types of data in each column

Basic Info about a data frame

- Get dimensions

```
> dim(datM)
[1] 25  7
> Mdim<-dim(datM)
> |
```

```
55 #get dimensions of the dataset
56 dim(datM)
57 #Note output is a vector of 2 values
58 #we can name this and refer to later
59 Mdim<-dim(datM)
```

- Names of columns

```
> names(datM)
[1] "Rank"      "Name"      "Region"    "Elev.m"    "Prom.m"    "Elev.ft"   "Prom.ft"
> |
```

```
60
61 #get the column names
62 names(datM)
63
64
65
```

- See what it looks like

```
> head(datM)
  Rank      Name      Region Elev.m Prom.m Elev.ft Prom.ft
1    1  Mt Everest Nepal Tibet  8848   8848   29028   29028
2    2  Aconcagua  Argentina  6962   6962   22841   22841
3    3 Mt McKinley Denali      US  6194   6138   20320   20138
4    4  Kilimanjaro Tanzania  5895   5885   19340   19308
5    5 Cristobal Colon Colombia  5700   5509   18701   18074
6    6    Mt Logan    Canada  5959   5250   19550   17224
> |
```

```
63
64 #look at the names and first 5 rows
65 head(datM)
66
67
68
69
70
71
72
73
74
75
```

Referring to data in data frames

- A column can be used by: *data.frame\$column*

```
#look at only the name columnne  
datM$Name
```

- Data frames can also be refered to like matrix where [rows,columns] notation is used
 - Refer to a column without calling its name:

```
#look at name in second column  
datM[,2]
```

- Multiple columns

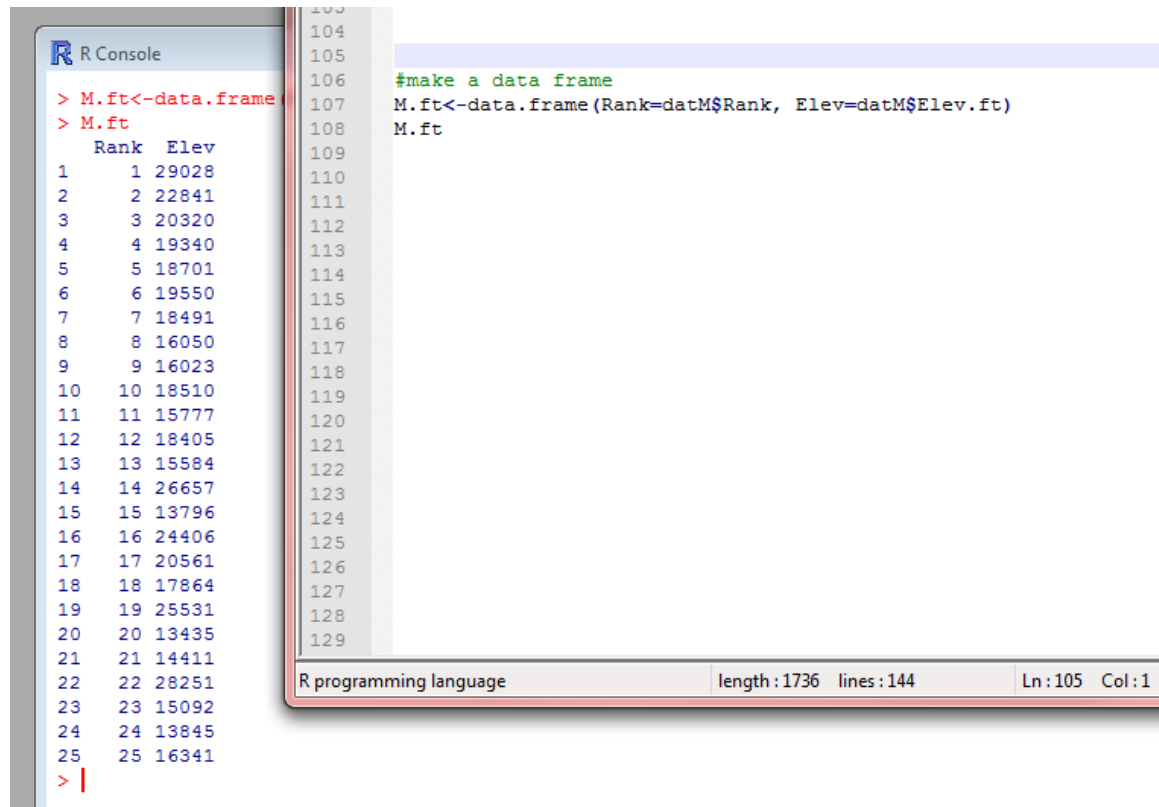
```
#refer to multiple columns  
datM[,2:4]
```

- Rows:

```
#refer to several rows  
datM[1:3,]
```


Creating a data frame

- Use the function `data.frame`
 - Note: all vectors must be equal lengths



The screenshot shows an R console window with two panes. The left pane displays the output of the `data.frame` function, showing a table with two columns: Rank and Elev. The right pane shows the R code used to create the data frame.

```
R Console
```

```
> M.ft<-data.frame
> M.ft
  Rank Elev
1    1 29028
2    2 22841
3    3 20320
4    4 19340
5    5 18701
6    6 19550
7    7 18491
8    8 16050
9    9 16023
10   10 18510
11   11 15777
12   12 18405
13   13 15584
14   14 26657
15   15 13796
16   16 24406
17   17 20561
18   18 17864
19   19 25531
20   20 13435
21   21 14411
22   22 28251
23   23 15092
24   24 13845
25   25 16341
> |
```

```
#make a data frame
M.ft<-data.frame(Rank=datM$Rank, Elev=datM$Elev.ft)
M.ft
```

R programming language length:1736 lines:144 Ln:105 Col:1

Subset data

- Data can be subset by a characteristic
- This is done using logical expressions (see R's guide for logical expressions.
 - <https://www.r-bloggers.com/logical-operators-in-r/>
- Subset with brackets:
 - Mountains in the US

```
> US.M
  Rank      Name Region Elev.m Prom.m Elev.ft Prom.ft
3     3 Mt McKinley Denali    US   6194   6138  20320  20138
15    15      Mauna Kea     US   4205   4205  13796  13796
21    21      Mt Rainier    US   4393   4023  14411  13196
> |
```

```
83 M.ft
84
85 #subset all of the tallest mountains in the US
86 US.M<-datM[datM$Region=="US",]
87
88
89
```

– Mountains above 20000 ft

```
> High.M<-datM[datM$Elev.ft>20000,]
> High.M
  Rank      Name      Region Elev.m Prom.m Elev.ft Prom.ft
1     1      Mt Everest  Nepal Tibet   8848   8848  29028  29028
2     2      Aconcagua  Argentina  6962   6962  22841  22841
3     3      Mt McKinley Denali    US   6194   6138  20320  20138
14    14      Nanga Parbat  Pakistan  8125   4608  26657  15118
16    16 Jengish Chokusu ex Pik Pobedy Kyrgyzstan China  7439   4148  24406  13609
17    17      Chimborazo  Ecuador   6267   4122  20561  13523
19    19      Namcha Barwa    China   7782   4106  25531  13471
22    22      K2          Pakistan China   8611   4017  28251  13179
~ |
```

```
92
93
94
95 #subset by mountains above 20,000 ft
96 High.M<-datM[datM$Elev.ft>20000,]
97 High.M
98
99
100
101
102
103
```

Missing data

- NA indicates that the data is missing in R
- If there are blank cells in a data file R will automatically fill them in with NA
- You can also designate what marks an NA if it differs in a data file:

Errors

- Error messages look intimidating at first in R, but they are actually very useful
- Some kinds of examples:
 - Trying to do something where vectors are different lengths

```
> High.M$Elev.ft-US.M$Prom.ft
[1] 8890 9045 7124 6519 10610 7365 5393 14455
Warning message:
In High.M$Elev.ft - US.M$Prom.ft :
  longer object length is not a multiple of shorter object length
> |
```

```
93
94 #look at difference between prominence and elevation
95 High.M$Elev.ft-US.M$Prom.ft
96
97
98
99
```

- Referring to names incorrectly (capitalization counts!)

```
> mean(High.M$elev.ft)
[1] NA
Warning message:
In mean.default(High.M$elev.ft) :
  argument is not numeric or logical: returning NA
> |
```