R packages

- Currently, the CRAN package repository features 12,591 available packages (6 June 2018).
- To install packages in RStudio, click on the **Packages** tab in the lower right, then:
  1. `Install`
  2. `Install from: Repository (CRAN)`
  3. Type the name of the package in `Packages`
  4. Click **Install**
- Or, you can type `install.packages("`*package name*`")`, e.g. `install.packages("plotrix")`.
- After the installation, use `library("`*package name*`")` to load it into **R**.

Reading other data files into **R**

- Base **R** only includes functions which read data sets saved in simple file formats, e.g. `csv`, `txt`, `tsv`, etc., but what if your data was saved in another format, e.g. STATA, SPSS or SAS spreadsheets?
- The https://cran.r-project.org/web/packages/haven/index.html contains functions that may help:

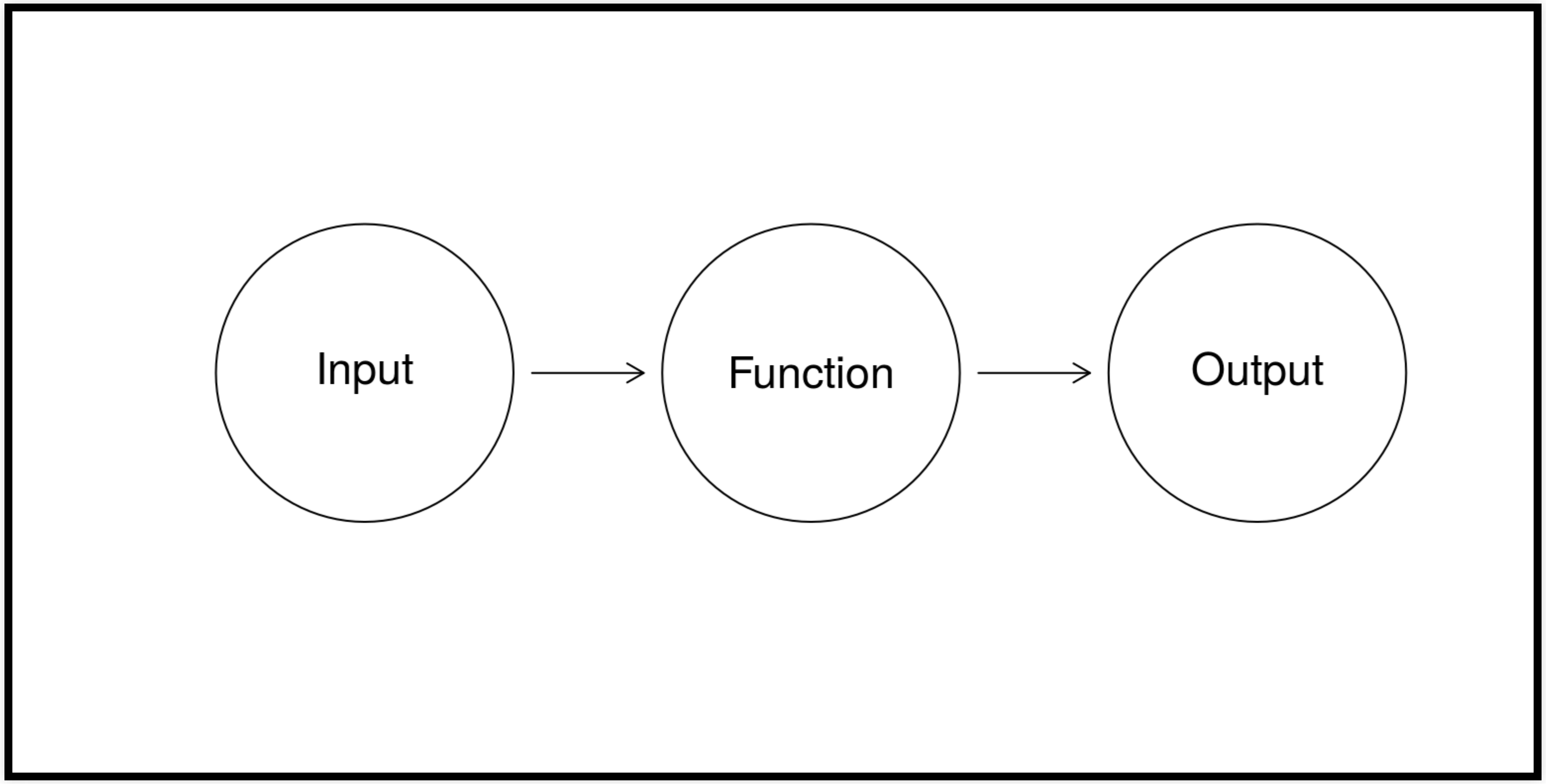
```r
library(haven)
stata.df = read_dta("data.dta")
spss.df = read_sav("data.sav")
sas.df = read_sas("data.sas7bdat")
sasxport.df = read_xpt("data.xpt")
```

However, it is always easiest to convert the data into a `csv` file.

Functions

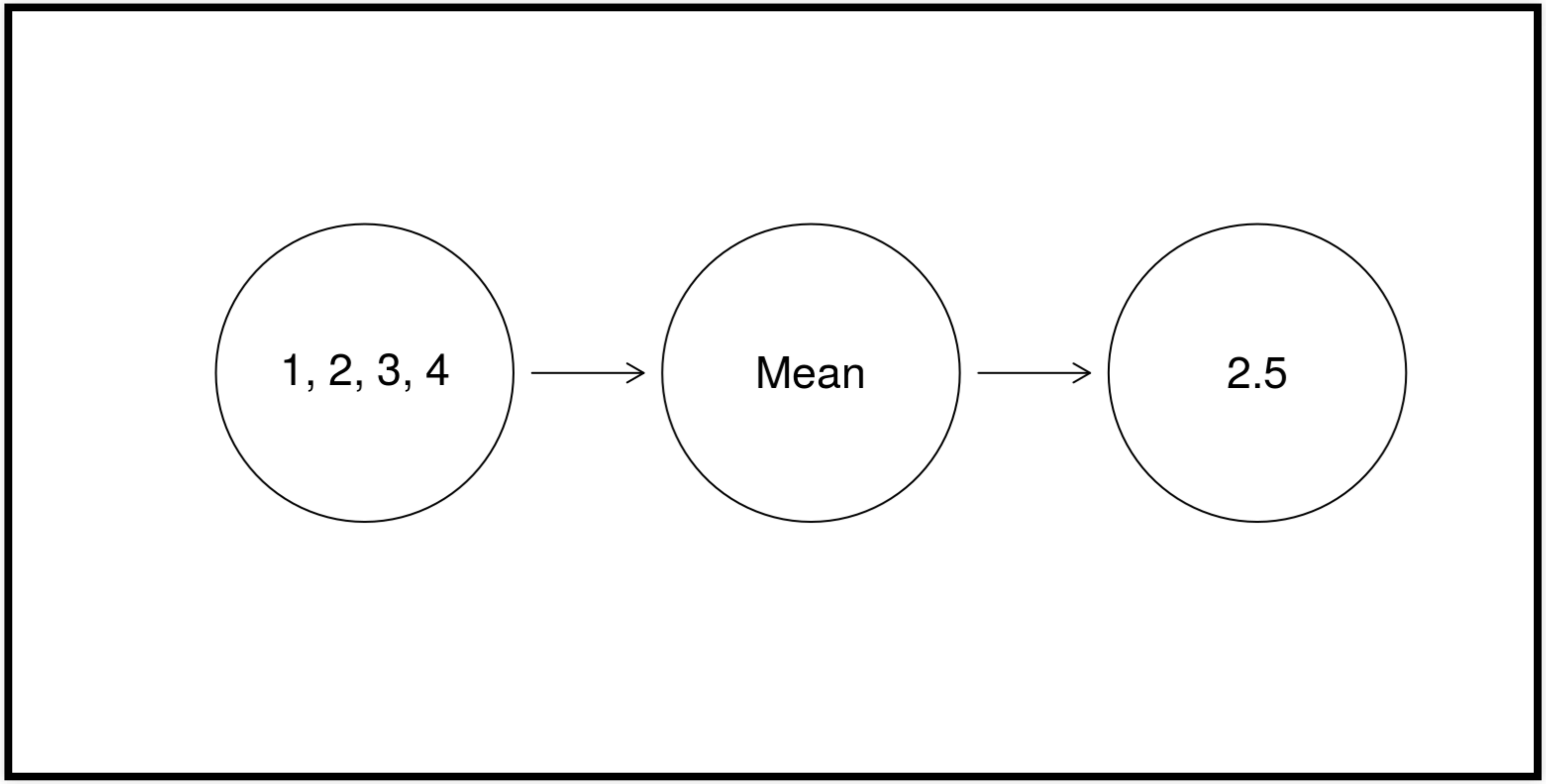
Functions

A function is a relationship between a set of inputs (arguments) and a set of outputs. The function is fed some information on which it operates, and the results are the output.



Functions

We have already seen some functions, e.g. `sd()`, `mean()`, `table()`, `with()`, etc.



Working with functions

- Functions can be user-defined (you can write your own).
- The output is the last line of the function.

Here is a function that calculates the standard error of the mean:

$$\widehat{\text{SEM}} = \frac{s}{\sqrt{N}}$$

```r
sem_function = function(input){
  s = sd(input, na.rm = TRUE) # Calc std. deviation
  N = length(input)           # Calc sample size
  s / sqrt(N)                 # Definition of SEM
}

sem_function(patient.df$Weight)
```

```
#R:   [1] 0.3032053
```

- A set of user-defined functions can be bundled together into an **R** package.

Subsetting

Square brackets

Square brackets `[` and `]` are used to extract subsets of data.

```r
# Extract the first patient's height
patient.df$Height[1]
```

```
#R:   [1] 70.4
```

```r
# Extract the first 5 patients' heights
patient.df$Height[1:5]
```

```
#R:   [1] 70.4 63.9 61.8 69.8   NA
```

```r
# Extract all but the first element in a vector
c("first", "second", "third")[-1]
```

```
#R:   [1] "second" "third"
```

Square brackets

Square brackets `[` and `]` are used to extract subsets of data.

```
# Extract the heights of patients 3 through to 8
patient.df$Height[3:8]
```

```
#R:   [1] 61.8 69.8   NA 70.2 62.6 64.4
```

```
# Extract the heights of the third and eighth patients
patient.df$Height[c(3, 8)]
```

```
#R:   [1] 61.8 64.4
```

Subsetting rows

Subsetting two-dimensional arrays, such as data frames, requires the use of *two* indices:

```
[rows, cols]
```

```
# Extract the first row
patient.df[1, ]
```

```
#R:     Patient.ID Age Gender Ethnicity Weight Height Smoke
#R:   1          3  21   Male         1  179.5   70.4    NA
```

Subsetting columns

Subsetting two-dimensional arrays, such as data frames, requires the use of *two* indices:

```
[rows, cols]
```

```
# Second column or variable (Age)
patient.df[, 2]
```

```
#R:      [1] 21 32 48 35 48 44 42 24 67 56 82 44 50 36 48 32 66 70 63 37
#R:     [23] 58 80 80 23 83 28 90 86 27 72 34 21 45 84 36 28 69 63 31 25
#R:     [45] 39 55 72 32 27 55 78 65 57 26 31 31 49 61 65 42 48 80 22 43
#R:     [67] 59 43 22 73 66 69 79 85 48 32 73 32 28 62 38 56 48 22 21 66
#R:     [89] 55 52 51 66 33 54 24 83 31 42 24 47 35 21 59 64 41 29 63 33
#R:    [111] 76 31 20 77 27 40 30 78 43 24 80 66 48 66 89 42 47 46 22 73
#R:    [133] 42 81 77 65 24 21 36 73 81 20 39 38 61 30 76 58 81 67 26 53
#R:    [155] 58 85 35 30 85 72 61 38 29 26 45 34 61 70 22 32 32 43 69 27
#R:    [177] 68 25 87 30 68 61 81 70 32 42 48 47 40 32 75 36 23 32 23 49
#R:    [199] 24 20 35 58 49 26 34 50 73 72 23 67 78 54 40 65 86 27 90 42
#R:    [221] 36 84 38 21 61 62 21 59 48 57 64 23 38 42 46 50 62 31 28 40
#R:    [243] 47 58 79 39 57 29 68 58 56 69 83 35 34 63 83 24 22 30 64 41
#R:    [265] 49 61 65 54 31 63 34 56 40 48 20 85 68 32 45 32 62 41 55 85
#R:    [287] 22 73 22 40 25 23 68 37 69 71 44 45 52 60 87 60 79 40 45 77
#R:    [309] 57 33 63 36 51 72 27 45 33 29 34 58 57 49 30 40 51 80 59 77
#R:    [331] 39 47 50 25 35 32 88 45 72 58 61 57 29 63 41 24 28 57 43 58
```

```
#R:     [353] 62 50 73 22 58 45 44 28 48 58 39 55 71 26 21 50 86 58 35 67
```

Subsetting columns

Subsetting two-dimensional arrays, such as data frames, requires the use of *two* indices:

```
[rows, cols]
```

```
# Second column or variable (Age)
patient.df[, "Age"]
```

```
#R:      [1] 21 32 48 35 48 44 42 24 67 56 82 44 50 36 48 32 66 70 63 37
#R:     [23] 58 80 80 23 83 28 90 86 27 72 34 21 45 84 36 28 69 63 31 25
#R:     [45] 39 55 72 32 27 55 78 65 57 26 31 31 49 61 65 42 48 80 22 43
#R:     [67] 59 43 22 73 66 69 79 85 48 32 73 32 28 62 38 56 48 22 21 66
#R:     [89] 55 52 51 66 33 54 24 83 31 42 24 47 35 21 59 64 41 29 63 33
#R:    [111] 76 31 20 77 27 40 30 78 43 24 80 66 48 66 89 42 47 46 22 73
#R:    [133] 42 81 77 65 24 21 36 73 81 20 39 38 61 30 76 58 81 67 26 53
#R:    [155] 58 85 35 30 85 72 61 38 29 26 45 34 61 70 22 32 32 43 69 27
#R:    [177] 68 25 87 30 68 61 81 70 32 42 48 47 40 32 75 36 23 32 23 49
#R:    [199] 24 20 35 58 49 26 34 50 73 72 23 67 78 54 40 65 86 27 90 42
#R:    [221] 36 84 38 21 61 62 21 59 48 57 64 23 38 42 46 50 62 31 28 40
#R:    [243] 47 58 79 39 57 29 68 58 56 69 83 35 34 63 83 24 22 30 64 41
#R:    [265] 49 61 65 54 31 63 34 56 40 48 20 85 68 32 45 32 62 41 55 85
#R:    [287] 22 73 22 40 25 23 68 37 69 71 44 45 52 60 87 60 79 40 45 77
#R:    [309] 57 33 63 36 51 72 27 45 33 29 34 58 57 49 30 40 51 80 59 77
#R:    [331] 39 47 50 25 35 32 88 45 72 58 61 57 29 63 41 24 28 57 43 58
```

```
#R:     [353] 62 50 73 22 58 45 44 28 48 58 39 55 71 26 21 50 86 58 35 67
```

Subsetting both rows and columns

Subsetting two-dimensional arrays, such as data frames, requires the use of *two* indices:

```
[rows, cols]
```

```
# Subset patient.df to only include the first 5 patients' data on
# the 4th, 5th, 6th, and 7th variables (Ethnicity, Weight, Height, Smoke,
patient.df[1:5, 4:7]
```

```
#R:    Ethnicity Weight Height Smoke
#R:  1         1  179.5   70.4    NA
#R:  2         1     NA   63.9    NA
#R:  3         1  149.7   61.8     2
#R:  4         1  203.5   69.8    NA
#R:  5         1  155.3     NA     2
```

Subsetting both rows and columns

Subsetting two-dimensional arrays, such as data frames, requires the use of *two* indices:

```
[rows, cols]
```

```
# Subset patient.df to only include the first 5 patients' data on
# the 4th, 5th, 6th, and 7th variables (Ethnicity, Weight, Height, Smoke,
patient.df[1:5, c("Ethnicity", "Weight", "Height", "Smoke")]
```

```
#R:    Ethnicity Weight Height Smoke
#R:  1         1  179.5   70.4    NA
#R:  2         1     NA   63.9    NA
#R:  3         1  149.7   61.8     2
#R:  4         1  203.5   69.8    NA
#R:  5         1  155.3     NA     2
```

Which individuals smoke?

Remember that in our `patient.df` data set, an individual who smoked had a value of `1` for the variable `Smoke`.

Let’s use **R**’s powerful subsetting capabilities to select those cases for which the value of Smoke is equal to 1.

```r
smokers <- which(patient.df$Smoke == 1)
smokers
```

```
#R:     [1]   6   7   8  12  15  16  18  19  20  21
#R:    [12]  23  28  33  35  38  45  47  48  52  53
#R:    [23]  56  57  58  64  70  75  79  80  81  89
#R:    [34]  91  93  99 108 109 110 116 118 119 122
#R:    [45] 124 127 136 137 144 146 148 154 155 162
#R:    [56] 174 180 182 187 188 189 191 192 195 198
#R:    [67] 205 210 213 221 225 229 230 232 233 234
#R:    [78] 239 246 248 251 255 256 258 259 265 269
#R:    [89] 279 280 290 291 293 300 302 316 317 318
#R:   [100] 322 332 337 341 345 346 352 354 361 362
#R:   [111] 367 368 378 379 385 386 390 393 395 397
#R:   [122] 408 411 414 416 438 440 445 448 450 453
#R:   [133] 459 461 463 469 471 474 476 477 492 498
```

```
#R:   [144]   506   508   510   518   519   527   528   532   533   534

#R:   [155]   538   539   542   546   553   555   565   566   572   573
#R:   [166]   578   579   581   585   587   589   608   609   614   615
#R:   [177]   620   621   623   625   631   635   636   638   639   644
```

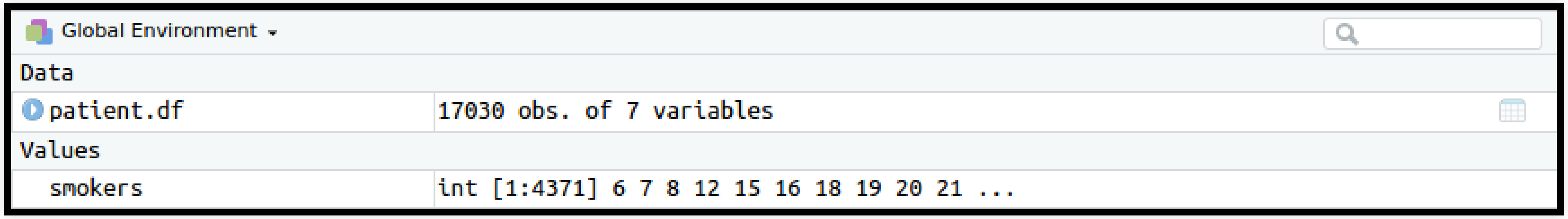
But how many smokers are there?

We can use the `length()` function to see how many smokers there are.

```r
length(smokers)
```

```
#R:   [1] 4371
```

We can also easily see this in RStudio in the Environment panel:



```
Data
patient.df     17030 obs. of 7 variables
Values
  smokers      int [1:4371] 6 7 8 12 15 16 18 19 20 21 ...
```

But how many smokers are there?

- `patient.df$Smoke == 1` gives a vector of `TRUE` or `FALSE` (or `NA`) for every observation.
- `which()` gives a vector of the position of the `TRUE`s .
- `length()` tells us how many elements there are in the vector.

Another way is to sum up how many `TRUE`s there are in the logical vector `patient.df$Smoke == 1`.

```
sum(patient.df$Smoke == 1, na.rm = TRUE)
```

```
#R:   [1] 4371
```

Who are the smokers?

The `smokers` vector contains the row numbers of all the smokers in our data set. We can use this as an index, to subset the patient ID's to only include those who are smokers.

```r
# Use square brackets to extract IDs corresponding to the row numbers
# contained in 'smokers'
patient.df$Patient.ID[smokers]
```

```
#R:     [1]   19   34   44   51   55   56   67   70   71   72
#R:    [12]   74   90  110  115  120  129  139  140  154  155
#R:    [23]  161  164  165  180  190  210  217  218  219  242
#R:    [34]  251  255  275  297  298  303  317  331  335  343
#R:    [45]  346  353  376  377  395  400  403  427  429  445
#R:    [56]  474  485  488  503  507  509  519  520  524  528
#R:    [67]  539  550  554  570  578  586  588  592  593  595
#R:    [78]  606  618  620  639  644  646  650  657  673  681
#R:    [89]  710  711  730  732  735  745  749  784  786  788
#R:   [100]  798  816  833  863  868  869  883  892  908  912
#R:   [111]  921  922  957  958  971  972  982  989  993  997
#R:   [122] 1015 1020 1026 1030 1096 1104 1122 1127 1129 1136
#R:   [133] 1155 1160 1163 1172 1181 1188 1196 1200 1242 1256
#R:   [144] 1273 1278 1300 1316 1317 1339 1342 1349 1351 1353
#R:   [155] 1362 1366 1373 1381 1395 1398 1412 1414 1425 1429
```

```
#R:   [166]  1445  1447  1452  1466  1468  1471  1524  1531  1547  1551
#R:   [177]  1567  1569  1573  1579  1592  1603  1604  1611  1613  1619
```

Missing values

Missing values

- **R** reserves the object `NA` for elements of a vector that are missing.
- We can use `is.na()` to return `TRUE` for each missing value.
- Use of `is.na()` to search for missing values requires that they are recorded as `NA`.
- `na` will not do because **R** is case sensitive!

```
# A vector with one missing value
c(1, 2, NA, 4, 5)
```

```
#R:   [1]  1  2 NA  4  5
```

```
# Remember R is case sensitive!
c(1, 2, na, 4, 5)
```

```
#R:  Error in eval(expr, envir, enclos): object 'na' not found
```

Missing values in Smoke

```
patient.df$Smoke[1:5]
```

```
#R:   [1] NA NA  2 NA  2
```

```
is.na(patient.df$Smoke[1:5])
```

```
#R:   [1]  TRUE  TRUE FALSE  TRUE FALSE
```

```
sum(is.na(patient.df$Smoke[1:5]))
```

```
#R:   [1] 3
```

```
# How many missing values in Smoke?
sum(is.na(patient.df$Smoke))
```

```
#R:   [1] 8404
```

Missing values in tables

The default option of `table()` ignores `NA`s when constructing frequency tables.

```r
table(patient.df$Smoke)
```

```
#R: 
#R:     1    2 
#R:  4371 4255 
```

Missing values in tables

If you still want to see how many `NA`s in the frequency tables, you can change the `useNA` argument to `"always"` in `table()`.

```
table(patient.df$Smoke, useNA = "always")
```

```
#R:
#R:      1    2 <NA>
#R:   4371 4255 8404
```

Data cleaning

The `ifelse()` function

```
IF (test is TRUE){
  THEN yes
  ELSE no
}
```

- `test`: a logical test.
- `yes`, what to return if the test is TRUE.
- `no`, what to return if the test is FALSE.

```r
test = c(TRUE, FALSE)

ifelse(test, "Yes", "No")
```

```
#R:  [1] "Yes" "No"
```

Note, the `ifelse()` in **R** has the same functionality as `IF()` in Excel.

Cleaning up the `Smoke` variable

Notice how we still need to remember if 1 or 2 denotes a smoker or non-smoker.

The `ifelse()` function provides a quick way to convert the `Smoke` variable in `patient.df` from numbers to words.

```
patient.df$Smoke = ifelse(patient.df$Smoke == 1, "Yes", "No")

table(patient.df$Smoke)
```

```
#R:
#R:     No  Yes
#R:   4255 4371
```

Cleaning up the `Ethnicity` variable

Remember that the `Ethnicity` variable has 3 levels (Caucasian, African, Other). We can nest an `ifelse()` statement inside another `ifelse()` statement!

```r
patient.df$Ethnicity = with(patient.df,
                            ifelse(Ethnicity == 1, "Caucasian",
                                   ifelse(Ethnicity == 2, "African", "Other")))

table(patient.df$Ethnicity)
```

```
#R:
#R:    African Caucasian     Other
#R:       4860     11612       553
```

Our clean data set

```r
head(patient.df)
```

```
#R:    Patient.ID Age Gender Ethnicity Weight Height Smoke
#R:  1          3  21   Male Caucasian  179.5   70.4  <NA>
#R:  2          4  32 Female Caucasian     NA   63.9  <NA>
#R:  3          9  48 Female Caucasian  149.7   61.8    No
#R:  4         10  35   Male Caucasian  203.5   69.8  <NA>
#R:  5         11  48   Male Caucasian  155.3     NA    No
#R:  6         19  44   Male   African  189.6   70.2   Yes
```

Creating new variables

A BMI variable

We will need to include a BMI variable for our analyses.

First we will need to calculate BMI:

$$\mathtt{BMI} = \frac{\mathtt{Weight(kg)}}{\mathtt{Height(m)}^2} = \frac{\mathtt{Weight(pounds)}}{\mathtt{Height(inches)}^2} \times 703$$

Then we will need to create a new variable `BMI_cat` for `normal`, `overweight` and `obese`, where:

- `normal` is associated with a BMI less than 25.
- `overweight` is a BMI greater than or equal to 25.
- `obese` is a BMI greater than or equal to 30.

Calculating BMI

$$\mathtt{BMI} = \frac{\mathtt{Weight(kg)}}{\mathtt{Height(m)}^2} = \frac{\mathtt{Weight(pounds)}}{\mathtt{Height(inches)}^2} \times 703$$

```r
# Create a BMI variable
patient.df$BMI = with(patient.df, Weight/Height^2 * 703)

# Average BMI
mean(patient.df$BMI, na.rm = TRUE)
```

```
#R:   [1] 27.03084
```

Categorizing BMI

```
Normal < 25 >= Overweight < 30 >= Obese
```

Since there are more than 2 possible outcomes, we will need to use nested `ifelse()` statements.

```
patient.df$bmi_cat = with(patient.df,
                          ifelse(BMI >= 30, "Obese",
                                 ifelse(BMI >= 25, "Overweight",
                                        "Normal")))
table(patient.df$bmi_cat)
```

```
#R: 
#R:      Normal      Obese Overweight 
#R:        6916       4185       5866 
```

Subsetting in calculations

Average BMI of females

```
with(patient.df, mean(BMI[Gender == "Female"], na.rm = TRUE))
```

```
#R:   [1] 27.45053
```

Average BMI of African males

```r
with(patient.df,
     mean(BMI[Gender == "Male" & Ethnicity == "African"],
     na.rm = TRUE))
```

```
#R:   [1] 26.44199
```

Summary

- Functions in **R**
- Installing and loading **R** packages
- Subsetting vectors and datasets
- `ifelse()` function
- Creating new variables