R Introduction - Frames

Structured data is usually organized in tables that have a certain number of rows and columns like an Excel spreadsheet or relational database table. R data frames are a type of data structure designed to hold such tabular data. A data frame consists of a number of rows and columns with each column representing some variable or feature of the data and each row representing a record, case or data point. A data frame is similar to a matrix in that it is a 2-dimensional data structure but unlike a matrix, different columns can hold data of different types. A data frame is actually just a list under the hood–a list where each object(column) is a vector with the same number of items.

Creating Data Frames

You can create a new data frame by passing vectors of the same length to the data.frame() function. The vectors you pass in become the columns of the data frame. The data you pass in can be named or unnamed:

```
a <- c(1,2,3,4,5)                    # Create some vectors
b <- c("Life","Is","Study!","Let's","Learn")
c <- c(TRUE,FALSE,TRUE,TRUE,FALSE)

my_frame <- data.frame(a,b,c)       # Create a new data frame

my_frame
```

```
##   a      b     c
## 1 1   Life  TRUE
## 2 2     Is FALSE
## 3 3 Study!  TRUE
## 4 4  Let's  TRUE
## 5 5  Learn FALSE
```

Since we did not supply column names, the columns took the names of the variables used to create the data frame. We could have assigned column names when creating the data frame like this:

```
my_frame <- data.frame(numeric = a, character = b, logical = c)

my_frame
```

```
##   numeric character logical
## 1       1      Life    TRUE
## 2       2        Is   FALSE
## 3       3    Study!    TRUE
## 4       4     Let's    TRUE
## 5       5     Learn   FALSE
```

You can check and reassign column names using the colnames() or names() functions:

```
colnames(my_frame)
```

```
## [1] "numeric"   "character" "logical"
```

```
names(my_frame)
```

```
## [1] "numeric"   "character" "logical"
```

```
colnames(my_frame) <- c("c1","c2","c3")

colnames(my_frame)
```

```
## [1] "c1" "c2" "c3"
```

Data frames also support named rows. You can create row names when creating a data frame by including the row.names argument and setting it equal to a character vector to be used for row names:

```
my_frame <- data.frame(numeric = a, character = b, logical = c,
                      row.names = c("r1","r2","r3","r4","r5"))

my_frame
```

```
##    numeric character logical
## r1       1      Life    TRUE
## r2       2        Is   FALSE
## r3       3    Study!    TRUE
## r4       4     Let's    TRUE
## r5       5     Learn   FALSE
```

You can check and alter row names after creating a data frame using the rownames() function:

```
rownames(my_frame)
```

```
## [1] "r1" "r2" "r3" "r4" "r5"
```

```
rownames(my_frame) <- 1:5

rownames(my_frame)
```

```
## [1] "1" "2" "3" "4" "5"
```

Another way to create a data frame is to coerce an existing matrix into data frame using the as.data.frame() function:

```
X <- matrix(seq(10,1000,10),10,10)    #Create a 10 x 10 matrix

X_frame <- as.data.frame(X)           #Turn the matrix into a data frame

X_frame
```

```
##     V1  V2  V3  V4  V5  V6  V7  V8  V9  V10
## 1   10 110 210 310 410 510 610 710 810  910
## 2   20 120 220 320 420 520 620 720 820  920
## 3   30 130 230 330 430 530 630 730 830  930
## 4   40 140 240 340 440 540 640 740 840  940
## 5   50 150 250 350 450 550 650 750 850  950
## 6   60 160 260 360 460 560 660 760 860  960
## 7   70 170 270 370 470 570 670 770 870  970
## 8   80 180 280 380 480 580 680 780 880  980
## 9   90 190 290 390 490 590 690 790 890  990
## 10 100 200 300 400 500 600 700 800 900 1000
```

In practice, most of the data frames you work with probably won't be data frames you create yourself. When you load data into R for analysis from a tabular data source like an Excel file or comma separated values file (CSV), it is usually structured as data frame. We will cover reading data into R in an upcoming lesson. For the rest of this lesson we'll work with the mtcars data set, a small set of car-related data built into R.

```
cars <- mtcars        # Load the mtcars data

print(cars)
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Summarizing Data Frames

When you load new into R, it is a good idea to explore the data to get a sense of the variables and values it contains before moving on to any kind of analysis. Real world data is often very messy and cluttered with things like oddly formatted values and missing (NA) values. Cleaning data to get it into a form that you can work with to perform analysis–often called data munging or data wrangling–can be of the most time intensive tasks necessary to work with data. Data exploration and summaries help determine out what, if anything, needs to be cleaned. Data frames support many of the summary functions that apply to matrices and lists. The summary() function is perhaps the most useful as it gives summary statistics for each variable in the data frame:

```
summary(cars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

The str() function provides a structural overview of a data frame including the number of observations and variables:

```
str(cars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

*Note: the environment pane in the upper right corner of RStudio also provides useful summary information for data frames. If a data frame is large, you won't want to try to print the entire frame to the screen. You can look at a few rows at the beginning or end of a data frame using the head() and tail() functions respectively:

```
head(cars, 5)      # Look at the first 5 rows of the data frame
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

```
tail(cars, 5)      # Look at the last 5 rows of the data frame
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino    19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E      21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

Data frames support a few other basic summary operations:

```
dim(cars)      # Get the dimensions of the data frame
```

```
## [1] 32 11
```

```
nrow(cars)
```

```
## [1] 32
```

```
ncol(cars)     # Get the number of columns
```

```
## [1] 11
```

Data Frame Indexing

Since data frame are lists where each list object is a column, they support all indexing operations that apply to lists:

```
head( mtcars[6]  )       # Single brackets take column slices
```

```
##                      wt
## Mazda RX4         2.620
## Mazda RX4 Wag     2.875
## Datsun 710        2.320
## Hornet 4 Drive    3.215
## Hornet Sportabout 3.440
## Valiant           3.460
```

```
typeof( mtcars[6] )      # And return a new data frame
```

```
## [1] "list"
```

```
head( mtcars[[6]]  )     # Double brackets get the actual object at the index
```

```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460
```

```
typeof( mtcars[[6]]  )
```

```
## [1] "double"
```

```
head( mtcars[["wt"]]  )  # Column name notation in double brackets works
```

```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460
```

```
head( mtcars$wt  )        # As does the $ notation
```

```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460
```

Data frames also support matrix-like indexing by using a single square bracket with a comma separating the index value for the row and column. Matrix indexing allows you get values by row or specific values within the data frame:

```
cars[2,6]    # Get the value at row 2 column 6
```

```
## [1] 2.875
```

```
cars[2, ]    # Get the second row
```

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
cars[ ,6]    # Get the 6th column
```

```
##  [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440
## [12] 4.070 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520
## [23] 3.435 3.840 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

```
cars["Mazda RX4", ]    # Get a row by using its name
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

```
cars[ ,"mpg"]    # Get a column by using its name
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

All of the indexing methods shown in previous lessons still apply, even logical indexing:

```
cars[(cars$mpg > 25), ]    # Get rows where mpg is greater than 25
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

Instead of logical indexing, you can also use the subset() function to create data frame subsets based on logical statements. subset() takes the data frame as the first argument and then a logical statement as the second argument create a subset:

```
subset(cars, (mpg > 20) & (hp > 70))    # Subset with over 20 mpg and 70 horsepower
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

The matrix functions cbind() and rbind() we covered work on data frames, providing an easy way to combine two data frames with the same number of rows or columns. You can also delete columns in a data frame by assigning them a value of NULL:

```
cars$vs <- NULL      # Drop the column "vs"

cars$carb <- NULL    # Drop the column "carb"
subset(cars, (mpg > 20) & (hp > 70))
```

```
##                 mpg cyl  disp  hp drat    wt  qsec am gear
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  1    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  1    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1    4
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  0    3
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  0    4
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  0    3
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  1    5
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1    5
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1    4
```

You cannot drop rows by assigning them a value of NULL due to the way data frames are stored as lists of columns. If you want to drop rows, you can use matrix-style subsetting with the -operator:

```
cars <- cars[-c(1, 3), ]    # Drop rows 1 and 3

head( cars )                # Note Mazda RX4 and Datsun 710 have been removed
```

```
##                    mpg cyl  disp  hp drat    wt  qsec am gear
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  1    4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  0    3
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0    3
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  0    3
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0    3
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  0    4
```

Data frames are one of the main reasons R is a good tool for working with data. Data in many common formats translate directly into R data frames and they are easy to summarize and subset.

Before we learn how to read data into R, there's one more data structure we need to discuss. Earlier in this lesson we created a data frame called my_frame with a column name "character":

```
my_frame
```

```
##   numeric character logical
## 1       1      Life    TRUE
## 2       2        Is   FALSE
## 3       3    Study!    TRUE
## 4       4     Let's    TRUE
## 5       5     Learn   FALSE
```

If we check the type of column "character", we have a surprise in store:

```
typeof( my_frame$character )
```

```
## [1] "integer"
```

How can a column that appears to hold characters be of type integer? It turns out that when you create a data frame, all character vectors in the data frame are converted into a special data structure called a factor by default. You can suppress this behavior by including the argument "stringsAsFactors = FALSE" when creating a data frame:

```
my_frame <- data.frame(numeric = a, character = b, logical = c,
                       stringsAsFactors = FALSE)

typeof( my_frame$character )
```

```
## [1] "character"
```