Base R Data Structures: Data Frames

Common Data Structures

A data scientist needs to deal with data! We need to have a firm foundation in the ways that we can store our data in R. This section goes through the most commonly used 'built-in' R objects that we'll use.

- There are five major data structures used in R

    1. Atomic Vector (1d)
    2. Matrix (2d)
    3. Array (nd)
    4. Data Frame (2d)
    5. List (1d)

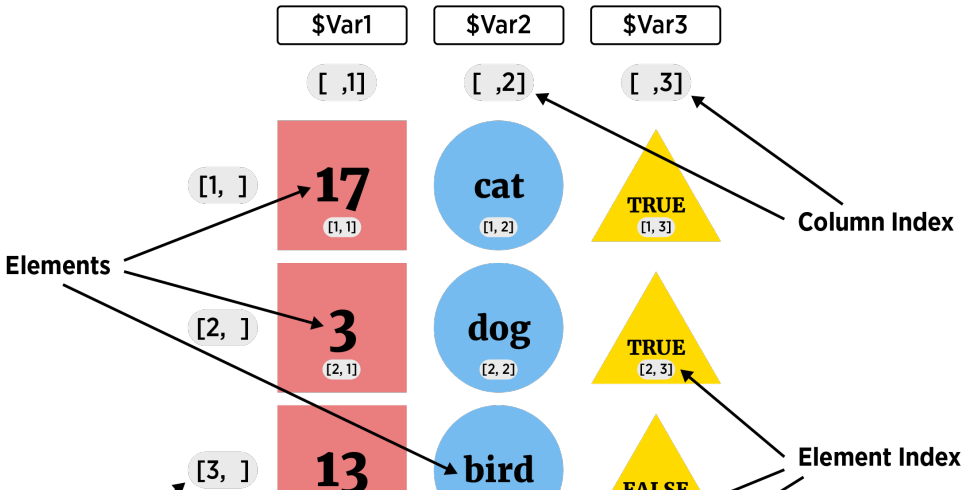
Dimension	Homogeneous (elements all the same)	Heterogeneous (elements may differ)
1d	Atomic Vector	List
2d	Matrix	Data Frame
nd	Array	

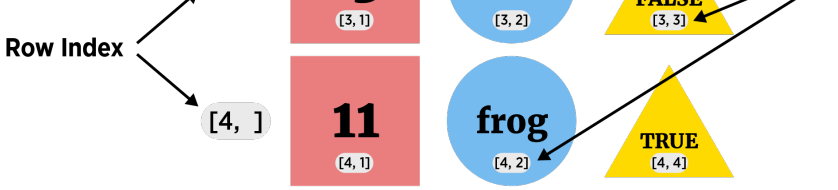
Data Frame

We've now gone through the common data structures that are homogeneous. Next we'll take on the data frame. These are 2D objects where the columns can be of different types!

Data Frames

- A collection (list) of **vectors** of the same *length*
- Perfect for most data sets!





Creating a Data Frame

We can create a data frame using the `data.frame()` function. From the help:

```
data.frame(..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
fix.empty.names = TRUE, stringsAsFactors = FALSE)
```

The help isn't super clear on what the first argument `...` should be. Essentially, this syntax allows us to pass vectors (each of the same length) as the columns of our data frame.

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
my_df <- data.frame(x, y, z)
my_df
```

```
  x  y  z
1 a  1 10
2 b  3 11
3 c  4 12
4 d -1 13
5 e  5 14
6 f  6 15
```

Data Frame Attributes

Data frames have one attribute of note: `names`.

```
str(my_df)
```

```
'data.frame':   6 obs. of  3 variables:
 $ x: chr  "a" "b" "c" "d" ...
 $ y: num  1 3 4 -1 5 6
 $ z: int  10 11 12 13 14 15
```

```
attributes(my_df)
```

```
$names
[1] "x" "y" "z"

$class
[1] "data.frame"

$row.names
```

```
[1] 1 2 3 4 5 6
```

- Our data frame originally inherited the names from the data objects passed in.
- If we didn't pass in objects with names, we get some default nonsense.

```
data.frame(1:5, c("a", "b", "c", "d", "e"))
```

```
  X1.5 c..a....b....c....d....e..
1    1                         a
2    2                         b
3    3                         c
4    4                         d
5    5                         e
```

- We can set the names explicitly when we create the data frame.

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
my_df <- data.frame(char = x, data1 = y, data2 = z)
my_df
```

```
  char data1 data2
1    a     1    10
2    b     3    11
3    c     4    12
4    d    -1    13
5    e     5    14
6    f     6    15
```

```
data.frame(number = 1:5, letter = c("a", "b", "c", "d", "e"))
```

```
  number letter
1      1      a
2      2      b
3      3      c
4      4      d
5      5      e
```

- Note: A syntactically valid name consists of letters, numbers and the dot or underline characters. It must start with a letter or a dot not followed by a number.
- You shouldn't use `.` with names generally. Periods are often used for **methods** and other things in programming.
- You also should avoid reserved words: `if else repeat while function for in next break TRUE FALSE NULL Inf NaN NA NA_integer_ NA_real_ NA_complex_ NA_character_` (type `?make.names` into the console for more details)

Accessing Elements of a Data Frame

Let's check out the 'built-in' `iris` data frame

```
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1
1 1 1 ...
```

- This is a 2D structure and we can access just like a matrix!

```
iris[1:4, 2:4] #returns a data frame
```

```
  Sepal.Width Petal.Length Petal.Width
1         3.5          1.4         0.2
2         3.0          1.4         0.2
3         3.2          1.3         0.2
4         3.1          1.5         0.2
```

```
iris[1, ] #returns a data frame
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
```

```
iris[1:10, 1] #returns a vector
```

```
 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

- Note again that R simplifies objects sometimes! The last return was simplified to a vector.
- This behavior actually comes from using the `[` function. **Everything you do in R is a function call!**
- We can avoid the behavior by adding the `drop = FALSE` argument

```
iris[1:10, 1, drop = FALSE] #return a data frame
```

```
   Sepal.Length
1           5.1
2           4.9
3           4.7
4           4.6
5           5.0
6           5.4
7           4.6
8           5.0
9           4.4
10          4.9
```

- Or we can call the `[` function as a **prefix** function (like our usual function calls)

```
`[`(iris, 1:10, 1, drop = FALSE)
```

```
   Sepal.Length
1           5.1
2           4.9
3           4.7
4           4.6
5           5.0
6           5.4
7           4.6
8           5.0
9           4.4
10          4.9
```

- Usually data frames have *meaningful* column names. We can use these for subsetting

```
iris[1:5 , c("Sepal.Length", "Species")]
```

```
  Sepal.Length Species
1          5.1  setosa
2          4.9  setosa
3          4.7  setosa
4          4.6  setosa
5          5.0  setosa
```

- **The most common way to access a single column is via the `$` operator.** This returns a vector.

```
iris$Sepal.Length
```

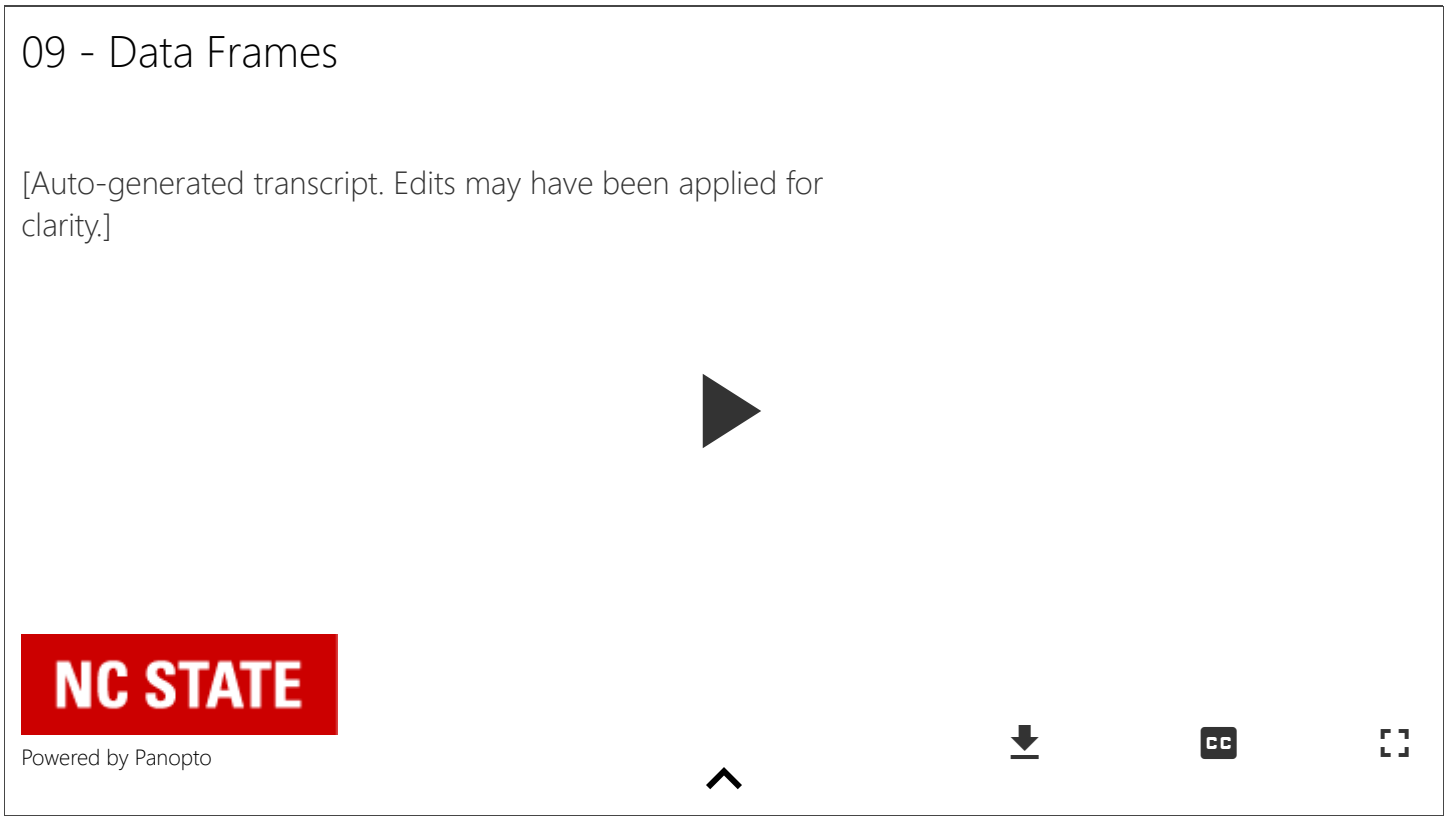
```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
5.1
 [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9
5.0
 [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9
5.5
 [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9
6.1
 [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6
5.5
 [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9
7.3
[109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7
7.2
[127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8
6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
```

- RStudio fills in options for us!
  - Type `iris$`
  - If no choices - hit tab

- Hit tab again to choose

Quick R Video

Please pop this video out and watch it in the full panopto player!



Recap!

Data Frame (2D data structure)

- Collection (list) of **vectors** of the same *length*
- Create with `data.frame()` function
- Access with `[ , ]` or `$`
- Perfect for most data sets!
- Most functions that read 2D data store it as a `data frame` (or `tibble` - a special data frame covered later)