Dimension

Before we begin, let's open up the `frame.r` file within RStudio. See instructions below:

info

Open the `frame.r` file

Within RStudio, open the `frame.r` file by selecting: `File` —> `Open File...` —> `code` —> `import` —> `frame.r`

Dimension

There are certain functions that you use to understand more about the data frame you have created. For example `dim()` allows you to see the dimension of the data frame.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(dim(df))
```

Console Result:

```
[1] 18  2
```

`18` represents the number of rows and `2` represents the number of columns.

Structure

Structure

You can use the `str()` command to see a more detailed description of the data frame.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(str(df))
```

Console Result:

```
'data.frame':   18 obs. of  2 variables:
 $ h: int  74 68 70 72 66 66 64 71 72 69 ...
 $ w: int  170 166 155 167 124 115 121 158 175 143 ...
NULL
```

The description tells you that your `data.frame` structure has `18` objects that are stored within `2` variables. It also lists the variable and its element type as well as the first **10** pieces of data. `NULL` is returned by default when using `str()` so do not worry about it.

Summary

Summary

`summary()` is an incredibly useful function that returns different types of statistics regarding your data frame.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(summary(df))
```

Console Result:

```
       h               w         
 Min.   :62.00   Min.   : 98.0  
 1st Qu.:66.25   1st Qu.:132.0  
 Median :69.50   Median :150.0  
 Mean   :69.06   Mean   :146.7  
 3rd Qu.:71.75   3rd Qu.:165.2  
 Max.   :75.00   Max.   :176.0  
```

It provides you the minimum, first quantile, median, mean, third quantile, and maximum values for each column of data within your data frame. From here, you can use this data for further data analysis.

Quantiles

Quantiles are sometimes referred to as **quartiles**. Each quantile represents 25% or a fourth of the data. In R:
* The **first quantile** represents the data point that is at the 25th percent mark of the sorted data set.
* The second quantile represents the data point that is at the 50th percent mark of the sorted data set (more commonly known as the **median**).
* The **third quantile** represents the data point that is at the 75th percent mark of the sorted data set.

* And lastly, the fourth quantile represents the data point that is at the 100th percent mark of the sorted data set (more commonly known as the **maximum**).

If you only want to calculate quantiles, you may use the `quantile()` function on a vector. For example:

```
print(quantile(h))
```

and

```
print(quantile(w))
```

will result in:

```
   0%   25%   50%   75%  100% 
62.00 66.25 69.50 71.75 75.00 
    0%    25%    50%    75%   100% 
 98.00 132.00 150.00 165.25 176.00 
```

You can see that the calculated quantiles, first and third, for `h` and `w` match the ones that are provided by the `summary()` function.

Column Names

Column Names

Use `colnames()` to return all of the column names or categories of data you have.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(colnames(df))
```

Console Result:

```
[1] "h" "w"
```

Note that we currently only have 2 columns of data within our data frame, `"h"` and `"w"`. Feel free to rename these variables as needed to be more descriptive or easier to understand.

Renaming with `colnames()`

You can actually use `colnames()` to change the names of your columns. Let's take a look at this example which has the addition of the column `Name`.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
n <- d$Name
df <- data.frame(h, w, n)

print(colnames(df))
```

To rename **one** specific column, you can use:

```r
colnames(df)[colnames(df)=="h"] <- "height"

print(colnames(df))
```

To rename **all** columns, you can use:

```r
colnames(df) <- c("height", "weight", "name")

print(colnames(df))
```

Note that if you don't provide each column with a new name, that column will have a new label of `NA`.

To rename **some** columns, you can use:

```r
colnames(df)[colnames(df) %in% c("h", "n")] <- c("height",
        "name")

print(colnames(df))
```

Head

Head

`head()` by default returns only the **first** 6 rows of data within the data frame.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(head(df))
```

Console Result:

```
  h   w
1 74 170
2 68 166
3 70 155
4 72 167
5 66 124
6 66 115
```

If you want to see more or fewer rows of data, you can specify it within the function by using a comma `,` followed by the letter `n`, followed by the equal `=`, and lastly the number of rows you want (for example, `10` or `3`).

```
print(head(df, n = 10))
print(head(df, n = 3))
```

n = 10 returns 10 rows of data

```
    h   w
1   74 170
2   68 166
3   70 155
4   72 167
5   66 124
6   66 115
7   64 121
8   71 158
9   72 175
10 69 143
```

n = 3 returns 3 rows of data

```
   h   w
1 74 170
2 68 166
3 70 155
```

`head()` is useful if you just want to peek at the data without having to see the entire data frame (which can be very long in some cases).

Tail

Tail

On the other hand, `tail()` by default returns only the **last** 6 rows of data within the data frame.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

print(tail(df))
```

Console Result:

```
    h   w
13 62  98
14 75 160
15 70 145
16 67 135
17 71 176
18 65 131
```

Note how you can see that the last 6 rows of data are shown by looking at the line numbers. Like with `head()`, you can see more or fewer rows of data by specifying it within the function using a comma `,` followed by the letter `n`, followed by the equal `=`, and lastly the number of rows you want (for example, `8` or `2`).

```
print(tail(df, n = 8))
print(tail(df, n = 2))
```

n = 8 returns 8 rows of data

```
    h   w
11 69 139
12 72 163
13 62  98
14 75 160
15 70 145
16 67 135
17 71 176
18 65 131
```

n = 2 returns 2 rows of data

```
    h   w
17 71 176
18 65 131
```

View

View

Lastly, `View()`, notice that the `V` is uppercase, will capture your data frame and present it in an actual table form for better readability. You **should not** use the `print()` function along with the `View()` function.

```
d <- read.csv("data/biostats.csv")

h <- d$Height..in.
w <- d$Weight..lbs.
df <- data.frame(h, w)

View(df) ## do not use the print() command
```

Note that you will see the table appear in a new tab in RStudio. The result will not be printed in the console like other previous examples.

Result:

	h	w
1	74	170
2	68	166
3	70	155
4	72	167
5	66	124
6	66	115
7	64	121
8	71	158
9	72	175
10	69	143
11	69	139
12	72	163
13	62	98
14	75	160
15	70	145
16	67	135
17	71	176
18	65	131

.guides/img/import/view-command

Slice

Slicing a Data Frame

To **slice** a data frame is to take only a portion of the data frame. Whatever is not included in that portion is left out. For example, if I have the following data frame:

```
df <- read.csv("data/biostats.csv")

print(class(df))
```

important

IMPORTANT

Note that using the `read.csv()` automatically converts the data from the file into a data frame. You don't have to use `data.frame()`.

I can then choose to slice the data frame `df` into portions by using brackets `[]` and a colon `:`. For example:

```
df <- read.csv("data/biostats.csv")

print(df[2])
```

returns:

```
      Sex
1       M
2       M
3       M
4       M
5       F
6       F
7       F
8       M
9       M
10      M
11      F
12      M
13      F
14      M
15      M
16      F
17      M
18      F
```

because specifying `2` in `print(df[2])` tells the system to print only the **second** column of data.

To print a **range** of columns of data, use a colon `:` and specify the start and end column numbers before and after the colon respectively. For example:

```
df <- read.csv("data/biostats.csv")

print(df[1:3])
```

will print all columns of data from and including column number `1` through `3`.

```
   Name      Sex Age
1  Alex        M  41
2  Bert        M  42
3  Carl        M  32
4  Dave        M  39
5  Elly        F  30
6  Fran        F  33
7  Gwen        F  26
8  Hank        M  30
9  Ivan        M  53
10 Jake        M  32
11 Kate        F  47
12 Luke        M  34
13 Myra        F  23
14 Neil        M  36
15 Omar        M  38
16 Page        F  31
17 Quin        M  29
18 Ruth        F  28
```

To store this subset of the data frame as another data frame, you can use:

```r
df <- read.csv("data/biostats.csv")

df2 <- df[1:3]
print(df2)
```

Slicing by Row and Column

Using a single colon `:` will slice the data frame by columns. For example, `df[1:3]` extracts all data from columns `1` through `3`. However, if I want particular **rows** and **columns** to be extracted, I need to specify more.

You can think of data frames as having coordinate points on a graph. Each pair of coordinate points tells you the "x" or row coordinate and "y" or column coordinate. These coordinates help you pinpoint the location on the graph. Similar to that, you can specify how the system should slice the data frame by specifying the **row range** and **column range**. For example, if I want rows **1 through 4** and columns **2 and 3** to be concluded in the slice, I can use:

```
df <- read.csv("data/biostats.csv")
df2 <- df[1:3]

print(df2[1:4, 2:3])
```

which results in:

```
      Sex Age
1       M  41
2       M  42
3       M  32
4       M  39
```

You can see that only data from rows 1 through 4 as well as from columns 2 and 3 are included in the slice.

Selecting by Condition

Lastly, I can select columns of data based on certain conditions using the function `subset()`. For example:

```
df <- read.csv("data/biostats.csv")
df2 <- df[1:4, 2:3]

print(subset(df2, Age < 40))
```

results in:

```
      Sex Age
3       M  32
4       M  39
```

because I specified the system to select and print only data where `Age` is less than `40` from the data frame `df2`. Here's another example:

```
df <- read.csv("data/biostats.csv")

print(subset(df, Height..in. > 70))
```

results in:

```
   Name      Sex Age Height..in. Weight..lbs.
1  Alex        M  41          74          170
4  Dave        M  39          72          167
8  Hank        M  30          71          158
9  Ivan        M  53          72          175
12 Luke        M  34          72          163
14 Neil        M  36          75          160
17 Quin        M  29          71          176
```

because I wanted only rows of data where `Height..in.` are greater than `70` from the data frame `df`.

Append & Sort

Appending to a Data Frame

You can append to a data frame by using the dollar sign `$` symbol.

For example, if I want to add a vector called `Month` with the following values `"Jan", "Feb", "Dec", "June"` to my data frame `df2`, I can use:

```
df <- read.csv("data/biostats.csv")
df2 <- df[5:8, 2:3]

df2$Month <- c("Jan", "Feb", "Dec", "June")
print(df2)
```

which results in:

```
      Sex Age Month
5       F  30   Jan
6       F  33   Feb
7       F  26   Dec
8       M  30  June
```

Notice how the column of data `Month` is added to my data frame at the end as the final column.

Sorting Data within a Data Frame

To sort data within a data frame, I can use the `order()` function. The basic syntax for `order()` is:

```
data_frame[order(vector_name),]
```

where:
* `data_frame` represents the data frame being used
* `order` represents the calling of the `order()` function
* `vector_name` represents the column of data you want the system to sort by
* `,` represents that you want all of the column data to be shown after sorting
* To show only a particular column after sorting, specify that column

number after the comma `,`. For example:
`data_frame[order(vector_name),2]` will only print column number `2` after the data is sorted.

Examples:

```
df <- read.csv("data/biostats.csv")
df2 <- df[5:8, 2:3]
df2$Month <- c("Jan", "Feb", "Dec", "June")

print(df2[order(df2$Age),])
```

results in:

```
      Sex Age Month
7       F  26   Dec
5       F  30   Jan
8       M  30  June
6       F  33   Feb
```

However, if I only want to see `Month` after the data frame is sorted according to `Age`, I can use:

```
df <- read.csv("data/biostats.csv")
df2 <- df[5:8, 2:3]
df2$Month <- c("Jan", "Feb", "Dec", "June")

print(df2[order(df2$Age),3]) # 3 refers to Month column
```

which results in:

```
[1] "Dec"  "Jan"  "June" "Feb"
```

and represented as a vector.

Sort in Descending Order

By default, the data will be sorted in **ascending** order. To sort in **descending** order, simply place a minus sign (`-`) in front of the column name you want sorted (`-df2$Age`):

```
df <- read.csv("data/biostats.csv")
df2 <- df[5:8, 2:3]
df2$Month <- c("Jan", "Feb", "Dec", "June")

print(df2[order(-df2$Age),])
```

which will now print:

```
       Sex Age Month
6        F  33   Feb
5        F  30   Jan
8        M  30  June
7        F  26   Dec
```

Combination

Combining Multiple Data Frames

If you have multiple data frames that contain the same categories of data, you can combine them all into one data frame using the `rbind()` function. First, let's take a look at what's inside the `colors1.csv` file.

```
df1 <- read.csv("data/colors1.csv")

print(df1)
```

which results in:

```
   Color Count Male Female
1    Red    56   30     26
2   Blue    78   38     40
3  Green    62   45     17
4 Yellow    48   18     30
```

Now let's say you gathered additional data in a separate file called `colors2.csv` and wanted to combine this data with the data in `colors1.csv`; first, you can store `colors2.csv` as a data frame (e.g. `df2`), and then use the `rbind()` function to combine the two data frames together into a new data frame (e.g. `df`).

```
df1 <- read.csv("data/colors1.csv")
df2 <- read.csv("data/colors2.csv")
df <- rbind(df1, df2)

print(df)
```

Your new data frame `df` now contains the combined data of `df1` and `df2`.

```
   Color Count Male Female
1    Red    56   30     26
2   Blue    78   38     40
3  Green    62   45     17
4 Yellow    48   18     30
5 Orange    17   10      7
6 Purple    32   21     11
7   Pink    23    8     15
```

important

IMPORTANT

As mentioned previously, your data frames must contain the **same** categories (or column names) in order for you to use `rbind()` to combine them. Try importing two CSV files which contain completely different categories of data and see what happens when you attempt to combine them using `rbind()`.

```
df1 <- read.csv("data/colors1.csv")
df2 <- read.csv("data/biostats.csv")
df <- rbind(df1, df2)

print(df)
```

You are not limited to just two data frames, so feel free to combine more data frames using `rbind()` if needed. For example:

```
df1 <- read.csv("data/colors1.csv")
df2 <- read.csv("data/colors2.csv")
df3 <- read.csv("data/colors3.csv")
df <- rbind(df1, df2, df3) #combining three data frames

print(df)
```

Result:

```
   Color Count Male Female
1    Red    56   30     26
2   Blue    78   38     40
3  Green    62   45     17
4 Yellow    48   18     30
5 Orange    17   10      7
6 Purple    32   21     11
7   Pink    23    8     15
8  Other    39   19     20
```