3. CREATE & SUBSET DATA

CREATE DATA

We use the assignment operator to create vectors, which are just collections of values, separated by commas. String values must be quoted with either single or double quotes. Numeric values are not quoted. The "c" command below means collect, combine or concatenate, and is necessary when we have multiple values that we wish to pull together into a vector.

```
age <-  c(20:25)
name <-  c('greg', 'sally', 'sean', 'carlos', 'becka', 'doug')
whoknows <-  age+5
```

Dataframes

Dataframes are the main data structure used for manipulating and analyzing data in R. A dataframe is essentially a rectangular arrangement of varaibles in columns and observations in rows. Dataframes can be created from vectors using the data.frame() command.

```
df <- data.frame(age, name, whoknows)
```

The extraction operator $

The base R extraction operator is the dollar sign, $. This command pulls a single column out of a dataframe as a vector. The extracted column is no longer a dataframe, but is instead a vector. More generally, the $ pulls smaller objects out of larger objects, such as a dataframe out of a list of output. Note that when writing this character in markdown space I sometimes put a backslash before the character. The backslash is an escape character and is used because the dollar sign can also be used to format text with italics in Markdown language, and here we wish to "escape" this functionality.

Let's extract a variable from the df dataframe and then show summary information for the extracted variable.

```
v1 <- df$whoknows
summary(v1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   25.00   26.25   27.50   27.50   28.75   30.00
```

Metadata

Metadata is information about data. Some metadata is stored as attributes. Every dataframe has at least three attributes, column names, row names and class. Click the white triangle in blue circle icon next to objects in the Environment tab to view useful metadata for dataframes. You can also use the str(df) command in the console window to view the same metadata. Values and variables (vectors) can also have metadata as shown below

```
# Show the attributes of df (or any object)
attributes(df)
```

```
## $names
## [1] "age"      "name"     "whoknows"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5 6
```

```
# names() is a command that adds naming metadata.
names(age) <- c('greg', 'sally', 'sean', 'carlos', 'becka', 'doug')

# Now the values of age are named. This is unnecessary and for demonstration
# purposes only, because we already have a separate column for names.
attributes(age)
```

```
## $names
## [1] "greg"   "sally"  "sean"   "carlos" "becka"  "doug"
```

```
# Don't confuse data with metadata. Even though age now has a name attribute, the
# age variable is still an integer variable.
2*age
```

```
##   greg  sally   sean carlos  becka   doug
##     40     42     44     46     48     50
```

Logical operators

Some variables and some statements are "logical" meaning that they take on a value of TRUE or FALSE. Logical statements use a collection of operators that appear similar to math expressions, but which always return a value of TRUE or FALSE. Logical statements are often used in ifelse statements to create new variables. Consequently, we need to understand logical statements before we can understand ifelse statements.

Important tip The "=", is NOT a logical operator. "=" is used to create objects by assignment, just like the assignment operator. For example...

```
x = 4
```

The Logical operators below return TRUE or FALSE. They do NOT assign values. Run the commands one at a time and pay close attention to the output (TRUE or FALSE).

```
x == 4
```

```
## [1] TRUE
```

```
x == 5
```

```
## [1] FALSE
```

```
x < 4
```

```
## [1] FALSE
```

```
x <= 4
```

```
## [1] TRUE
```

```
x > 1
```

```
## [1] TRUE
```

```
x != 0
```

```
## [1] TRUE
```

```
x < 1 | x > 3  # a vertical bar "|" means "or"
```

```
## [1] TRUE
```

```
x != 0 & x > 10 # The ampersand "&" means "and"
```

```
## [1] FALSE
```

Using ifelse() to create new variables

Now that we understand logicals, we can use the ifelse command to create new variables. Three arguments are essential to an ifelse statement:

1. test = , a logical statement that evalutes to TRUE or FALSE
2. yes = , identifies a value to return if the test returns TRUE.
3. no = , a value to return if the test returns FALSE.

Below we use an ifelse statement with a logical test argument to create a new variable called treat. Note that we are using the extraction operator with df$treat to insert a new variable into the df dataframe. In this case "extraction" isn't the best name for the operator, because really we are doing the opposite. View the df dataframe after running the code chunk below.

```
df$treat <- ifelse(test = df$age > 23,
                   yes = 'Young Adult',
                   no = 'Spring Chicken')
```

SUBSET DATA

Subsetting with Tidyverse commands (select and filter)

The Tidyverse package consists of several packages that form an ecosystem of commands that follow the same general structure. Many of the most popular commands for manipulating and summarizing dataframes come from the dplyr package within tidyverse. The select and filter commands shown below are examples, but many more are available. The dplyr vignette provides explanations and examples of how to use these commands, as well as a downloadable Cheat Sheet.

View each dataframe below after running the code.

```
# select() creates a smaller dataframe by selecting 1 or more specific columns
d1 <- select(df, name)
d2 <- select(df, name, age)

# filter() uses logical statements to reduce the rows of a dataframe to those that return TRUE for all
d3 <- filter(df, age >= 21, treat == 'Spring Chicken')

# select and filter can be combined using the pipe operator. White space has been added below to make t
d4 <-
    df %>%
```

```
    select(age, treat) %>%
    filter(age >= 21, treat == 'Spring Chicken')
```

Subset with Base R brackets (AKA braces)

Single Brackets [ ]

While I will mostly tidyverse commands for subsetting, students should also be familiar with the [ ] and [[ ]] methods. These are quick and useful and you'll come across them reviewing help vignettes and searching forums for coding solutions. Tidyverse commands are generally easier to read and understand. Also, the select() command always returns a dataframe, which is the data structure most used in data analysis and other tidyverse commands. The objects extracted by brackets may not be of the same type as the parent object and may not work with other tidyverse commands. To demonstrate the variety of bracket uses, run each code below one at a time and track the "temp" file in your environment. Note when temp is created as a dataframe versus a vector (AKA value).

```
temp <- df[c('name', 'whoknows')]
temp <- df[2]
temp <- df[2,3]
temp <- df[c(1,2), c(3,4)]
temp <- whoknows[2]
```

Double brackets [[ ]]

Two nested brackets are often used with a data structure called a list, but can also be used with vectors and dataframes to extract single elements as vectors. For most analysis and anytime we are working with dataframes, we won't use the double brackets.

```
temp <- df[[2]]
temp <- df[[2,3]]
# temp <- df[[c(1,2), c(3,4)]]
```