18.1 Lesson Summary - R We There Yet

R is a programming language specifically designed for statistical analysis. R has great performance when dealing with large sets of data and offers excellent data visualizations. There is significant overlap between Python and R when it comes to data processing functionality but having familiarity with both is critical to being able to easily transition to projects built with R or Python.

Concept: The **R** programming language is not too dissimilar from Python. In R, as with Python, you can assign variables, do conditional comparisons, and run loops. To assign a **variable** in R the variable name goes on the left of the statement with the value on the right but instead of an *'='* you use a *'<-'*. For example:

*my\_variable <- "Hello World"*

* Activity: 01\_Ins\_RBasics

Concept: There are different **variable types** in R. The way R represents basic data types such as integers or boolean values is a bit unique. Basic data types are represented as **vectors** with one element of that type. For example, an integer variable in R is represented as a vector with one value of type integer (R refers to these types as 'classes' but this shouldn't be confused with Python or traditional object-oriented programing classes). To create vector variables of boolean, numeric, and character type you could use the following code:

*my\_numeric\_variable <- 3.1415*

*my\_character\_variable <- "Hello World!"*

*my\_boolean\_variable <- TRUE*

* Activity: 01\_Ins\_RBasics
* Suppl link: <https://www.tutorialspoint.com/r/r_data_types.htm>

Concept: If you want to create a **group of objects** (like a **list** in Python) in R you can simply use the same vector object that you use for individual values though you must place a 'c' in front of it. All objects within a vector must be of the same type. To create a vector of letters you could use the following code:

*my\_letters <- c("a", "b", "c", "d", "e")*

To access the contents of a vector you can use syntax similar to a Python list, however R is a **one-index language**, meaning the first element in a vector is index 1, unlike Python where the first index of a list is 0. You can display a message to the console using the ***print*** command, just like in Python. To see the second element in the *my\_letters* vector you can use the following command:

*print(my\_letters[2])*

You can **combine vectors** of the same type quite easily. For example:

*my\_letters <- c("a", "b", "c", "d", "e")*

*my\_other\_letters <- c("w", "x", "y", "z")*

*my\_combined\_letters = c(my\_letters, my\_other\_letters)*

To **loop** through a vector, you can use a for loop similar to Python. For example:

*for (letter in my\_letters) {*

*print(letter)*

*}*

**If-then** statements in R have syntax similar to Python. For example:

*test\_var <- TRUE*

*if (test\_var == TRUE) {*

*print("That variable is true")*

*}*

* Activity: 01\_Ins\_RBasics

Concept: You can use functions in R to better organize your code. To create a function summing two numbers you could use the following code:

*sum\_nums\_function <- function(num1, num2){*

*my\_sum <- num1 + num2*

*return(my\_sum)*

*}*

*print(sum\_nums\_function(2,3))*

* Activity: 02\_Stu\_RBasics

Concept: You can get the **min**, **max**, **median**, **mean** and other **summary statistics** for a vector of numbers using the **summary** function. For example:

*my\_data <- c(4.1, 3.9, 4.5, 3.5, 4.5, 4.1)*

*my\_data\_summary <- summary(my\_data)*

*print(my\_data\_summary)*

*print(my\_data\_summary["Min."])*

You can use the ***sd*** function to get the **standard deviation** for a sequence of numbers. For example:

*my\_data <- c(4.1, 3.9, 4.5, 3.5, 4.5, 4.1)*

*print(sd(my\_data))*

The pipe character sequence, *%>%* can be used to chain values and functions together for easier readability. If you wanted to round the standard deviation of a series of numbers you could use the following code:

*my\_data <- c(4.1, 3.9, 4.5, 3.5, 4.5, 4.1)*

*without\_pipe <- round(sd(my\_data),2)*

*with\_pipe <- my\_data %>% sd() %>% round(2)*

* Activity: 03\_Ins\_Vectors

Concept: **Tibbles** in R are very similar to **DataFrames** in Pandas. Tibbles are table like organizations of columns and rows. A column should contain the same kind of data. You can work with a tibble in much the same way you work with a Pandas DataFrame. To get the first 5 rows you can use the following code:

*slice(my\_tibble, 1:5)*

To calculate the **number of rows** in a tibble you can use the following code:

*nrow(my\_tibble)*

To **filter** the table to just show the rows with "My Value" in "my\_column" you can use the following code:

filter(*my\_tibble, my\_column='My Value')*

To display the **mean** of a column of data you can use the following code:

*summarize*(*my\_tibble, mean(my\_column))*

To **group by** multiple columns, you can use the following code:

*my\_group <- group\_by(my\_tibble, my\_column\_1, my\_column\_2)*

To create a tibble from a CSV you can use the following code:

*my\_tibble <-read\_csv("my\_data.csv")*

* Activity: 05\_Ins\_Tibble, 06\_Stu\_Tibble