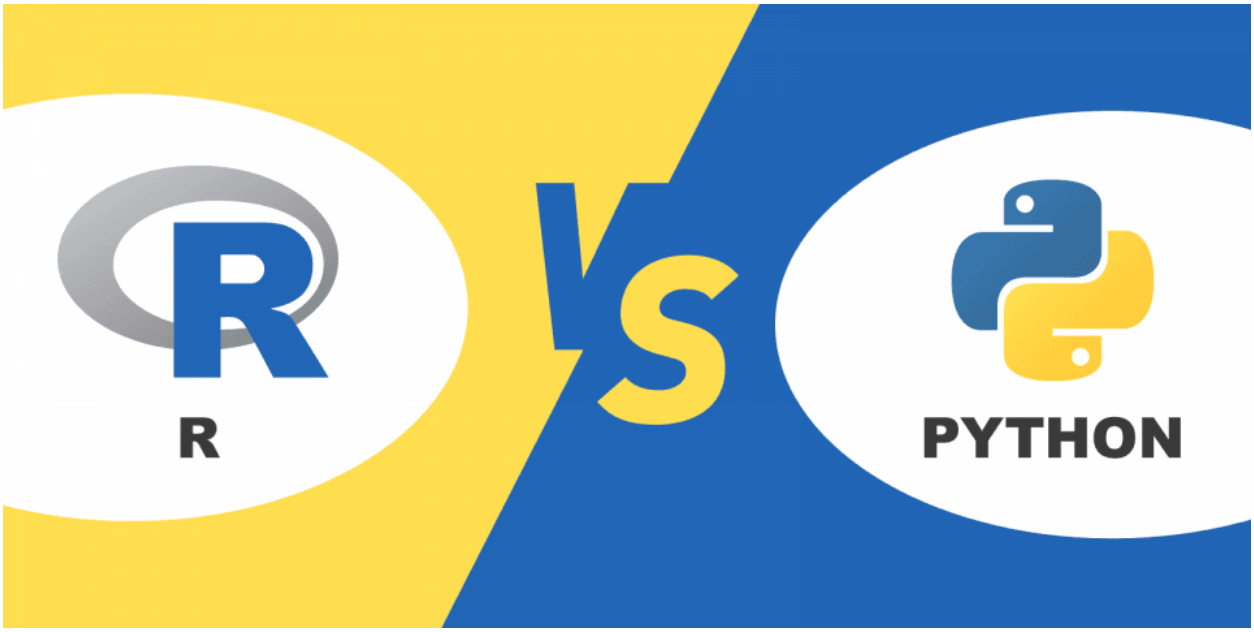
R and Python syntax comparison for Data Scientists



Compiled by Kiel Dang

Email: dang.v@northeastern.edu

Medium LinkedIn Git-hub Portfolio

Critical differences

This section highlights differences in Python and R that could result in inadvertent errors if the wrong convention is used (i.e. code may still run but would produce wrong results).

Topic	R	Python
General purposes	R was developed specifically for statistical computing and data analysis.	Python was developed as a general-purpose programming language.
Boolean	TRUE or T FALSE or F	True False ***All capital letters are not allowed***
Array indexing	Starts at 1	Starts at 0
Indentation	Has no impact on code – is purely cosmetic	Has a specific meaning in the code. Reducing the indentation level indicates the end of a block of code.
Length of a string	nchar(x) ***Do not use length(x)***	len(x)
Return statements in functions	If no return statement is specified, will return the last calculation done within the function	Return statement must be specified if we want the function to return an output; otherwise, it will return "None"
Interpretation of "="	The "=" sign will create an independent copy of the object. For example, if we do data_2 = data_1, and perform some manipulations on data_2, then data_1 will be unchanged.	The "=" sign will create a new pointer to the original object, which will not behave independently. For example, if we do data_2 = data_1, and perform some manipulations on data_2, the same operations will be applied to data_1. To make an independent copy of a dataset, use data_2 = data_1.copy() instead.

Structural differences

This section highlights differences in Python and R that represent significant differences in the way the code is structured, but which are unlikely to cause non-obvious errors (i.e. if the wrong approach is used then the code would not run).

Topic	R	Python

Code blocks	Are encased in braces { and }
	example = function(x){
	some code
	some more code
	return(something)
	}
		Begins with a line ending with a colon. On the next line, the indentation level increases by 1. The code block ends when the indentation level returns back to where it was at the start of the code block.
		def example(x):
		some code
		some more code
		return something
		more code that is not part of the function definition

Common ways to create unlabeled sequences of objects	In R, these are called **vectors**. Use the "c" command to create one, e.g. c(1, 2, 3) c here stands for combination. Elements in an R vector must all be of the same type.	In Python, this is called a **list**. Use square brackets with elements separated by commas, e.g. [1, 2, 3] Elements in a Python list can be of mixed type. Can also create a tuple using round parentheses, but these cannot be changed after being created. Example: (1, 2, 3)
Common ways to create labelled sequences of objects	In R, this is called a **list**. Use the "list" command to create one, separating key-value pairs with an equal sign, e.g. l = list('a' = 1, 'b' = 2, 'c' = 3) Access elements with the $ symbol, e.g. l$a is 1	In Python, this is called a **dictionary**. Use braces to create on, separating the list of key-value pairs with commas, e.g. d = {'a':1, 'b':2, 'c':3} Access elements using square brackets, e.g. d['a'] is 1
Applying a function across all elements of an array	Use the lapply command	Use the list comprehension syntax, e.g. [formula for x in list if condition]
Loop	for(i in 1:10) {...}	for i in range(10): ...
Conditional statement	if(x > 3) {...}	if x > 3: ...
Call a function	function(data)	function(data) data.function() In Python, we have more ways to call a function, in which data oriented is a common way: For example: mean(data) but we also call: data.mean()
Access a column in a data frame	1. Using the $ operator: data_frame$column 2. Using square brackets []: data_frame[, "column"]	1. Using square brackets []: data_frame["column"] 2. Using the dot notation: data_frame.column (only works if the column name does not contain any spaces or special characters)

Minor Differences

These are differences in naming or notational conventions that don't cause major changes in the structure of a code, but which might result in needing to change the name of a keyword or function. Items in this list will cause an obvious error (e.g. code won't run) if the wrong convention is used.

Topic	R	Python
Concatenating strings	Use "cat", paste() or "paste0" cat("Hello,", "world!") *Note: paste0() is similar to paste(), but it does not add any separator between the strings, while we can regulate the separator in paste(). For example, we can paste("Line 1", "Line 2", sep = "\n") to break line.*	Use "+" "Hello, " + "World!" **Other options**: format(), join() concatenated_string = "{}{}".format(string1, string2) concatenated_string = "_".join([string1, string2])
Displaying text	Use "print" – this can only display a single string	Use the print command. This can handle a sequence of strings / variables and will print them all out with a space between them.
Exponentiation	Can use a ** b or a^b	Use a ** b
Modular arithmetic	Use a %% b	Use a % b
Integer division, discarding remainder	Use a %/% b	Use a // b
Determine type of a variable	Use typeof(x)	Use type(x)
Change type of a variable	General format of the function is "to.type()". Example: to.integer(x)	General format of the function is "type()". Example: int(x)
Boolean variables	Use all-caps, TRUE and FALSE	Capitalize only first letter, True and False
Install package	install.packages('name')	pip install name
Importing additional functionality	These are called **packages** in R Use library(package) ***In R, when we access a library, all functions of that library will be available.***	These are called **modules** in Python Use from package import module from sklearn import metrics ***Note: In Python, every import only does with a specific function from that library.*** So if you need to import all modules, you need to use below syntax: from package import * For example: from pandas import *
Comment out	Ctrl + Shift + C	Windows: CTRL + 3 Mac: CMD + 3

Create a function	Use function()
	function_name <- function(arg1, arg2){
	return()
	}
		Use def()
		def function_name(arg1, arg2):
		return()

Lambda functions	Do not have. Still use using the function() keyword to create function.	Lambda functions are anonymous functions in Python, meaning they are functions without a name. They are used to perform a small task or calculation and are often used in combination with other functions like filter(), map() or reduce(). The syntax of a lambda function in Python is: lambda arguments: expression f = lambda x: x**2 print(f(5)) # 25
Condition ifelse()	R offers this.	Not offer. Need to use normal syntax: result = x if x > y else y
Call for help	Type in Console: ?function_name() ??function_name()→ To check the package that contains the function. For example: ?mutate()	Help(function_name) For example: Help(len) Note: Python also has dir(), a function used to return a list of valid attributes and methods of an object. For example, dir(list) returns a list of attributes and methods available for the built-in list type
Check available built-in functions	is("package:base") :This will return a character vector of all the functions in the base package or help(base) : see a list of all the functions in the base package, along with brief descriptions of each.	dir(__builtins__)
Unequal	!=	!= or <>

Some differences in Data Manipulation

These are differences in fundamental data cleaning steps when working with data frames. In R, **dplyr** is the main library for data frame manipulation along with R-based functions, while in Python, it's **Pandas**.

Topic	R	Python
Check structure	str(df)	df.info()
Data dimension	dim(df)	df.shape ***Note: No bracket here for shape***
Variables of data frame	colnames(df)	df.columns ***Note: No bracket here for columns***
Drop columns	- **Single:** df$column <- NULL - **Multiples:** ▪ By index df[, -c(column_index_1, column_index_2)] ▪ By name df[, !names(data_frame) %in% c("column_name1", "column_name2")]	- **Single**: del df["column_name"] or We can use below code with 1 column name. - **Multiples**: df.drop(columns=["column_1", "column_2"], inplace=True)
Check unique values	unique(df$column_name)	df["column_name"].unique() df["column_name"].value_counts() For value_counts() we can state an argument normalize = True to calculate **the proportion of** each element in the column.
Check duplicated observations	summary(duplicated(df))	df.duplicated().sum()
Drop duplicated values	df[!duplicated(df),] or you can use dplyr as follow: df %>% distinct()	df.drop_duplicates(inplace=True)
Check missing values	- **Single column** is.na(df$column_name) - **Multiple columns** sapply(df, function(x) sum(is.na(x)))	df.isnull().sum()
Drop NA	new_df <- na.omit(df)	new_df = df.dropna()
Fill NA	We can use replace_na() function in tidyr, or na.fill() from the zoo library, or we can use R base like this: df[is.na(df$col), "col"] <- value	It's easier to replace NA in Python. We just need to specify column and value then conduct this code: df.fillna(value) df.column_name.fillna(value)