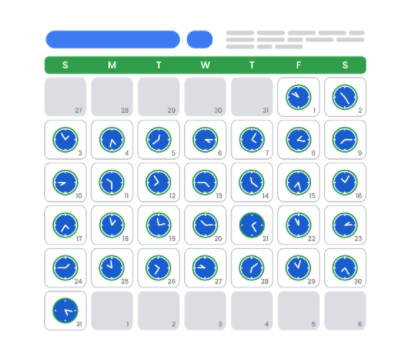
Dates and times in R

In this reading, you will learn how to work with dates and times in R using the **lubridate** package. Coming up, you will use tools in the lubridate package to convert different types of data in R into date and date-time formats.



Loading tidyverse and lubridate packages

Before you get started working with dates and times, you should load both **tidyverse** and **lubridate**. Lubridate is part of tidyverse.

First, open RStudio.

If you haven't already installed tidyverse, you can use the **install.packages()** function to do so:

* + **install.packages("tidyverse")**

Next, load the tidyverse and lubridate packages using the **library()** function. First, load the core tidyverse to make it available in your current R session:

* + **library(tidyverse)**

Then, load the lubridate package:

* + **library(lubridate)**

Now you’re ready to be introduced to the tools in the lubridate package.

Working with dates and times

This section covers the data types for dates and times in R and how to convert strings to date-time formats.

**Types**

In R, there are three types of data that refer to an instant in time:

* + A date **("2016-08-16")**
  + A time within a day **(“20:11:59 UTC")**
  + And a date-time. This is a date plus a time **("2018-03-31 18:15:48 UTC")**

The time is given in UTC, which stands for Universal Time Coordinated, more commonly called Universal Coordinated Time. This is the primary standard by which the world regulates clocks and time.

For example, to get the current date you can run the **today()** function. The date appears as year, month, and day.

**today()**

**#> [1] "2021-01-20"**

To get the current date-time you can run the **now()** function. Note that the time appears to the nearest second.

**now()**

**#> [1] "2021-01-20 16:25:05 UTC"**

When working with R, there are three ways you are likely to create date-time formats:

* + From a string
  + From an individual date
  + From an existing date/time object

R creates dates in the standard yyyy-mm-dd format by default.

Let's go over each.

**Converting from strings**

Date/time data often comes as strings. You can convert strings into dates and date-times using the tools provided by lubridate. These tools automatically work out the date/time format. First, identify the order in which the year, month, and day appear in your dates. Then, arrange the letters *y*, *m*, and *d* in the same order. That gives you the name of the lubridate function that will parse your date. For example, for the date *2021-01-20,* you use the order *ymd*:

**ymd("2021-01-20")**

When you run the function, R returns the date in yyyy-mm-dd format.

**#> [1] "2021-01-20"**

It works the same way for any order. For example, month, day, and year. R still returns the date in yyyy-mm-dd format.

**mdy("January 20th, 2021")**

**#> [1] "2021-01-20"**

Or, day, month, and year. R still returns the date in yyyy-mm-dd format.

**dmy("20-Jan-2021")**

**#> [1] "2021-01-20"**

These functions also take unquoted numbers and convert them into the yyyy-mm-dd format.

**ymd(20210120)**

**#> [1] "2021-01-20"**

**Creating date-time components**

The ymd() function and its variations create dates. To create a date-time from a date*,* add an underscore and one or more of the letters *h*, *m*, and s (hours, minutes, seconds) to the name of the function:

**ymd\_hms("2021-01-20 20:11:59")**

**#> [1] "2021-01-20 20:11:59 UTC"**

**mdy\_hm("01/20/2021 08:01")**

**#> [1] "2021-01-20 08:01:00 UTC"**

**Optional: Switching between existing date-time objects**

Finally, you might want to switch between a date-time and a date.

You can use the function **as\_date()** to convert a date-time to a date. For example, put the current date-time—now()—in the parentheses of the function.

**as\_date(now())**

**#> [1] "2021-01-20"**

Additional resources

To learn more about working with dates and times in R, check out the following resources:

* + [lubridate.tidyverse](https://lubridate.tidyverse.org/index.html): This is the “lubridate” entry from the official tidyverse documentation, which offers a comprehensive reference guide to the various tidyverse packages. Check out this link for an overview of key concepts and functions.
  + [Dates and times with lubridate: Cheat Sheet](https://rawgit.com/rstudio/cheatsheets/master/lubridate.pdf): This “cheat sheet” gives you a detailed map of all the different things you can do with the lubridate package. You don’t need to know all of this information, but the cheat sheet is a useful reference for any questions you might have about working with dates and times in R.

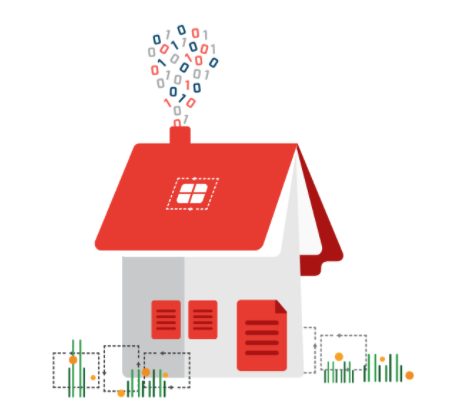
From <<https://www.coursera.org/learn/data-analysis-r/supplement/g0l4l/dates-and-times-in-r>>

Other common data structures

In this reading, you will continue on the topic of data structures with an introduction to data frames and matrices. You will learn about the basic properties of each structure, and simple ways to make use of them using R code. You will also briefly explore **files**, which are often used to access and store data and related information.

Data structures

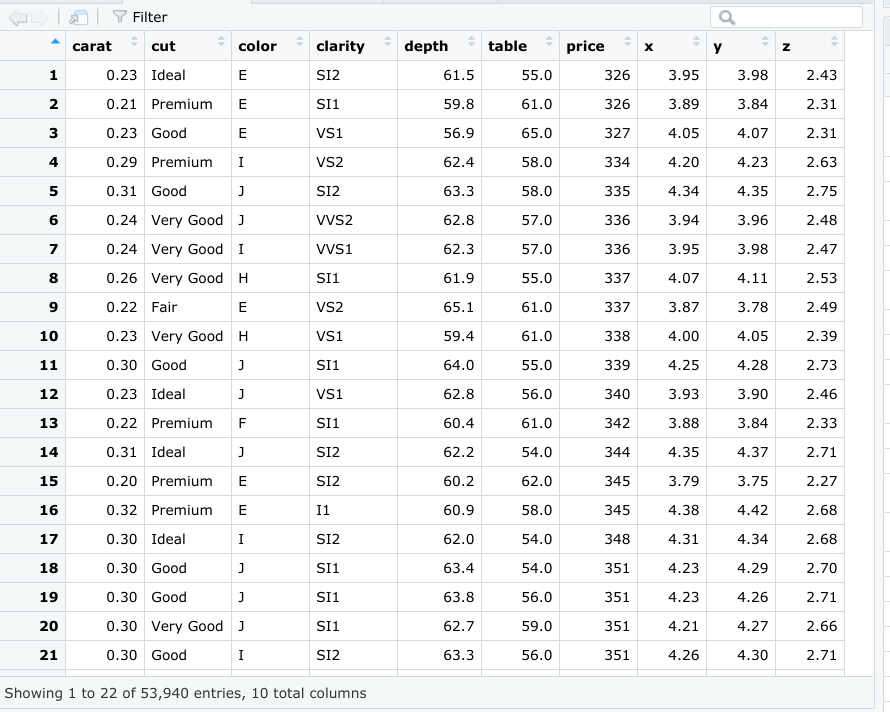
Recall that a data structure is like a house that contains your data.



**Data frames**

Data frames are the most common way of storing and analyzing data in R, so it’s important to understand what they are and how to create them. A **data frame** is a collection of columns–similar to a spreadsheet or SQL table. Each column has a name at the top that represents a variable, and includes one observation per row. Data frames help summarize data and organize it into a format that is easy to read and use.

For example, the data frame below shows the “diamonds” dataset, which is one of the preloaded datasets in R. Each column contains a single variable that is related to diamonds: carat, cut, color, clarity, depth, and so on. Each row represents a single observation.



There are a few key things to keep in mind when you are working with data frames:

* + First, columns should be named.
  + Second, data frames can include many different types of data, like numeric, logical, or character.
  + Finally, elements in the same column should be of the same type.

You will learn more about data frames later on in the program, but this is a great starting point.

If you need to manually create a data frame in R, you can use the **data.frame()** function. The data.frame() function takes vectors as input. In the parentheses, enter the name of the column, followed by an equals sign, and then the vector you want to input for that column. In this example, the *x* column is a vector with elements 1, 2, 3, and the *y* column is a vector with elements 1.5, 5.5, 7.5.

**data.frame(x = c(1, 2, 3) , y = c(1.5, 5.5, 7.5))**

If you run the function, R displays the data frame in ordered rows and columns.

**x y**

**1  1 1.5**

**2  2 5.5**

**3  3 7.5**

In most cases, you won’t need to manually create a data frame yourself, as you will typically import data from another source, such as a .csv file, a relational database, or a software program.

Files

Let’s go over how to create, copy, and delete files in R. For more information on working with files in R, check out [R documentation: files](https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/files). **R documentation** is a tool that helps you easily find and browse the documentation of almost all R packages on CRAN. It’s a useful reference guide for functions in R code. Let’s go through a few of the most useful functions for working with files.

Use the **dir.create** function to create a new folder, or directory, to hold your files. Place the name of the folder in the parentheses of the function.

**dir.create ("destination\_folder")**

Use the **file.create()** function to create a blank file. Place the name and the type of the file in the parentheses of the function. Your file types will usually be something like .txt, .docx, or .csv.

**file.create (“new\_text\_file.txt”)**

**file.create (“new\_word\_file.docx”)**

**file.create (“new\_csv\_file.csv”)**

If the file is successfully created when you run the function, R will return a value of **TRUE** (if not, R will return **FALSE**).

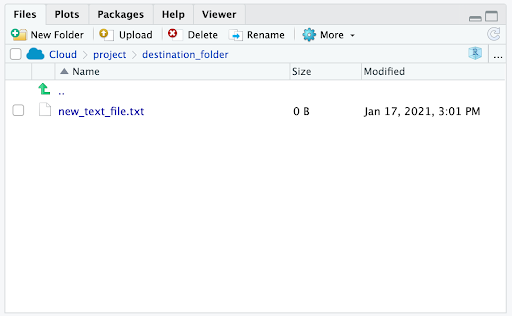
**file.create (“new\_csv\_file.csv”)**

**[1] TRUE**

Copying a file can be done using the **file.copy()** function. In the parentheses, add the name of the file to be copied. Then, type a comma, and add the name of the destination folder that you want to copy the file to.

**file.copy (“new\_text\_file.txt” , “destination\_folder”)**

If you check the Files pane in RStudio, a copy of the file appears in the relevant folder:



You can delete R files using the **unlink()** function. Enter the file’s name in the parentheses of the function.

**unlink (“some\_.file.csv”)**

Additional resource

If you want to learn more about working with data frames, matrices, and arrays in R, check out the [Data Wrangling](http://statseducation.com/Introduction-to-R/modules/getting data/data-wrangling/) section of Stat Education's Introduction to R course. The section includes modules on data frames, matrices, and arrays (and more), and each module contains helpful examples of key coding concepts.

--------------------------------------------------------------------------------------------------------------------------------------

Optional: Matrices

A **matrix** is a two-dimensional collection of data elements. This means it has both rows and columns. By contrast, a vector is a one-dimensional sequence of data elements. But like vectors, matrices can only contain a single data type. For example, you can’t have both logicals and numerics in a matrix.

To create a matrix in R, you can use the **matrix()** function. The matrix() function has two main arguments that you enter in the parentheses. First, add a vector. The vector contains the values you want to place in the matrix. Next, add at least one matrix dimension. You can choose to specify the number of rows or the number of columns by using the code **nrow =** or **ncol =**.

For example, imagine you want to create a 2x3 (two rows by three columns) matrix containing the values 3-8. First, enter a vector containing that series of numbers: **c(3:8)**. Then, enter a comma. Finally, enter **nrow = 2** to specify the number of rows.

**matrix(c(3:8), nrow = 2)**

If you run the function, R displays a matrix with three columns and two rows (typically referred to as a “2x3”) that contain the numeric values 3, 4, 5, 6, 7, 8. R places the first value (3) of the vector in the uppermost row, and the leftmost column of the matrix, and continues the sequence from left to right.

**[,1] [,2] [,3]**

**[1,]    3    5    7**

**[2,]    4    6    8**

You can also choose to specify the number of columns (**ncol =** ) instead of the number of rows (**nrow =** ).

**matrix(c(3:8), ncol = 2)**

When you run the function, R infers the number of rows automatically.

**[,1] [,2]**

**[1,]    3    6**

**[2,]    4    7**

**[3,]    5    8**

From <<https://www.coursera.org/learn/data-analysis-r/supplement/xEM9d/other-common-data-structures>>