The R-versus-Python debate

People often wonder which programming language they should learn first. You might be wondering about this, too. This certificate teaches the open-source programming language, R. R is a great starting point for foundational data analysis, and it has helpful packages that beginners can apply to projects. Python isn’t covered in the curriculum, but we encourage you to explore Python after completing the certificate. If you are curious about other programming languages, make every effort to continue learning.

Any language a beginner starts to learn will have some advantages and challenges. Let’s put this into context by looking at R and Python. The following table is a high-level overview based on a sampling of articles and opinions of those in the field. You can review the information without necessarily picking a side in the R vs. Python debate. In fact, if you check out RStudio’s blog article in the Additional resources section, it’s actually more about working together than winning a debate.

| **Languages** | **R** | **Python** |
| --- | --- | --- |
| **Common features** | - Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support | - Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support |
| **Unique advantages** | - Data manipulation, data visualization, and statistics packages - "Scalpel" approach to data: *find packages to do what you want with the data* | - Easy syntax for machine learning needs - Integrates with cloud platforms like Google Cloud, Amazon Web Services, and Azure |
| **Unique challenges** | - Inconsistent naming conventions make it harder for beginners to select the right functions - Methods for handling variables may be a little complex for beginners to understand | - Many more decisions for beginners to make about data input/output, structure, variables, packages, and objects - "Swiss army knife" approach to data: *figure out a way to do what you want with the data* |

**Additional resources**

For more information on comparing R and Python, refer to these resources:

* [R versus Python, a comprehensive guide for data professionals](https://medium.com/analytics-and-data/r-vs-python-a-comprehensive-guide-for-data-professionals-321e8dead598): This article is written by a data professional with extensive experience using both languages and provides a detailed comparison.
* [R versus Python, an objective comparison](https://www.dataquest.io/blog/python-vs-r/): This article provides a comparison of the languages using examples of code use.
* [R versus Python: What’s the best language for data science?](https://blog.rstudio.com/2019/12/17/r-vs-python-what-s-the-best-for-language-for-data-science/): This blog article provides RStudio’s perspective on the R vs. Python debate.

**Key takeaways**

Certain aspects make some programming languages easier to learn than others. But, that doesn’t make the harder languages impossible for beginners to learn. On the flip side, a programming language’s popularity doesn’t always make it the best language for beginners either.

R has been used by professionals who have a statistical or research-oriented approach to solving problems; among them are scientists, statisticians, and engineers. Python has been used by professionals looking for solutions in the data itself, those who must heavily mine data for answers; among them are data scientists, machine learning specialists, and software developers.

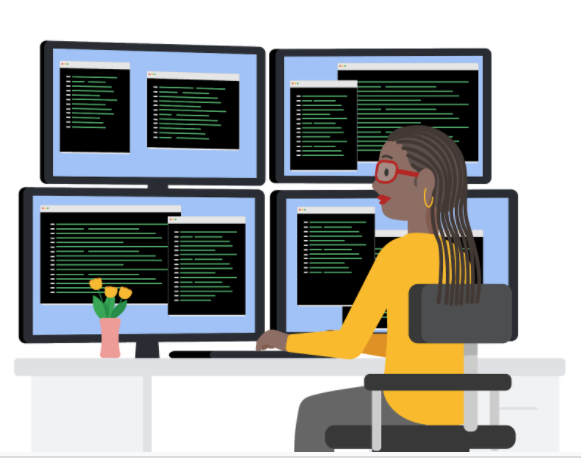
As you grow as a data analytics professional, you may need to learn additional programming languages. The skills and competencies you learn from your first programming experience are a good foundation. That's why this course focuses on the basics of R. You can develop the right perspective, that programming languages play an important part in the data analysis process no matter what job title you have.

The good news is that many of the concepts and coding principles that you will learn from using R in this course are transferable to other programming languages. You will also learn how to write R code in an Integrated Development Environment (IDE) called RStudio. RStudio allows you to manage projects that use R or Python, or even a combination of the two. Refer to [RStudio: A Single Home for R & Python](https://www.rstudio.com/solutions/r-and-python/) for more information. So, after you have worked with R and RStudio, learning Python or another programming language in the future will be more intuitive.

For a better idea of popular programming languages by job role, refer to [Ways to learn about programming](https://www.coursera.org/learn/data-analysis-r/supplement/y8zTf/ways-to-learn-about-programming). The programming languages most commonly used by data analysts, web designers, mobile and web application developers, and game developers are listed, along with links to resources to help you start learning more about those languages.

# Ways to learn about programming

Writing programming language code can be an exciting and rewarding experience. The programming field has a long history of people helping each other improve their skills and develop best practices. You will focus on the R programming language in this course, but in the future you might choose to pursue additional programming languages based on your interests and professional goals. This reading is a general guide to help you decide which programming languages are best suited for you.



## Popular programming languages by profession

Let’s go through some potential job titles you might encounter and the most popular programming languages used in those professions. Also included is a list of additional resources for you to explore and learn more about each of the programming languages introduced.

### **Data analyst**

A data analyst collects, transforms, and organizes data to draw conclusions, make predictions, and drive informed decision-making. The most popular programming languages used by data analysts are R and Python.

**R** offers convenient statistical features for data analysis and is useful for creating advanced data visualizations. Check out these resources to learn more about R:

* [The R Project for Statistical Computing](https://www.r-project.org/): a website for downloading R, documentation, and help
* [R Manuals](https://cran.r-project.org/manuals.html): links to manuals from the R core team, including introduction, administration, and help
* [Coding Club R Tutorials](https://ourcodingclub.github.io/tutorials.html): a collection of coding tutorials for R
* [R for Beginners](https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf): a starting guide to help you work with data, graphics, and statistics in R

**Python** is a general-purpose language that you can use to create what you need for data analysis. Here are a few resources to begin learning Python:

* [The Python Software Foundation (PSF)](https://www.python.org/about/gettingstarted/): a website with guides to help you get started as a beginner
* [Python Tutorial](https://docs.python.org/3/tutorial/): a Python 3 tutorial from the PSF site
* [Coding Club Python Tutorials](https://ourcodingclub.github.io/tutorials.html): a collection of coding tutorials for Python

### **Web designer**

A web designer is responsible for the styling and layout of web pages containing text, graphics, and video. Web designers generally use Hypertext Markup Language v5 (HTML5) and Cascading Style Sheets (CSS) to create web pages.

**HTML5** provides structure for web pages and is used to connect to hosting platforms. Learn more about HTML5 and CSS using these resources:

* [HTML Tutorial](https://www.tutorialrepublic.com/html-tutorial/): an introduction to HTML with links to HTML5 features, examples, and references
* [HTML5 Cheat Sheet](https://www.wpkube.com/html5-cheat-sheet/): a handy summary of HTML5 tags, attributes, and compatibility with HTML4
* [HTML5 and CSS Fundamentals course](https://www.edx.org/course/html5-and-css-fundamentals): a free W3C course on edX; a verified course certificate can be issued for $199

**CSS** is used for web page design and controls graphic elements (color, layout, and font) and page presentation on multiple devices (large screens, mobile screens, and printers). Check out these cheat sheets for CSS:

* [Interactive CSS Cheat Sheet](https://htmlcheatsheet.com/css/): includes the most common CSS snippets for gradient, background, font-family, border, and much more
* [50 Best HTML & CSS Cheat Sheets](https://sharethis.com/best-practices/2020/02/best-html-and-css-cheat-sheets/): a list of 50 cheat sheets–choose a few that are useful to you

### **Mobile application developer**

A mobile application developer uses programming to create applications used on laptops, mobile phones, and tablets. The most popular programming languages for mobile application developers are Swift, Java, and C#.

**Swift** (for Apple platforms) is an open source scripting language for macOS, iOS, watchOS, and tvOS. Its main goal is to make applications run faster. Browse these resources for more information about Swift:

* [Swift.org](https://swift.org/about/): an open source community with resources to learn how to use Swift, including videos and sample code
* [Swift developer site](https://developer.apple.com/swift/): an Apple developer website with information for developers who want to use Swift
* [Swift development resources](https://developer.apple.com/swift/resources/): Apple’s collection of documentation, sample code, videos, and recommended books

**Java** (for Android devices) is the official language for Android development. The article [I want to develop Android apps - which languages should I learn?](https://www.androidauthority.com/develop-android-apps-languages-learn-391008/) explores some other languages used for Android development. Check out these resources for Java:

* [Android Studio](https://developer.android.com/studio): a downloadable integrated development environment (IDE) with tools to build apps for Android devices
* [Build your first Android app in Java](https://developer.android.com/codelabs/build-your-first-android-app#1): instructions for installing Android Studio and creating your first app
* [Java tutorial for beginners: write a simple app with no previous experience](https://www.androidauthority.com/java-tutorial-for-beginners-write-a-simple-app-with-no-previous-experience-1121975/): an overview of how to learn Java, with examples

**C#** (pronounced C-sharp) is an object-oriented programming language that is widely used to create mobile apps in the .NET open source developer platform. Xamarin extends the .NET platform with a framework for developers to create cross-platform mobile apps for both iOS and Android. Here are a few resources to help you learn C#:

* [Microsoft .NET learning materials for C#](https://dotnet.microsoft.com/learn/csharp): includes free courses, tutorials, and videos to learn the programming language C#
* [Microsoft Xamarin learning materials](https://dotnet.microsoft.com/learn/xamarin): includes free courses, tutorials, and videos to learn about mobile development with Xamarin
* [Xamarin Tutorial - build your first iOS or Android app in C#](https://dotnet.microsoft.com/learn/xamarin/hello-world-tutorial/intro): instructions for building a mobile app that displays the text “Hello World”
* [Learn C# from Codecademy](https://www.codecademy.com/learn/learn-c-sharp): a website with free basic interactive lessons, and additional activities that can be accessed with a monthly subscription

### **Web application developer**

A web application developer designs and develops network applications used across the web. The most popular programming languages used by web application developers are Java, Python, Ruby, and PHP.

**Java** is widely used to create enterprise web applications that can run on multiple clients. Java’s main strength is its “Write Once, Run Anywhere” (WORA) approach.Browse these resources to learn more about Java:

* [Oracle Java Tutorials](https://docs.oracle.com/javase/tutorial/): Java tutorials from Oracle documentation
* [Java for Beginners](https://www.homeandlearn.co.uk/java/java.html): a free Java course for beginners from the website “Home and Learn”

**Python** is a general-purpose programming language. Check out the Python resources listed in the data analyst section.

**Ruby** is a general-purpose, object-oriented programming language used for web application development. Ruby isn't the same as Ruby on Rails, which is an open source web application framework that runs using Ruby. Browse these resources to learn more about Ruby:

* [Ruby news](http://ruby-doc.org/): information about the latest Ruby releases and links to other resources
* [Ruby documentation](http://www.ruby-lang.org/en/documentation/): includes guides, tutorials, and reference material to help you learn more about Ruby
* [Ruby programmer’s guide](http://ruby-doc.com/docs/ProgrammingRuby/): a tutorial and reference guide for Ruby
* [Learn Ruby from Codecademy](https://www.codecademy.com/learn/learn-ruby): a website with free basic interactive lessons, and additional activities that can be accessed with a monthly subscription

**PHP** is a scripting language particularly suited for web application development. It was based on Perl, another programming language. PHP is simple, flexible, and relatively easy to learn. Check out these resources to learn more about PHP:

* [PHP downloads and documentation](https://www.php.net/): information about the latest PHP releases and links to other resources
* [PHP the Right Way](https://phptherightway.com/): a quick reference for popular PHP coding standards
* [Interactive PHP tutorial](https://www.learn-php.org/): a free tutorial that runs PHP code in exercises

### **Game developer**

A game developer is an application developer who specializes in video game creation. Game developers most commonly use the programming languages C# and C++.

**C#** is an object-oriented programming language that is widely used to create games. Check out the C# resources listed in the mobile application developer section.

**C++** is an extension of the C programming language that is also used to create console games, like those for Xbox. Browse more information about C++:

* [Microsoft resources for C++](https://docs.microsoft.com/en-us/cpp/?view=msvc-160): learn how to install the Visual Studio IDE and write C++ code
* [Microsoft C++ and C# code samples for gaming](https://docs.microsoft.com/en-us/samples/browse/?languages=cpp&terms=gaming): a resource with over 40 C++ and C# code samples for gaming
* [Interactive C++ tutorial](https://www.learn-cpp.org/): a free tutorial that runs C++ code in exercises

## Tips for learning programming languages

Here are a few tips to follow when you start learning a new programming language:

* Define a practice project and use the language to help you complete it. This makes the learning process more practical and engaging.
* Keep previous concepts and coding principles in mind. Many of these are transferable between programming languages. So, after you have learned one language, learning a second or third programming language tends to be much easier.
* Create and keep good notes and cheat sheets in whatever format (handwritten or typed) that works best for you.
* Create an online filing system for information that you can easily access while you work in various programming environments.

From spreadsheets to SQL to R

Although the programming language R might be new to you, it actually has a lot of similarities to the other tools you have explored in this program. In this reading, you will compare spreadsheet programs, SQL, and R to have a better sense of how to use each moving forward.



**Spreadsheets, SQL, and R: a comparison**

As a data analyst, there is a good chance you will work with SQL, R, and spreadsheets at some point in your career. Each tool has its own strengths and weaknesses, but they all make the data analysis process smoother and more efficient. There are two main things that all three have in common:

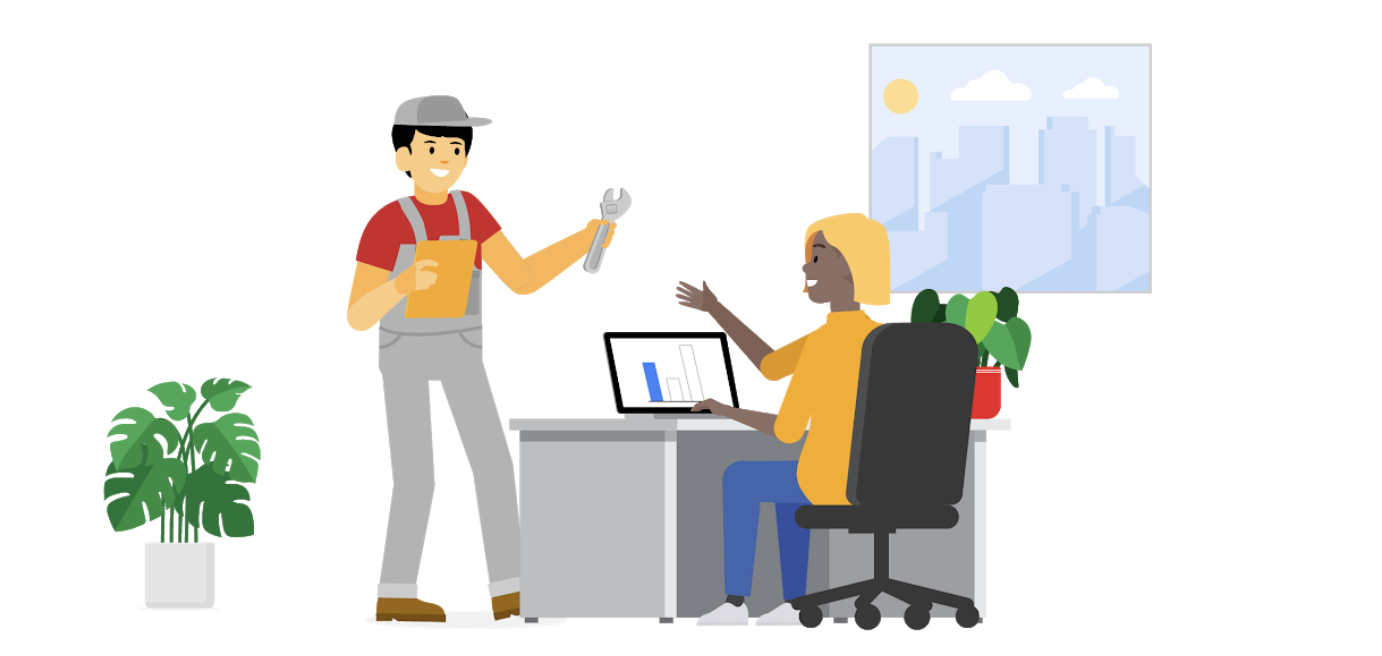
* **They all use filters:** for example, you can easily filter a dataset using any of these tools. In R, you can use the filter function. This performs the same task as a basic SELECT-FROM-WHERE SQL query. In a spreadsheet, you can create a filter using the menu options.
* **They all use functions:** In spreadsheets, you use functions in formulas, and in SQL, you include them in queries. In R, you will use functions in the code that is part of your analysis.

The table below presents key questions to explore a few more ways that these tools compare to each other. You can use this as a general guide as you begin to navigate R.

| **Key question** | **Spreadsheets** | **SQL** | **R** |
| --- | --- | --- | --- |
| **What is it?** | A program that uses rows and columns to organize data and allows for analysis and manipulation through formulas, functions, and built-in features | A database programming language used to communicate with databases to conduct an analysis of data | A general purpose programming language used for statistical analysis, visualization, and other data analysis |
| **W​hat is a primary advantage?** | I​ncludes a variety of visualization tools and features | A​llows users to manipulate and reorganize data as needed to aid analysis | P​rovides an accessible language to organize, modify, and clean data frames, and create insightful data visualizations |
| **Which datasets does it work best with?** | Smaller datasets | Larger datasets | Larger datasets |
| **What is the source of the data?** | Entered manually or imported from an external source | Accessed from an external database | Loaded with R when installed, imported from your computer, or loaded from external sources |
| **Where is the data from my analysis usually stored?** | In a spreadsheet file on your computer | Inside tables in the accessed database | In an R file on your computer |
| **Do I use formulas and functions?** | Yes | Yes | Yes |
| **Can I create visualizations?** | Yes | Yes, by using an additional tool like a database management system (DBMS) or a business intelligence (BI) tool | Yes |

When to use RStudio

As a data analyst, you will have plenty of tools to work with in each phase of your analysis. Sometimes, you will be able to meet your objectives by working in a spreadsheet program or using SQL with a database. In this reading, you will go through some examples of when working in R and RStudio might be your better option instead.



**Why RStudio?**

One of your core tasks as an analyst will be converting raw data into insights that are accurate, useful, and interesting. That can be tricky to do when the raw data is complex. R and RStudio are designed to handle large data sets, which spreadsheets might not be able to handle as well. RStudio also makes it easy to reproduce your work on different datasets. When you input your code, it's simple to just load a new dataset and run your scripts again. You can also create more detailed visualizations using RStudio.

**When RStudio truly shines**

When the data is spread across multiple categories or groups, it can be challenging to manage your analysis, visualize trends, and build graphics. And the more groups of data that you need to work with, the harder those tasks become. That’s where RStudio comes in.

For example, imagine you are analyzing sales data for every city across an entire country. That is a lot of data from a lot of different groups–in this case, each city has its own group of data.

Here are a few ways RStudio could help in this situation:

* Using RStudio makes it easy to take a specific analysis step and perform it for each group using basic code. In this example, you could calculate the yearly average sales data for every city.
* RStudio also allows for flexible data visualization. You can visualize differences across the cities effectively using plotting features like facets–which you’ll learn more about later on.
* You can also use RStudio to automatically create an output of summary stats—or even your visualized plots—for each group.

As you learn more about R and RStudio moving forward in this program, you’ll get a better understanding of when RStudio should be your data analysis tool of choice.

**For more information**

* [**The Advantages of RStudio**](https://www.theanalysisfactor.com/the-advantages-of-rstudio/): This web page explains some of the reasons why RStudio is many analysts’ preferred choice for interfacing with R. You’ll learn about the advantages of using RStudio for data analysis, from ease of use to accessibility of graphics and more.
* [**Data analysis and R programming**](https://lgatto.github.io/2017_11_09_Rcourse_Jena/before-we-start.html): This online introduction to data analysis and R programming is a good starting point for R and RStudio users. It also includes a list of detailed explanations about the advantages of using R and RStudio. You’ll also find a helpful guide for getting set up with RStudio.

Connecting with other analysts in the R community

R is a powerful tool in your data analysis toolkit–and it also has a powerful community of users who are excited to share, collaborate, and connect with others. This reading will give you a few places where you can start to connect, online and in-person, with other analysts in the R community.



**Online communities**

Online communities allow you to connect with other R users no matter where you live. This list includes forums and discussion channels where you can join the conversation. It also includes social media tags you can use on your existing social media platforms to connect with other data analysts.

* [**RStudio Community:**](https://community.rstudio.com/) The RStudio Community forum is a great place to get help and find solutions to challenges you have with R–and maybe help someone else out, too!
* [**r/RLanguage**](https://www.reddit.com/r/Rlanguage/): The R language subreddit is an active online community on the social media platform Reddit, where R users go to discuss R, ask questions, and share tips.
* [**rOpenSci**](https://discuss.ropensci.org/): rOpenSci has a community forum where R users can ask questions and search for solutions. It also includes links to their Best Practices guide and support pages.
* [**R4DS Online Learning Community and Slack channel:**](https://www.rfordatasci.com/) This is a community with another Slack channel where R learners and mentors can gather and connect. This is a great place to chat about using R for data science.
* [**Twitter #rstats**](https://twitter.com/hashtag/rstats?lang=en): If you use Twitter, you can connect with other R users using the hashtag #rstats; a lot of R developers and analysts are active on Twitter.

**Meetups**

Many organizations host both in-person and online meetups for R users; you should always practice caution and be safe whenever attending meetups in-person.

* [**Local Data Analytics meetups:**](https://www.meetup.com/topics/data-analytics/) These meetups are a great way to meet other people who are interested in data analytics and build your network. These meetups are location-based, so you can connect with other data analysts in your area.
* [**R User Groups:**](https://jumpingrivers.github.io/meetingsR/r-user-groups.html)This list contains links to regional R communities, including subreddits and meetup groups. This is a useful resource if you are interested in finding R users in your area.
* [**RLadies Meetups:**](https://www.meetup.com/pro/rladies) These are in-person and virtual meetups specifically for R enthusiasts who identify as underrepresented or marginalized. These meetups are also location-based and can help you connect with other data analysts in your area.

R can be tricky to learn, but luckily there is a strong community of R users who are interested in working together and helping each other out. These resources are a good starting point if you want to begin connecting with the larger data analyst community, so take advantage of them!

Vectors and lists in R

*You can save this reading for future reference. Feel free to download a PDF version of this reading below:*

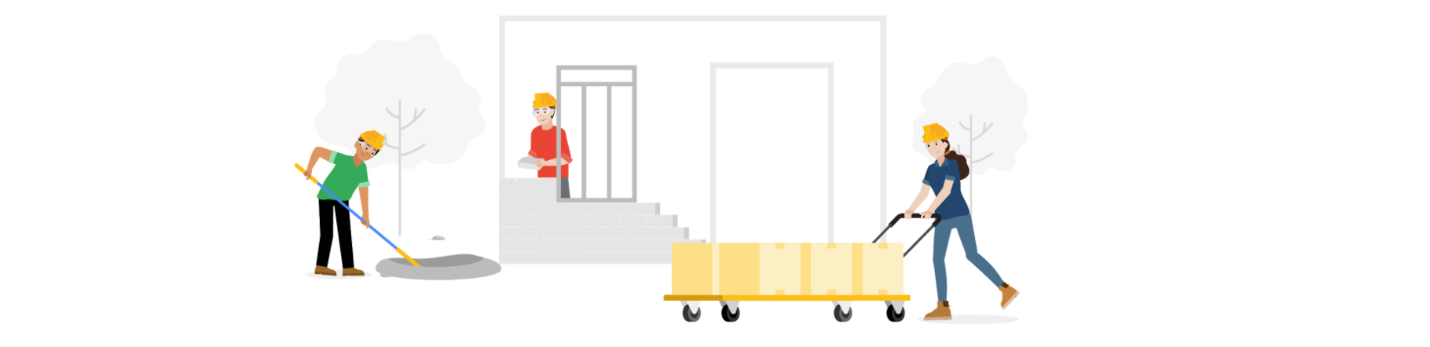
**Vectors and lists in R.pdf**PDF File

[Open file](https://d3c33hcgiwev3.cloudfront.net/b5iOOC7FSPKYjjguxYjyqQ_c3961931ea1d422da08da3ca484beff1_Vectors-and-lists-in-R.pdf?Expires=1648080000&Signature=IZDpRtyHd3Gj7yGIP2u1b1Q8q4JUNWp7-8qlvh4VNEPJ9kC6VDpp13aINX3pF08Jy6HDudQfdTo69QVdxFgk19LihLjcD034Sxvqkr8bBOt0rswrbGc8yZkOTI-WMKtOO4oOW-OU3hq~KM~13~iXW8DErkNSwasd2xlNfBZPyjE_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

In programming, a **data structure** is a format for organizing and storing data. Data structures are important to understand because you will work with them frequently when you use R for data analysis. The most common data structures in the R programming language include:

* Vectors
* Data frames
* Matrices
* Arrays

Think of a data structure like a house that contains your data.



This reading will focus on vectors. Later on, you’ll learn more about data frames, matrices, and arrays.

There are two types of vectors: **atomic vectors** and **lists**. Coming up, you’ll learn about the basic properties of atomic vectors and lists, and how to use R code to create them.

**Atomic vectors**

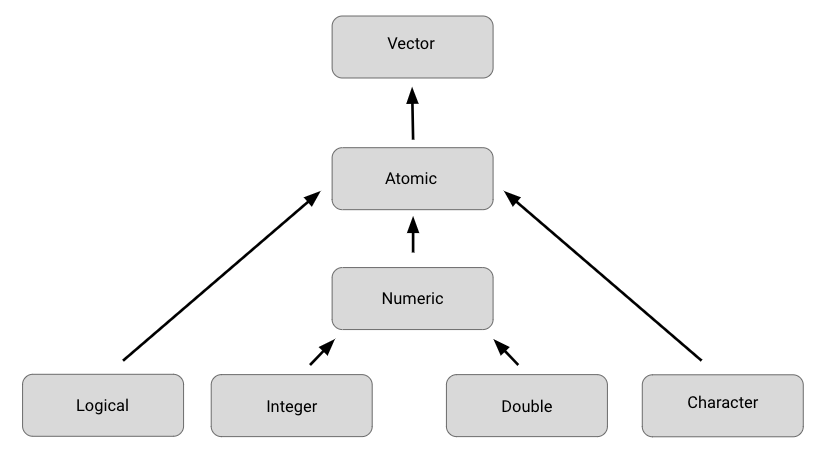
First, we will go through the different types of atomic vectors. Then, you will learn how to use R code to create, identify, and name the vectors.

Earlier, you learned that a **vector** is a group of data elements of the *same* type, stored in a sequence in R. You cannot have a vector that contains both logicals and numerics.

There are six primary types of atomic vectors: logical, integer, double, character (which contains strings), complex, and raw. The last two–complex and raw–aren’t as common in data analysis, so we will focus on the first four. Together, integer and double vectors are known as numeric vectors because they both contain numbers. This table summarizes the four primary types:

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| Logical | True/False | **TRUE** |
| Integer | Positive and negative whole values | **3** |
| Double | Decimal values | **101.175** |
| Character | String/character values | **“Coding”** |

This diagram illustrates the hierarchy of relationships among these four main types of vectors:

Bottom: logical (arrow points to atomic), integer (arrow points to numeric), double (arrow points to numeric), character (arrow points to atomic) Second to bottom: numeric (arrow points to atomic) second level: atomic (arrow points to vector) top: vector

**Creating vectors**

One way to create a vector is by using the **c()** function (called the “combine” function). The c() function in R combines multiple values into a vector. In R, this function is just the letter “c” followed by the values you want in your vector inside the parentheses, separated by a comma: c(x, y, z, …).

For example, you can use the c() function to store numeric data in a vector.

**c(2.5, 48.5, 101.5)**

To create a vector of integers using the c() function, you must place the letter "L" directly after each number.

**c(1L, 5L, 15L)**

You can also create a vector containing characters or logicals.

**c(“Sara” , “Lisa” , “Anna”)**

**c(TRUE, FALSE, TRUE)**

**Determining the properties of vectors**

Every vector you create will have two key properties: type and length.

 You can determine what type of vector you are working with by using the **typeof()** function. Place the code for the vector inside the parentheses of the function. When you run the function, R will tell you the type. For example:

**typeof(c(“a” , “b”))**

**#> [1] "character"**

 Notice that the output of the typeof function in this example is **“character”**. Similarly, if you use the typeof function on a vector with integer values, then the output will include **“integer”** instead:

**typeof(c(1L , 3L))**

**#> [1] "integer"**

You can determine the length of an existing vector–meaning the number of elements it contains–by using the **length()** function. In this example, we use an assignment operator to assign the vector to the variable *x*. Then, we apply the length() function to the variable. When we run the function, R tells us the length is **3**.

**x <- c(33.5, 57.75, 120.05)**

**length(x)**

**#> [1] 3**

You can also check if a vector is a specific type by using an **is** function: **is.logical(), is.double(), is.integer(), is.character()**. In this example, R returns a value of **TRUE** because the vector contains integers.

**x <- c(2L, 5L, 11L)**

**is.integer(x)**

**#> [1] TRUE**

In this example, R returns a value of **FALSE** because the vector does *not* contain characters, rather it contains logicals.

**y <- c(TRUE, TRUE, FALSE)**

**is.character(y)**

**#> [1] FALSE**

**Naming vectors**

All types of vectors can be named. Names are useful for writing readable code and describing objects in R. You can name the elements of a vector with the **names()** function. As an example, let’s assign the variable x to a new vector with three elements.

**x <- c(1, 3, 5)**

You can use the names() function to assign a different name to each element of the vector.

**names(x) <- c("a", "b", "c")**

Now, when you run the code, R shows that the first element of the vector is named **a**, the second **b**, and the third **c**.

**x**

**#> a b c**

**#> 1 3 5**

Remember that an atomic vector can only contain elements of the same type. If you want to store elements of different types in the same data structure, you can use a list.

**Creating lists**

**Lists** are different from atomic vectors because their elements can be of any type—like dates, data frames, vectors, matrices, and more. Lists can even contain other lists.

You can create a list with the **list()** function. Similar to the c() function, the list() function is just **list** followed by the values you want in your list inside parentheses: **list(x, y, z, …)**. In this example, we create a list that contains four different kinds of elements: character (**"a"**), integer (**1L**), double (**1.5**), and logical (**TRUE**).

**list("a", 1L, 1.5, TRUE)**

Like we already mentioned, lists can contain other lists. If you want, you can even store a list inside a list inside a list—and so on.

**list(list(list(1 , 3, 5)))**

**Determining the structure of lists**

If you want to find out what types of elements a list contains, you can use the **str()** function. To do so, place the code for the list inside the parentheses of the function. When you run the function, R will display the data structure of the list by describing its elements and their types.

Let’s apply the str() function to our first example of a list.

**str(list("a", 1L, 1.5, TRUE))**

We run the function, then R tells us that the list contains four elements, and that the elements consist of four different types: character (**chr**), integer (**int**), number (**num**), and logical (**logi**).

**#> List of 4**

**#>  $ : chr "a"**

**#>  $ : int 1**

**#>  $ : num 1.5**

**#>  $ : logi TRUE**

Let’s use the str() function to discover the structure of our second example.  First, let’s assign the list to the variable *z* to make it easier to input in the str() function.

**z <- list(list(list(1 , 3, 5)))**

Let’s run the function.

**str(z)**

**#> List of 1**

**#>  $ :List of 1**

**#>   ..$ :List of 3**

**#>   .. ..$ : num 1**

**#>   .. ..$ : num 3**

**#>   .. ..$ : num 5**

The indentation of the **$** symbols reflect the nested structure of this list. Here, there are three levels (so there is a list within a list within a list).

**Naming lists**

Lists, like vectors, can be named. You can name the elements of a list when you first create it with the list() function:

**list('Chicago' = 1, 'New York' = 2, 'Los Angeles' = 3)**

**$Chicago**

**[1] 1**

**$`New York`**

**[1] 2**

**$`Los Angeles`**

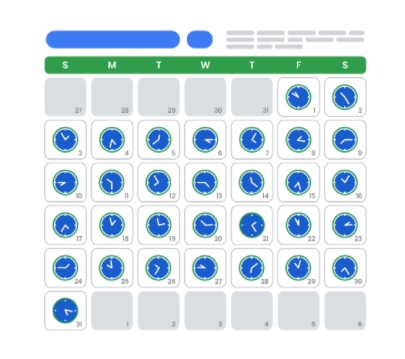
**[1] 3**

**Additional resource**

To learn more about vectors and lists, check out [R for Data Science, Chapter 20: Vectors](https://r4ds.had.co.nz/vectors.html#vectors). R for Data Science is a classic resource for learning how to use R for data science and data analysis. It covers everything from cleaning to visualizing to communicating your data. If you want to get more details about the topic of vectors and lists, this chapter is a great place to start.

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**

T​his 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:

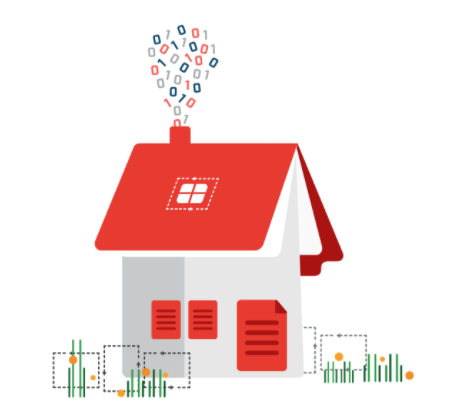
* [lubridate.tidyverse](https://lubridate.tidyverse.org/index.html): This is the “lubridate” entry from the official tidyverse documentation, which offers a comprehensive reference guide to the various tidyverse packages. Check out this link for an overview of key concepts and functions.
* [Dates and 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.

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**

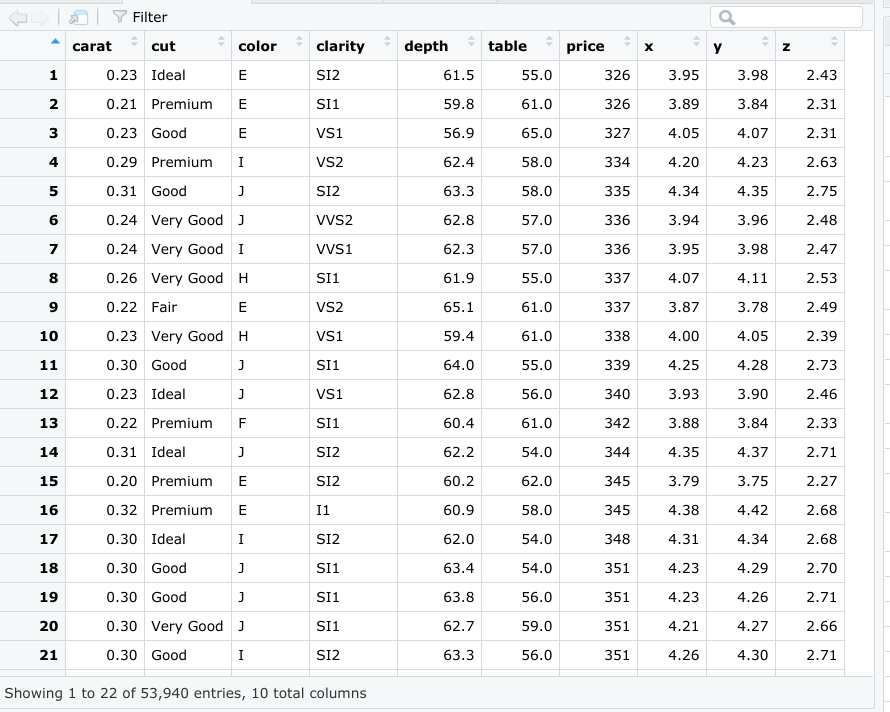
R​ecall that a data structure is like a house that contains your data.



**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**).

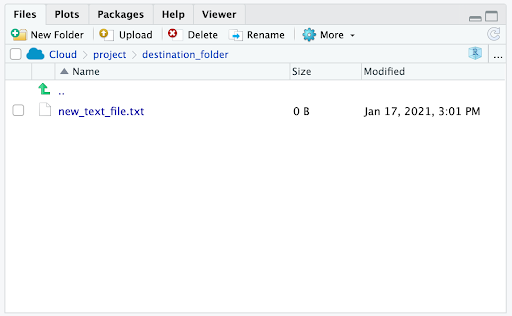
**file.create (“new\_csv\_file.csv”)**

**[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%20data/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**

Logical operators and conditional statements

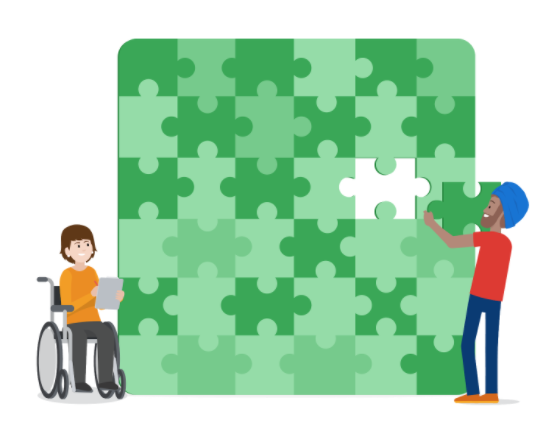
**Tip:** You may refresh on the concepts presented in [Understanding Boolean logic](https://www.coursera.org/learn/data-preparation/supplement/GgZMN/understanding-boolean-logic) to help you understand how logical operators work.

*You can save this reading for future reference. Feel free to download a PDF version of this reading below:*

**Logical operators and conditional statements.pdf**PDF File

[Open file](https://d3c33hcgiwev3.cloudfront.net/IH3jvjscStK94747HNrSCg_9681cba255d44707b891ea5e0eb0e2f1_Logical-operators-and-conditional-statements.pdf?Expires=1648080000&Signature=bpNmKb5-To8dmm-SdJx1ZOqGeZklgJM73yU6~6bBQhMIehz9y-JoiP1e0ED-RGjblgPWS3k4dnWzye4eIbzQ9UZCuhSzSl6zja9hRlmY5dU8AqQN6W3AF4UpELv-4ur8JYUdZYA9vKY1~CLmZa2jGzhjGovvRfchRMvQGaSjMa8_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

Earlier, you learned that an **operator** is a symbol that identifies the type of operation or calculation to be performed in a formula. In this reading, you will learn about the main types of logical operators and how they can be used to create conditional statements in R code.



**Logical operators**

**Logical operators** return a logical data type such as TRUE or FALSE.

There are three primary types of logical operators:

* AND (sometimes represented as & or && in R)
* OR (sometimes represented as | or || in R)
* NOT (!)

Review the summarized logical operators below.

**AND operator “&”**

* The AND operator takes two logical values. It returns **TRUE** only if both individual values are TRUE. This means that TRUE & TRUE evaluates to **TRUE**. However, FALSE & TRUE, TRUE & FALSE, and FALSE & FALSE all evaluate to **FALSE**.
* If you run the corresponding code in R, you get the following results: **> TRUE & TRUE [1] TRUE > TRUE & FALSE [1] FALSE > FALSE & TRUE [1] FALSE > FALSE & FALSE [1] FALSE** You can illustrate this using the results of our comparisons. Imagine you create a variable x that is equal to 10.  **x <- 10** To check if x is greater than 3 but less than 12, you can use x > 3 and x < 12 as the values of an “AND” expression.  **x > 3 & x < 12** When you run the function, R returns the result TRUE. **[1] TRUE** The first part, **x > 3** will evaluate to **TRUE** since 10 is greater than 3. The second part, **x < 12** will also evaluate to **TRUE** since 10 is less than 12. So, since both values are TRUE, the result of the AND expression is **TRUE**. The number 10 lies between the numbers 3 and 12.  However, if you make x equal to 20, the expression **x > 3 & x < 12** will return a different result.  **x <- 20 x > 3 & x < 12 [1] FALSE** Although **x > 3** is **TRUE** (20 > 3), **x < 12** is **FALSE** (20 < 12). If one part of an AND expression is FALSE, the entire expression is FALSE (TRUE & FALSE = FALSE). So, R returns the result **FALSE**.

**OR operator “|”**

* The OR operator (|) works in a similar way to the AND operator (&). The main difference is that at least one of the values of the OR operation must be TRUE for the entire OR operation to evaluate to **TRUE**. This means that TRUE | TRUE, TRUE | FALSE, and FALSE | TRUE all evaluate to **TRUE**. When both values are FALSE, the result is **FALSE**.
* If you write out the code, you get the following results:  **> TRUE | TRUE [1] TRUE > TRUE | FALSE [1] TRUE > FALSE | TRUE [1] TRUE > FALSE | FALSE [1] FALSE** For example, suppose you create a variable y equal to 7. To check if y is less than 8 or greater than 16, you can use the following expression: **y <- 7 y < 8 | y > 16** The comparison result is TRUE (7 is less than 8) | FALSE (7 is not greater than 16). Since only one value of an OR expression needs to be TRUE for the entire expression to be TRUE, R returns a result of TRUE.  **[1] TRUE** Now, suppose y is 12. The expression y < 8 | y > 16 now evaluates to FALSE (12 < 8) | FALSE (12 > 16). Both comparisons are FALSE, so the result is **FALSE**. **y <- 12 y < 8 | y > 16 [1] FALSE**

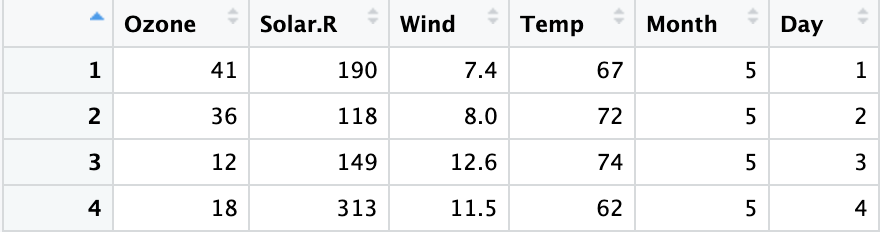
**NOT operator “!”**

* The NOT operator (!) simply negates the logical value it applies to. In other words, !TRUE evaluates to **FALSE**, and !FALSE evaluates to **TRUE**.
* When you run the code, you get the following results:  **> !TRUE  [1] FALSE > !FALSE [1] TRUE** Just like the OR and AND operators, you can use the NOT operator in combination with logical operators. Zero is considered FALSE and non-zero numbers are taken as TRUE. The NOT operator evaluates to the opposite logical value.  Let’s imagine you have a variable x that equals 2:  **x <- 2** The NOT operation evaluates to FALSE because it takes the opposite logical value of a non-zero number (TRUE).  **> !x [1] FALSE**

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

Let’s check out an example of how you might use logical operators to analyze data. Imagine you are working with the *airquality* dataset that is preloaded in RStudio. It contains data on daily air quality measurements in New York from May to September of 1973.

The data frame has six columns: *Ozone* (the ozone measurement), *Solar.R* (the solar measurement), *Wind* (the wind measurement), *Temp* (the temperature in Fahrenheit), and the *Month* and *Day* of these measurements (each row represents a specific month and day combination).



Let’s go through how the AND, OR, and NOT operators might be helpful in this situation.

**AND example**

Imagine you want to specify rows that are extremely sunny and windy, which you define as having a *Solar* measurement of over 150 anda *Wind* measurement of over 10.

In R, you can express this logical statement as **Solar.R > 150 & Wind > 10**.

Only the rows where *both* of these conditions are true fulfill the criteria:

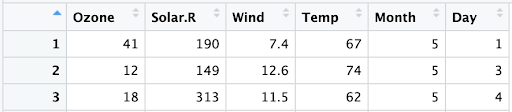
Image of a single row of the “airquality” dataset in the RStudio data viewer.

**OR example**

Next, imagine you want to specify rows where it’s extremely sunny or it’s extremely windy, which you define as having a *Solar* measurement of over 150 or a *Wind* measurement of over 10.

In R, you can express this logical statement as **Solar.R > 150 | Wind > 10**.

All the rows where *either* of these conditions are true fulfill the criteria:

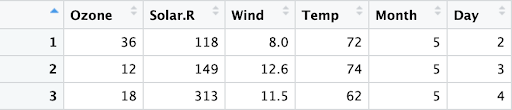


**NOT example**

Now, imagine you just want to focus on the weather measurements for days that aren't the first day of the month.

In R, you can express this logical statement as **Day != 1**.

The rows where this condition is true fulfill the criteria:



Finally, imagine you want to focus on scenarios that aren't extremely sunny and not extremely windy, based on your previous definitions of extremely sunny and extremely windy. In other words, the following statement should not be true: either a *Solar* measurement greater than 150 or a *Wind* measurement greater than 10.

Notice that this statement is the opposite of the OR statement used above. To express this statement in R, you can put an exclamation point (!) in front of the previous OR statement: **!(Solar.R > 150 | Wind > 10)**. R will apply the NOT operator to everything within the parentheses.

In this case, only one row fulfills the criteria:



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

**Optional: Conditional statements**

A **conditional statement** is a declaration that if a certain condition holds, then a certain event must take place. For example, “*If* the temperature is above freezing, *then* I will go outside for a walk.” If the first condition is true (the temperature is above freezing), then the second condition will occur (I will go for a walk). Conditional statements in R code have a similar logic.

Let’s discuss how to create conditional statements in R using three related statements:

* **if()**
* **else()**
* **else if()**

**if statement**

The **if** statement sets a condition, and if the condition evaluates to **TRUE**, the R code associated with the if statement is executed.

In R, you place the code for the condition inside the parentheses of the if statement. The code that has to be executed if the condition is TRUE follows in curly braces (**expr**). Note that in this case, the second curly brace is placed on its own line of code and identifies the end of the code that you want to execute.

**if (condition) {**

**expr**

**}**

For example, let’s create a variable *x* equal to 4.

**x <- 4**

Next, let’s create a conditional statement: if *x* is greater than 0, then R will print out the string **“x is a positive number".**

**if (x > 0) {**

**print("x is a positive number")**

**}**

Since x = 4, the condition is true (4 > 0). Therefore, when you run the code, R prints out the string **“x is a positive number"**.

**[1] "x is a positive number"**

But if you change x to a negative number, like -4, then the condition will be FALSE (-4 > 0). If you run the code, R will not execute the print statement. Instead, a blank line will appear as the result.

**else statement**

The **else** statement is used in combination with an if statement. This is how the code is structured in R:

**if (condition) {**

**expr1**

**} else {**

**expr2**

**}**

The code associated with the else statement gets executed whenever the condition of the if statement is *not* TRUE. In other words, if the condition is TRUE, then R will execute the code in the if statement (*expr1*); if the condition is *not* TRUE, then R will execute the code in the else statement (*expr2*).

Let’s try an example. First, create a variable *x* equal to 7.

**x <- 7**

Next, let’s set up the following conditions:

* If x is greater than 0, R will print **“x is a positive number”**.
* If x is less than or equal to 0, R will print **“x is either a negative number or zero”**.

In our code, the first condition (x > 0) will be part of the if statement. The second condition of x less than or equal to 0 is implied in the else statement. If x > 0, then R will print **“x is a positive number”**. Otherwise, R will print **“x is either a negative number or zero”**.

**x <- 7**

**if (x > 0) {**

**print ("x is a positive number")**

**} else {**

**print ("x is either a negative number or zero")**

**}**

Since 7 is greater than 0, the condition of the if statement is true. So, when you run the code, R prints out **“x is a positive number”**.

**[1] "x is a positive number"**

But if you make x equal to -7, the condition of the if statement is *not* true (-7 is not greater than 0). Therefore, R will execute the code in the else statement. When you run the code, R prints out **“x is either a negative number or zero”**.

**x <- -7**

**if (x > 0) {**

**print("x is a positive number")**

**} else {**

**print ("x is either a negative number or zero")**

**}**

**[1] "x is either a negative number or zero"**

**else if statement**

In some cases, you might want to customize your conditional statement even further by adding the **else if** statement. The else if statement comes in between the if statement and the else statement. This is the code structure:

**if (condition1) {**

**expr1**

**} else if (condition2) {**

**expr2**

**} else {**

**expr3**

**}**

If the if condition (*condition1*) is met, then R executes the code in the first expression (*expr1*). If the if condition is not met, and the else if condition (*condition2*) is met, then R executes the code in the second expression (*expr2*). If neither of the two conditions are met, R executes the code in the third expression (*expr3*).

In our previous example, using only the if and else statements, R can only print **“x is either a negative number or zero”** if x equals 0 or x is less than zero. Imagine you want R to print the string **“x is zero”** if x equals 0. You need to add another condition using the else if statement.

Let’s try an example. First, create a variable *x* equal to negative 1 (“-1”).

**x <- -1**

Now, you want to set up the following conditions:

* If x is less than 0, print **“x is a negative number”**
* If x equals 0, print **“x is zero”**
* Otherwise, print **“x is a positive number”**

In the code, the first condition will be part of the if statement, the second condition will be part of the else if statement, and the third condition will be part of the else statement. If x < 0, then R will print **“x is a negative number”.** If x = 0, then R will print **“x is zero”**. Otherwise, R will print **“x is a positive number”**.

**x <- -1**

**if (x < 0) {**

**print("x is a negative number")**

**} else if (x == 0) {**

**print("x is zero")**

**} else {**

**print("x is a positive number")**

**}**

Since -1 is less than 0,  the condition for the if statement evaluates to **TRUE**, and R prints **“x is a negative number”**.

**[1] "x is a negative number"**

If you make x equal to 0, R will first check the if condition **(x < 0)**, and determine that it is FALSE. Then, R will evaluate the else if condition. This condition, **x==0**, is TRUE. So, in this case, R prints **“x is zero”**.

If you make x equal to 1, both the if condition and the else if condition evaluate to **FALSE**. So, R will execute the else statement and print **“x is a positive number”**.

As soon as R discovers a condition that evaluates to TRUE, R executes the corresponding code and ignores the rest.

**Additional resource**

To learn more about logical operators and conditional statements, check out DataCamp's tutorial [Conditionals and Control Flow in R](https://www.datacamp.com/community/tutorials/conditionals-and-control-flow-in-r). DataCamp is a popular resource for people learning about computer programming. The tutorial is filled with useful examples of coding applications for logical operators and conditional statements (and relational operators), and offers a helpful overview of each topic and the connections between them.

vailable R packages

To make the most of R for your data analysis, you will need to install packages. **Packages** are units of reproducible R code that you can use to add more functionality to R. The best part is that the R community creates and shares packages so that other users can access them! In this reading, you will learn more about widely used packages and where to find them.



Packages can be found in repositories, which are collections of useful packages that are ready to install. You can find repositories on [**Bioconductor**](http://bioconductor.org/), [**R-Forge**](https://r-forge.r-project.org/), [**rOpenSci**](https://ropensci.org/), or [**GitHub**](https://github.com/), but the most commonly used repository is the Comprehensive R Archive Network or [**CRAN**](https://cran.r-project.org/). CRAN stores code and documentation so that you can install packages into your own RStudio space.

**Package documentation**

Packages will not only include the code itself, but also documentation that explains the package’s author, function, and any other packages that you will need to download. When you are using CRAN, you can find the package documentation in the DESCRIPTION file.

Check out Karl Broman's [**R Package Primer**](https://kbroman.org/pkg_primer/)to learn more.

**Choosing the right packages**

With so many packages out there, it can be hard to know which ones will be the most useful for your library or directory of installed packages. Luckily, there are some great resources out there:

* [**Tidyverse**](https://www.tidyverse.org/): the tidyverse is a collection of R packages specifically designed for working with data. It’s a standard library for most data analysts, but you can also download the packages individually.
* [**Quick list of useful R packages**](https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages): this is RStudio Support’s list of useful packages with installation instructions and functionality descriptions.
* [**CRAN Task Views**](https://cran.r-project.org/web/views/): this is an index of CRAN packages sorted by task. You can search for the type of task you need to perform and it will pull up a page with packages related to that task for you to explore.

You will discover more packages throughout this course and as you use R more often, but this is a great starting point for building your own library.

Working with biased data

Every data analyst will encounter an element of bias at some point in the data analysis process. That’s why it’s so important to understand how to identify and manage biased data whenever possible. You might recall we explored bias in detail in Course 3 of this program. In this reading, you will read a real-life example of an analyst who discovered bias in their data, and learn how they used R to address it.

**Addressing biased data with R**



This scenario was shared by a quantitative analyst who collects data from people all over the world. They explain how they discovered bias in their data, and how they used R to address it:

“I work on a team that collects survey-like data. One of the tasks my team does is called a side-by-side comparison. For example, we might show users two ads side-by-side at the same time. In our survey, we ask which of the two ads they prefer. In one case, after many iterations, we were seeing consistent bias in favor of the first item. There was also a measurable decrease in the preference for an item if we swapped its position to second.

So we decided to add randomization to the position of the ads using R. We wanted to make sure that the items appeared in the first and second positions with similar frequencies. We used sample() to inject a randomization element into our R programming. In R, the sample() function allows you to take a random sample of elements from a data set. Adding this piece of code shuffled the rows in our data set randomly. So when we presented the ads to users, the positions of the ads were now random and controlled for bias. This made the survey more effective and the data more reliable.”

**Key takeaways**

The sample() function is just one of many functions and methods in R that you can use to address bias in your data. Depending on the kind of analysis you are conducting, you might need to incorporate some advanced processes in your programming. Although this program won’t cover those kinds of processes in detail, you will likely learn more about them as you get more experience in the data analytics field.

To learn more about bias and data ethics, check out these resources:

* [**Bias function:**](https://www.rdocumentation.org/packages/SimDesign/versions/2.2/topics/bias) This web page is a good starting point to learn about how the bias function in R can help you identify and manage bias in your analysis.
* [**Data Science Ethics**](https://datasciencebox.org/ethics.html): This online course provides slides, videos, and exercises to help you learn more about ethics in the world of data analytics. It includes information about data privacy, misrepresentation in data, and applying ethics to your visualizations.