Data Analysis 101 – Data Wrangling and Visualization using R.

Sameer Mathur, Aryeman Gupta Mathur

2023-06-30

Table of contents

Preface

This book is about:::

- R programming.
- RStudio.

1 Overview and Getting Started

1.1 Overview of R programming?

1. R is a programming language and open-source software environment for statistical computing, data analysis, and visualization. It was created by Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand in the early 1990s.
2. R offers an extensive array of statistical and graphical techniques, such as linear and nonlinear modeling, classical statistical tests, time-series analysis, and clustering, among others. In addition, it supports data manipulation, data import/export, and multiple data formats.
3. It is frequently employed in the creation of statistical software, data analysis, and data science.
4. R is an **open-source** project, and the GNU General Public License makes its source code accessible. The S programming language was created at Bell Labs in the 1970s by John Chambers and others. R is an implementation of S.
5. The Comprehensive R Archive Network (CRAN) provides a significant number of downloadable packages for the extensible programming language R. These products include tools for machine learning, data mining, and visualization.
6. R is a prominent tool for data analysts, data scientists, statisticians, and researchers in academia, industry, and government. It has a large and active user community that contributes to the maintenance and development of R packages. [1]

1.2 Benefits of R

Numerous advantages are associated with mastering R programming:

1. As an open-source programming language, R is free to use, distribute, and modify. This makes it accessible to anyone, regardless of budget or resources, who desires to learn and use it.

2. R has a large and active community of users and developers who contribute to its maintenance and development. This community provides an abundance of resources, such as documentation, forums, and packages, making it simpler for users to learn and effectively use R.
3. R was designed particularly for statistical computing and has a vast array of statistical functions and models built in. In addition, the thousands of available packages on CRAN provide additional statistical tools and methodologies.
4. R's robust and adaptable graphics system enables users to create high-quality data visualizations for data exploration, analysis, and communication.
5. Reproducibility: R enables users to write scripts that are readily shareable and reproducible, which increases transparency and facilitates collaboration on data analysis projects.
6. R is readily compatible with other programming languages, such as Python and SQL, as well as data storage and manipulation tools, such as Hadoop and Spark. [2]

1.3 Key Features of R

Among the main characteristics of R programming are:

1. R is an object-oriented programming language, allowing users to construct and manipulate complex data structures and objects.
2. R also supports functional programming, allowing users to construct functions as first-class objects and use higher-order functions to create more complex algorithms.
3. Extensibility: R has a large number of packages available for distribution from the Comprehensive R Archive Network (CRAN). These products include tools for machine learning, data mining, and visualization.
4. Graphics and visualization: R's robust and adaptable graphics system enables users to generate high-quality visualizations for data exploration, analysis, and communication.
5. Data manipulation: R offers a vast array of data manipulation tools, including subsetting, merging, reshaping, and aggregating data.
6. R was designed particularly for statistical computing and has a vast array of built-in statistical functions and models, such as linear and nonlinear modeling, classical statistical tests, time-series analysis, and clustering, among others.
7. R offers a robust interactive environment, with support for data input/output, data exploration, and data visualization. [3]

1.4 Running R locally

Follow these general instructions to run R locally on your computer:

1. Install and download R
2. Select an IDE: To write and execute R code, you can use a text editor or a dedicated integrated development environment (IDE). Popular alternatives include RStudio, Emacs, and Vim. RStudio is a free and open-source integrated development environment (IDE) designed specifically for R that is extensively utilized by the R community.
3. Open your preferred IDE or text editor, and then initiate a new R session. This can be accomplished by typing R at the command prompt or launching a new R script file.
4. Use your preferred IDE or text editor to write and execute R code. You can execute your code by choosing the "Run" button or by pressing Ctrl+Enter (Windows) or Cmd+Enter (Mac) at the end of each line.
5. Protect your work: Save your R code to a file for reference and reuse in the future. Your plots and outputs can also be saved to files for use in reports and presentations. [4]

1.4.1 Installing R locally

Follow these general instructions to install R locally on your computer:

1. Go to the website of the R project: In your web browser, go to the R project website (https://www.r-project.org/).
2. Select an operating system: On the download page, choose the version of R compatible with your operating system (Windows, Mac, or Linux).
3. Select a mirror to obtain R from after clicking the download link for your operating system.
4. Install R: Double-click the downloaded file once the download is complete to begin the installation procedure. Install R on your computer by following the instructions. [5]

1.4.2 RStudio

RStudio is a specialized integrated development environment (IDE) for R programming. It provides an intuitive interface and a suite of tools for R-based data analysis, visualization, and modeling.

Among the features of RStudio are the following:

1. RStudio has a code editor with syntax highlighting, code completion, and other features that make writing R code simpler.
2. RStudio offers a data viewer that enables users to observe and investigate their data in a tabular format.
3. The plots pane in RStudio depicts the graphical outputs generated by R code.
4. Console pane: The console pane in RStudio displays R code and its output and enables users to execute R commands in an interactive environment.
5. RStudio provides tools for managing R packages, including the installation, update, and removal of packages.
6. Version control: RStudio integrates with version control systems like Git, enabling users to manage and collaborate on their code.
7. Shiny applications: RStudio allows users to create interactive web applications using the Shiny web development utility for R. [6]

To install RStudio locally on your computer, you can follow these general steps:

1. Download R: Before you can install RStudio, you will need to download and install R. You can download the appropriate version of R for your operating system from the R project website (https://www.r-project.org/).
2. Download RStudio: Go to the RStudio download page (https://www.rstudio.com/products/rstudio/downl and select the appropriate version of RStudio for your operating system.
3. Install RStudio: Once the RStudio installer has downloaded, run the installer and follow the instructions to install RStudio on your computer.
4. Open RStudio: Once the installation is complete, open RStudio by double-clicking the RStudio icon on your desktop or in your Applications folder.
5. Start an R session: In RStudio, click on the Console tab to start an R session. You can then type R commands in the console and run them using the "Run" button or by pressing Ctrl+Enter (Windows) or Cmd+Enter (Mac). [7]

1.5 Running R in the Cloud

1. Running R in the cloud enables users to access R and RStudio from any location with an internet connection, without deploying R locally.
2. Numerous cloud service providers offer virtual machines (VMs) with pre-installed R and RStudio. Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) are popular options.

Depending on the user's use case and requirements, running R in the cloud provides a variety of advantages and disadvantages. Here are some general advantages and disadvantages of cloud-based R:

Benefits:

1. Scalability: Cloud service providers provide scalable computing resources that can be adjusted to satisfy the requirements of a specific workload. This can be especially beneficial for data-intensive tasks that demand a great deal of computing power.
2. Running R in the cloud enables users to access R and RStudio from any location with an internet connection, making it simple to collaborate on projects and share data.
3. Cloud providers offer flexible pricing models that may be more cost-effective than running R on local hardware, especially for short-term or infrequent use cases.
4. Security: Cloud service providers offer a variety of security features, including firewalls and encryption, to safeguard data and applications from unauthorized access or attacks. [8]

Drawbacks:

1. Running R in the cloud requires a stable internet connection, which may not be accessible at all times or in all locations. This can hinder the ability to work on initiatives involving data analysis and modeling.
2. Running R in the cloud necessitates familiarity with cloud computing platforms and tools, which can necessitate a steep learning curve for users new to cloud computing.
3. Data privacy: Storing data in the cloud may raise concerns about data privacy, particularly for sensitive or confidential data. Cloud service providers offer security features, but users must be aware of the risks and secure their data accordingly.
4. Although cloud computing can be cost-effective in certain circumstances, it can also be costly for long-term or high-volume use cases, especially if data storage or other resources are required in addition to computing capacity. [8]

1.5.1 Cloud Service Providers – Posit, AWS, Azure, GCP

Posit is a comparatively new cloud service provider that offers high-performance computing resources for data-intensive applications. AWS, Azure, and GCP are three of the most prominent cloud service providers, each of which offers a variety of services and features for running cloud-based applications. Here is a comparison of the four service providers:

Posit:

1. Provides computing resources with high efficacy for data-intensive applications

2. Provides bare-metal instances for superior performance and adaptability
3. Dedicated to data security and compliance
4. Offers hardware configurations tailored to specific duties.

AWS:

1. Provides a variety of cloud computing services, such as computing, storage, and database services
2. A large and active user community with an abundance of resources and assistance.
3. Includes pay-as-you-go and reserved instance pricing options.
4. Offers a variety of tools and services for administering and securing cloud-based applications

Azure:

1. Provides a variety of cloud computing services, such as computing, storage, and networking Integrates tightly with Microsoft's enterprise software and services.
2. Provides flexible pricing models, including pay-as-you-go, reserved instance, and spot instance pricing
3. Provides a variety of tools and services for managing and securing cloud-based applications

GCP:

1. Provides a variety of cloud computing services, such as computing, storage, and networking
2. Includes a variety of tools and services for machine learning and artificial intelligence.
3. Offers flexible pricing models, including pay-as-you-go and sustained use pricing
4. Provides a variety of tools and services for managing and securing cloud-based applications. [9]

1.5.2 General Instructions for operating R in the cloud

1. Choose a cloud service provider.
2. Once you have selected a cloud service provider, you can construct a virtual machine with R and RStudio already installed. The precise steps for creating a virtual machine will differ depending on the provider you select, but the majority of providers offer an intuitive interface for creating and configuring virtual machines.
3. Once the virtual machine is operational, you can connect to it using a web browser or remote desktop software. This should be accompanied by instructions from the provider.
4. Start a R session: Once you are connected to the virtual machine, you can use RStudio or the R console to launch a R session.
5. You can transfer your data to the virtual machine and then access it through RStudio or the R console. You can also connect to external data sources with R programs like DBI and odbc. [9]

1.6 References

[1]

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics, 5(3), 299-314. https://www.jstor.org/stable/1390807

[2]

Peng, R. D. (2016). R programming for data science. O'Reilly Media.

Venables, W. N., Smith, D. M., & R Development Core Team. (2019). An introduction to R. Network Theory Ltd. Retrieved from https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

Wickham, H. (2016). ggplot2: Elegant graphics for data analysis. Springer-Verlag.

Gandrud, C. (2015). Reproducible research with R and RStudio. CRC Press.

Grolemund, G., & Wickham, H. (2017). R for data science: Import, tidy, transform, visualize, and model data. O'Reilly Media.

[3]

Chambers, J. M. (2016). Extending R (2nd ed.). CRC Press.

Wickham, H., & Grolemund, G. (2017). R packages: Organize, test, document, and share your code. O'Reilly Media.

Murrell, P. (2006). R graphics. CRC Press.

Wickham, H. (2014). Tidy data. Journal of Statistical Software, 59(10), 1-23.

Venables, W. N., Smith, D. M., & R Development Core Team. (2019). An introduction to R. Network Theory Ltd. Retrieved from https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[4]

The R Project for Statistical Computing. (2021). Download R for (Mac) OS X. https://cran.r-project.org/bin/macosx/

RStudio. (2021). RStudio. https://www.rstudio.com/

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Peng, R. D. (2016). R programming for data science. O'Reilly Media.

Gandrud, C. (2015). Reproducible research with R and RStudio. CRC Press.

[5]

The R Project for Statistical Computing. (2021). Download R for (Mac) OS X. https://cran.r-project.org/bin/macosx/

The R Project for Statistical Computing. (2021). Download R for Windows. https://cran.r-project.org/bin/windows/base/

The R Project for Statistical Computing. (2021). Download R for Linux. https://cran.r-project.org/bin/linux/

RStudio. (2021). RStudio. https://www.rstudio.com/products/rstudio/download/

Peng, R. D. (2016). R programming for data science. O'Reilly Media.

[6]

RStudio. (2021). RStudio. https://www.rstudio.com/products/rstudio/features/

Peng, R. D. (2016). R programming for data science. O'Reilly Media.

Wickham, H. (2014). Tidy data. Journal of Statistical Software, 59(10), 1-23.

Wickham, H., & Grolemund, G. (2017). R packages: Organize, test, document, and share your code. O'Reilly Media.

Chang, W., & Cheng, J. (2020). R Markdown: The definitive guide. CRC Press.

Chang, W., Cheng, J., Allaire, J. J., Xie, Y., & McPherson, J. (2021). shiny: Web application framework for R. R package version 1.6.0. https://CRAN.R-project.org/package=shiny

[7]

The R Project for Statistical Computing. (2021). Download R for (Mac) OS X. https://cran.r-project.org/bin/macosx/

The R Project for Statistical Computing. (2021). Download R for Windows. https://cran.r-project.org/bin/windows/base/

The R Project for Statistical Computing. (2021). Download R for Linux. https://cran.r-project.org/bin/linux/

RStudio. (2021). RStudio. https://www.rstudio.com/products/rstudio/download/

Peng, R. D. (2016). R programming for data science. O'Reilly Media.

[8]

Biecek, P., & Kosinski, M. (2018). Mastering Software Development in R. Packt Publishing.

Amazon Web Services. (2021). AWS. https://aws.amazon.com/

Microsoft Azure. (2021). Azure. https://azure.microsoft.com/

Google Cloud Platform. (2021). GCP. https://cloud.google.com/

RStudio. (2021). RStudio Server Pro. https://rstudio.com/products/rstudio-server-pro/

Amazon Web Services. (2021). EC2 User Guide for Linux Instances. https://docs.aws.amazon.com/AWSEC2/la

Microsoft Azure. (2021). Create a Windows virtual machine with the Azure portal. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal

Google Cloud Platform. (2021). Compute Engine Documentation. https://cloud.google.com/compute/docs

Amazon Web Services. (2021). Running RStudio Server Pro using Amazon EC2. https://docs.rstudio.com/rsp/quickstart/aws/

[9]

Posit. (2021). High-Performance Computing Services. https://posit.cloud/

Amazon Web Services. (2021). AWS. https://aws.amazon.com/

Microsoft Azure. (2021). Azure. https://azure.microsoft.com/

Google Cloud Platform. (2021). GCP. https://cloud.google.com/

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., … Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58. https://doi.org/10.1145/1721654.1721672

Xiao, Z., Chen, Z., & Zhang, J. (2014). Cloud computing research and security issues. Journal of Network and Computer Applications, 41, 1–11. https://doi.org/10.1016/j.jnca.2013.11.004

Cloud Spectator. (2021). Cloud Service Provider Pricing Models: A Comprehensive Guide. https://www.cloudspectator.com/cloud-service-provider-pricing-models-a-comprehensive-guide/

2 R Packages

1. R packages are compilations of code, data, and documentation that extend the functionality of R, a statistical computing and graphics programming language and software environment.
2. R packages are typically developed by R users and developers, and they offer additional tools, functions, and datasets that can be used for a variety of purposes, including data analysis, visualization, machine learning, and more.
3. Installation sources for R packages include the Comprehensive R Archive Network (CRAN), Bioconductor, GitHub, and additional online repositories.
4. The packages can be imported into R using the `library()` function, and the functions and data contained within the packages can be accessed and utilized in R scripts and interactive sessions. [1]

2.1 Benefits of R Packages

There are numerous advantages to using R packages:

1. Reusability: R packages enable users to write code that is readily reusable across applications. Once a package has been created and published, others can install and use it, sparing them time and effort in coding.
2. Collaboration: Individuals or teams can develop packages collaboratively, enabling the sharing of code, data, and ideas. This promotes collaboration within the R community and the creation of new tools and techniques.
3. Standardization: Packages help standardize the code and methodology used for particular duties, making it simpler for users to comprehend and replicate the work of others. This decreases the possibility of errors and improves the dependability of results.
4. Scalability: Packages can manage large data sets and sophisticated analyses, enabling users to scale up their work to larger, more complex problems.
5. Accessibility: R packages are freely available and can be installed on a variety of operating systems, making them accessible to a broad spectrum of users. [1]

2.2 Comprehensive R Archive Network (CRAN)

1. The Comprehensive R Archive Network (CRAN) is a global server network that maintains and distributes R packages, which are collections of code, data, and documentation that extend R's functionality.
2. CRAN provides a centralized and organized repository for R packages, making it simple for users to locate, acquire, and install the necessary packages.
3. CRAN hosts thousands of packages, which can be downloaded and installed using the `install.packages()` function in R. The packages are organized into categories such as graphics, statistics, and machine learning, making it easy to find packages that are relevant to your needs.
4. The R Development Core Team maintains CRAN, which is readily accessible to anyone with an internet connection. [2]

2.3 Installing a R Package

1. The `install.packages()` function can be used to deploy a R package.
2. For example, in order to install the `ggplot2` package, for instance, you would execute the following code in R:

```
install.packages("ggplot2")
```

```
Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.3'
(as 'lib' is unspecified)
```

```
also installing the dependencies 'colorspace', 'utf8', 'farver', 'labeling', 'munsell', 'RCol
```

3. This will obtain and install on your system the `ggplot2` package and its dependencies.
4. Note that a package only needs to be installed once on your system. After installation, you can import the package using the `library()` function in your R session.
5. To import the `ggplot2` package, for instance, you would execute the following code in R:

```
library(ggplot2)
```

6. This will make the `ggplot2` package's functions and data sets available for use within your R session.

2.3.1 Popular R Packages

There are many popular R packages, covering a wide range of topics and domains, such as the following examples:

1. `ggplot2`: A package for data visualization that provides a flexible system for creating high-quality graphics.
2. `dplyr`: A package for data manipulation that provides a set of functions for filtering, summarizing, and transforming data.
3. `tidyr`: A package for data manipulation that provides functions for reshaping data from wide to long and vice versa.
4. `tidyverse`: A collection of packages for data manipulation, visualization, and modeling that includes dplyr, ggplot2, and tidyr.
5. `data.table`: A package for data manipulation that provides a fast and efficient way to work with large data sets.
6. `stringi`: A package for working with strings that provides a wide range of string manipulation functions optimized for performance.
7. `stringr`: A package for working with strings, which are common data types in text processing and natural language processing.
8. `shiny`: A package for creating interactive web applications and dashboards.
9. `RODBC`: A package for connecting R to databases using the Open Database Connectivity (ODBC) standard.
10. `ggmap`: A package for creating maps and visualizing spatial data using `ggplot2`.
11. `rmarkdown`: A package for creating dynamic documents that combine text, code, and results in a single document.
12. `knitr`: A package for dynamic report generation, which allows users to create reports that automatically update as data or code changes.
13. `MASS`: A package for multivariate analysis and statistical modeling, which includes functions for fitting linear and nonlinear regression models, principal components analysis..
14. `lme4`: A package for fitting mixed-effects models, which are useful for modeling hierarchical data structures.
15. `caret`: A package for machine learning that provides tools for data preprocessing, model training, and model evaluation.
16. `forecast`: A package for time series forecasting that includes functions for modeling, visualization, and evaluation of time series data. [2]

2.3.2 Getting help

There are several ways to get help with an R package:

1. Consult the documentation: The documentation included with the majority of R packages describes the functions and data sets provided by the package and provides usage examples. Using the `help()` function or typing `?package name>` in the R console provides access to a package's documentation.
2. Utilize the integrated support system: R's integrated help system provides documentation and demonstrations for R functions and packages. In the R console, you can access the help system by typing `help(topic)` or `?topic`, where "topic" is the name of the function or package you require assistance with.
3. Online: There are many online resources available for assistance with R packages, including blogs, forums, and question-and-answer sites such as Stack Overflow. These resources are useful for locating solutions to specific problems and obtaining general guidance on how to use a package.

2.4 References

[1]

Wickham, H., & Bryan, J. (2018). R packages: Organize, test, document, and share your code. O'Reilly Media.

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[2]

The R Project for Statistical Computing. (2021). The Comprehensive R Archive Network (CRAN). https://cran.r-project.org/

3 Inbuilt R functions

3.1 Mathematical Operations

R is a powerful programming language for performing mathematical operations and statistical calculations. Here are some common mathematical operations in R:

1. Arithmetic Operations: R can perform basic arithmetic operations such as addition (+), subtraction (-), multiplication (*), and division (/).

```
# Addition and Subtraction
5+9-3
```

```
[1] 11
```

```
# Multiplication and Division
(5 + 3) * 7 /2
```

```
[1] 28
```

2. Exponentiation and Logarithms: R can raise a number to a power using the ^ or ** operator or take logarithms.

```
# exponentiation
2^6
```

```
[1] 64
```

```
# Exponential of x=2 i.e. e^2
exp(2)
```

```
[1] 7.389056
```

```r
# logarithms base 2 and base 10
log2(64) + log10(100)
```

```
[1] 8
```

3. Other mathematical functions: R has many additional useful mathematical functions.

- We can find the absolute value, square roots, remainder on division.

```r
# absolute value of x=-5
abs(-9)
```

```
[1] 9
```

```r
# square root of x=70
sqrt(70)
```

```
[1] 8.3666
```

```r
# remainder of the division of 11/3
11 %% 3
```

```
[1] 2
```

- We can round numbers, find their floor, ceiling or up to a number of significant digits

```r
# Value of pi to 10 decimal places
pi = 3.1415926536

# round(): This function rounds a number to the given number of decimal places
# For example, round(pi, 3) returns 3.142
round(pi,3)
```

```
[1] 3.142
```

```r
# ceiling(): This function rounds a number up to the nearest integer.
# For example, ceiling(pi) returns 4
ceiling(pi)
```

```
[1] 4
```

```r
# floor(): This function rounds a number down to the nearest integer.
# For example, floor(pi) returns 3.
floor(pi)
```

```
[1] 3
```

```r
# signif(): This function rounds a number to a specified number of significant digits.
# For example, signif(pi, 3) returns 3.14.
signif(pi,3)
```

```
[1] 3.14
```

4. Statistical calculations: R has many built-in functions for statistical calculations, such as mean, median, standard deviation, and correlation.

```r
x <- c(0, 1, 1, 2, 3, 5, 8)   # create a vector of 7 Fibonacci numbers
length(x) # count how many numbers do we have
```

```
[1] 7
```

```r
mean(x)   # calculate the mean
```

```
[1] 2.857143
```

```r
median(x) # calculate the median
```

```
[1] 2
```

```
sd(x)     # calculate the standard deviation
```

```
[1] 2.794553
```

```
y <- c(1, 2, 3, 4, 5, 6, 7) # create a new vector of positive integers
cor(x,y)  # calculate the correlation between x and y
```

```
[1] 0.938668
```

3.2 Assigning values to variables

1. A variable can be used to store a value. For example, the R code below will store the sales in a variable, say "sales":

```
# use the assignment operator <-
sales <- 9
# alternately, use =
sales = 9
```

2. It is possible to use <- or = for variable assignments.
3. R is case-sensitive. This means that `Sales` is different from `sales`
4. It is possible to perform some operations with it.

```
# multiply sales by 2
2 * sales
```

```
[1] 18
```

5. We can change the value stored in a variable

```
# change the value
sales <- 15
# display the revised sales
sales
```

```
[1] 15
```

6. The following R code creates two variables holding the sales and the price of a product and we can use them to compute the revenue.

```
# sales
sales <- 5

# price
price <- 7

# Calculate the revenue
revenue <- price*sales
revenue
```

```
[1] 35
```

4 Data Structures

4.1 Popular Data Structures

The R programming language includes a number of data structures that are frequently employed in data analysis and statistical modeling. These are some of the most popular data structures in R:

1. **Vector**: A vector is a one-dimensional array that stores identical data types, such as numeric, character, or logical. The "c()" function can be used to create vectors, and indexing can be used to access individual vector elements.
2. **Factor**: A factor is a vector representing categorical data, with each distinct value or category represented as a level. Using indexing, individual levels of a factor can be accessed using the "factor()" function.
3. **Dataframe**: Similar to a spreadsheet, a data frame is a two-dimensional table-like structure that can store various types of data in columns. The "data.frame()" function can be used to construct data frames, and individual elements can be accessed using row and column indexing.
4. **Matrix**: A matrix is a two-dimensional array of data with identical rows and columns. The "matrix()" function can be used to construct matrices, and individual elements can be accessed using row and column indexing.
5. **Array**: An array is a multidimensional data structure that can contain data of the same data type in user-specified dimensions. Arrays can be constructed using the "array()" function, and elements can be accessed using multiple indexing.
6. **List**: A list is an object that may comprise elements of various data types, including vectors, matrices, data frames, and even other lists. The "list()" function can be used to construct lists, while indexing can be used to access individual elements.

These data structures are helpful for storing and manipulating data in R, and they can be utilized in numerous applications, such as statistical analysis and data visualization.

We will focus our attention on Vectors, Factors and Dataframes, since we believe that these are the three most useful data structures. [1]

4.2 Vectors

1. A vector is a fundamental data structure in R that can hold a sequence of values of the same data type, such as integers, numeric, character, or logical values.
2. A vector can be created using the `c()` function.
3. R supports two forms of vectors: atomic vectors and lists. Atomic vectors are limited to containing elements of a single data type, such as numeric or character. Lists, on the other hand, can contain elements of various data types and structures. [1]

4.2.1 Vectors in R

1. The following R code creates a numeric vector, a character vector and a logical vector respectively.

```r
# Read data into vectors
names <- c("Ashok", "Bullu", "Charu", "Divya")
ages <- c(72, 49, 46, 42)
females <- c(FALSE, TRUE, TRUE, TRUE)
```

2. The `c()` function is employed to combine the four character elements into a single vector.
3. Commas separate the elements of the vector within the parentheses.
4. Individual elements of the vector can be accessed via indexing, which utilizes square brackets []. For instance, names[1] returns "Ashok", while names[3] returns "Charu".
5. We can also perform operations such as categorizing and filtering on the entire vector. For instance, `sort(names)` returns a vector of sorted names, whereas names[names!= "Bullu"] returns a vector of names excluding "Bullu."

4.2.2 Vector Operations

Vectors can be used to perform the following vector operations:

1. **Accessing Elements:** We can use indexing with square brackets to access individual elements of a vector. To access the second element of the "names" vector, for instance, we can use:

```r
names[2]
```

```
[1] "Bullu"
```

This returns “Bullu”, the second element of the “names” vector.

2. **Concatenation:** The “c()” function can be used to combine multiple vectors into a single vector. For instance, to combine the “names” and “ages” vectors into the “people” vector, we can use:

```
persons <- c(names, ages)
persons
```

```
[1] "Ashok" "Bullu" "Charu" "Divya" "72"    "49"    "46"    "42"
```

This generates an eight-element vector containing the names and ages of the four people.

3. **Subsetting:** We can use indexing with a logical condition to construct a new vector that contains a subset of elements from an existing vector. For instance, to construct a new vector named “female_names” containing only the females’ names, we can use:

```
female_names <- names[females == TRUE]
female_names
```

```
[1] "Bullu" "Charu" "Divya"
```

This generates a new vector comprising three elements containing the names of the three females (“Bullu”, “Charu”, and “Divya”).

4. **Arithmetic Operations:** We can perform element-wise arithmetic operations on vectors. To calculate the sum of the “ages” vector, for instance, we can use:

```
sum(ages)
```

```
[1] 209
```

This returns 209, the sum of the four ages.

5. **Logical Operations:** We can perform logical operations on vectors, which are also executed element-by-element. To create a new vector titled “middle_age” that indicates whether each individual is 45 to 55 years old, for instance, we can use:

```
middle_age <- (ages >= 45) & (ages <= 55)
middle_age
```

```
[1] FALSE  TRUE  TRUE FALSE
```

This generates a new vector with four elements containing logical values indicating whether each person is between 45 and 55 years of age.

To test whether any of the elements in the "ages" vector are greater than 50, we can use:

```
any(ages > 50)
```

```
[1] TRUE
```

6. **Unique Values:** We can find the unique values in a vector using the "unique()" function. For example, to find the unique values in the "ages" vector, we can use:

```
unique(ages)
```

```
[1] 72 49 46 42
```

7. **Sorting:** We can sort a vector in ascending or descending order using the "sort()" function. For example, to sort the "ages" vector in descending order, we can use:

```
sort(ages, decreasing = TRUE)
```

```
[1] 72 49 46 42
```

4.2.3 Statistical Operations on Vectors

1. **Length**: The length represents the count of the number of elements in a vector.

```
length(ages)
```

```
[1] 4
```

2. **Maximum** and **Minimum**: The maximum and minimum values are the vector's greatest and smallest values, respectively.
3. **Range**: The range is a measure of the spread that represents the difference between the maximum and minimum values in a vector.

```
min(ages)
```

```
[1] 42
```

```
max(ages)
```

```
[1] 72
```

```
range(ages)
```

```
[1] 42 72
```

4. **Mean**: The mean is a central tendency measure that represents the average value of a vector's elements.
5. **Standard Deviation**: The standard deviation is a measure of dispersion that reflects the amount of variation in a vector's elements.
6. **Variance**: The variance is another measure of the spread. It is square of the Standard Deviation.

```
mean(ages)
```

```
[1] 52.25
```

```
sd(ages)
```

```
[1] 13.47529
```

```
var(ages)
```

```
[1] 181.5833
```

7. **Median**: The median is a measure of central tendency that represents the middle value of a sorted vector.

```
median(ages)
```

```
[1] 47.5
```

8. **Quantiles**: The quantiles are a set of cut-off points that divide a sorted vector into equal-sized groups.

```
quantile(ages)
```

```
   0%   25%   50%   75%  100%
42.00 45.00 47.50 54.75 72.00
```

This will return a set of five values, representing the minimum, first quartile, median, third quartile, and maximum of the four ages.

Thus, we note that the R programming language provides a wide range of statistical operations that can be performed on vectors for data analysis and modeling. Vectors are clearly a potent and versatile data structure that can be utilized in a variety of ways.

4.2.4 Strings

Here are some common string operations that can be conducted using the provided vector examples.

1. **Substring**: The `substr()` function can be used to extract a substring from a character vector. To extract the first three characters of each name in the "names" vector, for instance, we can use:

```
substr(names, 1, 3)
```

```
[1] "Ash" "Bul" "Cha" "Div"
```

This returns a new character vector containing the initial three letters of each name ("Ash", "Bul", "Cha", and "Div").

2. **Concatenation**: Using the `paste()` function, we can concatenate two or more character vectors into a singular vector. To create a new vector containing the names and ages of the individuals, for instance, we can use:

```
persons <- paste(names, ages)
persons
```

```
[1] "Ashok 72" "Bullu 49" "Charu 46" "Divya 42"
```

This will generate a new eight-element character vector containing the name and age of each individual, separated by a space.

3. **Case Conversion:** The `toupper()` and `tolower()` functions can be used to convert the case of characters within a character vector. To convert the "names" vector to uppercase letters, for instance, we can use:

```
toupper(names)
```

```
[1] "ASHOK" "BULLU" "CHARU" "DIVYA"
```

This will generate a new character vector with all of the names converted to uppercase.

4. **Pattern Matching:** Using the `grep()` and `grepl()` functions, we can search for a pattern within the elements of a character vector. To find the names in the "names" vector that contain the letter "a", for instance, we can use:

```
grep("a", names)
```

```
[1] 3 4
```

This returns a vector containing the indexes of the "names" vector elements that contain the letter "a."

5. **Regular Expressions:** We can use regular expressions with the `grep()` and `grepl()` functions to search for patterns in the elements of a character vector. To find the names in the "names" vector that begin with the letter "C", for instance, we can use:

```
grep("^C", names)
```

```
[1] 3
```

This returns a vector containing the indexes of the elements in “names” that begin with the letter “C.” [1]

4.3 References

[1]

R Core Team. (2021). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. **https://www.R-project.org/**

R Core Team. (2022). Vectors, Lists, and Arrays. R Documentation. https://cran.r-project.org/doc/manuals/r-release/R-intro.html#vectors-lists-and-arrays

Wickham, H., & Grolemund, G. (2016). R for data science: Import, tidy, transform, visualize, and model data. O’Reilly Media, Inc.

5 Reading Data

Dataframes and Tibbles are frequently employed data structures in R for storing and manipulating data. They facilitate the organization, exploration, and analysis of data.

5.1 Dataframes

1. A dataframe is a two-dimensional table-like data structure in R that stores data in rows and columns, with distinct data types for each column.
2. Similar to a spreadsheet or a SQL table, it is one of the most frequently employed data structures in R. Each column in a data.frame is a constant-length vector, and each row represents an observation or case.
3. Using the `data.frame()` function or by importing data from external sources such as CSV files, Excel spreadsheets, or databases, dataframe objects can be created in R.
4. dataframe objects have many useful built-in methods and functions for manipulating and summarizing data, including subsetting, merging, filtering, and aggregation. [1]

5.1.1 Creating a dataframe using raw data

5. The following code generates a data.frame named `df` containing three columns - `names`, `ages`, and `heights`, and four rows of data for each individual.

```r
# Create input data as vectors
names <- c("Ashok", "Bullu", "Charu", "Divya")
ages <- c(72, 49, 46, 42)
heights <- c(170, 167, 160, 166)

# Combine input data into a data.frame
people <- data.frame(Name = names, Age = ages, Height = heights)

# Print the resulting dataframe
print(people)
```

```
   Name Age Height
1 Ashok  72    170
2 Bullu  49    167
3 Charu  46    160
4 Divya  42    166
```

5.2 Reading Inbuilt datasets in R

1. R contains a number of built-in datasets that can be accessed without downloading or integrating from external sources. Here are some of the most frequently used built-in datasets in R:

- `women`: This dataset includes the heights and weights of a sample of 15,000 women.
- `mtcars`: This dataset contains information on 32 distinct automobile models, including the number of cylinders, engine displacement, horsepower, and weight.
- `diamonds`: This dataset includes the prices and characteristics of approximately 54,000 diamonds, including carat weight, cut, color, and clarity.
- `iris`: This data set measures the sepal length, sepal width, petal length, and petal breadth of 150 iris flowers from three distinct species.

5.2.1 The `women` dataset

As an illustration, consider the `women` dataset inbuilt in R, which contains information about the heights and weights of women. It has just two variables:

1. `height`: Height of each woman in inches
2. `weight`: Weight of each woman in pounds
3. The `data()` function is used to import any inbuilt dataset into R. The `data(women)` command in R loads the `women` dataset

```
data(women)
```

4. The `str()` function gives the dimensions and data types and also previews the data.

```
str(women)
```

```
'data.frame':   15 obs. of  2 variables:
 $ height: num  58 59 60 61 62 63 64 65 66 67 ...
 $ weight: num  115 117 120 123 126 129 132 135 139 142 ...
```

5. The `summary()` function gives some summary statistics.

```
summary(women)
```

```
    height          weight
 Min.   :58.0   Min.   :115.0
 1st Qu.:61.5   1st Qu.:124.5
 Median :65.0   Median :135.0
 Mean   :65.0   Mean   :136.7
 3rd Qu.:68.5   3rd Qu.:148.0
 Max.   :72.0   Max.   :164.0
```

5.2.2 The `mtcars` dataset

The `mtcars` dataset inbuilt in R comprises data on the fuel consumption and other characteristics of 32 different automobile models. Here is a concise description of the 11 `mtcars` data columns:

1. `mpg`: Miles per gallon (fuel efficiency)
2. `cyl`: Number of cylinders
3. `disp`: Displacement of the engine (in cubic inches)
4. `hp`: gross horsepower
5. `drat`: Back axle ratio wt: Weight (in thousands of pounds)
6. `wt`: Weight (in thousands of pounds)
7. `qsec`: 1/4 mile speed (in seconds)
8. `vs`: Type of engine (0 = V-shaped, 1 = straight)
9. `am`: Type of transmission (0 for automatic, 1 for manual)
10. `gear`: the number of forward gears
11. `carb`: the number of carburetors

```
data(mtcars)
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

5.3 Reading different file formats into a dataframe

1. We examine how to read data into a dataframe in R when the original data is stored in prominent file formats such as CSV, Excel, and Google Sheets.
2. Before learning how to accomplish this, it is necessary to comprehend how to configure the Working Directory in R.

5.3.1 Working Directory

1. The working directory is the location where R searches for and saves files by default.
2. By default, when we execute a script or import data into R, R will search the working directory for files.
3. Using R's `getwd()` function, we can examine our current working directory:

```
getwd()
```

```
[1] "/cloud/project"
```

4. We are running R in the Cloud and hence we are seeing that the working directory is specified as `/cloud/project/DataAnalyticsBook101`. If we are doing R programming on a local computer, and if our working directory is the Desktop, then we may see a different response such as `C:/Users/YourUserName/Desktop`.
5. Using R's `setwd()` function, we can change our current working directory. For example, the following code will set our working directory to the Desktop:

```
setwd("C:/Users/YourUserName/Desktop")
```

6. We should choose an easily-remembered and accessible working directory to store our R scripts and data files. Additionally, we should avoid using spaces, special characters, and non-ASCII characters in file paths, as these can cause file handling issues in R. [2]

5.3.2 Reading a CSV file into a dataframe

1. CSV is the abbreviation for "Comma-Separated Values." A CSV file is a plain text file that stores structured tabular data.
2. Each entry in a CSV file represents a record, whereas each column represents a field. The elements in each record are separated by commas (hence the name Comma-Separated Values), semicolons, or tabs.
3. Before proceeding ahead, it is imperative that the file that we wish to read is located in the Working Directory.
4. Suppose we wish to import a CSV file named `mtcars.csv`, located in the Working Directory. We can use the `read.csv()` function, illustrated as follows.

```
df_csv <- read.csv("mtcars.csv")
```

4. In this example, the `read.csv()` function reads the mtcars.csv file into a data frame named `df_csv`.
5. If the file is not in the current working directory, the complete file path must be specified in the `read.csv()` function argument; otherwise, an error will occur.

5.3.3 Reading an Excel (xlsx) file into a dataframe

1. Suppose we wish to import a Microsoft Excel file named `mtcars.xlsx`, located in the Working Directory.
2. We can use the `read_excel` function in the R package `readxl`, illustrated as follows.

```
library(readxl)
df_xlsx <- read_excel("mtcars.xlsx")
```

5.3.4 Reading a Google Sheet into a dataframe

1. Google Sheets is a ubiquitous cloud-based spreadsheet application developed by Google. It is a web-based application that enables collaborative online creation and modification of spreadsheets.
2. We can import data from a Google Sheet into a R dataframe, as follows.

- Consider a Google Sheet whose preferences have been set such that anyone can view it using its URL. If this is not done, then some authentication would become necessary.
- Every Google Sheet is characterized by a unique Sheet ID, embedded within the URL. For example, consider a Google Sheet containing some financial data concerning S&P500 index shares.
- Suppose the Sheet ID is: `1nm688a3GsPM5cadJIwu6zj336WBaduglY9TSTUaM9jk`
- We can use the function `gsheet2tbl` in package `gsheet` to read the Google Sheet into a dataframe, as demonstrated in the following code.

```
# Read recent S&P500 data that is posted in a Google Sheet.
library(gsheet)

prefix <- "https://docs.google.com/spreadsheets/d/"
sheetID <- "1nm688a3GsPM5cadJIwu6zj336WBaduglY9TSTUaM9jk"
suffix <- "/edit#gid=0"

# Form the URL to connect to
url <- paste(prefix, sheetID, suffix)

# Read the Google Sheet located at the URL into a dataframe called gf
gf <- gsheet2tbl(url)
```

```
No encoding supplied: defaulting to UTF-8.
```

- The first line imports the `gsheet` package required to access Google Sheets into R.
- The following three lines define URL variables for Google Sheets. The `prefix` variable contains the base URL for accessing Google Sheets, the `sheetID` variable contains the ID of the desired Google Sheet, and the `suffix` variable contains the URL's suffix.
- The `paste()` function is used to combine the `prefix`, `sheetID`, and `suffix` variables into a complete URL for accessing the Google Sheet.

- The `gsheet2tbl()` function from the `gsheet` package is then used to read the specified Google Sheet into a dataframe called `gf`.
- Once the preceding code is executed, the `gf` dataframe will contain the Google Sheet data, which can then be analyzed further in R.

5.3.5 Joining or Merging two dataframes

- Suppose we have a second S&P 500 data located in a second Google Sheet and suppose that we would like to join or merge the data in this dataframe with the above dataframe gf.
- The ID of this second sheet is: `1F5KvFATcehrdJuGjYVqppNYC9hEKSww9rXYHCk2g6OA`
- We can read the data present in this Google Sheet using the following code, similar to the one discussed above, using the following code.

```
# Read additional S&P500 data presend in another Google Sheet.
library(gsheet)

prefix <- "https://docs.google.com/spreadsheets/d/"
sheetID <- "1F5KvFATcehrdJuGjYVqppNYC9hEKSww9rXYHCk2g6OA"
suffix <- "/edit#gid=0"

# Form the URL to connect to
url <- paste(prefix, sheetID, suffix)

# Read the Google Sheet located at the URL into a dataframe called tv
tv <- gsheet2tbl(url)
```

```
No encoding supplied: defaulting to UTF-8.
```

- We now have two dataframes named `tv` and `gf` that we wish to merge or join.
- The two dataframes have a column named `Stock` in common, which will serve as the key.
- The following code illusrates how to merge two dataframes:

```
# merging dataframes
M.df <- merge(tv, gf , id = "Stock")
```

- We now have a new dataframe named `M.df`, which contains the data got from merging the two dataframes `tv` and `gf`.

5.4 Tibbles

1. A tibble is a contemporary and enhanced variant of a R data frame that is part of the tidyverse package collection.
2. Tibbles are created and manipulated using the dplyr package, which provides a suite of functions optimized for data manipulation.
3. The following characteristics distinguish a tibble from a conventional data frame:
4. Tibbles must always have unique, non-empty column names. Tibbles do not permit the creation or modification of columns using partial matching of column names. Tibbles improve the output of large datasets by displaying by default only a few rows and columns.
5. Tibbles have a more consistent behavior for subsetting, with the use of [[always returning a vector or NULL, and [] always returning a tibble.
6. Here is an example of using the tibble() function in dplyr to construct a tibble:

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
# Create a tibble
my_tibble <- tibble(
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 35),
  gender = c("F", "M", "M")
)

# Print the tibble
my_tibble
```

```
# A tibble: 3 x 3
  name      age gender
  <chr>   <dbl> <chr>
1 Alice      25 F
2 Bob        30 M
3 Charlie    35 M
```

7. This will generate a tibble consisting of three columns (name, age, and gender) and three rows of data. Note that the column names are preserved and the tibble is printed in a compact and legible manner.

5.4.1 Converting a dataframe into a tibble

```r
# Create a data frame
my_df <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 35),
  gender = c("F", "M", "M")
)

# Convert the data frame to a tibble
my_tibble <- as_tibble(my_df)

# Print the tibble
my_tibble
```

```
# A tibble: 3 x 3
  name      age gender
  <chr>   <dbl> <chr>
1 Alice      25 F
2 Bob        30 M
3 Charlie    35 M
```

8. This assigns the tibble representation of the data frame my_df to the variable my_tibble.
9. Note that the resulting tibble has the same column names and data as the original data frame, but has the additional characteristics and behaviors of a tibble.

5.4.2 Converting a tibble into a dataframe

```
library(dplyr)

# Convert the tibble to a data frame
my_df <- as.data.frame(my_tibble)

# Print the data frame
my_df
```

```
     name age gender
1   Alice  25      F
2     Bob  30      M
3 Charlie  35      M
```

10. A tibble offers several advantages over a data frame in R:

- Large datasets can be printed with greater clarity and precision using Tibbles. By default, they only print the first few rows and columns, making it simpler to read and comprehend the data structure.
- Better subsetting behavior: With [[always returning a vector or NULL and [] always returning a tibble, Tibbles have a more consistent subsetting behavior. This facilitates the subset and manipulation of data without unintended consequences.
- Consistent naming: Tibbles always have column names that are distinct and non-empty. This makes it simpler to refer to specific columns and prevents errors caused by duplicate or unnamed column names.
- More informative errors: Tibbles provides more informative error messages that make it simpler to diagnose and resolve data-related problems.
- Fewer surprises: Tibbles have more stringent constraints than data frames, resulting in fewer surprises and unexpected behavior when manipulating data.

5.5 References

[1]

R Core Team. (2021). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. **https://www.R-project.org/**

R Core Team. (2022). Vectors, Lists, and Arrays. R Documentation. https://cran.r-project.org/doc/manuals/r-release/R-intro.html#vectors-lists-and-arrays

Wickham, H., & Grolemund, G. (2016). R for data science: Import, tidy, transform, visualize, and model data. O'Reilly Media, Inc.

R Core Team. (2022, March 2). Data Frames. R Documentation. https://www.rdocumentation.org/packages/ba

[2]

OpenIntro. (2022). 1.3 RStudio and working directory. In Introductory Statistics with Randomization and Simulation (1st ed.). https://www.openintro.org/book/isrs/

R Core Team. (2021). getwd(): working directory; setwd(dir): change working directory. In R: A language and environment for statistical computing. R Foundation for Statistical Computing. https://stat.ethz.ch/R-manual/R-devel/library/base/html/getwd.html

R Core Team. (2021). getwd(): working directory; setwd(dir): change working directory. In R: A language and environment for statistical computing. R Foundation for Statistical Computing. https://stat.ethz.ch/R-manual/R-devel/library/base/html/setwd.html