

Lecture 1

Introduction to R

Dr. Abbas Maarooof
aimaarooof@gmail.com
AITRS-1 June 2021

Course Outline

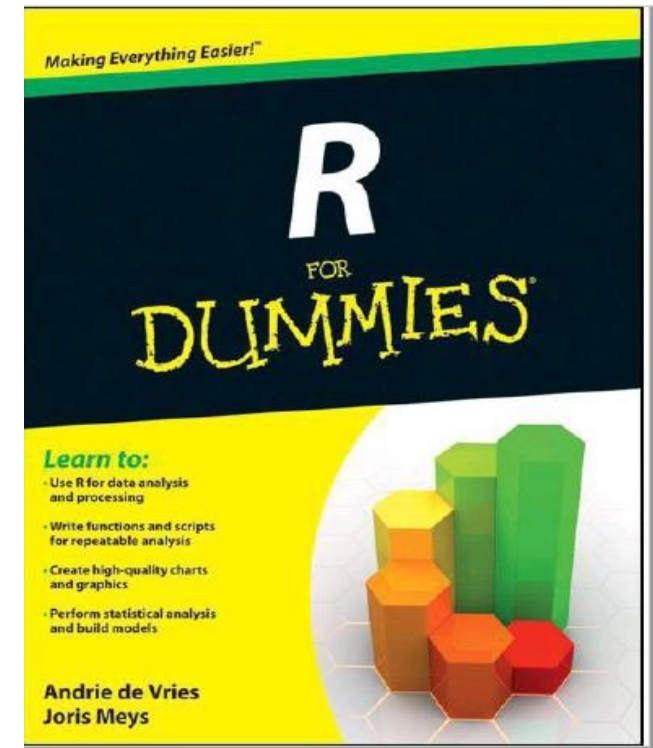
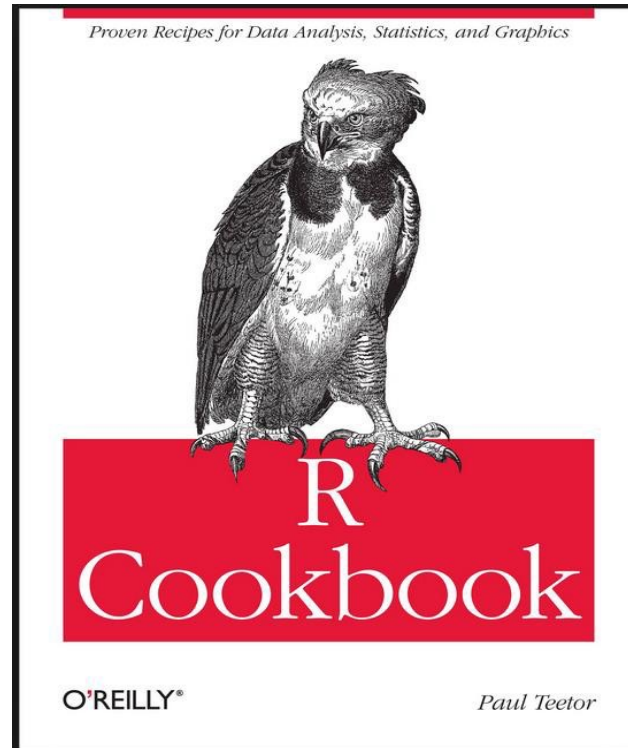
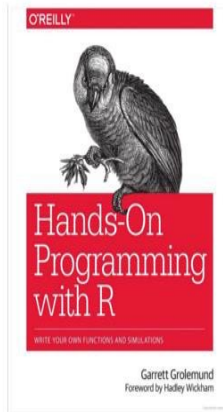
1. Introduction to R - Day #1
2. Fundamentals of R - Day #2
3. Visualizing Data with ggplot2 - Day #3

References

Hands-On Programming with R

Garrett Golemund

Foreword by Hadley Wickham



Hint: You can download these ebooks as a pdf files for free

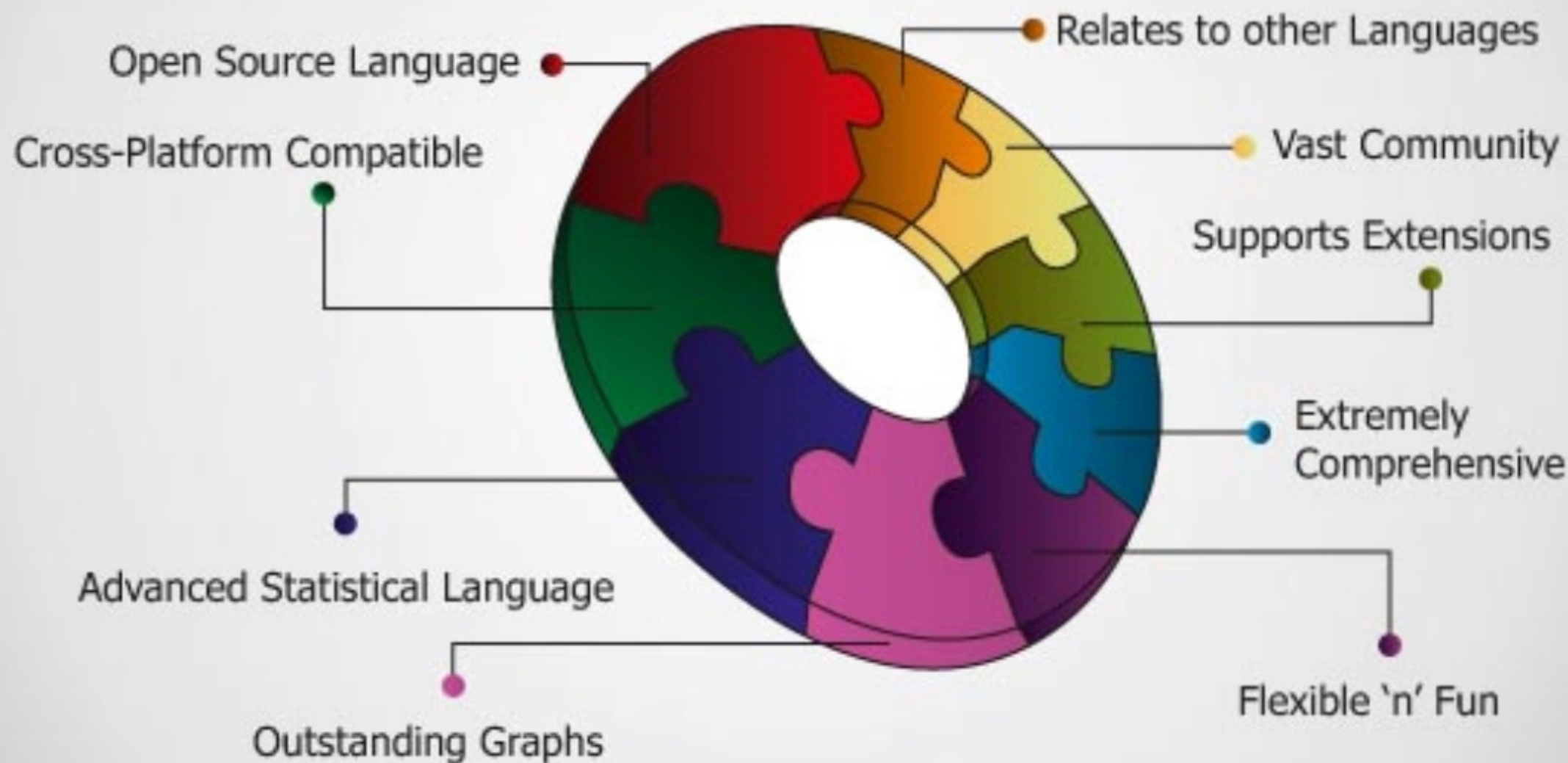
Why Learn R?

- R -the programming language favored by many statisticians because facilitate matrix arithmetic - carrying out complex, often automated calculations on data which is held in a grid of rows and columns.
- Named from the initials of the two men who first developed the language at Dept. of Statistics of University of Auckland, New Zealand during 1990s, **Robert Gentleman** and **Ross Ihaka**.
- R is very good for creating programs which can carry out calculations on these datasets, even when the datasets are constantly growing in size at an ever-increasing rate, and producing real-time visualizations based on this data.

- R is a computer programming language which is particularly well suited to handling and sorting the large datasets associated with Big Data projects.
- The software environment used to create code in R is **open sourced, meaning it is free to download**, anyone can use it, and there is a plethora of guidance and advice available on how to use it most effectively.
- R designers realized that **visualization** was key to being able to understand the complex datasets that are being explored, incorporated functionality to translate data into charts, graphs and complex multi-dimensional matrices - as well as many user-defined methods of visualization - into its core.

- Online, R code is everywhere although you won't see it, as it's always hidden behind pretty graphical interfaces. But when you use **Google, Facebook or Twitter** you are almost certainly executing R code running on the servers of those organizations.
- It is also capable of executing code written in other languages such as C++ or Java, so resources coded in those languages can be made available. Because it can be compiled to run on any major operating system, R code can easily be ported between Unix, Windows or Mac environments.
- With a reported more than **two million** users worldwide, and thousands of deployed applications created using it, **R is undoubtedly one of the backbone technologies of the Big Data revolution.**

Why Learn R?



Benefits of Using R

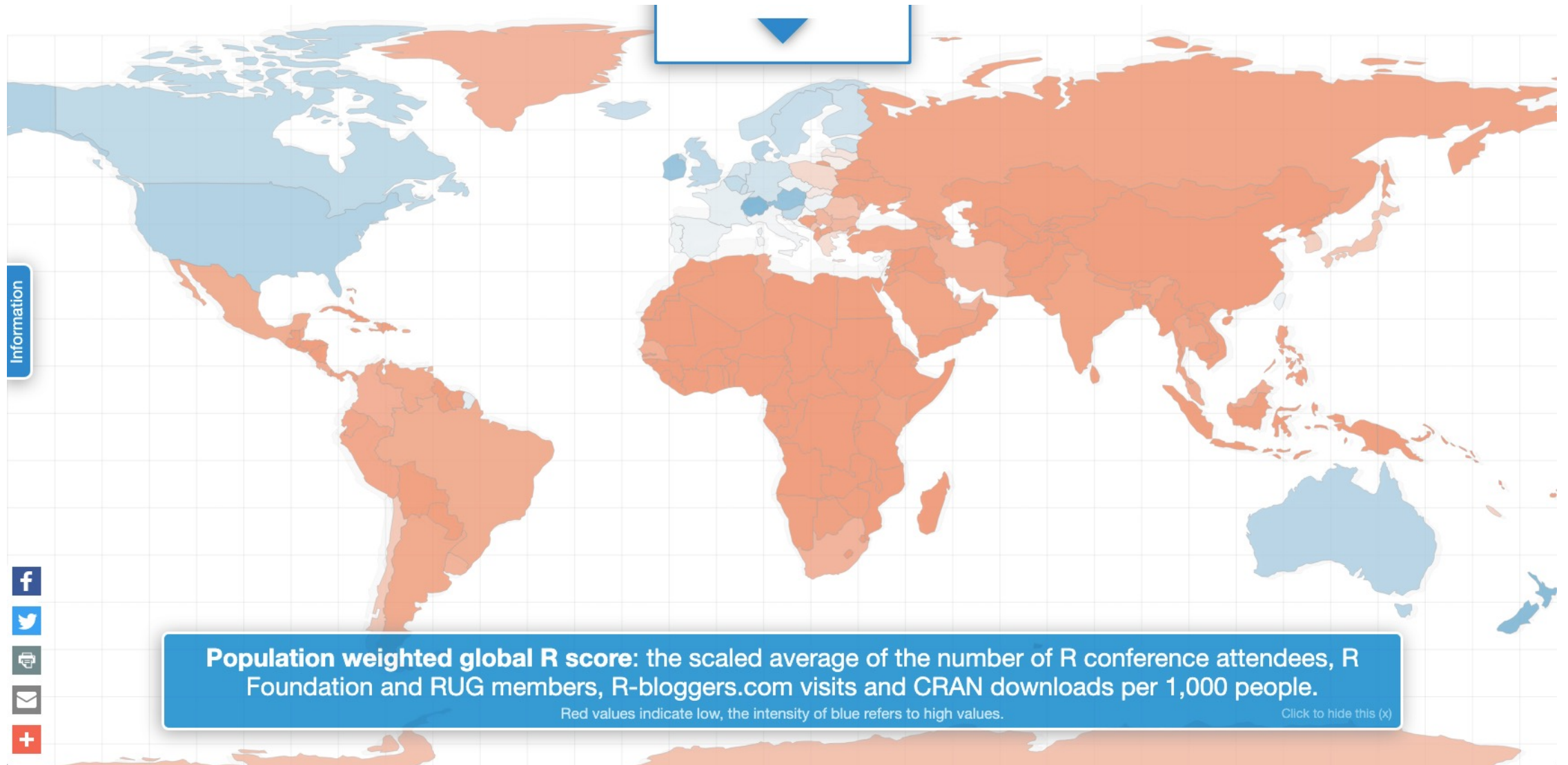
Here are some benefits I found after using R:

- The style of coding is quite easy.
- It's open source. No need to pay any subscription charges.
- Availability of instant access to over 17649 packages customized for various computation tasks.
- The community support is overwhelming. There are numerous forums to help you out.
- Get high performance computing experience (require packages)
- One of highly sought skill by analytics and data science companies.

Things R does and What R does not do

R does	R does not
<ul style="list-style-type: none">• Data handling and storage: numeric, textual• Matrix algebra• Has tables and regular• Expressions• high-level data analytic and statistical functions• classes (“Object Oriented”)• Graphics• programming language: loops, branching, Subroutines	<ul style="list-style-type: none">• Is not a database, but connects to DBMSs• has no graphical user interfaces, however it connects to Java, TclTk and it has R Studio• language interpreters are not fast. However, R could be extended by compiled C/C++ code• No spreadsheet view of data, but connects to MS Excel.• No professional / commercial Support but you can use help

A world map of R user activity



Statistical Packages

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors
- Most R packages deal with statistics and data analysis
- Many statistical researchers publish their state of the art methods as R packages.
- Comprehensive R Archive Network (CRAN) is a place where you can fetch those packages for free. You can get truly powerful tools at CRAN, you can find them at this link:
<https://cran.r-project.org/>

Looking At Some of the Unique Features of R

1. Performing multiple calculations with vectors

R is a vector-based language. You can think of a vector as a row or column of numbers or text. The list of numbers {1,2,3,4,5}, for example, could be a vector. Unlike most other programming languages, R allows you to apply functions to the whole vector in a single operation without the need for an explicit loop. We'll illustrate with some real R code. First, we'll assign the values 1:5 to a vector that we'll call x:

```
> x <- 1:5  
> x  
[1] 1 2 3 4 5
```

Next, we'll add the value 2 to each element in the vector x and print the result:

```
> x + 2  
[1] 3 4 5 6 7
```

You can also add one vector to another. To add the values 6:10 element-wise to x, you do the following:

```
> x + 6:10  
[1] 7 9 11 13 15
```

To do this in most other programming language would require an explicit loop to run through each value of x.

Looking At Some of the Unique Features of R

2. Processing more than just statistics

R was developed by statisticians to make statistical processing easier. This heritage continues, making R a very powerful tool for performing virtually any statistical computation.

The result is that R is now eminently suitable for a wide variety of nonstatistical tasks, including

Data processing,
Graphic visualization,
and analysis of all sorts

R is being used in

The fields of finance,
Natural language processing,
Genetics,
Biology,
Market research, to name just a few

Which means that you can
use R alone to program
anything you want

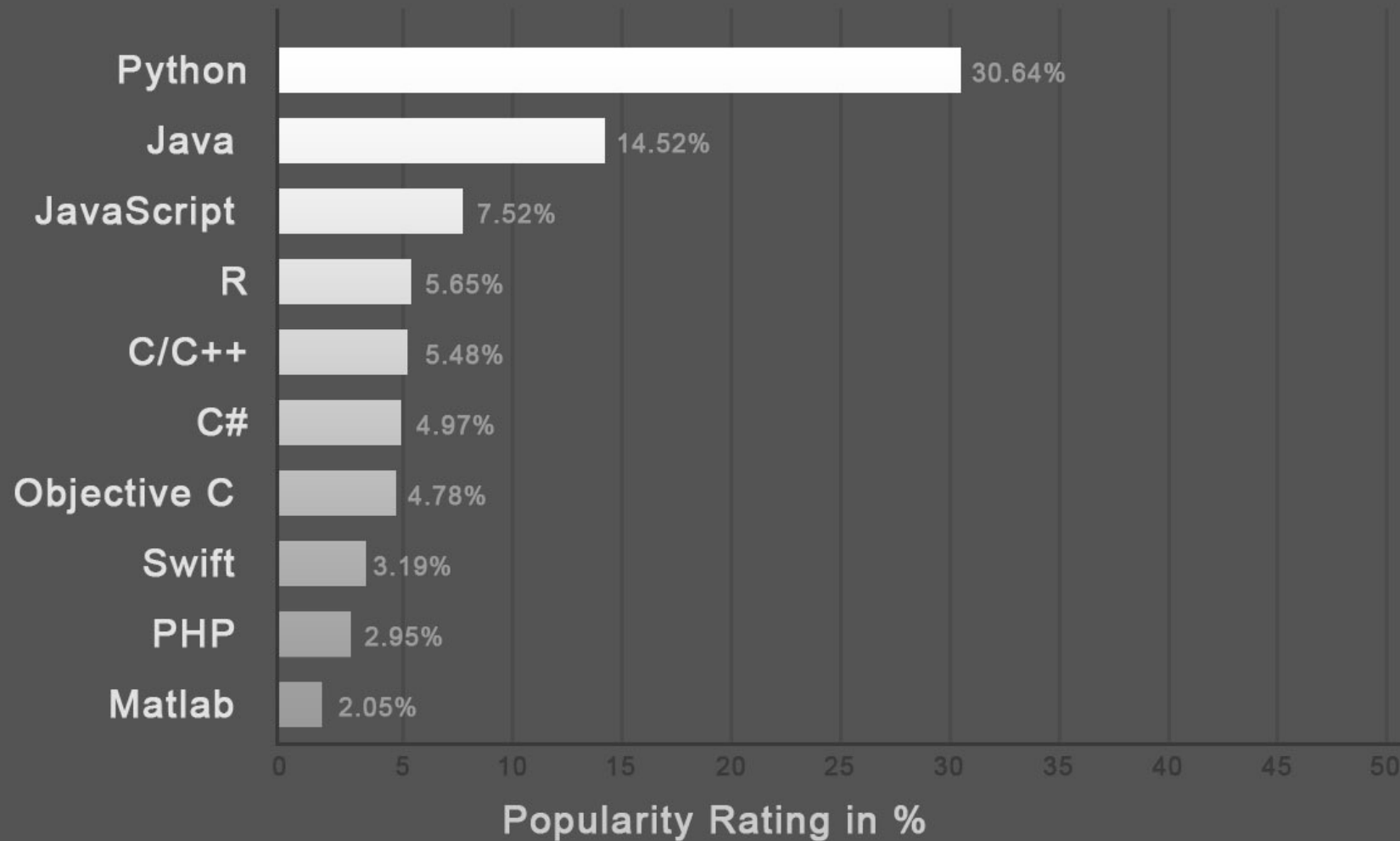
Looking At Some of the Unique Features of R

3. Running code without a compiler

R is an interpreted language, which means that — contrary to compiled languages like C and Java — you don't need a compiler to first create a program from your code before you can use it

The days of commercial statistical languages and packages such as **SAS**, **Stata** and **SPSS** are over,"

Popularity of Programming Language (PYPL) Ranking 2020



<devathon>

The days of commercial statistical languages and packages such as **SAS**, **Stata** and **SPSS** are over.

Any questions

How to install R and R Studio ?

Install R

Install R

To install R on your computer (legally for free!), go to the home website of R
Download R of Windows, Mac-OS or Linux from

<https://www.r-project.org/>

and do the following (assuming you work on a windows/Mac-OS computer):

1. Click download CRAN in the left bar
2. Choose a download site (normally 0-cloud) <https://cloud.r-project.org/>
3. Choose Windows as target operation system
4. Click base
5. Choose Download R 4.1.0 for Windows (86 megabytes, 32/64 bit) and
6. Choose default answers for all questions

Install R

- Download R of Windows, Mac-OS or Linux from <http://cran.r-project.org/>
- If you like command line interface, you do not need more than that.



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-05-18, Camp Pontanezen) [R-4.1.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

There are a lot of options for running R on your computer. Now, when you install R, it does have its own app, and you can open that and you can run commands.

This is one way to go.

I actually don't use this one very often, because it opens up several different windows and also because the keyboard commands

```
R Console

R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

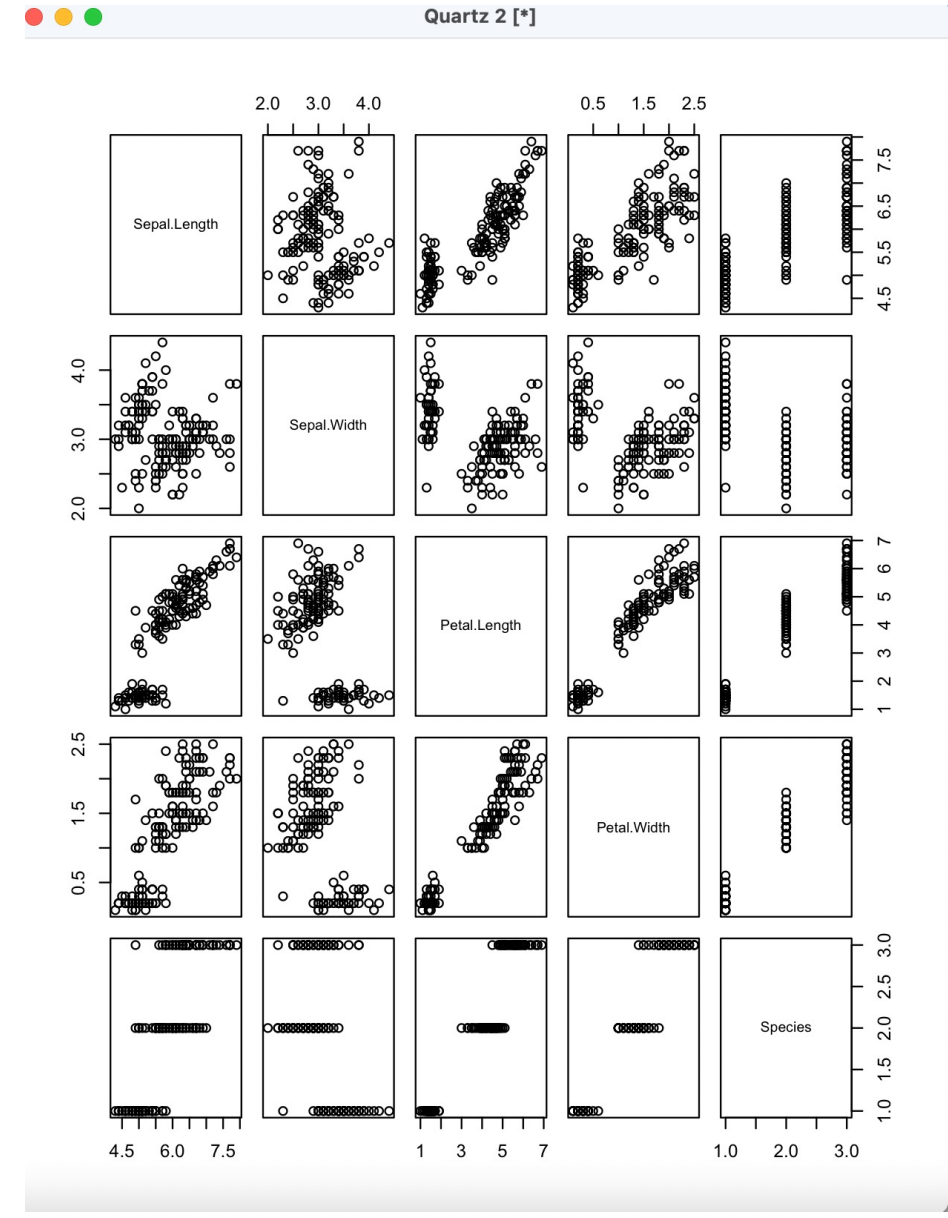
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
[R.app GUI 1.73 (7892) x86_64-apple-darwin17.0]

WARNING: You're using a non-UTF8 locale, therefore only ASCII characters will work.
Please read R for Mac OS X FAQ (see Help) section 9 and adjust your system preferences accordingly.
[Workspace restored from /Users/admin/.RData]
[History restored from /Users/admin/.Rapp.history]

> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> plot(iris)
> |
```



Environment for R

Install R Studio

- If you prefer an integrated development environment (IDE), an IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion. download **R Studio** from <http://www.rstudio.com>
- You will keep on fetching packages (libraries) from the CRAN <http://cran.r-project.org/> site.
- Run R installation first, then install R Studio. That is all.
- When we use R, we will use R Studio, except in rare circumstances.

Install R Studio

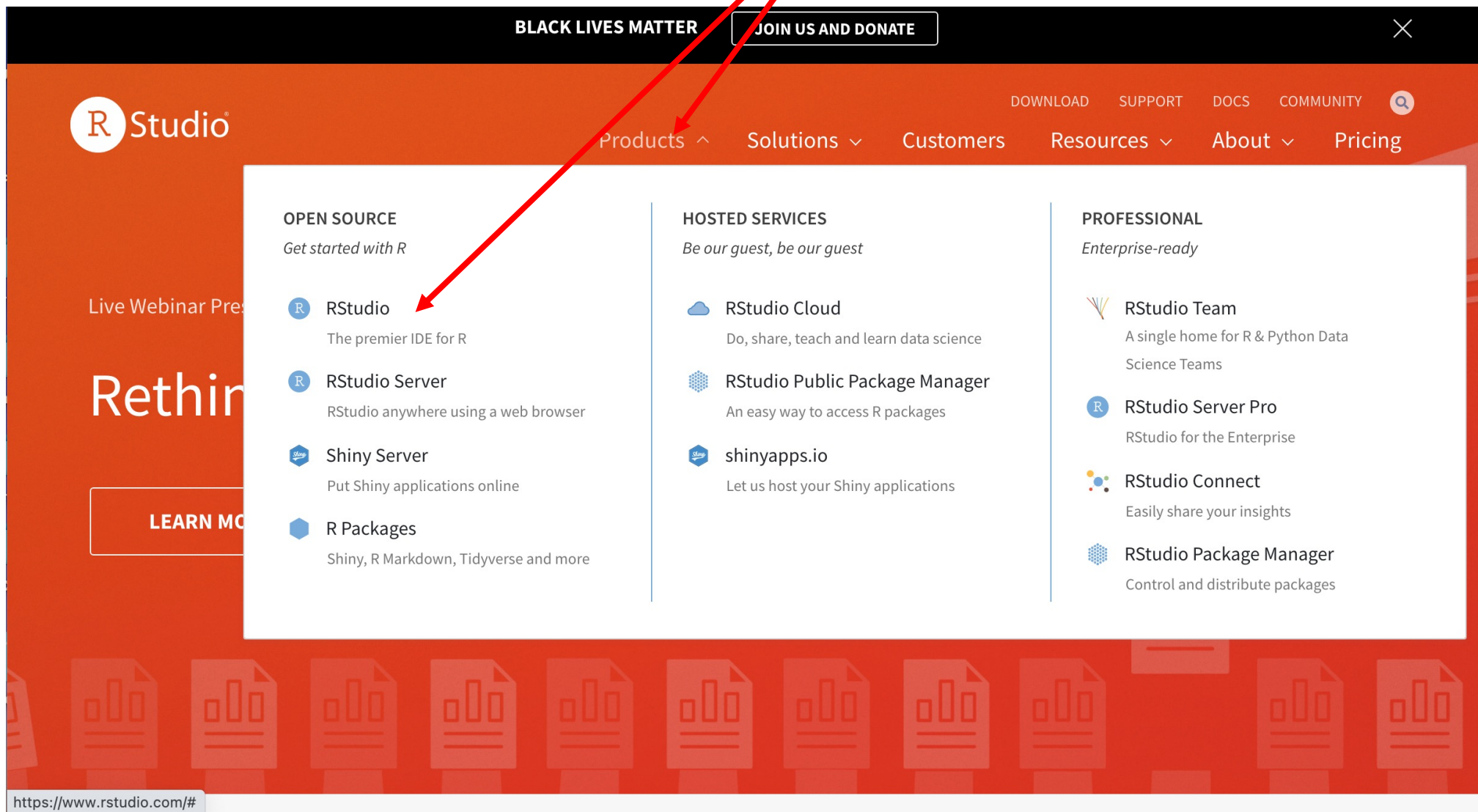
To install RStudio, go to:

<http://www.rstudio.com>

And do the following (assuming you work on a windows computer):

1. Click Download RStudio
2. Click Download RStudio Desktop
3. Click Recommended For Your System
4. Download the .exe file and run it (choose default answers for all questions)

Step#1 <http://www.rstudio.com/>



The screenshot shows the RStudio website homepage. At the top, there is a black banner with the text "BLACK LIVES MATTER" and a button "JOIN US AND DONATE". Below this is an orange navigation bar with the RStudio logo on the left and links for "Products", "Solutions", "Customers", "Resources", "About", and "Pricing" on the right. A search icon is also present. The "Products" link is highlighted with a red arrow. Below the navigation bar, the page is divided into three columns: "OPEN SOURCE", "HOSTED SERVICES", and "PROFESSIONAL". The "OPEN SOURCE" column lists "RStudio" (The premier IDE for R), "RStudio Server" (RStudio anywhere using a web browser), "Shiny Server" (Put Shiny applications online), and "R Packages" (Shiny, R Markdown, Tidyverse and more). The "HOSTED SERVICES" column lists "RStudio Cloud" (Do, share, teach and learn data science), "RStudio Public Package Manager" (An easy way to access R packages), and "shinyapps.io" (Let us host your Shiny applications). The "PROFESSIONAL" column lists "RStudio Team" (A single home for R & Python Data Science Teams), "RStudio Server Pro" (RStudio for the Enterprise), "RStudio Connect" (Easily share your insights), and "RStudio Package Manager" (Control and distribute packages). A red arrow points from the "Products" link in the navigation bar to the "RStudio" link in the "OPEN SOURCE" column. The background of the page features a pattern of faint RStudio logos.

BLACK LIVES MATTER [JOIN US AND DONATE](#)

R Studio

DOWNLOAD SUPPORT DOCS COMMUNITY

Products Solutions Customers Resources About Pricing

OPEN SOURCE
Get started with R

- RStudio**
The premier IDE for R
- RStudio Server**
RStudio anywhere using a web browser
- Shiny Server**
Put Shiny applications online
- R Packages**
Shiny, R Markdown, Tidyverse and more

HOSTED SERVICES
Be our guest, be our guest

- RStudio Cloud**
Do, share, teach and learn data science
- RStudio Public Package Manager**
An easy way to access R packages
- shinyapps.io**
Let us host your Shiny applications

PROFESSIONAL
Enterprise-ready

- RStudio Team**
A single home for R & Python Data Science Teams
- RStudio Server Pro**
RStudio for the Enterprise
- RStudio Connect**
Easily share your insights
- RStudio Package Manager**
Control and distribute packages

<https://www.rstudio.com/#>

Step#2

New in **RStudio 1.4**

Love what you code, from faster development with our Visual Markdown Editor, new Python capabilities, and a host of quality of life improvements, helping to bridge the gap between tools and teams.

[Read the Announcement](#)

There are two versions of RStudio:



RStudio Desktop

Run RStudio on your desktop



RStudio Server

Centralize access and computation

Step#3

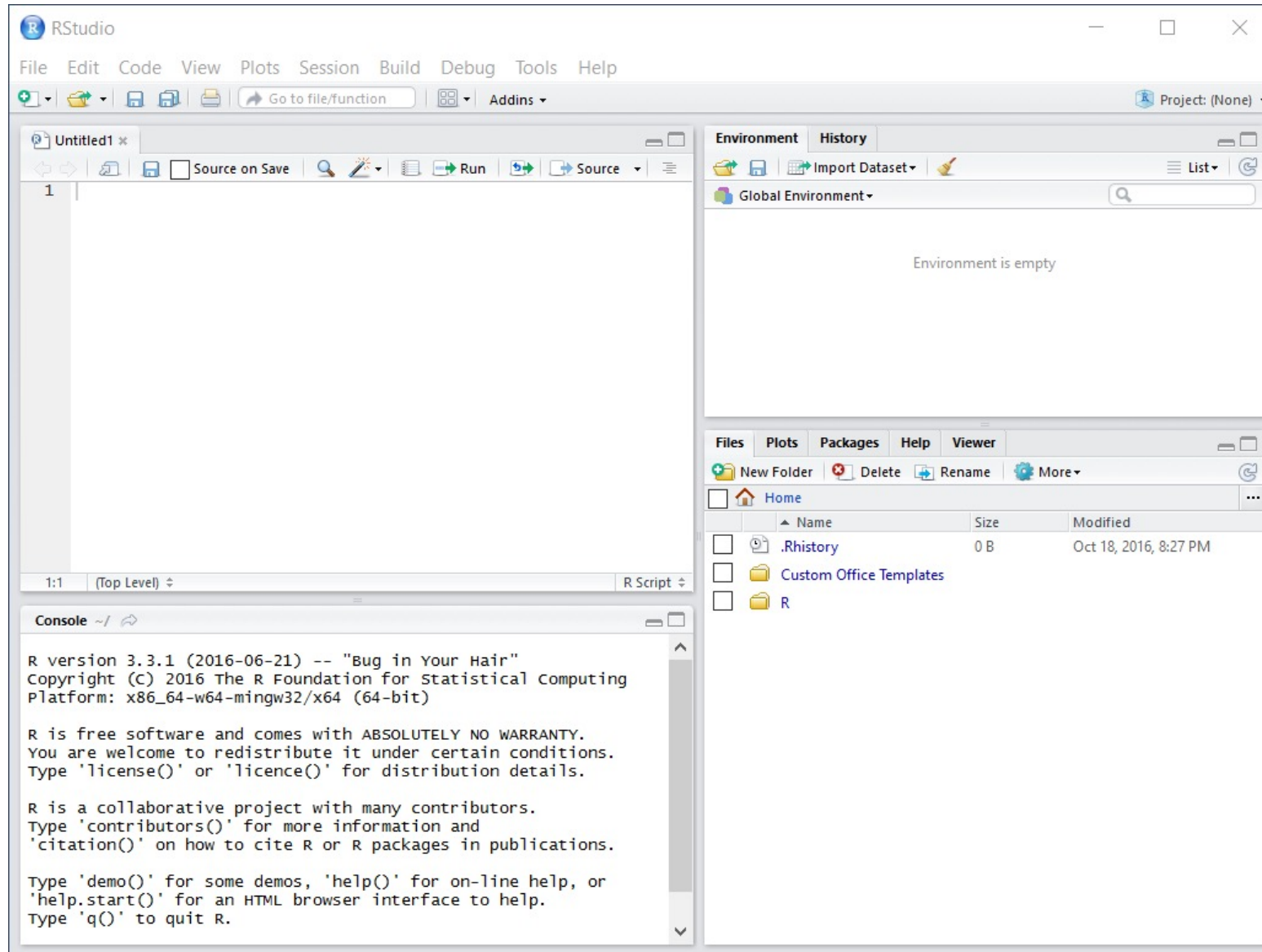


	Open Source Edition	RStudio Desktop Pro
Overview	<ul style="list-style-type: none">• Access RStudio locally• Syntax highlighting, code completion, and smart indentation• Execute R code directly from the source editor• Quickly jump to function definitions• View content changes in real-time with the Visual Markdown Editor• Easily manage multiple working directories using projects• Integrated R help and documentation• Interactive debugger to diagnose and fix errors• Extensive package development tools	<p>All of the features of open source; plus:</p> <ul style="list-style-type: none">• A commercial license for organizations not able to use AGPL software• Access to priority support• RStudio Professional Drivers• Connect directly to your RStudio Server Pro instance remotely
Support	Community forums only	<ul style="list-style-type: none">• Priority Email Support• 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	\$995/year

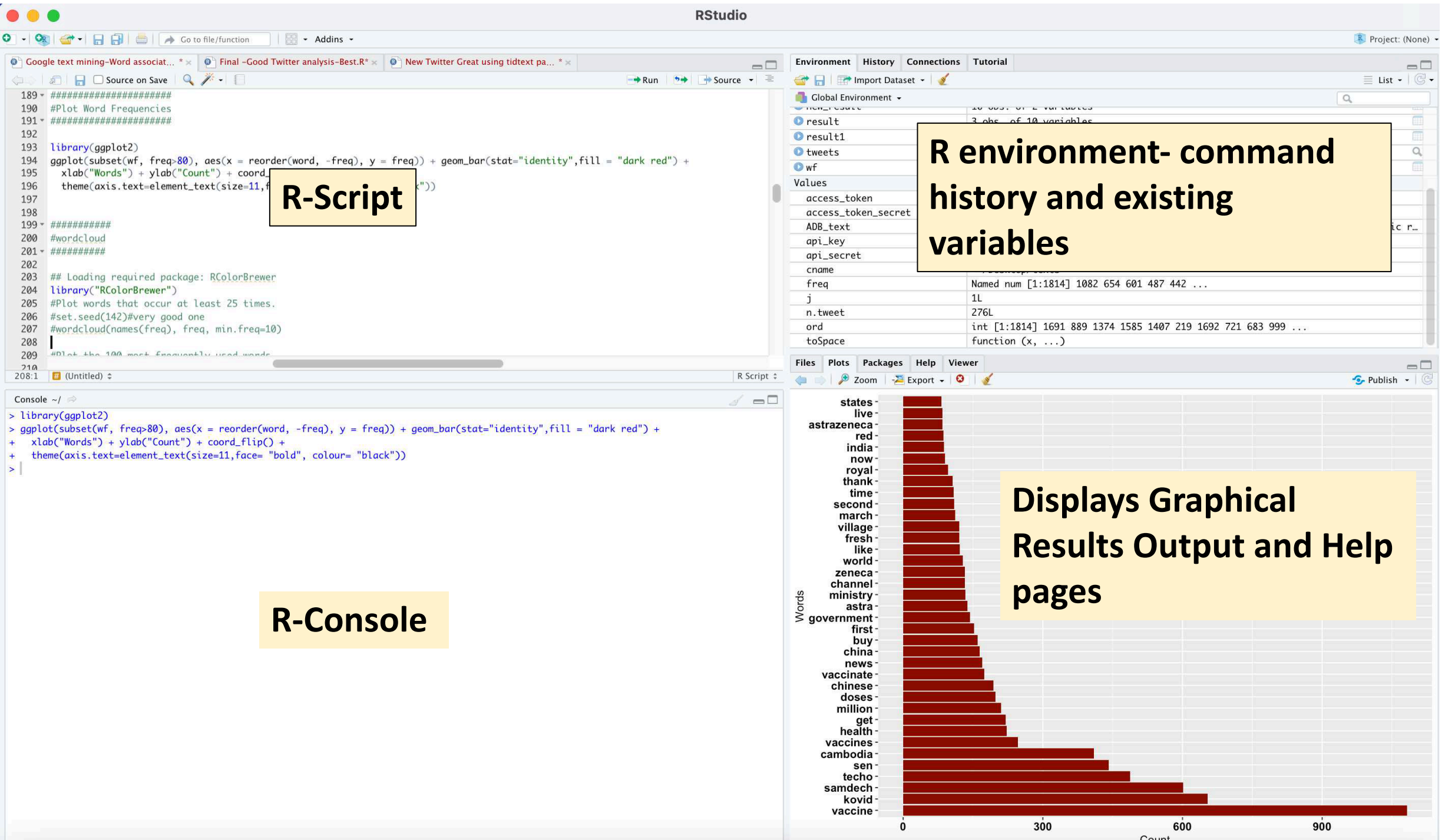
DOWNLOAD RSTUDIO DESKTOP

DOWNLOAD FREE RSTUDIO DESKTOP PRO TRIAL

R Studio



Navigating the RStudio Environment



Let's quickly understand the interface of R Studio:

R Console: This area shows the output of code you run. Also, you can directly write codes in console. Code entered directly in R console cannot be traced later. This is where R script comes to use. **Hint** : Ctl I to clear the screen of the Console.

R Script: As the name suggest, here you get space to write codes. To run those codes, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' button location at top right corner of R Script.

R environment: This space displays the set of external elements added and command history and existing variables. This includes data set, variables, vectors, functions etc. To check if data has been loaded properly in R, always look at this area.

Graphical Output: This space display the graphs created during exploratory data analysis. Not just graphs, you could select packages, seek help with embedded R's official documentation. Hit Help tab to get to the main help page.

Hint in running R

You can run R using a number of text editors or “integrated development environments” (IDEs). Most people prefer some other application than R’s native environment, which provides only limited functionality in terms of syntax highlighting, auto-completion, and debugging. Alternatives include *RStudio* and *Emacs/ESS*. I very much prefer the latter, but if you’ve never programmed before I would go with RStudio (installation instructions [here](#); note that you need both R and RStudio).

All IDEs include a console and a text editor. The console is where you’ll see the results (or output) of commands executed from the editor. You can type commands directly into the console, but this is generally not a good strategy. This is because the whole purpose of writing code is to make it reproducible. Typing commands in the text editor will let you come back to them later as long as you save them (using extension `.R`).

5. Packages and Help Pages

Statistical Packages

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors
- Most R packages deal with statistics and data analysis
- Many professors, programmers, and statisticians use R to design tools that can help people analyze data. They then make these tools free for anyone to use. To use these tools, you just have to download them. They come as preassembled collections of functions and objects called packages
- Comprehensive R Archive Network (CRAN) is a place where you can fetch those packages for free. You can get truly powerful tools at CRAN. you can find them at this link:
<https://cran.r-project.org/>



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

Click

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-05-18, Camp Pontanezen) [R-4.1.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.



[CRAN](#)
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

[About R](#)
[R Homepage](#)
[The R Journal](#)

[Software](#)
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

[Documentation](#)
[Manuals](#)
[FAQs](#)
[Contributed](#)

Contributed Packages

Available Packages

Currently, the CRAN package repository features 17648 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual [R Installation and Administration](#) (also contained in the R base sources) explains the process in detail.

[CRAN Task Views](#) allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

Package Check Results

All packages are tested regularly on machines running [Debian GNU/Linux](#), [Fedora](#), macOS (formerly OS X), Solaris and Windows.

The results are summarized in the [check summary](#) (some [timings](#) are also available). Additional details for Windows checking and building can be found in the [Windows check summary](#).

Writing Your Own Packages

The manual [Writing R Extensions](#) (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Repository Policies

The manual [CRAN Repository Policy \[PDF\]](#) describes the policies in place for the CRAN package repository.

Related Directories

[Archive](#)

Previous versions of the packages listed above, and other packages formerly available.

[Orphaned](#)

Packages with no active maintainer, see the corresponding [README](#).

[bin/windows/contrib](#)

Windows binaries of contributed packages

[bin/macosx/contrib](#)

macOS High Sierra binaries of contributed packages

[bin/macosx/el-capitan/contrib](#)

OS X El Capitan binaries of contributed packages



[CRAN](#)

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

[About R](#)

[R Homepage](#)

[The R Journal](#)

[Software](#)

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

[Documentation](#)

[Manuals](#)

[FAQs](#)

[Contributed](#)

Available CRAN Packages By Name

[A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)

[A3](#)

[aaSEA](#)

[AATtools](#)

[ABACUS](#)

[abbyyR](#)

[abc](#)

[abc.data](#)

[ABC.RAP](#)

[abcADM](#)

[ABCanalysis](#)

[abcdeFBA](#)

[ABCOptim](#)

[ABCp2](#)

[abcrf](#)

[abcrlda](#)

[abctools](#)

[abd](#)

[abdiv](#)

[abe](#)

[abess](#)

[abf2](#)

[abglasso](#)

[ABHgenotypeR](#)

[abind](#)

[abjutils](#)

[abn](#)

[abnormality](#)

[abodOutlier](#)

[ABPS](#)

[abstractr](#)

[abtest](#)

[Ac3net](#)

[ACA](#)

[academictwitterR](#)

[acc](#)

[accelerometry](#)

[accelmissing](#)

Accurate, Adaptable, and Accessible Error Metrics for Predictive Models

Amino Acid Substitution Effect Analyser

Reliability and Scoring Routines for the Approach-Avoidance Task

Apps Based Activities for Communicating and Understanding Statistics

Access to Abbyy Optical Character Recognition (OCR) API

Tools for Approximate Bayesian Computation (ABC)

Data Only: Tools for Approximate Bayesian Computation (ABC)

Array Based CpG Region Analysis Pipeline

Fit Accumulated Damage Models and Estimate Reliability using ABC

Computed ABC Analysis

ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package

Implementation of Artificial Bee Colony (ABC) Optimization

Approximate Bayesian Computational Model for Estimating P2

Approximate Bayesian Computation via Random Forests

Asymptotically Bias-Corrected Regularized Linear Discriminant Analysis

Tools for ABC Analyses

The Analysis of Biological Data

Alpha and Beta Diversity Measures

Augmented Backward Elimination

Adaptive Best Subset Selection in Polynomial Time

Load Gap-Free Axon ABF2 Files

Adaptive Bayesian Graphical Lasso

Easy Visualization of ABH Genotypes

Combine Multidimensional Arrays

Useful Tools for Jurimetrical Analysis Used by the Brazilian Jurimetrics Association

Modelling Multivariate Data with Additive Bayesian Networks

Measure a Subject's Abnormality with Respect to a Reference Population

Angle-Based Outlier Detection

The Abnormal Blood Profile Score to Detect Blood Doping

An R-Shiny Application for Creating Visual Abstracts

Bayesian A/B Testing

Inferring Directional Conservative Causal Core Gene Networks

Abrupt Change-Point or Aberration Detection in Point Series

Access the Twitter Academic Research Product Track V2 API Endpoint

Exploring Accelerometer Data

Functions for Processing Accelerometer Data

Missing Value Imputation for Accelerometer Data

Important: Install packages from (almost) anywhere The devtools R package makes it easy to install packages from locations other than the CRAN website. devtools provides functions like `install_github`, `install_gitorious`, `install_bitbucket`, and `install_url`. These work similar to `install.packages`, but they search new locations for R packages. `install_github` is especially useful because many R developers provide development versions of their packages on GitHub. The development version of a package will contain a sneak peek of new functions and patches but may not be as stable or as bug free as the CRAN version.

Important: What's the best way to learn about R packages?

It is difficult to use an R package if you don't know that it exists. You could go to the CRAN website and click the Packages link to see a list of available packages, but you'll have to wade through thousands of them. Moreover, many R packages do the same things. How do you know which package does them best? The R-packages mailing list is a place to start. It sends out announcements of new packages and maintains an archive of old announcements. Blogs that aggregate posts about R can also provide valuable leads. I recommend www.r-bloggers.com**[R-bloggers]**. RStudio maintains a list of some of the most useful R packages in the Getting Started section of <http://support.rstudio.com>. Finally, CRAN groups together some of the most useful—and most respected—packages by subject area. This is an excellent place to learn about the packages designed for your area of work.

How to install R packages?

Installing Packages:

The sheer power of R lies in its incredible packages. In R, most data handling tasks can be performed in 2 ways: Using R packages and R base functions. In this course, I'll also introduce you with the most handy and powerful R packages. To install a package, simply type:

```
install.packages("package name")
```

As a first time user, a pop might appear to select your CRAN mirror (country server), choose accordingly and press OK.

Note: You can type this either in console directly and press 'Enter' or in R script and click 'Run'.

Loading Packages:

Installing a package doesn't immediately place its functions at your fingertips. It just places them on your computer. To use an R package, you next have to load it in your R session with the command:

```
library(package name)
```

Updating R Packages:

For example if you already have ggplot2, reshape2, and dplyr on your computer, it'd be a good idea to check for updates before you use them:

```
update.packages(c("ggplot2", "reshape2", "dplyr"))
```

install.packages

Each R package is hosted at <http://cran.r-project.org>, the same website that hosts R.

However, you don't need to visit the website to download an R package; you can download packages straight from R's command line. Here's how:

1. Open RStudio.
2. Make sure you are connected to the Internet.
3. Run `install.packages("ggplot2")` at the command line (console)

Example of Packages

We're going to use the **qplot function** to make some quick plots. qplot comes in the **ggplot2 package**, a popular package for making graphs. **Before you can use qplot, or anything else in the ggplot2 package, you need to download and install it.**

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

also installing the dependencies 'stringi', 'magrittr',
'colorspace', 'Rcpp', 'stringr', 'RColorBrewer', 'dichromat',
'munsell', 'labeling', 'assertthat', 'digest', 'gtable', 'plyr',
'reshape2', 'scales', 'tibble', 'lazyeval'

Do you want to install from sources the packages which need
compilation?

y/n: y

Library

Installing a package doesn't place its functions at your fingertips just yet: it simply places them in your hard drive. To use an R package, you next have to load it in your R session with the command **library("ggplot2")**. If you would like to load a different package, replace ggplot2 with your package name in the code.

Package that brings in a whole bunch of other packages with it

Example important Package is

The tidyverse

These packages are for data science to make you work easier, and more efficient



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

tidyverse: Easily Install and Load the 'Tidyverse'

The 'tidyverse' is a set of packages that work in harmony because they share common data representations and 'API' design. This package is designed to make it easy to install and load multiple 'tidyverse' packages in a single step. Learn more about the 'tidyverse' at <https://www.tidyverse.org>.

Version: 1.3.1

Depends: R (≥ 3.3)

Imports: [broom](#) (≥ 0.7.6), [cli](#) (≥ 2.4.0), [crayon](#) (≥ 1.4.1), [dbplyr](#) (≥ 2.1.1), [dplyr](#) (≥ 1.0.5), [dtplyr](#) (≥ 1.1.0), [forcats](#) (≥ 0.5.1), [googledrive](#) (≥ 1.0.1), [googlesheets4](#) (≥ 0.3.0), [ggplot2](#) (≥ 3.3.3), [haven](#) (≥ 2.3.1), [hms](#) (≥ 1.0.0), [httr](#) (≥ 1.4.2), [jsonlite](#) (≥ 1.7.2), [lubridate](#) (≥ 1.7.10), [magrittr](#) (≥ 2.0.1), [modelr](#) (≥ 0.1.8), [pillar](#) (≥ 1.6.0), [purrr](#) (≥ 0.3.4), [readr](#) (≥ 1.4.0), [readxl](#) (≥ 1.3.1), [reprex](#) (≥ 2.0.0), [rlang](#) (≥ 0.4.10), [rstudioapi](#) (≥ 0.13), [rvest](#) (≥ 1.0.0), [stringr](#) (≥ 1.4.0), [tibble](#) (≥ 3.1.0), [tidyr](#) (≥ 1.1.3), [xml2](#) (≥ 1.3.2)

Suggests: [covr](#), [feather](#), [glue](#), [knitr](#), [rmarkdown](#), [testthat](#)

Published: 2021-04-15

Author: Hadley Wickham [aut, cre], RStudio [cph, fnd]

Maintainer: Hadley Wickham <hadley at rstudio.com>

BugReports: <https://github.com/tidyverse/tidyverse/issues>

License: MIT + file LICENSE

URL: <https://tidyverse.tidyverse.org>, <https://github.com/tidyverse/tidyverse>

NeedsCompilation: no

Citation: [tidyverse citation info](#)

Materials: [README NEWS](#)

CRAN checks: [tidyverse results](#)

Downloads:

Reference manual: [tidyverse.pdf](#)

Vignettes: [The tidy tools manifesto](#)
[Welcome to the tidyverse](#)

Package source: [tidyverse_1.3.1.tar.gz](#)

Windows binaries: r-devel: [tidyverse_1.3.1.zip](#), r-release: [tidyverse_1.3.1.zip](#), r-oldrel: [tidyverse_1.3.1.zip](#)

macOS binaries: r-release: [tidyverse_1.3.1.tgz](#), r-oldrel: [tidyverse_1.3.1.tgz](#)

Old sources: [tidyverse archive](#)

Reverse dependencies:

Reverse depends: [div](#), [NPBayesImputeCat](#), [optimos.prime](#), [qqr](#), [tidyMicro](#), [Tushare](#), [WinRatio](#), [wrappedtools](#)

Reverse imports: [APCI](#), [archetypes](#), [baystability](#), [bioOED](#), [cartools](#), [catenary](#), [catmaply](#), [cgmquantify](#), [DAPAR](#), [dce](#), [diffman](#), [dplyrAssist](#), [drhur](#), [drimmR](#), [dumbbell](#), [eda4trecR](#), [epca](#), [eph](#), [epitweetr](#), [fructimadapt](#), [ggplotAssist](#), [hdpGLM](#), [hettx](#), [hlaR](#), [IPWpn](#), [matman](#), [MedLEA](#), [metaprotr](#), [mosaicModel](#), [nonet](#), [OpenRepGrid.ic](#), [packDAMipd](#), [penalizedclr](#), [predictorR](#), [projpred](#), [PSS.Health](#), [puzzle](#), [RanglaPunjab](#), [riskCommunicator](#), [rticulate](#), [sdmApp](#), [ShapleyValue](#), [ShellChron](#), [sim2Dpredictr](#), [SIPDIBGE](#), [SPARSEMODr](#), [statVisual](#), [studyStrap](#), [SurvHiDim](#), [SurvIMChd](#), [synergyfinder](#), [UniprotR](#), [usfertilizer](#), [vannstats](#)

Reverse suggests: [admixr](#), [ANCOMBC](#), [anomalize](#), [babsim.hospital](#), [bayesmodels](#), [biogrowth](#), [BRDT](#), [causalCmprsk](#), [CelliD](#), [Clustering](#), [clusterPower](#), [ClusTorus](#), [convergEU](#), [covid19br](#), [COVIDIBGE](#), [DecomposeR](#), [DescrTab2](#), [distributionsrd](#), [dm](#), [eechidna](#), [episensr](#), [flpr](#), [fma](#), [fobitools](#), [fpp2](#), [garchmodels](#), [garma](#), [genesysr](#), [genogeographer](#), [geodiv](#), [GetQuandlData](#), [ggmulti](#), [gMOIP](#), [greta](#), [HSAR](#), [hypeR](#), [ibawds](#), [insee](#), [InteractiveComplexHeatmap](#),

tidyverse website

Packages



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

Tidyverse

[Packages](#)

[Blog](#)

[Learn](#)

[Help](#)

[Contribute](#)



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the ~~complete~~ tidyverse with:

```
install.packages("tidyverse")
```

Learn the tidyverse

See how the tidyverse makes data science



ggplot2

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details. [Go to docs...](#)



dplyr

dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges. [Go to docs...](#)



tidyr

tidyr provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable. [Go to docs...](#)



readr

readr provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes. [Go to docs...](#)



purrr

purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. Once you master the basic concepts, purrr allows you to replace many for loops with code that is easier to write and more expressive. [Go to docs...](#)



tibble

tibble is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles are data.frames that are lazy and surly; they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code. [Go to docs...](#)



stringr

stringr provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations. [Go to docs...](#)



forcats

forcats provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values. [Go to docs...](#)

Contents

[Installation and use](#)[Core tidyverse](#)[Import](#)[Wrangle](#)[Program](#)[Model](#)[Get help](#)

All Packages within tidyverse Package

Exploring R (Examples)

Exploring R

To open RStudio, click the RStudio icon in your menu system or on your desktop,
Once RStudio started, choose File⇒New⇒R Script.

Source: The top-left corner of the screen contains a text editor that lets you work with source script files. Here, you can enter multiple lines of code, save your script file to disk, and perform other tasks on your script. This code editor works a bit like every other text editor you've ever seen, but it's smart. It recognizes and highlights various elements of your code, for example (using different colors for different elements), and it also helps you find matching brackets in your scripts.

Console: In the bottom-left corner, you find the console. The console in Rstudio is identical to the console in RGui. This is where you do all the interactive work with R.

Workspace and history: The top-right corner is a handy overview of your workspace, where you can inspect the variables you created in your session, as well as their values. This is also the area where you can see a history of the commands you've issued in R.

Files, plots, package, and help: In the bottom-right corner, you have access to several tools:

- **Files:** This is where you can browse the folders and files on your computer.
- **Plots:** This is where R displays your plots (charts or graphs). We discuss plots in Part V.
- **Packages:** This is where you can view a list of all the installed packages. A package is self-contained set of code that adds functionality to R, similar to the way that an add-in adds functionality to Microsoft Excel.
- **Help:** This is where you can browse the built-in Help system of R.

Starting Your First R Session

Start a new R session, type the following in your console, and press Enter:

```
> print("Hello world!")
```

R responds immediately with this output:

```
[1] "Hello world!"
```

Doing simple math

Type the following in your console to calculate the sum of five numbers:

```
> 1+2+3+4+5
```

```
[1] 15
```

Using vectors

A vector is the simplest type of data structure in R. To construct a vector, type the following in the console:

```
> c(1,2,3,4,5)  
[1] 1 2 3 4 5
```

In constructing a vector, you tell the **c() function** to construct a vector with the first five integers. The entries inside the parentheses are referred to as ***arguments***.

One very handy operator is called sequence, and it looks like a colon (:). Type the following in your console:

```
> 1:5  
[1] 1 2 3 4 5
```

Type the following in your console to calculate the sum of this vector:

```
> sum(1:5)
```

```
[1] 15
```

Storing and calculating values

A much more useful capability is storing values and then doing calculations on these stored values.

Try the following:

```
> x <- 1:5
```

```
> x
```

```
[1] 1 2 3 4 5
```

In these two lines of code, you first assign the sequence 1:5 to a variable called x. Then you ask R to print the value of x by typing x in the console and pressing Enter.

Hint: In R, the assignment operator is <-, which you type in the console by using two keystrokes: the less-than symbol (<) followed by a hyphen (-). The combination of these two symbols represents assignment

In addition to retrieving the value of a variable, you can do calculations on that value. Create a second variable called `y`, and assign it the value 10. Then add the values of `x` and `y`, as follows:

```
> y <- 10
```

```
> x + y
```

```
[1] 11 12 13 14 15
```

The values of the two variables themselves don't change unless you assign a new value. You can check this by typing the following:

```
> x
```

```
[1] 1 2 3 4 5
```

```
> y
```

```
[1] 10
```

Now create a new variable `z`, assign it the value of `x+y`, and print its value:

```
> z <- x + y
```

```
> z
```

```
[1] 11 12 13 14 15
```

Variables also can take on text values. You can assign the value “Hello” to a variable called h, for example, by presenting the text to R inside quotation marks, like this:

```
> h <- "Hello"  
> h  
[1] "Hello"
```

Hint: You must present text or character values to R inside quotation marks — either single or double. R accepts both. So both `h <- "Hello"` and `h <- 'Hello'` are examples of valid R syntax

In “Using vectors,” earlier in this chapter, you use the `c()` function to combine numeric values into vectors. This technique also works for text. Try it:

```
> hw <- c("Hello", "world!")  
> hw  
[1] "Hello" "world!"
```

You can use the `paste()` function to concatenate multiple text elements. By default, `paste()` puts a space between the different elements, like this:

```
> paste("Hello", "world!")  
[1] "Hello world!"
```

Piping commands with %>%

The piping commands character with `%>%`, which is included as part of the Tidyverse.

Let me give you an example of command and how you would write it in base R. Base R uses nested commands, which mean you start in the middle and you go out.

```
> round(exp(diff(log(x))), 1)
```

R is a functional language, which means that your code often contains a lot of parenthesis, (and). When you have complex code, this often will mean that you will have to nest those parentheses together. This makes your R code hard to read and understand. Here's where `%>%` comes in to the rescue!

How Pipes %>% Work

Old

```
f3(f2(f1(data,arg1), arg2),arg3)
```

New Pipes command %>%

```
data %>%  
  f1(arg1) %>%  
  f2(arg2) %>%  
  f3(arg3)
```

Typical example, which is a typical example of nested code:

```
> # Initialize `x`  
> x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)  
>  
> # Compute the logarithm of `x`, return suitably lagged and iterated  
differences,  
> # compute the exponential function and round the result  
> round(exp(diff(log(x))), 1)  
[1] 3.3 1.8 1.6 0.5 0.3 0.1 48.8 1.1  
>
```

Note that you need to import the magrittr library to get the above code to work. That's because the pipe operator is, as you read above, part of the magrittr library and is, since 2014, also a part of dplyr. If you forget to import the library, you'll get an error like

Error in eval(expr, envir, enclos): could not find function "%>%".

With the help of %<%, you can rewrite the above code as follows:

```
> # Import `magrittr`  
> library(magrittr)  
>  
> # Perform the same computations on `x` as above  
> x %>% log() %>%  
+   diff() %>%  
+   exp() %>%  
+   round(1)  
[1] 3.3 1.8 1.6 0.5 0.3 0.1 48.8 1.1
```

Example 1: plot or (qplot) makes a scatterplot when you give it two vectors

Script

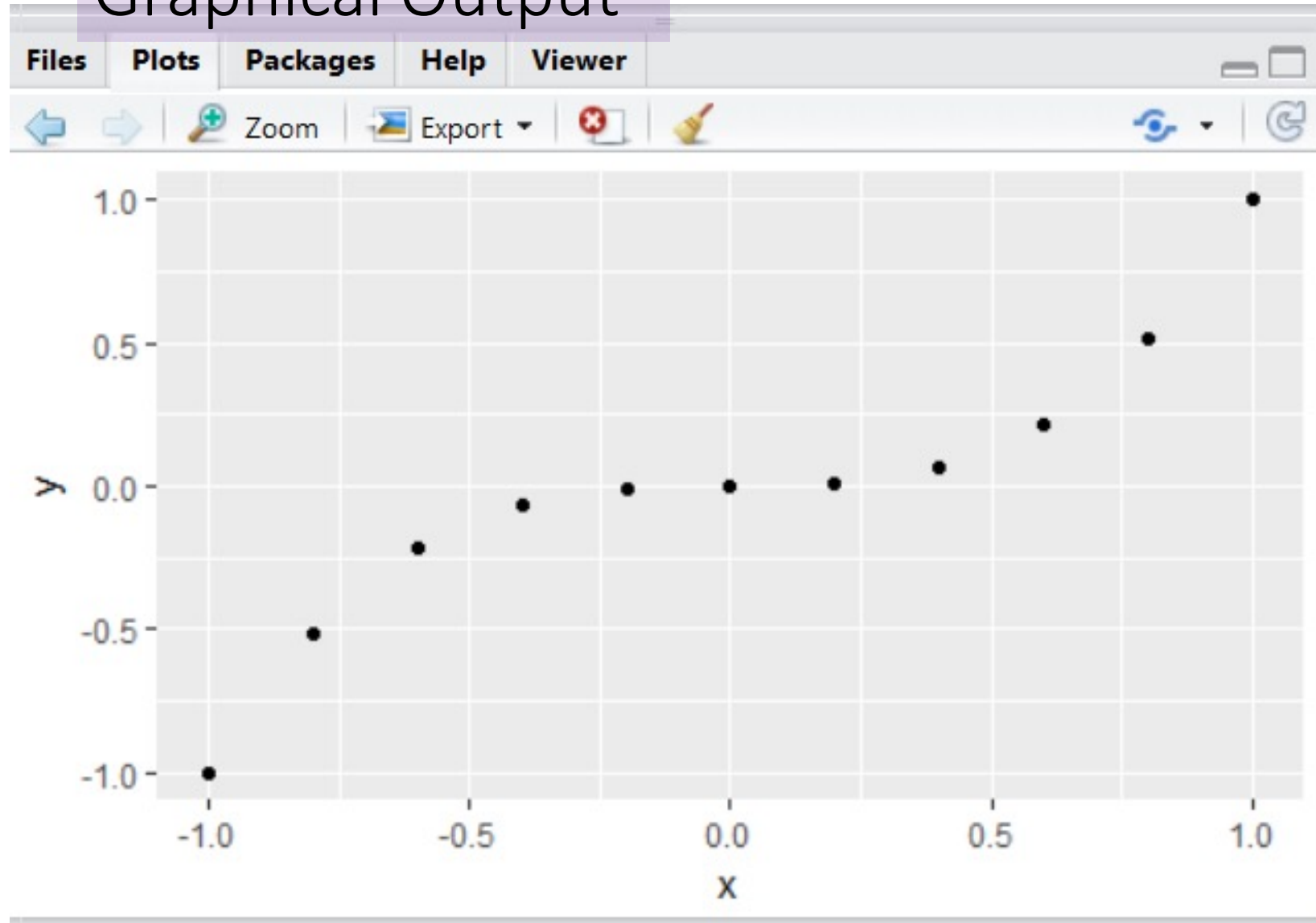
```
library(ggplot2)
x <- c(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1)
y <- x^3
qplot(x,y)
```

```
> library(ggplot2)
> x <- c(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1)
> y <- x^3
> qplot(x,y)
>
```

Console

Example 1: plot or (qplot) makes a scatterplot when you give it two vectors

Graphical Output



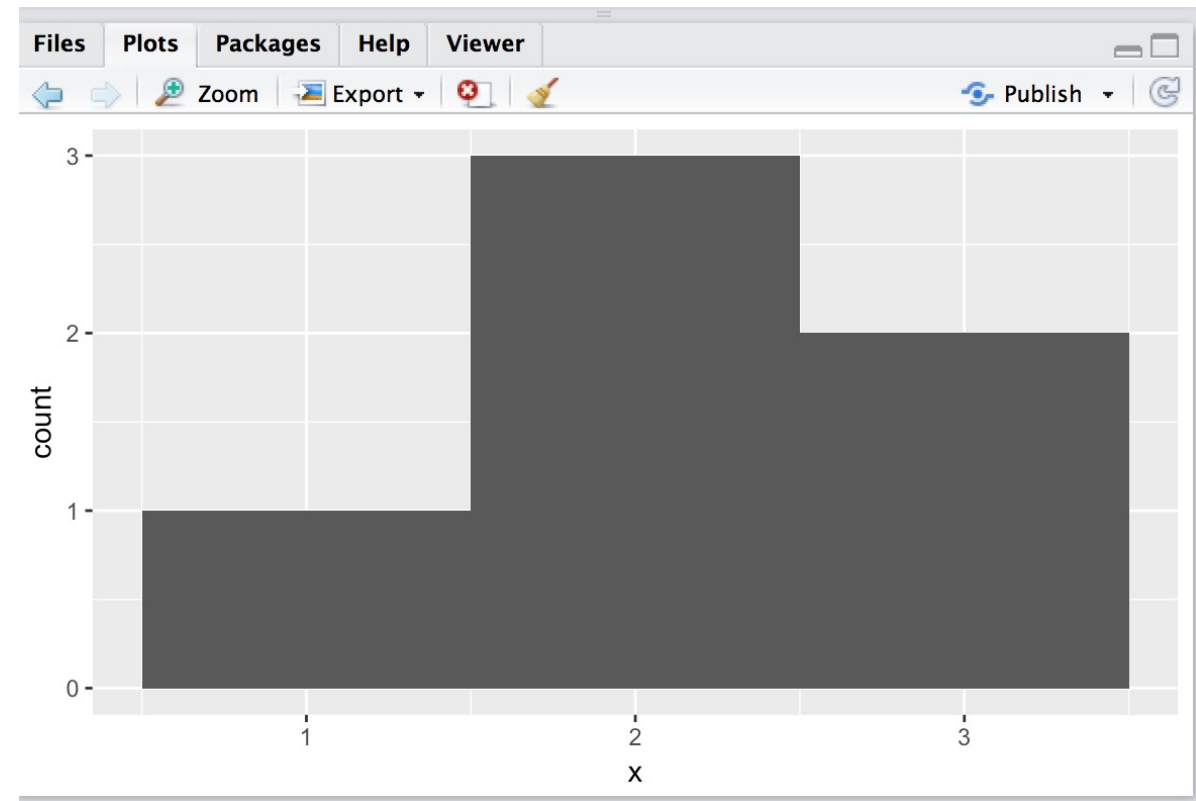
Scatterplots are useful for visualizing the relationship between two variables.

Example (2)

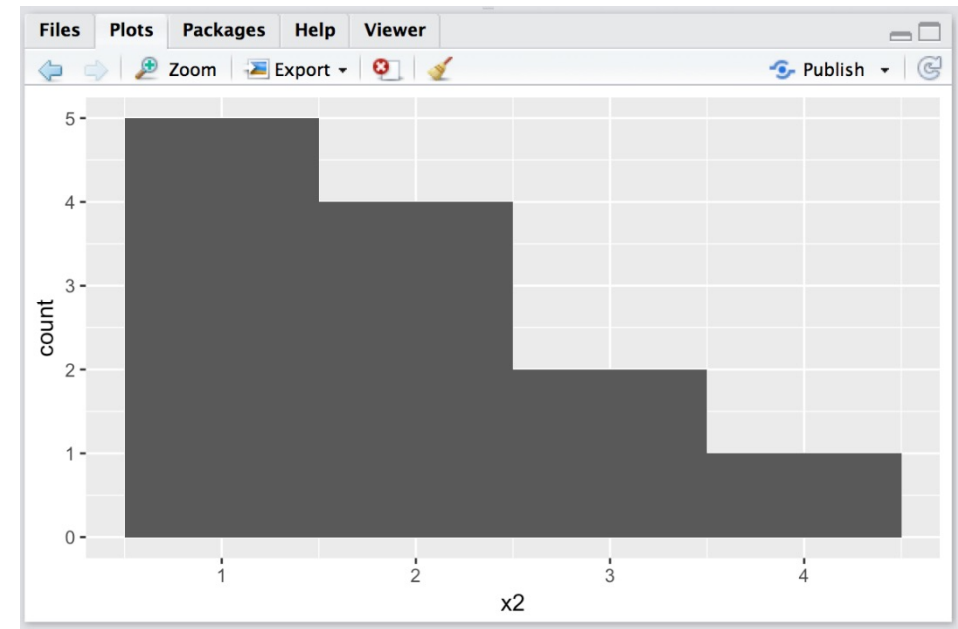
we're going to use a different type of graph, **a histogram**. A histogram visualizes the distribution of a single variable; it displays how many data points appear at each value of x .

```
> x <- c(1, 2, 2, 2, 3, 3)
> qplot(x, binwidth = 1)
>
```

Figure 2 . qplot makes a histogram when you give it a single vector.



```
> x2 <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 4)
> qplot(x2, binwidth = 1)
```



```
> x3 <- c(0, 1, 1, 2, 2, 2, 3, 3, 4)
> qplot(x3, binwidth = 1)
```

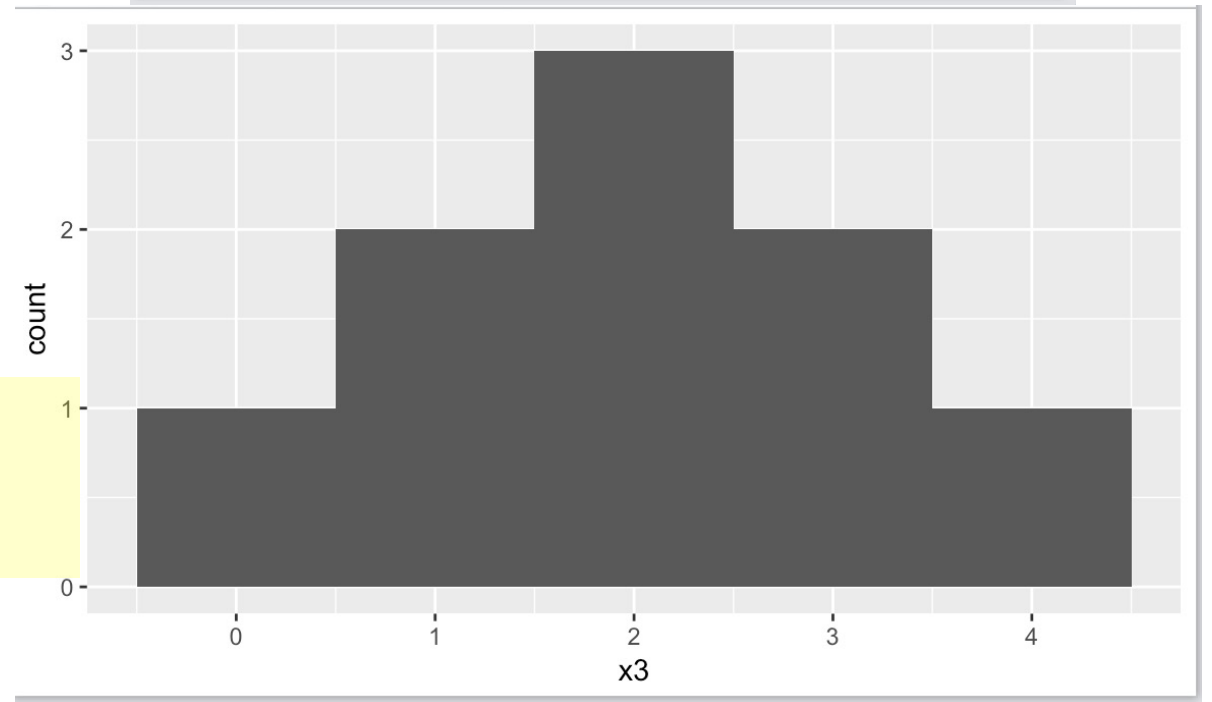


Figure 3. qplot makes a histogram when you give it a single vector.

Useful R Packages

Currently, the CRAN package repository features 17648 available packages, I've listed some of the most powerful and commonly used packages in predictive modeling in this article. Since, I've already explained the method of installing packages, you can go ahead and install them now. Sooner or later you'll need them.

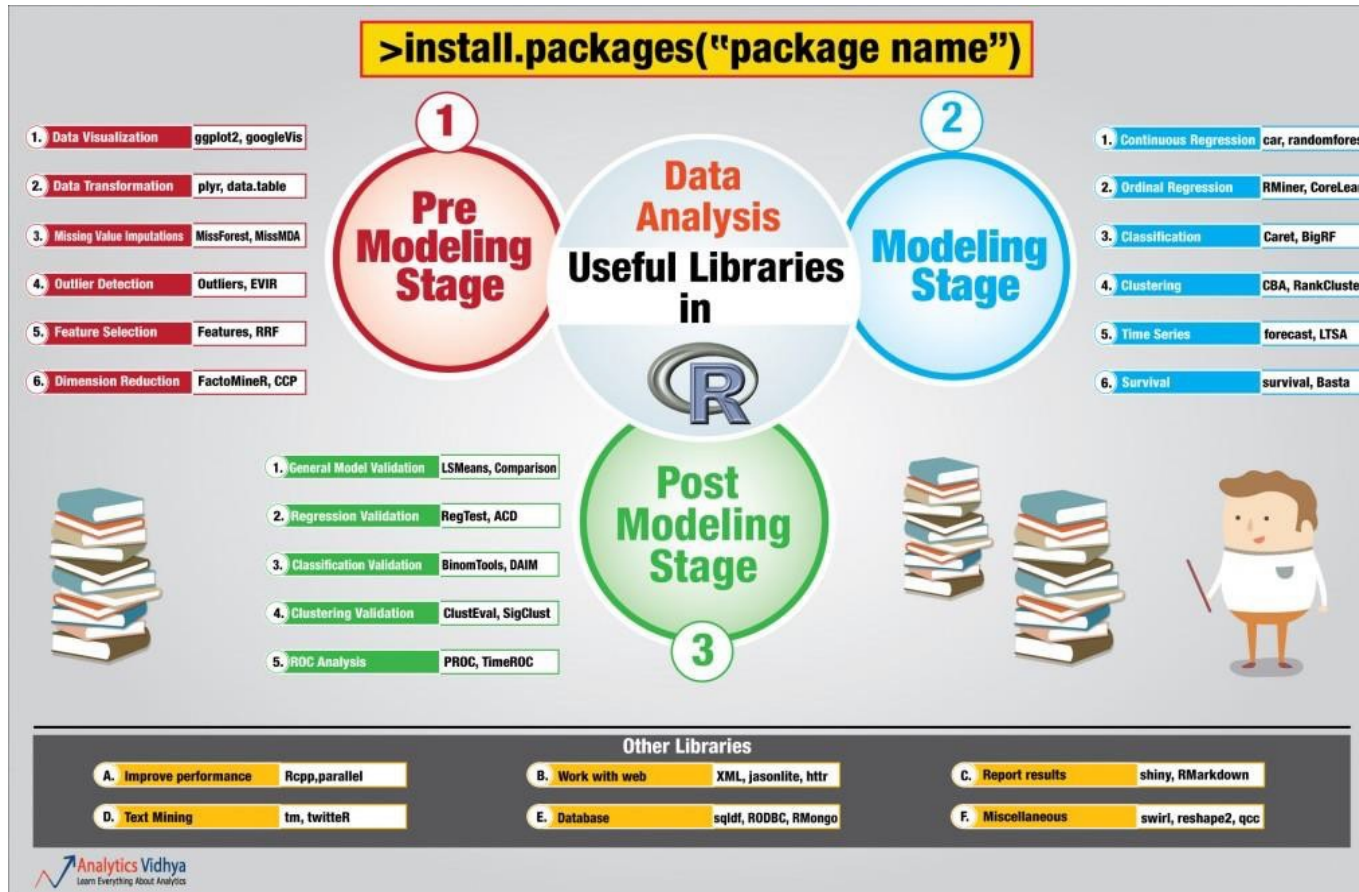
Importing Data: R offers wide range of packages for importing data available in any format such as .txt, .csv, .json, .sql etc. To import large files of data quickly, it is advisable to install and use data.table, readr, RMySQL, sqldf, jsonlite.

Data Visualization: R has in built plotting commands as well. They are good to create simple graphs. But, becomes complex when it comes to creating advanced graphics. Hence, you should install ggplot2.

Data Manipulation: R has a fantastic collection of packages for data manipulation. These packages allows you to do basic & advanced computations quickly. These packages are *dplyr*, *plyr*, *tidyr*, *lubridate*, *stringr*.

Modeling / Machine Learning: For modeling, *caret* package in R is powerful enough to cater to every need for creating machine learning model. However, you can install packages algorithms wise such as *randomForest*, *rpart*, *gbm* etc

Note: I've only mentioned the commonly used packages. You might like to check this interesting [infographic](#) on complete list of useful R packages.



Getting Help with Help Pages

There are over 1,000 functions at the core of R, and new R functions are created all of the time. This can be a lot of material to memorize and learn! Luckily, each R function comes with its own help page, which you can access by typing the function's name after a question mark.

For example, each of these commands will open a help page. Look for the pages to appear in the Help tab of RStudio's bottom-right pane:

```
> ?sqrt  
> ?log10  
> ?sample
```

Hint: If a function comes in an R package, R won't be able to find its help page unless the package is loaded.

Getting More Help?!

R also comes with a super active community of users that you can turn to for help on the R-help mailing list. You can email the list with questions, but there's a great chance that your question has already been answered. Find out by searching the archives.

Even better than the R-help list is **Stack Overflow**, a website that allows programmers to answer questions and users to rank answers based on helpfulness. **Personally, I find the Stack Overflow format to be more user-friendly than the R-help email list (and the respondents to be more human friendly)**. You can submit your own question or search through Stack Overflow's previously answered questions related to R. There are over 30,000.

For both the R help list and Stack Overflow, you're more likely to get a useful answer if you provide a reproducible example with your question. This means pasting in a short snippet of code that users can run to arrive at the bug or question you have in mind.

Getting More Help?!

- R puts a big emphasis on documentation. The previously mentioned <https://www.rdocumentation.org/> is a great website to look at the different documentation of different packages and functions.
- There are numerous blogs & posts on the web covering R such as [KDnuggets](#) and [R-bloggers](#).

Working Directory

Each time you open R, it links itself to a directory on your computer, which R calls the **working directory**. This is where R will look for files when you attempt to load them, and it is where R will save files when you save them. The location of your working directory will vary on different computers. To determine which directory R is using as your working directory, run:

```
> getwd()  
[1] "C:/Users/new/Documents"  
>
```

1. You can place data files straight into the folder that is your working directory, or
2. You can move your working directory to where your data files are.
3. You can move your working directory to any folder on your computer with the function `setwd`. Just give `setwd` the file path to your new working directory.

Hint: I prefer to set my working directory to a folder dedicated to whichever project I am currently working on. That way I can keep all of my data, scripts, graphs, and reports in the same place. For example:

Setting your Working Directory

I prefer to set my working directory to a folder dedicated to whichever project I am currently working on. That way I can keep all of my data, scripts, graphs, and reports in the same place. For example:

```
> setwd("C:/Users/new/Desktop/Data files")  
> getwd()  
[1] "C:/Users/new/Desktop/Data files"  
>
```

Should be like
this direction
of the slash to
make it work!

Tilde command

```
> setwd("~/Desktop/AITRS Course")
```

If the file path does not begin with your root directory, R will assume that it begins at your current working directory.

You can also change your working directory by clicking on Session > Set Working Directory
> Choose Directory in the RStudio menu bar.

You can see what files are in your working directory with *list.files()*. If you see the file that you would like to open in your working directory, then you are ready to proceed. How you open files in your working directory will depend on which type of file you would like to open.

```
> list.files()  
[1] "deck.csv"  "deck.RData"  
>
```

Designing projects

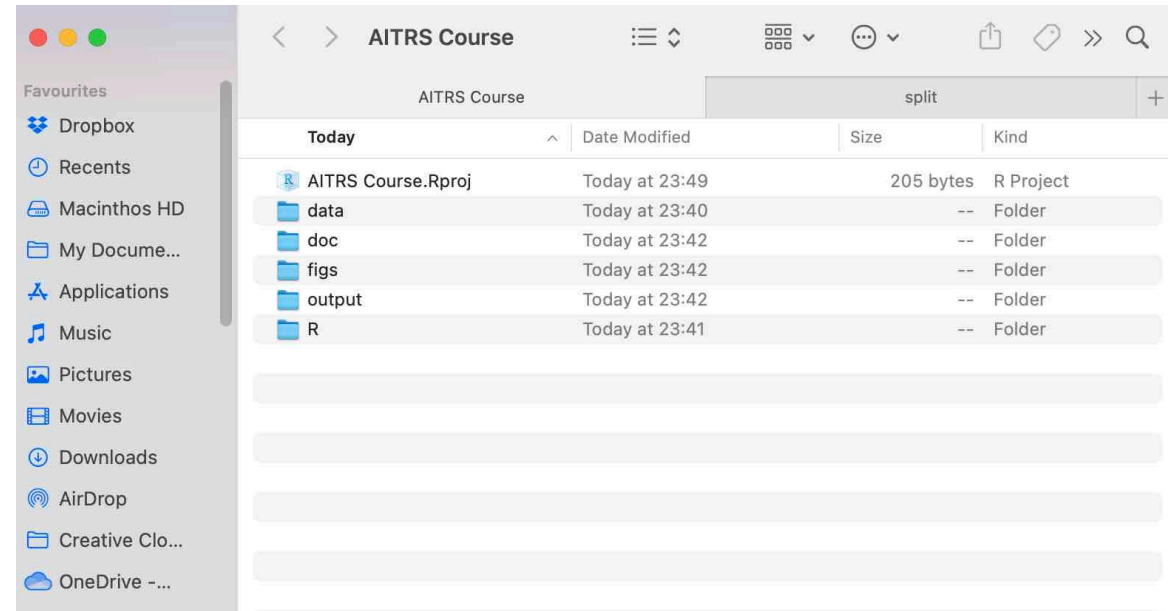
Managing your projects in a reproducible fashion doesn't just make your science reproducible, it makes your life easier.

Working directories are useful for keeping work organized. A working directory is one spot (e.g. a folder) that you have created for saving all of your work.

Here are a couple of different ideas for laying a project out. This is the basic structure that I usually use:

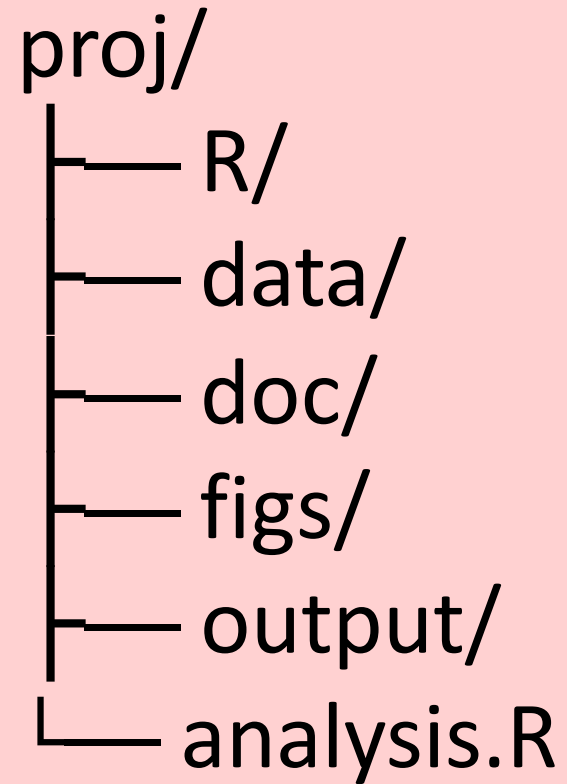
```
proj/  
├── R/  
├── data/  
├── doc/  
├── figs/  
└── output/
```

Hint: Open a folder in your Desktop and name it then create five subfolders as shown in the diagrams then you can open it as new project in R



- The R directory contains various files with function definitions (but only function definitions - no code that actually runs).
- The data directory contains data used in the analysis. This is treated as read only; in particular the R files are never allowed to write to the files in here. Depending on the project, these might be csv files, a database, and the directory itself may have subdirectories.
- The doc directory contains the paper. I work in LaTeX which is nice because it can pick up figures directly made by R. Markdown can do the same and is starting to get traction among biologists. With Word you'll have to paste them in yourself as the figures update.
- The figs directory contains the figures. This directory only contains generated files; that is, I should always be able to delete the contents and regenerate them.
- The output directory contains simulation output, processed datasets, logs, or other processed things.
- In this set up, I usually have the R script files that do things in the project root:

In this set up, I usually have the R script files that do things in the project root:



```
proj/  
├── R/  
├── data/  
├── doc/  
├── figs/  
├── output/  
└── analysis.R
```

For very simple projects, you might drop the R directory, perhaps replacing it with a single file **analysis-functions.R** which you source.

The Data Analysis Workflow

The Data Analysis Workflow

1. Importing Data
2. Data Manipulation
3. The Machine Learning Part
4. Data Visualization
5. Reporting your results

1. Importing Data

1.Importing Data

Before you can start performing analysis, you first need to get your data into R. The good thing is that you can import into R all sorts of data formats, the hard part this is that different types often need a different approach:

- Flat files: You can import **flat files with functions** such as [read.table\(\)](#) and [read.csv\(\)](#) from the pre-installed utils package. Specific R packages to import flat files data are [readr](#) and [fread\(\)](#) function of the data.table package.
- You can get your **excel** files into R with either the [readxl package](#), the [gdata package](#) and [XLConnect](#) package. <https://www.datacamp.com/community/tutorials/r-tutorial-read-excel-into-r>
- The [haven package](#) lets you import SAS, STATA and SPSS data files into R. The [foreign package](#) lets you import formats like Systat and Weka.
- Connecting with a database happens via specific packages like [RMySQL](#), [Rpostgresql](#) and the [ROracle](#) package. Accessing and manipulating the database happens via [DBI](#).
- For web scraping you can use a package like [rvest](#). (For more info on web scraping with R check <http://blog.rolffredheim.com/2014/02/web-scraping-basics.html>

If you want to learn more on how to import data into R check an https://www.datacamp.com/courses/importing-data-into-r?tap_a=5644-dce66f&tap_s=14201-e863d5

Loading and Saving Data in R

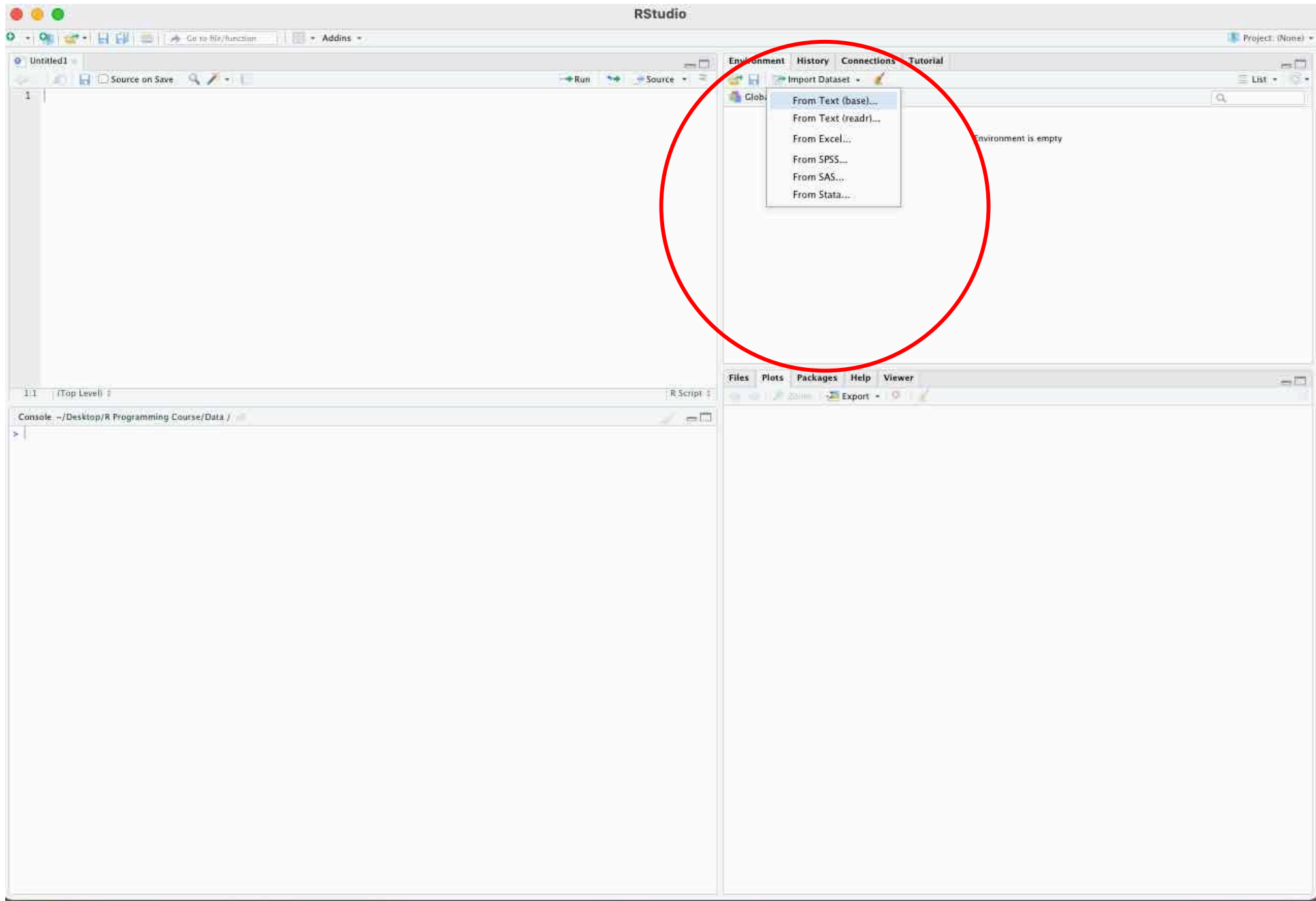
This Section will show you how to load and save data into R from plain text files, R files, and Excel spreadsheets. It will also show you the R packages that you can use to load data from databases and other common programs, like SAS and MATLAB, etc.

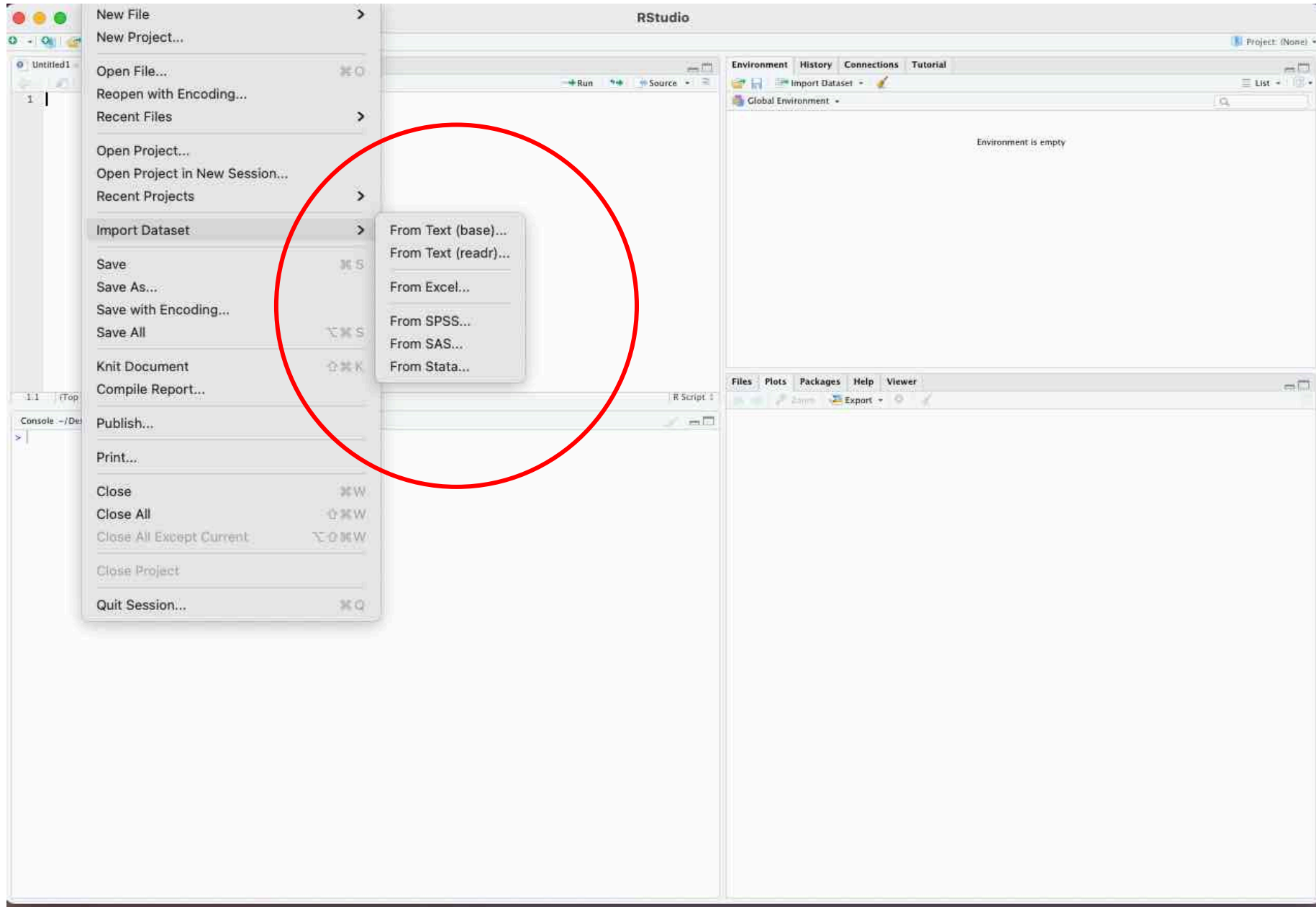
The data import features can be accessed from **the environment pane or from the tool's menu.**

The importers are grouped into 3 categories:

1. Text (base)-Delimited data (CSV)-CSVs are plain-text files
2. Text (readr)
3. Excel data
4. Statistical data (SPSS, SAS, and Stata)

To access this feature, use the **"Import Dataset"** dropdown from the "Environment" pane as shown below:





R's built-in datasets

Now, the datasets package comes with R, it's part of the default installation. However, it's not loaded, it's not active in memory by default. And so by using `library`, and then in parenthesis, `datasets`, we'll load it. And we'll make it available. You can also use `require`.

short description of each by running:

```
> library(package = "datasets")
```

When we run that command, it's going to give us this information and this is a list of all of the datasets that are included in that package

RStudio

Google text mining-Word associat... | Untitled1* | Final -Good Twitter analysis-Best.R* | New Twitter Great using tidtext pa... | Addins

Source on Save | Run | Source

```
17 x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
18
19 # Compute the logarithm of 'x', return suitably lagged and iterated differences,
20 # compute the exponential function and round the result
21 round(exp(diff(log(x))), 1)
22
23
24
25 help(package = "datasets")
26
27 library(package = "datasets")
28
29
30
31
32
33
34
35 # Import 'magrittr'
36 library(magrittr)
37
38 # Perform the same computations on 'x' as above
39 x %>% log() %>%
25.1 | (Top Level) | R Script
```

Environment | History | Connections | Tutorial

Files | Plots | Packages | Help | Viewer

R: the R Datasets Package | Find in Topic

Help Pages

[ABCDEFGHIJKLMNOPQRSTUVWXYZ](#)

[datasets-package](#) The R Datasets Package

-- A --

[ability.cov](#) Ability and Intelligence Tests
[airmiles](#) Passenger Miles on Commercial US Airlines, 1937-1960
[AirPassengers](#) Monthly Airline Passenger Numbers 1949-1960
[airquality](#) New York Air Quality Measurements
[anscombe](#) Anscombe's Quartet of 'Identical' Simple Linear Regressions
[attenu](#) The Joyner-Boore Attenuation Data
[attitude](#) The Chatterjee-Price Attitude Data
[austres](#) Quarterly Time Series of the Number of Australian Residents

-- B --

[beaver1](#) Body Temperature Series of Two Beavers
[beaver2](#) Body Temperature Series of Two Beavers
[beavers](#) Body Temperature Series of Two Beavers
[BJsales](#) Sales Data with Leading Indicator
[BJsales.lead](#) Sales Data with Leading Indicator
[BOD](#) Biochemical Oxygen Demand

-- C --

[cars](#) Speed and Stopping Distances of Cars
[ChickWeight](#) Weight versus age of chicks on different diets
[chickwts](#) Chicken Weights by Feed Type
[CO2](#) Carbon Dioxide Uptake in Grass Plants
[co2](#) Mauna Loa Atmospheric CO2 Concentration
[crimtab](#) Student's 3000 Criminals Data

-- D --

[datasets](#) The R Datasets Package
[discoveries](#) Yearly Numbers of Important Discoveries
[DNase](#) Elisa assay of DNase

-- E --

[esoph](#) Smoking, Alcohol and (O)esophageal Cancer
[euro](#) Conversion Rates of Euro Currencies
[euro.cross](#) Conversion Rates of Euro Currencies
[euro.dist](#) Distances Between European Cities and Between US Cities
[EuStockMarkets](#) Daily Closing Prices of Major European Stock Indices, 1991-1998

C

R comes with many data sets preloaded in the datasets package, which comes with base R. These data sets are not very interesting, but they give you a chance to test code or make a point without having to load a data set from outside R. You can see a list of R's data sets as well as a

To use a data set, just type its name. Each data set is already presaved as an R object. For example:

> iris

```
> iris
      Sepal.Length Sepal.Width Petal.Length
Petal.Width  Species
1          5.1       3.5         1.4         0.2    setosa
2          4.9       3.0         1.4         0.2    setosa
3          4.7       3.2         1.3         0.2    setosa
4          4.6       3.1         1.5         0.2    setosa
5          5.0       3.6         1.4         0.2    setosa
6          5.4       3.9         1.7         0.4    setosa
7          4.6       3.4         1.4         0.3    setosa
8          5.0       3.4         1.5         0.2    setosa
9          4.4       2.9         1.4         0.2    setosa
10         4.9       3.1         1.5         0.1    setosa
11         5.4       3.7         1.5         0.2    setosa
12         4.8       3.4         1.6         0.2    setosa
13         4.8       3.0         1.4         0.1    setosa
14         4.3       3.0         1.1         0.1    setosa
```

Importing data from a spreadsheet

Spreadsheets are the universal data containers. Billions of datasets in the rows and columns of a spreadsheet. And they're very easy to import in R as long as you have what's called tidy data and that means each column is a variable, each row is an observation.

1. Importing data from CSV files

The CSV importer provides support to:

- Import from the file system or a url
- Change column data types
- Skip or include-only columns
- Rename the data set
- Skip the first N rows
- Use the header row for column names
- Trim spaces in names
- Change the column delimiter
- Encoding selection
- Select quote, escape, comment and NA identifiers

Example (1): You can load the deck data frame from the file *deck.csv*.

<https://gist.github.com/garrettgman/9629323>

deck.csv is a comma-separated values file, or CSV for short. CSVs are plain-text files, which means you can open them in a text editor (as well as many other programs). If you open *deck.csv*, you'll notice that it contains a table of data that looks like the following table. Each row of the table is saved on its own line, and a comma is used to separate the cells within each row. Every CSV file shares this basic format: "face","suit","value"

"king","spades",13

"queen","spades",12

"jack","spades",11

"ten","spades",10

"nine","spades",9

... and so on.

Once everything looks right, click Import. RStudio will read in the data and save it to a data frame. RStudio will also open a data viewer, so you can see your new data in a spreadsheet format. This is a good way to check that everything came through as expected. If all worked well, your file should appear in a View tab of RStudio, like in Figure (3). You can examine the data frame in the console with *head(deck)*.

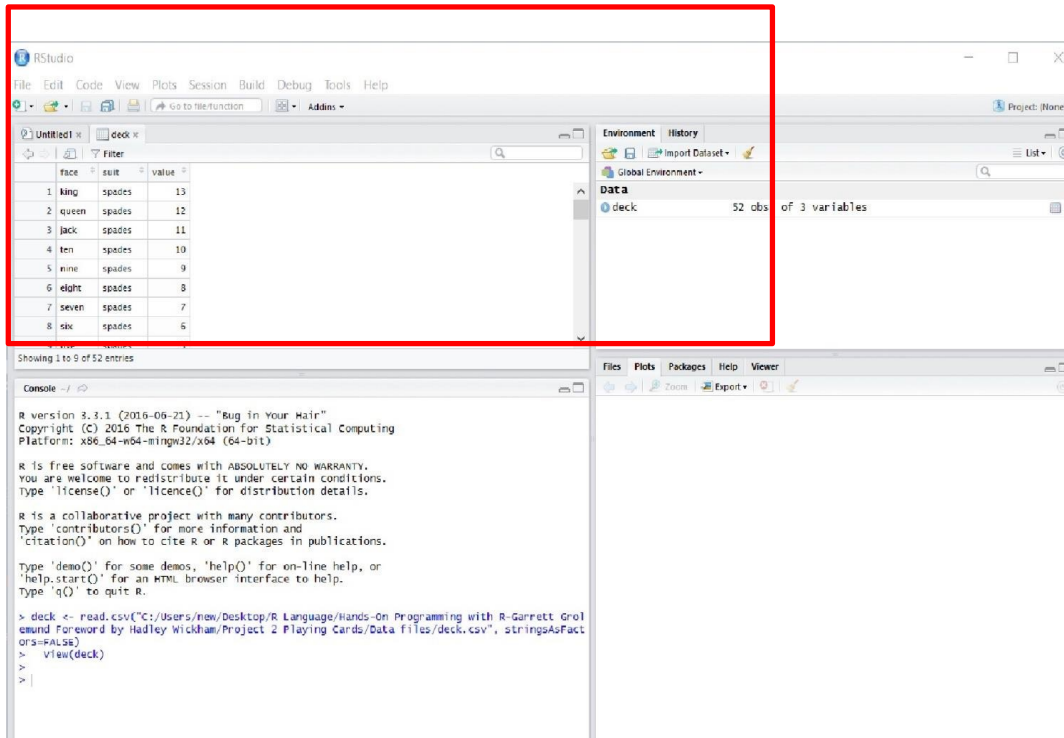


Figure 3(. When you import a data set, RStudio will save the data to a data frame and then display the data frame in a View tab. You can open any data frame in a View tab at any time with the View function.

head and tail are two functions that provide an easy way to peek at large data sets. `head` will return just the first six rows of the data set, and `tail` will return just the last six rows. To see a different number of rows, give `head` or `tail` a second argument, the number of rows you would like to view, for example, `head(deck, 10)`.

```
> head(deck)
  face suit value
1 king spades  13
2 queen spades 12
3 jack spades  11
4 ten spades   10
5 nine spades   9
6 eight spades  8
> tail(deck)
  face suit value
47 six hearts   6
48 five hearts  5
49 four hearts  4
50 three hearts 3
51 two hearts   2
52 ace hearts   1
>
```

2. Importing data from Excel files

The Excel importer provides support to:

- Import from the file system or a url
- Change column data types
- Skip columns
- Rename the data set
- Select an specific Excel sheet
- Skip the first N rows
- Select NA identifiers

Example select import StateData.xlsx

Example select import StateData.xlsx

AITRS Course - RStudio

Go to file/function Addins

AITRS Course — Desktop

Environment History Connections Tutorial

Import Dataset

Global Environment

Data

- deck 52 obs. of 3 variables
- slsummar 57 obs. of 7 variables
- StateData 48 obs. of 22 variables

Files Plots Packages Help Viewer

New Folder Delete Rename More

Home > Desktop > AITRS Course

	Name	Size	Modified
<input type="checkbox"/>	..		
<input type="checkbox"/>	.Rhistory	0 B	May 26, 2021, 11:48 PM
<input type="checkbox"/>	AITRS Course.Rproj	205 B	May 27, 2021, 12:39 AM
<input type="checkbox"/>	data		
<input type="checkbox"/>	doc		
<input type="checkbox"/>	figs		
<input type="checkbox"/>	output		
<input type="checkbox"/>	R		

	State	state_code	region	governor	psychRegions	extraversion	agreeableness	conscientiousness
1	Alabama	AL	South	Republican	Friendly and Conventional	55.5	52.7	55.5
2	Arizona	AZ	West	Republican	Relaxed and Creative	50.6	46.6	58.4
3	Arkansas	AR	South	Republican	Friendly and Conventional	49.9	52.7	41.0
4	California	CA	West	Democrat	Relaxed and Creative	51.4	49.0	43.2
5	Colorado	CO	West	Democrat	Friendly and Conventional	45.3	47.5	58.8
6	Connecticut	CT	Northeast	Democrat	Temperamental and Uninhibited	57.6	38.6	34.2
7	Delaware	DE	South	Democrat	Temperamental and Uninhibited	47.0	38.8	36.5
8	Florida	FL	South	Republican	Friendly and Conventional	60.9	50.7	62.7
9	Georgia	GA	South	Republican	Friendly and Conventional	63.2	60.0	68.8
10	Idaho	ID	West	Republican	Relaxed and Creative	40.7	52.9	44.5
11	Illinois	IL	Midwest	Republican	Friendly and Conventional	62.5	48.3	50.9
12	Indiana	IN	Midwest	Republican	Friendly and Conventional	48.9	50.2	56.2
13	Iowa	IA	Midwest	Republican	Friendly and Conventional	62.8	56.6	52.2
14	Kansas	KS	Midwest	Republican	Friendly and Conventional	45.5	48.9	50.8
15	Kentucky	KY	South	Republican	Friendly and Conventional	53.4	48.1	51.3
16	Louisiana	LA	South	Democrat	Friendly and Conventional	52.2	49.7	45.0

Showing 1 to 16 of 48 entries, 22 total columns

Console ~/Desktop/AITRS Course/

```
> |
```

3. Importing data from SPSS, SAS and Stata files

The SPSS, SAS and Stata importer provides support to:

Import from the file system or a url

Rename the data set

Specify a model file

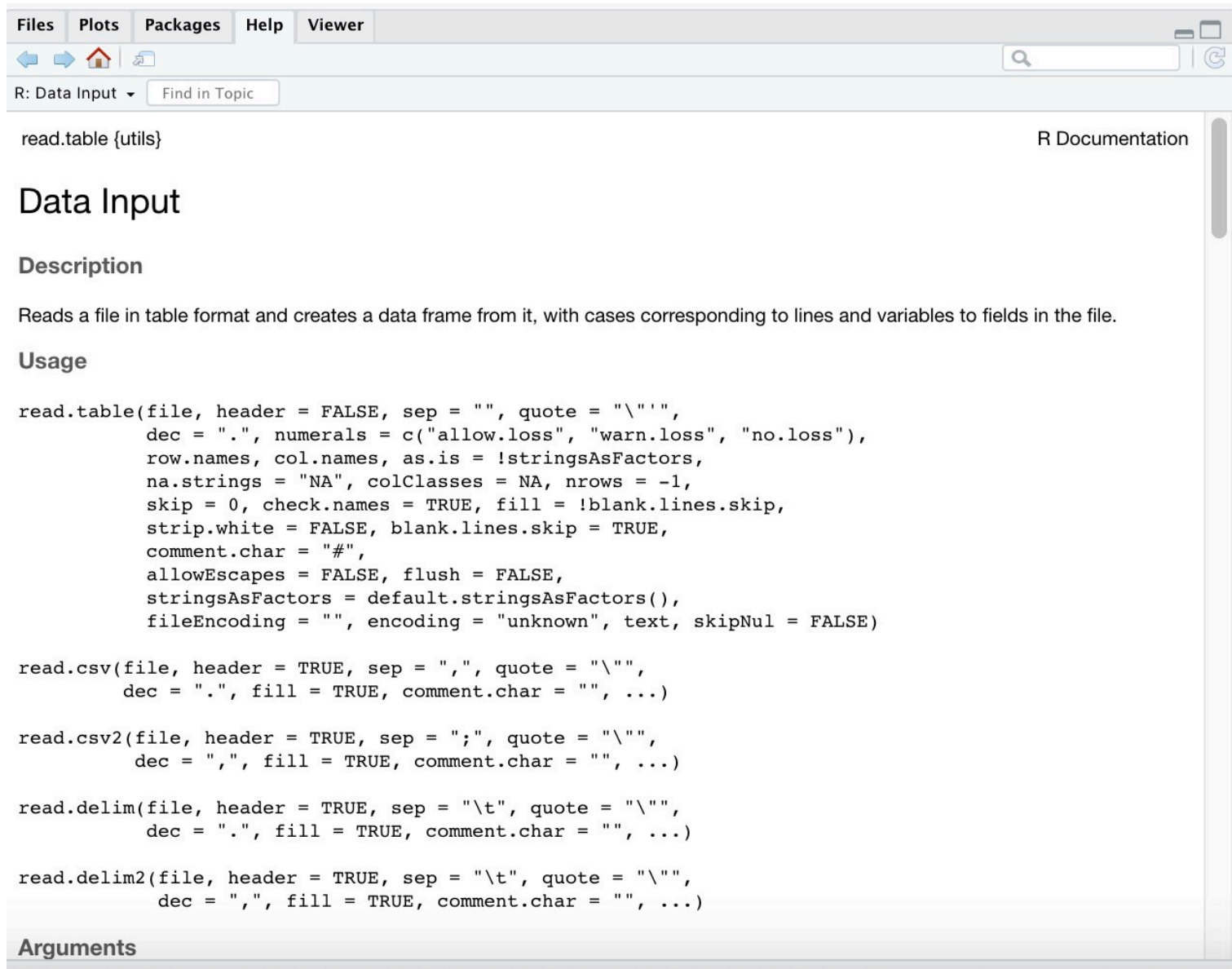
Plain-text Files

Plain-text files are one of the most common ways to save data. They are very simple and can be read by many different computer programs—even the most basic text editors. For this reason, *public data often comes as plain-text files*. For example,

- Census Bureau
- Social Security Administration
- Bureau of Labor Statistics all make

All plain-text files can be saved with the ***extension .txt (for text)***, but sometimes a file will receive a special extension that advertises how it separates data-cell entries. Another text file type would be a comma-separated-values file and would usually be saved with the ***extension .csv.***

?read.csv



The screenshot shows the R Documentation window for the `read.csv` function. The window has a menu bar with 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu bar is a toolbar with navigation icons and a search bar. The main content area displays the function signature `read.table {utils}` and the title 'Data Input'. The 'Description' section states: 'Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.' The 'Usage' section provides the following code examples:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
  dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
  row.names, col.names, as.is = !stringsAsFactors,
  na.strings = "NA", colClasses = NA, nrows = -1,
  skip = 0, check.names = TRUE, fill = !blank.lines.skip,
  strip.white = FALSE, blank.lines.skip = TRUE,
  comment.char = "#",
  allowEscapes = FALSE, flush = FALSE,
  stringsAsFactors = default.stringsAsFactors(),
  fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
  dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
  dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
  dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
  dec = ",", fill = TRUE, comment.char = "", ...)
```

The 'Arguments' section is partially visible at the bottom of the window.

Read/Import csv file using command

```
filename <- read.csv(file.choose())  
filename
```

Alternatively

```
getwd()  
setwd("/Users/admin/Desktop/AITRS Course/data")  
deck <- read.csv("deck.csv")  
deck
```

Read/Import csv file using command

```
filename <- read.csv(file.choose())  
filename
```

Alternatively

For Mac-OS

```
getwd()  
setwd("/Users/admin/Desktop/AITRS Course/data")  
deck <- read.csv("deck.csv")  
deck
```

For Window

```
getwd()  
setwd("C:\\Users\\abbas\\Desktop\\AITRS Course\\  
deck.csv data")  
deck <- read.csv("deck.csv")  
deck
```

read.table

To load a plain-text file, use read.table. The first argument of read.table should be the name of your file (if it is in your working directory), or the file path to your file (if it is not in your working directory). If the file path does not begin with your root directory, R will append it to the end of the file path that leads to your working directory. You can give *read.table* other arguments as well. ***The two most important are sep and header.*** If the royal flush data set was saved as a file named poker.csv in your working directory, you could load it with:

```
> deck <- read.table("deck.csv", sep = ",", header = TRUE)
>
```

```
> deck <- read.table("deck.csv", sep = ",", header = TRUE)  
>
```

sep

Use `sep` to tell `read.table` what character your file uses to separate data entries. To find this out, you might have to open your file in a text editor and look at it. If you don't specify a `sep` argument, `read.table` will try to separate cells whenever it comes to white space, such as a tab or space. R won't be able to tell you if `read.table` does this correctly or not, so rely on it at your own risk.

header

Use `header` to tell `read.table` whether the first line of the file contains variable names instead of values. If the first line of the file is a set of variable names, you should set `header = TRUE`.

na.strings

Oftentimes data sets will use special symbols to represent missing information. If you know that your data uses a certain symbol to represent missing entries, you can tell

read.table (and the preceding functions) what the symbol is with the na.strings argument. read.table will convert all instances of the missing information symbol to NA, which is R's missing information symbol

You could read the data set into R and convert the missing values into NAs as you go with the command:

```
> deck <- read.table("deck.csv", sep = ",", header = TRUE,  
na.string=".")  
>
```

skip and nrow

Sometimes a plain-text file will come with introductory text that is not part of the data set. Or, you may decide that you only wish to read in part of a data set. You can do these things with the **skip and nrow** arguments. Use `skip` to tell R to skip a specific number of lines before it starts reading in values from the file. Use `nrow` to tell R to stop reading in values after it has read in a certain number of lines.

You can read just the six lines (five rows plus a header) that you want with:

```
> deck <- read.table("deck.csv", sep = ",", header = TRUE, skip = 3, nrow = 5)
```

stringsAsFactors

R reads in numbers just as you'd expect, but when R comes across character strings (e.g., letters and words) it begins to act strangely. R wants to convert every character string into a factor. This is R's default behavior, but I think it is a mistake. Sometimes factors are useful. At other times, they're clearly the wrong data type for the job. Also factors cause weird behavior, especially when you want to display data

Setting the argument `stringsAsFactors` to `FALSE` will ensure that R saves any character strings in your data set as character strings, not factors. To use `stringsAsFactors`, you'd write:

```
> deck <- read.table("deck.csv", sep = ",", header = TRUE, stringsAsFactors = FALSE)
>
```

If you will be loading more than one data file, you can change the default factoring behavior at the global level with:

```
options(stringsAsFactors = FALSE)
```

This will ensure that all strings will be read as strings, not as factors, until you end your R session, or recharge the global default by running:

```
options(stringsAsFactors = TRUE)
```

The read Family

R also comes with some prepackaged short cuts for `read.table`, shown in Table

Function	Defaults	Use
<code>read.table</code>	<code>sep = " ", header = FALSE</code>	General-purpose read function
<code>read.csv</code>	<code>sep = ",", header = TRUE</code>	Comma-separated-variable (CSV) files
<code>read.delim</code>	<code>sep = "\t", header = TRUE</code>	Tab-delimited files
<code>read.csv2</code>	<code>sep = ";", header = TRUE, dec = ","</code>	CSV files with European decimal format
<code>read.delim2</code>	<code>sep = "\t", header = TRUE, dec = ","</code>	Tab-delimited files with European decimal format

For example:

```
> deck <- read.csv("deck.csv")  
>
```

Excel File

```
install.packages("readxl")  
# Loading  
library("readxl")  
# xlsx files  
my_data <- read_excel("StateData.xlsx")
```

It's also possible to choose a file interactively using the function **file.choose()**

```
my_data <- read_excel(file.choose())
```

Web Data

```
install.packages("RCurl")  
install.packages("XML")  
install.packages("stringr")  
install.packages("plyr")
```

```
library("RCurl")  
library("XML")  
library("stringr")  
library("plyr")
```

Read the URL.

```
url <-"http://www.geos.ed.ac.uk/~weather/jcmb_ws/"
```

```
url <-"https://gist.github.com/garrettgman/9629323"
```

Saving Data

Saving Plain-Text Files

Once your data is in R, you can save it to any file format that R supports. If you'd like to save it as a plain-text file, you can use the write family of functions. The three basic write functions appear in Table below. Use ***write.csv*** to save your data as a .csv file and write.table to save your data as a tab delimited document or a document with more exotic separators.

Table 2. R saves data sets to plain-text files with the write family of functions

File format	Function and syntax
.csv	<code>write.csv(r_object, file = filepath, row.names = FALSE)</code>
.csv (with European decimal notation)	<code>write.csv2(r_object, file = filepath, row.names = FALSE)</code>
tab delimited	<code>write.table(r_object, file = filepath, sep = "\t", row.names=FALSE)</code>

Before we go any further, let's save a copy of deck as a new .csv file. That way you can email it to a colleague, store it on a thumb drive, or open it in a different program. You can save any data frame in R to a .csv file with the command `write.csv`. To save deck, run:

1. You should give `write.csv` the name of the data frame that you wish to save

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)  
>
```

2. you should provide a file name to give your file. R will take this name quite literally, so be sure to provide an extension.

Finally, you should add the argument `row.names = FALSE`. This will prevent R from adding a column of numbers at the start of your data frame. These numbers will identify your rows from 1 to 52, but it is unlikely that whatever program you open `cards.csv` in will understand the row name system. More than likely, the program will assume that the row names are the first column of data in your data frame. In fact, this is exactly what R will assume if you reopen `cards.csv`. If you save and open `cards.csv` several times in R, you'll notice duplicate columns of row numbers forming at the start of your data frame. I can't explain why R does this, but I can explain how to avoid it: use `row.name = FALSE` whenever you save data with `write.csv`.

Saving your work

You have several options for saving your work:

- You can save individual variables with the ***save()*** function.
- You can save the entire workspace with the ***save.image()*** function.
- You can save your R script file, using the appropriate save menu command in your code editor.

Suppose you want to save the value of ***yourname***. To do that, follow these steps:

1. Find out which working directory R will use to save your file by typing the following:

```
> getwd()  
[1] "c:/users/andrie"
```

The default working directory should be your user folder. The exact name and path of this folder depend on your operating system.

Important Hint: If you use the Windows operating system, the path is displayed with slashes instead of backslashes.

2.Type the following code in your console, using a filename like *yourname.rda*, and then press Enter.

```
> save(yourname, file="yourname.rda")
```

R silently saves the file in the working directory. If the operation is successful, you don't get any confirmation message.

3.To make sure that the operation was successful, use your file browser to navigate to the working directory, and see whether the new file is there.

2. Data Manipulation

Data Manipulation in R can be carried out for further analysis and visualisation.

Let us see a few basic data structures in R:

1. **Vectors in R**- 1-dimensional data.

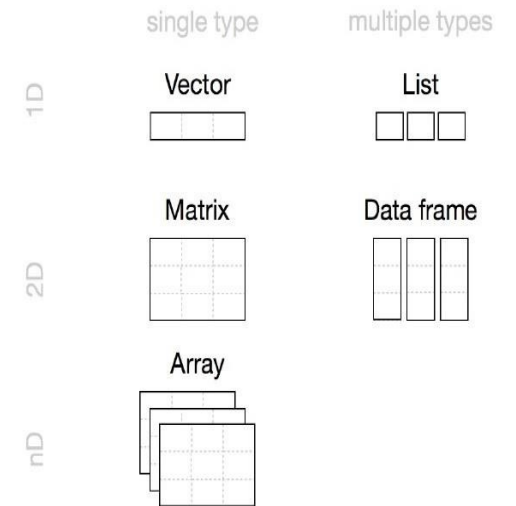
Types – *integer, numeric, logical, character, complex*.

2. **Matrices in R**- These are rectangular collections of elements and are useful when all data is of a single class that is numeric or characters. Dimensions – two, three, etc.

3. **Lists in R**- These are ordered containers for arbitrary elements and are used for higher dimension data, like customer data information of an organization

4. **Data Frames**- These are two-dimensional containers for records and variables and are used for representing data from spreadsheets etc. It is similar to a single table in the database.

5. **Arrays**- While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension



Sample() command in R

As we have seen, samples are created from data for analysis. To create samples, sample() command is used and the number of samples to be drawn are mentioned.

For example, to create a sample of 10 simulations of a die, below command is used:

```
> sample(1:6, 10, replace=TRUE)
[1] 4 1 4 2 5 5 1 2 3 4
```

>

sample() should always produce random values but it does not happen with the test code sometimes. If substituted with a seed value, the sample() command always produces random samples.

The seed value is the starting point for any random number generator formula. Seed value defines both, the initialization of the random number generator along with the path that the formula will follow.

Let us see how seed value is used:

```
> set.seed(100)
> sample(1:5, 10, replace = TRUE)
[1] 2 3 1 2 4 4 2 3 2 5
>
```

Hint: Computers don't generate truly random numbers—they are deterministic, which means that they operate by a set of rules. You can mimic randomness by specifying a set of rules. For example, “take a number x , add $900 + x$, then subtract 52.” In order for the process to start, you have to specify a starting number, x (the seed). Let's take the starting number 77:

Adding Calculated Fields to Data

R makes it easy to perform calculations on columns of a data frame because each column is itself a vector.

We will see how to calculate the ratio between the lengths and width of the sepals.

The command for the same is:

```
> data(iris)
> x <- iris$Sepal.Length / iris$Sepal.Width
> head(x)
[1] 1.457143 1.633333 1.468750 1.483871 1.388889 1.384615
>
```

Creating Subgroups or Bins of Data

Most statisticians often draw histograms to investigate their data. As this type of calculation is common when you use statistics, R has some functions for it

1. cut() function in R

`cut()` function groups values of a variable into larger bins. It creates bins of equal size and classifies each element into its appropriate bin.

Let us see how cut works in R with an example:

```
> frost <- c(1,2,3)
> cut(frost, 3, include.lowest=TRUE)
[1] [0.998,1.67] (1.67,2.33] (2.33,3]
Levels: [0.998,1.67] (1.67,2.33] (2.33,3]
> cut(frost, 3, include.lowest=TRUE, labels=c("Low", "Med", "High"))
[1] Low Med High
Levels: Low Med High
```

merge() Function in R

Let's see the use of merge() function.

The merge() function is used to combine data frames. Let us see this with an example: [every.states](#)

Try to run this code

```
every.states <- as.data.frame(state.x77)
every.states$Name <- rownames(state.x77)
rownames(every.states) <- NULL
str(every.states)

#Creating a subset of freezing states
freezing.states <- every.states[every.states$Frost>150
                               , c("Name", "Frost")]
freezing.states

#Creating a subset of big states
big.states <- every.states[every.states$Area>=100000
                           , c("Name", "Area")]
big.states

#Using the merge function
merge(freezing.states, big.states)
```

Results

```
> every.states <- as.data.frame(state.x77)
> every.states$Name <- rownames(state.x77)
> rownames(every.states) <- NULL
> str(every.states)
'data.frame':    50 obs. of  9 variables:
 $ Population: num  3615 365 2212 2110 21198 ...
 $ Income   : num  3624 6315 4530 3378 5114 ...
 $ Illiteracy: num  2.1 1.5 1.8 1.9 1.1 0.7 1.1 0.9 1.3 2 ...
 $ Life Exp : num  69 69.3 70.5 70.7 71.7 ...
 $ Murder   : num  15.1 11.3 7.8 10.1 10.3 6.8 3.1 6.2
10.7 13.9 ...
 $ HS Grad  : num  41.3 66.7 58.1 39.9 62.6 63.9 56 54.6
52.6 40.6 ...
 $ Frost    : num  20 152 15 65 20 166 139 103 11 60 ...
 $ Area     : num  50708 566432 113417 51945 156361 ...
 $ Name     : chr  "Alabama" "Alaska" "Arizona"
"Arkansas" ...
```

```
> #Creating a subset of freezing states  
> freezing.states <- every.states[every.states$Frost>150  
+                               , c("Name", "Frost")]
```

```
> freezing.states
```

	Name	Frost
2	Alaska	152
6	Colorado	166
19	Maine	161
23	Minnesota	160
26	Montana	155
28	Nevada	188
29	New Hampshire	174
34	North Dakota	186
41	South Dakota	172
45	Vermont	168
50	Wyoming	173

```
> #Creating a subset of big states
> big.states <- every.states[every.states$Area>=100000
+                               , c("Name", "Area")]
> big.states
```

	Name	Area
2	Alaska	566432
3	Arizona	113417
5	California	156361
6	Colorado	103766
26	Montana	145587
28	Nevada	109889
31	New Mexico	121412
43	Texas	262134

```
> #Using the merge function
```

```
> merge(freezing.states, big.states)
```

```
  Name Frost  Area
```

```
1  Alaska 152 566432
```

```
2 Colorado 166 103766
```

```
3  Montana 155 145587
```

```
4  Nevada 188 109889
```

Sorting and Ordering Data in R using sort() and order() in R

A common task in data analysis and reporting is sorting information. You can answer many everyday questions with sorted tables of data that tell you the best or worst of specific things;

Let's first create data frame and then we will sort it. Then, we will use some.states command to create a data fram

```
some.states <- data.frame( Region = state.region, + state.x77)
some.states <- some.states[1:10, 1:3]
sort(some.states$Population) #Command to sort Population in ascending order
sort(some.states$Population, decreasing=TRUE) #Command to sort Population
#in descending order
order.pop <- order(some.states$Population) #Another way of sorting
some.states[order.pop, ] #In ascending order
order(some.states$Population, decreasing=TRUE) #Descending Order
```

Results

```
> some.states <- data.frame( Region = state.region, + state.x77)
> some.states <- some.states[1:10, 1:3]
> sort(some.states$Population)  #Command to sort Population in ascending order
[1] 365 579 2110 2212 2541 3100 3615 4931 8277 21198
> sort(some.states$Population, decreasing=TRUE)  #Command to sort Population
[1] 21198 8277 4931 3615 3100 2541 2212 2110 579 365
>                                     #in descending order
> order.pop <- order(some.states$Population)    #Another way of sorting
> some.states[order.pop, ]      #In ascending order
```

	Region	Population	Income
Alaska	West	365	6315
Delaware	South	579	4809
Arkansas	South	2110	3378
Arizona	West	2212	4530
Colorado	West	2541	4884
Connecticut	Northeast	3100	5348
Alabama	South	3615	3624
Georgia	South	4931	4091
Florida	South	8277	4815
California	West	21198	5114

```
> order(some.states$Population, decreasing=TRUE)  #Descending Order
[1] 5 9 10 1 7 6 3 4 8 2
>
```

3. The Machine Learning Part

In case you are new to statistics, there are some very solid sources that explain the basic concepts while making use of R:

- [Andrew Conway's Introduction to statistics with R](#) (online interactive coding course)

<https://www.datacamp.com/tracks/learn-statistics-with-r>

- Furthermore there are some very interesting blogs to kickstart your Machine Learning <https://www.datacamp.com/community/tutorials/machine-learning-in-r>

- Good Book An Introduction to Machine Learning with R

<https://lgatto.github.io/IntroMachineLearningWithR/index.html#caution>

- Google crash course in Machine Learning

<https://developers.google.com/machine-learning/crash-course/descending-into-ml/video-lecture>

4. Data Visualization

If you want to get started with visualizations in R, take some time to study the [ggplot2](#) package. One of the (if not the) most famous packages in R for creating graphs and plots. ggplot2 makes intensive use of the [grammar of graphics](#), and as a result is very intuitive in usage (you're continuously building part of your graphs so it's a bit like playing with lego). There are tons of resources to get you started such as this

https://www.datacamp.com/courses/data-visualization-with-ggplot2-1?tap_a=5644-dce66f&tap_s=14201-e863d5

Besides ggplot2 there are multiple other packages that allow you to create highly engaging graphics and that have good learning resources to get you up to speed. Some of our favourites are:

- [ggvis](#) for interactive web graphics
<http://ggvis.rstudio.com/>
- [googleVis](#) to interface with google charts.
<https://developers.google.com/chart/interactive/docs/gallery>
- [Plotly for R](#)
<https://plotly.com/r/>

5.Reporting your results

One of the best way to share your models, visualizations, etc is through dynamic documents.

<https://rmarkdown.rstudio.com/>

(based on [knitr](#) and [pandoc](#)) is a great tool for reporting your data analysis in a reproducible manner though html, word, pdf, ioslides, etc. This 4 hour tutorial on https://www.datacamp.com/courses/reporting-with-r-markdown?tap_a=5644-dce66f&tap_s=14201-e863d5

explains the basics of R markdown. Once you are creating your own markdown documents, make sure

<https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>

is on your desk.

Big Data Types

Big Data Types

Variety is one of the principles of Big Data as described previously. The Big Data can be divided into three types:

- (1) Structured Data,
- (2) Semi-Structured Data, and
- (3) Unstructured Data

(1) Structured Data

Structured data generally refers to data that has a defined length and format. Most organizations are storing large amounts of structured data in various divisions, in normalised/deformalised formats in a database: Data warehouses, relational database management system (RDMSs), and various other environments.

The data can be queried using a language like structured query language (SQL) in which the datasets can be updated with new data, and deleted, read or any other activity. The evolution of technology provides newer sources of structured data being produced - often in real time and in large volumes. The sources of data are divided into three categories:

(i) Computer- or Machine-Generated Structured Data:

- Sensor data
- Web log data
- Point-of-sale data:
- Financial data

(ii) Human-Generated Data:

- Input data
- Click-stream data
- Gaming-related data

Example of Structure d Data:
Simple Invoices Tables from Northwind

Access's Datasheet view of an Invoices table, which is based on the Northwind.mdb sample database's Orders table. The Invoice No field is the primary key. Values in the OrderID, CustomerID, EmployeeID, and ShipperID fields relate to primary key values in Northwind's Orders, Customers, Employees, and Shippers tables. A field that contains values equal to those of primary key values in other tables is called a foreign key [field].

This simple Invoices table was created from the Northwind Orders table and doesn't take advantage of Access's extended properties, such as the field captions, lookup fields, and subdatasheets in the Datasheet view of the Orders table.

Foreign fields

Primary key		Fields					
Invoice No	OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, ShipDate, ShipVia						Records
100000100	10356 WA DK	6	11/1a/2005	12/16/2005	U/21/2005	2	
100000101	0357 LIAS	1		05			
1000001_1	0358 IAMA	S	11/9/2005	2/17/2006	11/2/2005	3	
10000012	0359 SEVES	S		5			
10000013	0360 BIO P		11/20/2005	2/11/2005	1/27/2005	1	
100000111	10361 QUIO	1	11/11/2005	U/19/2005	1/26/2005	3	
100000115	10362 BO P	3	11/12/2005	U/21/2005	2/2/2005	3	
100000116	10363 DRACO	4		5			
100000117	10364 EASIC	1	11/12/2005	U/20/2005	12/3/2005	2	
100000118	10365 S A ON	3		S			
100000119	10366 GALEO	8	11/25/2005	12/23/2005	11/28/2005	1	
100000120	10367 IJAFFE	7		>S			
100000121	0368 ER SH	2	11/26/2005	12/24/2005	U/4/2005	3	
100000122	10369 SPUR	8		OS			
100000123	0370 CHOPS	6	11/26/2005	1/7/2006	12/ /2005	1	
100000124	10371 IAMA	S	11/27/2005	12/25	12/2/2005	2	
100000125	0372 QUEEN	S	11/28/2005	1/9/2006	U/MJ/2005	2	
100000126		4	12/28/2005	2/M/2006	11/2/2005	3	
			12/29/2005	1/2/2006	12/11/2005	3	
				hrm	11/23/2005	2	

(2) Semi-Structured Data

Semi-structured data is a kind of data that falls between structured and unstructured data. This type of data became a talking point. **Mostly data coming from Facebook, Twitter, Blogs, publically available websites, etc.** makes the basis of semi-structured data. These data sources usually have defined structures and mostly contain text information. The free flow text generated through the social media is the only unstructured component whilst the remaining data is structured.

Most of the times, the social data is mistaken with unstructured data. The social data is NOT unstructured data, it is semi-structured and in fact, some of the social data contains industry standard structures. Social media data: This data is generated from the social media platforms such as YouTube, Facebook, Twitter, LinkedIn, Posts, Favourites, Sentiment, and Flickr, etc. **This creation data process or the process of gathering social media data is normally called “mining”.**

Example of Semi-Structured Data:

Google Trends: A Web-Based Tool for Real-Time Surveillance of Disease Outbreaks

Google Flu Trends can detect regional outbreaks of influenza 7-10 days before conventional Centers for Disease Control and Prevention surveillance systems. We describe the Google Trends tool, explain how the data are processed, present examples, and discuss its strengths and limitations. Google Trends shows great promise as a timely, robust, and sensitive surveillance system. It is best used for surveillance of epidemics and diseases with high prevalences and is currently better suited to track disease activity in developed countries because to be most effective, it requires large populations of Web search users. Spikes in search volume are currently hard to interpret but have the benefit of increasing vigilance. Google should work with public health care practitioners to develop specialized tools, using Google Flu Trends as a blueprint, to track infectious diseases. Suitable Web search query proxies for diseases need to be established for specialized tools or syndromic surveillance. This unique and innovative technology takes us one step closer to true real-time outbreak surveillance. Google Flu Trends Web tool interface ([available at http://www.google.com/trends/explore?q=flutrends](http://www.google.com/trends/explore?q=flutrends)).

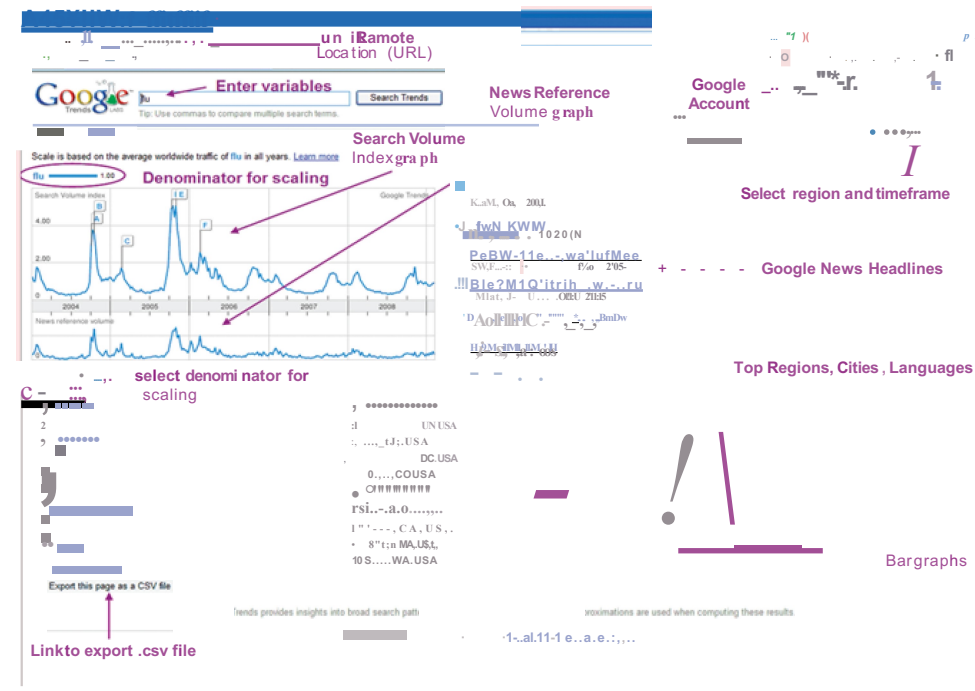


Figure (1): Google Trends output for Web search queries for the term "flu" worldwide from January 2004 to March 2009.

Reference : Herman Anthony Camero, and Eleftherios Mylonakis, "Google Trends: A Web-Based Tool for Real-Time Surveillance of Disease Outbreaks" SURFING THE WEB • CID 2009 :49 (15 November) • 1557-1565 (2016)

(3) Unstructured Data

Unstructured data does not have any defined, consistent fields and it may even do not have any numbers and text. Unstructured data can be divided also into either machine generated or human generated and described as flows:

(i) Machine-Generated Unstructured Data Examples:

- Satellite images
- Scientific data
- Photographs and video
- Radar or sonar data

(ii) Human-generated Unstructured Data Examples:

- Mobile and Voice data
- Web behavior and content
- Image and Video Data
- Machine Data

Creating Subsets of Data in R

The process of creating samples is called subsetting. Different methods of subsetting in R are:

1. \$

The dollar sign operator selects a single element of data. The result of this operator is always a vector when we use it with a data-frame.

2. [[

Similar to \$ in R, the double square brackets operator in R also returns a single element, but it offers the flexibility of referring to the elements by position rather than by name. It can be used for data frames and lists.

3. [

The single square bracket operator in R returns multiple elements of data. The index within the square brackets can be a numeric vector, a logical vector, or a character vector.

For example: To retrieve 5 rows and all columns of already built-in dataset iris, the below command, is used:

```
> data(iris)
> iris[1:5, ]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5         1.4         0.2   setosa
2          4.9         3.0         1.4         0.2   setosa
3          4.7         3.2         1.3         0.2   setosa
4          4.6         3.1         1.5         0.2   setosa
5          5.0         3.6         1.4         0.2   setosa
>
```

Example of Unstructured Data

Satellite data Example [7]

The use of satellites to predict weather is well known. But satellite data can also map differences in the Earth's surface so precisely that we can calculate how much water is stored in even the most isolated of aquifers (Sullivan, 2015) or analyse the quality of soil (Lenhardt, 2015).

Light emissions picked up by satellites are also being used to proxy poverty levels and track GDP growth to supplement national accounting in data-poor countries (Henderson et al., 2012).

In Indonesia, researchers have used data on electrification and economic growth for 5,000 sub-districts in Indonesia between 1992 and 2008 (Olivia et al 2014).



Figure (2): An example of raw and postprocessed satellite data. *(Left)* Along-track satellite observations of sea surface height from the JASON-II satellite for May 20, 2010. *(Middle)* A 12-day composite of five satellites centered on May 20, 2010. *(Right)* The postprocessed data from May 20, 2010. The altimeter products were produced by Ssalto/Duacs and distributed by AVISO, with support from CNES (www.aviso.oceanobs.com/duacs/).

Reference: James H. Faghmous, and Vipin Kumar/ " A Big Data Guide to Understanding Climate Change: The Case for Theory-Guided Data Science, BigData. 2(3): 155- 163 (2014).

Thank you