Software Applications and Tools

Matteo Orsoni & Sara Garofalo
University of Bologna

The powerful, free, and community-vetted tools discussed in this chapter arguably represent the best options currently available for statistical analysis, due to their zero cost, high quality, and constant peer review by the global statistical community. Additionally, if you are concerned about Open Science practices and the increasing demands for research transparency and reproducibility, the choice of software becomes a methodological necessity. The commitment to sharing analytical code and ensuring methodological transparency requires a deliberate choice of statistical software that inherently supports these goals. In the modern research landscape, tools fall primarily into two important categories: programming environments and Graphical User Interface (GUI)-based applications. Both categories, when utilizing Open Source technology, can effectively serve the goals of transparency and reproducibility.

Programming vs. GUI environments

Programming environments, such as R (often utilized within the RStudio integrated development environment) and Python, are generally regarded as the pinnacle of computational reproducibility. Their principal strength is rooted in their architecture: every single action taken (from initial data cleaning and transformation to the final execution of complex statistical models and visualization generation) is meticulously documented and executed via a script or code file. This inherent feature ensures the analysis is fully auditable and perfectly repeatable by any researcher with access to the code and the necessary dependencies. For the researcher, the investment required to master these environments, despite the steep learning curve, yields profound and personal benefits that extend well beyond mere adherence to reproducibility mandates. Programming grants unlimited analytical flexibility; researchers are never constrained by a software's built-in menus but can devise, implement, and even pioneer custom analyses, statistical methods, and highly specific visualizations tailored exactly to their unique research questions. Furthermore, developing skills in programming languages like R or Python is a significant career

advantage, providing a powerful, marketable skill set that is highly transferable across diverse academic and industry settings. This proficiency ultimately grants researchers total control over the entire analytical workflow, transforming the process from a series of fixed choices into a robust, traceable, and infinitely adaptable computational method. In contrast, Graphical User Interface (GUI) applications (i.e., interfaces that utilize Graphical User Interface elements like buttons, menus, and icons) offer unparalleled accessibility. They enable researchers, students, and those without a coding background to perform sophisticated statistical analysis quickly and efficiently through intuitive visual menus. The newest generation of Open Source GUIs has successfully integrated reproducible principles, making them powerful, intuitive tools that democratize access to rigorous statistical methods for the modern open scientist.

GUI-based software

The drive to make research both open and accessible has led to the development of robust, free, and cross-platform GUI-based statistical programs. These solutions combine the ease of traditional menu-driven interfaces with the transparent execution of powerful Open Source statistical engines. The leading examples, JASP and jamovi, are both built upon the R statistical language, assuring that their analytical power is both powerful and fully verifiable.

JASP (JASP Team, 2025), an acronym for Jeffreys's Amazing Statistics Program, is a sophisticated, free, and fully Open Source statistical software developed by the University of Amsterdam. The name is a direct tribute to Sir Harold Jeffreys, the prominent statistician who championed Bayesian inference, perfectly reflecting JASP's core methodological mission. JASP is conceived specifically to offer a robust, user-friendly, and methodologically rigorous alternative to legacy proprietary GUI programs.

- **Core philosophy:** JASP is distinguished by its seamless, integrated presentation of both Bayesian and Frequentist statistics for nearly every available analysis. This dual presentation encourages researchers to engage in deeper statistical inference beyond sole reliance on traditional p-values.
- **Engine and reproducibility**: The entire analytical engine is built upon the powerful and verifiable R statistical language. JASP enhances transparency by generating and storing an internal log of all user actions (analogous to code) and streamlines the reporting phase by automatically generating polished, APA-style tables directly from the output.
- **Modularity and Extension**: JASP's extension system is primarily focused on enhancing statistical analysis. Advanced features, such as the comprehensive Bayesian suite, Meta-Analysis, Factor Analysis, and Network Analysis, are organized into distinct, built-in modules. These modules are curated directly by the

JASP core team and collaborators, ensuring consistency and reliability across sophisticated statistical procedures.
- **Learning Resources**: The official JASP Resources page is an essential hub offering free, continuously updated materials, including downloadable textbooks, detailed video tutorials, and ready-to-use datasets for practice: https://jasp-stats.org/resources/.

Jamovi (jamovi Project, 2025) represents another premier free, open-source, and cross-platform statistical solution that utilizes a familiar spreadsheet. Its mission is to democratize advanced statistics by providing accessibility without sacrificing analytical power. Jamovi's entire engine is founded upon the R programming language, meaning every statistical test executed within the GUI is driven by highly validated and community-supported R packages.
- **Core Philosophy**: jamovi provides an intuitive, streamlined interface focusing on a broad range of standard frequentist statistical tests, making it immediately familiar to users transitioning from other GUI packages. It prioritizes ease of use and seamless integration of data input and output within the same interface.
- **Engine and Reproducibility**: Built on the R statistical language, jamovi also provides a syntax mode that allows users to view the underlying R code equivalent of their GUI actions, significantly enhancing the transparency of the analysis.
- **Modularity and Extension**: jamovi's primary strength in modularity lies in its expansive, community-driven module system. Users can effortlessly expand core capabilities by installing a wide range of third-party modules. This allows jamovi to offer specialized analytical procedures, from complex mediation and moderation models to dedicated clinical trial tools, granting it a high degree of flexibility that closely rivals the customizability of a full programming environment.
- **Learning Resources**: The jamovi Community page acts as the central clearinghouse for extensive, free instructional materials. It provides direct links to webinars, video guides, instructional articles, and access to all the specialized, user-created content for its expansive module library: https://www.jamovi.org/community.html.

Programming environments: R and Python

R (R Core Team, 2025) is a powerful, free, and open-source programming language for statistical computing and graphics, and RStudio (Posit team, 2025) is a highly recommended Integrated Development Environment (IDE) that makes using R much easier.

To use these, you need to download and install both on your PC. This two-

step process ensures you have both the engine (R) and the user-friendly dashboard (RStudio).

Before installing RStudio, you must install R.

- Navigate to the Comprehensive R Archive Network (CRAN) website: https://cran.r-project.org/
- Click on the download link relevant to your operating system (OS).
- Follow the instructions on the specific page.
- Download the latest installer file and run it. Accept the default settings during installation, unless you have a specific reason to change them.

RStudio

Once R is installed, you can install RStudio, which you will use for all your data analysis.

- Go to the RStudio website (Posit): https://posit.co/download/rstudio-desktop/
- Click on the download link relevant to your operating system (OS).
- Follow the instructions on the specific page.
- Download the latest installer file and run it. Accept the default settings during installation, unless you have a specific reason to change them.

After the installation is complete, you can launch RStudio. If the installation was successful, the RStudio interface will open, and you should see the console pane where the R version you installed is listed, confirming that both programs are communicating correctly.

RStudio Workflow for Data Analysis

Once you have R and RStudio installed, the following steps cover the basic workflow required to start any statistical analysis. You will perform these actions in the Console or in an R Script within RStudio.

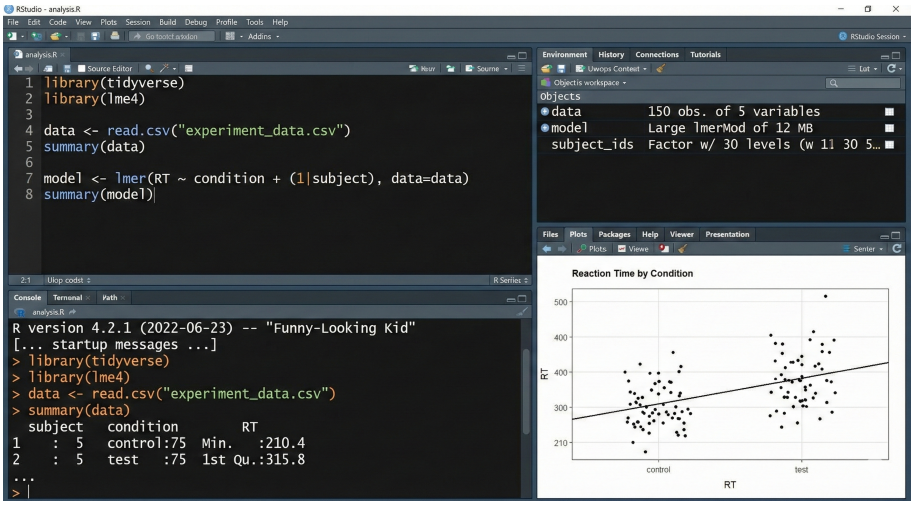
Figure 1, depict the typical RStudio interface.

RStudio, it usually presents four main rectangular sections (called "panes"). Understanding what each does is crucial for an efficient workflow.

The Source Editor (Top-Left).

Note: If you only see the Console on the left when you first open RStudio, go to File > New File > R Script to open the Editor.

This is where you will spend most of your time. It is essentially a smart text editor tailored for writing code, edit, and save your R scripts. Unlike the Console (Bottom-Left), code written here is not run immediately. This



1. Interface of the RStudio IDE.

allows you to write a complete analysis, save it, share it with your colleagues, and run it later. This is essential for reproducibility.

You can write code here and send it down to the Console to be executed either by clicking the "Run" button at the top of the pane or by pressing Ctrl + Enter (Windows) or Cmd + Enter (macOS) on your keyboard.

The Console (Bottom-Left).

This section allows to execute commands and show results.

When you "Run" code from the Source Editor, it automatically appears in the Console and is executed. You can also type commands directly into the Console next to the > prompt and press Enter for immediate results.

The Environment & History Pane (Top-Right).

This pane helps you manage what R currently "see" in your session. The Environment Tab, contains a list of everything you have currently loaded into R's memory.

If you load a dataset, in 1, a data object, it will appear here. It will show you a summary (e.g., "150 obs. of 5 variables"). If you run a regression model and save it as model, that object will also appear here.

The Files, Plots, Packages, & Help Pane (Bottom-Right).

This is a multi-purpose utility pane with several important tabs: ("Files Tab", "Plots Tab", "Packages Tab", "Help Tab").

In the Plots Tab, whenever you generate a graph (e.g., using ggplot2 or base R graphics), it will appear here. You can use the "Export" button

in this tab to save your graphs as images (PNG, JPEG) or PDFs for your manuscripts.

Now, you have a brief overview on how to approach to RStudio. The next stages are how to do the basics for Data Analysis.

1. Installing and Loading Packages

Packages are collections of functions and data sets created by the R community that extend R's capabilities (e.g., for advanced modeling, data manipulation, or specific statistical tests). To install a package, you only need to use the function `install.packages()`. This function will download and install the package from CRAN.

```
1
2 # Install the 'tidyverse' package (a collection of packages
    like ggplot2 and dplyr)
3 install.packages("tidyverse")
4
5 # Install a package often used for mixed-effects modeling
6 install.packages("lme4")
```

Once, you have installed the package/s, you must load it/them into your current R session every time you start RStudio. You can do this by using the `library()` function.

```
1
2 # Load the 'tidyverse' package for use in the current session
3 library(tidyverse)
```

2. Setting Your Working Directory

The working directory is the default location on your computer where RStudio looks for files to load and saves files to. Setting it correctly is crucial for reproducibility and ease of file access. You can set the directory using the `setwd()` function, providing the full path to the folder where your data is stored.

```
1
2 # Example of setting the working directory (adjust the path for
    your OS and folder)
3 # For Windows users:
4 setwd("C:/Users/YourName/Documents/PsychologyDissertationData")
5
6 # For macOS/Linux:
7 # setwd("~/Documents/PsychologyDissertationData")
```

Always verify the current working directory using the `getwd()` function. In RStudio, it is often easier to use RStudio Projects `File > New`

`Project...` A Project automatically sets the working directory to the project folder, making your code highly reproducible across different computers.

3. Loading Data

Once the working directory is set, you can load data files directly using their filename. We are going to see how to load two different but mostly common data format: `.csv`, and `.xlsx`.

Loading a CSV file (`.csv`). Comma Separated Values (CSV) files are the most common and robust format for data exchange. You can use the `read.csv()` function.

```

# Read a CSV file named 'experiment_results.csv'
# The data is saved into a variable called 'my_data'
my_data <- read.csv("experiment_results.csv")

# Important arguments:
# header = TRUE (tells R that the first row contains variable names)
# sep = "," (specifies the delimiter, which is a comma for a standard CSV)
```

Loading an Excel file (`.xlsx`). To load Excel files, you first need to install and load a specific package, such as `readxl` (part of the tidyverse). Then, use the `read_excel()` function.

```

# 1. Install and load the readxl package
install.packages("readxl")
library(readxl)

# 2. Read the Excel file named 'survey_data.xlsx'
# If your data is on a specific sheet (e.g., Sheet2), use the 'sheet' argument:
survey_data <- read_excel("survey_data.xlsx", sheet = "Sheet2")

# If the sheet is the first one, you can often omit the 'sheet' argument.
```

Python (Python Software Foundation, 2016) is a free, open source, and Oriented-Object-Programming (OOB) language. As for R, we'll follow a two-step process: installing Python itself and then setting up the highly popular Visual Studio Code (VS Code) editor to serve as your Integrated Development Environment (IDE).

You must install Python before configuring VS Code to use it.

- Navigate to the official Python website: https://www.python.org/downloads/
- The website should automatically suggest the latest stable version for your operating system (Windows, macOS, or Linux).
- Run the downloaded installer file. **Crucial Step for Windows Users**: On the first screen of the installer, make sure to check the box that says "Add python.exe to PATH" before clicking "Install Now." This makes it much easier for VS Code and other tools to find Python.

Configure Visual Studio Code

VS Code (Microsoft, 2025) is a powerful, lightweight, and customizable editor that is excellent for Python development.

- Navigate to the official VS Code website: https://code.visualstudio.com/
- Download the installer for your operating system and run it, accepting the default settings.
- Install the Python Extension: Once VS Code is installed and opened, you need to add the official Python Extension:
  - Click on the Extensions icon (four squares, usually in the far left sidebar).
  - Search for "Python"
  - Click the "Install" button. This extension provides features like code completion, debugging, and running code directly in the IDE.

VS Code Workflow for Data Analysis with Python

Once you have Python installed and the necessary **VS Code Extensions** (like the official Python extension) configured, the following steps cover the basic workflow required to start any data analysis. You will perform these actions primarily in the **Editor** and the **Terminal** panes within VS Code.

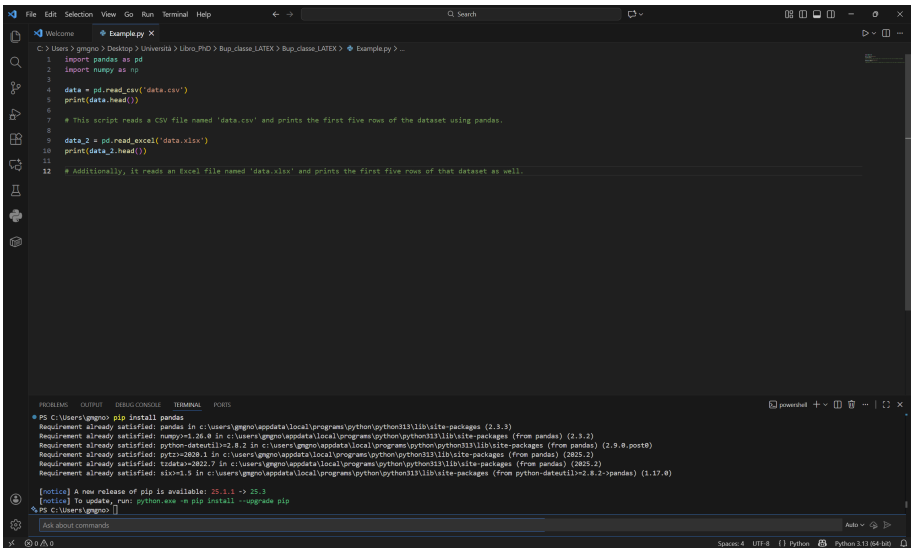
Figure 2, depicts the typical Visual Studio Code interface when set up for Python development.

The Visual Studio Code interface is highly customizable, but its typical layout presents several key functional areas crucial for data science:

The Editor Pane (Central Area) *Note: This is the equivalent of the RStudio Source Editor*.

This is the main area where you will spend most of your time. It is a powerful text editor tailored for writing and editing code.

Note: To start, navigate to File > New File and save it immediately with a `.py` extension (e.g., `analysis.py`).



2. Interface of the Visual Studio Code IDE for Python development.

- **Purpose**: To write, edit, and save your Python scripts (`.py` files).
- **Execution**: Code written here is not executed immediately. You can run the entire script by clicking the **"Run Python File"** button (often an arrow in the top right corner) or run selected lines by pressing **Shift + Enter** on your keyboard (which sends the code to the Terminal or the interactive Python console). This approach is essential for **reproducibility**.

The Integrated Terminal (Bottom Pane) *Note: This acts as the Console and package manager*.

This section is vital as it acts as both the **Console** (where code is executed) and the environment manager (where packages are installed).

- **Execution (Console)**: When you run code from the Editor, it is automatically processed here. You can also type commands directly into the prompt (e.g., `pip install pandas`).
- **Output**: Statistical output, print statements, warnings, and errors appear in this pane.

The Explorer and Sidebar (Left Pane) *Note: This is where you manage files and extensions*.

This vertical sidebar manages the overall file structure and access to VS Code's core features.

- **Explorer Tab**: Displays the file structure of your project folder. This is where you quickly navigate between your Python scripts and data files.

- **Extensions Tab**: Allows you to manage and install add-ons like the **Python extension**, essential for features like code completion and debugging.

The Variables and Debug Pane *Note: The Variables section is the closest equivalent to R's Environment Pane*.

This pane is crucial for inspecting your data and code integrity, though it often requires starting the **Debugger**.

- **Variables**: When you run your code in Debug mode, this section displays all active objects and variables in memory (e.g., your `pandas` Data Frames). This is the best way to inspect the contents and shape of your data objects.
- **Breakpoints**: Here you manage **breakpoints**, which are points in the code where execution pauses, allowing you to step through the logic line-by-line to identify errors.

Now that you have set up Python and the VS Code IDE, you are ready to begin the fundamental steps of any data analysis project. Unlike R, where many functions are native, Python relies on powerful external libraries (the equivalent of R's packages), primarily Pandas for data handling.

Installing and Importing Packages (Libraries)

Python packages are installed using the package manager **pip** in the command line (Terminal), and the equivalent of loading a package is done using the **import** command in your script.

1. Installing Packages with `pip`

You only install a package once. In VS Code, access the terminal via `Terminal > New Terminal`.

```

# In the VS Code Terminal
# Install the essential scientific packages: Pandas (for DataFrames) and NumPy (for numerical operations)
pip install pandas numpy
```

2. Importing Packages with `import`

You must **import** the required libraries at the beginning of every Python script (`.py` file) where you intend to use them. Standard practice uses aliases (`as`) for convenience.

```

# At the top of your analysis.py script

# Import Pandas and assign the common alias 'pd'
import pandas as pd
```

```
# Import NumPy and assign the common alias 'np'
import numpy as np
```

Managing Your Working Directory

In Python, setting the working directory is less common than in R. By default, Python uses the directory where your **script (`.py` file) is saved** as the working directory. The best practice is to **save your script in the same folder as your data files**.

```
# The OS library is used to check the current directory.
import os

# Check the current working directory (it should be the folder
    where your script is located)
print(os.getcwd())
```

Loading Data

Once the necessary libraries are imported, you can load data into a **Pandas DataFrame** (the primary object for data analysis in Python).

Loading a CSV file (`.csv`)

The **Pandas** (The pandas development team, 2020) library uses the function `pd.read_csv()` for this purpose.

```
# Read a CSV file named 'experiment_results.csv'
# The data is saved into a variable called 'df_data'
df_data = pd.read_csv("experiment_results.csv")

# Print the first 5 rows to verify the data loaded correctly
print(df_data.head())
```

Loading an Excel file (`.xlsx`)

Pandas handles Excel files using `pd.read_excel()`. This requires the installation of an engine like `openpyxl`.

```
# In the VS Code Terminal, install the necessary engine:
pip install openpyxl
```

```
# In your analysis.py script:
import pandas as pd

# Read the Excel file named 'survey_data.xlsx'
```

```
# We specify the sheet name:
df_excel = pd.read_excel("survey_data.xlsx", sheet_name="Sheet1
    ")
```

References

jamovi Project. (2025). *Jamovi (version 2.6) [computer software]*. The jamovi project. Retrieved December 30, 2025, from https://www.jamovi.org/

JASP Team. (2025). *Jasp (version 0.95.3) [computer software]*. Retrieved December 30, 2025, from https://jasp-stats.org/

Microsoft. (2025). *Visual studio code (version 1.95) [computer software]*. Retrieved December 30, 2025, from https://code.visualstudio.com/

Posit team. (2025). *Rstudio: Integrated development environment for r*. Boston, MA, Posit Software, PBC. Retrieved December 30, 2025, from http://www.posit.co/

Python Software Foundation. (2016). *Python (version 3.x) [computer software]*. Retrieved December 30, 2025, from https://www.python.org/

R Core Team. (2025). *R: A language and environment for statistical computing*. Vienna, Austria, R Foundation for Statistical Computing. Retrieved December 30, 2025, from https://www.R-project.org/

The pandas development team. (2020). *Pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.3509134