

An interactive tool for data analysis, presentation, and interpretation with the help of R Shiny app

Short name of the tool (for the header): Shiny “aRt”

Slogan for the header: „Help with your data!”

Table of Contents

Introduction	1
Basics of using R for beginners	3
Step 1: Installing R.....	3
Step 2: Installing RStudio.....	4
Step 3. Installing R Packages	5
Table 1 Common Useful Packages for Beginners	6
Step. 4. How to Run and R Script in RStudio	7
Step 5. How to Read Your Own Data into R Using RStudio	8
2. Set Your Working Directory.....	8
Exploratory data analysis	13
Creating a Word Cloud	15
Scripts for your own R shiny experience	18
Script 1. – Time series analysis with Savitzki-Golay filter.....	18
Script 2 – Explanatory Data Analysis using “explore” package.....	22
Script 3 – Creating World Clouds for your own text	23

Introduction

R shiny application makes it possible to create an interactive web surface which makes it possible to interpret, show, figure and analyse data without specific knowledge on statistical

programs. This tool can be appropriate for presentation of your data, however, for scientific papers there is a need for further – more detailed analysis with some statistical programs. We developed some shiny apps, that can be useful for Socially engaged Research (SER) in Life Sciences (LS), that can be extended with more similar apps, in case Early Career Researcher (ECR) needs.

So far we prepared three apps.

1. The first app shows how time series data can be smoothed. We used the dataset of annual waterflow in the Nile river as an example for showing how row data should be smoothened for a more successful data presentation. Similar time series datasets occur in SER.
2. Second app allows ECR to browse their own excel data files, and the script connects them to explanatory data analysis – after choosing the target variable, other data within the file will be plotted against it; in case of discrete variables as a boxplot, in case of continuous variables as a regression line. This helps in finding eventual relationships and forming the hypothesis. For further analysis, however, it is advised to get a proper statistical analysis.
3. The third app gives a possibility to ECR to present their data in a form of a word cloud. This is a fancy way of presenting the most important clues of a text. ECR can copy a text into the box, and choose word cloud colours.

These apps offer a preliminary tool for interpreting, presenting and analysing their data and also provides help in communication with stakeholders during a socially engaged research.

Shiny apps make data representation significantly easier and more interactive; however, it is generally the best practice to keep core algorithms and processing logic on a server to ensure better performance, security, and scalability. Since we currently do not have the infrastructure or permissions to store program files on a central server, we have instead provided a detailed guide on how to install R on a local machine.

Alongside this, users can copy and paste our algorithms directly into their R environment to analyze their own data sources. This approach ensures accessibility for users at varying skill levels while still offering the functionality of the original Shiny apps. For users with some experience in R, particularly ECRs, executing the scripts should be straightforward using

the provided command lines. For those who are new to R, the documentation includes step-by-step instructions starting from installation, making it possible for beginners to run the programs independently.

This solution not only overcomes the current technical limitations but also encourages skill development among ECRs by giving them the opportunity to interact with the code and understand the data analysis process more deeply. In future updates, we aim to integrate cloud-based options to streamline this process further and improve accessibility for a broader user base.

Basics of using R for beginners

To use our R-based programs or run Shiny apps on your local machine, you'll need to install two essential components: **R**, the programming language used for statistical computing and data analysis, and **RStudio**, a powerful integrated development environment (IDE) that makes working with R much more efficient and user-friendly. Both are free and open-source.

Step 1: Installing R

R is the core software that runs the code and handles the statistical computations. Here's how to install it:

1. **Go to the CRAN website:**

Open your browser and visit the Comprehensive R Archive Network (CRAN) at:
<https://cran.r-project.org>

2. **Choose your operating system:**

On the CRAN homepage, you will see links for different operating systems:

- **Windows users:** Click on "Download R for Windows," then choose "base," and finally click the link to download the latest R installer (e.g., R-4.x.x-win.exe).
- **Mac users:** Click on "Download R for macOS" and select the latest .pkg file compatible with your version of macOS.
- **Linux users:** Click on "Download R for Linux" and follow the detailed instructions for your Linux distribution (e.g., Ubuntu, Debian, Fedora). You may need to use terminal commands to add repositories and install dependencies.

3. Run the installer:

Once the download is complete, open the installer file and follow the setup instructions. You can typically proceed with the default options unless you have specific preferences or system requirements. The installation process usually takes just a few minutes.

Step 2: Installing RStudio

While R can be used on its own, **RStudio** provides a much better user experience, especially for beginners. It offers a structured layout with multiple panels—for code, plots, data views, and file management—making it easier to develop and debug your R scripts or Shiny apps.

1. Go to the RStudio download page:

Visit the official RStudio website (now branded as Posit) at:
<https://posit.co/download/rstudio-desktop>

2. Choose your version:

Scroll down to the free version of **RStudio Desktop** and click the download button for your operating system:

- Windows: .exe installer
- macOS: .dmg installer
- Linux: Various .deb or .rpm packages depending on your distribution

3. Install RStudio:

After downloading, run the installer and follow the prompts. RStudio will automatically detect your existing R installation, so no additional configuration is usually needed.

Verifying the Installation

Once both R and RStudio are installed:

- Open **RStudio** from your applications or start menu.
- In the RStudio Console pane (bottom left), type a simple command like `2 + 2` and press Enter. If you see the result (`[1] 4`), everything is working correctly.

After installation, you can begin using RStudio to:

- Write and run R scripts
- Install R packages (e.g., shiny, ggplot2)
- Load and analyze data files
- Build interactive Shiny apps

If you are new to R, we recommend starting with basic tutorials available on the RStudio Education site, which includes beginner-friendly content and example projects.

Step 3. Installing R Packages

R packages are **collections of functions, data, and documentation** that extend the base functionality of R. Think of them as "add-ons" or "toolkits" that help you do specific tasks more easily.

For example:

- ggplot2 helps you create high-quality plots.
- dplyr makes data manipulation faster and more intuitive.
- shiny lets you build interactive web apps.
- readxl allows you to read Excel files.

Some more, useful packages are mentioned in Table.1.

R comes with a basic set of functions, but for most real-world tasks—especially in data analysis, statistics, or app development—you'll need to use additional packages.

Using packages lets you:

- Avoid writing complex code from scratch.
- Access community-tested tools and methods.
- Work faster and more efficiently.
- Reproduce standard analysis workflows with ease.

There are over 19,000 packages available on CRAN (Comprehensive R Archive Network), covering a wide range of tasks in data science, machine learning, visualization, statistics, and more.

To install a package, you only need to do it **once per computer** (or per R setup):

```
r
copy this line
install.packages("package_name")
```

Example: (installing ggplot2 package)

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

R will download the package from CRAN and install it on your system.

Choose a CRAN Mirror (First Time Only)

When installing your first package, R may prompt you to **select a CRAN mirror**. Just choose a mirror close to your location (e.g., a country or university near you). This helps speed up the download.

After installing, you must **load the package each time you start R**:

```
r
copy
library(package_name)
```

Example: (opening ggplot2 package)

```
r
library(ggplot2)
```

Now the package's functions are available for use.

Table 1 Common Useful Packages for Beginners

Package	Purpose
tidyverse	Collection of packages for data science (includes ggplot2, dplyr, etc.)

Package	Purpose
shiny	Building interactive web applications
readr	Reading CSV and text files
readxl	Reading Excel files
lubridate	Working with dates and times
plotly	Creating interactive plots

Example: Installing and Using readxl to Read Excel Files

```
r
install.packages("readxl") # Install once
library(readxl)           # Load in each session
data <- read_excel("mydata.xlsx")
```

Step. 4. How to Run and R Script in RStudio

An R script is a plain text file containing a sequence of R commands, usually saved with the .R extension. It allows you to **save**, reuse, and share your code.

Option 1: Run Line by Line or Selection

1. Open your .R script in RStudio (File > Open File...).
2. Click on any line or select multiple lines.
3. Press **Ctrl + Enter** (Windows/Linux) or **Cmd + Enter** (Mac) to run the selected code.
 - The command will appear in the **Console** and be executed.

Option 2: Run the Entire Script

1. Press **Ctrl + Shift + Enter** (Windows/Linux) or **Cmd + Shift + Enter** (Mac). This runs the entire script from top to bottom.
2. Or click the **Source** button at the top-right of the script editor pane.

Option 3: Using the `source()` Function

In the Console or another script, you can run a saved script using:

```
r
source("path/to/your_script.R")
```

Example:

```
r
source("C:/Users/YourName/Documents/my_script.R")
```

This is helpful for automating workflows or running scripts from another R file.

Always make sure your **working directory** is set to where your script is located, or use the full file path. You can always check the working directory with `getwd()` and change it with `setwd("your/path")`.

Step 5. How to Read Your Own Data into R Using RStudio

Once you have R and RStudio installed, the next important step is to import your own data so you can start analyzing it. R supports many data formats, but the most commonly used are **CSV**, **Excel**, and **text files**.

1. Prepare Your Data File

Make sure your data file is saved in a known location on your computer, such as the **Desktop** or **Documents** folder. For best compatibility:

- Use **CSV (.csv)** or **Excel (.xlsx)** format.
- Ensure the file has a header row (column names).
- Avoid special characters or empty columns/rows.

2. Set Your Working Directory

Before importing data, it's good practice to tell R where your files are located.

In RStudio:

1. Click Session > Set Working Directory > Choose Directory.
2. Navigate to the folder where your data file is saved and click Open.
3. Alternatively, use this command:

```
r  
setwd("C:/Users/YourName/Documents/MyData")
```

This ensures that when you import data, R looks in the correct location.

3. Import Your Data

Option A: Using the RStudio GUI (recommended for beginners)

1. In the **Environment** pane (top-right), click Import Dataset.
2. Choose the file type:
 - From Text (readr) for CSV files
 - From Excel if you're using Excel files (you may need to install the readxl package)
3. Browse to your file and click Open.
4. Review the preview window and click Import.

RStudio will automatically generate the code for you in the script window, such as:

```
r  
data <- read.csv("mydata.csv")
```

Option B: Using Code (flexible and reproducible)

For CSV files:

```
r  
data <- read.csv("mydata.csv")
```

For Excel files (requires readxl package):

```
r
install.packages("readxl") # Run this once
library(readxl)
data <- read_excel("mydata.xlsx")
```

For tab-separated files (.txt or .tsv):

```
r
data <- read_delim("mydata.txt")
```

4. Check Your Data

After importing, check that everything loaded correctly:

```
r
head(data)      # View the first few rows
str(data)       # See the structure of the dataset
summary(data)   # Get a summary of each column
```

You can also click on the dataset name in the “Environment pane” to open it in a spreadsheet-style view.

If you're unsure about the file path or name, you can drag and drop the file into the **RStudio Console**—it will print the full path automatically.

In the following sessions we will show you three possible ways of interactive visualisation of your data. These examples can be applicable during your scientific work as well as in case of interactions with stakeholders.

Smoothing Time Series Data

Smoothing is a widely used technique in time series analysis, aimed at reducing short-term fluctuations or irregularities in data to reveal the underlying structure. By applying smoothing, analysts can identify trends, seasonal patterns, or cycles that may otherwise be obscured by noise or random variation.

Time series data often includes a combination of long-term trends, seasonal components and irregular, unpredictable fluctuations (noise). Smoothing techniques help separate the meaningful signal from the noise by averaging data points over a certain window or applying weighted methods. This allows analysts to better observe the true variance of the data over time and is especially useful for: visual interpretation, Trend detection, Model preparation for forecasting.

Smoothing is used by a wide range of professionals and researchers, including: data scientists and statisticians (working in analytics or modeling), Economists and financial analysts (dealing with economic indicators or stock prices), Epidemiologists and public health experts (checking disease progression over time), Environmental scientists (studying climate or pollution trends) and even **business analysts and managers** (monitoring key performance indicators and operational data)

In academic settings, smoothing is also used in research for time series decomposition and forecasting experiments.

Smoothing data enhances the visibility of trends and patterns by eliminating random fluctuations, **facilitating** clearer visualizations and more interpretable charts. It also **prepares** data for modeling, especially for forecasting and anomaly detection, **by** reducing noise-related errors **in** model fitting.

While smoothing is a valuable tool in time series analysis (e.g. for highlighting patterns and improving model performance) it should be carried out with caution. Small but meaningful fluctuations can be unintentionally removed leading to loss of important **information**. Moreover, Certain smoothing methods such as moving averages may introduce a lag effect, delaying the reflection of recent changes. Over-smoothing can also sometimes lead to misrepresentation of the original data. **Additionally, smoothing is not suitable** for all types of data, particularly when abrupt changes are critical (e.g., alarms, sudden events). The choice of smoothing method and its parameters should be guided by the nature of the data and the specific analytical goals.

There are several ways of smoothing data e.g. **Moving Average (MA)**: Simplest method; averages over a fixed window, **Exponential Smoothing**: Gives more weight to recent values; used in

forecasting models, **LOESS/LOWESS**: Flexible local regression method for nonlinear trends. **Spline Smoothing**: Fits smooth curves through data using mathematical splines.

Smoothing algorithm using Savitzsky-Golay filter

This method is a little more sophisticated algorithm as it by fitting a **low-degree polynomial** (typically 2nd or 3rd degree) to a **moving window** of consecutive data points using **least-squares regression**. The central point in each window is replaced by the value predicted from the fitted polynomial.

The algorithm follows the following steps:

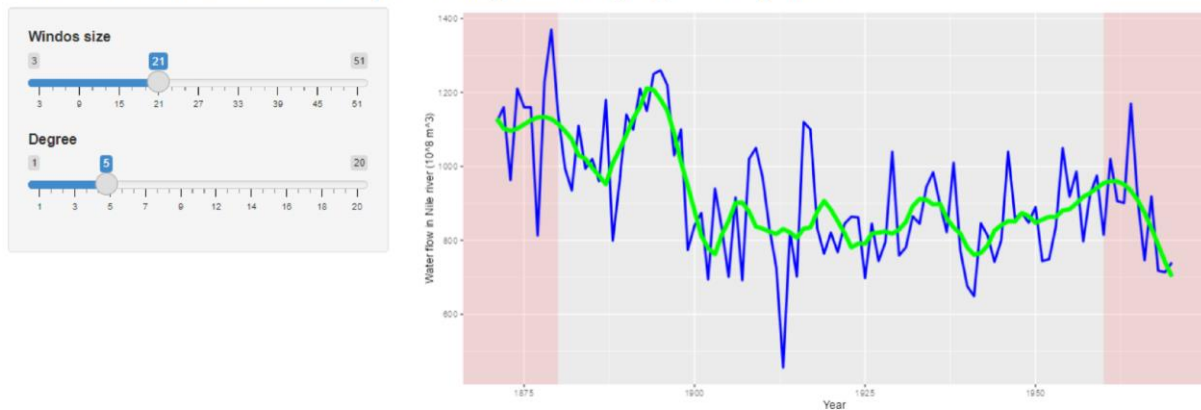
- A sliding window (e.g., 5, 7, or 11 points) moves across the data.
- For each window, a polynomial is fitted to the values using least squares.
- The center value of the window is replaced with the fitted value from the polynomial.
- This process is repeated across all points in the series.

It was originally developed for smoothing **spectroscopic data**, but it is widely used today in many fields involving **time series, and other purposes**. The algorithm has an advantage that **it preserves sharp features** such as peaks and edges. This makes it useful for preprocessing data before differentiation or peak detection. Savitsky- Golay filter outperforms other algorithms like moving averages at retaining the shape of the signal. Requires evenly spaced data points and selection of optimal parameters (window size and polynomial degree).

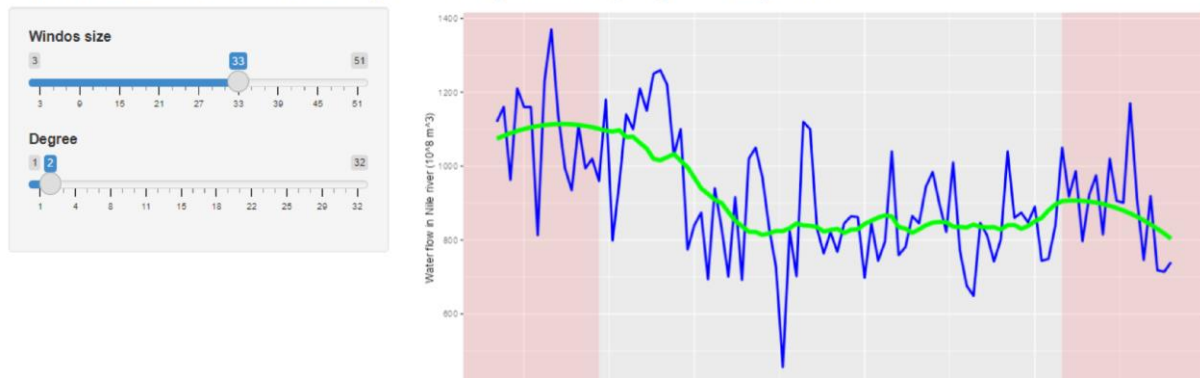
Our Example is made on smoothing the time series data on the Nile water flow between 1875 and 1950. The algorithm allows the user for making changes in degree of smoothing and demonstrating the importance of smoothing scales.

Figure 1. Demonstrating the importance of degree and scale in the cases of Nile water flow time series.

Smoothed time series analysis using Savitsky'ego-Golay filter



Smoothed time series analysis using Savitsky'ego-Golay filter



If you would like to use the algorithm yourself please copy “script 1” into your R environment, sign it and run the algorithm. Following this, you can change the window sizes and degree of smoothing.

Exploratory data analysis

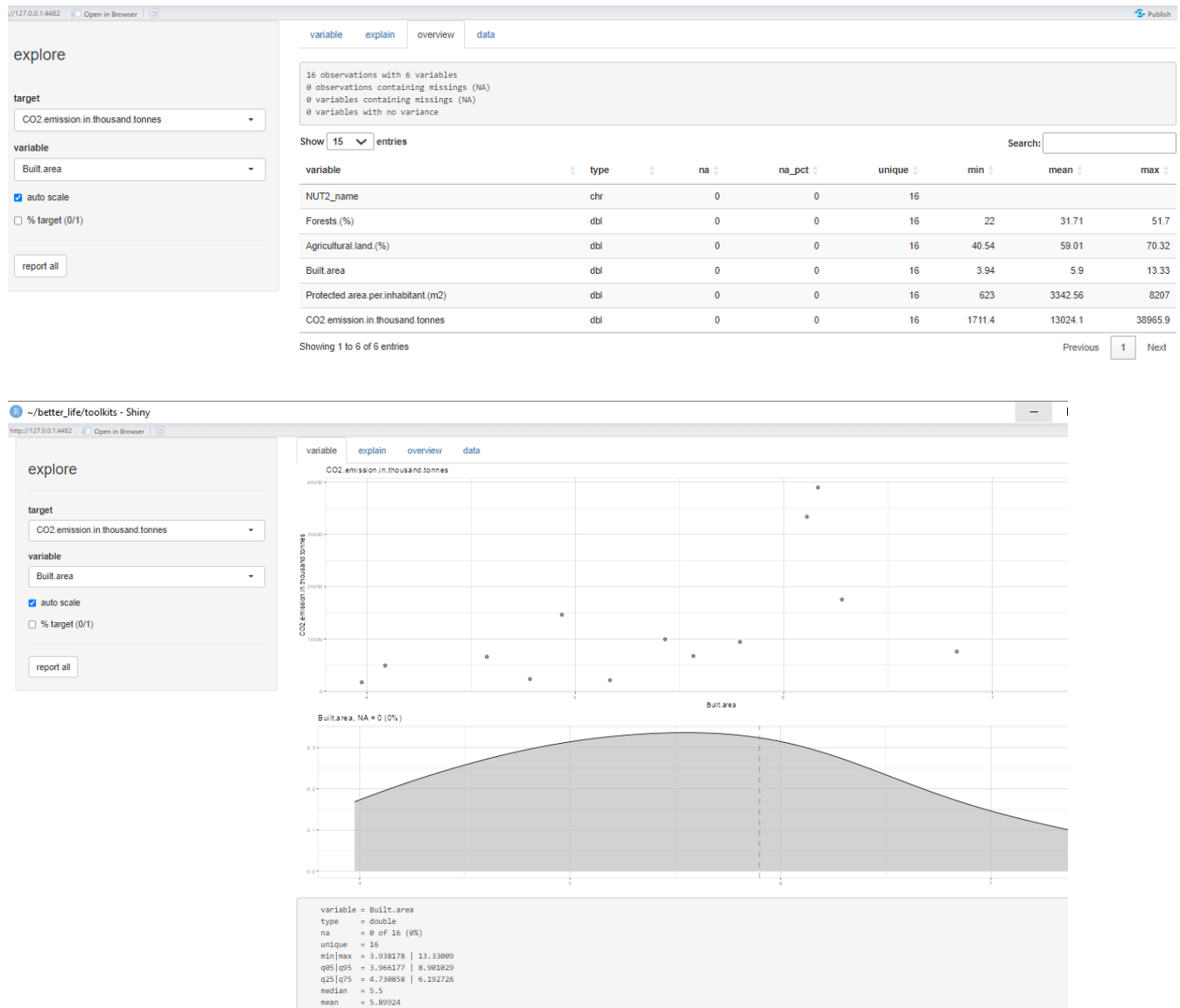
Exploratory Data Analysis (EDA) is the process of examining a dataset in detail to summarize its main characteristics, often through a combination of visualizations, summary statistics, and data transformations. It is a crucial early step in the data analysis workflow, serving as the foundation for understanding what the data can tell us before moving on to more formal modelling or hypothesis testing.

The goal of EDA is to understand the structure of the dataset, detect underlying patterns or trends, identify outliers or anomalies, and spot data quality issues such as missing or inconsistent values. By exploring the data in this way, analysts can make more informed decisions about which statistical models to apply or whether data cleaning and transformation steps are needed.

EDA can involve both numerical summaries—like means, medians, ranges, and standard deviations—and graphical methods, such as histograms, boxplots, scatterplots, or correlation matrices. These tools help make sense of the data's distribution, relationships between variables, and group-level differences allowing for rapid assessment before more detailed analysis. Also it can be useful for demonstrating material during work with stakeholders, or summarising material obtained from workshops.

The provided script allows you to choose your file in xcl, read it into the R environment and following this, enables users to select the variables, look at the tables and explore the dependencies. As an example we loaded the Polish regional landcover data, and CO2 emissions. CO2 emissions were chosen as target variable and we can explore eventual dependencies on elements of landcover data.

Figure 2. Examples of explanatory data analysis using “explore” packages



Creating a Word Cloud

A word cloud (also known as a *tag cloud*) is a visual representation of textual data where the size (and sometimes color or weight) of each word reflects its **frequency or importance** within a given body of text. More frequently occurring words are displayed in larger font sizes, while less common words appear smaller. Word clouds are often used to quickly convey the most prominent terms or themes within large text datasets such as articles, survey responses, speeches, social media content, or research reports. Typically, word clouds ignore common stop words like “the,” “and,” or “is,” focusing on more meaningful terms.

Word clouds are a powerful tool for engaging stakeholders in life sciences by making complex textual information more accessible and easier to interpret. They help quickly highlight key themes from scientific reports, survey responses, meeting transcripts, or public consultations, allowing stakeholders to visually grasp the most relevant or frequently mentioned terms. This kind of visualization supports transparency and shared understanding, especially when working with diverse groups such as researchers, clinicians, regulatory bodies, and patient advocacy organizations. Word clouds can encourage more inclusive discussions by providing a neutral, data-driven summary of opinions or feedback.

Additionally, Word Clouds are also useful during early phases of project planning or evaluation when collecting qualitative input from stakeholders. By summarizing this input visually, word clouds can help guide decision-making, identify shared priorities, and ensure that stakeholder voices are represented clearly and fairly. Word clouds might help in early-stage exploratory analysis when trying to get a general overview of text content.

Word clouds are a powerful and accessible tool for visually exploring and presenting textual data. While they don't replace deeper text analysis methods, they serve as a compelling first step for summarizing and communicating key insights quickly and effectively.

In conclusion, the R Shiny tool offers a valuable opportunity for early career researchers (ECRs) to conduct data analysis and visualization in a relatively simple and accessible way. While many ECRs are already familiar with statistical concepts and programming, this is not always the case for other key audiences, such as stakeholders, practitioners, and members of the general public. Among these groups, there is often a wide range of digital literacy and comfort with complex software tools.

R Shiny bridges this gap by enabling researchers to build interactive, web-based (in this case ECR computer based) applications that do not require users to write any code or have specialized technical knowledge. Data can be presented in user-friendly formats or simply pasted into a text field, allowing for immediate and intuitive visualization. This lowers the barrier to entry and makes it easier for broader audiences to explore, understand, and engage with research findings.

Moreover, this tool enhances transparency and encourages collaboration by making the research process more open and accessible. In contexts involving stakeholder participation—such as environmental monitoring, public health, or social sciences—interactive dashboards and visual tools can significantly improve communication and mutual understanding. They allow non-experts to see patterns, identify issues, and contribute meaningfully to decision-making.

Ultimately, R Shiny can support socially engaged research by helping ECRs design platforms that promote dialogue, foster inclusivity, and respond to real-world challenges. It empowers researchers not only to analyze data but also to share it in a way that invites participation, builds trust, and strengthens the impact of their work beyond academic circles.

Scripts for your own R shiny experience

Script 1. – Time series analysis with Savitzki-Golay filter

Please copy the following script into your R environment, select it all and press “Run App” in the top right part of the screen

```
library(shiny)

library(cowplot)

library(ggplot2)

library(dplyr)

library(rlist)

library(signal)


# Define UI for app
ui <- fluidPage(

  # App title ----

  titlePanel("Smoothed time series analysis using Savitsky'ego-Golay filter "),

  # Sidebar layout with input and output definitions ----

  sidebarLayout(

    # Sidebar panel for inputs ----

    sidebarPanel(

      # Input: Slider for the number of bins ----

      sliderInput(inputId = "window_size",

        label = "Windos size",

        min = 3,

        max = 51,

        step=2,

        value = 5),

      sliderInput(inputId = "degree",

        label = "Degree",

        min = 1,

        max = 10,

        step=1,
```

```

      value = 2)
    ),
    # Main panel for displaying outputs ----
    mainPanel(

      plotOutput(outputId = "distPlot")
    )
  )
)

# Define server logic
server <- function(input, output) {
  observe({
    updateSliderInput(inputId="degree",max=input$window_size-1)
  })

  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot
  output$distPlot <- renderPlot({

df=data.frame(year=as.numeric(time(Nile)),flow=as.numeric(Nile),smooth=sgolayfilt(Nile,n=input$window_size,p=input$degree,m=0,ts=1)
)

xborder1=df[(input$window_size-1)/2,'year']
xborder2=df[length(Nile)-(input$window_size-1)/2,'year']
ggplot()+
  annotate("rect",xmin=-Inf,xmax=time(Nile)[(input$window_size-1)/2],ymin=-Inf,ymax=Inf,fill="red",alpha=0.1)+
  annotate("rect",xmin=xborder2,xmax=Inf,ymin=-Inf,ymax=Inf,fill="red",alpha=0.1)+
  xlab("Year")+ylab("Water flow in Nile river (10^8 m^3)")+
  geom_line(data=df, aes(x=year,y=flow),color="Blue",size=1.1)+geom_line(data=df, aes(x=year,y=smooth),color="green",size=2)
  })
}

# Create Shiny app ----
shinyApp(ui = ui, server = server)

```

Please note, that this script used the average yearly water flow of Nile river. If you would like to try it with your own time series, please use the following code (“data” can be replaced with your filename and Amount column with the name of your time series column in the data. The modification of the code allows you to use this filter on your own data!

```
library(shiny)
library(signal)
library(ggplot2)

# Read the predefined data file
df <- read.csv("data.csv")

# Validate required columns
if (!all(c("Year", "Amount") %in% colnames(df))) {
  stop("The CSV file must contain 'Year' and 'Amount' columns.")
}

# Define UI
ui <- fluidPage(

  titlePanel("Smoothed Time Series Analysis Using Savitzky-Golay Filter"),

  sidebarLayout(
    sidebarPanel(
      sliderInput("window_size", "Window Size",
        min = 3, max = 51, step = 2, value = 5),

      sliderInput("degree", "Polynomial Degree",
        min = 1, max = 10, step = 1, value = 2),

      helpText("Note: Smoothing excludes beginning and end based on window size.")
    ),

    mainPanel(
      plotOutput("distPlot")
    )
  )

# Define server logic
```

```
server <- function(input, output, session) {

  observe({
    updateSliderInput(session, "degree", max = input$window_size - 1)
  })

  output$distPlot <- renderPlot( {
    smoothed <- sgolayfilt(df$Amount, n = input$window_size,
                          p = input$degree, m = 0, ts = 1)

    df$smooth <- smoothed

    xborder1 <- df[(input$window_size - 1) / 2 + 1, "Year"]
    xborder2 <- df[nrow(df) - (input$window_size - 1) / 2, "Year"]

    ggplot() +
      annotate("rect", xmin = -Inf, xmax = xborder1,
              ymin = -Inf, ymax = Inf, fill = "red", alpha = 0.1) +
      annotate("rect", xmin = xborder2, xmax = Inf,
              ymin = -Inf, ymax = Inf, fill = "red", alpha = 0.1) +
      xlab("Year") +
      ylab("Amount") +
      geom_line(data = df, aes(x = Year, y = Amount),
                color = "blue", size = 1.1) +
      geom_line(data = df, aes(x = Year, y = smooth),
                color = "green", size = 2)
  })
}

# Run the app
shinyApp(ui = ui, server = server)
```

Script 2 – Explanatory Data Analysis using “explore” package

This code allows you to choose, your excel file, read it into R and explore your data. It can be useful in case of rapid analysis!

```
# install and call the required directories

install.packages("explore")
install.packages("openxlsx ")

library(openxlsx)
library (explore)

#add source directory
setwd("~/better_life/toolkits")

#choose your file from the source directory
file_name <- file.choose() #Open file dialog, cannot restrict file types
file_name<-choose.files(filters=c("Excel files", "*.xlsx")) #Open file dialog - works under windows only
df=read.xlsx(file_name) #Read the excel file

#start EDA, choose the target variable and the explanatory variables (e.g. CO2 emissions in tonnes vs. agricultural land cover in the wojewodships)
explore(df)
```

Script 3 – Creating World Clouds for your own text

This script allows you to create your Word Cloud by copying any text into the dialogue box.

```
#Install and Load required libraries

install.packages("shiny","tm", "wordcloud"," RColorBrewer")

library(shiny)

library(tm)

library(wordcloud)

library(RColorBrewer)


# Define UI
ui <- fluidPage(

  titlePanel("Word Cloud Generator"),

  sidebarLayout(

    sidebarPanel(

      textAreaInput("text", "Enter your text here:", rows = 5),

      selectInput("color", "Choose a color scheme:",

        choices = rownames(brewer.pal.info), selected = "Dark2"),

      actionButton("generate", "Generate Word Cloud")

    ),

    mainPanel(

      plotOutput("wordcloudPlot", width = "100%", height = "600px") # Increased height here

    )

  )

)


# Define server
server <- function(input, output) {

  wordcloudPlot <- reactive({

    if (input$generate > 0) {

      text <- Corpus(VectorSource(input$text))

      text <- tm_map(text, content_transformer(tolower))

      text <- tm_map(text, removePunctuation)

      text <- tm_map(text, removeNumbers)

      text <- tm_map(text, removeWords, stopwords("en"))

      text <- tm_map(text, stripWhitespace)

      text <- tm_map(text, removeWords, c("the", "and", "in", "to", "for", "of", "a"))

    }

  })

}
```



```
wordcloud(words = text,
  min.freq = 2,
  scale = c(6, 0.5), # Larger scale for bigger words
  colors = brewer.pal(8, input$color),
  random.order = FALSE)
}
})

output$wordcloudPlot <- renderPlot({
  wordcloudPlot()
})
}

# Run the app
shinyApp(ui, server)
```