Briefings in Bioinformatics, 23(1), 2022, 1–15

https://doi.org/10.1093/bib/bbab415
Review

Development of interactive biological web applications with R/Shiny

Lihua Jia[†], Wen Yao[†], Yingru Jiang, Yang Li, Zhizhan Wang, Haoran Li, Fangfang Huang, Jiaming Li, Tiantian Chen and Huiyong Zhang

Corresponding authors: Wen Yao. College of Life Sciences, Henan Agricultural University, Zhengzhou 450002, China; E-mail: yaowen@henau.edu.cn; Huiyong Zhang. College of Life Sciences, Henan Agricultural University, Zhengzhou 450002, China; E-mail: huiyong.zhang@henau.edu.cn
[†]These authors contributed equally to this work.

Abstract

Development of interactive web applications to deposit, visualize and analyze biological datasets is a major subject of bioinformatics. R is a programming language for data science, which is also one of the most popular languages used in biological data analysis and bioinformatics. However, building interactive web applications was a great challenge for R users before the Shiny package was developed by the RStudio company in 2012. By compiling R code into HTML, CSS and JavaScript code, Shiny has made it incredibly easy to build web applications for the large R community in bioinformatics and for even non-programmers. Over 470 biological web applications have been developed with R/Shiny up to now. To further promote the utilization of R/Shiny, we reviewed the development of biological web applications with R/Shiny, including eminent biological web applications built with R/Shiny, basic steps to build an R/Shiny application, commonly used R packages to build the interface and server of R/Shiny applications, deployment of R/Shiny applications in the cloud and online resources for R/Shiny.

Key words: biological web application development; web server; biological database; data analysis; R; Shiny

Introduction

With the development of high-throughput sequencing and other biotechnologies, a tremendous amount of data was generated in biological studies [1]. Analysis of these data requires professional coding skills, which are not equipped by most of the biological science community. As a result, the development of interactive web applications to deposit and analyze huge datasets has become an important subject of bioinformatics for many years [1, 2]. Many famous web applications were developed and were applied in various fields of biological studies. Basic Local Alignment Search Tool [3] is the most popular program used in biological studies, which had been implemented as a graphical interface in many biological databases. UCSC Genome Browser [4] is a very popular web application for query, visualization and examination of the genome sequences of human and other major model organisms. For each genome, additional annotations, including gene models, genetic variations, the expression profile of messenger ribonucleic acid (RNA), etc., are integrated for browsing and visualizing. Since 1994, an issue of biological databases was launched annually by Nucleic Acids Research

Lihua Jia is a PhD student at the College of Agronomy, Henan Agricultural University.
Wen Yao is a professor at the College of Life Sciences, Henan Agricultural University. His research interests include biological web applications development and population genomic studies of crops.
Yingru Jiang is a master's student at the College of Life Sciences, Henan Agricultural University.
Yang Li is an associate professor at the College of Life Sciences, Henan Agricultural University.
Zhizhan Wang is a master's student at the College of Life Sciences, Henan Agricultural University.
Haoran Li is a master's student at the College of Life Sciences, Henan Agricultural University.
Fangfang Huang is a master's student at the College of Life Sciences, Henan Agricultural University.
Jiaming Li is a master's student at the College of Life Sciences, Henan Agricultural University.
Tiantian Chen is a master's student at the College of Life Sciences, Henan Agricultural University.
Huiyong Zhang is a professor at the College of Life Sciences, Henan Agricultural University. His research focuses on functional studies of plant microRNA and the genomic basis of disease resistance in maize.

Submitted: 9 June 2021; **Received (in revised form):** 9 September 2021

[5]. Later in 2004, Nucleic Acids Research launched an issue of web servers, which has continued up to the present year [6]. Several thousands of biological web applications have been published in Nucleic Acids Research and many other journals, which promoted the progress of various fields of biological studies.

Up to now, the LAMP web application software stack [7], which is an integration of Linux, Apache, MySQL and PHP/Python/Perl/Java, is the most popular framework used to build web applications in biological studies. This framework is a professional system extensively used to build commercial web applications in business and industry. However, large efforts and time are required to develop web applications using this framework, as most biological researchers are not trained in computer sciences. Each component of this framework is an independent system with powerful functionalities, which requires great efforts to learn and master. Configuration of each component of this framework to make them working together coordinately is also a great challenge for many biological researchers. In addition, the knowledge of HyperText Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, etc. are required to develop interactive web applications utilizing this framework.

R [8] is a remarkably efficient computation and visualization system that was created by Ross Ihaka and Robert Gentleman in the early 1990s, which is one of the most popular programming languages used in bioinformatics. In addition, many open-source R packages were developed by the R community to further enhance the functionalities of R. As of April 2021, over 17 000 R packages are deposited in the Comprehensive R Archive Network (CRAN, https://cran.r-project.org/) [9], while ≥1900 R packages are hosted by Bioconductor [10]. Despite its powerful features, R was rarely used to build web applications before the Shiny package [11] was developed by the RStudio company in 2012 (Figure 1). Shiny is an R package, making it incredibly easy to build web applications for the large R community in bioinformatics, as it is not required to be familiar with HTML, CSS or JavaScript to build web applications with R/Shiny. A recent review on the use of R/Shiny in research publications revealed that R/Shiny had promoted collaborations between researchers by allowing experts from different disciplines without professional programming knowledge to develop their own applications [12]. R/Shiny not only can be used to build lightweight web applications for data visualization or data exploration but also can be utilized to construct large databases or web servers with a huge number of user accesses. An interactive web application [13] for real-time visualization of the COVID-19 epidemic in Japan was built with R/Shiny, getting >13 million page views in 2020.

Here, we reviewed the utilization of R/Shiny to build interactive web applications in biological studies. The basic steps required to build a web application with R/Shiny were elaborated. Commonly used R packages for the design of the graphical interface at the client side and data analysis at the server side in R/Shiny development were introduced. We also compared different approaches to deploy an R/Shiny application in the cloud. Online resources for learning and mastering R/Shiny were also summarized. Finally, the pros and cons of R/Shiny were discussed.

Results

Survey of the web techniques used to build biological web applications in recent studies

To get a full picture of the web techniques utilized to build biological web applications in recent years, we surveyed the latest issues of web servers and databases published in Nucleic Acids Research [5, 6]. The web techniques utilized to build 175 out of a total of 79 web servers and 189 databases were well documented in the published papers (Supplementary Table S1 available online at http://bib.oxfordjournals.org/). It was found that MySQL was the most frequently used database tool in the building of biological web applications. Apache was the most popular web server used to deliver web content. Python and Java were the most common programming languages that were used to build the user interface (UI) and to program the server side to generate dynamic contents and to respond to user requests (Supplementary Table S1 available online at http://bib.oxfordjournals.org/). Besides, HTML, CSS and JavaScript were frequently used together with other techniques in the building of UIs and the communication with the server side. Dozens of other programming languages, including PHP, Perl, Julia, JSP, Ruby, Shell, JavaScript library, HTML5, Go, R/Shiny, etc., were also utilized in the building of biological web applications. Most of these techniques are the core techniques utilized to build web applications in industry and business. Among all the programming languages used to construct biological web applications, R, Python and Perl are the most popular languages used in biological data analysis and bioinformatics. Compared with Python, R was utilized in the building of fewer biological web applications. On the contrary, although Java is a very popular language in industry and business, its popularity in biological data analysis is far behind that of R. Other languages, including PHP, JSP, Ruby, JavaScript library, etc., are rarely used in biological data analysis. These results indicate that building interactive web applications is still a great challenge for most biological researchers and R users.

A full list of biological web applications built with R/Shiny

Ever since R/Shiny [11] was released by RStudio in 2012, a growing number of researchers have used it to build biological web servers, databases or data visualization applications (Figures 1 and 2 and Table 1). By searching Google Scholar and PubMed with the keyword 'R Shiny' and by manually checking the abstract and skimming the full text of each publication, we obtained a full list of 470 biological web applications developed with R/Shiny (Supplementary Table S2 available online at http://bib.oxfordjournals.org/).

R [8] is renowned for its powerful statistics and data visualization ecosystems. Different plotting systems are implemented in R, including the base plotting system, the lattice system [14] and the ggplot2 system [15]. The ggplot2 system [15] is one of the most popular visualization systems used by the science community, which was developed by Hadley Wickham in R based on the grammar of graphics [16]. Although ggplot2 has made it very easy to visualize scientific data, it is still required to be familiar with the grammars and features of R to make fancy scientific plots, which is still a great challenge for non-R users. With the development of R/Shiny, more and more interactive data visualization applications were developed, which enabled the relish of the powerful visualization ability of R by many more audiences (Table 1 and Supplementary Table S2 available online at http://bib.oxfordjournals.org/). To encourage the use of box plot instead of bar plot when visualizing sampled data, an R/Shiny application BoxplotR [17] was developed and published in Nature Methods (Figures 1 and 2). Inspired by BoxplotR, more versatile R/Shiny applications, including PlotsOfData [18] and PlotTwist [19], were developed utilizing the ggplot2 system. PlotsOfData was designed to visualize raw data points as dot plot and their summary statistics as box plots or violin plots, while

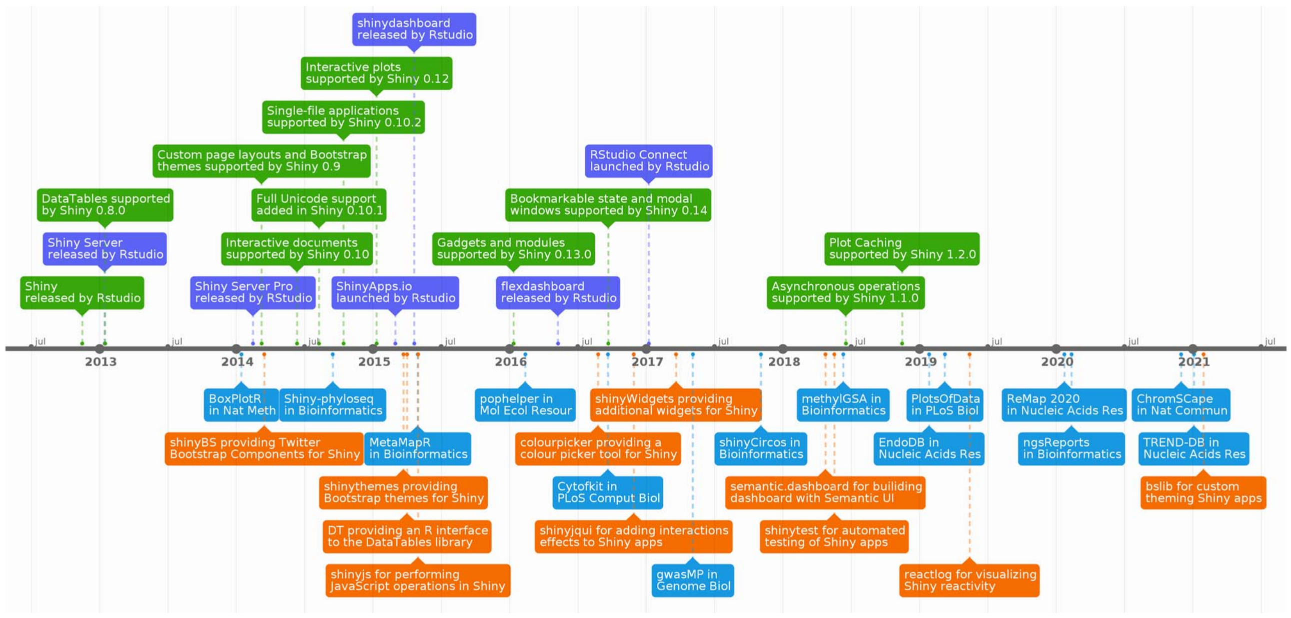

shinydashboard released by Rstudio
Interactive plots supported by Shiny 0.12
Single-file applications supported by Shiny 0.10.2
Custom page layouts and Bootstrap themes supported by Shiny 0.9
RStudio Connect launched by Rstudio
DataTables supported by Shiny 0.8.0
Full Unicode support added in Shiny 0.10.1
Bookmarkable state and modal windows supported by Shiny 0.14
Shiny Server released by Rstudio
Interactive documents supported by Shiny 0.10
Gadgets and modules supported by Shiny 0.13.0
Plot Caching supported by Shiny 1.2.0
Shiny released by Rstudio
Shiny Server Pro released by RStudio
ShinyApps.io launched by Rstudio
flexdashboard released by Rstudio
Asynchronous operations supported by Shiny 1.1.0
2013
2014
2015
2016
2017
2018
2019
2020
2021
jul
BoxPlotR in Nat Meth
Shiny-phyloseq in Bioinformatics
pophelper in Mol Ecol Resour
shinyWidgets providing additional widgets for Shiny
methylGSA in Bioinformatics
PlotsOfData in PLoS Biol
ReMap 2020 in Nucleic Acids Res
ChromSCape in Nat Commun
shinyBS providing Twitter Bootstrap Components for Shiny
MetaMapR in Bioinformatics
colourpicker providing a colour picker tool for Shiny
shinyCircos in Bioinformatics
EndoDB in Nucleic Acids Res
ngsReports in Bioinformatics
TREND-DB in Nucleic Acids Res
shinythemes providing Bootstrap themes for Shiny
Cytofkit in PLoS Comput Biol
semantic.dashboard for builiding dashboard with Semantic UI
bslib for custom theming Shiny apps
DT providing an R interface to the DataTables library
shinyjqui for adding interactions effects to Shiny apps
shinytest for automated testing of Shiny apps
shinyjs for performing JavaScript operations in Shiny
gwasMP in Genome Biol
reactlog for visualizing Shiny reactivity

Figure 1. A timeline view of the development history of Shiny and biological web applications built with R/Shiny. The development history and feature updating of Shiny are indicated in dark green. Supporting tools for R/Shiny application development released by RStudio are in soft blue. Commonly used R packages developed by RStudio or the R community to enhance the capability of Shiny are shown in vivid orange. Representative biological web applications built with R/Shiny are indicated in vivid blue.

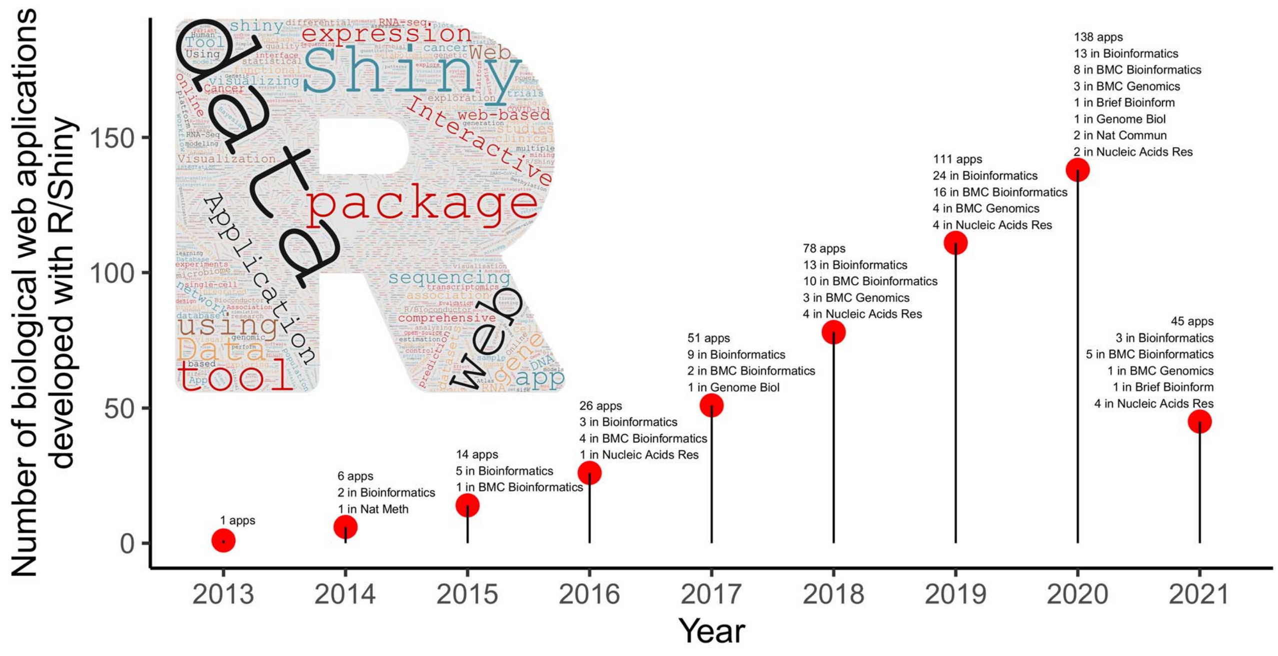


Figure 2. An overview of biological web applications developed with R/Shiny in different years. For each year, the total number of R/Shiny applications developed and the number of R/Shiny applications published in major bioinformatics journals are indicated. In the upper left is a word cloud created using the titles of all publications of the 470 R/Shiny applications. Note: For the year 2021, data up to May 18 are included.

PlotTwist was designed to plot and annotate continuous data. Circos [20] was originally developed using Perl to visualize genomic data in a circular layout, which has become a mandatory plot to demonstrate newly sequenced genomes. Tens of programs, including several R packages, were developed to create Circos plot, aiming to resolve the complex configurations of the Perl-based Circos program [21, 22]. By wrapping the circlize R package [23] with graphical interfaces, an R/Shiny application shinyCircos [24] was developed for creating Circos plot, requiring no coding efforts from the users (Table 1). Similarly, shinyChromosome [25] is an R/Shiny application for non-circular visualization of genomic data by aligning genomic data along all chromosomes of a single genome. trackViewer [26] is a Bioconductor package used for the visualization of multi-omics data, including data of coverage, annotation, methylation and mutation, along a specified genomic region with various

Table 1. Representative biological web applications built with R/Shiny

Name	Title	Year	Category	URL
BoxPlotR	BoxPlotR: a web tool for generation of box plots	2014	Visualization	http://shiny.chemgrid.org/boxplotr/
Cerebro	Cerebro: interactive visualization of scRNA-seq data	2019	Visualization	https://github.com/romanhaa/cerebroApp
PlotsOfData	PlotsOfData—a web app for visualizing data together with their summaries	2019	Visualization	https://huygens.science.uva.nl/PlotsOfData/
PlotTwist	PlotTwist: a web app for plotting and annotating continuous data	2020	Visualization	https://huygens.science.uva.nl/PlotTwist/
shinyCircos	shinyCircos: an R/Shiny application for interactive creation of Circos plot	2018	Visualization	https://venyao.xyz/shinycircos/
shinyChromosome	shinyChromosome: an R/Shiny application for interactive creation of non-circular plots of whole genomes	2019	Visualization	https://venyao.xyz/shinyChromosome/
ECOGEMS	ECOGEMS: efficient compression and retrieve of SNP data of 2058 rice accessions with integer sparse matrices	2019	Database	https://venyao.xyz/ECOGEMS/
EndoDB	EndoDB: a database of endothelial cell transcriptomics data	2018	Database	https://endotheliomics.shinyapps.io/endodb/
IR-TEx	IR-TEx: an open source data integration tool for big data transcriptomics designed for the malaria vector *Anopheles gambiae*	2018	Database	https://www.lstmed.ac.uk/projects/ir-tex
PCAT	PCAT: an integrated portal for genomic and preclinical testing data of pediatric cancer patient-derived xenograft models	2021	Database	http://pcat.zhenglab.info
TREND-DB	TREND-DB—a transcriptome-wide atlas of the dynamic landscape of alternative polyadenylation	2021	Database	http://shiny.imbei.uni-mainz.de:3838/trend-db/
BRepertoire	BRepertoire: a user-friendly web server for analyzing antibody repertoire data	2018	Web server	http://mabra.biomed.kcl.ac.uk/BRepertoire
CELLector	CELLector: genomics-guided selection of cancer *in vitro* models	2020	Web server	https://ot-cellector.shinyapps.io/CELLector_App/
ChromSCape	Interactive analysis of single-cell epigenomic landscapes with ChromSCape	2020	Web server	https://vallotlab.shinyapps.io/ChromSCape/
GAM	GAM: a web service for integrated transcriptional and metabolic network analysis	2016	Web server	https://artyomovlab.wustl.edu/shiny/gam/
LOLAweb	LOLAweb: a containerized web server for interactive genomic locus overlap enrichment analysis	2018	Web server	http://lolaweb.databio.org/
sleuth	Differential analysis of RNA-seq incorporating quantification uncertainty	2017	Web server	http://pachterlab.github.io/sleuth/

types of plots. An interactive application of trackViewer was developed using R/Shiny, which further increased the appeals of trackViewer to a broader user community. No coding efforts are required from the users to use these web applications.

Besides the powerful statistics and visualization system, another merit of R is the freely available of tens of thousands of packages for data analysis in almost every field of biological studies. This merit is further enhanced with the application of the Shiny package in the building of interactive web servers (Table 1) [11]. ChromSCape [27] is a user-friendly R/Shiny application for the interactive analysis of single-cell epigenomic data (Figures 1 and 2). Various R packages were utilized by ChromSCape, including SingleCellExperiment [28], scater [29] and scran [30] for the manipulation of single-cell data; GenomicRanges [31] for the manipulation of genomic regions; Rsamtools [32] and BiocParallel [33] for the manipulation of genomic files; ggplot2 [15], colourpicker [34] and gplots [35] for data visualization; etc. LOLAweb [36] is an interactive web server built with R/Shiny for the enrichment analysis of genomic locus overlaps (Table 1). The core functionalities of LOLAweb rely on the R package Locus Overlap Analysis (LOLA) [37]. GAM [38] is a web server built with R/Shiny for the integrative network analysis of transcriptomic and metagenomic data, which can be used to identify the most regulated metabolic subnetworks by comparing data of different conditions or cell states (Table 1).

R/Shiny can also be used to build biological databases (Table 1). MySQL is an open-source database management system widely used in the management of large datasets. The advantages of MySQL can be utilized by R with the help of the RMySQL [39] and RMariaDB [40] packages, which function as the interface to MySQL from R. PCAT [41] is a comprehensive database built with R/Shiny and MySQL, providing hundreds of patient-derived xenograft models and their associated clinical and genomic data. For small- to medium-scale datasets, we can use the data structure implemented in R instead of MySQL. Utilizing the sparse matrix data structure implemented in R, the genotype data of 2058 rice accessions across over 8 million SNP sites were compressed and stored in the ECOGEMS database [42]. ECOGEMS was built using R/Shiny, taking advantage of several R packages for data analysis, including APE [43], pegas [44], LDheatmap [45], ggtree [46], etc. IR-TEx [47] is a database built with R/Shiny for exploration and comparison of transcriptomic data of insecticide-resistant and -susceptible *Anopheles gambiae* populations from different geographical regions of Africa. Users can identify the expression pattern of resistance candidates and explore the functions or pathways of novel resistance-associated genes with IR-TEx. TREND-DB [48] is another database built with R/Shiny, cataloging the dynamic landscape of alternative polyadenylation for >3600 human genes.

The basic structure of an R/Shiny application

To develop an R/Shiny application, the R environment is indispensable, which can be downloaded from CRAN and installed on Windows, macOS and a wide variety of UNIX platforms [8]. Although it is not mandatory, it is recommended to develop R/Shiny applications with RStudio [49], which is one of the best Integrated Development Environment for R code development. Installation of the Shiny package [11] is also a prerequisite for R/Shiny development.

The development of an R/Shiny application is similar to the development of interactive web applications using other techniques, which also includes the design of a graphical UI for the client side and the process of user requests received from the client side (Figure 3**A**). A typical R/Shiny application is composed of two R scripts, ui.R and server.R, which should be placed in the root directory of the R/Shiny application (Figure 3**B**). ui.R is used to define the graphical interfaces of an R/Shiny application. The appearance and layout of an R/Shiny application are configured in ui.R (Figure 3**C**). In addition, various input widgets can be defined in ui.R, which can be used to receive user inputs, including file uploading, text pasting, parameter settings, etc. User inputs collected by ui.R would then be conveyed to the server side and processed by server.R (Figure 3**D**). The calculation outputs by server.R are then displayed as figures, tables, etc. at positions specified by ui.R in the UI. The code of ui.R and server.R can be integrated into a single R script for mini R/Shiny applications. Furthermore, a www directory can be created in the root directory of an R/Shiny application to place scripts and files that can be directly referenced in the R/Shiny application.

The development of an R/Shiny application is a dynamic process, which means the application needs to be frequently launched to examine its appearance and functionality during the development process. The runApp() function of the Shiny package [11] can be used to launch an R/Shiny application, with the directory path of the Shiny application as the parameter. An R/Shiny application can be launched and displayed in the viewer panel or a new window of RStudio, or the default web browser of the system. It is recommended to launch an R/Shiny application in the web browser. The update in the appearance or functionality of an R/Shiny application, resulting from code update, can be observed instantly by refreshing the web browser.

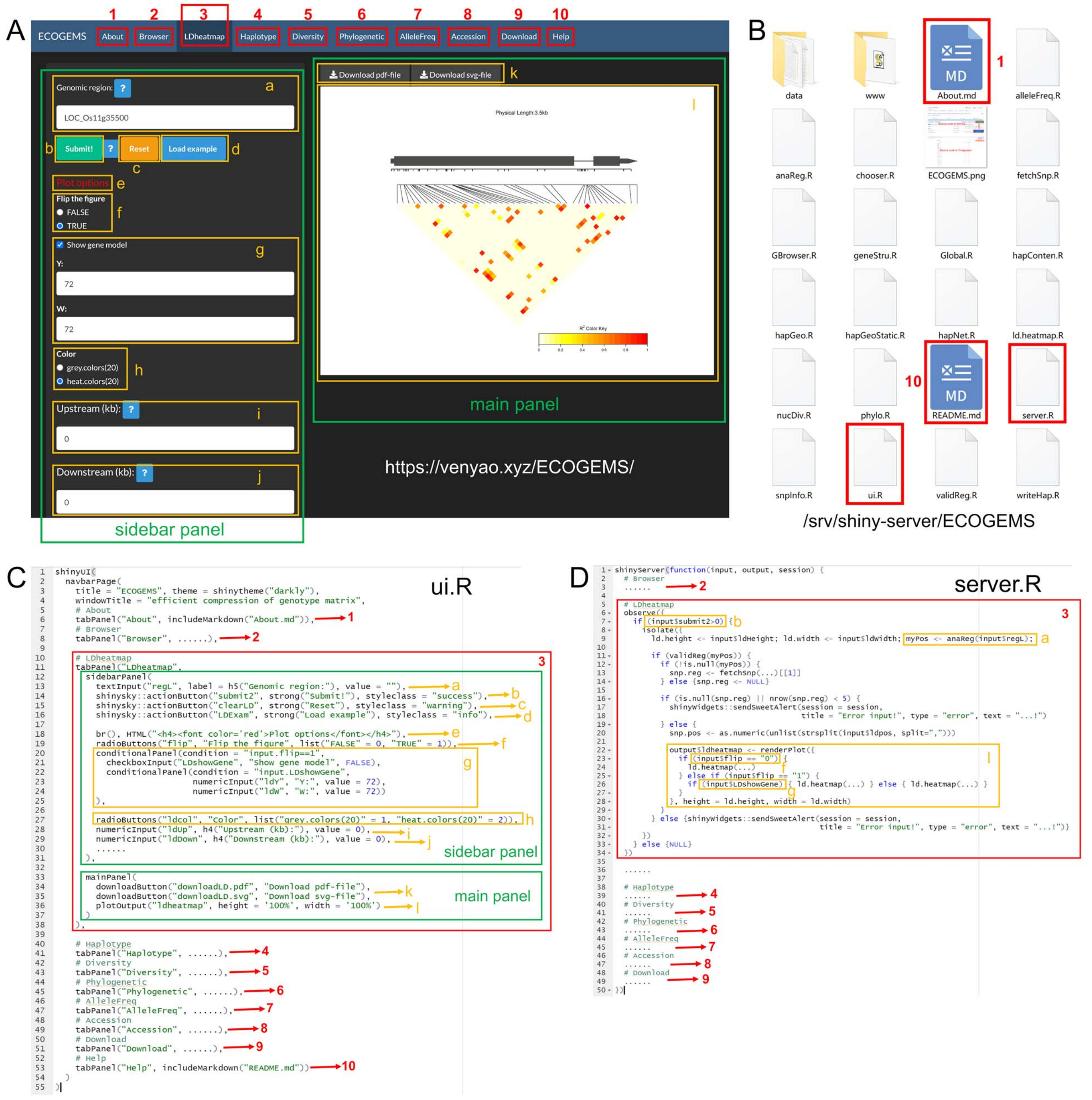
Design the graphical interface of an R/Shiny application with ui.R

The first step in the development of an R/Shiny application is to design the UI with ui.R. The *Page() functions of the Shiny package can be used to initialize the page layout of an R/Shiny application [11]. The navbarPage() function can be used to create a multiple-page layout, including a navigation bar, while the fluidPage() function can be used to create a single page layout that automatically adjusts to the dimensions of the user's browser window (Figure 3). The interface of the R/Shiny application can then be laid out by placing UI elements inside the page layout. Usually, a single page would be further split into multiple panels by placing *Panel() functions inside the *Page() functions. Functions, including headerPanel(), titlePanel(), sidebarPanel() and mainPanel(), are used to combine a set of UI elements into a unified panel. The header panel and title panel are used to define the title region of an R/Shiny application, while the sidebar panel is used to place various widgets to collect user inputs from files, keyboard or the mouse (Figure 3). The main panel is the main area which is used to place the outputs rendered by server.R in the form of images, tables, texts, etc. (Figure 3). By default, the sidebar panel is placed on the left side of the main panel, which can be moved to the right side by setting the position parameter. Another frequently used panel is a conditional panel, which is used to hide a set of UI elements in the UI when the required condition is not satisfied. The condition is usually defined by the status of one or multiple input widgets.

In any panel, we can add UI elements by placing the corresponding functions inside the *Panel() functions in ui.R (Figure 3). Many pre-built widgets are provided as R functions by the Shiny package. A widget is a UI element that the users can interact with, providing a way for the users to send inputs to the R/Shiny application. For instance, the actionButton() function can be used to create an action button, the fileInput() function can be used to create a file upload control wizard, etc. For the R function of each widget, three mandatory arguments are required to define the identifier, the label and the initial value of the widget. The identifier of the widget is used to access the widget on the server side, while the label is used to display a text beside the widget in the UI. A widget can also have other optional arguments, which are used to set the ranges or increments for the corresponding widgets, etc. Except for the pre-built widgets, UI elements can also be added using HTML tag functions of the Shiny package [11]. For example, the h1() function can be used to create a top-level header, the strong() function can be used to emphasize a text in bold, etc. All these functions are simply HTML tag wrappers with identical names, which makes it quite easy to design UI elements in R/Shiny without the knowledge of HTML. A full list of HTML tag functions is embedded as elements of the tags list defined in the Shiny package, which can be used to recreate a total of 110 HTML tags [50].

Data analysis conducted by server.R based on user inputs collected by ui.R

The user inputs collected by various UI elements are stored in the input variable in the R environment, which can be accessed



```
shinyUI(
  navbarPage(
    title = "ECOGEMS", theme = shinytheme("darkly"),
    windowTitle = "efficient compression of genotype matrix",
    # About
    tabPanel("About", includeMarkdown("About.md")),
    # Browser
    tabPanel("Browser", ......),

    # LDheatmap
    tabPanel("LDheatmap",
      sidebarPanel(
        textInput("regL", label = h5("Genomic region:"), value = ""),
        shinysky::actionButton("submit2", strong("Submit!"), styleclass = "success"),
        shinysky::actionButton("clearLD", strong("Reset"), styleclass = "warning"),
        shinysky::actionButton("LDExam", strong("Load example"), styleclass = "info"),

        br(), HTML("<h4><font color='red'>Plot options</font></h4>"),
        radioButtons("flip", "Flip the figure", list("FALSE" = 0, "TRUE" = 1)),
        conditionalPanel(condition = "input.flip==1",
          checkboxInput("LDshowGene", "Show gene model", FALSE),
          conditionalPanel(condition = "input.LDshowGene",
                           numericInput("ldY", "Y:", value = 72),
                           numericInput("ldW", "W:", value = 72))
        ),

        radioButtons("ldcol", "Color", list("grey.colors(20)" = 1, "heat.colors(20)" = 2)),
        numericInput("ldUp", h4("Upstream (kb):"), value = 0),
        numericInput("ldDown", h4("Downstream (kb):"), value = 0),
        ......
      ),

      mainPanel(
        downloadButton("downloadLD.pdf", "Download pdf-file"),
        downloadButton("downloadLD.svg", "Download svg-file"),
        plotOutput("ldheatmap", height = '100%', width = '100%')
      )
    ),

    # Haplotype
    tabPanel("Haplotype", ......),
    # Diversity
    tabPanel("Diversity", ......),
    # Phylogenetic
    tabPanel("Phylogenetic", ......),
    # AlleleFreq
    tabPanel("AlleleFreq", ......),
    # Accession
    tabPanel("Accession", ......),
    # Download
    tabPanel("Download", ......),
    # Help
    tabPanel("Help", includeMarkdown("README.md"))
  )
)
```

```
shinyServer(function(input, output, session) {
  # Browser
  ......

  # LDheatmap
  observe({
    if (input$submit2>0) {
      isolate({
        ld.height <- input$ldHeight; ld.width <- input$ldwidth; myPos <- anaReg(input$regL);

        if (validReg(myPos)) {
          if (!is.null(myPos)) {
            snp.reg <- fetchSnp(...)[[1]]
          } else {snp.reg <- NULL}

          if (is.null(snp.reg) || nrow(snp.reg) < 5) {
            shinyWidgets::sendSweetAlert(session = session,
                                         title = "Error input!", type = "error", text = "...!")
          } else {
            snp.pos <- as.numeric(unlist(strsplit(input$ldpos, split=",")))

            output$ldheatmap <- renderPlot({
              if (input$flip == "0") {
                ld.heatmap(...)
              } else if (input$flip == "1") {
                if (input$LDshowGene) { ld.heatmap(...) } else { ld.heatmap(...) }
              }
            }, height = ld.height, width = ld.width)
          }
        } else {shinyWidgets::sendSweetAlert(session = session,
                                             title = "Error input!", type = "error", text = "...!")}
      })
    } else {NULL}
  })

  ......

  # Haplotype
  ......
  # Diversity
  ......
  # Phylogenetic
  ......
  # AlleleFreq
  ......
  # Accession
  ......
  # Download
  ......
})
```

Figure 3. Illustration of the basic structure and workflow of an R/Shiny application. (**A**) The graphical interface of the LDheatmap menu of the R/Shiny application ECOGEMS accessible at https://venyao.xyz/ECOGEMS/. The sidebar and main panels of the LDheatmap menu are indicated in green frames. 1–10, the 10 menus of the ECOGEMS application providing different functionalities. a-l, user input and output widgets implemented under the LDheatmap menu. (**B**) List of files under the root directory of ECOGEMS (/srv/shiny-server/ECOGEMS) on a Linux server. About.md and README.md are markdown files that would be compiled as HTML files by R/Shiny and displayed under the About and Help menus of ECOGEMS, respectively. R script files except for server.R and ui.R are implemented with R functions which would be invoked by server.R for specific functionality. (**C**) The content of ui.R used to define the UI of ECOGEMS. 1–10, the 10 tabPanel() functions used to define the ten menus of the ECOGEMS application shown in (**A**). a-l, R functions used to define the user input and output widgets in panel A. (**D**) The content of server.R used to program the server side of ECOGEMS. 2–9, R code used to collect user inputs from ui.R and perform downstream analysis under different menus of ECOGEMS. To save space, lots of R codes are deleted and represented as '......'.

in server.R with the identifiers of the UI elements defined in ui.R (Figure 3C and D). For instance, input$id01 can be used to access the latest value of a UI element with the identifier id01. Based on user inputs and other variables defined in server.R, various operations can then be conducted, including reading in user-uploaded files, manipulating data, creating graphics, building statistical models, altering the parameters of specific R functions, etc. These operations can be conducted using R functions or by using non-R programs installed in the operating system, which can be invoked from inside of R using the system() function of R. The outcome of these operations can then be displayed as tables, plots, texts, images, etc. in the UI by using rendering functions in server.R, including renderTable(), renderPlot(), renderText() and renderImage(), respectively. The positions of UI elements presented by the rendering functions are defined by *Output() functions in ui.R. For instance, the position of a rendered table in the UI is determined by the position of the tableOutput() function in ui.R. The result of a rendering function must be assigned to an element of the output variable in server.R, the name of which is then used as the identifier of the corresponding *Output() function. In this way, the UI elements presented by a rendering function in server.R is bound to an *Output() function in ui.R, which creates placeholders for the UI elements. Each *Output() function in ui.R should have a corresponding rendering function in server.R.

A typical workflow in an R/Shiny application starts from the design of the graphical interface and the configurations of user input widgets in ui.R (Figure 3). The user inputs are then collected and goes through some expressions on the server side, the results of which are then rendered on the client side. Each time the user changes the value of one or multiple input widgets, the expression on the server side will be automatically re-executed and the corresponding output of the expression will be re-rendered based on the updated values of the input widgets. Components of this workflow are designated as reactive inputs, reactive expressions and reactive outputs, respectively. One reactive input can be connected to multiple reactive outputs, and one reactive output can be bound to multiple reactive inputs as well. In addition, the reactive expressions used to perform some calculations in server.R can be stopped, triggered or delayed with the R functions isolate(), observeEvent() and eventReactive(), respectively.

Prominent R packages used in server.R for data visualization and analysis

As a prominent programming language for data science, tens of thousands of R packages are freely available from CRAN [9], Bioconductor [10], rOpenSci [51] and GitHub [52]. These packages can be seamlessly utilized in R/Shiny applications. Based on an investigation of the download stats of R packages deposited in Bioconductor and CRAN, we obtained a list of 109 top downloaded R packages utilized in biological data analysis (Table 2 and Figure 4). ggplot2 is an R package developed by Hadley Wickham based on the Grammar of Graphics, which is one of the most popular tools for data visualization in science and industry [15, 16]. ggplot2 can be very easily extended to make new types of plots, and >80 ggplot2 extensions were developed as independent R packages to promote the visualization ability of R [53]. ggtree [46] is a ggplot2 extension which is widely used to manipulate and visualize phylogenetic trees and other trees with annotation data. ggplot2 is a component of tidyverse [54], which is a set of R packages designed for data science, including R packages to read and write plain text files, R packages to read and write files used by other statistical software, R packages to manipulate strings and dates, R packages to tidy and manipulate data and R packages for modeling and machine learning.

Bioconductor [10] is another repository hosting R packages, specially, for biological data analysis. Many R packages in Bioconductor were extensively used in biological data analysis, including Biostrings, clusterProfiler, enrichplot, GenomicRanges, Rsamtools, edgeR, DESeq2, etc. (Table 2). Biostrings [55] is an R package for manipulating biological strings, such as DNA, RNA and protein sequences. GenomicRanges [31] is an R package used to efficiently represent and manipulate genomic intervals. edgeR [56] and DESeq2 [57] are R packages used to conduct differential expression analysis, with count data calculated based on high-throughput sequencing. clusterProfiler [58] is an R package for enrichment analysis of functional profiles of genes and gene clusters. A variety of plotting methods are implemented in clusterProfiler and enrichplot [59] for visualization of the enrichment analysis output.

rOpenSci [51] is a community-driven project aiming to promote open and reproducible research with open-source tools and shared data implemented as R packages. Hundreds of R packages for analyzing scientific papers, handling taxonomic information, manipulating geospatial data, visualizing data, etc. were collected in rOpenSci. GitHub [52] is an open-source community for software development, with millions of developers. The source code of many R packages hosted by CRAN, Bioconductor and rOpenSci are also deposited in GitHub. In addition, many R packages under development are only available in GitHub.

Customize the overall appearance of an R/Shiny application

The default theme of an R/Shiny application is a bit monotonous. The shinythemes package [60] was developed to modify the overall appearance of an R/Shiny application by setting the theme argument of *Page() function in ui.R (Figures 1 and 3). A total of 16 pre-packaged Bootstrap themes are provided by the shinythemes package. Additional pre-packaged themes are provided by another R package bslib [61] (Figure 1). Compared with shinythemes, any Bootswatch 3 or 4 theme [62] can be applied to an R/Shiny application using the bslib R package. In addition, bslib can be used to customize individual UI elements, including navbars, headings, radio buttons, etc. It is also capable to customize specific features, including background color, foreground color or font of the UI elements of an R/Shiny application, with bslib.

Like the development of web applications using other techniques, HTML and CSS code can be utilized to further customize the graphical interface of an R/Shiny application. Several functions, including includeCSS(), includeScript(), includeHTML(), includeMarkdown() and includeText(), are provided by the R Shiny package to insert the contents of CSS files, JS files, HTML files, Markdown files and text files into ui.R of an R/Shiny application [11]. Furthermore, images can be placed under the www subdirectory of an R/Shiny application, which can be inserted into the graphical interface using the img() function.

Developing dashboards with R/Shiny

A dashboard is a graphical UI providing interactive approaches for tracking, analyzing and visualizing metrics and key data points relevant to a particular objective, which have become the norm for interacting with data. Dashboards have been extensively utilized in business, industry and science, supporting strategic decision-making and data analysis. During the COVID-19 pandemic, many dashboards were developed to track COVID-19 in real time, including famous dashboards built by the World Health Organization [63] and the Johns Hopkins University [64]. Dozens of software is available to develop dashboards, including Google Data Studio [65], Tableau [66], DashThis [67], Geckoboard [68], Cyfe [69], IBM Watson Analytics [70], the R/Shiny framework, etc. Among these tools, R/Shiny is one of the best open-source tools to make dashboards.

Dashboards built with R/Shiny are interactive Shiny applications developed with well-designed dashboard themes provided

Table 2. Representative R packages for biological data visualization and analysis

Package	Function	URL
APE	An R package to perform phylogenetics and evolution analyses	https://cran.r-project.org/web/packages/ape/index.html
Biostrings	An R package to store and manipulate biological strings	https://www.bioconductor.org/packages/release/bioc/html/Biostrings.html
clusterProfiler	Statistical analysis and visualization of functional profiles for genes and gene clusters	https://bioconductor.org/packages/release/bioc/html/clusterProfiler.html
data.table	High-performance data manipulation in R	https://cran.r-project.org/web/packages/data.table/index.html
DESeq2	An R package to perform differential gene expression analysis	https://bioconductor.org/packages/release/bioc/html/DESeq2.html
dplyr	An R package for easy and efficient data manipulation in R	https://github.com/tidyverse/dplyr/
DT	An R package providing interface to the JavaScript library DataTables	https://rstudio.github.io/DT/
edgeR	Empirical Analysis of Digital Gene Expression Data in R	https://www.bioconductor.org/packages/release/bioc/html/edgeR.html
enrichplot	An R package for visualization of functional enrichment result	https://www.bioconductor.org/packages/release/bioc/html/enrichplot.html
foreach	An R package for parallel computing	https://github.com/RevolutionAnalytics/foreach
GenomicRanges	An R package for storing and manipulating genomic intervals	https://bioconductor.org/packages/release/bioc/html/GenomicRanges.html
ggmap	An R package for plotting maps	https://github.com/dkahle/ggmap
ggplot2	An R package for creating graphics in R	https://cran.r-project.org/web/packages/ggplot2/index.html
ggtree	An R package for visualization of tree and annotation data	https://www.bioconductor.org/packages/release/bioc/html/ggtree.html
LDheatmap	Display of pairwise linkage disequilibria between SNPs with heatmap	https://cran.r-project.org/web/packages/LDheatmap/index.html
lubridate	An R package for working with dates and times	https://github.com/tidyverse/lubridate/
plotly	An R package for creating interactive web graphics	https://cran.r-project.org/web/packages/plotly/index.html
stringr	An R package for easy and efficient string operations	https://cran.r-project.org/web/packages/stringr/index.html
RColorBrewer	ColorBrewer palettes	https://cran.r-project.org/web/packages/RColorBrewer/
Rcpp	Seamless R and C++ Integration	https://cran.r-project.org/web/packages/Rcpp/
Rsamtools	Binary alignment FASTA, variant call and tabix file import	https://www.bioconductor.org/packages/release/bioc/html/Rsamtools.html
tidyr	An R package to create tidy data	https://cran.r-project.org/web/packages/tidyr/index.html
tidyverse	A set of R packages designed for data science	https://github.com/tidyverse/tidyverse/

by three R packages, including shinydashboard [71], flexdashboard [72] and semantic.dashboard [73] (Figure 1). flexdashboard and shinydashboard were developed by the RStudio company based on the CSS flexbox layout and the Bootstrap grid layout, respectively. semantic.dashboard was developed based on the Semantic UI [74]. flexdashboard is mainly used to publish dashboard-like documents for data visualizations using R Markdown, which can be added with more interactivity by the htmlwidgets [75] and Shiny packages. By providing templates for Shiny, shinydashboard and semantic.dashboard sacrifice flexibility for simplicity and convenience, which have made it much easier to build fancy dashboards with R. On the contrary, the Shiny package can be used to build web applications with any customized layout, offering much more flexibility. The server side of dashboards built with R/Shiny is the same as regular R/Shiny applications.

A typical dashboard is composed of three parts, including a header, a sidebar and a body, which can be built with the R functions dashboardHeader(), dashboardSidebar() and dashboardBody() of the shinydashboard package, respectively. The three functions should be placed inside the dashboardPage() function, which creates a dashboard page. The header and sidebar are optional for a dashboard, which can be disabled. The header is simply used to display the title of a dashboard, which is usually at the top left of the dashboard page. At the top right of the dashboard page, various types of dropdown menus, including message menus, notification menus and task menus, can be implemented. In the sidebar, menu items that behave like tabs can be added by using the menuItem() function for navigation of the dashboard application. A menu item is usually used to place regular Shiny UI input widgets to receive user inputs. In the body of a dashboard, any Shiny UI element can be added. For a dashboard with multiple menu items, contents in the body should be placed in tabItems with different tab names. The content inside a specific tabItem in the body only appears when the menu item with the same tab name is clicked. To

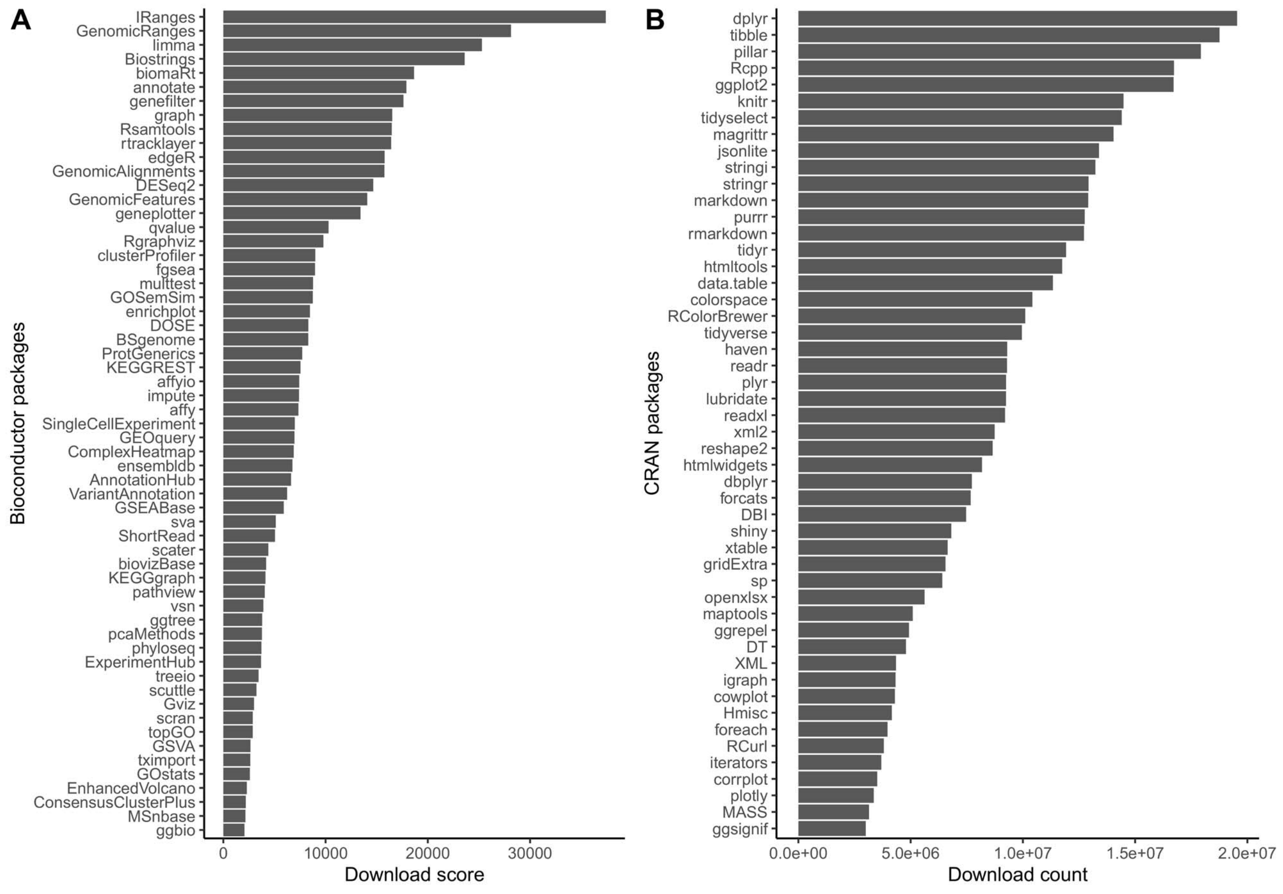


Figure 4. Top downloaded R packages from Bioconductor and CRAN. (**A**) Download scores of 59 Bioconductor packages obtained from https://bioconductor.org/packages/stats/. (**B**) Download counts of 50 CRAN packages calculated based on the download logs of all R packages in the last 3 years (from 31 July 2018 to 31 July 2021), obtained from http://cran-logs.rstudio.com/.

create a more structured dashboard, the contents in the body are usually placed in different boxes, which are the main building blocks of dashboard pages and the central components of shinydashboard. Several types of customized boxes are provided by shinydashboard for rapid building of dashboard, including tabbed box, info box, value box, etc. Finally, the boxes can be laid out using regular Shiny layout functions, including column(), fluidRow(), etc.

R packages to enhance the capability of R/Shiny

The major force underlying the popularity of R is the large number of open-source R packages contributed by the R community, including ones to further enhance the capability of R/Shiny (Figure 1, Table 3 and Supplementary Table S3 available online at http://bib.oxfordjournals.org/). Functions of these packages can be directly integrated into the source code of an R/Shiny web application. Based on their functionalities, these packages are used for the improvement of the graphical interface or for facilitating the communications between the server and the client side.

shinyWidgets [76] is an R package providing various widgets not available from R/Shiny or widgets, with more functional features to enhance the UI of an R/Shiny application (Figure 1 and Table 3). For instance, switch buttons can be used to replace classical checkboxes of R/Shiny, and slider text can be used to replace select input of R/Shiny. shinySky is another R package providing additional widgets, including styled action buttons and event buttons monitoring mouse events triggered by the users (Table 3) [77]. shinyThings [78] is also an R package with several UI widgets, including radio switch buttons, toggle button groups, etc. In addition, several functions of shinyThings can be used to add pagination to any collection of UI widgets. shinyforms [79] is an R package developed to create questionnaire-type forms with R/Shiny. Questionnaires, polls and surveys can be very easily created in an R/Shiny application using shinyforms (Table 3).

Sending informative messages from the server side to the client side based on user input is critical to improving the user experiences of a web application. For example, a pop-up window with indicative messages is very user-friendly when input data of the wrong format were uploaded by the user or when invalid information was entered, which would lead to errors in the downstream analysis and would even interrupt the R/Shiny application. Fortunately, this functionality was implemented in several R packages, including shinySky [77], shinyalert [80] and shinyWidgets [76] (Figure 1 and Table 3). With these packages, user inputs can be checked before being sent to downstream analysis, and wrong input will be intercepted with indicative messages displayed on the screen. By default, a Shiny output is still visible but gets greyed out while it is recalculating, which is not quite elegant. With the help of R packages

Table 3. Commonly used R packages to enhance the capacity of R/Shiny

Package	Function	URL
bslib	Bootstrap themes to customize the appearance of Shiny applications	https://rstudio.github.io/bslib/
cicerone	Step-by-step guided tours for Shiny applications	https://cicerone.john-coene.com/
colourpicker	A color picker tool for Shiny	https://daattali.com/shiny/colourInput/
reactlog	Reactivity visualizer for Shiny	https://rstudio.github.io/reactlog/
shinyalert	Create pretty popup messages in Shiny applications	https://github.com/daattali/shinyalert
shinyBS	Add extra functionality and interactivity to Shiny applications	https://ebailey78.github.io/shinyBS/
shinybusy	Add busy indicator for Shiny applications	https://github.com/dreamRs/shinybusy
shinycssloaders	Add loading animations to a Shiny output while it is recalculating	https://github.com/daattali/shinycssloaders
shinycustomloader	Custom css/html or gif/image file for the loading screen of a Shiny output	https://github.com/emitanaka/shinycustomloader
shinydashboard	Create dashboards with Shiny	https://rstudio.github.io/shinydashboard/
shinydisconnect	Show a nice message when the Shiny application disconnects	https://github.com/daattali/shinydisconnect
shinyFiles	A server-side file system viewer for Shiny	https://github.com/thomasp85/shinyFiles
shinyforms	Easily create questionnaire-type forms with Shiny	https://github.com/daattali/shinyforms
shinyglide	Add carousel-like components to Shiny applications	https://github.com/juba/shinyglide
shinyjs	Perform common useful JavaScript operations in Shiny applications	https://deanattali.com/shinyjs/
shinyjqui	Add interactions and animation effects to Shiny applications	https://github.com/Yang-Tang/shinyjqui
shinymaterial	Implement Google's material design in Shiny applications	https://ericrayanderson.github.io/shinymaterial/
shinySky	Additional widgets for Shiny	https://github.com/AnalytixWare/ShinySky
shinytest	Automated testing of Shiny applications	https://rstudio.github.io/shinytest/
shinythemes	Bootstrap themes for use with Shiny	https://rstudio.github.io/shinythemes/
shinyThings	Additional widgets for Shiny	https://pkg.garrickadenbuie.com/shinyThings/
shinyWidgets	Additional widgets for Shiny	http://shinyapps.dreamrs.fr/shinyWidgets/
shiny.semantic	Semantic UI Support for Shiny	https://appsilon.github.io/shiny.semantic/
thematic	Styling of R plots in Shiny	https://rstudio.github.io/thematic/index.html

shinycssloaders [81] and shinycustomloader [82], a loading animation can be added to a Shiny output to replace the default behavior of R/Shiny (Table 3). The connection to an R/Shiny application disconnects when the user's computer goes to sleep or when the user lost internet connection or for other reasons, which is annoying as no prompt messages were given. The shinydisconnect package [83] can be used to prompt a pretty message to the users when the R/Shiny application disconnects (Table 3).

JavaScript is one of the core web technologies used to program the behavior of web pages. JavaScript code can be written to customize and extend a web application developed by R/Shiny. The quickest way to use JavaScript in R/Shiny is to include a short piece of JavaScript code in ui.R by utilizing the R functions tags$script() and HTML(). However, this approach is not appropriate for long JavaScript code. Another option to use JavaScript in R/Shiny is to create a separate file with multiple lines of JavaScript code and to save it in the www subdirectory of an R/Shiny application. Then, the JavaScript code in the file can be included in ui.R by utilizing the R function tags$script(). Nevertheless, programming JavaScript code is still a great challenge for R users. The shinyjs R package [84] was developed to enable R users to perform common JavaScript operations in an R/Shiny application without any knowledge of JavaScript (Figure 1 and Table 3). shinyjs can be used to hide an UI element, disable an input element, delay the execution of R code, etc. shinyjqui [85] is an R package providing an R wrapper for the jQuery UI JavaScript library [86], which can be used to drag, sort or resize UI elements of an R/Shiny application using the mouse.

Colorful images are indispensable for an R/Shiny application. Setting appropriate colors interactively for plots created in an R/Shiny application is critical to the improvement of user experiences. By default, setting colors in an R/Shiny application is achieved by receiving color names or HEX color codes from the users, which can be frustrating. The colourpicker package [34] can be used to implement a color picker widget in an R/Shiny application (Figure 1 and Table 3). The color to be used for a plot created in an R/Shiny application can be interactively selected and modified by clicking a color palette with the mouse. In addition, the colourpicker package also provides functions and RStudio add-ins for users to select colors from a color palette for any R code inside or outside an R/Shiny application. An alternative color picker widget is implemented in the RLumShiny package (Table 3) [87].

A comprehensive tutorial or prompt messages for different UI elements are important for the improvement of user experiences of an R/Shiny application. R Markdown documents can be used to write tutorials for an R/Shiny application, which can be compiled into markdown or HTML files by the knitr R package (Table 3) [88, 89]. The markdown or HTML files can be directly included in the source code of an R/Shiny application, which would be displayed as structured HTML pages. The shinyBS R package [90] was designed to provide some Twitter Bootstrap components to an R/Shiny application (Figure 1 and Table 3). For example, tooltips and popovers can be added for UI elements to display explaining messages on mouse hover or clicking. The cicerone R package [91] can be used to create step-by-step guided tours to walk the user through an R/Shiny application, which is very helpful for new users. The code used to create the guided tours by cicerone can be separated from the key code of an R/Shiny application, leaving the structure and the layout of the application intact.

Deployment of R/Shiny applications

An R/Shiny application can be deployed and launched locally on personal computers, provided the dependent R packages and other prerequisite software were properly installed. With this approach, an R/Shiny application can be used without internet access. Many R/Shiny applications, including shinyCircos [24], shinyChromosome [25], BoxplotR [17], PlotsOfData [18], etc., can be deployed on personal computers and can be used without internet access. Using the runGitHub() function of the Shiny package, a lightweight R/Shiny application can be automatically downloaded from GitHub and can be deployed on personal computers with a one-line command.

To make an R/Shiny application available to a much larger community, the application needs to be deployed on a publicly accessible web server. For developers who do not own a server, shinyapps.io [92] developed by the RStudio company is a superb choice, which is a platform for hosting R/Shiny applications in the cloud (Figure 1). It is quite easy to deploy and manage an R/Shiny application in shinyapps.io as it is not required to configure a firewall or other system software. To use shinyapps.io, an account at https://www.shinyapps.io/ is required and the rsconnect R package should be installed on the local computer. A token and secret would be automatically generated for the account by shinyapps.io, which were then used by the rsconnect package to upload and deploy R/Shiny applications from local computers to shinyapps.io. Up to five R/Shiny applications can be deployed and 25 h of uptime each month is allocated for a free shinyapps.io account. To deploy more applications with more active hours, a paid account is required. For a paid account, advanced features are provided, including more computing resources, custom domain names, etc. Except for shinyapps.io, RStudio Connect is another service provided by the RStudio company to host R/Shiny applications in the cloud [93]. Compared with shinyapps.io, RStudio Connect supports much more features and is only open to paid users. Most R packages in CRAN, Bioconductor and GitHub are supported by shinyapps.io and RStudio Connect.

However, for R/Shiny applications depending on non-R programs, including commonly used bioinformatics programs BWA [94], SAMtools [95], etc., the Shiny Server program [96] is required to deploy the R/Shiny applications on a self-owned Linux server (Figure 1). Shiny Server is free and open-source, which only runs on 64-bit Linux operating systems. Multiple R/Shiny applications can be deployed behind a firewall on a single Linux server with Shiny Server installed. The data and code of an R/Shiny application should be uploaded to a directory on the Linux server, which is usually the directory/srv/shiny-server created by Shiny Server. Then, Shiny Server should be configured by editing the file/etc/shiny-server/shiny-server.conf, including the setting of the location of the R/Shiny application and the listening port of Shiny. The R/Shiny application would be accessible at an URL address composed of the IP address of the Linux server, the listening port of Shiny and the directory name of the R/Shiny application when the Shiny Server service is activated. Apart from Shiny Server, ShinyProxy [97] is another free, open-source software to deploy R/Shiny applications on a self-owned Linux server, developed based on Java and Docker by OpenAnalytics. For each R/Shiny application, ShinyProxy supports multiple R processes, while Shiny Server only supports a single R process, which is of great value to computationally intensive applications. In addition, ShinyProxy also provides several advanced features, including LDAP authentication and authorization, logging and user statistic tracking. All the advanced features of ShinyProxy are also supported by RStudio Connect for paid users.

Online resources for R/Shiny

Like any other techniques used to develop interactive web applications, many online resources are available for learning and mastering R/Shiny (Table 4). The best place to learn R/Shiny is the official site of Shiny hosted by the RStudio company. Videos and written tutorials are provided to guide the users through the basics of Shiny, including the architecture of an R/Shiny application, rendering functions, reactive programming and the steps to build a UI [98]. For users who are already familiar with the basics of Shiny, many technical articles [99] are provided, covering the advanced usage of R/Shiny, including using relational databases in R/Shiny, using CSS or JavaScript in R/Shiny, performance and scalability improvement of an R/Shiny application, etc. A complete function reference of the Shiny package is available at https://shiny.rstudio.com/reference/shiny/. Furthermore, many elegant open-source R/Shiny applications developed by the RStudio company and the R community are hosted in the Shiny Gallery to demonstrate the basic and advanced features of R/Shiny.

Except for the official site of R/Shiny, many other online resources for the development of R/Shiny applications are available. The book, Mastering Shiny, written by Hadley Wickham, is an excellent resource for R/Shiny developers, which is freely available online (Table 4) [100]. Engineering Production-Grade Shiny Apps by Colin Fay *et al.* is another free online book on the development of production-grade R/Shiny applications (Table 4). Dean Attali is an R/Shiny expert who has developed several R packages, including colourpicker [34], shinyjs [84], etc., to extend the features of R/Shiny (Table 3). His personal website, with many blogs on R/Shiny, is also a great place for R/Shiny developers (Table 4). Four online courses on the development of R/Shiny skills are available on the interactive learning platform DataCamp [101], covering the basic and advanced contents of web application development with R/Shiny. Especially, two courses are devoted to the learning of dashboards building with the shinydashboard and flexdashboard packages, respectively.

To further promote the development of interactive web applications with R/Shiny, several forums, including RStudio Community and Google groups, are operated by the RStudio company for users to ask questions and to seek answers on R/Shiny development (Table 4). Developers are also recommended to post questions and search for answers at Stack Overflow, which is the largest online community for programmers.

Table 4. Online resources for R/Shiny

URL	Description
https://shiny.rstudio.com/	Official website and resources for R/Shiny
https://mastering-shiny.org/	The Mastering Shiny book by Hadley Wickham
https://deanattali.com/	Blogs of Dean Attali
https://engineering-shiny.org/	The Engineering production-grade Shiny apps book by Colin Fay *et al.*
https://laderast.github.io/gradual_shiny/	A gradual introduction to Shiny
https://mastering-shiny-solutions.org/	Solutions to the exercises from the Mastering Shiny book by Hadley Wickham
https://business-science.github.io/shiny-production-with-aws-book/	A book for deploying applications in the Cloud with Shiny and AWS
https://kellobri.github.io/shiny-prod-book/	Supplement to the Shiny in Production Workshop delivered at rstudio::conf 2019
https://unleash-shiny.rinterface.com/	Outstanding UIs with Shiny
https://javascript-for-r.com/	JavaScript for R
https://plotly-r.com/index.html	Interactive web-based data visualization with R, plotly and shiny
https://bookdown.org/msharkey3434/ShinyDB_Book/	Building web applications with Shiny and SQL server
https://bookdown.org/hadrien/how_to_build_a_shiny_app_from_scratch/	How to Build a Shiny Application from Scratch
https://bookdown.org/weicheng/shinyTutorial/	A simple tutorial on R/Shiny
https://community.rstudio.com/tag/shiny	RStudio Community
https://github.com/grabear/awesome-rshiny	A curated list of resources for R/Shiny
https://github.com/nanxstats/awesome-shiny-extensions	A curated list of R packages used to extend UI or server components of Shiny
https://groups.google.com/g/shiny-discuss	Google group for R/Shiny
https://stackoverflow.com/questions/tagged/shiny	Stack overflow questions on R/Shiny
https://www.reddit.com/r/rshiny/	R/Shiny on Reddit
http://asas.yingjiehu.com/	A curated list of R/Shiny applications for statistics
http://shinyapps.org/	A list of R/Shiny applications for experience statistics

The pros and cons of web application development with R/Shiny

R/Shiny [11] has quickly become a very popular framework to develop interactive web applications in biological studies since it was released by the RStudio company in 2012. R/Shiny can be used to develop complex web applications with pure R code, which is the largest advantage for R users who are not very familiar with Web techniques, including HTML, CSS and JavaScript. An R/Shiny application can be built even without the use of MySQL and Apache. R/Shiny can also promote cross-team collaborations and reproducible researches by allowing specialists from different disciplines to develop their own applications. Another merit of web application development with R/Shiny is the free availability of tens of thousands of R packages covering almost every field of biological studies, which can be fully exploited in the analyses of biological datasets. The powerful visualization systems of the R environment further boost the advantages of R/Shiny. In addition, CSS, HTML, and JavaScript codes can be incorporated into the code of R/Shiny applications to further customize the appearance and functionality of R/Shiny applications. Nevertheless, there are also disadvantages of R/Shiny. Currently, distributed computing and high concurrency are not very well supported by R/Shiny. To process a dataset, R would read the whole dataset into RAM at once and the created R object would live in memory entirely. As a result, R/Shiny is not suitable for the development of web applications aiming to process very large datasets on the server side or very large files needed to be uploaded from the user side. Moreover, an R/Shiny application can be crashed without warning when dependent R packages are updated, as the parameters of an R function implemented in R packages might be altered in the updating. It can also be challenging to debug an R/Shiny application due to the lack of a good automated testing tool. Although the shinytest package [102] is available for testing R/Shiny applications, some repetitive testing tasks have to be performed by hand sometimes (Figure 1).

Conclusions

R/Shiny is an easy-to-use and powerful framework for R users to build interactive web applications, which promotes the analysis and visualization of biological datasets in interactive web pages. An R/Shiny application can be a powerful tool to convey scientific findings and insights, which also allows users to explore their own datasets. More and more interactive web applications were developed by biologists and data scientists with R/Shiny, which also enable the enjoyment of various data visualization and analysis methods of the R environment by a wider audience. With the continuous development of the R environment and the unceasing improvement of R/Shiny, we expect more functional features and higher efficiency of R/Shiny to be realized in the near future.

Key Points

- Development of interactive biological web applications is a major subject of bioinformatics.
- The R/Shiny package had made it incredibly easy to build web applications for the biological science community.
- Utilization of R/Shiny in the building of biological web applications was reviewed.
- Basic and advanced features of biological web applications development with R/Shiny were reviewed.

Supplementary Data

Supplementary data are available online at https://academic.oup.com/bib.

Consent for publication

All authors have reviewed the manuscript and approved the final draft for publication.

Data availability

Supplementary tables are available online at https://github.com/venyao/shinyReview.

Funding

National Natural Science Foundation of China (31900451); the research start-up fund to topnotch talents of Henan Agricultural University (30500581); Scientific and Technological Research Project of Henan Province (202102110015).

References

1. National Genomics Data Center Members and Partners. Database resources of the National Genomics Data Center in 2020. *Nucleic Acids Res* 2020;**48**:D24–33.
2. Mercatelli D, Holding AN, Giorgi FM. Web tools to fight pandemics: the COVID-19 experience. *Brief Bioinform* 2021;**22**:690–700.
3. Altschul SF, Gish W, Miller W, *et al.* Basic local alignment search tool. *J Mol Biol* 1990;**215**:403–10.
4. Navarro Gonzalez J, Zweig AS, Speir ML, *et al.* The UCSC genome browser database: 2021 update. *Nucleic Acids Res* 2021;**49**:D1046–57.
5. Rigden DJ, Fernández XM. The 2021 Nucleic Acids Research database issue and the online molecular biology database collection. *Nucleic Acids Res* 2021;**49**:D1–9.
6. Editorial: the 18th annual Nucleic Acids Research web server issue 2020. *Nucleic Acids Res* 2020;**48**:W1–4.
7. Parker S. *How to Build a LAMP Server*. North Charleston: CreateSpace Independent Publishing Platform, 2015.
8. R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2020.
9. Hornik K. The comprehensive R archive network. *Wiley Interdiscip Rev Comput Stat* 2012;**4**:394–8.
10. Huber W, Carey VJ, Gentleman R, *et al.* Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods* 2015;**12**:115–21.
11. Chang W, Cheng J, Allaire J, *et al. Shiny: Web Application Framework for R*. https://shiny.rstudio.com/ (18 May 2021, date last accessed).
12. Kasprzak P, Mitchell L, Kravchuk O, *et al.* Six years of Shiny in research—collaborative development of web tools in R. *R J* 2020;**12**:155–62.
13. Su W, Fu B. *COVID-19 BULLETIN BOARD*. https://covid-2019.live/ (18 May 2021, date last accessed).
14. Sarkar D. *Lattice: Multivariate Data Visualization with R*. New York: Springer, 2008.
15. Wickham H. ggplot2: elegant graphics for data analysis. *J Stat Softw* 2010;**35**:65–88.
16. Wilkinson L. The grammar of graphics. In: *Handbook of computational statistics*. New York: Springer, 2012, 375–414.
17. Spitzer M, Wildenhain J, Rappsilber J, *et al.* BoxPlotR: a web tool for generation of box plots. *Nat Methods* 2014;**11**:121–2.
18. Postma M, Goedhart J. PlotsOfData-a web app for visualizing data together with their summaries. *PLoS Biol* 2019;**17**:e3000202.
19. Goedhart J. PlotTwist: a web app for plotting and annotating continuous data. *PLoS Biol* 2020;**18**:e3000581.
20. Krzywinski MI, Schein JE, Birol I, *et al.* Circos: an information aesthetic for comparative genomics. *Genome Res* 2009;**19**:1639–45.
21. Cui Y, Chen X, Luo H, *et al.* BioCircos.js: an interactive Circos JavaScript library for biological data visualization on web applications. *Bioinformatics* 2016;**32**:1740–2.
22. Cui Y, Cui Z, Xu J, *et al.* NG-Circos: next-generation Circos for data visualization and interpretation. *NAR Genom Bioinform* 2020;**2**:lqaa069.
23. Gu Z, Gu L, Eils R, *et al.* circlize implements and enhances circular visualization in R. *Bioinformatics* 2014;**30**:2811–2.
24. Yu Y, Ouyang Y, Yao W. shinyCircos: an R/Shiny application for interactive creation of Circos plot. *Bioinformatics* 2018;**34**:1229–31.
25. Yu Y, Yao W, Wang Y, *et al.* shinyChromosome: an R/Shiny application for interactive creation of non-circular plots of whole genomes. *Genomics Proteomics Bioinformatics* 2019;**17**:535–9.
26. Ou J, Zhu LJ. trackViewer: a Bioconductor package for interactive and integrative visualization of multi-omics data. *Nat Methods* 2019;**16**:453–4.
27. Prompsy P, Kirchmeier P, Marsolier J, *et al.* Interactive analysis of single-cell epigenomic landscapes with ChromSCape. *Nat Commun* 2020;**11**:5702.
28. Amezquita RA, Lun ATL, Becht E, *et al.* Orchestrating single-cell analysis with Bioconductor. *Nat Methods* 2020;**17**:137–45.
29. McCarthy DJ, Campbell KR, Lun ATL, *et al.* Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics* 2017;**33**:1179–86.
30. Lun A, McCarthy D, Marioni J. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research* 2016;**5**:2122.
31. Lawrence M, Huber W, Pagès H, *et al.* Software for computing and annotating genomic ranges. *PLoS Comput Biol* 2013;**9**:e1003118.
32. Morgan M, Pagès H, Obenchain V, *et al. Rsamtools: Binary Alignment (BAM), FASTA, Variant Call (BCF), and tabix File Import*. https://bioconductor.org/packages/Rsamtools (18 May 2021, date last accessed).
33. Morgan M, Obenchain V, Lang M, *et al. BiocParallel: Bioconductor Facilities for Parallel Evaluation*. https://github.com/Bioconductor/BiocParallel (18 May 2021, date last accessed).
34. Attali D. *Colourpicker: A Colour Picker Tool for Shiny and for Selecting Colours in Plots*. https://cran.r-project.org/package=colourpicker (18 May 2021, date last accessed).
35. Warnes GR, Bolker B, Bonebakker L, *et al. gplots: Various R Programming Tools for Plotting Data*. https://cran.r-project.org/package=gplots (18 May 2021, date last accessed).
36. Nagraj VP, Magee NE, Sheffield NC. LOLAweb: a containerized web server for interactive genomic locus overlap enrichment analysis. *Nucleic Acids Res* 2018;**46**:W194–9.
37. Sheffield NC, Bock C. LOLA: enrichment analysis for genomic region sets and regulatory elements in R and Bioconductor. *Bioinformatics* 2016;**32**:587–9.

38. Sergushichev AA, Loboda AA, Jha AK, *et al.* GAM: a web-service for integrated transcriptional and metabolic network analysis. *Nucleic Acids Res* 2016;**44**:W194–200.
39. Ooms J, James D, DebRoy S, *et al. RMySQL: Database Interface and 'MySQL' Driver for R.* https://cran.r-project.org/package=RMySQL (18 May 2021, date last accessed).
40. Müller K, Ooms J, James D, *et al. RMariaDB: Database Interface and 'MariaDB' Driver.* https://cran.r-project.org/package=RMariaDB (18 May 2021, date last accessed).
41. Yang J, Li Q, Noureen N, *et al.* PCAT: an integrated portal for genomic and preclinical testing data of pediatric cancer patient-derived xenograft models. *Nucleic Acids Res* 2021;**49**:D1321–d1327.
42. Yao W, Huang F, Zhang X, *et al.* ECOGEMS: efficient compression and retrieve of SNP data of 2058 rice accessions with integer sparse matrices. *Bioinformatics* 2019;**35**:4181–3.
43. Paradis E, Claude J, Strimmer K. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics* 2004;**20**:289–90.
44. Paradis E. pegas: an R package for population genetics with an integrated–modular approach. *Bioinformatics* 2010;**26**:419–20.
45. Shin J-H, Blay S, McNeney B, *et al.* LDheatmap: an R function for graphical display of pairwise linkage disequilibria between single nucleotide polymorphisms. *J Stat Softw* 2006;**16**:9.
46. Yu G, Smith DK, Zhu H, *et al.* GGTREE: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods Ecol Evol* 2017;**8**:28–36.
47. Ingham VA, Wagstaff S, Ranson H. Transcriptomic meta-signatures identified in *Anopheles gambiae* populations reveal previously undetected insecticide resistance mechanisms. *Nat Commun* 2018;**9**:5282.
48. Marini F, Scherzinger D, Danckwardt S. TREND-DB-a transcriptome-wide atlas of the dynamic landscape of alternative polyadenylation. *Nucleic Acids Res* 2021;**49**:D243–d253.
49. Allaire J. *RStudio: integrated development environment for R* In: The R User Conference 2011, Coventry, UK. The R Foundation for Statistical Computing, Indianapolis, Indiana, USA. 2011, 14.
50. Grolemund G. *Shiny HTML Tags Glossary.* https://shiny.rstudio.com/articles/tag-glossary.html (18 May 2021, date last accessed).
51. Boettiger C, Chamberlain S, Hart E, *et al.* Building software, building community: lessons from the rOpenSci project. *J Open Res Softw* 2015;**3**:e8.
52. Dabbish L, Stuart C, Tsay J, *et al.* Social coding in GitHub: transparency and collaboration in an open software repository. In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, Seattle: The ACM Digital Library, Washington, DC, USA. 2012, 1277–86.
53. Emaasit D. *ggplot2 Extensions.* https://exts.ggplot2.tidyverse.org/index.html (18 May 2021, date last accessed).
54. Wickham H, Averick M, Bryan J, *et al.* Welcome to the Tidyverse. *J Open Source Softw* 2019;**4**:1686.
55. Pages H, Aboyoun P, Gentleman R, *et al. Biostrings: Efficient Manipulation of Biological Strings.* https://bioconductor.org/packages/Biostrings (18 May 2021, date last accessed).
56. Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 2010;**26**:139–40.
57. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol* 2014;**15**:550.
58. Yu G, Wang LG, Han Y, *et al.* clusterProfiler: an R package for comparing biological themes among gene clusters. *OMICS* 2012;**16**:284–7.
59. Yu G. *Enrichplot: Visualization of Functional Enrichment Result.* https://yulab-smu.top/biomedical-knowledge-mining-book/ (18 May 2021, date last accessed).
60. Chang W, Park T, Dziedzic L, *et al. shinythemes: Themes for Shiny.* https://cran.r-project.org/package=shinythemes (18 May 2021, date last accessed).
61. Sievert C, Cheng J, RStudio. *bslib: Custom 'Bootstrap' 'Sass' Themes for 'Shiny' and 'rmarkdown'.* https://cran.r-project.org/package=bslib (18 May 2021, date last accessed).
62. Park T. *bootswatch: Free Themes for Bootstrap.* https://bootswatch.com/ (18 May 2021, date last accessed).
63. WHO. *WHO Coronavirus (COVID-19) Dashboard.* https://covid19.who.int/ (18 May 2021, date last accessed).
64. Tebé C, Valls J, Satorra P, *et al.* COVID19-world: a Shiny application to perform comprehensive country-specific data visualization for SARS-CoV-2 epidemic. *BMC Med Res Methodol* 2020;**20**:235.
65. Google. *Google Data Studio.* https://datastudio.google.com/ (18 May 2021, date last accessed).
66. Tableau Software. *Tableau, A Visual Analytics Platform.* https://www.tableau.com/ (18 May 2021, date last accessed).
67. Moment Zero inc. *DashThis, An Automated Marketing Reporting Tool.* https://dashthis.com/ (18 May 2021, date last accessed).
68. Geckoboard. *Geckoboard, Data Dashboards Made for Sharing.* https://www.geckoboard.com/ (18 May 2021, date last accessed).
69. Traject. *Cyfe, All-in-One Business Dashboard.* https://www.cyfe.com/ (18 May 2021, date last accessed).
70. IBM. *IBM Watson Analytics.* https://www.ibm.com/analytics (18 May 2021, date last accessed).
71. Chang W, Borges Ribeiro B. *Shinydashboard: Create Dashboards with 'Shiny'.* https://rstudio.github.io/shinydashboard/ (18 May 2021, date last accessed).
72. Iannone R, Allaire J, Borges B. *flexdashboard: R Markdown Format for Flexible Dashboards.* https://rmarkdown.rstudio.com/flexdashboard/ (18 May 2021, date last accessed).
73. Stachura F, Krzeminski D, Igras K, *et al. semantic.dashboard: Dashboard with Fomantic UI Support for Shiny.* https://appsilon.github.io/semantic.dashboard/ (18 May 2021, date last accessed).
74. Semantic Organization. *Semantic UI Framework Designed for Theming.* https://semantic-ui.com/ (18 May 2021, date last accessed).
75. Vaidyanathan R, Xie Y, Allaire J, *et al. htmlwidgets: HTML Widgets for R.* https://www.htmlwidgets.org/ (2 August 2021, date last accessed).
76. Perrier V, Meyer F, Granjon D, *et al. shinyWidgets: Custom Inputs Widgets for Shiny.* https://cran.r-project.org/package=shinyWidgets (18 May 2021, date last accessed).
77. AnalytixWare. *shinySky: A Set of Shiny Components and Widgets.* https://github.com/AnalytixWare/ShinySky (18 May 2021, date last accessed).
78. Aden-Buie G. *shinyThings: Reusable Shiny Modules and Other Shiny Things.* https://github.com/gadenbuie/shinyThings (18 May 2021, date last accessed).

79. Attali D. *shinyforms: Easily Create Questionnaire-Type Forms with Shiny*. https://github.com/daattali/shinyforms (18 May 2021, date last accessed).
80. Attali D, Edwards T, Wang Z. *shinyalert: Easily Create Pretty Popup Messages (Modals) in 'Shiny'*. https://cran.r-project.org/package=shinyalert (18 May 2021, date last accessed).
81. Sali A, Hass L, Attali D. *shinycssloaders: Add Loading Animations to a 'Shiny' Output While It's Recalculating*. https://cran.r-project.org/package=shinycssloaders (18 May 2021, date last accessed).
82. Tanaka E. *shinycustomloader: Custom Loader for Shiny Outputs*. https://cran.r-project.org/package=shinycustomloader (18 May 2021, date last accessed).
83. Attali D. *shinydisconnect: Show a Nice Message When a 'Shiny' App Disconnects or Errors*. https://cran.r-project.org/package=shinydisconnect (18 May 2021, date last accessed).
84. Attali D. *shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*. https://cran.r-project.org/package=shinyjs (18 May 2021, date last accessed).
85. Tang Y. *shinyjqui: 'jQuery UI' Interactions and Effects for Shiny*. https://cran.r-project.org/package=shinyjqui (18 May 2021, date last accessed).
86. jQuery UI Team, Schmitz A. *jQuery UI, a Curated Set of User Interface Interactions, Effects, Widgets, and Themes Built on Top of the jQuery JavaScript Library*. https://jqueryui.com/ (18 May 2021, date last accessed).
87. Burow C, Kreutzer S, Dietze M, *et al*. RLumShiny: a graphical user interface for the R package "luminescence". *Ancient TL* 2016;**34**:22–32.
88. Xie Y. *Dynamic Documents with R and knitr*. Boca Raton: CRC Press, 2017.
89. Xie Y, Allaire JJ, Grolemund G. *R Markdown: The Definitive Guide*. Boca Raton: CRC Press, 2018.
90. Bailey E. *shinyBS: Twitter Bootstrap Components for Shiny*. https://cran.r-project.org/package=shinyBS (18 May 2021, date last accessed).
91. Coene J, Bacher E. *cicerone: Provide Tours of 'Shiny' Applications*. https://cran.r-project.org/package=cicerone (18 May 2021, date last accessed).
92. RStudio. *shinyapps.io, an Online Service for Hosting Shiny Apps in the Cloud*. https://www.shinyapps.io/ (18 May 2021, date last accessed).
93. RStudio. *RStudio Connect, a Standalone Publishing Platform for the Work Your Teams Create in R*. https://www.rstudio.com/products/connect/ (18 May 2021, date last accessed).
94. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 2009;**25**:1754–60.
95. Li H, Handsaker B, Wysoker A, *et al*. The sequence alignment/map format and SAMtools. *Bioinformatics* 2009;**25**:2078–9.
96. RStudio. *Shiny Server, put Shiny Web Apps Online*. https://www.rstudio.com/products/shiny/shiny-server/ (18 May 2021, date last accessed).
97. Verbeke T, Michielssen F. *ShinyProxy–Open Source Enterprise Deployment for Shiny*. https://www.shinyproxy.io/ (18 May 2021, date last accessed).
98. RStudio. *Official Tutorial on R/Shiny*. https://shiny.rstudio.com/tutorial/ (18 May 2021, date last accessed).
99. RStudio. *Official Technique Articles on R/Shiny*. https://shiny.rstudio.com/articles/ (18 May 2021, date last accessed).
100. Wickham H. *Mastering Shiny*. Sebastopol: O'Reilly Media, 2021.
101. Datacamp. *R/Shiny Courses in Datacamp*. https://www.datacamp.com/search?q=shiny (18 May 2021, date last accessed).
102. Chang W, Csárdi G, Wickham H, *et al*. *shinytest: Test Shiny Apps*. https://rstudio.github.io/shinytest/ (18 May 2021, date last accessed).