Hands-on R and Shiny:

**Interactive Visualization**

[**What is R? 1**](#_n826y3u7i06u)

[**What is RStudio? 1**](#_fydb4aj0dpus)

[**Installation 2**](#_h1fnd361boi3)

[**Let’s do R practice - First Session: 6**](#_a42dqzhxl6y4)

[**Data Import and Cleaning 8**](#_pmsqhjag1n8a)

[**ggplot2: A static graph pack 8**](#_f3dtayrk1d8n)

[**Interactive Charts with R Plotly and Other Packs 13**](#_xbc66quyia89)

[**ShinyApp 13**](#_xfh4g0paxckr)

[**Publish 17**](#_jos8gzn8nndy)

[**Advanced (skip if not comfortable) - Dashboard with shinydashboard 21**](#_mojxtl3q972l)

[**Submission 22**](#_2tpbswzcoosw)

[**Troubleshooting 22**](#_ogk87uuhrkt)

The goal of this hands-on activity is to demonstrate how R, ggplot2 and Shiny App are used to generate visualizations, especially interactive visualizations.

Briefly you just need R and RStudio installed and create an account on R Server for publication.

## What is R?

R is a programming and interpreted language for statistical computing and graphics supported by the R Core Team and the R Foundation for Statistical Computing.

Created by statisticians **Ross** Ihaka and **Robert** Gentleman, R is used among data miners, bioinformaticians and statisticians for data analysis and developing statistical software.

R is maintained by an international team of developers who make the language available through the web page of The Comprehensive R Archive Network (CRAN).

R is one of the most commonly used programming languages in data mining.

R's data structures include vectors, arrays, lists, and data frames.

## What is RStudio?

RStudio is an integrated development environment (IDE) for R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging, and workspace management. RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux).

## Installation

The order matters: R, then RStudio

* First, install the latest and stable R:

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

Latest stable version is R 4.2.2

* Second, install RStudio Desktop

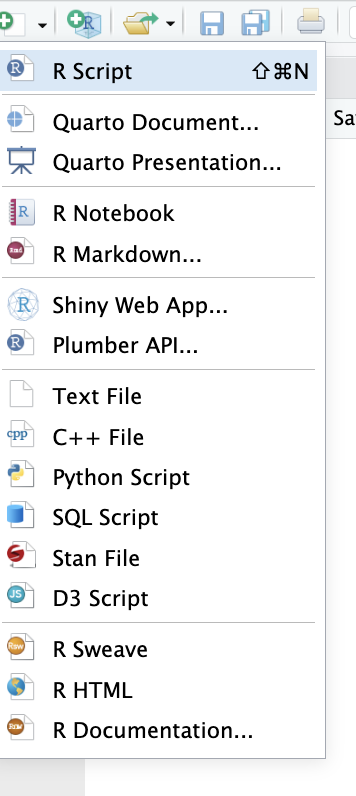
<https://posit.co/downloads/>

* Open RStudio from your computer (don’t open R!)

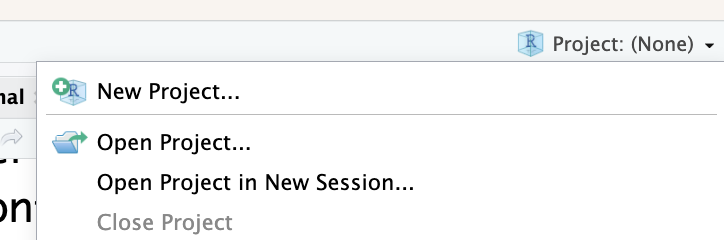
Do you see a blue button for the RStudio start? Open it and see the panels: you don’t need to open R, it is already integrated with RStudio if you installed R and RStudio in order.



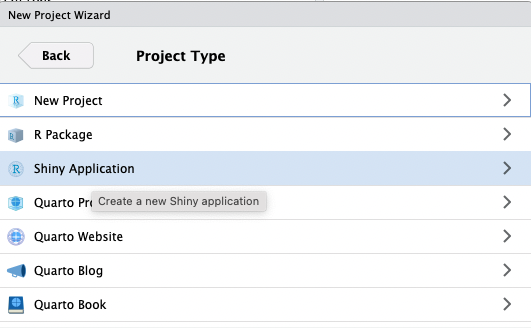
You can create many types of files under the + sign on the top left.

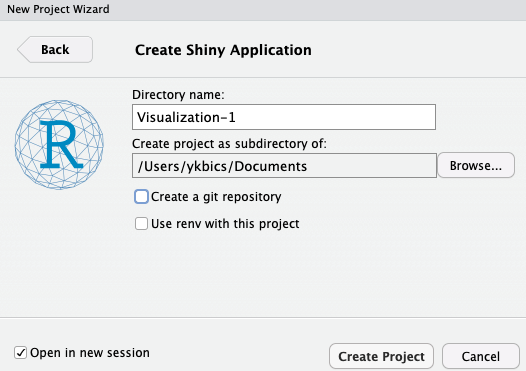


You can create a project from the top right menu: in general, we create a project, then we will work on it. Create a new project with a name.



You can choose a Shiny Application Project if creating a Shiny App. For general, just pick New Project. IN



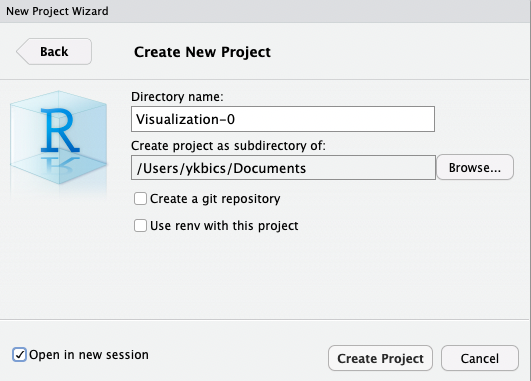


Once you create a new Shiny project, it will start installing (just approve).

In the tutorial, let’s create two projects:

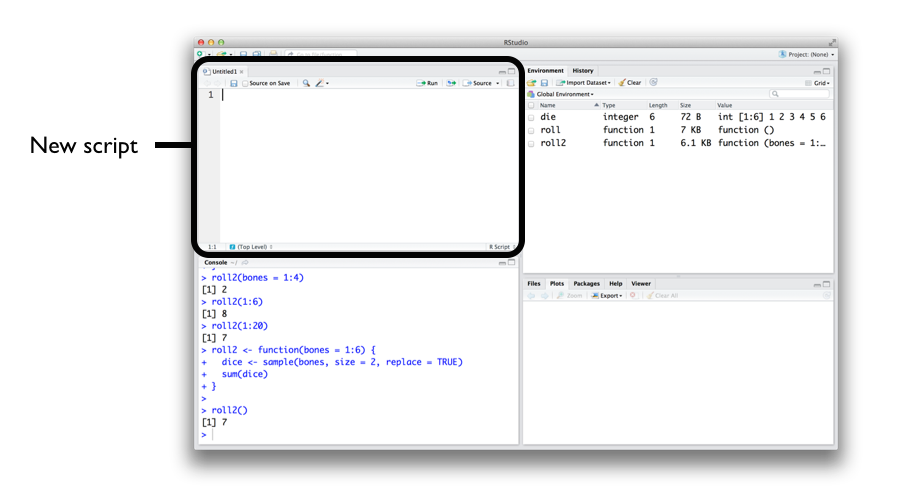
* one for introduction (playground, data cleaning, ggplot2 practices etc) and
* one for Shiny application (interactive visualization).

This is for our playground project:



Check Open in New Session.

Open the R Script panel from the top left menu. Save it.



## 

## Let’s do R practice - First Session

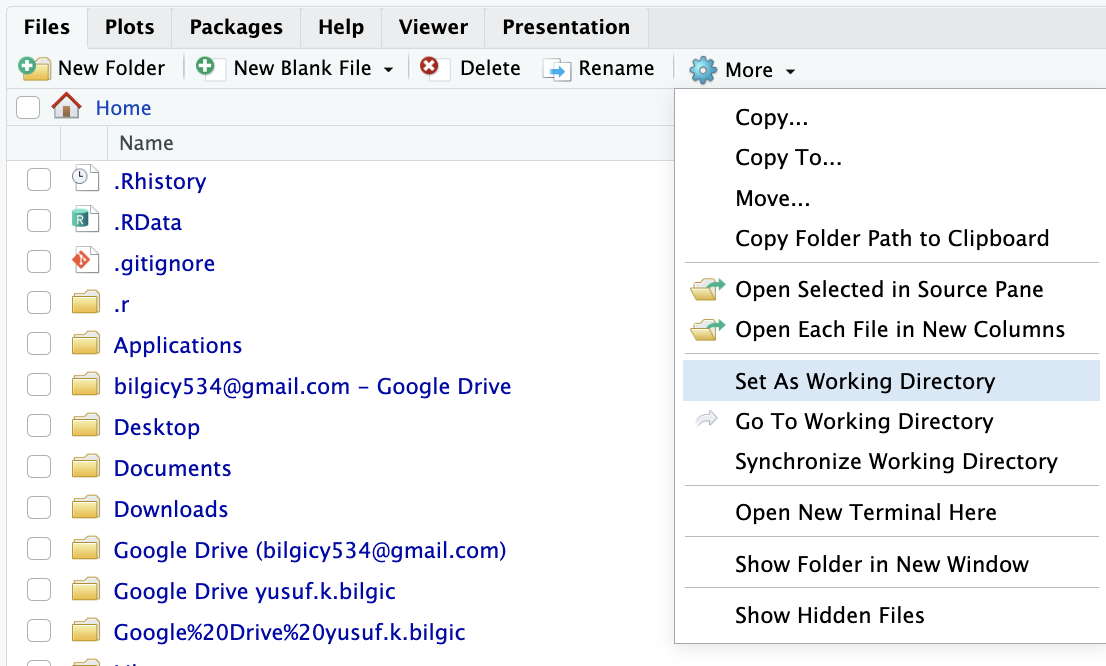
You type R code into the bottom line of the RStudio console pane and then click Enter to run it: try

> 1+1

R Script panel is for typing code island saving in .r file.

If you work under projects, you won’t need to set the working directory. Just put datasets and necessary files under the project’s directory.

If you want to change the working directory to the current directory you open, you can set from the Files menu:



R comes with many data sets preloaded in the datasets package, which comes with base R. You can select any. Run one by one:

help(package = "datasets")

data(Titanic)

View(Titanic)

glimpse(Titanic) #you need to install and load the tidyverse pack

#make data frame

df=data.frame(Titanic)

dim(df)

colnames(df)

df$Freq #this pick the column

summary(df)

Try this chunk of code one by one:

# Import dataset

data\_iris <- iris

View(data\_iris)

head(data\_iris, 5)

print(ncol(data\_iris))

print(nrow(data\_iris))

dim(data\_iris)

str(data\_iris)

attributes(data\_iris)

colnames(data\_iris)

summary(data\_iris)

data\_iris[,1] #one column

data\_iris$Sepal.Length

data\_iris[,c("Sepal.Length", "Sepal.Width")] #subsetting

Just use any dataset in csv format from our previous work (midterm’s data, Tableau hands-on data etc), call it data1, import and practice to summarize and access the columns:

df <- read.csv("data1.csv")

df <- read.table("poker.csv", sep = ",", header = TRUE)

Try this and run:

df <- read.csv("https://gist.githubusercontent.com/garrettgman/9629323/raw/ee5dfc039fd581cb467cc69c226ea2524913c3d8/deck.csv")

If the data is in excel format, <https://rstudio-education.github.io/hopr/dataio.html#excel-spreadsheets> and <https://rstudio-education.github.io/hopr/dataio.html> helps to show how to import.

The Very Basics about R

Complete most of the practices (skip many)

* Go over very basics about R: <https://rstudio-education.github.io/hopr/basics.html>
* Installing packages and help pages: <https://rstudio-education.github.io/hopr/packages.html>
* Loading and saving the data: <https://rstudio-education.github.io/hopr/dataio.html>

## Data Import and Cleaning

Once you are familiar with the environment of RStudio, time to import and clean data. This tutorial is great: you can use the data for the assignment.

* Complete this tutorial about data import and cleaning: <https://www.dataquest.io/blog/load-clean-data-r-tidyverse/>

## ggplot2: A static graph pack

This pack was developed by Wichkam. ggplot2 is a R package dedicated to data visualization. It can greatly improve the quality and aesthetics of your graphics.

The qplot() function used to make the histogram is from the ggplot2 package, which is a core tidyverse package.

* Gallery

<https://r-graph-gallery.com/ggplot2-package.html>

Scroll down and read the text and code

* Layers:

Layers are the fundamental elements that create the objects that we perceive on a plot. They have a visual & an aesthetic function, but they also tie together aspects of both the story and its presentation. The use of layers in a system is one of the most common ways to organize information and to present results in multiple formats.

A layer consists of **four components**: 1) data and mapping, 2) a statistical transformation, 3) a geometric object, and 4) a position adjustment.

From Ahmed’s article:

* Every ggplot2 graph starts with the function ggplot(). It basically creates the **coordinate** system. Over this the **graphical layers** are added.
* The first argument of this function is the **input dataset** for the intended graph.
* Then comes the **geom functions** which add the layers of plotting on the coordinate system according to its geom i.e. geom\_point, geom\_line, geom\_bar etc.
* Every geom function needs **a mapping argument**. This defines how the variables in the dataset are **mapped to visual properties**.
* The **aesthetic function aes()** is assigned to the mapping argument. The main arguments of the aes() function are **axes augments-x, y** and differentiating arguments like **color, size, fill, alpha**.
* The differentiating arguments become common featured arguments when they are put outside of the aes() function. **ggtitle(), xlab(), ylab(), theme()** these functions are used for the labeling and thematic attributes.
* You can find details of these functions in the help tab in R by executing the command — ?function\_name , like ?geom\_point
* A nice presentation was made a student about ggplot2’s philosophy based one one of our readings:

[GrammerOfGraphics-ISTE782\_Suvrat\_Ritambara.pptx](https://docs.google.com/presentation/d/1pGF6rO1MZQrgIP4xCwiv0izGz6rmxDzZ/edit?usp=sharing&ouid=115698378242200697041&rtpof=true&sd=true) See the grammar of ggplot2.

* Now, time to start some practices

<https://r-graph-gallery.com/277-marginal-histogram-for-ggplot2.html>

Get the code, copy-paste into the script panel, save and run. Install necessary packs first by doing

> install.packages(‘...’)

> library(...)

Then, copy paste the code and run chunk by chunk using ggplot, which is shortcut for ggplot2:

**Example-0:**

# Install packs and load

install.packages('ggplot2')

library(ggplot2)

data\_iris <- iris

# Scatterplot

ggplot(data\_iris) +

geom\_point(aes(x = Sepal.Length , y = Sepal.Width,

color = Species, shape = Species), size = 4) +

ggtitle("Scatter Plot with feature differentiation - 1") +

xlab("SEPAL LENGTH") +

ylab("SEPAL WIDTH") +

theme\_bw() +

theme(axis.text.x = element\_text(face = 'bold.italic',

color = 'darkgreen',

size = 10, angle = 0),

axis.text.y = element\_text(face = 'bold',

color = 'blue',

size = 10, angle = 45))

**Example-1:**

# Install packs

install.packages(‘ggplot’)

library(ggplot2)

# Data

data("midwest", package = "ggplot2") # load the data

# Or,

#midwest <- read.csv("http://goo.gl/G1K41K") # alt source

# Init Ggplot

ggplot(midwest, aes(x=area, y=poptotal)) # area and poptotal are columns in 'midwest'

# Simple scatterplot

library(ggplot2)

ggplot(midwest, aes(x=area, y=poptotal)) + geom\_point()

# Drawing the line of best fit

g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom\_point() + geom\_smooth(method="lm") # set se=FALSE to turnoff confidence bands

plot(g)

# Adjusting limits

g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom\_point() + geom\_smooth(method="lm") # set se=FALSE to turnoff confidence bands

# Delete the points outside the limits

g + xlim(c(0, 0.1)) + ylim(c(0, 1000000)) # deletes points

# g + xlim(0, 0.1) + ylim(0, 1000000) # deletes points

# Zoom in

g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom\_point() + geom\_smooth(method="lm") # set se=FALSE to turnoff confidence bands

# Zoom in without deleting the points outside the limits.

# As a result, the line of best fit is the same as the original plot.

g1 <- g + coord\_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) # zooms in

plot(g1)

# Add title

g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom\_point() + geom\_smooth(method="lm") # set se=FALSE to turnoff confidence bands

g1 <- g + coord\_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) # zooms in

# Add Title and Labels

g1 + labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area", caption="Midwest Demographics")

# or

g1 + ggtitle("Area Vs Population", subtitle="From midwest dataset") + xlab("Area") + ylab("Population")

# Change color and size

ggplot(midwest, aes(x=area, y=poptotal)) +

geom\_point(col="steelblue", size=.3) + # Set static color and size for points

geom\_smooth(method="lm", col="firebrick") + # change the color of line

coord\_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +

labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area", caption="Midwest Demographics")

# Categories

gg <- ggplot(midwest, aes(x=area, y=poptotal)) +

geom\_point(aes(col=state), size=.3) + # Set color to vary based on state categories.

geom\_smooth(method="lm", col="firebrick", size=.2) +

coord\_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +

labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area", caption="Midwest Demographics")

plot(gg)

#

gg + theme(legend.position="None") # remove legend

#

gg + scale\_colour\_brewer(palette = "Set1") # change color palette

# Color Brewer

library(RColorBrewer)

head(brewer.pal.info, 10) # show 10 palettes

**Example-2:**

# load package and data

install.packages('ggExtra', 'ggplot2')

library(ggplot2)

library(ggExtra)

# Data

data(mpg, package="ggplot2")

# mpg <- read.csv("http://goo.gl/uEeRGu")

# Scatterplot

theme\_set(theme\_bw()) # pre-set the bw theme.

mpg\_select <- mpg[mpg$hwy >= 35 & mpg$cty > 27, ]

g <- ggplot(mpg, aes(cty, hwy)) +

geom\_count() +

geom\_smooth(method="lm", se=F)

ggMarginal(g, type = "histogram", fill="transparent")

ggMarginal(g, type = "boxplot", fill="transparent")

**Example-3:**

# library

library(ggplot2)

library(ggExtra)

# The mtcars dataset is proposed in R

head(mtcars)

# classic plot :

p <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, size=cyl)) +

geom\_point() +

theme(legend.position="none")

# Set relative size of marginal plots (main plot 10x bigger than marginals)

p1 <- ggMarginal(p, type="histogram", size=10)

# Custom marginal plots:

p2 <- ggMarginal(p, type="histogram", fill = "slateblue", xparams = list( bins=10))

# Show only marginal plot for x axis

p3 <- ggMarginal(p, margins = 'x', color="purple", size=4)

p1

p2

p3

**Example-4: Optional**

Next tutorials on ggplot2 require installing many packs: in case the packs don’t work, just read the code.

* <https://towardsdatascience.com/data-visualization-with-ggplot2-db04c4956236>
* <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html> (also many nice templates available here)

## 

## Interactive Charts with R Plotly and Other Packs

An interactive chart allows the user to perform actions: zooming, hovering a marker to get a tooltip, choosing a variable to display and more. R offers a set of packages called the html widgets: they allow you to build interactive dataviz directly from R.

<https://r-graph-gallery.com/interactive-charts.html>

See the gallery.

See the code. Wanna try some. Practice one-two. Get the plots. **You may wanna first screenshot it to post.**

## Shiny App

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions.

Shiny is a web application framework for R. It does not require the user to code HTML. There are normally two sets of code files: **the server** and **the user interface** (UI). Both sets of code work on top of a Shiny server. A Shiny server can reside on one of your machines or in the cloud (via several hosting companies).

* The UI file only establishes the layout of the page
* The Server file load packs, data, and make plots

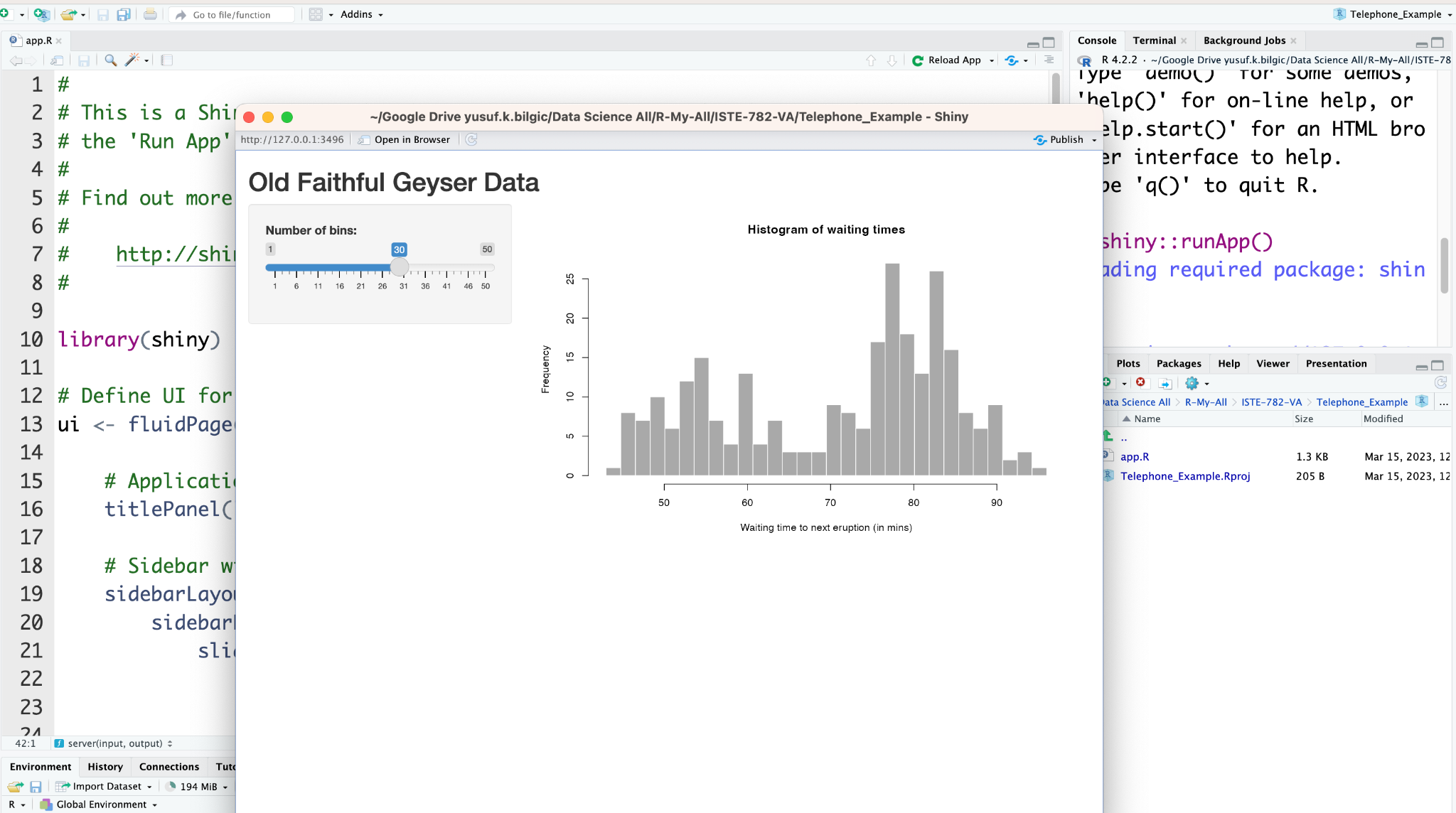
With RStudio you can develop the code sets for the **server.R** and **ui.R** components (though it is possible to put them together under **app.R**) and then run them. Running a Shiny application will:

* Open a new browser window
* Use the coding to generate the corresponding HTML for your coding
* Display the generated HTML in the new browser window
* Did you create a Shiny Project? Use the code from the templates and practice to generate the plots.
* Now, discover Shiny

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

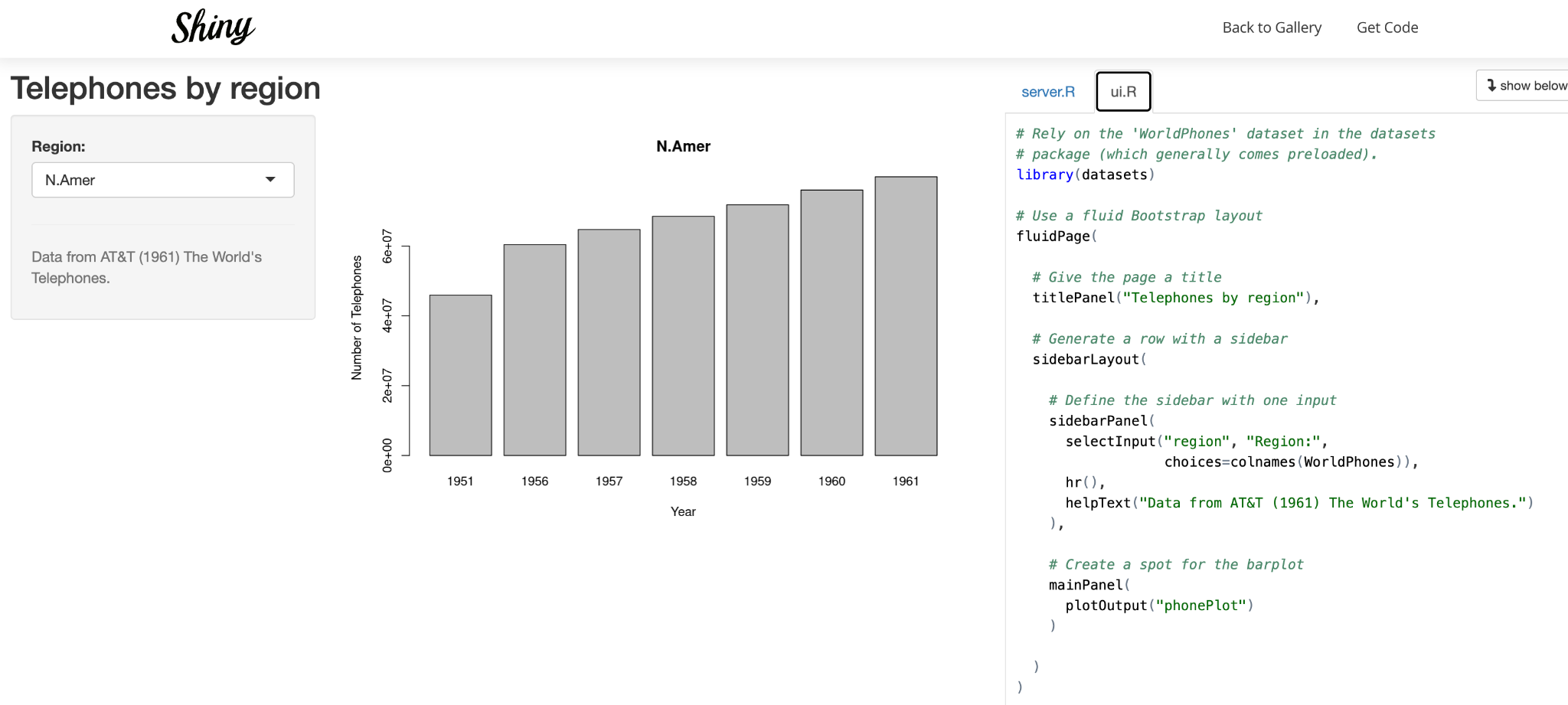
* A simple example from <https://shiny.rstudio.com/gallery/telephones-by-region.html>. Practice on your Shiny project. Either you need to put all these two code files under app.r (see the R makes the template ready when you create a new Shiny project) OR create two files under the Shiny project, and run:

Initially see the template built-in: run.



Then, you can replace the template with the telephone data:

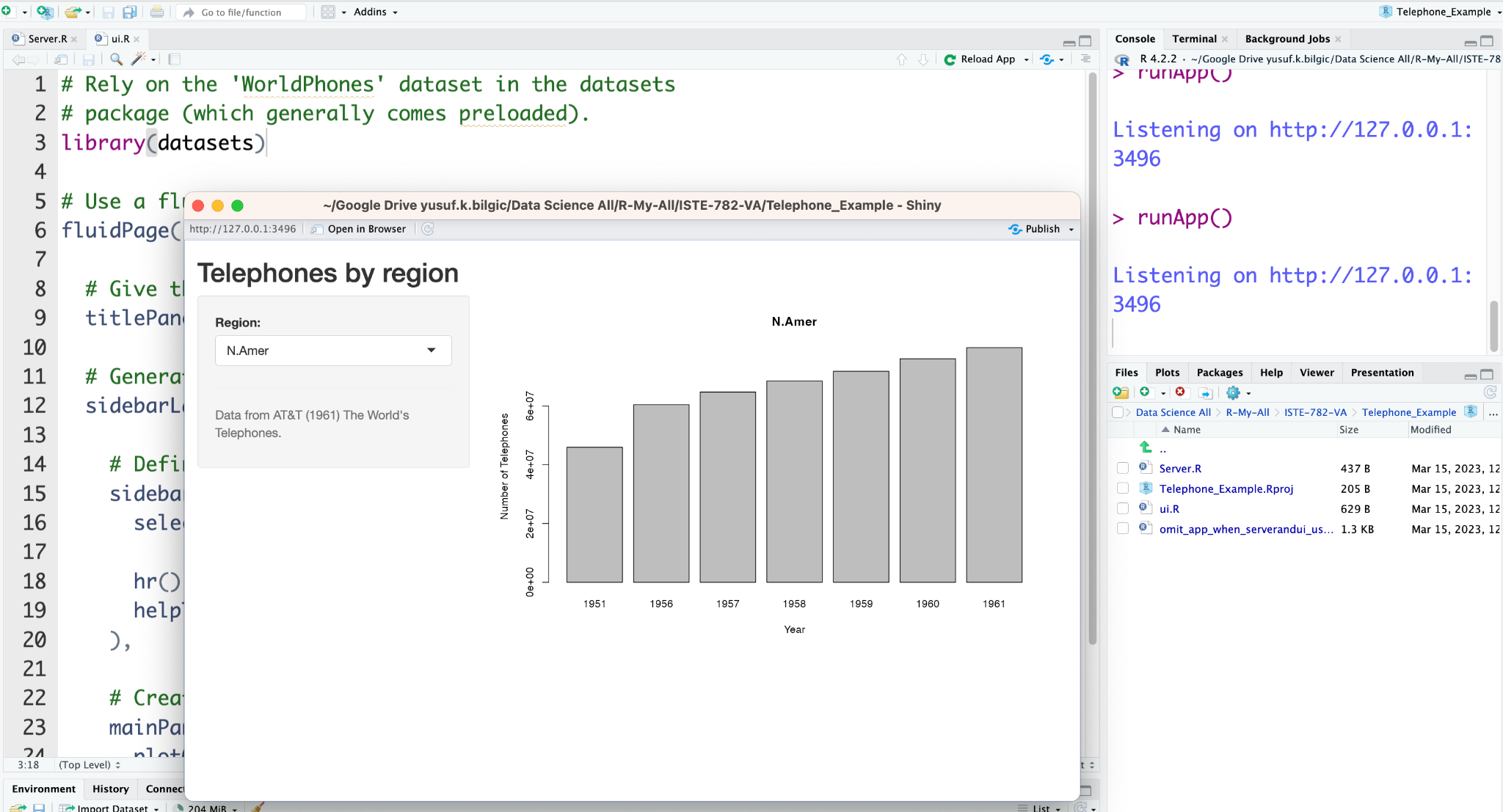




See how data set is imported from the R base and specified what column to use on drop-down menu:

WorldPhones[,input$region]

You need to change the name of app.r if you want to use the two files to generate the Shiny app:



* See the gallery

<https://shiny.rstudio.com/gallery/>

Spend time on this site. See the code structure, source files on github etc.

* Now, visit Shiny Demo

Either scroll down on the gallery or visit one of them: <https://shiny.rstudio.com/gallery/kmeans-example.html>

See the code from Get Code on top right: two files: server.R and ui.R

These two files are always needed to create Shiny charts. Also, app.R can combine both code in a single R file to create Shiny app.

You will practice some of these visualizations.

* Now, see Interactive visualizations

<https://shiny.rstudio.com/gallery/superzip-example.html> is a good one

Get Code. You will practice and generate something similar to this dashboard.

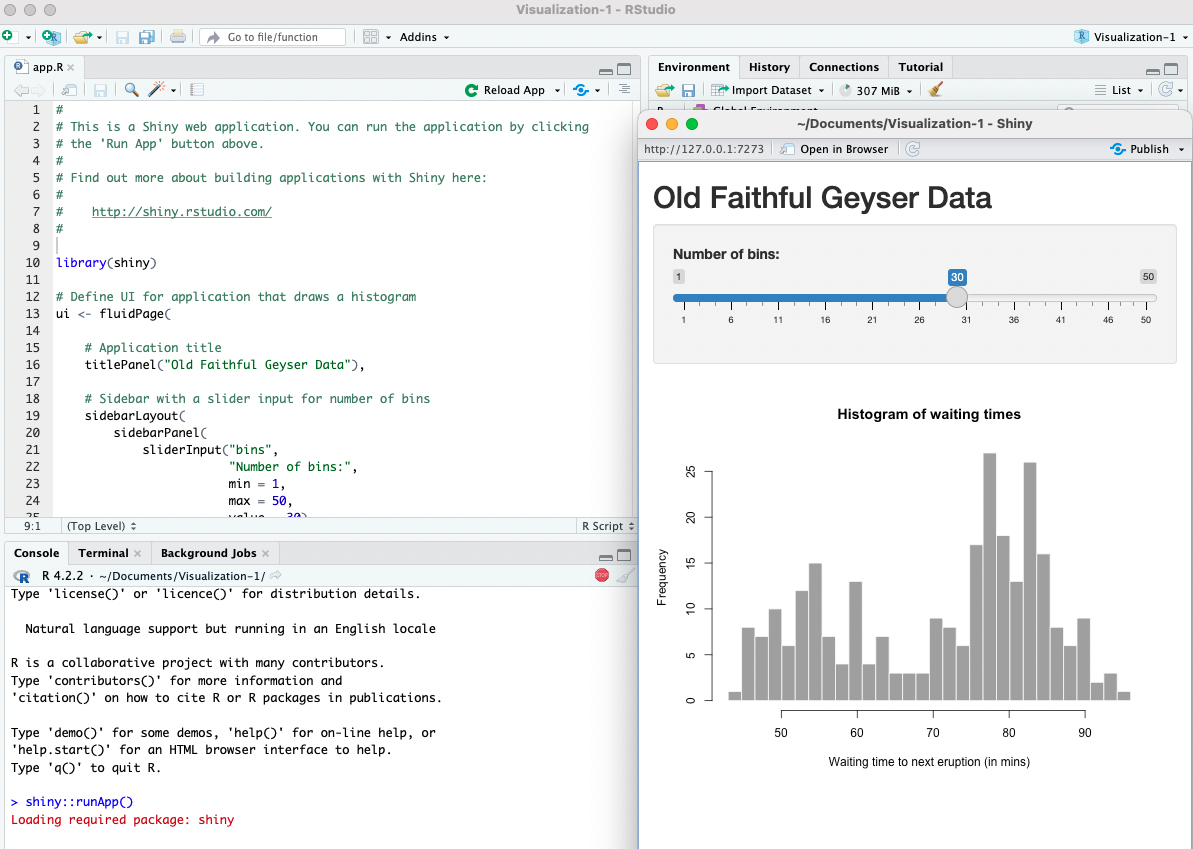
* Visit the Application layout

<https://shiny.rstudio.com/gallery/tabsets.html>

Get Code. Practice some. Explore web capacities.

* Now, time to create Shiny applications

Remember we created a Shiny project. Open it and run. You will see the interactive panel/browser. Read the code. It is ready to modify with the ones you just visited.



## Publish

* Now, time to publish:

Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications). You will need to create a shinyapps.io account and then this will allow you to publish/deploy the Shiny applications to the cloud. Free.

Go to shinyapps.io and click “Dashboard.” The site will ask you to sign in using your email and password, your Google account, or your GitHub account.

The first time you sign in, shinyapps.io prompts you to set up your account. Shinyapps.io uses the account name as the domain name for all your apps.

Once you set up your account in shinyapps.io, you can configure the rsconnect package to use your account. Shinyapps.io automatically generates a token and secret for you, which the rsconnect package can use to access your account.

Before you get started with publishing on shinyapps.io, you will need: 1) An R development environment, such as the RStudio IDE (you have!), and 2) the latest version of the rsconnect R package

install.packages('rsconnect')

library(rsconnect)

**Read the instructions from:**

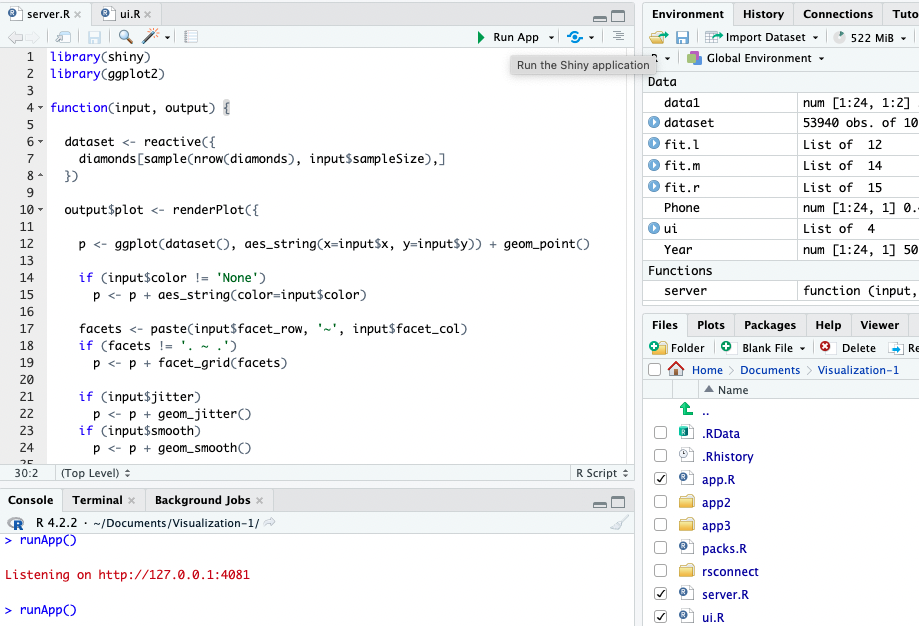
<https://shiny.rstudio.com/articles/shinyapps.html?_ga=2.253239434.220891967.1678218702-1656129218.1678218702>

**You will design your first Shiny App and probably get the screenshot as second.**

* Scroll down and read `A Demo app`: Follow the steps to create your first own Shiny app.

On the R console, run > install.packages(c('ggplot2', 'shiny'))

Once you created the two files, server.r and ui.r, copied the code in the demo, Run App. Open in Browser.



The browser will show the Shiny output.

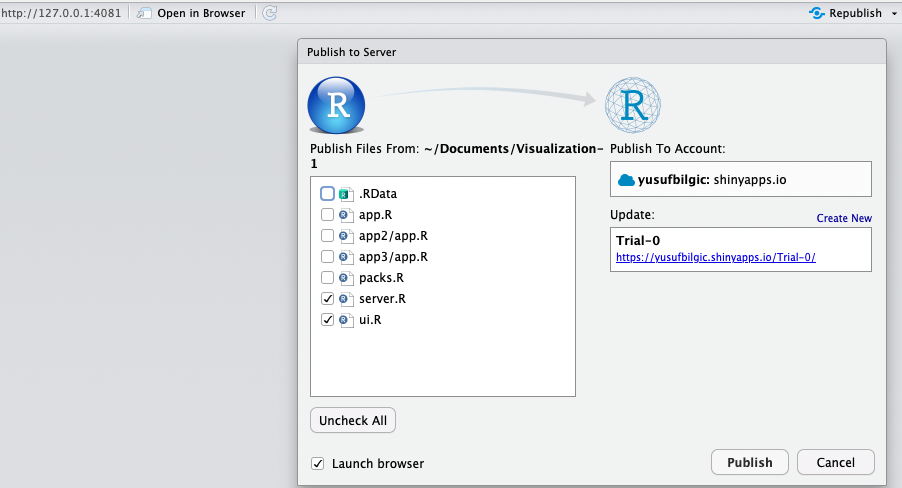


Publishing (deploying) the server.r and ui.r files, let’s call it Shiny App, (also, if you put them altogether in app.r, you can publish it as well) has two ways: either use the buttons on RStudio or use the deployApp command from the **rsconnect** packages.

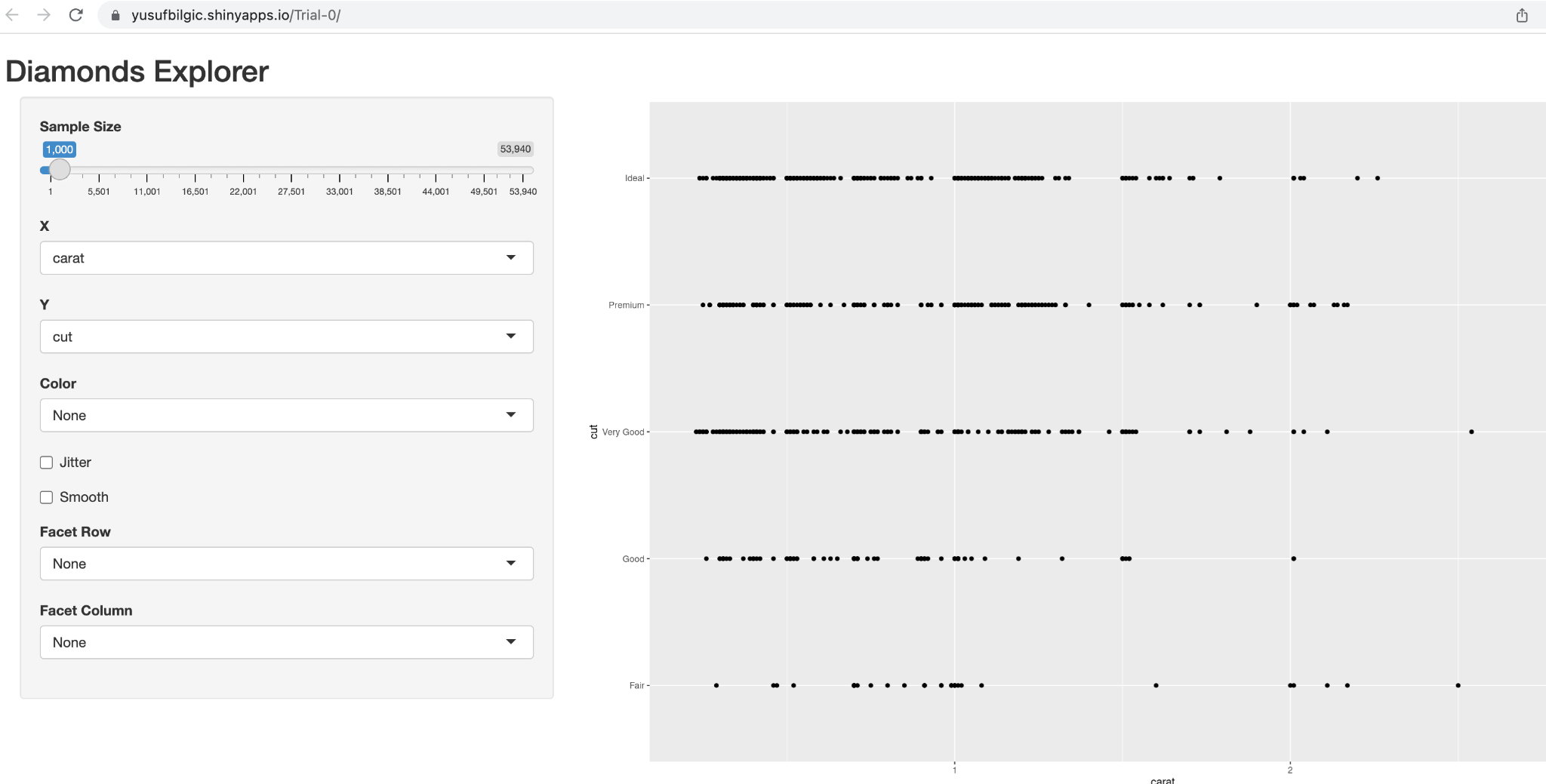
> library(rsconnect)

> deployApp()

I preferred the button (not the commands) and the two files to publish/deploy on the web:



Then, the trivial work is published on the web:



Please publish your first trivial work on the web. Feel free to use different templates.

Get the link, paste on a different browser and see if it is working.

Now, change the file with the code for interactive plots you visited. Save, run, publish, see the changes.

Do a couple of examples: importing data is a crucial portion that needs attention. How to import data?

By default, shinyapps.io deploys all applications on ‘medium’ instances, which are allowed to use 512 MB of memory.

In console,

> library(rsconnect)

> terminateApp("<your app's name>")

> terminateApp('Trial-0') #my case

## 

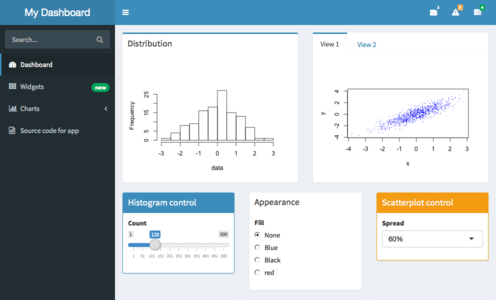
## Advanced (skip if not comfortable) - Dashboard with shinydashboard

So far, we haven’t designed any dashboard yet. You may wanna consider doing deep in the project.

**shinydashboard** makes it easy to use Shiny to create dashboards.

**shinydashboard** calls upon shiny to create a dashboard page with header, sidebar and body# each of these components is defined in the preceding app.R code.

Start reading from <https://rstudio.github.io/shinydashboard/>



This gallery is worth taking look at: <https://gallery.shinyapps.io/087-crandash/> with its source code <https://github.com/rstudio/shiny-examples/tree/main/087-crandash>

Once you design a dashboard with two plots, publish it.

**—**

**As much as possible, complete the instructions above. Skip some. It will take time.**

**—**

## 

## Submission

Submit on Discussions: (1) the **four** screenshots of the visualizations and dashboard that show the best parts of your practice and (2) one published **web link** of any of your Shiny app work.

## Troubleshooting

* Visit the Slack server and start sharing issues and suggestions.
* ?