DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 1 : Introducing shinyjs: perform common JavaScript operations in Shiny apps using plain R code

Action buttons and action links are different from other Shiny widgets because they are intended to be used exclusively with observeEvent() or eventReactive(). How action buttons work Create an action button with actionButton() and an action link with actionLink().

Shiny has a wide array of input widgets e. Take a look at the RStudio widget gallery for a complete list. Note that both of these, and all widgets, have a unique input id inputId â€" the text is mytext and the slider is myslider. Careful, this often causes trouble â€" the input ids must be unique. In this particular example, we are including both a slider and a text input. Allow user input Simple app with widgets, though the widgets are not connected to the server yet. You can enter text and play with the slider but nothing will happen. There is no code in the server to tell it to listen or react. In order to listen, it needs to know what to listen to and this is where our unique input ids come in. Our widgets all have a unique ID that the server will listen for and react to. Each of these input ids is mapped to the input argument on our server. Instead you need to wrap these reactive values in one of the functions designed to handle interactive widget output. Try running the code below in your own console and you will get an error operation not allowed without an active reactive context. This means that to read the reactive value you need to wrap it in a function designed to listen to the reactive elements. In a console, you will see the error. Here the app will fail to load. In the next few apps I will be using a function called updateTextInput instead of print â€" essentially I will use a text box as a console and will print results to the text box with updateTextInput. Here is an example: They were designed to listen to reactive elements and respond by causing side effects, like updates to text boxes or pull-downs. Unlike the reactive function, which we cover next, they should not be used to return data or values. There are two flavors of observe. With observe the code inside will get triggered when any of the reactive values inside change. With observeEvent code will only be triggered by specified reactive values. I would suggest that you use observeEvent whenever possible because observeEvent forces you to think through and specify the reactions you want to see. So, back to the example from above. Any time the user makes a change to mytext the observer code will run and myresults will be updated. So in the following code the results text box will update if the user interacts with the input text box or with the slider. Any reactive value in the observe function will trigger all the code in the observe function to run. Note that in the next couple of apps I paste a random number to the text to make it easier to see updates. With observeEvent the code inside will only run if the specified reactive value s change. So in the following code the update will only execute if the user makes changes to the text box. Only specified reactive s trigger the code to run using observeEvent. Using observeEvent instead of observe allows you to specify the reactive values to listen for and react to. You can use an observe function priority argument to do this. The default priority is 0 and higher numbers mean higher priority and you can use negative numbers. No priority specified The order of execution is not always predictable in Shiny apps. In this example we have two observers and they both write to the same output text box. This app uses default priorities and the second observer will run second and will, therefore, write over the updates from the first observer. Prioritize to control order of execution Instead of default priorities we will force the first observer to run second so that it writes over the updates from the second observer. In order to do this we have a higher priority for the second observer so that it runs first. Because using reactive creates a function and returns results you generally save a reactive as an object and use it elsewhere in your server as you would use any R function. The reactives are NOT supposed to generate side effects, they should essentially be self-contained. So in the example, below I use reactive to create a self-contained function called myresults. Since I want to print the results to the console a side effect , I run the reactive function from within an observer. When it hears a change it generates a string as output. Using eventReactive to prevent unwanted reactions Sometimes you only want your reactive function to listen for specific reactive values and this is when you use eventReactive or observeEvent. Each time the user changes the text value the server updates the text box. But, and here is a mini test,: What happens if you have no reactive value in an observe function? What happens if your reactive value is in the observe function but is not involved in any calculations? So in

this server code the values will print to the console once and then never again. Nevertheless, any time your user changes the text box this server will print 1 to 10 to the console. This is a very important concept â€" a reactive value in your observe or reactive functions will trigger that function to run if the user interacts with it even if the reactive element is not part of the calculations. In other words, keep calculations separated as much as possible. Reactive functions are a good way to do this. You can use reactive functions to isolate code and only run that code when necessary. Take, for example, the following app. I have included the reactive associated with both the slider and the text input in the same observer. As a result, even if the user only changes the slider, all the code in the observer will get run, even the pieces associated with the text box. The reverse is also true. You might want to isolate these pieces. Reactive values are kept separate better I refactored the code above so that the slider and text reactive values are in separate reactive functions. This way if the text reactive changes only the relevant code gets run. Likewise for the slider. An alternative to observeEvent or eventReactive Above you saw that observeEvent and eventReactive can limit reactions to specified reactive values. Use isolate to avoid triggering reactions In this example we have two text boxes that update with the input text. In the previous mini-apps we listened and reacted to reactive values with observe or reactive but simply updated a text box. The functions like renderText or renderPlot and their UI counterparts, textOutput and plotOutput, enable you to create meaningful output. An initial example with renderText and textOutput In order to return values to the user we need a strategy to 1 grab needed values from the UI in the server; 2 process as necessary and then 3 return the result to the UI. To do this RStudio created a suite of functions that tag-team and circulate the value to and from the server and UI. The User-to-Server-back-to-User process in broad strokes: The renderText function, in the server, would be used to read the text box reactive value and process as necessary. Then the renderText function would send the result back to the Shiny UI by attaching the results to the output object. The User-to-Server-back-to-User process in detail: Using our own app as an example. Our UI has a text input box called mytext. Also similar to the text example, the reactive values move from the UI to the server and back to the UI. Dynamic user interface Instead of returning a single object, we are returning a list of objects. The selector gets updated when the user clicks on the button note observeEvent. Each time the user clicks the button the selections gets updated. In a later example, I will link them together in a more meaningful way but there is one more topic I want to cover before I do that. The observe is designed to update the text box a side effect, but does not produce output. The renderTable function returns the table to the UI. They require a Shiny server, a server that can run your R commands. R and click on Run App in the top right. For a single or multi-page app you can use the function runApp where you specify the directory your app. R files are housed in. Running your own Shiny server There is a free, open source version of the Shiny server that you can run on, for example, Amazon Web Services or your own server. This is designed for apps with a relatively low number of visitors. Running Shiny Server Pro RStudio also sells a yearly subscription to Shiny Server Pro that provides security, admin and other enhancements when compared to the open source version. You can view a comparison of the open source and pro version here. Add-on packages shinyjs The shinyjs package, created by Dean Attali , allows you to use common JavaScript operations in your Shiny applications such as hiding an element, delaying code etc. The package provides more than a dozen useful functions that are described in a page on GitHub. His talk on the package at the Shiny Developers Conference is also worth watching and will be posted by RStudio in the near future. In order to use the functionality you need to load the package and then activate it in the UI with the useShinyjs function.

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 2 : Interactive data visualization with Shiny

Description. Creates an action button or link whose value is initially zero, and increments by one each time it is pressed.

The ease of working with Shiny has what popularized it among R users. These web applications seamlessly display R objects like plots, tables etc. Shiny provides automatic reactive binding between inputs and outputs which we will be discussing in the later parts of this article. It also provides extensive pre-built widgets which make it possible to build elegant and powerful applications with minimal effort. Any shiny app is built using two components: This file creates the user interface in a shiny application. It provides interactivity to the shiny app by taking the input from the user and dynamically displaying the generated output on the screen. This file contains the series of steps to convert the input given by user into the desired output to be displayed. Setting up shiny Before we proceed further you need to set up Shiny in your system. Follow these steps to get started. Create a new project in R Studio 2. Select type as Shiny web application. It creates two scripts in R Studio named ui. Each file needs to be coded separately and the flow of input and output between two is possible. The user interface can be broadly divided into three categories: The content in the title panel is displayed as metadata, as in top left corner of above image which generally provides name of the application and some other relevant information. Sidebar layout takes input from the user in various forms like text input, checkbox input, radio button input, drop down input, etc. It is represented in dark background in left section of the above image. It is part of screen where the output s generated as a result of performing a set of operations on input s at the server. R with an example: R loading shiny library library shiny shinyUI fluidPage fluid page for dynamically adapting to screens of different resolutions. R This acts as the brain of web application. R is written in the form of a function which maps input s to the output s by some set of logical operations. The inputs taken in ui. We will be discussing a few examples of server. R in the coming sections of the article for better understanding. Deploying the Shiny app on the Web The shiny apps which you have created can be accessed and used by anyone only if, it is deployed on the web. It provides free of cost platform as a service [PaaS] for deployment of shiny apps, with some restrictions though like only 25 hours of usage in a month, limited memory space, etc. You can also use your own server for deploying shiny apps. Steps for using shiny cloud: Sign up on shinyapps. Go to Tools in R Studio. Open publishing tab Step 5: Manage your account s. Using Shiny Cloud is that easy! Creating interactive visualization for data sets The basic layout for writing ui. Drawing histograms for iris dataset in R using Shiny Writing ui. Drawing Scatterplots for iris dataset in R using Shiny Writing ui. To brief you about the data set, the dataset we will be using is a Loan Prediction problem set in which Dream Housing Finance Company provides loans to customers based on their need. We will be creating an explanatory analysis of individual variables of the practice problem. Explanatory analysis of multiple variables of Loan Prediction Practice problem. Explore Shiny app with these add-on packages To add some more functionality to your Shiny App, there are some kick-ass packages available at your disposal. Here are few from RStudio.

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 3 : A nice RShiny On/Off switch button Â· StatnMap

Jun 13, Â· R shiny: Add weblink to actionButton. Ask Question. I have a box in my shiny application that has a button included within a shiny dashboard box like this.

The updated version of this post is available on my new blog: In this tutorials sequence, we are going to see three tricks to do the following in a Shiny app: Today, we are going to see how to add buttons in each rows of a datatable to delete, edit or compare it with other rows. The app is live here. Buttons to delete, edit and compare Datatable rows Here is what we want to achieve: The final application As you can seen, the final application offer a lot of functionalities, like selecting row with checkboxes, comparing, deleting them and also modifying them. Even though you can select the rows via the classical datatable functions, checkboxes make it more visual for the users he knows there is something it can do. In addition to this, the button are a clear call to action so that the users do not have to serach how to do the different actions. It all begins with a data. R library shinydashboard library data. The apps create fakes yearly sales data for 10 brands, with a name, an email and a fake estimated growth. The data is purely cosmetic. The body contains a title, three buttons which are grouped using the html div btn-group. When Shiny generates the html code, the div will encapsulate the action buttons which will be showed as grouped. Storing the table in a reactive value: Since we will want to perform action on the data with buttons and eventObserver the modifications of the data will be done with side effects. Hence reactiveValues are a must. On which we added some buttons Now, lets add the buttons to our datatable. Basically, they will be added through html code and the datatble will be asked not to escape strings. The checkboxes will be used to select the rows on which the user want to perform some actions Deletion, comparison. Since they all have the same name it will be easy to access their value through JS. Each row will contain a group of two buttons, one to delete the row and the other one to modify it. Again, the app will access the row to delete and the action to do through the button ID. But these buttons were doing nothing. No matter you click on the buttons or not, nothing will happen. They need to be linked with the Shiny App. Now, to be able to compare the selected row and to delete them, we need to create a new Shiny input with the list of selected checkbox. Then when the user is clicking on an input, the script runs a function. We are storing all the checkboxes and we create an empty arrow that will be filled with the id of ticked checkbox. To compare the rows, the code is similar. The modal is generate using the code below: Now when you click on the compare or delete selected rows, the actions should happen correctly. We are only keeping the end of the id which contains the number of the row to delete. One modal to modify them all. Now, we want to be able to modify a given line of the data. To do see, we will pop a modal with text field when the user click on the modify button. The modal to modify a row The datatable inside has the following structure: How to catch these new fields? Now that the fiels are created Shiny need to be able to detect them and use them as an input. Then, it is put in a data. Finally, we can get the proper row to change by using the last modify button the user clicked on. Here we are, the apps should work perfectly now! You can find the code HERE. Thanks for reading the tutorial, Antoine.

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 4 : Creating Interactive data visualization using Shiny App in R

Mar 08, Â· Change the color of action button in shiny. Ask Question. up vote 23 down vote favorite. 7. I am trying to change the color of the action button from gray to orange.

The package comes with eleven built-in examples that each demonstrate how Shiny works. Each of these examples is a self-contained app. Users can change the number of bins with a slider bar, and the app will immediately respond to their input. To run Hello Shiny, type: It is defined in a source file named ui. Below is the ui. R script for the Hello Shiny example. R script contains the instructions that your computer needs to build the app. Here is the server. R file for the Hello Shiny example. The expression is wrapped in a call to renderPlot to indicate that: It does some calculations and then plots a histogram with the requested number of bins. Try to develop a feel for how the app works. Your R session will be busy while the Hello Shiny app is active, so you will not be able to run any R commands. To get your R session back, hit escape or click the stop sign icon found in the upper right corner of the RStudio console panel. At a minimum, an app has ui. R files, and you can create an app by making a new directory and saving the ui. R file inside it. Each Shiny app will need its own unique directory. You can run a Shiny app by giving the name of its directory to the function runApp. The code above assumes that the app directory is in your working directory; in such case, the file path is just the name of the directory. Alternatively, you can also launch that app by calling runApp system. The project will start with ui. R containing the familiar code from the Hello Shiny app. To launch your app, run in the R console:

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 5 : Shiny - actionButton

Demos. You can check out a demo Shiny app that lets you play around with some of the functionality that shinyjs makes available, or have a look at a very basic Shiny app that uses shinyjs to enhance the user experience with very minimal and simple R code.

Restrict access to previous data to admins only Motivation Last year I was fortunate enough to be a teaching assistant for STAT â€" a course at the University of British Columbia, taught by Jenny Bryan , that introduces R into the lives of student scientists. It was especially special to me because just 12 months prior, that course taught me how to write my first line in R. To facilitate communication with the students, we wanted to gather some basic information from them, such as their prefered name, email, and Twitter and Github handles. I was given the task of developing this shiny app, which was a great learning experience. You can view the original code for that app on GitHub or visit the app yourself to see it in action. The idea of recording user-submitted form data can be applied to many differente scenarios. Seeing how successful the previous app was for our course, we decided to also collect all peer reviews of assignments in a similar shiny app. This worked great for us â€" you can see the original code on GitHub or try the app out yourself. You can see the result of this tutorial on my shiny server and the corresponding code on GitHub. It looks like this: The main idea is simple: Sounds simple, and it is! In this tutorial each response will be saved to a. To see all submissions that were made, we simply read all csv files and join them together. When using Shiny Server Pro or paid shinyapps. Note about persistent storage One major component of this app is storing the user-submitted data in a way that would allow it to be retrieved later. This is an important topic of its own, and in a few days I will write a detailed post about all the different storage options and how to use them. In this tutorial I will use the simplest approach for saving the data: Using the local filesystem in shinyapps. You can get a bit more information about why shinyapps. Build the basic UI inputs I generally prefer to split shiny apps into a ui. R file with an additional helpers. Create a new file named app. R and copy the following code into it to build the input elements. After saving this file, you should be able to run it either with shiny:: The app simply shows the input fields and the submit button, but does nothing yet. We need to use shinyjs for that, so you need to add a call to shinyjs:: In the global scope above the definition of shinyApp, outside the UI and server code , define the mandatory fields: The condition is whether or not all mandatory fields have been filled. To calculate that, we can loop through the mandatory fields and check their values. Add the following code to the server portion of the app: Show which fields are mandatory in the UI If you want to be extra fancy, you can add a red asterisk to the mandatory fields. For example, textInput "name", labelMandatory "Name" , "". The complete code so far should look like this it might be a good idea to just copy and paste this, to make sure you have the right code: First we need to define a what input fields we want to store and b what directory to use to store all the responses. I also like to add the submission timestamp to each submission, so I also want to define c a function that returns the current time as an integer. Next we need to have a way to gather all the form data plus the timestamp into a format that can be saved as a csv. We can do this easily by looping over the input fields. Add the following reactive expression to the server: When saving the user responses locally to a file, there are two options: The first approach might sound like it makes more sense, but I wanted to avoid it for two reasons: Secondly, this approach is not thread-safe, which means that if two people submit at the same time, one of their responses will get lost. So I opted to use the second solution â€" each submission is its own file. It might seem weird, but it works. However, instead of having turly random characters in the filename, I went a slightly different way: I make the filename a concatenation of the current time and the md5 hash of the submission data. This way the only realistic way that two submissions will overwrite each other is if they happen at the same second and have the exact same data. Here is the function to save the response add to the server: If you get an error when saving, make sure the responses directory exists and you have write permissions. By default, all apps are run as the shiny user, and that user will probably not have write permission on folders you create. You should either add write permissions to shiny, or change the running user to yourself. See more information on how to do this in this post. Better user feedback while submitting and on error Right now there

is no feedback to the user when their response is being saved and if it encounters an error, the app will crash. We want to reverse these actions when saving the data is finished. If an error occurs while saving the data, we want to show the error message. All these sorts of actions are why shinyjs was created, and it will help us here. Remember that all the responses are saved locally, so you can also just open the files manually or use any approach you want to open the files. Add table that shows all previous responses Note: First we need to add a dataTable placeholder to the UI add it just before the form div, after the titlePanel: You can define it in the global scope. Add the following to the server: Add ability to download all responses It would also be very handy to be able to download all the reponses into a single file. Restrict access to previous data to admins only The only missing piece is that right now everyone will see all the responses, and you might want to restrict that access to admins only. This is only possible if you enable authentication, which is available in Shiny Server Pro and in the paid shinyapps. Without authentication, everyone who goes to your app will be treated equally, but with authentication you can give different people different usernames and decide which users are considered admins. The first thing we need to do is remove all the admin-only content from the UI and only generate it if the current user is an admin. The following code ensures that for non-admins, nothing gets rendered in the admin panel, but admins can see the table and download button add this to the server: If there is no authentication, it will be NULL. You are now ready to create forms with shiny apps. You can see what the final app code looks like on GitHub with a few minor modifications , or test it out on my shiny server. Related Share Tweet To leave a comment for the author, please follow the link and comment on their blog:

Chapter 6 : Mimicking a Google Form with a Shiny app | R-bloggers

Designed to be used from HTML and R: Shiny user interfaces can either be written using R code (that generates HTML), or by writing the HTML directly. A well-designed Shiny input component will take both styles into account: offer an R function for creating the component, but also have thoughtfully designed and documented HTML markup.

Today, we are going to see how to add buttons in each rows of a datatable to delete, edit or compare it with other rows. The app is live here. Buttons to delete, edit and compare Datatable rows Here is what we want to achieve: The final application As you can seen, the final application offer a lot of functionalities, like selecting row with checkboxes, comparing, deleting them and also modifying them. Even though you can select the rows via the classical datatable functions, checkboxes make it more visual for the users he knows there is something it can do. In addition to this, the button are a clear call to action so that the users do not have to serach how to do the different actions. It all begins with a data. R library shinydashboard library data. The apps create fakes yearly sales data for 10 brands, with a name, an email and a fake estimated growth. The data is purely cosmetic. The body contains a title, three buttons which are grouped using the html div btn-group. When Shiny generates the html code, the div will encapsulate the action buttons which will be showed as grouped. Storing the table in a reactive value: Since we will want to perform action on the data with buttons and eventObserver the modifications of the data will be done with side effects. Hence reactiveValues are a must. On which we added some buttons Now, lets add the buttons to our datatable. Basically, they will be added through html code and the datatble will be asked not to escape strings. The checkboxes will be used to select the rows on which the user want to perform some actions Deletion, comparison. Since they all have the same name it will be easy to access their value through JS. Each row will contain a group of two buttons, one to delete the row and the other one to modify it. Again, the app will access the row to delete and the action to do through the button ID. But these buttons were doing nothing. No matter you click on the buttons or not, nothing will happen. They need to be linked with the Shiny App. Now, to be able to compare the selected row and to delete them, we need to create a new Shiny input with the list of selected checkbox. Then when the user is clicking on an input, the script runs a function. We are storing all the checkboxes and we create an empty arrow that will be filled with the id of ticked checkbox. To compare the rows, the code is similar. The modal is generate using the code below: Now when you click on the compare or delete selected rows, the actions should happen correctly. We are only keeping the end of the id which contains the number of the row to delete. One modal to modify them all. Now, we want to be able to modify a given line of the data. To do see, we will pop a modal with text field when the user click on the modify button. The modal to modify a row The datatable inside has the following structure: How to catch these new fields? Now that the fiels are created Shiny need to be able to detect them and use them as an input. Then, it is put in a data. Finally, we can get the proper row to change by using the last modify button the user clicked on. Here we are, the apps should work perfectly now! You can find the code HERE. Thanks for reading the tutorial, Antoine You can follow me on Twitter:

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 7 : File download example for R Shiny - calendrierdelascience.com

An extensive tutorial on how to add buttons to add, modify and delete rows in a data table in R Shiny. It also covers group selection and plotting.

Try my new interactive online video course: It lets you perform common useful JavaScript operations in Shiny applications without having to know any JavaScript. Important note The package has improved a lot since writing this post. I highly recommend you stop reading this page and instead go the shinyjs website. Availability shinyjs is available through both CRAN install. Motivation Shiny is a fantastic R package provided by RStudio that lets you turn any R code into an interactive webpage. Now I can simply call hide "panel" or disable "button". I was lucky enough to have previous experience with JS so I knew how to achieve the results that I wanted, but for any Shiny developer who is not proficient in JS, hopefully this package will make it easy to extend the power of their Shiny apps. There are arguments that control the animation as well, though animation is off by default. Was originally developed with the sole purpose of running a shinyjs function when an element is clicked, though any R code can be used. Basic use case - working example You can view the final Shiny app developed in this simple example here. Here is what that app would look like Now suppose we want to add a few features to the app to make it a bit more user-friendly. This is required to set up all the JavaScript and a few other things. To do that, replace p "Timestamp: Some users may find it hard to read the small text in the app, so there should be an option to increase the font size First, we need to add checkbox to the UI checkboxInput "big", "Bigger text", FALSE In order to make the text bigger, we will use CSS. Altenatives using native Shiny shiny:: I mostly intended for this function to be used to change the text, though it can also be used to add HTML elements. There are many Shiny functions that allow you to change the text of an element. I still use the Shiny functions often, but I find html useful as well. This means that observeEvent can be used for any input element not only clickable things , but onclick can be used for responding to a click on any element, even if it is not an input tag. This would not be used for most basic apps, but for more complex dynamic apps it might come in handy. This would mean that you could just write a JavaScript function, tell shinyjs where to find it, and then shinyjs will do some magic to let you call that function as if it was regular R code. Known issues There are some input tags that shiny wraps in extra HTML, and this can interfere with shinyj functions. For example, using selectInput "foo" by default uses selectize JS, which hides the real select box that has id foo and instead makes a more visually appealing box. The previous workaround works for some shinyjs functions such as hide and show, but not all. Contributions If anyone has any suggestions or feedback, I would love to hear about it. If you have improvements, feel free to make a pull request.

Chapter 8 : Shiny - Using Action Buttons

This is because R is busy-your R session is currently powering a Shiny app and listening for user interaction (which won't happen because the app has nothing in it yet). Click the stop button to stop the app, or press the Escape key.

The checkboxes will be used to select the rows on which the user want to perform some actions Deletion, comparison. Since they all have the same name it will be easy to access their value through JS. Each row will contain a group of two buttons, one to delete the row and the other one to modify it. Again, the app will access the row to delete and the action to do through the button ID. But these buttons were doing nothing. No matter you click on the buttons or not, nothing will happen. They need to be linked with the Shiny App. Now, to be able to compare the selected row and to delete them, we need to create a new Shiny input with the list of selected checkbox. Then when the user is clicking on an input, the script runs a function. We are storing all the checkboxes and we create an empty arrow that will be filled with the id of ticked checkbox. To compare the rows, the code is similar. The modal is generate using the code below: Now when you click on the compare or delete selected rows, the actions should happen correctly. We are only keeping the end of the id which contains the number of the row to delete. One modal to modify them all. Now, we want to be able to modify a given line of the data. To do see, we will pop a modal with text field when the user click on the modify button. The modal to modify a row The datatable inside has the following structure: How to catch these new fields? Now that the fiels are created Shiny need to be able to detect them and use them as an input. Then, it is put in a data. Finally, we can get the proper row to change by using the last modify button the user clicked on. Here we are, the apps should work perfectly now! You can find the code HERE. Thanks for reading the tutorial, Antoine.

DOWNLOAD PDF BUTTON FOR R SHINY

Chapter 9 : html - R shiny: Add weblink to actionButton - Stack Overflow

Shiny Cheat Sheet learn more at calendrierdelascience.com Shiny Updated: 6/14 widget's current value in server.R with input$ Action button checkbox.

Get a feel for the wide range of things you can do with Shiny. Shiny app basics Every Shiny app is composed of a two parts: In Shiny terminology, they are called UI user interface and server. The UI is responsible for creating the layout of the app and telling Shiny exactly where things go. If you look at the app we will be building , the page that you see is built with the UI code. The UI is responsible for creating these controls and telling Shiny where to place the controls and where to place the plot and table, while the server is responsible for creating the actual plot or the data in the table. Create an empty Shiny app All Shiny apps follow the same template: It initializes an empty UI and an empty server, and runs an app using these empty parts. Copy this template into a new file named app. R in a new folder. A few things you should keep in mind: It is very important that the name of the file is app. R, otherwise it would not be recognized as a Shiny app. That line needs to be the last line in your file. It is good practice to place this app in its own folder, and not in a folder that already has other R scripts or files, unless those other files are used by your app. After saving the file, RStudio should recognize that this is a Shiny app, and you should see the usual Run button at the top change to Run App. Click the Run App button, and now your app should run. Click the stop button to stop the app, or press the Escape key. You can run that command instead of clicking the button if you prefer. However, do not place the runApp function inside the shiny app code! Try running the empty app using the runApp function instead of using the Run App button. If you want to break up your app into these two files, you simply put all code that is assigned to the ui variable in ui. R and all the code assigned to the server function in server. Note that if you use this method instead of having one app. Try making a new Shiny app by creating the two files ui. Rememeber that they have to be in the same folder. Also remember to put them in a new, isolated folder not where your app. If you do this, RStudio will let you choose if you want a single-file app app. R or a two-file app ui. RStudio will initialize a simple functional Shiny app with some code in it. They provide a direct link to download a csv version of the data, and this data has the rare quality that it is immediately clean and useful. You can view the raw data they provide, but I have taken a few steps to simplify the dataset to make it more useful for our app. I removed some columns, renamed other columns, and dropped a few rare factor levels. Download it now and place this file in the same folder as your Shiny app. Make sure the file is named bcl-data. Add a line in your app to load the data into a variable called bcl. Try to run the app to make sure the file can be loaded without errors. If you want to verify that the app can successfully read the data, you can add a print statement after reading the data. You can place the following line after reading the data: How big is it, what variables are there, what are the normal price ranges, etc. This is usually the first thing you do when writing a Shiny app - add elements to the UI. The entire UI will be built by passing comma-separated arguments into the fluidPage function. By passing regular text, the web page will just render boring unformatted text. Add several more strings to fluidPage and run the app. Nothing too exciting is happening yet, but you should just see all the text appear in one contiguous block. There are also functions that are wrappers to other HTML tags, such as br for a line break, img for an image, a for a hyperlink, and others. All of these functions are actually just wrappers to HTML tags with the equivalent name. You can add any arbitrary HTML tag using the tags object, which you can learn more about by reading the help file on tags. Notice the formatting of the text and understand why it is rendered that way. For people who know basic HTML: Overwrite the fluidPage that you experimented with so far, and replace it with the simple one below, that simply has a title and nothing else. Look at the documentation for the titlePanel function and notice it has another argument. Use that argument and see if you can see what it does. It provides a simple two-column layout with a smaller sidebar and a larger main panel. Add the following code after the titlePanel sidebarLayout sidebarPanel "our inputs will go here" , mainPanel "the results will go here" Remember that all the arguments inside fluidPage need to be separated by commas. Add some UI into each of the two panels sidebar panel and main panel and see how your app now has two columns. To convince yourself of this, look

at the output when printing the contents of the ui variable. Add inputs to the UI Inputs are what gives users a way to interact with a Shiny app. Shiny provides many input functions to support many kinds of interactions that the user could have with an app. For example, textInput is used to let the user enter text, numericInput lets the user select a number, dateInput is for selecting a date, selectInput is for creating a select box aka a dropdown menu. All input functions have the same first two arguments: The inputId will be the name that Shiny will use to refer to this input when you want to retrieve its current value. It is important to note that every input must have a unique inputId. The label argument specifies the text in the display label that goes along with the input widget. Every input can also have multiple other arguments specific to that input type. The only way to find out what arguments you can use with a specific input function is to look at its help file. Read the documentation of? Experiment with the different arguments. Run the app and see how you can interact with this input. Then try different inputs types. The most sensible types of input for this are either numericInput or sliderInput since they are both used for selecting numbers. To create a slider input, a maximum value needs to be provided. By looking at the documentation for the slider input function, the following piece of code can be constructed. Run the code of the sliderInput in the R console and see what it returns. Change some of the parameters of sliderInput , and see how that changes the result. The same is true in our app, we should be able to choose what type of product we want. For this we want some kind of a text input. We could either use radio buttons or a select box for our purpose. It should look like this: We should add one last input, to select a country. The most appropriate input type in this case is probably the select box. Look at the documentation for selectInput and create an input function. If you followed along, your entire app should have this code: Add placeholders for outputs After creating all the inputs, we should add elements to the UI to display the outputs. Outputs can be any object that R creates and that we want to display in our app - such as a plot, a table, or text. Each output needs to be constructed in the server code later. Shiny provides several output functions, one for each type of output. Similarly to the input functions, all the ouput functions have a outputId argument that is used to identify each output, and this argument must be unique for each output. Since we want a plot, the function we use is plotOutput. Add the following code into the mainPanel replace the existing text: To remind yourself that we are still merely constructing HTML and not creating actual plots yet, run the above plotOutput function in the console to see that all it does is create some HTML. To get a table, we use the tableOutput function. Here is a simple way to create a UI element that will hold a table output: Implement server logic to create outputs So far we only wrote code inside that was assigned to the ui variable or code that was written in ui. Now we have to write the server function, which will be responsible for listening to changes to the inputs and creating outputs to show in the app. You must define these two arguments! Both input and output are list-like objects. As the names suggest, input is a list you will read values from and output is a list you will write values to. We need to write code in R that will tell Shiny what kind of plot or table to display.