Generalizable Shiny App: Interactive Data Visualization

**Introduction**

Traditionally, data are visualized in static plots. With the prevalence of R, Python, and other customizable data science software, data owners are interested in exploring customizable dashboards for monitoring the status of the data collection process. For rigorous statistical analysis and survey specialization, it creates challenges for using traditional platforms, such as Excel and Tableau. Excel can generate static plots, but the raw data must be analyzed in another statistical package first. Similarly difficult, Tableau can create refined dynamic plots, but the specialized nature of some work often require more sophisticated back-end work that cannot be done in the program itself. R Shiny, on the other hand, provides an innovative solution to the problems presented above. Creating an interactive data visualization app with R Shiny allows data owners to visualize dynamic plots that require specialized research statistics.

Thus, the purpose of the interactive data visualization app was to explore the feasibility of a generalizable prototype for dynamically visualizing survey data. This prototype could provide a skeleton for others to recreate other data visualization apps without starting from scratch. For users without R experience, they can use the interactive app simply by running the app. Furthermore, since R Shiny can be used for a lower cost than other data visualization tools, it offers an economic advantage as well.

In order to explore the feasibility and applicability of the project, the interactive data visualization app focuses on visualizing line graphs and bar charts. When the end user has ensured that the data input source has been formatted correct, he or she can choose, without even opening the R code, to visualize the data with either a single interactive plot per page or a dynamic number of interactive plots per page.

While the end user would not need to open any of the Shiny code, it is expected that a developer has general familiarity with R and Shiny. It is strongly encouraged to review an introductory tutorial found here: <https://shiny.rstudio.com/tutorial/>. While most of the programming has been completed, understanding the code can be useful for personalizing a dashboard to a specific project’s needs. Shiny is an extension of R, so this brief pre-test quiz can be a good measure for readiness in R that would be recommended for Shiny: <https://shiny.rstudio.com/tutorial/quiz/>. In the next sections, each of the four key component areas (Excel Input File, helperFunctions.R, ui.R, and server.R) of this interactive data visualization app are described in detail.

**Excel Input File**

In an Excel file, each spreadsheet should correspond to one tab panel in the Shiny application. Two example templates can be found for R-indicators and American Community Survey (ACS) quality measures, which are the input files Mock\_R-ind\_input.xlsx and ACS\_Quality\_Input.xlsx.

Each spreadsheet corresponds to a different tab panel on the Shiny. For instance, the first spreadsheet would correspond to the first tab panel. Please note that the spreadsheet must follow a specific naming pattern as listed below.

***Spreadsheet Naming System***

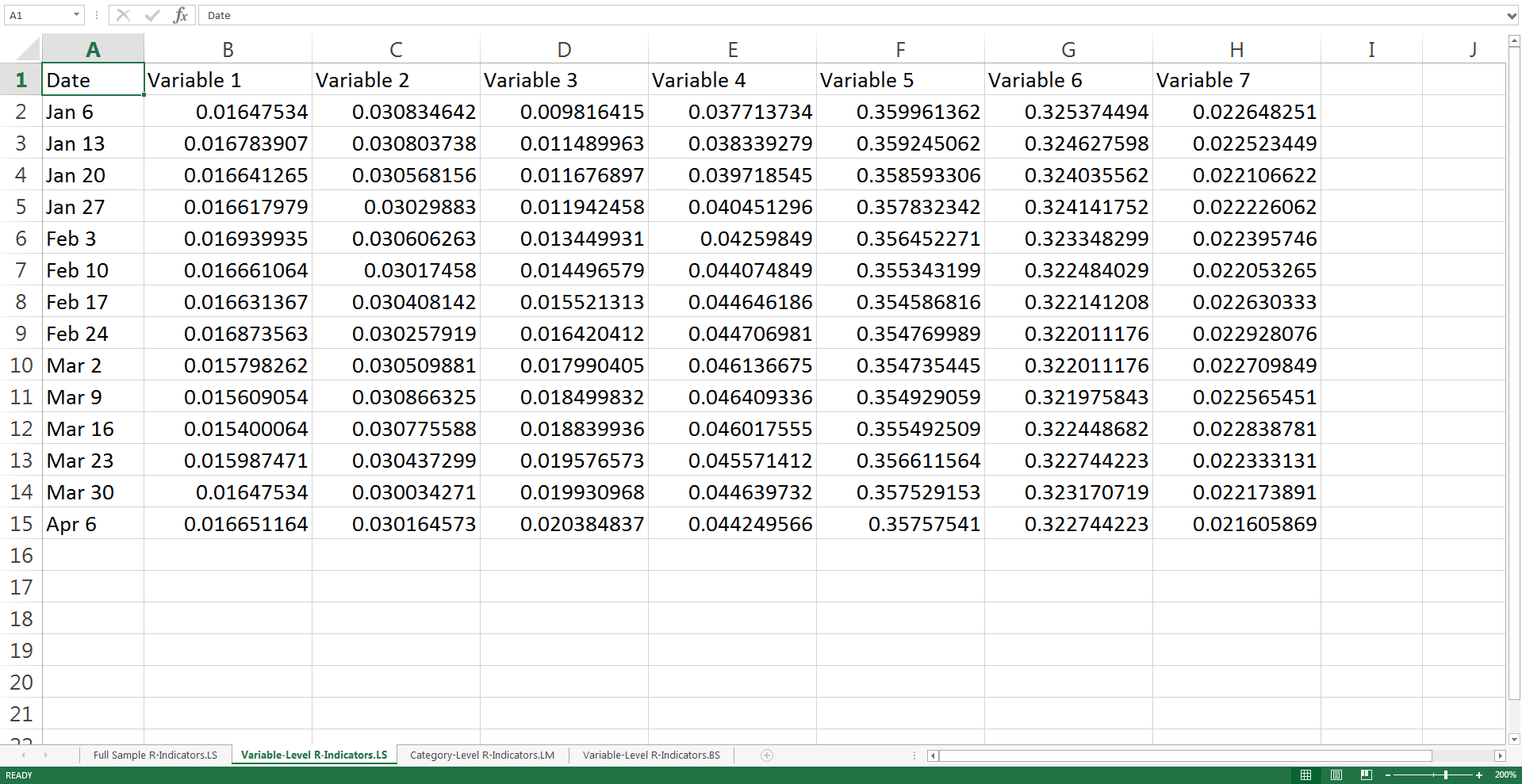
|  |  |  |
| --- | --- | --- |
| **Type of Graph** | **Suffix of Spreadsheet Name** | **Example** |
| Single Line Graph | “.LS” | “InputSheet1.LS” |
| Multiple Line Graph | “.LM” | “InputSheet2.LM” |
| Single Bar Graph | “.BS” | “InputSheet3.BS” |

Note that both upper-case and lower-case letters work for end of spreadsheet name. For example, both InputSheet1.LS and InputSheet1.ls will work.

***Line Graphs***

The first column of each spreadsheet should contain the y-axis variable. For example, in a time-series graph, the first column would be week, day, or year. Afterwards, each column should contain the series and its associated values. The first cell of every column should be the time series name. The sheet name must end in “.LS”.

For making multiple plots appear on one page, follow the instructions as above. After finishing with the data for the first plot, leave an entire row blank. Then, add the data for the second plot. Continue in this manner until all datasets are completed. The sheet name must end in “.LM”. Figure 1 shows an example of single line graph Excel formatting. Figure 2 shows an example of multiple line graph Excel formatting.

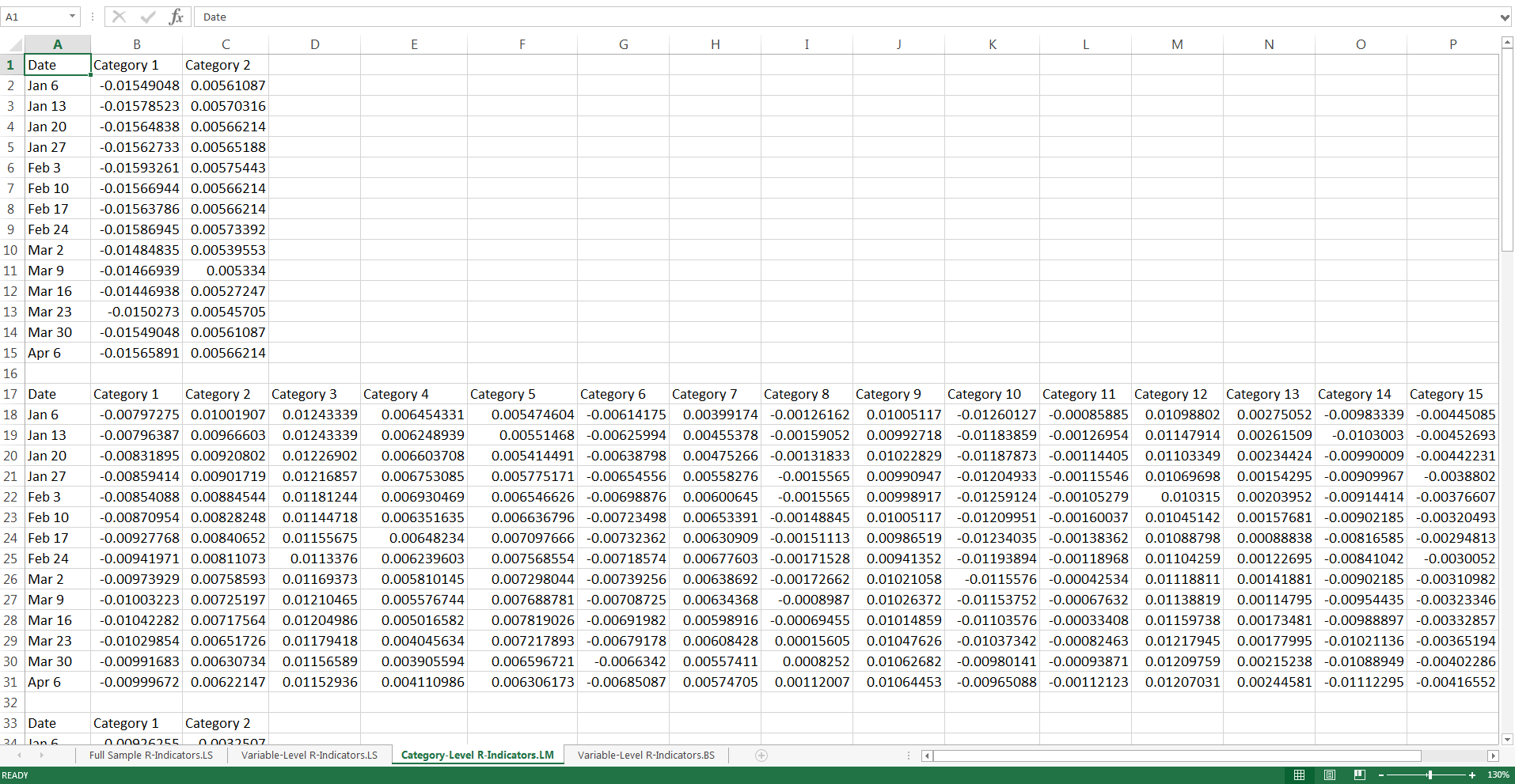


Other columns – time series

First column – Time Unit

**Figure 1: Single line graph to appear per page**

**(Additional Note: Spreadsheet ends in “.LS”)**

****

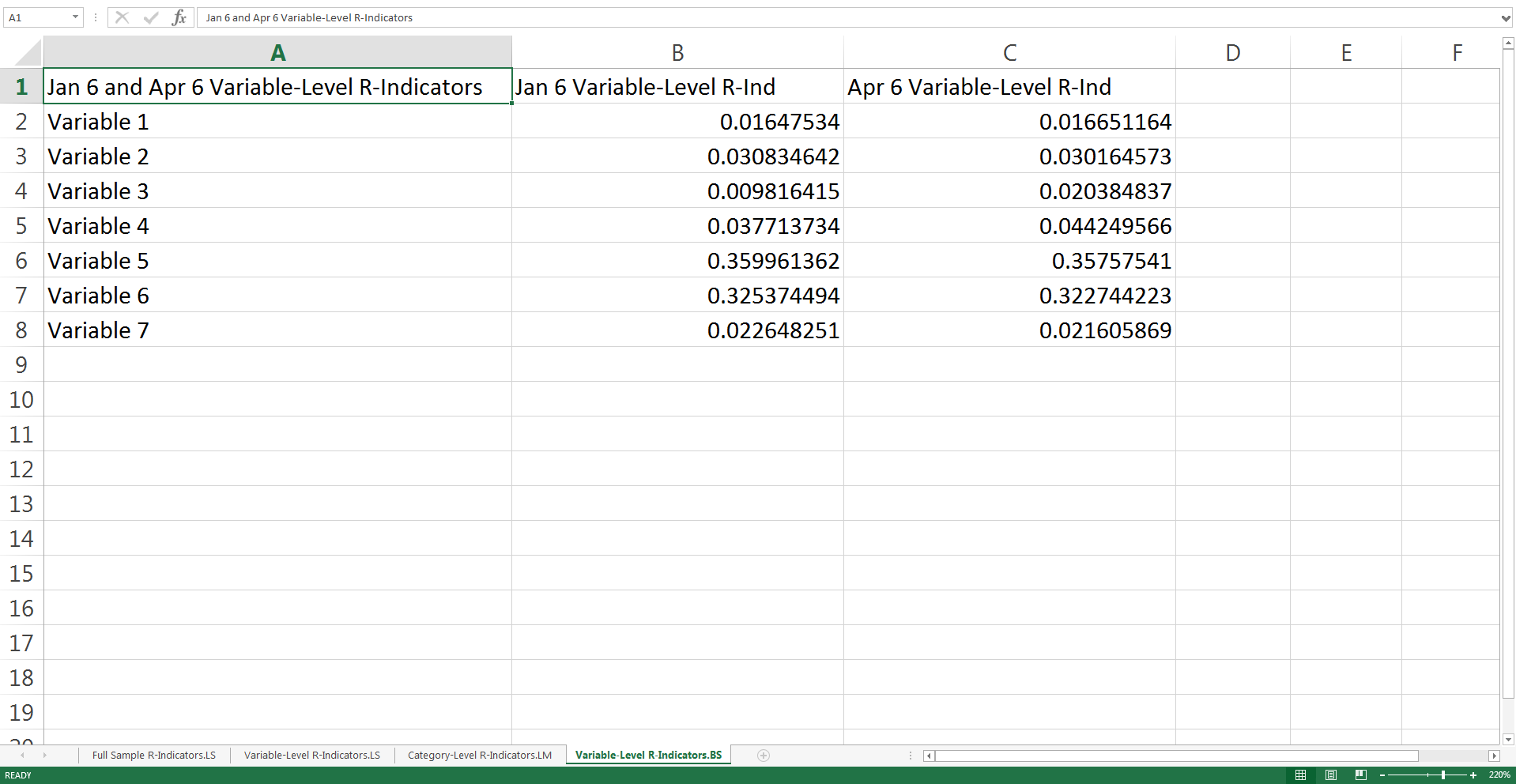
Skip a row between datasets

**Figure 2: Multiple line graph to appear per page**

**(Additional Note: Spreadsheet name ends in “.LM”)**

***Bar Charts***

The first column of each spreadsheet should contain the names of each bar (i.e. Category 1, Category 2, Category 3, etc.). Afterwards, the next columns should each represent measures of interest. Each column should have data that corresponds to the name of each of the bars from the first column. Figure 3 gives an example of the input file. The sheet name must end in “.BS”.

****

Other columns – data for each bar

First column – Group names for the bars

**Figure 3: Single bar plot to appear per page**

**(Additional note: Spreadsheet name ends in “.BS”)**

**R Files**

**helperFunctions.R Structure**

The following functions were called upon to automate the code and prevent redundancies.

* timeClean(dataset) : This function converts the time variable from a character to a factor. Furthermore, it keeps the original order as intended, so changing the class of the variable will not affect the visualization.
* dataDownload(dataset) : The functions returns a file to download. This was not shown in the December demonstration, but previous versions of the Shiny app have allowed for the user to download only the relevant data files.
* read\_excel\_allsheets(filename) : This reads in datasets from all except sheets. The input is the filename, including the directory. The output is a three-tiered list of organized data according to (1) graph type; (2) Excel sheet; (3) single/multiple datasets.
* multiDatasets\_DFlist(dataset): This splits one dataset into smaller datasets and forms a list. It calls on the splitDataset() function, which recognizes that there is an entirely empty row in the dataset. The input is a dataset from a single Excel sheet and then creates a list of the split datasets as the output. The list is what allows the app to separate the data into different graphs.
* splitDataset(dataset): This allows for the use of multiple datasets in 1 Excel sheet. The function recognizes when to subset the data by identifying when an empty row appears. It is called upon by the mulitDatasets\_DFlist() function.
* get\_plot\_output\_list(selectVar\_indices) : This function allows for a dynamic number of graphs on a single page. Its parameter `selectVar\_indices` is a list of indices of user-selected vairables. The output is a list of plot output objects for each graph.
* dataMeltFun(colVars, dataset, multiplePlots = FALSE) : This prepares the data frame for ggplot2 and Plotly by melting the function.
* LG\_ggPlotly (dataset\_long, denseData = FALSE, width = 800, height = 600, title = “”) : This function first creates the plot in ggplot2 and then converts it to an interactive Plotly graph. It takes in `dataset\_long` as the ggplot2-ready graph. ‘denseData’ will change the x-axis labeling to make dense data frames easier to read though it defaults to normal data. `height` and `width` affect the size of the plot, but they have a default dimension of 600 by 800. The ‘title’ default is blank, but it can be edited to a desired title.
* BC\_ggplotly(dataset\_list, data\_input, var\_input, yaxis\_input) : This function first creates the plot in ggplot2 and then converts it to an interactive Plotly graph. It takes in `dataset\_list` as list of datasets used for the Shiny. ‘data\_input` allows the selection of the data input. `var\_input` then allows for the selection of the specific variable input. Finally, the parameter `yaxis\_input` allows the y-axis of the bar chart to be changed.
* tableFun(dataset\_long) : This function takes the melted original data frame and reconverts it back to the original format. This is useful because it allows for the table to be reactive. This was not shown in the December 2017 demonstration, but previous versions of the Shiny app have allowed for the user to view the data in a table format.

**ui.R Structure**

The UI is actually constructed together from 5 different tabs: Home, Data, Line Graphs, Bar Charts, and Help. The UI section begins around line 31.

* Home: This contains a welcome page for the Shiny app.
* Data: This is the menu for uploading the data. It contains a UI element for inputting a file into the Shiny app.
* Line Graphs: This contains the UI elements for the various controls and plots to appear. It is in a sidebar layout with the controls on the sidebar and the graphs on the main panel.
* Bar Charts: This contains the UI elements for various controls and plots to appear. This contains the UI elements for the various controls and plots to appear. It is in a sidebar layout with the controls on the sidebar and the graphs on the main panel.
* Help: This is a tab panel that can offer the user the ability to access additional resources for help.

**server.R Structure**

The server contains the code for controlling the other side of the bidirectional communication between the UI and the server. This usually handles the events that occur as a result as a change of the user’s input. The data structure of the datasets is a three-tiered list: (1) graph-type, (2) Excel-Sheet, and (3) Single/Multiple Dataset.

The code can be divided into three parts:

Reactive list of datasets

* `dataset\_list` is a reactive object that creates a list of sheet dataframe from an input Excel file. It reads in bar charts and line graphs data differently. The input is the user-selected Excel file when they are inputting the Excel file while the Shiny app is running.
* `LG\_list` is a reactive function that filters only the line graph elements of the `dataset\_list`.
* `LG\_sheet` is a reactive function that filters only the line graph elements that the user selects when using the Shiny app.
* `selected\_inputIndex` is a reactive function that is dependent on the specific datasets in which the user is interested. Only the selected indices will update.
* `BC\_list` is a reactive function that filters only the bar charts in the `dataset\_list`.

Selection outputs

* ` output$select\_LG\_dataset` lets the user select the line graph dataset(s) (one Excel line graph sheet)
* ` output$select\_LG\_variable` lets the user select the line graph variable(s)
* ` output$select\_LG\_subVariable` lets the user select line graph sub-variable(s)
* ` output$select\_BC\_dataset` lets the user select the bar chart dataset(s) (one Excel bar chart sheet)
* ` output$select\_BC\_var` lets the user select the bar chart variable(s)
* ` output$yaxis\_singleBC` lets the user control the y-axis of the single bar chart

Plotting of the Graphs

* ` output$LGgraphs` has the UI elements for the plotting of the line graphs in the main panel for the Line Graphs tab. It is plotted with the Plotly package.
* ` output$single\_LGplot` renders the ggplot that is made in `plotInput`. The data input for `plotInput` is a reactive function of the data that are selected by the user for the line graph.
* `output$multiPlots` makes the dynamic number of plots appear as desired. First, it calls on the reactive function `updatePlots` that aligns the variable input and the datasets. Then the function `get\_plot\_output\_list(selected\_inputIndex())` updates the number of plots that appear in the main panels.
* `output$BCgraphs` creates a bar chart by calling on the `BC\_ggplotly` function.
* Additional work would allow for the automatic creation of multiple bar charts on a single page. The code occasionally worked in developing the interactive data visualization app, but there was a bug that made the bar charts appear squished on different computers.

**Further Work and Limitations**

For additional dashboards, this can be applied to other popular templates. Examples might include statistics by key variables (dot plots or bar charts). Finally, since it is important to make the Shiny app accessible to all potential users, a video tutorial could be helpful on how to modify the Shiny app. This would ideally quell any fears that any code-anxious user may have about R if they wanted to further use the code.

Currently, the code is being loaded into an Excel file. However, additional code could be written to directly connect a data source to the Shiny app. This would then not require the user to manually create their own input file. Given the nature of the targeted investment of the project, it was not possible to write specialized code for the direction connection of the data source to the Shiny app.