User guide for Shiny interactive visualizer template

Cristian Homescu

May 2020

Abstract

This document is an user guide for Shiny interactive visualizer template based on semantic.dashboard and shiny.semantic R packages (which incorporates SemanticUI and its newer community fork FomanticUI). It provides an overview, the main concepts and the rationales for the corresponding folder and code structure. It also identifies specific steps:

- to change visual elements (interactive plots and tables) within existing tabs and subtabs
- to modify Shiny actions within existing tabs and/or subtabs
- to add visual elements to existing tabs and/or subtabs
- to incorporate new Shiny actions to existing tabs and/or subtabs
- to add new tabs and/or subtabs
- to create another visualizer starting from the template

Contents

1 Overview of the Shiny visualizer template . . . 2
1.1 Main concepts . . . 2
1.2 Required Software (and versions) . . . 2
1.3 Overall code structure . . . 2
1.4 Working directories . . . 2
1.5 Running the code to get the interactive visualizer showing up on your computer . . . 2
1.6 Deploying the interactive visualizer on website shinyapps.io . . . 3
2 Main UI and Server functions . . . 3
2.1 UI function . . . 3
2.2 Server function . . . 3
3 Implementation details regarding external data and functions . . . 3
3.1 Types of input data files . . . 3
3.2 When data and functions are stored in files outside Shiny working directory . . . 4
3.3 Default values for inputs and other variables . . . 4
3.4 Icons and other useful info for SemanticUI . . . 4
4 How to create another visualizer starting from the template . . . 4
5 How to add a new tab to visualizer . . . 5
6 How to add a new subtab to an existing tab in visualizer . . . 5
7 How to add new visual outputs to a visualizer tab . . . 5
7.1 Procedure steps . . . 6
7.2 Going through an example . . . 6
8 How to add new inputs to visualizer . . . 7
9 How to add new Shiny actions to visualizer . . . 7
10 Snapshots of visualizer template . . . 8

1 Overview of the Shiny visualizer template

1.1 Main concepts

The visualizer consists of multiple tabs and subtabs, with each tab encapsulated using modules (each one implemented in its own file). We rely on Shiny reactive values to use as arguments to functions within the modules. Each function in the code is either a collection of function calls or an implementation of a specific task, making the code simpler to understand and to debug.

This approach is meant to significantly simplify the development, enhancement and maintenance of codes used to construct the Shiny visualizer.

1.2 Required Software (and versions)

You would need RStudio version 1.3.959 (or newer) and R version 3.6.3 (not another version).

To download and set up the appropriate versions of required R packages, you would need to run file **SetupPackages.R**, which contains a snapshot date of May 25, 2020.

1.3 Overall code structure

The file to run is **Run_Shiny_Viz.R**.

The rest of the code is within folder Viz_Shiny_Modules, which contains five subfolders:

- Inputs
- Server_Modules
- UI_Modules
- Utils
- www

The Shiny visualizer is created through an UI function and an Server function, implemented in file **app.R** (in folder Viz_Shiny_Modules)

Each tab of the visualizer is implemented through two corresponding files: one in folder Server_Modules and one in folder UI_Modules.

The www folder contains any static images (such as snapshots or background images) which are to be incorporated within Shiny visualizer. As an example it contains the file **LogoGraphite.png**, which is displayed in the title of the dashboard.

The Inputs folder contains two files, one containing saved data and the other one saved functions (if necessary), which are used within Shiny.

The Utils folder contains functions to save and to retrieve data and functions, and to set default values for input values and other variables.

Additional details for these folders are given in next sections.

1.4 Working directories

One of the challenges with Shiny codes is due to the fact that the working directory changes as soon as we run the Shiny codes: from working directory of the main folder (as defined through an RProj file associated with the main folder) to a subfolder (associated with Shiny) of the main folder.

For the Shiny visualizer template described in this document, the Shiny working directory is the subfolder VizTemplate/Viz_Shiny_Modules

1.5 Running the code to get the interactive visualizer showing up on your computer

Open the RProject file, run **SetupPackages.R**, then run the file **Run_Shiny_Viz.R**.

Data and functions located in files outside Shiny working directory have to be retrieved and stored in files within Shiny working directory. The main function RunShinyViz has an argument **saveDataAndFunctions_RSV** which is set to TRUE or FALSE, controlling whether the operation of "save and retrieve data" has to be performed (either for the first time, or reperformed due to change in data) or not.

The second argument of RunShinyViz main function contains the Shiny working directory.

1.6 Deploying the interactive visualizer on website shinyapps.io

Assuming that the local interactive visualizer (obtained in the previous subsection) has the desired visual features, you can also deploy it ("publish" it) on website shinyapps.io, after creating an account on that website.

To deploy it, you need to run the file **PublishViz.R**, with appropriate changes. For example, you would need to change the inputs of function rsconnect::setAccountInfo (within function SetShinyAppAccount), namely name, token and secret corresponding to account shinyapps.io.

If you want to deploy another visualizer, you may need to add another function, similar to function PublishViz_VizTemplate, and then update accordingly the if-else within function PublishViz.

The deployed visualizer can be viewed in Google Chrome by clicking on a shinyapps.io link (if interested in vieweing it, please reach out to me for the link).

2 Main UI and Server functions

The Shiny visualizer relies on two main functions: an UI function and an Server function, both implemented in file **app.R**

2.1 UI function

Since the visualizer relies on R semantic.dashboard package, the UI function calls 3 functions (implemented in file **UI_Main.R** in folder UI_Modules):

1) DisplayDashboardHeader
2) DisplayDashboardSidebar
3) DisplayDashboardBody

2.2 Server function

The server function fulfills three major tasks:

1) Creates three structures of Shiny reactive values
   - 1.a) structure (named **reactiveUserInputs**) storing the IDs of inputs explicitly passed by the user
   - 1.b) structure (named **reactiveInnerVariables**) storing the variables calculated or updated throughout the code
   - 1.c) structure (named **reactiveTriggers**) storing the IDs of triggers (such as Shiny action buttons or action links) that would activate Shiny actions
2) Observes events to update the reactive values of the 3 structures defined above
3) Renders content in the visualizer (one function per visualizer tab)

The structure of Shiny reactive values is similar to a list, but with special capabilities for reactive programming. When one writes to it, it notifies any reactive functions within the code that depend on that value.

The first two major tasks are implemented through functions in the file **Server_ReactiveVars.R**, while rendering the content is performed through functions implemented in a file corresponding to each tab.

3 Implementation details regarding external data and functions

3.1 Types of input data files

The input files can be of any type that you want (Excel spreadsheets, text files, JSON files, RData or RDS files, files with qs, feather, arrow extensions), provided that there are R functions (or packages) which can read those files within R code. The input datafiles used within the file **SaveInfo.R** are meant only to exemplify the procedure.

3.2 When data and functions are stored in files outside Shiny working directory

One of the main challenges of programming for a Shiny visualizer is that Shiny has a different working directory then the one defined in the RProj file of the project. Moreover, in case we want to "publish" the Shiny visualizer through RStudio Connect or shinyapps.io, we could only use files within the Shiny working directory, otherwise these files cannot be retrieved for packaging into a container.

This challenge is addressed by a combination of two approaches:

1) within the Shiny code use only relative paths which are inside the Shiny working directory

2) for any data files and functions in files outside the Shiny working directory, store them into files located within the Shiny working directory

The operations of saving of data and functions is performed within the file **SaveInfo.R** in the folder Utils. Data and functions located in files outside the Shiny working directory are stored in 2 files in the folder Inputs within the Shiny working directory:

- **SavedDataForShiny.qs**
- **SavedFunctionsForShiny.qs**

The file **GetSavedInfo.R** in folder Utils reads the content of these files, and stores them into two structures which are used within Shiny: **objSavedData** and, respectively, **objSavedFunctions**.

An example of using functions which were stored as elements in the list retrieved from file **SavedFunctionsForShiny.qs** is in function CalcPlotTimeSeries CalcPlotTimeSeries of file **Server__TabAnalysisOne.R**. Instead of using

```
ggplot2::ylab(label = "Value")
```

the code uses

```
funcs_MVSTA1_CPTS[["Func_ggplot_ylab"]](label = "Value")
```

where the function argument named funcs_MVSTA1_CPTS is the list of saved functions retrieved from file **SavedFunctionsForShiny.qs**

3.3 Default values for inputs and other variables

These default values are set within the file **DefaultValues.R** in folder Utils. The functiions within this file are called within file **SaveInfo.R**, and the resulting default values are stored as *DefaultInputs* component or *DefaultInnerVariables* component in the list saved in file **SavedDataForShiny.qs**

3.4 Icons and other useful info for SemanticUI

The visualizer framework is relying on R packages shiny.semantic and semantic.dashboard, which in turn rely on SemanticUI (and its newer community fork FomanticUI).

Semantic is a development framework that helps create beautiful, responsive layouts using human-friendly HTML.

The 2 R packages makes it asy to incorporate the elements of SemanticUI (and FomanticUI), including the icon set, into the interactive visualizer.

4 How to create another visualizer starting from the template

Copy the subfolder in which the template is located (Visualization/SemanticDashboardTemplate) to another subfolder within Visualization folder. Rename the folder according to your preferences.

To exemplify, let's assume that the copy of the SemanticDashboardTemplate folder is renamed TheDashboardFolder. Then one also need to change the path of the argument shinyWorkingDir__RSV. For this exemplification the code

```
shinyWorkingDir__RSV = base::file.path("Visualization","SemanticDashboardTemplate"),
```

becomes

```
shinyWorkingDir__RSV = base::file.path("Visualization","TheDashboardFolder"),
```

5 How to add a new tab to visualizer

Adding a new tab to visualizer is done by creating two files for this tab, one in folder UI_Modules and one in folder Shiny_Modules. To exemplify, let us consider that these files are named as **UI__TabExtra.R** and **Server__TabExtra.R**

Then one needs to

- source file **Server__TabExtra.R** at end of file **GetSavedInfo.R** (in folder Utils), after the other files sourced there
- source file **UI__TabExtra.R** at beginning of file **UI__Main.R**, after the other files sourced there.

DisplayContent is the main function in file **UI__TabExtra.R**, while RenderContent is the main function in file **Server__TabExtra.R**. Then DisplayContent is called within function DisplayDashboardBody of file **UI__Main.R**, while RenderContent is called within function theServer of file **MainAppViz.R**.

The corresponding sidebar is added to function DisplayDashboardSidebar in file **UI__Main.R**. We also add the corresponding tab ID to function SetTabIDs_SidebarMenu in file **UI__Main.R**.

The functions implemented in the file **Server__TabExtra.R** would use as arguments the set (or a subset) of the 5 main data structures utilized within the Shiny code:

- **objSavedData**
- **objSavedFunctions**
- **reactiveUserInputs**
- **reactiveInnerVariables**
- **reactiveTriggers**

We would also need to add subtabs and visual outputs (tables and/or interactive plots) to this new visualizer tab. The steps on how to do this are described in other sections.

Regarding examples in the current visualizer template, the user may consider the tabs Data, Summary, ResultsTwo and AnalysisOne.

6 How to add a new subtab to an existing tab in visualizer

To exemplify, let's consider that the existing tab is named TabOne and thus implemented through files **UI__TabOne.R** and **Server__TabOne.R**. We want to add a subtab named SubTabExtra.

This additional subtab will be incorporated into the visualizer UI through the function DisplayContent of file **UI__TabOne.R**, while its content will be rendered through a newly created function called within function RenderContent of file **Server__TabOne.R**

The functions implemented in the file **Server__TabOne.R** would use as arguments the set (or a subset) of the 5 main data structures utilized within the Shiny code:

- **objSavedData**
- **objSavedFunctions**
- **reactiveUserInputs**
- **reactiveInnerVariables**
- **reactiveTriggers**

We would also need to add visual outputs (tables and/or interactive plots) to this new visualizer subtab. The steps on how to do this are described in other sections.

Regarding examples in the current visualizer template, the user may consider the subtabs of tabs AnalysisOne, ResultsOne and ResultsTwo.

7 How to add new visual outputs to a visualizer tab

When adding new visual outputs (such as interactive plots or tables) we need to consider inputs, datasets, display and rendering of the inputs and outputs. We describe the steps in next subsection.

7.1 Procedure steps

1) Decide what is needed

   1.a) dataset(s)

   1.b) input(s)

   1.c) type of visual output

2) Decide which tab/subtab should contain the new visual output

3) Steps related to dataset

   3.a) if dataset is already used or incorporated within Shiny code for the visualizer, it can be reused for the new visual output

   3.b) if dataset is stored within file(s) located in folder(s) outside Shiny working directory, then follow the steps described in subsection 3.2

4) Steps related to input(s)

   4.a) if input(s) are already used or incorporated within Shiny code for the visualizer, it can be reused for the new visual output

   4.b) if input(s) need to be added to Shiny code, then follow the steps described in section 8

5) Steps related to functions for visual output

   5.a) if the function(s) to obtain the visual output belong to an R package (external or internal), it can be used directly within Shiny code

   5.b) if the function(s) to obtain the visual output are implemented in within file(s) located in folder(s) outside Shiny working directory, then follow the steps described in subsection 3.2

6) Steps related to adding new visual output to a subtab

   6.a) if tab or subtab does not exist yet in Shiny code of the visualizer, create it following the steps described in sections 5 and 6.

   6.b) if subtab already exists within Shiny code of the visualizer, add the new visual output following the steps described in section 7

7.2 Going through an example

To exemplify, let's consider that the existing tab is named TabOne and thus implemented through files **UI__TabOne.R** and **Server__TabOne.R**. We also assume that name of existing subtab is subTabA.

We would need to update some existing functions, and add new functions, in both **UI__TabOne.R** and **Server__TabOne.R**.

We need to add functions in file **Server__TabOne.R** to obtain the required new visual outputs (tables and/or interactive plots). The rendering function will provide the identifier ID of the new visual output, ID which is then used within the UI file. Let's denote this ID as **outputID__theNewVisualOutput**.

To ensure that this visual ouput would change automatically when a corresponding input or variable would change, we use observeEvent to connect this visual output with one of the three structures of reactive variables:

- **reactiveUserInputs**
- **reactiveInnerVariables**
- **reactiveTriggers**

We need to update function DisplayContent in file **UI__TabOne.R**, by incorporating the ID of the corresponding visual component. Depending on the situation, it can be done through function calls such as

- shiny::uiOutput(outputId = "**outputID__theNewVisualOutput**")
- plotly::plotlyOutput(outputId = "**outputID__theNewVisualOutput**")

- DT::dataTableOutput(outputId = "**outputID__theNewVisualOutput**")
- rhandsontable::rHandsontableOutput(outputId = "**outputID__theNewVisualOutput**")
- semantic.dashboard::valueBoxOutput(outputId = "**outputID__theNewVisualOutput**")

If we also need to connect Shiny actions to the new visual outputs, steps on how to do this are described in other sections.

Regarding examples in the current visualizer template, the user may consider the time series plot in second subtab of tab AnalysisOne, the table and the plot in tab AnalysisTwo, the tables in tabs AnalysisThree and ResultsTwo.

8 How to add new inputs to visualizer

The tab Data is currently used for user inputs, and it is likely that the corresponding code needs to be updated. If the new inputs are to be utilized in other tabs, then the codes for those tabs would have to be updated as well.

We need to update function SetValues__UserInputs in file **defaultValues.R**

We also need to update the functions related to the structure reactiveUserInputs:

- function SetInitialValues__ReactiveUserInputs in file **Server__ReactiveVars.R**
- function ObserveEvents__ReactiveUserInputs in file **Server__ReactiveVars.R**

Then perform the following:

- update existing functions and, if necessary, add new functions in UI file of corresponding visualizer tab
- update existing functions and, if necessary, add new functions in Server file of corresponding visualizer tab
- check consistency of the identifiers used for inputs in both UI and Server files of corresponding visualizer tab

If we also need to add visual outputs (tables and/or interactive plots), steps on how to do this are described in other sections.

Regarding examples in the current visualizer template, the user may consider the inputs shown in tab Data.

9 How to add new Shiny actions to visualizer

We need to update the functions related to the structure reactiveTriggers:

- function SetInitialValues__ReactiveTriggers in file **Server__ReactiveVars.R**
- function ObserveEvents__ReactiveTriggers in file **Server__ReactiveVars.R**

If we also need to add visual outputs (tables and/or interactive plots), steps on how to do this are described in other sections.

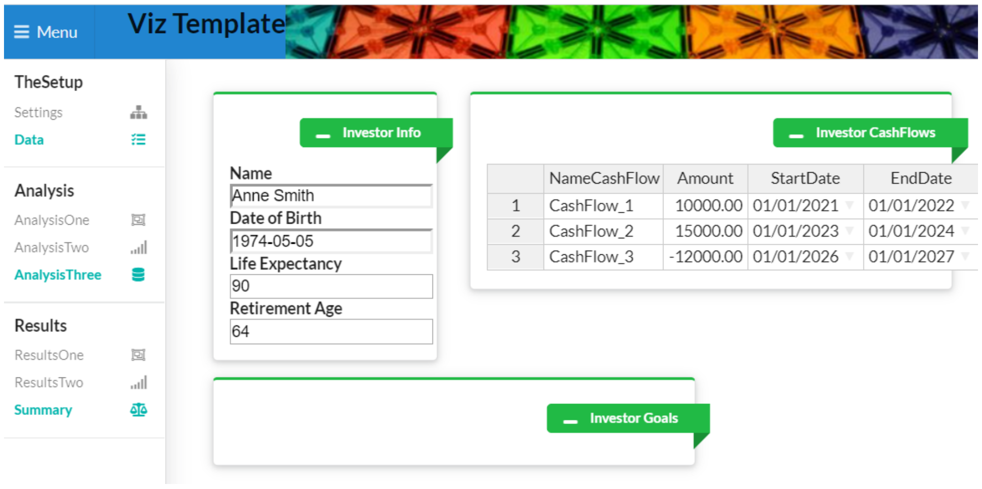
Regarding examples in the current visualizer template, the user may consider the action buttons associated with the tab Summary through the identifiers **inputID__ActionButton__TabSummary__PortfolioValue** and **inputID__ActionButton__TabSummary__DateLastReport**. The actions buttons are implemented in file **UI__TabSummary.R**

The corresponding actions are implemented in function PerformShinyEvents of file **Server__TabSummary.R**.

10 Snapshots of visualizer template

We exemplify the visualizer output through a few snapshots.

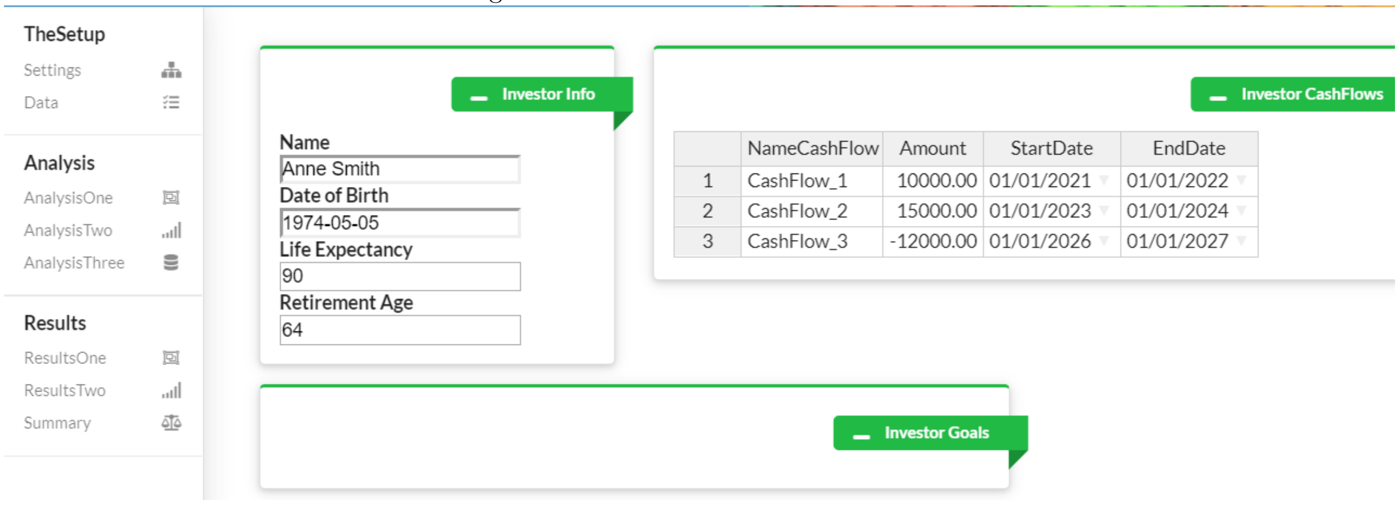
Figure 1: Visualizer sidebar



Name
Anne Smith
Date of Birth
1974-05-05
Life Expectancy
90
Retirement Age
64

	NameCashFlow	Amount	StartDate	EndDate
1	CashFlow_1	10000.00	01/01/2021	01/01/2022
2	CashFlow_2	15000.00	01/01/2023	01/01/2024
3	CashFlow_3	-12000.00	01/01/2026	01/01/2027

Figure 2: Content of Data visualizer tab



Name
Anne Smith
Date of Birth
1974-05-05
Life Expectancy
90
Retirement Age
64

	NameCashFlow	Amount	StartDate	EndDate
1	CashFlow_1	10000.00	01/01/2021	01/01/2022
2	CashFlow_2	15000.00	01/01/2023	01/01/2024
3	CashFlow_3	-12000.00	01/01/2026	01/01/2027

Figure 3: Content of Data visualizer tab with minimized visual outputs

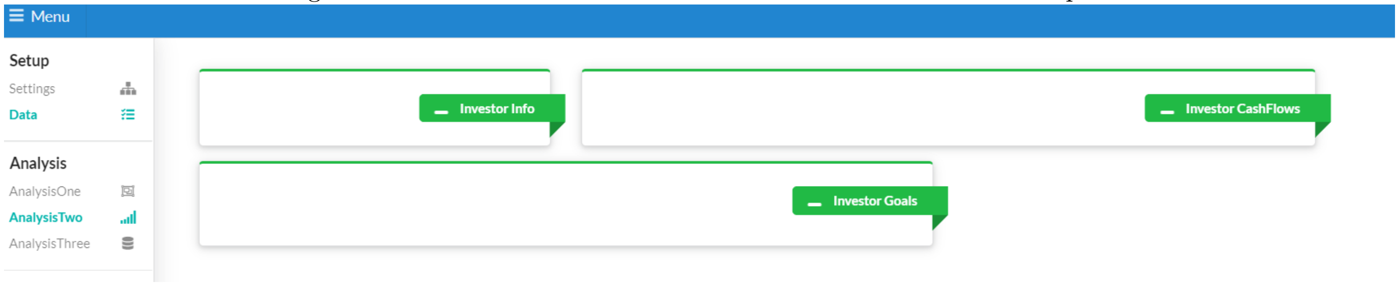
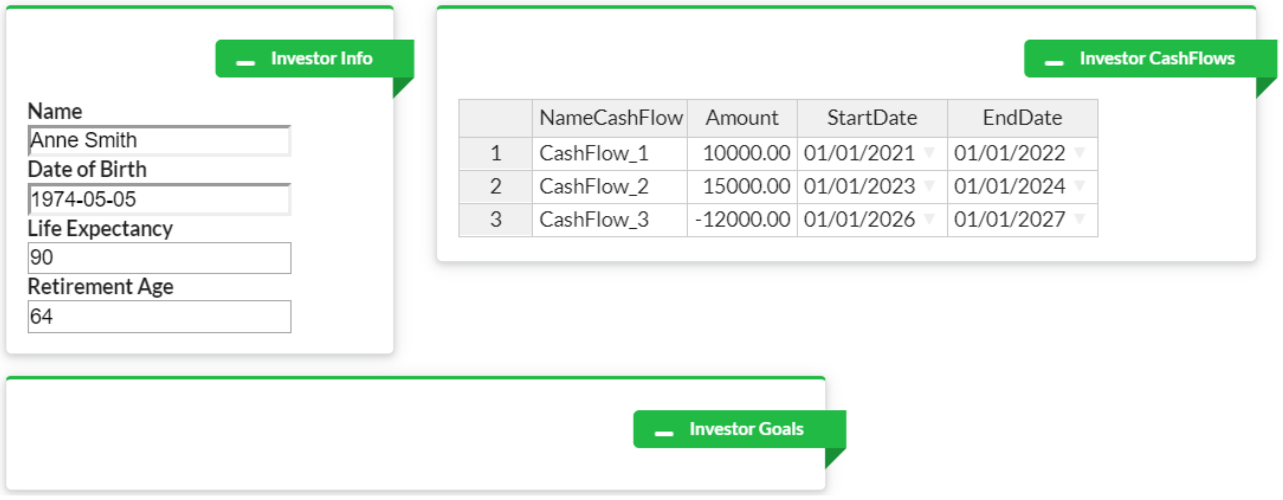


Figure 4: Content of Data visualizer tab with no sidebar shown



Investor Info

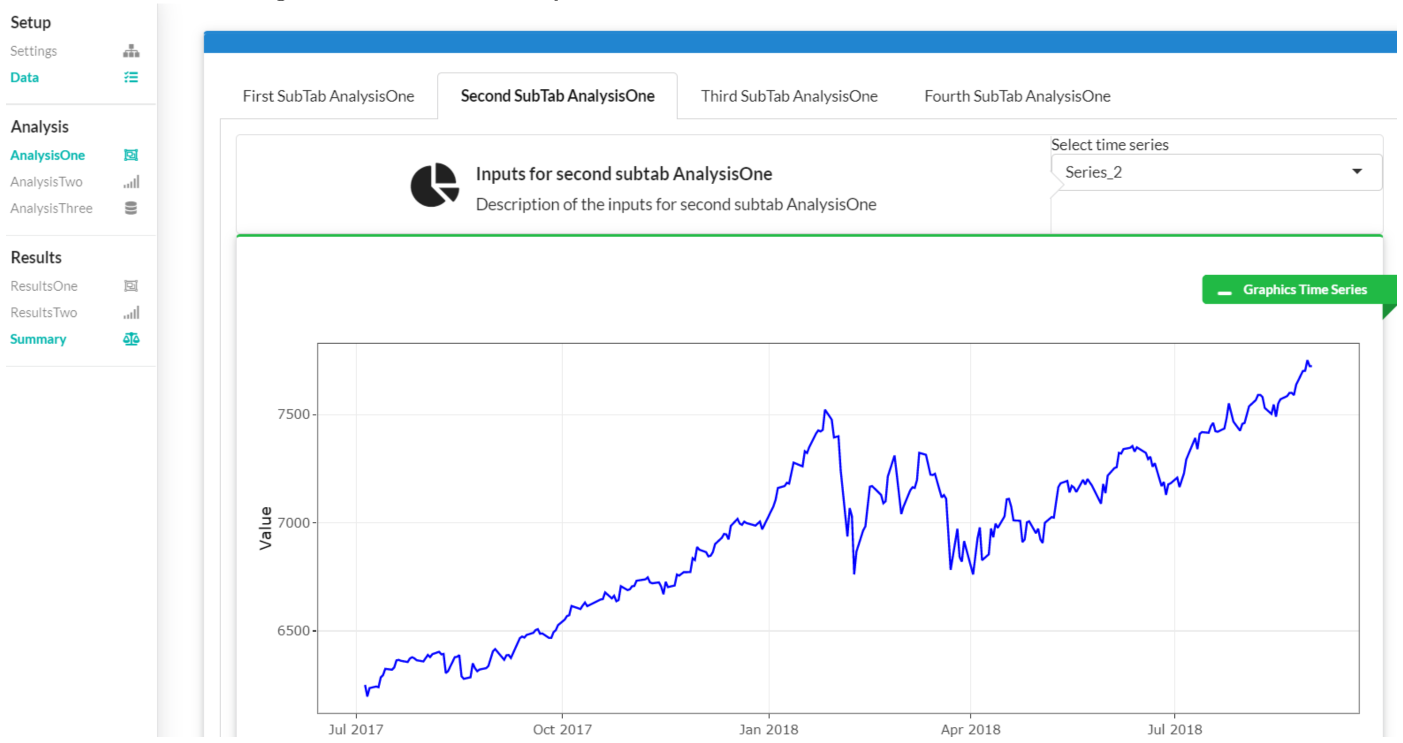
Name	Anne Smith
Date of Birth	1974-05-05
Life Expectancy	90
Retirement Age	64

Investor CashFlows

	NameCashFlow	Amount	StartDate	EndDate
1	CashFlow_1	10000.00	01/01/2021	01/01/2022
2	CashFlow_2	15000.00	01/01/2023	01/01/2024
3	CashFlow_3	-12000.00	01/01/2026	01/01/2027

Investor Goals

Figure 5: Content of AnalysisOne visualizer tab: time series in second subtab

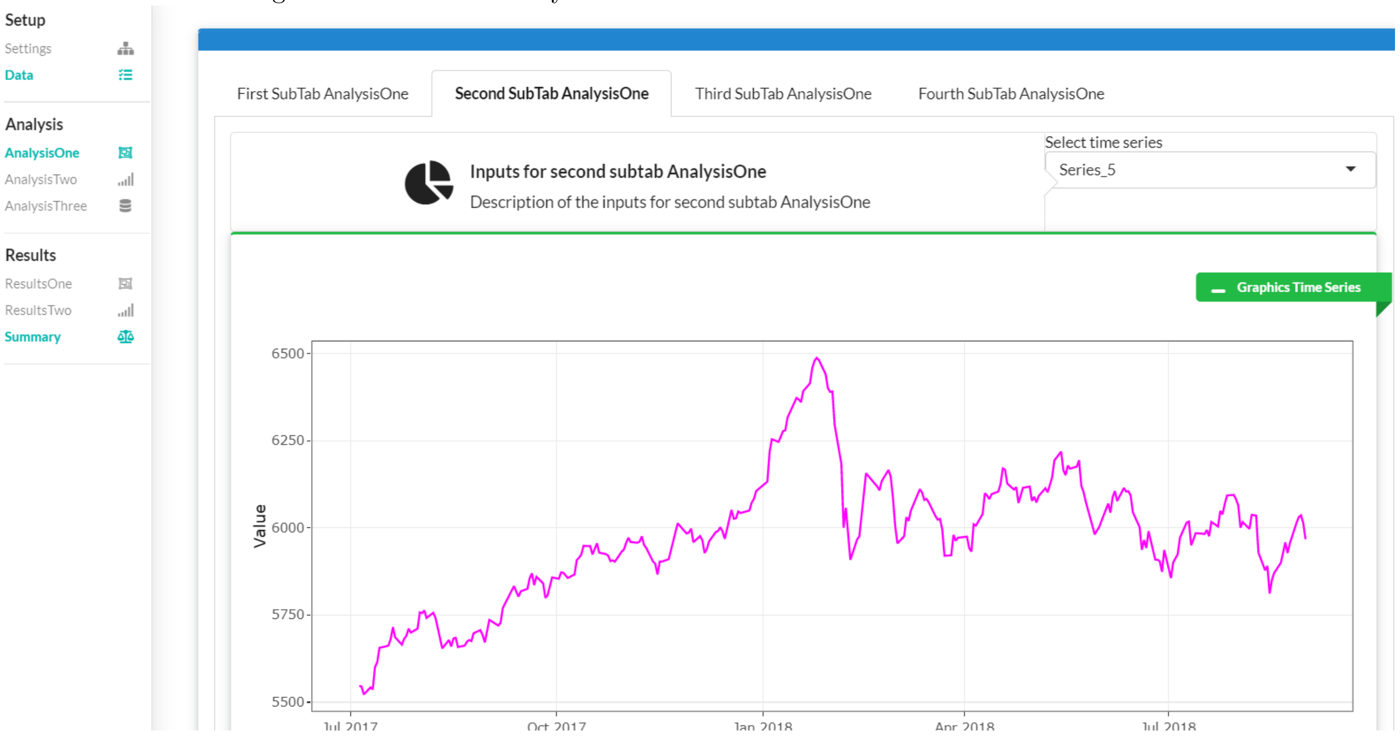


Inputs for second subtab AnalysisOne

Description of the inputs for second subtab AnalysisOne

Select time series: Series_2

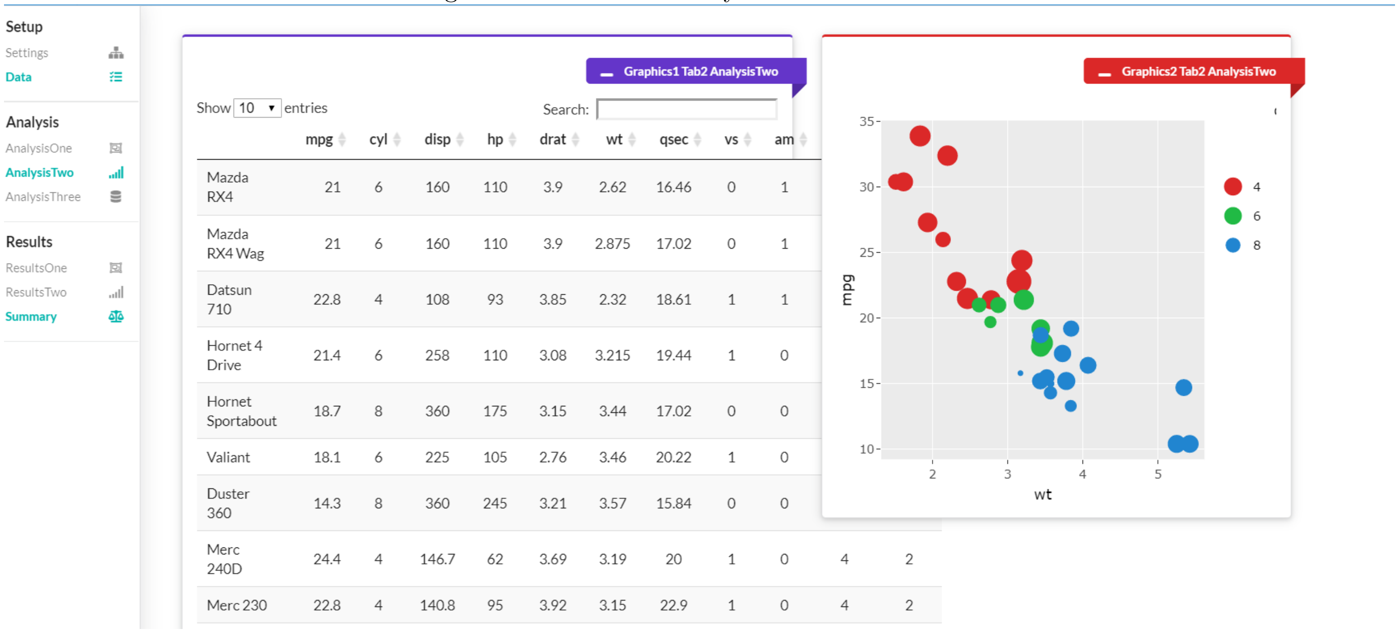
Figure 6: Content of AnalysisOne visualizer tab: time series in second subtab



Inputs for second subtab AnalysisOne

Description of the inputs for second subtab AnalysisOne

Figure 7: Content of AnalysisTwo visualizer tab

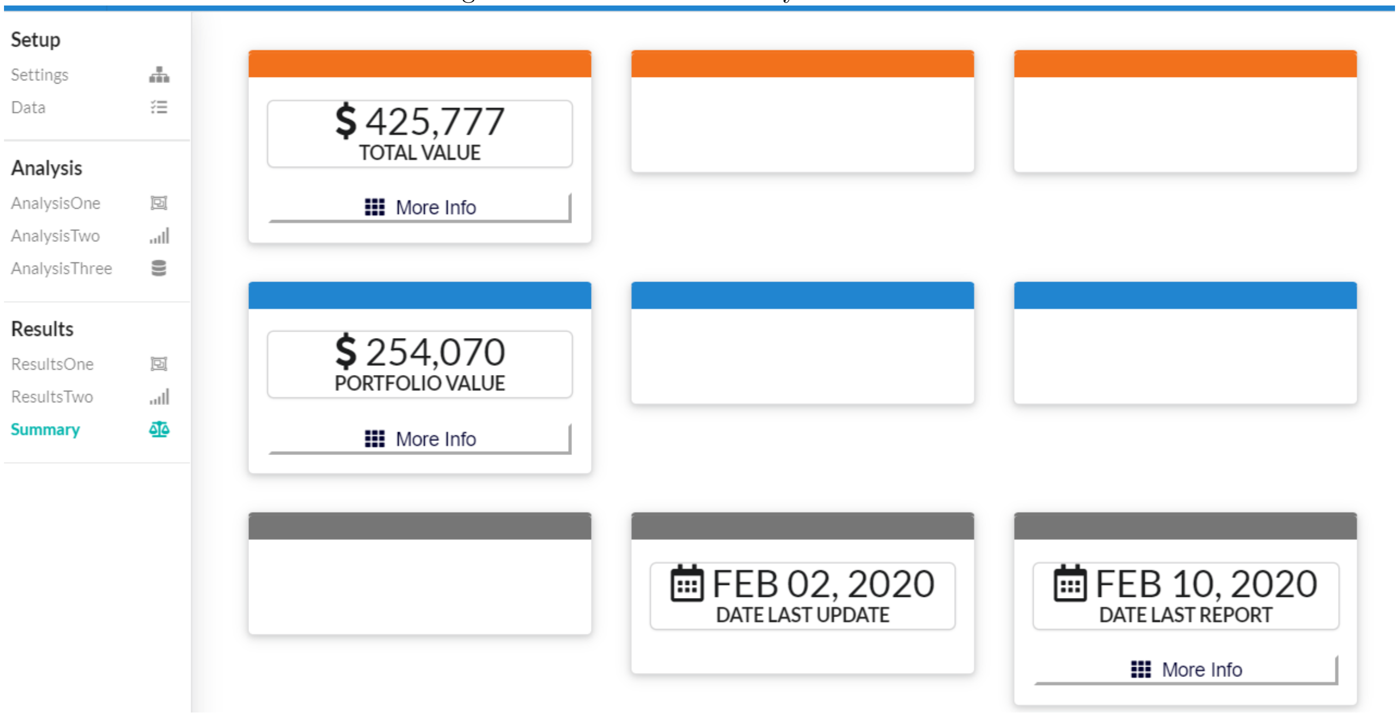


Show 10 entries

Search:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am		
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1		
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1		
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1		
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0		
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0		
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0		
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0		
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2

Figure 8: Content of Summary visualizer tab



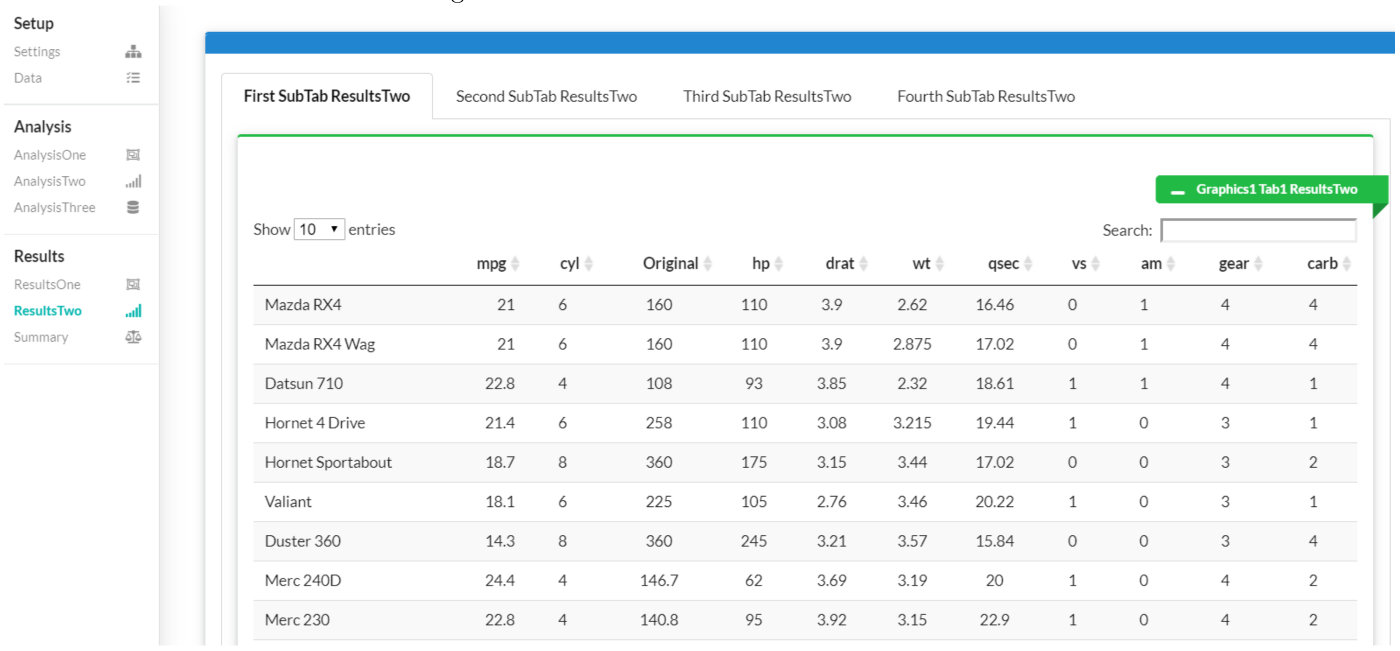
$ 425,777
TOTAL VALUE

$ 254,070
PORTFOLIO VALUE

FEB 02, 2020
DATE LAST UPDATE

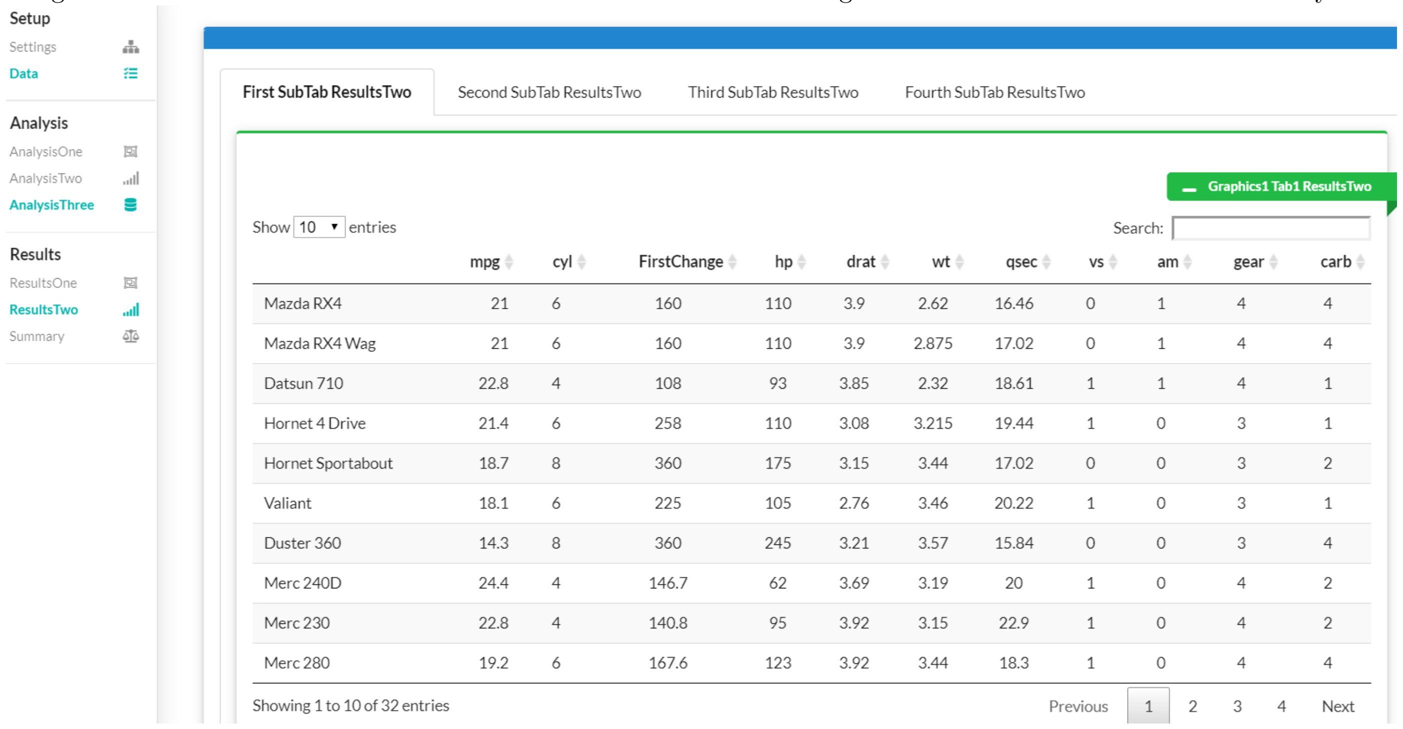
FEB 10, 2020
DATE LAST REPORT

Figure 9: Content of ResultsTwo visualizer tab



	mpg	cyl	Original	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2

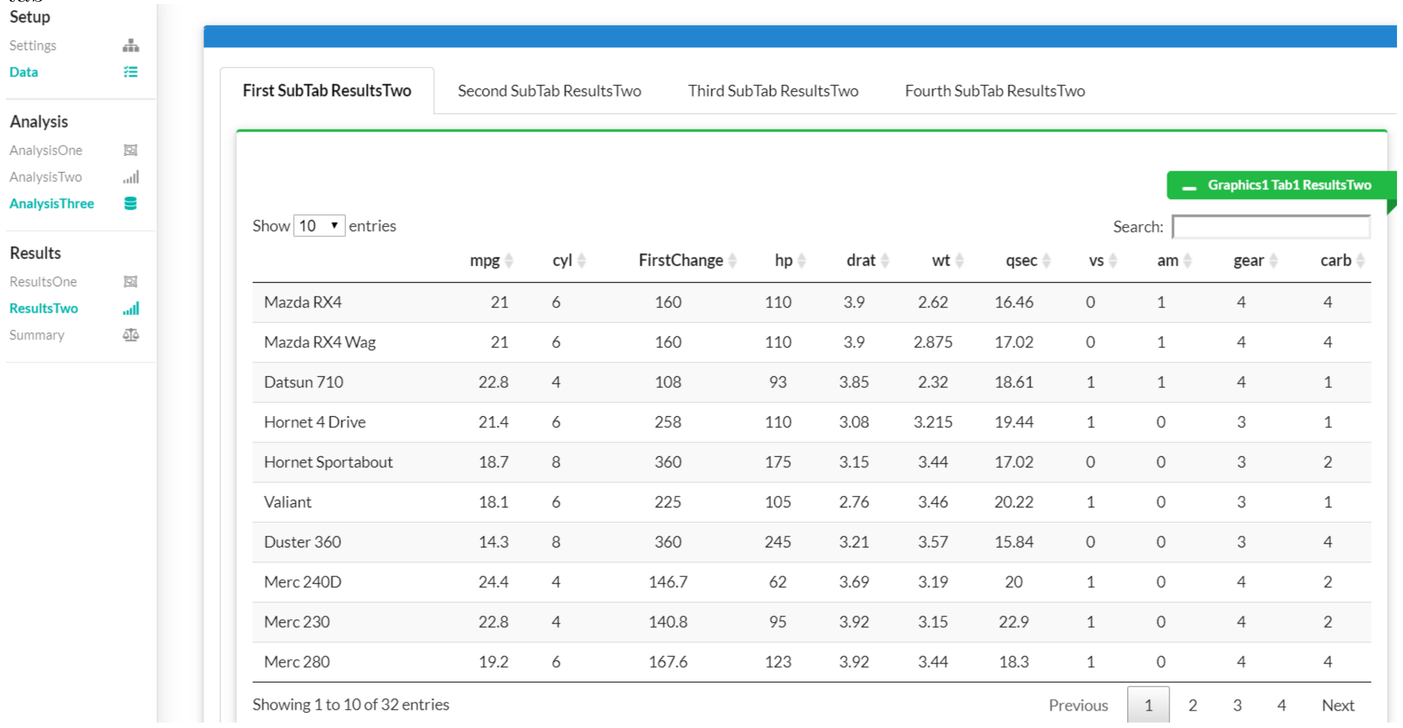
Figure 10: ResultsTwo visualizer tab: name of third column was changed as result of actionButton in Summary tab



	mpg	cyl	FirstChange	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4

Showing 1 to 10 of 32 entries

Figure 11: ResultsTwo visualizer tab: name of third column was changed again as result of actionButton in Summary tab



	mpg	cyl	FirstChange	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4

Showing 1 to 10 of 32 entries