**SCJS ShinyApp – Information/Instructions**

Contents

[**1.** **What is it?** 1](#_Toc519778193)

[**2.** **Why is it?** 1](#_Toc519778194)

[**3.** **Where is it?** 1](#_Toc519778195)

[**4.** **How to: Updating the App** 2](#_Toc519778196)

[**5.** **How to: Adding New Variables** 6](#_Toc519778197)

[**6.** **How to: Understanding the App** 8](#_Toc519778198)

# **What is it?**

An interactive web app which aims to display a selection of areas of the SCJS disaggregated by police division.

# **Why is it?**

Currently, the SCJS releases police division data in excel tables.

This is only done for two sets of questions: police confidence, and attitudes to the police. Proportions are tested between each division and the national average, and between current survey year and first/previous year.

The development of the RShiny App offers more accessible dissemination of the data, and allows for more versatility in testing proportions (users can pick and choose what to test). The interactivity of ShinyApps means that uses are more engaged, and can hone in on the data which best suits their interests and best answers their questions.

Given that users will likely have a range of interests/perspectives on the data, the app attempts to give both a high-level overview of all police divisions (which are above/below the national average) and trends, as well as allowing more detail on specific divisions should that be desired.

# **Where is it?**

The current version can be found at the following url: <https://scotland.shinyapps.io/sg-scottish-crime-justice-survey>

A compressed folder of the required files for the app is on eRDM in the **Scottish Crime and Justice Survey (SCJS): Restricted Documents 2016/17: 2017-2022** directory

# **How to: Updating the App**

**Requirements**

* the SCJS\_ShinyApp R Project (this is the entire directory named *SCJS\_ShinyApp*, which contains the *SCJS\_ShinyApp.Rproj* file, a compressed version is currently stored on eRDM).
* R (version 3.4.4 or later)
* Rstudio
* R packages:
  + tidyverse OR (individually): tidyr, dplyr, magrittr, forcats, ggplot2
  + shiny, plotly, leaflet, sp, rsconnect\*
* a secret/token for the scotland.shinyapps account

(I emailed [Victoria.Avila@nrscotland.gov.uk](mailto:Victoria.Avila@nrscotland.gov.uk) to get these)

\*At the time of writing, the SG supported version of R does not include the package “rsconnect” which is required for publishing ShinyApps online.

Most app developers within SG tend to use their own laptops.

I am not sure if this can be done over scots wifi, as I have only done it on my own laptop tethered to my phone.

**Step 0 - Preliminaries**

* Copy the *SCJS\_ShinyApp.zip* compressed folder to a location on your computer.
* Extract/Unzip
* Place any new data into into *SCJS\_ShinyApp/data/* and make sure it is named, as the others, beginning with “SCJS”

\* Data needs to be in .csv format. If you’re working in SAS, then simply open your dataset, and export the data as .csv. If you're using the .sav files provided by the contractors, then open in SPSS, and File > Export > CSV Data, making sure that the box "Save value labels where defined instead of data values" is unchecked.

**Step 1 – Opening RStudio, Loading the project.**

* Open RStudio.
* From RStudio, open the project file *SCJS\_ShinyApp.Rproj*

File > Open Project > …

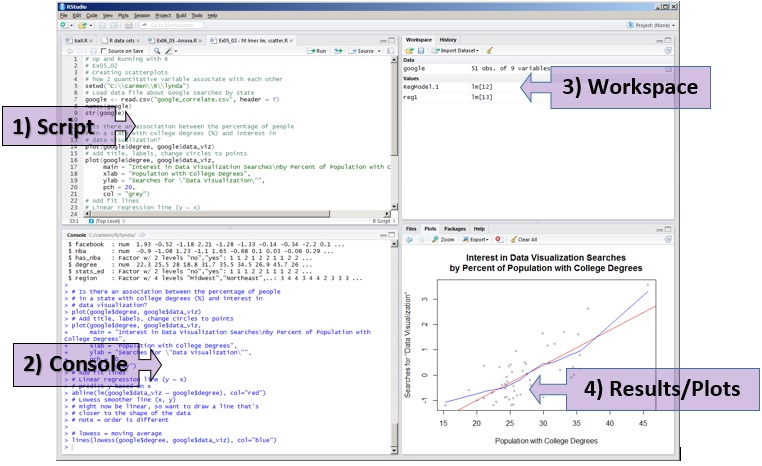
The project should now open. It will load the environment (top right window) might open some scripts (top left window) and in the very top right corner it should show:



|  |
| --- |
| **If you have never used R and RStudio before, the next page is for you! If you have then skip ahead to Step 1b** |

**Step 1a – Understanding R and Rstudio.**

R studio normally has 4 windows



**1) script**

This is where you write and save code. You can just see in the above picture there are several tabs open. Each one is a script, which can contain R code (and comments).

**2) console**

This is the bit which executes code. You can type in it (e.g. 2+3), and it will evaluate it (and return, e.g. 5), but you can’t edit/save code which is typed in there.

You can send code from the script to be evaluated (in the console) by clicking on a line in the script and pressing ctrl+enter or clicking run (at which point you will see it appear in the console). You can run multiple lines of code by highlighting it in the script and running it.

**3) workspace**

The workspace is the working environment. This includes any user defined objects (data, vectors, lists, functions etc).

e.g. if you assign an object called x to be of a value 5, you will notice x appears in the workspace window.

Examples to type in the console:

x<-5

y<-"sheep"

z<-c(1,2,3,4,x)

#this is a comment. It won’t be evaluated because it starts with a #

**4) results/plots/help**

This is where plots/results and help appear.

e.g. typing this into the console

plot(1,1)

will open a plot of the number 1 against the number 1.

**Step 2 – Setting up the Scotland ShinyApp using the token/secret.**

(Skip if you have already done this on your machine)

After receiving the token and secret (probably from Vicky), you need to setup the account info. Broadly, what this does is link your RStudio to the scotland.shinyapps.io account, allowing you to publish the app online.

In the console type the following line, replacing <TOKEN> with your token, and <SECRET> with your secret:

rsconnect::setAccountInfo(name="scotland", token="<TOKEN>", secret="<SECRET>")

More Details Here: <https://shiny.rstudio.com/articles/shinyapps.html>

**Step 3 – Opening & Understanding the script which updates the app.**

* Open the *loader.R* script

(File > Open File > …)

WHAT THE *LOADER.R* SCRIPT DOES

|  |
| --- |
| The loader.R script does the following (in order):   * Reads in the relevant variables from NVF datasets (all years)   (because SG machines appear to give limited CPU resources to R, loading all variables was time-consuming)   * Tidies demographic variables (police divisions, age etc.)   (not all of this is necessary, and is leftover from before the decision to focus on police divisions, but it's harmless, and may one day be useful)   * Recodes the questions.   This involves separating out groups of variables which match patterns of responses we are interested in, and then recoding them in the data  (e.g. for polconf, dconf, polop questions, we are interested in responses of 1&2 compared to responses of -2,-1,3,4,&5. So we collapse these to just be 0s and 1s)   * Checks to see what variables are present in which year.   (completely unnecessary for the app, but if you're working through it and not familiar with the survey, then useful)   * Creates datasets of proportions, by police div and by year   This involves:  choosing the variables of interest (separated by individual vs household weighting)  calculating sample sizes and weighted proportions for each variable/police division/year  reshaping these into long format  joining with design factors  calculating confidence intervals   * Gets the data ready for the app   This involves:  joining information about the variables (contained in the ***data/variable\_information.csv***)  setting up things like user input choices and colour schemes  loading the map data (from ***data/pd\_mapdata.RDS***)   * Updates/Overwrites the .Rdata file used in the app   The .Rdata file is the workspace which the app runs from.   * Deploys the app locally (i.e. opens it on your machine) |

**Step 4 – Updating the script**

There are 2 lines of code to change to give the script enough information for it to correctly update the app.

* **Line 18** - append the new year of the survey to the list of years
* **Line 20** - append the new year’s design factor to the list of design factor

**Step 5 – Running the script**

You can either run the entire script by selecting all the code (ctrl+A) and running it (ctrl+enter) - this will run all the lines of code in it sequentially.

Barring any mishaps, it should open up a window displaying the updated app.

**OR**

You can go through the code slowly. Starting at the top of the script, run each line of code one at a time. Click on a line of code (so that the flashing cursor is somewhere on that line) and press ctrl+enter. This will run that line and the cursor will move down to the next line (so you can just keep pressing ctrl+enter and it will slowly run each line).

This is a useful approach as it means you can see where (if) errors arise.

The final line of the script will open up a window displaying the updated app.

|  |
| --- |
| **Got stuck? Getting errors? Nothing happening?**  **Now might be the time to seek assistance from someone who knows R.**  **Failing that, feel free to contact me at** [**josiahpjking@gmail.com**](mailto:josiahpjking@gmail.com) |

**Step 6 – Check the app.**

If the app has successfully run on your machine (hooray!!) then now you should check some basic things:

* Are the plots loading?
* Do the plots show the year you have added? Are the percentages what you'd expect?
* Is there loss of functionality? (Can you still select divisions/variables etc?)

(probably not necessary to check all the selections, just a few)

**Step 7 - Publish the app!**

In the window which has opened, you can publish the app via the button in the top right corner.

Alternatively, if you close the window, you can publish it via R by running the command:

rsconnect::deployApp(appDir="./app/")

(see Deploying Apps at <https://shiny.rstudio.com/articles/shinyapps.html> )

# **How to: Adding New Variables**

To add new variables to the app, you need to do several things.

* Add details of the variables into variables\_information.csv

(open this in excel, and add rows for new questions). It’s self-explanatory what to add in, given the current rows.

Make sure you save it as .csv

* Edit the variable\_information.R file (this is the text which appears on the app).

This will maybe look imposing but it's really quite simple. It is a list of divs (html elements). It should be reasonably clear on inspection how to add questions to a group.

* Read in the new variables (i.e. load them from the data files into R)

**Line 26**

This line looks in the data files for variables which contain the specified strings (all lowercase), separated by a vertical bar.

e.g. to search for variables containing qworr or qhworr, you could use "qworr|qhworr" or "worr" and they would find the variables you want.

* Recode them correctly to 0/1 (for calculating proportions).

**Lines 96-172**

Survey questions vary in the possible responses. They also vary in which options we are interested in looking at.

e.g. qpolconf has 6 possible responses, which want to be mapped to 2.

|  |  |
| --- | --- |
| -2: Refused  -1: Don't know  1: Very confident  2: Fairly confident  3: Not very confident  4: Not at all confident  NA | 1: Confident  0: Other Responses  NA |

If the new variables you are adding match the response mapping of some of the current variables (e.g. for polconf, dconf, lcpeop, qhworr .. etc., we are interested in responses of 1 or 2 compared to all others) then you may be able to simply add that question to the appropriate list to get recoded in the same way.

If the new variables are quite different, you can simply replicate one of the current lines of code and specify which values you wish to map to which.

If there are any requirements to change sample sizes, this should be done here. Currently, this is only needed for the qworr\_01-03 variables, which in 2014/15 were asked prior to establishing whether the respondent owned any vehicle. Thus, for any respondent who has a car, we set their response to qworr\_01-03 as NA, thus excluding them from the calculated proportions.

* Add them to the lists of either indiv or hhd variables (how are they weighted?)

**Lines 200-210**

These lines simply make 2 lists of variables/questions. These lists are then referenced in calculating proportions, determining what weighting to apply.

* Add them to the list of variables which the app uses. (either in a current grouping or creating a new one).

**Lines 385-395**

This separates the variables into groups, which are then referenced by the app when showing inputs and working out what to plot.

By now you should be able to identify which bits of code are doing what – each list item is as follows:

'<Group Name>' = getnames("<variable\_prefix>")

e.g. 'Worries of Crime' = getnames("QWORR")

**N.B** – the variable prefix for this is in uppercase, because it is referring to the "label" column in variable\_information.csv

Only add new groups *to the end of the list.* (the app relies on national indicators being the first group).

* Run the entire script (see Step 5 of How to Update).

# **How to: Understanding the App**

The app is contained inside the *SCJS\_ShinyApp/app* folder.

There is an associated .Rproj file which preserves information about workspace etc required for updating the app. This is found in the *SCJS\_ShinyApp* folder.

Updating the app can be done via the *loader.R* script.

To simply update the app, skip ahead. If you’re interested in changing the app, read on.

**Other folders:**

**/app** : the code for the app itself includes *ui.R, server.R* and the *.Rdata* file.

/**app/source** : app ui and server scripts.

/**app/www** : css and images

**/data** : includes the current data file, and information on variables, used by the *loader.R* script to join with new data

**/setup** : functions for updating the app, as used in *loader.R*

**Inside the App**

The app consists of 6 tabs, and each of these relies on a set of R scripts.

Details of what each tab does, and what the scripts do, can be found below.

*This information is only necessary should you desire to change the app layout/plots/functionality. You should have some knowledge of R before editing any of the below scripts as it may cause severe loss of app functionality.*

|  |  |
| --- | --- |
| **PAGE INFO** | **FILES** |
| **Home Tab**  Aims to give a (very broad) overview of the SCJS by focusing on the 3 national indicators included in the survey.  Infographics on whether they each indicator is improving, maintaining, or worsening.  Map of police divisions coloured by National Indicators (user selection).  Button links to the other tabs in the app. | **ui\_home.R** : layout, including text.  **server\_home.R** :  - national indicator graphics (call the correct icon and text to display).  - observes button clicks to update tab selection.  **server\_map.R** :  - map output  (icons for buttons and indicators are found in */app/www/* ) |
| **Breakdown by police divisions**  Highlights how police divisions have performed relative to the national average.  Users select a survey area, and a year/police division (depending on graph).  There are two graphs  - A single year with all divisions visible and the national average as a solid black line. Divisions are coloured as significantly better/worse than national average.  - A single division with all years present and the national average for each year as a solid black line. Results for each year are coloured as significantly better/worse than national average. | **ui\_overview.R** : layout  **server\_overview.R** : some data manipulation and creates plots.  Compares all data points to national average in that year for that variable (also wraps some string variables for plotting titles/annotations etc).  - creates plots. If user selects national indicator, returns barplot, if whole sections of questions returns scatterplots.  Updates the input to show either police div or year, depending on which plot is visible.  text output for info on plot (depending on what plot is displayed)  text output for info on variables (depending on what variable/s selected)  Observes plot clicks and updates which plot is visible.  **variable\_information.R** : contains text info on all variables |
| **Comparison Tool**  This is designed to provide users with a proportion testing tool which can display whole sections of the survey at a time (e.g. if there’s lots of red and green bars, then the two selections are quite different in the selected survey area).  Users choose a survey area, and make two selections of a police division (or national average) and year.  Results are compared and colour coded. | **ui\_comparison.R** : layout  **server\_compare.R** : some data manipulation and creates plots.  - subsets data into the two user selections (police division, year).  - checks for common variables  - tests proportions.  Makes plots based on above data.  Text output for info on variables selected  **variable\_information.R** : contains text info on all variables. |
| **Trends**  Simply displays line graphs across time for any variable and any division.  Does not add much to the app, but offers option to display confidence intervals, and may be preferable for some users.  Users choose survey variable(s) and police divisions. | **ui\_trends.R** : layout  **server\_trendplot.R** : filters the data and produces the plots.  Filters data based on user input of variables and police divisions.  Makes plots (with or without confidence intervals based on user input).  Updates variable selection based on survey area selection.  Observes a "RESET" button to reset plot to initial state.  Observes clicks on link to comparison tab. |
| **Tables**  Displays customizable tables of percentages and samplesizes, with the ability to download them as .csv files.  User chooses area of survey, police divisions, and two years to compare between.  Table of percentages includes column saying yes/no based on whether there is a signif difference between the two selected years. | **ui\_tables.R** : layout  **server\_table.R** : does a little data manipulation and makes the tables.  Filters the data to the variable and police division inputs.  Creates downloadable data (no testing)  Creates table of percentages (with proportion testing)  Creates table of samplesizes (no testing)  Updates selection when "select all" is chosen. (for both variables and police divisions)  Observes "reset table" button.  Download data button |
| **Help & Info**  Links to the SCJS publications page, contains info on stats testing, confidence intervals etc. | **ui\_help.R** : layout, and all text.  **ui\_links.R** : links present at footer of all tabs |