## Introduction and definitions

The purpose of this paper is to set out guidance for using the SMRA databases with RStudio, specifically RStudio server (although the code examples and much of the other information are transferable to RStudio desktop). It is intended to be understandable to first time users of R/SMRA/SQL, and includes step by step instructions to access the server and basic examples of R and SQL code to allow new users to get started using SMRA with R. It is NOT intended to be a comprehensive introduction to R, RStudio or SQL, although it does include links to further resources on these topics.

### Definitions

### The SMRA database

The SMRA database is a relational database, specifically an [Oracle database](https://en.wikipedia.org/wiki/Oracle_Database). The data contained in the SMRA database is secondary care data in the form of the [Scottish Morbidity Records](http://www.ndc.scot.nhs.uk/National-Datasets/index.asp?ID=1&Topic=Hospital%20Activity%20Statistics) (SMRs). The SMRs record [hospital activity](http://www.isdscotland.org/Health-Topics/Hospital-Care/Inpatient-and-Day-Case-Activity/) (including [outpatien](http://www.isdscotland.org/Health-Topics/Hospital-Care/Outpatient-Datamart/)t clinics). The SMRA also contains [NRS death records](https://www.nrscotland.gov.uk/statistics-and-data/statistics/statistics-by-theme/vital-events/deaths), which can similarly be linked to hospital records.

### SQL

[SQL](https://en.wikipedia.org/wiki/SQL) stands for Structured Query Language. It is a language used in programming and designed for managing and extracting data held in a relational database management system. We use SQL to 'query' the database to extract data. It is rare that you would ever want to extract all the information held in a database, so SQL queries will contain instructions so that only data you need. NSS databases use the Oracle system, therefore we use Oracle SQL to access it.

### RStudio server

[R](https://cran.r-project.org/) is a system for statistical computation and graphics. [RStudio](https://www.rstudio.com/products/RStudio/) is a software application for the R language. It is essentially a more user friendly and versatile way to use R than the basic R environment. RStudio can be run as a desktop version or on a server accessible through a webpage. An introduction to use of RStudio can be accessed for free [here](https://www.datacamp.com/courses/working-with-the-rstudio-ide-part-1).

## Advantages of using RStudio server to access SMRA

### Why use SMRA with RStudio server?

* The ability to access SMRA through RStudio means that if further analysis is to be done in R, there is no break in the workflow through having to access SMRA through another program (e.g. SPSS).
* There is often pressure on SPSS licences in ISD, using R and RStudio instead, which is available on a free software licence, will help relieve this pressure
* On RStudio desktop, the size and complexity of analyses that can be run is limited by the processing power and memory of your PC or laptop as all the R code executes locally. Accessing the server allows you to access greater processing power and more memory, permitting larger and more complex analyses to be run than on the desktop.
* Accessing the server allows analyses to be run on larger datasets even on slower internet connections (an advantage when working at home), as all the data is accessed and processed remotely.

## Requesting access

### Databases

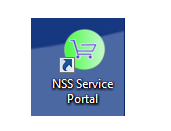
* First you need to be able to access some or all of the SMR datasets. Request access using the Access to Data form in the usual way (ask your line manger if you are not sure how to do this/where to find the form).
* If you already have access to the SMR databases you need, e.g. you already access these data sources through SPSS or business objects, you can move onto the next step

### R /RStudio Desktop

* If you wish to access SMRA via the desktop version of RStudio, you will need R and RStudio installed on your machine. Check the Microsoft system centre to see if R or RStudio is available for you to install.
* If not, make a request to IT to have the current R version *with the connection to the ODBC* installed on your machine.[[1]](#footnote-1)

### RStudio server

* Access to the RStudio Server can be requested on an individual basis via the NSS service portal.



* To assist the Customer Support Desk with your request, it may be helpful to mention that your request can be actioned by the UNIX team.

### ODBC connection

* For RStudio server, make a request (through the NSS service portal) to be set up with a connection to the SMRA ODBC on the server, giving your username.
* For RStudio desktop, connecting to the SMRA database relies on an *SMRA* ODBC DSN being set up on your PC by NSS IM&T.
  + You can check if the DSN is present on your PC in the *ODBC Data Source Administrator*. Click on the Start button and search for *Microsoft ODBC Administrator*. Click on the link with the same name to open the *ODBC Data Source Administrator*.
  + On the *System DSN* tab you should see an entry called *SMRA.* If not, raise a Change Request with the Customer Support Desk to have the *SMRA ODBC DSN* installed on your PC.

## RStudio server: Logging in and basic navigation.

### Logging in

The RStudio server is located at: [http://nssRStudio.csa.scot.nhs.uk/](http://nssrstudio.csa.scot.nhs.uk/). Login using your username and the password you normally use to access the stats area through the file system and through SPSS.

If you are unfamiliar with RStudio, a basic introduction to finding your way around RStudio can be found [here](https://www.sitepoint.com/introduction-r-rstudio/). Many articles on use of RStudio are available on the [RStudio support website](https://support.rstudio.com/hc/en-us/sections/200107586-Using-the-RStudio-IDE).

#### Accessing Files on the Stats Server

The default directory on the RStudio server that is displayed in the bottom right pane is the 'Home' directory', which is only accessible to the logged in user. You will normally want to use and save files from the rest of the network. To navigate to these other areas using the file browser in RStudio Server, click on the icon with the 3 dots to the right hans side of the files pane.

In the box that appears type in the appropriate directory name (e.g. the 'PHI\_conf' area or the older 'conf' area containing subdirectories belonging to different teams. Files in cl-out can be found at /conf/linkage/output). The Files tab will then show the directory you have linked through and you can click through to access files in subdirectories.

The same technique can be used in the File Browser window accessible via the File Open File… menu option.

NB RStudio server cannot connect to all servers/file areas in use in NSS. If you cannot see your file area through the file browser, raise a request with IT to find out whether it can be connected or not.

If your area cannot be connected you can either:

* Run your analysis on RStudio desktop (this may not be possible for large analyses)

or

* Store files in an alternative (appropriately secure!) location that can be accessed from RStudio: e.g. /conf/bss/03-Workspace/... and move final syntax/output files to the appropriate directory when the analysis is finished.

### Sign Out vs. Quit Session

There are two ways to exit RStudio Server:

1. Sign Out

2. Quit Session

Signing out of RStudio Server leaves your R session open, and if you are currently running R code, this will continue to run in the background. Leaving your R session open allows you to return to where you left off next time you log in. To sign out of RStudio Server, click on the close window button of your web browser or click the “Sign out” button in the RStudio Server interface

**Signing out and leaving your R session open has the downside of continuing to consume server processing time and memory even if you are not actively running any R code, consequently reducing the available server resources for other users. It is therefore recommended that if you have finished an analysis that you quit your R session.**

You can quit your R session by selecting the “Quit Session…” option in the File menu, selecting the “Quit Session…” option in the Session menu or clicking the “Quit Session” button at the top-right of the RStudio Server interface.

## Opening a connection to SMRA

### Loading required packages

First, load the required libraries using the following code. Copy the following code into the upper left hand pane in RStudio, highlight all the text and hit ctrl+return or click on the Run icon above the pane:

library(dplyr)

library(readr)

library(odbc)

The odbc package is required to connect to the database. The readr and dplyr packages (both belonging to the [*tidyverse*](https://www.tidyverse.org/) set of packages) are also recommended if you would like to execute SQL queries saved in external files and store data in [*tibbles*](http://tibble.tidyverse.org/)*.*

N.B. if you don't already have these packages installed, you will get an error saying that there is no such package in the library. In this case, click the "install" button in the 'Packages' tab in the lower right hand pane of the RStudio server. Type odbc(or other required package name) in the pop up and it will be installed. Then run the library(package) code again. Alternatively, you can use code to install the package(s), e.g.:

install.packages("odbc").

#### Problems and alternatives to odbc

There are several R packages which provide the functionality to connect to external ODBC databases. The odbc package is currently the recommended package for connecting to ODBC databases, it is faster and more efficient than alternatives. You should use odbc with the SMRA datasets, and all code in this document pertains to the odbc package. However, there are known problems with using odbc with some of the other datasets used in ISD (e.g. Ecossstats). If you experience problems using odbc with your database (and you are certain it is not a coding error), try using package [RODBC](https://CRAN.R-project.org/package=RODBC) instead.

### Set up the connection

The code to set up the connection is:

channel <- suppressWarnings(dbConnect(odbc(), dsn="SMRA",

                    uid=.rs.askForPassword("SMRA Username:"),

                    pwd=.rs.askForPassword("SMRA Password:")))

When you run the code popup boxes will appear for you to enter your username and password.

If the connection is successful an object called 'channel' will appear in the upper right hand pane (Environment tab).

Wrapping the *dbConnect* function inside the *suppressWarnings* function prevents your password for connecting to the SMRA database being shown in clear text on the console should the connection attempt be unsuccessful.

### Close the connection

At the end of the session close the channel using:

odbcClose(channel)

## Code to get information on the views and variables available to you in SMRA

Now the connection is open, you can inspect the data views available to you and the variables contained within those views. The analysis 'views' can be thought of as tables in the database containing a category of information. For example the Scottish morbidity records (SMR00, SMR01, SMR02, SMR04) each have their separate views.

### Getting started

1. Show all tables/views available - N.B. this is a large output and takes some time to run.

dbListTables(channel)[1:50]

The [1:50,] restricts the output to the first 50 views available

Look at the table 'scheme' types that are available by running the code below

odbcListObjects(channel)

"Analysis" schemes are the analysis views, likely the ones you want to use. Tables with a 'scheme' which is a username are data uploaded by individual users. If you are unsure which view you require, consult the information in [this folder](file:///\\Freddy\Projects\SAF\Subgroups\SMRA%20Analysis%20views%20subgroup\File%20layouts).

1. Show tables/views again, but this time restrict to "Analysis" views only.

odbcListObjects(channel, schema="ANALYSIS")

name type

1 GP\_PRACTICE\_SCOTLAND view

2 GRO\_DEATHS\_C view

3 GRO\_DEATHS\_WEEKLY\_C view

4 GRO\_INFANT\_DEATHS\_C view

5 GRO\_MULTIPLE\_STILLBIRTHS\_C view

6 GRO\_STILLBIRTHS\_C view

.....

16 SMR00\_PI view

17 SMR00\_PI\_AUG2017 view

18 SMR01\_1E\_PI view

19 SMR01\_HAH\_PI view

20 SMR01\_HISTORIC view

21 SMR01\_PI view

22 SMR01\_PI\_APR2017 view

23 SMR01\_PI\_AUG2017 view

24 SMR04\_HISTORIC view

25 SMR04\_PI view

26 SMR06\_PI view

27 SPECIALTY view

You can now see the names of the analysis views, these are the same names/data as can be viewed in SPSS through the wizard. Most of the views are versions of the various SMR databases: [SMR00](http://www.ndc.scot.nhs.uk/Data-Dictionary/SMR-Datasets/SMR00-Outpatient-Attendance/SMR00-Summary-of-Rules/) (outpatients), [SMR01](http://www.ndc.scot.nhs.uk/Dictionary-A-Z/Definitions/index.asp?Search=S&ID=460&Title=SMR01%20-%20General/Acute%20Inpatient%20and%20Day%20Case) (Acute hospital activity), [SMR04](http://www.ndc.scot.nhs.uk/Dictionary-A-Z/Definitions/index.asp?Search=S&ID=999&Title=SMR04%20-%20Summary%20of%20Rules) etc.

The "\_PI" suffix means that these views contain personally identifying information (names, dates of birth etc). SMRXX\_PI is the usual version used for analyses. SMRA also includes NRS death records (GRO\_DEATHS\_C).

Other files are reference files which contain information about coded fields e.g. if you have a list of codes referring to hospitals and you want to know the hospital names or postcodes, you would use the LOCATION view to look up this information.

1. **Viewing variables names**

In order to view the variables, we can preview the table. Further details of variable names and attributes in each of the SMR views can also be found in the relevant [file](file:///\\freddy\projects\SAF\Subgroups\SMRA%20Analysis%20views%20subgroup\File%20layouts).

odbcPreviewObject(channel, table="ANALYSIS.SMR01\_PI", rowLimit=0)

## Extract data from SMRA

To extract data from SMRA, you need to use SQL code, to 'query' the database.

To make an SQL query of the database through RStudio server, you need to use the dbGetQuery function.

You will need to give the channel name and the SQL query (statement). Other parameters are optional.

The SMRA connection was simply called "channel" in code above, so we can fill in dbGetQuery (conn=channel, statement=..)

When filling in the "statement" argument, you have two choices.

1. Write the SQL query in a separate SQL script file and import it to R
2. Write the query directly into the statemtent option in the dbGetQuery function.

The choice of which method to use is personal choice, as it doesn’t affect efficiency. Writing the SQL query into the R script means that all the code is kept in one file. On the other hand, you may find it easier to write and edit the SQL query in a separate file. If SQL is saved in a separate file, it is easier to open it in an SQL editor to check and debug. Even in RStudio, once the text is saved as an SQL file, RStudio will recognise this, and commands and functions will be shown in different colours, making reading and editing the syntax easier.

The two methods are illustrated below.

#### Method 1.

1. Open a new text file in RStudio. Go to File>New File> Text File

This opens a new blank text file

1. Write the SQL query into the text file.

E.g. To extract age, sex and location (i.e. hospital code) from all SMR01 records for men over 100:

SELECT AGE\_IN\_YEARS, SEX, LOCATION FROM ANALYSIS.SMR01\_PI WHERE AGE\_IN\_YEARS>100 AND SEX=1 rownames<=10

Save the file with a ".sql" extension.

NB rownames <=10 is used to limit the number of rows returned to 10. This will reduce the time it takes to run this example. In practice, you may want to retrieve a limited number of rows when setting up a query in order to check that the query is returning the expected information/ the variables selected are the correct ones.

1. Read the SQL query to execute against the SMRA database from an external file:

SQL <- read\_file("/path/to/file/query.sql")

1. Execute the SQL query against the SMRA database, fetch the records and store in a tibble:

table1 <- dbGetQuery (channel, query=SQL))

#### Method 2

1. In your existing .R script file, write the query into the sqlQuery function.

table1<- tbl\_df (dbGetQuery(channel, statement="SELECT

AGE\_IN\_YEARS, SEX, LOCATION

FROM ANALYSIS.SMR01\_PI

WHERE AGE\_IN\_YEARS>100 AND SEX=1 rownames<=10"))

The SQL code is exactly the same as written in a separate file, but in this case it is wrapped in paste("").

### Viewing the result

With either method the returned object "table1" can be viewed by typing ' table1' into the code window and running, or clicking on 'table1' in the Environment tab.

It should look something like this:

table1

# A tibble: 10 × 3

AGE\_IN\_YEARS SEX LOCATION

\* <int> <int> <fctr>

1 102 1 N102H

2 102 1 S226H

3 101 1 F805H

4 105 1 G516H

5 101 1 S308H

6 102 1 N102H

7 103 1 S116H

8 102 1 S226H

9 101 1 T208H

10 104 1 G516H

## Using SQL to extract data efficiently

It is best practice to make good use of SQL code to reduce the amount of data you extract from the database and to perform as much data wrangling as possible in the database, especially for operations that are repeated many times. This increases efficiency. A comprehensive guide to efficient use of SQL code is beyond the scope of this paper, however:

* The most important rule is to minimise your extract as much as possible, i.e. only extract the data you need and as little extra as you can. Extracting a large amount of data which you then immediately refine to discard values is inefficient in terms of both computing time and memory, it is better to limit your extract in the first place.
* the two functions you are likely to use most are:
  + WHERE – to extract records based on certain [conditions](#_Conditional_selection.) e.g. date of record, age of patient
  + [JOIN](#_Matching_information_from) or match records from different tables to avoid having to perform multiple extracts

Examples of these two operations are shown below.

* If you are extracting information about a particular group of patients for whom you have a LINK\_NO or UPI or other identifying information, it is best to use the **cohort method.** This involves uploading a table to the SMRA database and using this table to match records in other table. See next section for the cohort method.

**IMPORTANT:** When retrieving data from SMRA you must also SORT the data in a certain order first. SMRA data must *always* be sorted on the following fields: LINK\_NO, ADMISSION\_DATE, DISCHARGE\_DATE, ADMISSION, DISCHARGE, URI. See SAG guidance [here](file:///\\freddy\projects\SAF\Agendas,%20Minutes%20and%20Bulletins\Bulletins\SAF%20Bulletin%20No.16%20Special%20Edition%20-%20SMRA%20Guidance.docx).

Do this by including these variables in the [ORDER BY](https://www.1keydata.com/sql/sqlorderby.html) command e.g.

table1<- tbl\_df(sqlQuery(channel=channel, query=paste("

SELECT AGE\_IN\_YEARS, SEX, LOCATION

FROM

ANALYSIS.SMR01\_PI

WHERE

AGE\_IN\_YEARS>100 AND SEX=1

ORDER BY LINK\_NO, ADMISSION\_DATE,

DISCHARGE\_DATE, ADMISSION, DISCHARGE, URI"), max=10))

You do not have to include the variables in the select part of your query for this to work.

Other useful functions include:

* [CREATE VIEW](https://www.1keydata.com/sql/sql-create-view.html): to create customised temporary views which can be queried multiple times. E.g. views based on a particular cohort of patients
* Use of [sub-queries](https://www.1keydata.com/sql/sql-subquery.html)

Links to further SQL resources can be found in [Section 8](#_Further_resources)

### WHERE command for conditional selection

* This is possibly the most important basic function to master.
* You can select data based on one condition or many using the WHERE command e.g. in the previous examples records were only extracted WHERE the conditions sex=1 (i.e. men) and age over 100 were met. The conditions are specified after the WHERE command.

table1 <- tbl\_df(dbGetQuery(SMRA,statement="SELECT

AGE\_IN\_YEARS, SEX, LOCATION

FROM

ANALYSIS.SMR01\_PI

WHERE

**AGE\_IN\_YEARS>100 AND SEX=1**

ORDER BY LINK\_NO, ADMISSION\_DATE, DISCHARGE\_DATE, ADMISSION, DISCHARGE, URI "))

* You can also limit extracts by date. The following code restricts the extract to a time period of BETWEEN 1st January 2010 AND 30th of April 2010.

SMR\_date<-tbl\_df(dbGetQuery(SMRA, statement="SELECT

LPAD(LINK\_NO,10,0) AS LINK\_NO ,LOCATION, ADMISSION\_DATE, DISCHARGE\_DATE,

MAIN\_CONDITION, CIS\_MARKER

FROM ANALYSIS.SMR01\_PI

WHERE

**ADMISSION\_DATE BETWEEN {d '2010-01-01'} AND {d '2010-04-30'}**

ORDER BY LINK\_NO, ADMISSION\_DATE, DISCHARGE\_DATE, ADMISSION, DISCHARGE, URI" ))

The BETWEEN clause is *inclusive* of both dates.

Run:

summary(SMR\_time$ADMISSION\_DATE)

To prove this.

#### Note on variables with leading zeros

Any number with leading zeros will have these dropped by default when you query the database. E.g. '00001234' will be returned as '1234'. To retrieve the variable with leading zeros included, we use the LPAD function:

e.g. for the variable "LINK\_NO" in the query above, which should be 10 characters long and sometimes contains leading zeros:

"SELECT LPAD(LINK\_NO,10,0), ....."

Will return 1234 as 0000001234.

A further point to note on the use of LPAD, is that SELECT LPAD(LINK\_NO,10,0) will, by default, change the name of the imported variable to "LPAD(LINK\_NO, 10, 0,)". To import the variable in the correct format with the name "LINK\_NO", we specify: "SELECT LPAD(LINK\_NO,10,0),**AS LINK\_NO**", which renames the variable back to LINK\_NO.

The AS clause can be used to rename any variable.

SELECT VAR1 AS V\_ONE, VAR2 AS V\_TWO

Would give the first variable the name "V\_ONE" and the second variable the name "V\_TWO".

### Matching information from two or more tables.

* This is called a JOIN in SQL language.

e.g. if you want to know when (or if) a patient has died, you can match death records data onto hospital records using the LINK\_NO field.

* There are a number of things you have to specify in the SQL query when you extract data from more than one table:

1. You must specify, for each variable, which table it comes from
2. You must specify on which variable(s) the join is to be made
3. You must specify the [type of join](http://www.sql-join.com/sql-join-types/)

Example:

We wish to JOIN death dates on to a group of SMR01 records.

SMR\_Join <- dbGetQuery(channel, statement="SELECT

LPAD(T1.Link\_NO,10,0) AS LINK\_NO, T1.LOCATION, T1.SPECIALTY, T1.ADMISSION\_TYPE, T1.ADMISSION\_DATE, T1.DISCHARGE\_DATE, T1.MAIN\_CONDITION, T2.DATE\_OF\_DEATH

FROM

ANALYSIS.SMR01\_PI T1

**LEFT JOIN**

ANALYSIS.GRO\_DEATHS\_C T2

**ON T1.LINK\_NO= T2.LINK\_NO**

WHERE rownum<=100")

head(SMR\_Join)

* We label ANALYSIS.SMR01\_PI as "T1" and ANALYSIS.GRO\_DEATHS\_C as "T2" . The variables are then named as T1.var1, T2, var2 etc...
  + You can perform joins without labelling the tables and just use the original names, in which case the SQL query would read "("SELECT ANALYSIS.SMR01\_PI.Link\_NO, ANALYSIS.SMR01\_PI.LOCATION, ..... ANALYSIS.GRO\_DEATHS\_C.DATE\_OF\_DEATH...". This is cumbersome and harder to read than assigning a shorthand name.
* The WHERE clause specifies the fields to be used in the join. The fields do not have to have the same name in both tables as you specify both names in the query.
* The join type is specified as a LEFT JOIN, i.e. return all records matching other conditions from the first table, and only matching records from the second table. (So SMR01 records are returned whether or not there is a matching death record, however death records with no corresponding hospital record are not returned).
  + See [www.sql-join.com/sql-join-types/](http://www.sql-join.com/sql-join-types/) for a visual illustration of the available join types.
* It is also possible to join more than two tables at once

## Cohort method

This method is used to extract data from SMRA based on individual patient identifiers this might be used to extract information about a study cohort for example.

The process of uploading and using a table is as follows:

1. Create a table of UPI or LINK numbers in R as a datatable

Example:

LINK\_NO<-c(00000119, 75881029, 45960570, 00019300, 01959251, 01959200, 11949200, 02939132, 00000383)

INCOHORT<-rep("yes", 9)

x<-as.data.frame(cbind(LINK\_NO, INCOHORT))

1. Upload to SMRA using the dbWriteTable function

dbWriteTable(SMRA, "test", x)

The table will be uploaded to a schema that is your username. To see a list of tables that you have uploaded run:

dbListTables(SMRA,schema="<USERNAME>")

Replacing <USERNAME> with the username you use to access SMRA, in all capitals.

1. Use the uploaded table in a query.

The table name must be enclosed in double quotes, this *only* applies to your uploaded tables, NOT to SMRA tables/views generally. The query statement as a whole must therefore be enclosed in single quotes when using a temporary table.

Example:

This syntax works:

test <- dbGetQuery(SMRA,statement='SELECT \* FROM <USERNAME>."test" ')

This syntax, with the double and single quote positions inverted, will result in an error.

test <- dbGetQuery(SMRA,statement="select \* from '<USERNAME>.test'")

1. Using the table to retrieve information from SMRA using a JOIN command:

SMR\_cohort <- tbl\_df(dbGetQuery(SMRA,

statement='SELECT

T2.LINK\_NO,T2.LOCATION, T2.ADMISSION\_DATE, T2.DISCHARGE\_DATE,

T2.MAIN\_CONDITION, T2.CIS\_MARKER, T1.INCOHORT

FROM

EMILYM02."test" T1

LEFT JOIN

ANALYSIS.SMR01\_PI T2

ON T1.LINK\_NO = T2.LINK\_NO

ORDER BY T2.LINK\_NO, T2.ADMISSION\_DATE, T2.DISCHARGE\_DATE, T2.ADMISSION, T2.DISCHARGE, T2.URI' ))

1. Delete the table once you have finished using it using dbRemoveTable

dbRemoveTable(SMRA, "test")

To check it has been removed:

dbListTables(SMRA,schema="<USERNAME>")

NB It is of course possible to extract information based on a personal table containing other fields in addition to LINK\_NO to further refine the selection, perhaps years of interest in combination with LINK\_NO.

## Further resources

SQL code examples & tutorials

<https://oracle-base.com/articles/sql/articles-sql>

<https://www.w3schools.com/sql/default.asp>

Resources for learning R and RStudio, linked from RStudio's website

[https://www.RStudio.com/online-learning/#R](https://www.rstudio.com/online-learning/#R)

General coding help, both for R and SQL, can be found by searching on

<https://stackoverflow.com/>

1. At time of writing there is no regular update scheduled for R on local machines and on the R Server. The latest desktop version in use in NSS is R3.3.2 and the R Server is on 3.2.3. However, it is planned to establish regular updates*,* at least annually, for both desktop and serve versions. [↑](#footnote-ref-1)