|  |
| --- |
| CSC-40042 |
| SQLite & dplyr in R |
| Statistical Techniques for Data Analytics |

|  |
| --- |
| Alex Farrell  15005594 |

Contents

[Introduction 2](#_Toc532421500)

[RSQlite 2](#_Toc532421501)

[Dplyr 2](#_Toc532421502)

[Census data tasks – Discussion 3](#_Toc532421503)

[Task 1 3](#_Toc532421504)

[Task 2 3](#_Toc532421505)

[Task 3 3](#_Toc532421506)

[Task 4 4](#_Toc532421507)

[Task 5 4](#_Toc532421508)

[Task 6a 4](#_Toc532421509)

[Task 6b 5](#_Toc532421510)

[Conclusion 5](#_Toc532421511)

[References 6](#_Toc532421512)

# Introduction

This report discusses the similarities and differences between the RSQlite package and the dplyr package, when used in RStudio. These packages were used to manipulate census data from a data file to give certain information. This report will also determine which package is preferred for each task, along with overall efficiency and time taken to complete each task.

## RSQlite

The RSQlite package embeds the ‘SQlite’ database engine in R and provides an interface that complies with the ‘DBI’ package (Muller, et al., 2018). SQlite is a single-user, light-weight database engine, implementing a decent subset of the SQL 92 standard, including core table creation, updating insertion and selection operations, along with transaction management (r-project, n.d.). Unlike SQL, SQlite does not offer user management and is ideal for mobile application and testing. It is also not designed to work with big-scale data and is not recommended for use in enterprises due to security reasons (Yang, 2014).

## Dplyr

Dplyr is a package which provides a set of tools for the manipulation of datasets in R. Hadley Wickham discusses three key ideas that underlie dplyr. One of these key ideas is that performance will only get better over time, especially once the creators figure out the best way to make the most of multiple processors. Another key idea is that you can use dplyr when working on a local data frame or a remote database table, with support included for database engines such as SQlite. Finally, the bottleneck when conducting most data analyses is the time it takes for you to figure out what to do with data; in dplyr, this is made easier by the use of individual functions that correspond to the most common operations (Wickham, 2014).

# Census data tasks – Discussion

This section will compare the use of RSQlite and dplyr when completing manipulation tasks on the census data. The comparisons will summarise the amount of code required to complete the task and the overall efficiency of the code in terms of memory and time taken.

## Task 1

The first task required the data to be put into an SQlite database, however as dplyr does not require a database, as it can manipulate data frames directly, the data from the file was read directly into a new data frame using the ‘read.csv’ function. Overall, this task was better suited to the dplyr package as the data was being read into a local data frame, only requiring the use of 2 functions which read the data and renamed the columns respectively. Compared to when this task was completed using RSQlite, an extra function was required to create a database connection. Also, in terms of file space, dplyr was more efficient as the data was being stored in a local data frame, instead of a separate database file.

## Task 2

The second task required the addition of a column, ‘SS\_ID’, which should subsequently be set to the primary key of the table. When this was attempted in RSQlite, an error was thrown which stated that this could not be done. The reason for this, trails back to referential integrity. When a table is created in SQlite without a primary key being referenced, the table automatically assigns a ‘hidden primary key’ in the way of the ‘rowId’. Therefore, the easiest way to add a new column as a primary key, is to create a new table. This means that temporarily, extra memory is used, until the original table is deleted.

In comparison, due to dplyr not requiring a database, the ‘mutate’ function could be used to add another column to the data frame and set it to the value of the row names. However, due to not being able to add primary keys to a data frame, this criterion could not be fulfilled. If dplyr was to be used with a database, the same issue as discovered when using RSQlite would appear due to the referential integrity as discussed above.

## Task 3

The next task required the total number of males and females to be displayed, grouped by their race. Due to the task specifying the genders required, it was reasonable to assume that these could be hard-coded into the statements. Therefore, a statement which retrieved females and males was used and stored in respective data frames. These data frames were then combined using the merge function. This method was not very memory efficient and required multiple lines of code.

In comparison, when using dplyr, the ‘group\_by’ function can be used to select individual columns. When combined with the ‘tally’ function, a data frame will be created which displays only the necessary fields, along with a column displaying the number of records that fall into each group. Compared to RSQlite, this method using dplyr only requires a single line of code, creating a single data frame. Therefore, it is much more memory and time efficient.

## Task 4

The next task required the calculation of the average annual income for each race group. In this calculation, it was crucial that only those contracts that were not zero-hour were considered. Also, from analysis of the census data, it became apparent that the values had been multiplied by 40 (as specified in the task, an assumption is made that each person works a 40-hour week).

When completed using RSQlite, the first step was to create a data frame containing the different race groups represented in the table, utilising the ‘select distinct’ function. This method of retrieving the race groups eliminated the possibility of incorrectly entering the data into the main query, as the string is lifted directly from the table. Then, a for loop was used to loop through this data frame, using the ‘avg’ function to calculate the average of the sum of the hours pay (AHRSPAY) and the weeks worked (WKSWORK). Each time an iteration of the loop was completed, the result was added as a new row to a data frame, created before the for loop was initiated.

In comparison, when using dplyr, the amount of code required, as well as the complexity, was much lower. To begin with, the ‘filter’ function was used to remove the rows from the table which contained zero-hour contracts. Next, the ‘group\_by’ function was used to group the data by the race group. Finally, the ‘summarise’ function was used, containing the ‘mean’ function to calculate the average income. Overall, dplyr was much simpler to use here as less code was required to complete the same task. Using this package also reduced the time taken to run the code as there was no loop involved.

## Task 5

The next task required the creation of 3 new tables containing data extracted from the census income table. Computationally, this task was similar when completed using both RSQlite and dplyr, however due to the necessity for a primary key in RSQlite, an extra step was required to first create the table with the data types and keys, before inserting the data using a combination of the ‘insert’ and ‘select’ functions. In dplyr, the select function could be used to select the data directly from the census income data frame.

## Task 6a

The first part of the final task required the retrieval of the number of people employed in the highest paying job, grouped by their state. When completed using RSQlite, this could be completed in a single query where you could use the ‘max’ function to specify the retrieval of the maximum value for the hours pay (AHRSPAY), then specify the remaining fields required. Then an inner-join could be used to join all three tables together, specifying the field (SS\_ID), in which the data would be joined. The ‘where’ clause in this statement was nested, containing a further ‘where’ clause. The reason for this is to select the job type which corresponds to the highest wage, as this data is not contained within the same table.

When using dplyr for this task, the job type corresponding to the highest wage needed to be retrieved before completing the query. As the data was stored in a data frame, even though this only contained one data item, when referenced in the main query, the dollar sign ($) symbol was required to specify the column in which the data resided.

## Task 6b

The last part of the final task required the retrieval of the average wage and average number of weeks worked of those of Hispanic origin and with a degree, grouped by the type of industry they work in. When completed using RSQlite, this could be completed in a single query, where the industry could be selected, along with use of the ‘avg’ function to calculate the average of the hours pay and weeks worked. Then an inner-join could be used to join all three tables together, specifying the field (SS\_ID), in which the data would be joined. The ‘where’ clause made use of the ‘in’ and ‘not in’ functions to filter out values that didn’t match the criteria specified in the task. Finally, the results were grouped by the industry.

When using dplyr for this task, a single query could also be used, resulting in similar computational efficiency. The necessary fields were selected, along with inner joins being used to combine the data. The ‘filter’ function was then used to remove all non-matching records. In terms of code required to do this, there is no function like the ‘in’ function in RSQlite, therefore each clause must be entered separately, using the logical AND, and OR operators. The ‘group\_by’ function was then used to group the results by the industry, followed by the ‘summarise’ function to calculate the average wage and average weeks worked, utilising the ‘round’ function to reduce the decimal places displayed in the results.

# Conclusion

Overall, the preferred package would be dplyr for several reasons. Firstly, for simple tasks such as those required in the assignment, the simplified functions of dplyr offer enough complexity to retrieve the data you want in a more efficient way than RSQlite. However, for more complex queries, such as those required in task 6, although dplyr was sufficient to complete the tasks, computationally, there is an argument to suggest that RSQlite is better since both tasks could be completed in one query. This extends the restrictions of the dplyr package with its limited main functions.

Also, in terms of efficiency, the amount of code required to complete the tasks was considerably less with dplyr than it was with RSQlite. This directly correlates to the time and memory efficiency which was also considerably better with dplyr. For example, task 2 for RSQlite required the creation of another table since data was being queried from a database table as appose to a data frame. However, as stated, dplyr can also be used with an SQlite database. Therefore, it is reasonable to assume that if this was the case, the same issues with referential integrity would arise.

# References

Muller, K., Wickham, H., James, D. & Falcon, S., 2018. *RSQLite.* [Online]   
Available at: https://cran.r-project.org/web/packages/RSQLite/RSQLite.pdf  
[Accessed 12 12 2018].

r-project, n.d. *RSQLite.* [Online]   
Available at: https://www.r-project.org/nosvn/pandoc/RSQLite.html  
[Accessed 12 12 2018].

Wickham, H., 2014. *Introducing dplyr.* [Online]   
Available at: https://blog.rstudio.com/2014/01/17/introducing-dplyr/  
[Accessed 12 12 2018].

Yang, Y., 2014. *Introduction to RSQLite.* [Online]   
Available at: http://users.stat.umn.edu/~yang3175/lit\_sem/RSQLite\_Tutorial.html#1  
[Accessed 12 12 2018].