THE UNIVERSITY OF AUCKLAND

HONOURS PROJECT

Survey statistics in a database

Author:
Charco HUI

Supervisor:
Professor Thomas LUMLEY

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science (Honours)

in the

Department of Statistics

June 05, 2018

THE UNIVERSITY OF AUCKLAND

Abstract

Department of Statistics

Bachelor of Science (Honours)

Survey statistics in a database

by Charco HUI

Multistage surveys can rise to moderately large data sets (tens of millions of rows). Most current software for survey analysis reads the data into memory, the **survey** package in R provides fairly comprehensive analysis features for complex surveys which are small enough to fit into memory easily, however, most of the computations can actually be expressed as database operations. There is already a similar approach with the **sqlsurvey** package in R which performs substantial computation in SQL in the database, importing only small summary tables into R, this approach scales to very large surveys such as the American Community Survey and the Nationwide Emergency Department Sample, but this approach causes compatibility issues with different types of databases. Therefore, in this project I will work on implementing R functions and testing some survey computations using the **dplyr** and **dbplyr** R package as a efficient and portable database interface.

Acknowledgements

I would like to acknowledge my supervisor, Professor Thomas Lumley with my deepest appreciation. I would like to thank him for his patience in sharing his expertise, without his help, this project would not be possible.

Lastly, i would like to thank my friends and family for their continuous support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Currently, there is already a **survey** package (Lumley, 2004) in R which is at a stable production status, it provides survey analysis, including graphics, estimation and inference. It also supports both replicate-weight and Taylor linearisation standard errors, and can efficiently handle multistage stratified designs without replacements. However, it requires the data sets to be stored in a data frame in memory. For most survey data sets this is not a problem, however, nowadays there are a number of large survey data sets, for example the American Community Survey (ACS) includes 3,000,000 people per year, and the Nationwide Emergency Department Sample (NEDS) includes more than 25,000,000 hospital visit records per year.

In R, there are currently two approaches to analyse survey data sets in a database. The first is to use the **survey** package, with its database back-end function, the data sets can be loaded into memory without any problem, but the time to analysis the data may not be promising when the data sets are too large.

Another approach is to perform as much computation as possible directly in the database, so that only small bits of data or numbers are transferred into memory when necessary. This approach is more efficient but is less flexible, since mathematics and statistical operations are limited in a database. Another advantage of this approach is that if the database is powerful, then the computation would be faster than just using a standard laptop or desktop.

The second approach is implemented in the **sqlsurvey** R package (Lumley, 2014), however, codes which communicates with the databases are written in "hand-written SQL code". Therefore, it would be hard to maintain and would cause compatibility issues between different types of databases. Not only codes may look different, there is also a major inconsistency in evaluating the code, for example dealing with missing values. Despite the attempt of standardising the SQL standards between multiple companies, issues of portability still remains.

The better approach to analyse survey data sets in a database would be to use the R packages **dplyr** (Wickham et al., 2017) and **dbplyr** (Wickham and Ruiz, 2018) as a database interface. Since these packages are maintained by experts at Rstudio, it is likely that these packages are more stable than others, bug fixes and updates would also be quick. Most importantly, it is more portable where its compatibility extends to powerful databases back ends like PostgreSQL and Google BigQuery. So, this project will implement a set of functions to analyse survey data sets with the second approach, named **svydb**, and evaluate its speed on large survey data sets.

1.2 Survey data in SQL

As mentioned in section 1.1, when we are analysing large survey data sets, it would be more feasible to do it in a database. Some commonly used survey statistics are survey mean, survey total, summaries and regression.

Survey totals, means and summaries can be easily computed in a database, since it only requires simple arithmetic like summing, multiplications, divisions along with some grouping.

Regression may require a bit more work, since it requires matrix operations which are not supported in SQL, however by loading a few chunks of small matrices into memory, it can still be easily implemented in a database, since after all regression coefficients and their variances only requires sums and multiplications.

More details of the calculations and difficulties will be discussed later on in Chapter 2.

1.3 Coding with dplyr and dbplyr

1.3.1 Introduction to dplyr

The **dplyr** package was implemented to manipulate, clean and construct data. With this package, data manipulation and data exploration can be done easily and quickly, since they are written in a computationally efficient manner.

The package contains a few common data manipulating functions such as selecting specific columns, arranging or creating new columns, filtering rows, merging data (joins) and summarising data by groups. Other features such as simple statistics operations are also included in the package.

1.3.2 Pipes

The pipe operator (%>%) first appeared in **magrittr** package (Bache and Wickham, 2014), and is created to make codes more readable. The pipe operator inputs the object on the left-hand side of the pipe into the function on right-hand side. Some basic piping are as follows:

- `x %>% f` is equivalent to `f(x)`.
- `x %>% f(y)` is equivalent to `f(x, y)`.

It is rather useful when we have multiple steps while we are transforming data sets, because naturally we read from left to right. For example, with traditional coding, reading is always from inside out,

```
> x = sample(10)
> summary(diff(exp(floor(cos(x)))))
```

with piping, it is much easier to read,

```
> x %>% cos() %>% floor() %>% exp() %>% diff()
    %>% summary()
```

Though the pipe has its advantages, there are also times that it is not useful. It would not be useful when the intermediate variables are needed or when the intermediate variables require heavy computation.

1.3.3 dplyr's SQL compatibility (dbplyr)

As mentioned in section 1.3.1, there are six basic functions in **dplyr**. These functions are all related to the basic SQL queries.

dplyr Function	Description	Equivalent SQL
select()	Selecting columns (variables)	SELECT
filter ()	Filter (subset) rows.	WHERE
group_by()	Group the data	GROUP BY
arrange()	Sort the data	ORDER BY
join()	Joining tables	JOIN
mutate()	Creating New Variables (Columns)	COLUMN ALIAS

With **dbplyr**, when these **dplyr** functions are applied onto a sql table, they automatically translate itself into SQL queries. For example,

```
> mtdb %>% select(mpg, gear) %>% group_by(gear) %>%
    summarise(sum_mpg = sum(mpg)) %>% head(3) %>%
    show_query()

 # <SQL>
 # SELECT "gear", SUM("mpg") AS "sum_mpg"
 # FROM (SELECT "mpg" AS "mpg", "gear" AS "gear"
 # FROM "mtcars") "gaecowztcc"
 # GROUP BY "gear"
 # LIMIT 3
```

With this approach, **dplyr** does not actually do any work, its job is only to translate the codes into SQL and gives the database instructions. Another advantage of this method is that the intermediate variables between the pipes only builds up the query and does not get evaluated nor is stored anywhere.

1.3.4 Quasi-quotation

Programming with **dplyr** relies on a concept called the quasi-quotation, also known as non-standard evaluation, it means that while we are doing some evaluation with **dplyr** in R, we are not using R's standard evaluation method. For example, with R's standard method,

```
test_func = function(x, y){
    x + y
}
> x1 = 1; x2 = 2
> test_func(x1, x2)
```

R looks for the variables `x1` and `x2` in the environment, evaluates them and input their values into the function `test_func`.

However, while programming in **dplyr**,

```
> mtcars %>% select(mpg)
```

The variable `mpg` is a variable in the data set and cannot be found in the environment, it is quoted and evaluated in a non standard way.

Though, it might look useful to use this non-standard evaluation method, but it is more difficult to program with, for example while writing a function,

```
test_fun2 = function(data, x){
    data %>% select(x) %>% head(2)
}
> test_fun2(mtcars, mpg)
# Error: `x` must resolve to integer column positions,
# not a list
```

Since **dplyr** does not evaluate in the standard way, we cannot just pass a variable in like the standard method. We will need to quote it with the `quo()` or `enquo()` function.

```
test_fun2 = function(data, x){
    x = enquo(x)
    data %>% select(!!x) %>% head(2) %>% tbl_df()
}
> test_fun2(mtcars, mpg)

#    mpg
#* <dbl>
#1    21
#2    21
```

`enquo()` allows us to quote the variable resulting in a quosure where it contains its expression along with an evaluation environment, so we can pass it into the `select()` function, and `!!` (bang bang) allows us to unquote the variable at evaluation.

There are also other similar functions to help us overcome the difficulties while programming with **dplyr**, like `quos()`, `sym()` and `quo_name()` which are used in different situations.

1.3.5 Common issues while coding with dplyr in a database

- No factor types in SQL.
- Difficult to code with quasi-quotation.
- Cannot do row-wise operations due to the lazy interface. That is, the data sets within a database in R will not be loaded into memory unless required.
- No matrix operations.
- No base R functions.
- No distributions.
- Inconsistent availability of functions between databases.

1.4 Layout

In chapter 2, estimation methods and functions will be discussed, and in chapter 3, graphics.

In chapter 4, speed of database-based and memory-based implementations will be compared, chapter 5 discuss the usability of the functions, and lastly, a discussion in chapter 6.

Chapter 2

Methodology

2.1 Survey Design

When analysing survey data sets in the **survey** package, a survey design (**svydesign()**) is always required. The survey design object combines the data set and all the survey design information needed to analyse it. These objects are used by the survey modelling and summary functions.

The set of functions that **surveydb** provides also adapted this concept but with a few modifications, it uses the **R6** (Chang, 2017) class system which is encapsulated object orientation programming and is different to the standard **S3** and **S4** in base R which uses functional object orientation programming. The main difference between the two is that **S3** and **S4** methods and objects are separate and in **R6**, object contains methods and data.

The advantage of encapsulated object orientation programming is that information within the objects are not computed unless it is needed, for example when the sum of all sampling weights are needed, we can compute it by using a method within the object and the value also updates when it's been called on a subset of the object.

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
    id = SDMVPSU, data = nhdb)
> nh.dbsurv$getwt()
[1] 306590681

> nh.dbsurv$subset(Race3 == 3)$getwt()
[1] 192721267
```

Therefore, it is much more time efficient than creating a survey design that contains all the information, since the user may not need every information within the object.

The `svydbdesign()` function, has four basic arguments,

```
svydbdesign(st = NULL, id = NULL, wt, data)
```

- *st* = Column name specifying the strata column. *NULL* for no strata.
- *id* = Column name specifying the cluster column. *NULL* for no cluster.
- *wt* = Column name specifying the sampling weights column.
- *data* = A data frame or sql table of the survey data set.

When the `svydb.design` object is called directly, a brief description of the design will be displayed.

```
> nh.dbsurv
svydb.design, 9756 observation(s), 31 Clusters
```

Functions within the **svydb.design** object for users to use are,

```
Classes 'svydb.design', 'R6' <svydb.design>
.
.
clone: function (deep = FALSE)
getmh: function ()
getwt: function ()
subset: function (..., logical = T)
subset_rows: function (from, to)
```

- *clone*() = Create a clone of the object.
- *getmh*() = A table indicating strata and cluster information.
- *getwt*() = Compute the sum of the sampling weights.
- *subset*() = Subset the design by conditions, similar to *base* :: *subset*.
  e.g.(design$subset(Race3 == 3)
- *subset_rows*() = Subset the design by rows.
  e.g.(`design$subset_rows(1, 100)`)

2.2 Population Total

The function `svydbtotal()` was designed to estimate the population total in R by using **dplyr**, it is compatible with data frames and sql tables and was designed to do as much computation as possible in a database.

In the function `svydbtotal()`, the total is computed by using the Horvitz-Thompson estimator (Horvitz and Thompson, 1952), it is an unbiased estimator of the population total.

$$\hat{Total} = \sum_{h=1}^{L} \sum_{i=1}^{m_h} z_{hi}$$

where,

$$z_{hi} = \sum_{j \in PSU} w_{hij} x_{hij}$$

- L = number of stratum
- m_h = number of clusters in stratum h
- w_{hij} = the sample weight in stratum h and cluster i for observation j.

Variance estimation of the total uses the Horvitz-Thompson estimator on an influence function.

$$Var(\hat{Total}) = \sum_{h=1}^{L} \frac{m_h}{m_h - 1} \sum_{i=1}^{m_h} (z_{hi} - \bar{z}_h)^T (z_{hi} - \bar{z}_h)$$

where,

$$\bar{z}_h = \frac{1}{m_h} \sum_{i=1}^{m_h} z_{hi}$$

2.2.1 Usage

```
svydbtotal(x, num = T, return.total = F, design)
```

2.2.2 Arguments

- *x* = Name indicating the variable.
- *num* = `TRUE` or `FALSE` indicating whether x is numeric or categorical.
- *return.total* = `TRUE` to return only totals, no standard errors.
- *design* = svydb.design object.

2.2.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                id = SDMVPSU, data = nhdb)
> svydbtotal(x = Race3, design = nh.dbsurv, num = T)
#          Total       SE
# Race3 959380842 61432595

> svydbtotal(x = Race3, design = nh.dbsurv, num = F)
#            Total       SE
# Race3_1   29812316  6112527
# Race3_2   21416164  4485865
# Race3_3  192721267 23431296
# Race3_4   38131538  5561161
# Race3_6   15519529  2367723
# Race3_7    8989867  1468813

> svydbtotal(x = Race3, design = nh.dbsurv , num = T,
  return.total = T)
#          Total
# Race3 959380842
```

Generic functions like `coef()` and `SE()` were also implemented to extract the coefficients and standard errors from a **svydbstat** object.

```
> class(svydbtotal(x = Race3, design = nh.dbsurv, num = T))
# [1] "svydbstat"

> coef(svydbtotal(x = Race3, design = nh.dbsurv , num = T))
# [1] 959380842

> SE(svydbtotal(x = Race3, design = nh.dbsurv , num = T))
#    Race3
# 61432595
```

2.2.4 Difficulties

1. In SQL if a column contains only numbers, it is not possible to identify whether a column type is factor or numeric, therefore in the `svydbtotal()` function, the user needs to specify it. `num = TRUE`: Numeric, `num = FALSE`: Factor.

2. To compute the population total for a categorical variable, dummy variables are needed. In SQL, there are two ways to do this. The first way would be to create new columns for every levels of the variable manually and apply `ifelse/CASE WHEN` to each of the columns. Another way would be to create a small contrast table in memory and use `INNER JOIN` to join the small table onto the data set. The second approach is much faster, and the function `dummy_mut()` adapts that approach. It creates (mutate) new dummy columns on the right side of the data set.

3. Calculating the variance/standard error is the most complex part of `svydbtotal()`. Therefore the `svyVar()` function is written to calculate the variance of a variable. If the variable is a categorical variable with multiple levels, the calculations will be replicated with `sapply()`.

2.3 Population Mean

The function `svydbmean()` was designed to estimate the population mean in R by using **dplyr**, it is compatible with data frames and sql tables and was designed to do as much computation as possible in a database.

In the function `svydbmean()`, the mean is computed by using a ratio estimator rather than the Horvitz-Thompson estimator. This is a standard in survey software as N may not be known.

$$M\hat{e}an = \frac{1}{\hat{N}} \sum_{h=1}^{L} \sum_{i=1}^{m_h} z_{hi}$$

where,

$$\hat{N} = \sum_{j \in PSU} w_j, \quad z_{hi} = \sum_{j \in PSU} w_{hij} x_{hij}$$

- L = number of stratum
- m_h = number of clusters in stratum h
- w_{hij} = the sample weight in stratum h and cluster i for observation j.

Variance estimation of the mean uses the Horvitz-Thompson estimator on an influence function.

$$Var(M\hat{e}an) = \sum_{h=1}^{L} \frac{m_h}{m_h - 1} \sum_{i=1}^{m_h} (d_{hi} - \bar{d}_h)^T (d_{hi} - \bar{d}_h)$$

where,

$$d_{hi} = \frac{1}{\hat{N}} \sum_{j \in PSU} w_{hij}(x_{hij} - \bar{x}), \quad \bar{d}_h = \frac{1}{m_h} \sum_{i=1}^{m_h} d_{hi}$$

2.3.1 Usage

```
svydbmean(x, num = T, return.mean = F, design)
```

2.3.2 Arguments

- x = Name indicating the variable.
- *num* = `TRUE` or `FALSE` indicating whether x is numeric or categorical.
- *return.mean* = `TRUE` to return only means, no standard errors.
- *design* = svydb.design object.

2.3.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                id = SDMVPSU, data = nhdb)
> svydbmean(x = Race3, design = nh.dbsurv , num = T)
#          Mean      SE
# Race3 3.1292 0.0674

> svydbmean(x = Race3, design = nh.dbsurv, num = F)
#              Mean      SE
# Race3_1 0.097238 0.0208
# Race3_2 0.069853 0.0154
# Race3_3 0.628595 0.0407
# Race3_4 0.124373 0.0239
# Race3_6 0.050620 0.0080
# Race3_7 0.029322 0.0045
```

Generic functions like `coef()` and `SE()` were also implemented to extract the coefficients and standard errors from a **svydbstat** object. This is useful since the **svydbstat** objects are rounded when they are printed, by using `coef()` and `SE()`, the unrounded value can be extracted.

```
> class(svydbmean(x = Race3, design = nh.dbsurv, num = T))
# [1] "svydbstat"

> coef(svydbmean(x = Race3, design = nh.dbsurv, num = T))
# [1] 3.129191

> SE(svydbmean(x = Race3, design = nh.dbsurv, num = T))
#      Race3
# 0.06735437
```

2.3.4 Difficulties

1. In SQL cannot recognise factor variables. Explained in difficulty 1 from chapter 2.2 (Population Total).

2. SQL does not have built-in support for dummy variables. Explained in difficulty 2 from chapter 2.2 (Population Total).

3. Difficult to compute the variances. Explained in difficulty 3 from chapter 2.2 (Population Total).

2.4 Regression

The function `svydblm()` was designed to fit a linear model to survey data in R by using **dplyr**, it is compatible with data frames and sql tables and was designed to do as much computation as possible in a database.

In the function `svydblm()`, the coefficients are computed by,

$$\hat{\beta} = (X^T WX)^{-1}(X^T WY)$$

- W = Sampling weights

Variance estimation of the coefficients uses a similar approach as survey mean/total,

$$Var_{pq}(\hat{\hat{\beta}}) = \sum_{h=1}^{L} \frac{m_h}{m_h - 1} \sum_{i=1}^{m_h} (z_{hip} - \bar{z}_{hp})^T (z_{hiq} - \bar{z}_{hq})$$

where,

$$z_{hi} = x_{hij} w_{hij} (y_{hij} - \mu_{hij}), \quad \bar{z}_h = \frac{1}{m_h} \sum_{i=1}^{m_h} z_{hi}$$

And by using the sanwich estimator,

$$cov(\hat{\beta}) = (X^T WX)^{-1} Var_{pq}(\hat{\beta})(X^T WX)^{-1}$$

- L = number of stratum
- m_h = number of clusters in stratum h
- w_{hij} = the sample weight in stratum h and cluster i for observation j.
- p, q = Indicator function for variables p and q.

2.4.1 Usage

```
svydblm(formula, design)
```

2.4.2 Arguments

- $formula$ = Model formula.
- $design$ = svydb.design object.

2.4.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                        id = SDMVPSU, data = nhdb)
> svydblm(DirectChol ~ Age + BMI + factor(Gender),
    design = nh.dbsurv)
# svydb.design, 9756 observation(s), 31 Clusters
#
# Survey design:
# svydbdesign(st = SDMVSTRA, id = SDMVPSU, wt = WTMEC2YR,
#    data = nhdb)
#
# Call:
# svydblm(formula = DirectChol ~ Age + BMI + factor(Gender),
#    design = nh.dbsurv)
#
# Coefficients:
#       intercept   Age         BMI         Gender_2
# [1,]   1.632111   0.003254   -0.018636   0.218086
```

The summary() function to obtain the summary of the model and predict() function to predict using new data-sets with the model was also implemented,

```
# dbfit = svydblm(formula = DirectChol ~ Age + BMI,
#   design = nh.dbsurv)

> summary(dbfit)
# Call:
# svydblm(formula = DirectChol ~ Age + BMI,
#   design = nh.dbsurv)
#
# Survey design:
# svydbdesign(st = SDMVSTRA, id = SDMVPSU, wt = WTMEC2YR,
#   data = nhdb)
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# intercept  1.7263135  0.0279703  61.719  < 2e-16 ***
# Age        0.0034161  0.0003428   9.966 5.23e-08 ***
# BMI       -0.0182468  0.0011277 -16.181 6.63e-11 ***
# ---
# Signif. codes:  0    ***    0.001    **    0.01    *
#   0.05    .    0.1         1

> predict(dbfit, newdata = data.frame(Age = 1:3, BMI = 4:6))
#     link     SE
# 1 1.6567 0.0242
# 2 1.6419 0.0233
# 3 1.6271 0.0225
```

Other generic function includes `coef()`, `SE()`, `vcov()` and `residuals()`.

```
# dbfit = svydblm(formula = DirectChol ~ Age + BMI,
#   design = nh.dbsurv)

> coef(dbfit)
#      intercept         Age         BMI
# [1,]  1.726314 0.003416124 -0.01824676

> SE(dbfit)
#    intercept          Age          BMI
# 0.0279703171 0.0003427746 0.0011276877

> vcov(dbfit)
#                intercept           Age           BMI
# intercept  7.823386e-04  3.769663e-06 -2.806340e-05
# Age        3.769663e-06  1.174944e-07 -2.496403e-07
# BMI       -2.806340e-05 -2.496403e-07  1.271680e-06

> head(residuals(dbfit), 3)
# Source:   lazy query [?? x 1]
# Database: MonetDBEmbeddedConnection
#   residuals
#       <dbl>
# 1    -0.316
# 2    -0.318
# 3    -0.733
```

2.4.4 Difficulties

1. SQL does not have built-in support for dummy variables. Explained in difficulty 2 from chapter 2.2 (Population Total).

2. In SQL matrix multiplications are not supported, however, it can still be implemented since matrix multiplications only requires addition and multiplication. For example with matrix X, by calculating the sums of products of the first column of the matrix with the rest of the columns (including the first column), we get the first row of the X^TX matrix. To get the whole X^TX matrix we only need to repeat the process with different columns. However, the inverse of a matrix is not possible in SQL, so to compute $(X^TX)^{-1}$, we need to pull the matrix into memory.

3. The variance for the regression coefficients are a bit more complicated than computing the variance for mean/total, since for mean/total we only need the diagonal of the co-variance matrix. However, the variances of the regression coefficients requires the whole covariance matrix. This means that more replication with different combinations of z_{hip}/z_{hiq} will be needed, but we only need the upper triangle or the lower triangle of the matrix, since it is a symmetric matrix.

2.5 Quantiles

The function `svydbquantile()` was designed to compute the medians/quantiles from survey data in R by using **dplyr**, it is compatible with data frames and a **few types** of sql tables and was designed to do as much computation as possible in a database.

To estimate the median/quantiles, a standard probabilistic algorithm is used. For example to estimate the median,

1. Take a sample of size $n^{2/3}$ from the data set and read it into memory. (Proof below)
2. Compute the 99% confidence interval $[a, b]$ of the 0.5 quantile by using `svyquntile()` from the survey package.
3. Read in the data set where the observations are between a and b.
4. Sort the data and compute the cumulative sum of the weights of the read in observations, $w_n = \sum_{i=1}^{n} w_i$
5. Find out if the median is within the read in data set. Since $median = 0.5 \times W$, we can find out by searching if median equals or is between w_n.
6. If the median is not found, repeat.

Other quantiles are computed with the same method.

Though by using this method to compute the survey quantile requires at least two sets of data to be read into memory, however it is still much more efficient than sorting and calculating the cumulative sum for the whole data set, since $n^{2/3}$ is relatively small compared to whole data set (One million observations, $10000000^{2/3} = 10000$).

Let M be a random sample of N.

The first set of data read into memory has M points and its confidence interval length is $\propto M^{-1/2}$. (step 1)

Number of points in the second read is $\beta N M^{-1/2}$. (step 3)

Total number of points read in is $M + \beta N M^{-1/2}$. Minimise,

$$\frac{\partial}{\partial M} = 1 + N\left(\frac{-1}{2}M^{-3/2}\right) = 0$$

$$1 = \frac{1}{2}NM^{-3/2}$$

$$2M^{-3/2} = N$$

$$M \propto N^{2/3}$$

2.5.1 Usage

```
svydbquantile(x, quantiles = 0.5, design)
```

2.5.2 Arguments

- *x* = Name indicating the variable.
- *quantiles* = Quantiles to estimate, a number, or a vector of numbers for multiple quantiles. Default to 0.5.
- *design* = svydb.design object.

2.5.3 Examples

```
> db.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                          id = SDMVPSU, data = nhdb)
> svydbquantile(x = Age, quantile = 0.5, design = nh.dbsurv)
# 0.5
#  37

> svydbquantile(x = BMI, quantile = c(0.25,0.75),
                              design = nh.dbsurv)
# 0.25 0.75
# 21.7 30.6
```

2.5.4 Difficulties

1. To compute the survey quantile, a sample of the data-set is needed. However, different types of SQL uses different queries for sampling tables and the `sample_frac()` function from **dplyr** currently only works with **Spark** (Luraschi et al., 2018) database connections and local dataframes/tibbles. Sampling in **MonetDB** (MonetDB-B.V., 2008) was also implemented. Therefore currently, the `svydbquantile()` is only tested with local dataframes and database connections that are mentioned above. It may be possible that it is compatible with other types of connections but they are to be tested. Another possibility is that the `sample_frac()` function will be extended in the future to support more database connections.

2. Since to compute the survey quantile in `svydbquantile()`, the inputted data-set will be sized down into $n^{2/3}$, so if the inputted data-set is a pre-subsetted data-set then it means that the data-set will be subsetted at least twice to compute the quantiles. This could be a problem because `svydbquantile()` runs through `svyquantile()` form the **survey** package which uses `svymean()` within it. If the data-set is too small it may lose enough information and may cause mathematical errors. For example, if there are only one cluster within a strata then it will cause the equation to divide by zero.
   To overcome this, there is a option within the **survey** package called "survey.loney.psu", if we set this option to "adjust", e.g. `options("survey.lonely.psu" = "adjust")`, it will allow survey statistic computations even if there is only one cluster within a strata.

2.6 Survey Tables

The function `svydbtable()` was designed to create contingency tables for survey data, it is compatible with data frames and sql tables and was designed to do as much computation as possible in a database.

Each cell within the table is computed with the same method as `svydbtotal()`.

2.6.1 Usage

```
svydbtable(formula, design, as.local = F)
```

2.6.2 Arguments

- *formula* = A formula specifying margins for the table, only + can be used.
- *design* = svydb.design object.
- *as.local* = A logical value indicating the returning object type. Default is database tables, `tbl_sql`.

2.6.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                          id = SDMVPSU, data = nhdb)

> svydbtable(~MaritalStatus, design = nh.dbsurv)
# Source:     lazy query [?? x 2]
# Database:   MonetDBEmbeddedConnection
# Ordered by: MaritalStatus
#   MaritalStatus         wt
#           <int>      <dbl>
# 1             1 118752657.
# 2             2  12600347.
# 3             3  23868539.
# 4             4   5486968.
# 5             5  44543092.
# 6             6  18664186.

> svydbtable(~MaritalStatus, design = nh.dbsurv,
    as.local = T)
#   MaritalStatus         wt
#           <int>      <dbl>
# 1             1 118752657.
# 2             2  12600347.
# 3             3  23868539.
# 4             4   5486968.
# 5             5  44543092.
# 6             6  18664186.
```

```
> svydbtable(~Race3 + Smoke100, design = nh.dbsurv,
    as.local = T)
# A tibble: 6 x 5
#   Race3 Smoke100_1 Smoke100_2 Smoke100_7 Smoke100_9
#   <int>      <dbl>      <dbl>      <dbl>      <dbl>
# 1     1   6177437.  11124548.          0          0
# 2     2   5375656.   9313414.          0          0
# 3     3  71234238.  77533263.          0      29247.
# 4     4   9687657.  16031251.     22142.      14686.
# 5     6   3019921.   8645276.          0      14966.
# 6     7   3070772.   2650765.          0          0


> svydbtable(~Race3 + Work + Gender, design = nh.dbsurv,
    as.local = T)

# $`Gender = 1`
# A tibble: 6 x 6
#   Race3    Work_1   Work_2   Work_3    Work_4 Work_7
#   <int>     <dbl>    <dbl>    <dbl>     <dbl>  <dbl>
# 1     1  7063412.  118589.  805525.  2148047. 54555.
# 2     2  5066507.   50966.  283436.  2056117.     0
# 3     3 48916000. 1670881. 3052152. 22756262.     0
# 4     4  6557730.  166729.  812680.  5201428.     0
# 5     6  3689443.   79793.  191406.  1764683.     0
# 6     7  1747070.   34664.  258597.  1182825.     0

# $`Gender = 2`
# A tibble: 6 x 6
#   Race3    Work_1   Work_2   Work_3    Work_4 Work_7
#   <int>     <dbl>    <dbl>    <dbl>     <dbl>  <dbl>
# 1     1  4839422.   80842.  273991.  4040679.      0
# 2     2  4052770.   20057.  400463.  3907392.      0
# 3     3 42775016. 1370755. 2995602. 34142499.      0
# 4     4  7486159.  280862.  915095.  7070385.      0
# 5     6  3557006.  111248.  209497.  2803386.      0
# 6     7  1810965.   17984.  114653.  1255186.      0
```

2.7 Survey Statistic on Subsets

The function `svydbby()` was designed to compute survey statistics on subsets of the data in R by using **dplyr**, it is compatible with data frames and sql tables and was designed to do as much computation as possible in a database.

This function creates a number of subsets based on the conditions given by the user and computes the desired survey statistic on all the subsets. Currently, it is only compatible with `svydbtotal()` and `svydbmean()`.

2.7.1 Usage

```
svydbby(x, by, FUN, design, ...)
```

2.7.2 Arguments

- *x* = A variable specifying the variable to pass to FUN.
- *by* = A variable specifying factors that define the subsets.
- *FUN* = A function indicating the desired survey statistics.
- *design* = svydb.design object.
- ... = Other arguments to pass to FUN.

2.7.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
                          id = SDMVPSU, data = nhdb)
> svydbby(x = Age, by = Gender, FUN = svydbmean,
                design = nh.dbsurv, num = T)
# $Age
#                 Mean        SE
# Gender == 1 36.21035 0.8387459
# Gender == 2 38.09899 0.6721789

> svydbby(x = BMI, by = Race3, FUN = svydbtotal,
                design = nh.dbsurv, num = T)
# $BMI
#                 Total        SE
# Race3 == 3 5004822100 612188966
# Race3 == 1  737939057 153551662
# Race3 == 6  347435902  52876464
# Race3 == 4 1020476221 154630793
# Race3 == 7  227729724  39637006
# Race3 == 2  550207484 118331941
```

2.8 Replicate Weights

In survey data sets, standard errors can never be known with any certainty and are only estimated. Replicate weights lets us to use a single sample with different sampling weights to capture the characteristics of multiple samples, it allows us to compute more informed standard error estimates, this method is similar to bootstrap and jackknife sampling. Though computing standard errors from replicate weights usually result in them getting bigger, but the increase usually is not large enough that it can alter the significance level.

Another reason for us to use replicate weights is that it provides a less complex way to compute the standard errors.

Currently, replicate weights are available in a number of data sets, for example the American Community Survey and Puerto Rican Community Survey data. In these data sets, there are 80 separate replicate weights at the household/person level which allows us to compute the standard errors.

2.8.1 Replicate Standard Errors

To compute the replicate standard errors, there are three steps.

1. Compute the survey statistics of interest with the full sample weights.
2. Rerun the analysis with each set of the replicate weights.
3. Calculate the standard error,

$$SE(X) = \sqrt{s \sum_{r=1}^{n(r)} (X - X_r)^2}$$

- X = Result of the survey statistics using the full sample weights.
- X_r = Result of the survey statistics using the r'th set of the replicate weights.
- s = Scale multiplier. i.e. $\frac{4}{80}$ for the American Community Survey.

2.8.2 Survey Replicate Design

Similarly, there is a survey replicate design like the survey design from section 2.1.

Currently, replicate statistics that are supported are survey totals and means. Therefore, there is no need to provide the stratification and clustering information to `svydbrepdesign()`.

```
svydbrepdesign(wt, repwt, scale, data)
```

- *wt* = Column name specifying the sampling weights column.
- *repwt* = A regular expression that matches the names of the replication weight variables.
- *data* = A data frame or sql table of the survey data set.

2.8.3 Examples

```
> hde.repsurv = svydbrepdesign(wt = WGTP, repwt="wgtp[0-9]+",
    scale = 4/80, data = ss16hde)
> hde.repsurv
# svydb.repdesign, 4582 observation(s),
#   80 sets of replicate weights, scale = 0.05
```

2.8.4 Population Total with replicate weights

Arguments are the same as `svydbtotal()`, but with an extra argument.

- *return.replicate* = *TRUE* to return the replicate statistics.

```
> hde.dbrepsurv = svydbrepdesign(wt = WGTP,
  repwt = "wgtp[0-9]+", scale = 4/80, data = ss16hde)
> svydbreptotal(x = BATH, design = hde.dbrepsurv , num = T)
#       Total     SE
# BATH 429410 755.04

> svydbreptotal(x = FS, design = hde.dbrepsurv , num = F)
#       Total     SE
# FS_1  37392 2151.8
# FS_2 313694 3033.6

> (svydbreptotal(x = HHT, design = hde.dbrepsurv ,
      num = T, return.replicates = T)$replicates)[,1:3]
#    wgtp1  wgtp2  wgtp3
#    <dbl>  <dbl>  <dbl>
# 1 975397 973284 987185

> coef(svydbreptotal(x = BATH, design = hde.dbrepsurv,
      num = T))
# [1] 429410

> SE(svydbreptotal(x = BATH, design = hde.dbrepsurv,
  num = T))
#     BATH
# 755.0406
```

2.8.5 Population Mean with replicate weights

All arguments are the same as `svydbreptotal()`.

```
> svydbrepmean(x = BATH, design = hde.dbrepsurv , num = T)
#        Mean     SE
# BATH 1.0076 0.0018

> svydbrepmean(x = FS, design = hde.dbrepsurv , num = F)
#        Mean     SE
# FS_1 0.1065 0.0061
# FS_2 0.8935 0.0061
```

Chapter 3

Out-of-memory Graphics

Graphs are useful in statistics because it allows us to have a visual perspective of the data sets or to present them in a more meaningful way. However, when implementing graphics with survey data sets, it may become more difficult because the data sets are often large. For example, if every point of the large data set is plotted in a scatter plot, there will be too many points in the graph which will overlap and hide each other. There are two ways to over come this problem,

1. Use different symbols, colour or sizes to represent different points or group of points. This is only suitable for data sets that are moderately large. If the data sets are too large, the points will still overlap each other.

2. Condense multiple points into one point with an algorithm and use colours to represent the density of the sampling weights within the point. This will be discussed with more detail in section 3.3.

Another approach would be to take a sample from the full data set based on the estimated population distribution and plot the sample.

Currently, there are very few R packages which allows us to plot graphs with data sets which are stored in a database. One of them is **dbplot** (Ruiz, 2018), it is simple and light-weighted but it is in its early development stage. As for survey data sets, except for the **sqlsurvey** package mentioned in section 1.1, there are no packages that will allows us to plot data sets with sampling weights without having the data sets stored in memory.

Like the **survey** package, **svydb** also provides a set of tools for plotting survey data sets and is implemented with **ggplot2** (Wickham, 2009), but with less flexibility and options. Similar to the previous chapters, every graphical functions within **svydb** is designed to do as much computation in the database as possible.

For some type of graphs, it is reasonable to produce them with data sets outside of memory, since to plot them, we don't need the whole data set in memory (computing the statistics for the graphs within the database may even be faster than transferring the whole data set into memory). For example, plotting a histogram with one variable, the only information that are needed to plot it is the location of the breaks for the x-axis and the density of the corresponding break for the y-axis. Say if there are 30 breaks, then at most we will only need 60 numbers in memory, 30 for the breaks and 30 for the density.

An example of a graph that is not possible to plot without having the whole data set in memory is scatter plot, since we will need to know the location of all the points.

3.1 Histogram

Traditionally, the heights of each bin for a histogram while plotting the frequency, the height of each bin is determined by the number of values that are within the ranges of a certain bin, and plotting the density is determined by the proportion of data in each bin divided by the width of each bin.

To calculate the proportions, we first need to determine which bin each value is in. To do this we can use the `base::cut()` and perform `svydbmean()` on the cuts. However this function is not compatible with database tables, therefore a new function `db_cut2` was implemented to overcome this difficulty, this will be discussed with more detail in section 3.1.4.

3.1.1 Usage

```
svydbhist(x, design, binwidth = NULL, xlab = "x", ylab = "Density")
```

3.1.2 Arguments

- x = Name indicating the variable.
- *design* = svydb.design object.
- *binwidth* = The width of each bin. Binswidths are calculated with Sturges' formula (Sturges, 1926) by default, $k = [log_2 n] + 1$..
- $xlab, ylab$ = labels for xlab and ylab.

3.1.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
    id = SDMVPSU, data = nhdb)
> svydbhist(x = DirectChol , design = nh.dbsurv,
    binwidth = 0.25)
```

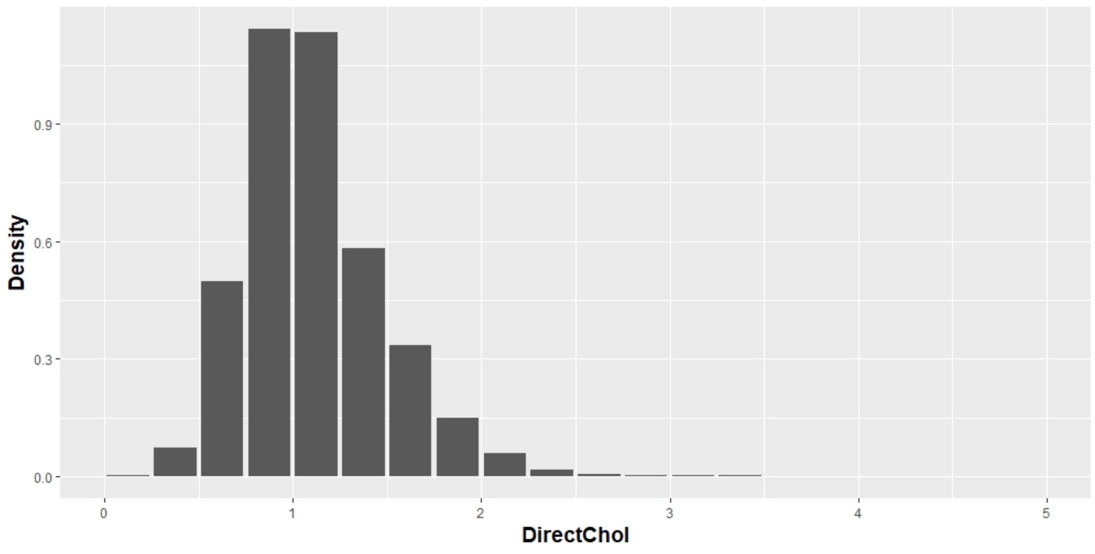

FIGURE 3.1: svydbhist example

3.1.4 Difficulty

1. In R, it is simple to divide numeric columns into intervals by using `base::cut()`, but this function is not supported in SQL. To overcome this problem a new function was written, `db_cut2`, it is designed to be compatible with SQL tables and is a simple version of `base::cut()`.

3.2 Box Plot

Boxplots are based on the quantiles of each variables or groups of variables. Traditionally, by default in R, the boxes of the boxplot only uses the median, 25th percent and 75th percent quantile. The ends of the boxplots only extends out to the observations that are within the 1.5 $\times$ interquantile range, the rest of the observation outside this range are plotted as points. In `svydbboxplot` it is the opposite, due to efficiency.

3.2.1 Usage

```
svydbboxplot(x, groups = NULL, design, varwidth = F, outlier = F)
```

3.2.2 Arguments

- $x, groups$ = If `groups` is defined, boxes of x will be split by `groups`.
- *design* = svydb.design object.
- *varwidth* = If `varwidth = T`, the width of the boxes will be proportional to the number of observations in that box.
- *outlier* = If `outlier = T`, Any observations above or below the $1.5IQR$ will be plotted as points.
- *all.outlier* = `TRUE` to plot all the outlier points, default `FALSE`.

3.2.3 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
    id = SDMVPSU, data = nhdb)
> svydbboxplot(x = Weight, groups = Race3,
    design = nh.dbsurv, outlier = T,
    all.outlier = F, varwidth = T)
```

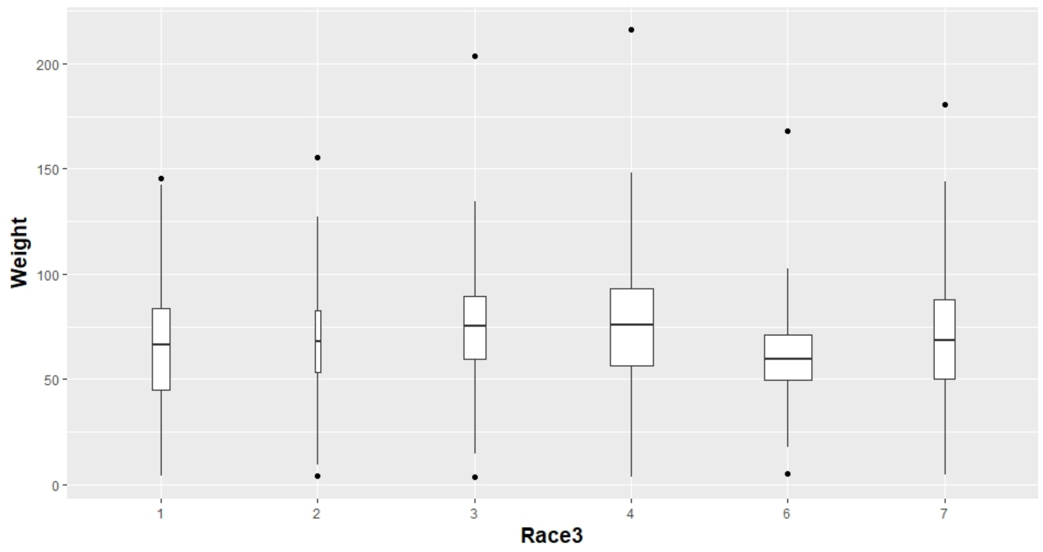

FIGURE 3.2: `svydbboxplot` example

3.3 Hexagon Binning

As mentioned at the beginning of this chapter, since survey data sets can be large and scatter plots does not work well with large data sets because the points will hide or overlap each other. A solution would be to use hexagon binning (Carr et al., 1987), group points that are close to each other into a hexagon and use colours or sizes of the hexagons to represent the sampling weights of all the points in every hexagon. The reasons of using hexagons will not be discussed here, but it is proven to be efficient and less biased visually.

Briefly, the hexagon binning algorithm works as follows,

1. Figure 3.4 (Page 29) Panel 1 - The red point represents the point to be binned. Blue and black point represents the dual lattice, each point represents the corner of each rectangle.
2. Figure 3.4 (Page 29) Panel 2 - Figure out which rectangles are closest to the red point. The intersect of the two rectangles should contain the red point.
3. Figure 3.4 (Page 29) Panel 3 - Figure out which hexagon the red point should be in by calculating the distance between the point and the centres of two hexagons. The centre for the blue hexagon is the top left corner of the black rectangle and the centre for the black hexagon is the bottom right corner of the blue rectangle.
4. Repeat for the next point.

3.3.1 Usage/Arguments

To compute the hexagon bins:

```
svydbhexbin(formula, design, xbins = 30, shape = 1)
```

- *formula* = A formula indicating x and y. i.e. y x.
- *design* = svydb.design object.
- *xbins* = Number of bins on range of the x-axis.
- *shape* = plotting region, shape = height of y/width of x.

To plot the hexagon bins:

```
svydbhexplot(d, xlab = d$xlab, ylab = d$ylab)
```

- *d* = returning object of `svydbhexbin()`.
- *xlab*, *ylab* = labels for x and y axis.

3.3.2 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
    id = SDMVPSU, data = nhdb)
> hb = svydbhexbin(Height~Weight, design = nh.dbsurv)
> svydbhexplot(hb)
```

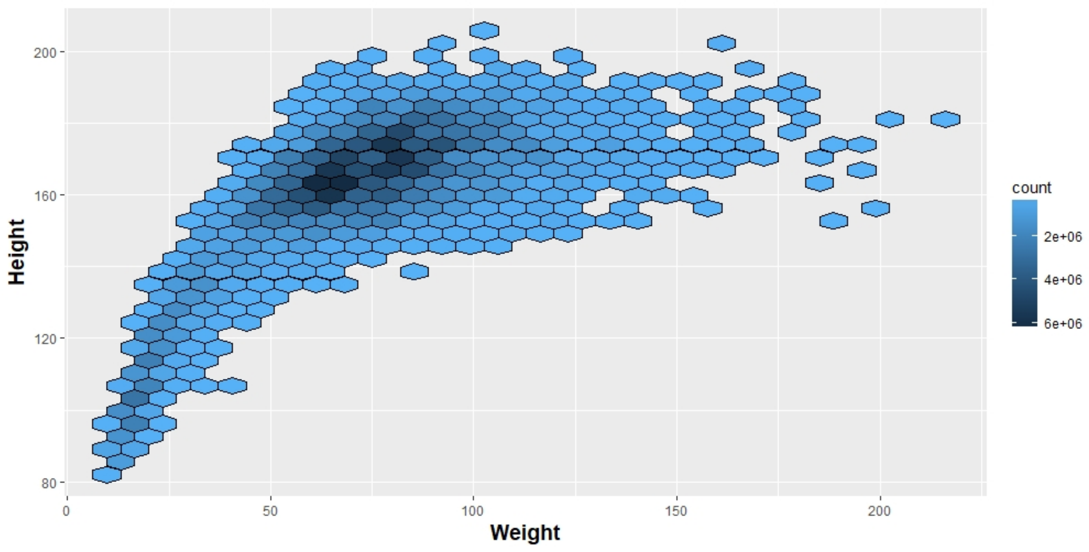

FIGURE 3.3: `svydbhexplot` example

3.3.3 Difficulties

1. The original code for hexagon binning in the **hexbin** R package (Carr et al., 2018) used for-loops to run the algorithm, but it is not possible in SQL. Therefore, new columns and extra calculations were needed to overcome this problem. This caused a efficiency problem.

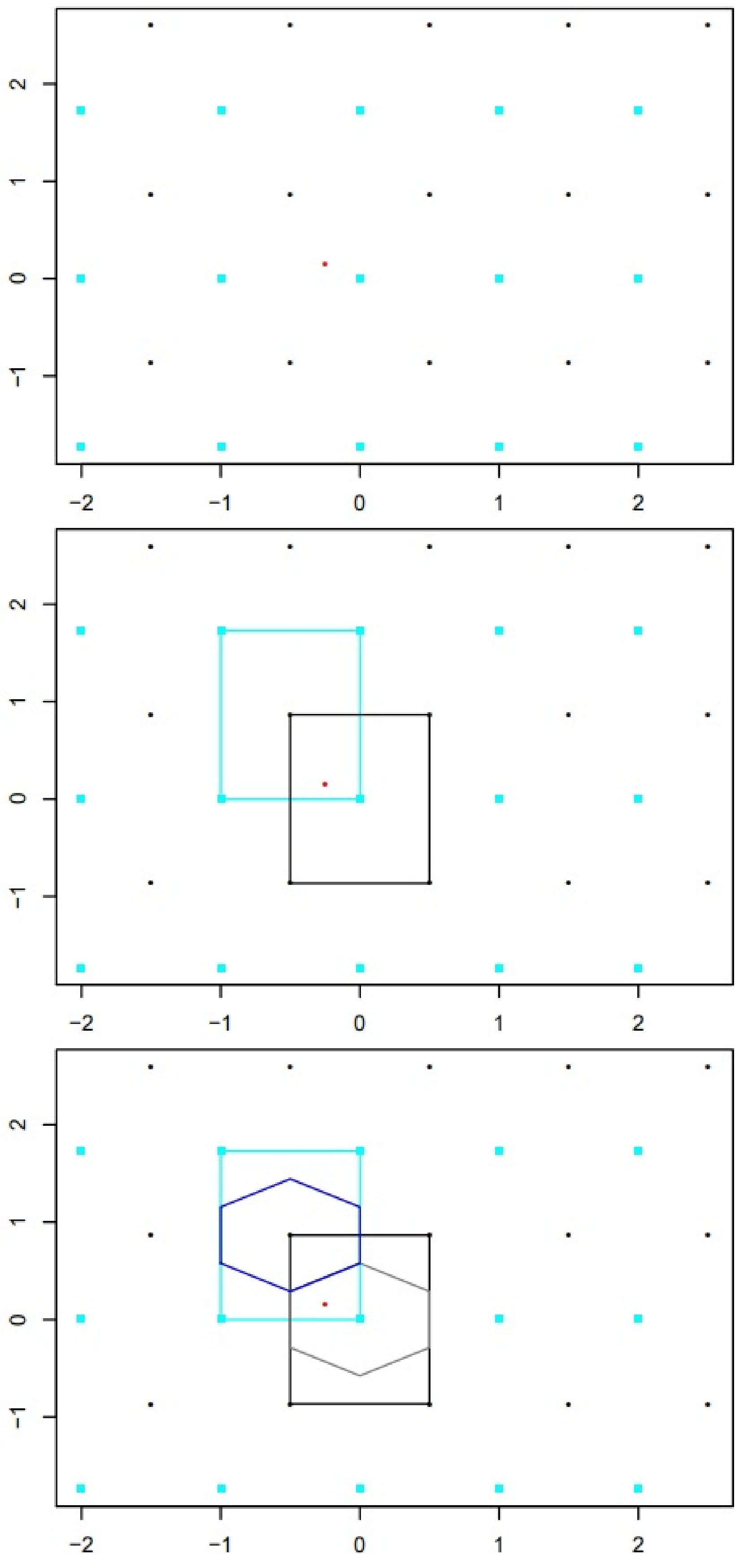

FIGURE 3.4: Hexagon binning explanation, (Lewin-Koh, 2016)

3.4 Conditional Plots

Conditional plots in **svydb** products sets of hexagon binning graphs conditioned by the condition given by the user, both x and y axis will remain the same for all graphs. Currently it only supports conditions applied on factors.

3.4.1 Function description

The `svydbcoplot()` function, has three basic arguments,

```
svydbcoplot(formula, by, design)
```

- *formula* = Formula indicating x and y. i.e. y x.
- *by* = Formula indicating the conditions of each plot. i.e by1 by2.
- *design* = svydb.design object.

3.4.2 Examples

```
> nh.dbsurv = svydbdesign(st = SDMVSTRA, wt = WTMEC2YR,
    id = SDMVPSU, data = nhdb)
> svydbcoplot(Age~Height, by = SmokeNow~Gender,
    design = nh.dbsurv)
```

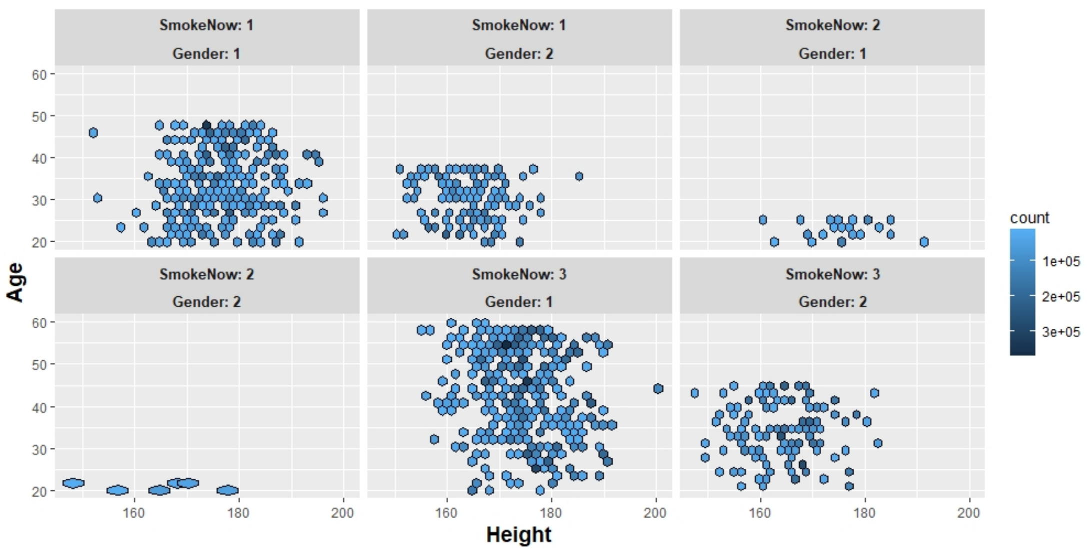

FIGURE 3.5: `svydbcoplot` example

3.5 Why ggplot?

There are several reasons why graphics in **svydb** is implemented with **ggplot2**, one of them is that **ggplot2**, **dplyr** and **dbplyr** is maintained by the same group of people and is a part of a project at Rstudio. Though currently, **ggplot2** is only compatible with data sets in memory, but it is possible this will change in the future. Another reason could be that the user would like to have a interactive plot, this done by **plotly** (Sievert et al., 2017).

However, the main reason is that it provides more options for the users in terms of customising the plots. In base graphics, it would be difficult to customise plots when the arguments provided does not include what we want, also it would be difficult for the author to consider all the arguments when creating a function.

A simple example is that, when a gg or a `ggplot` object is created,

```
> p = ggplot(mpg, aes(class, hwy))
```

Customising on the object is just as simple as adding other functions on,

```
> p + geom_boxplot(varwidth = T) +
    geom_jitter(width = 0.2) + coord_flip() +
    ggtitle("plot") + theme_bw()
```

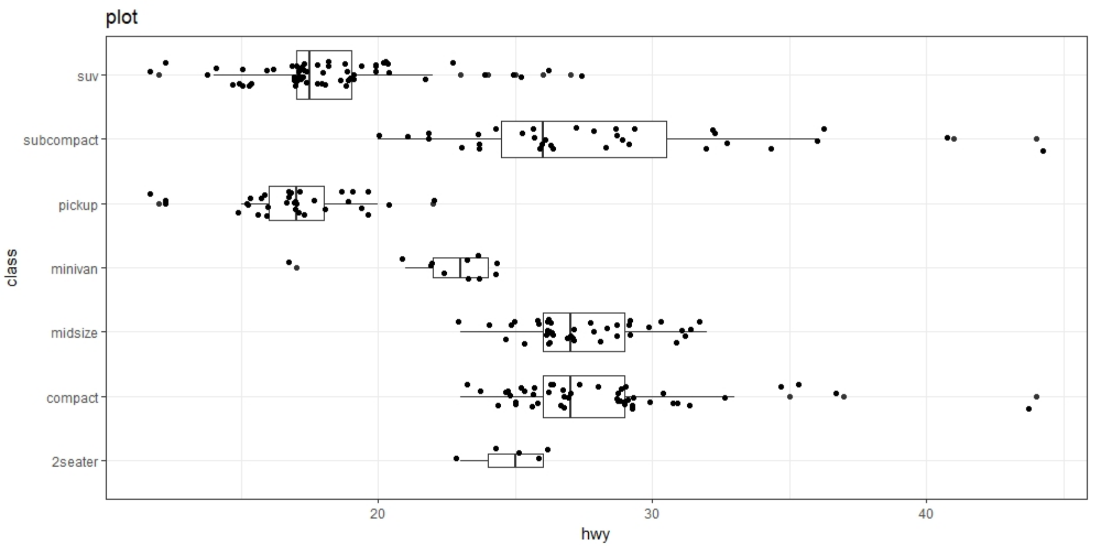

FIGURE 3.6: Why use `ggplot`

Or switching the style of the plot is as simple as changing the `geom`,

```
> p + geom_violin()
```

Chapter 4

Results

In this chapter, the time it takes to compute each survey statistics with **svydb** will be tested against the **survey** package, both data sets in memory (local) and in database will be used for **svydb**. Up to two million observations was tested with data sets in memory and up to four million observations for data sets in a data base, data set used was The National Health and Nutrition Examination Survey.

All computation was done with the same computer, specs as follows,

- **MacBook Pro Early 2015**
- **Processor** 2.7 GHz Intel Core i5
- **Memory** 8 GB 1867 MHz DDR3
- **Graphics** Intel Iris Graphics 6100 1536 MB

Data sets from 250,000 to 2,000,000 observations were tested on functions which computes replicate survey statistics, up to 4,000,000 observations for other survey statistics and graphics. For all the times, please refer to Appendix A.

Legend on graphs corresponds to,

- **survey** - **survey** package on local data sets.
- **svydb.local** - **svydb** on local data sets.
- **svydb.database** - **svydb** package on data sets stored in a database.

4.1 Total

4.1.1 Numeric variable

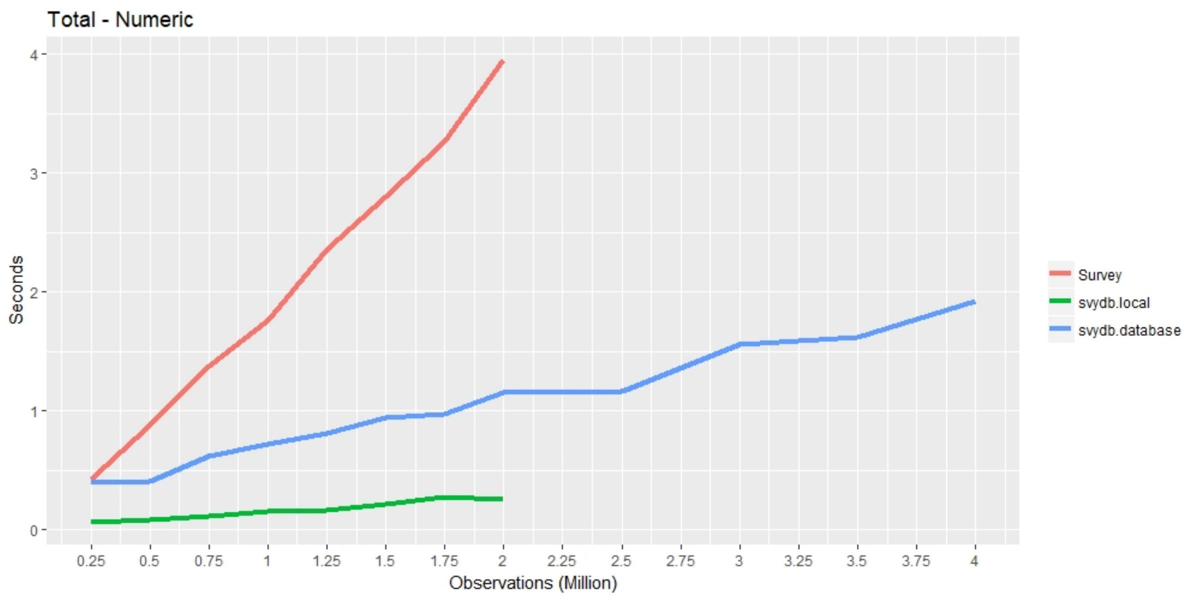

FIGURE 4.1: Population total time comparison - Numeric

4.1.2 Categorical variable with 7 levels

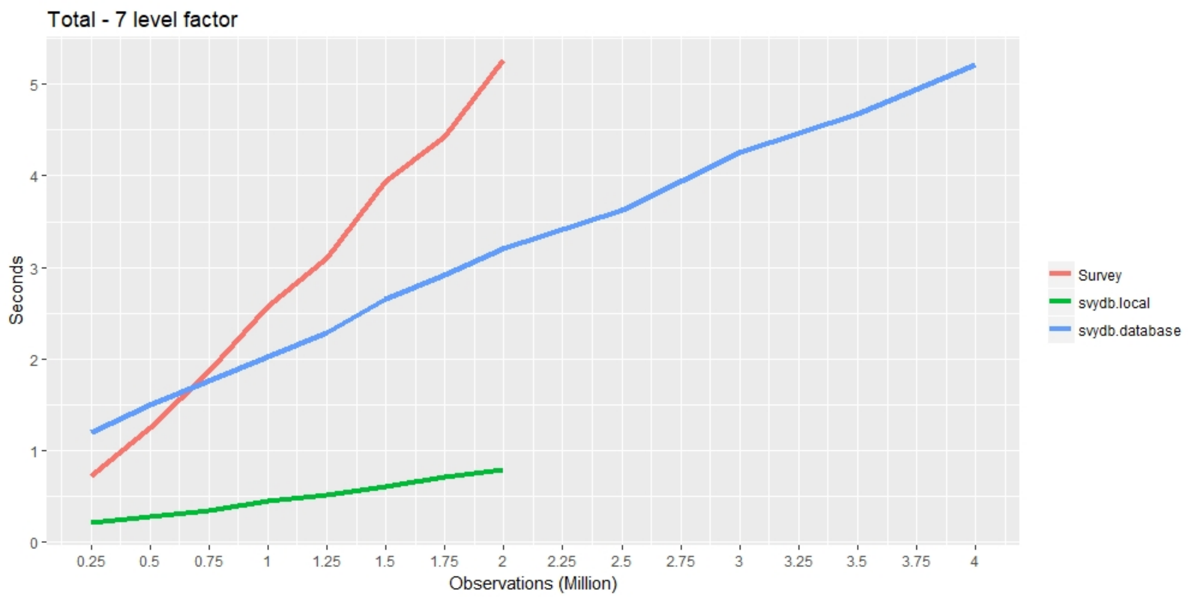

FIGURE 4.2: Population total time comparison - Categorical

4.2 Mean

4.2.1 Numeric variable

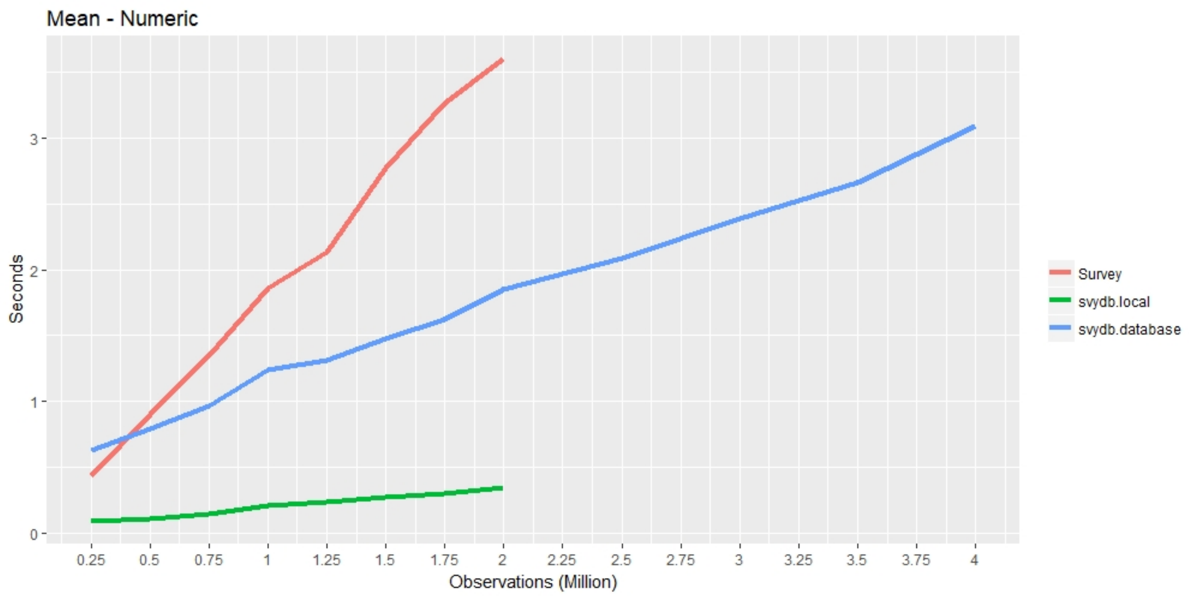

FIGURE 4.3: Population mean time comparison - Numeric

4.2.2 Categorical variable with 7 levels

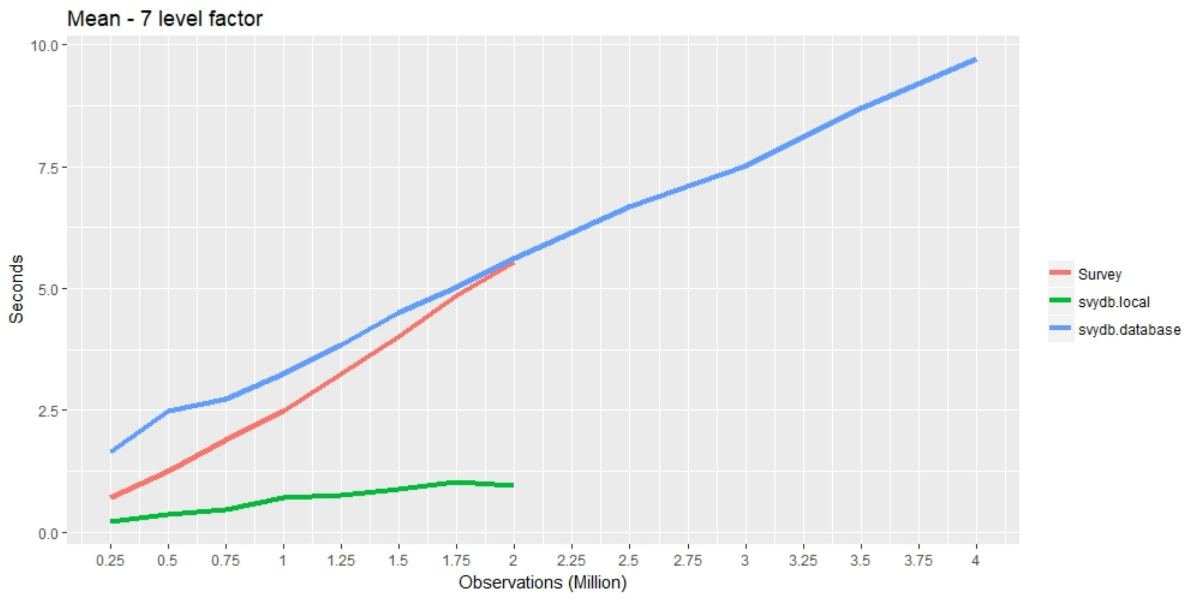

FIGURE 4.4: Population mean time comparison - Categorical

4.3 Regression

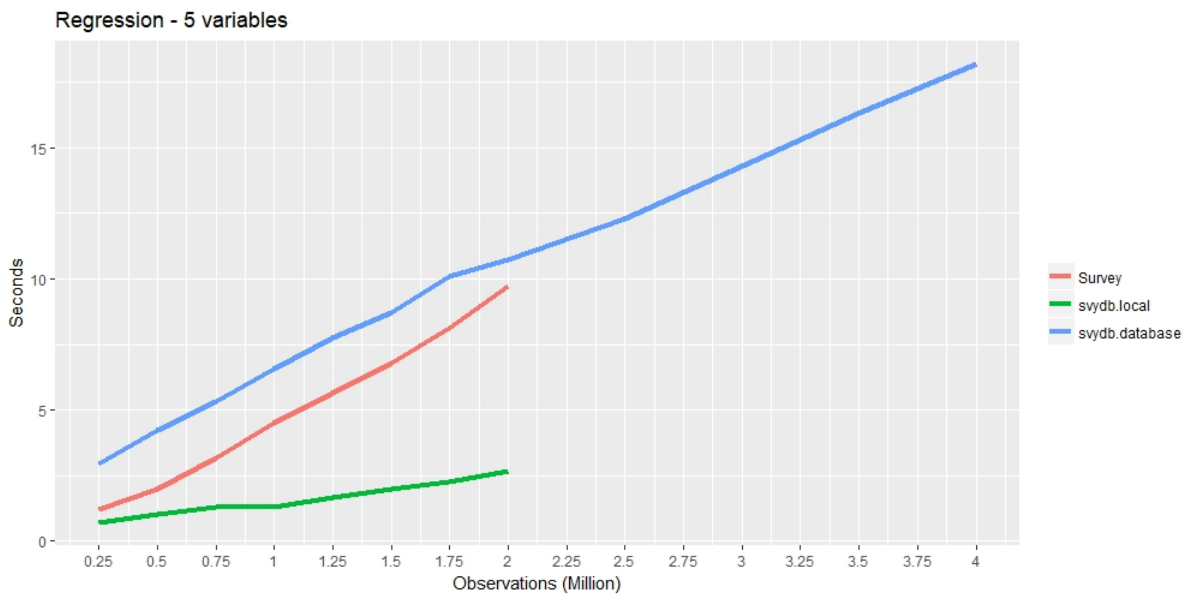

FIGURE 4.5: Regression time comparison - 5 variables

4.4 Quantiles

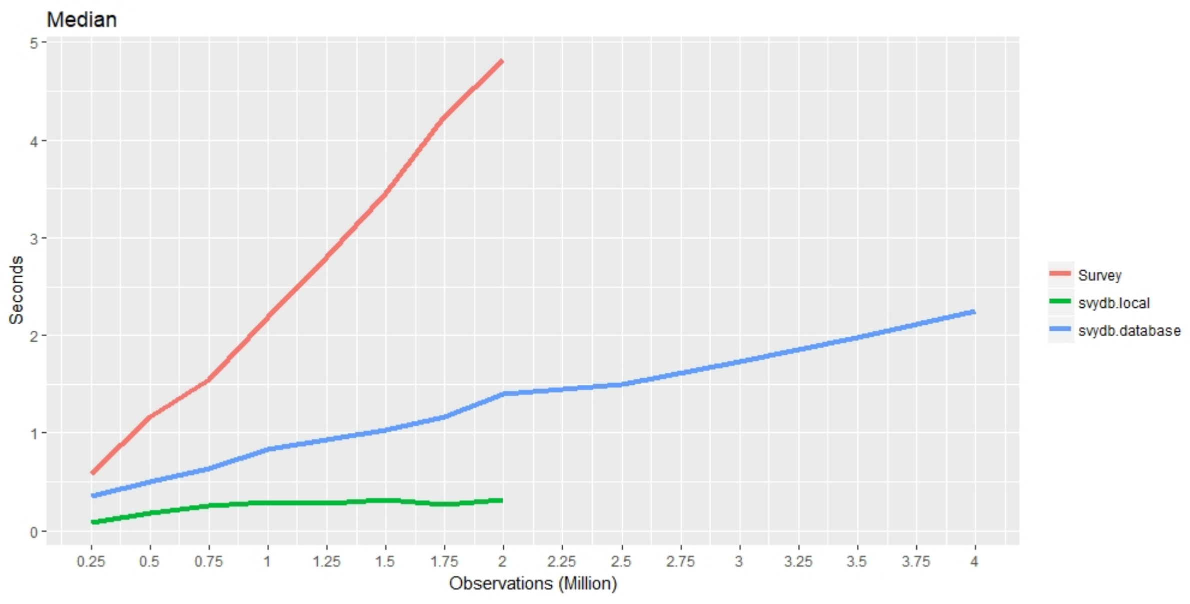

FIGURE 4.6: Median time comparison - Categorical

4.5 Survey Tables

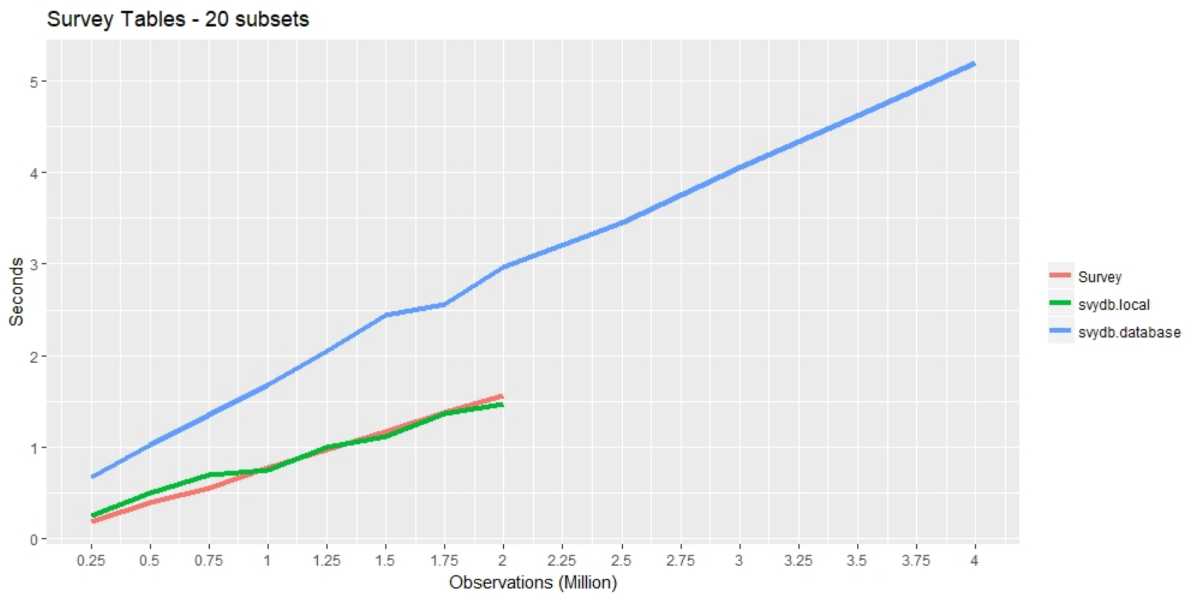

FIGURE 4.7: Tables time comparison - Categorical

4.6 Total - Replicate Weights

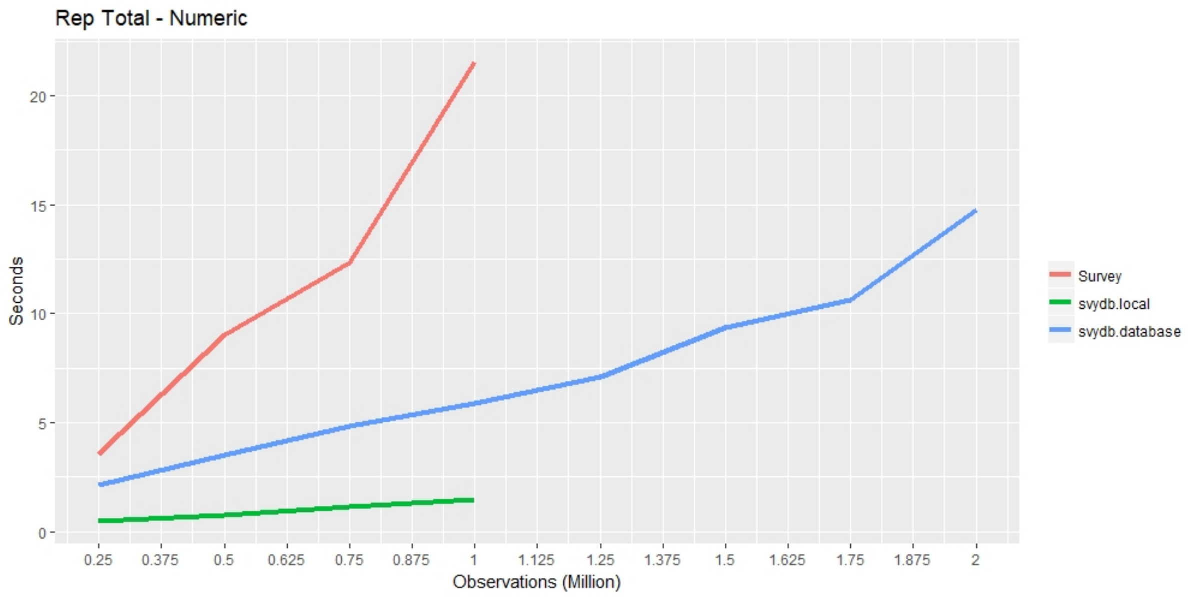

FIGURE 4.8: Replicate total time comparison - Numeric

4.7 Mean - Replicate Weights

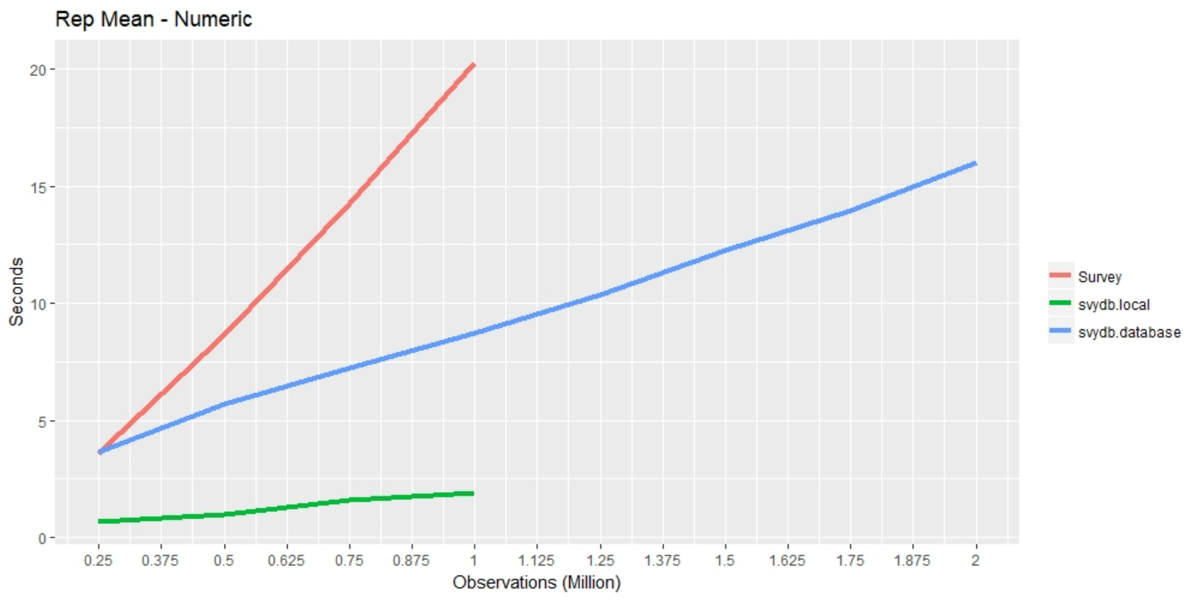

FIGURE 4.9: Replicate mean time comparison - Numeric

4.8 Histogram

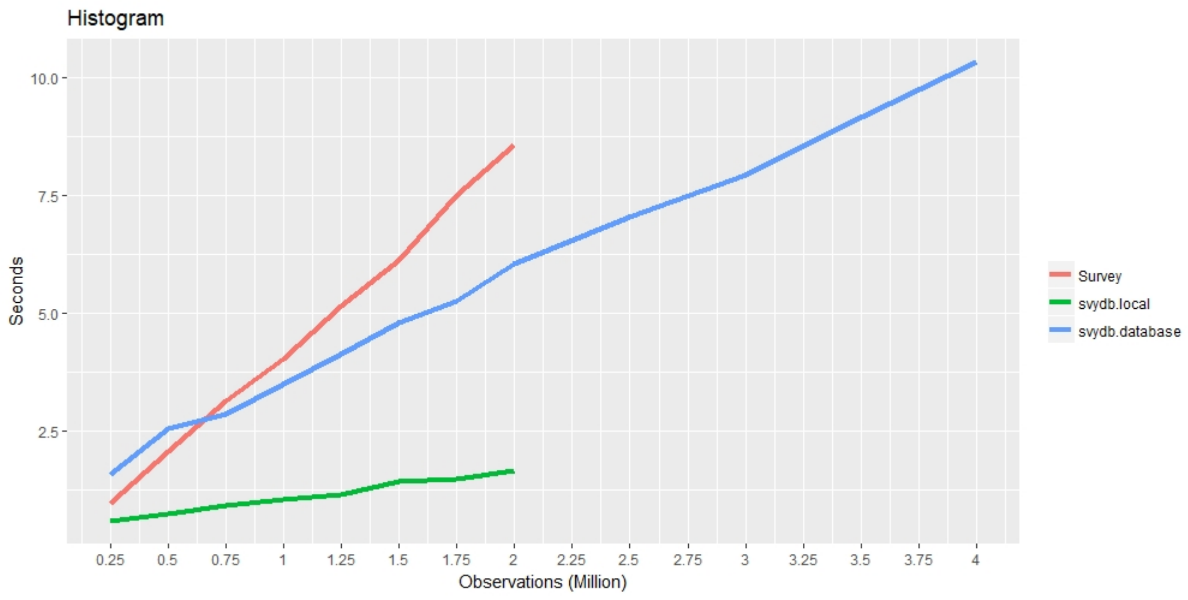

FIGURE 4.10: Histogram time comparison - Categorical

4.9 Boxplot

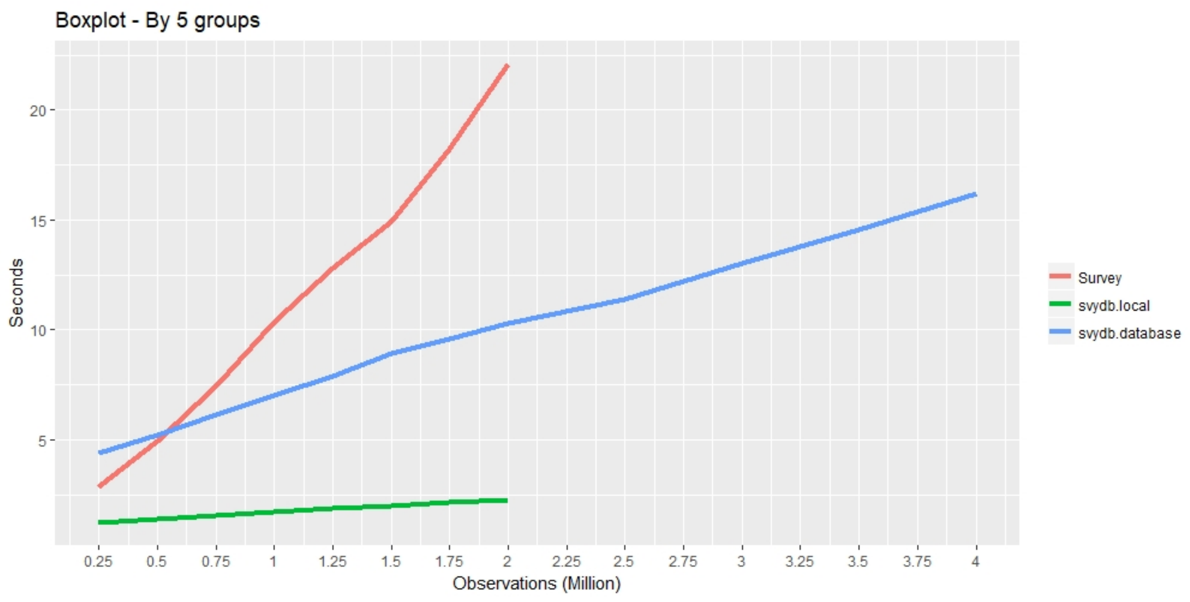

FIGURE 4.11: Boxplot time comparison

4.10 Hexagon Binning

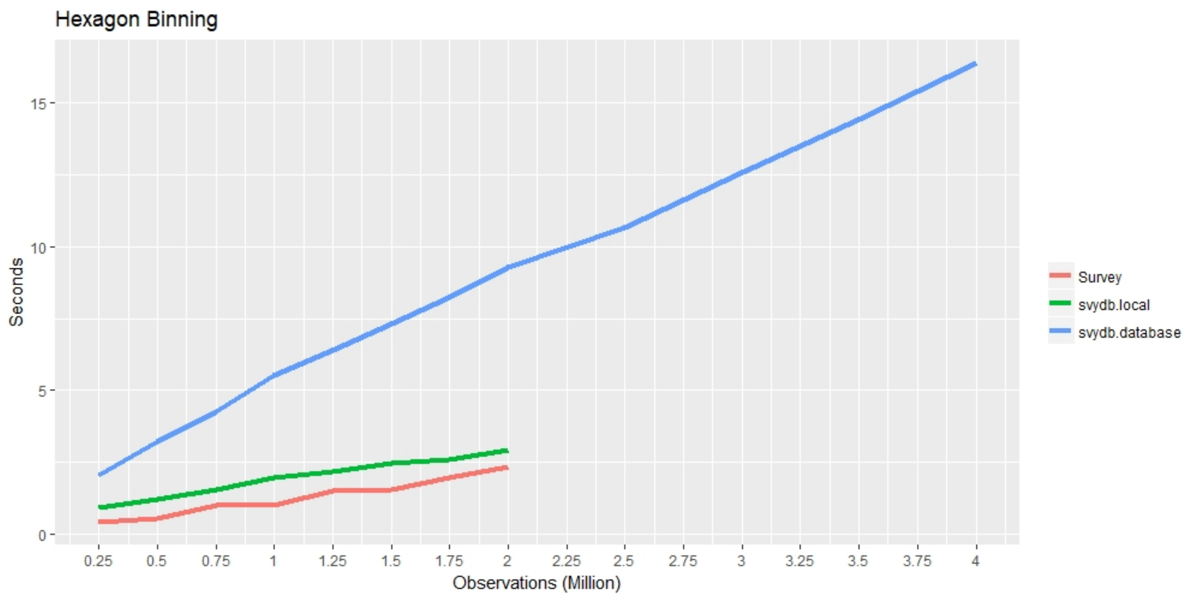

FIGURE 4.12: Hexagon Binning time comparison

4.11 Time Comparison

In general, using **svydb** with local data sets is always faster than both **svydb** with data sets in a database or with the **survey** package, except for computing the survey tables (Figure 4.7) and hexagon binning (Figure 4.12), where the times were similar for survey tables and hexagon binning was always about 0.5 seconds slower.

In terms of data sets in a database, If using **svydb** with local data sets is faster than the **survey** package then the time it takes to compute survey statistics in a database with **svydb** is predicted to be faster than the **survey** package if the data set is large enough. When the data set gets larger, the increase in time for the **survey** package is almost always larger than **svydb**. In some cases with data sets up to two million observations, **svydb** is already faster.

Most of the time we can see that **svydb** with data sets in memory is the fastest, followed by the **survey** package, and **svydb** with data sets in a database is always the slowest. The reason is that it takes time for the computer to communicate with the database, as in telling it what to do and collecting the results. However, if the data sets are too large and cannot be fitted into memory, **svydb** would be useful.

Some functions are just a replicate of another function but with more iterations and more features. For example, within **svydbboxplot()** (Figure 4.11), **svydbquantile()** (Figure 4.6) is called to obtain the quantiles, **svydbhist()** (Figure 4.10) calls **svydbmean()** (Figure 4.3, Figure 4.4) to obtain the proportions of each cut, and **svydbreptotal()** (Figure 4.8) is computed similarly to **svydbtotal()** (Figure 4.1, Figure 4.2) but with more iterations with different sets of weights. Therefore, their computation time would be similar across different sizes of data set but with some sort of a scale factor.

Regression (Figure 4.5) is the most interesting, though the slope of the line for **svydb.database** seems to be decreasing at 1.75 million observations, it is still around 1 to 1.5 seconds slower than the **survey** package at the two million observations mark. Since matrix operations are not supported in a database, to get the result of a matrix multiplication is to calculate it row by row which would be slow. Therefore **svydb.database** would only be expected to be faster than the **survey** package if the data set is large enough, or if the model contains enough explanatory variables which results in an unfeasible matrix operation.

Though it seems as if **svydb** is faster in most cases, but the structure of the **survey** package is more complex, and what it can do is outside the capability of **svydb**, for example, handling missing values, post-stratification and log models. At this stage it is unclear how much faster the **survey** package would be if was only computing statistics that are implemented in **svydb**.

Chapter 5

Usability

5.1 The package

With the help of the **devtools** package (Wickham, Hester, and Chang, 2018) for package building, the **roxygen2** package (Wickham, Danenberg, and Eugster, 2017) for package documentation amd the **formatR** (Xie, 2017) package for formatting. The full **svydb** package has been uploaded and can be found on Github, where it also includes basic help pages for each functions that computes survey statistics.

The dependant packages of **svydb** are,

- **R6 (Chang, 2017)**
- **dplyr (Wickham et al., 2017)**
- **rlang (Henry and Wickham, 2018)**
- **survey (Lumley, 2004)**
- **ggplot2 (Wickham, 2009)**

5.2 Supporting functions

Along with the main functions which computes survey statistics, there are also a number of other functions that were written to help with the implementation of the main functions, testing those functions and to overcome difficulties where there are inconsistency in terms of how to query between different types of databases. Details of these functions can be found in Appendix B.15.

5.3 Testing

Examples within the **survey** package help page was used to test **svydb**, all results were identical, except for when the data sets are large enough that it causes rounding error. When examples were not available for certain functions, alternatives were used.

Chapter 6

Discussion

6.1 Overview

The objective of this project is to build a tool in R to compute survey statistics for large data sets which cannot be fit into memory and investigate whether it is feasible to do so.

From the results in chapter 4 we can see that overall, computing survey statistics in a database is definitely feasible for certain types of statistics, that is statistics that does not require many iterations. It is also possible that other statistics that does not heavily depend on statistical or mathematical operations can be computed in a database efficiently.

Additionally, we have found that **dplyr** and **dbplyr** seems to be implemented in a very efficient manner.

6.2 Future work

6.2.1 Error Messages

More user friendly error messages should be added. Currently while computing statistics with **svydb**, when there is an error in the code, it will only show where the error is without any explanations.

6.2.2 Functions

In general, more functions should be built into **svydb**.

Appendix A

Result Tables

All units are in seconds.

A.1 Total

A.1.1 Numeric variable

	Observations	Survey	svydb.local	svydb.database
1	250000	0.43	0.06	0.40
2	500000	0.88	0.08	0.41
3	750000	1.38	0.11	0.61
4	1000000	1.76	0.15	0.72
5	1250000	2.35	0.17	0.81
6	1500000	2.80	0.21	0.94
7	1750000	3.26	0.28	0.97
8	2000000	3.95	0.26	1.15
9	2500000	-	-	1.17
10	3000000	-	-	1.56
11	3500000	-	-	1.61
12	4000000	-	-	1.92

A.1.2 Categorical variable with 7 levels

	Observations	Survey	svydb.local	svydb.database
1	250000	0.73	0.22	1.20
2	500000	1.25	0.28	1.50
3	750000	1.87	0.34	1.76
4	1000000	2.58	0.46	2.03
5	1250000	3.10	0.51	2.28
6	1500000	3.94	0.61	2.65
7	1750000	4.42	0.71	2.92
8	2000000	5.27	0.79	3.20
9	2500000	-	-	3.63
10	3000000	-	-	4.25
11	3500000	-	-	4.67
12	4000000	-	-	5.21

A.2 Mean

A.2.1 Numeric variable

	Observations	Survey	svydb.local	svydb.database
1	250000	0.43	0.09	0.63
2	500000	0.90	0.11	0.79
3	750000	1.35	0.14	0.96
4	1000000	1.86	0.21	1.24
5	1250000	2.13	0.23	1.31
6	1500000	2.77	0.27	1.48
7	1750000	3.26	0.30	1.62
8	2000000	3.60	0.34	1.85
9	2500000	-	-	2.09
10	3000000	-	-	2.39
11	3500000	-	-	2.66
12	4000000	-	-	3.09

A.2.2 Categorical variable with 7 levels

	Observations	Survey	svydb.local	svydb.database
1	250000	0.72	0.21	1.65
2	500000	1.25	0.35	2.49
3	750000	1.90	0.47	2.73
4	1000000	2.49	0.70	3.26
5	1250000	3.26	0.76	3.83
6	1500000	4.02	0.89	4.50
7	1750000	4.85	1.02	5.04
8	2000000	5.55	0.94	5.63
9	2500000	-	-	6.68
10	3000000	-	-	7.53
11	3500000	-	-	8.70
12	4000000	-	-	9.72

A.3 Regression

	Observations	Survey	svydb.local	svydb.database
1	250000	1.18	0.68	2.95
2	500000	2.00	1.03	4.24
3	750000	3.16	1.30	5.31
4	1000000	4.50	1.31	6.56
5	1250000	5.65	1.65	7.75
6	1500000	6.77	1.97	8.70
7	1750000	8.13	2.24	10.09
8	2000000	9.75	2.68	10.72
9	2500000	-	-	12.31
10	3000000	-	-	14.31
11	3500000	-	-	16.32
12	4000000	-	-	18.22

A.4 Quantiles

	Observations	Survey	svydb.local	svydb.database
1	250000	0.58	0.08	0.36
2	500000	1.16	0.18	0.51
3	750000	1.55	0.26	0.63
4	1000000	2.18	0.29	0.83
5	1250000	2.80	0.28	0.93
6	1500000	3.45	0.31	1.04
7	1750000	4.24	0.27	1.17
8	2000000	4.82	0.32	1.40
9	2500000	-	-	1.49
10	3000000	-	-	1.73
11	3500000	-	-	1.98
12	4000000	-	-	2.25

A.5 Survey Tables

	Observations	Survey	svydb.local	svydb.database
1	250000	0.18	0.26	0.67
2	500000	0.39	0.49	1.02
3	750000	0.56	0.69	1.35
4	1000000	0.78	0.76	1.69
5	1250000	0.97	1.00	2.05
6	1500000	1.16	1.11	2.44
7	1750000	1.38	1.37	2.56
8	2000000	1.56	1.47	2.97
9	2500000	-	-	3.45
10	3000000	-	-	4.05
11	3500000	-	-	4.63
12	4000000	-	-	5.20

A.6 Total - Replicate Weights

	Observations	Survey	svydb.local	svydb.database
1	250000	3.54	0.46	2.12
2	500000	9.03	0.74	3.49
3	750000	12.31	1.13	4.81
4	1000000	21.57	1.45	5.86
5	1250000	-	-	7.10
6	1500000	-	-	9.34
7	1750000	-	-	10.62
8	2000000	-	-	14.75

A.7 Mean - Replicate Weights

	Observations	Survey	svydb.local	svydb.database
1	250000	3.57	0.66	3.66
2	500000	8.69	0.98	5.68
3	750000	14.25	1.61	7.23
4	1000000	20.27	1.89	8.72
5	1250000	-	-	10.36
6	1500000	-	-	12.24
7	1750000	-	-	13.97
8	2000000	-	-	16.02

A.8 Histogram

	Observations	Survey	svydb.local	svydb.database
1	250000	0.97	0.58	1.57
2	500000	2.07	0.75	2.56
3	750000	3.14	0.90	2.84
4	1000000	4.03	1.05	3.49
5	1250000	5.15	1.15	4.12
6	1500000	6.12	1.42	4.79
7	1750000	7.50	1.48	5.24
8	2000000	8.57	1.66	6.05
9	2500000	-	-	7.03
10	3000000	-	-	7.94
11	3500000	-	-	9.15
12	4000000	-	-	10.34

A.9 Boxplot

	Observations	Survey	svydb.local	svydb.database
1	250000	2.84	1.21	4.37
2	500000	4.96	1.39	5.20
3	750000	7.44	1.54	6.16
4	1000000	10.36	1.69	7.01
5	1250000	12.83	1.90	7.88
6	1500000	14.96	2.01	8.92
7	1750000	18.22	2.14	9.61
8	2000000	22.12	2.28	10.31
9	2500000	-	-	11.38
10	3000000	-	-	13.01
11	3500000	-	-	14.56
12	4000000	-	-	16.20

A.10 Hexagon Binning

	Observations	Survey	svydb.local	svydb.database
1	250000	0.41	0.92	2.05
2	500000	0.56	1.23	3.23
3	750000	0.99	1.53	4.27
4	1000000	1.00	1.97	5.53
5	1250000	1.50	2.17	6.40
6	1500000	1.54	2.47	7.33
7	1750000	1.97	2.61	8.24
8	2000000	2.32	2.92	9.29
9	2500000	-	-	10.65
10	3000000	-	-	12.59
11	3500000	-	-	14.41
12	4000000	-	-	16.40

Appendix B

Codes

B.1 svydbdesign

```
makesvydbdesign <- R6Class("svydb.design",
                            public = list(dataOg = NULL,
                            data = NULL, dataSub = NULL,
                            vars = NULL, st = NULL,
                            id = NULL, wt = NULL,
                            call = NULL, names = list(),
                            levels = list(),
    initialize = function(vars = NA,
        st = NA, id = NA, wt = NA, data) {
        if (quo_is_null(wt)) {
            stop("Please provide sampling weights")
        } else {
            self$wt = as.character(wt)[2]
        }

        if (quo_is_null(st)) {
            data = data %>% mutate(st = 1)
            self$st = "st"
        } else {
            self$st = as.character(st)[2]
        }

        if (quo_is_null(id)) {
            data = data %>% mutate(id = row_number())
            self$id = "id"
        } else {
            self$id = as.character(id)[2]
        }
        self$data = data %>% select(everything())
        self$dataOg <<- self$data
    },
    setx = function(x) {
        tc = tryCatch(class(x), error = function(e) e)

        if ("formula" %in% tc) {
            x = all.vars(x)
            self$data <<- self$data %>%
                select(!!x, self$st, self$id, self$wt) %>%
                filter_all(any_vars(!is.na(.)))
            self$vars <<- x
        } else {
            x = enquo(x)
            self$data <<- self$data %>%
                select(!!x, self$st, self$id, self$wt) %>%
                filter(!is.na(!!x))
            self$vars <<- as.character(x)[2]
        }
        self$names[["logged"]] =
            c(self$st, self$id, self$wt, "m_h")
    },
    addx = function(x) {
        l = enquo(x)
        r = syms(colnames(self$data))
        self$data = self$dataOg %>% select(!!l, !!!r)
    },
    getwt = function() {
        self$data %>% select(self$wt) %>%
            summarise_all(sum) %>% pull()
    },
    getmh = function() {
        self$data %>% group_by(!!sym(self$st)) %>%
            summarise(m_h = n_distinct(!!sym(self$id)))
    },
    subset = function(..., logical = T) {
        d = self$clone()

        if (logical == T) {
            d$data = d$data %>% filter(...)
        } else {
            d$data = d$data %>% filter(!!parse_expr(...))
        }
        return(d)
    },
    subset_rows = function(from, to) {
        self$dataSub = self$data %>%
            db_selectRows(., from = from, to = to)
    },
    storename = function(name, obj, force = FALSE) {
        if (force == TRUE) {
            self$names$logged =
                self$names$logged[-which(
                        self$names$logged %in% obj)]
        }
        if (!all(obj %in% self$names$logged)) {
            new = setdiff(obj, self$names$logged)
            self$names[[name]] = c(new)
            self$names$logged = c(self$names$logged, new)
        }
    },
    removename = function(name, obj) {
        self$names$logged =
            self$names$logged[-which(
                self$names$logged %in% obj)]
        self$names[[name]] =
            (self$names[[name]])[-which(
                self$names[[name]] %in% obj)]
    },
    storelevel = function(x, name) {
        ll = list(x)
        names(ll) = name
        self$levels = c(self$levels, ll)
    },
    storecall = function(x) {
        self$call = x
    },
    print = function() {
        nid = self$getmh() %>% pull(m_h) %>% sum()
        rows = self$data %>% db_nrow()
        txt = sprintf("svydb.design,
            %s observation(s), %s Clusters\n",
            rows, nid)
        cat(txt)
    }))

svydbdesign = function(st = NULL, id = NULL,
                            wt = NULL, data){
  st = enquo(st)
  id = enquo(id)
  wt = enquo(wt)

  d = makesvydbdesign$new(st = st, id = id, wt = wt,
                          data = data)
  d$storecall(match.call())
  d
}
```

B.2 svydbtotal

```
svydbtotal = function(x, num, design,
                      return.total = F, ...) {

    if (!("svydb.design" %in% class(design))) {
        stop("Please provide a svydb.design")
    }

    if (missing(num)) {
        stop("Is x a numeric or categorical variable?
            , num = T OR num = F?")
    }

    dsn = design$clone()
    dsn$setx(!!enquo(x))
    d = dsn$data
    dsn$storename("x", colnames(d))

    if (num == F) {
        d = dummy_mut(d, !!sym(dsn$names$x), withBase = T)
    }
    dsn$storename("x", colnames(d))
    d = d %>% mutate_at(vars(dsn$names$x),
                        funs((. * !!sym(dsn$wt)))) %>%
            compute(temporary = T)

    totTbl = d %>% select(dsn$names$x) %>%
                summarise_all(sum) %>% collect() %>% t()

    if (return.total == TRUE) {
        colnames(totTbl) = "Total"
        return(totTbl)
    }

    varTbl = d %>% select(dsn$st, dsn$id, dsn$names$x) %>%
                    group_by(!!!syms(c(dsn$st, dsn$id))) %>%
                    summarise_at(vars(dsn$names$x),
                                funs(sum(.))) %>%
                    compute(temporary = T)
    varTbl = inner_join(varTbl, dsn$getmh(), by = dsn$st)

    barTbl = varTbl %>% select(-one_of(dsn$id)) %>%
                        group_by(!!sym(dsn$st)) %>%
                        summarise_at(vars(dsn$names$x),
                                    funs(bar = sum(./m_h)))
    dsn$storename("bar", colnames(barTbl))

    varTbl = inner_join(varTbl, barTbl, by = dsn$st)
    ‘zhi-zbar‘ = paste(dsn$names$x, "-",
                       dsn$names$bar, collapse = " ; ")
    varTbl = varTbl %>%
        mutate(!!!parse_exprs(‘zhi-zbar‘)) %>%
        ungroup() %>% compute(temporary = T)
    dsn$storename("diff", colnames(varTbl))

    varTbl = sapply(dsn$names$diff, svydbVar,
                    st = dsn$st, m_h = "m_h", data = varTbl)

    class(totTbl) = "svydbstat"
    attr(totTbl, "var") = varTbl
    attr(totTbl, "statistic") <- "Total"
    attr(totTbl, "name") = dsn$names$x
    return(totTbl)
}
```

B.3 svydbmean

```
svydbmean = function(x, num, design,
                    return.mean = F, ...) {

    if (!("svydb.design" %in% class(design))) {
        stop("Please provide a svydb.design")
    }

    if (missing(num)) {
        stop("Is x a numeric or categorical variable?,
             = T OR num = F?")
    }

    dsn = design$clone()
    dsn$setx(!!enquo(x))
    d = dsn$data
    dsn$storename("x", colnames(d))

    if (num == F) {
        d = dummy_mut(d, !!sym(dsn$names$x), withBase = T)
    }

    dsn$storename("x", colnames(d))
    N = dsn$getwt()
    meanTbl = d %>%
              transmute_at(vars(dsn$names$x),
                           funs((. * !!sym(dsn$wt))
                                 /!!quo(N))) %>%
              summarise_all(sum) %>%
              compute(temporary = T) %>% collect()

    if (return.mean == TRUE) {
        colnames(meanTbl) = dsn$names$x
        return(meanTbl)
    }

    dhi_exprs = paste(dsn$names$x, " - ", "meanTbl$",
                      dsn$names$x, sep = "",
                      collapse = " ; ")
    varTbl = d %>% mutate(dhi = !!!parse_exprs(dhi_exprs))
    dsn$storename("dhi", colnames(varTbl))

    varTbl = varTbl %>%
               mutate_at(vars(dsn$names$dhi),
               funs((. * !!sym(dsn$wt))/!!quo(N))) %>%
               select(dsn$st, dsn$id,
                dsn$names$dhi)

    barTbl = varTbl %>% select(dsn$st, dsn$names$dhi) %>%
             group_by(!!sym(dsn$st)) %>% summarise_all(sum)
    barTbl = inner_join(barTbl, dsn$getmh(),
        by = dsn$st) %>%
        mutate_at(vars(dsn$names$dhi),
        funs(bar = ./m_h)) %>%
        select(-one_of(dsn$names$dhi))
    dsn$storename("bar", colnames(barTbl))

    varTbl = varTbl %>%
             group_by(!!!syms(c(dsn$st, dsn$id))) %>%
             summarise_all(sum) %>% compute(temporary = T)
    varTbl = inner_join(varTbl, barTbl, by = dsn$st)

    `dhi-dbar` = paste("`", dsn$names$dhi, "`", "-",
                       "`", dsn$names$bar, "`",
                       collapse = " ; ", sep = "")
    varTbl = varTbl %>%
             mutate(!!!parse_exprs(`dhi-dbar`)) %>%
             ungroup() %>% compute(temporary = T)
    dsn$storename("diff", colnames(varTbl))

    varTbl = sapply(dsn$names$diff, svydbVar,
                    st = dsn$st, m_h = "m_h", data = varTbl)

    meanTbl = t(meanTbl)
    class(meanTbl) = "svydbstat"
    attr(meanTbl, "var") = varTbl
    attr(meanTbl, "statistic") <- "Mean"
    attr(meanTbl, "name") = dsn$names$x

    return(meanTbl)
}
```

B.4 svydblm

```
svydblm = function(formula, design) {
    if (!("svydb.design" %in% class(design))) {
        stop("Please provide a svydb.design")
    }

    if (missing(formula)) {
        stop("Please provide a formula")
    }

    fit = list()
    fit$call = match.call()
    dsn = design$clone()
    dsn$setx(formula)
    d = dsn$data
    dsn$storename("y", all.vars(formula)[1])
    dsn$storename("variables", all.vars(formula)[-1])

    d = d %>% mutate(`:=`(!!sym(dsn$wt),
        case_when(is.na(!!sym(dsn$names$y)) ~
        0, TRUE ~ !!sym(dsn$wt))))
    d = d %>% mutate(`:=`(!!sym(dsn$names$y),
        case_when(is.na(!!sym(dsn$names$y)) ~
            0, TRUE ~ !!sym(dsn$names$y))))
    d = d %>% filter_all(all_vars(!is.na(.)))
    d = d %>% mutate(intercept = 1)
    dsn$storename("intercept", colnames(d))
    xy = d

    facVar = attr(terms(formula, specials = "factor"),
        "specials")$factor
    if (!is.null(facVar)) {
        facVar = facVar - 1
        dsn$storename("factor", dsn$names$variables[facVar],
            force = T)
        for (i in 1:length(facVar)) {
            dd = dummy_mut(xy, by =
                !!sym(dsn$names$factor[i]),
                withBase = F, return.level = T)
            xy = dd$dum
            dsn$storelevel(x = dd$levels,
                name = dsn$names$factor[i])
            dsn$names$variables = c(dsn$names$variables,
                dsn$names$factor[i])
            dsn$names$variables =
                dsn$names$variables[-(grep(
                dsn$names$factor[i],
                dsn$names$variables)[1])]
        }
    }

    fit$terms = terms(paste("~", paste(dsn$names$variables,
        collapse = " + ")) %>% as.formula)
    dsn$storename("dummy", colnames(xy))
    xy = compute(xy)
    dsn$storename("variables", c(dsn$names$variables,
        dsn$names$dummy), force = T)

    if (!is.null(facVar)) {
        dsn$removename("variables", dsn$names$factor)
    }
    db_xtwx_i = function(x, col, wt, data) {
        data = data %>% summarise_at(vars(x),
            funs(xtx = sum(. * (!!sym(col)) *
                (!!sym(wt)))))
        return(data)
    }
    xtx = lapply(c(dsn$names$intercept,
        dsn$names$variables), db_xtwx_i,
        x = c(dsn$names$intercept,
        dsn$names$variables), wt = dsn$wt,
        data = xy)
    xtx = Reduce(full_join, xtx) %>% collect() %>%
        as.matrix()
    colnames(xtx) =
        c(dsn$names$intercept, dsn$names$variables)

    xty = xy %>% transmute_at(vars(dsn$names$intercept,
        dsn$names$variables), funs(. * (!!sym(dsn$wt)) *
        (!!sym(dsn$names$y)))) %>% summarise_all(sum) %>%
        collect() %>% t()

    xy = xy %>% filter(!(!!sym(dsn$wt)) == 0) %>%
        compute()
    beta = solve(xtx) %*% xty
    fit$coefname =
        c(dsn$names$intercept, dsn$names$variables)
    fit$coefficients = beta %>% t()


    dsn$storename("xb", paste(dsn$names$variables,
        "_xb", sep = ""))
    e = paste(dsn$names$y, " - ", beta["intercept",
        ], " - ", paste(rownames(beta)[-1], " * ",
        beta[-1, ], sep = "", collapse = " - "),
        sep = "")

    xy = xy %>% mutate(residuals = !!parse_expr(e)) %>%
        select(-one_of(dsn$names$y))
    dsn$storename("residuals", colnames(xy))
    res = xy %>% select(residuals)
    fit$residuals = res

    varTbl = xy %>% mutate_at(vars(dsn$names$intercept,
        dsn$names$variables),
        funs(. * (!!sym(dsn$names$residuals)) *
        (!!sym(dsn$wt)))) %>% group_by(!!sym(dsn$st),
        !!sym(dsn$id)) %>% summarise_all(sum) %>%
        compute()
    dsn$storename("zhi", c(dsn$names$intercept,
        dsn$names$variables), force = T)
    barTbl = varTbl %>% select(dsn$st, dsn$names$zhi) %>%
        group_by(!!sym(dsn$st)) %>% summarise_all(sum)

    mh = dsn$getmh()
    barTbl = inner_join(barTbl, mh, by = dsn$st) %>%
        mutate_at(vars(dsn$names$intercept,
        dsn$names$variables), funs(bar = ./m_h)) %>%
        compute()
    dsn$storename("bar", colnames(barTbl))

    varTbl = inner_join(varTbl %>% select(dsn$st,
        dsn$names$zhi), barTbl %>% select(dsn$st,
        dsn$names$bar, m_h), by = dsn$st)
    ‘zhi-zbar‘ = paste("‘", dsn$names$zhi, "‘",
        "-", "‘", dsn$names$bar, "‘", collapse = " ; ",
        sep = "")
    varTbl = varTbl %>%
        mutate(!!!parse_exprs(‘zhi-zbar‘)) %>%
        ungroup() %>% compute()
    dsn$storename("diff", colnames(varTbl))

    covDiag = sapply(dsn$names$diff, svydbVar,
        st = dsn$st, m_h = "m_h", data = varTbl)
    e = outer(dsn$names$diff, dsn$names$diff,
        paste, sep = ",")
    e = e[lower.tri(e)] %>% strsplit(., ",")
    covLt = sapply(e, svydbVar2, st = dsn$st,
        m_h = "m_h", data = varTbl)
    cov.mat = diag(covDiag)
    cov.mat[lower.tri(cov.mat)] = covLt
    cov.mat[upper.tri(cov.mat)] =
        t(cov.mat)[upper.tri(cov.mat)]

    fit$cov.unscaled = solve(xtx) %*% cov.mat %*%
        solve(xtx)
    colnames(fit$cov.unscaled) = rownames(fit$cov.unscaled)

    fit$formula = formula
    fit$df.residual = sum(mh %>% select(m_h) %>%
        pull) - db_nrow(mh) - nrow(beta) + 1
    fit$call = match.call()
    fit$design = dsn

    class(fit) = c("svydblm", "lm")
    return(fit)
}
```

B.5 svydbquantile

```
svydbquantile = function(x, quantiles = 0.5, design) {
    oldoptions = options("survey.lonely.psu")
    options(survey.lonely.psu = "adjust")

    dsn = design$clone()
    dsn$setx(!!enquo(x))
    d = dsn$data
    dsn$storename("x", colnames(d))

    q_name = quantiles
    qvec = c()
    b = T

    for (i in 1:length(quantiles)) {
        if (quantiles[i] >= 1) {
            qvec[i] =
                dbmax(data = d, var = !!sym(dsn$names$x))
        } else if (quantiles[i] <= 0) {
            qvec[i] =
                dbmin(data = d, var = !!sym(dsn$names$x))
        } else {
            if (b) {
                N = dsn$getwt()
                nrowdata = db_nrow(d)
                sampN = ceiling(nrowdata^(2/3))
                d = compute(d)

                if (!is.null(d$src$con)) {
                  sdata = d %>%
                    svydb_monet_sampleN(sampN) %>%
                    tbl_df()
                } else {
                  sdata = d %>% sample_n(sampN) %>%
                    tbl_df()
                }

                sdata =
                    sdata %>% rename(x = !!sym(dsn$names$x))
                s.surv = svydesign(id = c2f(dsn$id),
                  st = c2f(dsn$st), weights = c2f(dsn$wt),
                  data = sdata, nest = T)
                b = F
            }
            notfound = TRUE

            while (notfound) {
                q = svyquantile(~x, s.surv, quantiles[i],
                  alpha = 0.1, ci = TRUE, na.rm = T)

                temp_lq = q$CIs[1]
                temp_uq = q$CIs[2]

                readIn = d %>% select(x = dsn$names$x,
                  wt = dsn$wt) %>% filter(x >= temp_lq &
                  x <= temp_uq)
                readIn_wts = readIn %>% select(wt) %>%
                  summarise(sum(wt)) %>% pull()
                notRead = d %>% select(x = dsn$names$x,
                  wt = dsn$wt) %>% filter(x < temp_lq)
                notRead_wts = notRead %>% select(wt) %>%
                  summarise(sum(wt)) %>% pull()

                thold = (N * quantiles[i])
                if ((notRead_wts <= thold) & (thold <
                  (readIn_wts + notRead_wts))) {
                  qs = readIn %>% arrange(x)
                  qs = qs %>% collect() %>%
                    mutate(wt2 = cumsum(wt))
                  c_wts = N * quantiles[i] - notRead_wts
                  qs = qs %>%
                    filter(wt2 >= !!quo(c_wts)) %>%
                    select(x) %>% slice(1) %>% pull()
                  qvec[i] = qs
                  notfound = F
                } else {
                  if (!is.null(d$src$con)) {
                    sdata = d %>%
                        svydb_monet_sampleN(sampN) %>%
                        tbl_df() %>%
                        rename(x = !!sym(dsn$names$x))
                  } else {
                    sdata = d %>% sample_n(sampN) %>%
                      tbl_df() %>%
                      rename(x = !!sym(dsn$names$x))
                  }
                  s.surv = svydesign(id = c2f(dsn$id),
                    st = c2f(dsn$st), weights = c2f(dsn$wt),
                    data = sdata, nest = T)
                }
            }
        }
    }
    names(qvec) = as.character(q_name)
    options(oldoptions)
    return(qvec)
  }
```

B.6 svydbtable

```
svydbtable = function(formula, design, as.local = F) {

  dsn = design$clone()
  dsn$setx(formula)
  d = dsn$data
  d = d %>% filter_all(all_vars(!is.na(.)))
  dsn$storename("all", colnames(d))

  ff = all.vars(formula)
  dsn$storename("base", ff[1], force = T)
```

```
if (length(ff) == 1) {
    out = d %>% group_by(!!sym(dsn$names$base)) %>%
        summarise(wt = sum(!!sym(dsn$wt))) %>%
        arrange(!!sym(dsn$names$base))
    if (as.local == T) {
        out = collect(out)
    }
    return(out)
}

dsn$storename("by", ff[2], force = T)
d = d %>% select(dsn$names$all, dsn$wt) %>%
    dummy_mut(data = ., by = !!sym(dsn$names$by),
        withBase = T)
dsn$storename("dummy", colnames(d))
d = d %>% select(-one_of(dsn$names$by))
d = compute(d)

if (length(ff) == 2) {
    out = d %>% group_by(!!sym(dsn$names$base)) %>%
        summarise_at(vars(dsn$names$dummy),
            funs(sum(. * (!!sym(dsn$wt))))) %>%
        arrange(!!sym(dsn$names$base))
    if (as.local == T) {
        out = collect(out)
    }
    return(out)
}

dsn$storename("others", ff[-c(1, 2)], force = T)
d = db_columnAsCharacter(d, dsn$names$others)
combnTbl = d %>% select(dsn$names$others) %>%
    distinct() %>%
    arrange(!!!syms(dsn$names$others)) %>%
    collect()
combnLst = split(combnTbl, seq(1, nrow(combnTbl)))

sTbls = function(by) {
    nname = paste(colnames(by), " = ", by, sep = "",
        collapse = " & ")
    con = gsub(pattern = "=", replacement = "==",
        nname)
    d = d %>% filter(!!parse_expr(con)) %>%
        select(-one_of(colnames(by))) %>%
        group_by(!!sym(dsn$names$base)) %>%
        summarise_at(vars(dsn$names$dummy),
            funs(sum(. * (!!sym(dsn$wt))))) %>%
        arrange(!!sym(dsn$names$base)) %>%
        list(.)
    names(d) = nname
```

```
        d
    }
    out = lapply(combnLst, sTbls) %>% flatten()

    if (as.local == T) {
        out = lapply(out, collect)
    }

    return(out)
}
```

B.7 svydbby

```
svydbtable = function(formula, design, as.local = F) {

    dsn = design$clone()
    dsn$setx(formula)
    d = dsn$data
    d = d %>% filter_all(all_vars(!is.na(.)))
    dsn$storename("all", colnames(d))

    ff = all.vars(formula)
    dsn$storename("base", ff[1], force = T)

    if (length(ff) == 1) {
        out = d %>% group_by(!!sym(dsn$names$base)) %>%
            summarise(wt = sum(!!sym(dsn$wt))) %>%
            arrange(!!sym(dsn$names$base))
        if (as.local == T) {
            out = collect(out)
        }
        return(out)
    }

    dsn$storename("by", ff[2], force = T)
    d = d %>% select(dsn$names$all, dsn$wt) %>%
        dummy_mut(data = ., by = !!sym(dsn$names$by),
            withBase = T)
    dsn$storename("dummy", colnames(d))
    d = d %>% select(-one_of(dsn$names$by))
    d = compute(d)

    if (length(ff) == 2) {
        out = d %>% group_by(!!sym(dsn$names$base)) %>%
            summarise_at(vars(dsn$names$dummy),
                funs(sum(. * (!!sym(dsn$wt))))) %>%
            arrange(!!sym(dsn$names$base))
        if (as.local == T) {
            out = collect(out)
        }
        return(out)
    }

    dsn$storename("others", ff[-c(1, 2)], force = T)
    combnTbl = d %>% select(dsn$names$others) %>%
        distinct() %>%
        arrange(!!!syms(dsn$names$others)) %>%
        collect()
    combnLst = split(combnTbl, seq(1, nrow(combnTbl)))

    sTbls = function(by) {
        nname = paste(colnames(by), " = ", by,
            collapse = " & ")
        con = gsub(pattern = "=", replacement = "==",
            nname)
        d = d %>% filter(!!parse_expr(con)) %>%
            select(-one_of(colnames(by))) %>%
            group_by(!!sym(dsn$names$base)) %>%
            summarise_at(vars(dsn$names$dummy),
                funs(sum(. * (!!sym(dsn$wt))))) %>%
            arrange(!!sym(dsn$names$base)) %>%
            list(.)
        names(d) = nname
        d
    }
    out = lapply(combnLst, sTbls) %>% flatten()

    if (as.local == T) {
        out = lapply(out, collect)
    }

    return(out)
}
```

B.8 svydbrepdesign

```
makesvydbrepdesign <- R6Class("svydb.repdesign",
    public = list(dataOg = NULL, data = NULL,
        vars = NULL, st = NULL, id = NULL, wt = NULL,
        repwt = NULL, scale = NULL, names = list(),
        initialize = function(vars = NA, st = NA,
            id = NA, wt = NA, repwt = NULL, scale,
            data) {

            if (quo_is_null(wt)) {
                stop("Please provide sampling weights")
            } else {
                self$wt = as.character(wt)[2]
            }

            if (is.null(repwt)) {
                stop("Please provide replicate weights")
            } else {
                self$repwt = grep(pattern = repwt,
                  colnames(data), value = T)
            }

            if (quo_is_null(st)) {
                data = data %>% mutate(st = 1)
                self$st = "st"
            } else {
                self$st = as.character(st)[2]
            }

            if (quo_is_null(id)) {
                data = data %>% mutate(id = row_number())
                self$id = "id"
            } else {
                self$id = as.character(id)[2]
            }
            self$scale = scale
            self$data = data %>% select(everything())
            self$dataOg <<- self$data
        }, setx = function(x) {
            tc = tryCatch(class(x), error = function(e) e)

            if ("formula" %in% tc) {
                x = all.vars(x)
                self$data <<- self$data %>% select(!!x,
                  st = self$st, id = self$id,
                  self$wt, self$repwt) %>%
                  filter_all(any_vars(!is.na(.)))
                self$vars <<- x
            } else {
                x = enquo(x)
                self$data <<- self$data %>% select(!!x,
                  st = self$st, id = self$id,
                  self$wt, self$repwt) %>%
                  filter(!is.na(!!x))
                self$vars <<- as.character(x)[2]
            }
            self$names[["logged"]] = c(self$st,
                self$id, self$wt, self$repwt,
                "m_h")
        }, addx = function(x) {
            l = enquo(x)
            r = syms(colnames(self$data))
            self$data = self$dataOg %>% select(!!l,
                !!!r)
        }, getwt = function() {
            self$data %>% select(self$wt) %>%
                summarise_all(sum) %>% pull()
        }, getmh = function() {
            self$data %>% group_by(!!sym(self$st)) %>%
                summarise(m_h = n_distinct(!!sym(self$id)))
        }, subset = function(..., logical = T) {
            d = self$clone()
            if (logical == T) {
                d$data = d$data %>% filter(...)
            } else {
                d$data = d$data %>%
                    filter(!!parse_expr(...))
            }
            return(d)
        }, subset_rows = function(from, to) {
            self$dataSub = self$data %>% db_selectRows(.,
                from = from, to = to)
        }, storename = function(name, obj, force = FALSE) {
            if (force == TRUE) {
                self$names$logged =
                    self$names$logged[-which(
                        self$names$logged %in% obj)]
            }
            if (!all(obj %in% self$names$logged)) {
                new = setdiff(obj, self$names$logged)
                self$names[[name]] = c(new)
                self$names$logged = c(self$names$logged,
                  new)
            }
        }, removename = function(name, obj) {
            self$names$logged =
                self$names$logged[-which(
                    self$names$logged %in% obj)]
            self$names[[name]] =
                (self$names[[name]])[-which(
                        self$names[[name]] %in% obj)]
        }, print = function() {
            rows = self$data %>% db_nrow()
            txt = sprintf("svydb.repdesign,
                %s observation(s), %s sets of
                replicate weights, scale = %s",
                rows, length(self$repwt), self$scale)
            cat(txt)
}))

svydbrepdesign = function(st = NULL, id = NULL,
    wt = NULL, repwt = NULL, scale, data) {
    st = enquo(st)
    id = enquo(id)
    wt = enquo(wt)

    d = makesvydbrepdesign$new(st = st, id = id,
        wt = wt, repwt = repwt, scale = scale,
        data = data)
    d
}
```

B.9 svydbreptotal

```
svydbreptotal = function(x, design,
            num, return.replicates = F) {
    x = enquo(x)

    if (!("svydb.repdesign" %in% class(design))) {
        stop("Please provide a svydb.repdesign")
    }

    if (missing(num)) {
        stop("Is x a numeric or categorical variable?,
            num = T OR num = F?")
    }

    dsn = design$clone()
    dsn$setx(!!enquo(x))

    d = dsn$data

    dsn$storename("x", colnames(d))

    if (num == F) {
        d = dummy_mut(d, !!sym(dsn$names$x), withBase = T)
    }

    dsn$storename("x", colnames(d))

    fullTotTbl = d %>% summarise_at(vars(dsn$names$x),
        funs(sum(. * (!!sym(dsn$wt))))) %>% collect()

    cnt = 1
    getRepTots = function(names, fullTot) {
        replicates = d %>% summarise_at(vars(dsn$repwt),
            funs(sum((. * !!sym(names))))) %>%
            collect()
        repTot = replicates %>% summarise_all(funs((. -
            !!quo(fullTot[cnt]))^2))
        cnt <<- cnt + 1
        if (return.replicates == T) {
            list(replicates = replicates,
                repVar = db_rowSums(repTot) %>%
                transmute_all(
                    funs(. * !!quo(dsn$scale))) %>%
                collect())
        } else {
            list(repVar = db_rowSums(repTot) %>%
                transmute_all(funs(. *
                    !!quo(dsn$scale))) %>%
                collect())
        }
    }

    ans = lapply(colnames(fullTotTbl), getRepTots,
        fullTot = as.vector(t(fullTotTbl)))
    repVar = lapply(ans, function(x) x$repVar) %>%
        Reduce(rbind, .) %>% pull()

    tot = fullTotTbl %>% t() %>% as.vector()
    attr(tot, "var") = repVar
    attr(tot, "statistic") <- "Total"
    attr(tot, "name") = dsn$names$x

    if (return.replicates == T) {
        replicates =
            lapply(ans, function(x) x$replicates %>%
            collect()) %>% Reduce(rbind, .)
        tot = list(svydbrepstat = tot,
            replicates = replicates)
    }
    class(tot) = c("svydbrepstat")
    return(tot)
}
```

B.10 svydbrepmean

```
svydbrepmean = function(x, design, num,
                    return.replicates = F) {
    x = enquo(x)

    if (!("svydb.repdesign" %in% class(design))) {
        stop("Please provide a svydb.repdesign")
    }

    if (missing(num)) {
        stop("Is x a numeric or categorical variable?,
            num = T OR num = F?")
    }

    dsn = design$clone()
    dsn$setx(!!enquo(x))

    d = dsn$data

    dsn$storename("x", colnames(d))

    if (num == F) {
        d = dummy_mut(d, !!sym(dsn$names$x), withBase = T)
    }
    dsn$storename("x", colnames(d))

    N = dsn$getwt()
    fullMeanTbl = d %>% summarise_at(vars(dsn$names$x),
        funs(sum(. * (!!sym(dsn$wt)))/!!quo(N))) %>%
        collect()

    repN = d %>% select(dsn$repwt) %>%
        summarise_all(sum) %>%
        collect()
    repN = paste(colnames(repN), "/", repN,
        collapse = " ; ")
    cnt = 1
    getRepTots = function(names, fullMean) {
        replicates = d %>% summarise_at(vars(dsn$repwt),
            funs(sum(. * !!sym(names)))) %>%
            transmute(!!!parse_exprs(repN)) %>%
            compute()
        repMean = replicates %>% summarise_all(funs((. -
            !!quo(fullMean[cnt]))^2))
        cnt <<- cnt + 1
        if (return.replicates == T) {
            list(replicates = replicates,
                repVar = db_rowSums(repMean) %>%
                transmute_all(funs(. *
                    !!quo(dsn$scale))) %>%
                collect())
        } else {
            list(repVar = db_rowSums(repMean) %>%
                transmute_all(funs(. *
                    !!quo(dsn$scale))) %>%
                collect())
        }
    }
    ans = lapply(colnames(fullMeanTbl), getRepTots,
        fullMean = as.vector(t(fullMeanTbl)))
    repVar = lapply(ans, function(x) x$repVar) %>%
        Reduce(rbind, .) %>% pull()

    means = fullMeanTbl %>% t() %>% as.vector()
    attr(means, "var") = repVar
    attr(means, "statistic") <- "Mean"
    attr(means, "name") = dsn$names$x

    if (return.replicates == T) {
        replicates = lapply(ans, function(x)
                x$replicates %>%
            collect()) %>% Reduce(rbind, .)
        means = list(svydbrepstat = means,
            replicates = replicates)
    }
    class(means) = c("svydbrepstat")
    return(means)
}
```

B.11 svydbhist

```
svydbhist = function(x, design, binwidth = NULL,
    xlab = "x", ylab = "Density", ...) {

    if (!("svydb.design" %in% class(design))) {
        stop("Please provide a svydb.design")
    }

    dsn = design$clone()
    dsn$setx(!!enquo(x))
    d = dsn$data
    dsn$storename("x", colnames(d))
    d_n = d %>% db_nrow()

    x_max = d %>% dbmax(!!sym(dsn$names$x), asNum = T)
    x_min = d %>% dbmin(!!sym(dsn$names$x), asNum = T)
    x_range = x_max - x_min

    if (is.null(binwidth)) {
        binwidth = ceiling(log2(d_n) + 1)
        pbreaks = pretty(c(x_min, x_max), n = binwidth,
            min.n = 1)
    } else {
        pbreaks = seq(from = floor(x_min),
            to = ceiling(x_max),
            by = binwidth)
    }

    d = db_cut2(var = !!sym(dsn$names$x), breaks = pbreaks,
        data = d) %>% arrange(cut)
    dsn$data = d

    props = svydbmean(x = cut, design = dsn, num = F,
        return.mean = T) %>% collect() %>% t()
    colnames(props) = "Mean"

    mids = pbreaks[-length(pbreaks)] + diff(pbreaks)/2

    if (length(mids) != nrow(props)) {
        props = tbl_df(props) %>%
            mutate(uqid = row_number())
        d = left_join(mids %>% tbl_df() %>%
            mutate(uqid = row_number()),
            props, by = "uqid")
        d = d %>% mutate(Mean = case_when(is.na(Mean) ~
            0, TRUE ~ Mean)) %>% select(-uqid)
    } else {
        d = cbind(mids, props) %>% tbl_df()
    }

    colnames(d) = c("x", "y")
    d$y = d$y/diff(pbreaks)
    p = ggplot(d) + geom_col(aes(x, y)) +
        labs(x = dsn$names$x, y = ylab)

    print(p)

    invisible(p)
}
```

B.12 svydbboxplot

```
svydbboxplot = function(x, groups = NULL, design,
    varwidth = F, outlier = F, all.outlier = F) {

    groups = enquo(groups)
    dsn = design$clone()
    dsn$setx(!!enquo(x))
    d = dsn$data
    dsn$storename("x", colnames(d))

    if (quo_is_null(groups)) {
        boxes = svydbquantile(x = !!sym(dsn$names$x),
            quantile = c(0, 0.25, 0.5, 0.75, 1),
            design = dsn) %>% t() %>% tbl_df() %>%
            mutate("")
        ax = c(x = "", y = dsn$names$x)
        colnames(boxes) = c(as.character(letters[1:5]),
            "x")
    } else {
        group_name = as.character(groups)[2]
        dsn$addx(group_name)
        d = dsn$data
        dsn$storename("groups", colnames(d))

        group_levels = distinct(d,
            !!sym(dsn$names$groups)) %>%
            collect()
        group_names = paste(colnames(group_levels),
            pull(group_levels), sep = " ")
        group_names2 = paste(colnames(group_levels),
            paste("'", pull(group_levels), "'",
                sep = ""), sep = " ")

        f = function(x) {
            svydbquantile(x = !!sym(dsn$names$x),
                quantile = c(0, 0.25, 0.5, 0.75,
                  1), design = dsn$subset(x, logical = F))
        }

        boxes = sapply(gsub(pattern = " ",
            replacement = "==",  x = ,
            group_names2), f)
        boxes = t(boxes) %>% tbl_df() %>%
            bind_cols(group_levels)
        ax = c(x = dsn$names$groups, y = dsn$names$x)
        colnames(boxes) = c(as.character(letters[1:5]),
            "x")
        boxes$x = as.character(boxes$x)
    }

    haveOut = F
    outlsLst = list()
    if (outlier == T) {
        boxes = boxes %>% mutate(outUP = d + 1.5 *
            (d - b), checkUP = ifelse(outUP <
            e, T, F), outLow = b - 1.5 * (d -
            b), checkLow = ifelse(outLow > a,
            T, F))
        if (any(boxes$checkLow) == T) {
            haveOut = T
            boxes = boxes %>% mutate(a = ifelse(checkLow ==
                T, outLow, a))
            outls = paste(gsub(pattern = " ",
                replacement = "==", x = group_names2),
                "&", dsn$names$x, "<", boxes$a,
                collapse = " | ")
            outlsLow = d %>%
                filter(!!!parse_exprs(outls)) %>%
                select(x = dsn$names$groups,
                    y = dsn$names$x) %>%
                mutate(x = as.character(x))
            if (all.outlier == F) {
                outlsLow = outlsLow %>% group_by(x) %>%
                  summarise(y = min(y))
            }
            outlsLow = outlsLow %>% tbl_df()
            outlsLst = c(outlsLst, list(outlsUP))
        }

        if (any(boxes$checkUP) == T) {
            haveOut = T
            boxes = boxes %>% mutate(e = ifelse(checkUP ==
                T, outUP, e))
            outls = paste(gsub(pattern = " ",
                replacement = "==", x = group_names2),
                "&", dsn$names$x, ">", boxes$e,
                collapse = " | ")
            outlsUP = d %>%
                filter(!!!parse_exprs(outls)) %>%
                select(x = dsn$names$groups,
                    y = dsn$names$x) %>%
                mutate(x = as.character(x))
            if (all.outlier == F) {
                outlsUP = outlsUP %>% group_by(x) %>%
                  summarise(y = max(y))
            }
            outlsUP = outlsUP %>% tbl_df()
        }
        outls = bind_rows(outlsUP, outlsLow)
        outls = bind_rows(outlsUP, outlsLow)
    }

    p = ggplot(boxes) + labs(x = ax["x"], y = ax["y"])

    if (varwidth == T) {
        boxwid = d %>%
            group_by(!!sym(dsn$names$groups)) %>%
            summarise(wid = n()) %>% collect()
        p$data = p$data %>%
            mutate(width = (boxwid$wid/sum(boxwid$wid)))
        p = p + geom_boxplot(aes(x = as.factor(x),
            ymin = a, lower = b, middle = c, upper = d,
            ymax = e, width = width), stat = "identity")

    } else {
        p = p + geom_boxplot(aes(x = as.factor(x),
            ymin = a, lower = b, middle = c, upper = d,
            ymax = e), stat = "identity")
    }

    if (haveOut == T) {
        utlsLst = Reduce(rbind, outlsLst)
        p = p + geom_point(data = outls, aes(x = x,
            y = y))
    }

    print(p)

    return(p)
}
```

B.13 svydbhexbin, svydbhexplot

```
svydbhcell2xy = function(d) {
    xbins = d$xbins
    xbnds = d$xbnds
    c3 = diff(xbnds)/xbins
    ybnds = d$ybnds
    c4 = (diff(ybnds) * sqrt(3))/(2 * d$shape *
        xbins)
    jmax = d$dimen[2]
    cell = d$cell - 1
    i = cell%/%jmax
    j = cell%%jmax
    y = c4 * i + ybnds[1]
    x = c3 * ifelse(i%%2 == 0, j, j + 0.5) + xbnds[1]

    return(list(x = x, y = y))
}


svydbhbin = function(xy, x, y, xName, yName, cell,
    cnt, xcm, ycm, size, shape, rx, ry, bnd, n) {
    xmin = rx[1]
    ymin = ry[1]
    xr = rx[2] - xmin
    yr = ry[2] - ymin
    c1 = size/xr
    c2 = size * shape/(yr * sqrt(3))

    jinc = floor(bnd[2])
    lat = floor(jinc + 1)
    iinc = floor(2 * jinc)
    lmax = floor(bnd[1] * as.integer(bnd[2]))
    con1 = 0.25
    con2 = 1/3

    xy = xy %>% mutate(sx = !!quo(c1), sy = !!quo(c2),
        xmin = !!quo(xmin), ymin = !!quo(ymin))
    xy = xy %>% mutate(sx = sx * (x - xmin))
    xy = xy %>% mutate(sy = sy * (y - ymin))
    xy = xy %>% mutate(j1 = floor(sx + 0.5),
        i1 = floor(sy + 0.5))
    xy = xy %>% mutate(dist1 = (sx - j1)^2 + 3 *
        (sy - i1)^2, iinc = !!quo(iinc), lat = !!quo(lat))
    xy = xy %>% mutate(con1 = !!quo(con1),
        con2 = !!quo(con2), j2 = floor(sx),
        i2 = floor(sy))
    xy = xy %>% mutate(con3 = (sx - j2 - 0.5)^2 +
        3 * (sy - i2 - 0.5)^2)
    xy = xy %>% mutate(L = case_when(dist1 < con1 ~
        floor(i1 * iinc + j1 + 1), dist1 > con2 ~
        floor(floor(sy) * iinc + floor(as.double(sx)) +
            lat), TRUE ~ case_when(dist1 <= con3 ~
        floor(i1 * iinc + j1 + 1), TRUE ~ floor(i2 *
        iinc + j2 + lat))))

    Lfulltbl = xy %>% select(x, y, L)
    cmsTbl = Lfulltbl %>% group_by(L) %>%
        summarise(xcm = mean(x), ycm = mean(y)) %>%
        arrange(L) %>% select(-L) %>%
        collect()
    Ltbl = xy %>% select(L2 = L, wt) %>% group_by(L2) %>%
        summarise(cnt = sum(wt))
    xy = xy %>% select(x, y) %>% mutate(L2 = row_number())
    xy = left_join(xy, Ltbl, by = "L2") %>%
        mutate(cnt = case_when(is.na(cnt) ~
            0, TRUE ~ cnt)) %>%
        arrange(L2)
    cntsTbl = xy %>% rename(cell = L2) %>%
        mutate(lt1 = case_when(cnt >  0 ~ 1, TRUE ~ 0)) %>%
        filter(lt1 == 1) %>%
        select(-lt1) %>% collect()

    out = list(cell = cntsTbl$cell, count = cntsTbl$cnt,
        xcm = cmsTbl$xcm, ycm = cmsTbl$ycm, xbins = size,
        shape = shape, xbnds = rx, ybnds = ry,
        dimen = bnd, n = n, ncells = nrow(cntsTbl),
        xlab = xName, ylab = yName)
    xy = svydbhcell2xy(out)
    out = c(xy, out)

    return(out)
}

svydbhexbin = function(formula, design, xbins = 30,
    shape = 1) {

    dsn = design$clone()
    dsn$setx(formula)
    dsn$storename("y", all.vars(formula)[1])
    dsn$storename("x", all.vars(formula)[-1])
    d = dsn$data
    d = d %>% rename(x = !!sym(dsn$names$x)) %>%
        rename(y = !!sym(dsn$names$y)) %>%
        rename(wt = !!sym(dsn$wt)) %>%
        filter(!is.na(x)) %>% filter(!is.na(y))
    d = compute(d)

    n = d %>% db_nrow()
    x = d %>% select(x)
    y = d %>% select(y)

    xbnds = c(dbmin(d, x), dbmax(d, x))
    ybnds = c(dbmin(d, y), dbmax(d, y))

    jmax = floor(xbins + 1.5001)
    c1 = 2 * floor((xbins * shape)/sqrt(3) + 1.5001)
    imax = trunc((jmax * c1 - 1)/jmax + 1)
    lmax = jmax * imax

    ans = svydbhbin(xy = d, x = x, y = y,
        xName = dsn$names$x, yName = dsn$names$y,
        cell = as.integer(lmax), cnt = as.integer(lmax),
        xcm = as.integer(lmax), ycm = as.integer(lmax),
        size = xbins, shape = shape,
        rx = as.double(xbnds), ry = as.double(ybnds),
        bnd = as.integer(c(imax, jmax)), n = n)

    return(ans)
}

svydbhexplot = function(d, xlab = d$xlab, ylab = d$ylab) {
    pdata = tibble(x = d$x, y = d$y, count = d$count,
        xcm = d$xcm, ycm = d$ycm)

    p = ggplot(pdata) + geom_hex(aes(x = x, y = y,
        fill = count),
            color = "black", stat = "identity") +
        labs(x = xlab, y = ylab) +
        scale_fill_continuous(trans = "reverse")

    print(p)

    invisible(p)
}
```

B.14 svydbcoplot

```
svydbcoplot = function(formula, by, design) {
    if (!is_formula(by)) {
        stop("by must be a formula")
    }

    y = all.vars(formula)[1]
    x = all.vars(formula)[-1]
    dsn = design$clone()

    by_var = all.vars(by)
    by = dsn$data %>% select(!!!syms(by_var)) %>%
        distinct() %>% arrange(!!sym(by_var[1])) %>%
        collect()
    by = split(by, seq(nrow(by)))

    filterData = function(by, dsn, x, y) {
        dsn = dsn$subset(paste(colnames(by), " == ",
            by, collapse = " & "), logical = F)
        hb = svydbhexbin(formula, design = dsn)
        if (length(hb$x) | length(hb$y)
                | length(hb$count) != 0) {
```

```
            cbind(tibble(x = hb$x, y =
                hb$y, count = hb$count),  by)
        }
    }

    p = lapply(by, filterData, dsn = dsn) %>%
        Reduce(rbind, .)

    p = ggplot(p) + geom_hex(aes(x = x, y = y,
        fill = count),
            color = "black", stat = "identity") +
        labs(x = x, y = y) +
        scale_fill_continuous(trans = "reverse") +
        facet_wrap(by_var, labeller = "label_both")

    print(p)

    invisible(p)
}
```

B.15 Other Functions

```
dbmin = function(data, var, asNum = T) {
    var = enquo(var)
    data = data %>% ungroup() %>% select(!!var) %>%
        summarise(min = min(!!var))
    if (asNum == T) {
        data %>% pull
    } else {
        data
    }
}

dbmax = function(data, var, asNum = T) {
    var = enquo(var)
    data = data %>% ungroup() %>% select(!!var) %>%
        summarise(max = max(!!var))
    if (asNum == T) {
        data %>% pull
    } else {
        data
    }
}

db_nrow = function(data) {
    data %>% ungroup() %>% count() %>% pull()
}

db_dim = function(data) {
    n_rows = data %>% ungroup() %>% count() %>%
        pull()
    n_cols = ncol(data)

    out = c(n_rows, n_cols)
    names(out) = c("rows", "cols")
    out
}

db_view = function(data, num = 1) {
    if (class(data) == "list") {
        View(data[[num]] %>% tbl_df())
    } else {
        View(data %>% tbl_df)
    }
}

db_rowSums_mut = function(data, vars = NULL,
                            newRowName = "rSum") {
    if (is.null(vars)) {
        s = paste(colnames(data), collapse = " + ")
    } else {
        s = paste(vars, collapse = " + ")
    }
    q = quote(mutate(data, rSum = s))

    eval(parse(text = sub("rSum", newRowName,
        sub("s", s, deparse(q)))))
}

db_rowSums = function(data) {
    cn = colnames(data)
    rs = paste("‘", cn, "‘", sep = "", collapse = " + ")
    data %>% transmute(rowsum = !!parse_expr(rs))
}

db_cbind = function(x, y) {
    x = x %>% mutate(‘___i‘ = "key")
    y = y %>% mutate(‘___i‘ = "key")
    out = inner_join(x, y, by = "___i", copy = T) %>%
        select(-‘___i‘)
    return(out)
}

db_slice = function(data, n) {
    n = enquo(n)
    data %>% mutate(‘___i‘ = row_number()) %>%
        filter(‘___i‘ <= !!n) %>% select(-‘___i‘)
}

dummy_mut = function(data, by, withBase = T,
                            return.level = F) {
    by = enquo(by)
    dum = data %>% distinct(!!by) %>% arrange(!!by) %>%
        data.frame() %>% na.omit()
    level = as.character(dum[, 1])
    cs = contrasts(as.factor(dum[, 1]))
    by_name = colnames(dum)

    if (withBase == T) {
        c1 = c(1, rep(0, nrow(dum) - 1))
        dum = cbind(dum, c1, contrasts(as.factor(dum[,
            1])))
        name = paste(colnames(dum)[1], as.character(dum[,
            1]), sep = "_")
        colnames(dum) = c(colnames(dum)[1], name)
    } else {
        dum = cbind(dum, contrasts(as.factor(dum[,
            1])))
        name = paste(colnames(dum)[1], as.character(dum[,
            1])[-1], sep = "_")
        colnames(dum) = c(colnames(dum)[1], name)
    }

    dum = inner_join(data, dum, by = by_name,
        copy = T)

    if (withBase == F) {
        dum = dum %>% select(-!!by)
    }

    if (return.level == T) {
        dum = list(dum = dum, levels = level)
        return(dum)
    } else {
        return(dum)
    }
}

db_cut = function(var, breaks, data) {
    var = enquo(var)
    var_name = as.character(var)[2]
    data = data %>% rename(vars = !!var) %>%
        filter(!is.na(vars))
    breaks = breaks[-1]
    trues = seq(length(breaks))
    temp_exprs = paste("ifelse(vars", "<= ", breaks,
        ",", trues, ", _f_)")
    temp_exprs[length(breaks)] = gsub(pattern = "[_]f[_]",
        replacement = "NA", x = temp_exprs[length(breaks)])
    mut_exprs = temp_exprs[1]

    for (i in 2:length(breaks)) {
        mut_exprs = gsub(pattern = "[_]f[_]",
            replacement = temp_exprs[i], x = mut_exprs)
    }

    mut_exprs = parse_expr(mut_exprs)
    data = data %>% mutate(‘_cuts_‘ = !!mut_exprs) %>%
        rename(‘:=‘(!!sym(var_name), vars)) %>%
        rename(cuts = ‘_cuts_‘)
    data
}

db_cut2 = function(var, breaks, right = TRUE,
    data) {
    var = enquo(var)
    mult = diff(breaks)[1]

    if (right == TRUE) {
        data = data %>% mutate(cut = ((!!quo(mult)) *
            ceiling((!!var)/(!!quo(mult)))) -
            (!!quo(mult)))
    } else {
        data = data %>% mutate(cut = ((!!quo(mult)) *
            floor((!!var)/(!!quo(mult)))))
    }

    return(data)
}

svydbVar2 = function(x, xleft = 1, xright = 2,
    st, m_h, data) {
    xleft = x[xleft]
    xright = x[xright]
    m = data
    m = m %>% select(xleft = !!sym(xleft),
        xright = !!sym(xright), st = st, m_h = m_h)
    m_h = m %>% select(st, m_h) %>% mutate(m_h = 1) %>%
        group_by(st) %>% summarise(m_h = sum(m_h))
    m = m %>% select(-m_h)
    m = m %>% mutate(ztz = xleft * xright) %>%
        group_by(st) %>% summarise(ztz = sum(ztz))
    m = left_join(m, m_h, by = "st")
    m = m %>% mutate(scaled = ztz * (m_h/(m_h - 1))) %>%
        select(scaled) %>% summarise(sum(scaled)) %>%
        compute(temporary = T) %>% pull()
    return(m)
}
svydbVar = function(x, st, m_h, data) {
    m = data
    m = m %>% select(x = x, st = st, m_h = m_h)
    m_h = m %>% select(st, m_h) %>% mutate(m_h = 1) %>%
        group_by(st) %>% summarise(m_h = sum(m_h))
    m = m %>% select(-m_h)
    m = m %>% mutate(ztz = x * x) %>% group_by(st) %>%
        summarise(ztz = sum(ztz))
    m = left_join(m, m_h, by = "st")
    m = m %>% mutate(scaled = ztz * (m_h/(m_h - 1))) %>%
        select(scaled) %>% summarise(sum(scaled)) %>%
        compute(temporary = T) %>% pull()
    return(m)
}

c2f = function(x) {
    as.formula(paste("~", x, sep = ""))
}

db_columnAsCharacter = function(x, cols){

  checktype =  x %>% select(!!!syms(cols)) %>%
    head(1) %>% collect %>% lapply(type_sum) %>%
    unlist()

  numCols = checktype[checktype %in%
                c("int", "dbl")] %>% names()

  for(i in 1:length(numCols)){
    x = x %>%
        mutate(!!quo_name(numCols[1]) :=
            as.character(!!sym(numCols[1])))
  }

  x
}

svydb_monet_sampleN = function(data, n) {
    q = paste("SELECT * FROM", data$ops$x, "SAMPLE",
        n)
    dbGetQuery(data$src$con, q)
}

as.data.frame.svydbstat = function(x) {
    ans = cbind(coef(x), SE(x))
    colnames(ans) = c(attr(x, "statistic"), "SE")
    ans
}

print.svydbstat = function(xx, ...) {
    v <- attr(xx, "var")
    m = cbind(xx, sqrt(v))
    colnames(m) = c(attr(xx, "statistic"), "SE")
    printCoefmat(m)
}

coef.svydbstat = function(object, ...) {
    attr(object, "statistic") = NULL
    attr(object, "name") = NULL
    attr(object, "var") = NULL
    unclass(object) %>% t() %>% as.vector()
}

SE.svydbstat = function(x, ...) {
    s = attr(x, "var") %>% sqrt()
    names(s) = attr(x, "name")
    return(s)
}

print.svydbrepstat = function(xx, ...) {
    if (is.list(xx)) {
        xx = xx$svydbrepstat
    }
    v <- attr(xx, "var")
    m = cbind(xx, sqrt(v))
    colnames(m) = c(attr(xx, "statistic"), "SE")
    rownames(m) = attr(xx, "name")
    printCoefmat(m)
}

coef.svydbrepstat = function(object, ...) {
    if (is.list(object)) {
        object = object$svydbrepstat
    }
    attr(object, "statistic") = NULL
    attr(object, "name") = NULL
    attr(object, "var") = NULL
    unclass(object) %>% t() %>% as.vector()
}

SE.svydbrepstat = function(x, ...) {
    if (is.list(x)) {
        x = x$svydbrepstat
    }
    s = attr(x, "var") %>% sqrt()
    names(s) = attr(x, "name")
    return(s)
}

print.svydblm = function(x, digits =
    max(3L, getOption("digits") - 3L), ...) {
    print(x$design)
    cat("\nSurvey design:\n")
    print(x$design$call)
    cat("\nCall:\n", paste(deparse(x$call), sep = "\n",
        collapse = "\n"), "\n\n", sep = "")
    if (length(coef(x))) {
        cat("Coefficients:\n")
        print.default(format(coef(x), digits = digits),
            print.gap = 2L, quote = FALSE)
    } else cat("No coefficients\n")
    cat("\n")
    invisible(x)
}
summary.svydblm = function(object) {
    df.r = object$df.residual
    coef.p = coef(object)
    covmat = vcov(object)
    dimnames(covmat) = list(colnames(coef.p),
        colnames(coef.p))
    var.cf = diag(covmat)
    s.err = sqrt(var.cf)
    tvalue = coef.p/s.err
    dn = c("Estimate", "Std. Error")
    pvalue <- 2 * pt(-abs(tvalue), df.r)
    coef.table <- rbind(coef.p, t(as.matrix(s.err)),
        tvalue, pvalue) %>% t()
    dimnames(coef.table) <- list(colnames(coef.p),
        c(dn, "t value", "Pr(>|t|)"))
    ans = list(df.residual = df.r,
        coefficients = coef.table,
        cov.unscaled = covmat, cov.scaled = covmat,
        call = object$call, design = object$design)
    class(ans) <- c("summary.svydblm", "summary.glm")
    return(ans)
}
print.summary.svydblm = function(x, digits = max(3,
    getOption("digits") - 3),
    signif.stars = getOption("show.signif.stars"), ...) {
    cat("\nCall:\n")
    cat(paste(deparse(x$call), sep = "\n", collapse = "\n"),
        "\n\n", sep = "")
    cat("Survey design:\n")
    print(x$design$call)
    cat("\nCoefficients:\n")
    coefs <- x$coefficients
    if (!is.null(aliased <- is.na(x$coefficients[,
        1])) && any(aliased)) {
        cn <- names(aliased)
        coefs <- matrix(NA, length(aliased), 4,
            dimnames = list(cn, colnames(coefs)))
        coefs[!aliased, ] <- x$coefficients
    }
    printCoefmat(coefs, digits = digits,
        signif.stars = signif.stars,
        na.print = "NA", ...)
    invisible(x)
}
```

```
predict.svydblm = function(object, newdata = NULL,
    ...) {
    tt = delete.response(object$terms)
    mf = model.frame(tt, data = newdata,
        xlev = object$design$levels)
    mm = model.matrix(tt, mf)
    eta = drop(mm %*% as.vector(coef(object)))
    attr(eta, "var") = drop(rowSums((mm %*% vcov(object)) *
        mm))
    attr(eta, "statistic") = "link"
    class(eta) <- "svydbstat"
    eta
}

vcov.svydblm = function(x, ...) {
    x$cov.unscaled
}
```

Bibliography

Bache, Stefan Milton and Hadley Wickham (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5. URL: https://CRAN.R-project.org/package=magrittr.

Carr, D. B. et al. (1987). “Scatterplot Matrix Techniques for Large N”. In: *Journal of the American Statistical Association* 82.398, pp. 424–436. ISSN: 01621459. URL: http://www.jstor.org/stable/2289444.

Carr, Dan et al. (2018). *hexbin: Hexagonal Binning Routines*. R package version 1.27.2. URL: https://CRAN.R-project.org/package=hexbin.

Chang, Winston (2017). *R6: Classes with Reference Semantics*. R package version 2.2.2. URL: https://CRAN.R-project.org/package=R6.

Henry, Lionel and Hadley Wickham (2018). *rlang: Functions for Base Types and Core R and 'Tidyverse' Features*. R package version 0.2.0. URL: https://CRAN.R-project.org/package=rlang.

Horvitz, D. G. and D. J. Thompson (1952). “A Generalization of Sampling Without Replacement From a Finite Universe”. In: *Journal of the American Statistical Association* 47.260, pp. 663–685. ISSN: 01621459. URL: http://www.jstor.org/stable/2280784.

Lewin-Koh, Nicholas (2016). *Hexagon Binning: an Overview*. URL: https://cran.r-project.org/web/packages/hexbin/vignettes/hexagon_binning.pdf.

Lumley, Thomas (2004). “Analysis of Complex Survey Samples”. In: *Journal of Statistical Software* 9.1. R package verson 2.2, pp. 1–19.

— (2014). *sqlsurvey: analysis of very large complex survey samples (experimental)*. R package version 0.6-11/r41. URL: https://R-Forge.R-project.org/projects/sqlsurvey/.

Luraschi, Javier et al. (2018). *sparklyr: R Interface to Apache Spark*. R package version 0.8.3. URL: https://CRAN.R-project.org/package=sparklyr.

MonetDB-B.V. (2008). *MonetDB Database System*. URL: https://www.monetdb.org/.

Ruiz, Edgar (2018). *dbplot: Simplifies Plotting Data Inside Databases*. R package version 0.3.0. URL: https://CRAN.R-project.org/package=dbplot.

Sievert, Carson et al. (2017). *plotly: Create Interactive Web Graphics via 'plotly.js'*. R package version 4.7.1. URL: https://CRAN.R-project.org/package=plotly.

Sturges, Herbert A. (1926). “The Choice of a Class Interval”. In: *Journal of the American Statistical Association* 21.153, pp. 65–66. ISSN: 01621459. URL: http://www.jstor.org/stable/2965501.

Wickham, Hadley (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org.

Wickham, Hadley, Peter Danenberg, and Manuel Eugster (2017). *roxygen2: In-Line Documentation for R*. R package version 6.0.1. URL: https://CRAN.R-project.org/package=roxygen2.

Wickham, Hadley, Jim Hester, and Winston Chang (2018). *devtools: Tools to Make Developing R Packages Easier*. R package version 1.13.5. URL: https://CRAN.R-project.org/package=devtools.

Wickham, Hadley and Edgar Ruiz (2018). *dbplyr: A 'dplyr' Back End for Databases*. R package version 1.2.1. URL: https://CRAN.R-project.org/package=dbplyr.

Wickham, Hadley et al. (2017). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.4. URL: https://CRAN.R-project.org/package=dplyr.

Xie, Yihui (2017). *formatR: Format R Code Automatically*. R package version 1.5. URL: https://CRAN.R-project.org/package=formatR.