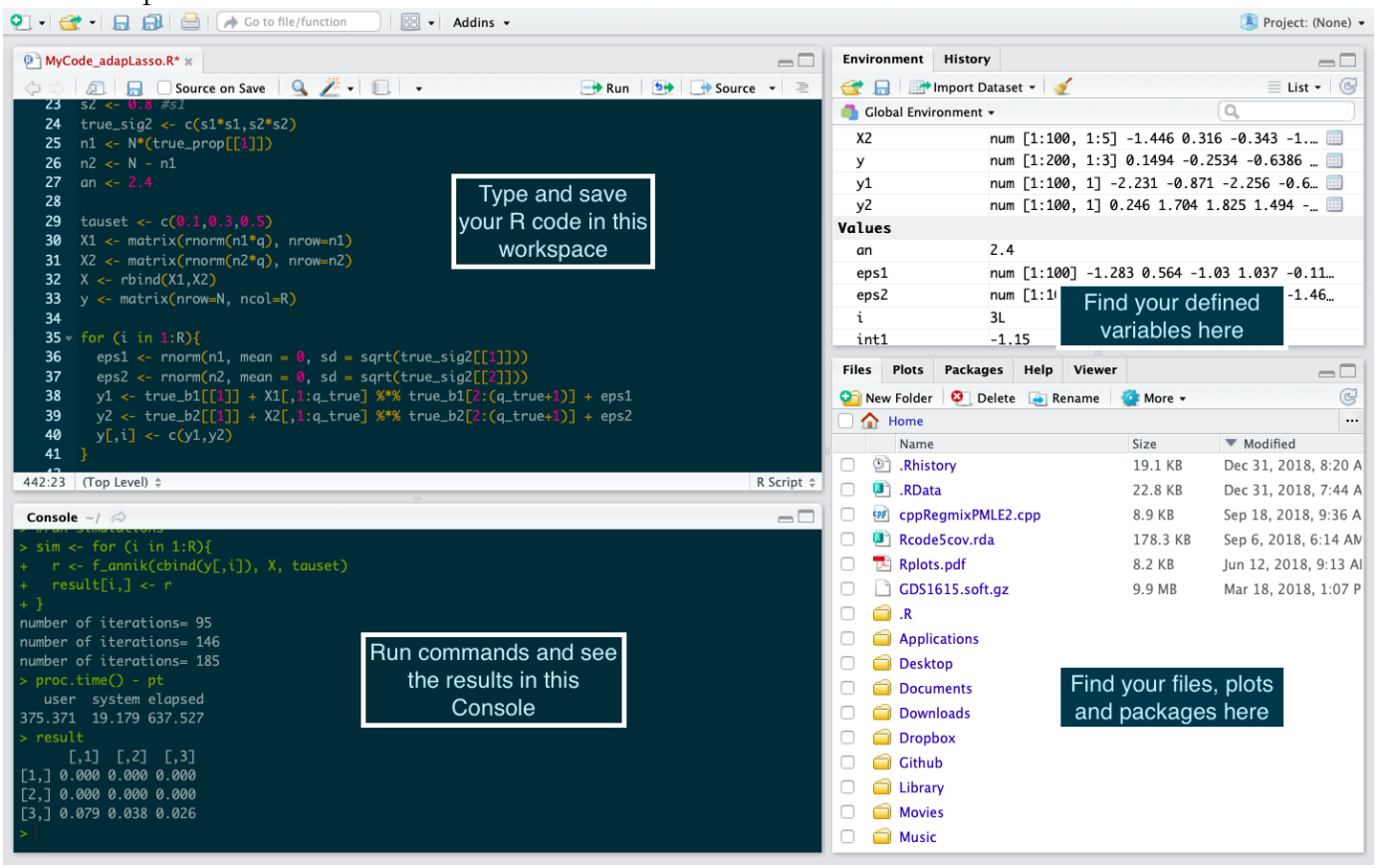
Canadian Housing Statistics Program

Contents

1 Intro to RStudio

RStudio is an interface that makes R easier to use. Base R can be used if you are familiar with the software already. However, the interface is greatly improved with RStudio. Make sure that both R and RStudio are installed on your Net A computer.

Here is a quick look at RStudio:



Here are a few starting tips with RStudio:

- To create a new R Script: File → New File → R Script
- To set the working directory: Session → Set Working Directory → Choose Directory (or simply enter the command `setwd("C:/myfolder/data")` in the console)
- Type your code in the workspace instead of in the console. Then, highlight the code you want to run and click the Run button on the top of the workspace (or control+enter on your keyboard). R will output your answer in the console. By working in the workspace instead of the console, you can save it and edit your code as you go!

2 Installing Packages in R on Net A

This information is adapted from the following page on Confluence, written by Jason Jenson.

Packages in R are collections of R functions, data and code within a defined format. The directory where they are stored is called a library. R comes with standard functions and others are available for download and installation. To use a package in R, you must first install the package on your computer. Then, you need to call on the library every session in which you wish to use a function from that package.

StatCan maintains an internal CRAN repository, which is a repository for R packages. Therefore, you can install packages from this repository onto their Net A computers.

Usually, you can install packages in R using the following command:

```
install.packages("your_package_name")
```

However, this does not run on Net A since we don't have internet access. We must indicate to retrieve the packages from StatCan's CRAN repository. We can do so by creating an .Rprofile file in the documents folder. This can be created under your F drive, in the Profile-Data folder, then in Documents. The file path where I created my file is //stchsfs/gougann$/Profile-Data/Documents.

Open Notepad and copy and paste the following:

```
## set packages repo
options(repos =
c(CRAN="https://artifactory.statcan.ca:8443/artifactory/cran"))

## set package install directory
Sys.setenv(R_LIBS_USER = "F:/config/R/win-library/3.4")

## make sure that packages get installed there
invisible(.libPaths(c(unlist(strsplit(Sys.getenv("R_LIBS"), ";")),
unlist(strsplit(Sys.getenv("R_LIBS_USER"), ";"))
)))

## Confirm that this profile has been loaded
message("**** Loaded .Rprofile ****")
```

Then save this file with the extension type "All types (*.*)" and name it .Rprofile. Open RStusdio, set your working directory (Session → Set Working Directory → Choose Directory) and select the folder with your .Rprofile.

Now, packages should be easy to install! When you open your RStudio, you should see the following message in the console:

```
**** Loaded .Rprofile ****
```

Now you can install packages in the usual way, using the install.packages function in R! You should only have to run this once. If your packages are not loading properly, check if your working directory is still set to your folder containing the .Rprofile.

To use a package in your code, call on to the library with the following code. This has to be run every new session.

```
library(your_package_name)
```

If you are still encountering issues with installing packages on the VDI, please consult : https://rpug.pages.cloud.statcan.ca/en/r-vdi/
Additional details on the installation of packages on Net A are also available here : https://rpug.pages.cloud.statcan.ca/en/r-packages/

3 Importing SAS Datasets in R

The next step is to load data (usually kept in .sas7bdat format) into RStudio. Luckily, this is very straightforward. We will use the haven library for this, so ensure that it is installed and loaded in your session.

Here is the code we will use to import the entire dataset

```
library(haven)
dataframe_name <- read_sas("Y:/filepath/filename.sas7bdat")
```

Since importing the entire dataset might take lots of memory, we can select the number of columns by using the cols_only argument:

```
library(haven)
dataframe_name <- read_sas("Y:/filepath/filename.sas7bdat",
                           cols_only = c("col1_name","col2_name"))
```

To import data using SQL instead, allowing for more flexibility in the import process, is significantly more complex. I included those steps in the Appendix.

4 R Programming

4.1 Getting Started

Here are quick examples of the basics of R programming

- You can do the usual arithmetic computations in R, the same way you would in any other programming language.
- To assign values to an object name, we use the following synthax: *name of object* $<-$ *value of object*. Note that the # denotes a comment. When assigning values to an object, R will not output those values in the console. To see the values of an object, run the name of the object

```
my_nume_name <- 2 #numeric variable
my_nume_name #run this line and R will output 2 in the console
my_chara_name <- "CHSP" #character variable
logic_name <- TRUE #logical variable
```

- To create a **vector**, we use the c() function for concatenate

```
my_vect <- c(2,4,5,6,7)
```

- We can now do calculations with the vector, such as addition and substraction. Note that R will execute this for each number in the vector individually. You can also use functions such as the mean, median, sd, etc.

```
my_vect + 2 #the output is 4 6 7 8 9
mean(my_vect) #the output is 4.8
sd(my_vect) #the output is 1.923538
length(my_vect) #this returns the number of elements, so 5
```

- We can create a **matrix**, containing mutliple rows and columns, which is a combination of multiple vectors with the same types (num, char or logical).

```
my_vect2 <- c(3,5,8,9,10)
my_matr <- cbind(my_vect, my_vect2)
#cbind stands for column bind, creating columns with the two vectors
dim(my_matr) #the output is 5 2
colSums(my_matr) #sums over the columns, the output is 24 35
```

- A **data frame** is like a matrix but with columns of any type. It's the format we use most often to store data. Let's create a data frame with 3 observations and 3 columns, one for the property ID, one for the CSD and one with the assessment value. The columns are called ID, CSD and AV.

```
my_dat <- data.frame("ID" = c(1,2,3),
"CSD" = c("Whistler","Whistler","Quesnel"),
"AV" = c(250000, 567000, 459000))
```

We can work with a specific column by using the $ symbol. For example, to find the mean assessment value of those three properties:

```
my_dat$AV #the output is the vector containing 250000 567000 459000
mean(my_dat$AV) #the output is 425333.3
```

- If you ever want more information on a particular R function (included those in loaded packages), simply type a ? followed by the name of the function in the console

```
?mean
?length
#this will output instructions in the lower right hand side of RStudio
```

4.2 Extraction and Subsetting

We will define extraction as pulling out specific numbers from a vector, while subsetting is extracting certain rows and columns from a data frame. These two objectives require similar synthax in R.

4.2.1 Extracting values from vectors

Remember that we defined our vector as my_vect.

- To extract a value (e.g. the number at the fourth position) of my_vect, we use the following synthax

```
my_vect[4] #the output is 9
```

- To modify the fourth value of the vector

```
my_vect[4] <- 900
my_vect #the output is 3 5 8 900 10
```

- Or we can save the value in a new object

```
my_object <- my_vect[4]
```

- We can also extract multiple values at a time, using a series. The following two lines of code are equivalent

```
my_vect[c(1,2,3)] #the output is 3 5 8
my_vect[1:3] #1:3 produces 1 2 3 so the output is the same as above
```

- We can use logical operators to extract certain values. Note that the == in R is used to test equality.

```
my_vect == 3 #this returns TRUE FALSE FALSE FALSE FALSE FALSE
my_vect > 5 #this outputs FALSE FALSE TRUE TRUE TRUE
my_vect[my_vect > 5] #this outputs all values with TRUE, so 8 900 10
```

4.2.2 Subsetting from data frames

Let's begin by importing some synthetic CHSP data, available in the Stage 5 folders. We will import it as a data frame into the name CHSP, with the following columns: CHSP_PROP_ID, AV_RES_TOT and CSD_2016, LV_AREA_TOT.

```
library(haven)
CHSP <- read_sas("//fld5filer/CHSP/Stage5_analysis/Data/Vintage/2018_12/
35_ON/Synthetic/fake_data_on.sas7bdat", cols_only = c("CHSP_PROP_ID",
"AV_RES_TOT", "CSD_2016", "LV_AREA_TOT"))
CHSP <- as.data.frame(CHSP) #this transforms the data into dataframe format
```

To extract rows and/or columns from a dataframe, we use the [,] seperated by a comma. The first index is for the extraction of rows and the second is for the extraction of columns. Keeping the index empty keeps all observations from that row or column.

```
CHSP[1,3] #this extracts the observation in the 1st row and 3rd column
CHSP[,3] #this extracts all observations (rows) from the 3rd column
CHSP[1:10,] #this extracts the first 10 rows and all columns
```

We can also select columns using the $ symbol and the name of the column. In RStudio, a list of available column names will appear once you type the dataframe's name and include the $ symbol.

```
CHSP$CHSP_PROP_ID
CHSP$AV_RES_TOT
```

We can select observations based on the values of variables, creating a subset of selected observations. It is good practice to use the which() function, which returns the row numbers of the observations of interest.

```
which(CHSP$AV_RES_TOT > 1000000)
#returns row index of all properties with an AV greater than one million
which(CHSP$CSD_2016 == 3520005)
#returns row index of all properties in CSD 3520005
```

Let's combine these concepts to subset all rows who have an assessment value greater than one million dollars in the Toronto CSD (CSD 3520005). We will save our subset (which is still in the data frame format) under the name TO_CSD.
**Note the comma, followed by the]. This indicates that we wish to keep all columns in our subset.

```
TO_CSD <- CHSP[which(CHSP$AV_RES_TOT > 1000000 & CHSP$CSD_2016 == 3520005),]
```

4.3 Plots

The package "ggplot2" in R is definitely the most versatile and customizable tool to create graphs in R. A good tip is to google the type of plot you want to create in ggplot, and find nice examples you can ~~steal~~ borrow.
Make sure the package is installed on your computer, then call in the library in your session.

```
library(ggplot2)
```

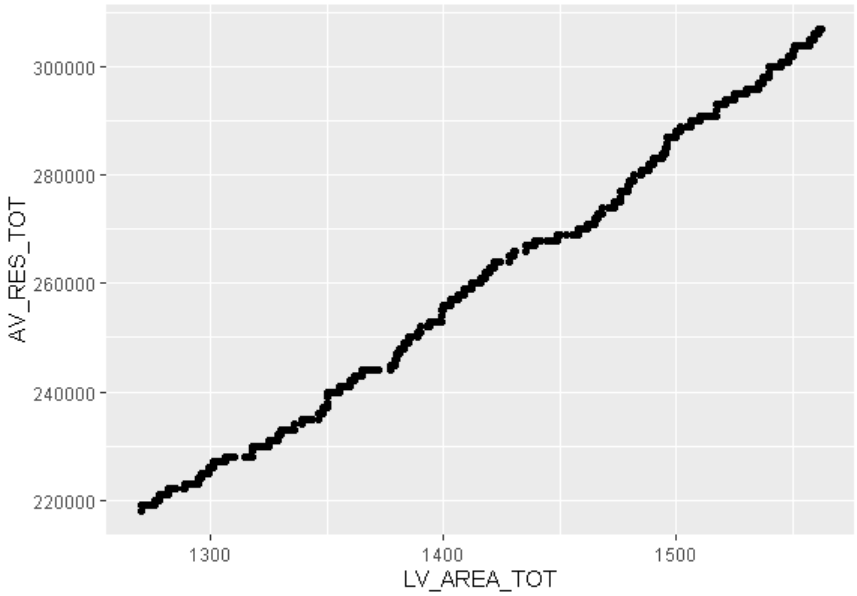
Every ggplot figure has 3 components:

1. **Data**: defining the source data set
2. **Aesthetic mapping**: defines which variables will be visualized from the dataset
3. **Layers**: establish what the plot will look like. All ggplots have at least one layer with the geom function to define the type of graph (scatter, histogram, boxplot). Other layers allow for further customization with labels, legends, colours, etc.

Let's create a scatterplot examining the impact of living area on assessment value in the Toronto CSD. First, we create a subset of all the properties in the Toronto CSD. We run the ggplot function ggplot() with the dataset name first, then open the aes function aes() to define our x and y variables. Note that the aes function is located inside the ggplot function (notice the parenthesis). To add a layer, we include a + and follow it with geom_point() which is used to create a scatter plot.

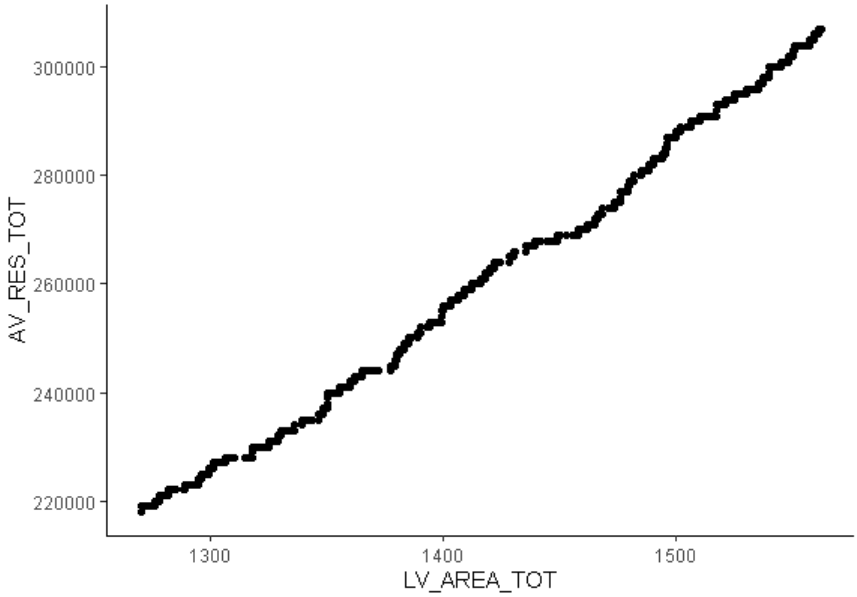
```
TO_CSD <- CHSP[which(CHSP$CSD_2016 == 3520005),]
ggplot(data = TO_CSD, aes(x = LV_AREA_TOT, y = AV_RES_TOT)) +
        geom_point()
```

Here is the result. Clearly, this data is fake since this is almost a perfect relationship.



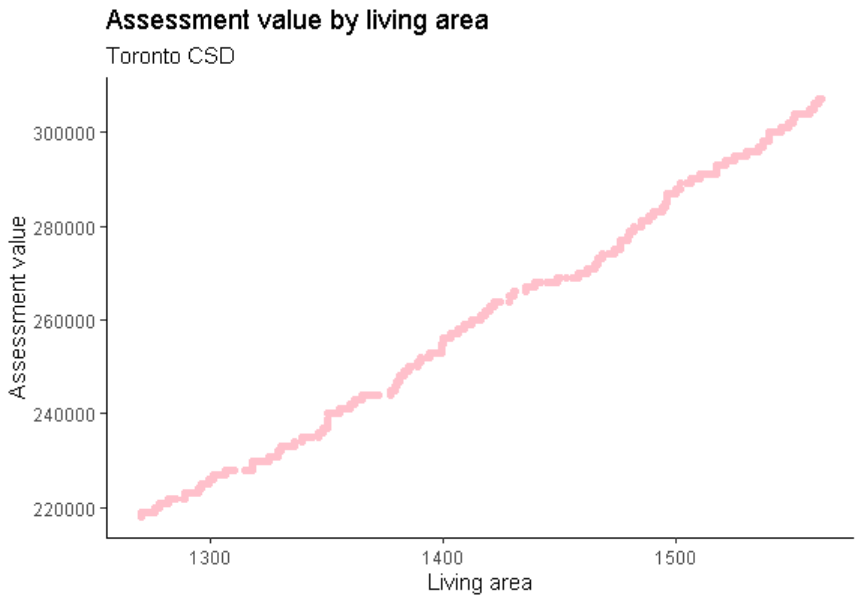
We can easily customize the graph by modifying the current layers or adding additional ones. For example, R has built-in themes that will change the axis, background colour, etc. My favourite one is theme_classic(), which we can add as an additional layer. This is what the code and graph looks like with this theme

```
ggplot(data = TO_CSD, aes(x = LV_AREA_TOT, y = AV_RES_TOT)) +
        geom_point() +
        theme_classic()
```



We can change the colours of the points by adding a parameter to the geom_point function. We can also add another layer (called labs) to change the title and axis titles. Here is an example

```
ggplot(data = TO_CSD, aes(x = LV_AREA_TOT, y = AV_RES_TOT)) +
        geom_point(colour = "pink") +
        labs(title="Assessment value by living area",
                subtitle = "Toronto CSD",
                x = "Living area",
                y = "Assessment value") +
        theme_classic()
```

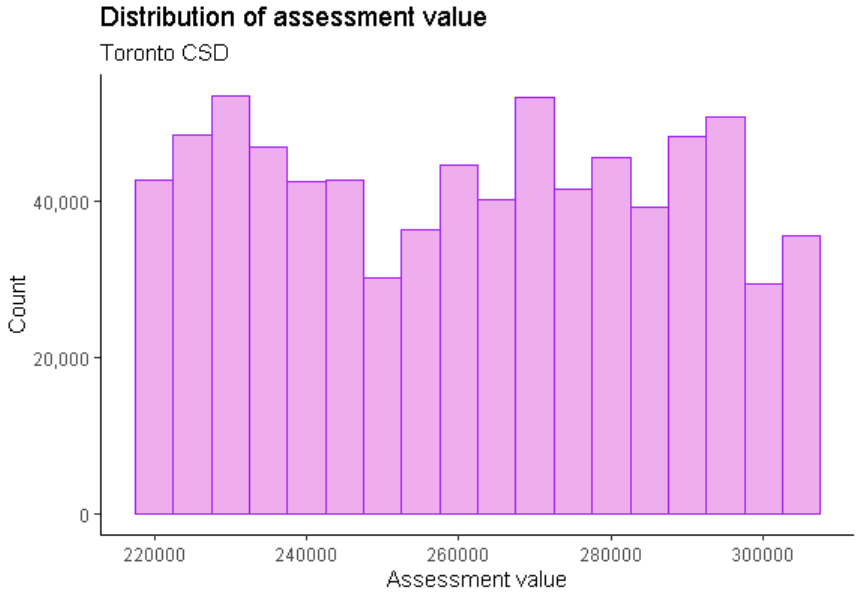


Let's add a regression line through our scatterplot. We add a layer with the function geom_smooth, which adds a trend line. We will specify that this should be a simple linear regression so we define the method as lm. The size parameter decreases the size of the line by 50

```
ggplot(data = TO_CSD, aes(x = LV_AREA_TOT, y = AV_RES_TOT)) +
        geom_point(colour = "pink") +
        geom_smooth(method="lm", colour = "black", size=0.5) +
        labs(title="Assessment value by living area",
                subtitle = "Toronto CSD",
                x = "Living area",
                y = "Assessment value") +
        theme_classic()
```

We can also create histograms to view the distribution of variables. For example

```
ggplot(data = TO_CSD, aes(x = LV_AREA_TOT, y = AV_RES_TOT)) +
geom_point(colour = "pink") +
labs(title="Assessment value by living area",
subtitle = "Toronto CSD",
x = "Living area",
y = "Assessment value") +
theme_classic()
```



This is just scratching the surface of types of fully customizable plots available with ggplot. As mentioned earlier, google is an excellent reference to learn more and see examples.

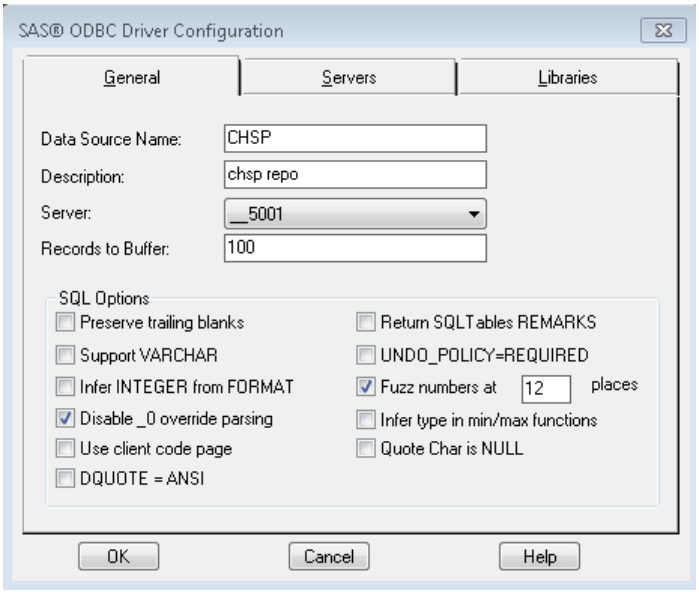
5 CHSP Queries

6 Appendix

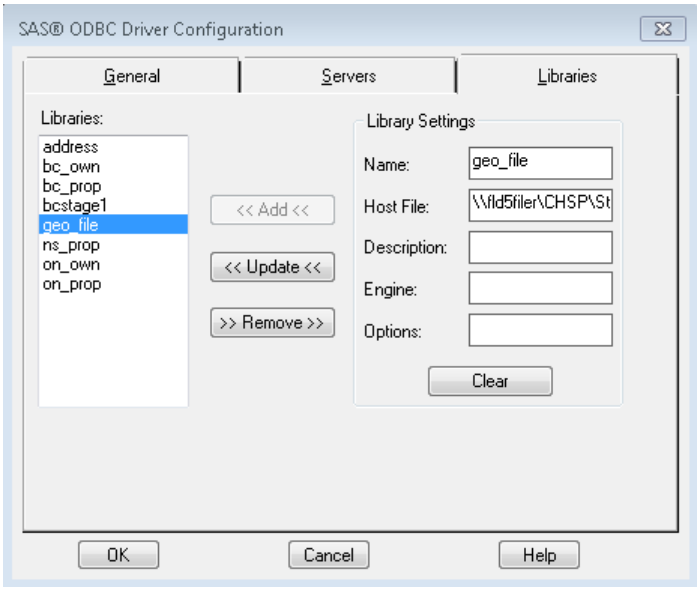
6.1 Importing data in R using SQL

This information is adapted from the following page on Confluence, written by Jason Jenson.

Let's say we want to import the geography information which includes the CMA and CSD names. We must first establish a connection through the ODBC. Simply search for ODBC in your Windows search bar. Now, we will create the SAS connection.
Click on Add, then select a SAS driver (note that we can also create one for SQL drivers, if data is stored in an SQL format such as the Business Register). Type a Data Source Name (e.g. CHSP) and a description (e.g. cshp repo). Ensure the Server is set at __5001. Click ok and you should now see CHSP under your User Data Sources.



Double click on your new data source then go to the Libraries tab. Create a library name (e.g. geo_files) and type the file path as the Host File (e.g. \\fld5filer\CHSP \Stage5_Analysis\Data\Vintage\2016_00\00_CA\Geography\OutputFiles). Click Add and make sure it is listed under library settings.



Now we can get out of ODBC and go open RStudio. We will use the package RODBC to import the data, so make sure this package is installed and the library is loaded. We must now open a connection, create an SQL query to pull the required data, assign this to a dataframe name and close the connection. Here is an example of code, continuing with the geography example.

```
conn <- odbcConnect("CHSP", believeNRows = FALSE, rows_at_time = 1)
query <- "SELECT CMA_CA_2016_E as CMA_CA_NAME,
   CMA_CA_2016 as CMA_2016, CSD_2016,
   CSD_2016_E as CSD_NAME, GEO_TYPE
  FROM   geo_files.geomaster
  WHERE  GEO_TYPE in ('CSD')"
geo <- sqlQuery(conn, query)
odbcClose(conn)
```

Of note here is that, in the first line of code, we call in the Data Source we created named CHSP. The query section is SQL code where we denote which columns we wish to import, and any other restrictions (e.g. in the WHERE statement). It is important that the dataset defined in the FROM statement first contains the library name (geo_files) and the actual dataset name (geomaster). Finally, we name our imported dataset (geo) and close the connection.

We now have our data in R! Note that a new library must be created in ODBC whenever the files are pulled from a different folder.