Introduction to Numeric Data Cleaning and Visualization Using R

Berenica Vejvoda, Data Librarian

berenica.vejvoda@mcgill.ca

McGill Library Data Lab Workshop

November 7, 2017

Introduction

R is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing.

Why R?

- Free
- Works on all operating systems
- Excellent and efficient graphical capacities
- Large and active community
- Many methods not available in other commercial software
- Good integration between programming language and statistical functions
- Good integration with a number of database systems and other languages

Setting up the environment

For this workshop R and R Studio are already installed on the workstations in the McGill Library Data Lab. If you have a laptop and wish to install R and R Studio, here are post-workshop instructions.

R

Download the latest version of R from the following link:

Windows

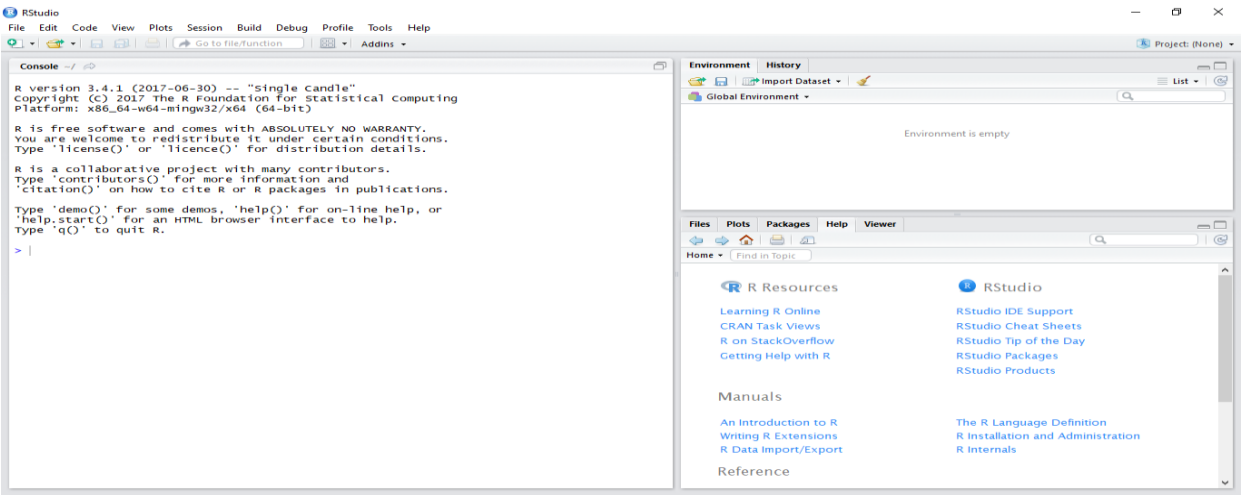
- https://cran.r-project.org/bin/windows/base/
- Install the (.exe) by double clicking it and accepting the default settings.
- You can use the R-GUI which comes by default with the installation or as specified in the next section use an IDE (RStudio)

Linux

- The instruction to install Linux varies from flavor to flavor
- Easiest way –
  - yum install R

R Studio

RStudio is a free and open source integrated development environment (IDE) for R. It consists of a console where the user can type the R commands and an environment which shows the data that has been loaded in the R environment. R should be installed first. R Studio once downloaded and installed should detect your R installation.



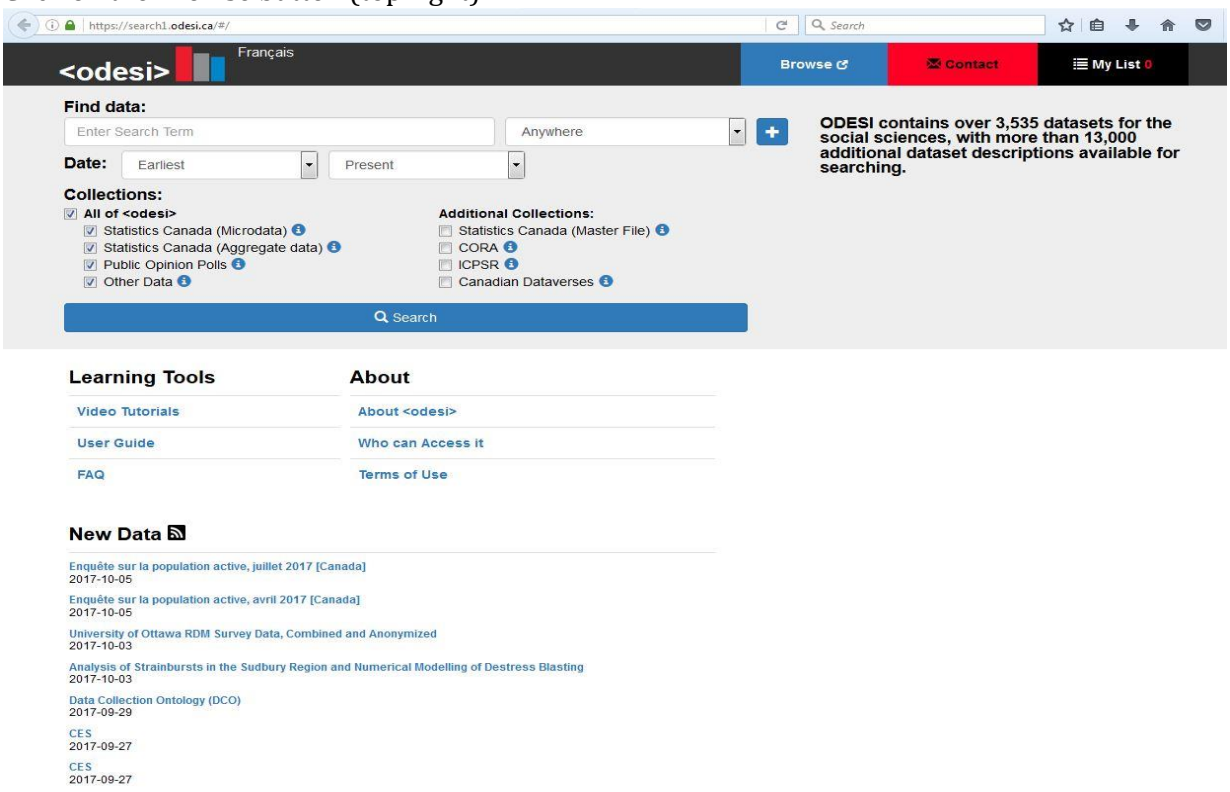
- Download the latest version of R Studio from the following link:

  https://www.rstudio.com/products/rstudio/download/

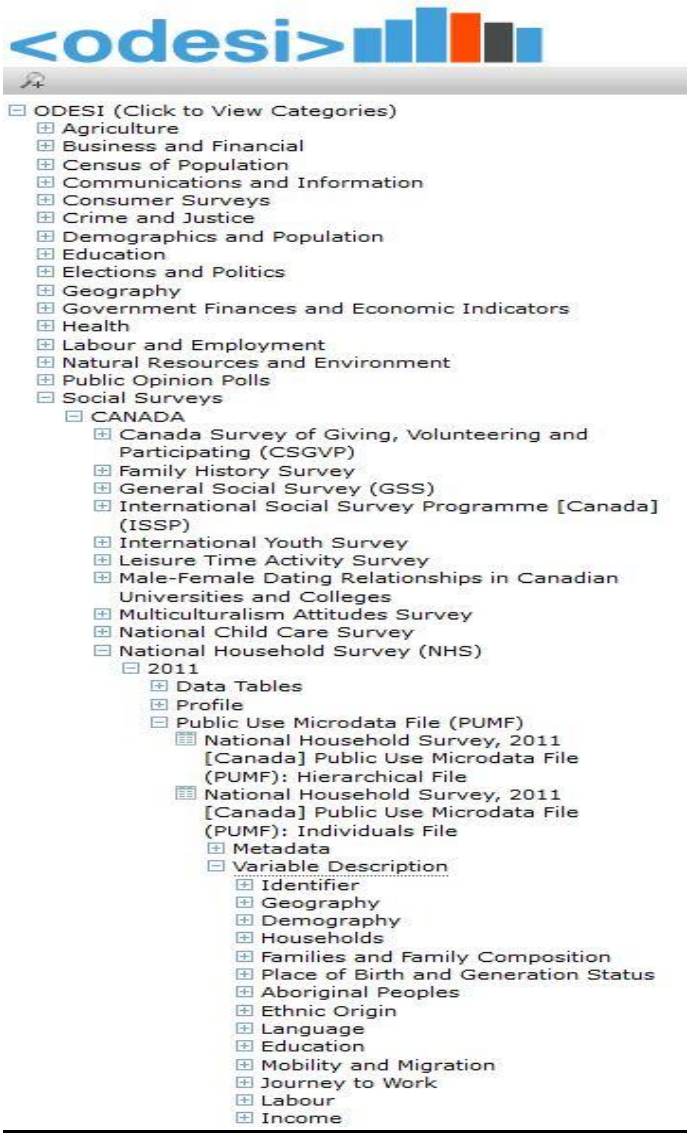
- Select the free "RStudio Desktop version" and download the Installer specific for your OS.

Downloading the National Household Survey, 2011 (NHS) data from ODESI

- Go to http://search1.odesi.ca/# /
- Click on the Browse button (top right)



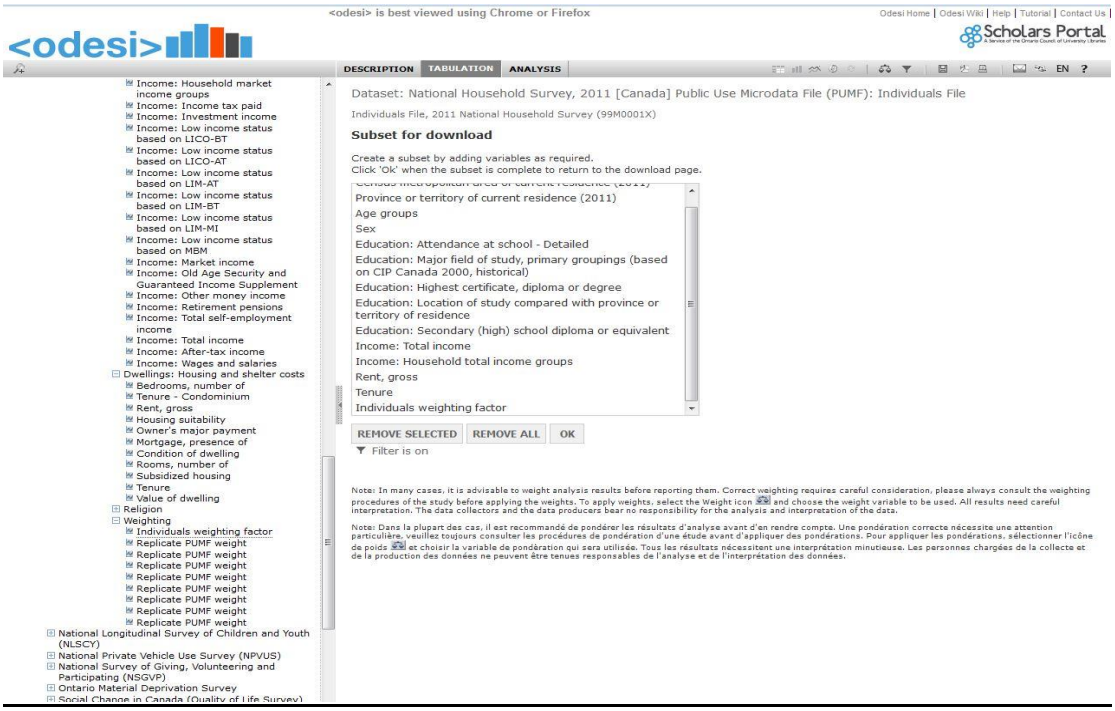
- Drill Down under ‘Social Surveys’
    - Click ‘Canada’
    - Click ‘National Household Survey’
    - Click ‘2011’
    - Click ‘Public Use Microdata File (PUMF)’
    - Click ‘Individuals file’
    - Click ‘Variable Description’



- Clicking on the ‘floppy disk’ icon on top right
- Choose the Data Format as ‘Comma Separated Value file’
- Since we only want the subsets –
    - Click on SUBSET

- Choose the following 14 variables to add to the 'subset for download' window. To add a variable to subset, simply click on the variable and select "add to subset".

    - Geography
        - Census metropolitan area of current residence (2011)
        - Province or territory of current residence (2011)
    - Demography
        - Age Groups
        - Sex
    - Education
        - Education: Attendance at school – Detailed
        - Education: Major field of study, primary groupings (based on CIP Canada 2000, historical)
        - Education: Highest certificate, diploma or degree
        - Education: Location of study compared with province or territory of residence
        - Education: Secondary (high) school diploma or equivalent
    - Income
        - Income: Household total income groups
        - Income: Total income
    - Dwelling
        - Rent, gross
        - Tenure
    - Weighting
        - Individuals weighting factor



- Click OK
- Select “include documentation”
- Click on “DOWNLOAD”

After downloading the dataset, create a new folder on drive P. Name that folder ‘r_tutorial’. Save the dataset to that folder. Unzip the file.

OPEN Rstudio from your workstation.

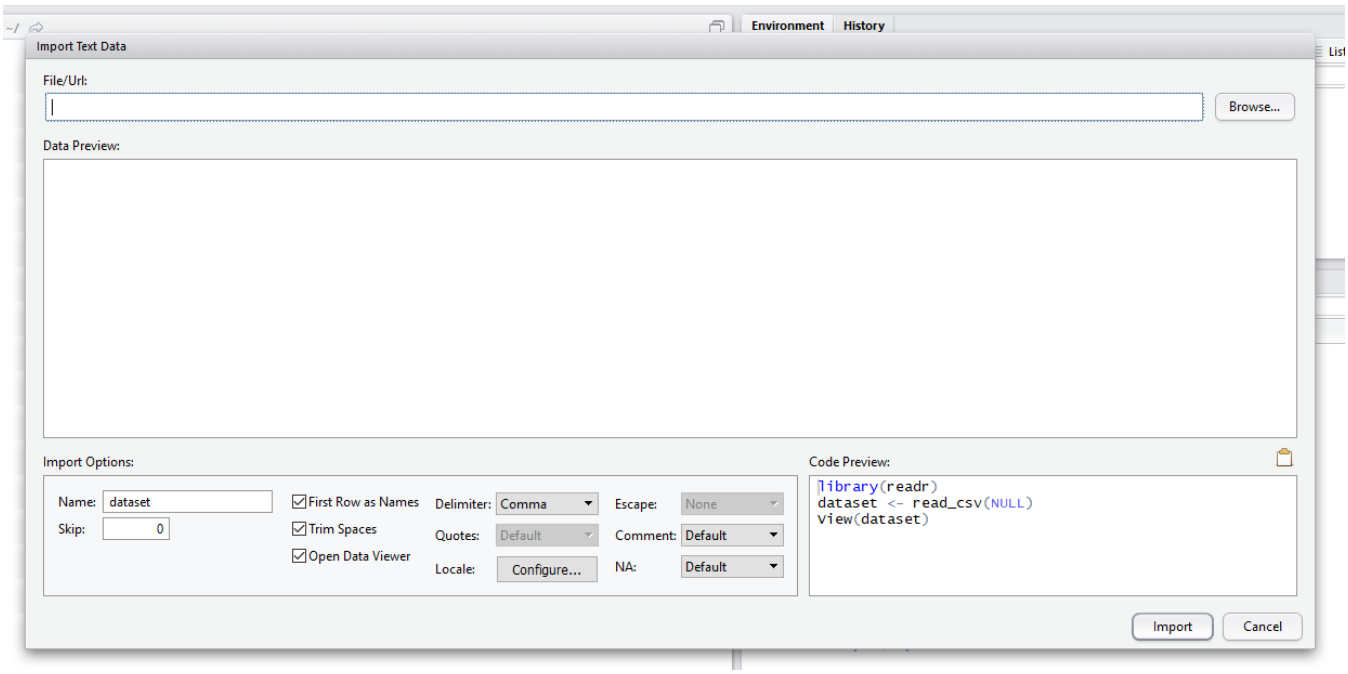
Importing data into R

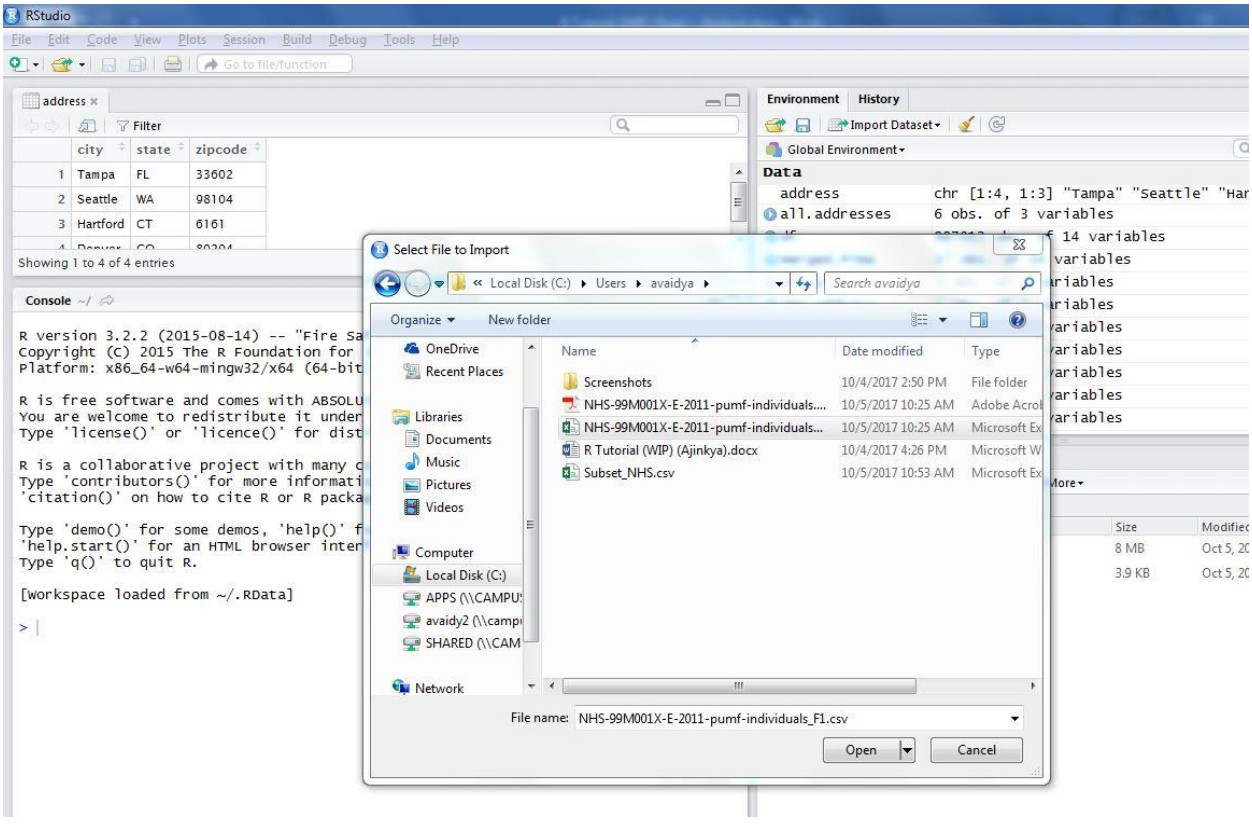
There are 2 ways to import data in R, through the GUI and using a command.

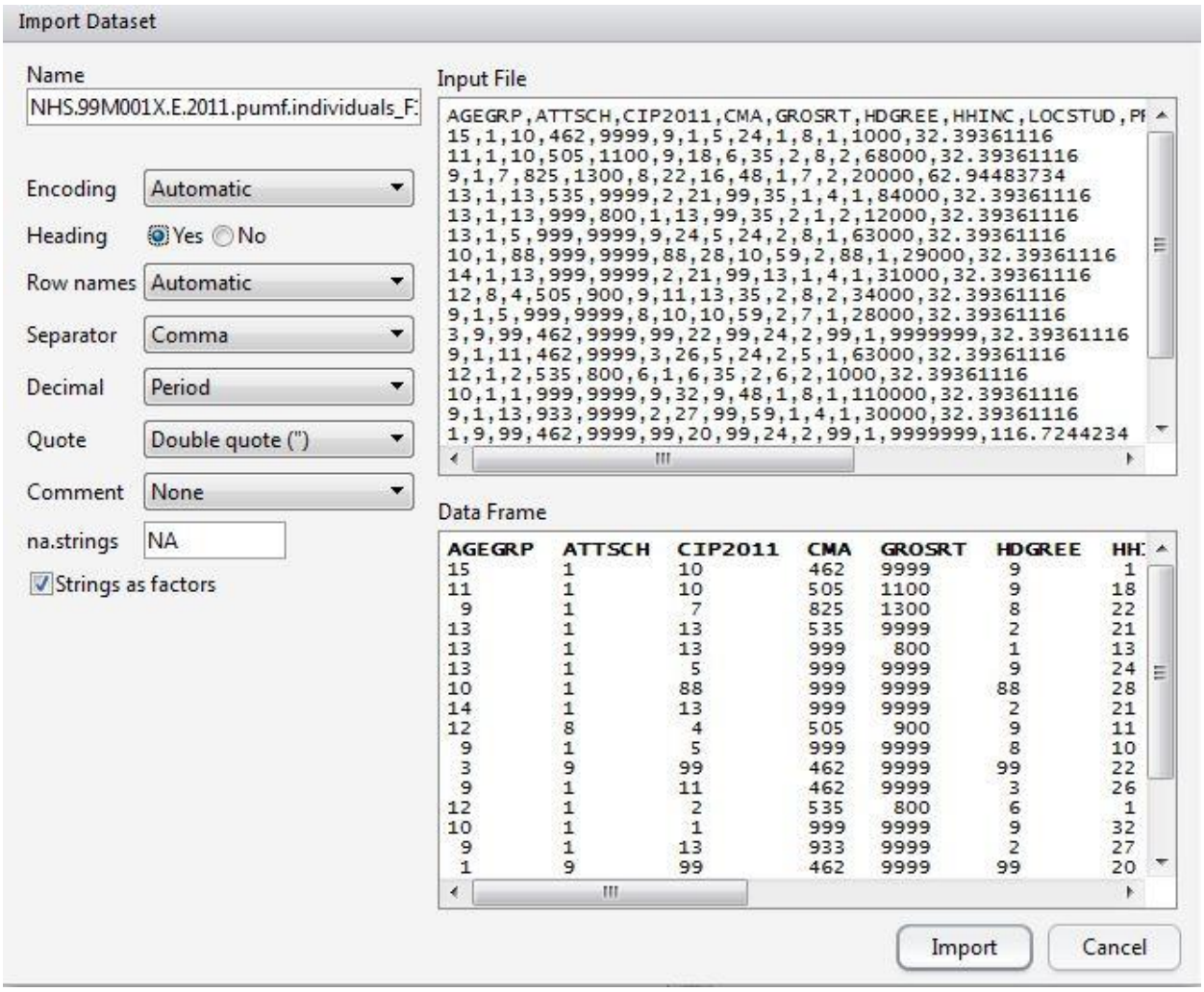
GUI Import

Click on File, then “Import Dataset”. Select the type of data you want to import. Choose csv. A dialog box will appear.

Browse to the csv dataset. Select it and click on “Open” and then “Import”.


Environment
History
Import Text Data
File/Url:
Browse...
Data Preview:
Import Options:
Name: dataset
Skip: 0
First Row as Names
Trim Spaces
Open Data Viewer
Delimiter: Comma
Quotes: Default
Locale: Configure...
Escape: None
Comment: Default
NA: Default
Code Preview:
library(readr)
dataset <- read_csv(NULL)
View(dataset)
Import
Cancel

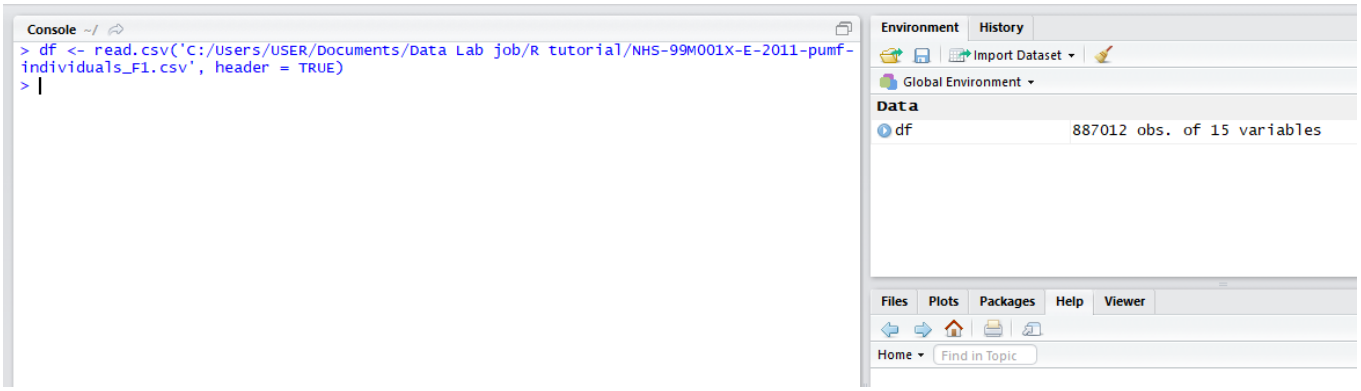

RStudio
File Edit Code View Plots Session Build Debug Tools Help
address
Filter
city state zipcode
1 Tampa FL 33602
2 Seattle WA 98104
3 Hartford CT 6161
Showing 1 to 4 of 4 entries
Console
R version 3.2.2 (2015-08-14) -- "Fire Sa
Copyright (C) 2015 The R Foundation for
Platform: x86_64-w64-mingw32/x64 (64-bit
R is free software and comes with ABSOLU
You are welcome to redistribute it under
Type 'license()' or 'licence()' for dist
R is a collaborative project with many c
Type 'contributors()' for more informati
'citation()' on how to cite R or R packa
Type 'demo()' for some demos, 'help()' f
'help.start()' for an HTML browser inter
Type 'q()' to quit R.
[workspace loaded from ~/.RData]
>
Environment
History
Import Dataset
Global Environment
Data
address chr [1:4, 1:3] "Tampa" "Seattle" "Har
all.addresses 6 obs. of 3 variables
Select File to Import
Local Disk (C:) Users avaidya
Search avaidya
Organize
New folder
OneDrive
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
APPS (\\CAMPU
avaidy2 (\\camp
SHARED (\\CAM
Network
Name
Date modified
Type
Screenshots 10/4/2017 2:50 PM File folder
NHS-99M001X-E-2011-pumf-individuals.... 10/5/2017 10:25 AM Adobe Acro
NHS-99M001X-E-2011-pumf-individuals... 10/5/2017 10:25 AM Microsoft Ex
R Tutorial (WIP) (Ajinkya).docx 10/4/2017 4:26 PM Microsoft W
Subset_NHS.csv 10/5/2017 10:53 AM Microsoft Ex
File name: NHS-99M001X-E-2011-pumf-individuals_F1.csv
Open
Cancel
Size
8 MB
3.9 KB



Import using command

Type: df <- read.csv('P:/r_tutorial/NHS-99M001X-E-2011-pumf-individuals_F1.csv', header = TRUE)

the data is stored in data frame df



NOTE: make sure to keep forward slashes (/) in the pathname.

The CSV is imported as a **df – data frame**

- A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.
- Header is TRUE if file contains the names of the variables as its first line

Basics of R

The R environment can be used to compute calculations and assign variables. As a new R-user, you might want to practice these simple exercises by typing them into the console window:

```
> 2+2
[1] 4
> sqrt(64)
[1] 8
> b<-"Hello"
> b
[1] "Hello"
> vector<-c(1,2,3,4)
> vector
[1] 1 2 3 4
```

There are different data types in R. These data types can be numeric, integer, logical/boolean, character/string, vector, matrix, array, list, data-frame. It is useful to know the data type in order to know what functions can be performed on the object.

To determine the type of data, you can use the **class()**, **mode()** or **typeof()** functions. The following commands create different variables and check their type using the **class()** function.

It is possible to convert from one data type to another by using functions such as **as.integer()**, **as.vector()**, **as.matrix()** etc.

```
> numbers <- pi
> class(numbers)
[1] "numeric"
> integers <- as.integer(numbers)
> class(integers)
[1] "integer"
> vector <- as.vector(integers)
> class(vector)
[1] "integer"
> matrix <- as.matrix(vector)
> class(matrix)
[1] "matrix"
```

Type typeof(df). You will see the dataset we are using is type “data frame”.

Managing the R environment

Introduction

The R environment refers to your current workspace. You can think of it like the memory of all assigned values, which you can draw on as you work. For example, if 33 is assigned to black ("black<-33"), then "black" now exists within the R environment with a value of 33, and if the environment is cleared, then "black" will no longer have that value.

Packages in R

R is a bare bones program. Its utility comes from the 3500+ packages available for installation. You may think of these packages as similar to extensions or apps, which some other programs use, like Google, Chrome or Apple products. Each package serves a purpose and has specific commands you can use. In short, Packages are user-created programs which can be used to run a specific task or set of tasks.

Using packages is a twostep process. First you must install the package you want, and then you must load it into R.

Installing packages

install.package("nameofpackage") # installs a package

library("nameofpackage") # loads an already installed package, you can use a package only after loading it

installed.packages() # lists installed packages (not necessarily loaded)

Installing R packages for this workshop

Install the following packages:

- install.packages("plyr")
- install.packages("dplyr")
- install.packages("ggplot2")

You can install multiple packages at once with a single call by placing the names of the packages in a character vestor.

Type: install.packages(c("tidyr", "devtools", "dplyr", "ggplot2"))

Working directory

getwd() # returns current working directory

setwd("/path/to/new/working/directory") # changes current working directory

For this tutorial set the working directory to the folder created previosly for saving the dataset – 'r_tutorial'.

To do this: type:

setwd("P://r_tutorial/")

Common Operators in R

?	One of the most important operators in R because this is the equivalent of a help button. When used in combination with a function, such as "?read.csv", a webpage on the CRAN will open explaining what the function does, what its arguments are, and what are some associated functions.
<-	This operator assigns the value on the right side to that of the left, so "five<-5" means that the numerical value 5 is assigned to the character value of five. Therefore, inputting "five+5" would result in 10.
$	To access one variable in a dataset, use the dollar sign "$". For example, $vote1 returns the vote1 variable (the vote1 column).
""	All information put between quotation marks must be literal because R will search for those exact characters.

#	You might find this operator in at the start of a command line in CRAN files or any descriptions of R functions. This tells R not to run that line and to move on to the next, so it is a way to provide line-by-line commentary without interrupting R's ability to run the script.
	`> five<-5`
	`> #we have assigned 5 to five`
	`> six<-6`
	`> five + six`
	`[1] 11`

c()	As seen in the vector example on the previous page, **c()**, which stands for concatenate, will combine its arguments, both numbers and words, into a vector.
+	Another form of concatenation used typically in writing long scripts which span multiple lines, so rather than R interpreting each new line as a new command, it reads it all as one single command.
==	A boolean (meaning there are two possible values, true or false) operator which ensures that the values on the left side are the exact same as the values on the right, i.e., 5==5 would be true and 5==9 would be false.

Functions in R to view data

str(df) # shows the internal structure of the data frame

```
> str(df)
'data.frame':   887012 obs. of  14 variables:
 $ AGEGRP : int  15 11 9 13 13 13 10 14 12 9 ...
 $ ATTSCH : int  1 1 1 1 1 1 1 1 8 1 ...
 $ CIP2011: int  10 10 7 13 13 5 88 13 4 5 ...
 $ CMA    : int  462 505 825 535 999 999 999 999 505 999 ...
 $ GROSRT : int  9999 1100 1300 9999 800 9999 9999 9999 900 9999 ...
 $ HDGREE : int  9 9 8 2 1 9 88 2 9 8 ...
 $ HHINC  : int  1 18 22 21 13 24 28 21 11 10 ...
 $ LOCSTUD: int  5 6 16 99 99 5 10 99 13 10 ...
 $ PR     : int  24 35 48 35 35 24 59 13 35 59 ...
 $ SEX    : int  1 2 1 1 2 2 2 1 2 2 ...
 $ SSGRAD : int  8 8 7 4 1 8 88 4 8 7 ...
 $ TENUR  : int  1 2 2 1 2 1 1 1 2 1 ...
 $ TOTINC : int  1000 68000 20000 84000 12000 63000 29000 31000 34000 28000 ...
 $ WEIGHT : num  32.4 32.4 62.9 32.4 32.4 ...
>
```

head(df) # tells you the first six lines of a vector, matrix, table, data frame, or function

```
> head(df)
  AGEGRP ATTSCH CIP2011 CMA GROSRT HDGREE HHINC LOCSTUD PR SEX SSGRAD TENUR TOTINC
1     15      1      10 462   9999      9     1       5 24   1      8     1   1000
2     11      1      10 505   1100      9    18       6 35   2      8     2  68000
3      9      1       7 825   1300      8    22      16 48   1      7     2  20000
4     13      1      13 535   9999      2    21      99 35   1      4     1  84000
5     13      1      13 999    800      1    13      99 35   2      1     2  12000
6     13      1       5 999   9999      9    24       5 24   2      8     1  63000
    WEIGHT
1 32.39361
2 32.39361
3 62.94484
4 32.39361
5 32.39361
6 32.39361
```

Some other similar functions which can be used are as follows:

tail(df) # tells you the last six lines of a vector, matrix, table, data frame, or function

summary(df) # shows the statistical summary

ls() # lists all of the objects available in the workspace, i.e., all the data and variables you have defined

ls.str() # combines ls and str to lists all objects with details about data type and content

class() # tells you the data type, i.e., vector, matrix, character, etc.

names() # can either change the name of an object or tells you all of the defined names in the object, i.e., all of the column titles in an excel file

object.size() # tells you how much memory is taken up on your computer by the object

dim() # tells you the dimensions of the object

length() # tells you the length (number) of the vectors and factors in the object

ncol() # tells you the number of columns

nrow() # tells you the number of rows

Accessing columns in a data frame

Let's say we want to find the mean of the input column "TOTINC" in R. We do that as shown below:

```
> mean(df$TOTINC)
[1] 1756655
>
```

Hence, to access TOTINC, we need to use df$TOTINC. However, to access the columns directly in R, you can use the function attach, after which you can refer to the columns directly as shown below:

```
> mean(df$TOTINC)
[1] 1756655
> attach(df)
> mean(TOTINC)
[1] 1756655
>
```

You can and should use detach(df) to detach the data frame once you don't need to directly access the variables.

Note: When working with multiple data frames, DO NOT use attach as it might cause a lot of confusion, rather use WITH() function, the details of which are beyond the scope of this workshop.

Subsetting Data

Selecting (Keeping) Variables and Selecting Observations:

We wish to select the variables AGEGRP, TOTINC & WEIGHT. We create a new vector called myvars and then assign a subset data frame using myvars.

Type:

myvars <- c("AGEGRP", "TOTINC", "WEIGHT")

new_data_frame <- df[myvars]

=

```
> str(df)
'data.frame':   887012 obs. of  14 variables:
 $ AGEGRP : int  15 11 9 13 13 13 10 14 12 9 ...
 $ ATTSCH : int  1 1 1 1 1 1 1 1 8 1 ...
 $ CIP2011: int  10 10 7 13 13 5 88 13 4 5 ...
 $ CMA    : int  462 505 825 535 999 999 999 999 505 999 ...
 $ GROSRT : int  9999 1100 1300 9999 800 9999 9999 9999 900 9999 ...
 $ HDGREE : int  9 9 8 2 1 9 88 2 9 8 ...
 $ HHINC  : int  1 18 22 21 13 24 28 21 11 10 ...
 $ LOCSTUD: int  5 6 16 99 99 5 10 99 13 10 ...
 $ PR     : int  24 35 48 35 35 24 59 13 35 59 ...
 $ SEX    : int  1 2 1 1 2 2 2 1 2 2 ...
 $ SSGRAD : int  8 8 7 4 1 8 88 4 8 7 ...
 $ TENUR  : int  1 2 2 1 2 1 1 1 2 1 ...
 $ TOTINC : int  1000 68000 20000 84000 12000 63000 29000 31000 34000 28000 ...
 $ WEIGHT : num  32.4 32.4 62.9 32.4 32.4 ...
> myvars <- c("AGEGRP", "TOTINC", "WEIGHT")
> new_data_frame <- df[myvars]
> 
```

There is another method to subset variables based on index. Suppose we wish to select 1st and 5th thru 10th variables.

Type:

```
> new_data_frame <- df[c(1,5:10)]
> 
```

Selecting the first 5 observations.

Type:

```
> new_data_frame <- df[1:5,]
> 
```

Selecting first 5 observations and first 3 variables.

Type:

```
> new_data_subset <- df[1:5, 1:3]
> 
```

Finally, let's try creating a new data frame with FEMALES (SEX = 1) earning more than 35000

sub-setting based on values of variables

Type:

```
> new_data_frame <- df[ which(df$SEX=='1'  & df$TOTINC > 35000), ]
> 
```

External resource for subsetting in R: http://www.statmethods.net/management/subset.html

Data Reshaping

- It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from the format in which we received it.
- Data Reshaping in R is about changing the way data is organized into rows and columns.
- Joining columns and rows in a data frame is achieved by 2 functions:
    - rbind
        - 1:3Combines vector, matrix or data frame *by rows*

```
> nhs_subset2<-df[11:21,1:3]
> nhs_subset1<-df[1:10,1:3]
>
>
>
>
> nhs_subset3<-rbind(nhs_subset1,nhs_subset2)
>
>
> str(nhs_subset3)
'data.frame':   21 obs. of  3 variables:
 $ AGEGRP : int  15 11 9 13 13 13 10 14 12 9 ...
 $ ATTSCH : int  1 1 1 1 1 1 1 1 8 1 ...
 $ CIP2011: int  10 10 7 13 13 5 88 13 4 5 ...
```

    - cbind
        - Combines vector, matrix or data frame *by columns*

```
> nhs_subset4 <- df[1:20, 1:3]
> nhs_subset5 <- df[1:20, 4:6]
> nhs_subset6 <- cbind(nhs_subset4, nhs_subset5)
> head(nhs_subset6)
  AGEGRP ATTSCH CIP2011 CMA GROSRT HDGREE
1     15      1      10 462   9999      9
2     11      1      10 505   1100      9
3      9      1       7 825   1300      8
4     13      1      13 535   9999      2
5     13      1      13 999    800      1
6     13      1       5 999   9999      9
> 
```

Transpose

To convert the rows into columns and columns into rows (essentially flipping the data frame) we can use the t() function in R.

```
> nhs_subset1
   AGEGRP ATTSCH CIP2011
1      15      1      10
2      11      1      10
3       9      1       7
4      13      1      13
5      13      1      13
6      13      1       5
7      10      1      88
8      14      1      13
9      12      8       4
10      9      1       5
>
>
> nhs_subset1.transpose<- as.data.frame(t(nhs_subset1))
>
>
> nhs_subset1.transpose
         1  2 3  4  5  6  7  8  9 10
AGEGRP  15 11 9 13 13 13 10 14 12  9
ATTSCH   1  1 1  1  1  1  1  1  8  1
CIP2011 10 10 7 13 13  5 88 13  4  5
```

The t function is used to create the transpose, after which as.data.frame is used to convert the result into a data frame.

Tapply and Aggregate functions in R

tapply() function

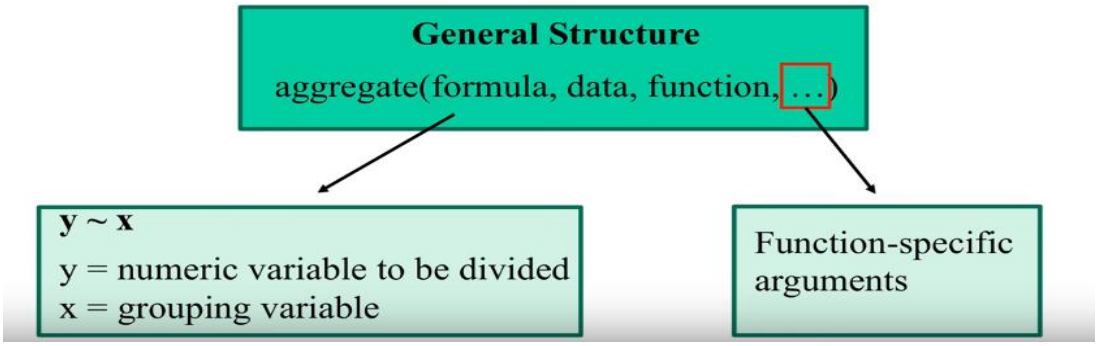
tapply(TOTINC, SEX, mean) # finds the mean of Total_Income based on Sex.

```
> tapply(TOTINC, SEX, mean)
      1       2
1675920 1840389
```

As per the data documentation, 1 is female and 2 is male, so we can see their respective means of Total_Income.

aggregate() function

Aggregate is a function similar to tapply but it is much more flexible compared to tapply. A general structure of aggregate formula is given below:



By running aggregate() with the same parameters as tapply() in previous example, we get

```
> aggregate(TOTINC~SEX, df, mean)
  SEX  TOTINC
1   1 1675920
2   2 1840389
>
```

Supposed we want to subset the data on more than one variable, say SEX and PR, we use:

```
> aggregate(TOTINC~SEX+PR, df, mean)
   SEX PR  TOTINC
1    1 10 1545185
2    2 10 1658997
3    1 11 1674461
4    2 11 1864334
5    1 12 1513492
6    2 12 1700714
7    1 13 1572150
8    2 13 1694964
9    1 24 1613734
10   2 24 1745807
11   1 35 1677769
12   2 35 1870890
13   1 46 1914047
14   2 46 2043743
15   1 47 1952193
16   2 47 2082047
17   1 48 1905374
18   2 48 2026555
19   1 59 1517707
20   2 59 1710347
21   1 60 2515043
22   2 60 2586150
>
```

You can use any of the standard functions instead of mean in the examples above such as sum, sd (standard deviation), etc.

Data Visualization using ggplot2

Why gglplot2?

- mature and complete graphics system
- consistent underlying grammar of graphics
- plot specification at a high level of abstraction
- very flexible
- includes a theme system for polishing plot appearances
- many users, active mailing list

What is ggplot2?

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same few components: a dataset, a set of geoms (which are visual marks that represent data points), and a coordinate system.

General syntax

- ggplot(data ='data_frame' ,aes (x='column name' , y='column name'))
    - aes -> aesthetic mappings describing how variables in the data are mapped.
        - x -> variable(column name) that must be mapped to X-axis.

        - y -> variable(column name) that must be mapped to Y-axis.

- This creates a plot that you must finish by adding more layers.
- Various plots can be created using ggplot2, some of the popular ones include:
    - geom_area()
    - geom_density()
    - geom_dotplot()
    - geom_freqpoly()
    - geom_histogram()
    - geom_bar()
    - geom_hex()
    - geom_boxplot()
    - geom_dotplot()
    - geom_map()
    - geom_contour()
    - Many more exist…

- The simplest way to create a comprehensive plot using ggplot is to keep adding layers rather than combining everything into one.

Example

Let look at an example of this by creating a bar graph of gross rent paid by people across different provinces.

Some background regarding the dataset:

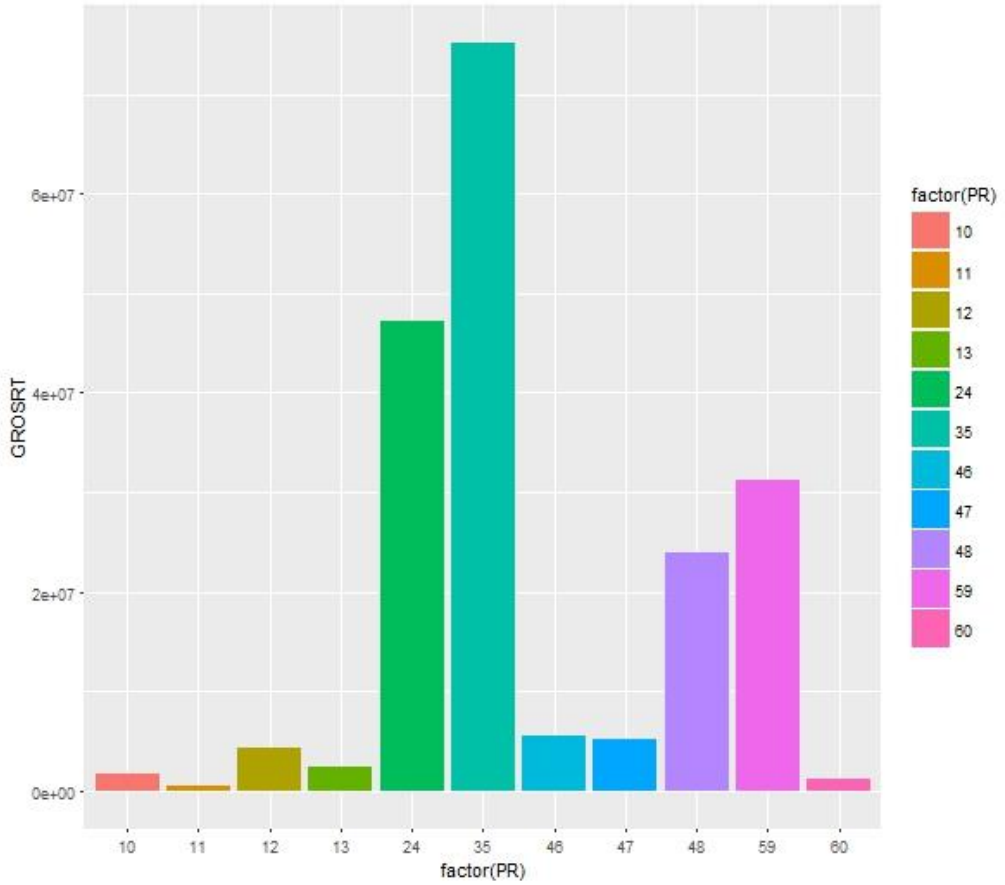
- Gross rent paid per month is stored in the column – GROSRT
    - For the sake of accuracy, we should exclude the values of 9999 & 8888 as both mean that the person either lives in his own house without paying any rent (e.g. a farmhouse) or the data is not available/missing.
- The current residence of the people surveyed is stored in the column –PR
    - The values in the column are numbers, each representing a province:
        - 10- Newfoundland and Labrador
        - 11- Prince Edward Island
        - 12- Nova Scotia
        - 13- New Brunswick
        - 24- Quebec
        - 35- Ontario
        - 46- Manitoba
        - 47- Saskatchewan
        - 48- Alberta
        - 59- British Columbia
        - 60- Northern Canada

Let's start with the basic plotting:

```
> ggplot(NHS.99M001X.E.2011.pumf.individuals_F1[!(NHS.99M001X.E.2011.pumf.indivi
duals_F1$GROSRT==9999 | NHS.99M001X.E.2011.pumf.individuals_F1$GROSRT==8888),],a
es(x=factor(PR),y=GROSRT,fill = factor(PR))) + geom_bar(stat = 'identity')
```

- As we highlighted earlier, we have excluded all the entries where the gross rent is either 9999 or 8888.
- We want the province on the X axis and the rent on the Y axis, hence:
  - aes(x=factor(PR),y=GROSRT)
    - factor is used to keep the original discrete values rather than letting ggplot use an auto scale making the X axis continuous.
- We want the color of each bar to different based on the province, hence:
  - fill = factor(PR)
- Now we want a bar chart, where instead of counting the number of values which geom_bar does by default, we want to leave the values as they are, therefore:
  - geom_bar(stat='identity')
- Remember, each layer can be added by concatenating it with a '+'

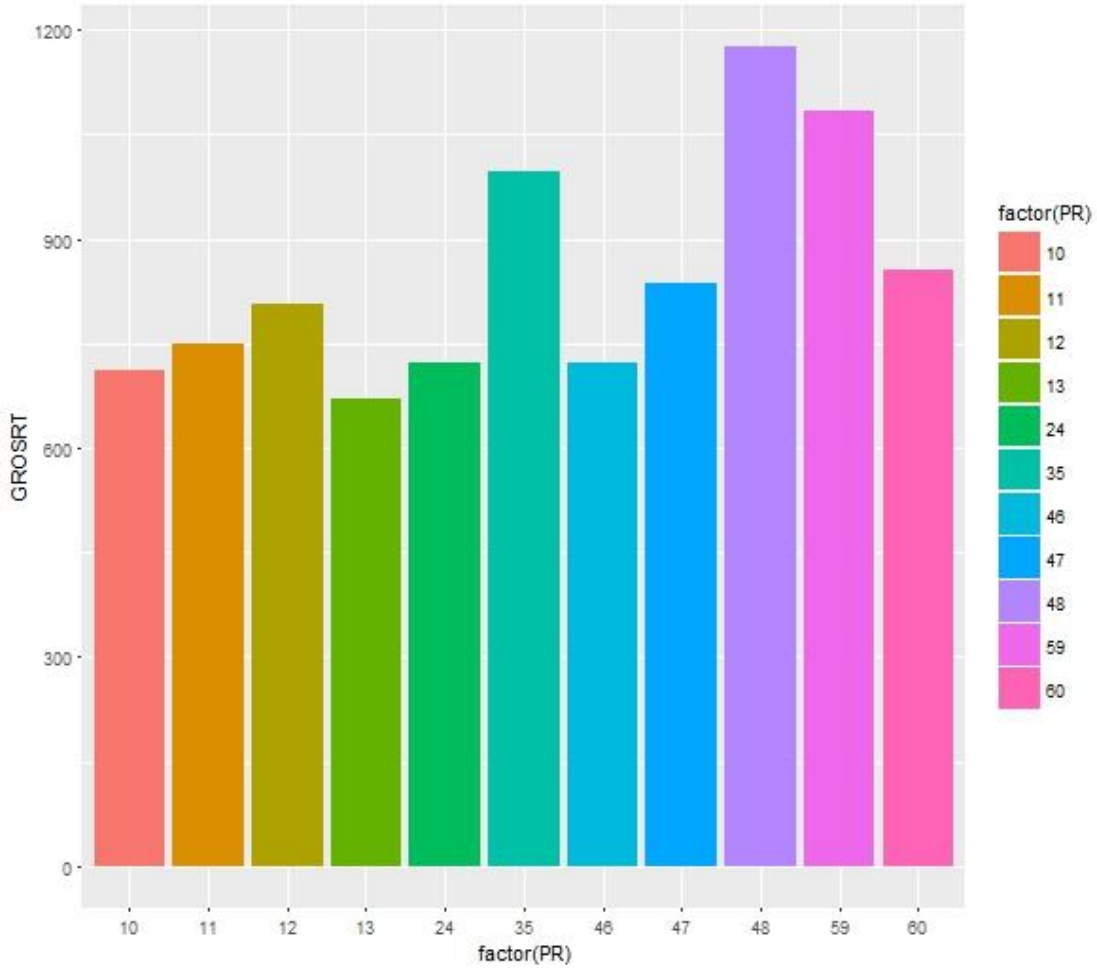
So, what does this give?



- Now, as you can see, ggplot by default has summed all the values in the GROSRT column, we don't want that. This just illustrates the total sum of rent of all the people surveyed province wise.
- We want to calculate the average rent paid by people province wise, hence we need to add a statistical layer that will average the rent province wise. Hence:

```
>
> ggplot(NHS.99M001X.E.2011.pumf.individuals_F1[!(NHS.99M001X.E.2011.pumf.indivi
duals_F1$GROSRT==9999 | NHS.99M001X.E.2011.pumf.individuals_F1$GROSRT==8888),],a
es(x=factor(PR),y=GROSRT,fill = factor(PR))) + stat_summary(fun.y = "mean",geom=
"bar")
>
```

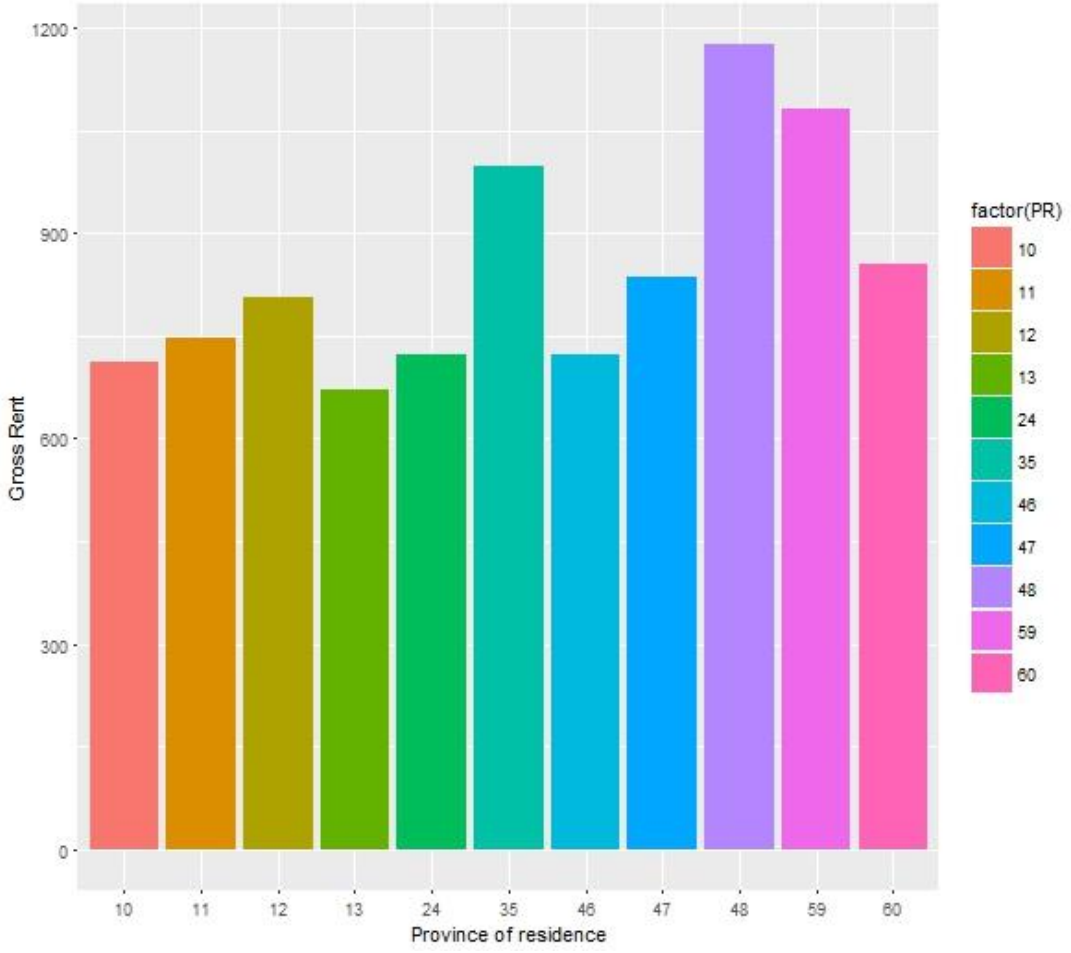
- So, we have just added a layer of stats **(stat_summary**) telling ggplot to apply the function **'mean'** to all the values on the Y axis based on the province. Notice, how we have combined the geom_bar previously added as a layer with stats_summary as a parameter.
- The result:



- Now, we have got the graph we wanted but it's less intelligible because the X and Y axis labels are just column names which are not understandable to someone looking at the graph for the first time. So, let's add labels to the graph.

```
>
> ggplot(NHS.99M001X.E.2011.pumf.individuals_F1[!(NHS.99M001X.E.2011.pumf.indivi
duals_F1$GROSRT==9999 | NHS.99M001X.E.2011.pumf.individuals_F1$GROSRT==8888),],a
es(x=factor(PR),y=GROSRT,fill = factor(PR))) + stat_summary(fun.y = "mean",geom=
"bar") + xlab("Province of residence") + ylab("Gross Rent")
>
```

The result:



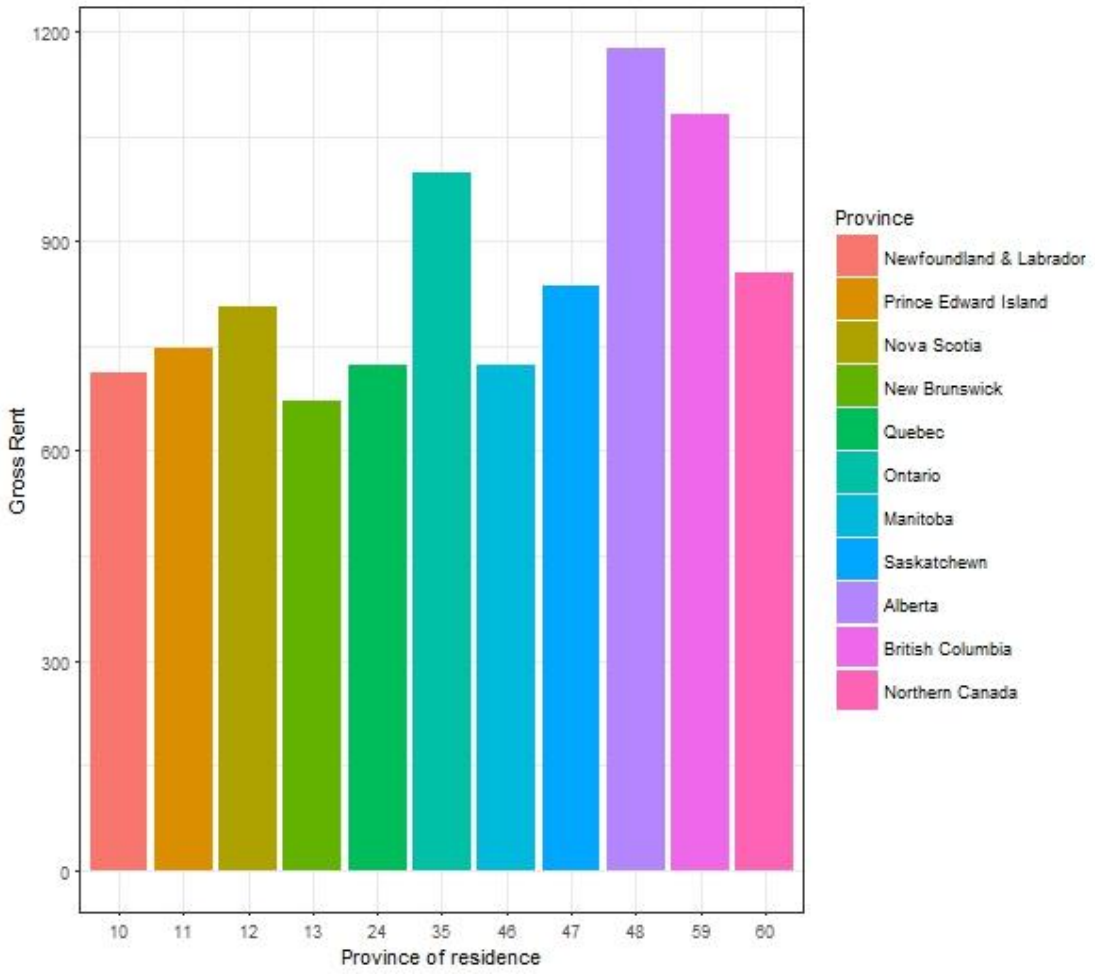
- This is now readable, except we don't really know what the numbers on the X axis stand for. We know, as previously stated, that each number is a province. So, let's add that to the graph, while making the background prettier.

```
>
> ggplot(NHS.99M001X.E.2011.pumf.individuals_F1[!(NHS.99M001X.E.2011.pumf.indivi
duals_F1$GROSRT==9999 | NHS.99M001X.E.2011.pumf.individuals_F1$GROSRT==8888),],a
es(x=factor(PR),y=GROSRT,fill = factor(PR))) + stat_summary(fun.y = "mean",geom=
"bar") + theme_bw() + xlab("Province of residence") + ylab("Gross Rent") + scale
_fill_discrete(name="Province",labels=c("Newfoundland & Labrador","Prince Edward
 Island","Nova Scotia","New Brunswick","Quebec","Ontario","Manitoba","Saskatchew
n","Alberta","British Columbia","Northern Canada"))
>
>
```

  - To name the legend and individual bars on the X axis we use – scale_fill_discrete()
    - name = name that should be given to the legend.

- labels = names given to each individual bar.

Final graph:



This example shows you how to create a bar graph. Similarly, you can create many other types of data visualizations.