Guillermo Romera Rodriguez | Spring 2018 | DA5020

STEAMSPY'S DATA COLLECTION STORE AND RETRIEVAL SCRIPT

Preface

The initial idea for this project was to create an automated script that allowed collection of data through the SteamSpy's propietary API https://steamspy.com/api.php and made it open source for anyone interested (especially for the students at Northeastern University in the GSND department). Since the change of terms in Steam Privacy Policy unfortunately the API is not able to retrieve the data anymore and the half the project was rendered not functional. Because of this I also had to work with a back up copy of the JSON file I retrieved from the API forcing me also to change a few things from its original concept

SteamSpy Home Page: http://steamspy.com/
Links related to the Privacy Policy change on Steam:
https://www.pcgamesn.com/game-devs-react-steam-privacy-change
https://www.digitaltrends.com/gaming/privacy-changes-steam-spy-useless/
http://variety.com/2018/gaming/news/steam-privacy-update-gaming-libraries-1202750358/
https://www.eurogamer.net/articles/2018-04-11-why-steam-spy-has-to-close-from-the-creator-himself
https://www.polygon.com/2018/4/12/17229752/steam-spy-charts-new-privacy-rules-valve

Because of this I was forced myself to extend my project further than anticipated and divide into two part

- Part 1 : Will be the part that doesn't work because of the shutdown fo the API, it will still have the same explanations as it had at the beginning since it is crucial to understand the project. The introduction of snip shots help to further understand how everything was intended to work before this setback
- Part 2 : It is the working part of the project, some departments have been extended intentionally to prove proper coding skills and knowledge although some lines have been changed due to the issue mentioned above

Note: Most of the code in part 1 still remains functional due to the API still being operative but it doesn't retrieve data, I have code it so if it doesn't retrieve the data the code will move onto the next chunk, I do mention this because as of 4/19/2018 the API is still online but by the time this is read that might change.

PART 1 - Introduction and non-functional code

Understanding Steamspy's webpage

Steamspy webpage focuses on retrieving the data from the videogame software application called Steam, and it does so with their propietary algorithms and tools since Steam's company (Valve) do not release their analytics and data to the public. This has become more and more valuable over time since performing some analysis on that data and building predictive models over it can shed some light of the market trends for videogames.

The webpage displays the data in a visually appealing manner with well made graphs and properly structured data. See *Figure*1.1 as an example.

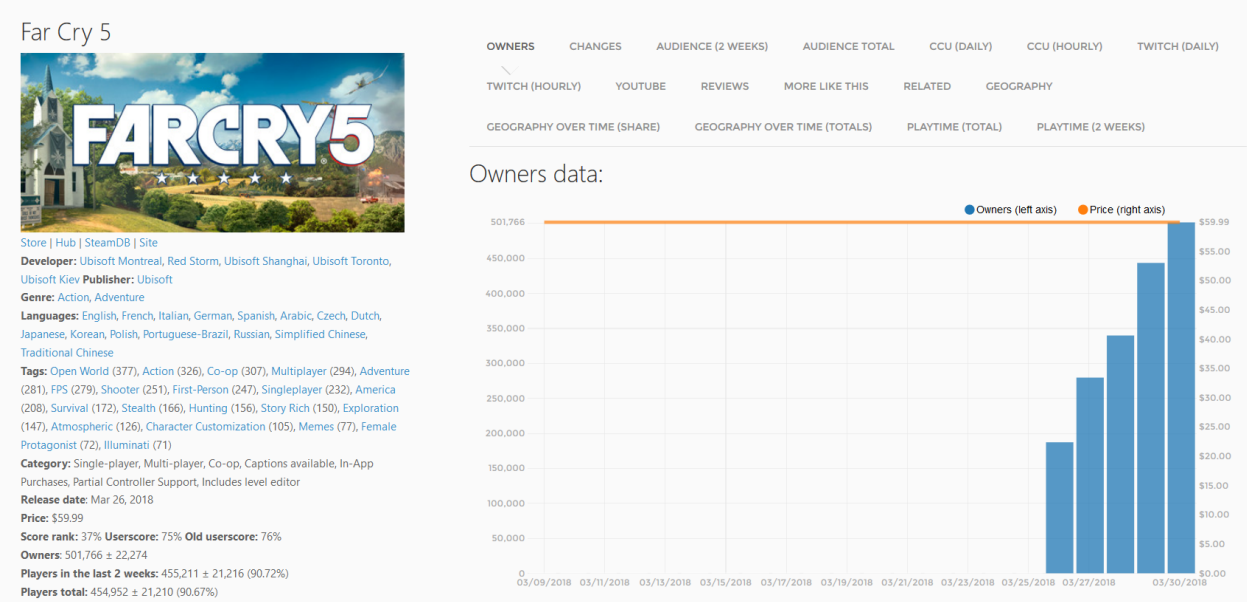
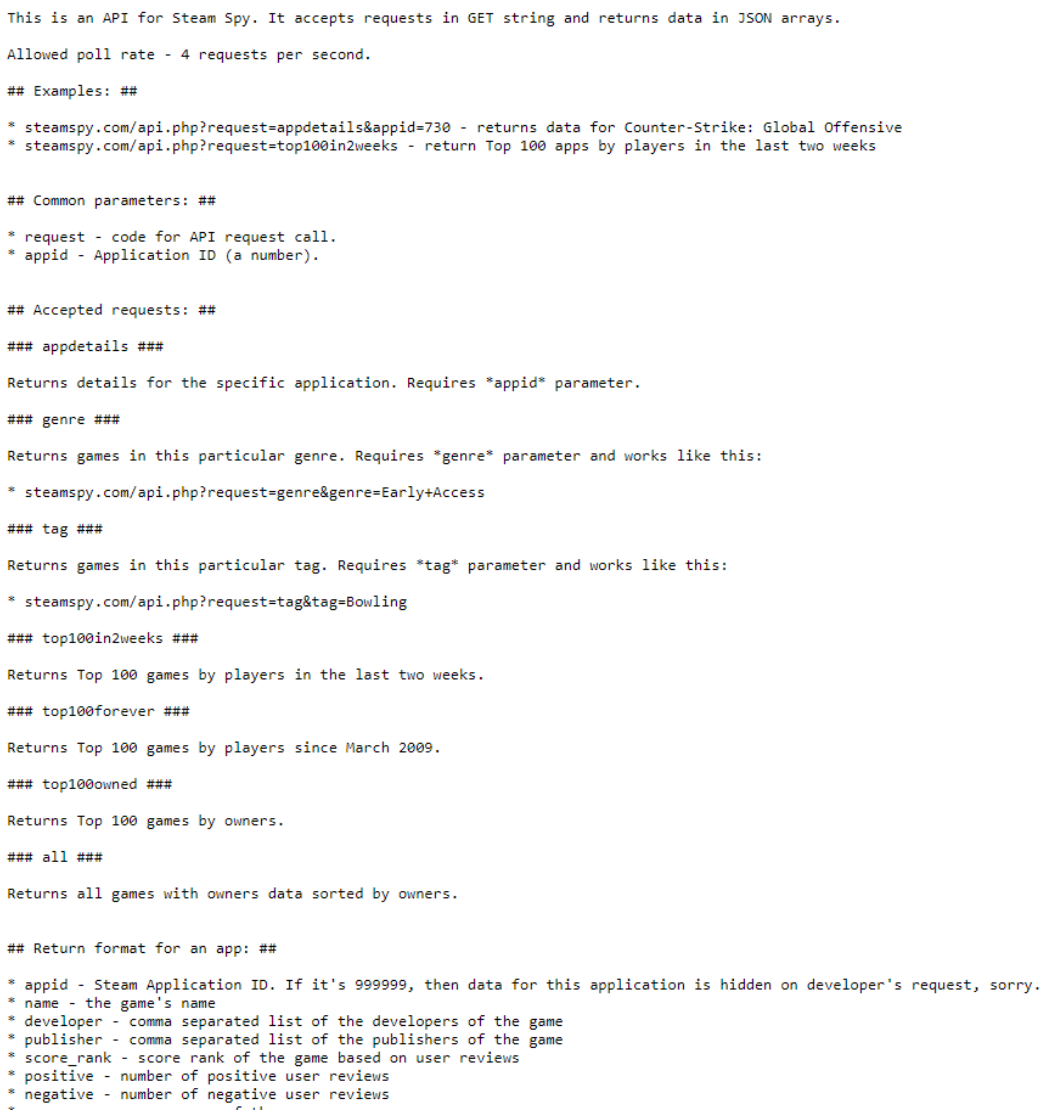


*Figure*1.1

The problem with that is that for anyone to see different data we need to navigate the webpage from one game to another, adn that besides being time consuming is pointless if we want to do some analysis over the data.

Fortunately Steamspy has their own propietary API which has been made with ease of use in mind so anyone can collect the data and start doing some analysis over it.
The instructions for the API as seen in *Figure*1.2 are quite simple and have a few options to make the collection of data easier.



This is an API for Steam Spy. It accepts requests in GET string and returns data in JSON arrays.

Allowed poll rate - 4 requests per second.

Examples:

* steamspy.com/api.php?request=appdetails&appid=730 - returns data for Counter-Strike: Global Offensive
* steamspy.com/api.php?request=top100in2weeks - return Top 100 apps by players in the last two weeks

Common parameters:

* request - code for API request call.
* appid - Application ID (a number).

Accepted requests:

appdetails

Returns details for the specific application. Requires *appid* parameter.

genre

Returns games in this particular genre. Requires *genre* parameter and works like this:

* steamspy.com/api.php?request=genre&genre=Early+Access

tag

Returns games in this particular tag. Requires *tag* parameter and works like this:

* steamspy.com/api.php?request=tag&tag=Bowling

top100in2weeks

Returns Top 100 games by players in the last two weeks.

top100forever

Returns Top 100 games by players since March 2009.

top100owned

Returns Top 100 games by owners.

all

Returns all games with owners data sorted by owners.

Return format for an app:

* appid - Steam Application ID. If it's 999999, then data for this application is hidden on developer's request, sorry.
* name - the game's name
* developer - comma separated list of the developers of the game
* publisher - comma separated list of the publishers of the game
* score_rank - score rank of the game based on user reviews
* positive - number of positive user reviews
* negative - number of negative user reviews

*Figure*1.2
But as it can be seem there are a few issues that need to be addressed and they will in this script.

- First, the API returns a JSON file, and while that is useful in most situation, even to store in a NOSQL data abse as I have seen in this course it might not be the most suitable format to work with R
- The API accepts certain tags to download the data at will, but what if we want to download the whole chunk of data, store it in a relational SQL data base and the retrieve it?

These issues will be addressed so the use of the API is minimized and the ease of collecting the data is improved so the user doesn't need to know the intricancies of the code to collect the data and start working with it.

First Collection of Data from Steamspy's API

As I mentioned before the first issue of retrieving the data from the webpage is the fact that the data comes in a JSON format and we need something to pull that data from from the webapge throught the API

Fortunately someone has already thought about that before me and create a package for it.

The package is called *jsonlite* and combined with the package *httr* will alow me to uyse the API in the Rstudio environment.

What follows next is a demostration of how the Steamspy's API is used in combination with the *jssonlite* and *httr* packages in the Rstudio to retrieve the data.

```
library(httr)
library(jsonlite)

  # The code below saves the API url and the exact path
  # where the information will be taken from

url  <- "https://steamspy.com/api.php"
path <- "steamspy.com/api.php?request=appdetails&appid=730"

raw_result <- GET(url = path)

  # This makes sure that the reques has been accepted
  # "200" means is accepted "400" means it has failed
raw_result$status_code
```

```
## [1] 200
```

```
  # The line below converts the raw data into readable content for R
raw_content <- rawToChar(raw_result$content)

  # A fail safe implemented after the Steam Privacy change to see
  # that the code runs but it doesn't collect data anymore

if (raw_content==""){
  print("Their API doesn't work anymore but the my code still works")
} else {
  r_content <- fromJSON(raw_content)

  # Creating a data frame out of the collected data

df <- do.call(what = "cbind",
                            args = lapply(r_content, as.data.frame))

head(df)
}
```

```
## [1] "Their API doesn't work anymore but the my code still works"
```

Now as we can see above there are a few issues with this.

- There are more columns that I need, the columns marked as tags are not needed, mostly because not every game has their tags pulled from Steam, hence for the sake of consitency I will get rid of those columns.
- There data within the data frame is highly unestructured, there are spaces and breaks when shouldn't be, so that will need some cleaning on my end
- The columns that I need and I want and not properly named, and in order to avoid doing it manually a few extra lines of code will be implemented

This fortunately is just a quick fix while setting up the parameters above

Let's apply those fixes down below

```
library(httr)
library(jsonlite)

  # The code below saves the API url and the exact path
  # where the information will be taken from

url  <- "https://steamspy.com/api.php"
path <- "steamspy.com/api.php?request=appdetails&appid=730"

raw_result <- GET(url = path)

  # This makes sure that the reques has been accepted
  # "200" means is accepted "400" means it has failed
raw_result$status_code
```

```
## [1] 200
```

```
  # The line below converts the raw data into readable content for R
raw_content <- rawToChar(raw_result$content)

  # A fail safe implemented after the Steam Privacy change to see
  # that the code runs but it doesn't collect data anymore
if (raw_content==""){
  print("Their API doesn't work anymore but the my code still works")
} else {

r_content <- fromJSON(raw_content)

df2<-data.frame()
tmp<- data.frame(r_content)
df2 <- rbind(df2, tmp[,1:20])
head(df2)

}
```

```
## [1] "Their API doesn't work anymore but the my code still works"
```

As we can see now the data has been properly extracted by deleting a line of code and adding a few others making the code more solid an fail proof

Of course that is collecting only one videogame, but what if I want to collect more? There are a few methods to do that.

Collecting data by "Brute Force" from the API

The first method is the collection of the data by brute force, that is using a hand made function to collect the data from the API.

As seen in *Figure*1.3 the webapage allows to collect the data by using the videogame's Steam Appid, which functions as a Primary key, one wuld say that Valve already thought about it when creating the Steam Software.

This method has two issues :

- In the webpage not every number is a Steam Appid, that means that for example steamspy.com/api.php?request=appdetails&appid=730 calls for the game Counter Strike, but maybe steamspy.com/api.php?request=appdetails&appid=340 might be empty, I haven't manually checked since this is just for explanation purposes.
- The API only allows for 4 request per second,as seen in *Figure*1.3 , and while that might seem enough, it gets time consuming over time.

```
This is an API for Steam Spy. It accepts requests in GET string and returns data in JSON arrays.

Allowed poll rate - 4 requests per second.

## Examples: ##

* steamspy.com/api.php?request=appdetails&appid=730 - returns data for Counter-Strike: Global Offensive
* steamspy.com/api.php?request=top100in2weeks - return Top 100 apps by players in the last two weeks


## Common parameters: ##

* request - code for API request call.
* appid - Application ID (a number).
```

*Figure*1.3
The function is as follows:

```
index<-c(725:732)
sci_df <- data.frame()
for (i in 1:7){ #This number never changes unles you want to make the loop different

  add.wb <- paste0("steamspy.com/api.php?request=appdetails&appid=",index[i])
  # the ID number for said video game
  # putting the video game and its ID together
  # path<-paste(add.wb,test, sep="")
  # calling the url
url  <- "https://steamspy.com/api.php"
raw_result <- GET(url = add.wb)
raw_result$status_code
raw_content <- rawToChar(raw_result$content)

if (raw_content==""){
  print("Their API doesn't work anymore but the my code still works")
} else {
this.content <- fromJSON(raw_content)

 if (this.content$score_rank=="") {
print(paste("no data",index[i]))}
else{

tmp<- data.frame(this.content)
sci_df0 <- rbind(sci_df, tmp[,1:20])
print(paste0("Data Collected , Game: ",sci_df0[1,2] ))

}
}
  # this line makes sure that no more than 4 pulls per second occur
```

```
Sys.sleep(0.30)

}
```

```
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
## [1] "Their API doesn't work anymore but the my code still works"
```

As it can be seen there is a fatal flaw with the function, it needs to go over every index number within the webpage to retrieve the data, which makes the process prohibitely long and tedious.

This process would the equivalent of scraping the webpage with a package such as Rvest but automatized, so the user would only have to set up the parameters, run it and then wait until the script collects all the possible data.

Collecting data by Steamspy's API with Tag =ALL

Now this is were the power of the API can be seen and harnessed; If when I use the API I put the tag ALL such as this : steamspy.com/api.php?request=appdetails&appid=$all the API will collect all data from theri webpage an put it together for anyone to perform any kind of analysis.

Nonetheless there are a few drawbacks:

- The API returns a JSON file, which as we can imagine at this point it will be of quite a bit of size, making it for example too big for some softwares to handle
- While there are tools to convert JSON to CSV on the internet it is better to do it manually in R studio and code it so it works no matter how big or small the JSON file is.
- The data obtained while quite organized and clean has still some imperfections that will need to be addressed no matter what, for that the script will implement some fail safes so it works everytime data is collected through the API

The code to retrieve all the data is as follows :

```
library(httr)
library(jsonlite)
library(ndjson)
```

```
## Warning: package 'ndjson' was built under R version 3.4.4
```

```
##
## Attaching package: 'ndjson'
```

```
## The following objects are masked from 'package:jsonlite':
##
##     flatten, stream_in, validate
```

```
sci_df2 <- data.frame()
tmp<-data.frame()
  add.wb <- paste0('steamspy.com/api.php?request=all' )
  # the ID number for said video game
  # putting the video game and its ID together
  # path<-paste(add.wb,test, sep="")
  # calling the url
```

```
url  <- "https://steamspy.com/api.php"
raw.result <- GET(url = add.wb)
raw.result$status_code
```

```
## [1] 200
```

```
this.raw.content <- rawToChar(raw.result$content)

validate(this.raw.content)
```

```
## [1] TRUE
```

```
if (this.raw.content==""){
  print("Their API doesn't work anymore but the my code still works")
} else {
this.content <- fromJSON(this.raw.content)
}
```

```
## [1] "Their API doesn't work anymore but the my code still works"
```

This collect two types of files,a JSON file and a R list, a big one.

The first idea that comes into mind is to directly upload the JSON file to a NOSQL data such as MongoDB, but there is a problem with the collected JSON file.

It seems as if the author (Sergey Galyonkin) already thought about what I was about to do in the project and made the JSON file to work with relational data bases.

The main issue is that if we try to insert the JSON file into MongoDB it won't have the proper "structure" and will make the retrieval of the data a tedious task.

The code for the JSON insertion can be seen below, with both packages MONGOLITE and RMONGODB and the result can be seem in *Figure*1.5

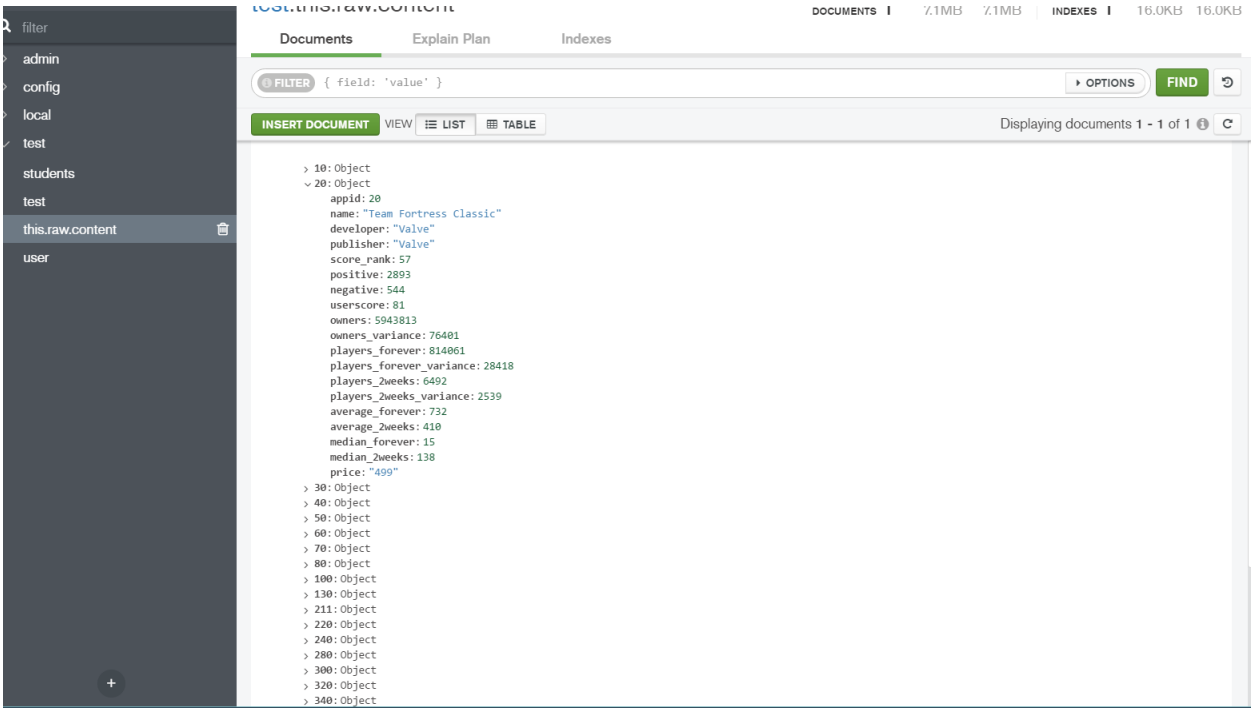


*Figure*1.5

```
  # THIS CODE IS INTENDED AS A DEMONSTRATION PURPOSES BUT NOT INTENDED TO BE RUN
  # AT THIS STAGE IN THE PROJECT
library(mongolite)
library(tidyverse)


list1<-this.raw.content

books<-mongo("this.raw.content")
books$insert(this.raw.content)

test1<-mongo("test1")
test1$insert(df_1)


books$find('{"10":1,"developer":"Valve"}')

v1<-data.frame(books$find(fields = '{"developer":"Valve"}'))
v2<-data.frame(books$find(fields = '{"30":1}'))

v1<-unlist(v1)
v2<-unlist(v2)

v3<-rbind.data.frame(v1,v2)


books$

index<-"20"

fields1<-paste0("{\"",index,"\"",':1}' )

v2<-books$find(fields = fields1)

fields2<-'{"20":1}'
"hola\""

library(ndjson)
validate(this.raw.content)

library(rmongodb)
mongo = mongo.create(host = "localhost")
mongo.is.connected(mongo)
mongo.get.databases(mongo)
mongo.get.database.collections(mongo, db = "test")

bs <- mongo.bson.from.JSON(this.raw.content)

db <- "test"
coll <- "test"

bs <- mongo.bson.from.JSON(this.raw.content)
```

```
mongo.insert(mongo, ns = paste0(db ,".", coll), b = bs)
```

Nonetheless saving the information in a database will always be easier and faster than directly collecting it from the internet

It seems that after a few hours of work and trying to get over this setback I'm only left with one option (that I think of) create a data frame out of the collected data from the API.

A few things need to be taken into consideration:

- The data is real time, that means the size of the data can cahnge over time
- Because of the above that also means that the collected data can have different dimensions, hence the code has to be fail proof in case the size of the data varies

Let's begin create the code for it

PART 2: Working code and extension

```
library(httr)
library(jsonlite)
library(data.table)

  # Taking the back-up copy of the Json file I had
raw_contet <- fromJSON("D:/DA5020/Final Project/test.json")

  # Putting into a data frame

steam1<-data.frame(do.call(rbind, raw_contet), stringsAsFactors=FALSE)

  # Creating a data table out of it so it can be more manageable later
steamdf_1<-as.data.table(steam1)
```

Small Setback

Due to the encoding of the JSON file and the structure resulting from it's conversion to an R readable format the nested data frame needs some small cleaning before being properly transformed into a data frame.

As it can be seen in the *Figure*1.51 the data has different lengths and thus the JSON parser created by me doesn't properly function unless this small code is run

l1	Large integer (17640 elements, 1.1 Mb)
l10	Large integer (17640 elements, 1.4 Mb)
l11	Large integer (17640 elements, 1.4 Mb)
l12	Large integer (17640 elements, 1.4 Mb)
l13	Large integer (17640 elements, 1.4 Mb)
l14	Large integer (17640 elements, 1.4 Mb)
l15	Large integer (17640 elements, 1.4 Mb)
l16	Large integer (17640 elements, 1.4 Mb)
l17	Large integer (17640 elements, 1.4 Mb)
l18	Large integer (17640 elements, 1.4 Mb)
l19	Large character (17382 elements, 1.2 Mb)
l2	Large character (17640 elements, 2.3 Mb)
l3	Large character (17640 elements, 1.9 Mb)
l4	Large character (17619 elements, 1.7 Mb)
l5	Large character (17640 elements, 1.2 Mb)
l6	Large integer (17640 elements, 1.1 Mb)
l7	Large integer (17640 elements, 1.1 Mb)
l8	Large integer (17640 elements, 1.1 Mb)
l9	Large integer (17640 elements, 1.1 Mb)

*Figure*1.51

```
  # These lines clean the empty character and Null values from the data
  # if there still are
steamdf_1[steamdf_1 == ""] <- NA

steamdf_1[steamdf_1 == " "] <- NA

steamdf_1[steamdf_1 == "NULL"] <- NA
```

After running the code we can see the data is equal in lenght within the whole document as seen in *Figure*1.52

l1	Large integer (17640 elements, 1.1 Mb)
l10	Large integer (17640 elements, 1.4 Mb)
l11	Large integer (17640 elements, 1.4 Mb)
l12	Large integer (17640 elements, 1.4 Mb)
l13	Large integer (17640 elements, 1.4 Mb)
l14	Large integer (17640 elements, 1.4 Mb)
l15	Large integer (17640 elements, 1.4 Mb)
l16	Large integer (17640 elements, 1.4 Mb)
l17	Large integer (17640 elements, 1.4 Mb)
l18	Large integer (17640 elements, 1.4 Mb)
l19	Large character (17382 elements, 1.2 Mb)
l2	Large character (17640 elements, 2.3 Mb)
l3	Large character (17640 elements, 1.9 Mb)
l4	Large character (17619 elements, 1.7 Mb)
l5	Large character (17640 elements, 1.2 Mb)
l6	Large integer (17640 elements, 1.1 Mb)
l7	Large integer (17640 elements, 1.1 Mb)
l8	Large integer (17640 elements, 1.1 Mb)
l9	Large integer (17640 elements, 1.1 Mb)

*Figure*1.52

Good! It seemed rather easy to create a data frame out of it, and MongoDB seems to like this format a lot

more

```
  # LIKE THE OTHER CODE THIS IS FOR DEMONSTRATION PURPOSES NOT TO BE RUN

#test1<-mongo("test1")
#test1$insert(df_1)
```

Althought still not quite good enough as it can be seen in *Figure*1.6, and that is because there is an issue with how the dataframe was created.

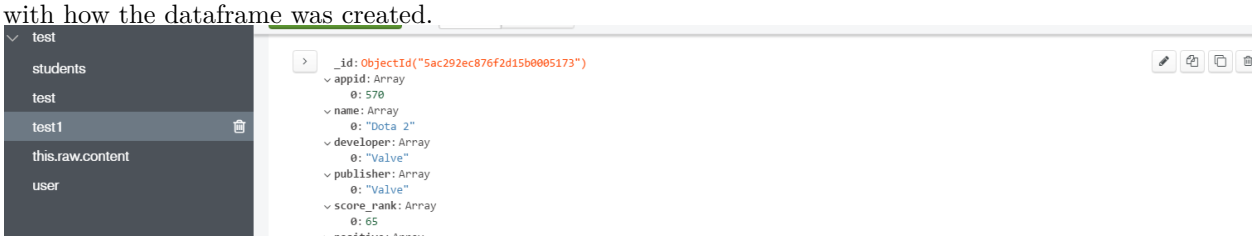


*Figure*1.6

Here lies the issue:

```
  #These lines allow to see class of some of the columns of the data frame
class(steamdf_1$name)
```

```
## [1] "list"
```

```
class(steamdf_1$developer)
```

```
## [1] "list"
```

As it can be seen every object within the data frame is a list, and that can't be allowed, because it a waste of memory and space and also because it's not the correct format for a database whether is NOSQL or SQL

Let's now create the code to create a proper data frame

```
  # This code allows us to take every column of the data frame
  # individually by unlisting them

col<-vector() #Vector to save the unlisted columns
for (i in seq(19)) {

  col <- paste("col", i, sep = "") #saving the columns within the vector
  #assigning the unlisted datra to each new created variable
 assign(col, unlist(steamdf_1[,i,with=FALSE]))
 #print(i)

}
```

The trick here is to take appid as a Primary Key in order to account for the total number of rows Also as I want the code to be as fail proof as possible the next lines need to run no matter the amount of data involved

This is the code needed to bind all the data together is as follows:

```
  # This whole chunk of code puts together all the columns created before
  # and bind them into a new data frame

dat1<-cbind.data.frame(col1,col2) #taking the first to columns to make the code run

dat12<-as.matrix(dat1)

  # function created to add columns to a data frame independent of their length
```

```
add.col<-function(df, new.col) {

n.row<-dim(df)[1]
length(new.col)<-n.row
cbind(df, new.col)

}

new_df<-dat12 #this is an iteration used as a failsafe and for debugging

  # As mentioned before this loop binds all the columns together in a new data frame
  # independently of their length
for (n in c("col3","col4","col5","col6","col7","col8",
            "col9","col10","col11","col12","col13","col14",
            "col15","col16","col17","col18","col19"))

{

  v<-get(n)
  v1<-assign(n,v)
  new_df<-add.col(new_df,v1)
}

steamdf_2<-as.data.table(new_df)

colnames<-names(steamdf_1) #using the original names obtained at the beginning

names(steamdf_2)<-colnames #using the original names obtained at the beginning


head(steamdf_2,5)
```

```
##     appid                             name            developer
## 1:    570                         Dota 2                Valve
## 2:    440                 Team Fortress 2                Valve
## 3:    730 Counter-Strike: Global Offensive                Valve
## 4: 304930                       Unturned Smartly Dressed Games
## 5: 578080    PLAYERUNKNOWN'S BATTLEGROUNDS      PUBG Corporation
##                publisher score_rank positive negative userscore    owners
## 1:                 Valve         65   764587   107047        87 121375328
## 2:                 Valve         86   469664    30783        93  44383892
## 3:                 Valve         71  2213809   253577        89  40686790
## 4: Smartly Dressed Games         74   276005    27648        90  35177709
## 5:      PUBG Corporation         13   370586   285295        55  32215840
##    owners_variance players_forever players_forever_variance players_2weeks
## 1:          298512       121375328                  298512        8762016
## 2:          197704        44383892                  197704        1301225
## 3:          190045        38748194                  185847       10293774
## 4:          177751        28132835                  160141         733720
## 5:          170636        31999936                  170102       21305378
##    players_2weeks_variance average_forever average_2weeks median_forever
## 1:                   91160           11716           1024            256
## 2:                   35392            4655            562            220
## 3:                   98655           17384            785           3795
```

```
## 4:                   26591            1320             436             154
## 5:                  140350           12061            1294            7300
##    median_2weeks price
## 1:           515     0
## 2:           140     0
## 3:           304  1499
## 4:            67     0
## 5:           874  2999
```

Now what I consider the hardest part has been completed. The code will run indepented of how many entries the data frame has and it will properly create a data frame out of it.

Now let's move on to the cleaning before we finish the script and upload everything to a data base

Cleaning the data frame

Almost the final step in this journey is cleaning the data set, fortunately the data comes somewhat clean with of course a few exceptions here and there.

This part will address those imperfections within the data and get the files ready to be uploaded to a data base

- First the empty characters will be addressed to see if the cleaning done before has not worked entirely

```
  # This chunk makes sure there are not more noise in the data remaining

steamdf_3<-as.data.frame(steamdf_2)#another iteration for debugging and testing

  # this loop goes through every column of the data frame
  # looking for any unsued character such as an empty space

for (i in seq(19))

{
  uc<-nrow(steamdf_3[steamdf_3[,i] %in% "" ,])

  print(paste0("The column ",colnames(steamdf_3)[i]," has ", uc, " Unused characters" ))

}
```

```
## [1] "The column appid has 0 Unused characters"
## [1] "The column name has 0 Unused characters"
## [1] "The column developer has 0 Unused characters"
## [1] "The column publisher has 0 Unused characters"
## [1] "The column score_rank has 0 Unused characters"
## [1] "The column positive has 0 Unused characters"
## [1] "The column negative has 0 Unused characters"
## [1] "The column userscore has 0 Unused characters"
## [1] "The column owners has 0 Unused characters"
## [1] "The column owners_variance has 0 Unused characters"
## [1] "The column players_forever has 0 Unused characters"
## [1] "The column players_forever_variance has 0 Unused characters"
## [1] "The column players_2weeks has 0 Unused characters"
## [1] "The column players_2weeks_variance has 0 Unused characters"
## [1] "The column average_forever has 0 Unused characters"
## [1] "The column average_2weeks has 0 Unused characters"
```

```
## [1] "The column median_forever has 0 Unused characters"
## [1] "The column median_2weeks has 0 Unused characters"
## [1] "The column price has 0 Unused characters"
```

As it can be seen from the output the small cleaning performed at the beginning has worked well enough.

Now the next part is a little bit more tedious. The names of the developers and the publishers have some noise, that means some of the names have not been properly encoded and then an unreadable string is displayed, some others do not have not been properly named or just simply some entries that do not fit for the purpose.

```
  # This code checks for every video game name that has the word
  # Demo at the end and codes it as a missing value (NA)
  # because demos don't have much value here

steamdf_4<-steamdf_3#another iteration for debugging and testing

  # Seeing how many columns have the Demo word at the end
grep("[Dd][Ee][Mm][Oo]*$",steamdf_4$name)
```

```
##  [1]  1659  3668  4433  4595  5448  5458  5469  5920  7047  7150  7285
## [12]  7389  9223  9294 12427 15104 15544 15969 16555 17032
```

```
  # Saving those into a variable
NotWanted<-grep("[Dd][Ee][Mm][Oo]*$",steamdf_4$name)

  # The loops run through those columns stored in the variable
  # and changes their names for missing values
for (i in NotWanted)

{

  steamdf_4$name[i]<-NA

}
  # This checks that the loop above has worked
grep("[Dd][Ee][Mm][Oo]*$",steamdf_4$name)
```

```
## integer(0)
```

```
  # This codes makes sure that Arcane Raise is properly displayed
  # As it can bee see below it was poorly coded
steamdf_4$name[3719]
```

```
## [1] "- Arcane preRaise -"
```

```
steamdf_4$name[3410]
```

```
## [1] "- Arcane Raise -"
```

```
  # This line stores the rows where that name appears
arc<-grep("Arcane Raise",steamdf_4$developer)

  # The loop runs through those rows and rewrites the name properly
for (i in arc)

{

  steamdf_4$name[i]<-"Arcane Raise"
```

```
}
  # This is to proof that the code has worked correctly
grep("Arcane Raise",steamdf_4$name)
```

```
## [1]  3410  3443  3719  8579  9209 11325
```

```
steamdf_4$name[3719]
```

```
## [1] "Arcane Raise"
```

```
steamdf_4$name[3410]
```

```
## [1] "Arcane Raise"
```

```
  # This code cleans up some of the names that had an @
  # at the beginning because they out their twitter handle
  # instead of their name on Steam
steamdf_4$developer<-gsub(".*@","",steamdf_4$developer)
```

This is interesting to say the least. As it can be seen from *Figure*1.7 it seems that some of the data is not properly encoded and they would need to be cleaned.

	'n Verlore Verstand	Skobbejak Games	Skobbejak Games
	-	GY Games	GY Games
	- Arcane preRaise -	Arcane Raise	ArcaneRaise
	- Arcane Raise -	Arcane Raise	ArcaneRaise
	- Arcane RERaise -	Arcane Raise	ArcaneRaise
	- Occult preRaise -	Arcane Raise	ArcaneRaise
	- Occult Raise -	Arcane Raise	ArcaneRaise
	- Occult RERaise -	Arcane Raise	ArcaneRaise
	-(A Bit Crosser-Three Kingdoms)	singi	singi
	-Monobeno-	Lose	HIKARI FIELD
	- Old Watch	<U+7070><U+70EC><U+5929><U+56FD>	<U+7070><U+70EC><U+5929><U+56FD>
	- Red Obsidian Remnant	Red Obsidian Studio	Beijing New Era Network Technology Co., Ltd.
	- ShP	Xitilon	Xitilon
	- Tales of Hongyuan	<U+5F18><U+539F><U+6E38><U+620F>	<U+5F18><U+539F><U+6E38><U+620F>
	/ Barrage Musical ~A Fantasy of Tempest~	SlimeSmile	SlimeSmile
	Hidden Star in Four Seasons.	<U+4E0A><U+6D77><U+30A2><U+30EA><U+30B9>...	Mediascape Co., Ltd.
	The Disappearing of Gensokyo	MyACG Studio	MyACG Studio
	(Chinese PaladinSword and Fairy 6)	SOFTSTAR ENTERTAINMENT INC.	SOFTSTAR ENTERTAINMENT INC.
	(Chinese PaladinSword and Fairy)	SOFTSTAR TECHNOLOGY BEIJINGCOLTD	SOFTSTAR TECHNOLOGY BEIJINGCOLTD
	| Spring Breeze	<U+6728><U+5B50><U+5DE5><U+574A>	HongbinGame
	1 Sakura no Mori Dreamers part.1	MOONSTONE	HIKARI FIELD
	Antinomy of Common Flowers.	<U+9EC4><U+660F><U+30D5><U+30ED><U+30F3>...	SUNFISH Co., Ltd.
	Chinese PaladinSword and Fairy 5 Prequel	Softstar Technology (Beijing) Co.,Ltd	Beijing New Era Network Technology Co., Ltd.
	Circles	Jeroen Wimmers	Jeroen Wimmers
	Cryste: the Faith of Fire Vol.1	Salvation	<U+51C0><U+571F><U+5236><U+4F5C><U+5BA4>

*Figure*1.7

If a simple pattern matching is run we see that none of those strings are recognized. Let's inspect further

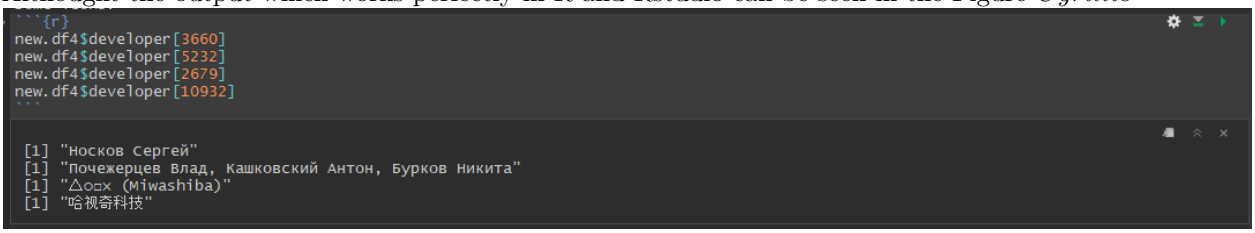
```
  # Attempt to take the rows with different encoding
grep(".*<","",steamdf_4$developer)
```

```
## integer(0)
```

If the rows are taken manually, their names are properly displayed, but in another alphabet cyrillic and/or sinograms (Chinese) and some icons.

After a lenghty research and multiple trials, errors and breaking LateK a few times I discovered that unfortunately the way encoding works with LaTeK and PDF these lines will not display properly on the final report, that's why I have decided no to make it run for the final report.

Althought the output which works perfectly in R and Rstudio can be seen in the Figure *Cyrillic*



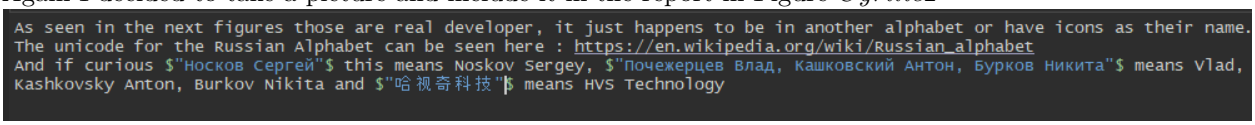
FigureCyrillic

```
  # THESE LINES CAN BE UNCOMMENTED TO SEE THEIR REAL NAMES
  # IN CYRILLIC AND CHINESE AS R HAS NO ISSUE WITH IT

#steamdf_4$developer[3660]
#steamdf_4$developer[5232]
#steamdf_4$developer[2679]
#steamdf_4$developer[10932]
```

As I was not happy by not being able to display these lines and their names properly I decided to explain it verbally outside a chunk of code but then again that didn't work either.

Again I decided to take a picture and include it in the report in Figure *Cyrilic*2



*FigureCyrillic*2

Although quite a bit of time has been spent trying to clean that part only to see that it was an encoding issue all along this makes things a little bit easier and shortens the task.

Suprinsingly enough after further inspection I decided not to continue with more cleaning as most of the names that seem rather awkard or wrongly encoded are really not; Take for example : $[erka : es], DarksquidMedia$ {COMPANY_NAME_GOES_HERE} $+7Software$ and $.ezGames$

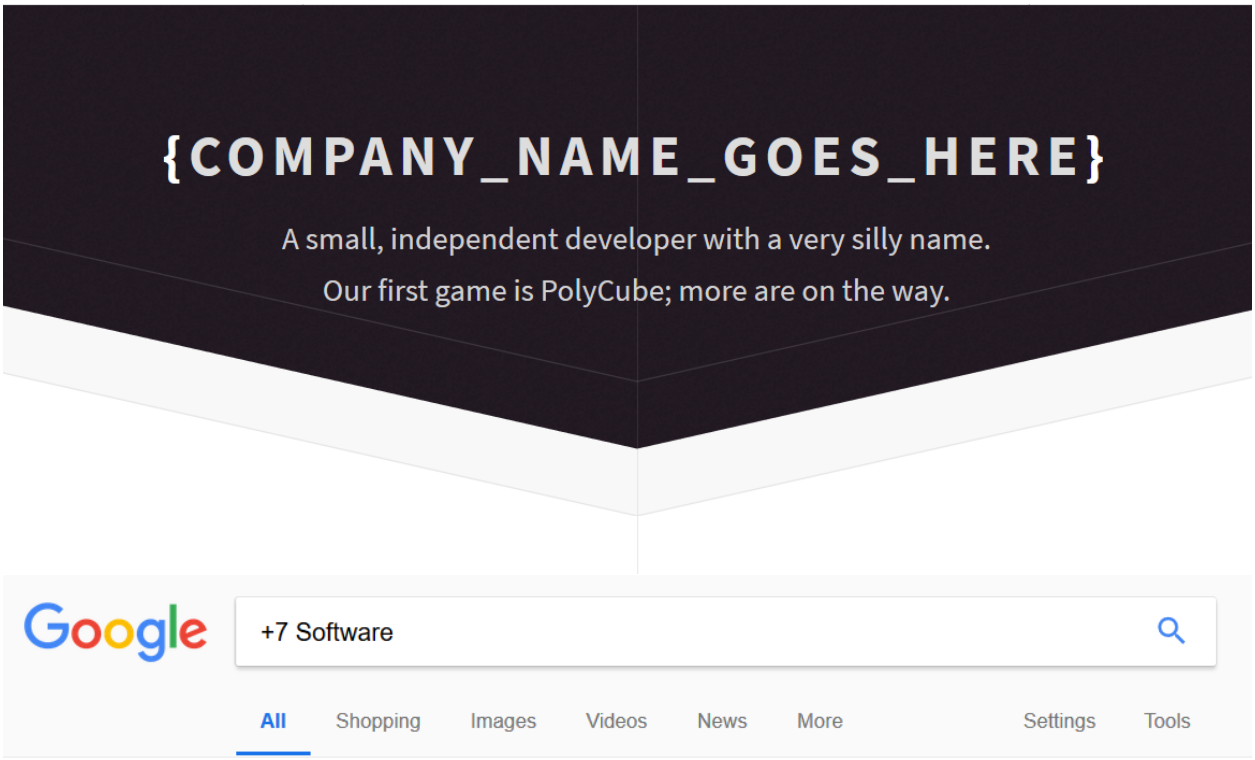
I have decided not to manipulate the data further as it might be counter productive if an analysis shall be performed as some company names might be wrongly typed or deleted.

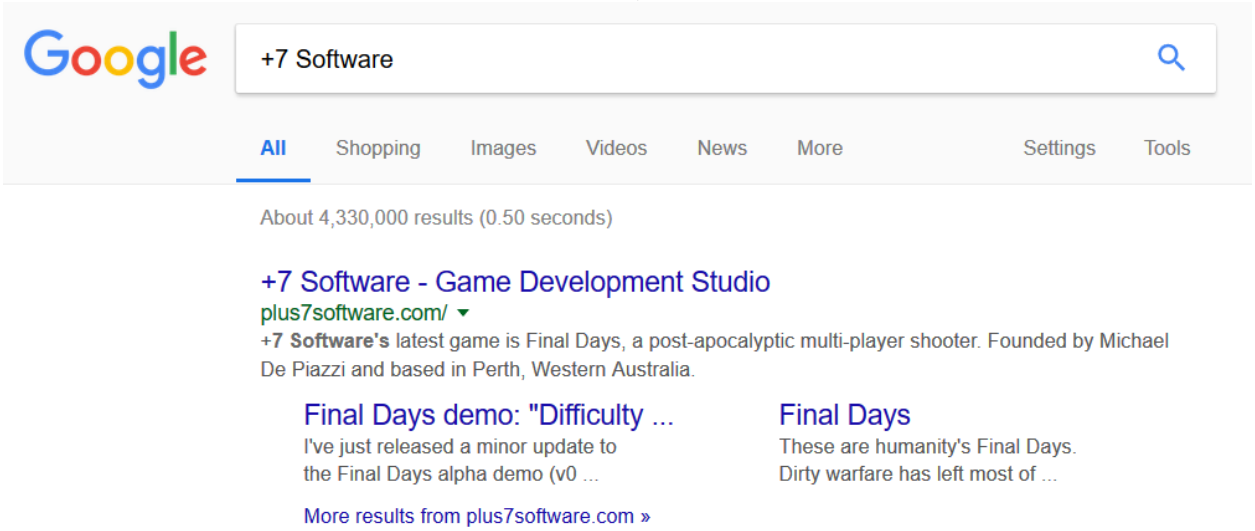
Here's a proof of some weird names that I have found in the data frame and they ended up being real

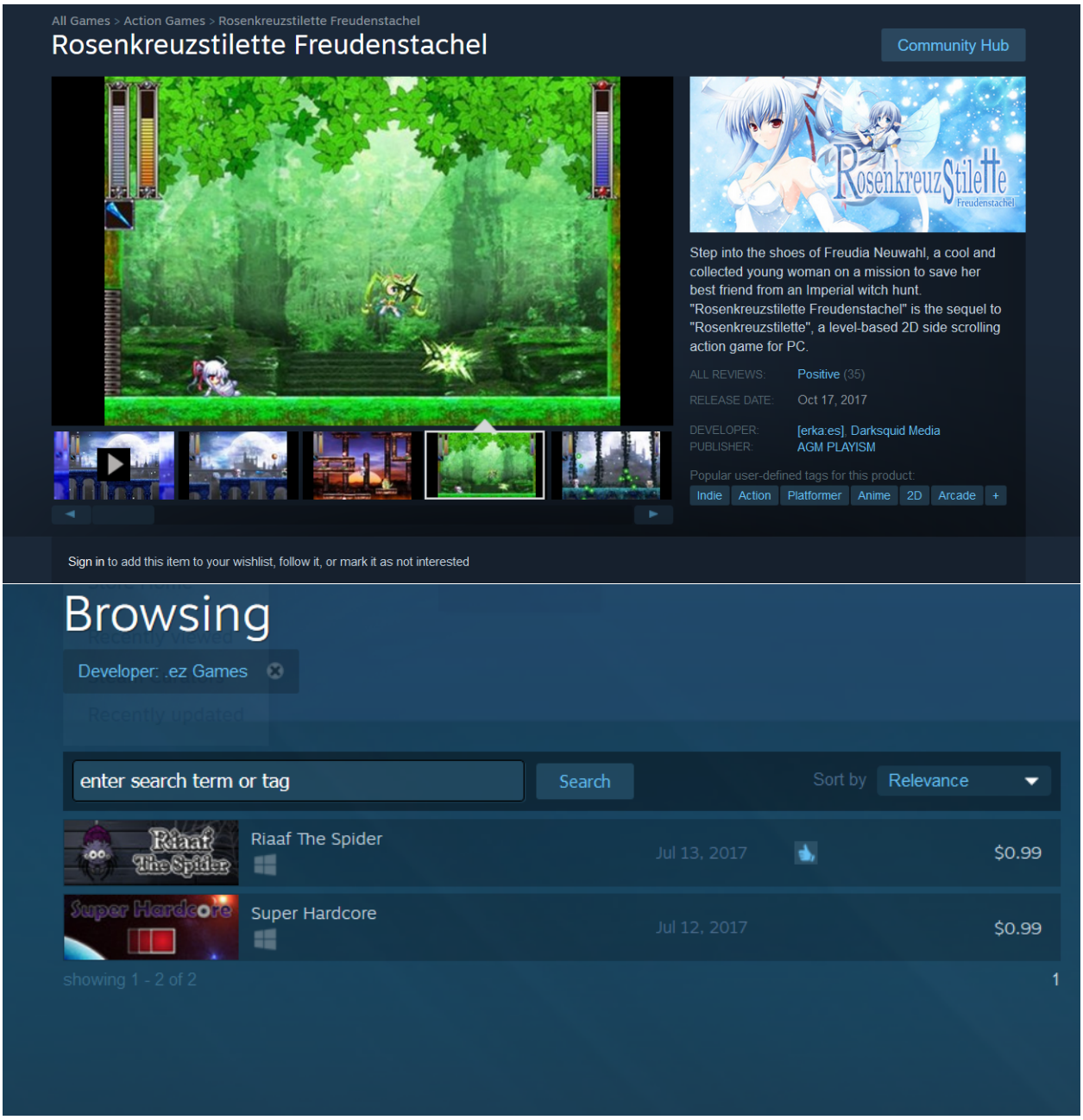
```
  # This code shows some of the weird names that some companies
  # have and a reason not to change more their names
WeirdNames<- as.data.frame(steamdf_4$developer[c(8470,11026,17441,17368)])
WeirdNames
```

```
##   steamdf_4$developer[c(8470, 11026, 17441, 17368)]
## 1                                          .ez Games
## 2                          [erka:es], Darksquid Media
## 3                            {COMPANY_NAME_GOES_HERE}
## 4                                         +7 Software
```

And their corresponding real life names...


{COMPANY_NAME_GOES_HERE}
A small, independent developer with a very silly name.
Our first game is PolyCube; more are on the way.


Google
+7 Software
All
Shopping
Images
Videos
News
More
Settings
Tools
About 4,330,000 results (0.50 seconds)
+7 Software - Game Development Studio
plus7software.com/
+7 Software's latest game is Final Days, a post-apocalyptic multi-player shooter. Founded by Michael De Piazzi and based in Perth, Western Australia.
Final Days demo: "Difficulty ...
I've just released a minor update to the Final Days alpha demo (v0 ...
Final Days
These are humanity's Final Days. Dirty warfare has left most of ...
More results from plus7software.com »



One last thing I've noticed while working one the data is that every column within the data frame is a character value and we don't want that, to fix this we will run this simple chunk of code

```
for (i in seq(from =5, to=19))#running from the 5th column to the 19th

{


  steamdf_4[,i]<-as.integer(steamdf_4[,i])#converting those columns into integer class
  print(paste0("Object class of the column ",colnames(steamdf_4)[i]
               ," has been converted to: ",class(steamdf_4[,i])))
```

```
}
```

```
## [1] "Object class of the column score_rank has been converted to: integer"
## [1] "Object class of the column positive has been converted to: integer"
## [1] "Object class of the column negative has been converted to: integer"
## [1] "Object class of the column userscore has been converted to: integer"
## [1] "Object class of the column owners has been converted to: integer"
## [1] "Object class of the column owners_variance has been converted to: integer"
## [1] "Object class of the column players_forever has been converted to: integer"
## [1] "Object class of the column players_forever_variance has been converted to: integer"
## [1] "Object class of the column players_2weeks has been converted to: integer"
## [1] "Object class of the column players_2weeks_variance has been converted to: integer"
## [1] "Object class of the column average_forever has been converted to: integer"
## [1] "Object class of the column average_2weeks has been converted to: integer"
## [1] "Object class of the column median_forever has been converted to: integer"
## [1] "Object class of the column median_2weeks has been converted to: integer"
## [1] "Object class of the column price has been converted to: integer"
```

```
steamdf_4$appid<-as.integer(steamdf_4$appid)#the last column is the appid
```

Final-Step : Storing the Data

The final step of the script will be storing the data in a data base. Since the purpose of the script is to facilitate the collecting and storing of data through SteamSpi's API the slected databased will be MongoDB and a relational SQL data base.

Things to consider:

- The data has been automatically flattened by the script, converted into a data frame object and ready to be uploaded to MongoDb

*The SQL data base will have as its Primary Key the Appid because it's the most suitable column for the given data (it seems as if someone already thought about it)

First let's put it into a SQL relational data base

To make the size of the data base smaller the data set will be splitted into three equal parts, all of them containing at least the Primary Key

The first part will include the Appid , Name, Developer and Publisher The second part will include the Appid, Score Rank, Positive, Negative, and User Score The third part will be the biggest one and it will contain the Appid, and the remaining columns : owners, owners_variance, players_forever, players_forever_variance, players_2weeks, players_2weeks_variance, average_forever, average_2weeks, median_forever, median_2weeks, price

```
library(RSQLite)
library(sqldf)
```

```
## Loading required package: gsubfn
```

```
## Loading required package: proto
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
```

```
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(DBI)

  #This chunk creates a SQL database with the parameters mentioned above

  # Conneting to the database object
steamS <- dbConnect(SQLite(),'steamdf_4.db')

  #Setting up the parameters for the first table
dbSendQuery(
  steamS,
  "CREATE TABLE table11 (
  appid INTEGER,
  name TEXT,
  developer TEXT,
  publisher TEXT,
  PRIMARY KEY (appid))"
  )
```

```
## <SQLiteResult>
##   SQL  CREATE TABLE table11 (
##   appid INTEGER,
##   name TEXT,
##   developer TEXT,
##   publisher TEXT,
##   PRIMARY KEY (appid))
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
  #Writting the first table
dbWriteTable(
  conn = steamS,
  name = "table11",
  steamdf_4[,1:4],
  append = T,
  row.names = F
)
```

```
## Warning: Closing open result set, pending rows
```

```
  #Writting the second table
dbWriteTable(
  conn = steamS,
  name = "table12",
  steamdf_4[,c(1,5,6,7,8)],
  append = T,
  row.names = F
)
```

```
  #Writting the thirdtable
dbWriteTable(
  conn = steamS,
  name = "table13",
  steamdf_4[,c(1,9,10,11,12,13,14,15,16,17,18,19)],
  append = T,
  row.names = F
)

#dbDisconnect(steamS)
```

With this done let's pass a few simple queries to see if the data as been correctly uploaded to the database.

```
  # This chunk performs simple queries to test if the tables have been
  # created with the correct parameters

  #looking for any video game named Dota 2
dbGetQuery(
  steamS,
  "SELECT appid,name
  FROM table11
  WHERE name
  LIKE 'Dota 2'"
)
```

```
##   appid   name
## 1   570 Dota 2
```

```
  # its R counterpart to verify
steamdf_4%>%
  select(appid,name)%>%
  filter(name=="Dota 2")
```

```
##   appid   name
## 1   570 Dota 2
```

```
  #looking for any video game with a score_rank of 65
head(dbGetQuery(
  steamS,
  "SELECT appid,score_rank
  FROM table12
  WHERE score_rank
  LIKE '65'"
),5)
```

```
##    appid score_rank
## 1    570         65
## 2 438100         65
## 3   3910         65
## 4  16450         65
## 5 232090         65
```

```
  # its R counterpart to verify
head(steamdf_4%>%
  select(appid,score_rank)%>%
  filter(score_rank=="65"),5)
```

```
##    appid score_rank
## 1    570         65
## 2 438100         65
## 3   3910         65
## 4  16450         65
## 5 232090         65
```

```
  #looking for any video game with a price of 0
head(dbGetQuery(
  steamS,
  "SELECT appid,price
  FROM table13
  WHERE price
  LIKE '0'"
),5)
```

```
##    appid price
## 1    570     0
## 2    440     0
## 3 304930     0
## 4 230410     0
## 5 227940     0
```

```
  # its R counterpart to verify
head(steamdf_4%>%
  select(appid,price)%>%
  filter(price=="0"),5)
```

```
##    appid price
## 1    570     0
## 2    440     0
## 3 304930     0
## 4 230410     0
## 5 227940     0
```

Nice, it seems the data has been passed properly to the data base.

I will pass some more complex queries to see if the Primary Key has been correctly assigned and if the relational schema is set up the way I intended to.

```
  # This queries retrieves any game called Dota 2 and displays
  # the appid, the name, score_rank, userscore and price
dbGetQuery(
  steamS,
  "SELECT table11.appid ,table11.name ,table12.score_rank,table12.userscore,table13.price
  FROM table11
  INNER JOIN table12 ON table11.appid = table12.appid
  INNER JOIN table13 ON table11.appid= table13.appid
  WHERE table11.name = 'Dota 2'")
```

```
##   appid   name score_rank userscore price
## 1   570 Dota 2         65        87     0
```

```
steamdf_4[1,c(1,2,5,6,19)]
```

```
##   appid   name score_rank positive price
## 1   570 Dota 2         65   764587     0
```

We can see in the code above that with one query the relational database has been set up and it works as intended

After having stored the data in a relation SQL database the data can be now stored in a NOSQL data base such as MongoDb.

One of the advantages of NOSQL databases as we have learnt this course is that they do not require schema and in some cases they don't require a primary key so uploading the data frame to MongoDb is very straightforward

```
library(mongolite)
```

```
## Warning: package 'mongolite' was built under R version 3.4.4
```

```
  # uploading the cleaned data into MongoDb
mongonew.steamdf_4<-mongo("steamdf_4")
mongonew.steamdf_4$insert(steamdf_4)
```

```
## List of 5
##  $ nInserted  : num 17640
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

To test if the data frame has beencorreclty uploaded into MongoDb a simple query can be run

```
  # Query looking for any video game with its publisher being Valve
head(mongonew.steamdf_4$find(fields='{"name":1,"publisher":1}',
                             query='{"publisher":"Valve"}'),5)
```

```
##                         _id                            name publisher
## 1 5adc44c0876f2d0818003d12                          Dota 2     Valve
## 2 5adc44c0876f2d0818003d13                 Team Fortress 2     Valve
## 3 5adc44c0876f2d0818003d14 Counter-Strike: Global Offensive     Valve
## 4 5adc44c0876f2d0818003d18                   Left 4 Dead 2     Valve
## 5 5adc44c0876f2d0818003d1c          Counter-Strike: Source     Valve
```

Good, it seems that after all the work put into cleaning and re-modelling the data it finally fits MongoDb structure and the query works

As it can be seen in *Figure*1.71 the data as been properly uploaded to the database now with the correct format and ready to be retrieved for whatever analysis or usage want to be applied to it

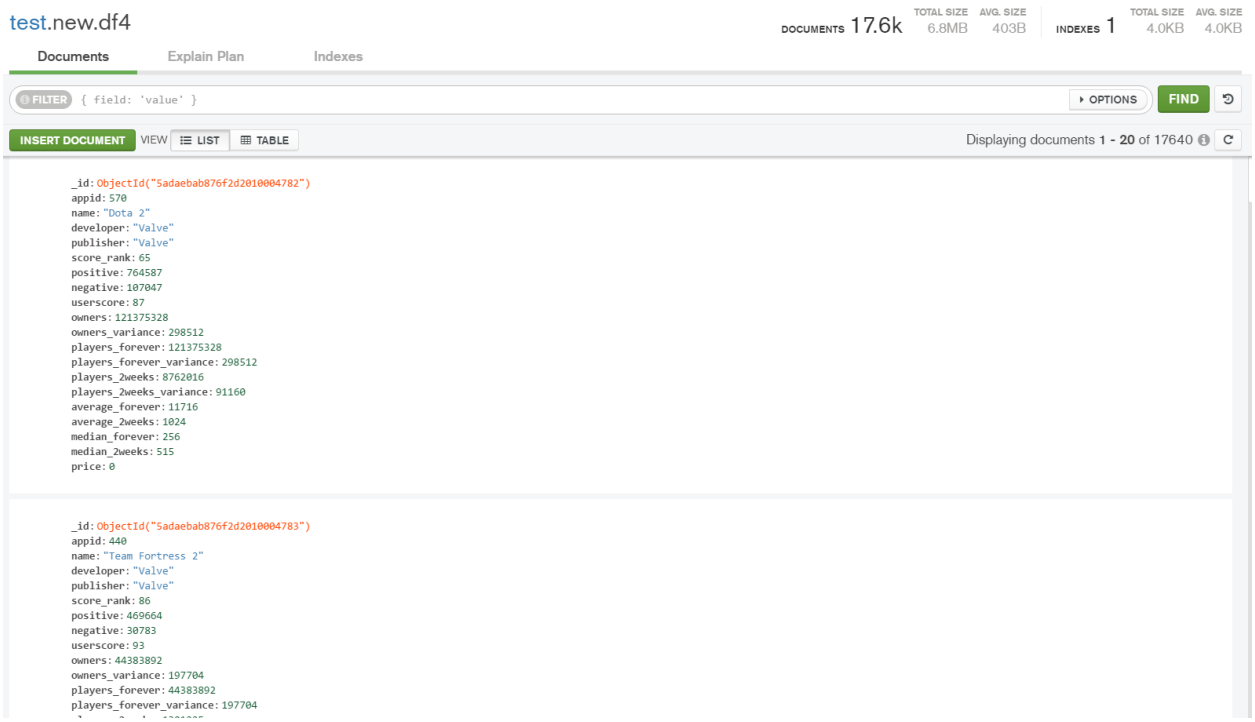


*Figure*1.71

This ends the part of collecting and storing the data. Since I was hit by the setback of Steam changing I did not have intended to continue the script much more since my initial idea was to create a script that did everything automatically.

Because of that I will add some data visualization to the report and some retrieval of said data as well as some functions that given the data at hand might be very useful.

This function for example will return the appid the name, score_rank, userscore, negative and positive reviews as well as the price of any videogame the user decides to enter

```
VideoGameViewer<-function(GameName)


{


query1<-paste0(
  "SELECT table11.appid ,table11.name ,table12.score_rank,table12.userscore,
  table12.negative,table12.positive,table13.price
  FROM table11
  INNER JOIN table12 ON table11.appid = table12.appid
  INNER JOIN table13 ON table11.appid= table13.appid
  WHERE table11.name = '",GameName,"'")

sql<-dbGetQuery(
  steamS,query1)

return(sql)
```

```
}


VideoGameViewer("Metro 2033")
```

```
##   appid       name score_rank userscore negative positive price
## 1 43110 Metro 2033         82        93     1213    15592  1499
```

Now there's an issue with the way the queries are sent to the SQL database. That issue is that there can be no words with the special character ′ such as *Don′tStarve* the query will be stopped and an error message will be displayed.

The error can be seen below.

```
> VideoGameViewer<-function(GameName)
+
+
+ {
+
+
+ query1<-paste0(
+   "SELECT table11.appid ,table11.name ,table12.score_rank,table12.userscore,
+   table12.negative,table12.positive,table13.price
+   FROM table11
+   INNER JOIN table12 ON table11.appid = table12.appid
+   INNER JOIN table13 ON table11.appid= table13.appid
+   WHERE table11.name = '",GameName,"'")
+
+ sql<-dbGetQuery(
+   steamS,query1)
+
+ return(sql)
+
+ }
>
>
> VideoGameViewer("Dont't Starve")
Error in rsqlite_send_query(conn@ptr, statement) : near "t": syntax error
>
```

Now there's a solution for that, but I do not approve since I believe it is transforming the data too much for what it's intended.

Now the function would work given that the user enters the name without the ′ symbol such as *DontStarve*.

It can't be tested "on the spot" because there are only three data frames on the SQL database and none of them have their names modified that much.

```
  # This code deletes every ' character found in the data
steamdf_5<-steamdf_4#Iteration to avoid doing this in the original data

for (i in seq(17640))

{

  steamdf_5$name[i]<-gsub("'","",steamdf_5$name[i])

}
```

After that function let's move one with another function that can be very useful for the user. Retrieve a game information based on it's userscore

```
  # This functions will retrieve any videogame name based on the userscore
  # inputed by the user
UserScoreViewer<-function(userscore)


{


query1<-paste0(
  "SELECT table11.appid ,table11.name ,table12.score_rank,table12.userscore,
  table12.negative,table12.positive,table13.price
  FROM table11
  INNER JOIN table12 ON table11.appid = table12.appid
  INNER JOIN table13 ON table11.appid= table13.appid
  WHERE table12.userscore = '",userscore,"'")

sql<-dbGetQuery(
  steamS,query1)

return(sql)

}


head(UserScoreViewer("85"),5)
```

```
##    appid                          name score_rank userscore negative
## 1 291550                     Brawlhalla         59        85     9167
## 2 359550 Tom Clancy's Rainbow Six Siege         59        85    25118
## 3 588430                Fallout Shelter         59        85     2753
## 4  42700        Call of Duty: Black Ops         68        85     1624
## 5  42710        Call of Duty: Black Ops         68        85     1625
##   positive price
## 1    54657     0
## 2   149050  1499
## 3    16247     0
## 4    12748  3999
## 5    12749  3999
```

The last function is a little bit more fancy but also migth eb a little bit more useful. The idea is to let the user give an input of an userscore and the price range and the function would return a game that meets said criteria.

```
UserPrice<-function(userscore, PriceBelow)


{


query1<-paste0(
  "SELECT table11.appid ,table11.name ,table12.score_rank,table12.userscore,
  table12.negative,table12.positive,table13.price
  FROM table11
  INNER JOIN table12 ON table11.appid = table12.appid
  INNER JOIN table13 ON table11.appid= table13.appid
```

```
  WHERE table12.userscore = '",userscore,"' AND table13.price<",PriceBelow,"")

sql<-dbGetQuery(
  steamS,query1)

return(sql)

}


head(UserPrice("85","500"),5)
```

```
##    appid                       name score_rank userscore negative positive
## 1 291550                  Brawlhalla         59        85     9167    54657
## 2 588430             Fallout Shelter         59        85     2753    16247
## 3 204300 Awesomenauts - the 2D moba         59        85     3395    20111
## 4 206500               AirMech Strike         59        85     1182     6954
## 5   4540                  Titan Quest         59        85      328     1894
##   price
## 1     0
## 2     0
## 3     0
## 4     0
## 5     0
```

While the functions above don't pose much of a coding challenge they can be quite useful for any user who decides to run the script.

Small Data Visualization

For the final part I will perform some visualization of the given data to see what the data I've been dealing looks like in terms of their variables.

Since the data his quite big and the amount of data can be overwhelming I have decided to notch down the dataframe only to the Top 50 video games in the data frame.

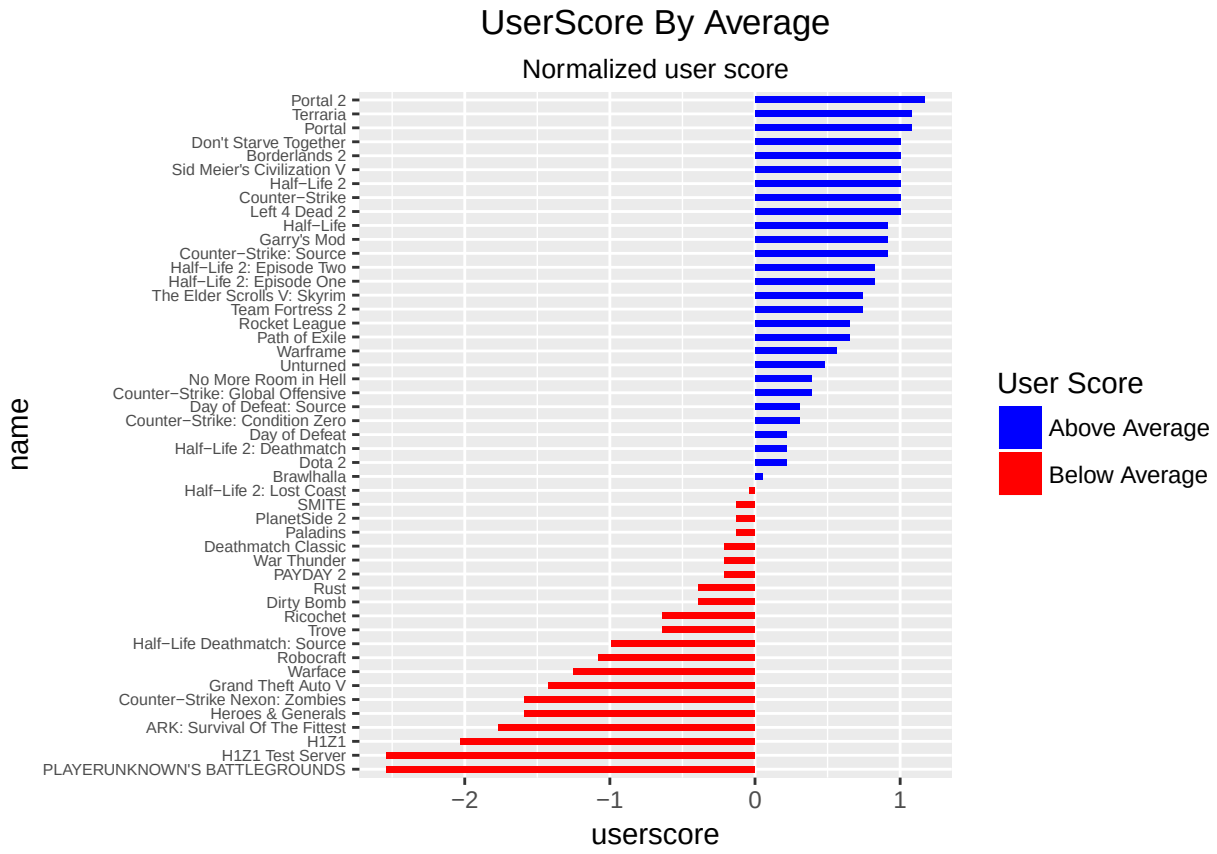
The results, for someone who knows a little bit of the video game industry are quite revealing althought maybe somewhat expected.

The first plot revelas something interesting and some people dare to speculate that is in part why Steam changed their pricacy policiy , the plot shows whether a gamer is above or below the average within their group and there is a game that is quite an outlier PLAYER UNKNWOWN's BATTLEGROUNDS and it is quite far from the average. This means that the game is "Over-Hyped" a term widely used in this industry and that means that the game has gotten a lot of popularity and hence a lot of sales but the game necessarily is not that good.

```
library(ggplot2)
top501<-steamdf_4[1:50,] #taking only the Top 50

top501$userscore<-round((top501$userscore - mean(top501$userscore))/sd(top501$userscore),
                        2) #normalizing the userscore average
top501$userscore_type <- ifelse(top501$userscore < 0, "below", "above")
top501 <- top501[order(top501$userscore), ]#sorting in desceding order
top501<-top501[-26,]
top501$name <- factor(top501$name, levels = top501$name)
```

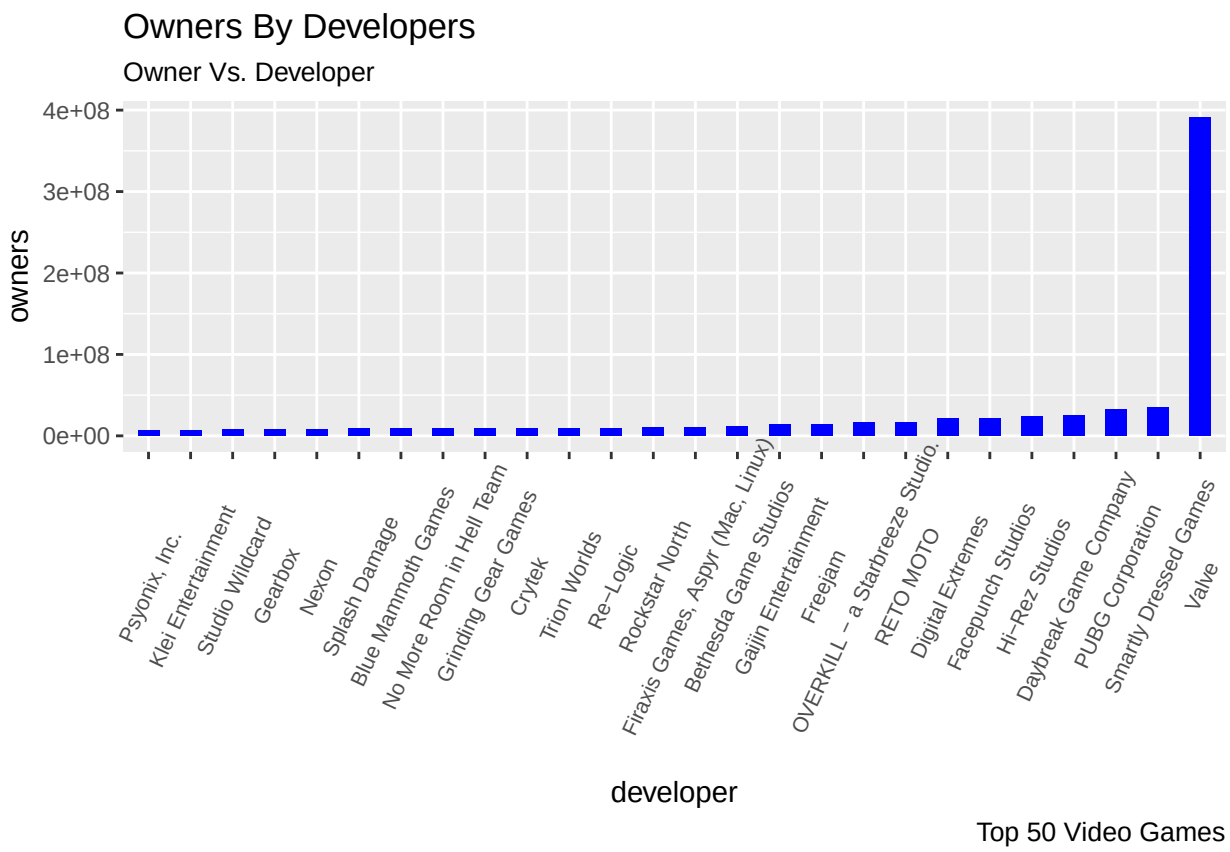
```
ggplot(top501, aes(x=name, y=userscore, label=userscore)) +
  geom_bar(stat='identity', aes(fill=userscore_type), width=.5)  +
  scale_fill_manual(name="User Score",
                    labels = c("Above Average", "Below Average"),
                    values = c("above"="blue", "below"="red")) +
  labs(subtitle="Normalized user score",
       title= "UserScore By Average") +
  coord_flip()+
  # adjusting parameters to fit the plot with the given data
  theme(plot.title = element_text(hjust = .5),plot.subtitle = element_text(hjust = .5),
        axis.text.y = element_text(size = 6,hjust=1))
```



This second plot shows something not that unexpected and that is that Valve account for the majority of owners in the Top 50, not suprising taking into account that Valve is also the Developer of Steam, and I dare to say quite good video game developers.

```
#agregatting the developer by the ammount of owners
OwnersDevelopers <- aggregate(top501$owners, by=list(top501$developer), FUN=sum)
colnames(OwnersDevelopers) <- c("developer", "owners")#renaming the columns again
#sorting in increasing order
OwnersDevelopers <- OwnersDevelopers[order(OwnersDevelopers$owners), ]
OwnersDevelopers$developer[3]<-"Studio Wildcard"
OwnersDevelopers$developer[4]<-"Gearbox"
OwnersDevelopers$developer <- factor(OwnersDevelopers$developer,
                                     levels = OwnersDevelopers$developer)
```

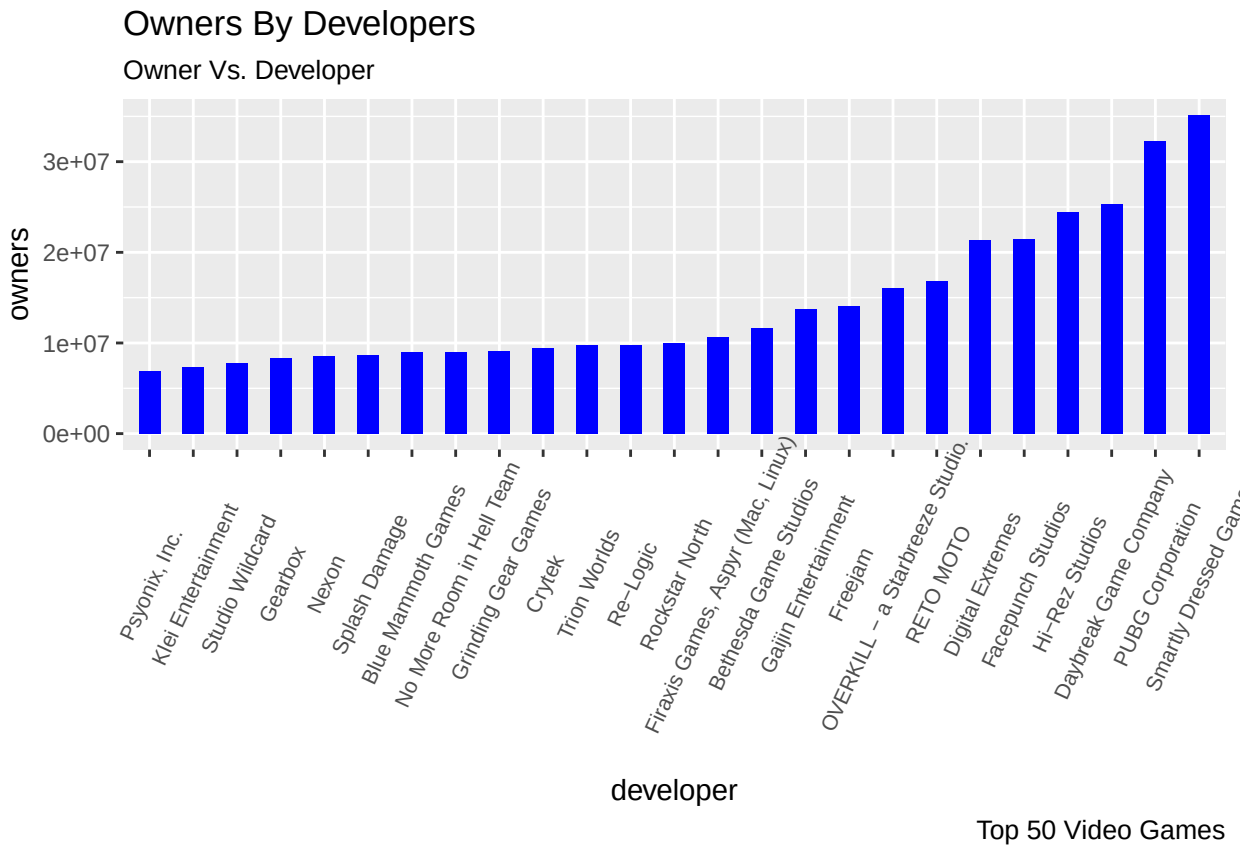
```
ggplot(OwnersDevelopers, aes(x=developer, y=owners)) +
  geom_bar(stat="identity", width=.5, fill="blue") +
  labs(title="Owners By Developers",
       subtitle="Owner Vs. Developer",
       caption="Top 50 Video Games") +
    # adjusting parameters to fit the plot with the given data
  theme(axis.text.x = element_text(angle=65, vjust=0.6,size = 8,hjust=0.5))
```



Since Valve was skewing the plot above I have decided to remove it from the enxt plot, and as we can see the distribution is more normal and not that sweked than the previous plot

```
OwnersDevelopers1<-OwnersDevelopers[1:25,]


ggplot(OwnersDevelopers1, aes(x=developer, y=owners)) +
  geom_bar(stat="identity", width=.5, fill="blue") +
  labs(title="Owners By Developers",
       subtitle="Owner Vs. Developer",
       caption="Top 50 Video Games") +
  theme(axis.text.x = element_text(angle=65, vjust=0.6,size = 8,hjust=0.5))
```



Lesson Learnt

As this is the end of the project it is now time to reflect one things that could have been improved and lessons that I've learnt along the way.

Let's summarise those:

- Contigency plans; They are always needed no matter how much one thinks the project will be smooth, at the shortest notice something can go south and turn everything upside down rendering all the work you have put in a hold until you come up with a solution.
- Document everything; This one is also intertwined with contigency plans, if someone goes wrong and not as expected you always have a report for what has happened, when, why and how it happened.
- Back ups: Nowadays there is no excuse not to have multiple back ups of everything in different clouds, servers or devices, if something breaks or a last minute issue arises, your back ups can save you from a certain doom (The example is my backed-up JSON file)
- Start early; The sooner you start the better, you can tackle unexpected issues and problems while also making you coding effort less of a painful proccess

All in all this has been a great learning experience, even the fact that I ran into the issue of the Steam Privacy Policy, that has provided me with an insight of how to react when something goes as bad as it can be, but if one maintains a rather well disciplined working ethic and take into account the points mentioned above any issue can be tackled and any (almost) problem solved

Sources

I've tried to save every web page I visited during the creation of the project so it could be listed in here.

https://companynamegoeshere.com/
https://en.wikipedia.org/wiki/Russian_alphabet
https://stackoverflow.com/questions/12357592/efficient-multiplication-of-columns-in-a-data-frame
https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf
http://steamspy.com/
https://stackoverflow.com/questions/16566799/change-variable-name-in-for-loop-using-r
https://stackoverflow.com/questions/40225650/how-to-cbind-many-data-frames-with-a-loop
https://stat.ethz.ch/pipermail/r-help/2004-October/059752.html
https://stat.ethz.ch/pipermail/r-help/2016-December/443790.html
https://ryouready.wordpress.com/2009/01/23/r-combining-vectors-or-data-frames-of-unequal-length-into-one-data-frame/

https://www.r-bloggers.com/applying-an-operation-to-a-list-of-variables/
http://r.789695.n4.nabble.com/Loop-over-several-variables-td4648112.html
http://r.789695.n4.nabble.com/Creating-and-assigning-variable-names-in-loop-td4221080.html
http://r.789695.n4.nabble.com/looping-variable-names-td3255711.html
https://stats.stackexchange.com/questions/10838/produce-a-list-of-variable-name-in-a-for-loop-then-assign-values-to-them

http://www.dummies.com/programming/r/how-to-create-an-array-in-r/
https://stackoverflow.com/questions/7448960/combining-columns-from-multiple-data-frames-with-a-loop
https://stackoverflow.com/questions/1923273/counting-the-number-of-elements-with-the-values-of-x-in-a-vector

https://stackoverflow.com/questions/30664012/extract-column-name-to-vector-by-calling-the-column-name

http://www.stat.cmu.edu/~cshalizi/rmarkdown/
http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html
https://rmarkdown.rstudio.com/developer_parameterized_reports.html
https://rmarkdown.rstudio.com/tufte_handout_format.html#comment-1582377678
http://www.tex.ac.uk/FAQ-formatstymy.html
https://latex.org/forum/viewtopic.php?t=22321
http://texblog.net/latex-archive/distributions/babel-update-miktex/
https://tex.stackexchange.com/questions/83440/inputenc-error-unicode-char-u8-not-set-up-for-use-with-latex