

Facebook Draft Report V2

paul

March 3, 2017

Contents

Summary of Key Findings:	2
Quick Introduction To Facebook API:	2
Structure:	2
Request Format:	3
Pagnation:	3
Rate Limit:	3
Further Information:	3
Rfacebook Package:	4
Summary of Findings For Rfacebook:	4
List of Deprecated Functions:	4
Setting Up (Libraries and Folder)	4
Creating Test Account:	5
Creating Long Lasting Authorization Token With <i>fbOAuth</i> Function:	5
Exploring the Package:	5
<i>Location:</i>	5
Exploring Facebook API for Nodes that Contain Location Information:	5
<i>Albums:</i>	5
<i>Comments:</i>	6
<i>Events:</i>	6
<i>Pages:</i>	6
<i>Posts:</i>	6
<i>Photos:</i>	8
<i>callAPI</i> Function:	8
Trying to extract my profile information:	8
Extracting information about a public facebook page:	9
Extracting information about a comment.	9
Developing a function to extract the “ID” of facebook object through using the URL of the object:	10
Conclusion:	11
Exploring <i>getPage</i>, <i>getPost</i>, <i>getCommentReplies</i>:	11
What if the posts are pictures?	13
Exploring <i>searchGroup</i> & <i>getGroup</i> functions:	13
Exploring <i>searchPage</i>:	14
Trying <i>getLikes</i>:	15
Search API:	16
Mini Project1:	16
Description:	16
Code:	16

MiniProject 2:	18
Description:	18
Code:	18
Gathering RawData:	18
Planning:	19
Query Response Structure:	19
Paging Response Structure:	19
Desired Organized Output Structure:	19
Desired Final DF:	19
Pesudocode For Parsing:	19
Parsing and Processing information.	23

Summary of Key Findings:

- It is easy to retrieve data from Facebook API using the functions provided by the **Rfacebook**.
 - The facebook documentation does not provide specific information regarding **Rate Limit**, it specifies that the limit **differs depending on request**. There was no rate limit related issues while working on this report.
 - The **Utils.R** package provides example frameworks to parse the response JSON nested list into an organized data frame.
 - Refer to *MiniProject2* for an example.
 - Facebook API V2.0 have heavy restrictions in terms of accessing personal information such as profile information, posts and comments of a user. Thus it is **not recommended** to scrape the facebook api to search for specific users.
 - It is easy to access public data of facebook through its API. Information such as public pages, posts, comments and groups.
 - Facebook API can be used to provide data for sentiment/text analysis.
 - Can be used to provide extra information regarding locations, events, maybe even tracking public stories.
 - It was observed that many pages, posts does not include *location* tags as part of its content, but the location information is sometimes mentioned as part of the main body content.
 - It is **recommended** to use the access token created through a developer's application. There is no expiration date for this type of access token.
 - The [graph API Explorer] tool is useful for testing API calls to understand structure as well as exploring possible fields and edges.
-

Quick Introduction To Facebook API:

Structure:

The Facebook API is in the form of a *graph*:

Structure	Description
nodes	Objects of facebook API. Nodes are “things” like Users, page, Group, comments etc.
Each	node has its' own ID that which is used to access it via the Graph API.

Structure	Description
edges	The connections that leads from one node to another.
For	example: the cover photo of a user, the comments and posts on a user's timeline.
Edge	s are identified by a name
fields	Information about the nodes, like the names, Id, birthday etc of a user. Fields are identified by a name.

Request Format:

- The *API request* is in the format of a standard HTTP request. Methods like **GET, POST, DELETE, etc** can be used to retrieve and modify information of facebook through the API.
 - Format of a request: `GET graph.facebook.com /{node-id}/{edge-name}?fields = {first-level fields}{Second-level fields}`.
 - An access token is required to access any information.
-

Pagination:

- To manage the amount of information returned per API Call, facebook divides the data using several pagination techniques (Cursors-based, Time-based and Offset-based).
- Most commonly used and the only ones encountered during this report are the cursor based paginations. The information for pagination is located under the *\$paging* subset of the response data and provides a *next* token which is a HTTP GET request for the next page of data and a *previous* token which is a HTTP GET request for the previous page of data.
 - If the provided paging request call returns *NULL*, the end/very beginning of the data is reached.

Rate Limit:

- The type of rate limit dealt with mainly in this report is **Application-Level Rate Limiting**.
- The facebook API states that the limit for the application is **200 calls per hour per user in aggregate**.
- However not all API calls are subjected to the rate limit.
- There is a tool that monitors the application's API calls and it is located under the dashboard of the application. Application. Have to log in using the test user account.

Further Information:

- facebook API Overview
 - Facebook API Request Information
 - Facebook API Rate Limit
 -
-

Rfacebook Package:

- Rfacebook Source Code
- Rfacebook Function Documentation

Summary of Findings For Rfacebook:

Advantages	Disadvantages
Clear documentation.	Assumes that the data being searched is public.
If an error is thrown by the Facebook API, the search stops.	
High level of Abstraction, so no need to worry about the specifics.	Some functions are no longer supported.
Good variety of functions that can provide data for data mining.	Limited fields and edges, does not support all.
The source code provides a structure to parse any arbitrary facebook API response.	Many functions are deprecated due to API changes.

List of Depreciated Functions:

functions	description
getCheckins	deprecated
getFQLS	deprecated
getFriends	only your friends who uses your application
getNetwork	only applicable to users of your application
searchFacebook	deprecated

Setting Up (Libraries and Folder)

```
# creating a folder under current working directory to store any data.
```

```
if(!dir.exists("Rfacebook Exploration"))  
{  
  dir.create("Rfacebook Exploration")  
}  
  
folder_Path = "Rfacebook Exploration"  
  
rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId") ))
```

```
##Loading Libraries needed for the rest of the report:
```

```
library(jsonlite)  
library(knitr)  
library(Rfacebook)  
knitr::opts_chunk$set(error=T,warning=TRUE)  
  
rm(list = ls())
```

Creating Test Account:

- *First name:* Api-Testing
- *Last name:* nrc
- *Email:* NRC.API.Testing@gmail.com
- *Password:* NRCTesting123456!
- *Birthday:* July 1st 1997
- *Gender:* Males

Creating Long Lasting Authorization Token With *fbOAuth* Function:

- Normally, a temporary access token normally has a 2 hour expiration time. A long-lived token(OAuth token) can be used for longer access. The following outlines how to obtain a long lived token:
 - ___The *fbOAuth* function documentation in Rfacebook outlines the steps.
 - Mining Facebook Data Using R & Facebook API! also provides steps to creating a long lived token.

```
app_Id = 1353820787971442
app_Secret = "4841ab73f4f68960ebbf37e5705e2610"

## The following shows the code that calls the fbOAuth function and creates the OAuth token,
##it has been commented once executed and the result is saved in a file
##called "my_OAuth.txt" Please run the following code in case the file does not exist.

# token = fbOAuth(app_id = app_Id,app_secret = app_Secret,extended_permissions = TRUE)
# if(!file.exists("My_OAuth"))
# {
#     file.create("My_OAuth.txt")
#     save(token, file= "My_OAuth.txt")
# }

load("My_OAuth.txt")
rm(list = setdiff(ls(),c("token","FbObjectId") ))
```

- The following pictures indicates where to navigate to set the extended permissions.

Exploring the Package:

Location:

GOAL: To try and explore what the facebook api offer regarding extracting location information.

Exploring Facebook API for Nodes that Contain Location Information:

Albums:

- Contains a *location* field. It is the “The textual location of the album.” **Note, for general data collection, only public albums can be accessed with any access token.**
- The documentation is Album

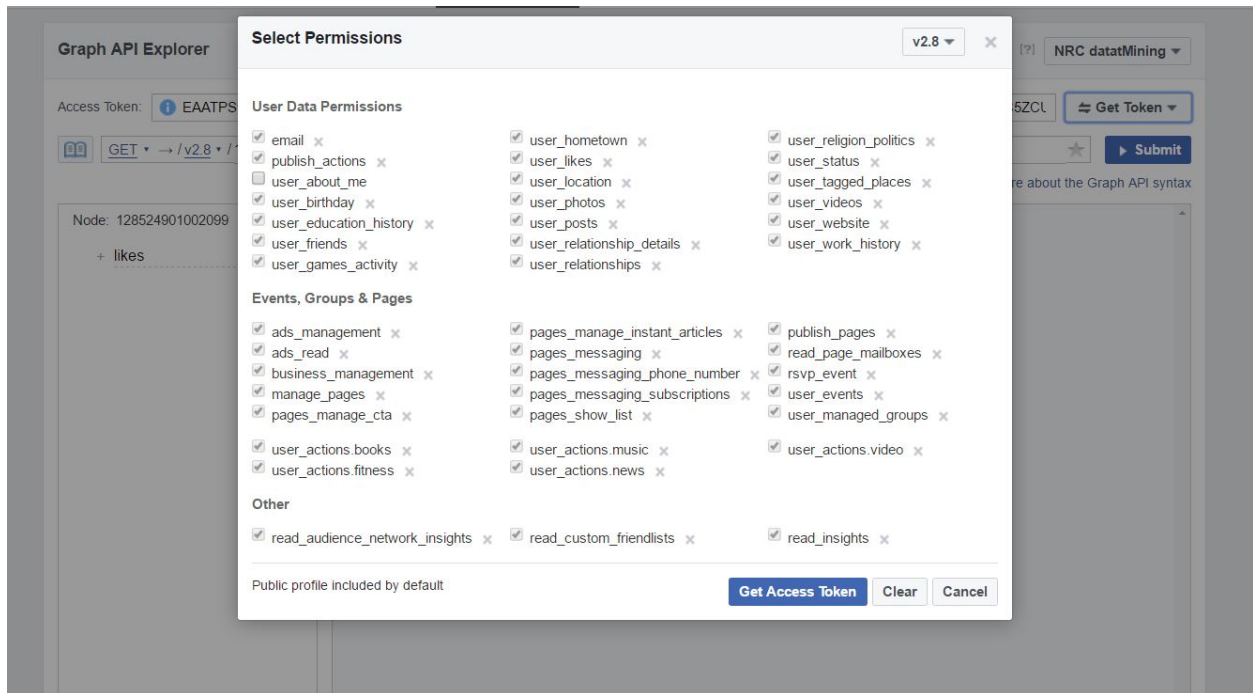


Figure 1: Extended Permissions

- I tried it on the Masterchef Timeline Album + the *location* field returned no information on the facebook graph explorer. + the album id is marked in the URL.`album_id=148022941877464`.
- Often the *location* field returns null.

Comments:

- Does not provide a location field.
- Comments Documentation

Events:


- Events Documentation
- Provides a *place* field which outlines the location of the event.
- Only public events
- Example:
 - URL: <https://www.facebook.com/events/175802919572863/>.
 - Event Id: 175802919572863

Pages:

- *Pages* have a *location* field. This field is automatically returned, if available, as part of the *Rfacebook's searchPage* function.

Posts:

- Might have location associated with the post under the *place* field.
- Page Documentation
- The *getPost* function in *Rfacebook* does not provide the *place* information.


 GET → /v2.8 / 175802919572863?fields=place

Node: 175802919572863

- ☒ place
- + Search for a field

```
{
  "place": {
    "name": "The Oliver House",
    "location": {
      "city": "Middleboro",
      "country": "United States",
      "latitude": 41.90898,
      "longitude": -70.91188,
      "state": "MA",
      "street": "445 Plymouth St",
      "zip": "02346"
    },
    "id": "1686350934986001"
  },
  "id": "175802919572863"
}
```

Figure 2: Return from Facebook API Explorer

 GET → /v2.8 / 10158714687065725?fields=place

Node: 10158714687065725

- ☒ place
- + Search for a field

```
{
  "place": {
    "name": "The White House",
    "location": {
      "city": "Washington",
      "country": "United States",
      "latitude": 38.896844674148,
      "longitude": -77.036604881287,
      "state": "DC",
      "street": "1600 Pennsylvania Avenue",
      "zip": "20500"
    },
    "id": "1191441824276882"
  },
  "id": "10158714687065725"
}
```

Figure 3: Post API Explorer Return

Photos:

- Also has a *place* field to indicate any location information.
 - Photos Documentation
-

callAPI Function:

- The “callAPI” function delegates the task of retrieving information from the Facebook API to the *GET()* method in the *httr* package and uses the *rjson* package to parse the returned json formatted response.

Trying to extract my profile information:

```
##Extracting my personal information
my_Data = callAPI("https://graph.facebook.com/me", token)
## Extracting my ID, this can be passed on as the ids for other functions
my_id = my_Data$id
print(my_Data)

## $name
## [1] "Api-Testing Nrc"
##
## $id
## [1] "128524901002099"

##Adding fields to the URL to gain specific information (birthday, age_range)
#no spaces anywhere within the URL should be included
my_Specific_Information_Request = "https://graph.facebook.com/me?fields=birthday,age_range"
my_Specific_Data = callAPI(my_Specific_Information_Request,token)

print("detailed Personal Data")

## [1] "detailed Personal Data"
print(my_Specific_Data)

## $birthday
## [1] "07/01/1997"
##
## $age_range
## $age_range$max
## [1] 20
##
## $age_range$min
## [1] 18
##
##
## $id
## [1] "128524901002099"

rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId") ))
```

- NOTE: Since the APP is made by the test account, the application already had permissions to the test account’s personal information. If this was applied to a generic user ID,

only the name and the ID would be returned.

Extracting information about a public facebook page:

```
##Extracting facebook page basic information.
##The url of the page is: https://www.facebook.com/harrypottermovie/

harry_Potter_Request = "https://graph.facebook.com/https://www.facebook.com/harrypottermovie/"
Harry_Potter_Page = callAPI(harry_Potter_Request,token=token)
print(Harry_Potter_Page)

## $name
## [1] "Harry Potter"
##
## $id
## [1] "156794164312"

##Trying to extract other fields as well using the callAPI function
harry_Potter_More = paste("https://graph.facebook.com",Harry_Potter_Page$id
                          ,"posts?fields=message",sep="/")
information = callAPI(harry_Potter_More,token=token)

names(information)

## [1] "data"    "paging"

length(information$data)

## [1] 25

rm(list = setdiff(ls(),c("token","FbObjectId") ))
```

Extracting information about a comment.

- Note: You can not paste the URL directly into the GET query for the callAPI function. The comment ID is needed.
 - The comment ID is imbedded as part of the URL.
 - The comment Id is: postID_comment_id

```
comment_URL = "https://www.facebook.com/harrypottermovie/videos/10155070549019313
/?comment_id=10155070752519313&comment_tracking=%7B%22tn%22%3A%22R0%22%7D"

comment_Id = "10155070549019313_10155070752519313"
comment_Query = paste("https://graph.facebook.com/",comment_Id,sep="")

comment_Data = callAPI(comment_Query,token=token)
comment_Data

## $created_time
## [1] "2017-03-03T19:25:16+0000"
##
## $from
## $from$name
## [1] "Suzanne Marie King"
```

```
##
## $from$id
## [1] "10209935847807594"
##
##
## $message
## [1] "Professor... I'm sorry! Your bird... It just caught fire!\n\nPitty you had to see him on a burn."
##
## $id
## [1] "10155070549019313_10155070752519313"

rm(list = setdiff(ls(),c("token","FbObjectId") ))
```

Developing a function to extract the “ID” of facebook object through using the URL of the object:

```
## ONLY VALID for SIMPLE Webaddresses with no id in the URL, ex: www.facebook.com/harrypotter

FbObjectId = function(object_URL,token)
{
  beginning_Index = 1
  # Checking the formatting of the "object_URL"
  if(class(object_URL) != "character"){
    warning("Object_URL is not a string")
  }else if( grep(pattern = "(http|https)://www.facebook.com/(.*)*"
    , x=object_URL) != beginning_Index ) {

    warning("Invalid URL")
  }

  #formatting and checking the complete URL request to the facebook Graph API
  complete_URL = paste("https://graph.facebook.com",object_URL,sep = "/")
  #print(complete_URL)

  #Extracting the ID of the facebook object
  url_Data = Rfacebook::callAPI(complete_URL,token)
  url_id = url_Data$id

  # If the Id is not numeric, something is wrong.
  if(length( grep(pattern="www.facebook.com",x=url_id))!=0 ) {
    warning("Error In Returned ID: Not numeric values.
      URL likely not supported.")
  }

  url_id
}
```

- Test cases:

```
## For URLs that yield no ids, the facebook would return a version of the
##URL address as the object ID.(just discovered)

## A URL of a post on the public facebook photo of Harry Potter
##(This should also return an error)
```

```
test1 = "https://www.facebook.com/harrypottermovie/photos/
a.422515109312.180796.156794164312/10155042492264313"
original_Return = "https://www.facebook.com/harrypottermovie/photos/
a.422515109312.180796.156794164312/10155042492264313:
https://www.facebook.com/harrypottermovie/photos/
a.422515109312.180796.156794164312/10155042492264313"
```

```
FbObjectId(test1,token)
```

```
## Error in curl::curl_fetch_memory(url, handle = handle): URL using bad/illegal format or missing URL
```

```
## A comment on a public page (This should return an error)
test2 = "https://www.facebook.com/harrypottermovie/photos
/a.422515109312.180796.156794164312/10155042492264313/
?type=3&comment_id=10155042702764313&
comment_tracking=%7B%22tn%22%3A%22R4%22%7D"
FbObjectId(test2,token)
```

```
## Error in curl::curl_fetch_memory(url, handle = handle): URL using bad/illegal format or missing URL
```

```
## A URL of a public facebook page
test3 = "https://www.facebook.com/harrypottermovie"
FbObjectId(test3,token)
```

```
## [1] "156794164312"
```

```
## A URL of a person's timeline (not my own)
test4 = "https://www.facebook.com/FSXAC"
FbObjectId(test4,token)
```

```
## Warning in FbObjectId(test4, token): Error In Returned ID: Not numeric values.
## URL likely not supported.
```

```
## [1] "https://www.facebook.com/FSXAC"
```

```
rm(list = setdiff(ls(),c("token","FbObjectId")))
```

Conclusion:

- the *callAPI* function allows you to call API using the GET methods using regular Facebook Query format. The weakness is that it does not format the returned nested list into a data frame.
- **The strength is that it can be used to retrieve information regarding any arbitrary request.**

Exploring *getPage*, *getPost*, *getCommentReplies*:

- **GOAL:** To extract information regarding comments on a public facebook webpage

1: Starting with getting the post_Ids (mainly post Ids) using *getPage*

```
## the getPage function returns a matrix, and one of the columns is the "postId"
BBC_URL = "https://www.facebook.com/bbcnews"
```

```
BBC_Id = FbObjectId(BBC_URL,token)
print(BBC_Id)
```

```

## [1] "228735667216"
page_Information = getPage(BBC_Id,token,n=2)

## 2 posts
class(page_Information)

## [1] "data.frame"
dim(page_Information)

## [1] 2 11
names(page_Information)

## [1] "from_id"      "from_name"    "message"      "created_time"
## [5] "type"         "link"         "id"           "story"
## [9] "likes_count"  "comments_count" "shares_count"

post_Ids = page_Information$id

2: Getting information about each post using _getPost_
(mainly interested in comment ids)

## trying to see getPost can be vectorized, it appears to be working through the use of lapply

## The return type is a list of posts and each element within this represents
##the information for a single post. Each "post" is a list composed of
##three sections: "post","likes","comments".

post_Information = lapply(post_Ids,getPost,n=100,token=token)

names(post_Information[[2]])

## [1] "post"      "likes"     "comments"

3: Now that I have obtained a list of comments, I can use the _getCommentReplies_
to further investigate each reply to a comment.

## I have extracted the list of comments_Id from the post_information
## the "comments" is a data frame where the column "id" represents the list of comments
## to the posts and each element within this "id" is another list of comment_ids.
## I have printed out such a comment Id below (note, I had to subset the list twice).

comments = unlist( lapply(post_Information, function(x) {x["comments"]}), recursive = F)
comments_Id = unlist(comments[[1]]["id"])
comments_Id[[1]]

## [1] "1296812350402975_1296817243735819"

#Extracting the comment replies to each posts' comments

## First I tested the getCommentReplies with a single comment_Id
single_Comment_Reply2 = getCommentReplies(comments_Id[[1]],token=token)
names(single_Comment_Reply2)

## [1] "comment" "replies"

##Now for all of the comment Ids.

```

```
comment_Replies = lapply(comments_Id, function(x) getCommentReplies(x,token = token))

length(comment_Replies)
```

```
## [1] 100
```

What if the posts are pictures?

- The information regarding attachments (such as photos and videos) can be accessed using the *attachments* edge. However the Rfacebook package does not seem to have explicit functions that support the extraction of information regarding attachments.
- One way to extract this information is to use the *callAPI* method with the *attachments* edge specified as part of the query.

Exploring *searchGroup* & *getGroup* functions:

- *GOAL: To extract information regarding a public facebook group: couponboutique using the getGroup function*

1: Suppose the ID is not known, or not certain. *searchGroup()* can be used to extract the group_Id:

```
group_Name = URLencode("The Coupon Boutique")
group_Id = searchGroup(group_Name,token=token)
print(group_Id)
```

```
##               name privacy          id
## 1 The Coupon Boutique - Public Group    OPEN 1696255650601823
## 2               The Coupon Boutique  CLOSED 1504395996479976
## 3 The Coupon Boutique - Public Group    OPEN 1580411485549988
```

+ An attempted solution found on [Stackoverflow](https://github.com/pablobarbera/Rfacebook/issues/7). Try to use `_URLencode_` function to encode the name to in URL format.

```
group_Name_URL = URLencode("The Coupon Boutique")
group_Id2 = searchGroup(group_Name_URL,token=token)
class(group_Id2)
```

```
## [1] "data.frame"

print(group_Id2)
```

```
##               name privacy          id
## 1 The Coupon Boutique - Public Group    OPEN 1696255650601823
## 2               The Coupon Boutique  CLOSED 1504395996479976
## 3 The Coupon Boutique - Public Group    OPEN 1580411485549988
```

+ This solution appears to be working. Upon reading the documentation, it did specify that the name needs to be in "URL"

2: Extracting information using *getGroups*

```
start_Time = as.numeric(as.POSIXct("2016-12-01"))
id = group_Id2$id[[1]]
##I have verified that this id is linked with the URL
```

```

#for the group I have shown at the beginning of this section.
print(id)

## [1] "1696255650601823"
##NOTE: the "groupId" is different from the "pageId"
##where the group is posting and hosting.

#NOTE: the default number of posts to return is 25.
group_posts = getGroup(id,token=token)

## 25 posts

class(group_posts)

## [1] "data.frame"

names(group_posts)

## [1] "from_id"      "from_name"    "message"      "created_time"
## [5] "type"        "link"         "id"           "story"
## [9] "likes_count"  "comments_count" "shares_count"

rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId") ))

```

Exploring *searchPage*:

- This method works as shown in the documentation.

```

keyword ="Apples"
data = searchPages(keyword,token, n=100)

## 100 pages

## Warning in vect[notnulls] <- unlist(lapply(lst, function(x) x[[field]])):
## number of items to replace is not a multiple of replacement length

class(data)

## [1] "data.frame"

names(data)

## [1] "id"           "about"        "category"
## [4] "description"  "general_info" "likes"
## [7] "link"        "city"         "state"
## [10] "country"     "latitude"     "longitude"
## [13] "name"        "talking_about_count" "username"
## [16] "website"

tail(data,n=2)

##           id
## 99 438628792848540
## 100 205141136166746
##
## 99 U-Pick Apple Orchard in Las Cruces! The best tasting apples in NM and home-made apple pie oozing
## 100                                     100% All Natural, Local Produce Applesauce

```

```
## category
## 99 Fruit & Vegetable Store
## 100 Food & Beverage Company
##
## 99 Fresh, orchard ripened apples hanging around
## 100 Holmes Made Foods LLC and Holmes Mouthwatering Applesauce were established in Dec. 2008 by 15 years
## general_info likes link
## 99 <NA> The Jazz Cafe https://www.facebook.com/FarmFreshApplePie/
## 100 <NA> 50097478644 https://www.facebook.com/HolmesApplesauce/
## city state country latitude longitude
## 99 Las Cruces NM United States 32.32366 -106.84714
## 100 Shaker Heights OH United States 41.48885 -81.52386
## name
## 99 U-Pick Mesilla Valley Apples and Farm Fresh Apple Pie
## 100 Holmes Mouthwatering Applesauce
## talking_about_count username website
## 99 28 FarmFreshApplePie http://www.appleofjoy.com
## 100 25 HolmesApplesauce holmesapplesauce.com

rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId")))

```

Trying *getLikes*:

- *GOAL:* The documentation stated that it could provide information about a user or a page's likes given the id. I want to see if it works with user Ids that have been granted permissions to my application.
- Does not work on users but work on pages.

```
rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId")))
user_Id = 313298645491893

```

```
# The following produces facebook error message:
# likes = getLikes(user_Id, token=token)

```

```
rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId")))

```

```
##trying to see if I can get information about the test account's likes.
## It would appear that this function is not useable for my own personal account.
##It also returns a facebook error. So this means
my_Id = 128648664323056
# likes = getLikes(my_Id, token=token)
rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId")))

```

- Result, a random user who has been given permission to your application will result a facebook error. The same for the app owner (test account)

```
page_Id = 156794164312
page_Likes = getLikes(page_Id, token=token)

```

```
##This works as it should.

```

Search API:

- Allows public searches on public information.
- Search Documentation
- **This API provides the same functionality as facebook's Search Bar.**
- Query example: `https://graph.facebook.com/search?q={string to search for}&type={type name}&{fields of type}`
- Allows:
 - users
 - page
 - place-topic
 - event
 - group
 - places

Mini Project1:

Description:

1: To scrape the facebook API for all web pages that contains a series of keywords. Up to 300 pages, then graph distribution based on location.

Code:

```
rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId","ExtractList",
                        "RoughDF","MinRow","UnravelList") ))
```

```
#Loading libraries
```

```
library(Rfacebook)
```

```
# setting up authorization token
```

```
## refer to above section for details
```

```
load("My_OAuth.txt")
```

```
#Gathering webpage Locations:
```

```
keywords = c("flower")
```

```
##Note: there might not be the specified number of pages
```

```
##that will be returned as the result.
```

```
webpage_Id= searchPages(keywords, token=token,n=2000)
```

```
## 200 pages
```

```
## Warning in vect[notnulls] <- unlist(lapply(lst, function(x) x[[field]])):
```

```
## number of items to replace is not a multiple of replacement length
```

```
## 400
```

```
## Warning in vect[notnulls] <- unlist(lapply(lst, function(x) x[[field]])):
```

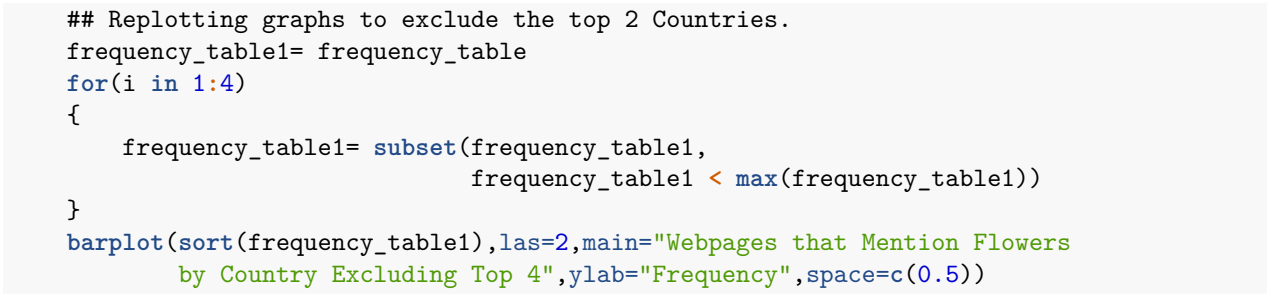
```
## number of items to replace is not a multiple of replacement length
```

```
## 516
```

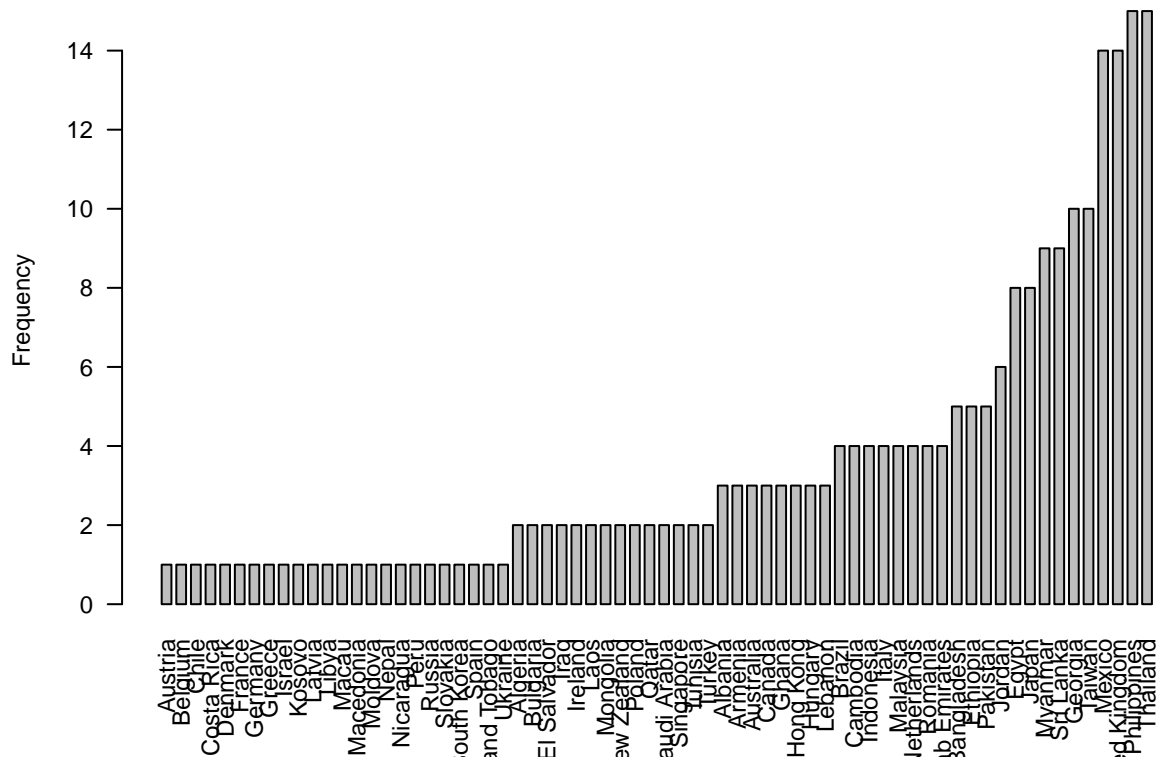


```
webpage_Location = webpage_Id$country
webpage_Location[is.na(webpage_Location)] = "Not Provided"

#presenting information:
frequency_table = table(webpage_Location)
par(cex=0.75)
barplot(sort(frequency_table), las=2, main="Webpages that Mention Flowers by Country",
        , ylab="Frequency", space=c(0.5))
```



Webpages that Mention Flowers by Country Excluding Top 4



MiniProject 2:

Description:

_To find the place most posted on a travel website from 500 posts. The main point is trying to illustrate the structure of the response and how I went about organizing it into a dataframe.

* note: sometimes there is no location tag, could consider searching for it in the text of the comment. * note: The *getPost* function

Code:

Gathering RawData:

```
website_URL = "https://www.facebook.com/thousandamazingplacesonearth/"

website_Id = FbObjectId(website_URL, token)

#Making the query. I first tested this out on the Graph API Explorer
## Note: The limit for number of posts
query= paste("https://graph.facebook.com",website_Id,
             "?fields=posts.limit(100){place}",sep="/")
```

```

#Pulling RawData
raw_Data= callAPI(query,token)

#Formatting the raw Data:
## I looked at the returned format again at the graph API explorer.

```

Planning:

Query Response Structure:

Paging Response Structure:

Desired Organized Output Structure:

Desired Final DF:

Pesudocode For Parsing:

____NOTE: This process is based on similar procedure as the source code of the Rfacebook Package. I do not take credit for the idea behind this parsing. I merely studied the structure of the Rfacebook source code and tried to adapts some parts of it. The functions that I studied are the *getPage* function, the *pageToDF* & *UnlistWithNA* functions under the *Utils.R* package

I strongly recommand to use the function *UnlistWithNA*(unavailable for direct access under the Rfacebook Package) under the *Utils.R* package for parsing of queries not supported by the methods of the Rfacebook. *UnlistWithNa* function includes pre-made parsing function for many possible fields. Furthermore, the general structure of this function can be adapted to parse JSON formatted information.____

- 1: Navigate to the *data* section of the *roughData*, through format like: *rough_Data\$posts\$data*.
- 2: At the *data* section, the underlying strcture is a list of elements. Each element containing it's own list of information. + In order to subset the list, the general structure of `list[element][[field_Lvl_1]][[field_Lvl_2]]`. for example. The *place* field has a nested field of *location*. To extract it from the *data* section. Something like: `data[[1]][["place"]][["location"]]` can be used. + The general format can be applied to multiple levels of nesting if the data is organized correctly.
- 3: Since some of the *posts* won't have *location* (or other fields), when trying to access them, a *NULL* element will be the response. It will be needed to be replaced with *NA*. The end result is a list where each element corresponds to the post information.
- 4: By going through step 2 & 3 for all the desired fields (note they should all be the same length), should be in their own lists. Then a data frame can be merged from these lists. **The key is that the fields to be coerced into a data frame all need to be the same length. Which is one of the main difficulties for coming up with an automated program to format data frames.**
- 5: The *paging* URLs are already in the correct query format and just need to be extracted using *callAPI* function. Repeat steps 1 to 4 for the data that is pointed to by the *paging* information.
- 6: Finally merge all the data frames of the initial query and the subsequent paging matrix can be merged into one big matrix to be the final output.

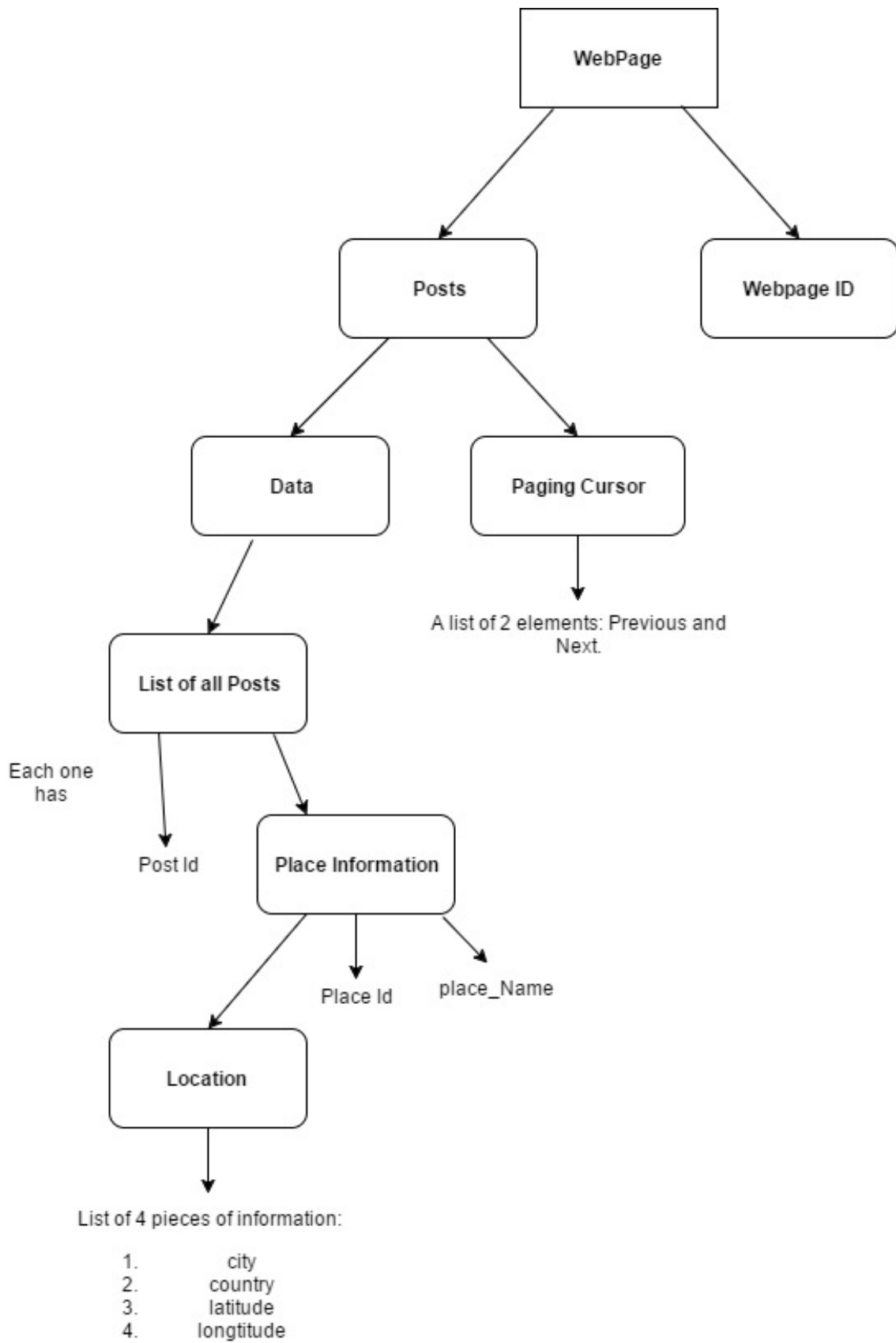


Figure 4:
20

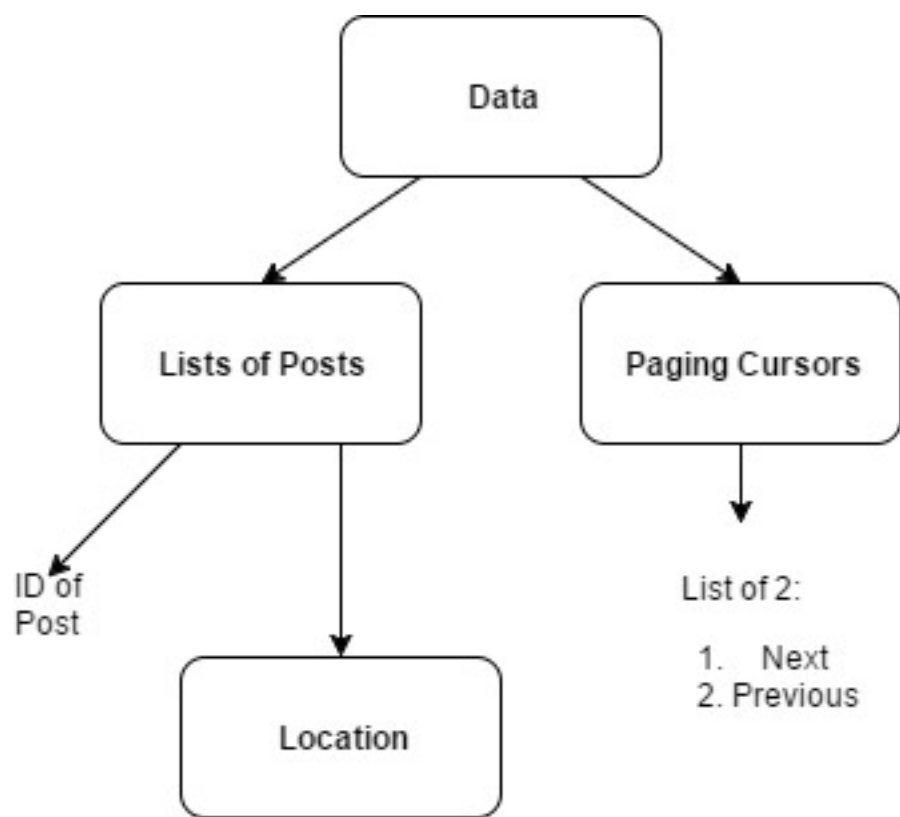


Figure 5:

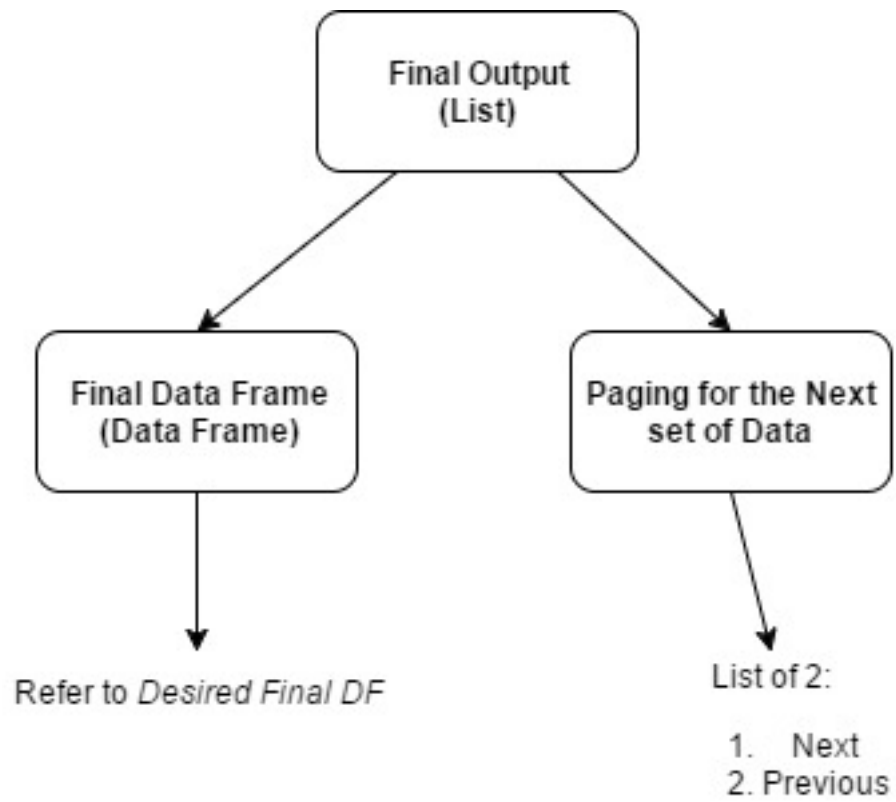


Figure 6:

Post_Id	Place_Id	Place_Name	City	Country	Latitude	Longitude
data	data	data	data	data	data	data

Figure 7:

Parsing and Processing information.

PERHAPS ANOTHER WAY TO GO ABOUT IT using tidyJSON

```
num_Posts = 500
post_Limit = 100

#step 1 of described procedure: navigating to list of posts data.
query_Data = raw_Data$posts$data

#step 2 and 3; extracting information and formatting.
#This process is mostly done by the UnlistWithNA function.
#I will be producing a simple replicatation of it.

UnlistWithNA_Copy = function(field, list)
{
  ## produce a list of NAs.
  complete = rep(NA, length(list))

  if(length(field)==1)
  {
    ## produces a vector indicating which indices are null.
    ##This is used to indicate which values to replace with NA.
    notNull = unlist(lapply(list,function(x) !is.null(x[[field]])))

    ## Combine the information within the fields as well as the NA_List to
    ##form the complete and correct list regarding the field.
    ##NOTE: differences between [[]] selects the content and [] selects the corresponding
    ##container

    complete[notNull]= unlist(lapply(list, function(x) x=x[[field]]))
  }

  if(length(field)==2)
  {
    ##similar logic as the above.
    notNull=unlist(lapply(list,function(x) !
                        is.null(x[[field[1]]][[field[2]]])))
    complete[notNull]= unlist(lapply(list,function(x)
                        x=x[[field[1]]][[field[2]]]))
  }

  if (length(field)==3){
    notnull <- unlist(lapply(list, function(x)
      !is.null(x[[field[1]]][[field[2]]][[field[3]]])))

    complete[notnull] <- unlist(lapply(list[notnull],
      function(x) x = x[[field[1]]][[field[2]]][[field[3]]] ))
  }

  ##Note: for lists, data.frame corece it as a row. While for vectors,
  ##each vector is a column. Thus in order for each field to be its own column,
  ##the list needs to be a vector
}
```

```

    return(as.vector(complete))

}

# Takes a list of data and extract the fields of interest and organize into a data frame.
OrganizingDF = function(list)
{
    # Constructing lists corresponding to the desired fields using the above
    # function.

    ##Similar Warning appeared that stated multiple of replacement length when
    ##the package is called originally.

    post_Id=UnlistWithNA_Copy(c("id"), list)
    place_Id=UnlistWithNA_Copy(c("place","id"),list)
    place_Name=UnlistWithNA_Copy(c("place","name"),list)
    city=UnlistWithNA_Copy(c("place","location","city"),list)
    country=UnlistWithNA_Copy(c("place","location","country"),list)
    latitude=UnlistWithNA_Copy(c("place","location","latitude"),list)
    longitude=UnlistWithNA_Copy(c("place","location","longitude"),list)
    street=UnlistWithNA_Copy(c("place","location","street"),list)

    df = data.frame(post_Id, place_Id,place_Name,city,country,
                    latitude,longitude,street,stringsAsFactors = F)

    return(df)
}

```

```

##Continuing on with Data Processing: The query information is organized into a data frame.

query_DF = OrganizingDF(query_Data)

#Moving on to organizing the paging informations. This can be done in a loop.
    ## Depending the number of posts desired, a custom function can be made
##to calculate how many posts are requested.
    ## By modifying the ".limit" modifier in the paging cursor,
##you can specify the number of posts to return.
    ##In this case, the maximum is 100, so i will need to loop through the cursors 4 times.

    ## The format of the paging token location varies: For the initial query,
    ##it is under raw_Data$posts$paging[["next"]]. For any subsequent navigations,
    ##it is under raw_Data$paging[["next"]]

    paging_Url = raw_Data$posts$paging[["next"]]
    for(i in (1:4))
    {
        #formatting paging_Url to extract 100 posts (post limit) eaech time
        paging_Url = gsub(pattern="place&limit=25", replacement =
                        paste("place&limit=",post_Limit,sep=""), x=paging_Url)

        # Extracting the raw_Data from the new paging_Url

```



```

raw_Data_Paging = callAPI(paging_Url,token)

#formatting the data using the above functions into a data frame.
response_DF = OrganizingDF(raw_Data_Paging$data)

#Formating final output:
query_DF = rbind(query_DF,response_DF)

#Navigating to next paging cursor
paging_Url = raw_Data_Paging$paging[["next"]]
}

dim(query_DF)

```

```
## [1] 500    8
```

```

# Now the matrix is organized, time to get it into the final output format.

```

```
final_Out = list(query_DF, raw_Data_Paging$paging)
```

```

#finding most visited location:

```

```

location = query_DF$country
max_Location = max(location, na.rm=T)
print(max_Location)

```

```
## [1] "Zimbabwe"
```