atividade data engineer greenpeace

2023-04-24

Pacotes Necessários

```
library(dplyr)
```

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

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

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

```
library(bigrquery)
library(xtable)
library(httr)
library(jsonlite)
library(googledrive)
```

Exercício 1

1.

```
# Auth in google account
bq_auth()
```

```
## ! Using an auto-discovered, cached token.

##   To suppress this message, modify your code or options to clearly consent to
##   the use of a cached token.

##   See gargle's "Non-interactive auth" vignette for more details:

##   <https://gargle.r-lib.org/articles/non-interactive-auth.html>

## i The bigrquery package is using a cached token for 'pedro.h.castr8@gmail.com'.
```

```r
# Store the project ID
projectid = "gp-data-engineer"

# Set your query
sql <- "SELECT *
FROM `bigquery-public-data.fda_food.food_events`
WHERE consumer_gender = 'Male'
ORDER BY date_created DESC"

# Run the query; this returns a bq_table object that you can query further
tb <- bq_project_query(projectid, sql)
```

```
## Auto-refreshing stale OAuth token.
```

```r
# Store the first 100 rows of the data in a tibble
sample <-bq_table_download(tb, n_max = 100)
```

```r


sample
```

Exemplo que a tabela foi importada com sucesso limitada a 100 observações via lazy query

```
## # A tibble: 100 x 12
##    report_number   reactions outcomes products_brand_name products_industry_code
##    <chr>           <chr>     <chr>    <chr>               <chr>
##  1 2022-CFS-014599 Chemical~ Other S~ FLAGSTONE FOODS PR~ 23
##  2 2022-CFS-014640 Foreign ~ Other S~ CENTRUM SILVER MEN~ 54
##  3 2022-CFS-014598 Abdomina~ Other S~ GENERAL MILLS HONE~ 5
##  4 2022-CFS-014598 Abdomina~ Other S~ DAILY MULTIVITAMIN  54
##  5 2022-CFS-014602 Diarrhoe~ Hospita~ ENFAMIL PROSOBEE S~ 40P
##  6 2022-CFS-014604 Overdose  Hospita~ THC EDIBLE GUMMY    33
##  7 2022-CFS-014605 Chest pa~ Hospita~ EXHALE WELLNESS DE~ 33
##  8 2022-CFS-014605 Chest pa~ Hospita~ FISH OIL            54
##  9 2022-CFS-014605 Chest pa~ Hospita~ MULTIVITAMINS       54
## 10 2022-CFS-014605 Chest pa~ Hospita~ TUMERIC             54
## # i 90 more rows
## # i 7 more variables: products_role <chr>, products_industry_name <chr>,
## #   date_created <date>, date_started <date>, consumer_gender <chr>,
## #   consumer_age <dbl>, consumer_age_unit <chr>
```

2.1

```r
sql_A <- "SELECT
  reaction_type,
  COUNT(*) as reaction_count
FROM (
  SELECT
    SPLIT(reactions, ',') as reaction_types
  FROM `bigquery-public-data.fda_food.food_events`
), UNNEST(reaction_types) as reaction_type
GROUP BY reaction_type
ORDER BY reaction_count DESC;
"
# Run the query; this returns a bq_table object that you can query further
tb_A <- bq_project_query(projectid, sql_A)

# Lazy query to download query informations
sample_A <-bq_table_download(tb_A, n_max = 1)

sample_A
```

```
## # A tibble: 1 x 2
##   reaction_type  reaction_count
##   <chr>                   <int>
## 1 Ovarian cancer          18615
```

2.2

```
# letra B

sql_B <- "SELECT
  products_industry_name,
  COUNT(*) AS death_count
FROM
  `bigquery-public-data.fda_food.food_events`
WHERE
  products_industry_name IS NOT NULL
  AND LOWER(reactions) LIKE '%death%'
GROUP BY
  products_industry_name
ORDER BY
  death_count DESC"
# Run the query; this returns a bq_table object that you can query further
tb_B <- bq_project_query(projectid, sql_B)

# Store the first 100 rows of the data in a tibble
sample_B <-bq_table_download(tb_B, n_max = 1)

sample_B
```

```
## # A tibble: 1 x 2
##   products_industry_name death_count
##   <chr>                        <int>
## 1 Cosmetics                    17554
```

2.3

```
# letra C

sql_C <- "SELECT
  reactions,
  COUNT(*) AS reaction_count
FROM
  (
    SELECT
      products_industry_name,
      SPLIT(reactions, ',') AS reactionss
    FROM
      `bigquery-public-data.fda_food.food_events`
    WHERE
      lower(products_industry_name) LIKE '%cosmetics%'
      AND consumer_age BETWEEN 18 AND 25
  ), UNNEST(reactionss) AS reactions
GROUP BY
  reactions
ORDER BY
  reaction_count DESC"
# Run the query; this returns a bq_table object that you can query further
tb_C <- bq_project_query(projectid, sql_C)

# Store the first 100 rows of the data in a tibble
sample_C <-bq_table_download(tb_C, n_max = 3)

sample_C
```

```
## # A tibble: 3 x 2
##   reactions        reaction_count
##   <chr>                     <int>
## 1 "Alopecia"                  398
## 2 "Ovarian cancer"            289
## 3 " Pruritus"                 183
```

Exercício 2

Passo 1 - gerar uma tabela com 1000 usuários brasileiros

```
# # Function for user in API
# get_random_user <- function() {
#   # Make the API request
#   response <- httr::GET("https://randomuser.me/api/?nat=BR")
#   # Parse the JSON response
#   json <- jsonlite::fromJSON(httr::content(response, "text"), simplifyDataFrame = TRUE)
#   # Extract the user data
#   user <- json$results
#   return(user)
# }
#
# # Single user
# users <- list()
# for (i in 1:1000) {
#   users[[i]] <- get_random_user()
# }
#
# # Convert to df
# users_df <- dplyr::bind_rows(users)
```

Exercício 3

3.1

- Para garantir que o usuário "gp_user" tenha acesso a tabela "press_data" hospedada no AWS redshift é necessário ter uma conta com privilégios de administrador e rodar o seguinte comando no console:

```
GRANT SELECT ON TABLE press_data TO gp_user;
```

O comando GRANT SELECT garante que esta tabela press_data poderá ser consultada pelo usuário gp_user. Aqui está a documentação em que a aws redshift informa sobre permissões a usuários além de alguns exemplos

3.2

- Quando a tabela é reescrita tendo feito o comando DROP é necessário garantir que o schema sempre tenha as permissões anteriores. Herdando sempre que for feito como no código abaixo:

```
ALTER DEFAULT PRIVILEGES IN SCHEMA schema_press_data
GRANT SELECT ON TABLES TO gp_user;

GRANT SELECT ON TABLE schema_press_data.press_data TO gp_user;
```

A primeira linha de comando garante que os privilégios para o schema "schema_press_data" onde está a tabela press_data sejam herdados. E para garantir novamente a tabela "press_data" é selecionada novamente. Os exemplos buscados seguem os padrões da documentação sobre privilégios

3.3

- Para mudar o proprietário de uma tabela deve-se usar o comando

```
ALTER TABLE press_data OWNER TO gp_new_user;
```

Foi usada a mesma tabela que nos exemplos anteriores. Uma observação que deve ser feita é que somente usuários que tenham os privilégios necessários podem alterar o acesso de cada usuário, como por exemplo superusuários. Os exemplos de como realizar a alteração estão neste link