reddit_analysis

January 10, 2021

1 About Dataset

Reddit is a discussion website which users can post images and text in a subforum called subreddit which users can discuss about shared contents in comment section. This dataset contains 05/2015 comment submissions from reddit users with 54.504.410 rows and 22 columns.

I got my data from kaggle unfornutely this dataset is too big to run on kaggle so I needed to download it. > https://www.kaggle.com/reddit/reddit-comments-may-2015/notebooks

If you want a JSON format of this data you can download it from: https://files.pushshift.io/reddit/comments/

2 Accessing data from sqlite and cleaning it

Used this sqlite query to clean the dataset before extracting it to csv because it caused problems while trying to import the data

I didn't import authorflaircss_class field because it is not important for our analysis

```
create table reddit_2015_05 as
select
rd.created_utc,
rd.ups,
rd.subreddit_id,
rd.link_id,
rd.name,
rd.score_hidden,
replace(
    replace(
        replace(
                replace(
                    replace(
                        replace(
                            replace(rd.author_flair_text,'\','')
                        ,'*','')
                    ,'#','')
                , X'0A', ' ')
        ,char(13),' ')
    ,';','')
,'"','') as author_flair_text,
```

```
rd.subreddit,
rd.id,
rd.removal_reason,
rd.gilded,
rd.downs,
rd.archived,
rd.author,
rd.score,
rd.retrieved_on,
replace(
    replace(
        replace(
                replace(
                    replace(
                        replace(
                            replace(rd.body,'\','')
                        ,'*','')
                    ,'#','')
                , X'0A', ' ')
        ,char(13),' ')
    ,';','')
,'"','') as body,
rd.distinguished,
rd.edited,
rd.controversiality,
rd.parent_id
from may2015 rd;
```

3 Splitting csv data to make it ready for import

I needed to split my csv file so I can import it to PostgreSQL because PostgreSQL copy command doesn't support files bigger than 4GB

I used csvsplitter from erdconcepts

Opened up cmd and inserted these lines;

```
cd C:\data\reddit\csvsplitter
```

```
CSVSplitter.exe filename="C:\data\reddit\reddit_2015_05.csv" rowcount=5000000
```

It spliced my csv to 11 files ranging from 1.2GB to 1.5GB

4 Creating table in PostgreSQL to import our dataset

I created my PostgreSQL table with this query

```
CREATE TABLE "ODS"."EXT_REDDIT_COMMENTS"
(
```

```
    created_utc integer,
    ups integer,
    subreddit_id text COLLATE pg_catalog."default",
    link_id text COLLATE pg_catalog."default",
    name text COLLATE pg_catalog."default",
    score_hidden text COLLATE pg_catalog."default",
    author_flair_text text COLLATE pg_catalog."default",
    subreddit text COLLATE pg_catalog."default",
    id text COLLATE pg_catalog."default",
    removal_reason text COLLATE pg_catalog."default",
    gilded integer,
    downs integer,
    archived text COLLATE pg_catalog."default",
    author text COLLATE pg_catalog."default",
    score integer,
    retrieved_on integer,
    body text COLLATE pg_catalog."default",
    distinguished text COLLATE pg_catalog."default",
    edited text COLLATE pg_catalog."default",
    controversiality integer,
    parent_id text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE "ODS"."EXT_REDDIT_COMMENTS"
    OWNER to postgres;
```

5 Importing dataset

Then used PostgreSQL copy command to import my data;

```
SET STATEMENT_TIMEOUT TO 3000000;

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-000.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-001.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-002.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-003.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-004.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-005.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-006.CSV' DELIMITER ';';
```

```
COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-007.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-008.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-009.CSV' DELIMITER ';';

COPY "ODS"."EXT_REDDIT_COMMENTS" FROM 'C:/data/reddit/REDDIT_2015_05-010.CSV' DELIMITER ';';

COMMIT;
```

6 Analyzing our data

Original dataset is too big to handle(54.504.410 rows with 33.3GB size) maybe we should check if it is possible to reduce our data while not affecting our analysis

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2;
```

This query reduces our data to 54.333.604 rows while removing comments like 'OK' which is not meaningful on its own.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%');
```

This would remove 958 bot comments with comment author names contains "-bot-" or "*bot*", it is not that a huge decrease.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND NOT (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''','''))) LIKE '%im a bot%';
```

We could also filter comments with "I'm a bot" text, this also decreases dataset with 24.918 rows.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
```

```
AND NOT (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''','')) LIKE '%im a bot%')
AND ERS.BODY <> '[deleted]';
```

This query removes deleted comments which is 3.138.587 rows.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND NOT (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''','')) LIKE '%im a bot%')
AND ERS.BODY <> '[deleted]'
AND LENGTH(ERS.REMOVAL_REASON) = 0;
```

We should also remove removed comments which is replaced by removal reason instead of original comments.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND NOT (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''','')) LIKE '%im a bot%')
AND ERS.BODY <> '[deleted]'
AND LENGTH(ERS.removal_reason) = 0
AND ERS.BODY LIKE '% %';
```

We should remove single word comments(1.885.966 rows) because they are not important for our analysis.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND NOT (LOWER(ERS.AUTHOR) LIKE '%\_bot\_%'
OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''','')) LIKE '%im a bot%')
AND ERS.BODY <> '[deleted]'
AND LENGTH(ERS.removal_reason) = 0
AND ERS.BODY LIKE '% %'
AND ERS.AUTHOR <> 'AutoModerator';
```

With this query we remove “AutoModerator” user which every subreddit uses it for moderation purposes, It filters 286.444 rows.

```
SELECT
COUNT(*)
FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
WHERE 1=1
AND LENGTH(ERS.BODY) > 2
AND NOT(LOWER(ERS.AUTHOR) LIKE '%\_bot\_%' OR LOWER(ERS.AUTHOR) LIKE '%\-bot\-%')
AND NOT(LOWER(REPLACE(ERS.BODY,'''',')) LIKE '%im a bot%')
AND ERS.BODY <> '[deleted]'
AND LENGTH(ERS.removal_reason) = 0
AND ERS.BODY LIKE '% %'
AND ERS.AUTHOR <> 'AutoModerator'
AND ERS.AUTHOR <> '[deleted]'
```

Filtering authors which they deleted their account removes 305.983 rows.

7 Cleaning data

Using sql analysis we found out which data to ignore, we must clean data before working on it.

[1]:
```
import pandas as pd
import numpy as np
import psycopg2
import time
import math

conn_string = 'host={pghost} port={pgport} dbname={pgdatabase} user={pguser}␣
↪password={pgpassword}'.
↪format(pgdatabase='MEF-BDA-PROD',pguser='postgres',pgpassword='123',pghost='localhost',pgpo
conn=psycopg2.connect(conn_string)
cur=conn.cursor()

def check_if_table_exists(schema,table):
    cur.execute("select exists(select * from information_schema.tables where␣
↪table_schema='{schema}' AND table_name='{table}')".format(schema=schema,␣
↪table=table))
    return cur.fetchone()[0]

def check_if_index_exists(index):
    cur.execute("SELECT EXISTS(SELECT * FROM PG_CLASS WHERE relname =␣
↪'{index}')".format(index=index))
    return cur.fetchone()[0]

if(check_if_table_exists('EDW','DWH_REDDIT_COMMENTS')):
    print('Table already exists')
else:
    start_time = time.time()
    cur.execute('set time zone UTC;')
```

```
    cur.execute("""
    CREATE TABLE "EDW"."DWH_REDDIT_COMMENTS" AS
    SELECT
    ROW_NUMBER() OVER (ORDER BY ERS.ID) AS ID,
    TO_TIMESTAMP(GREATEST(ERS.CREATED_UTC ,CAST(ERS.EDITED AS INTEGER))) AS␣
 ↪DATE,
    ERS.SUBREDDIT,
    ERS.AUTHOR,
    ERS.AUTHOR_FLAIR_TEXT,
    ERS.SCORE,
    ERS.BODY AS COMMENT
    FROM "ODS"."EXT_REDDIT_COMMENTS" ERS
    WHERE 1=1
    AND LENGTH(ERS.BODY) > 2
    AND NOT(LOWER(ERS.AUTHOR) LIKE '%\_bot\_%' OR LOWER(ERS.AUTHOR) LIKE␣
 ↪'%\-bot\-%')
    AND NOT(LOWER(REPLACE(ERS.BODY,'''','')) LIKE '%im a bot%')
    AND ERS.BODY <> '[deleted]'
    AND LENGTH(ERS.removal_reason) = 0
    AND ERS.BODY LIKE '% %'
    AND ERS.AUTHOR <> 'AutoModerator'
    AND ERS.AUTHOR <> '[deleted]';
    """)
    cur.execute('COMMIT;')
    print("Table created in {execute_time} seconds".format(execute_time=math.
 ↪trunc(time.time()-start_time)))

if(check_if_index_exists('IDX_DWH_REDDIT_COMMENTS#01')):
    print('Index already exists')
else:
    start_time = time.time()
    cur.execute("""
    CREATE INDEX "IDX_DWH_REDDIT_COMMENTS#01"
    ON "EDW"."DWH_REDDIT_COMMENTS" USING BTREE(
        "id" ASC NULLS LAST,
        "date" ASC NULLS LAST
    )
    TABLESPACE PG_DEFAULT;
    """)
    cur.execute('COMMIT;')
    print("Index created in {execute_time} seconds".format(execute_time=math.
 ↪trunc(time.time()-start_time)))
```

```
Table already exists
Index already exists
```

1. We filtered our data and transformed epoch date to readable date and added numeric id to work our data with batch processing. It reduced our row count 54.504.410(with 33.3GB) to

48.690.746(with 24.5GB) with 11% reduction in rows and 27% reduction in size.

2. Added index to increase our read speed from table.

[2]:
```
import os
import re
import urllib
from urllib.request import urlopen
import fsspec
import xlrd
import xlsxwriter
from pandas import DataFrame

def download_file_if_not_exists(file_url,file_name):
    start_time = math.trunc(time.time())
    if(os.path.exists(file_name) and os.stat(file_name).st_size==0):
        os.remove(file_name)
    if(not(os.path.exists(file_name))):
        urllib.request.urlretrieve(file_url,file_name)
        with open(file_name, 'r+', errors='ignore', encoding="utf-8") as f:
            file_text = f.read()
            file_text = re.sub(r'"[^"]*"', lambda m: m.group(0).replace(',', '␣
↪'), file_text).replace('\\','').replace('"','').replace("'",'')
            f.seek(0)
            f.write(file_text)
            f.truncate()
    end_time = math.trunc(time.time())
    if(start_time!=end_time):
        print("File downloaded and cleaned in {execute_time} seconds".
↪format(execute_time=end_time-start_time))

file_name = "DatafinitiElectronicsProductsPricingData.csv"
file_url = "https://query.data.world/s/n7byb65oqj47oro2btcqqyas62zclv"
download_file_if_not_exists(file_url,file_name)

file_name = "electronic_products_pricing_df.xlsx"
if(os.path.exists(file_name)):
    electronic_products_pricing_df = pd.read_excel(file_name, engine='openpyxl')
else:
    electronic_products_pricing_df = pd.
↪read_csv("DatafinitiElectronicsProductsPricingData.csv", encoding="utf-8")
    electronic_products_pricing_df = electronic_products_pricing_df.loc[:,␣
↪~electronic_products_pricing_df.columns.str.contains('^Unnamed')]
    electronic_products_pricing_df =␣
↪electronic_products_pricing_df[electronic_products_pricing_df["prices.
↪currency"] == "USD"]
```

```
    electronic_products_pricing_df =␣
↪electronic_products_pricing_df[["name","brand","categories","prices.
↪amountMax"]]
    electronic_products_pricing_df = electronic_products_pricing_df.
↪groupby(["name","brand","categories"]).mean()
    electronic_products_pricing_df = electronic_products_pricing_df.
↪reset_index()
    electronic_products_pricing_df = electronic_products_pricing_df.
↪rename(columns = {"prices.amountMax":"average_price"})
    electronic_products_pricing_df["keys"] = ␣
↪electronic_products_pricing_df[["name","brand","categories"]].agg(' '.join,␣
↪axis=1, ).str.lower()
    electronic_products_pricing_df.to_excel(file_name, engine='xlsxwriter')

file_name = "keys_pricing_df.xlsx"
if(os.path.exists(file_name)):
    keys_pricing_df = pd.read_excel(file_name, index_col=0, engine='openpyxl')
else:
    keys_pricing_df = DataFrame(electronic_products_pricing_df["keys"].
↪replace('&','').str.split(' ').tolist(),␣
↪index=electronic_products_pricing_df["average_price"]).stack()
    keys_pricing_df = keys_pricing_df.reset_index()
    keys_pricing_df.columns = ["average_price","level","key"]
    keys_pricing_df = keys_pricing_df[keys_pricing_df["key"] != "&"]
    keys_pricing_df = keys_pricing_df[["key","average_price"]]
    keys_pricing_df["key"] = keys_pricing_df["key"].str.
↪replace(r'[^A-Za-z0-9]+','')
    keys_pricing_df["key"] = keys_pricing_df["key"][~keys_pricing_df["key"].str.
↪isnumeric()]
    keys_pricing_df = keys_pricing_df[keys_pricing_df["key"] != ""] #Used this␣
↪after realizing dropna doesn't work for some reason
    keys_pricing_df = keys_pricing_df.dropna()
    keys_pricing_df = keys_pricing_df.groupby(["key"]).max()
    keys_pricing_df = keys_pricing_df.reset_index()
    keys_pricing_df = keys_pricing_df.rename(columns = {"average_price":
↪"max_price"})
    keys_pricing_df.to_excel(file_name, engine='xlsxwriter')

keys_pricing_df
```

[2]:

	key	max_price
0	000mah	93.374000
1	0g03674	209.851250
2	1000watt	998.326667
3	1000x	191.996667
4	100400mm	1574.278571

```
…                                    …          …
2926                                 zubehr    58.730000
2927                                  zubie    99.990000
2928  zuxbbweveteztffywrybecztecqtezfbc  1799.970000
2929                       zxqyvbwuwyydcq    18.158000
2930      zxsqvcbvvywsrwuaasvczufystyyy    75.882857

[2931 rows x 2 columns]
```

We are getting some price and key data to analize how much we would gain if we presented adds based on their comments.

8 Removing stopwords and using Natural Language Processing while Detecting Language

Stopwords are commonly used words that must be not stored because they do not give any significant value to analyze. We are going to remove them from our table so we can lower our datasize and proccess our data faster.

We must test which libraries to use and how to use them;

[3]:
```
import pycld2

value = 'Testing this value'
print("String '{test_value}' returns: {most_confident_language_tuple}\n".
 ↪format(test_value=value,most_confident_language_tuple=pycld2.
 ↪detect(value)[2][0])) #This will return the language with highest confidence␣
 ↪score.
value = 'Bu değeri deniyoruz'
print("String '{test_value}' returns: {most_confident_language_tuple}\n".
 ↪format(test_value=value,most_confident_language_tuple=pycld2.
 ↪detect(value)[2][0]))
value = 'Test text'
print("String '{test_value}' returns: {most_confident_language_tuple}\n".
 ↪format(test_value=value,most_confident_language_tuple=pycld2.
 ↪detect(value)[2][0]))
```

```
String 'Testing this value' returns: ('ENGLISH', 'en', 95, 1077.0)

String 'Bu değeri deniyoruz' returns: ('TURKISH', 'tr', 95, 1706.0)

String 'Test text' returns: ('Unknown', 'un', 0, 0.0)
```

[77]:
```
import nltk
import pycld2
from nltk.corpus import stopwords
```

```
nltk.download('stopwords', quiet=True)
stop = stopwords.words('english')

sql_command = 'SELECT * FROM "{schema}"."{table}" WHERE ID = 15;'.
 ↪format(schema='EDW', table='DWH_REDDIT_COMMENTS')
df = pd.read_sql(sql_command,conn)
print("Original comment:\n{comment}\n".format(comment=df['comment'][0]))

df['comment'] = df['comment'].apply(lambda x: ' '.join([word for word in x.
 ↪lower().split() if word not in (stop)]))

print("Comment after removing stopwords:\n{comment}\n".
 ↪format(comment=df['comment'][0]))

start_time = time.time()
cur.execute("""
UPDATE "EDW"."DWH_REDDIT_COMMENTS"
SET "comment" = %(comment)s
WHERE "id" = %(id)s
""", {'comment': str(df['comment'][0]), 'id': int(df['id'][0])})

print("Updated record(s) in {execute_time} seconds\n".format(execute_time=math.
 ↪trunc(time.time()-start_time)))

sql_command = 'SELECT * FROM "{schema}"."{table}" WHERE ID = 15;'.
 ↪format(schema='EDW', table='DWH_REDDIT_COMMENTS')
df = pd.read_sql(sql_command,conn)
cur.execute('ROLLBACK;')
print("Comment in table:\n{comment}\n\n".format(comment=df['comment'][0]))

df['comment_language_code'] = pycld2.detect(str(df['comment']))[2][0][1]
df['comment_language'] = pycld2.detect(str(df['comment']))[2][0][0].lower().
 ↪replace('unknown','english')

df
#Since we are testing here we didn't commit to database so our changes are␣
 ↪going to be rolled back after our session dies. It will be used after␣
 ↪completing our test.
```

```
Original comment:
It's not your fault don't think that. Hey if this helps it does then if it
doesn't well. At least I can talk about it.  I shot my dog. I had him for 10
years. I can home from partying and it was New Years so I was going to shoot off
my .40 so…  Well I was loading it outside and I shot it off accidentally.
Right into my dog. Me having about 5-7 tequila shots into me though if I go to
sleep I would wake up and he would be okay. Well if I called a vet when I shot
him I could have saved him. Or at least from his pain. So I woke up saw a dead
```

```
dog in my yard… i cried for hours. That old dog still had a few good years in
him.   That was my fault and you sitting on a little tiny mouse is more
understandable than a man discharging a firearm into a shitzu. Don't feel to
bad.

Comment after removing stopwords:
fault think that. hey helps well. least talk it. shot dog. 10 years. home
partying new years going shoot .40 so… well loading outside shot accidentally.
right dog. 5-7 tequila shots though go sleep would wake would okay. well called
vet shot could saved him. least pain. woke saw dead dog yard… cried hours. old
dog still good years him. fault sitting little tiny mouse understandable man
discharging firearm shitzu. feel bad.

Updated record(s) in 0 seconds

Comment in table:
fault think that. hey helps well. least talk it. shot dog. 10 years. home
partying new years going shoot .40 so… well loading outside shot accidentally.
right dog. 5-7 tequila shots though go sleep would wake would okay. well called
vet shot could saved him. least pain. woke saw dead dog yard… cried hours. old
dog still good years him. fault sitting little tiny mouse understandable man
discharging firearm shitzu. feel bad.
```

[77]:

```
    id                       date   subreddit      author author_flair_text  \
0  15 2015-05-01 00:00:00+00:00  offmychest  Zekkystyle

   score                                                comment  \
0     14  fault think that. hey helps well. least talk i…

  comment_language_code comment_language
0                    en          english
```

It reduces our data well and it still makes sense.

[4]:

```
import nltk
import pycld2
from nltk.corpus import stopwords

nltk.download('stopwords', quiet=True)

sql_command = """SELECT id, date, subreddit, author,␣
 ↪replace(author_flair_text,'''','') as author_flair_text, score,␣
 ↪replace(COMMENT,'''','') as comment FROM "{schema}"."{table}" WHERE ID␣
 ↪BETWEEN 30545495 AND 30545595;""".format(schema='EDW',␣
 ↪table='DWH_REDDIT_COMMENTS')
df = pd.read_sql(sql_command,conn)
```

```
#30545495 AND 30545595

df['comment_language_code'] = df['comment'].str.replace(r'[^A-Za-z0-9]+',' ').
 ↪apply(pycld2.detect).apply(lambda x: x[2][0][1])
df['comment_language'] = df['comment'].str.replace(r'[^A-Za-z0-9]+',' ').
 ↪apply(pycld2.detect).apply(lambda x: x[2][0][0]).str.lower()

def remove_stopwords(text,text_language):
    text_without_stopwords = ' '
    try:
        text_without_stopwords = text_without_stopwords.join([word for word in␣
 ↪text.lower().split() if word not in (stopwords.words(text_language))])
    except:
        text_without_stopwords = text_without_stopwords.join([word for word in␣
 ↪text.lower().split() if word not in (stopwords.words('english'))])

    text_without_stopwords = re.sub("[\W]"," ",text_without_stopwords,re.
 ↪UNICODE)
    #text_without_stopwords = re.sub("[^\p{L} 0-9]","␣
 ↪",text_without_stopwords,re.UNICODE)
    return text_without_stopwords

df['word_count'] = df['comment'].str.count(' ') #We are getting original word␣
 ↪count without removing stopwords

df['comment'] = df.apply(lambda x: remove_stopwords(x['comment'],␣
 ↪x['comment_language']), axis=1)

df['comment'] = df['comment'].str.replace(',','').replace("'","")
df['author_flair_text'] = df['author_flair_text'].str.replace(',','').
 ↪replace("'","")

df
```

[4]:

	id	date	subreddit	\
0	30545495	2015-05-20 12:13:05+00:00	Turkey	
1	30545496	2015-05-20 12:13:06+00:00	cats	
2	30545497	2015-05-20 12:13:06+00:00	Database	
3	30545498	2015-05-20 12:13:06+00:00	unitedkingdom	
4	30545499	2015-05-20 12:13:06+00:00	skyrim	
..	…	…	…	
96	30545591	2015-05-20 12:13:12+00:00	Coffee	
97	30545592	2015-05-20 12:13:12+00:00	news	
98	30545593	2015-05-20 12:13:12+00:00	h1z1	
99	30545594	2015-05-20 12:13:12+00:00	BeforeNAfterAdoption	

```
100  30545595 2015-05-20 12:13:12+00:00              FlashTV

                  author         author_flair_text  score  \
0               shiguree                                1
1               lovechip                               10
2                cojajoc                                7
3                JamDunc  ex-Yorkshire now Sverige     11
4         Open_Info_of94                                2
..                   …                          …    …
96    Italian_Not_Jewish                                1
97        PrivateHazzard                                1
98               Tenetri                                1
99             PantsGiver                              14
100  SuperBattleFranky37                               3

                                                comment comment_language_code  \
0    i nsana insan yaşama fırsatı verildiğinde şimd…                      tr
1    looks like cat used know  that sounds like wei…                      en
2    partitioning table would useful   partition da…                      en
3    well swedens system open run government   alth…                      en
4           lol said role play hard  thats real thing                        en
..                                                 …                     …
96   ive issue  brought french press help hard clea…                      en
97   logic incorrect    dont need war crime    real…                      en
98   but  ive gotten many streamers  lt5 viewers bl…                      en
99   inside cats really helps  also live fiv  cats …                      en
100   im curious ciscos source dna tested crime scene                       en

    comment_language  word_count
0            turkish         127
1            english          24
2            english         111
3            english          40
4            english          12
..               …         …
96           english          96
97           english          19
98           english          37
99           english          32
100          english          16

[101 rows x 10 columns]
```

[6]:
```
import os
import psycopg2
import numpy as np
import psycopg2.extras as extras
```

```
from io import StringIO

nltk.download('stopwords', quiet=True)

def df_column_conversation(df, column_name, type):
    if(type == 'timestamp'):
        df[column_name] = df[column_name].apply(lambda x: f"'{x}'::timestamp")
    if(type == 'text'):
        df[column_name] = df[column_name].apply(lambda x: f"'{x}'")

def remove_stopwords(text,text_language):
    text_without_stopwords = ' '
    try:
        text_without_stopwords = text_without_stopwords.join([word for word in␣
↪text.lower().split() if word not in (stopwords.words(text_language))])
    except:
        text_without_stopwords = text_without_stopwords.join([word for word in␣
↪text.lower().split() if word not in (stopwords.words('english'))])

    text_without_stopwords = re.sub("[\W]"," ",text_without_stopwords,re.
↪UNICODE)
    return text_without_stopwords

def execute_mogrify(conn, df, schema, table):
    """
    Using cursor.mogrify() to build the bulk insert query
    then cursor.execute() to execute the query
    """
    # Create a list of tupples from the dataframe values
    tuples = [tuple(x) for x in df.to_numpy()]
    # Comma-separated dataframe columns
    cols = ','.join(list(df.columns))
    # SQL quert to execute
    cursor = conn.cursor()
    try:
        for tup in tuples:
            query  = """INSERT INTO "{schema}"."{table}"({cols}) VALUES␣
↪({values})""".format(schema=schema,table=table, cols=cols, values=",".
↪join(map(str,tup)))
            cursor.execute(query)
            conn.commit()
    except (Exception, psycopg2.DatabaseError) as error:
        print("Error: %s" % error)
        conn.rollback()
        cursor.close()
        return 1
    #print("execute_mogrify() done")
```

```
    cursor.close()

cur.execute('SELECT MAX(ID) FROM "EDW"."DWH_REDDIT_COMMENTS_DETAIL"')
start_row = cur.fetchone()[0]
row_per_loop = 10000
cur.execute('SELECT COUNT(1) FROM "EDW"."DWH_REDDIT_COMMENTS"')
end_row = cur.fetchone()[0]

start_time = math.trunc(time.time())
for i in range(start_row,end_row,row_per_loop):
    sql_command = """SELECT id, date, subreddit, author,␣
 ↪replace(author_flair_text,'''','') as author_flair_text, score,␣
 ↪replace(COMMENT,'''','') as comment FROM "{schema}"."{table}" WHERE ID␣
 ↪BETWEEN {start_row} AND {end_row};""".format(schema='EDW',␣
 ↪table='DWH_REDDIT_COMMENTS', start_row=i, end_row=i+row_per_loop-1)
    df = pd.read_sql(sql_command,conn)
    df['comment_language_code'] = df['comment'].str.replace(r'[^A-Za-z0-9]+','␣
 ↪').apply(pycld2.detect).apply(lambda x: x[2][0][1])
    df['comment_language'] = df['comment'].str.replace(r'[^A-Za-z0-9]+',' ').
 ↪apply(pycld2.detect).apply(lambda x: x[2][0][0]).str.lower()
    df['word_count'] = df['comment'].str.count(' ') #We are getting original␣
 ↪word count without removing stopwords
    df['comment'] = df.apply(lambda x: remove_stopwords(x['comment'],␣
 ↪x['comment_language']), axis=1)
    df['comment'] = df['comment'].str.replace(',','')
    df['author_flair_text'] = df['author_flair_text'].str.replace(',','')
    df_column_conversation(df, 'date', 'timestamp')
    df_column_conversation(df, 'subreddit', 'text')
    df_column_conversation(df, 'author', 'text')
    df_column_conversation(df, 'author_flair_text', 'text')
    df_column_conversation(df, 'comment', 'text')
    df_column_conversation(df, 'comment_language_code', 'text')
    df_column_conversation(df, 'comment_language', 'text')
    execute_mogrify(conn,df,"EDW","DWH_REDDIT_COMMENTS_DETAIL")
    print("Inserted rows between {start_row} and {end_row}".format(start_row=i,␣
 ↪end_row=i+row_per_loop-1))

end_time = math.trunc(time.time())
print("Data transformation completed in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))
```

```
Inserted rows between 1 and 10000
Inserted rows between 10001 and 20000
Inserted rows between 20001 and 30000
Inserted rows between 30001 and 40000
Inserted rows between 40001 and 50000
Inserted rows between 50001 and 60000
```

```
Inserted rows between 60001 and 70000
Inserted rows between 70001 and 80000
Inserted rows between 80001 and 90000
Inserted rows between 90001 and 100000
Inserted rows between 100001 and 110000
Inserted rows between 110001 and 120000
Inserted rows between 120001 and 130000
Inserted rows between 130001 and 140000
Inserted rows between 140001 and 150000
Inserted rows between 150001 and 160000
Inserted rows between 160001 and 170000
Inserted rows between 170001 and 180000
Inserted rows between 180001 and 190000
Inserted rows between 190001 and 200000
Inserted rows between 200001 and 210000
Inserted rows between 210001 and 220000
Inserted rows between 220001 and 230000
Inserted rows between 230001 and 240000
Inserted rows between 240001 and 250000
Inserted rows between 250001 and 260000
Inserted rows between 260001 and 270000
Inserted rows between 270001 and 280000
Inserted rows between 280001 and 290000
Inserted rows between 290001 and 300000
Inserted rows between 300001 and 310000
Inserted rows between 310001 and 320000
Inserted rows between 320001 and 330000
Inserted rows between 330001 and 340000
Inserted rows between 340001 and 350000
Inserted rows between 350001 and 360000
Inserted rows between 360001 and 370000
Inserted rows between 370001 and 380000
Inserted rows between 380001 and 390000
Inserted rows between 390001 and 400000
Inserted rows between 400001 and 410000
Inserted rows between 410001 and 420000
Inserted rows between 420001 and 430000
Inserted rows between 430001 and 440000
Inserted rows between 440001 and 450000
Inserted rows between 450001 and 460000
Inserted rows between 460001 and 470000
Inserted rows between 470001 and 480000
Inserted rows between 480001 and 490000
Inserted rows between 490001 and 500000
Inserted rows between 500001 and 510000
Inserted rows between 510001 and 520000
Inserted rows between 520001 and 530000
Inserted rows between 530001 and 540000
```

```
Inserted rows between 540001 and 550000
Inserted rows between 550001 and 560000
Inserted rows between 560001 and 570000
Inserted rows between 570001 and 580000
Inserted rows between 580001 and 590000
Inserted rows between 590001 and 600000
Inserted rows between 600001 and 610000
Inserted rows between 610001 and 620000
Inserted rows between 620001 and 630000
Inserted rows between 630001 and 640000
Inserted rows between 640001 and 650000
Inserted rows between 650001 and 660000
Inserted rows between 660001 and 670000
Inserted rows between 670001 and 680000
Inserted rows between 680001 and 690000
Inserted rows between 690001 and 700000
Inserted rows between 700001 and 710000
Inserted rows between 710001 and 720000
Inserted rows between 720001 and 730000
Inserted rows between 730001 and 740000
Inserted rows between 740001 and 750000
Inserted rows between 750001 and 760000
Inserted rows between 760001 and 770000
Inserted rows between 770001 and 780000
Inserted rows between 780001 and 790000
Inserted rows between 790001 and 800000
Inserted rows between 800001 and 810000
Inserted rows between 810001 and 820000
Inserted rows between 820001 and 830000
Inserted rows between 830001 and 840000
Inserted rows between 840001 and 850000
Inserted rows between 850001 and 860000
Inserted rows between 860001 and 870000
Inserted rows between 870001 and 880000
Inserted rows between 880001 and 890000
Inserted rows between 890001 and 900000
Inserted rows between 900001 and 910000
Inserted rows between 910001 and 920000
Inserted rows between 920001 and 930000
Inserted rows between 930001 and 940000
Inserted rows between 940001 and 950000
Inserted rows between 950001 and 960000
Inserted rows between 960001 and 970000
Inserted rows between 970001 and 980000
Inserted rows between 980001 and 990000
Inserted rows between 990001 and 1000000
Inserted rows between 1000001 and 1010000
Inserted rows between 1010001 and 1020000
```

```
Inserted rows between 1020001 and 1030000
Inserted rows between 1030001 and 1040000
Inserted rows between 1040001 and 1050000
Inserted rows between 1050001 and 1060000
Inserted rows between 1060001 and 1070000
Inserted rows between 1070001 and 1080000
Inserted rows between 1080001 and 1090000
Inserted rows between 1090001 and 1100000
Inserted rows between 1100001 and 1110000
Inserted rows between 1110001 and 1120000
Inserted rows between 1120001 and 1130000
Inserted rows between 1130001 and 1140000
Inserted rows between 1140001 and 1150000
Inserted rows between 1150001 and 1160000
Inserted rows between 1160001 and 1170000
Inserted rows between 1170001 and 1180000
Inserted rows between 1180001 and 1190000
Inserted rows between 1190001 and 1200000
Inserted rows between 1200001 and 1210000
Inserted rows between 1210001 and 1220000
Inserted rows between 1220001 and 1230000
Inserted rows between 1230001 and 1240000
Inserted rows between 1240001 and 1250000
Inserted rows between 1250001 and 1260000
Inserted rows between 1260001 and 1270000
Inserted rows between 1270001 and 1280000
Inserted rows between 1280001 and 1290000
Inserted rows between 1290001 and 1300000
Inserted rows between 1300001 and 1310000
Inserted rows between 1310001 and 1320000
Inserted rows between 1320001 and 1330000
Inserted rows between 1330001 and 1340000
Inserted rows between 1340001 and 1350000
Inserted rows between 1350001 and 1360000
Inserted rows between 1360001 and 1370000
Inserted rows between 1370001 and 1380000
Inserted rows between 1380001 and 1390000
Inserted rows between 1390001 and 1400000
Inserted rows between 1400001 and 1410000
Inserted rows between 1410001 and 1420000
Inserted rows between 1420001 and 1430000
Inserted rows between 1430001 and 1440000
Inserted rows between 1440001 and 1450000
Inserted rows between 1450001 and 1460000
Inserted rows between 1460001 and 1470000
Inserted rows between 1470001 and 1480000
Inserted rows between 1480001 and 1490000
Inserted rows between 1490001 and 1500000
```

```
Inserted rows between 1500001 and 1510000
Inserted rows between 1510001 and 1520000
Inserted rows between 1520001 and 1530000
Inserted rows between 1530001 and 1540000
Inserted rows between 1540001 and 1550000
Inserted rows between 1550001 and 1560000
Inserted rows between 1560001 and 1570000
Inserted rows between 1570001 and 1580000
Inserted rows between 1580001 and 1590000
Inserted rows between 1590001 and 1600000
Inserted rows between 1600001 and 1610000
Inserted rows between 1610001 and 1620000
Inserted rows between 1620001 and 1630000
Inserted rows between 1630001 and 1640000
Inserted rows between 1640001 and 1650000
Inserted rows between 1650001 and 1660000
Inserted rows between 1660001 and 1670000
Inserted rows between 1670001 and 1680000
Inserted rows between 1680001 and 1690000
Inserted rows between 1690001 and 1700000
Inserted rows between 1700001 and 1710000
Inserted rows between 1710001 and 1720000
Inserted rows between 1720001 and 1730000
Inserted rows between 1730001 and 1740000
Inserted rows between 1740001 and 1750000
Inserted rows between 1750001 and 1760000
Inserted rows between 1760001 and 1770000
Inserted rows between 1770001 and 1780000
Inserted rows between 1780001 and 1790000
Inserted rows between 1790001 and 1800000
Inserted rows between 1800001 and 1810000
Inserted rows between 1810001 and 1820000
Inserted rows between 1820001 and 1830000
Inserted rows between 1830001 and 1840000
Inserted rows between 1840001 and 1850000
Inserted rows between 1850001 and 1860000
Inserted rows between 1860001 and 1870000
Inserted rows between 1870001 and 1880000
Inserted rows between 1880001 and 1890000
Inserted rows between 1890001 and 1900000
Inserted rows between 1900001 and 1910000
Inserted rows between 1910001 and 1920000
Inserted rows between 1920001 and 1930000
Inserted rows between 1930001 and 1940000
Inserted rows between 1940001 and 1950000
Inserted rows between 1950001 and 1960000
Inserted rows between 1960001 and 1970000
Inserted rows between 1970001 and 1980000
```

```
Inserted rows between 1980001 and 1990000
Inserted rows between 1990001 and 2000000
Inserted rows between 2000001 and 2010000
Inserted rows between 2010001 and 2020000
Inserted rows between 2020001 and 2030000
Inserted rows between 2030001 and 2040000
Inserted rows between 2040001 and 2050000
Inserted rows between 2050001 and 2060000
Inserted rows between 2060001 and 2070000
Inserted rows between 2070001 and 2080000
Inserted rows between 2080001 and 2090000
Inserted rows between 2090001 and 2100000
Inserted rows between 2100001 and 2110000
Inserted rows between 2110001 and 2120000
Inserted rows between 2120001 and 2130000
Inserted rows between 2130001 and 2140000
Inserted rows between 2140001 and 2150000
Inserted rows between 2150001 and 2160000
Inserted rows between 2160001 and 2170000
Inserted rows between 2170001 and 2180000
Inserted rows between 2180001 and 2190000
Inserted rows between 2190001 and 2200000
Inserted rows between 2200001 and 2210000
Inserted rows between 2210001 and 2220000
Inserted rows between 2220001 and 2230000
Inserted rows between 2230001 and 2240000
Inserted rows between 2240001 and 2250000
Inserted rows between 2250001 and 2260000
Inserted rows between 2260001 and 2270000
Inserted rows between 2270001 and 2280000
Inserted rows between 2280001 and 2290000
Inserted rows between 2290001 and 2300000
Inserted rows between 2300001 and 2310000
Inserted rows between 2310001 and 2320000
Inserted rows between 2320001 and 2330000
Inserted rows between 2330001 and 2340000
Inserted rows between 2340001 and 2350000
Inserted rows between 2350001 and 2360000
Inserted rows between 2360001 and 2370000
Inserted rows between 2370001 and 2380000
Inserted rows between 2380001 and 2390000
Inserted rows between 2390001 and 2400000
Inserted rows between 2400001 and 2410000
Inserted rows between 2410001 and 2420000
Inserted rows between 2420001 and 2430000
Inserted rows between 2430001 and 2440000
Inserted rows between 2440001 and 2450000
Inserted rows between 2450001 and 2460000
```

```
Inserted rows between 2460001 and 2470000
Inserted rows between 2470001 and 2480000
Inserted rows between 2480001 and 2490000
Inserted rows between 2490001 and 2500000
Inserted rows between 2500001 and 2510000
Inserted rows between 2510001 and 2520000
Inserted rows between 2520001 and 2530000
Inserted rows between 2530001 and 2540000
Inserted rows between 2540001 and 2550000
Inserted rows between 2550001 and 2560000
Inserted rows between 2560001 and 2570000
Inserted rows between 2570001 and 2580000
Inserted rows between 2580001 and 2590000
Inserted rows between 2590001 and 2600000
Inserted rows between 2600001 and 2610000
Inserted rows between 2610001 and 2620000
Inserted rows between 2620001 and 2630000
Inserted rows between 2630001 and 2640000
Inserted rows between 2640001 and 2650000
Inserted rows between 2650001 and 2660000
Inserted rows between 2660001 and 2670000
Inserted rows between 2670001 and 2680000
Inserted rows between 2680001 and 2690000
Inserted rows between 2690001 and 2700000
Inserted rows between 2700001 and 2710000
Inserted rows between 2710001 and 2720000
Inserted rows between 2720001 and 2730000
Inserted rows between 2730001 and 2740000
Inserted rows between 2740001 and 2750000
Inserted rows between 2750001 and 2760000
Inserted rows between 2760001 and 2770000
Inserted rows between 2770001 and 2780000
Inserted rows between 2780001 and 2790000
Inserted rows between 2790001 and 2800000
Inserted rows between 2800001 and 2810000
Inserted rows between 2810001 and 2820000
Inserted rows between 2820001 and 2830000
Inserted rows between 2830001 and 2840000
Inserted rows between 2840001 and 2850000
Inserted rows between 2850001 and 2860000
Inserted rows between 2860001 and 2870000
Inserted rows between 2870001 and 2880000
Inserted rows between 2880001 and 2890000
Inserted rows between 2890001 and 2900000
Inserted rows between 2900001 and 2910000
Inserted rows between 2910001 and 2920000
Inserted rows between 2920001 and 2930000
Inserted rows between 2930001 and 2940000
```

```
Inserted rows between 2940001 and 2950000
Inserted rows between 2950001 and 2960000
Inserted rows between 2960001 and 2970000
Inserted rows between 2970001 and 2980000
Inserted rows between 2980001 and 2990000
Inserted rows between 2990001 and 3000000
Inserted rows between 3000001 and 3010000
Inserted rows between 3010001 and 3020000
Inserted rows between 3020001 and 3030000
Inserted rows between 3030001 and 3040000
Inserted rows between 3040001 and 3050000
Inserted rows between 3050001 and 3060000
Inserted rows between 3060001 and 3070000
Inserted rows between 3070001 and 3080000
Inserted rows between 3080001 and 3090000
Inserted rows between 3090001 and 3100000
Inserted rows between 3100001 and 3110000
Inserted rows between 3110001 and 3120000
Inserted rows between 3120001 and 3130000
Inserted rows between 3130001 and 3140000
Inserted rows between 3140001 and 3150000
Inserted rows between 3150001 and 3160000
Inserted rows between 3160001 and 3170000
Inserted rows between 3170001 and 3180000
Inserted rows between 3180001 and 3190000
Inserted rows between 3190001 and 3200000
Inserted rows between 3200001 and 3210000
Inserted rows between 3210001 and 3220000
Inserted rows between 3220001 and 3230000
Inserted rows between 3230001 and 3240000
Inserted rows between 3240001 and 3250000
Inserted rows between 3250001 and 3260000
Inserted rows between 3260001 and 3270000
Inserted rows between 3270001 and 3280000
Inserted rows between 3280001 and 3290000
Inserted rows between 3290001 and 3300000
Inserted rows between 3300001 and 3310000
Inserted rows between 3310001 and 3320000
Inserted rows between 3320001 and 3330000
Inserted rows between 3330001 and 3340000
Inserted rows between 3340001 and 3350000
Inserted rows between 3350001 and 3360000
Inserted rows between 3360001 and 3370000
Inserted rows between 3370001 and 3380000
Inserted rows between 3380001 and 3390000
Inserted rows between 3390001 and 3400000
Inserted rows between 3400001 and 3410000
Inserted rows between 3410001 and 3420000
```

```
Inserted rows between 3420001 and 3430000
Inserted rows between 3430001 and 3440000
Inserted rows between 3440001 and 3450000
Inserted rows between 3450001 and 3460000
Inserted rows between 3460001 and 3470000
Inserted rows between 3470001 and 3480000
Inserted rows between 3480001 and 3490000
Inserted rows between 3490001 and 3500000
Inserted rows between 3500001 and 3510000
Inserted rows between 3510001 and 3520000
Inserted rows between 3520001 and 3530000
Inserted rows between 3530001 and 3540000
Inserted rows between 3540001 and 3550000
Inserted rows between 3550001 and 3560000
Inserted rows between 3560001 and 3570000
Inserted rows between 3570001 and 3580000
Inserted rows between 3580001 and 3590000
Inserted rows between 3590001 and 3600000
Inserted rows between 3600001 and 3610000
Inserted rows between 3610001 and 3620000
Inserted rows between 3620001 and 3630000
Inserted rows between 3630001 and 3640000
Inserted rows between 3640001 and 3650000
Inserted rows between 3650001 and 3660000
Inserted rows between 3660001 and 3670000
Inserted rows between 3670001 and 3680000
Inserted rows between 3680001 and 3690000
Inserted rows between 3690001 and 3700000
Inserted rows between 3700001 and 3710000
Inserted rows between 3710001 and 3720000
Inserted rows between 3720001 and 3730000
Inserted rows between 3730001 and 3740000
Inserted rows between 3740001 and 3750000
Inserted rows between 3750001 and 3760000
Inserted rows between 3760001 and 3770000
Inserted rows between 3770001 and 3780000
Inserted rows between 3780001 and 3790000
Inserted rows between 3790001 and 3800000
Inserted rows between 3800001 and 3810000
Inserted rows between 3810001 and 3820000
Inserted rows between 3820001 and 3830000
Inserted rows between 3830001 and 3840000
Inserted rows between 3840001 and 3850000
Inserted rows between 3850001 and 3860000
Inserted rows between 3860001 and 3870000
Inserted rows between 3870001 and 3880000
Inserted rows between 3880001 and 3890000
Inserted rows between 3890001 and 3900000
```

```
Inserted rows between 3900001 and 3910000
Inserted rows between 3910001 and 3920000
Inserted rows between 3920001 and 3930000
Inserted rows between 3930001 and 3940000
Inserted rows between 3940001 and 3950000
Inserted rows between 3950001 and 3960000
Inserted rows between 3960001 and 3970000
Inserted rows between 3970001 and 3980000
Inserted rows between 3980001 and 3990000
Inserted rows between 3990001 and 4000000
Inserted rows between 4000001 and 4010000
Inserted rows between 4010001 and 4020000
Inserted rows between 4020001 and 4030000
Inserted rows between 4030001 and 4040000
Inserted rows between 4040001 and 4050000
Inserted rows between 4050001 and 4060000
Inserted rows between 4060001 and 4070000
Inserted rows between 4070001 and 4080000
Inserted rows between 4080001 and 4090000
Inserted rows between 4090001 and 4100000
Inserted rows between 4100001 and 4110000
Inserted rows between 4110001 and 4120000
Inserted rows between 4120001 and 4130000
Inserted rows between 4130001 and 4140000
Inserted rows between 4140001 and 4150000
Inserted rows between 4150001 and 4160000
Inserted rows between 4160001 and 4170000
Inserted rows between 4170001 and 4180000
Inserted rows between 4180001 and 4190000
Inserted rows between 4190001 and 4200000
Inserted rows between 4200001 and 4210000
Inserted rows between 4210001 and 4220000
Inserted rows between 4220001 and 4230000
Inserted rows between 4230001 and 4240000
Inserted rows between 4240001 and 4250000
```

[46]:
```
cur.close()
conn.close()
```

9 Sources:

1. About Reddit
2. Data source
3. Checking if a table exist with psycopg2 on postgreSQL
4. Using current time in UTC as default value in PostgreSQL. This is important because date is utc in the data

5. Creating multicolumn index on PostgreSQL
6. Checking if index exist
7. How to execute start time and end time in python
8. Truncating numbers in python
9. Removing stopwords
10. Prevent SQL Injection in Python
11. Preventing SQL Injection resulted errors but It needed to be done, data type conversation is the key here
12. About stopwords
13. How to detect language
14. Increasing timeout while installing new packages
15. PyCld2 is only works in linux systems
16. Replacing text to change unknown values to english
17. Electronic Products and Pricing Data
18. Remove all commas between quotes
19. Check if file exist in directory
20. Download files
21. Deleting empty files
22. Removing unnamed columns
23. Split (explode) pandas dataframe string entry to separate rows
24. Dropping numeric values
25. Join function
26. Apply function with two arguments to columns
27. Pandas to PostgreSQL using Psycopg2: Bulk Insert Performance Benchmark

[ ]: