reddit_analysis

January 28, 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 for preprocessing

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 EDW.DWH_REDDIT_COMMENTS 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 EDW.DWH_REDDIT_COMMENTS 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 EDW.DWH_REDDIT_COMMENTS 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.

7.1 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)
```

[4]:
```
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:
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.

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.
```

[4]:

```
   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.

[5]:

```
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
```

[5]:

	id	date	subreddit	author	author_flair_text	score
0	30545495	2015-05-20 12:13:05+00:00	Turkey	shiguree		1
1	30545496	2015-05-20 12:13:06+00:00	cats	lovechip		10
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			

```
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]+1
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))
```

```
Data transformation completed in 0 seconds.
```

Inserting about 15,000,000 rows took about 11.13 hours. Data processing in this part took tremendous amount of time.

8 Data Analysis

8.1 Top 20 Most Commented Subreddits on Average

In this analysis we are going to see most commented subreddits on average. We are going to ignore subreddits which has less than 200 comments because they might be skewed from low comment count.

[7]:
```
if(check_if_table_exists('EDW','DWH_REDDIT_SUBREDDIT_COMMENTS')):
    print('Table EDW.DWH_REDDIT_SUBREDDIT_COMMENTS already exists')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_REDDIT_SUBREDDIT_COMMENTS" AS
    SELECT
    SUBREDDIT,
    AVG(WORD_COUNT) AS AVERAGE_WORD_COUNT,
    MIN(WORD_COUNT) AS MINIMUM_WORD_COUNT,
    MAX(WORD_COUNT) AS MAXIMUM_WORD_COUNT,
    COUNT(*) AS COMMENT_COUNT
    FROM "EDW"."DWH_REDDIT_COMMENTS_DETAIL"
    GROUP BY SUBREDDIT;
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_REDDIT_SUBREDDIT_COMMENTS created in {execute_time}␣
 ↪seconds".format(execute_time=end_time-start_time))
```

```
Table EDW.DWH_REDDIT_SUBREDDIT_COMMENTS already exists
```

[8]:
```
sql_command = """
SELECT
*
FROM "EDW"."DWH_REDDIT_SUBREDDIT_COMMENTS"
WHERE 1=1
AND COMMENT_COUNT >= 200
ORDER BY AVERAGE_WORD_COUNT DESC;
"""
df_subreddit_comments = pd.read_sql(sql_command,conn)
df_subreddit_comments['average_word_count'] =␣
 ↪df_subreddit_comments['average_word_count'].apply(round)
df_subreddit_comments.head(20)
```

[8]:

	subreddit	average_word_count	minimum_word_count	\
0	subredditreports	503	1	
1	dancing	473	1	
2	dailyprogrammer	344	1	
3	cissp	344	1	
4	csgocritic	323	1	
5	DanceDanceRevolution	299	1	
6	JonTronCirclejerk	283	1	
7	DestructiveReaders	234	1	
8	Kochen	198	1	
9	kundalini	183	1	
10	Worldprompts	171	1	
11	PSYC2371	170	1	

12	IronThroneRP	170	1
13	Dance	169	1
14	SeireiteiRP	162	1
15	Oneirosophy	160	1
16	WritingPrompts	155	1
17	Target	147	1
18	Sikh	145	1
19	ReasonableFaith	145	1

	maximum_word_count	comment_count
0	1135	3303
1	2044	663
2	5024	1577
3	1943	311
4	2002	239
5	3640	1007
6	3332	245
7	1986	1532
8	1021	221
9	1636	218
10	1746	537
11	1103	709
12	1625	2512
13	2069	506
14	757	223
15	1225	630
16	3224	44070
17	1698	995
18	1665	1159
19	1802	1429

8.2 Hourly Comment

[9]:
```
if(check_if_table_exists('EDW','DWH_REDDIT_HOURLY_COMMENTS')):
    print('Table EDW.DWH_REDDIT_HOURLY_COMMENTS already exists')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_REDDIT_HOURLY_COMMENTS" AS
    WITH COMMENT_HOURLY AS(
    SELECT
    EXTRACT('HOUR' FROM DATE) AS UTC_HOUR,
    COUNT(*) AS COMMENT_COUNT
    FROM "EDW"."DWH_REDDIT_COMMENTS_DETAIL"
    WHERE 1=1
    GROUP BY EXTRACT('HOUR' FROM DATE)
    )
```

```
    SELECT
    CH.*,
    SUM(COMMENT_COUNT) OVER() AS TOTAL_COMMENT_COUNT,
    COMMENT_COUNT/SUM(COMMENT_COUNT) OVER() AS PERCENTAGE
    FROM COMMENT_HOURLY CH;
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_REDDIT_HOURLY_COMMENTS created in {execute_time}␣
↪seconds".format(execute_time=end_time-start_time))
```

```
Table EDW.DWH_REDDIT_HOURLY_COMMENTS already exists
```

[10]:
```
sql_command = """
SELECT
*
FROM "EDW"."DWH_REDDIT_HOURLY_COMMENTS";
"""
df_hourly_comments = pd.read_sql(sql_command,conn)
df_hourly_comments['percentage'] =␣
↪round(df_hourly_comments['percentage'],ndigits=4)*100
df_hourly_comments.head(24)
```

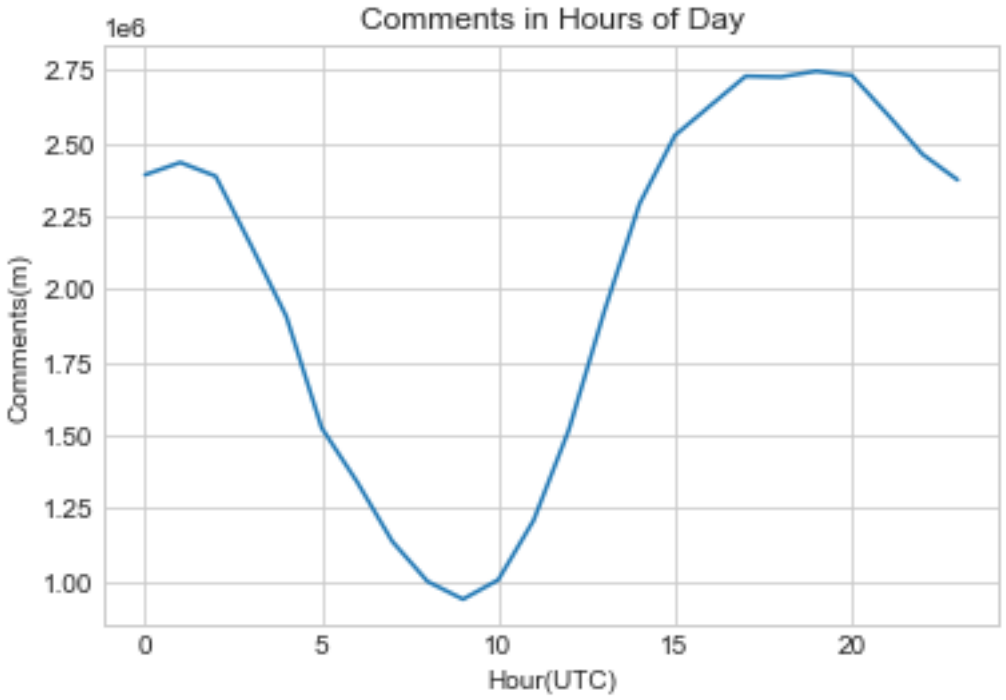
[10]:

	utc_hour	comment_count	total_comment_count	percentage
0	0.0	2391819	48690748.0	4.91
1	1.0	2434179	48690748.0	5.00
2	2.0	2387353	48690748.0	4.90
3	3.0	2150783	48690748.0	4.42
4	4.0	1906910	48690748.0	3.92
5	5.0	1525142	48690748.0	3.13
6	6.0	1340774	48690748.0	2.75
7	7.0	1136063	48690748.0	2.33
8	8.0	998797	48690748.0	2.05
9	9.0	938141	48690748.0	1.93
10	10.0	1004587	48690748.0	2.06
11	11.0	1208687	48690748.0	2.48
12	12.0	1519600	48690748.0	3.12
13	13.0	1924457	48690748.0	3.95
14	14.0	2293496	48690748.0	4.71
15	15.0	2527095	48690748.0	5.19
16	16.0	2628807	48690748.0	5.40
17	17.0	2729762	48690748.0	5.61
18	18.0	2727110	48690748.0	5.60
19	19.0	2747001	48690748.0	5.64
20	20.0	2732594	48690748.0	5.61
21	21.0	2600597	48690748.0	5.34
22	22.0	2462657	48690748.0	5.06

```
23      23.0        2374337          48690748.0        4.88
```

[11]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

plt.plot(df_hourly_comments['utc_hour'], df_hourly_comments['comment_count'])
plt.xlabel('Hour(UTC)')
plt.ylabel('Comments(m)')
plt.title('Comments in Hours of Day')
```

[11]:
```
Text(0.5, 1.0, 'Comments in Hours of Day')
```



While this analysis made some sense we can't exactly understand it because timezones are in UTC and it is worldwide.

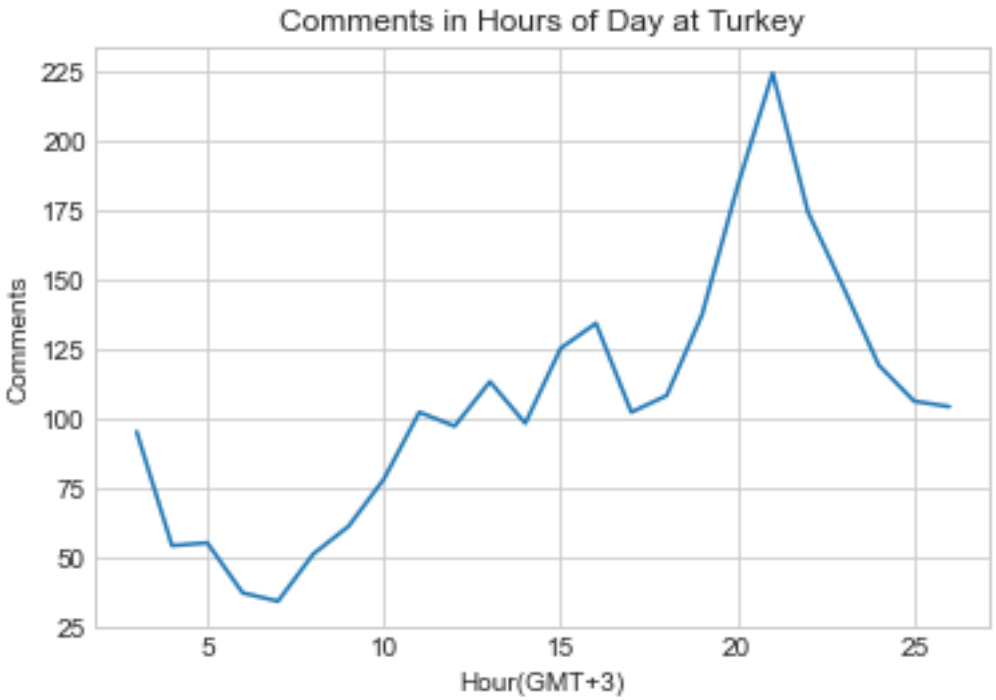
[12]:
```
if(check_if_table_exists('EDW','DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS')):
    print('Table EDW.DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS already exists')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS" AS
    WITH COMMENT_HOURLY AS(
    SELECT
    COMMENT_LANGUAGE_CODE AS LANGUAGE_CODE,
```

```
    COMMENT_LANGUAGE AS LANGUAGE,
    EXTRACT('HOUR' FROM DATE) AS UTC_HOUR,
    COUNT(*) AS COMMENT_COUNT
    FROM "EDW"."DWH_REDDIT_COMMENTS_DETAIL"
    WHERE 1=1
    GROUP BY
    COMMENT_LANGUAGE_CODE,
    COMMENT_LANGUAGE,
    EXTRACT('HOUR' FROM DATE)
    )
    SELECT
    CH.*,
    SUM(COMMENT_COUNT) OVER(PARTITION BY LANGUAGE_CODE) AS TOTAL_COMMENT_COUNT,
    COMMENT_COUNT/SUM(COMMENT_COUNT) OVER(PARTITION BY LANGUAGE_CODE) AS␣
 ↪PERCENTAGE
    FROM COMMENT_HOURLY CH;
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS created in␣
 ↪{execute_time} seconds".format(execute_time=end_time-start_time))
```

```
Table EDW.DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS already exists
```

[13]:
```
sql_command = """
SELECT
*
FROM "EDW"."DWH_REDDIT_LANGUAGE_HOURLY_COMMENTS"
WHERE LANGUAGE_CODE = 'tr';
"""
df_hourly_language_comments = pd.read_sql(sql_command,conn)
df_hourly_language_comments['percentage'] =␣
 ↪round(df_hourly_language_comments['percentage'],ndigits=4)*100
df_hourly_language_comments['gmt_p3_hour'] =␣
 ↪df_hourly_language_comments['utc_hour']+3%24

plt.plot(df_hourly_language_comments['gmt_p3_hour'],␣
 ↪df_hourly_language_comments['comment_count'])
plt.xlabel('Hour(GMT+3)')
plt.ylabel('Comments')
plt.title('Comments in Hours of Day at Turkey')
```

[13]:
```
Text(0.5, 1.0, 'Comments in Hours of Day at Turkey')
```



We can clearly see that around 17:00 there is a huge increase of comments, this is probably caused by people leaving from work/school and have free time.

At 21:00 comment count peaks at it's highest then starts to decline because people starting to sleep.

8.3 End Goal - Smart Marketing Analytics

For our end goal we are going split every comment word by word and assume we are showing ads to our customers based on their comments and we sell every 5% of the ads we show to our customers. We must **estimate** how much profit we can get based on our predictions.

[14]:
```
if(check_if_table_exists('EDW','TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD') or␣
 ↪check_if_table_exists('EDW','DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD')):
    print('Table EDW.TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD or EDW.
 ↪DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD already exists')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD" AS
    SELECT
    subreddit,
    comment_language,
    regexp_split_to_table(comment, '\s+(\w\s+)*') AS WORD
    FROM "EDW"."DWH_REDDIT_COMMENTS_DETAIL"
```

```
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD created in␣
 ↪{execute_time} seconds".format(execute_time=end_time-start_time))
```

```
Table EDW.TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD or
EDW.DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD already exists
```

After our analysis we are just going to drop this table because it's size is 98,8GB(transformed in 18.5 min)

[15]:
```
if(check_if_table_exists('EDW','DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD')):
    print('Table EDW.DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD already exists')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD" AS
    SELECT
    subreddit,
    comment_language,
    word,
    COUNT(*) AS word_count
    FROM "EDW"."TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD"
    GROUP BY subreddit,
    comment_language,
    word;
    DROP TABLE "EDW"."TRF_REDDIT_SUBREDDIT_LANGUAGE_WORD";
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD created in␣
 ↪{execute_time} seconds".format(execute_time=end_time-start_time))
```

```
Table EDW.DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD already exists
```

Transforming this dataset took around 1 hour.

[16]:
```
if(check_if_table_exists('EDW','DWH_ELECTRONIC_PRICES')):
    print('Table EDW.DWH_ELECTRONIC_PRICES already exists.')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_ELECTRONIC_PRICES"
    (
    key text,
    max_price numeric
    );
```

```
    ALTER TABLE "EDW"."DWH_ELECTRONIC_PRICES"
    OWNER to postgres;
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_ELECTRONIC_PRICES created in {execute_time} seconds.".
 ↪format(execute_time=end_time-start_time))

cur.execute('SELECT COUNT(*) FROM "EDW"."DWH_ELECTRONIC_PRICES"')
table_count = cur.fetchone()[0]
data_frame_count = len(keys_pricing_df.index)

if(table_count<data_frame_count):
    start_time = math.trunc(time.time())
    keys_pricing_df['key'] = keys_pricing_df['key'].str.replace("'",'')
    df_column_conversation(keys_pricing_df, 'key', 'text')
    execute_mogrify(conn,keys_pricing_df,"EDW","DWH_ELECTRONIC_PRICES")
    end_time = math.trunc(time.time())
    print("Data import to EDW.DWH_ELECTRONIC_PRICES completed in {execute_time}␣
 ↪seconds.".format(execute_time=end_time-start_time))
else:
    print("Data already imported to EDW.DWH_ELECTRONIC_PRICES")
```

```
Table EDW.DWH_ELECTRONIC_PRICES already exists.
Data already imported to EDW.DWH_ELECTRONIC_PRICES
```

[17]:
```
if(check_if_table_exists('EDW','DWH_TOTAL_PRICE_BY_WORDS')):
    print('Table EDW.DWH_TOTAL_PRICE_BY_WORDS already exists.')
else:
    start_time = math.trunc(time.time())
    cur.execute("""
    CREATE TABLE "EDW"."DWH_TOTAL_PRICE_BY_WORDS" AS
    SELECT
    SLW.*,
    ROUND(PRC.MAX_PRICE) AS PRICE_PER_ITEM,
    ROUND(SLW.WORD_COUNT*PRC.MAX_PRICE) AS TOTAL_PRICE
    FROM "EDW"."DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD" SLW
    INNER JOIN "EDW"."DWH_ELECTRONIC_PRICES" PRC ON (SLW.word = PRC.key);
    """)
    end_time = math.trunc(time.time())
    cur.execute('COMMIT;')
    print("Table EDW.DWH_TOTAL_PRICE_BY_WORDS created in {execute_time} seconds.
 ↪".format(execute_time=end_time-start_time))
```

```
Table EDW.DWH_TOTAL_PRICE_BY_WORDS already exists.
```

[18]:
```
cur.execute("""
SELECT
```

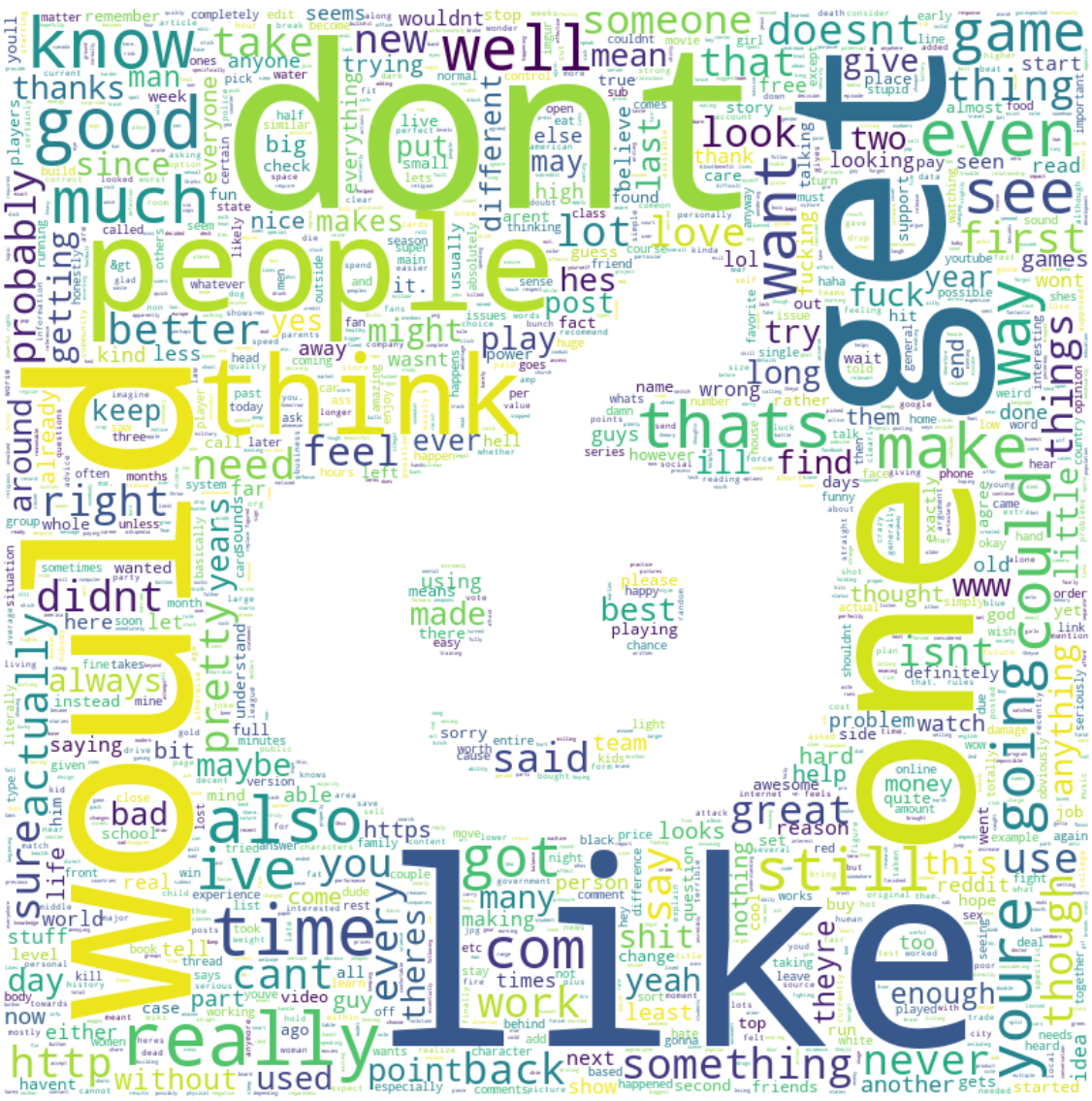
```
SUM(TOTAL_PRICE) AS TOTAL_PRICE
FROM "EDW"."DWH_TOTAL_PRICE_BY_WORDS"
ORDER BY SUM(TOTAL_PRICE) DESC
LIMIT 200;
""")
total_price = float(cur.fetchone()[0])
print("""From this analysis we could potential sell {total_price:,}$ of value␣
 ↪items.
If we assume we can sell only 5% items from customers we show ad based on their␣
 ↪comments we could sell {estimated_price_value_from_ads:,}$ of value.
However this doesn't calculates net gain or sites gain from profit if we assume␣
 ↪manufacturers get %20 percent profit per item and each successful purchase␣
 ↪is shared with website that shows the ad by 1% we can assume that website␣
 ↪would gain {website_profit:,}$ monthly.""".format(total_price=total_price,␣
 ↪estimated_price_value_from_ads=round(total_price*0.
 ↪05),website_profit=round(total_price*0.05*0.2*0.01)))
```

```
From this analysis we could potential sell 103,426,338,391.0$ of value items.
If we assume we can sell only 5% items from customers we show ad based on their
comments we could sell 5,171,316,920$ of value.
However this doesn't calculates net gain or sites gain from profit if we assume
manufacturers get %20 percent profit per item and each successful purchase is
shared with website that shows the ad by 1% we can assume that website would
gain 10,342,634$ monthly.
```

[19]:
```
from wordcloud import WordCloud
from PIL import Image
import os.path
import matplotlib.image as mpimg

def check_if_file_exists(filepath):
    return os.path.isfile(filepath)

if(not(check_if_file_exists('images/word_cloud_by_count_all.png'))):
    sql_command = """
    SELECT
    WORD,
    SUM(WORD_COUNT) AS WORD_COUNT
    FROM "EDW"."DWH_REDDIT_SUBREDDIT_LANGUAGE_WORD"
    WHERE 1=1
    AND TRIM(WORD) IS NOT NULL
    AND LENGTH(TRIM(WORD)) > 2
    AND NOT(TRIM(WORD) ~ '^([0-9]+[.]?[0-9]*|[.][0-9]+)$')
    GROUP BY WORD
    ORDER BY SUM(WORD_COUNT) DESC;
    """
    df_word_cloud = pd.read_sql(sql_command,conn)
```

```
    reddit_mask = np.array(Image.open('images/reddit.png'))
    word_cloud_data = dict(zip(df_word_cloud['word'].tolist(),␣
↪df_word_cloud['word_count'].tolist()))

    wc = WordCloud(background_color='white', width=8000, height=4000,␣
↪max_words=10000, mask=reddit_mask).generate_from_frequencies(word_cloud_data)
    wc.to_file('images/word_cloud_by_count_all.png')
Image.open('images/word_cloud_by_count_all.png')
```

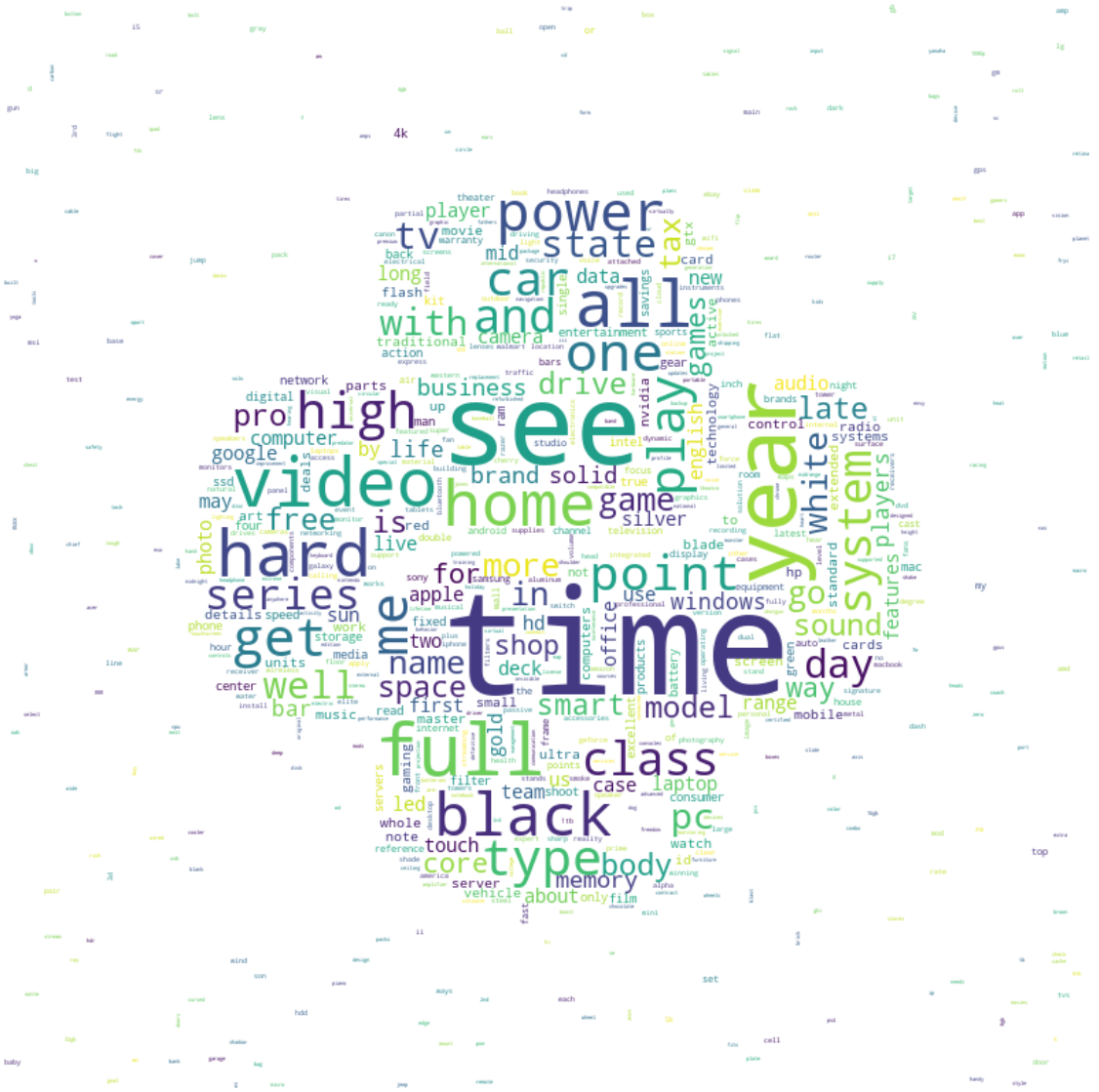
[19]:



Word cloud based on word counts where words are more than 2 characters and not numeric.

[20]:
```
if(not(check_if_file_exists('images/word_cloud_by_value_all.png'))):
    sql_command = """
    SELECT
    WORD,
    SUM(TOTAL_PRICE) AS TOTAL_PRICE
    FROM "EDW"."DWH_TOTAL_PRICE_BY_WORDS"
    GROUP BY WORD
    ORDER BY SUM(TOTAL_PRICE) DESC;
    """
    df_word_cloud = pd.read_sql(sql_command,conn)

    python_mask = np.array(Image.open('images/python.jpg'))
    word_cloud_data = dict(zip(df_word_cloud['word'].tolist(),␣
 ↪df_word_cloud['total_price'].tolist()))

    wc = WordCloud(background_color='white', width=8000, height=4000,␣
 ↪max_words=10000, mask=python_mask).generate_from_frequencies(word_cloud_data)
    wc.to_file('images/word_cloud_by_value_all.png')
Image.open('images/word_cloud_by_value_all.png')
```

[20]:



Word cloud based on total price.

[21]:
```
#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
28. Print number with commas as thousands separators
29. How to create a wordcloud according to frequencies in a pandas dataframe
30. Choosing the right colormaps