Amazon Kindle Reviews

Maddie Warndorf
5/2/19
EMSE 6586

Overview of Dataset

- Dataset was downloaded from Kaggle.
- Product reviews from Amazon Kindle Store from May 1996 - July 2014.
- Review limits: each reviewer has at least 5 reviews and each product has at least 5 reviews.
- There are 982,619 reviews, 68,223 reviewers, and 61,934 products in this dataset.

Columns

- `asin` - ID of the product, like B000FA64PK
- `helpful` - helpfulness rating of the review - example: 2/3.
- `overall` - rating of the product.
- `reviewText` - text of the review (heading).
- `reviewTime` - time of the review (raw).
- `reviewerID` - ID of the reviewer, like A3SPTOKDG7WBLN
- `reviewerName` - name of the reviewer.
- `summary` - summary of the review (description).
- `unixReviewTime` - unix timestamp.

MongoDB

Importing Dataset into MongoDB

```
Maddies-MacBook-Pro-3:~ maddiewarndorf$ mongoimport --db MMWFinalProject --collection kindle_review  --drop --file ~/Downloads/kin
dle_reviews.json
2019-04-02T20:43:22.726-0400    connected to: localhost
2019-04-02T20:43:22.727-0400    dropping: MMWFinalProject.kindle_review
2019-04-02T20:43:25.721-0400    [##......................] MMWFinalProject.kindle_review        92.8MB/789MB (11.8%)
2019-04-02T20:43:28.720-0400    [#####...................] MMWFinalProject.kindle_review        181MB/789MB (22.9%)
2019-04-02T20:43:31.720-0400    [########................] MMWFinalProject.kindle_review        268MB/789MB (33.9%)
2019-04-02T20:43:34.724-0400    [##########..............] MMWFinalProject.kindle_review        349MB/789MB (44.2%)
2019-04-02T20:43:37.722-0400    [#############...........] MMWFinalProject.kindle_review        444MB/789MB (56.2%)
2019-04-02T20:43:40.724-0400    [################........] MMWFinalProject.kindle_review        537MB/789MB (68.1%)
2019-04-02T20:43:43.720-0400    [####################....] MMWFinalProject.kindle_review        630MB/789MB (79.7%)
2019-04-02T20:43:46.720-0400    [######################...] MMWFinalProject.kindle_review        723MB/789MB (91.6%)
2019-04-02T20:43:48.705-0400    [########################] MMWFinalProject.kindle_review        789MB/789MB (100.0%)
2019-04-02T20:43:48.705-0400    imported 982619 documents
Maddies-MacBook-Pro-3:~ maddiewarndorf$
```

The dataset stored as kindle_reviews.json was imported to MongoDB via my Terminal.

- EMSE6586mmwFinal (4)
  - System
  - MMWFinalProject
    - Collections (1)
      - kindle_review
    - Functions
    - Users
  - config

Welcome | db.getCollection('kindle_review').find({}

EMSE6586mmwFinal | localhost:27017 | MMWFinalProject

```
db.getCollection('kindle_review').find({})
```

kindle_review | 0.044 sec. | 0 | 50

Key	Value	Type
(1) ObjectId("5ca401aae258e...	{ 10 fields }	Object
_id	ObjectId("5ca401aae258e377db2bc...	ObjectId
reviewerID	A1FV0SX13TWVXQ	String
asin	B000F83SZQ	String
reviewerName	Elaine H. Turley "Montana Songbird"	String
helpful	[2 elements]	Array
[0]	1	Int32
[1]	1	Int32
reviewText	I'd never read any of the Amy Brewst...	String
overall	5.0	Double
summary	I really liked it.	String
unixReviewTime	1392768000	Int32
reviewTime	02 19, 2014	String
(2) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(3) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(4) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(5) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(6) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(7) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(8) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(9) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(10) ObjectId("5ca401aae258...	{ 10 fields }	Object
(11) ObjectId("5ca401aae258e...	{ 10 fields }	Object
(12) ObjectId("5ca401aae258...	{ 10 fields }	Object
(13) ObjectId("5ca401aae258...	{ 10 fields }	Object

MySQL

Setting Up Information for MySQL Database

(1) ObjectId("5ca401aae258e...	{ 10 fields }	Object
_id	ObjectId("5ca401aae258e377db2bc...	ObjectId
reviewerID	A1FV0SX13TWVXQ	String
asin	B000F83SZQ	String
reviewerName	Elaine H. Turley "Montana Songbird"	String
helpful	[2 elements]	Array
[0]	1	Int32
[1]	1	Int32
reviewText	I'd never read any of the Amy Brewst...	String
overall	5.0	Double
summary	I really liked it.	String
unixReviewTime	1392768000	Int32
reviewTime	02 19, 2014	String

Reviewer

Review

Product

Setting Up Information for MySQL Database

Reviewer

reviewer_id

reviewer_name

*review_count

Review

created_time
unix_created_time
review_id
reviewer_id
helpful_rating
product_asin
text
*truncated
summary
product_rating

Product

product_asin

*review_count

*product_ratingavg

The attributes that are * indicate new attributes created by querying MongoDB to add additional information.

Final Tables Created from Set Up

reviewers

PK reviewer_id VARCHAR(225)
reviewer_name VARCHAR(225)
review_count INT

```
"""Create reviewer table to track Kindle Reviewers"""
make_reviewer_table = """CREATE TABLE reviewers(
                        reviewer_id VARCHAR(255),
                        reviewer_name VARCHAR(255),
                        review_count INT,
                        PRIMARY KEY(reviewer_id));"""

cursor.execute(make_reviewer_table)
connection.commit()
```

products

PK product_asin VARCHAR(255)
review_count INT
product_ratingavg FLOAT

```
"""Create product table to track Kindle products"""
make_product_table = """CREATE TABLE products(
                        product_asin VARCHAR(255),
                        product_ratingavg FLOAT,
                        review_count INT,
                        PRIMARY KEY(product_asin));"""

cursor.execute(make_product_table)
connection.commit()
```

reviews

created_time DATE
unix_created_time VARCHAR(225)
PK review_id VARCHAR(225)
FK reviewer_id VARCHAR(225)
helpful_rating VARCHAR(225)
FK product_asin VARCHAR(225)
text VARCHAR(500)
truncated BOOL
summary VARCHAR(225)
product_rating INT

```
"""Create review table to track Kindle Reviews"""
make_review_table = """CREATE TABLE reviews(
                        created_time DATE,
                        unix_created_time VARCHAR(225),
                        review_id VARCHAR(255),
                        reviewer_id VARCHAR(255),
                        helpful_rating VARCHAR(225),
                        product_asin VARCHAR(255),
                        text VARCHAR(500),
                        truncated BOOL,
                        summary VARCHAR(225),
                        product_rating INT,
                        PRIMARY KEY(review_id),
                        FOREIGN KEY (reviewer_id)
                            REFERENCES reviewers(reviewer_id)
                            ON DELETE CASCADE
                        FOREIGN KEY (product_asin)
                            REFERENCES products(product_asin)
                            ON DELETE CASCADE);"""

cursor.execute(make_review_table)
connection.commit()
```

Importing MongoDB Data into MySQL Database

- Used the SQL_Writer class given from Professor Klein.
- Created three other Python Classes. One for each table.

```python
class SQL_Writer():
    """This class is used to assist the insertion of objects into a MySQL database"""

    def __init__(self, cursor, connection):
        self.cursor = cursor
        self.conn = connection


    def insert(self, objs):
        """Inserts a list of objects into the given connection
        Args:
            objs (list) - list of SQL helper objects
        """

        for ix, obj in enumerate(objs):
            if not self.check_existence(obj):
                try:
                    if 'is_valid' in dir(obj):
                        if not self.check_validity(obj): continue
                    self.cursor.execute(obj.get_insert_query(), obj.get_values())
                except:
                    print(obj.get_insert_query() % obj.get_values())
                    raise

            if ix % 100 == 0:
                self.conn.commit()
        self.conn.commit()


    def check_validity(self, obj):
        """Uses object's built in validity query to check object validity.
        This is primarily focused around if an insert would be successful.
        Args:
            obj (SQL Helper Class) - Object to check for validity in the database
        """

        if self.cursor.execute(obj.is_valid()) < 1:
            return False
        return True


    def check_existence(self, obj):
        """Determine if the given object already exists
        Args:
            obj (SQL Helper Class) - Object to check existence
        """

        if self.cursor.execute(obj.existence_query()) < 1:
            return False
        return True
```

Reviewers Class

Reviewers

reviewer_id
reviewer_name
*review_count

```python
"""This would be done by taking unique reviewerID"""

class Reviewer():
    def __init__(self, reviewer, format='json'):
        """Load in json data into our object
        Args:
            reviewer - serialized object
            format (str) - what format is the serialized object
        """
        if format == 'json':
            self.load_json(reviewer)

    def load_json(self, reviewer):
        """Load data from JSON object
        Args:
            reviewer (dict) - json of object
        """
        self.reviewer_id = reviewer['reviewerID']
        try:
            self.reviewer_name = reviewer['reviewerName']
        except:
            self.reviewer_name = 'Null'
        self.review_count = collection.find({'reviewerID': self.reviewer_id}).count()

    def get_values(self):
        """Get the values used for inseritng a SQL record
        Returns:
            tuple - tuple in ordered format for SQL table
        """
        values = (self.reviewer_id, self.reviewer_name, self.review_count)

        return values

    def get_insert_query(self):
        """Get the string SQL insert statement
        Returns:
            str - insert statement
        """
        reviewer_insert = f"""INSERT INTO reviewers VALUES (%s, %s, %s)"""
        return reviewer_insert

    def existence_query(self):
        """Checks if the object already exists in the database
        Returns:
            str - existence query
        """
        reviewer_exists = "SELECT count(reviewer_id) FROM reviewers WHERE reviewer_id =
        return reviewer_exists
```

Products Class

Products

product_asin

*review_count

*product_ratingavg

```python
"""This is for the product"""
class Product():
    def __init__(self, product, format='json'):
        """Load in json data into our object
        Args:
            product - serialized object
            format (str) - what format is the serialized object
        """
        if format == 'json':
            self.load_json(product)

    def load_json(self, product):
        """Load data from JSON object
        Args:
            product (dict) - json of object
        """
        self.product_asin = product["asin"]
        self.product_rating_avg = self.finding_avg_rating()
        self.review_count = collection.find({'asin': self.product_asin}).count()

    def finding_avg_rating(self):
        """Get the average product rating from all the reviews
        Returns:
            avg_rating - average rating of product
        """
        answer = collection.aggregate([{"$match" : {"asin" : self.product_asin}},
                                       {"$group" : {"_id": "$asin", "average" :
                                                    {"$avg" : "$overall"}}}])
        an = list(answer)
        ans = an[0]
        avg_rating =  ans['average']
        return avg_rating

    def get_values(self):
        """Get the values used for inseritng a SQL record
        Returns:
            tuple - tuple in ordered format for SQL table
        """
        values = (self.product_asin, self.product_rating_avg, self.review_count)

        return values

    def get_insert_query(self):
        """Get the string SQL insert statement
        Returns:
            str - insert statement
        """
        product_insert = f"""INSERT INTO products VALUES (%s, %s, %s)"""
        return product_insert
    def existence_query(self):
        """Checks if the object already exists in the database
        Returns:
            str - existence query
        """
        product_exists = "SELECT product_asin FROM products WHERE product_asin = '" +
        return product_exists
```

Reviews

Reviews

created_time
unix_created_time
review_id
reviewer_id
helpful_rating
product_asin
text
*truncated
summary
product_rating

```python
"""This is for the review"""
class Review():
    def __init__(self, review, format='json'):
        """Load in json data into our object
        Args:
            review - serialized object
            format (str) - what format is the serialized object
        """
        if format == 'json':
            self.load_json(review)

    def load_json(self, review):
        """Load data from JSON object
        Args:
            review (dict) - json of object
        """
        self.created_at = parse(review['reviewTime'])
        self.unix_created_time = review['unixReviewTime']
        rvid = review["_id"]
        self.review_id = str(rvid)
        self.reviewer_id = review["reviewerID"]

        first_help = review.get('helpful')[0]
        second_help = review.get('helpful')[1]

        self.helpful_rating = f'{first_help}/{second_help}'

        self.product_asin = review["asin"]

        if len(review['reviewText']) > 500:
            text = review['reviewText']
            self.text = text[0:500]
            self.truncated = True
        else:
            self.text = review['reviewText']
            self.truncated = False

        if len(review['summary']) > 225:
            summary = review['summary']
            self.summary = summary[0:225]
        else:
            self.summary = review['summary']

        self.product_rating = review['overall']
```

FinalProjectEMSE6586 | Migration | EMSE6586mmwFinal (finalproject) | EMSE6586mmwFinal (

Administration | Schemas | Query 1 | reviewers | reviewers | products

SCHEMAS

Filter objects

- finalproject
  - Tables
    - products
    - reviewers
  - Views
  - Stored Procedures
  - Functions
- sys

Object Info | Session

Table: reviewers

Columns:

reviewer_id	varchar(255) PK
reviewer_nam	varchar(255)

Limit to 1000 rows

```
1 • SELECT * FROM finalproject.reviewers;
```

100% 1:1

Result Grid Filter Rows: Search Edit:

reviewer_id	reviewer_name	review_count
A00085083TSCV82430YT4	Ja'netHayes	6
A0010876CNE3ILIM9HV0	JassyR	6
A00207583M69Q8KX3BOFQ	debra wolstenholme	6
A002359833QJM7OQHCXWY	Dawn Peterson	14
A00328401T70RFN4P1IT6	Susan	5
A00463782V7TKAP9EMNL	Diane	25
A006458827ALF2J23JJTO	Matthew Fudge	15
A0089401235VSN3Z6F3HK	Draven Valverde	9
A0090953K7LNUG6UPMI6	Alice Herde	5
A0092581WFYQNV4KMUZ3	Jody	5
A0093003C4D9BVJ1YFA	Kindle Customer	5
A0099735VDZ3HDCAAYKL	M.Asa "MELVENA A…	109
A01024073VQNJIY6SIY50	Christine Zamora	8
A010971113OD625HDB6X8	BookaddictBieke "Ist…	5
A01473781YX8UGSS8JYKH	alan puga	5

reviewers 1

SCHEMAS

Filter objects

- finalproject
  - Tables
    - products
      - Columns
        - product_asin
        - product_ratingavg
        - review_count
      - Indexes
      - Foreign Keys
      - Triggers
    - reviewers
    - reviews
  - Views
  - Stored Procedures
  - Functions

Object Info | Session

Table: reviews

Columns:

created_time	date
unix_created_ti	varchar(225)

```
1 • SELECT * FROM finalproject.products;
```

100% 1:1

Result Grid Filter Rows: Search

product_asin	product_ratingavg	review_count
B000F83SZQ	4.25	8
B000FA64PA	4.2	5
B000FA64PK	4.375	8
B000FA64QO	3.8	5
B000FBFMVG	4.33333	9
B000FC1BN8	3.44444	9
B000FC1TG2	5	6
B000FC26RI	4.16667	6
B000FC2MB8	4.625	8
B000FDJ0FS	2.46667	15
B000GFK7L6	3.40909	22
B000HA4FKY	4	6
B000HC48T0	3.85714	7
B000JMKNQ0	3.8	10
B000JMKU0Y	3.92857	14
B000JMKX4W	3.54545	11

products 1

Administration | Schemas | Query 1 | reviewers | reviewers | products | products | reviews

SCHEMAS

Filter objects

- reviewers
- reviews
  - Columns
    - created_time
    - unix_created_time
    - review_id
    - reviewer_id
    - helpful_rating
    - product_asin
    - text
    - truncated
    - summary
    - product_rating
  - Indexes
  - Foreign Keys

Object Info | Session

Table: reviews

Columns:

created_time	date
unix_created_time	varchar(225)

Limit to 1000 rows

```
1 •  SELECT * FROM finalproject.reviews;
```

100% 1:1

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

created_time	unix_created_time	review_id	reviewer_id	helpful_rating	product_asin	text
2014-01-06	1388960400	5ca401aae258e377db2bc33b	ANUN05A9LIJEQ	2/2	B000F83SZQ	This book is a rel
2014-04-04	1396569600	5ca401aae258e377db2bc33c	A795DMNCJILA6	2/2	B000F83SZQ	This was a fairly i
2014-05-05	1399248000	5ca401aae258e377db2bc33d	A1F6404F1VG29J	0/0	B000F83SZQ	I enjoy vintage bo
2014-03-19	1395187200	5ca401aae258e377db2bc33e	A3SPTOKDG7WBLN	0/1	B000F83SZQ	If you like period
2014-05-26	1401062400	5ca401aae258e377db2bc33f	A1RK2OCZDSGC6R	0/0	B000F83SZQ	A beautiful in-dep
2014-03-22	1395446400	5ca401aae258e377db2bc340	A3DE6XGZ2EPADS	1/1	B000F83SZQ	Never heard of A
2013-10-11	1381449600	5ca401aae258e377db2bc341	A1UG4Q4D3OAH3A	0/0	B000FA64PA	Darth Maul worki
2014-06-10	1402358400	5ca401aae258e377db2bc342	A2HSAKHC3IBRE6	0/0	B000F83SZQ	I enjoyed this one
2014-01-27	1390780800	5ca401aae258e377db2bc343	A1ZT7WV0ZUA0OJ	0/0	B000FA64PA	I think I have this
2011-02-13	1297555200	5ca401aae258e377db2bc344	AQZH7YTWQPOBE	0/0	B000FA64PA	This is a short sto
2011-09-17	1316217600	5ca401aae258e377db2bc345	A2ZFR72PT054YS	0/0	B000FA64PA	Title has nothing
2013-12-31	1388448000	5ca401aae258e377db2bc346	A2QK1U70OJ74P	0/0	B000FA64PA	Well written. Inter
2013-05-12	1368316800	5ca401aae258e377db2bc347	A3H8PE1UFK04JZ	0/0	B000FA64PK	I am not for sure
2009-04-16	1239840000	5ca401aae258e377db2bc348	A38Z3Q6DTDIH9J	4/4	B000FA64PK	Another well writt
2012-03-15	1331769600	5ca401aae258e377db2bc349	A3SZMGJMV0G16C	0/0	B000FA64PK	Troy Denning's n
2013-10-29	1383004800	5ca401aae258e377db2bc34a	A1UG4Q4D3OAH3A	0/0	B000FA64PK	Great read enjoy
2013-12-31	1388448000	5ca401aae258e377db2bc34b	A2QK1U70OJ74P	0/0	B000FA64PK	Excellent! Very w

reviews 1

Apply | Revert

Result Grid | Form Editor | Field Types | Query Stats

Analysis

Top 10 Reviewers with the Most Reviews

```python
#Finding top 10 Reviewers who wrote the most reviews
res = cursor.execute("SELECT reviewer_id, review_count FROM reviewers ORDER BY review_count DESC LIMIT 10")
top_10_reviewers = cursor.fetchall()


print("__Number of Reviews Per User__")
for i in top_10_reviewers:
    for k, v in i.items():
        try:
            int(v)
            count = v
        except:
            user = v
    print("%s: %d" % (user, count))

```

```
__Number of Reviews Per User__
A13QTZ8CIMHHG4: 1173
A2WZJDFX12QXKD: 1007
A320TMDV6KCFU: 847
A3PTWPKPXOG8Y5: 789
A1JLU5H1CCENWX: 782
A37LY77Q2YPJVL: 744
A3A7FF87LEVCQ1: 658
A2JZCZYHNQHSCP: 625
A328S9RN3U5M68: 587
A2YJ8VP1SSHJ7: 579
```

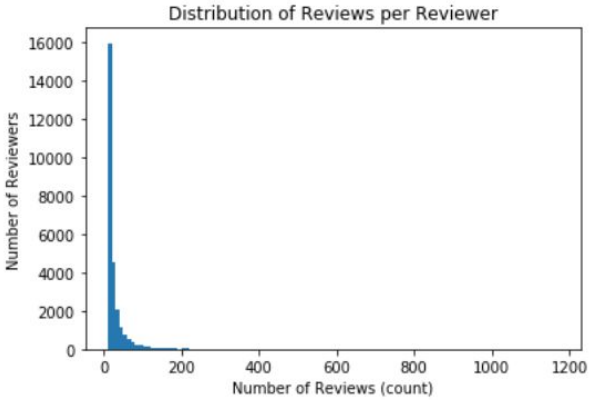
The Most Reviewed Reviews

Using the Helpful attribute in MongoDB it shows how many people found it useful out of the total number of people that reviewed that review. Using the second number in the Helpful attribute dictionary, I was able to determine the most reviewed reviews.

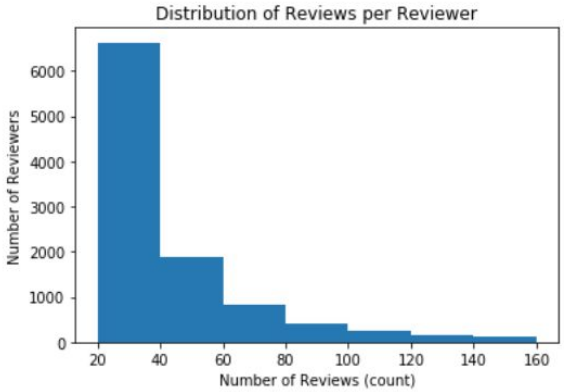
```
#Finding most reviewed review
most_reviewed = collection.find({'helpful.1': {'$gt':100}})
import pandas as pd
import numpy as np

review_id = []
reviewer_id = []
num_of_reviews = []
for rev in most_reviewed:
    rev_id = rev['_id']
    review_id.append(rev_id)
    rever_id = rev['reviewerID']
    reviewer_id.append(rever_id)
    num = rev['helpful']
    num_tot = num[1]
    num_of_reviews.append(num_tot)


df = pd.DataFrame()
df["ReviewID"] = review_id
df["NumberOfReviews"] = num_of_reviews

df.sort_values("NumberOfReviews", ascending=False)
```

	ReviewID	NumberOfReviews
1	5ca401aee258e377db2df695	2537
0	5ca401b2e258e377db301da6	1834
3	5ca401ade258e377db2d2951	1773
9	5ca401b0e258e377db2ee219	1575
5	5ca401b0e258e377db2f38bc	922
1	5ca401b2e258e377db301e2c	868
9	5ca401aee258e377db2df50c	692
4	5ca401afe258e377db2e83d8	668
5	5ca401ade258e377db2d813e	587
1	5ca401abe258e377db2be433	503
4	5ca401abe258e377db2c01a1	482

Distribution of Reviews per Reviewer



This first plot shows the full range of review count per reviewer.



This graph shows the close up of the most frequent number of review counts.

Correlation Between Swear Words and Product Review

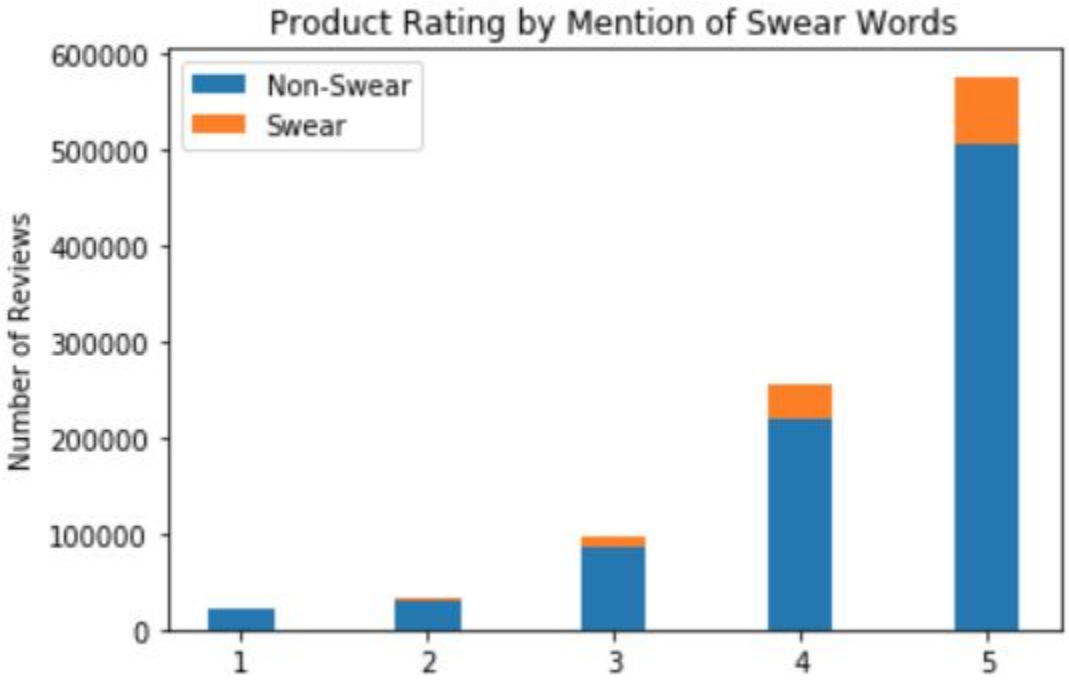
- Used regular expressions to pull reviews that mention a swear word.
  - I created a list of the top 7 swear words.
- Then created a new dataframe that had review id, reviewer id, number of swear words, and product rating.
  - Number of swear words ranged from 0 to 6.
- There were 124,560 unique reviews that mentioned one of the selected swear words.

	number_swear_words	product_review
number_swear_words	1.000000	0.004077
product_review	0.004077	1.000000

```
Product review mode for reviews mentioning a swear word: 5.0
Product review mode for reviews that did not mention a swear word: 5
Minimum number of swear words mentioned: 0
Maximum number of swear words mentioned: 6
```

```
Breakdown by Product Rating for Reviews that Mentioned Swear Words:
Total number of reviews: 124560
Number of products rated as 5: 70668
Number of products rated as 4: 36077
Number of products rated as 3: 11459
Number of products rated as 2: 3868
Number of products rated as 1: 2488


Breakdown by Product Rating for Reviews that Did Not Mentioned Swear Words:
Total number of reviews: 858037
Number of products rated as 5: 504578
Number of products rated as 4: 217933
Number of products rated as 3: 84734
Number of products rated as 2: 30262
Number of products rated as 1: 20530
```

Product Rating in Reviews by Mention of Swear Words



Final Thoughts on MongoDB vs. MySQL

MongoDB

Pros:

- Easy to import dataset
- ReviewText was not limited to characters

Cons:

- Hard to query general things like distribution of reviews by reviewer.
- Having to query through dictionaries

MySQL

Pros:

- Can add attributes such as review count while the data is being imported
- Being able to query individual tables

Cons:

- If the column type limits were set up wrong, it is hard to fix
  - Setting an attributes VARCHAR too low

Questions?