MSDS 420 Final Project Phase 1 – Analysis

Requirement 1: (10 Points) Inspect the provided datasets and run the provided **Extract_Amazon_Appliances_Reviews_data.ipynb.script**. Comment on the different fields (data types, NaN values, etc.) in the created CSV files (**Amazon_Appliances_Metadata.csv** and **Amazon_Appliances_Reviews.csv**) and what data preprocessing is needed to prepare the data to be loaded into a database. The asin - ID of the product is the common field between the two tables, do you think that is sufficient to cross-reference the data entries/rows in the two datasets?

Descriptions of columns in the files:

Review CSV:

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - ID of the product, e.g. 0000013714
- reviewerName - name of the reviewer
- helpful - helpfulness rating of the review, e.g. 2/3
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)

Appliance Metadata CSV:

- asin - ID of the product, e.g. 0000013714
- price – price of product e.g. 00.00
- imURL – website link to image of product
- description of product
- category – appliance category
- title – brand title of product
- brand – manufacturer of product
- related – items similar to product
- salesrank – sales performance ranking by category

To prepare the data to be uploaded into a database preprocessing steps have to occur.

1. Data Quality Assessment

   The Review CSV has 143,686 rows and 9 columns. Data types seems to be consistent within the columns, so there is no need to convert data values. The data types include strings (reviewID, asin, reviewtext, reviewername, and summary), integers(helpful, overall) timestamp(unixreviewtime) and date(reviewtime). Reviewer name has some inconsistencies with the permitted format of names rather than the standard “first name, last name” or vice versa reviews can be designated by nicknames, what seem to be usernames, not require letters to provide a name, or do not have a requirement to respond to the prompt at all. Most null values are coming from the reviewer name and review text columns. It would be best to make these descriptors consistent. Dropping the data is an option but it you may be ablet to determine the

reason why those were left blank. In addition, it would be most efficient to only include products that have over 500 reviews.

The Appliance Metadata CSV data types are consistent within the columns, so there is no need to convert data values. The data types include strings (description, asin, categories, titles, brand, related, sales rank), varchar max(imUrl), and float(price).

2. Data Cleaning

   The Review CSV data can be cleaned by, if possible, filling in missing values, and/or using a constant link N/A to fill the missing data. For example, data that has a completed asin column, overall review, and reviewerID can be supplemented with N/A values for the other columns as they allow for categorization with and query retrieval. There is also an unclear notation of review helpfulness as [x,y], changing this notation to a/b which will prevent any incorrect interpretation. In addition, removing punctuation, links, and unnecessary symbols.

   The Appliance Metadata CSV data can be cleaned by, if possible, filling in missing values for product price, if not providing N/A as a substitution to the missing data.

3. Data Reduction

   Review and Appliance Metadata CSV can undergo data elimination to provide a more concise dataset, remove missing values, and decrease the overall amount of data.

Using pandas, the data frame can be modified in a meaningful way that aids in data analysis tasks.

No, I do not believe the asin ID being the only common field between the two tables will be sufficient in cross-referencing entries in the two datasets. It would be more effective to have a common field that allows for descriptors and price to be used in search. For example, with the current fields if a query was generated that requested the dishwasher that was recently reviewed with a highest sales rank, requested a specific price range, and wanted a specific brand. The query would only be able to join on the asin ID. If that was unknown, there aren't any other fields to join to receive a result.

Requirement 2: *(10 Points) Create a table for a sample list of 3 queries (NOT listed in Requirements Specification Document) for every category listed below: o Quantitative Queries o Fuzzy/Exact-Search Queries o Hybrid-Search Queries*

```
CREATE TABLE 'Full Appliance Review'

reviewerID VARCHAR (255),
asin ID VARCHAR (255) NOT NULL,
description VARCHAR (255),
category VARCHAR (100),
brand VARCHAR (100),
helpful numerator INT,
helpful denominator INT,
reviewtext VARCHAR (255),
overall DECIMAL(3,1) CHECK (overall>=0 AND overall<=5)
summary VARCHAR (100)
unixreviewtime bigint(20),
reviewtime DATE
PRIMARY KEY ASIN ID
```

Quantitative Queries:

1. How many reviews for asin ID B000FPDHKA received an overall rating of 5?
2. How many reviews were submitted in January of 2014?
3. How many reviews received a helpfulness score?

Fuzz/Exact Searches:

1. Pull list of reviews for humidifiers where reviewers reported it **reduced humidity** and kept the space **dry**?
2. Get a list of product reviews where the product is **description is heavy duty** but the reviewers found it to feel **cheap**?
3. Get a list of product reviews where reviewers complained **Hoses** were **short**?

Hybrid Search Queries:

1. Get a list of reviews that are similar to this text: "This filter was a disappointment. The mesh is thin, has large holes, and did not keep the pool clean.
2. Get a list of reviews from 2014 that are similar to this text: "This water heater is horrible. It takes forever for the water to get hot, then when it does it is boiling! There is not in between."
3. Pull a list of reviews for the brand Leviton in 2013 where reviews are similar to this text: "I am satisfied with my purchase and this product. It worked very well."

Requirement 3: *(10 Points) Use ChatGPT and enter the* ***CoT*** *text (Section 5, Page 18 of* ***Final Project - Requirements Specification.pdf****) into ChatGPT prompt to get the recommended database that best meets the needs of every query you identified above. Create a table that has two columns: the query and the recommended database for it as printed by ChatGPT.*

-and-

Requirement 4: *(10 Points) Use ChatGPT to generate* ***code*** *for every query you identified above using the recommended database. Please see sample prompts and output in* ***Appendix A*** *of this document.*

ChatGPT included the codes in the initial response, it is shown in the ChatGPT response PDF

Quantitative Searches	
Query	**ChatGPT Response**

How many reviews for asin ID B000FPDHKA received an overall rating of 5?	Step 3: Use Postgres to execute the queries.
	
	Query 1 (Count of reviews for ASIN ID B000FPDHKA with an overall rating of 5):
	
	sql Copy code
	`SELECT COUNT(*)`
	`FROM reviews`
	`WHERE asin = 'B000FPDHKA' AND overall_rating = 5;`

How many reviews were submitted in January of 2014?	Query 2 (Count of reviews submitted in January 2014):
	
	sql Copy code
	`SELECT COUNT(*)`
	`FROM reviews`
	`WHERE submission_date >= '2014-01-01' AND submission_date < '2014-02-01';`

How many reviews received a helpfulness score?	Query 3 (Count of reviews with a helpfulness score):
	
	sql Copy code
	
	``sql
	SELECT COUNT(*)
	FROM reviews
	WHERE helpfulness_score IS NOT NULL;
	``
	
	These queries would efficiently retrieve the desired information from the Amazon product reviews dataset using Postgres.	

Fuzzy/Exact Searches	
Query	**ChatGPT Response**

Pull list of reviews for humidifiers where reviewers reported it **reduced humidity** and kept the space **dry**?	Step 3: Use Elasticsearch to execute the queries.
	
	Query 1 (List of reviews for humidifiers where reviewers reported it reduced humidity and kept the space dry):
	
	sql Copy code
	`GET /amazon_reviews/_search`
	`{`
	`  "query": {`
	`  "bool": {`
	`  "must": [`
	`  { "match": { "product_description": "humidifier" }},`
	`  { "match": { "review_text": "reduced humidity" }},`
	`  { "match": { "review_text": "kept the space dry" }}`
	`  ]`
	`  }`
	`  }`
	`}`

Get a list of product reviews where the product is **description is heavy duty** but the reviewers found it to feel **cheap**?	Query 2 (List of product reviews where the product description is "heavy duty" but reviewers found it to feel cheap):
	
	sql Copy code
	`GET /amazon_reviews/_search`
	`{`
	`  "query": {`
	`  "bool": {`
	`  "must": [`
	`  { "match": { "product_description": "heavy duty" }},`
	`  { "match": { "review_text": "feel cheap" }}`
	`  ]`
	`  }`
	`  }`
	`}`

Get a list of product reviews where reviewers complained **Hoses** were **short**?	Query 3 (List of product reviews where reviewers complained hoses were short):
	
	sql Copy code
	`GET /amazon_reviews/_search`
	`{`
	`  "query": {`
	`  "match": { "review_text": "hoses were short" }`
	`  }`
	`}`
	
	These Elasticsearch queries would efficiently retrieve the desired information from the Amazon product reviews dataset based on the specific text matching criteria provided in each query.

Hybrid Searches	
Query	**ChatGPT Response**

Get a list of reviews that are similar to this text: “This filter was a disappointment. The mesh is thin, has large holes, and did not keep the pool clean.	Step 3: Create queries using Elasticsearch for each scenario:
	
	Query 1 (List of reviews similar to provided text):
	
	json Copy code
	
	{
	"query": {
	"more_like_this": {
	"fields": ["review_text"],
	"like": "This filter was a disappointment. The mesh is thin, has large
	"min_term_freq": 1,
	"min_doc_freq": 1
	}
	}
	}

Get a list of reviews from 2014 that are similar to this text: “This water heater is horrible. It takes forever for the water to get hot, then when it does it is boiling! There is no in between.”	Query 2 (List of reviews from 2014 similar to provided text):
	
	json Copy code
	
	{
	"query": {
	"bool": {
	"must": [
	{"match": {"review_year": 2014}},
	{
	"more_like_this": {
	"fields": ["review_text"],
	"like": "This water heater is horrible. It takes forever for the
	"min_term_freq": 1,
	"min_doc_freq": 1
	}
	}
	]
	}
	}
	}

Get a list of reviews from 2013 that are similar to this text: “I am satisfied with my purchase and this product. It worked very well.”	Query 3 (List of reviews from 2013 similar to provided text):
	
	json Copy code
	
	{
	"query": {
	"bool": {
	"must": [
	{"match": {"review_year": 2013}},
	{
	"more_like_this": {
	"fields": ["review_text"],
	"like": "I am satisfied with my purchase and this product. It wo
	"min_term_freq": 1,
	"min_doc_freq": 1
	}
	}
	]
	}
	}
	}
	
	These queries are designed for Elasticsearch as it's optimized for similarity search and can efficiently handle queries involving text matching and similarity calculations.

Requirement 5: *(10 Points) Analyze, synthesize, and comment on the code (Postgres SQL statements) generated by ChatGPT for every* ***Quantitative Query*** *you identified above for completeness, correctness, and consistency.*

Query 1

```
#Count the number of reviews
SELECT COUNT(*)
FROM reviews
#Add conditions to 5 star ratings for the specific product/asin ID
WHERE asin = 'B000FPDHKA' AND overall_rating = 5;
```

The query is concise, straightforward, and easy to understand. The main specifications COUNT(*) and WHERE are used correctly to retrieve the correct value. Due to how concise the code it, it is easy to see where conditions can be added to the code.

Query 2

```
#Count the number of reviews
SELECT COUNT(*)
FROM reviews
#Add conditions that specify montth
WHERE date_part('month', review_date) = 1
#Add conditions that specify year
AND date_part('year', review_date) = 2014;
```

The query is easy to understand. Using data_part to split the review date into months and years allows for further conditions to be added, such as selecting reviews for the month of May for years 2013-2015.

Query 3

```
#Count the number of reviews
SELECT COUNT(*)
FROM reviews
#Add condition of list including reviews that received a helpfulness score
WHERE helpfulness_score IS NOT NULL;
```

The code is focuses on the requirement requested in the query, returning the number of reviews where a helpfulness score is provided. The NULL condition opposed to Zero, or Blank ensures that reviews where a helpfulness score is provided are counted.