Project Milestone 2

Scott Walters, Md Muhtasim Billah, Md Mahedi Hasan, Joseph Patten

1. Datasets and tools

a. Data model:

Our project relies upon a relational database management system (RDMS). The data focuses on user's reviews and comments on products. So, the utility of a RDMS will be helpful in managing, categorizing and preparing our dataset for our co-purchasing analysis algorithm. The data comes to us categorized, so the data is smoothly integrated into a relational database.

b. Datasets statistics:

The number of tuples corresponds to the number of reviews in the larger dataset. We have over `200` million tuples (`224,538,272`) representing the `10` attributes of each review. See below for the table defining the tuples. The complete data is `34 GB`. We are concerned about the size of this data and it is recommended to use a subset of this data that is a third the size (`14.3 GB`). This data covers `75,258,750` reviews. In addition to this data, there is metadata that has additional information pertaining to the items reviewed. Due to the significant time it takes to upload, parse and insert into the database, we have an estimate of just as many tuples as the reviews themselves. However, this metadata is separated into three tables - `Item`, `AlsoBuy` and `AlsoView` - to compartmentalize the relation. The queries for creating the relations are provided in Appendix A.

c. Parsing and Database Insertion:

The `PostgreSQL` RDMS was used to house the data locally before migrating it to the AWS EC2 instance. The tuples are inserted into the database via the python `psycopg2` library which allows us to create SQL statements to insert the tuples. To obtain the tuples, we were assisted with code that loads the data from the `.json.gz` files to pandas dataframe objects that nicely separate each tuple's attributes into their respective columns. From here, our program creates an insert statement string made of this information in an additional function that is quite simple and efficient. The python script for parsing the data is provided in Appendix B.

d. Estimated Loading Time:

Because of the vast amount of data, the transfer from json file to dataframe object takes considerable amount of time. The `Books.json.gz` file detailing `27 million` reviews has taken over two hours to transfer on Scott's 16 GB ram, 8 processor cpu. Despite this being in line with a 10 million tuple data set for a previous database class (using python and Posgres), we would like feedback as to the expectations on performing these tasks on the raw data. We estimate that it should take a `24 hours` of operation of our code to load the full dataset into Postgres.

2. Data Structure:

The dataset was collected in the form of semi-structured data (json files). Then then the data was parsed into structured data, (in the form of relational data) containing multiple relations. The details of the procedures have been provided in the previous section.

3. Plans for Milestone 3:

For milestone 3, we plan to develop and complete both the sequential and distributed algortihm based on Apache Spark engine on the AWS EC2 instance. Then we will work on applying the machine learning algorithm (collaborative filtering) to analyze the co-purchasing pattern in the Amazon data. The formal description of the algorithms used and and the related outcomes will be presented on the milestone 3 statement.

Appendix A

The following SQL queries have been used to create the relations metioned in section 1 (b).

```sql
CREATE TABLE Reviews (
    rating NUMERIC,
    verified Boolean,
    votes Integer,
    reviewTime CHAR (20),
    reviewerID CHAR (20),
    asin CHAR (20),
    reviewerName CHAR (300),
    reviewText CHAR (40000),
    summary CHAR (10000),
    style CHAR (20000),

    FOREIGN KEY (asin) REFERENCES Item (asin)
);

CREATE TABLE Item (
    asin CHAR (20) PRIMARY KEY,
    title CHAR (300),
    feature CHAR (10000),
    description CHAR (20000),
    price NUMERIC,
    salesRank CHAR (1000),
    brand CHAR (300),
    categories CHAR (10000)
);


CREATE TABLE AlsoBuy (
    asin CHAR (20),
    alsoBuy CHAR (20),
    FOREIGN KEY (asin) REFERENCES Item (asin),
    FOREIGN KEY (alsoBuy) REFERENCES Item (asin)
);

CREATE TABLE AlsoView (
    asin CHAR (20),
    alsoView CHAR (20),
    FOREIGN KEY (asin) REFERENCES Item (asin),
    FOREIGN KEY (alsoView) REFERENCES Item (asin)
);
```

Appendix B

The following python script has been used to parse the data

```
#code for parsing the data
import json
import psycopg2
import math
import pandas as pd
import gzip

# function from https://nijianmo.github.io/amazon/index.html#code
# data set webpage detailing how to obtain data
def parse(path):
  g = gzip.open(path, 'rb')
  for l in g:
    yield json.loads(l)


# function from https://nijianmo.github.io/amazon/index.html#code
# data set webpage detailing how to obtain data
def getDF(path):
  i = 0
  df = {}
  for d in parse(path):
    df[i] = d
    i += 1
  return pd.DataFrame.from_dict(df, orient='index')


# This function is from the database class, credit to Dr. Sakire
def cleanStr4SQL(s):
  return s.replace("'", "`").replace("\n", " ")

def cleanStr4Int(s):
  return s.replace(",", "").replace("$", "")


def insert_reviews(df):
  try:
    conn = psycopg2.connect("dbname='group1' user='postgres'host='localhost' password='sophie'")
    print('Connected to local database')
  except:
    print('Unable to connect to the database!')
    exit()
  cur = conn.cursor()
  inserted = 0
  for items in df.iloc:
    rating = str(items[0])
    if (rating == 'nan'):
      rating = '0'
    verified = str(items[1])
    votes = cleanStr4Int(str(items[9]))
    if (votes == 'nan'):
      votes = '0'
    reviewTime = str(items[2])
    reviewerID = str(items[3])
    asin = str(items[4])
    reviewerName = cleanStr4SQL(str(items[5]))
    reviewText = cleanStr4SQL(str(items[6]))
    summary = cleanStr4SQL((str(items[7])))
    style = cleanStr4SQL((str(items[10])))

    sql_str = "INSERT INTO Reviews (rating, verified, votes, reviewTime, reviewerID, asin, "\
    + "reviewerName, reviewText, summary, style) VALUES ('" + rating + "', '" + verified \
    + "', '" + votes + "', '" + reviewTime + "', '" + reviewerID + "', '" + asin  \
    + "', '" + reviewerName + "', '" + reviewText + "', '" + summary + "', '" + style + "');"

    try:
      cur.execute(sql_str)
    except:
      print("didn't work: " + sql_str)
      break
    conn.commit()
    inserted += 1
  return inserted



# There appears to be duplicates of item entries from the data set that is causing issues
def insert_metadata(df):
  try:
    conn = psycopg2.connect("dbname='group1' user='postgres'host='localhost' password='sophie'")
    print('Connected to local database')
  except:
    print('Unable to connect to the database!')
    exit()

  cur = conn.cursor()
  inserted = 0
  for items in df.iloc:
    asin = str(items['asin'])
    title = cleanStr4SQL(str(items[4]))
    feature = cleanStr4SQL(str(items[9]))
    if feature == '[]':
      feature = 'NA'
    description = cleanStr4SQL(str(items[2]))
    if description == '[]':
      description = 'NA'
    price = cleanStr4Int(str(items['price']))
    ## the 16th item has price as description or so it looks
    if price == 'nan' or price == '' or len(price) > 6:
      price = '0'
    salesRank = str(items[10])
    brand = cleanStr4SQL(str(items[8]))
    categories = str(items[0])
    if categories == '[]':
      categories = 'NA'

    sql_str = "INSERT INTO Item (asin, title, feature, description, price, salesRank, " + \
              "brand, categories) VALUES ('" + asin + "', '" + title + "', '" + \
              feature + "', '" + description + "', '" + price + "', '" + salesRank + "', '" + \
              brand + "', '" + categories + "');"

    try:
      cur.execute(sql_str)
    except:
      print("didn't work: " + sql_str)
      break
    conn.commit()

    if str(items['also_buy']) != '[]':
      for alsos in items['also_buy']:
        sql_str = "INSERT INTO AlsoBuy (asin, alsoBuy) VALUES ('" + asin + "', '" + str(alsos) + "');"
        try:
          cur.execute(sql_str)
        except:
          print("didn't work: " + sql_str)
          break
        conn.commit()

    if str(items['also_view']) != '[]':
      for alsos in items['also_view']:
        sql_str = "INSERT INTO AlsoView (asin, alsoView) VALUES ('" + asin + "', '" + str(alsos) + "');
        try:
          cur.execute(sql_str)
        except:
          print("didn't work: " + sql_str)
          break
        conn.commit()

    inserted += 1
  return inserted
```