Amazon Electronics Reviewer Value

Vinu Lakkur and Camaron Mangham

Background

Our original motivation was to gain a stronger understanding of consumer behavior for two categories of products - electronics and clothing - and how macroeconomic factors such as GDP/inflation and season tend to affect consumer behavior. The primary dataset, derived from a dataset of Amazon product reviews developed by researchers at UC San Diego[1], was utilized in a statistical analysis of marketing bias in product recommendations[3]. Our aim was to analyze and consider metrics including market basket analysis, user lifetime value, and larger trends in spend volume. As we explored the data we discovered limitations such as a lack of dollar values for products and inconsistent primary keys needed to obtain those values from the original dataset. This resulted in a shift in our objectives.

We focused our goal toward assessing **"reviewer lifetime value"** and larger trends in spend volume by analyzing reviews of Amazon purchases. The Amazon reviews dataset contains 233.1 million reviews across all categories of product, so we will only focus on a subset of the electronics category, which only contains about **6.7 million records** (the full electronics dataset contains almost 21 million records). The results of such analysis could provide insight into the characteristics of different customer groups that leave reviews for items. This information could also be compared and contrasted with other groups of customers such as those that do not leave reviews. In a business setting, such information could be valuable in order to determine how much business effort to put toward either capturing or communicating with certain sets of reviewers/customers. Overall, the motivation for this analysis is inspired by the common practice of determining customer lifetime value (CLV) to understand customer segments and build customer relationships[3-4], and a desire to apply big data and data mining concepts.

Key Findings and Challenges

- Storing this large data in memory was not viable, and required the leveraging of distributed data processing via PySpark.
- Compressed files must be decompressed to take advantage of distributed processing via PySpark.
- A majority of the reviews are from 2014-2017 which marks a period of rapid growth for Amazon.
- Median spending by reviewers is around $11 on average. Further analysis could be performed regarding price category and review count correlation.
- Information about a reviewers first Amazon review gives us marginal predictive value regarding their Lifetime Value as a Reviewer
- Initial analysis suggests there is potential for further feature engineering and modeling to yield higher, and more useful, predictive ability

Keywords

Consumer Behavior, Product Reviews, Amazon Electronics, Customer Lifetime Value, Big Data, Predictive Value

Methodology

Data Sources

The datasets are available as compressed JSON files via download at: https://nijianmo.github.io/amazon/index.html:

Amazon Electronics Reviews 5-core (4.19 GB JSON)

Contains **6,739,590 user reviews** from 1999 to 2018. These data have been reduced to extract the k-core, such that each of the remaining users and items have k reviews each. Key variables include:

- the **ID of the reviewer**
- the **ID of the product** and,
- the **time** of the review.

Amazon Electronics transaction metadata (11 GB JSON)

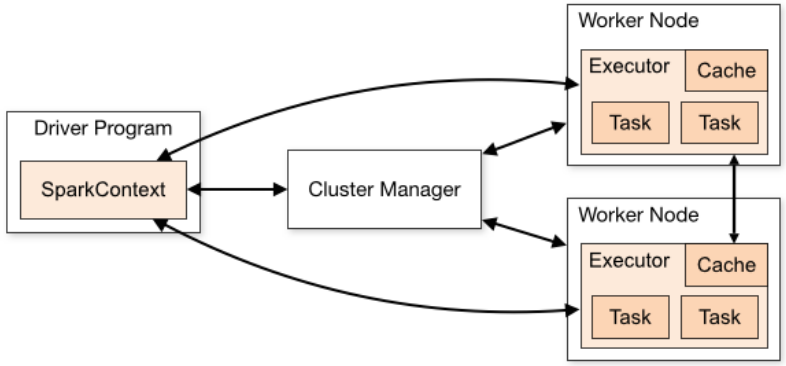
Includes descriptions, price, sales-rank, brand info, and co-purchasing links for **786,868 products**. Key variables include:

- **the ID of the product**,
- the **product price** in US dollars,
- **brand name** and,
- the **top category** to which the product belongs.

Data Manipulation Methods

Loading these large datasets into the typical Pandas workflow requires a sizable fraction of memory on our local machines and quickly becomes unwieldy. To prepare these large datasets for analysis, we leveraged distributed data processing with the **PySpark API**. Spark provides an analytics engine that represents a dataset as a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

By utilizing the SparkContext and partitioning the data across machine cores (worker nodes), we significantly accelerate our data manipulation and analysis:



Source: spark.apache.org

We begin by importing the necessary packages:

```
# Import packages
import pandas as pd
import numpy as np
import json
import gzip
import re

# PySpark specific modules
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

Then we create our SparkSession:

```
# Create Spark session
spark = SparkSession.builder.master("local[*]").appName("Milestone_I").getOrCreate()
sc = spark.sparkContext
```

The downloaded data must be decompressed[2] from the GZIP format to gain the benefit of PySpark partitioning:

```
%%sh
gzip -d meta_Electronics.json.gz
```

We then generate a spark schema for the dataset so that Spark does not have to infer the data types, which results in an increase in speed and accuracy of loading the data:

```
schema = StructType(
    [
        StructField("category", ArrayType(StringType()), True),
        StructField("tech1", StringType(), True),
        StructField("description", ArrayType(StringType()), True),
        StructField("fit", StringType(), True),
        StructField("title", StringType(), True),
        StructField("also_buy", ArrayType(StringType()), True),
        StructField("tech2", StringType(), True),
        StructField("brand", StringType(), True),
        StructField("feature", ArrayType(StringType()), True),
        StructField("rank", ArrayType(StringType()), True),
        StructField("also_view", ArrayType(StringType()), True),
        StructField("main_cat", StringType(), True),
        StructField("similar_item", StringType(), True),
        StructField("date", StringType(), True),
        StructField("price", StringType(), True),
        StructField("asin", StringType(), True),
        StructField("imageURL", ArrayType(StringType()), True),
        StructField("imageURLHighRes", ArrayType(StringType()), True),
    ]
)
```

Finally we load the electronics metadata into a variable that represents the raw data and a variable which we will work with to clean the data. These steps are performed for both datasets:

```
meta_elect_df_raw = spark.read.format("json").load("meta_Electronics.json", schema=schema)
meta_elect_df = meta_elect_df_raw
```

After loading the data, each dataset was subsequently reviewed and cleaned. Price is a key variable in our analysis. As such, the transaction dataset was filtered using regex to contain only products with values in the price attribute. The resulting dataset contained 304,323 products, with 5 attributes (product ID, brand, main category, price, title):

```
# Regex for anything that starts with a dollar sign ($)
expr = "\$.*"

# Filter with expression
meta_elect_df = meta_elect_df.filter(meta_elect_df.price.rlike(expr)).select(
    cols_to_use
)
```

The reviews dataset (6.7 million reviews) was reduced to the following attributes: product ID, review score, review time, reviewer ID, and review text. The datasets were then merged on the product ID:

```
elect_df = e5_core_df.join(meta_elect_df, on="asin", how="left")
```

After merging, 2.3 million records with missing values were removed:

```
elect_df = elect_df.na.drop()
```

This new merged dataset contains 4.5 million reviews as a result of these manipulations.

With the datasets merged, the dates and prices columns were prepared into the proper data types for analysis. Dates were represented in Unix time and were converted to dates in the YYYY-MM-DD format. The Year and Month for each date were extracted to create two new attributes useful as grouping categories during analysis:

```
elect_df = (
    elect_df.withColumn(
        "unixReviewTime", from_unixtime(col("unixReviewTime"), "MM-dd-yyyy")
    )
    .withColumn("unixReviewTime", to_date(col("unixReviewTime"), "MM-dd-yyyy"))
    .withColumn("year", year("unixReviewTime"))
    .withColumn("month", month("unixReviewTime"))
)
```

Using regex, the dollar signs ($) were removed from the price values and the datatype was converted from string to numerical:

```
elect_df = elect_df.withColumn(
    "price", regexp_replace("price", "[$,]", "").cast("double")
)
```

The first 5 rows resulting clean dataset:

```
elect_df.show(5)
```

```
+----------+-------+--------------+--------------+--------------------+--------------------+--------+-----+--------------------+----+-----+
|      asin|overall|unixReviewTime|    reviewerID|          reviewText|               brand|main_cat|price|               title|year|month|
+----------+-------+--------------+--------------+--------------------+--------------------+--------+-----+--------------------+----+-----+
|0446697192|    5.0|    2009-07-13|A3LXXYBYUHZWS5|Fresh from Connec...|Visit Amazon's Zo...|   Books|17.99|Hollywood Is like...|2009|    7|
|0446697192|    5.0|    2009-07-09|A1X4L7AO1BXMHK|I don't know abou...|Visit Amazon's Zo...|   Books|17.99|Hollywood Is like...|2009|    7|
|0446697192|    3.0|    2009-09-01|A1Y9RUTH5GG3MU|Obviously the pre...|Visit Amazon's Zo...|   Books|17.99|Hollywood Is like...|2009|    9|
|0446697192|    4.0|    2009-08-29| AAR8E3JF9K93P|I am very happy t...|Visit Amazon's Zo...|   Books|17.99|Hollywood Is like...|2009|    8|
|0446697192|    4.0|    2009-07-28|A277GP2U2TXH51|"Hollywood is Lik...|Visit Amazon's Zo...|   Books|17.99|Hollywood Is like...|2009|    7|
+----------+-------+--------------+--------------+--------------------+--------------------+--------+-----+--------------------+----+-----+
only showing top 5 rows
```

The clean data was exported to a parquet file for further analysis of the data:

```
elect_df.write.partitionBy("year").parquet("electronics_cleaned.parquet")
```

Finally samples of the raw and merged datasets were created for reader review:

```
meta_elect_df_raw.limit(100).toPandas().to_json('meta_elect_df_sample.json')
e5_core_df_raw.limit(100).toPandas().to_json('e5_core_df_sample.json')
elect_df.limit(100).toPandas().to_json('clean_merged_df_sample.json')
```

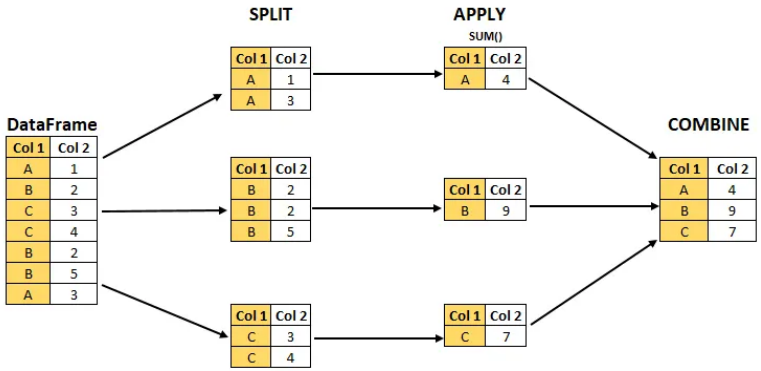
In summary, by using PySpark to handle data this large on a local machine, we were able to efficiently pre-process the data for further analysis. Key steps included:

- decompressing the data,
- merging the data,
- removing missing information, and
- transforming variables to the appropriate formats.

The full Amazon electronics dataset contains almost 21 million records, with the entire Amazon reviews dataset containing 233.1 million reviews across all categories of product. Operations on data this large would certainly require the use of cloud services or other dedicated machines optimized for big data.

Analysis

In addition to leveraging distributed processing with PySpark, we also leveraged the Split-Apply-Combine strategy to rapidly group, aggregate, and transform the data (primarily by year, product, and reviewer) into a format suitable for exploratory analysis with Pandas and visualization with Altair.



Source: Medium.com

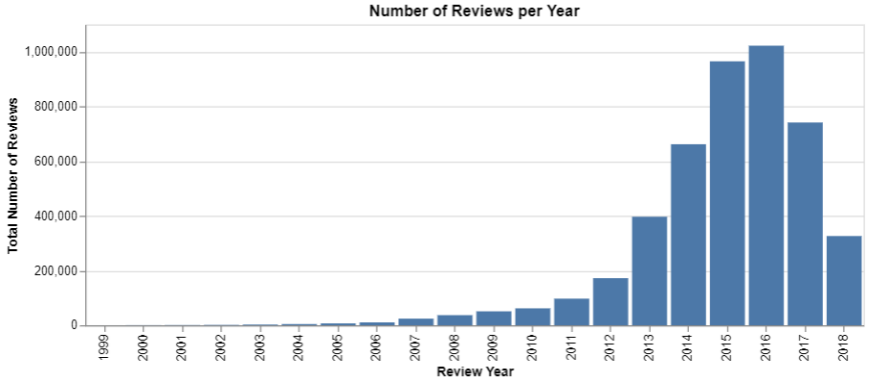
Exploratory Analysis

Descriptive statistics including the number of reviews, the number of reviewers, and the distribution of reviewer spending were generated for each year of data.

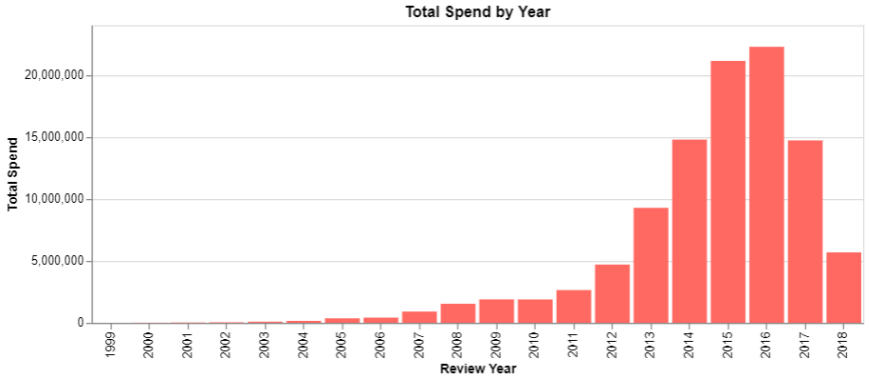
A majority of the reviews and spending fall between 2014-2017 which marks a period of rapid growth for Amazon (*finding #1 and #2*). Key events during this time include a partnership with the US Postal Service for Sunday deliveries in 2014, the acquisition of a live streaming platform Twitch in 2014 (100 million monthly users in 2015), and the acquisition of Whole Foods in 2017[6].

Median spending by reviewers is close to $11 on average, with the trend being mostly consistent each year (*finding #3*). Further analysis could provide insight as to how the price category (cheap, average, expensive) correlates with the volume of reviews.

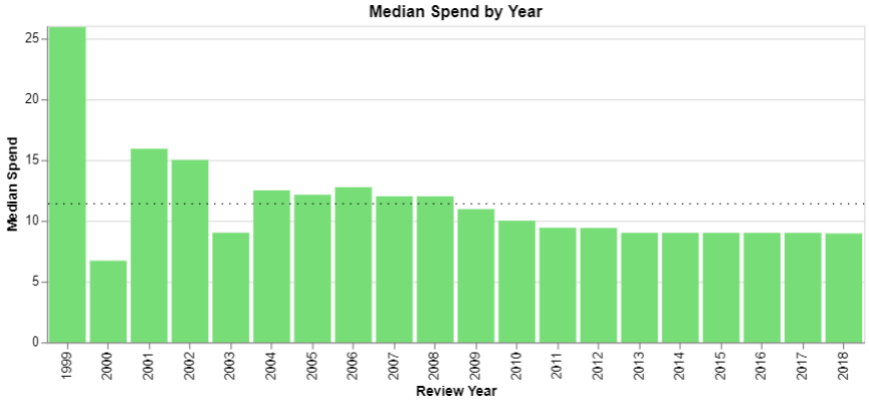
Finding #1: A majority of the reviews and reviewers are from 2014-2017.



Finding #2: Accordingly, most spending also falls in the same range of dates.



Finding #3: Median spending by reviewers is around $11 on average.



Reviewer Lifetime Value

With our dataset cleaned and properly explored we want to develop a sense of reviewer "lifetime value" and, *more* importantly, what factors tend to determine their lifetime value. That way, we can use initial attributes about a reviewer to predict what their long-term lifetime value may be; in a real world business setting, this would be useful to determine how much we want to invest in that customer/consumer.

In our case, we decided to use the total amount a user has spent over their history in this dataset (sum of the 'price' column for all products they've reviewed) as a reasonable proxy for their lifetime value. While we considered other options, such as total number of reviews per user, we felt the financial information would be useful. We want to use attributes about the first review they write to get a sense of whether or not they are likely to have a high/low lifetime value.

In order to get there, the first thing we'll do is some feature engineering to create the interesting and relevant attributes:

- LTV: Total money each customer has spent on the products they reviewed
- Star rating for customers' first review
- Length of customers' first review
- Price of the first product customer reviewed
- Main category of the first product customer reviewed

From there, we'll run a regression model and use star rating, length of first review, and price of first product reviewed to try to determine the predictive relationship with total money the customer spent over their lifetime. We'll use total money the customer spent as our dependent variable, and the rest of the attributes as our independent variables.

We'll use our model to hopefully predict the lifetime value of a customer based on their first review.

Feature Engineering:

```
import pyspark.sql.functions as F

review_data_user_spend = df.groupBy("reviewerID").agg(F.sum("price").alias("Total_Spend"))

review_data_user_spend.show()
```

```
# Create a dataframe to capture just the first review for each reviewer as well as additional features

from pyspark.sql.window import Window

scratch2 = Window.partitionBy("reviewerID").orderBy(F.col("unixReviewTime"))
review_data_first_review = df.withColumn("row",F.row_number().over(scratch2)) \
  .filter(F.col("row") == 1).drop("row")

# Add a feature that shows the length of the text of each review
review_data_first_review = review_data_first_review.withColumn("review_length", F.length("reviewText"))

# Select only the columns that matter to us; Drop rows with any na's in them
review_data_first_review = review_data_first_review.select("reviewerID", "overall", "review_length", "main_cat", "price")
review_data_first_review = review_data_first_review.na.drop()
```

```
# Combine first review dataset with the total spend dataset to develop the final dataframe we will use for our regression models
review_final_LTV = review_data_first_review.join(review_data_user_spend, ['reviewerID'], how = "left_outer")
review_final_LTV = review_final_LTV.withColumnRenamed("overall", "first_purchase_rating").withColumnRenamed("Price", "first_purchase_price")\
.withColumnRenamed("main_cat", "first_purchase_category")\
.withColumnRenamed("review_length", "first_review_length")
review_final_LTV.show()
```

First we needed to prepare our cleaned electronics data to engineer the new features and prepare it for a regression analysis. We built a table using a combination of grouping by reviewerID and aggregating the total price of the products they reviewed in the dataset to develop our LTV feature. To get values for the first review each user wrote, we partitioned the dataset by reviewerID and assigned numbers for each row (review) they left, ordered by date. From there, we only selected every reviewer's first entry. An example of the final dataset is below:

reviewerID	first_purchase_rating	first_review_length	first_purchase_category	first_purchase_price	Total_Spend
A0232309XTWAG6FEGW93	5.0	266	Camera & Photo	12.01	65.89999999999999
A0243759LWJA50LV06FT	5.0	2828	Computers	8.49	144.73
A0263994QBSZJDIHLWE	5.0	66	Cell Phones & Acc...	1.96	22.79
A02836981FYG9912C66F	5.0	2051	Computers	9.99	580.0
A02946063P41YZHKAQV6	5.0	25	Computers	4.99	35.38
A0455374D4OQJ3XCOGGT	4.0	9	Home Audio & Theater	8.39	29.36
A0530852MBVZJDVHZI4T	4.0	545	All Electronics	2.0	60.86
A0806400UK9TYOSLG1HR	1.0	75	Cell Phones & Acc...	9.74	72.4
A086178746FA6U3YDNCR	5.0	28	All Electronics	5.08	352.77
A1000I81ZDTJVE	5.0	553	Computers	19.99	57.010000000000005
A100C21JEL43QI	5.0	62	Computers	7.49	54.96
A100QXF0E860E9	5.0	217	All Electronics	14.04	125.0
A100UD67AHFODS	5.0	1851	Cell Phones & Acc...	14.95	1512.1600000000003
A1011ZSJQ2EOFO	5.0	38	Home Audio & Theater	2.0	109.75000000000001
A101AWVL2DA6EG	5.0	60	Computers	10.99	101.30000000000001
A101G9J0LB8UZ4	5.0	31	All Electronics	1.54	154.92000000000002
A101H4T6LFETV6	5.0	156	Computers	2.38	116.76
A101LNL1G2YDR5	5.0	17	Home Audio & Theater	5.99	34.089999999999996
A101OJPDXI44GP	5.0	127	Computers	17.99	35.98
A101OU5ZZAA1NG	5.0	139	Computers	39.99	130.97

Regression Modeling:

With our new features ready to go, we used Spark's machine learning library to run a linear regression on LTV (Total Spend). While we considered other tools, such as scikit-learn, using Spark proved to be the most workable solution considering the size of our dataset. We vectorized the dataset into two columns, split it into a training and testing set, and fit the model to the training set.

```
from pyspark.ml.feature import VectorAssembler

vectorAssembler = VectorAssembler(inputCols= ['first_purchase_rating','first_review_length','first_purchase_price'],
                                  outputCol= 'features')
test_vec_df = vectorAssembler.transform(review_final_LTV)
test_vec_df = test_vec_df.select(['features', 'Total_Spend'])
test_vec_df.count()
```

```
# Split the dataset into a training and test dataset
splits = test_vec_df.randomSplit([0.7, 0.3])
train_df = splits[0]
test_df = splits[1]
```

```
# Train the model on our training set

from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol = 'features', labelCol='Total_Spend', maxIter=10, regParam=0.3, elasticNetParam=0.8)
lr_model = lr.fit(train_df)
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
```

This yielded the following coefficients for each independent variable:

- Star Rating: 0.9499642372899711
- Review Length: 0.03995774898882538
- Price: 1.0274694155656496
- Intercept: 98.67039618960274

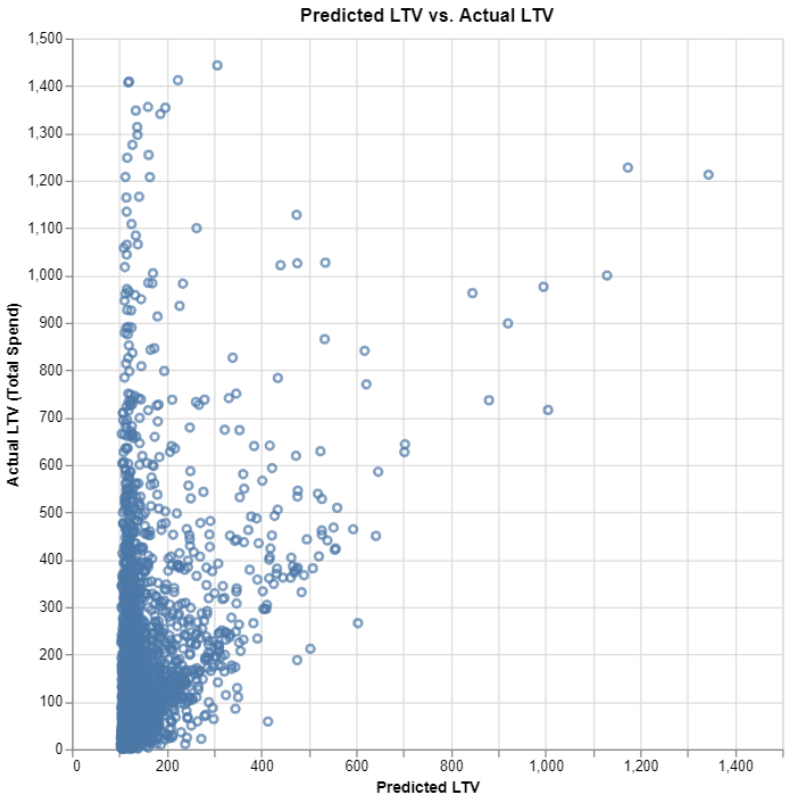
The model explained variability in the data with mixed results:

- RMSE: 274.590376
- r2: 0.384534

From there, we used the fitted model to predict values for the holdout test set. An example table of predicted values vs. actual reviewer LTV shows how the predicted LTV our model generated on the test set compares to the actual values of the test set.

prediction	Total_Spend
150.278697	64.34
139.221925	40.30
132.683717	187.65
113.472368	18.89
187.223553	140.53
160.017879	150.43
114.677903	51.79

The chart of a sample (5000 data points) of these rows is below. The chart shows the predicted values on the X axis and the actual values on the Y axis. Perfect prediction would result in a straight, diagonal line.



Discussion:

Ultimately the features we created were able to add some predictive value regarding reviewers' LTV. The R2 value of .38 showed this. However, we don't feel it was a smoking gun in terms of predicting future LTV in order to make business decisions. Additionally, price of first product reviewed was the biggest determinant of future LTV, and that could potentially be because some reviewers only reviewed one product, and, in those cases, price of first product reviewed is equal to their LTV.

While it was a successful first attempt, there are clear areas we've identified that can be improved upon immediately:

- Incorporating categorical variables (product category - which we didn't get to use in this first model, using Star Rating as a category) into our model.
- Diving deeper into the subset of users for which the model *does* seem to predict more accurately (see the diagonal trend on the scatter chart).
- Creating other variables that may have more explanatory power, such as # of reviews left in the first month, or using total number of reviews as a proxy for LTV instead of total spend.

Statement of Work

Both authors contributed equally to problem formulation, literature review, and data analysis performed in Python/Pandas. Data manipulation was performed by Camaron Mangham via Apache Spark (PySpark) ; visualizations were created by Vinu Lakkur via Vega-Altair. The project files were hosted on Github which served as the primary repository for collaboration. Analysis was performed on MacOS and Windows operating systems resulting in minor dependency issues that should be addressed in the early stages of future collaborations. This report was created via Google Docs.

References

1. Jianmo Ni, Jiacheng Li, Julian McAuley. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. *Empirical Methods in Natural Language Processing (EMNLP)*, 2019
2. Khandelwal, Subham. PySpark — Read Compressed gzip files
3. Mengting Wan, Jianmo Ni, Rishabh Misra, Julian McAuley. Addressing Marketing Bias in Product Recommendations. *WSDM*, 2020
4. Sun, Y., Liu, H., & Gao, Y. (2023). Research on customer lifetime value based on machine learning algorithms and customer relationship management analysis model. *Heliyon*, *9*(2), e13384. https://doi.org/10.1016/j.heliyon.2023.e13384
5. Qualtrics. How to Calculate Customer Lifetime Value.
6. Wikipedia. Amazon (company).