CIS 4130

Sunehra Simin Farhana

SUNEHRA.FARHANA@baruchmail.cuny.edu

PROPOSAL

The data set that I will be using for this project is the "Amazon US Customer Reviews Dataset". It contains the reviews that customers wrote about Amazon products on Amazon.com, from 1995 to 2015. It contains 37 tsv files full of information about how satisfied customers were over their books, furniture, electronics, shoes, and other product categories that they purchased from Amazon. It even includes what they rated the product, on a scale from 1 star to 5 stars, and how many other customers voted that their review was "helpful".

Here is the URL/Location for downloading the data:
https://www.kaggle.com/datasets/cynthiarempel/amazon-us-customer-reviews-dataset

There are fifteen data set attributes. They are marketplace, customer_id, review_id, product_id, product_parent, product_title, product_category, star_rating, helpful_votes, total_votes, vine, verified_purchase, review_headline, review_body, and review_date. It records information about the content of the review, and the product that it is reviewing.

This data set gives me twenty years' worth of reviews of common Amazon products. Using this data set, I intend to model which product categories have the highest and lowest star-ratings. Comparing the "helpful votes" to the "total votes", and checking whether "verified_purchase" is true, will allow me to gauge how useful and reliable those reviews are. In addition, I can model the most favorably-reviewed and least favorably-reviewed products within each product category. I will use this data set to predict which product categories will generate the most customer satisfaction, so that I will know which products need to be improved upon.

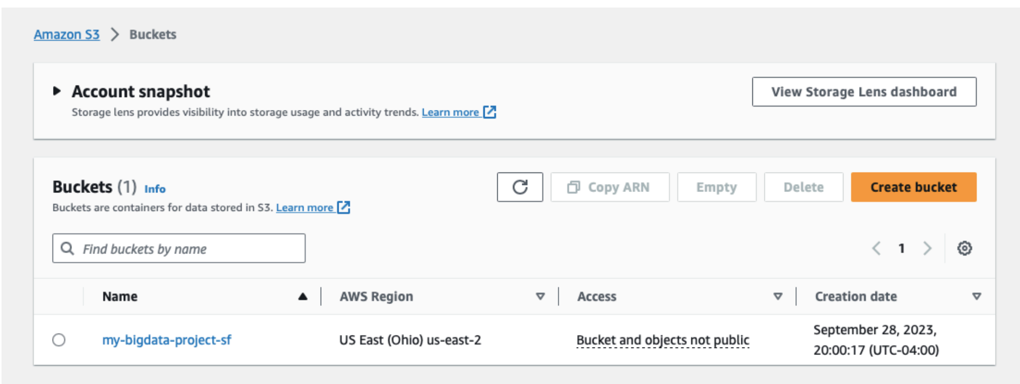
This data set can also be used to predict whether customers will have a generally negative or positive opinion of a product, based on the customers' reactions to similar products under that category. I can also use the data to predict the star_rating of a product, based on the review_headline, review_body, product_category, product_parent, and its other attributes.

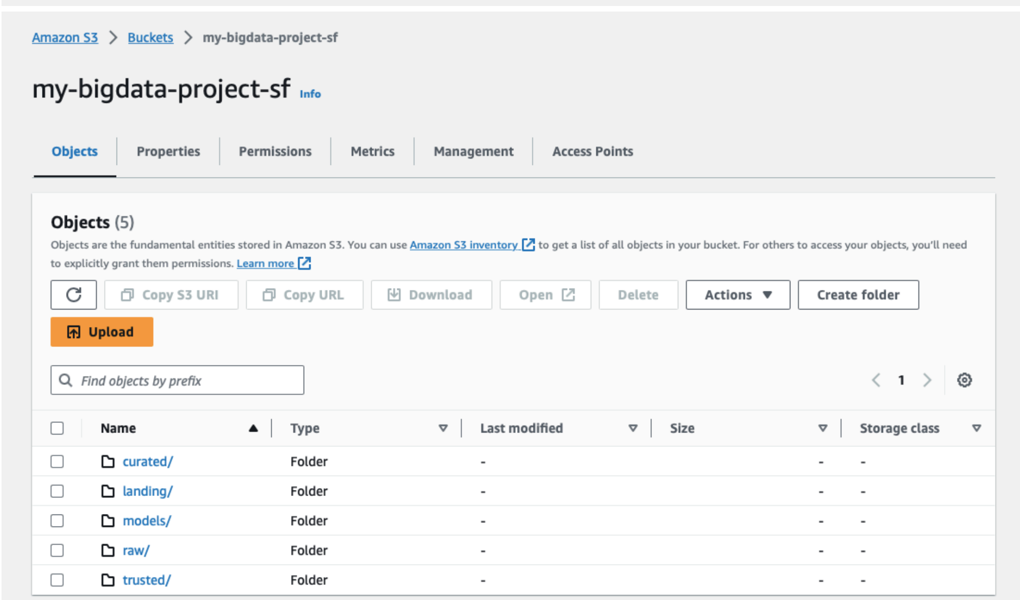
DATA ACQUISITION

In order to extract and collect the "Amazon US Customer Reviews Dataset" from Kaggle, I first launched an EC2 instance with 30gb of storage, and then connected to that instance. To configure the AWS CLI, I ran the command "aws configure", pasted in my Access Key ID, pasted in my Secret Access Key, typed "us-east-2" as the region name, and then typed "json" as the output format. I then ran "sudo yum -y install python3-pip" to install pip in the EC2 instance.

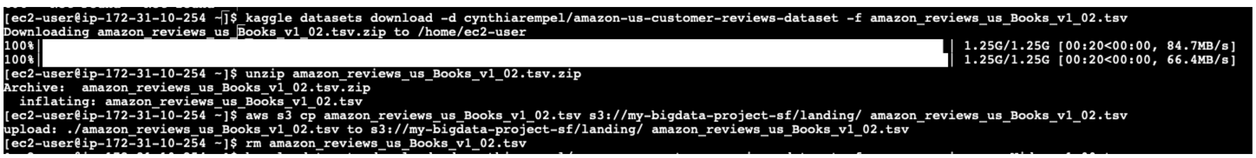
The next step was to log into my account on Kaggle.com, create an API token, and download the resulting json file. I installed the Kaggle CLI by running "pip3 install kaggle", and then made a directory by running "mkdir .kaggle". I entered "nano .kaggle/kaggle.json" to create a new file, and then pasted my username and key from the json file into it. I then saved the file, exited nano, and secured the file by typing "chmod 600 .kaggle/kaggle.json".

After that, I created an S3 bucket titled "my-bigdata-project-sf", and created a "landing" folder within that bucket in order to store the tsv files from Kaggle.

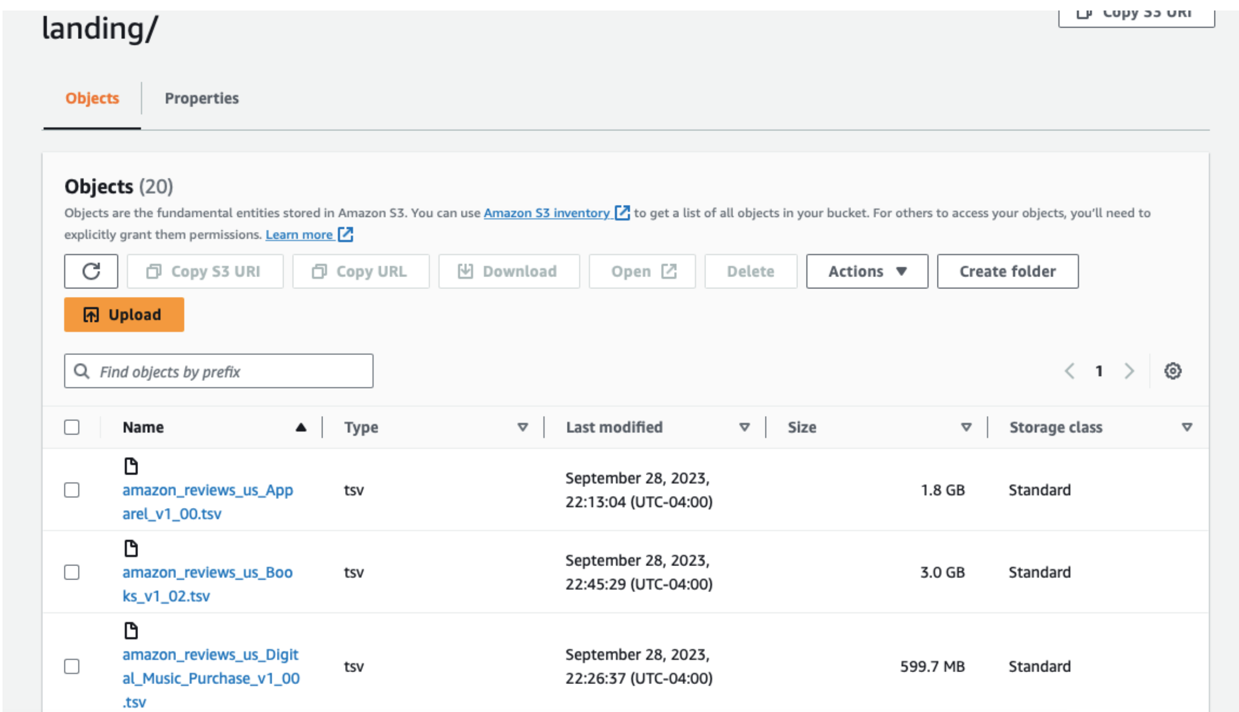


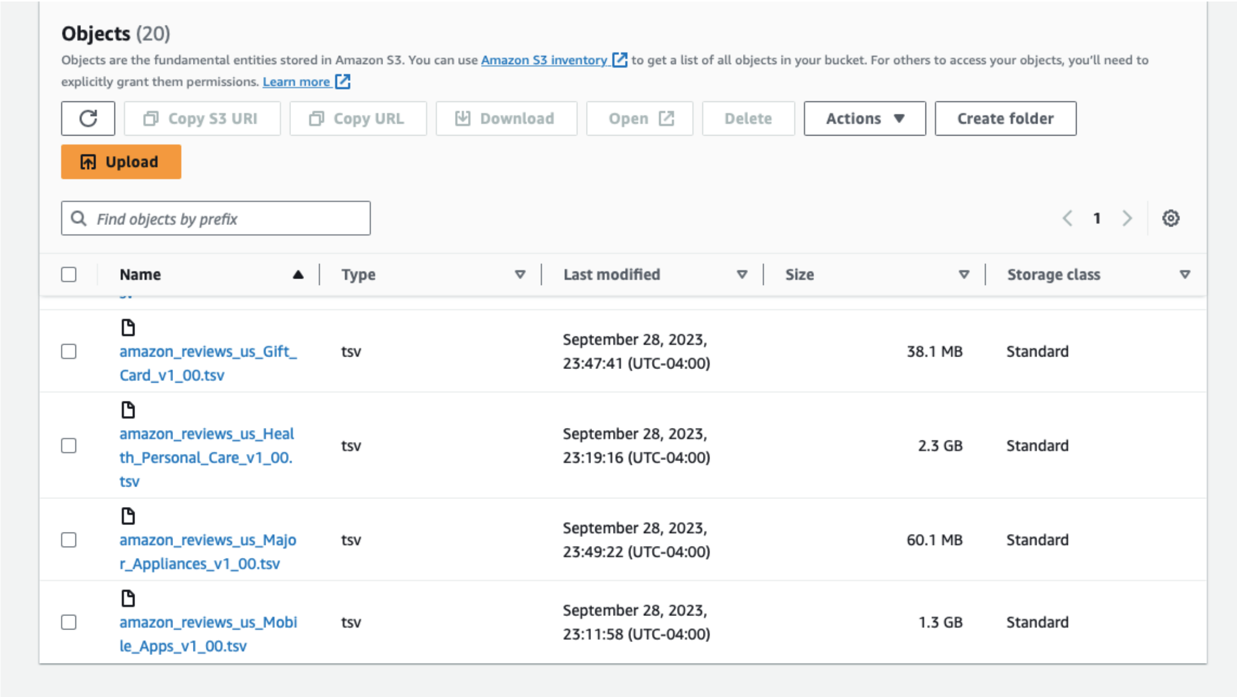


Now that my EC2 instance and S3 bucket was set up, I could finally begin extracting and collecting from the “Amazon US Customer Reviews Dataset” on Kaggle. I first ran “kaggle datasets files cynthiarempel/amazon-us-customer-reviews-dataset” to get a list of the 37 files in the dataset. The EC2 instance did not have enough disk space for me to download and unzip the entire dataset at once. Therefore, I had to individually download each file by running “kaggle datasets download -d cynthiarempel/amazon-us-customer-reviews-dataset -f amazon_reviews_us_Apparel_v1_00.tsv”, unzip it by running “unzip amazon_reviews_us_Apparel_v1_00.tsv.zip”, put it in the “landing” folder of the “my-bigdata-project-sf” S3 bucket by running “aws s3 cp amazon_reviews_us_Apparel_v1_00.tsv s3://my-bigdata-project-sf/landing/ amazon_reviews_us_Apparel_v1_00.tsv”, and finally remove the file by running “rm amazon_reviews_us_Apparel_v1_00.tsv”.



I repeated these steps 37 times, for each of the 37 tsv files in the Kaggle dataset. Unfortunately, there was a bug that didn’t allow me successfully download and move all 37 files into the S3 bucket. 17 of them gave me a “404 – Not Found” error, so I had to move on.





I was able to successfully transport the 20 following tsv files into the S3 bucket:

amazon_reviews_multilingual_US_v1_00.tsv
amazon_reviews_us_Apparel_v1_00.tsv
amazon_reviews_us_Automotive_v1_00.tsv
amazon_reviews_us_Baby_v1_00.tsv
amazon_reviews_us_Beauty_v1_00.tsv
amazon_reviews_us_Books_v1_02.tsv
amazon_reviews_us_Camera_v1_00.tsv
amazon_reviews_us_Digital_Ebook_Purchase_v1_01.tsv
amazon_reviews_us_Digital_Music_Purchase_v1_00.tsv
amazon_reviews_us_Digital_Video_Download_v1_00.tsv
amazon_reviews_us_Digital_Video_Games_v1_00.tsv
amazon_reviews_us_Digital_Software_v1_00.tsv
amazon_reviews_us_Electronics_v1_00.tsv
amazon_reviews_us_Furniture_v1_00.tsv
amazon_reviews_us_Gift_Card_v1_00.tsv
amazon_reviews_us_Grocery_v1_00.tsv
amazon_reviews_us_Health_Personal_Care_v1_00.tsv
amazon_reviews_us_Major_Appliances_v1_00.tsv
amazon_reviews_us_Mobile_Apps_v1_00.tsv
amazon_reviews_us_Mobile_Electronics_v1_00.tsv

EXPLORATORY DATA ANALYSIS

The Python code that I used to load the data set from my Amazon S3 bucket's "landing" folder, and to produce descriptive statistics about the data, is placed in the Appendix of this document. I used the amazon_reviews_us_Gift_Card_v1_00.tsv (reviews_df) and amazon_reviews_us_Digital_Video_Games_v1_00.tsv (reviews_dff) files for these examples. I used different variables so that it would be easier to switch back and forth between them.

I was able to successfully perform exploratory data analysis on the tsv files. The files had null values in the review_headline, review_body, and review_date columns, which makes sense since not everyone who rates a product would write out a review. The earliest review in the files was from the early 2000s, while the most recent review was from 2015. I also noticed how star_ratings of 5 were the most common rating for products. One of the challenges that I will have in cleaning and feature engineering will be handling all of the null values scattered throughout every tsv file. I will have to drop or replace those values while cleaning the data. It will also be a challenge to analyze the text in review_body. I observed during the analysis how the number of words in reviews ranged from 1 to over 400. Some people write short and concise reviews, while others write a paragraph detailing their experience with the product. Reviews are usually unique, since they contain different sets of words in different orders, so I'm not sure how I will incorporate this attribute into the predictive models.

FEATURE ENGINEERING AND MODELING

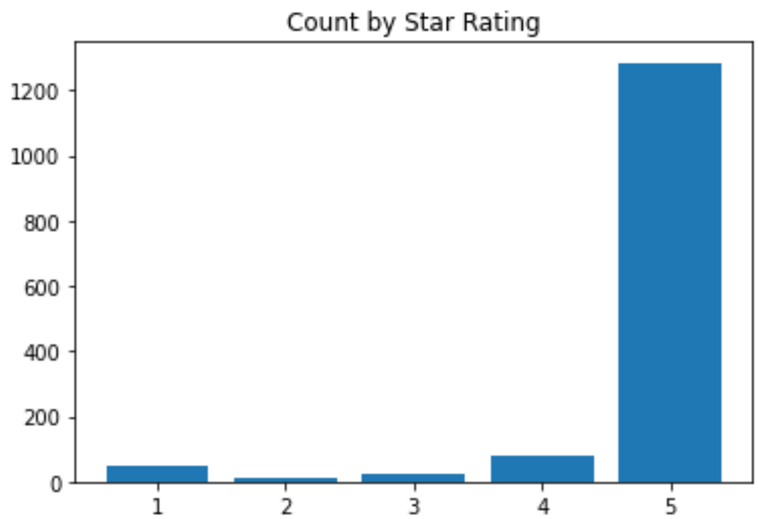
The main steps that my program takes in order to clean the data is first taking each tsv file from the "landing" folder of my S3 bucket, dropping any null records from the columns, then removing non-ascii characters from any review_headline and review_body records, and finally saving the cleaned data frame into the "raw" folder of my S3 bucket as a parquet file. This was challenging for larger files, since they need more time to process. However, the wait was worth it, because feature engineering is much easier without the null values, and without emojis or other symbols cluttering up the review text.

For feature engineering, I dropped the columns that I feel contain irrelevant data for my predictions. I will predict the star_rating based on the remaining attributes. I've outlined what I will do with each variable when feature engineering. I will encode the integer datatypes as a vector, index the strings and then encode them as a vector, and assemble the features into one vector. I will also tokenize the words within the clean_review_body and clean_review_headline variables, record their various frequencies, and run a sentiment analysis. I came across a challenge, in which I couldn't include clean_review_body and clean_review_headline in the encoder since some records were just blank spaces, even though I dropped all the null values. I would like to find a way to resolve this in the future.

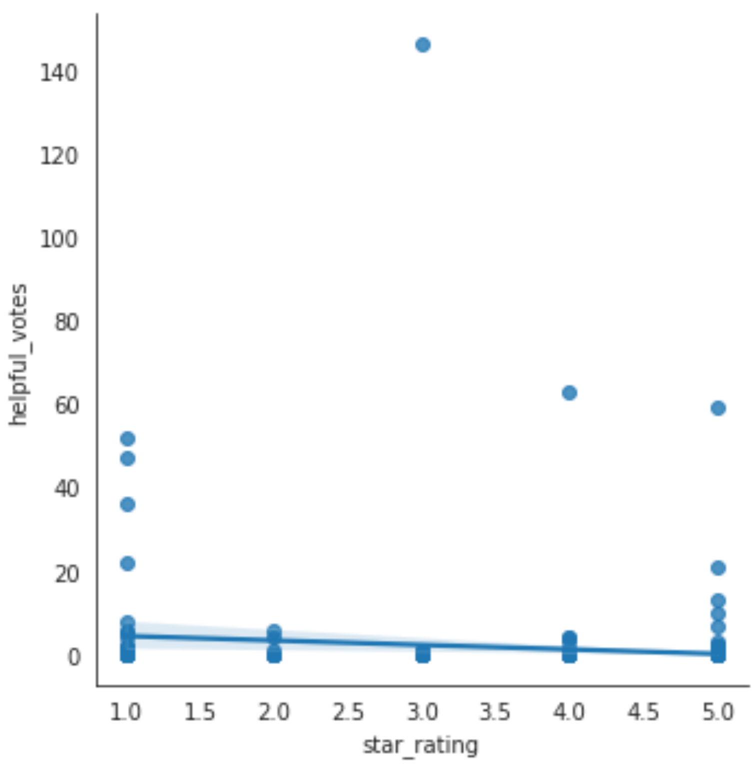
marketplace	string	drop
customer_id	integer	drop
review_id	string	drop
product_id	string	drop
product_parent	integer	drop
product_title	string	index, then encode as a vector
product_category	string	index, then encode as a vector
star_rating	integer	encode as a vector
helpful_votes	integer	encode as a vector
total_votes	integer	encode as a vector
vine	string	index, then encode as a vector
verified_purchase	string	index, then encode as a vector
clean_review_headline	string	tokenize, term-frequency, IDF, sentiment analysis
clean_review_body	string	tokenize, term-frequency, IDF, sentiment analysis
review_date	date	drop
body_sentiment_score	double	encode as a vector
headline_sentiment_score	double	encode as a vector

I’ve tested out my feature engineering and modeling process on three of my cleaned data files: cleaned_amazon_reviews_us_Gift_Card_v1_00.parquet, cleaned_amazon_reviews_us_Digital_Video_Games_v1_00.parquet, and cleaned_amazon_reviews_us_ Major_Appliances_v1_00.parquet. My model was ultimately successful. After looking over the test results, I could see that it was able to accurately predict the star_rating of a product based on the features. The accuracy, precision, and recall scores were between 87% to 96%, and the CrossValidator was about 0.7.

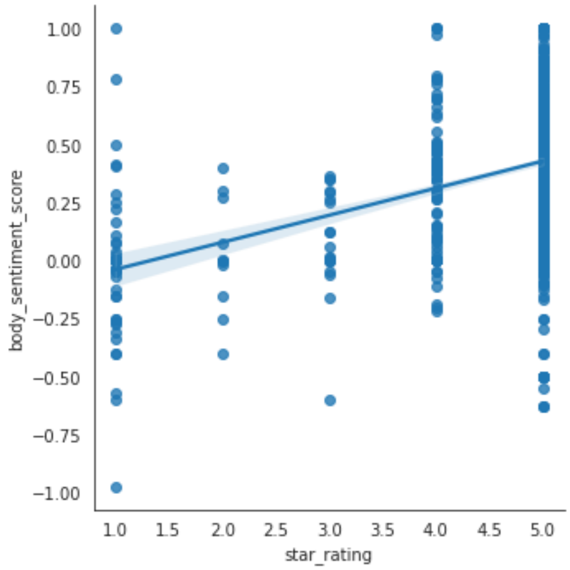
DATA VISUALIZING

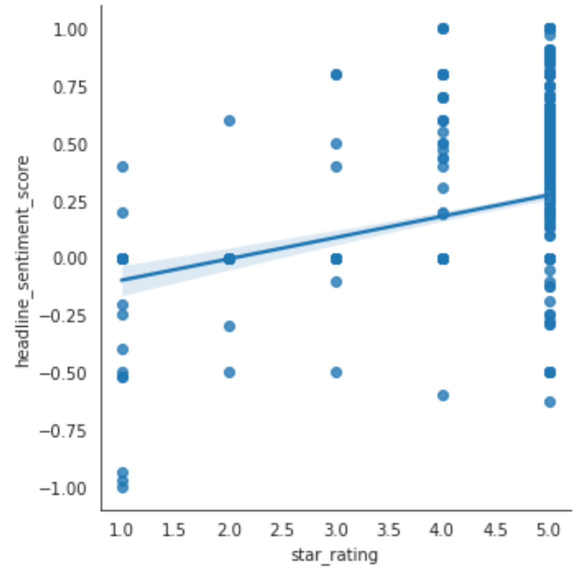


The first visualization of my data that I created is a bar plot that shows the frequency of star_rating, using Matplotlib. This is useful because it shows what is the most common rating among each data file. Having 5 as the most frequent rating would mean that this type of product is of good quality, and customers are generally satisfied with what they received from Amazon.

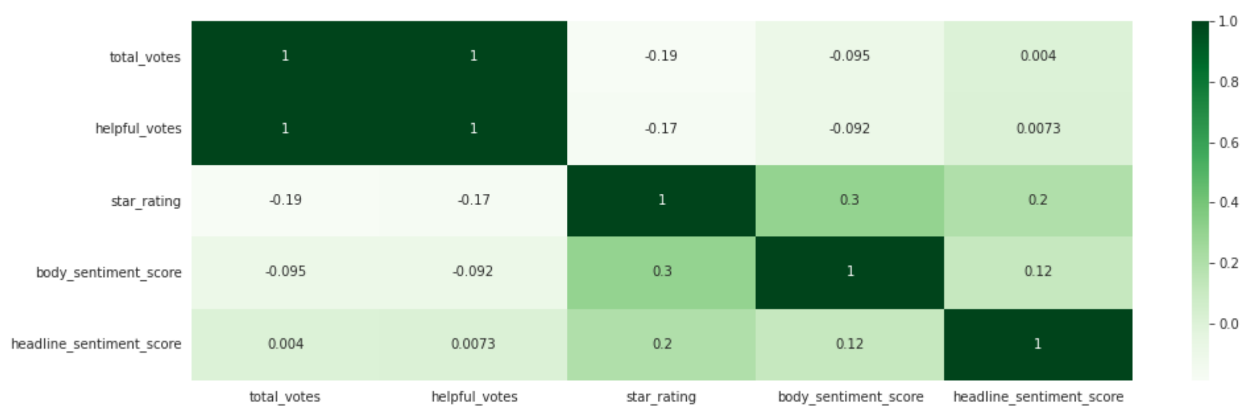


The second visualization is a relationship plot that shows the relationship between star_rating and helpful_votes, using Seaborn. This is useful because it shows whether there is a trend between the star rating of the review, and the number of people who voted that the review was helpful. This lends to the credibility of the reviewer's star rating.

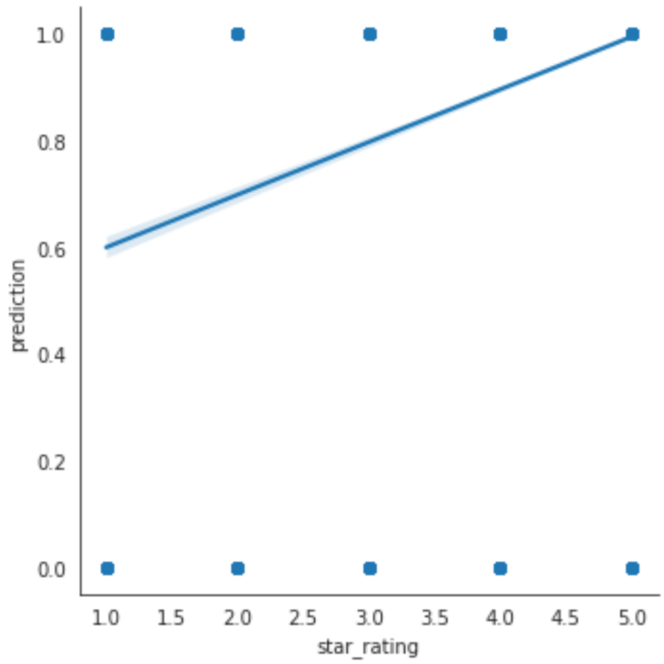




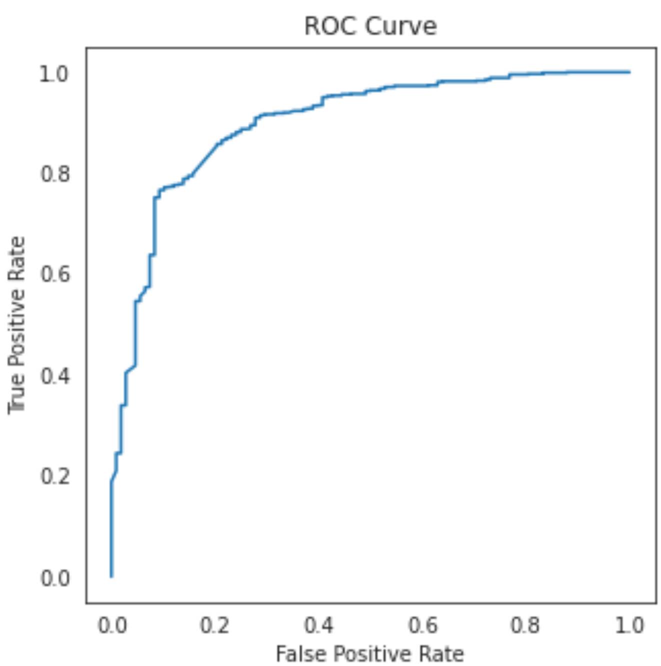
The third and fourth visualizations are relationship plots of the star_rating and body_sentiment_score, and then the star_rating and headline_sentiment_score, using Seaborn. The sentiment score assesses the words in the review and assigns it a score based on how negative or positive the review seems. Logically, there should be a higher star rating where there is a positive review score.



The fifth visualization is a correlation matrix of the numeric columns in the data, which are total_votes, helpful_votes, star_rating, body_sentiment_score, and headline_sentiment_score. This heatmap shows the strength of the relationship between every single combination of numeric variables. I can see that the star_rating has a weaker relationship with the total_votes and helpful_votes since its correlation is negative. I can also see that the star_rating has a stronger relationship with the body_sentiment_score and headline_sentiment_score, since its relationship is closer to 1. This means that the body_sentiment_score and headline_sentiment_score would be more useful for predicting the star_rating.



The sixth visualization is a relationship plot of the star_rating and prediction, which is the star_ratings that were predicted using the other variables, such as total_votes and body_sentiment_score. It is important to see how the predicted results aligned with the actual results, because that assesses the accuracy of the prediction.



The seventh visualization is an ROC curve of the bestModel data, using Matplotlib. This visualization plots the different true-positive and false-positive measures in order to assess the performance of the bestModel. This ROC curve indicates that my model was generally accurate when predicting the star_rating.

SUMMARY AND CONCLUSIONS

I have learned a lot of skills throughout this big data project. I discovered how useful the Amazon Web Services, EC2 and S3, are for processing and analyzing large amounts of data. I also learned how to use EC2 to download files from a website and upload them directly into my S3 bucket. I thought it was impressive how I didn't need to download a single Amazon Reviews file into my computer for the data acquisition.

One of the main challenges of this project was dealing with the small storage space in the EC2 instance. 30gb wasn't enough space to handle 22gb of Amazon Reviews data, so I had to combat this by loading a file into the instance one at a time, and then removing the file from the instance once I was done loading it to my S3 bucket. I also transitioned to writing the Python code in Databricks when I started working on feature engineering, modeling, and data visualization. This program had much more space, so I could work on the data files without much issue.

I realized that a lot of the Amazon Reviews data files had null values when doing exploratory data analysis, so I dropped the rows containing null values while cleaning the data, since they would not be useful for modeling. I also dropped the columns that contained the information that I would not be using to predict the star_rating, such as product_parent or review_id. Then, I removed the non-ascii characters from review_headline and review_body so that it would be easier to use that text for modeling. I also engineered a sentiment score column for those two variables, to record how positive each review is according to the words within it.

The purpose of my project was to use the product_title, product_category, helpful_votes, total_votes, vine, verified_purchase, body_sentiment_score, and headline_sentiment_score varibles to predict the star_rating. I encoded the integer datatypes as vectors, indexed and then encoded the string datatypes as vectors, and finally assembled the features into one vector. Analyzing the data was a long and tedious process. The large data files needed several minutes to process through the Python code, especially during the testing and training portion. However, I persisted through it and managed to train and test my model. I was satisfied with my model's ability to accurately predict the star_rating of a product based on the aforementioned features. The accuracy, precision, and recall scores were between 87% to 96%, and the CrossValidator was about 0.7.

I can conclude that it is possible to predict the star_rating of Amazon Reviews given the other attributes of the review. The correlation matrix that I created during the data visualization phase proved that the body_sentiment_score and headline_sentiment_score were most useful for predicting the star_rating. This makes sense, considering that the text of a review would be a big indicator of whether or not the reviewer was satisfied by the product. The ROC curve that I created of my model was closer to the upper left corner of the graph, indicating that the star_rating was correctly identified most often. The data visualizations were useful for further representing the data and the accuracy of my model.

This is the GitHub URL of my project:
https://github.com/SunehraFarhana/amazon-reviews-bigdata-sf.git

APPENDIX A
DATA ACQUISITION CODE

```
# Configure AWS CLI.
aws configure
_____           # AWS Access Key ID
_____           # Secret Access Key
us-east-2       # Default region name
json            # Default output format

# Install pip.
sudo yum -y install python3-pip
# Instal Kaggle CLI.
pip3 install Kaggle

# Made directory for Kaggle.
mkdir .kaggle
# Create file.
nano .kaggle/kaggle.json
# [paste username and key]
# Secure file.
chmod 600 .kaggle/kaggle.json

# Download each file from Kaggle.
kaggle datasets download -d cynthiarempel/amazon-us-customer-reviews-dataset -f
amazon_reviews_us_Apparel_v1_00.tsv
# Unzip file.
unzip amazon_reviews_us_Apparel_v1_00.tsv.zip
# Put file in landing folder of S3 bucket.
aws s3 cp amazon_reviews_us_Apparel_v1_00.tsv s3://my-bigdata-project-sf/landing/
amazon_reviews_us_Apparel_v1_00.tsv

# Remove file to save space in EC2 instance.
rm amazon_reviews_us_Apparel_v1_00.tsv
```

APPENDIX B
EXPLORATORY DATA ANALYSIS CODE

Import the necessary functions.
python3
import boto3
import pandas as pd

# Read in Amazon Review tsv file into a data frame.
reviews_df = pd.read_csv("s3://my-bigdata-project-sf/landing/amazon_reviews_us_Gift_Card_v1_00.tsv", sep='\t', on_bad_lines='skip')

Get the data type of each column in the file.
reviews_df.dtypes

```
>>> reviews_df = pd.read_csv("s3://my-bigdata-project-sf/landing/amazon_reviews_us_Gift_Card_v1_00.tsv", sep='\t', on_bad_lines='skip')
>>> reviews_df.dtypes
marketplace          object
customer_id           int64
review_id            object
product_id           object
product_parent        int64
product_title        object
product_category     object
star_rating           int64
helpful_votes         int64
total_votes           int64
vine                 object
verified_purchase    object
review_headline      object
review_body          object
review_date          object
dtype: object
```

```
>>> reviews_dff.dtypes
marketplace                  object
customer_id                   int64
review_id                    object
product_id                   object
product_parent                int64
product_title                object
product_category             object
star_rating                   int64
helpful_votes                 int64
total_votes                   int64
vine                         object
verified_purchase            object
review_headline              object
review_body                  object
review_date          datetime64[ns]
dtype: object
>>>
```

# Look at the first five rows of data.
print(reviews_df.head(5))

```
>>> print(reviews_df.head(5))
  marketplace  customer_id       review_id  ...                                review_headline                                    review_body review_d
ate
0          US     24371595  R27ZP1F1CD0C3Y  ...                                     Five Stars             Great birthday gift for a young adult.  2015-08
-31
1          US     42489718   RJ7RSBCHUDNNE  ...  Gift card for the greatest selection of items ...  It's an Amazon gift card and with over 9823983...  2015-08
-31
2          US       861463  R1HVYBSKLQJI5S  ...                                     Five Stars                                               Good  2015-08
-31
3          US     25283295  R2HAXF0IIYQBIR  ...                                       One Star                                               Fair  2015-08
-31
4          US       397970   RNYLPX611NB7Q  ...                                     Five Stars  I can't believe how quickly Amazon can get the...  2015-08
-31

[5 rows x 15 columns]
```

```
>>> print(reviews_dff.head(5))
  marketplace  customer_id       review_id  ...                  review_headline                                        review_body review_date
0          US     21269168   RSH1OZ87OYK92  ...  A slight improvement from last year.  I keep buying madden every year hoping they ge...  2015-08-31
1          US       133437  R1WFOQ3N9BO65I  ...                          Five Stars                                            Awesome  2015-08-31
2          US     45765011   R3YOOS71KM5M9  ...             Hail to the great Yuri!  If you are prepping for the end of the world t...  2015-08-31
3          US       113118  R3R14UATT3OUFU  ...                          Five Stars                                            Perfect  2015-08-31
4          US     22151364   RV2W9SGDNQA2C  ...                          Five Stars                                           Awesome!  2015-08-31

[5 rows x 15 columns]
>>>
```

# Look at the basic information about the data frame, such as non-null values in each column.
print(reviews_df.info())

```
>>> print(reviews_df.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148310 entries, 0 to 148309
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   marketplace        148310 non-null  object
 1   customer_id        148310 non-null  int64
 2   review_id          148310 non-null  object
 3   product_id         148310 non-null  object
 4   product_parent     148310 non-null  int64
 5   product_title      148310 non-null  object
 6   product_category   148310 non-null  object
 7   star_rating        148310 non-null  int64
 8   helpful_votes      148310 non-null  int64
 9   total_votes        148310 non-null  int64
 10  vine               148310 non-null  object
 11  verified_purchase  148310 non-null  object
 12  review_headline    148304 non-null  object
 13  review_body        148303 non-null  object
 14  review_date        148309 non-null  object
dtypes: int64(5), object(10)
memory usage: 17.0+ MB
None
>>>
```

```
>>> print(reviews_dff.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144724 entries, 0 to 144723
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   marketplace        144724 non-null  object
 1   customer_id        144724 non-null  int64
 2   review_id          144724 non-null  object
 3   product_id         144724 non-null  object
 4   product_parent     144724 non-null  int64
 5   product_title      144724 non-null  object
 6   product_category   144724 non-null  object
 7   star_rating        144724 non-null  int64
 8   helpful_votes      144724 non-null  int64
 9   total_votes        144724 non-null  int64
 10  vine               144724 non-null  object
 11  verified_purchase  144724 non-null  object
 12  review_headline    144721 non-null  object
 13  review_body        144721 non-null  object
 14  review_date        144721 non-null  datetime64[ns]
dtypes: datetime64[ns](1), int64(5), object(9)
memory usage: 16.6+ MB
None
>>>
```

# Get the count, mean, standard deviation, minimum, and maximum for the numerical data.
print(reviews_df.describe())

```
>>> print(reviews_df.describe())
        customer_id  product_parent    star_rating  helpful_votes    total_votes
count  1.483100e+05    1.483100e+05  148310.000000  148310.000000  148310.000000
mean   2.628931e+07    5.406163e+08       4.731333       0.397424       0.490493
std    1.587236e+07    2.661563e+08       0.829255      20.701385      22.823494
min    1.063700e+04    1.100879e+06       1.000000       0.000000       0.000000
25%    1.289732e+07    3.612555e+08       5.000000       0.000000       0.000000
50%    2.499530e+07    4.730483e+08       5.000000       0.000000       0.000000
75%    4.139731e+07    7.754865e+08       5.000000       0.000000       0.000000
max    5.309648e+07    9.992742e+08       5.000000    5987.000000    6323.000000
```

```
>>> print(reviews_dff.describe())
        customer_id  product_parent    star_rating  helpful_votes    total_votes                    review_date
count  1.447240e+05    1.447240e+05  144724.000000  144724.000000  144724.000000                         144721
mean   2.509395e+07    4.585574e+08       3.852443       1.487915       2.703636  2013-11-10 12:03:39.401468928
min    1.063200e+04    2.435250e+05       1.000000       0.000000       0.000000            2006-08-08 00:00:00
25%    1.196164e+07    2.328037e+08       3.000000       0.000000       0.000000            2013-03-02 00:00:00
50%    2.291884e+07    3.889339e+08       5.000000       0.000000       0.000000            2014-01-03 00:00:00
75%    3.985036e+07    6.866072e+08       5.000000       1.000000       2.000000            2014-11-13 00:00:00
max    5.309600e+07    9.996412e+08       5.000000    5068.000000    5251.000000            2015-08-31 00:00:00
std    1.572963e+07    2.720975e+08       1.539966      21.654493      24.179442                            NaN
>>>
```

# Find out which columns in the data frame have null values.
print(reviews_df.columns[reviews_df.isnull().any()].tolist())
# Find out how many records in the data frame have null values.
print("Rows with null values:", reviews_df.isnull().any(axis=1).sum())

```
>>> print(reviews_df.columns[reviews_df.isnull().any()].tolist())
['review_headline', 'review_body', 'review_date']
>>> print("Rows with null values:", reviews_df.isnull().any(axis=1).sum())
Rows with null values: 13
```

```
>>> print(reviews_dff.columns[reviews_dff.isnull().any()].tolist())
['review_headline', 'review_body', 'review_date']
>>> print("Rows with null values:", reviews_dff.isnull().any(axis=1).sum())
Rows with null values: 9
>>>
```

# See how each star_rating compares to the total_votes.
results = reviews_df.groupby('star_rating').total_votes.agg(['count', 'min', 'max', 'mean'])
print(results)

```
>>> results = reviews_df.groupby('star_rating').total_votes.agg(['count', 'min', 'max', 'mean'])
>>> print(results)
              count  min   max      mean
star_rating
1              4766    0  6323  7.917331
2              1560    0  2557  5.360897
3              3147    0   436  1.373371
4              9808    0  2763  0.503467
5            129029    0  2253  0.134760
```

```
>>> results = reviews_dff.groupby('star_rating').total_votes.agg(['count', 'min', 'max', 'mean'])
>>> print(results)
             count  min   max      mean
star_rating
1            24848    0  5251  7.170597
2             7727    0   420  4.004788
3            11589    0   220  2.490810
4            20328    0   387  1.822166
5            80232    0  2974  1.448973
>>>
```

# Get the number of words for each record of review text in the review_body column.
num_words = reviews_df["review_body"].str.split().str.len()
print(num_words)

```
>>> num_words = reviews_df["review_body"].str.split().str.len()
>>> print(num_words)
0           7.0
1          19.0
2           1.0
3           1.0
4          14.0
          ...
148305     48.0
148306     16.0
148307     51.0
148308     49.0
148309    113.0
Name: review_body, Length: 148310, dtype: float64
```

```
>>> num_words = reviews_dff["review_body"].str.split().str.len()
>>> print(num_words)
0         173.0
1           1.0
2          29.0
3           1.0
4           1.0
          ...
144719     59.0
144720     79.0
144721     83.0
144722    460.0
144723     46.0
Name: review_body, Length: 144724, dtype: float64
>>>
```

# Look at the most popular products in the file.
print(reviews_df['product_title'].value_counts())

```
>>> reviews_df['product_title'].value_counts()
product_title
Amazon.com eGift Cards                                                   36870
Amazon.com Gift Card in a Greeting Card (Various Designs)                 9344
Amazon.com Gift Cards - Print at Home                                     8399
Amazon eGift Card - Happy Birthday (Candles)                              6037
Amazon eGift Card - Smile                                                 5034
                                                                           ...
Amazon Video Gift Card - E-mail - Hard Rockin' Chanukah                      1
Amazon Gift Card - Email - Chanukah (Gelt Talks) [Someecards]                1
Amazon Gift Card - Facebook - Christmas (Buy for Yourself) [Someecards]      1
Amazon eGift Card - A Birthday So Memorable                                  1
Amazon Gift Card - Facebook - Amazon Watches                                 1
Name: count, Length: 1193, dtype: int64
>>>
```

```
>>> print(reviews_dff['product_title'].value_counts())
product_title
Playstation Network Card                                  13517
Xbox Live Subscription                                     7263
Playstation Plus Subscription                              4677
SimCity - Limited Edition                                  3419
Xbox Live Gift Card                                        3406
                                                            ...
XIII: Lost Identity [Download]                                1
Time Breaker [Download]                                       1
Suspension Railroad Simulator 2013 MAC [Download]             1
The Timebuilders: Pyramid Rising (Mac) [Download]             1
Enchanted Gardens [Download]                                  1
Name: count, Length: 6946, dtype: int64
>>>
```

# Look at the most popular star_rating among the products in the file.
print(reviews_df['star_rating'].value_counts())

```
>>> reviews_df['star_rating'].value_counts()
star_rating
5    129029
4      9808
1      4766
3      3147
2      1560
Name: count, dtype: int64
>>>
```

```
>>> print(reviews_dff['star_rating'].value_counts())
star_rating
5    80232
1    24848
4    20328
3    11589
2     7727
Name: count, dtype: int64
>>>
```

# Find the maximum and minimum review_date in the file.
reviews_df = pd.read_csv("s3://my-bigdata-project-sf/landing/amazon_reviews_us_Gift_Card_v1_00.tsv", sep='\t', on_bad_lines='skip', parse_dates=['review_date'])
print(reviews_df['review_date'].max())
print(reviews_df['review_date'].min())

```
>>> reviews_df = pd.read_csv("s3://my-bigdata-project-sf/landing/amazon_reviews_us_Gift_Card_v1_00.tsv", sep='\t', on_bad_lines='skip', parse_dates=['review_d
ate'])
>>> reviews_df['review_date'].max()
Timestamp('2015-08-31 00:00:00')
>>> reviews_df['review_date'].min()
Timestamp('2004-10-14 00:00:00')
>>>
```

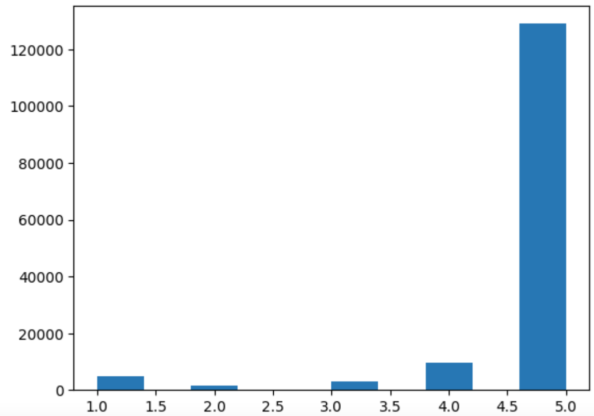
```
>>> reviews_dff = pd.read_csv("s3://my-bigdata-project-sf/landing/amazon_reviews_us_Digital_Video_Games_v1_00.tsv", sep='\t', on_bad_lines='skip', parse_dates
=['review_date'])
>>> reviews_dff['review_date'].max()
Timestamp('2015-08-31 00:00:00')
>>> reviews_dff['review_date'].min()
Timestamp('2006-08-08 00:00:00')
>>>
```

# Create a histogram of the star_rating data.
import matplotlib.pyplot as plt
plt.hist(reviews_df['star_rating'])

```
[30]: import matplotlib.pyplot as plt
```

```
[34]: # Create a histogram of the star_rating data.
      plt.hist(reviews_df['star_rating'])
```

```
[34]: (array([  4766.,      0.,   1560.,      0.,      0.,   3147.,      0.,
           9808.,      0., 129029.]),
       array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
       <BarContainer object of 10 artists>)
```



APPENDIX C
FEATURE ENGINEERING AND MODELING CODE

Data Cleaning Code:

```
# Install the necessary functions.
%pip install textblob
%pip install s3fs

# Import the necessary functions.
from pyspark.sql.functions import col, isnan, when, count, udf
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import sklearn
import numpy
import scipy
import plotly
import bs4 as bs
import urllib.request
import boto3

import os
# To work with Amazon S3 storage, set the following variables using AWS Access Key and
Secret Key.
# Set the Region to where the files are stored in S3.
access_key = '_____'
secret_key = '_____'
# Set the environment variables so boto3 can pick them up later.
os.environ['AWS_ACCESS_KEY_ID'] = access_key
os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key
encoded_secret_key = secret_key.replace("/", "%2F")
aws_region = "us-east-2"

# Update the Spark options to work with AWS Credentials.
sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region + ".amazonaws.com")
```

```
# Import the necessary functions.
from pyspark.sql.functions import col, isnan, when, count, udf

# Set the Spark logging level to only show errors.
sc.setLogLevel("ERROR")
# Set up the path to an Amazon reviews data stored on S3.
bucket = 'my-bigdata-project-sf/landing/'
filename = 'amazon_reviews_us_Gift_Card_v1_00.tsv'
file_path = 's3a://' + bucket + filename

# Create a Spark Dataframe from the file on S3.
sdf = spark.read.csv(file_path, sep='\t', header=True, inferSchema=True)

# Check the schema.
sdf.printSchema()

# Get the number of records in the dataframe.
sdf.count()

# Get some statistics on each of the columns.
sdf.summary().show()

# Check to see if any relevant columns have null values.
sdf.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in ["product_title",
"product_category", "star_rating", "helpful_votes", "total_votes", "vine", "verified_purchase",
"review_headline", "review_body"]] ).show()

# Drop some of the records where the relevant columns are empty.
sdf = sdf.na.drop(subset=["product_title", "product_category", "star_rating", "helpful_votes",
"total_votes", "vine", "verified_purchase", "review_headline", "review_body"])

# Define a function to strip out any non-ascii characters.
def ascii_only(mystring):
    if mystring:
        return mystring.encode('ascii', 'ignore').decode('ascii')
    else:
        return None

# Turn this function into a User-Defined Function (UDF).
ascii_udf = udf(ascii_only)
```

# Clean up the review_headline and review_body.
sdf = sdf.withColumn("clean_review_headline", ascii_udf('review_headline'))
sdf = sdf.withColumn("clean_review_body", ascii_udf('review_body'))

# Review the cleaned review_headline and review_body.
sdf.select("clean_review_headline", "clean_review_body").summary("count", "min", "max").show()

# Drop the columns that aren't relevant for feature engineering and modeling.
# Also drop review_headline and review_body, since the cleaned version will be used for feature engineering and modeling.
sdf = sdf.drop("marketplace", "customer_id", "review_id", "product_id", "product_parent", "review_date", "review_headline", "review_body")

# Check the schema now that null values are removed and irrelevant columns were dropped.
sdf.printSchema()

# Check the number of records now that null values are removed and irrelevant columns were dropped.
sdf.count()

# Save the cleaned dataframe in raw folder of S3 bucket as a Parquet file.
output_file_path="s3://my-bigdata-project-sf/raw/cleaned_amazon_reviews_us_Gift_Card_v1_00.parquet"
sdf.write.parquet(output_file_path)

Modeling Code:

# Import the necessary functions.
from pyspark.sql.functions import col, isnan, when, count, udf

# Set the Spark logging level to only show errors.
sc.setLogLevel("ERROR")

# Set up the path to an Amazon reviews data stored on S3.
bucket = 'my-bigdata-project-sf/raw/'
filename = 'cleaned_amazon_reviews_us_Gift_Card_v1_00.parquet'
file_path = 's3a://' + bucket + filename

Create a Spark Dataframe from the file on S3.

```
sdf = spark.read.parquet(file_path, sep='\t', header=True, inferSchema=True)

# Take a 1% sample of the data.
sdf = sdf.sample(False, 0.01)

# Check the schema.
sdf.printSchema()

# Tokenize clean_review_body, and then repeat process for clean_review_headline.
from pyspark.ml.feature import Tokenizer, RegexTokenizer

regexTokenizer = RegexTokenizer(inputCol="clean_review_body",
outputCol="review_body_words", pattern="\\w+", gaps=False)
review_body_words_sdf = regexTokenizer.transform(sdf)

review_body_words_sdf.select("clean_review_body",
"review_body_words").show(truncate=False)

# Generate the term-frequency for clean_review_body, and then repeat process for
clean_review_headline.
from pyspark.ml.feature import HashingTF

hashingTF = HashingTF(numFeatures=4096, inputCol="review_body_words",
outputCol="review_body_vector")
term_freq_sdf = hashingTF.transform(review_body_words_sdf)
term_freq_sdf.select(['review_body_words','review_body_vector']).show(truncate=False)

# Generate the inverse document frequency for clean_review_body, and then repeat process for
clean_review_headline.
from pyspark.ml.feature import IDF

idf = IDF(inputCol='review_body_vector', outputCol="review_body_features", minDocFreq=1)
idfModel = idf.fit(term_freq_sdf)
scaled_sdf = idfModel.transform(term_freq_sdf)
scaled_sdf.select("clean_review_body", "review_body_features").show(truncate=False)

# Textblob for sentiment analysis.
from textblob import TextBlob
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import col, isnan, when, count, udf
```

```
# Create a function to perform sentiment analysis on clean_review_body, and then repeat process
for clean_review_headline.
def sentiment_analysis(some_text):
    sentiment = TextBlob(some_text).sentiment.polarity
    return sentiment

# Turn the function into a UDF.
sentiment_analysis_udf = udf(sentiment_analysis, DoubleType())
# Apply the sentiment analysis function to the text-based columns and then create a new column.
sdf = sdf.withColumn("body_sentiment_score",
sentiment_analysis_udf(sdf['clean_review_body']))

# Display the results.
sdf.show(truncate=False)

# Check the schema, now that the sentiment score columns were added.
sdf.printSchema()

# Import the necessary functions.
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler

# Set up the StringIndexer and OneHotEncoder.
indexer = StringIndexer(inputCols=['product_title', 'product_category', 'vine',
'verified_purchase'], outputCols=['product_titleIndex', 'product_categoryIndex', 'vineIndex',
'verified_purchaseIndex'])
indexed_sdf = indexer.fit(sdf).transform(sdf)

encoder = OneHotEncoder(inputCols=['helpful_votes', 'total_votes', 'product_titleIndex',
'product_categoryIndex', 'vineIndex', 'verified_purchaseIndex'], outputCols=['star_ratingVector',
'helpful_votesVector', 'total_votesVector', 'product_titleVector', 'product_categoryVector',
'vineVector', 'verified_purchaseVector'], dropLast=False)
encoded_sdf = encoder.fit(indexed_sdf).transform(indexed_sdf)

# Assemble all of the vectors into one.
assembler = VectorAssembler(inputCols=[ 'body_sentiment_score', 'headline_sentiment_score',
'helpful_votesVector', 'total_votesVector', 'product_titleVector', 'product_categoryVector',
'vineVector', 'verified_purchaseVector'],outputCol= "features")
assembled_sdf = assembler.transform(encoded_sdf)
```

```
assembled_sdf.select(['body_sentiment_score', 'headline_sentiment_score', 'helpful_votesVector',
'total_votesVector', 'product_titleVector', 'product_categoryVector', 'vineVector',
'verified_purchaseVector', 'features']).show (truncate=False)

from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
# Import the logistic regression model.
from pyspark.ml.classification import LogisticRegression, LogisticRegressionModel
# Import the evaluation module.
from pyspark.ml.evaluation import *
# Import the model tuning module.
from pyspark.ml.tuning import *
import numpy as np

# Create a label. =1 if star_rating = 5, =0 otherwise.
sdf = sdf.withColumn("label", when(sdf.star_rating == "5", 1.0).otherwise(0.0) )

# Create the pipeline.
pipe = Pipeline(stages=[indexer, encoder, assembler])

# Call .fit to transform the data.
transformed_sdf = pipe.fit(sdf).transform(sdf)

# Review the transformed features.
transformed_sdf.select('star_rating', 'body_sentiment_score', 'headline_sentiment_score', 'label',
'features').show(30, truncate=False)
```

```
|star_rating|body_sentiment_score|headline_sentiment_score|label|features
|
+-----------+--------------------+------------------------+-----+--------------------------------------------------------------------------------------------------
------+
|4          |0.09074074074074073 |0.0                     |0.0  |(616,[0,6,153,337,612,613,615],[0.09074074074074073,1.0,1.0,1.0,1.0,1.0,1.0])
|
|5          |0.4866666666666667  |0.0                     |1.0  |(616,[0,2,149,326,612,613,615],[0.4866666666666667,1.0,1.0,1.0,1.0,1.0,1.0])
|
|5          |0.09880952380952382 |0.625                   |1.0  |(616,[0,1,2,149,327,612,613,615],[0.09880952380952382,0.625,1.0,1.0,1.0,1.0,1.0,1.0])
|
|5          |0.10833333333333334 |0.43333333333333335     |1.0  |(616,[0,1,2,149,326,612,613,614],[0.10833333333333334,0.43333333333333335,1.0,1.0,1.0,1.0,1.0
,1.0])|
|5          |0.3                 |0.0                     |1.0  |(616,[0,3,152,450,612,613,615],[0.3,1.0,1.0,1.0,1.0,1.0,1.0])
|
|5          |0.5966666666666667  |0.3833333333333333      |1.0  |(616,[0,1,3,151,450,612,613,615],[0.5966666666666667,0.3833333333333333,1.0,1.0,1.0,1.0,1.0,1
.0])  |
|4          |0.4166666666666667  |0.0                     |0.0  |(616,[0,2,149,327,612,613,614],[0.4166666666666667,1.0,1.0,1.0,1.0,1.0,1.0])
|
|4          |0.471875            |0.0                     |0.0  |(616,[0,2,149,343,612,613,614],[0.471875,1.0,1.0,1.0,1.0,1.0,1.0])
|
```

```
# Split the data into 70% training and 30% test sets.
trainingData, testData = transformed_sdf.randomSplit([0.7, 0.3], seed=42)
```

```
# Create a LogisticRegression Estimator.
lr = LogisticRegression()

# Fit the model to the training data.
model = lr.fit(trainingData)

# Show model coefficients and intercept.
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)

# Test the model on the testData.
test_results = model.transform(testData)
# Show the test results.
test_results.select('star_rating', 'body_sentiment_score', 'headline_sentiment_score',
'rawPrediction', 'probability','prediction', 'label').show(truncate=False)
```

```
|star_rating|body_sentiment_score|headline_sentiment_score|rawPrediction                            |probability                              |prediction|lab
el|
+-----------+--------------------+------------------------+-----------------------------------------+-----------------------------------------+----------+---
--+
|4          |-0.05               |0.30625                 |[-35.274997448071,35.274997448071]       |[4.789203146376992E-16,0.9999999999999996] |1.0       |0.0
|
|5          |0.49642857142857144 |0.0                     |[-26.849947495362002,26.849947495362002]|[2.183815596682211E-12,0.9999999999978162] |1.0       |1.0
|
|5          |0.16666666666666669 |0.0                     |[-25.123126517971762,25.123126517971762]|[1.2279050488480122E-11,0.9999999999877209]|1.0       |1.0
|
|4          |0.29583333333333334 |0.18791666666666668     |[-0.6440842965519242,0.6440842965519242]|[0.34432386304575824,0.6556761369542418]   |1.0       |0.0
|
|5          |0.4583333333333333  |0.3833333333333333      |[-2.927691962361378,2.927691962361378]   |[0.05080150483235604,0.949198495167644]    |1.0       |1.0
|
|5          |0.75                |1.0                     |[-5.071696589561954,5.071696589561954]   |[0.006232680547042919,0.993767319452957]   |1.0       |1.0
|
|5          |0.4339646464646465  |0.0                     |[-33.57771079349486,33.57771079349486]   |[2.614480984208731E-15,0.9999999999999973] |1.0       |1.0
|
|1          |0.0                 |0.0                     |[-2.3233666812124505,2.3233666812124505]|[0.0892061432767791,0.9107938567232209]    |1.0       |0.0
|
```

```
# Show the confusion matrix.
test_results.groupby('label').pivot('prediction').count().sort('label').show()
```

```
+-----+---+---+
|label|0.0|1.0|
+-----+---+---+
|  0.0| 12| 42|
|  1.0| 15|384|
+-----+---+---+
```

```
# Save the confusion matrix.
cm = test_results.groupby('label').pivot('prediction').count().fillna(0).collect()

def calculate_recall_precision(cm):
    tn = cm[0][1]              # True Negative
    fp = cm[0][2]              # False Positive
    fn = cm[1][1]              # False Negative
    tp = cm[1][2]              # True Positive
```

```
    precision = tp / ( tp + fp )
    recall = tp / ( tp + fn )
    accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
    f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
    return accuracy, precision, recall, f1_score

print(calculate_recall_precision(cm))
```

```
(0.8741721854304636, 0.9014084507042254, 0.9624060150375939, 0.9309090909090908)
```

```
# Create a BinaryClassificationEvaluator to evaluate how well the model works.
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

# Create the parameter grid.
grid = ParamGridBuilder().build()

# Create the CrossValidator.
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Use the CrossValidator to Fit the training data.
cv = cv.fit(trainingData)

# Show the average performance over the three folds.
cv.avgMetrics

# Evaluate the test data using the cross-validator model.
# Reminder: We used Area Under the Curve.
evaluator.evaluate(cv.transform(testData))
```

```
0.6858349577647823
```

```
# Create a grid to hold hyperparameters.
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0] )
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid.
grid = grid.build()
print('Number of models to be tested: ', len(grid))

# Create the CrossValidator using the new hyperparameter grid.
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)
```

```
# Call cv.fit() to create models with all of the combinations of parameters in the grid.
all_models = cv.fit(trainingData)

print("Average Metrics for Each model: ", all_models.avgMetrics)
```

```
Number of models to be tested:  12
Average Metrics for Each model:  [0.5202635398361826, 0.5202635398361826, 0.6925221432961791, 0.5, 0.6997706824848405, 0.5, 0.7029375091967353, 0.5, 0.70467085
19883266, 0.5, 0.7054619375942507, 0.5]
```

```
# Gather the metrics and parameters of the model with the best average metrics.
hyperparams = all_models.getEstimatorParamMaps()[np.argmax(all_models.avgMetrics)]

# Print out the list of hyperparameters for the best model.
for i in range(len(hyperparams.items())):
    print([x for x in hyperparams.items()][i])

# Choose the best model.
bestModel = all_models.bestModel
print("Area under ROC curve:", bestModel.summary.areaUnderROC)
```

```
(Param(parent='LogisticRegression_36e91a28c1c3', name='regParam', doc='regularization parameter (>= 0).'), 1.0)
(Param(parent='LogisticRegression_36e91a28c1c3', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is a
n L2 penalty. For alpha = 1, it is an L1 penalty.'), 0.0)
Area under ROC curve: 0.8987490594431905
```

```
# Use the model 'bestModel' to predict the test set.
test_results = bestModel.transform(testData)

# Show the results. test_results.select('star_rating', 'helpful_votes', 'total_votes', 'vine',
'probability', 'prediction', 'label').show(truncate=False)

# Evaluate the predictions. Area Under ROC curve.
print(evaluator.evaluate(test_results))
```

star_rating	body_sentiment_score	headline_sentiment_score	prediction	label
4	-0.05	0.30625	1.0	0.0
5	0.49642857142857144	0.0	1.0	1.0
5	0.16666666666666669	0.0	1.0	1.0
4	0.29583333333333334	0.18791666666666668	1.0	0.0
5	0.4583333333333333	0.3833333333333333	1.0	1.0
5	0.75	1.0	1.0	1.0
5	0.4339646464646465	0.0	1.0	1.0
1	0.0	0.0	1.0	0.0
5	0.385	0.5	1.0	1.0
5	0.3333333333333333	0.0	1.0	1.0
5	0.705	0.0	1.0	1.0
5	0.24285714285714285	0.8	1.0	1.0
5	0.39999999999999997	0.6000000000000001	1.0	1.0
5	0.14285714285714288	0.7	1.0	1.0
5	0.27291666666666664	0.0	1.0	1.0
5	0.28433441558441563	1.0	1.0	1.0
5	0.5	0.5	1.0	1.0
5	0.75	1.0	1.0	1.0

```
# Save the best model.
model_path = "s3://my-bigdata-project-sf/models/amazon_reviews_logistic_regression_model"
bestModel.write().overwrite().save(model_path)

# Load up the model from disk.
myModel = LogisticRegressionModel.load(model_path)
```

APPENDIX D
DATA VISUALIZING CODE

```
# Import the necessary functions and modules.
import io
import pandas as pd
import s3fs
import boto3
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import col, isnan, when, count, udf, to_date, year, month,
date_format, size, split
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler

# Show the frequency of the 'star_rating' column.
star_counts_df = sdf.groupby('star_rating').count().sort('star_rating').toPandas()
# Set up a figure.
fig = plt.figure(facecolor='white')
# Bar plot of star_rating and count.
plt.bar(star_counts_df['star_rating'],star_counts_df['count'])
# fig.tight_layout()
plt.title("Count by Star Rating") plt.savefig("frequency_star_rating_matplotlib.png")

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the Matplotlib figure to the buffer.
fig.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/frequency_star_rating_matplotlib.png', 'wb') as
f:
    f.write(img_data.getbuffer())

# Take the star_rating and helpful_votes columns, and convert to a Pandas dataframe.
df = sdf.select('star_rating', 'helpful_votes').toPandas()
# Set the style for Seaborn plots.
sns.set_style("white")
```

```
# Create the relationship plot.
relationship_plot = sns.lmplot(x='star_rating', y='helpful_votes', data=df)

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
relationship_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/
star_rating_helpful_votes_relationship_seaborn.png', 'wb') as f:
    f.write(img_data.getbuffer())

# Convert the numeric values to vector columns.
vector_column = "correlation_features"
# Choose the numeric (Double) columns.
numeric_columns = ['total_votes', 'helpful_votes', 'star_rating', 'body_sentiment_score',
'headline_sentiment_score']
assembler = VectorAssembler(inputCols=numeric_columns, outputCol=vector_column)
sdf_vector = assembler.transform(sdf).select(vector_column)

# Create the correlation matrix, then get just the values and convert to a list.
matrix = Correlation.corr(sdf_vector, vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()
# Convert the correlation to a Pandas dataframe.
correlation_matrix_df = pd.DataFrame(data=correlation_matrix, columns=numeric_columns,
index=numeric_columns)
plt.figure(figsize=(16,5))

# Set the style for Seaborn plots.
sns.set_style("white")

sns.heatmap(correlation_matrix_df,
            xticklabels=correlation_matrix_df.columns.values,
            yticklabels=correlation_matrix_df.columns.values,  cmap="Greens", annot=True)

plt.savefig("correlation_matrix.png")
```

```
# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/correlation_matrix.png', 'wb') as f:
    f.write(img_data.getbuffer())

# Take the star_rating and body_sentiment_score columns, and convert to a Pandas dataframe.
df = sdf.select('star_rating', 'body_sentiment_score').toPandas()

# Set the style for Seaborn plots.
sns.set_style("white")

# Create the relationship plot.
body_sentiment_plot = sns.lmplot(x='star_rating', y='body_sentiment_score', data=df)

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
body_sentiment_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/star_rating_body_sentiment_seaborn.png', 'wb')
as f:
    f.write(img_data.getbuffer())

# Take the star_rating and headline_sentiment_score columns, and convert to a Pandas
dataframe.
df = sdf.select('star_rating', 'headline_sentiment_score').toPandas()

# Set the style for Seaborn plots.
sns.set_style("white")
```

```
# Create the relationship plot.
headline_sentiment_plot = sns.lmplot(x='star_rating', y='headline_sentiment_score', data=df)

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
headline_sentiment_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://my-bigdata-project-sf/models/star_rating_headline_sentiment_seaborn.png',
'wb') as f:
    f.write(img_data.getbuffer())

# Visualize regression results.
# Plot star_rating against predicted star_rating.
# Select and convert to a Pandas dataframe.
df = test_results.select('star_rating','prediction').toPandas()

# Set the style for Seaborn plots.
sns.set_style("white")

# Create a relationship plot between tip and prediction.
prediction_plot = sns.lmplot(x='star_rating', y='prediction', data=df)

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
prediction_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/star_rating_prediction_plot_seaborn.png', 'wb')
as f:
    f.write(img_data.getbuffer())
```

```
#Use bestModel to make an ROC curve.
plt.figure(figsize=(5,5))
plt.plot(bestModel.summary.roc.select('FPR').collect(),
         bestModel.summary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.savefig("roc1.png")
plt.show()

# Create a buffer to hold the figure.
img_data = io.BytesIO()
# Write the figure to the buffer.
plt.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data.
img_data.seek(0)
# Connect to the s3fs file system.
s3 = s3fs.S3FileSystem(anon=False)

with s3.open('s3://my-bigdata-project-sf/models/ROC_curve.png', 'wb') as f:
    f.write(img_data.getbuffer())
```