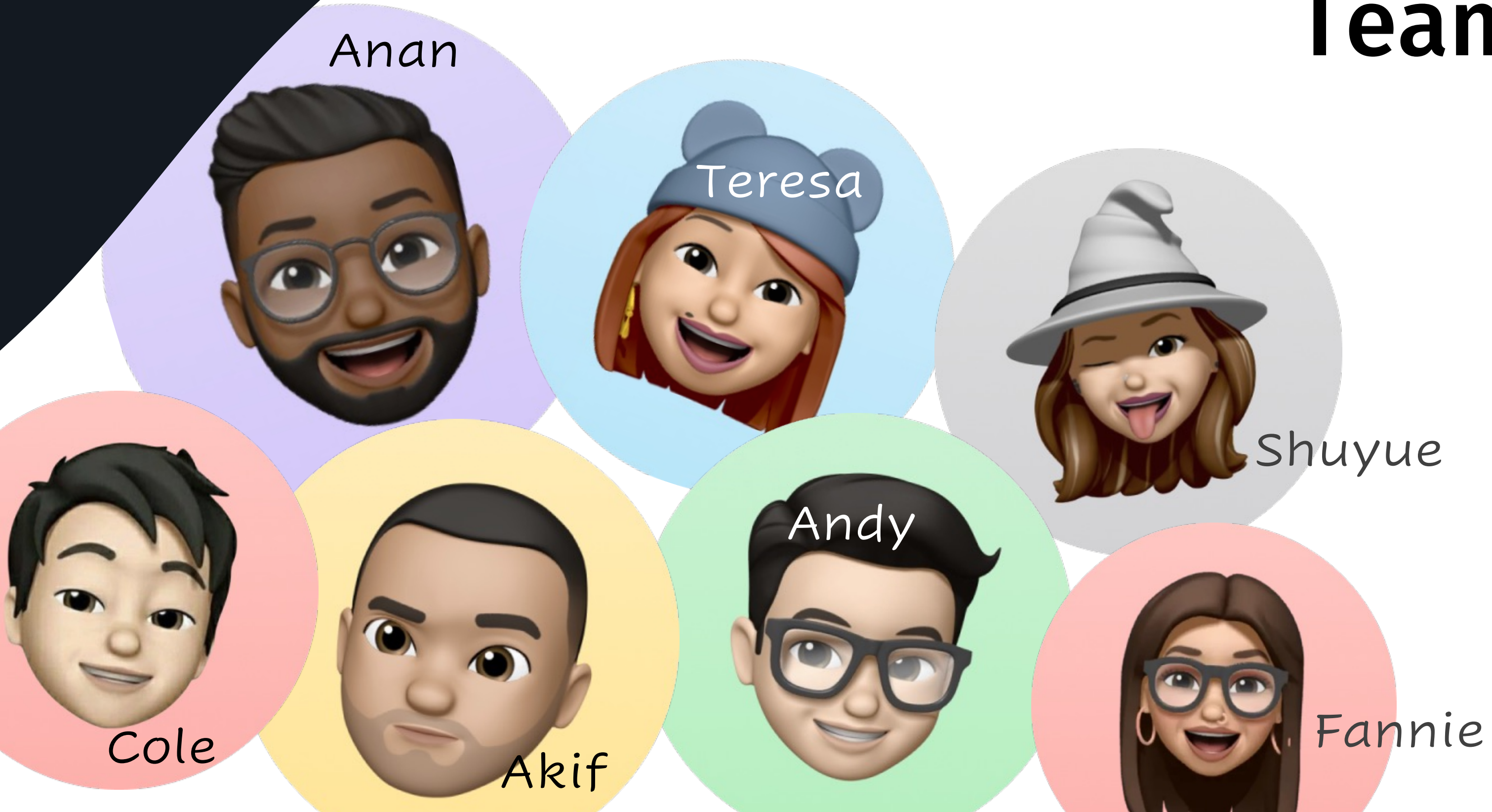
amazon

Understanding Customer Reviews
Team Cape Town



Our team: **Team Cape Town**

Agenda

- Overview
- Data Driven Insights
- Modelling Journey
- Model Evaluation
- LLM Integration
- Key Takeaways

Our team: **Team Cape Town**

Overview

Challenge

Lots of reviews per product, making it difficult for customers to find useful, relevant information, to assist with their purchase decisions

Current Situation

Poorly Written Reviews
Language & grammar errors make feedback difficult to decipher

Biased Reviews
Incentivized by discounts or brand affiliation

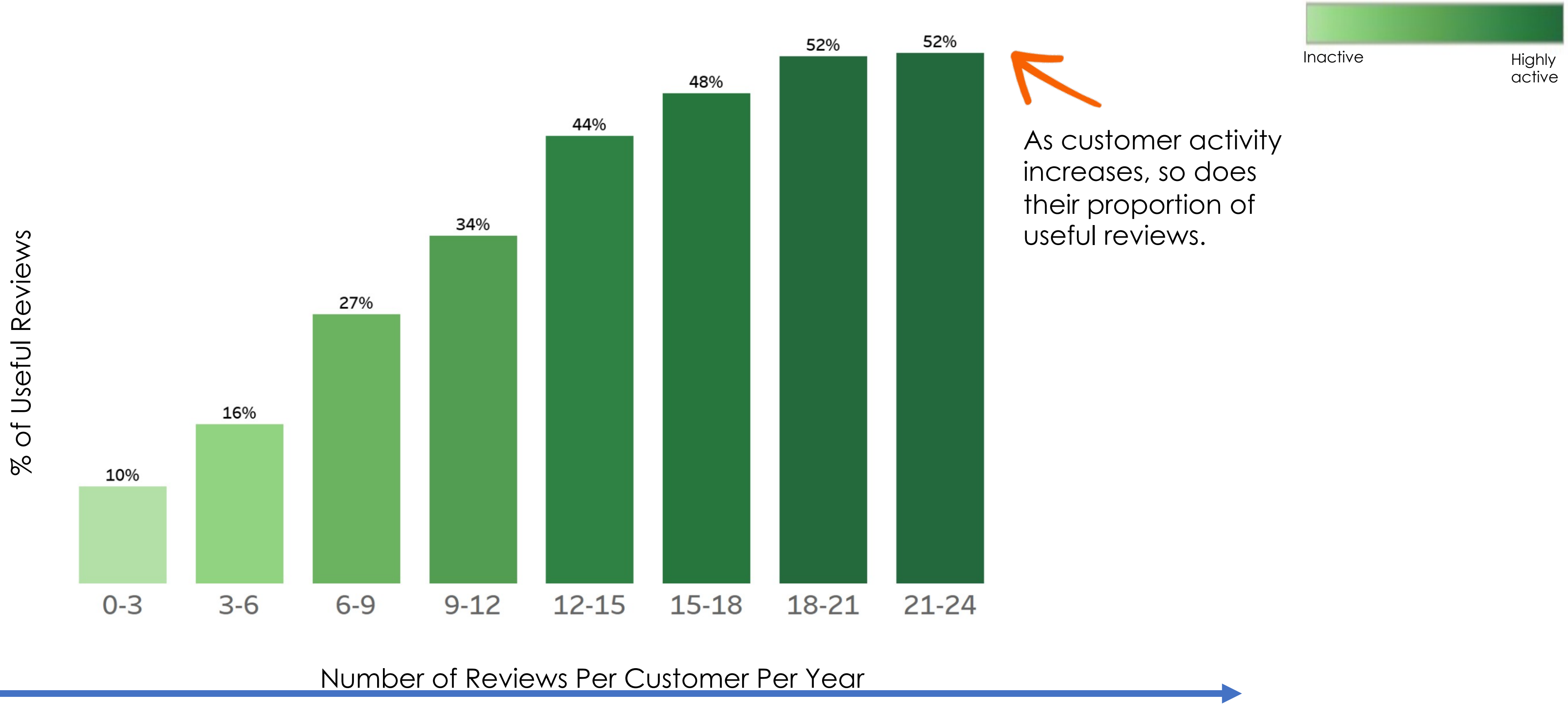
Unverified Reviews
No authentication for whether reviewer purchased the product

Duplicated Reviews
Reposting positive or negative comments

Goal

Identify reviews which are helpful to **customers** in their purchase decisions and to **Amazon** to improve their product assortment

Customer Engagement: More engaged customers leave useful reviews



As customer activity increases, so does their proportion of useful reviews.

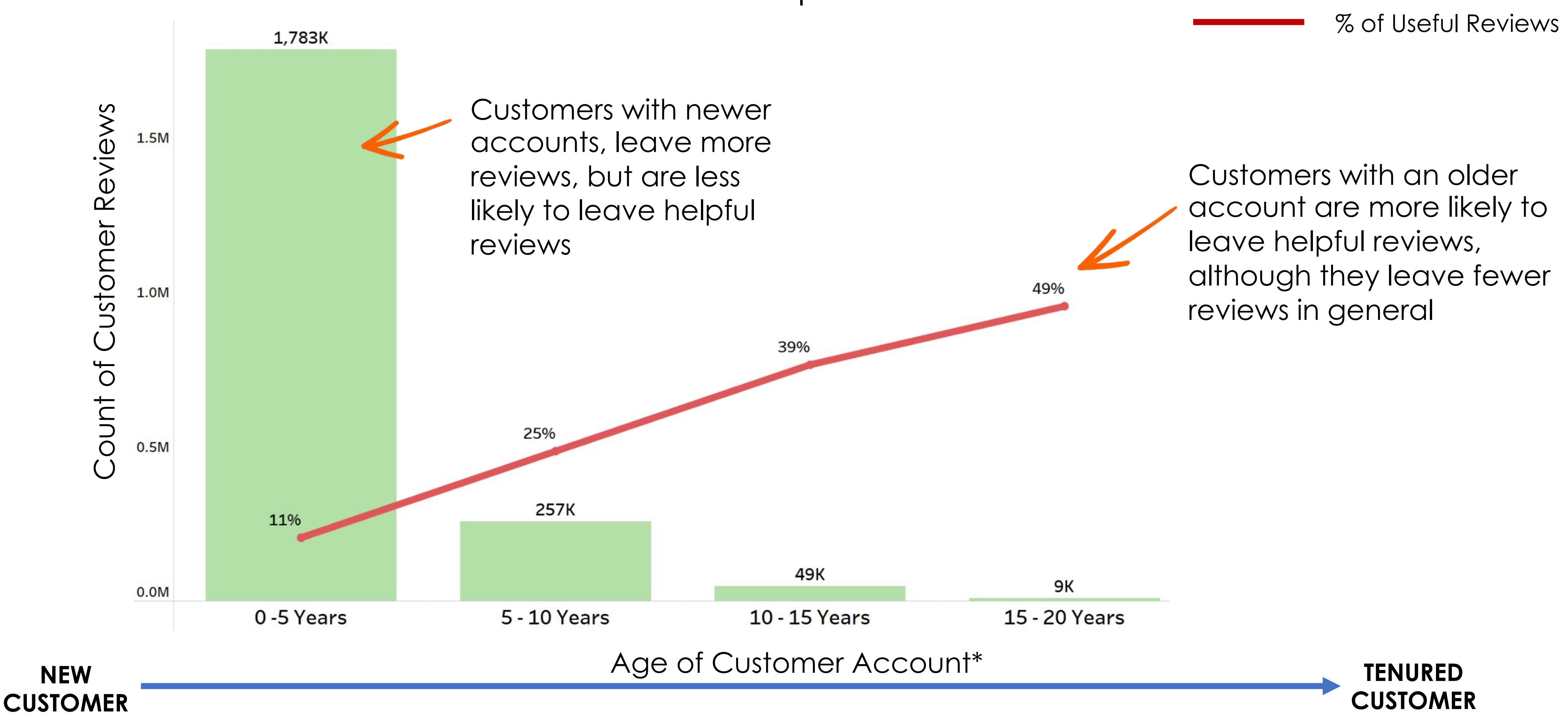
INACTIVE CUSTOMER

Increase in customer engagement

HIGHLY ACTIVE CUSTOMER

Data Driven Insights

Age of Customer Account*: Older accounts leave helpful reviews

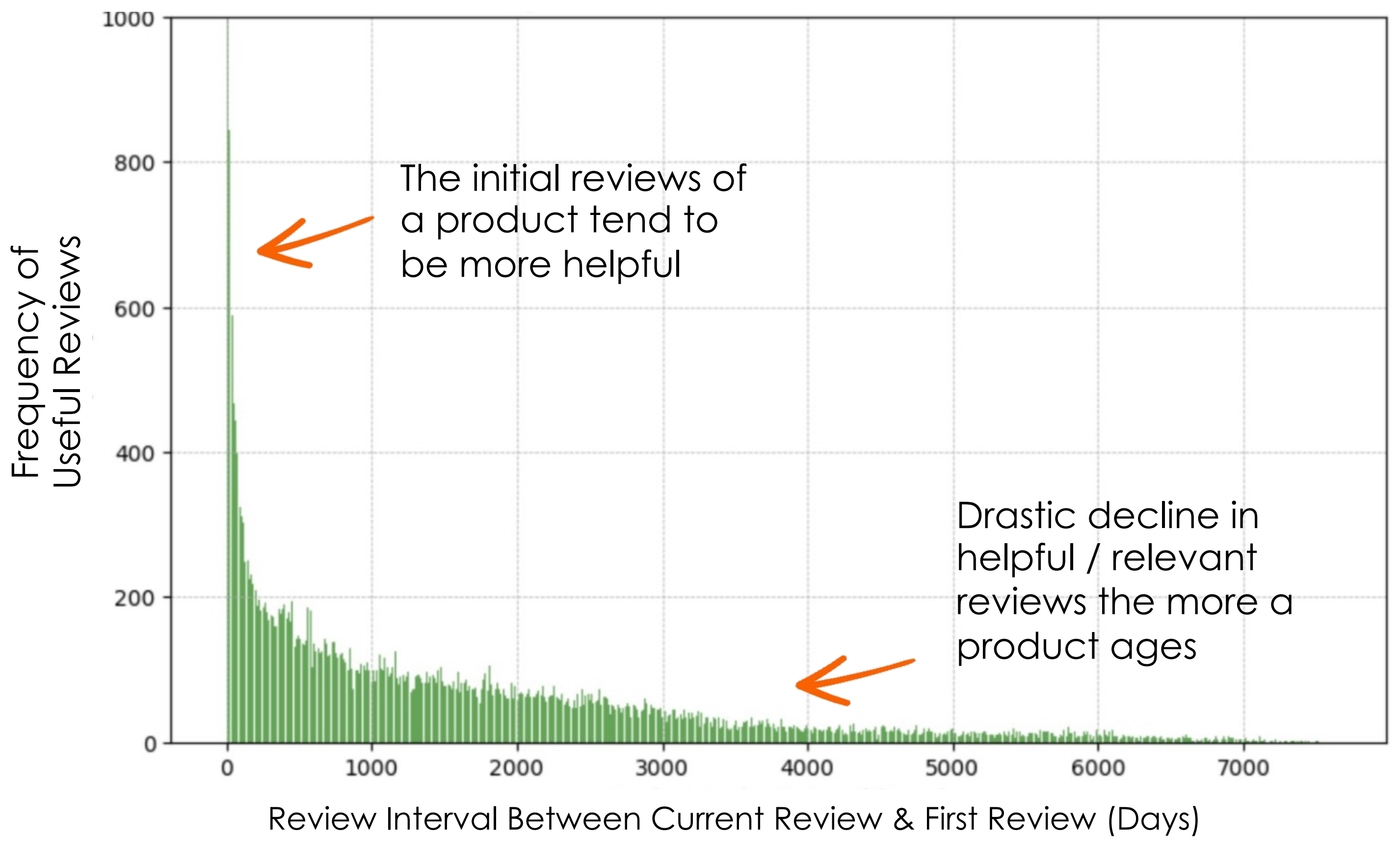


* This is a proxy for the age of a customer account, calculated as the time since the first review of a customer.

Our team: **Team Cape Town**

Data Driven Insights

Reviews Per Product: The initial reviews of a product are usually the helpful ones

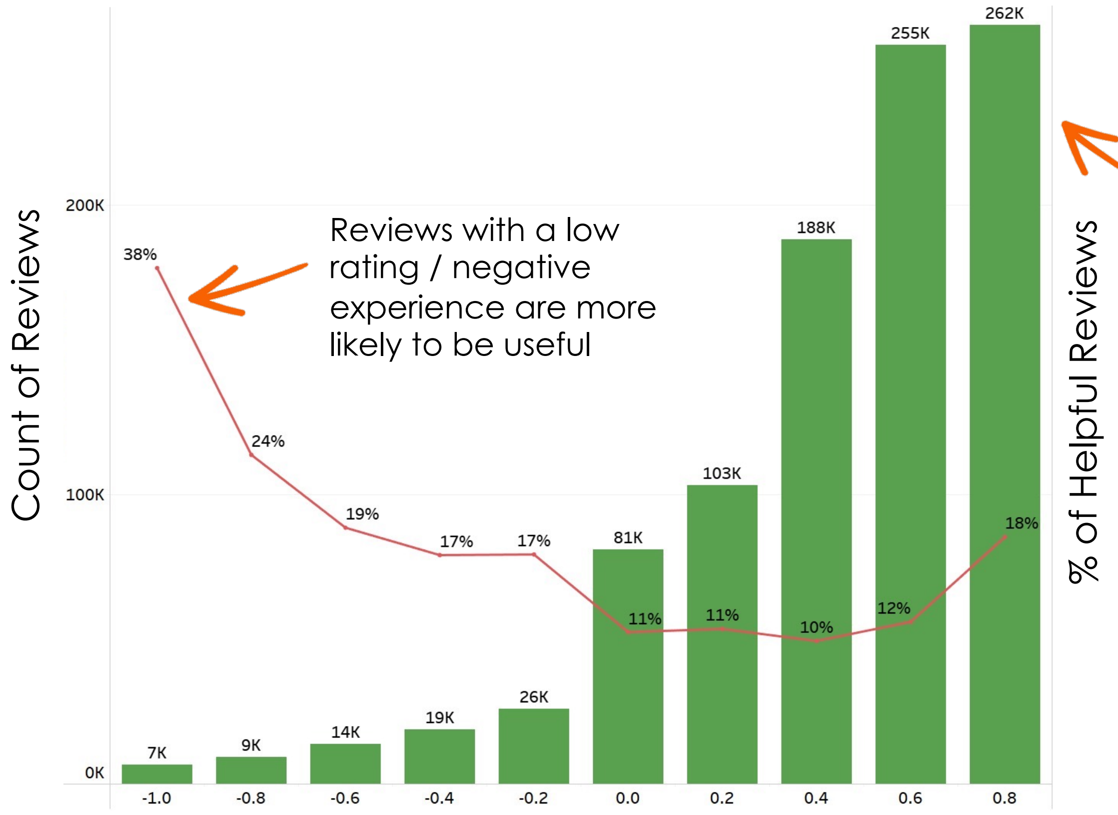


Our team: **Team Cape Town**

Data Driven Insights

Customer Sentiment: Negative reviews are more likely to be useful

% of Useful Reviews



Although more reviews tend to have a higher rating / positive sentiment, they are less likely to be helpful

NEGATIVE SENTIMENT — Sentiment Score — **POSITIVE SENTIMENT**

Our team: **Team Cape Town**

Modelling Journey

Objective: To flag customer reviews as helpful or not

Approach: Assess the recency, frequency & content of labelled, historically helpful reviews to build a predictive model, to spot similar patterns in recently generated reviews

CRISP-DM Modelling Process

Data Cleaning & Pre-Processing → Pipeline → Helpful / Not Helpful

01 Data Generation

Customers reviewing products on the Amazon website

02 Feature Engineering

Features created at the customer, product & review levels

03 Classification

The winning model for the Binary Classifier is **Light GBM.**

04 Model Tuning & Evaluation

Grid Search to identify best hyper parameters, optimizing for AUC

Our team: **Team Cape Town**

Modelling Journey

Feature Engineering

34 Features that Help Identify What Makes a Review Useful

CUSTOMER LEVEL

- Total reviews per customer
- Unique products reviewed
- Recency & Frequency of reviewing

CONTENT OF REVIEW

- Word count of each review
- Sentiment of review
- Count of question marks, caps lock & exclamation points in review

TEXT PREPROCESSING

- Tokenizing words
- Removing stop words
- Count Vector (Word Frequency)
- TF-IDF of frequently occurring words

SEASONALITY

- Day of week / month of review creation
- Black Friday, Boxing Day & Prime Week

PRODUCT LEVEL

- Reviews generated per product
- General sentiment of product
- Recency & Frequency of reviews

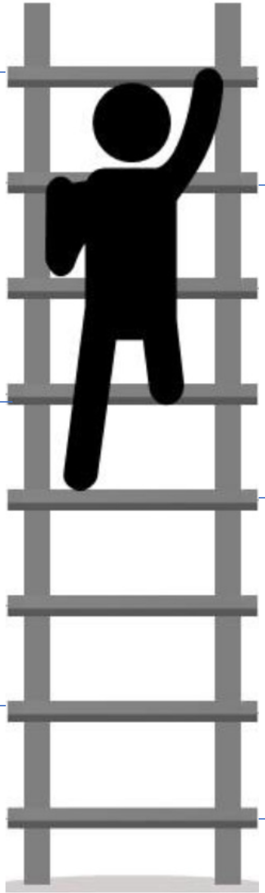
DIVERGENCE METRICS

- Does the review go against majority opinion?
- Does the customer only review when they have a negative experience?

Our team: **Team Cape Town**

Modelling Journey

Model Improvement



6 **Feature Selection + Hyperparameter Tuning**
AUC: ~ 0.89

5 **Fit Light GBM on Training Data**
AUC: ~ 0.88

4 **Added Features about the Content & Sentiment of Reviews**
AUC: ~ 0.86

3 **Added Date, Customer & Product Level Features**
AUC: ~ 0.85

2 **Applied Text Preprocessing Steps**
AUC: ~ 0.83

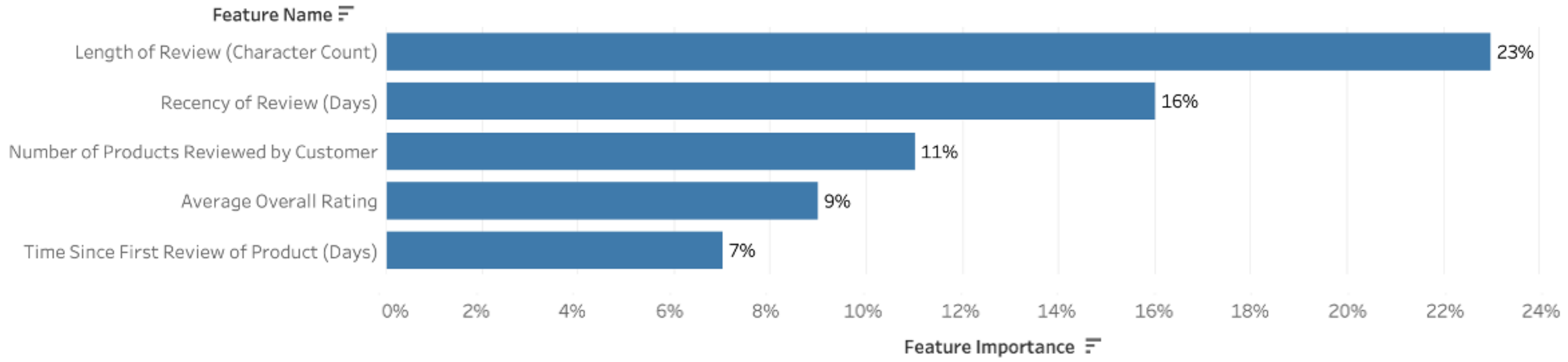
1 **Starting Model**
AUC: ~ 0.82

* All AUC values here represent the AUC on the Kaggle leader board.

Our team: **Team Cape Town**

Model Evaluation

Feature Importance Chart of Winning Model



Only top 5 features shown

Our team: **Team Cape Town**

amazon

LLM Integration

LLM Integration

✓ **Debugging Code:** Pasting the code and the error from the code provided a debugged version of the code, especially correcting for syntax issues.

Error Message:

```
AnalysisException: [AMBIGUOUS_REFERENCE] Reference `avg_sentiment_score_per_product` is ambiguous, could be: [`avg_sentiment_score_per_product`, `avg_sentiment_score_per_product`].
```

LLM Solution:

M can you debug the following code for me?

[AMBIGUOUS_REFERENCE] Reference `avg_sentiment_score_per_product` is ambiguous, could be: [`avg_sentiment_score_per_product`, `avg_sentiment_score_per_product`].

Explanation of error message

The error message `[AMBIGUOUS_REFERENCE] Reference 'avg_sentiment_score_per_product' is ambiguous` typically indicates that there are two columns in the dataframe with the same name, which is causing confusion during operations.

This situation can arise in a couple of scenarios:

1. **Joining Two DataFrames**: If you join two DataFrames and both have a column with the same name, then the resulting DataFrame will have ambiguous references for that column.
2. **Adding a Column Twice**: You might have accidentally added a column with the same name to the DataFrame more than once.

To fix the error, you'll need to resolve the ambiguous reference. Here's a potential solution based on a general scenario:

Regenerate

Scenario: Joining Two DataFrames

Let's assume the ambiguity arose because of joining two DataFrames `df1` and `df2`, both of which have the column `avg_sentiment_score_per_product`.

When joining the DataFrames, you can rename the columns in one of the DataFrames to avoid ambiguity:

python Copy code

```python
# Renaming the column in one DataFrame before joining
df2 = df2.withColumnRenamed("avg_sentiment_score_per_product", "avg_sentime

# Now perform the join
joined_df = df1.join(df2, on="common_column", how="inner")
```

Code sample

Now, the joined DataFrame `joined_df` will have two distinct columns: `avg_sentiment_score_per_product` (from `df1`) and `avg_sentiment_score_per_product_df2` (from `df2`).

Explanation of code snippet / solution

Now, the joined DataFrame `joined_df` will have two distinct columns: `avg_sentiment_score_per_product` (from `df1`) and `avg_sentiment_score_per_product_df2` (from `df2`).

Remember to adjust the column operations accordingly after renaming. If you need to access the value from the original column in `df1`, use `avg_sentiment_score_per_product`. If you need to access the value from the renamed column in `df2`, use `avg_sentiment_score_per_product_df2`.

In your given code, you might need to look at where the `df` dataframe is joined with `sentiment_product` and `result` dataframes. Ensure that none of the joining DataFrames have the column `avg_sentiment_score_per_product` before joining or, if they do, resolve the ambiguity as shown above.

LLM Integration

✓ **Code Refactoring:** ChatGPT made the code better formatted, legible and easier to edit, increasing efficiency, as the code becomes more "readable".

Pre LLM: 24 lines

```
def review_counts_per_product(data_input):

    review_counts = data_input.groupBy("asin").agg(
        countDistinct("reviewID").alias("reviews_per_product"),
        countDistinct('reviewerID').alias("reviewers_per_product"),
        min('reviewDate').alias('product_earliest_review'),
        max('reviewDate').alias('product_latest_review'))

    data_input = data_input.join(review_counts, on="asin", how="inner")

    data_input = data_input.withColumn('product_review_interval',
                                       datediff(data_input['product_latest_review'],
                                                data_input['product_earliest_review']))

    data_input = data_input.withColumn("product_review_interval",
                                       when(col("product_latest_review") == col
                                       ("product_earliest_review"), 0).otherwise(col
                                       ("product_review_interval")))

    data_input = data_input.withColumn('product_earliest_review',
                                       datediff(current_date(), data_input
                                       ['product_earliest_review']))

    data_input = data_input.withColumn('product_latest_review',
                                       datediff(current_date(), data_input
                                       ['product_latest_review']))

    return data_input
```

Post LLM: 17 lines (~29% reduction)

```
def review_counts_per_product(data_input):

    review_counts = (data_input.groupBy("asin")
                     .agg(F.countDistinct("reviewID").alias("reviews_per_product"),
                          F.countDistinct('reviewerID').alias("reviewers_per_product"),
                          F.min('reviewDate').alias('product_earliest_review'),
                          F.max('reviewDate').alias('product_latest_review')))

    review_counts = (review_counts.withColumn('product_review_interval',
                                              F.when(F.col("product_latest_review") == F.col
                                              ("product_earliest_review"), 0)
                                              .otherwise(F.datediff(F.col('product_latest_review'),
                                              F.col('product_earliest_review'))))
                                  .withColumn('product_earliest_review_age', F.datediff(F.
                                  current_date(), F.col('product_earliest_review')))
                                  .withColumn('product_latest_review_age', F.datediff(F.current_date
                                  (), F.col('product_latest_review'))))

    data_input = data_input.join(review_counts, on="asin", how="inner")

    return data_input
```

Compact code

Multiple new columns created all at once

Key Takeaways

Learnings

- ✓ **Feature Engineering:** The biggest improvement in model performance was brought about by adding features on how reviews are perceived by customers.
- ✓ **Feature Selection:** Relevant features need to be included in the model. Too many / too little features can erode performance.
- ✓ **Fitting Different Models:** Switching from Logistic Regression to Boosting models (Light GBM) improved model performance.
- ✓ **Cross Validation:** Cross-validation to optimize for hyper parameter values (using Optuna, Grid Search) improved model performance.
- ✓ **Sampling Data:** Data must be sampled to complete model training within a reasonable amount of time.

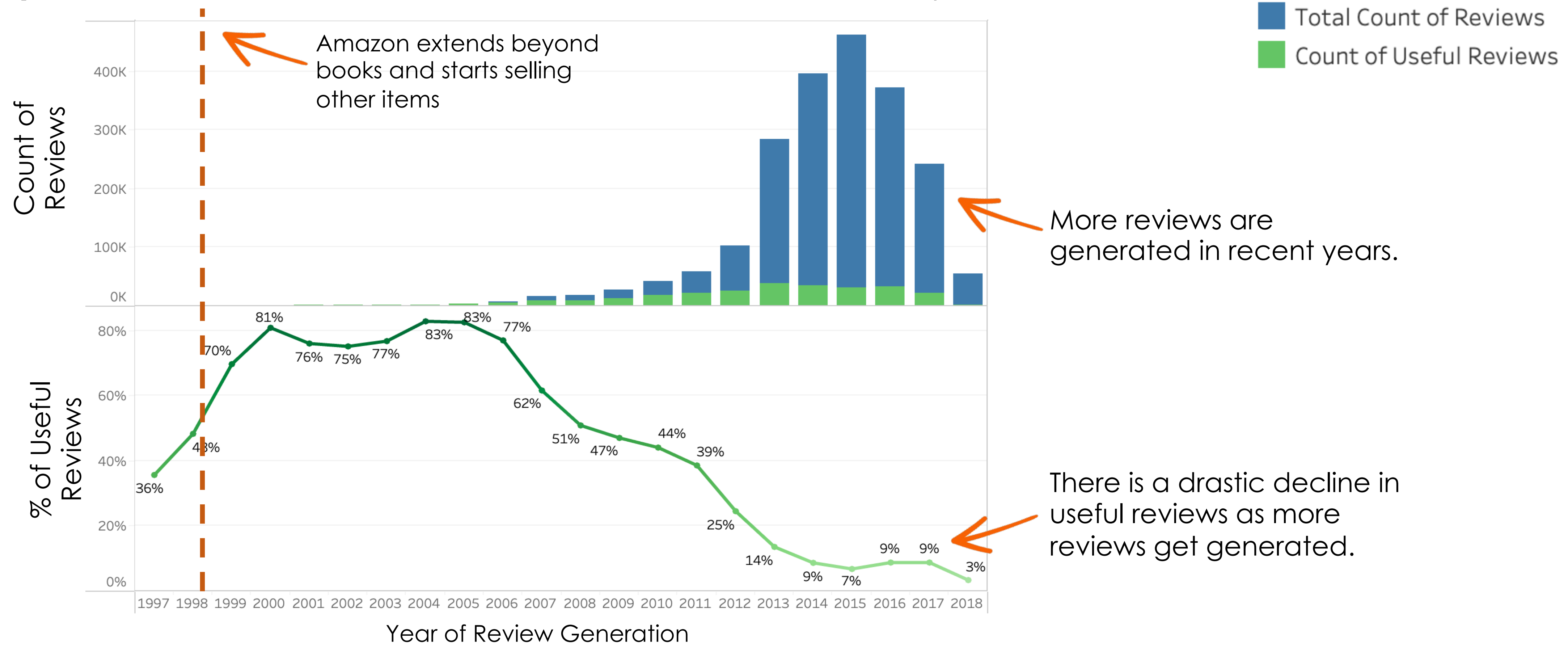
Thanks!

APPENDIX

Our team: **Team Cape Town**

Data Driven Insights

Recency of Reviews: A recent review does not mean it is a helpful one!



Our team: **Team Cape Town**

Data Driven Insights

Seasonality of Reviews: Review generation does not follow a seasonal trend

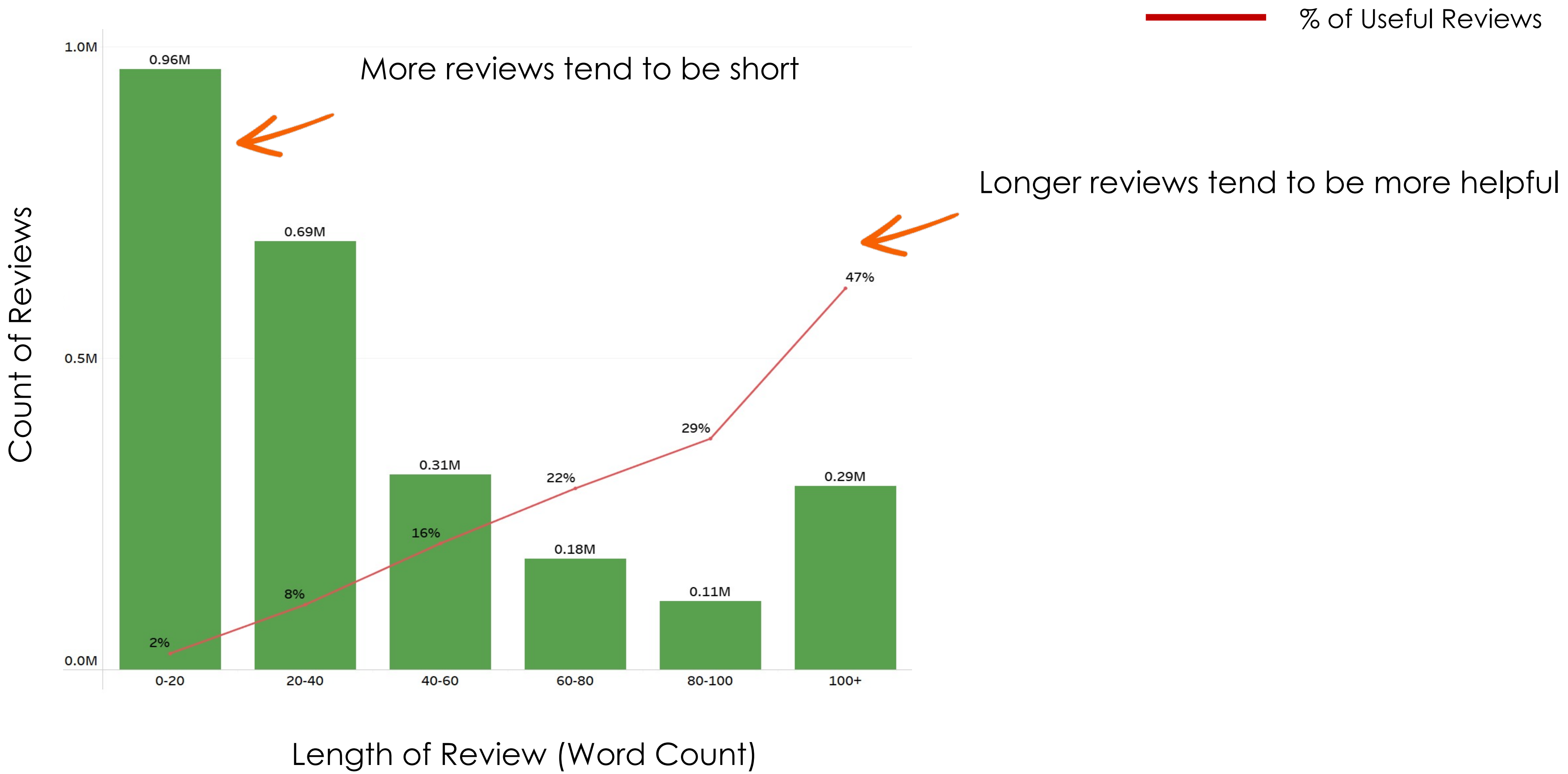
Review generation slightly increases in Dec & Jan, corresponding to Boxing Day & New Years. **There is no seasonality in review creation.**

The creation of **useful** reviews follows a steady trend, without demonstrating any seasonality.

Our team: **Team Cape Town**

Data Driven Insights

Length of Reviews: Longer reviews tend to be more helpful



Our team: **Team Cape Town**

Model Evaluation

Feature Importance Chart of Winning Model