# Assignment 2 NLP Analysis & Tasks

## Introduction

In this assessment a dataset was to be chosen and analyzed using a number of NLP techniques. This is important to do as we need to be able to understand and get meaningful conclusions from the data we can collect. Using NLP techniques big data can be processed and analyzed to discover meaning trends, meaning and tangible conclusions.

## Dataset

The most important part of this process is finding and choosing a suitable dataset that is free to use and will be good for analysis. When selecting a dataset some key things to consider are the following; data size, noise, usefulness, and usability. A dataset with a small amount of data is not good for analysis due to the potential lack of training data and bias. A bigger dataset is better for analysis as it gives you more to work with, giving consistent results. Datasets can become too big, but this can be scaled back easily enough. The dataset I chose has thousands of amazon reviews featuring a lot of columns, I do end up using most of them, but it is good to have the option of more if needed. The dataset was sourced from Kaggle.

## Process

With a wide range of things I was able to potentially do with the dataset I set a goal of making the analysis relevant to what the business would want to see analyzed. This led me to the analyzing of the sentiment, number of reviews for good and bad, topic modelling, summarization and finally spam detection. These all have value to the seller from Amazon.

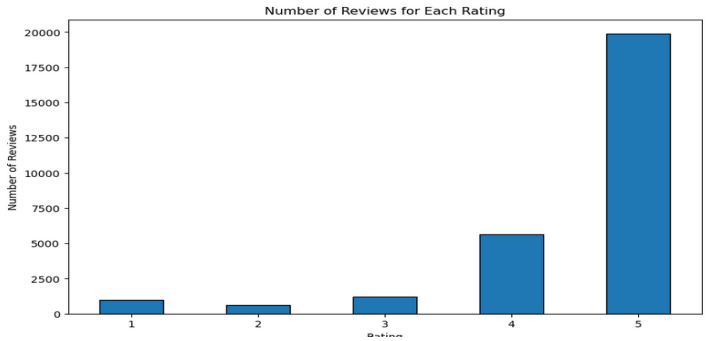
Firstly, I started with sentiment analysis. This is a good place to start as all sellers can find value in knowing the general sentiments that surround their product or service. I started by importing pandas and then loading in the dataset while saving it as a dataframe. Then I printed info about it so I could see it better in more detail.

Filtering out the unwanted columns was important as it makes working with the dataframe much easier, this helped. A crucial step I took next was preprocessing the data. This makes sure that the reviews are in a more raw and uniform way, making results more consistent. A defined a function to process each review. Starting with making all the text lowercase, then removing punctuation, tokenizing, removing stop words and lemmatization and then finally joining them back up again.


```
# Def preprocessing function to clean text
def preprocess_text(text):
    # Lowercasing, set to lowercase
    text = text.lower()

    # Removing punctuation and special characters
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Removing stop words and lemmatization
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]

    # Join tokens back into a single string
    preprocessed_text = ' '.join(tokens)

    # Return cleaned text
    return preprocessed_text
```

I checked the review distribution to see if there was a bias towards a certain way and there turned out there was. There were many more positive reviews than anything else. This was important to discover as any analysis would be biased towards positive reviews which can be bad for certain operations.

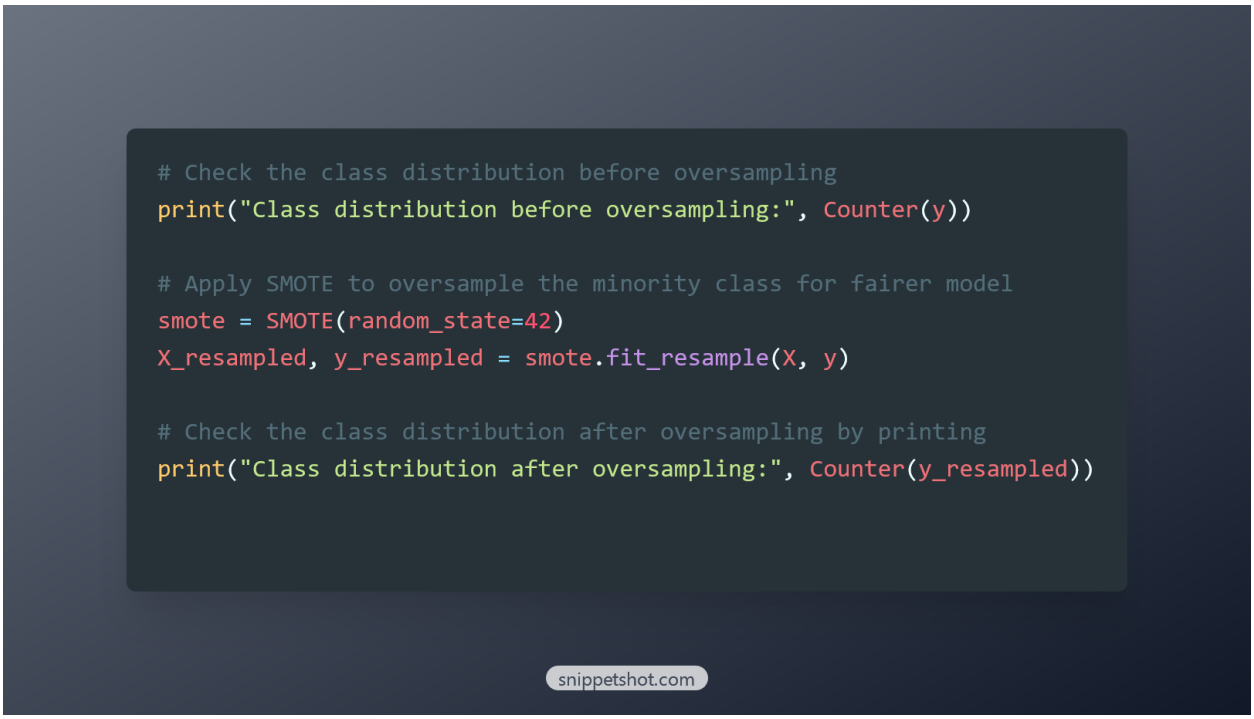
```
5    19897
4     5648
3     1206
1      965
2      616
Name: reviews.rating, dtype: int64
```




```python
# Import Matplot to show review rating distribution
import matplotlib.pyplot as plt

# Count the instances of each rating
rating_counts = df_filtered['reviews.rating'].value_counts()

# Display the counts
print(rating_counts)

# Plot the counts
plt.figure(figsize=(10, 6))
rating_counts.sort_index().plot(kind='bar', edgecolor='black')
plt.title('Number of Reviews for Each Rating')
plt.xlabel('Rating')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=0)
plt.show()
```



For sentiment analysts, you need a sentiment score to learn from. My dataset had ratings from 1-5. To make this more obvious I set ratings of 4 and 5 to a sentiment of 1 (being positive) and then set 1-2 to a sentiment of 0 (being negative). Finally, the ratings that were of 3 were removed to avoid ambiguous reviews. I split the dataset into test and train, 20-80. Then I trained the model. Testing the model revealed it would predict most things as positive despite being negative. I knew this meant the model was using the bias in positive reviews too much. I used SMOTE to rebalance the data by making synthetic data to match up to the number of positive reviews making it more balanced.




```
Accuracy: 0.9590917987864552
              precision    recall  f1-score   support

         0.0       0.94      0.98      0.96      5161
         1.0       0.98      0.94      0.96      5057

    accuracy                           0.96     10218
   macro avg       0.96      0.96      0.96     10218
weighted avg       0.96      0.96      0.96     10218
```

This made the model perform much better overall. Now the tests were getting all the sample reviews correct.


```
# Example new reviews ranging in sentiment from positive to negative
new_reviews = [
    "This product exceeded my expectations. I'm thrilled with its performance!",
    "I absolutely adore this product. It's a game-changer for me.",
    "The quality of this product is outstanding. I highly recommend it to everyone.",
    "I'm impressed by the durability and functionality of this product. It's worth every penny.",
    "This product has significantly improved my daily routine. I can't imagine life without it.",
    "I'm satisfied with this purchase. It meets my needs perfectly.",
    "The product works as advertised, but it's nothing extraordinary.",
    "It's an average product. Nothing special, but it gets the job done.",
    "I'm somewhat disappointed with the quality of this product. It doesn't live up to the hype.",
    "I regret buying this product. It's a waste of money and time.",
    "I hate this product, it sucks!",
    "This product could be better"
]

# Preprocess the new reviews
preprocessed_reviews = [preprocess_text(review) for review in new_reviews]

# Transform the preprocessed reviews into TF-IDF features
X_new = tfidf.transform(preprocessed_reviews).toarray()

# Predict sentiment for the new reviews
predictions = model.predict(X_new)

# Map the predictions back to sentiment labels
sentiment_labels = ['Negative' if pred == 0 else 'Positive' for pred in predictions]

# Display the results
for review, sentiment in zip(new_reviews, sentiment_labels):
    print(f"Review: {review}\nPredicted Sentiment: {sentiment}\n")
```

The next technique I worked on was topic modelling. I set to analysis the dataset and get 5 topics which make up 10 words related to it. I used a LDA model to get this.

Topic 0:

device amazon love kindle bought new just use old problem

Topic 1:

great good works tablet battery like screen life kindle use

Topic 2:

batteries great good price work long amazon just brand buy

Topic 3:

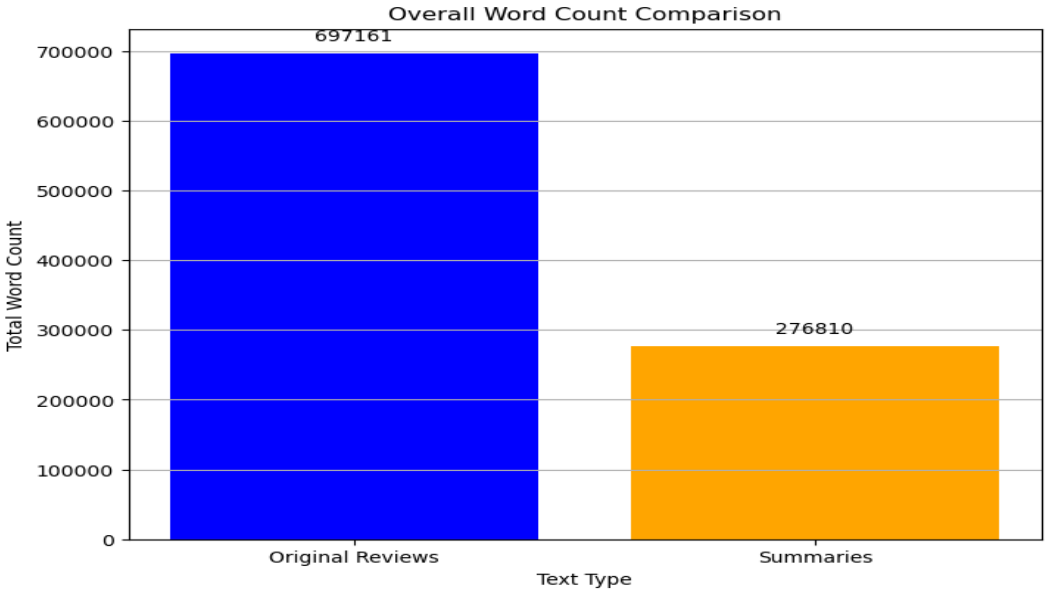
tablet great loves kids use old bought love easy year

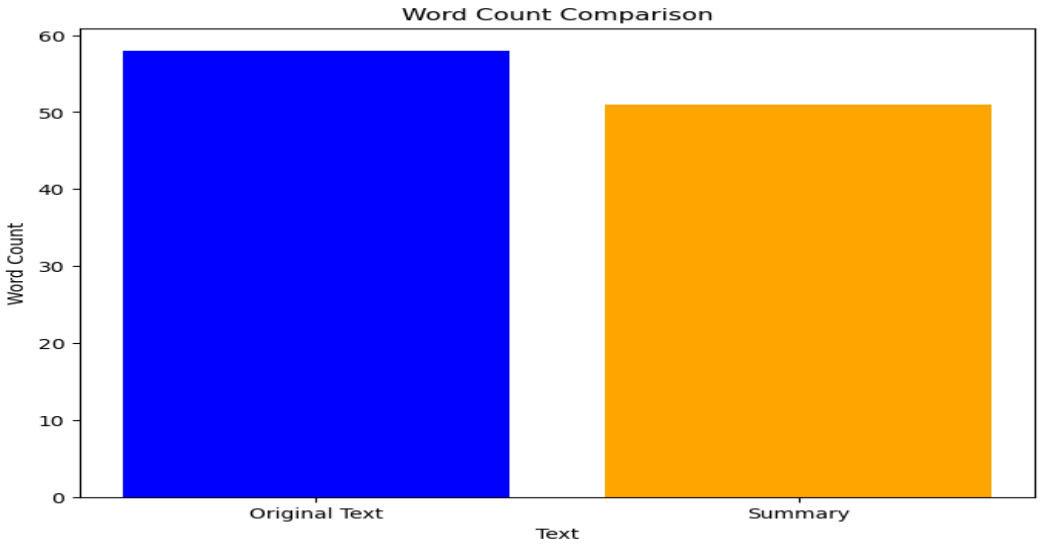
Topic 4:

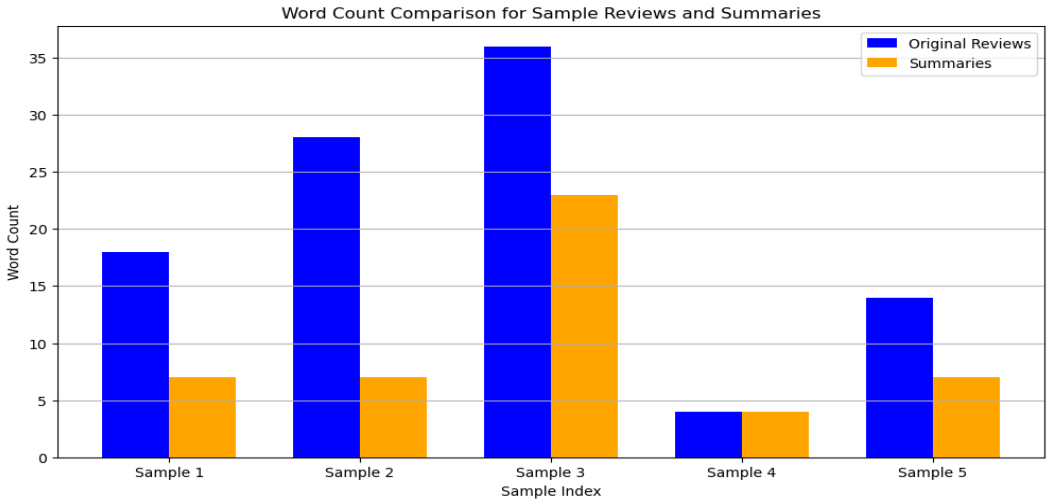
kindle tablet great use easy price amazon love books screen

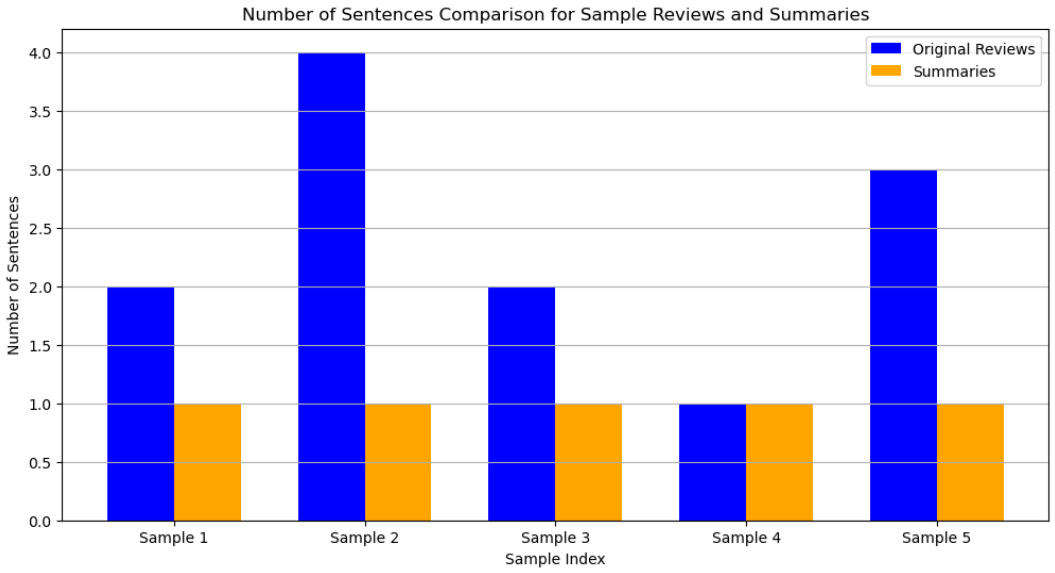
From this we can see that the first topic is about the amazon device kindle which is loved. I then made wordclouds of the topics showing the more important words.


Summarization can be an effective way to reduce a paragraph without losing its meaning. I started by testing a pretrained model which worked effectively. I did not want to use a pretrained model to do all the work, so I started developing my own way of summarizing text. Using NLTK and counter I achieved this. I tokenized the words and sentences, calculated the word frequency, scored each sentence, sorted the sentence by score, got the top sentences and then joined the sentences together. Using the most frequent words can identify the more important words and sentences in a review. I applied this model to the dataset and then visualized the results.


Overall Word Count Comparison
697161
276810
700000
600000
500000
400000
300000
200000
100000
0
Total Word Count
Original Reviews
Summaries
Text Type


Word Count Comparison
60
50
40
30
20
10
0
Word Count
Original Text
Summary
Text




The last technique was a spam filter or identification system. I did this by checking the reviews and looking for excessive capitalization. This found a good amount of spam. I could have expanded this to also include blacklisted words or repeated words or phrases. I used the dataset without any preprocessing.

```
Potential Spam Reviews:
HAVE NOT HAD CHANCE TO USE ALL OF THEM BUT HOPEFULLY ALL OF THEM WILL REMAIN IN GOOD ORDER FOR A REASONABLE AMOUNT OF TIME ME
ANING A FEW YEARS OUT.
GOT THE 48 PACK, I USEDD THEM UP FASTER THAN I THOUGHT I WOULD!!,SO IM ORDERING MORE !! SO IM ORDERING MORE NOW
LOVE THESE BATTERIES! BETTER THAN DURACELL AND SAVES ME MONEY! THANK YOU FOR A GREAT PRODUCT AMAZON!
NICE PRODUCT
A+A+A+A+A+ FAST DELIVERY JUST AS DESCRIBE
GOOD ALKALINE BATTERIES. AREN'T ALL ALKALINE THE SAME ADVERTISING STEERS SOME PEOPLE.
OK
GREAT!
HOPE THEY WORK!
AOK
AOK-AS ORDERED
GREAT VALUE
BUY THE BEST OR SUFFER LIKE THE REST
EXCELLENT BATTERIES FOR THE PRICE.
WORK GREAT
EXCELLENT PRODUCT
EXCELLENT PRODUCT,GREAT QUALITY AND VALUE
A GREAT BARGAIN ! ! !
AAA+
ALWAYS GOOD
AS EXPECTED
ASDFGHJ
BATTERIES! CHEAP!
BUENOS PRODUCTOS
CHEAP
COMES IN AWESOME AND SAFE PACKAGE AND JUST DOES THE JOD :)
EXCELLENT
EXCELLENT
EXCEPTIONAL.
F
FINE BATTERIES AT A WONDERFUL PRICE
FOR THE PRICE....THEY ARE OK
```

```python
# Load your dataset
df = pd.read_csv('Dataset.csv')

# Function to check for excessive capitalization
def has_excessive_capitalization(text):
    return sum(1 for char in text if char.isupper()) > len(text) * 0.5

# Apply rule-based filtering to identify potential spam reviews
df['potential_spam'] = df['reviews.text'].apply(has_excessive_capitalization)

# Display potential spam reviews
potential_spam_reviews = df[df['potential_spam'] == True]['reviews.text']
print("Potential Spam Reviews:")
for review in potential_spam_reviews:
    print(review)

# Count the number of reviews flagged as spam
num_spam_reviews = df['potential_spam'].sum()

# Count the number of reviews that weren't flagged as spam
num_non_spam_reviews = len(df) - num_spam_reviews

print("\nNumber of Reviews Flagged as Spam:", num_spam_reviews)
print("Number of Reviews That Weren't Flagged as Spam:", num_non_spam_reviews)

# remvoe from dataset to exclude spam if wanted
```

## Conclusion

In conclusion, I used a range of NLP techniques to analysis a kaggle dataset of amazon customer reviews to gain insight and knowledge. I applied visualizations to also understand the findings of the analysis better.

## References

*Consumer Reviews of Amazon Products*. (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/datafiniti/consumer-reviews-of-amazon-products