FINAL POE

ST10082757

Nande Mzantsi
PDAN8411

Table of Contents

INTRODUCTION AND PURPOSE

In this data driven day and age, customer feedback offers a lot of critical insights into user experience. With platforms like HelloPeter containing thousands of unstructured reviews which can be analysed to extract only topics of concern and emotional tone.

This report focuses on helping a medical aid provider understand how their services are being perceived by their customers. With the use of machine learning to HelloPeter reviews, the goal is to identify recurring issues and the overall sentiments expressed in these reviews. This will support evidence-based decision making and enable the medical aid scheme to respond efficiently and proactively to negative experiences online therefore improving the company's overall customer satisfaction.

After conducting research I have chosen machine learning models for the sentiment classification. These models are known for their high performance. To ensure a robust analysis this report also includes a step-by-step plan and the implementation.

Following a detailed analytical workflow allows the medical aid provider to gain a clear data driven understanding of the customer sentiment and the recurring themes that shape the public image therefore empowering the company to make informed decisions.

MODEL OF CHOICE AND JUSTIFICATION

According to IBM (2019) “With a machine learning (ML) approach, an algorithm is used to train software to gauge sentiment in a block of text using words that appear in the text as well as the order in which they appear”. Sentiment analysis is used to teach software how to identify emotions in text (similar to the way humans do). Sentiment analysis will help the medical aid provider find out the main areas of concern and to have a general sentiment of how the customer base is experiencing their services. This model will be analysing the areas of concern and customer sentiment. From conducting research I have found that there are a few most commonly used classification algorithms:

Logistic Regression

- Acharaya (2023) states that logistic regression is a statistical method which is used for binary classification. The goal is to predict the probability of an event or outcome occurring. This model is widely used in machine learning and statistics because it is simplistic and interpretable. Once a logistic regression model is trained it can be used to make predictions on new data. This model has several advantages but more specifically it can manage both categorical and continuous independent variables.

Naïve Bayes

- An algorithm that uses the Bayes Theorem to categorise words in a block of text, in short it classifies text into sentiment categories. Ashfaque (2023) states that this this model is used to predict the class of am observation given a set of features. Ashfaque (2023) adds that this model may appear overly simplistic, but it performs surprisingly well for sentiment analysis. It assumes that the presence of a particular feature in a class is unrelated to the presence of the other features in that class. This model can be applied to any classification problem where there are a set of features where we want to predict the class label.

Support Vector Machine

- Classification model used to solve two group classification problems. Reddy (2018) adds that “SVM can be used for both classification and regression challenges”. This model performs classification by finding the hyper-plane which differentiates the classes that were plotted in n-dimensional space.

Logistic Regression, Naïve Bayes and Support Vector Machine offer effective, robust and interpretable solutions for sentiment analysis. These models were selected because they are well-suited for text classification tasks like analysing the customers reviews to uncovers the areas of concern and the overall sentiment. Applying these models to our scenario, the medical aid company will gain valuable insights into the emotional tone expressed by their customers to ensure that data driven improvements are implemented for better customer satisfaction.

PLANNING

This model aims to build a sentiment analysis model which will classify medical aid reviews as either positive or negative based on text data. The goal is to extract the insights of the patient experience (at the hospitals the medical aids pay for) and support driven decision-making using machine learning. The dataset I used was sourced from Kaggle.

EXPLORATORY DATA ANALYSIS

- This tool is used to ensure that the data is clean, understandable and ready for modelling. To explore my dataset I will be taking steps to (not in this exact order):
  1. Understand the dataset, its structure and shape.
  2. Check for nulls and duplicates.
  3. Visualising the distributions using histograms and boxplots
  4. Detecting outliers using IQR or Z-score
  5. Checking and analysing basic correlations using a heatmap

FEATURE SELECTION

- From analysing the correlations, this tool will allow me to select the most important features to improve my model’s accuracy and performance. To do this I will (not in this exact order)::
  1. Remove any irrelevant or low variance features.
  2. I will be using statistical methods.

TRAIN MODEL

Here I will be training my model using the best algorithm and then tune it for optimal performance. I will do this by (not in this exact order):

1. Splitting the dataset into train and test data
2. Choosing an algorithm between Logistic Regression, SVM, and Naïve Bayes
3. Hyperparameter tuning will be applied.
4. Grid search will be applied.
5. Cross validation will be applied.

INTERPRET AND EVALUATE MODEL

Finally the model performance is evaluated, and the results are interpreted for decision-making. I will be conducting the steps by (not in this exact order):

1. Using evaluation metrics
2. Classification: Accuracy, Precision, Recall, F1-score, AUC-ROC
3. Regression: RMSE, MAE R Squared
4. Confusion Matrix for classification

5. Feature importance plots
6. Residual plots for regression

IMPORTS

Imports are tools that are needed to explore the data, clean it, analyse it and visualise it.

Data Manipulation : pandas

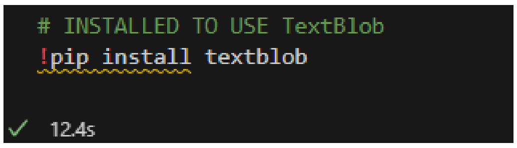
Visualisation : seaborn, matplotlib, WordCloud, STOPWORDS

Text Analysis and Preprocessing : Counter, TextBlob, Count Vectorizer, TfidVectorizer

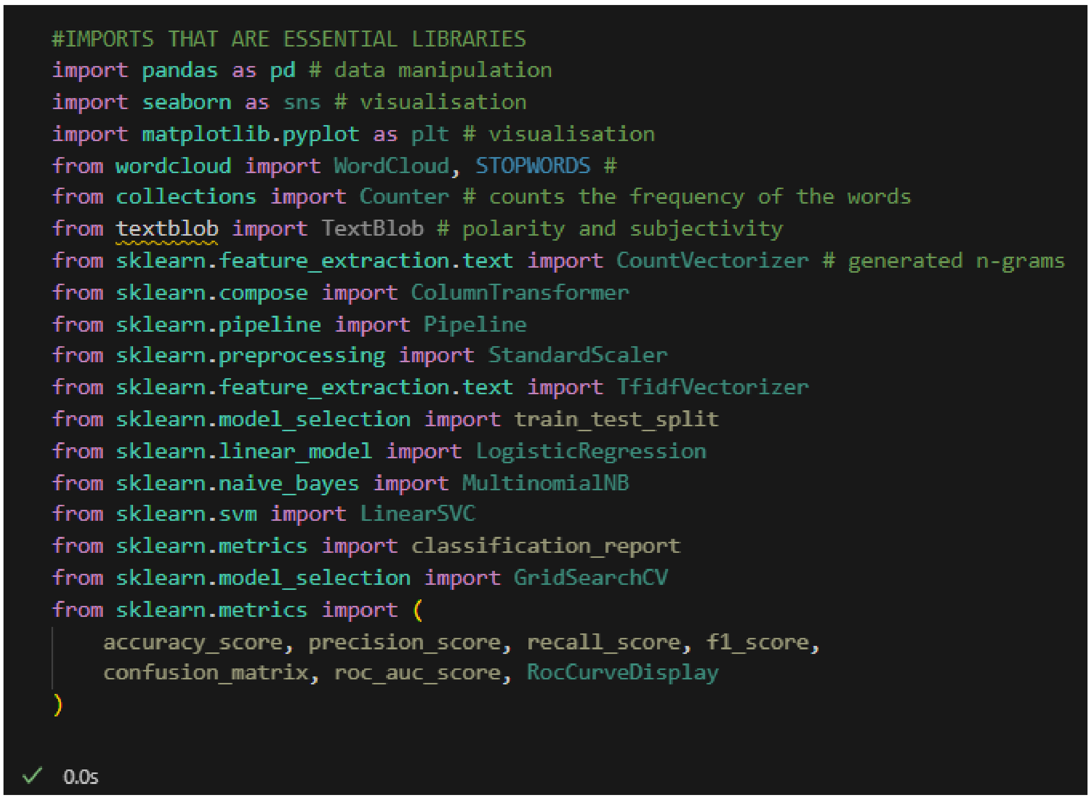
Pipeline Construction and feature engineering : ColumnTransformer, Pipeline, StandardScaler

Machine Learning Algorithms : LogisticRegression, MultinomialNB, LinearSVC

Model Evaluation and Tuning : classification_report, accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score, RocCurveDisplay, GridSearchCV



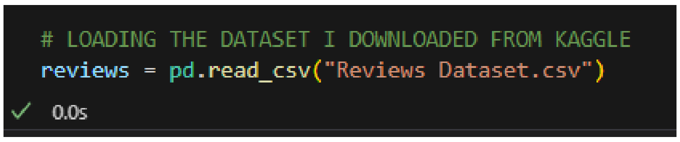
```
# INSTALLED TO USE TextBlob
!pip install textblob

✓ 12.4s
```



```
#IMPORTS THAT ARE ESSENTIAL LIBRARIES
import pandas as pd # data manipulation
import seaborn as sns # visualisation
import matplotlib.pyplot as plt # visualisation
from wordcloud import WordCloud, STOPWORDS #
from collections import Counter # counts the frequency of the words
from textblob import TextBlob # polarity and subjectivity
from sklearn.feature_extraction.text import CountVectorizer # generated n-grams
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_auc_score, RocCurveDisplay
)

✓ 0.0s
```

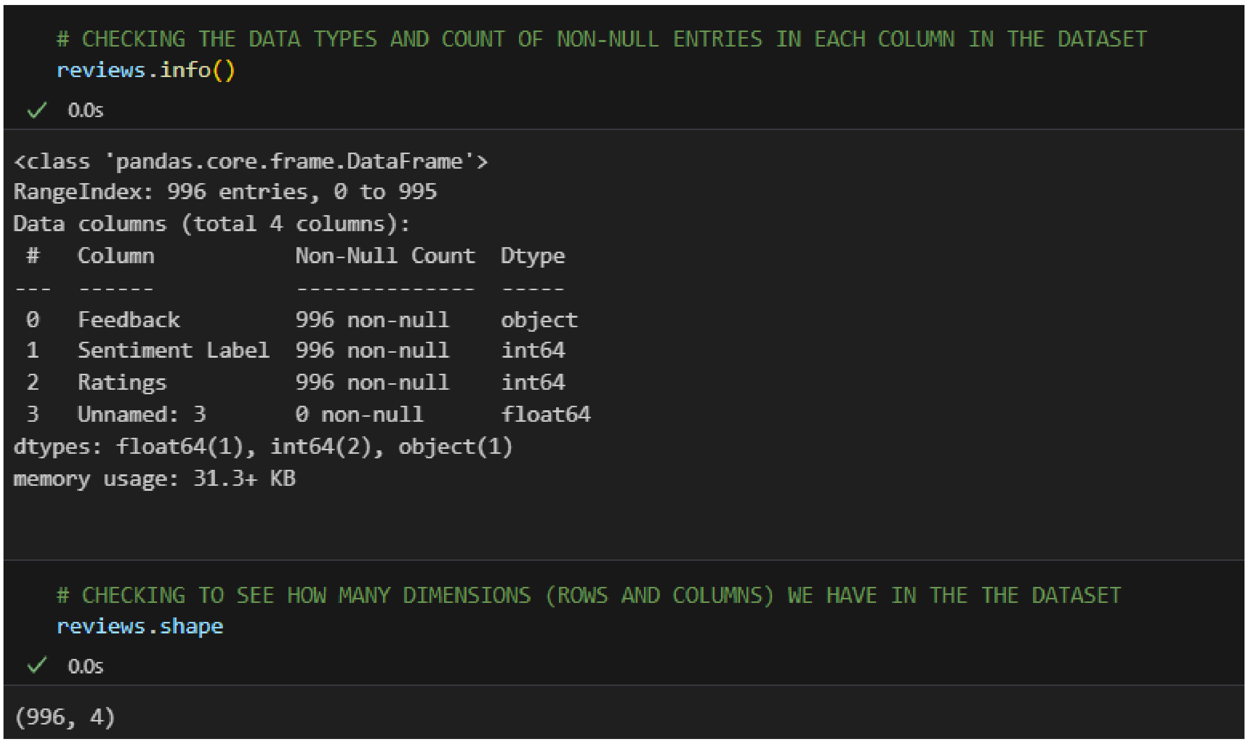
EXPLORATORY DATA ANALYSIS

Once the essential libraries are loaded, the exploration of the dataset can begin. The exploration starts with loading the dataset into a DataFrame. Without loading the dataset, the analysis cannot begin. The insights of this analysis are dependent on the data being loaded correctly. Running this line of code came with no errors which gave me an indication to continue with my analysis.



```
# LOADING THE DATASET I DOWNLOADED FROM KAGGLE
reviews = pd.read_csv("Reviews Dataset.csv")
✓ 0.0s
```

My analysis starts with displaying a summary of the dataset using **.info()** which includes the data types of each column, the number of non-null entries and the memory usage. With this information I can identify missing values or incorrect datatypes. Making use of **.shape()** allowed me to get an idea of the dimensions (columns and rows) of the DataFrame. This dataset has 996 rows and four columns before processing. This is to ensure that the expected number of rows and columns was loaded correctly. This step is important for the analysis to identifying things that require attention prior to processing.

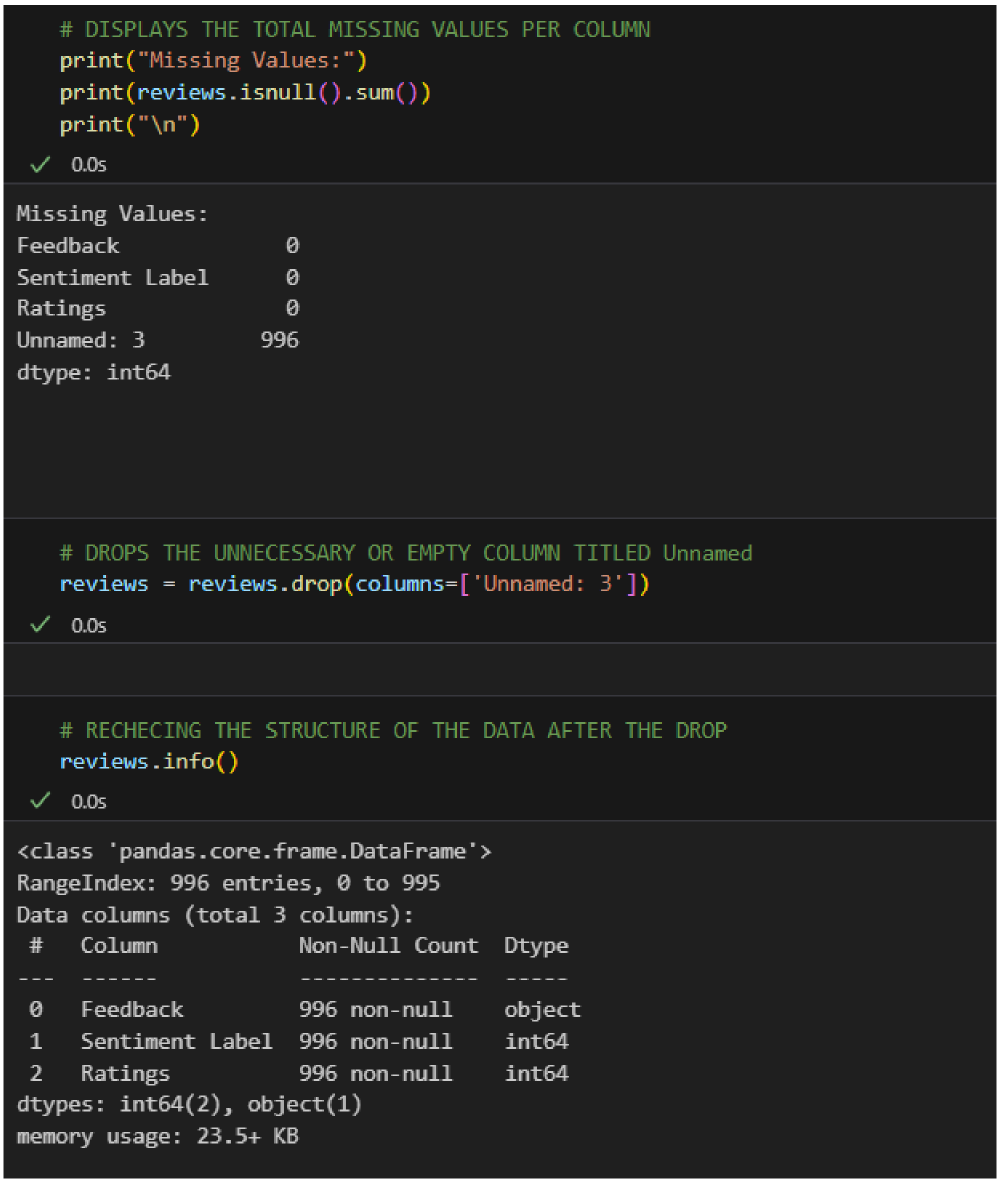


```
# CHECKING THE DATA TYPES AND COUNT OF NON-NULL ENTRIES IN EACH COLUMN IN THE DATASET
reviews.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996 entries, 0 to 995
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Feedback         996 non-null    object
 1   Sentiment Label  996 non-null    int64
 2   Ratings          996 non-null    int64
 3   Unnamed: 3       0 non-null      float64
dtypes: float64(1), int64(2), object(1)
memory usage: 31.3+ KB

# CHECKING TO SEE HOW MANY DIMENSIONS (ROWS AND COLUMNS) WE HAVE IN THE THE DATASET
reviews.shape
✓ 0.0s

(996, 4)
```

Having an idea of how my data is structured and what it consists of allows me to start planning on how to interpret it accordingly. This code allows me to see missing values per column to start planning on how to tackle missing data strategies like dropping some columns.

From this output we can confirm that the dataset has no missing reviews (each row has text), all the rows are labelled (positive (1) or negative (0)), there are no ratings missing and there is a column which is empty which means that every value is missing and there is no useful information to extract.

This column is now deemed as unnecessary and needs to be dropped otherwise the data model is at risk of being larger than necessary therefore leading to slower performance and increased storage requirements. After dropping the column, I run the same **.info()** to confirm that the column has been deleted.



```
# DISPLAYS THE TOTAL MISSING VALUES PER COLUMN
print("Missing Values:")
print(reviews.isnull().sum())
print("\n")
```
✓ 0.0s

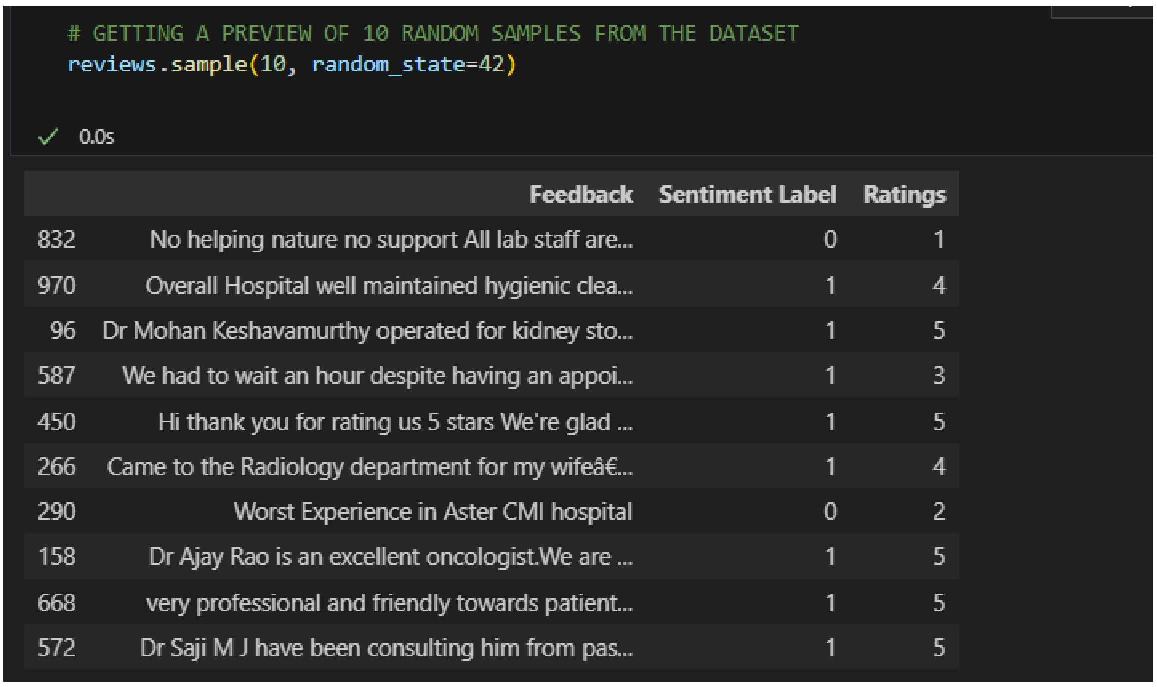
```
Missing Values:
Feedback            0
Sentiment Label     0
Ratings             0
Unnamed: 3        996
dtype: int64
```

```
# DROPS THE UNNECESSARY OR EMPTY COLUMN TITLED Unnamed
reviews = reviews.drop(columns=['Unnamed: 3'])
```
✓ 0.0s

```
# RECHECING THE STRUCTURE OF THE DATA AFTER THE DROP
reviews.info()
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996 entries, 0 to 995
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Feedback         996 non-null    object
 1   Sentiment Label  996 non-null    int64 
 2   Ratings          996 non-null    int64 
dtypes: int64(2), object(1)
memory usage: 23.5+ KB
```

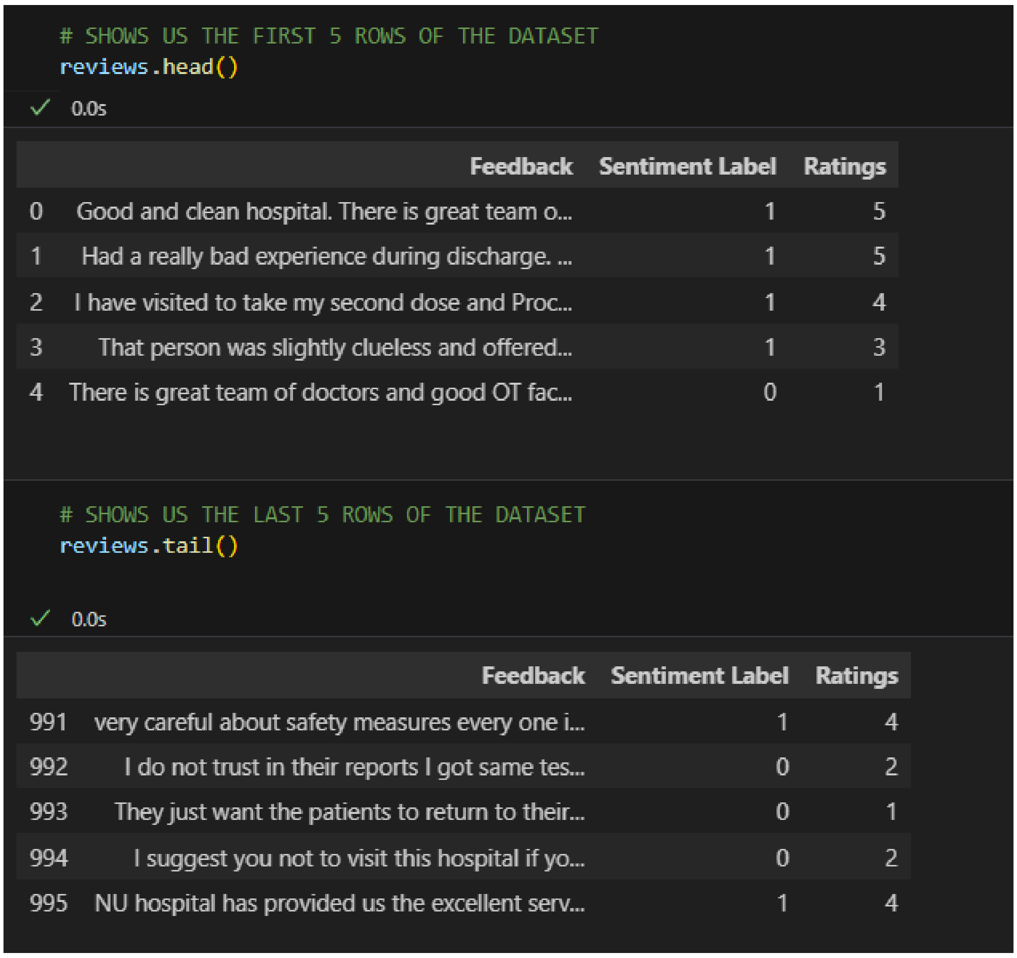
After getting rid of the unnecessary code, I run the code below to manually inspect the different parts of the data. I inspected random rows, the top five rows and the five rows at the bottom of the dataset to see if there are any noticeable duplicates, typos or anomalies that might cause bias for my analysis.



```
# GETTING A PREVIEW OF 10 RANDOM SAMPLES FROM THE DATASET
reviews.sample(10, random_state=42)
```

✓ 0.0s

	Feedback	Sentiment Label	Ratings
832	No helping nature no support All lab staff are...	0	1
970	Overall Hospital well maintained hygienic clea...	1	4
96	Dr Mohan Keshavamurthy operated for kidney sto...	1	5
587	We had to wait an hour despite having an appoi...	1	3
450	Hi thank you for rating us 5 stars We're glad ...	1	5
266	Came to the Radiology department for my wifeâ€...	1	4
290	Worst Experience in Aster CMI hospital	0	2
158	Dr Ajay Rao is an excellent oncologist.We are ...	1	5
668	very professional and friendly towards patient...	1	5
572	Dr Saji M J have been consulting him from pas...	1	5



```
# SHOWS US THE FIRST 5 ROWS OF THE DATASET
reviews.head()
```

✓ 0.0s

	Feedback	Sentiment Label	Ratings
0	Good and clean hospital. There is great team o...	1	5
1	Had a really bad experience during discharge. ...	1	5
2	I have visited to take my second dose and Proc...	1	4
3	That person was slightly clueless and offered...	1	3
4	There is great team of doctors and good OT fac...	0	1

```
# SHOWS US THE LAST 5 ROWS OF THE DATASET
reviews.tail()
```

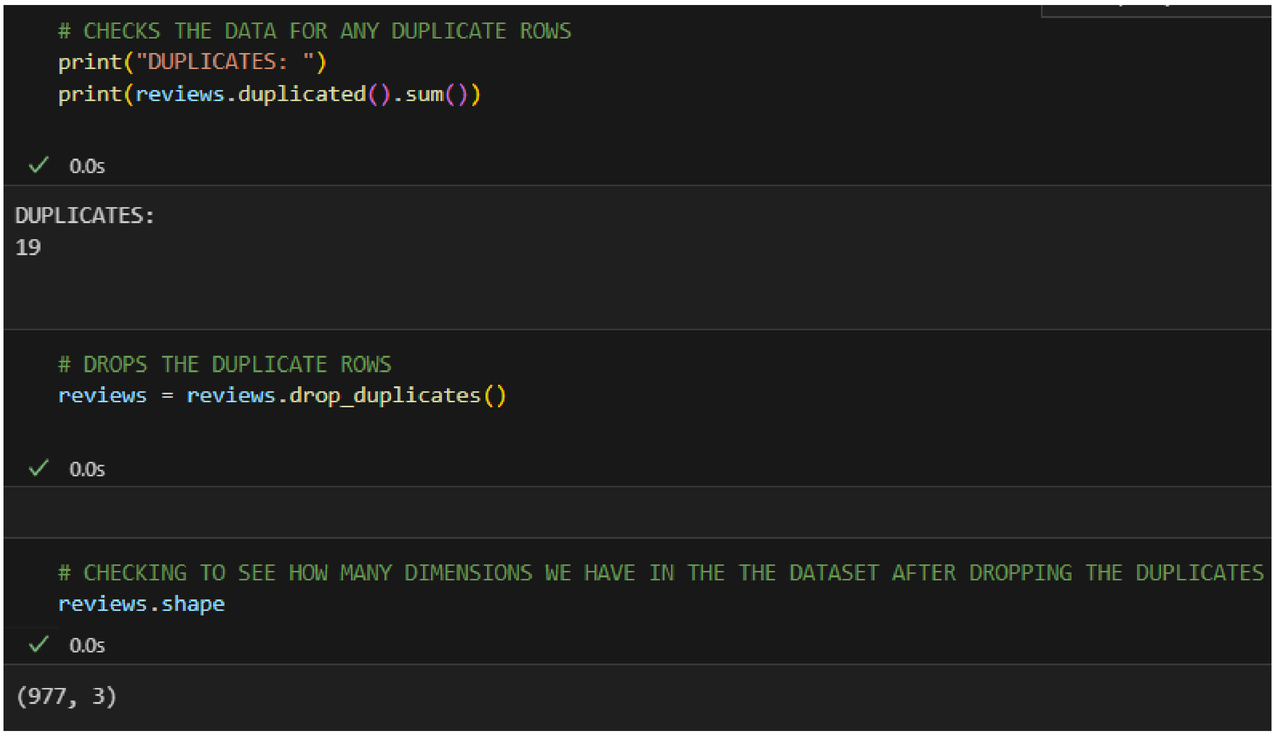
✓ 0.0s

	Feedback	Sentiment Label	Ratings
991	very careful about safety measures every one i...	1	4
992	I do not trust in their reports I got same tes...	0	2
993	They just want the patients to return to their...	0	1
994	I suggest you not to visit this hospital if yo...	0	2
995	NU hospital has provided us the excellent serv...	1	4

The code below checks and tells me how many rows in my dataset are exact copies of one another (duplicates) with the output of nineteen. This means that my dataset has nineteen rows that were duplicates of other issues. Not addressing this could cause bias in my model (overemphasis of one client's opinion by repetition).

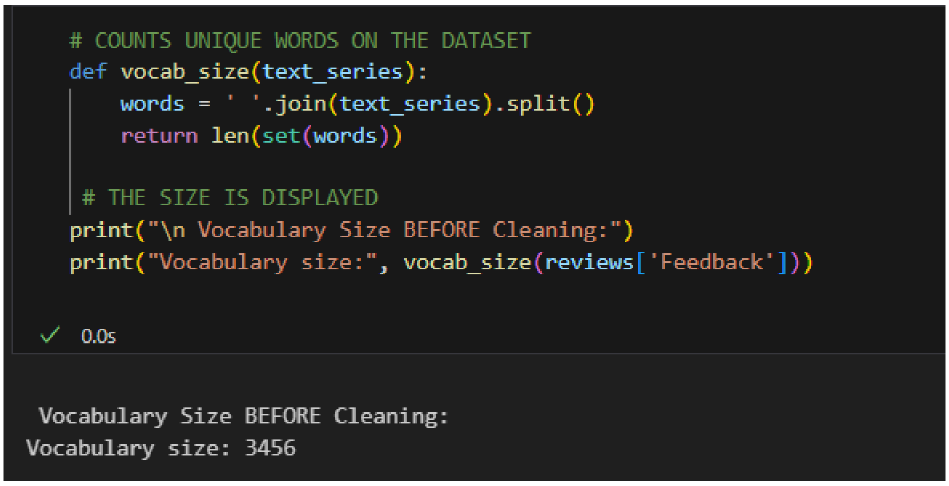
I dropped the nineteen duplicates from my dataset keeping only one of each to ensure each review contributes equally. This is to prevent overfitting as well as making each row or review unique.

I rechecked the shape of the data once again after dropping those rows. The count of rows in the dataset did get reduced from 996 to 977.



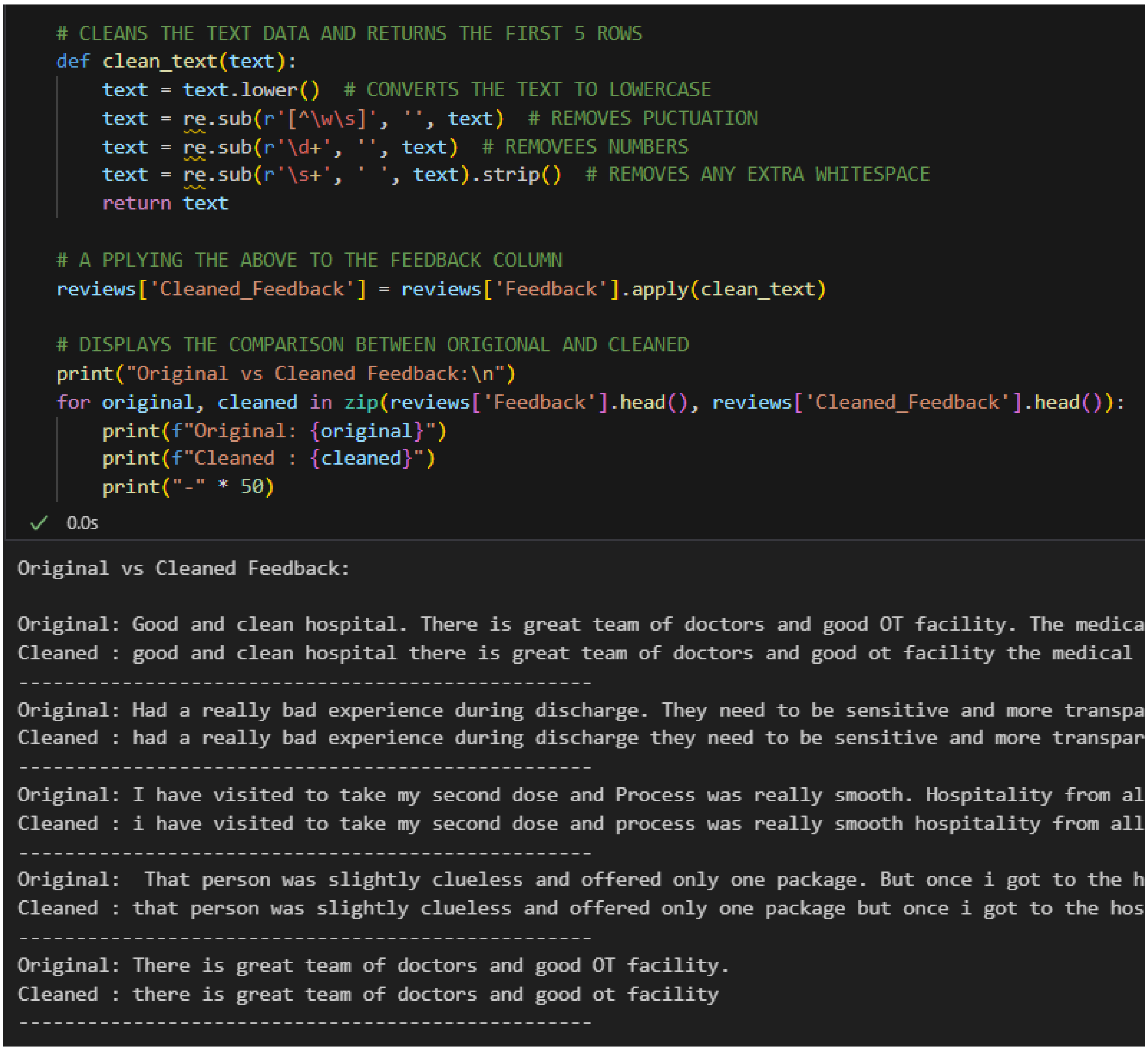
```
# CHECKS THE DATA FOR ANY DUPLICATE ROWS
print("DUPLICATES: ")
print(reviews.duplicated().sum())
```
```
✓ 0.0s
```
```
DUPLICATES:
19
```
```
# DROPS THE DUPLICATE ROWS
reviews = reviews.drop_duplicates()
```
```
✓ 0.0s
```
```
# CHECKING TO SEE HOW MANY DIMENSIONS WE HAVE IN THE THE DATASET AFTER DROPPING THE DUPLICATES
reviews.shape
```
```
✓ 0.0s
```
```
(977, 3)
```

This code checks the vocabulary size before cleaning it. The reviews are joined into one long string, then they are split into individual words and then the number of unique words are returned (vocab size).



```
# COUNTS UNIQUE WORDS ON THE DATASET
def vocab_size(text_series):
    words = ' '.join(text_series).split()
    return len(set(words))

# THE SIZE IS DISPLAYED
print("\n Vocabulary Size BEFORE Cleaning:")
print("Vocabulary size:", vocab_size(reviews['Feedback']))
```
```
✓ 0.0s
```
```

 Vocabulary Size BEFORE Cleaning:
Vocabulary size: 3456
```

The code below cleans the input text by converting those in uppercase to lowercase for case normalisation to ensure that the “Good” and “good” are seen as the same, removes punctuation to not confuse the models, removing any numbers because numbers add noise in text reviews unless they are analysed separately, removing extra whitespaces to clean the structure for word counting and then returning the cleaned text. This is done to reduce the vocabulary size, improve consistency and enhance the accuracy of our model. The cleaner the text the stronger the predictions

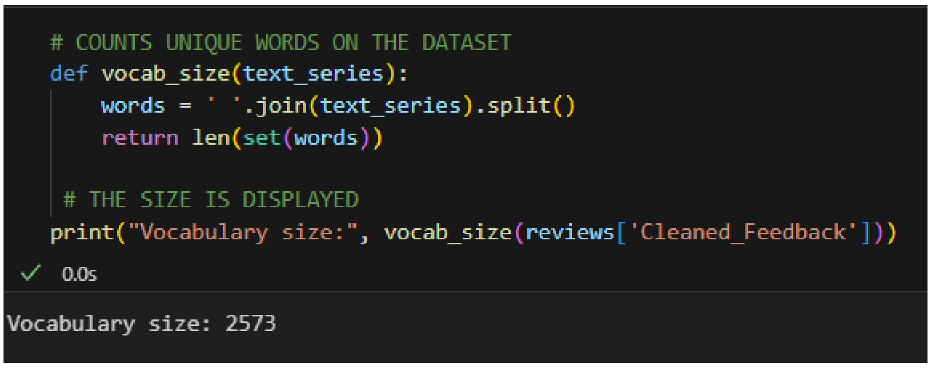


```python
# CLEANS THE TEXT DATA AND RETURNS THE FIRST 5 ROWS
def clean_text(text):
    text = text.lower()  # CONVERTS THE TEXT TO LOWERCASE
    text = re.sub(r'[^\w\s]', '', text)  # REMOVES PUCTUATION
    text = re.sub(r'\d+', '', text)  # REMOVEES NUMBERS
    text = re.sub(r'\s+', ' ', text).strip()  # REMOVES ANY EXTRA WHITESPACE
    return text

# A PPLYING THE ABOVE TO THE FEEDBACK COLUMN
reviews['Cleaned_Feedback'] = reviews['Feedback'].apply(clean_text)

# DISPLAYS THE COMPARISON BETWEEN ORIGIONAL AND CLEANED
print("Original vs Cleaned Feedback:\n")
for original, cleaned in zip(reviews['Feedback'].head(), reviews['Cleaned_Feedback'].head()):
    print(f"Original: {original}")
    print(f"Cleaned : {cleaned}")
    print("-" * 50)
```

✓ 0.0s

```
Original vs Cleaned Feedback:

Original: Good and clean hospital. There is great team of doctors and good OT facility. The medica
Cleaned : good and clean hospital there is great team of doctors and good ot facility the medical
--------------------------------------------------
Original: Had a really bad experience during discharge. They need to be sensitive and more transpa
Cleaned : had a really bad experience during discharge they need to be sensitive and more transpar
--------------------------------------------------
Original: I have visited to take my second dose and Process was really smooth. Hospitality from al
Cleaned : i have visited to take my second dose and process was really smooth hospitality from all
--------------------------------------------------
Original:  That person was slightly clueless and offered only one package. But once i got to the h
Cleaned : that person was slightly clueless and offered only one package but once i got to the hos
--------------------------------------------------
Original: There is great team of doctors and good OT facility.
Cleaned : there is great team of doctors and good ot facility
--------------------------------------------------
```

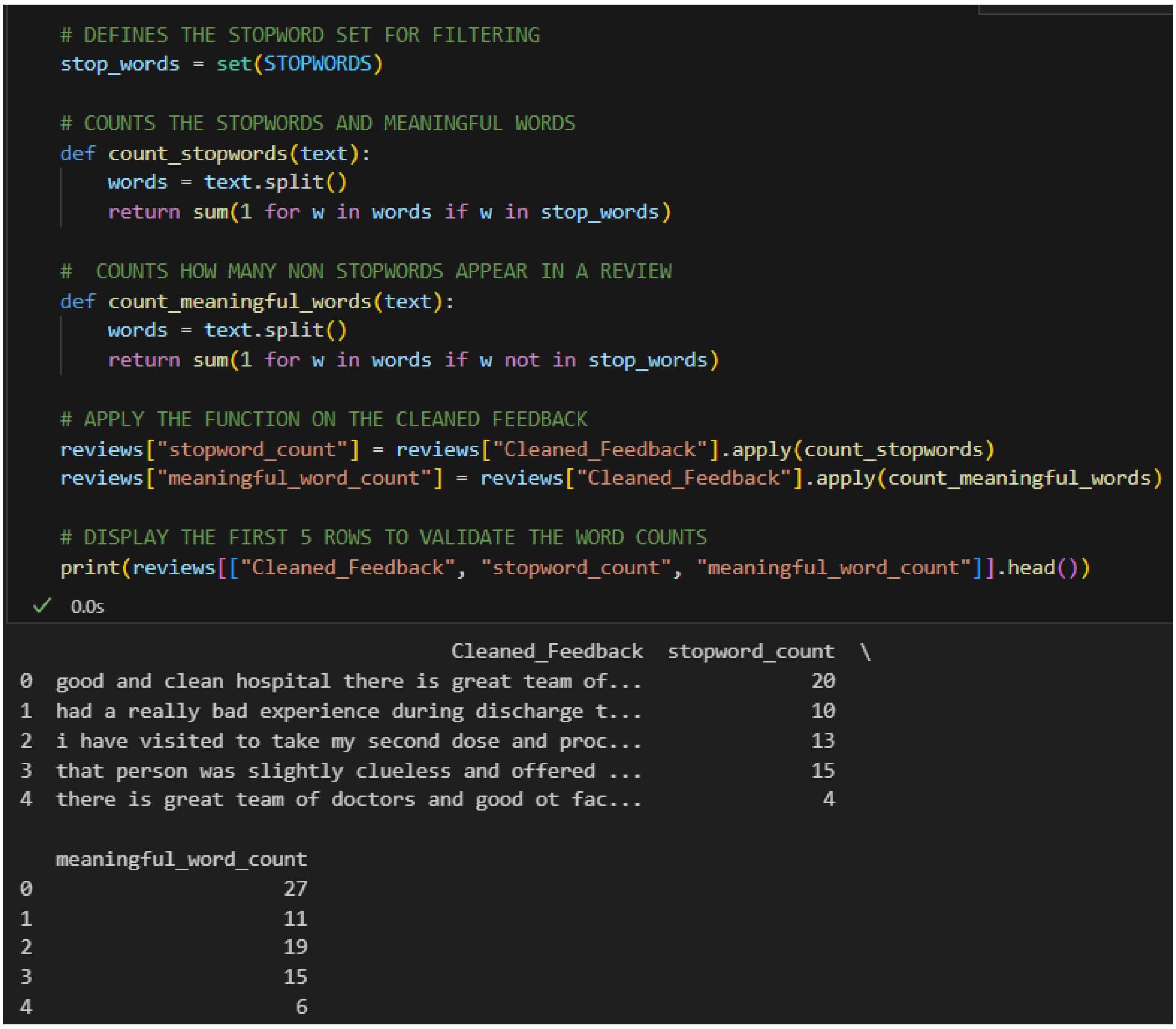
Prior to the cleaning, the size was 3456. After applying the cleaning steps above the size has dropped which tells us how much noise was removed after cleaning. From the filtering above this reduction is expected and good because it lowers the dimensionality and improves model generalisation. Returning a vocab size of 2573 tells us that the preprocessing steps were not aggressive as the size is still fine with more than 2500 unique words. A foundation is built where my model can learn from normalised text.



```
# COUNTS UNIQUE WORDS ON THE DATASET
def vocab_size(text_series):
    words = ' '.join(text_series).split()
    return len(set(words))

 # THE SIZE IS DISPLAYED
print("Vocabulary size:", vocab_size(reviews['Cleaned_Feedback']))
✓ 0.0s

Vocabulary size: 2573
```

The code below defines a set of common words like “and” etc using the built-in list STOPWORDS, splits a review into words and counts how many of those words are stopwords and non stopwords (words with meaning). Once that is ran or applied there are two new columns that are added into the dataset: stopword_count and meaningful_word_count.

This allows for the detection of reviews that can confuse the model (stuffed with filler words or overly short ones). Reviews with higher meaningful words may be more informative as opposed to those with too many stopwords which may be neutral or less useless.

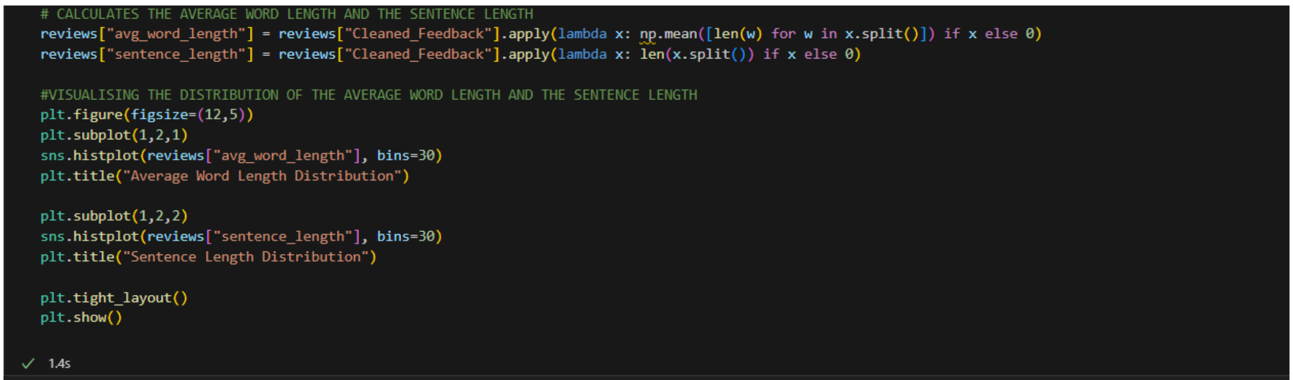


```
# DEFINES THE STOPWORD SET FOR FILTERING
stop_words = set(STOPWORDS)

# COUNTS THE STOPWORDS AND MEANINGFUL WORDS
def count_stopwords(text):
    words = text.split()
    return sum(1 for w in words if w in stop_words)

#  COUNTS HOW MANY NON STOPWORDS APPEAR IN A REVIEW
def count_meaningful_words(text):
    words = text.split()
    return sum(1 for w in words if w not in stop_words)

# APPLY THE FUNCTION ON THE CLEANED FEEDBACK
reviews["stopword_count"] = reviews["Cleaned_Feedback"].apply(count_stopwords)
reviews["meaningful_word_count"] = reviews["Cleaned_Feedback"].apply(count_meaningful_words)

# DISPLAY THE FIRST 5 ROWS TO VALIDATE THE WORD COUNTS
print(reviews[["Cleaned_Feedback", "stopword_count", "meaningful_word_count"]].head())
✓ 0.0s

                                    Cleaned_Feedback  stopword_count  \
0  good and clean hospital there is great team of...              20
1  had a really bad experience during discharge t...              10
2  i have visited to take my second dose and proc...              13
3  that person was slightly clueless and offered ...              15
4  there is great team of doctors and good ot fac...               4

   meaningful_word_count
0                     27
1                     11
2                     19
3                     15
4                      6
```

After cleaning my data I was comfortable enough to interpret and analyse it visually. Below is a histogram. For each review, this code splits the cleaned text into words and then computes the average number of characters per word, counts how many words there are in each review that is clean and then it creates two histograms: one for average word lengths and one for sentence (review) lengths.

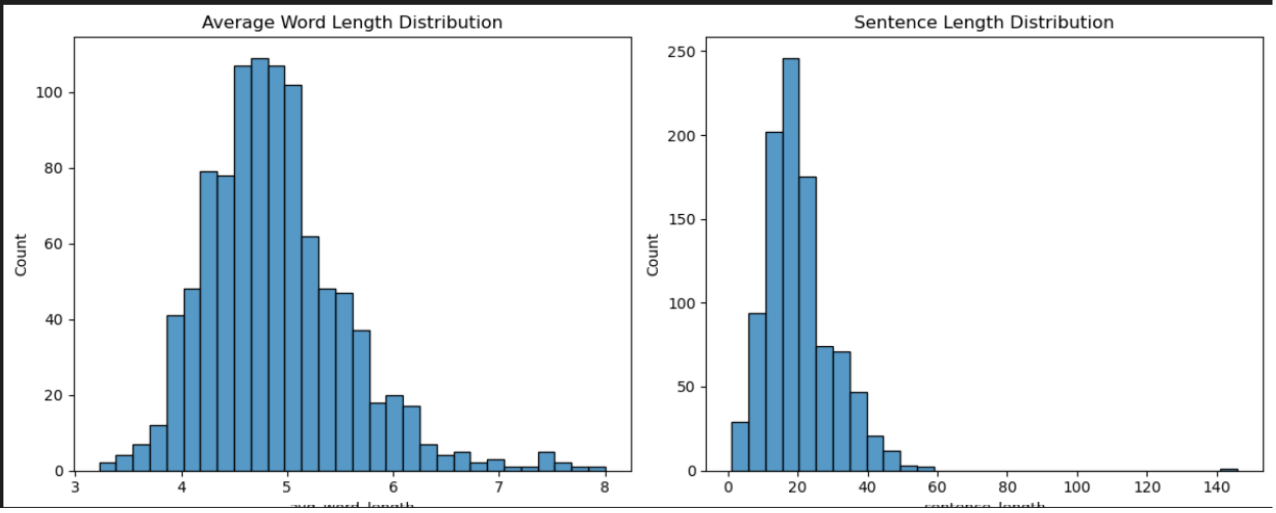
Avg_word_length helps the model distinguish between casual and formal tone whilst sentence_length is longer reviews which might carry more emotion and sentiment.

Shorter words may indicate simple language, might contain more context whilst longer words might indicate a more formal language, might be less helpful. Both of these graphs depict a positive or right-skewed graph. With regards to Average Word Length, this graph interprets that most words are short, and Sentence Length Distribution indicates that there may be detailed or emotional reviews.



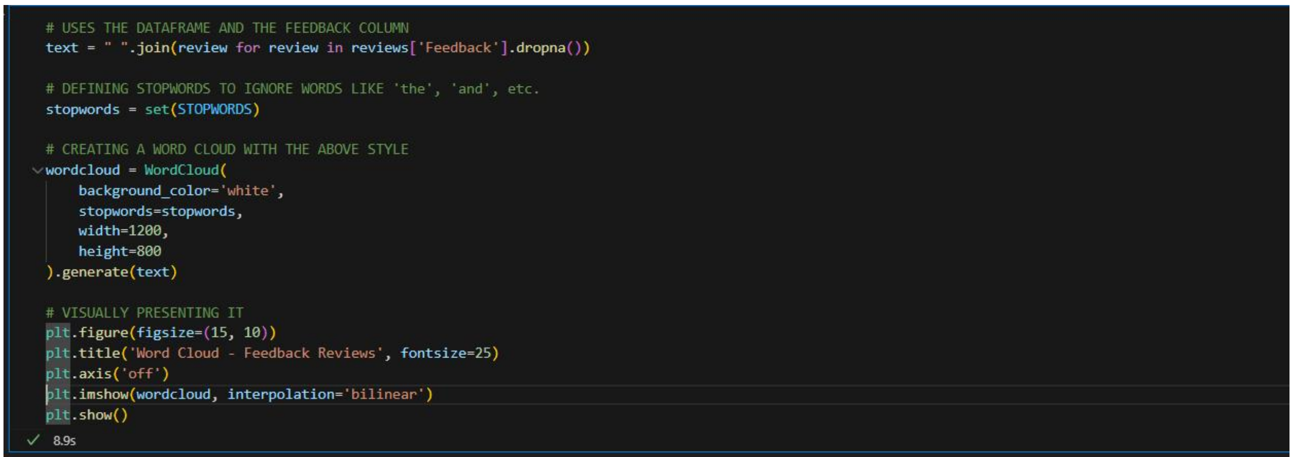
```
# CALCULATES THE AVERAGE WORD LENGTH AND THE SENTENCE LENGTH
reviews["avg_word_length"] = reviews["Cleaned_Feedback"].apply(lambda x: np.mean([len(w) for w in x.split()]) if x else 0)
reviews["sentence_length"] = reviews["Cleaned_Feedback"].apply(lambda x: len(x.split()) if x else 0)

#VISUALISING THE DISTRIBUTION OF THE AVERAGE WORD LENGTH AND THE SENTENCE LENGTH
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(reviews["avg_word_length"], bins=30)
plt.title("Average Word Length Distribution")

plt.subplot(1,2,2)
sns.histplot(reviews["sentence_length"], bins=30)
plt.title("Sentence Length Distribution")

plt.tight_layout()
plt.show()
```

✓ 1.4s

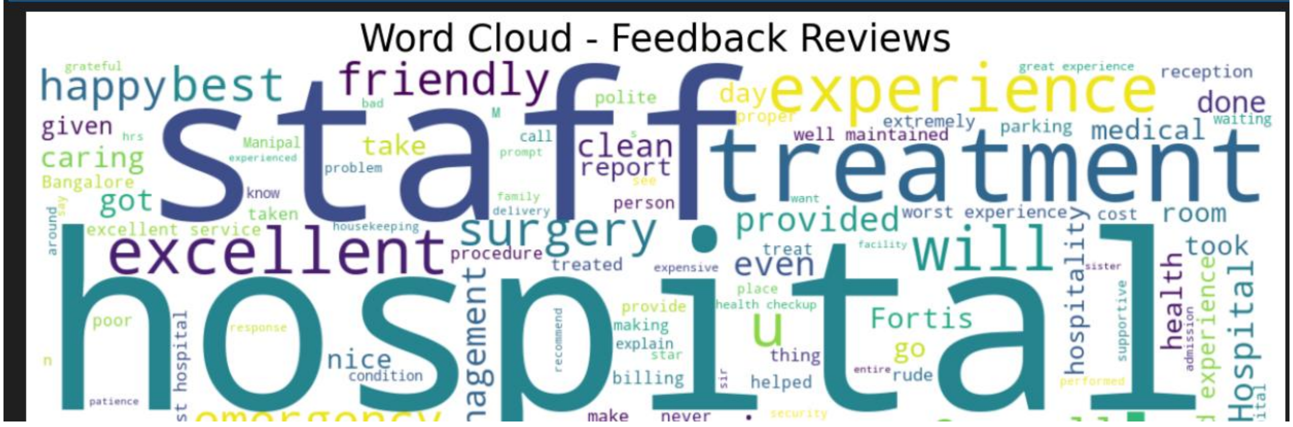


The code below visualises the words that appear the most across all the feedback. It joins the reviews into a single string, creates a word cloud to visualise the most frequent words out of all of them and then filters out the common words which have no meaning.

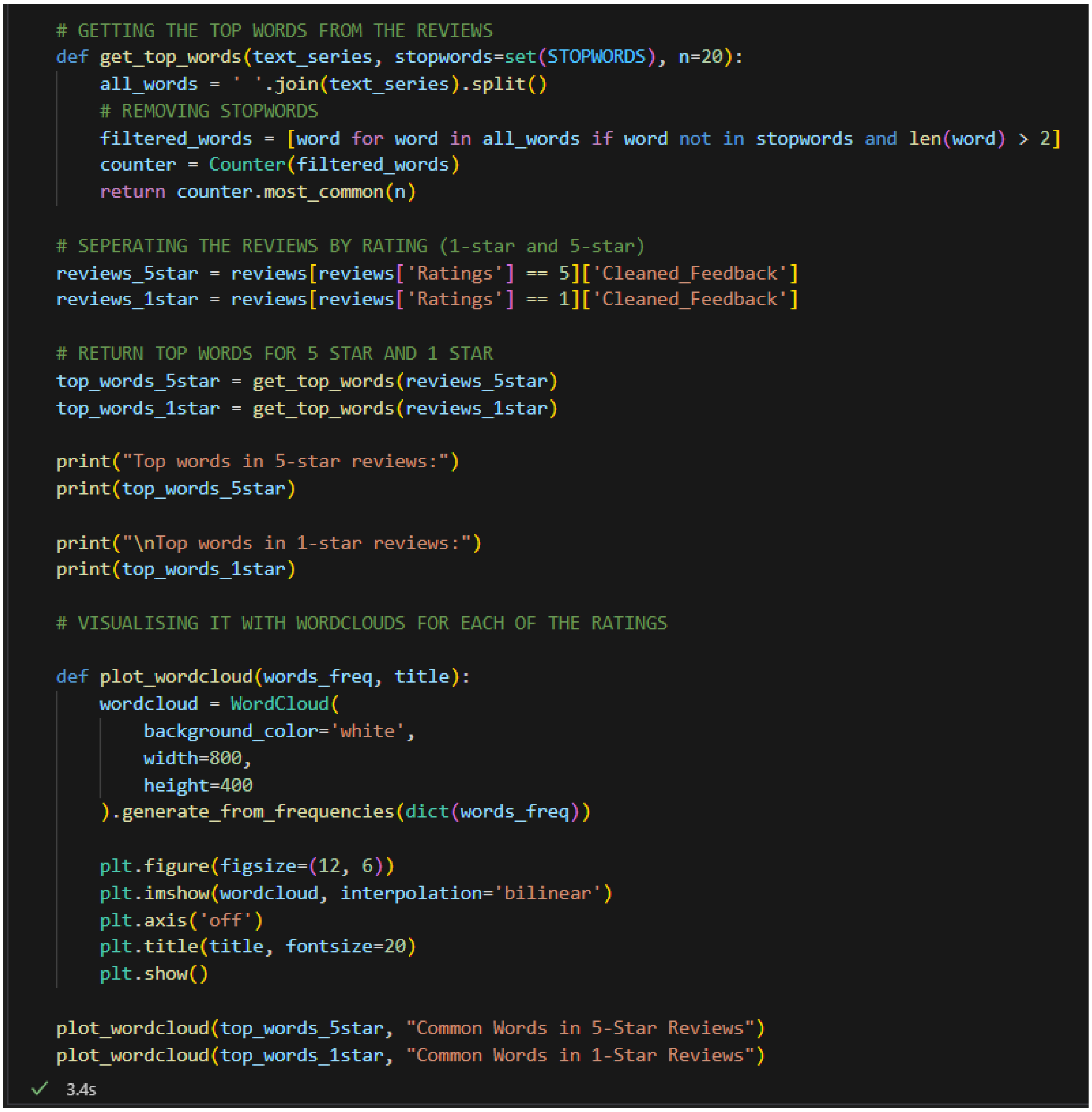
The large words means that they are most frequent as opposed to the small words meaning they are not frequently used. This aids in identifying the positive or negative words that are commonly used for the indication of sentiment trends.



```
# USES THE DATAFRAME AND THE FEEDBACK COLUMN
text = " ".join(review for review in reviews['Feedback'].dropna())

# DEFINING STOPWORDS TO IGNORE WORDS LIKE 'the', 'and', etc.
stopwords = set(STOPWORDS)

# CREATING A WORD CLOUD WITH THE ABOVE STYLE
wordcloud = WordCloud(
    background_color='white',
    stopwords=stopwords,
    width=1200,
    height=800
).generate(text)

# VISUALLY PRESENTING IT
plt.figure(figsize=(15, 10))
plt.title('Word Cloud - Feedback Reviews', fontsize=25)
plt.axis('off')
plt.imshow(wordcloud, interpolation='bilinear')
plt.show()
```
✓ 8.9s



This section compares commonly used words in the five star and 1-star reviews to uncover patterns. This is done by removing common stopwords like "the," etc, because they typically have little meaning. The remaining words are counted and then the top twenty most frequent are extracted. The dataset is then filtered to separate reviews according to ratings of five star and one star. This is applied to the 5-star group and one star group to see which words are most frequent when associated with satisfaction as opposed to dissatisfaction. Finally the frequent words are visualised to make it easy to grasp a theme in both reviews groups to help identify patterns.

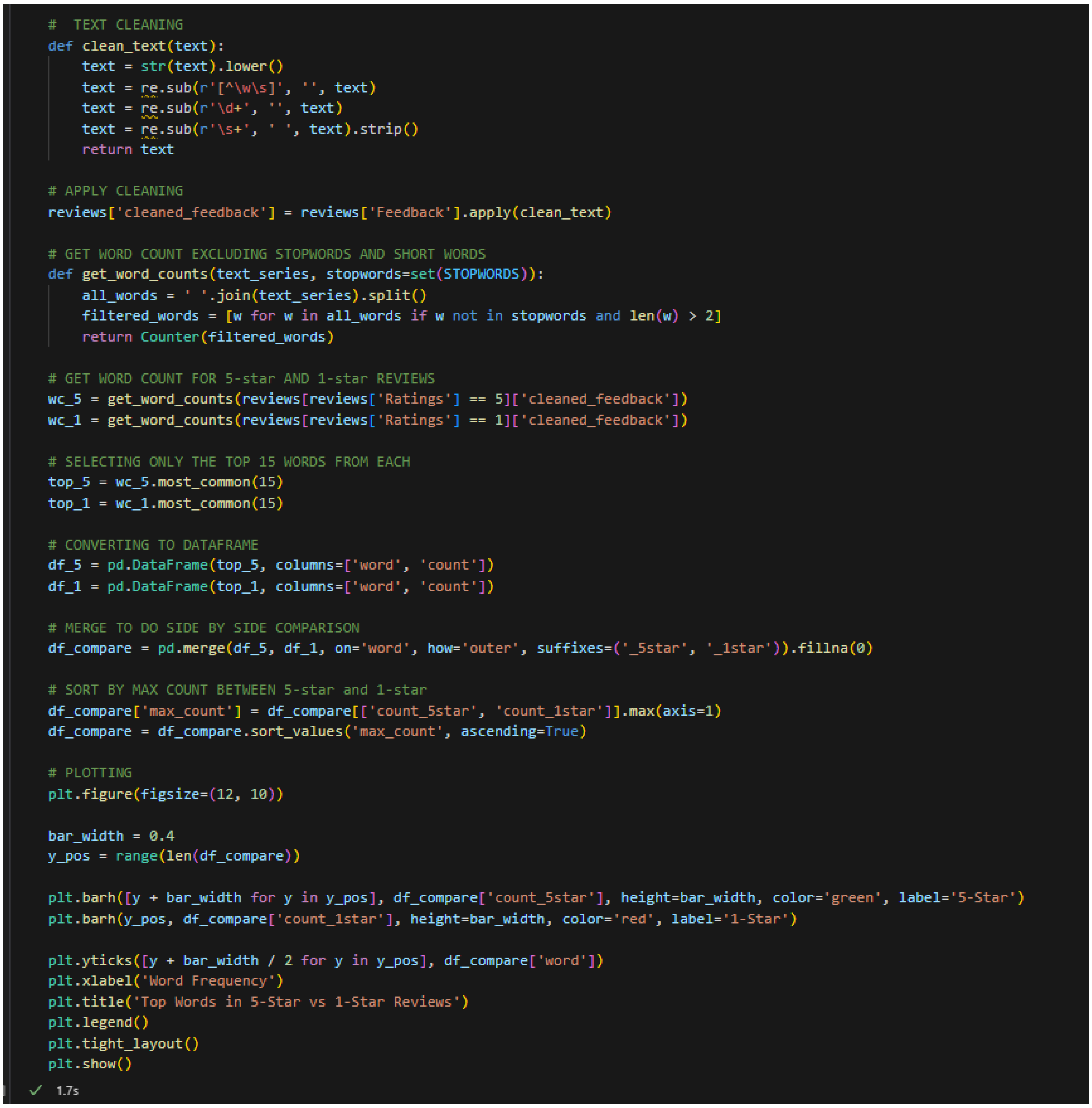


```python
# GETTING THE TOP WORDS FROM THE REVIEWS
def get_top_words(text_series, stopwords=set(STOPWORDS), n=20):
    all_words = ' '.join(text_series).split()
    # REMOVING STOPWORDS
    filtered_words = [word for word in all_words if word not in stopwords and len(word) > 2]
    counter = Counter(filtered_words)
    return counter.most_common(n)

# SEPERATING THE REVIEWS BY RATING (1-star and 5-star)
reviews_5star = reviews[reviews['Ratings'] == 5]['Cleaned_Feedback']
reviews_1star = reviews[reviews['Ratings'] == 1]['Cleaned_Feedback']

# RETURN TOP WORDS FOR 5 STAR AND 1 STAR
top_words_5star = get_top_words(reviews_5star)
top_words_1star = get_top_words(reviews_1star)

print("Top words in 5-star reviews:")
print(top_words_5star)

print("\nTop words in 1-star reviews:")
print(top_words_1star)

# VISUALISING IT WITH WORDCLOUDS FOR EACH OF THE RATINGS

def plot_wordcloud(words_freq, title):
    wordcloud = WordCloud(
        background_color='white',
        width=800,
        height=400
    ).generate_from_frequencies(dict(words_freq))

    plt.figure(figsize=(12, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title, fontsize=20)
    plt.show()

plot_wordcloud(top_words_5star, "Common Words in 5-Star Reviews")
plot_wordcloud(top_words_1star, "Common Words in 1-Star Reviews")
```

✓ 3.4s

The code below is text cleaning and word frequency analysis. I use it to compare the most common words in 5-star vs 1-star reviews from the database. It is a visualisation of the overlapping and divergence of word usage between positive and negative reviews aiding in the depiction of words that are sentiment specific.

This is done by standardising the text, removing digits, whitespaces and punctuations. The feedback is cleaned and stored in a new column. All the cleaned reviews are combined into a string and then split into a list of words. Common words are filtered, and short words are removed (under assumption that they are less meaningful). The dataset is then filtered by ratings and then the frequency of the words are counted separately. The top fifteen most frequent words are then used for each star rating group.



```python
#  TEXT CLEANING
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# APPLY CLEANING
reviews['cleaned_feedback'] = reviews['Feedback'].apply(clean_text)

# GET WORD COUNT EXCLUDING STOPWORDS AND SHORT WORDS
def get_word_counts(text_series, stopwords=set(STOPWORDS)):
    all_words = ' '.join(text_series).split()
    filtered_words = [w for w in all_words if w not in stopwords and len(w) > 2]
    return Counter(filtered_words)

# GET WORD COUNT FOR 5-star AND 1-star REVIEWS
wc_5 = get_word_counts(reviews[reviews['Ratings'] == 5]['cleaned_feedback'])
wc_1 = get_word_counts(reviews[reviews['Ratings'] == 1]['cleaned_feedback'])

# SELECTING ONLY THE TOP 15 WORDS FROM EACH
top_5 = wc_5.most_common(15)
top_1 = wc_1.most_common(15)

# CONVERTING TO DATAFRAME
df_5 = pd.DataFrame(top_5, columns=['word', 'count'])
df_1 = pd.DataFrame(top_1, columns=['word', 'count'])

# MERGE TO DO SIDE BY SIDE COMPARISON
df_compare = pd.merge(df_5, df_1, on='word', how='outer', suffixes=('_5star', '_1star')).fillna(0)

# SORT BY MAX COUNT BETWEEN 5-star and 1-star
df_compare['max_count'] = df_compare[['count_5star', 'count_1star']].max(axis=1)
df_compare = df_compare.sort_values('max_count', ascending=True)

# PLOTTING
plt.figure(figsize=(12, 10))

bar_width = 0.4
y_pos = range(len(df_compare))

plt.barh([y + bar_width for y in y_pos], df_compare['count_5star'], height=bar_width, color='green', label='5-Star')
plt.barh(y_pos, df_compare['count_1star'], height=bar_width, color='red', label='1-Star')

plt.yticks([y + bar_width / 2 for y in y_pos], df_compare['word'])
plt.xlabel('Word Frequency')
plt.title('Top Words in 5-Star vs 1-Star Reviews')
plt.legend()
plt.tight_layout()
plt.show()
```

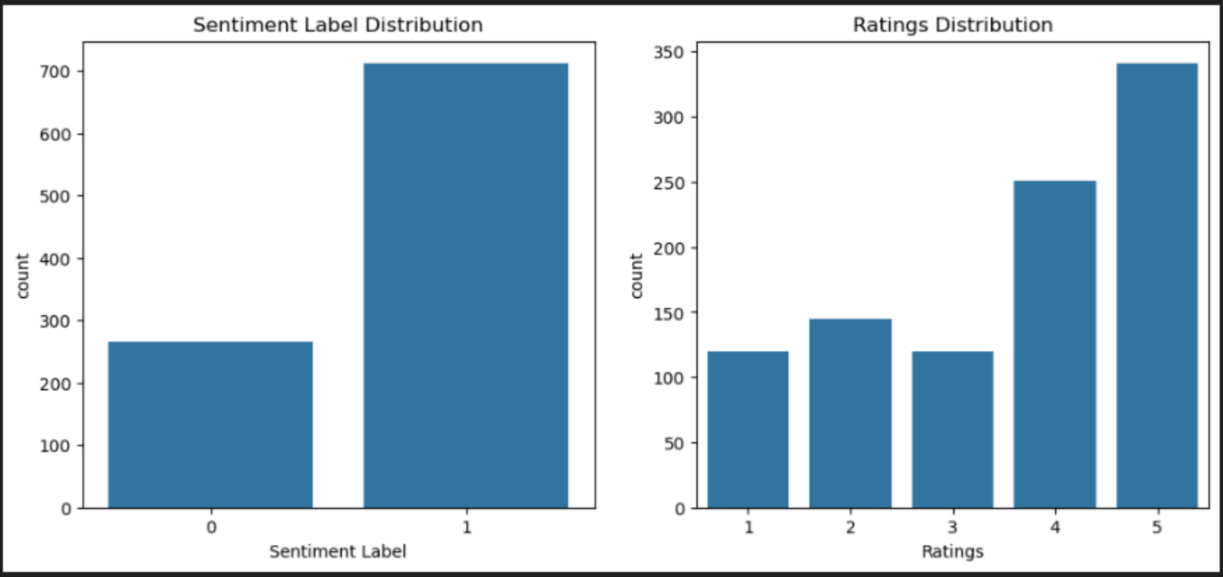
The block below allows us to visually analyse the distribution of the target variable which is Sentiment Label and the feature Ratings using bar plots. This aids in deciding if I will be sampling up or down when we are training the model. Not checking class imbalances can affect the model's performance and its biasing accuracy towards the majority.

From the Sentiment Label graph there is an imbalance. Sentiment Label has 70% positive and 30%, this means that the model might learn to predict positive sentiments all the time and still appear accurate. This creates false confidence.

From the Ratings Distribution most of the ratings are between 4 and 5. This means that the dataset the dataset leans toward positivity because there is a correlation between ratings and sentiment label therefore making ratings a useful feature (1 star and three stars are underrepresented).

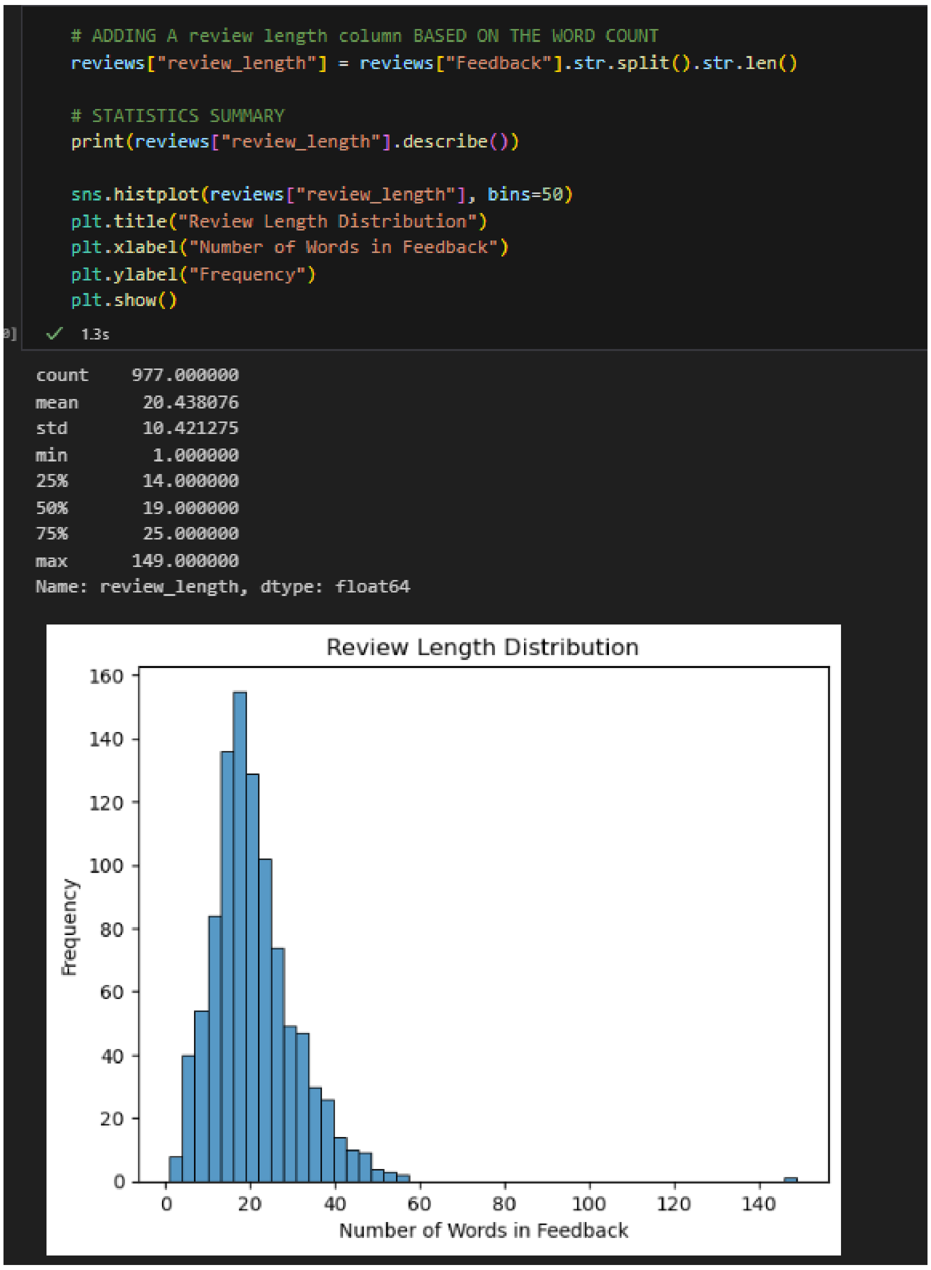
```
# CHECKS CLASS DISTRIBUTION IN Sentiment Label AND Ratings
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.countplot(x="Sentiment Label", data=reviews)
plt.title("Sentiment Label Distribution")

plt.subplot(1,2,2)
sns.countplot(x="Ratings", data=reviews)
plt.title("Ratings Distribution")

plt.show()
```

```
✓ 0.8s
```



The code below calculates the length of each review (text feature engineering) and then visualises the distribution of those lengths. It checks to see if the longer review contains stronger emotions or not. This is displayed using a histogram.

Each review is split into a list of words and then it counts the number of words in that list storing it into a new column named review_length. With this, it becomes easier to identify if the longer reviews are negative or positive, whether the review lengths correlate with sentiment and if there are any outliers.

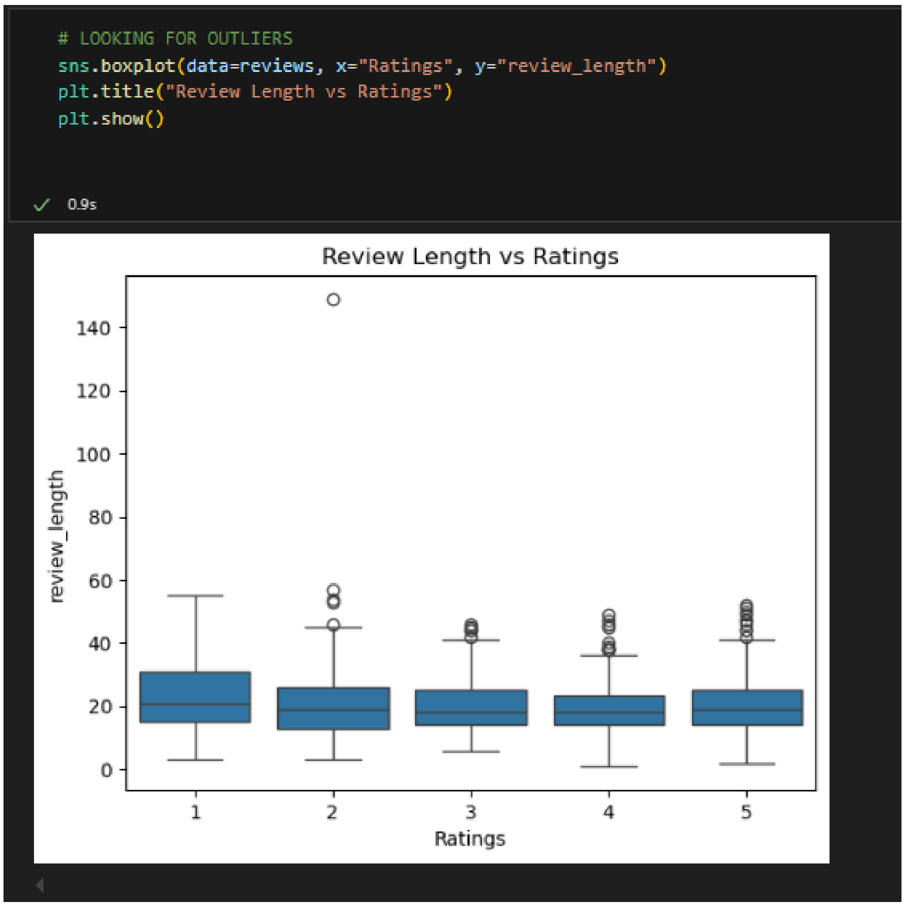


```
# ADDING A review length column BASED ON THE WORD COUNT
reviews["review_length"] = reviews["Feedback"].str.split().str.len()

# STATISTICS SUMMARY
print(reviews["review_length"].describe())

sns.histplot(reviews["review_length"], bins=50)
plt.title("Review Length Distribution")
plt.xlabel("Number of Words in Feedback")
plt.ylabel("Frequency")
plt.show()
```

```
count    977.000000
mean      20.438076
std       10.421275
min        1.000000
25%       14.000000
50%       19.000000
75%       25.000000
max      149.000000
Name: review_length, dtype: float64
```

Below I am creating a boxplot which is known for aiding in identifying outliers and understanding the distribution of data. This graph is how review lengths vary across the different rating levels, and the box plot is used to visualise it. This will allow me to see if the people tend to write longer reviews for low or high ratings or if its just unexpectedly long or short reviews.

From the graph I see that there are nit a lot of points floating away from my box plots which means that there are not a lot of outliers in this dataset. One star has a longer tail, which means that people often have a lot to say when they hate the services from the hospital provided by the medical aid.



```
# LOOKING FOR OUTLIERS
sns.boxplot(data=reviews, x="Ratings", y="review_length")
plt.title("Review Length vs Ratings")
plt.show()
```

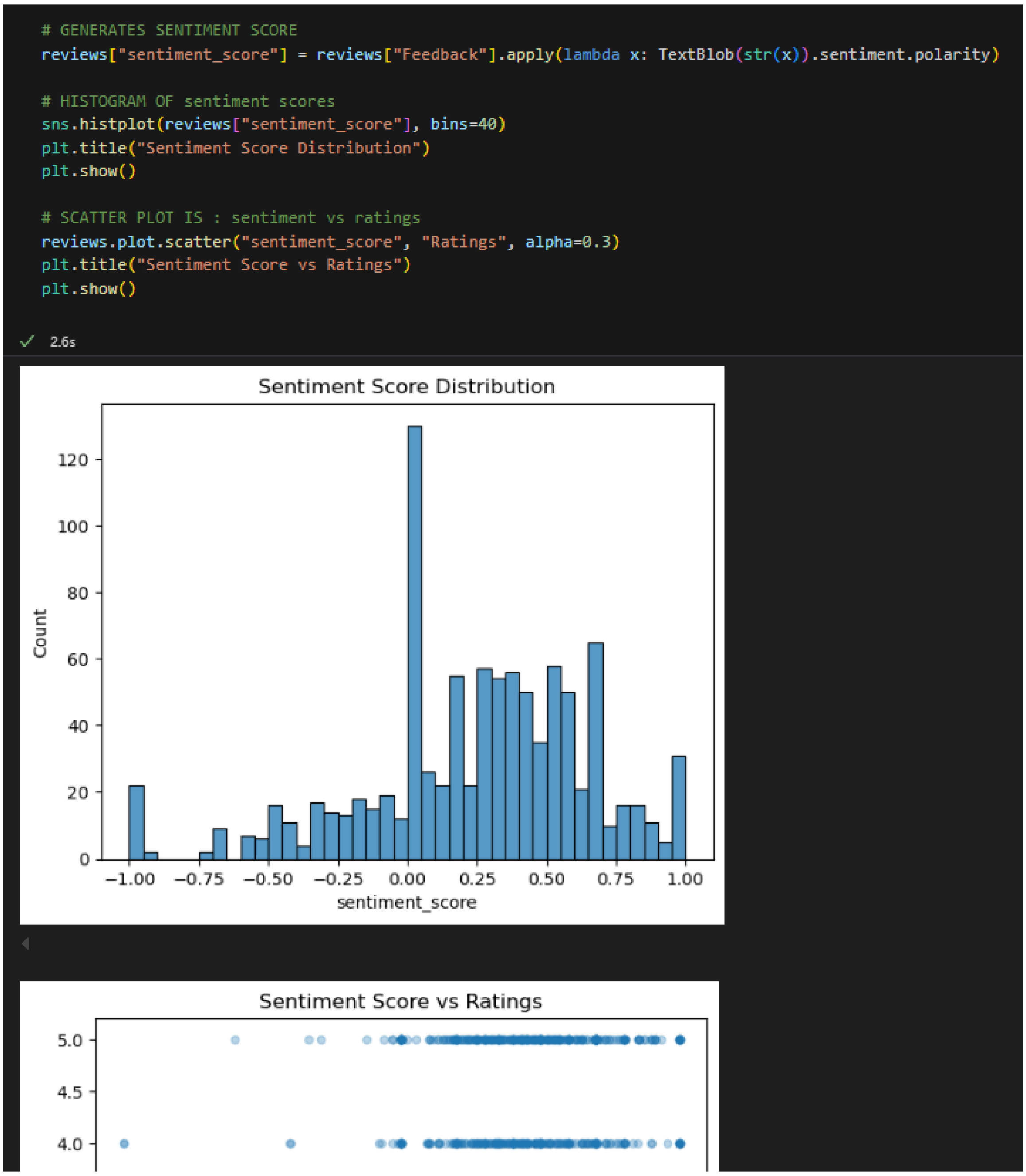
✓ 0.9s

With the code block below, we can get an understanding of how people feel in their feedback and how that correlates with the ratings that they provided. Extracting the polarity from positive to negative of each review. This allows us to set a baseline sentiment score to compare against the rating.

This is done by taking text from the Feedback column and converting it into a Textblob object so that it can be analysed. This then computes a sentiment score ranging from -1 and one, -1 being very negative, zero being neutral, and +1 being positive. The results are then stored in a new column named sentiment_score. A histogram is then created to show how sentiment scores are distributed across all the reviews.

From the graph majority of them are neutral.

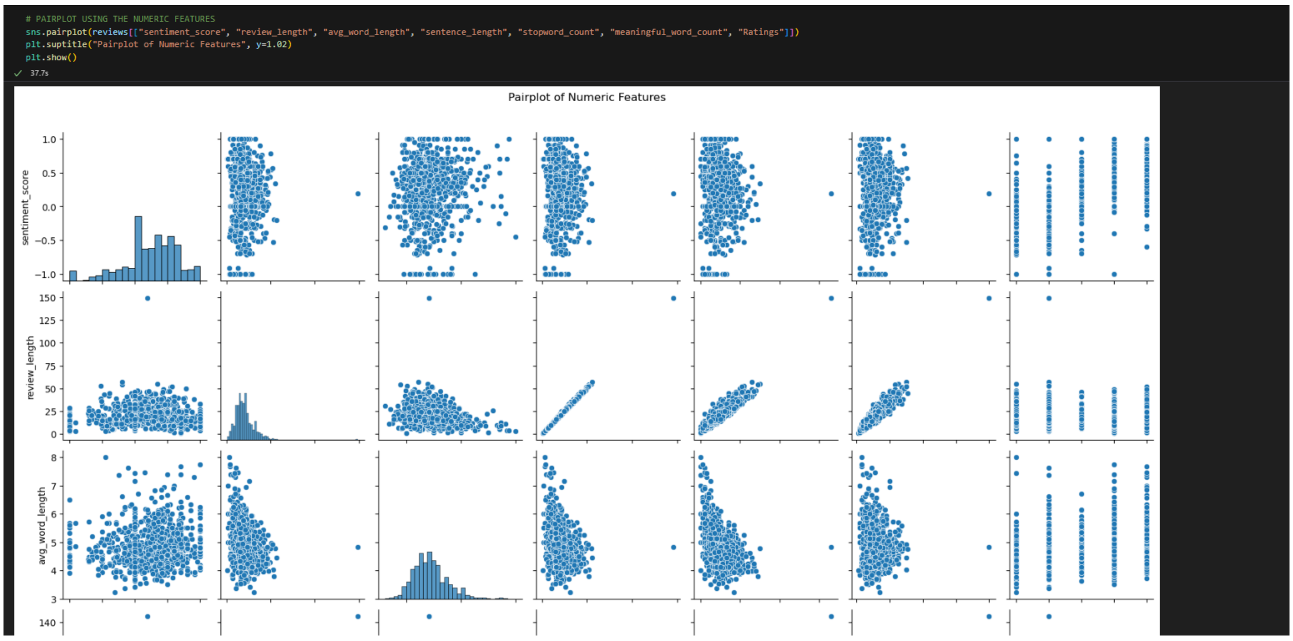
.



```
# GENERATES SENTIMENT SCORE
reviews["sentiment_score"] = reviews["Feedback"].apply(lambda x: TextBlob(str(x)).sentiment.polarity)

# HISTOGRAM OF sentiment scores
sns.histplot(reviews["sentiment_score"], bins=40)
plt.title("Sentiment Score Distribution")
plt.show()

# SCATTER PLOT IS : sentiment vs ratings
reviews.plot.scatter("sentiment_score", "Ratings", alpha=0.3)
plt.title("Sentiment Score vs Ratings")
plt.show()
```

✓ 2.6s

The code below takes all the numeric columns and plots a scatterplot to show the distribution of each individual variable. This grid of plots is all the numeric columns where the x axis and y axis change per cell to show all the variable combinations so that I can spot correlations, outliers and clusters.

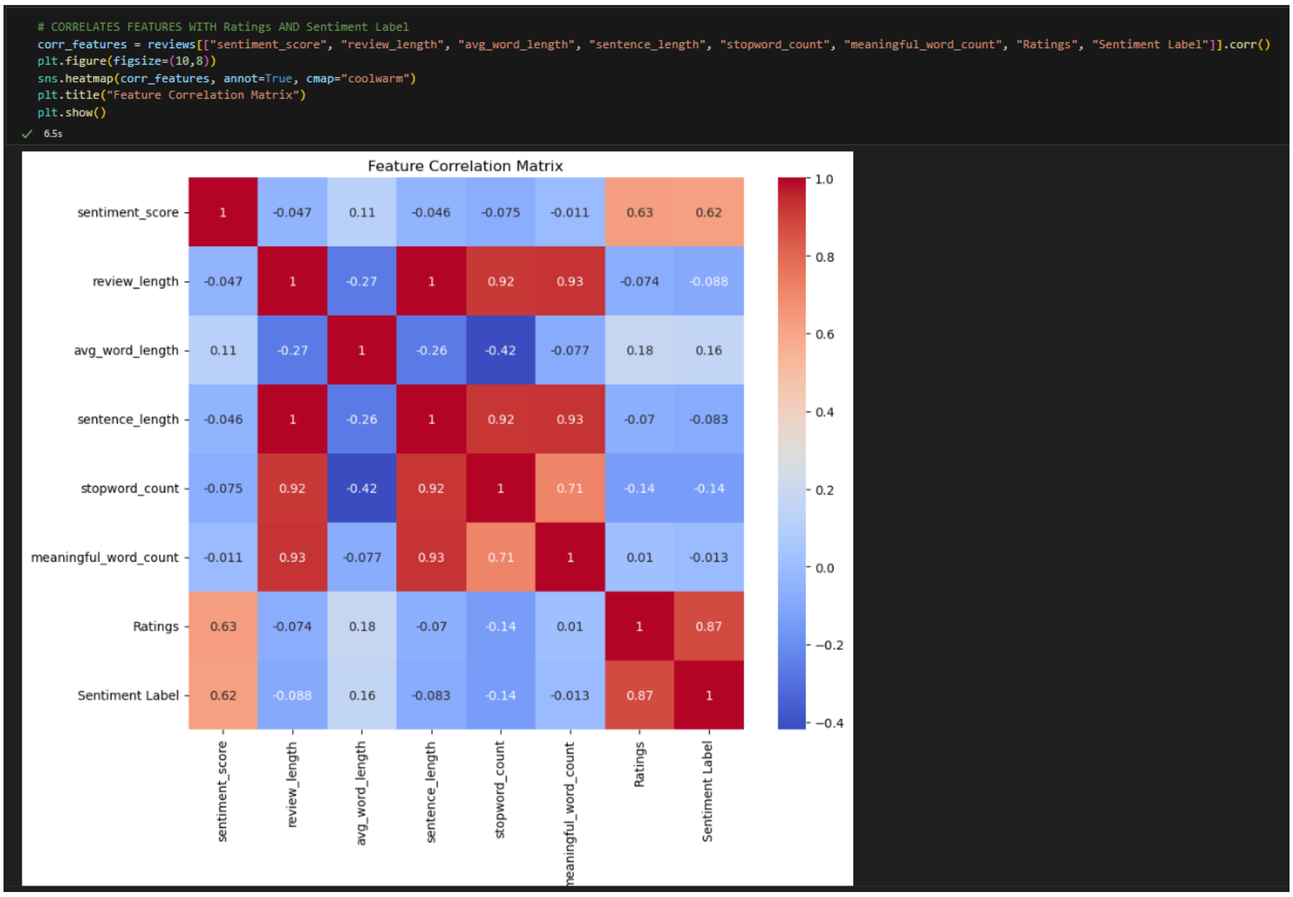


```python
# PAIRPLOT USING THE NUMERIC FEATURES
sns.pairplot(reviews[["sentiment_score", "review_length", "avg_word_length", "sentence_length", "stopword_count", "meaningful_word_count", "Ratings"]])
plt.suptitle("Pairplot of Numeric Features", y=1.02)
plt.show()
```

✓ 37.7s

The correlation coefficient between numeric features is computed ranging from -1 to 1, where -1 is a string negative correlation, zero is no linear relationship and +1 is a strong positive correlation. A heatmap is created with different colours and a legend that highlights the strength of the relationship.

Here we check how each feature relates to Ratings and Sentiment Label.



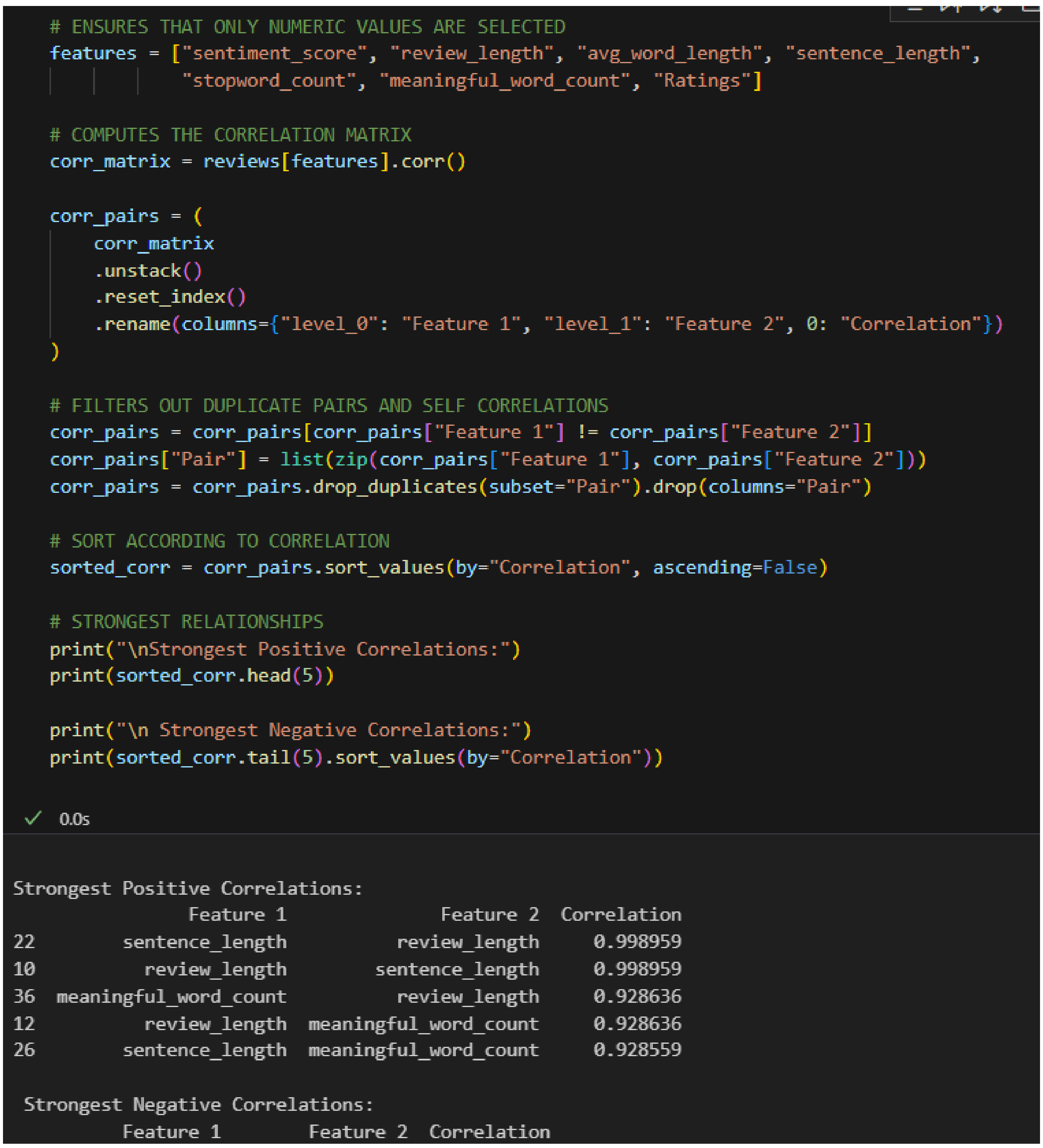
```python
# CORRELATES FEATURES WITH Ratings AND Sentiment Label
corr_features = reviews[["sentiment_score", "review_length", "avg_word_length", "sentence_length", "stopword_count", "meaningful_word_count", "Ratings", "Sentiment Label"]].corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr_features, annot=True, cmap="coolwarm")
plt.title("Feature Correlation Matrix")
plt.show()
```

✓ 6.5s

Feature Correlation Matrix

	sentiment_score	review_length	avg_word_length	sentence_length	stopword_count	meaningful_word_count	Ratings	Sentiment Label
sentiment_score	1	-0.047	0.11	-0.046	-0.075	-0.011	0.63	0.62
review_length	-0.047	1	-0.27	1	0.92	0.93	-0.074	-0.088
avg_word_length	0.11	-0.27	1	-0.26	-0.42	-0.077	0.18	0.16
sentence_length	-0.046	1	-0.26	1	0.92	0.93	-0.07	-0.083
stopword_count	-0.075	0.92	-0.42	0.92	1	0.71	-0.14	-0.14
meaningful_word_count	-0.011	0.93	-0.077	0.93	0.71	1	0.01	-0.013
Ratings	0.63	-0.074	0.18	-0.07	-0.14	0.01	1	0.87
Sentiment Label	0.62	-0.088	0.16	-0.083	-0.14	-0.013	0.87	1

Below we have the strongest positive and strongest negative correlations listed.


```python
# ENSURES THAT ONLY NUMERIC VALUES ARE SELECTED
features = ["sentiment_score", "review_length", "avg_word_length", "sentence_length",
            "stopword_count", "meaningful_word_count", "Ratings"]

# COMPUTES THE CORRELATION MATRIX
corr_matrix = reviews[features].corr()

corr_pairs = (
    corr_matrix
    .unstack()
    .reset_index()
    .rename(columns={"level_0": "Feature 1", "level_1": "Feature 2", 0: "Correlation"})
)

# FILTERS OUT DUPLICATE PAIRS AND SELF CORRELATIONS
corr_pairs = corr_pairs[corr_pairs["Feature 1"] != corr_pairs["Feature 2"]]
corr_pairs["Pair"] = list(zip(corr_pairs["Feature 1"], corr_pairs["Feature 2"]))
corr_pairs = corr_pairs.drop_duplicates(subset="Pair").drop(columns="Pair")

# SORT ACCORDING TO CORRELATION
sorted_corr = corr_pairs.sort_values(by="Correlation", ascending=False)

# STRONGEST RELATIONSHIPS
print("\nStrongest Positive Correlations:")
print(sorted_corr.head(5))

print("\n Strongest Negative Correlations:")
print(sorted_corr.tail(5).sort_values(by="Correlation"))
```

✓ 0.0s

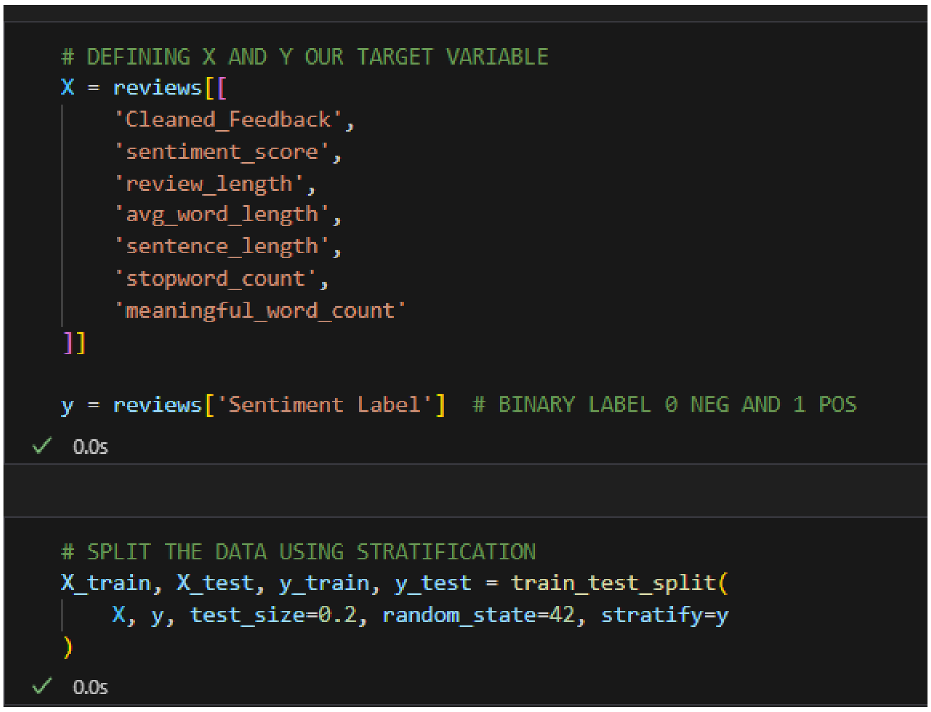
```

Strongest Positive Correlations:
                Feature 1              Feature 2  Correlation
22        sentence_length          review_length     0.998959
10          review_length        sentence_length     0.998959
36  meaningful_word_count          review_length     0.928636
12          review_length  meaningful_word_count     0.928636
26        sentence_length  meaningful_word_count     0.928559

 Strongest Negative Correlations:
          Feature 1        Feature 2   Correlation
```

FEATURE SELECTION

This first code block is building my feature matric(independent variables I am feeding into my model). The x values are the numeric features that are derived from the text that captured the client's tone and length. The y value is the predictor.

The second block splits my dataset into 80% training data and 20% testing data. The 20% will be kept aside for testing later. Random_state allows the split to be reproducible every instance. Stratify = y ensures that the test and training datasets have the same proportion of positive and negative labels to avoid any imbalanced splits (leads to skew model evaluation).

This is the first step towards model training. When selecting the relevant features and stratifying the split, the model has a better chance at generalising on unseen data especially when the sentiment classes are unbalanced.



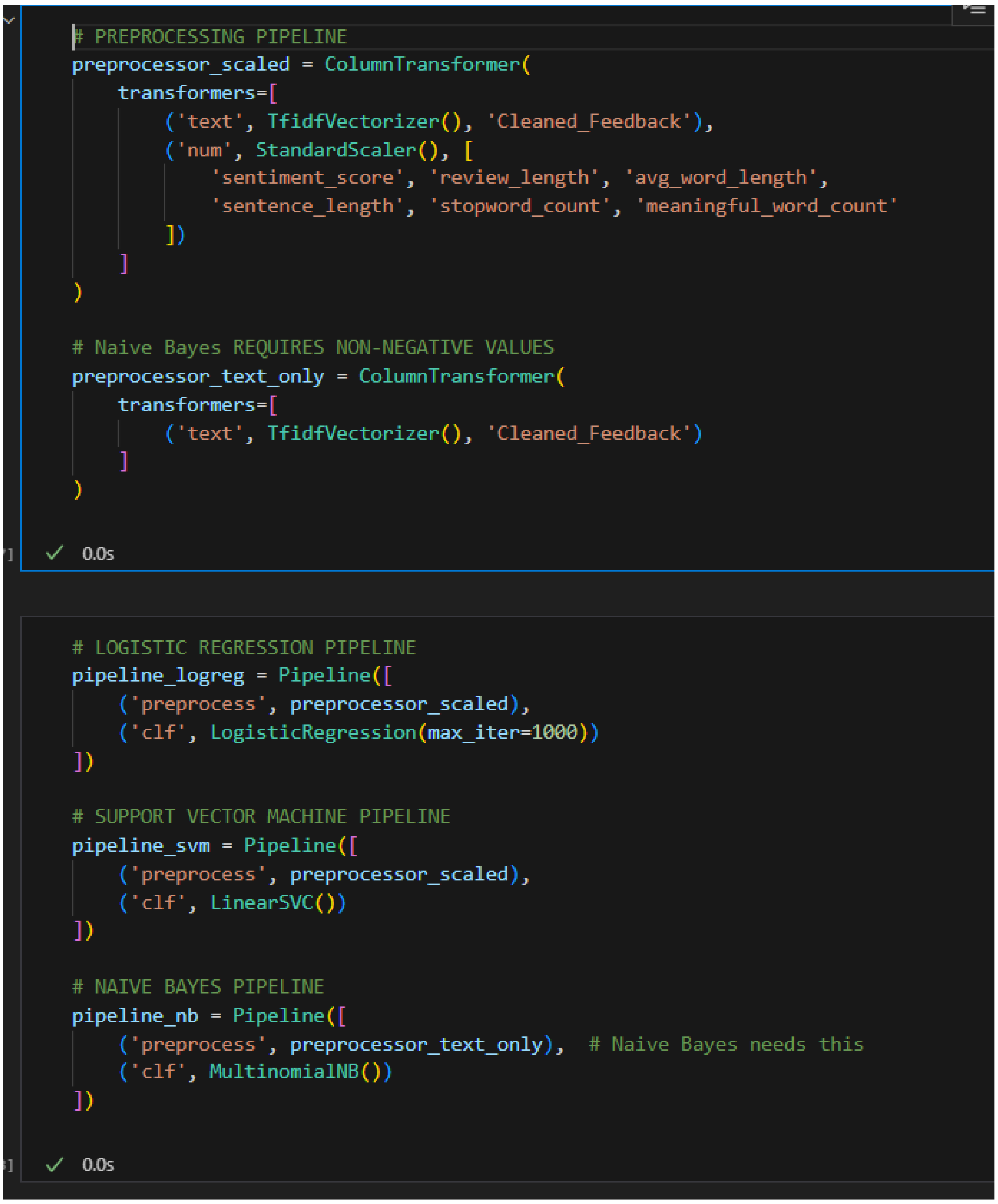
```
# DEFINING X AND Y OUR TARGET VARIABLE
X = reviews[[
    'Cleaned_Feedback',
    'sentiment_score',
    'review_length',
    'avg_word_length',
    'sentence_length',
    'stopword_count',
    'meaningful_word_count'
]]

y = reviews['Sentiment Label']  # BINARY LABEL 0 NEG AND 1 POS
```
✓ 0.0s

```
# SPLIT THE DATA USING STRATIFICATION
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```
✓ 0.0s

Below (first code block) I am preparing text and numerical features for models requiring standardised input. The TfidfVectorizer turns the pre cleaned text into numerical vectors based on the word importance and the StandardScaler standardised features to balance the influence of the features. Then with preprocessor_text_only I am only processing text (no numerical features to be included).

The second code block bundles the preprocessors with classifiers using Pipeline.

With the use of separate pipelines I am able to run comparisons of the models I have selected side by side, I can also swap features in and out and avoid compatibility issues like how SVM requires scaling and Naïve requires non negatives.



```
# PREPROCESSING PIPELINE
preprocessor_scaled = ColumnTransformer(
    transformers=[
        ('text', TfidfVectorizer(), 'Cleaned_Feedback'),
        ('num', StandardScaler(), [
            'sentiment_score', 'review_length', 'avg_word_length',
            'sentence_length', 'stopword_count', 'meaningful_word_count'
        ])
    ]
)

# Naive Bayes REQUIRES NON-NEGATIVE VALUES
preprocessor_text_only = ColumnTransformer(
    transformers=[
        ('text', TfidfVectorizer(), 'Cleaned_Feedback')
    ]
)
```

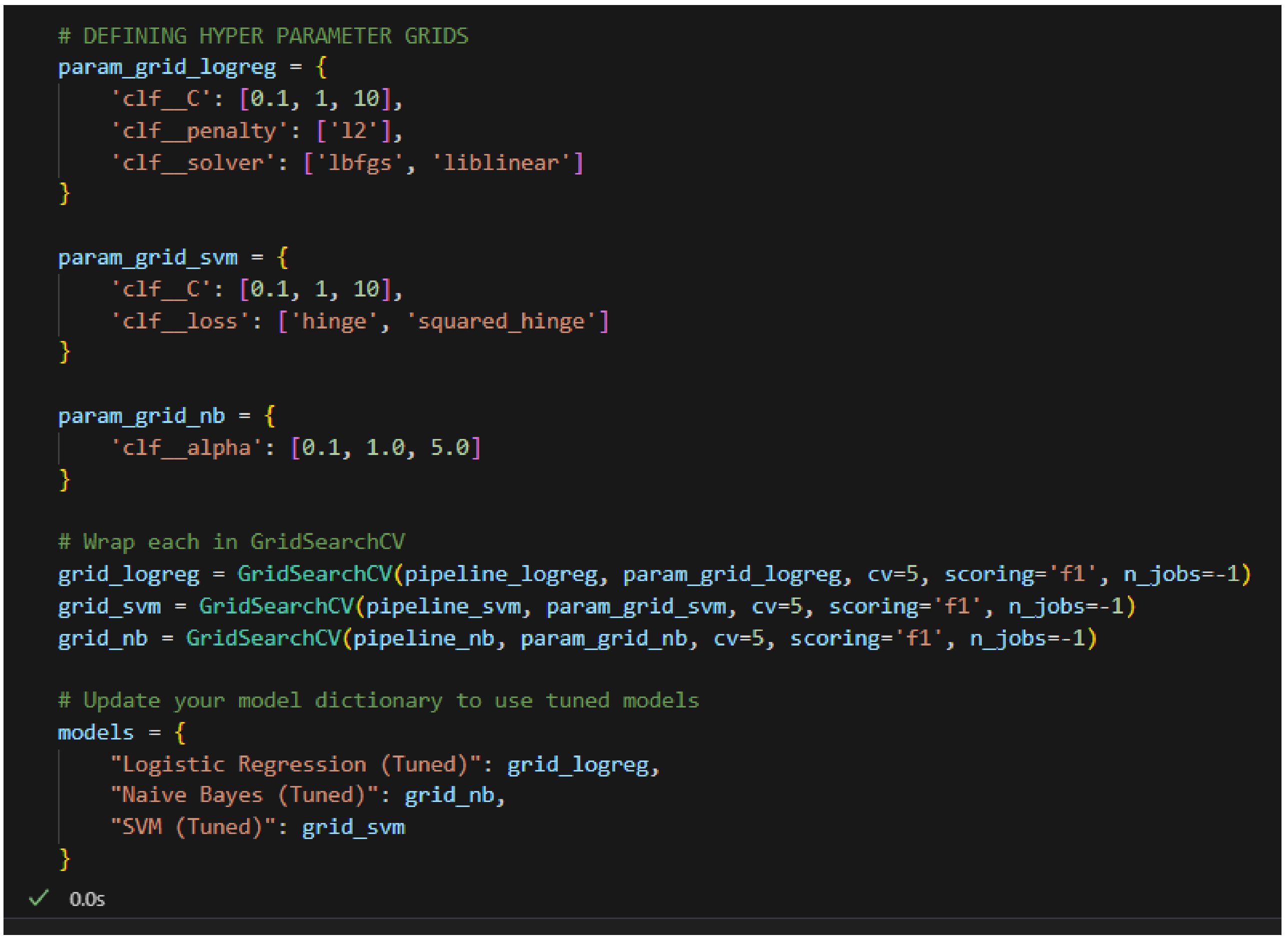
```
✓ 0.0s
```

```
# LOGISTIC REGRESSION PIPELINE
pipeline_logreg = Pipeline([
    ('preprocess', preprocessor_scaled),
    ('clf', LogisticRegression(max_iter=1000))
])

# SUPPORT VECTOR MACHINE PIPELINE
pipeline_svm = Pipeline([
    ('preprocess', preprocessor_scaled),
    ('clf', LinearSVC())
])

# NAIVE BAYES PIPELINE
pipeline_nb = Pipeline([
    ('preprocess', preprocessor_text_only),  # Naive Bayes needs this
    ('clf', MultinomialNB())
])
```

```
✓ 0.0s
```

This is where my hyperparameter tuning takes place to optimise my models and get the best possible performance from all of them.

The code below is setting up for hypermeter tuning for the three different models I selected. GridSearchCV automates the process of finding the best settings for each model by testing combinations of parameters. The parameter grids for each model are defines. Each of the models are paired with their preprocessing pipeline to ensure the data is prepared correctly for training. The pipelines are then wrapped with GridSearchCV to perform 5-fold cross validation. This is to evaluate each parameter combination that makes use of the F1 score as a performance metric.



```
# DEFINING HYPER PARAMETER GRIDS
param_grid_logreg = {
    'clf__C': [0.1, 1, 10],
    'clf__penalty': ['l2'],
    'clf__solver': ['lbfgs', 'liblinear']
}

param_grid_svm = {
    'clf__C': [0.1, 1, 10],
    'clf__loss': ['hinge', 'squared_hinge']
}

param_grid_nb = {
    'clf__alpha': [0.1, 1.0, 5.0]
}

# Wrap each in GridSearchCV
grid_logreg = GridSearchCV(pipeline_logreg, param_grid_logreg, cv=5, scoring='f1', n_jobs=-1)
grid_svm = GridSearchCV(pipeline_svm, param_grid_svm, cv=5, scoring='f1', n_jobs=-1)
grid_nb = GridSearchCV(pipeline_nb, param_grid_nb, cv=5, scoring='f1', n_jobs=-1)

# Update your model dictionary to use tuned models
models = {
    "Logistic Regression (Tuned)": grid_logreg,
    "Naive Bayes (Tuned)": grid_nb,
    "SVM (Tuned)": grid_svm
}
```

TRAIN MODEL

This is where the models are evaluated and the performance of the three is compared.

The preconfigured pipelines are stored in a dictionary called models (makes iteration easier). For each model in the dictionary, the name Training and Evaluating “name of model,” is printed, the model is fit on the training set, the labels of the test set are predicted and finally a classification report shows the performance metrics (F1, accuracy, etc). Each of the blocks in the output cell show the models ability to classify reviews as zero being negative or one being positive.

Logistic Regression

- Positive Reviews are very strong.
- Negative Reviews are weaker.
- Overall Accuracy 84%
- Overall the model is great.

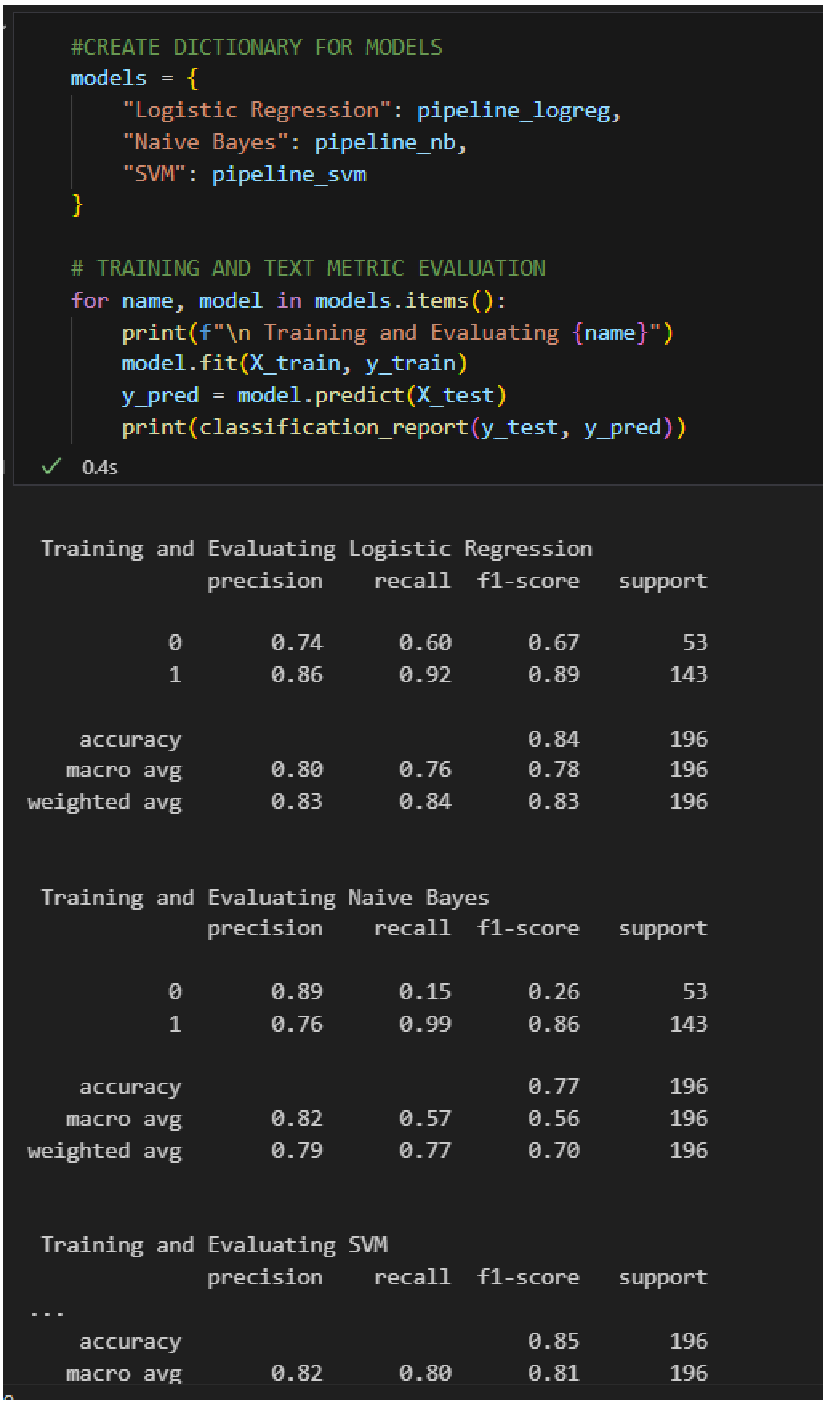
Naïve Bayes

- Positive Reviews almost perfect with a score of 0.99
- Negative Reviews very poor
- Overall Accuracy is 77%
- Not well balanced

SVM

- Positive Reviews strong performance
- Negative Reviews strong performance
- Overall Accuracy 85%
- Most balanced model of the three

From the performances below, the SVM slightly outperforms the other models in terms of accuracy and balance with logistic regression behind it. Although Naïve Bayes has a strong positive class precision it struggles with detecting negatives.



```python
#CREATE DICTIONARY FOR MODELS
models = {
    "Logistic Regression": pipeline_logreg,
    "Naive Bayes": pipeline_nb,
    "SVM": pipeline_svm
}

# TRAINING AND TEXT METRIC EVALUATION
for name, model in models.items():
    print(f"\n Training and Evaluating {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

✓ 0.4s

```

 Training and Evaluating Logistic Regression
              precision    recall  f1-score   support

           0       0.74      0.60      0.67        53
           1       0.86      0.92      0.89       143

    accuracy                           0.84       196
   macro avg       0.80      0.76      0.78       196
weighted avg       0.83      0.84      0.83       196


 Training and Evaluating Naive Bayes
              precision    recall  f1-score   support

           0       0.89      0.15      0.26        53
           1       0.76      0.99      0.86       143

    accuracy                           0.77       196
   macro avg       0.82      0.57      0.56       196
weighted avg       0.79      0.77      0.70       196


 Training and Evaluating SVM
              precision    recall  f1-score   support
...
    accuracy                           0.85       196
   macro avg       0.82      0.80      0.81       196
```

INTERPRET AND EVALUATE MODEL

This is the final step where I am doing a performance evaluation of the multiple trained models on a classification task.

The code loops over each model that is stored in the models' dictionary, the model is trained using the training data. The trained model is then used to predict the labels on the test dataset to produce the predictor. From there a detailed report is printed with the key metrics.

Logistic Regression

- Precision of all reviews, the model predicted it to be 86% truly positive.
- Recall where the model correctly identified 92% of the actual positive reviews.
- F1 score where there is great balance between precision and recall (0.89)

However this model seems to struggle with negative reviews, only identifying 60% of them correctly. The model's overall metrics were weighed towards the positive class which means they do show a very strong performance.

Naïve Bayes

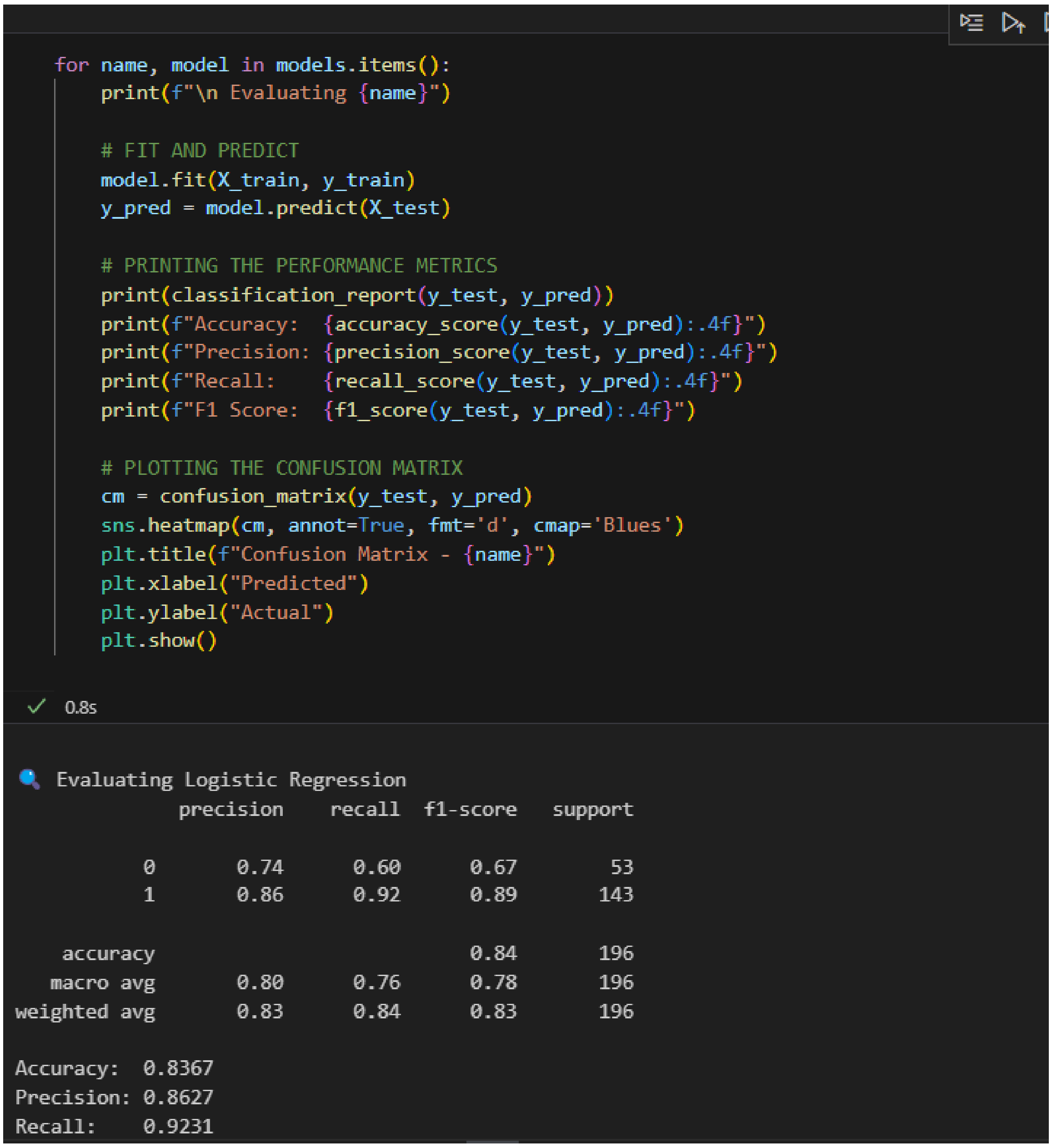
- The model is good at identifying these correctly with a score of 99%. With a high F1 score that means the models confident and consistent when it comes to positive feedback.
- Despite having a high precision rate of 0.89, the recall was very low. This means that most of the actual negative reviews are missed.

This model is highly biased toward predicting positive reviews.

SVM

- When it comes to positive reviews the model performs well with high recall precision meaning that most positive cases were correctly identified.
- When it comes to negative revies it is also good with 70% recall and 74% precision which means that it was able to capture most of the complaints and barely mislabelled the positives as negatives.

The SVM is the most balanced of the other models. It is able to manage positive and negative reviews with strong consistency.



```python
for name, model in models.items():
    print(f"\n Evaluating {name}")

    # FIT AND PREDICT
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # PRINTING THE PERFORMANCE METRICS
    print(classification_report(y_test, y_pred))
    print(f"Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_test, y_pred):.4f}")
    print(f"F1 Score:  {f1_score(y_test, y_pred):.4f}")

    # PLOTTING THE CONFUSION MATRIX
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

✓ 0.8s

```
🔍 Evaluating Logistic Regression
              precision    recall  f1-score   support

           0       0.74      0.60      0.67        53
           1       0.86      0.92      0.89       143

    accuracy                           0.84       196
   macro avg       0.80      0.76      0.78       196
weighted avg       0.83      0.84      0.83       196

Accuracy:  0.8367
Precision: 0.8627
Recall:    0.9231
```

References

Acharya, N., 2023. *Sentimental Analysis using Logistic regression.* [Online] Available at: https://medium.com/@nirajan.acharya777/sentimental-analysis-using-linear-regression-86764bfde907
[Accessed 20 June 2025].

Ashfaque, Z., 2023. *Sentiment Analysis with Naive Bayes Algorithm.* [Online] Available at: https://medium.com/@zubairashfaque/sentiment-analysis-with-naive-bayes-algorithm-a31021764fb4
[Accessed 20 June 2025].

IBM, 2023. *What is sentiment analysis?.* [Online] Available at: https://www.ibm.com/think/topics/sentiment-analysis
[Accessed 19 June 2025].

Reddy, V., 2018. *Sentiment Analysis using SVM.* [Online] Available at: https://medium.com/scrapehero/sentiment-analysis-using-svm-338d418e3ff1
[Accessed 20 June 2025].