NLP Task 1: Topic Modelling (750 words) 20%

***Literature Review and Rationale: (150 words)** 15% 311w*

As the internet affects how business is conducted, customer-generated reviews are becoming an increasingly important part of e-commerce. Collecting and presenting such information valuable to consumers is a core part of companies such as TripAdvisor.

TripAdvisor reviews are typically written by customers who have previously used a service or product. TripAdvisor makes the reviews widely accessible so potential customers can read them and use the information for their own purchasing decisions. A standard review format is a block of text and a numeric rating (1-5 bubbles) that summarises the customer's sentiment toward the accommodation in a single number. However, such a rating scheme is of limited use to the customer and the hotels themselves. Hotels are interested in the overall quality of the customer's experience and its quality in specific aspects. For example, a hotel interested in customer reviews may be interested in distinct aspects such as quality of the room, quality of the hotel facilities, location of the hotel, helpfulness of the staff and perceived value concerning the price. This suggests that one overall rating may be less meaningful than a set of ratings specific to each aspect of interest.

An area of Natural Language Processing (NLP) that can assist hotels better to understand the needs of their customers' needs and ultimately improve their customer experience is Topic Modelling (Shen, 2012). Topic Modelling, a type of unsupervised data mining technique, constitutes a popular tool for extracting important themes (topics) from unstructured data and is employed to reveal and annotate extensive documents collection with thematic information (Christodoulou, 2020). Latent Dirichlet Allocation (LDA) will be used in this study as a topic-modelling technique. In LDA, a topic is a probability distribution function over a set of words used as a type of text summarisation. Using Bayesian probabilities, LDA then expresses the relationships between words in terms of their affinity to certain latent variables (topics) (Alexander, Blank, & Hale, 2018).

***Data: (200)** 20% 122w*

Wrangling

To tokenise the words in the customer reviews, the **Gensim** package and **simple_preprocess()** function was used.

As discussed in part *2.3.3*, bigrams and trigram of the text was created to understand the frequency of words occurring together. The *min_count* and *threshold* arguments were employed to increase the difficulty of combining words to improve the quality of the bigrams and trigrams. The following code was used to achieve this:

```
# Create bigram and trigram

bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold
fewer phrases.

trigram = gensim.models.Phrases(bigram[data_words], threshold=100)
```

Lemmatisation is the process of converting a word to its base form. Lemmatisation first considers the context and then converts the word to its meaningful base form whereas stemming removes the last few characters, potentially leading to incorrect meanings and spelling errors (Balakrishnan & Llyod-Yemoh, 2014). Therefore, lemmatisation was used over stemming and carried out via the following code:

```
# Define functions for stopwords, bigrams, trigrams and lemmatisation
def removeStopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for
doc in texts]

def makeBigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def makeTrigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Summary of data for NLP task

Visualisation of data

NLP Task Implementation: (200) 35% 163w

Feature normalisation

To perform LDA Topic Modelling, a dictionary and a corpus was created. The dictionary was generated by implementing the **corpora.Dictionary()** function on the lemmatized text. The corpus was then created by generating a term document frequency matrix by employing the bag-of-words (BOW) method.

```
import gensim.corpora as corpora
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)
# Create Corpus
texts = data_lemmatized
# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]
```

ML algorithm

The LDA model used for this study was trained using the **LdaModel()** function from the **Gensim** package. The study tested several different topics to represent the data including, 20, 10 and 5 topics. Given the coherence score discussed later, the study determined that seven topics were appropriate. This was implemented by using the *num_topics* argument when training the LDA model, as shown below:

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=7,

                                           random_state=100,

                                           update_every=1,

                                           chunksize=100,

                                           passes=10,

                                           alpha='auto',

                                           per_word_topics=True)
```

Hyper-parameters

The following hyper-parameters were optimised when training the LDA model:

- *Chunksize* refers to the number of documents used in each training chunk.
- *Passes* refer to the total number of training passes.
- *Alpha* determines the sparsity of the topics.
- *Update_every* specifies how often the model parameters should be updated.

NLP Task Output: (200) 30% 306w

Quality metrics

The model Perplexity and model Coherence Score was computed using the following code:

```
# Compute Perplexity

print('\nPerplexity: ', lda_model.log_perplexity(corpus))

# Compute Coherence Score

coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized,
dictionary=id2word, coherence='c_v')

coherence_lda = coherence_model_lda.get_coherence()
```

The model Perplexity was -7.11. Perplexity is a measurement of how well a probability model predicts test data (Fig. 1). The lower the Perplexity, the more influential the model.

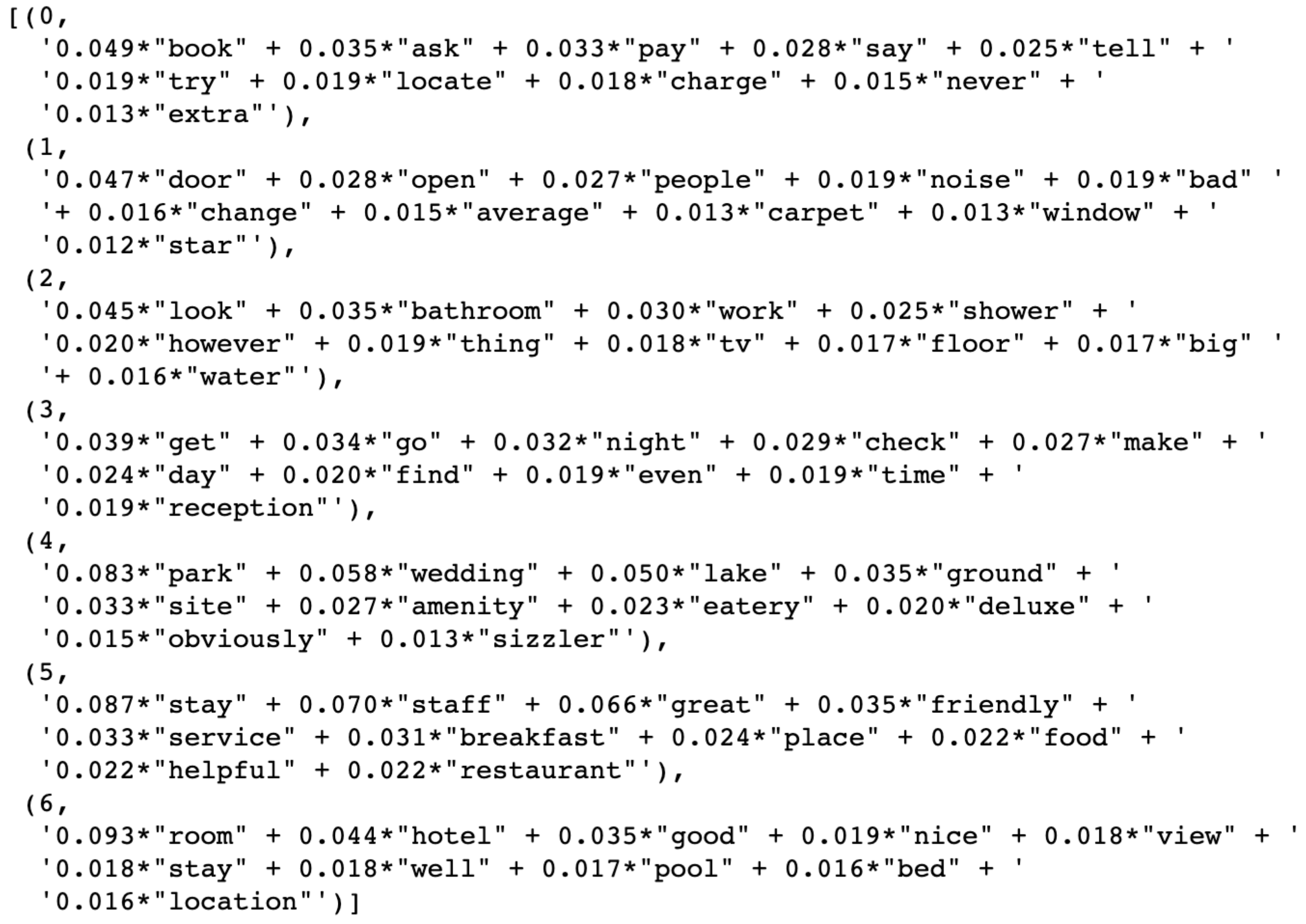
The model Coherence Score was 0.427. Topic Coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. The perplexity and coherence score give a sense of the relevance of the words categorised in each topic—the higher the Coherence Score, the better the model.

	Perplexity	**Coherence Score**
LDA model	-7.11	0.427

Fig 1. Model performance.

Summary of outputs

Below is a screenshot of the output representing the words in each topic and their relative weight (Fig. 2).



```
[(0,
  '0.049*"book" + 0.035*"ask" + 0.033*"pay" + 0.028*"say" + 0.025*"tell" + '
  '0.019*"try" + 0.019*"locate" + 0.018*"charge" + 0.015*"never" + '
  '0.013*"extra"'),
 (1,
  '0.047*"door" + 0.028*"open" + 0.027*"people" + 0.019*"noise" + 0.019*"bad" '
  '+ 0.016*"change" + 0.015*"average" + 0.013*"carpet" + 0.013*"window" + '
  '0.012*"star"'),
 (2,
  '0.045*"look" + 0.035*"bathroom" + 0.030*"work" + 0.025*"shower" + '
  '0.020*"however" + 0.019*"thing" + 0.018*"tv" + 0.017*"floor" + 0.017*"big" '
  '+ 0.016*"water"'),
 (3,
  '0.039*"get" + 0.034*"go" + 0.032*"night" + 0.029*"check" + 0.027*"make" + '
  '0.024*"day" + 0.020*"find" + 0.019*"even" + 0.019*"time" + '
  '0.019*"reception"'),
 (4,
  '0.083*"park" + 0.058*"wedding" + 0.050*"lake" + 0.035*"ground" + '
  '0.033*"site" + 0.027*"amenity" + 0.023*"eatery" + 0.020*"deluxe" + '
  '0.015*"obviously" + 0.013*"sizzler"'),
 (5,
  '0.087*"stay" + 0.070*"staff" + 0.066*"great" + 0.035*"friendly" + '
  '0.033*"service" + 0.031*"breakfast" + 0.024*"place" + 0.022*"food" + '
  '0.022*"helpful" + 0.022*"restaurant"'),
 (6,
  '0.093*"room" + 0.044*"hotel" + 0.035*"good" + 0.019*"nice" + 0.018*"view" + '
  '0.018*"stay" + 0.018*"well" + 0.017*"pool" + 0.016*"bed" + '
  '0.016*"location"')]
```

Fig 2. Screenshot of the ten keywords and their weighting for each topic.

The seven topics that were extracted via the LDA model are summarised in Table 1. The keywords were determined by those words with the highest beta value within the topic. The keywords with the highest relative probability of belonging to the given topic and were organised in order from the most important to least important keyword per topic. As LDA is an admixture model, the same words can belong to more than one topic.

Topic themes were derived from the keywords identified following the LDA Topic Modelling process. These themes are assumed are

Topic Assumption	Relevant keywords
Hotel booking process	Book, ask, pay, say, tell, try, locate, charge, never, extra.
Ambience	Door, open, people, noise, bad, change, average, carpet, window, star.
Room Features	Look, bathroom, work, shower, however, thing, tv, floor, big, water.
Reception	Get, go, night, check, make, day, find, even, time, reception.
Neighbourhood	Park, wedding, lake, ground, site, amenity, eatery, deluxe, obviously sizzler.
Staff Professionalism	Stay, staff, great, friendly, service, breakfast, place, food, helpful, restaurant.
Facilities	Room, hotel, good, nice, view, stay, well, pool, bed, location.

Table 1. Assumed topic themes and their keywords.

Service-related topics include the hotel booking process, reception, staff professionalism. The topic of hotel booking process includes the tasks associated with the booking process such as booking the accommodation (e.g., ‘book’, ‘ask’, ‘say’, ‘tell’), and payment (e.g., ‘pay’, ‘charge’, ‘extra’). The reception topic encompasses guest services such as luggage assistance (e.g., ‘get’, ‘go’, ‘check’, ‘day’, ‘night’) and booking services (e.g., ‘make’, ‘find’, ‘time’). The staff professionalism topic emphasizes the emotional warmth of the experience (e.g., ‘friendly’, ‘service’, ‘helpful’) and hospitality (e.g., ‘breakfast’, ‘food’, ‘restaurant’).

Topics related to the physical condition of the accommodation include the facilities and ambience. The topic dubbed facilities relate to the accommodation facilities (e.g., 'pool') and the view of the surrounding area (e.g., 'view', 'location'). The ambience topic mainly encompasses two distinct ambient qualities of the room, including sound (e.g., 'people', 'noise') and airflow (e.g., 'open', 'door', 'window', 'carpet').

Topics related to the physical condition of the room include the room features. The room features topic relates specifically to room amenities (e.g., ‘bathroom’, ‘shower’, ‘floor’, ‘water’) and dimensions/style (e.g., ‘look’, ‘floor’, ‘big’).

Location-related topics include the surrounding neighbourhood. The neighbourhood topic encompasses the local attractions (e.g., ‘park’, ‘wedding’, ‘lake’, ‘eatery’).

Visualise outputs

An Intertopic Distance Map (via multidimensional scaling) was used to interactively visualise the outputs of the Topic Model (Fig. 3). This was achieved using the pyLDAvis package and the following code:

```
# Visualise the topics

pyLDAvis.enable_notebook()

vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
```

From our topic model, we were able to obtain top keywords from each topic:

1. 39.7% of tokens about the hotel facilities such as view, pool, and location.

2. 19.8% of tokens about staff professionalism such as friendly, service, and helpful.

3. 15.6% of tokens about the hotel reception such as night, check-in, and reception.

4. 10.1% of tokens about room features such as bathroom, shower and tv.

5. 7% of tokens about hotel booking process such as paying and booking accommodation.

6. 5.6% of tokens about accommodation ambience such as noise and décor.

7. 2.2% of tokens about the neighbourhood such as parks, weddings, lakes, and eateries.

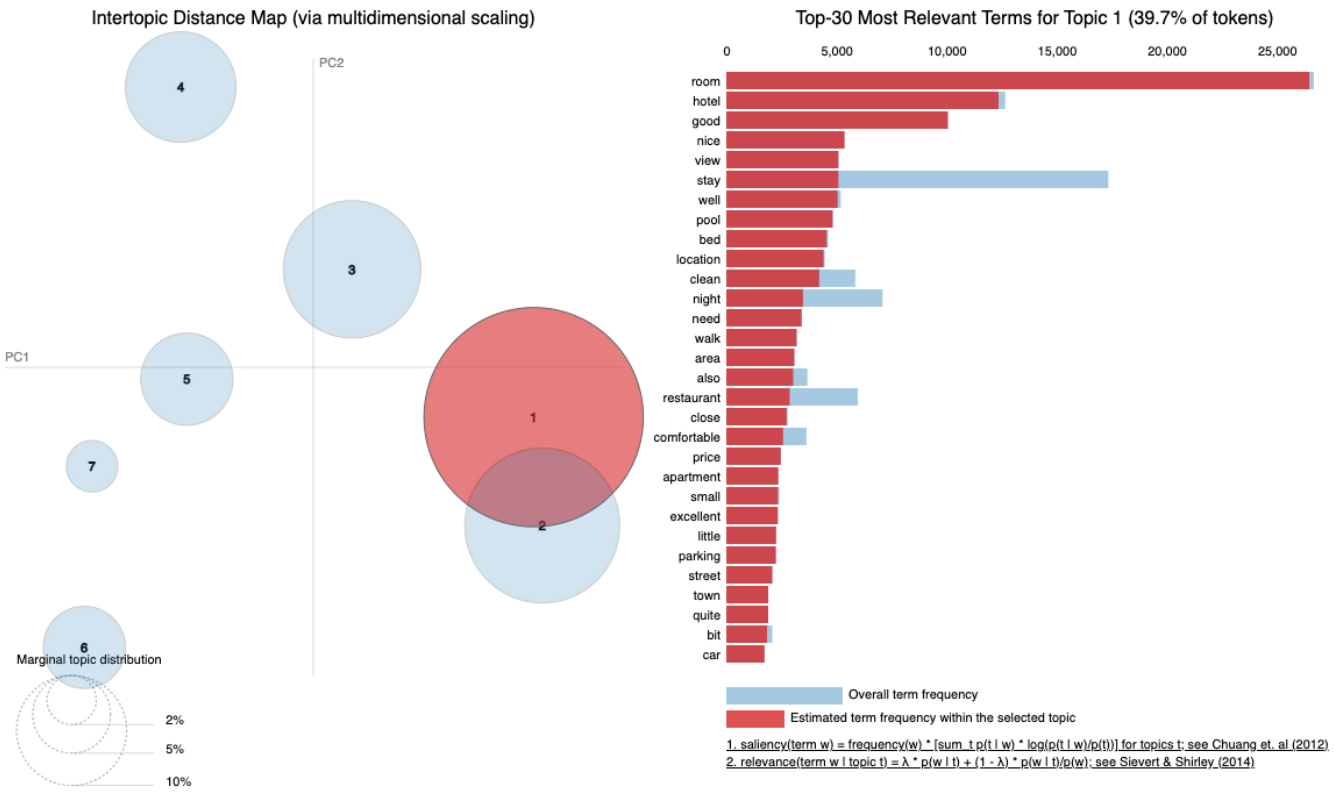


Fig 3. Screenshot of interactive visualisation representing the seven modelled topics and their associated words.

NLP task alignment with issue

Topic Modelling provides an inductive, data-driven approach that validates and extends current theory regarding the dimensions that affect customers in the tourism and hospitality industry (Sutherland, Sim, Lee, & Byun, 2020). This is done by utilising a large amount of text data from the customers. LDA extends the theory by offering more precise distinctions between the dimensions. This study extracted seven valuable topics in the customer reviews. The three topics outlining the most text include hotel facilities, staff professionalism and reception service.

By performing Topic Modelling, hotels can better understand the topic themes that customers discuss in their reviews left on TripAdvisor. For hotels to improve their customer experience, they could look at how they approach their facilities, how their staff interact with the customers and their reception processes.

NLP Task 2: Sentiment Analysis (750 words)

Literature Review and Rationale: (150 words) 15% 174w

Sentiment analysis is an NLP process that aims to classify language data concerning underlying attitudes, emotions, or sentiments (Valdivia, Luzon, & Herrera, 2017). A statement can be classified along a continuum from positive to negative sentiment. These data can then be used to evaluate opinions towards any number of topics. Sentiment analysis can be applied to individuals, but it is beneficial for evaluating public sentiment on a given topic. User-generated content websites are an ideal source of sentiment data, and the results can inform decision-making in several situations.

The interest in sentiment analysis has increased significantly due to the large amount of stored text in web applications and the importance of online customer opinions. As a result, more than 1 million research papers contain the term "sentiment analysis," and various start-ups have been created to analyse sentiments in social media companies (Valdivia, Luzon, & Herrera, Sentiment Analysis on TripAdvisor: Are There Inconsistencies in User Reviews?, 2017).

Multiple studies on TripAdvisor exist, but there is no complete analysis from the sentiment analysis viewpoint. This article proposes TripAdvisor as a source of data for sentiment analysis tasks.

Data: (200) 20% 695w

Wrangling

VADER (Valence Aware Dictionary and Sentiment Reasoner) was used to get sentimental scores of the user reviews and convert them into three categorical sentiments (positive, negative, and neutral). VADER is a lexicon and rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media (Borg & Boldt, 2020). The following code was used to create two new columns that give the sentiment score and the sentiment category of the review:

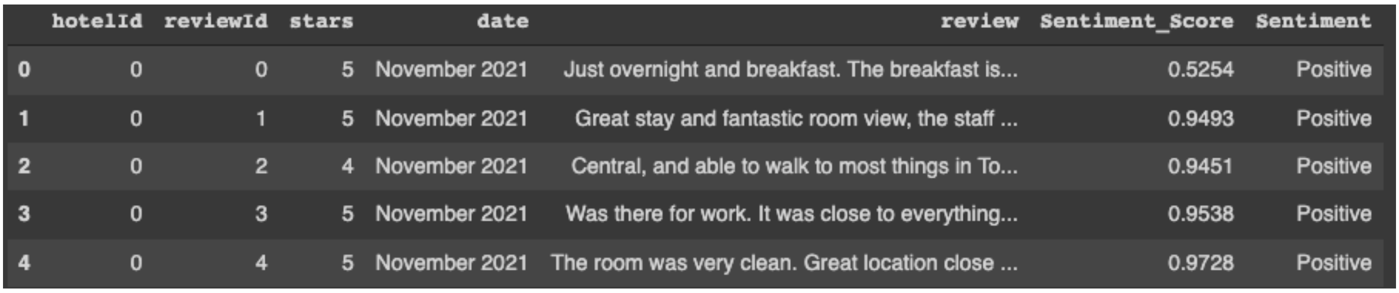
```
# Creating sentimental polarity
analyzer = SentimentIntensityAnalyzer()
def compound_score(txt):
    return analyzer.polarity_scores(txt)["compound"]

# Sentiments
def sentiment(score):
    emotion = ""
    if score >= 0.5:
        emotion = "Positive"
    elif score <= -0.5:
        emotion = "Negative"
    else:
        emotion = "Neutral"
    return emotion

# Applying compound score
polarity_scores = df["review"].astype("str").apply(compound_score)
df["Sentiment_Score"] = polarity_scores

## Applying Sentiment
df["Sentiment"] = df["Sentiment_Score"].apply(sentiment)
```

Fig. 1 displays the output of the dataframe.



	hotelId	reviewId	stars	date	review	Sentiment_Score	Sentiment
0	0	0	5	November 2021	Just overnight and breakfast. The breakfast is...	0.5254	Positive
1	0	1	5	November 2021	Great stay and fantastic room view, the staff ...	0.9493	Positive
2	0	2	4	November 2021	Central, and able to walk to most things in To...	0.9451	Positive
3	0	3	5	November 2021	Was there for work. It was close to everything...	0.9538	Positive
4	0	4	5	November 2021	The room was very clean. Great location close ...	0.9728	Positive

Fig 1. Screenshot of top five rows in '*hotelReviews*' data frame.

It was determined that no missing data were present in the data frame using the **isna().sum()** function.

Summary of data for NLP task

After checking our data for class balance, this report found that most of the user reviews were positive (Fig. 2).

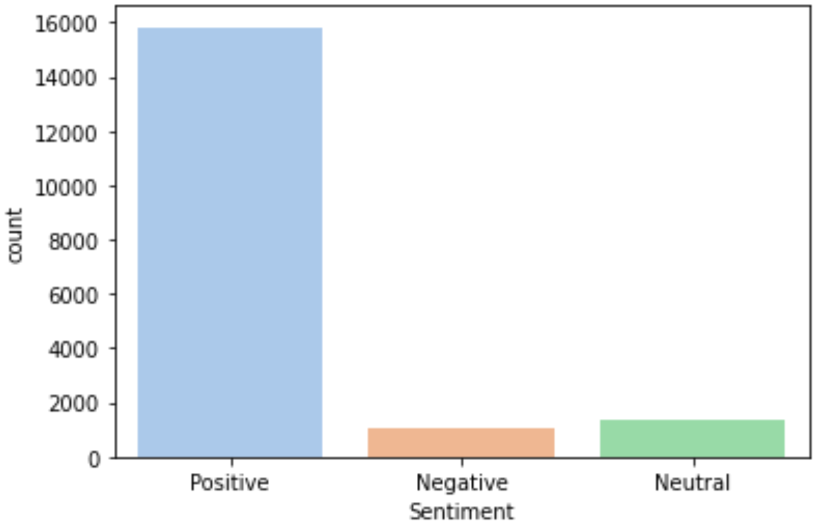


Fig 2. Distribution of sentiment class in '*hotelReviews*' data frame.

This study performed down-sampling to balance the classes and created a subset of the data which contained the top 1000 rows from each of the three 'Sentiment' categories (Fig. 3). This was achieved via the following code:

```
# Function to retrieve top few numbers of each category
def get_top_data(top_n = 1000):
    top_data_df_positive = df[df['Sentiment'] == 'Positive'].head(top_n)
    top_data_df_negative = df[df['Sentiment'] == 'Negative'].head(top_n)
    top_data_df_neutral = df[df['Sentiment'] == 'Neutral'].head(top_n)
    top_data_df_small = pd.concat([top_data_df_positive, top_data_df_negative, top_data_df_neutral])
    return top_data_df_small
# Function call to get the top 1000 from each sentiment
df = get_top_data(top_n=1000)
# After selecting top few samples of each sentiment
print("After segregating and taking equal number of rows for each sentiment:")
print(df['Sentiment'].value_counts())
df.head(10)
```

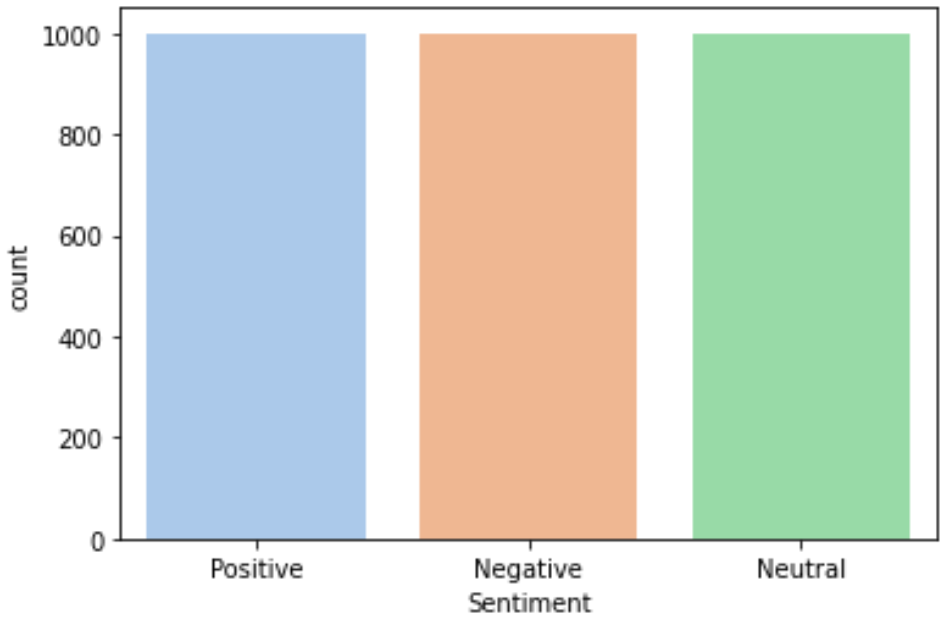


Fig 3. Distribution of sentiment class in '*hotelReviews*' data frame after downsampling.

Visualisation of data

Fig. 4 illustrates that people with 5-star ratings have the highest positive sentiment. Whereas at lower ratings, the sentiment is primarily negative.

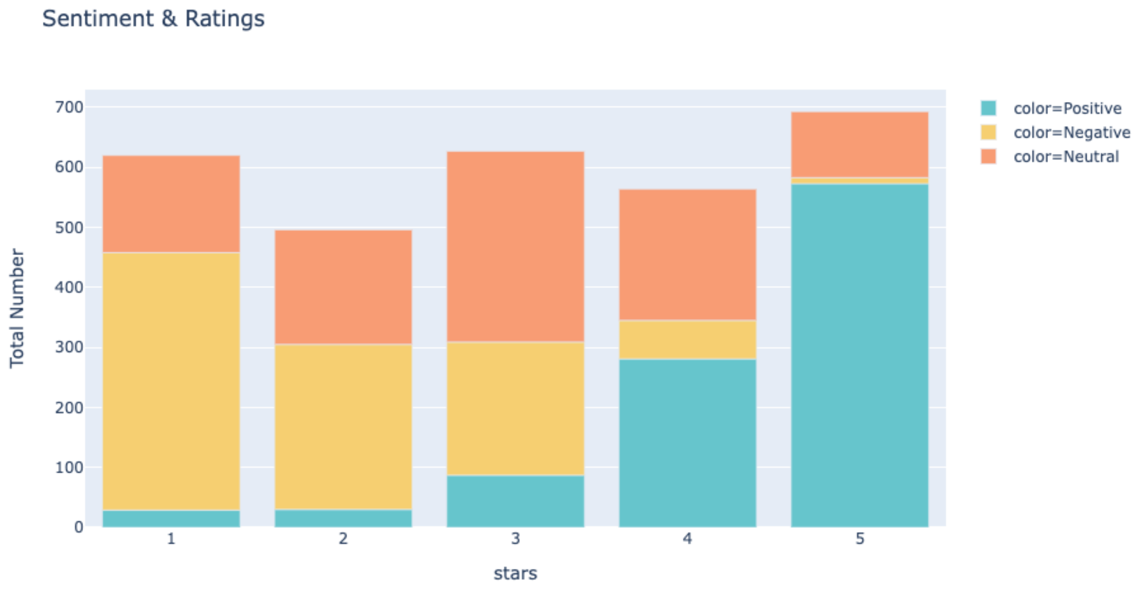


Fig 4. Distribution of sentiment class and customer ratings.

A pie chart using the **Plotly** library (Fig. 5) displays the distribution of different ratings. The range is between 16.5% (2-star review) and 23.1% (5-star review), which demonstrates the distribution is well spread across the ratings.

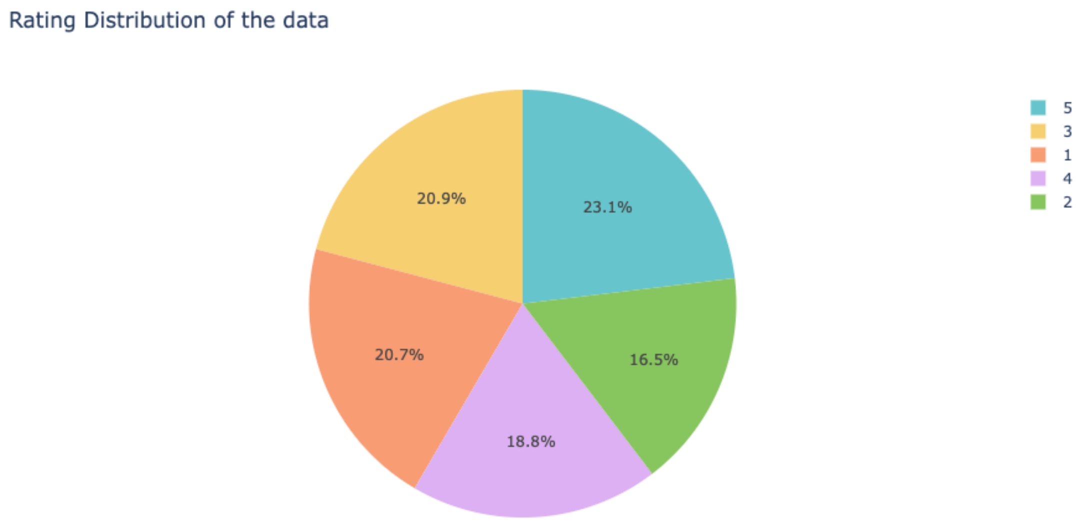


Fig 5. Distribution of customer ratings after downsampling.

Using the **WordCloud** package, three separate word clouds were created to understand the frequency of different words used in the reviews. The most common words used in all three Sentiments were hotel, room and stay (Fig. 7).

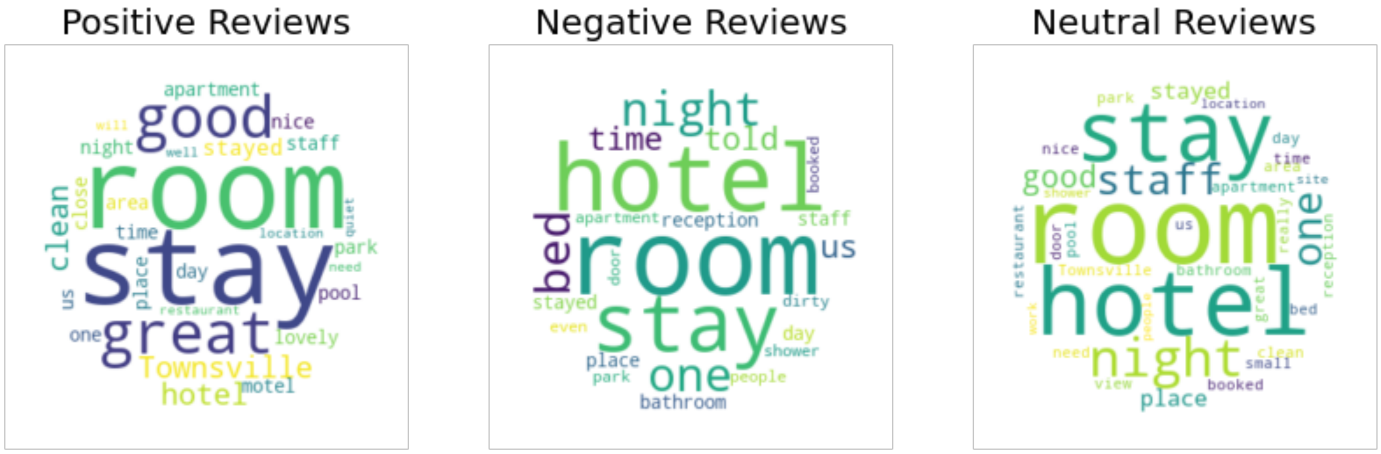


Fig 7. Frequency of words for each sentiment class.

The **Gensim** and **Counter** function packages were used to find keywords and build a graph using tokens from the text (Fig. 8). Apart from hotel rooms, users talked about staff and parking.

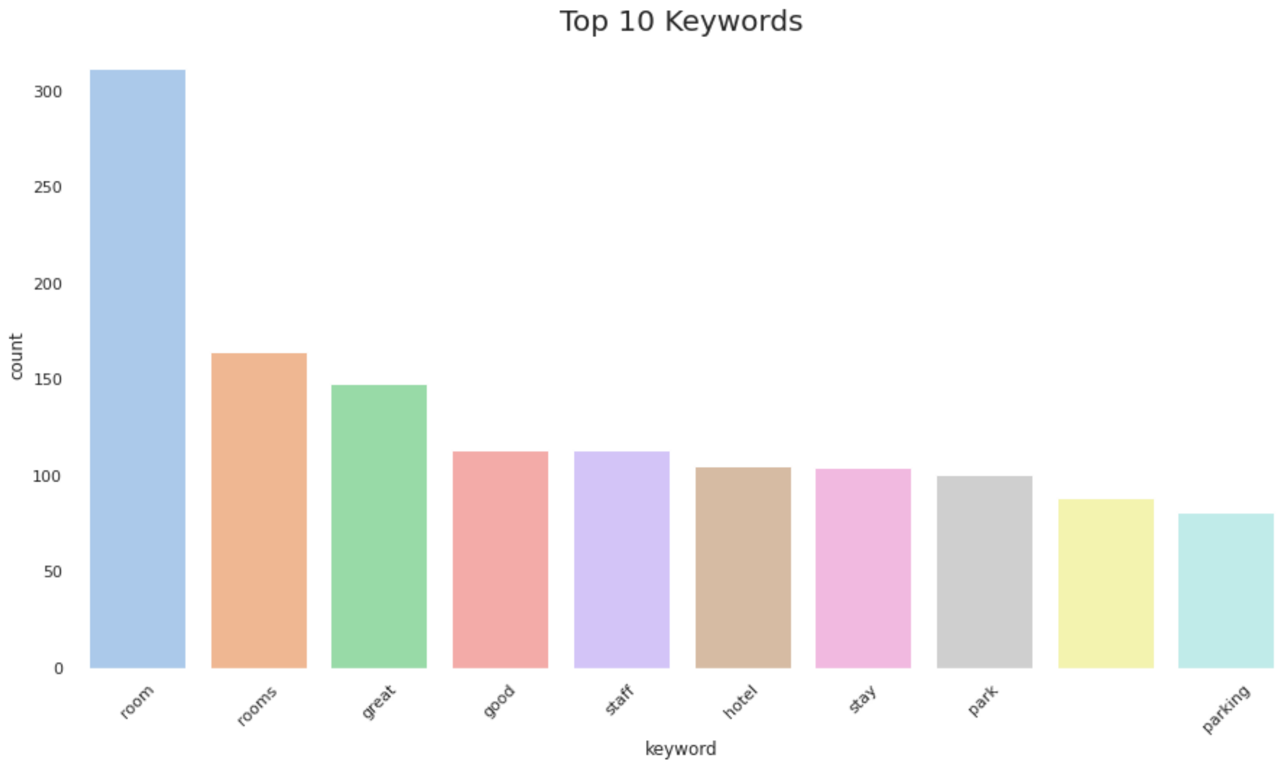


Fig 8. Distribution of top ten keywords.

Most used words were removed to improve the performance and increase accuracy of the model.

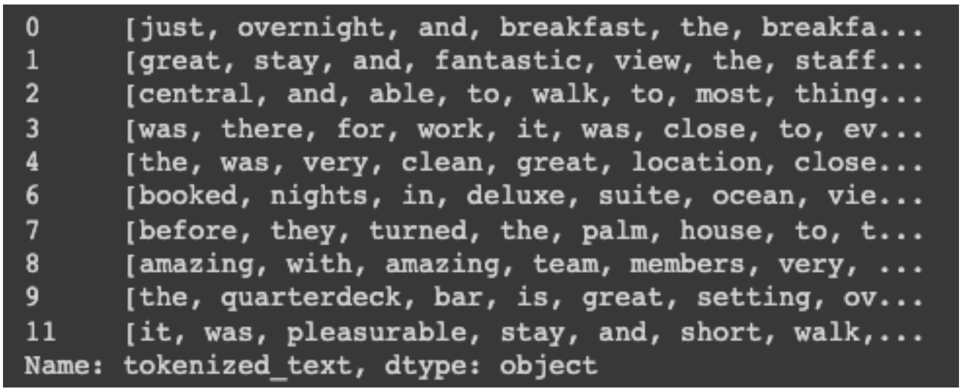
```
words = ["hotel","room","rooms","hotels"]
for x in words:
    data["review"] = data["review"].astype(str).str.replace(x,"")
```

In computing, stop words are words that are filtered out before or after the natural language data (text) are processed. An example of stop words are: 'a', 'an', 'the', 'this', 'not'. There is no universal set of stop words, and the list of stop words varies between packages., Removal of stop words removes the necessary words required to get the sentiment, and sometimes it can change the sentence's meaning (Arumugam & Shanmugamani, 2018). For this reason, the study skipped the removal of stop words for the sentiment analysis.

Tokenisation is when the sentence/text is split into an array of words called tokens (Reese & Bhatia, 2018). This helps to transform each word separately and is also required to transform words into numbers. The **simple_preprocess** function converts text to lower case and remove punctuations. It has min and max length parameters, which help filter out rare words that will fall in that range of lengths (Fig. 9).**Simple_preprocess** was used to get the tokens for the data frame as it does most of the pre-processing under the hood. The following code was used to tokenise the review text.

```
# Tokenisation

from gensim.utils import simple_preprocess
# Tokenize the text column to get the new column 'tokenized_text'
df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['review']]
```



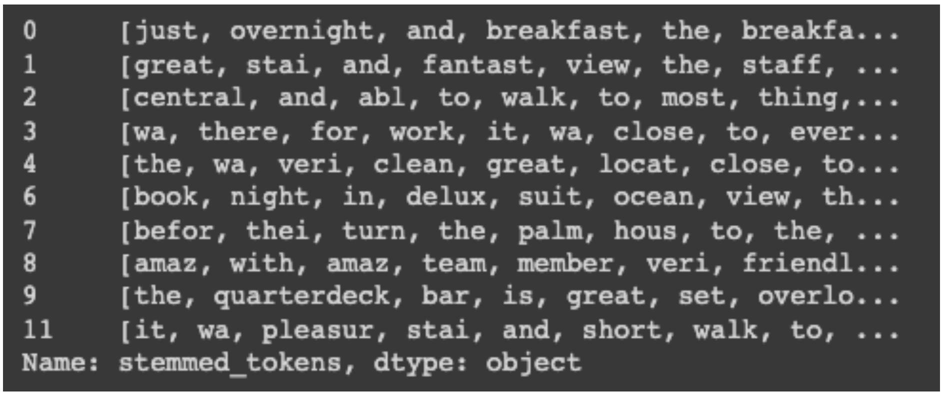
```
0     [just, overnight, and, breakfast, the, breakfa...
1     [great, stay, and, fantastic, view, the, staff...
2     [central, and, able, to, walk, to, most, thing...
3     [was, there, for, work, it, was, close, to, ev...
4     [the, was, very, clean, great, location, close...
6     [booked, nights, in, deluxe, suite, ocean, vie...
7     [before, they, turned, the, palm, house, to, t...
8     [amazing, with, amazing, team, members, very, ...
9     [the, quarterdeck, bar, is, great, setting, ov...
11    [it, was, pleasurable, stay, and, short, walk,...
Name: tokenized_text, dtype: object
```

Fig 9. Output following Tokenisation of customer reviews text.

The Stemming process reduces the words to its' root word. Unlike Lemmatization which uses grammar rules and dictionaries for mapping words to root form, stemming removes suffixes/prefixes (Fig. 10). Stemming is widely used in the application of search engine optimisation (SEOs), web search results, and information retrieval. If the root word matches in the text somewhere, it will help retrieve all the related documents in the search. The following code was used to stem the tokens in ‘tokenized_text’:

```
# Stemming
from gensim.parsing.porter import PorterStemmer
porter_stemmer = PorterStemmer()

# Get the stemmed_tokens
df['stemmed_tokens'] = [[porter_stemmer.stem(word) for word in tokens] for tokens in
df['tokenized_text'] ]
```



```
0     [just, overnight, and, breakfast, the, breakfa...
1     [great, stai, and, fantast, view, the, staff, ...
2     [central, and, abl, to, walk, to, most, thing,...
3     [wa, there, for, work, it, wa, close, to, ever...
4     [the, wa, veri, clean, great, locat, close, to...
6     [book, night, in, delux, suit, ocean, view, th...
7     [befor, thei, turn, the, palm, hous, to, the, ...
8     [amaz, with, amaz, team, member, veri, friendl...
9     [the, quarterdeck, bar, is, great, set, overlo...
11    [it, wa, pleasur, stai, and, short, walk, to, ...
Name: stemmed_tokens, dtype: object
```

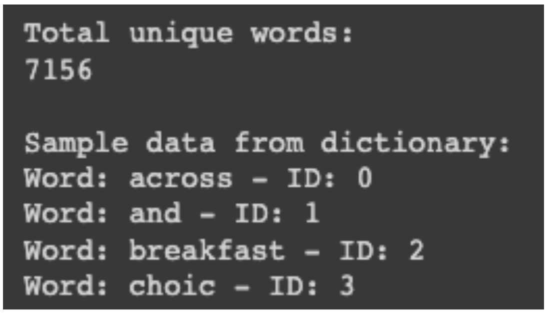
Fig 10. Output following Stemming of tokenised text.

A unique id identified each unique word in the dictionary object. This was used for creating representations of texts. A Bag-of-Words (BOW) corpus was created using this method to build the TF-IDF model. The list of words created a dictionary. This allowed the sentences to be converted to a list of words and then fed to the **corpora. Dictionary** as a parameter:

```
# Building a dictionary

from gensim import corpora
# Build the dictionary
mydict = corpora.Dictionary(df['stemmed_tokens'])
print("Total unique words:")
print(len(mydict.token2id))
print("\nSample data from dictionary:")
i = 0
# Print top 4 (word, id) tuples
for key in mydict.token2id.keys():
    print("Word: {} - ID: {} ".format(key, mydict.token2id[key]))
    if i == 3:
        break
    i += 1
```

After building the dictionary, the report identified a total of 7156 unique words in the ‘stemmed_tokens’ column (Fig.11).



```
Total unique words:
7156

Sample data from dictionary:
Word: across - ID: 0
Word: and - ID: 1
Word: breakfast - ID: 2
Word: choic - ID: 3
```

Fig 11. The output of the dictionary.

NLP Task Implementation: (200 words) 35% 208 w

Feature normalisation

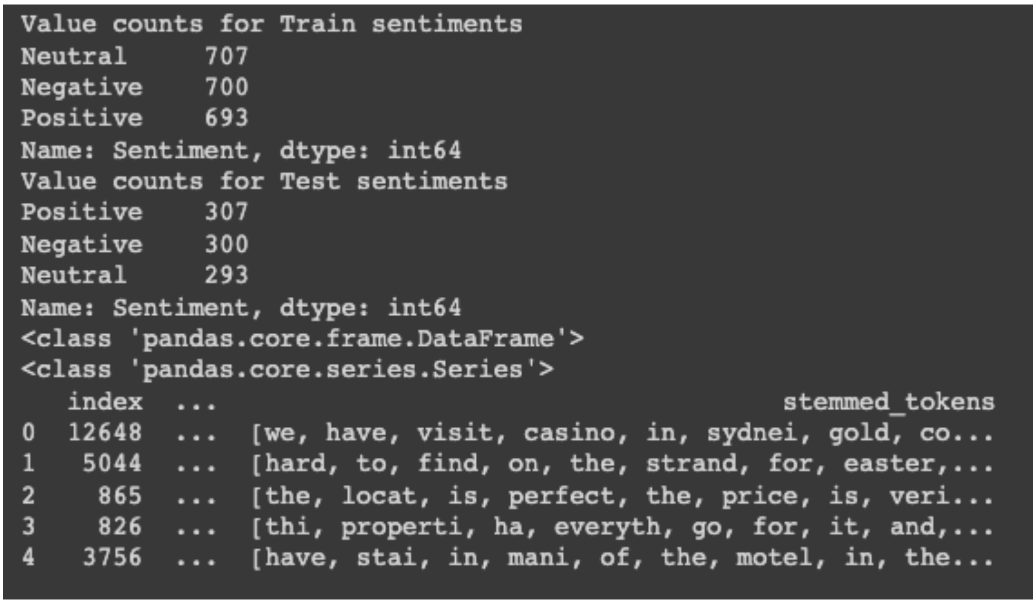
The data was split into training (70%) and test (30%) sets to train the model on a separate data set to the validation data set. The test data is what the model will predict the classes on, and it will be compared with the original labels to check the accuracy and other model performance metrics.

```
# Splitting into train and test sets

from sklearn.model_selection import train_test_split
# Train Test Split Function
def split_train_test(df, test_size=0.3, shuffle_state=True):
    X_train, X_test, Y_train, Y_test = train_test_split(df[['stars', 'stemmed_tokens']],
                                                        df['Sentiment'],
                                                        shuffle = shuffle_state,
                                                        test_size = test_size,
                                                        random_state = 15)
    print("Value counts for Train sentiments")
    print(Y_train.value_counts())
    print("Value counts for Test sentiments")
    print(Y_test.value_counts())
    print(type(X_train))
    print(type(Y_train))
    X_train = X_train.reset_index()
    X_test = X_test.reset_index()
    Y_train = Y_train.to_frame()
    Y_train = Y_train.reset_index()
    Y_test = Y_test.to_frame()
    Y_test = Y_test.reset_index()
    print(X_train.head())
    return X_train, X_test, Y_train, Y_test
```

```
# Call the train_test_split
X_train, X_test, Y_train, Y_test = split_train_test(df)
```

As seen below, the train and test sets both have an even distribution of all three sentiment categories (Fig. 12). This was crucial to prevent model bias.



```
Value counts for Train sentiments
Neutral     707
Negative    700
Positive    693
Name: Sentiment, dtype: int64
Value counts for Test sentiments
Positive    307
Negative    300
Neutral     293
Name: Sentiment, dtype: int64
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
   index  ...                                     stemmed_tokens
0  12648  ...  [we, have, visit, casino, in, sydnei, gold, co...
1   5044  ...  [hard, to, find, on, the, strand, for, easter,...
2    865  ...  [the, locat, is, perfect, the, price, is, veri...
3    826  ...  [thi, properti, ha, everyth, go, for, it, and,...
4   3756  ...  [have, stai, in, mani, of, the, motel, in, the...
```

Fig 12. Output after splitting the data into test and training sets.

TF-IDF is computed by multiplying a local component like term frequency (TF) with a global component, inverse document frequency (IDF), and optionally normalising the result to unit length. The Gensim package was used to create the TF-IDF model. The TF-IDF model was trained, and TF-IDF vectors were generated using the following code:

```
# Created TFIDF model

# Train the tfidf Model
from gensim.models import TfidfModel
# Make sure the dictionary is created from the previous block
# BOW corpus is required for tfidf model
corpus = [mydict.doc2bow(line) for line in df['stemmed_tokens']]

# TF-IDF Model
tfidf_model = TfidfModel(corpus)
# Generating TFIDF vectors
start_time = time.time()
OUTPUT_FOLDER = '/content/drive/My Drive/A3/'

tfidf_filename = OUTPUT_FOLDER + 'train_review_tfidf.csv'
# Storing the tfidf vectors for training data in a file
vocab_len = len(mydict.token2id)
with open(tfidf_filename, 'w+') as tfidf_file:
    for index, row in X_train.iterrows():
        doc = mydict.doc2bow(row['stemmed_tokens'])
        features = gensim.matutils.corpus2csc([tfidf_model[doc]],
num_terms=vocab_len).toarray()[:,0]
        if index == 0:
            header = ",".join(str(mydict[ele]) for ele in range(vocab_len))
            print(header)
            print(tfidf_model[doc])
            tfidf_file.write(header)
            tfidf_file.write("\n")
        line1 = ",".join( [str(vector_element) for vector_element in features] )
        tfidf_file.write(line1)
        tfidf_file.write('\n')
print("Time taken to create tfidf for :" + str(time.time() - start_time))
```

ML algorithm

A decision tree classification model was used to perform the sentiment classification. A decision tree classifier is a supervised machine learning algorithm for classification problems. The **scikit-learn** package was used for implementing the decision tree classifier. The fit function is used to fit the input feature vectors against the sentiments in the train data set. The following code shows how this was achieved:

```
# Training sentiment classification model using TF-IDF vectors

from sklearn.tree import DecisionTreeClassifier
import time
start_time = time.time()

# Read the TFIDF vectors
tfidf_df = pd.read_csv('/content/drive/My Drive/A3/train_review_tfidf.csv')

# Initialize the model
clf_decision_tfidf = DecisionTreeClassifier(random_state=2)

# Fit the model
clf_decision_tfidf.fit(tfidf_df, Y_train['Sentiment'])
print("Time to taken to fit the TF-IDF as input for classifier: " + str(time.time() -
start_time))
```

Hyper-parameters

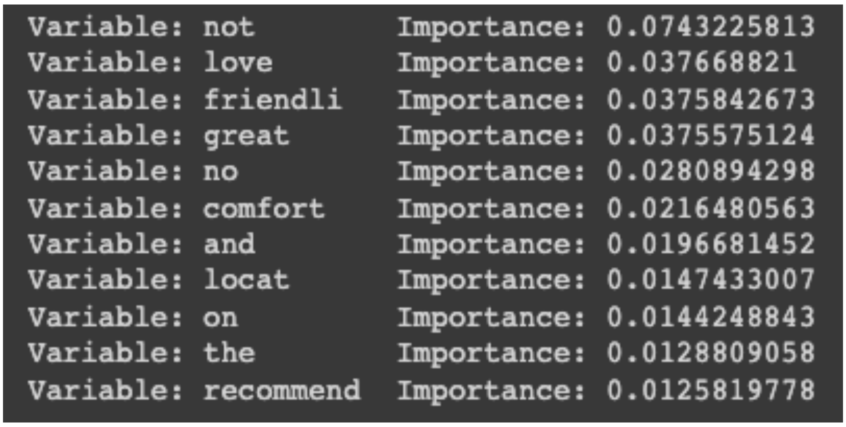
NLP Task Output: (200) 30% 286 w

Quality metrics

The **feature_importances_** attribute was used to get the important features of the model. The value for each feature was produced with the higher the value suggesting more importance (Fig. 13). Getting the important features was implemented via the following code:

```
# Find out the important features from the TFIDF classification model

importances = list(clf_decision_tfidf.feature_importances_)
feature_importances = [(feature, round(importance, 10)) for feature, importance in
zip(tfidf_df.columns, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# print(feature_importances)
top_i = 0
for pair in feature_importances:
    print('Variable: {:10} Importance: {}'.format(*pair))
    if top_i == 10:
        break
    top_i += 1
```



```
Variable: not        Importance: 0.0743225813
Variable: love       Importance: 0.037668821
Variable: friendli   Importance: 0.0375842673
Variable: great      Importance: 0.0375575124
Variable: no         Importance: 0.0280894298
Variable: comfort    Importance: 0.0216480563
Variable: and        Importance: 0.0196681452
Variable: locat      Importance: 0.0147433007
Variable: on         Importance: 0.0144248843
Variable: the        Importance: 0.0128809058
Variable: recommend  Importance: 0.0125819778
```

Fig 13. Ten most important features of the TFIDF classification model.

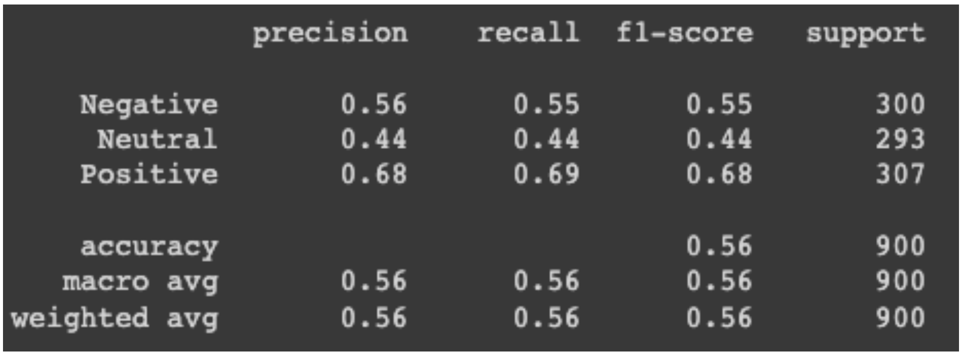
Fig. 13 displays the three most important features: ' not', 'love' and 'friendli', respectively.

Summary of outputs

As shown below, the decision tree classifier generates an overall accuracy of 56% (Fig 14). The model appears to predict the 'positive' sentiment class more accurately. This may be due to the slightly higher number of observations.

```
# Testing the model

from sklearn.metrics import classification_report
test_features_tfidf = []
import time
start_time = time.time()
for index, row in X_test.iterrows():
    doc = mydict.doc2bow(row['stemmed_tokens'])
    features = gensim.matutils.corpus2csc([tfidf_model[doc]],
num_terms=vocab_len).toarray()[:,0]
    test_features_tfidf.append(features)
test_predictions_tfidf = clf_decision_tfidf.predict(test_features_tfidf)
print(classification_report(Y_test['Sentiment'],test_predictions_tfidf))
print("Time taken to predict using TF-IDF:" + str(time.time() - start_time))
```



```
              precision    recall  f1-score   support

    Negative       0.56      0.55      0.55       300
     Neutral       0.44      0.44      0.44       293
    Positive       0.68      0.69      0.68       307

    accuracy                           0.56       900
   macro avg       0.56      0.56      0.56       900
weighted avg       0.56      0.56      0.56       900
```

Fig 14. Summary of model performance.

The model's precision, also known as positive pred value, is 68% for the 'positive' class. This means the model correctly predicted 68% of the ‘positive’ class out of all the predicted ‘positive’ class observations. The recall, also known as sensitivity, was 55% for the ‘negative’ class. This means the model correctly 55% of the 'negative' class out of the actual 'negative' class observations. The F1 score can be interpreted as the harmonic mean of precision and recall. The measure is between 0-1, with a score closer to 1 representing a higher performance. The F1 score for the ‘positive’ class is 68%. The accuracy of the model from the test data is 56%. This means the model correctly classified 56% of the total 3000 observations.

Visualise outputs

NLP task alignment with issue

By performing Sentiment Analysis, hotels can better understand the sentiment of the customer review without relying on the overall customer rating. The overall customer rating can be limiting in expressing the various aspects of a customer's experience. Hotels could improve their customer experience by using a sentiment analyser to expand their knowledge of customer reviews on TripAdvisor.