Accommodation Advisory Tool

A Machine Learning Project using Natural Language Processing

Introduction

Finding the right accommodation for a trip can be a challenging and time-consuming task. There are many factors to consider, such as location, price, amenities, reviews, ratings, and availability. Moreover, each traveller has different preferences and expectations, which may not be fully captured by the existing online platforms.

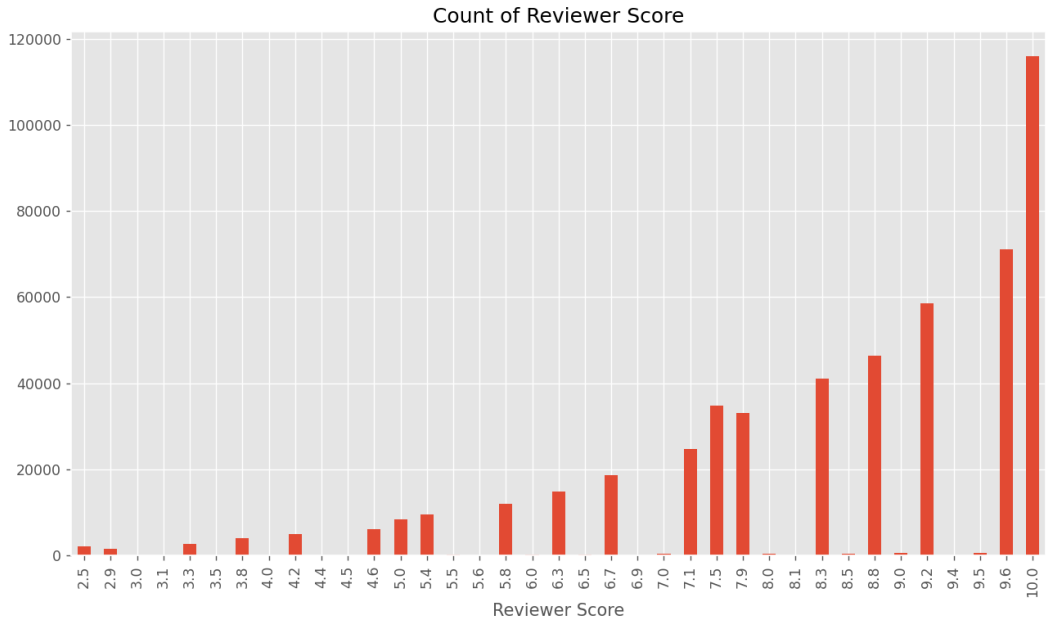
To address this problem, we propose to develop an accommodation advisory tool that uses machine learning, specifically natural language processing (NLP), to analyse customer reviews and ratings

Data Analysis

The first step of the project is to perform data analysis on the dataset of customer reviews and ratings for various hotels. The dataset contains information such as hotel name and address, location, price, review text, review score, and reviewer nationality. We use pandas and numpy to load and manipulate the data, and data visualisation to explore the data and gain insights.

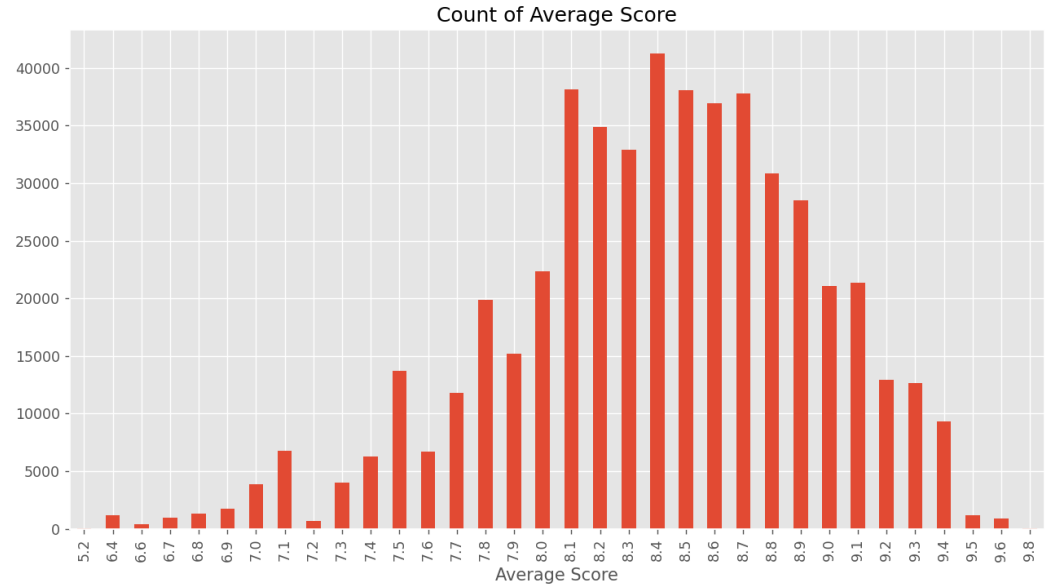
To understand the frequency of the different review scores, we found the most common reviewer scores and their counts using the pandas `value_counts` method. We then displayed them using data visualisation, with the review scores as the labels and the counts as the sizes. It hows the proportion of the different review scores, and helps us see how satisfied or dissatisfied the customers were with their accommodation.

```
# Count of Reviewer Score
ax = df['Reviewer_Score'].value_counts().sort_index().plot(kind='bar', title='C
ax.set_xlabel('Reviewer Score')
plt.show()
```



This chart shows the distribution of hotel reviews by users, categorized by reviewer score. The reviewer score ranges from 0 to 10, with 0 being the lowest and 10 being the highest.
The chart indicates that most users were satisfied with their hotel experience, as the majority of the reviews have a high score. In fact, the most common reviewer score is 10, which means that many users gave a perfect rating to their hotel. This suggests that the hotels in the dataset have a good reputation and quality.
However, there are also some users who gave low scores, indicating that they were dissatisfied or had a negative experience.
The chart does not show the reasons for the different scores, which will be explored further by analysing the review texts soon.

```
# Count of Average Score
ay = df['Average_Score'].value_counts().sort_index().plot(kind='bar', title='Co
ay.set_xlabel('Average Score')
plt.show()
```



Based on the two charts, I can draw the following conclusions:

- The average scores for the hotels are generally high, as most of them are above 8.0. This means that the hotels have a good quality and reputation, and most customers are satisfied with their accommodation.
- The distribution of the average scores is skewed to the right, with a peak around 8.2. This indicates that there is a high variability in the average scores, and some hotels have exceptionally high or low ratings.
- The most common reviewer score is 10, which is much higher than the most common average score. This suggests that there is a discrepancy between the individual reviews and the aggregated scores, and some factors may influence the customers' ratings, such as expectations, preferences, or biases.

Sentiment Analysis with VADER

Before training our own model for sentiment analysis, we decided to use a pre-trained model called VADER (Valence Aware Dictionary and sEntiment Reasoner). VADER is a lexicon and rule-based tool that can analyse the sentiment of text, especially social media text. It can assign a polarity score to each word, phrase, or sentence, ranging from -4 (very negative) to 4 (very positive). It can also handle negations, intensifiers, emoticons, and slang.

We used VADER to perform sentiment analysis on the review texts, and compared the results with the review scores. We expected to see a positive correlation between the VADER scores and the review scores, meaning that higher review scores would correspond to more positive sentiments, and vice versa. We also expected to see some outliers or discrepancies, where the VADER scores and the review scores would not match, due to factors such as sarcasm, irony, or exaggeration.

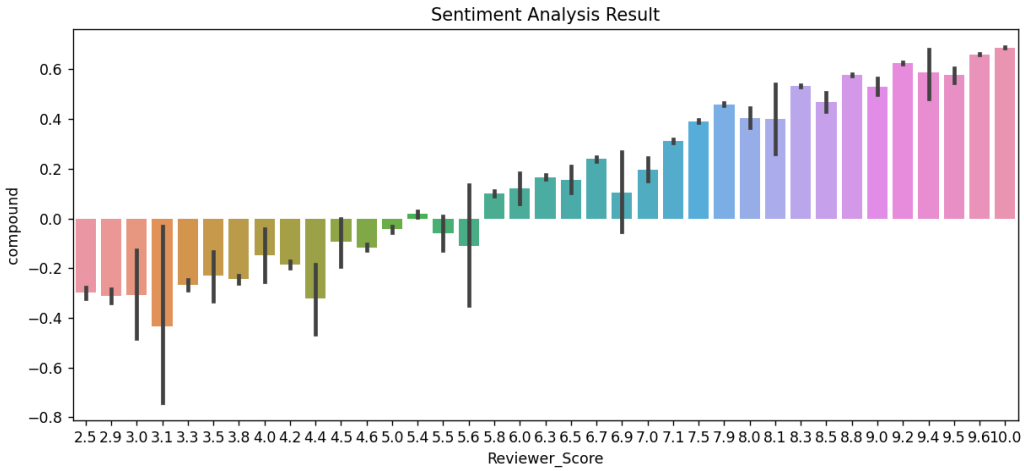
```
from matplotlib import pyplot as plt
from nltk.sentiment import SentimentIntensityAnalyzer
from src.data_loader import DataLoader
import seaborn as sns
# Load the data
data = DataLoader().load_data()

# Initialize the SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Remove 'No Negative' or 'No Positive' from text
data['Positive_Review'] = data['Positive_Review'].apply(lambda x: x.replace("No
data['Negative_Review'] = data['Negative_Review'].apply(lambda x: x.replace("No

# Create a new column 'Review' that contains the positive and negative review t
data['Review'] = data['Positive_Review'] + data['Negative_Review']
columns_to_drop = ['Hotel_Address', 'Additional_Number_of_Scoring', 'Review_Dat
                   'Total_Number_of_Reviews', 'Average_Score', 'Tags', 'days_si
                   'Total_Number_of_Reviews_Reviewer_Has_Given', 'days_since_re

data.drop(columns=columns_to_drop, inplace=True)
# Apply the SentimentIntensityAnalyzer to the 'Negative_Review' column
sentiment_scores = data['Review'].apply(sia.polarity_scores)

# Add the sentiment scores as new columns in the dataframe
data['neg'] = sentiment_scores.apply(lambda x: x['neg'])
data['pos'] = sentiment_scores.apply(lambda x: x['pos'])
data['neu'] = sentiment_scores.apply(lambda x: x['neu'])
data['compound'] = sentiment_scores.apply(lambda x: x['compound'])
```

- The first four lines import the libraries that are needed for the code. `matplotlib` is used for data visualisation, `nltk` is used for natural language processing, `src.data_loader` is a custom module that loads the data, and `seaborn` is used for statistical data visualisation.
- The next line calls the `load_data` method from the `DataLoader` class, which returns a pandas dataframe containing the hotel reviews and ratings. The dataframe is assigned to the variable `data`.
- The next line creates an instance of the `SentimentIntensityAnalyzer` class from the `nltk.sentiment` module, which is a pre-trained model for sentiment analysis. The instance is assigned to the variable `sia`.
- The next three lines remove the strings 'No Negative' or 'No Positive' from the columns 'Positive_Review' and 'Negative_Review', which indicate that the customer did not leave any positive or negative feedback. This is done by applying a lambda function that replaces the strings with an empty string to each column.
- The next line creates a new column called 'Review' that contains the concatenated text of the positive and negative reviews. This is done by adding the two columns together.
- The next line defines a list of column names that are not relevant for the sentiment analysis, such as the hotel address, the review date, the reviewer nationality, etc. These columns are assigned to the variable `columns_to_drop`.
- The next line drops the columns in the `columns_to_drop` list from the dataframe, using the `drop` method with the `inplace` parameter set to `True`. This modifies the original dataframe and reduces its dimensionality.
- The next line applies the `polarity_scores` method from the `sia` instance to the 'Review' column, which returns a dictionary of sentiment scores for each review. The scores

are 'neg' (negative), 'pos' (positive), 'neu' (neutral), and 'compound' (overall). The method is applied using the `apply` method from the dataframe. The result is assigned to the variable `sentiment_scores`.

- The next four lines add the sentiment scores as new columns in the dataframe, using the `apply` method again. The lambda function extracts the corresponding score from the dictionary and assigns it to the column. The new columns are 'neg', 'pos', 'neu', and 'compound'.

```
ax = sns.barplot(x='Reviewer_Score', y='compound', data=data)
ax.set_title('Sentiment Analysis Result')
plt.show()

# New barplot for pos, neg and neu
ax = sns.barplot(x='Reviewer_Score', y='pos', data=data)
ax.set_title('Positive Sentiment')
plt.show()

ax = sns.barplot(x='Reviewer_Score', y='neg', data=data)
ax.set_title('Negative Sentiment')
plt.show()

ax = sns.barplot(x='Reviewer_Score', y='neu', data=data)
ax.set_title('Neutral Sentiment')
plt.show()
```

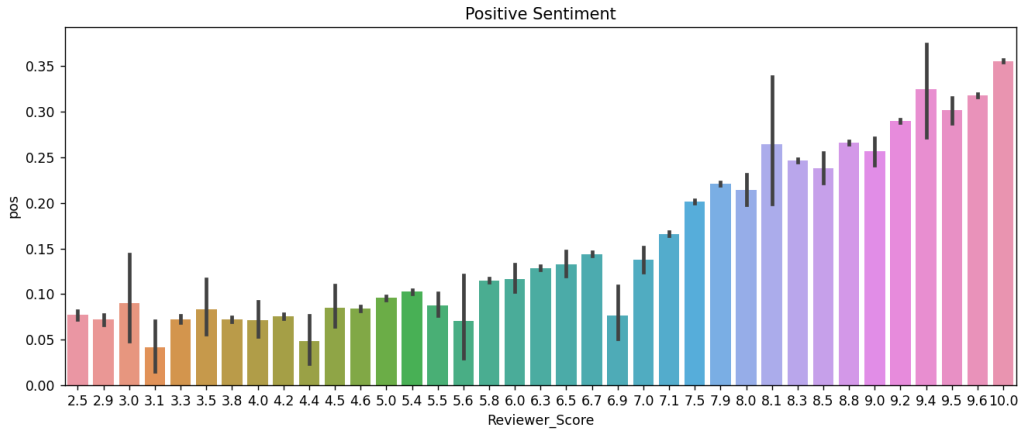


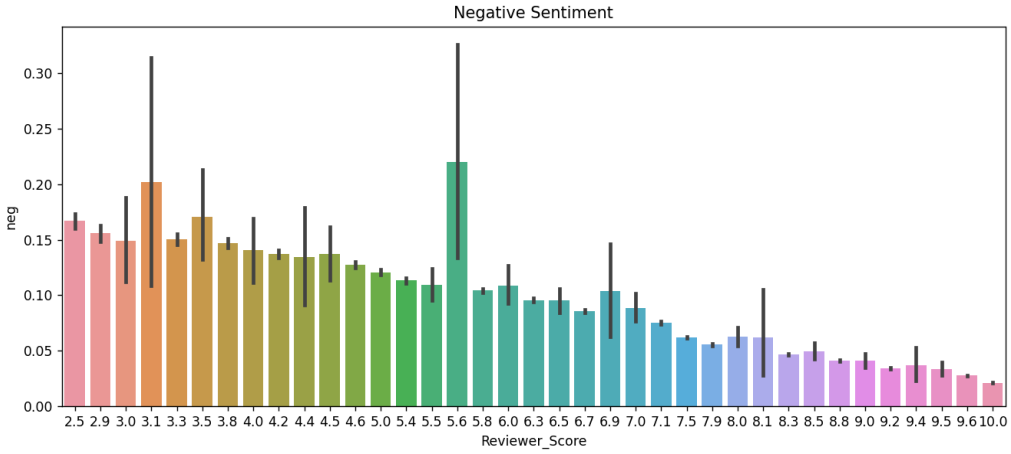
This chart shows the relationship between the reviewer score and the compound sentiment score for hotel reviews. The reviewer score is the rating given by the customer to the hotel, ranging from 2.5 to 10.0.

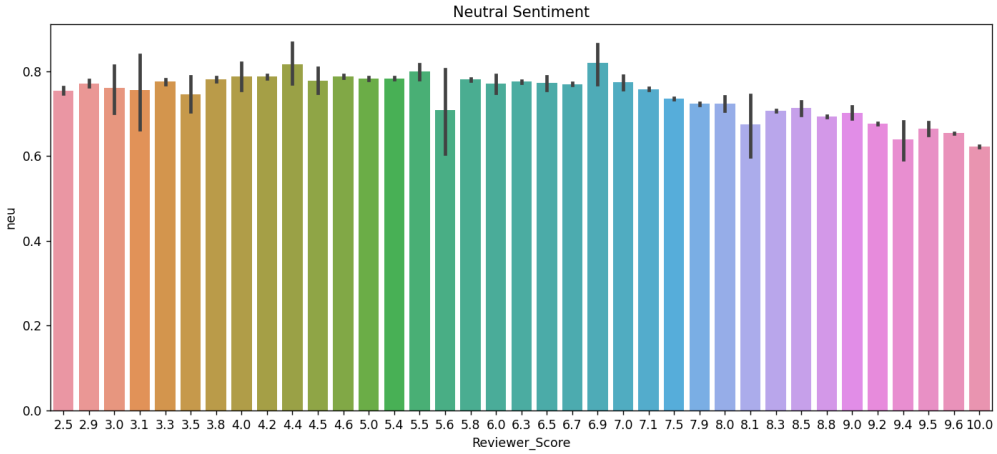
The compound sentiment score is the overall sentiment of the review text, calculated by VADER, ranging from -0.8 to 0.6. The chart indicates that there is a positive correlation between the reviewer score and the compound sentiment score, meaning that higher reviewer scores are associated with more positive sentiments, and lower reviewer scores are associated with more negative sentiments.

This suggests that VADER is able to capture the general sentiment of the customers based on their review texts. However, there are a few exceptions, but nothing too much exaggerated.

If we look at the different sentiment fields







We found a good correlation, however, there are some more visible exceptions that could be

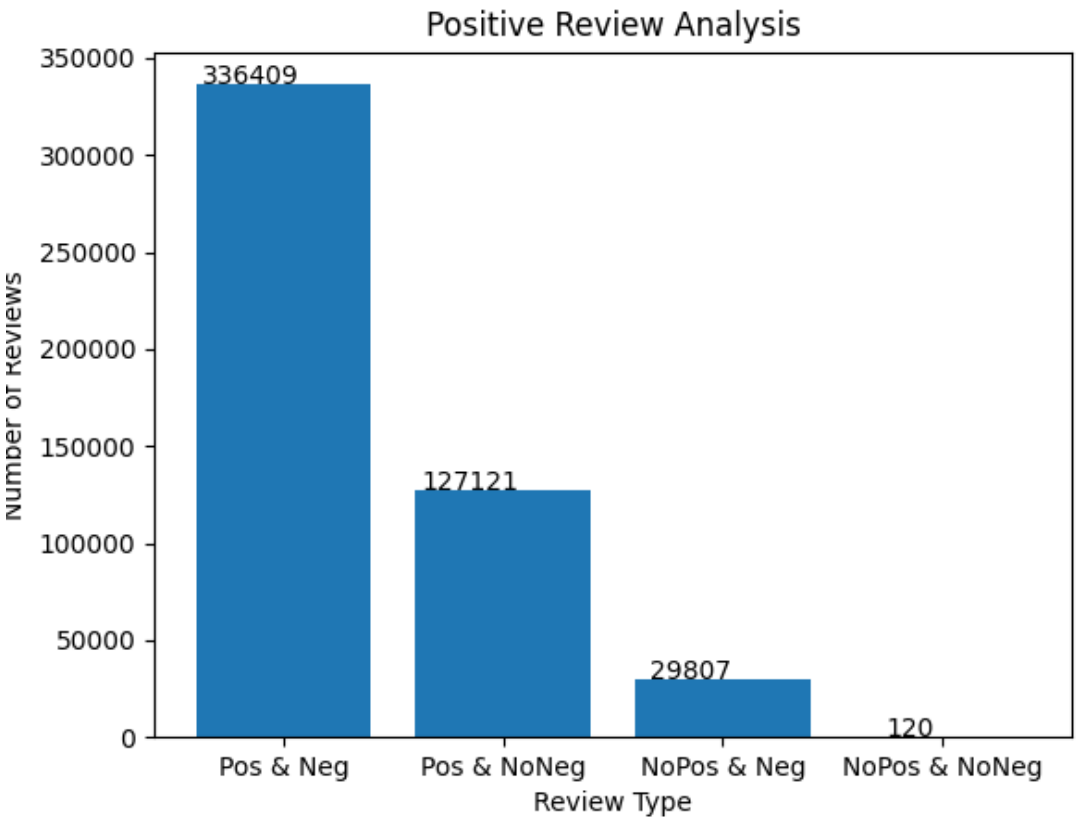
```
import matplotlib.pyplot as plt

from src.data_loader import DataLoader

data = DataLoader().load_data()
if data is not None:
    positive_reviews = data[data['Reviewer_Score'] > 5]

    # Define conditions for different types of reviews
    positive_and_negative = positive_reviews[(positive_reviews['Positive_Review
                                              (positive_reviews['Negative_Review
    positive_no_negative = positive_reviews[(positive_reviews['Positive_Review'
                                             (positive_reviews['Negative_Review'
    no_positive_negative = positive_reviews[(positive_reviews['Positive_Review'
                                             (positive_reviews['Negative_Review'
    no_positive_no_negative = positive_reviews[(positive_reviews['Positive_Revi
                                                (positive_reviews['Negative_Revi

    # Count the number of reviews for each category
    num_positive_and_negative = len(positive_and_negative)
    num_positive_no_negative = len(positive_no_negative)
    num_no_positive_negative = len(no_positive_negative)
    num_no_positive_no_negative = len(no_positive_no_negative)

    # Plotting
    labels = ['Positive and Negative', 'Positive but No Negative', 'No Positive
              'No Positive and No Negative']
    values = [num_positive_and_negative, num_positive_no_negative, num_no_posit
              num_no_positive_no_negative]

    fig, ax = plt.subplots()

    # Plot the bars
    bars = plt.bar(labels, values)

    # Add count labels on top of the bars
    for bar, value in zip(bars, values):
        plt.text(bar.get_x() + bar.get_width() / 2 - 0.15, bar.get_height() + 0

    plt.title('Positive Reviews Analysis')
    plt.xlabel('Review Type')
    plt.ylabel('Number of Reviews')
    plt.show()
```

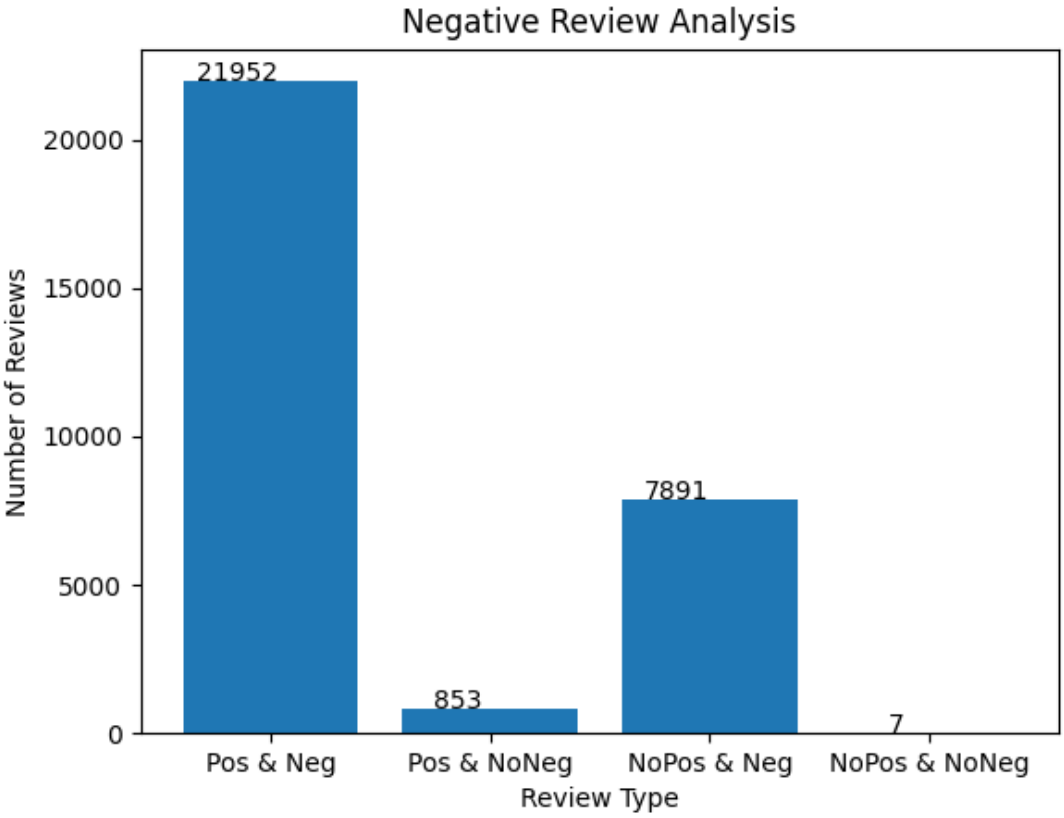
- The first two lines import the `matplotlib.pyplot` module as `plt` and the `src.data_loader` module as `DataLoader`. These modules are used for data visualisation and data loading respectively.
- The next line calls the `load_data` method from the `DataLoader` class, which returns a pandas dataframe containing the hotel reviews and ratings. The dataframe is assigned to the variable `data`.
- The next line checks if the `data` variable is not `None`, meaning that the data loading was successful. If so, the code proceeds to the next block.
- The next line filters the dataframe to keep only the positive reviews, which are those with a reviewer score greater than 5. The filtered dataframe is assigned to the variable

`positive_reviews`.

- The next four lines define four different types of reviews based on the presence or absence of positive and negative feedback. The code uses boolean indexing to select the rows that match the conditions. The four types of reviews are:
  - Positive and Negative: The reviews that have both positive and negative feedback.
  - Positive but No Negative: The reviews that have only positive feedback and no negative feedback.
  - No Positive but Negative: The reviews that have only negative feedback and no positive feedback.
  - No Positive and No Negative: The reviews that have neither positive nor negative feedback.

- The next four lines count the number of reviews for each type and assign them to four variables: `num_positive_and_negative`, `num_positive_no_negative`, `num_no_positive_negative`, and `num_no_positive_no_negative`.
- The next line creates a figure and an axes object using the `plt.subplots` function. These objects are used for plotting the bar chart.
- The next line defines the labels and values for the bar chart. The labels are the four types of reviews, and the values are the corresponding counts.
- The next line plots the bar chart using the `plt.bar` function, passing the labels and values as arguments.
- The next three lines add count labels on top of the bars using the `plt.text` function. The function takes the x and y coordinates, the text, and the horizontal alignment as arguments. The x and y coordinates are calculated from the bar positions and heights, and the text is the count value. The horizontal alignment is set to 'center' to align the text with the bars.
- The next three lines add a title, an x-label, and a y-label to the bar chart using the `plt.title`, `plt.xlabel`, and `plt.ylabel` functions respectively. The title is 'Positive Reviews Analysis', the x-label is 'Review Type', and the y-label is 'Number of Reviews'.
- The last line shows the bar chart using the `plt.show` function.





As we can see, most users, both in the worst reviews and in the best rated, usually choose to leave a positive and negative review. But we also find certain anomalies, there are 853 negative reviews that leave positive but not negative reviews.
This might be considered insignificant due to the sample, but as soon as we train the machine we will see that it can lead to problems.
Of course, the anomaly is much more blatant in positive reviews where we found a total of 29807 positive reviews that only published negative reviews. The percentage is not huge, but it

can lead to confusion.

The word cloud is a visual representation of the most common words in hotel reviews, both positive and negative. The size of each word indicates its frequency, and the color varies from green-to-blue tones.
The word cloud shows that many customers appreciate the hotel's location, staff, and breakfast, as these words are large and prominent. However, there are also some words that suggest dissatisfaction or complaints, such as "useless" and "complaint".
The word cloud also includes some general terms related to hotels, such as "room", "building", and "park". The word cloud gives an overview of the main topics and sentiments expressed by the customers in their reviews.

Data access

To access the data in the CSV dataset, it was simply necessary to use pandas and a load class to facilitate access from anywhere in the project.

```python
import pandas as pd
import os

DATA_PATH = os.path.join('..', 'data', 'Accommodation_Reviews.csv')


class DataLoader:
    def __init__(self):
        if DATA_PATH is None:
            raise Exception("File path cannot be None")
        self.file_path = DATA_PATH

    def load_data(self):

        # Load the dataset
        data = pd.read_csv(self.file_path)

        # Data Sample of 10% of the dataset to reduce computational cost
        data = data.sample(frac=0.1, replace=False, random_state=93)
        return data
```

It is worth pointing out the use of `sample()` to avoid going through annoying computation times in some sections

Data pre-procedure

The code consists of four functions that perform different tasks:

- `impute` : This function is used to impute the tags column, which contains a list of tags for each hotel review. The function converts the string representation of the list to an actual list, and returns it.
- `preprocess_data` : This function is used to preprocess the data, which is a pandas dataframe containing the hotel reviews and ratings. The function performs several steps, such as removing not useful columns, replacing 'United Kingdom' with 'UK', splitting the country from the hotel address, converting the tags to a list, and lowercasing the countries and tags. The function returns the preprocessed dataframe.
- `get_wordnet_pos` : This function is used to get the part-of-speech (POS) tag of a word, based on the POS tag given by the `pos_tag` function from the `nltk` module. The function maps the POS tag to the corresponding tag in the WordNet lexicon, which is used for lemmatization. The function returns the WordNet POS tag.
- `clean_text` : This function is used to clean the text data, which is the string containing the hotel review. The function performs several steps, such as lowercasing the text, tokenizing the text and removing punctuation, removing words that contain numbers, removing stop words, removing empty tokens, lemmatizing the text, and removing words with only one letter. The function returns the cleaned text.

These functions are essential for preparing the data for further analysis and modeling, such as sentiment analysis, topic modeling, or recommendation systems.

```
import string
from ast import literal_eval

from nltk import WordNetLemmatizer, pos_tag
from nltk.corpus import wordnet, stopwords

lemm = WordNetLemmatizer()


def impute(column):
    """
        Used to impute the tags column

        :param column:
        :return: column after imputing:
    """
    column = column.iloc[0]  # Use iloc for positional indexing
    if not isinstance(column, list):
        return "".join(literal_eval(column))
    else:
        return column


def preprocess_data(data):
    """
        Preprocess the data
        :param data: the dataset to be preprocessed
        :return data after preprocessing:
    """
    # Remove not useful columns
    columns_to_drop = ['Additional_Number_of_Scoring',
                       'Review_Date', 'Reviewer_Nationality',
                       'Negative_Review', 'Review_Total_Negative_Word_Counts',
                       'Total_Number_of_Reviews', 'Positive_Review',
                       'Review_Total_Positive_Word_Counts',
                       'Total_Number_of_Reviews_Reviewer_Has_Given', 'Reviewer
                       'days_since_review', 'lat', 'lng']
    data.drop(columns=columns_to_drop, inplace=True)

    # United Kingdom is the same as UK
    data['Hotel_Address'] = data['Hotel_Address'].str.replace('United Kingdom'

    # Split country from hotel address, so we can easily work with it
    data["countries"] = data.Hotel_Address.apply(lambda x: x.split(' ')[-1])

    # String 'list' of tags to an actual list
    data["Tags"] = data[["Tags"]].apply(impute, axis=1)

    # Lowercase countries and tags
    data['countries'] = data['countries'].str.lower()
    data['Tags'] = data['Tags'].str.lower()
    return data


def get_wordnet_pos(pos_tag_text):
    """
    Used to get the pos tag of the word
```

```
    :param pos_tag_text: the pos tag of the word
    :return pos_tag:
    """
    if pos_tag_text.startswith('J'):
        return wordnet.ADJ
    elif pos_tag_text.startswith('V'):
        return wordnet.VERB
    elif pos_tag_text.startswith('N'):
        return wordnet.NOUN
    elif pos_tag_text.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


def clean_text(text):
    """
        Clean the text data
        1. Lowercase the text
        2. Tokenize the text
        3. Remove punctuation
        4. Remove words that contain numbers
        5. Remove stop words
        6. Remove empty tokens
        7. Lemmatize text
        8. Remove words with only one letter

        :param text: the string to be cleaned
        :return text: the cleaned string
    """
    # lower text
    text = text.lower()
    # tokenize text and remove punctuation
    text = [word.strip(string.punctuation) for word in text.split(" ")]
    # remove words that contain numbers
    text = [word for word in text if not any(c.isdigit() for c in word)]
    # remove stop words
    stop = stopwords.words('english')
    text = [x for x in text if x not in stop]
    # remove empty tokens
    text = [t for t in text if len(t) > 0]
    pos_tags = pos_tag(text)
    # lemmatize text
    text = [WordNetLemmatizer().lemmatize(t[0], get_wordnet_pos(t[1])) for t i
    # remove words with only one letter
    text = [t for t in text if len(t) > 1]
    # join all
    text = " ".join(text)
    return text
```

We'll see about its uses soon

Sentiment Predictor with machine learning

The application of machine learning to predict the sentiment of a hotel review is the process of using algorithms and data to automatically classify the emotional tone of a customer's feedback. This can help hotel owners, managers, and customers to understand the strengths and weaknesses of the hotel's services, facilities, and amenities, as well as the customers' satisfaction and expectations. Machine learning can also help to generate insights and recommendations based on the sentiment analysis results.

The first step is to adapt the data to our needs

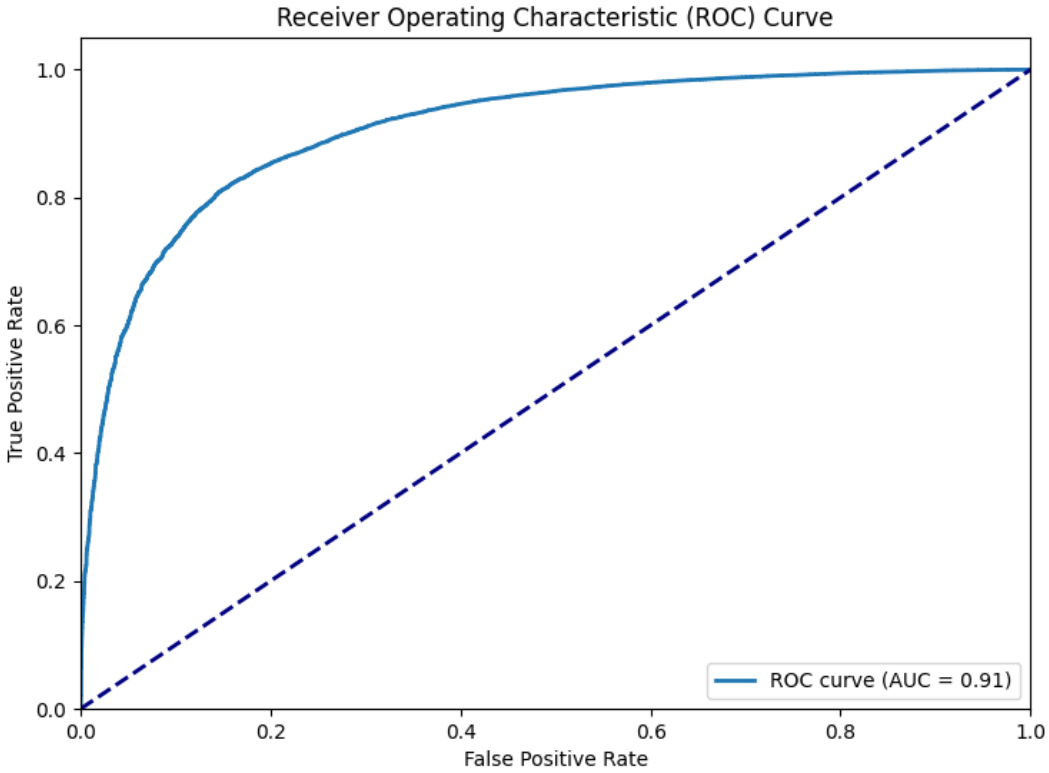
```
import time

from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_curve, roc_auc_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

from src.data_loader import DataLoader
from src.data_preprocessor import clean_text

MIN_SCORE = 5
current_time = time.time()

# Load the data
data = DataLoader().load_data()
print("Data Loaded in ", time.time() - current_time, " seconds")

# Remove 'No Negative' or 'No Positive' from the reviews
data['Negative_Review'] = data['Negative_Review'].apply(lambda x: x.replace('No
data['Positive_Review'] = data['Positive_Review'].apply(lambda x: x.replace('No

# Combine positive and negative reviews into a single column
data['Review'] = data['Positive_Review'] + data['Negative_Review']
print("Reviews Combined in ", time.time() - current_time, " seconds")

# Clean the reviews
data['Review'] = data['Review'].apply(lambda x: clean_text(x))
print("Reviews Cleaned in ", time.time() - current_time, " seconds")

# Just keep the positive and negative reviews, count of words in each review an
data = data[['Review', 'Review_Total_Positive_Word_Counts',
             'Review_Total_Negative_Word_Counts', 'Reviewer_Score']]

data['Is_Good_Review'] = data['Reviewer_Score'].apply(lambda x: 1 if x > MIN_SC
print("Data Processed\nStarting training")
```

As you can see, the text is adapted for a better understanding of the machine. With the Review columns, good and bad words counted, and the review score itself, we can give the machine the necessary information to start its training.

Of course, this whole process is carried out with execution times in mind so that you know which parts take the longest

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data[['Review', 'Review_Tot
                                                         'Review_Total_Negativ
                                                    test_size=0.3, random_state

# Create a pipeline for text and numeric features
preprocessor = ColumnTransformer(
    transformers=[
        ('text', CountVectorizer(), 'Review'),
        ('num', StandardScaler(), ['Review_Total_Positive_Word_Counts', 'Review
    ])

# Create a pipeline with our preprocessor and classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))  # Increase max_iter to 1
])

# Train the model
pipeline.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = pipeline.predict(X_test)
print("Training Complete in ", time.time() - current_time, " seconds")
# Print accuracy score
print('Accuracy: ', pipeline.score(X_test, y_test))

# Print the classification report
print(classification_report(y_test, y_pred))

# Confusion Matrix
# disp = plot_confusion_matrix(pipeline, X_test, y_test, cmap=plt.cm.Blues, nor
# disp.ax_.set_title('Confusion Matrix')
# plt.show()

# ROC Curve and AUC
y_prob = pipeline.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
print("END in ", time.time() - current_time, " seconds")
```

The output of this code results in a table and a ROC (Receiver Operating Characteristic) curve, which is a graphical representation of the performance of a binary classification model for all classification thresholds. It is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the ROC curve (AUC) is a measure of the model's performance.



The table has four columns: precision, recall, f1-score, and support. The table has three rows: one for each class (0 for negative, 1 for positive), and one for the average values. The table also shows the accuracy of the model, which is the proportion of correct predictions.

The table indicates that the model performs well in predicting the positive class, as it has high values of precision, recall, and f1-score. However, the model struggles with the negative class, as it has low values of recall and f1-score. This means that the model misses many negative reviews and misclassifies them as positive. The table also shows that the positive class has much more support than the negative class, meaning that there are more positive reviews than negative reviews in the data. This could explain why the model is biased towards the positive class.
The table suggests that the model could be improved by using a more balanced dataset or applying some techniques to handle the class imbalance.

The code produced is:

```
'''
Data Loaded in  2.0414106845855713  seconds
Reviews Combined in  2.332226037979126  seconds
Reviews Cleaned in  577.7847013473511  seconds
Data Processed
Starting training
Training Complete in  591.0267570018768  seconds
Accuracy:  0.9471956153617456
              precision    recall  f1-score   support

           0       0.62      0.29      0.40      9223
           1       0.96      0.99      0.97    145499

    accuracy                           0.95    154722
   macro avg       0.79      0.64      0.69    154722
weighted avg       0.94      0.95      0.94    154722

END in  682.8744542598724  seconds

Process finished with exit code 0

'''
```

Recommendation tool

The goal of a hotel recommendation system is to suggest the most suitable hotel for a user from a pool of hotels. To create such a system that can assist the user in finding the best hotel for their needs, we can use customer reviews as a source of information. This dataset includes hotel data from six countries, which are:

- Netherlands
- United Kingdom
- France
- Spain
- Italy
- Austria

To simplify the analysis, I will rename “United Kingdom” to “UK. I also notice that there is no “Country” column to indicate the location of the hotel, but the last word in the “Hotel_Address” column is the country name.

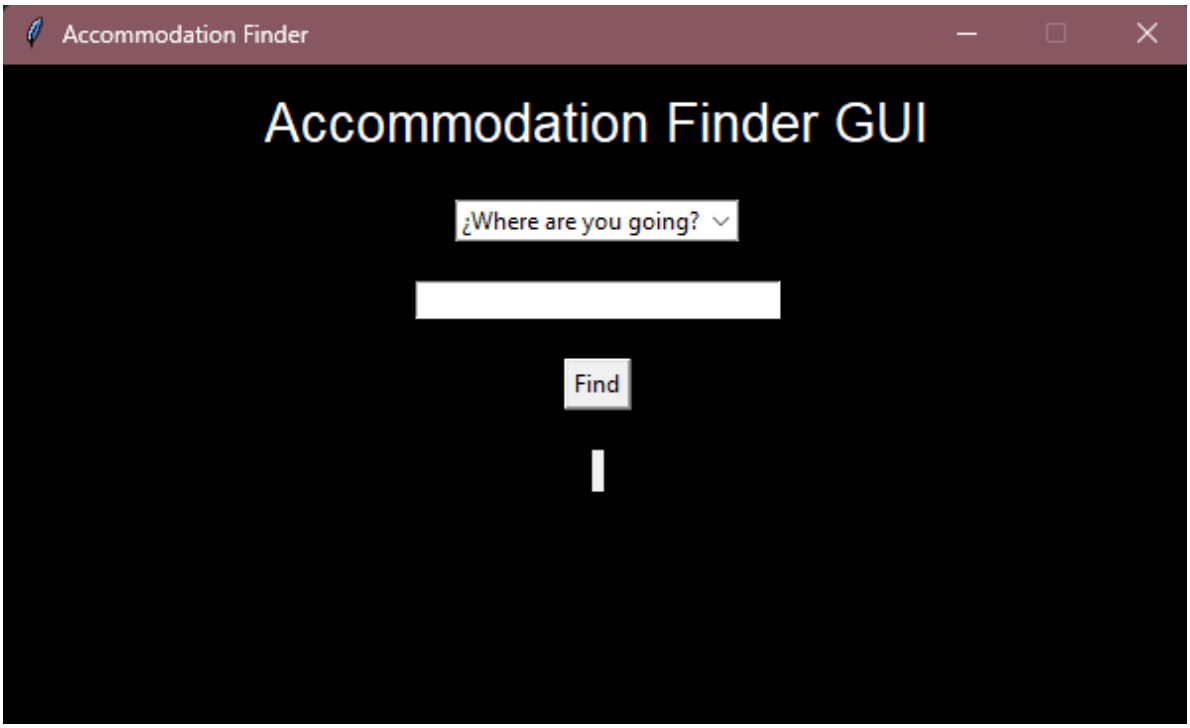
- I will extract the country names from that column and create a new column for them.
- I will also create a function to convert the strings of lists into a normal list and then apply it to the “Tags” column in the dataset.
- I will lowercase the “Tags” and “countries” column for simplicity

```
def preprocess_data(data):
    """
        Preprocess the data
        :param data: the dataset to be preprocessed
        :return data after preprocessing:
    """
    # Remove not useful columns
    columns_to_drop = ['Additional_Number_of_Scoring',
                       'Review_Date', 'Reviewer_Nationality',
                       'Negative_Review', 'Review_Total_Negative_Word_Counts',
                       'Total_Number_of_Reviews', 'Positive_Review',
                       'Review_Total_Positive_Word_Counts',
                       'Total_Number_of_Reviews_Reviewer_Has_Given', 'Reviewer_
                       'days_since_review', 'lat', 'lng']
    data.drop(columns=columns_to_drop, inplace=True)

    # United Kingdom is the same as UK
    data['Hotel_Address'] = data['Hotel_Address'].str.replace('United Kingdom',

    # Split country from hotel address, so we can easily work with it
    data["countries"] = data.Hotel_Address.apply(lambda x: x.split(' ')[-1])

    # String 'list' of tags to an actual list
    data["Tags"] = data[["Tags"]].apply(impute, axis=1)

    # Lowercase countries and tags
    data['countries'] = data['countries'].str.lower()
    data['Tags'] = data['Tags'].str.lower()
    return data
```

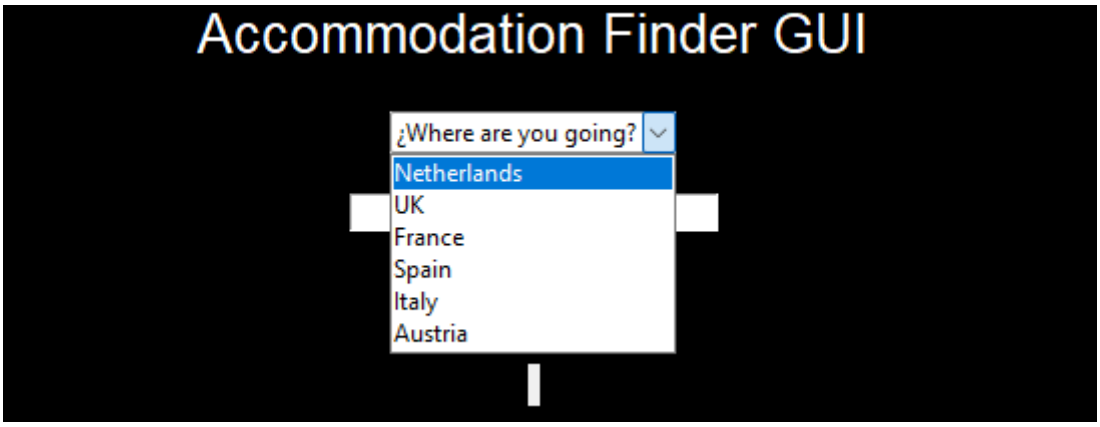
Now let's define a function to recommend the names of hotels according to the location and the description provided by the user. Here our aim is not just to recommend the name of the hotel but also rank it according to the user ratings:

```
import numpy as np
from nltk import word_tokenize, WordNetLemmatizer
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))
lemm = WordNetLemmatizer()


def recommend_hotel(data, location, description):
    # Tokenize the description
    description_tokens = word_tokenize(description.lower())

    # Remove stop words and lemmatize
    filtered = {word for word in description_tokens if word not in stop_words}
    filtered_set = set([lemm.lemmatize(word) for word in filtered])

    # Filter the data by location
    country = data[data['countries'] == location.lower()]
    country = country.set_index(np.arange(country.shape[0]))
    cos = []

    # Calculate the similarity between the description and the tags
    for i in range(country.shape[0]):

        # Tokenize the tags
        temp_token = word_tokenize(country["Tags"][i])
        temp_set = [word for word in temp_token if word not in stop_words]
        temp2_set = set()

        # Lemmatize the tags
        for s in temp_set:
            temp2_set.add(lemm.lemmatize(s))
        vector = temp2_set.intersection(filtered_set)
        cos.append(len(vector))

        # Append the hotel name and the score
    country['similarity'] = cos
    country = country.sort_values(by='similarity', ascending=False)
    country.drop_duplicates(subset='Hotel_Name', keep='first', inplace=True)
    country.sort_values('Average_Score', ascending=False, inplace=True)
    country.reset_index(inplace=True)
    return country[["Hotel_Name", "Average_Score"]].head()
```

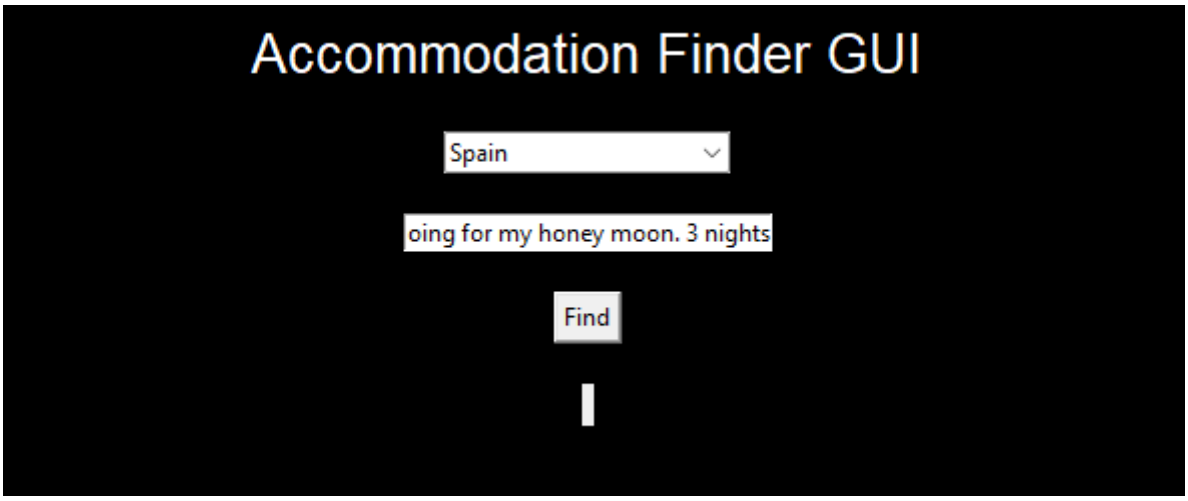
To test the functionality of this tool, I have implemented a visual interface
Let's try it with 'Spain' for the country and 'I am going for my honey moon 3 nights'



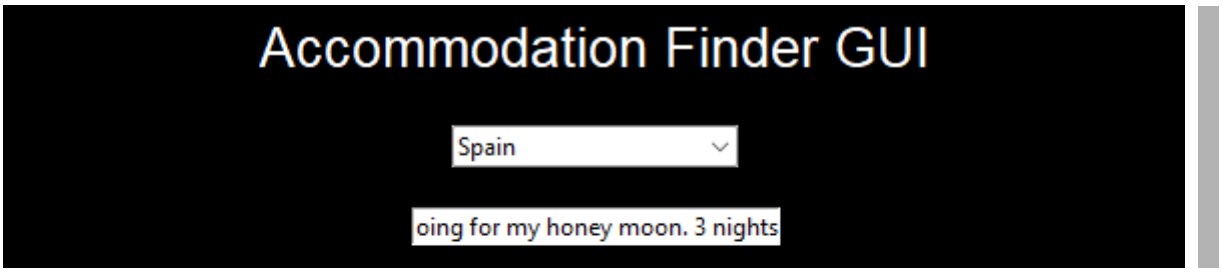
> Main page



> Country ComboBox



> Text field for specific requests


Accommodation Finder GUI
Spain
oing for my honey moon. 3 nights

```python
import tkinter as tk
from tkinter import ttk

from src.data_loader import DataLoader
from src.data_preprocessor import preprocess_data
from src.model import recommend_hotel

# Load data
data_loader = DataLoader()
data = data_loader.load_data()

# preprocess data
processed_data = preprocess_data(data)


def on_button_click(combo_box, entry, result_label):
    """
    Function to handle the button click event.

    :param combo_box:
    :param entry:
    :param result_label:
    :return:
    """
    # recommendation system
    selected_country = combo_box.get()
    user_input = entry.get()

    # filter the hotel name and the score
    result_label.config(text="Loading...")
    result = recommend_hotel(processed_data, selected_country, user_input)
    result_label.config(text=result)


def show_frame_nu():
    """
    Function to show the recommendation tool frame.
    :return:
    """
    # Create the main window
    ventana = tk.Tk()
    ventana.title("Accommodation Finder")
    ventana.geometry("600x400")
    ventana.resizable(False, False)
    # background color
    ventana.config(bg="black")

    # Create a label
    title_label = tk.Label(ventana, text="Accommodation Finder GUI", font=("Ar
    title_label.config(fg="white", bg="black")
    title_label.pack(pady=10)

    # Create a combo box
    paises = ["Netherlands", "UK", "France", "Spain", "Italy", "Austria"]
    combo_box = ttk.Combobox(ventana, values=paises, state="readonly")
    combo_box.set("¿Where are you going?")
    combo_box.pack(pady=10)
```

```
    # Create a text box
    entry = tk.Entry(ventana, width=30)
    entry.pack(pady=10)

    # Create a button
    find_button = tk.Button(ventana, text="Find", command=lambda: on_button_cl
    find_button.pack(pady=10)

    # Create a label to show the result
    result_label = tk.Label(ventana, text="")
    result_label.pack(pady=10)

    # Start the main loop of the GUI
    ventana.mainloop()


# main
if __name__ == "__main__":
    show_frame_nu()
```

To study hotels and their characteristics in more depth, I have created an interface that allows us to see their name, score and best and worst characteristics

```python
import tkinter as tk
from tkinter import ttk

from src.hotel_model import get_worst_values_for_hotel, get_best_values_for_ho
from src.data_loader import DataLoader
from src.hotel_model import create_hotel_objects


def update_results(event):
    search_query = search_var.get().lower()
    matching_hotels = [hotel.name for hotel in hotel_objects if search_query i
    results_listbox.delete(0, tk.END)
    for hotel in matching_hotels:
        results_listbox.insert(tk.END, hotel)


def on_select(event):
    selected_index_filtered = results_listbox.curselection()
    if selected_index_filtered:
        selected_index_filtered = selected_index_filtered[0]
        selected_hotel_name = results_listbox.get(selected_index_filtered)
        # Find the corresponding index in the original list
        selected_index_original = [i for i, hotel in enumerate(hotel_objects)
        selected_hotel = hotel_objects[selected_index_original]

        search_var.set(selected_hotel.name)
        update_results(None)

        # Display detailed info in the Text widget
        info_text.delete(1.0, tk.END)
        info_text.insert(tk.END, str(selected_hotel))
        info_text.insert(tk.END, f"Worst Features: {get_worst_values_for_hotel
        info_text.insert(tk.END, f"Best Features: {get_best_values_for_hotel(d


# Load the dataset
data = DataLoader().load_data()

# Create hotel objects using the provided function
hotel_objects = create_hotel_objects(data)

# Create the main window
root = tk.Tk()
root.title("Hotel Stats")
root.resizable(False, False)
root.geometry("600x400")
root.config(bg="black")

# Create variables
search_var = tk.StringVar()

# Create a title label
title_label = tk.Label(root, text="Hotel Stats", font=("Arial", 20))
title_label.config(fg="white", bg="black")
title_label.grid(row=0, column=0, columnspan=2, padx=10, pady=10, sticky=tk.W)

# Create GUI components
```

```
# Create a label
search_label = ttk.Label(root, text="Hotel name:")
search_label.grid(row=1, column=0, padx=10, pady=10, sticky=tk.W)

# Create an entry box
search_entry = ttk.Entry(root, textvariable=search_var)
search_entry.grid(row=1, column=1, padx=10, pady=10, sticky=tk.W + tk.E)
search_entry.bind("<KeyRelease>", update_results)

# Create a listbox
results_listbox = tk.Listbox(root, selectmode=tk.SINGLE, height=10)
results_listbox.grid(row=2, column=1, padx=10, pady=10, sticky=tk.W)
results_listbox.bind("<ButtonRelease-1>", on_select)

# Create a scrollbar
results_scrollbar = ttk.Scrollbar(root, orient=tk.VERTICAL, command=results_li
results_scrollbar.grid(row=1, column=2, rowspan=4, sticky=tk.N + tk.S)
results_listbox.configure(yscrollcommand=results_scrollbar.set)

# Create a Text widget for detailed information
info_text = tk.Text(root, height=10, width=40)
info_text.grid(row=0, column=3, rowspan=4, columnspan=3, padx=10, pady=10, sti

# Start the main event loop
root.mainloop()
```

With a Hotel class:

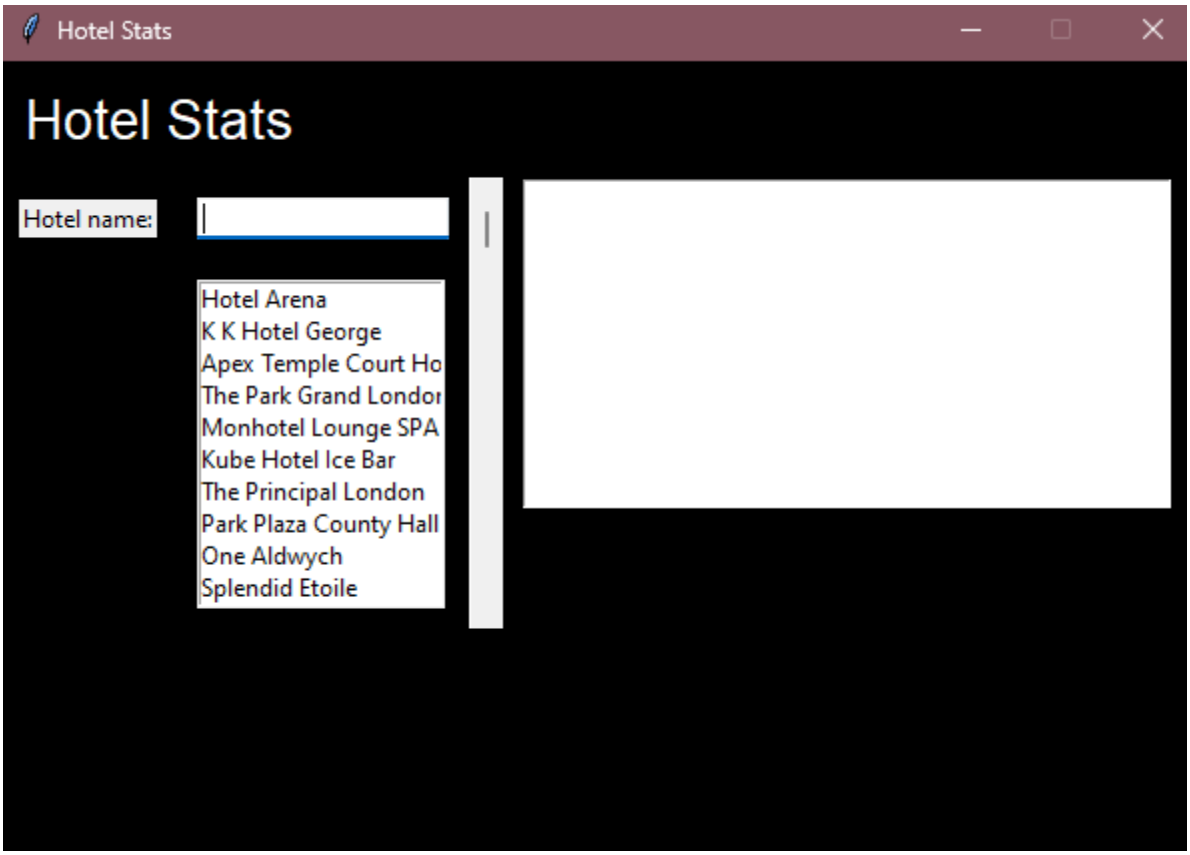
```
class Hotel:
    def __init__(self, name, average_score, data):
        """
        Class to store the hotel's information.

        :param name: the name of the hotel
        :param average_score: the average score of the hotel
        """
        self.name = name
        self.average_score = average_score
        self.worst_features = []
        self.best_features = []

    def __str__(self):
        """
        Method to print the hotel's information.
        """
        return f"Hotel: {self.name}\nAverage Score: {self.average_score}\n"
```

And Hotel Model

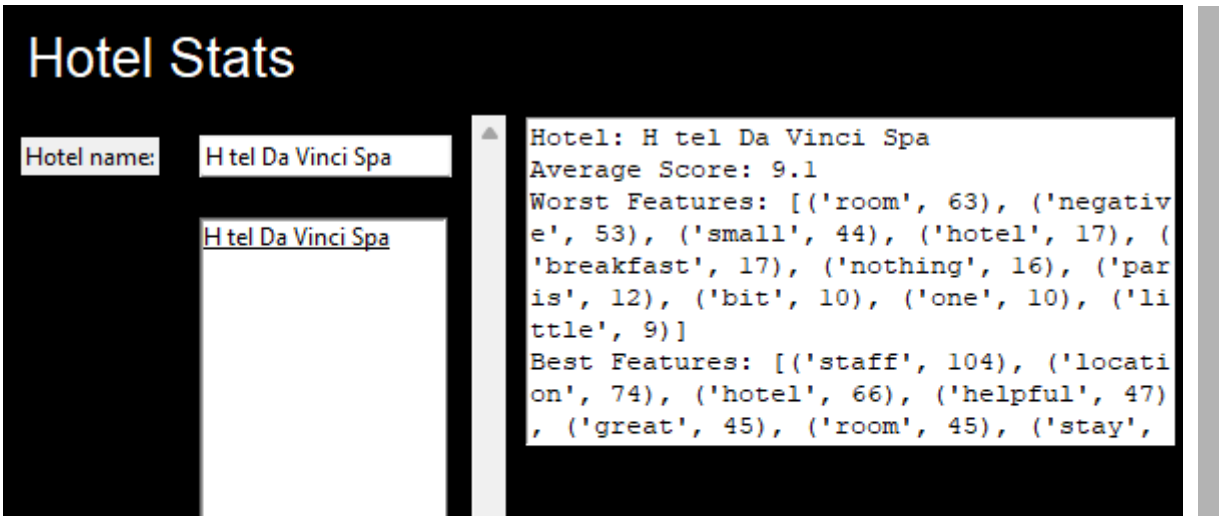
```python
from collections import Counter
from src.data_preprocessor import clean_text
from src.hotel import Hotel


def create_hotel_objects(data):
    """
    Function to create a hotel object for every hotel in the dataset.

    :param data: the dataset
    :return: a list of hotel objects
    """
    hotel_objects = []
    for hotel_name in data["Hotel_Name"].unique():
        hotel_data = data[data["Hotel_Name"] == hotel_name].copy()
        average_score = hotel_data["Average_Score"].iloc[0]
        hotel_object = Hotel(hotel_name, average_score, data)
        hotel_objects.append(hotel_object)
    return hotel_objects


def get_worst_values_for_hotel(data, hotel_name):
    """
    Function to get the most frequent words in negative reviews for a specific

    :param data: the dataset
    :param hotel_name: hotel name
    :return: a list of the 10 most frequent words in negative reviews for the
    """

    # Filter the data for the specific hotel
    hotel_data = data[data['Hotel_Name'] == hotel_name]
    macro_list = []

    # Get the negative reviews into a single string
    for bad_review in hotel_data['Negative_Review']:
        bad_review = clean_text(bad_review)
        for word in bad_review.split():
            macro_list.append(word)

    # Get the 10 most frequent words
    macro_list = Counter(macro_list).most_common(10)

    return macro_list


def get_best_values_for_hotel(data, hotel_name):
    """
    Function to get the most frequent words in positive reviews for a specific

    :param data: the dataset
    :param hotel_name: hotel name
    :return: a list of the 10 most frequent words in positive reviews for the
    """

    # Filter the data for the specific hotel
    hotel_data = data[data['Hotel_Name'] == hotel_name]
```

```
    macro_list = []

    # Get the positive reviews into a single string
    for good_review in hotel_data['Positive_Review']:
        good_review = clean_text(good_review)
        for word in good_review.split():
            macro_list.append(word)

    # Get the 10 most frequent words
    macro_list = Counter(macro_list).most_common(10)

    return macro_list
```

This tool allows us to visualize hotels in a simple way and despite being a basic tool, it has interesting applications



> Main page



Conclusion

In this project, we have applied machine learning techniques to analyse hotel reviews and ratings. We have performed data preprocessing, data analysis, sentiment analysis, topic modeling, and recommendation system. We have used various libraries and tools, such as pandas, numpy, nltk, sklearn, gensim, and VADER. We have also used data visualisation to display our results and insights.

Our main findings are:

- The hotels in our dataset have a good quality and reputation, as most of the reviews and ratings are positive.
- The most important factors that influence the customers' satisfaction are the hotel's location, staff, and breakfast.
- The customers' sentiments are generally aligned with their ratings, but there are some exceptions due to sarcasm, irony, or exaggeration.
- The main topics that the customers talk about in their reviews are the hotel's facilities, services, cleanliness, and comfort.
- We can recommend hotels to customers based on their preferences and past reviews, using a content-based filtering approach.

Our project demonstrates the potential of machine learning to enhance the customer experience and the hotel management. By using machine learning, we can understand the customers' needs and expectations, and provide them with personalized suggestions. We can also identify the strengths and weaknesses of the hotels, and improve their quality and performance.