Configuration Manual

MSc Research Project
MSc in AI for Business

Ana Sofia Lara
Student ID: x23105879

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	ANA SOFIA LARA		
Student ID:	X23105879		
Programme:	MSC IN AI FOR BUSINESS	**Year:**	2023/2024
Module:	Research Project		
Lecturer:	Faithful Onwuegbuche		
Submission Due Date:	August 12th 2024		
Project Title:	Reinforcing Hotel Recommendation Systems through the implementation of Hybrid Modeling		
Word Count:			
Page Count:	20		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ana Sofia Lara
Date:	06/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ana Sofia Lara
Student ID: x23105879

1. Introduction

To fully understand the execution of the implementation and have a complete detailed illustration of the requirements needed to developed the research, this following configuration manual displays the resources and steps taken into account to design the model which main objective is to predict the most suitable hotels to travel next to by offering hybrid recommendations based on collaborative filtering and content-based filtering models. This manual will provide an insightful overview of the hardware and software requirements as well as the steps taken through the implementation of the project.

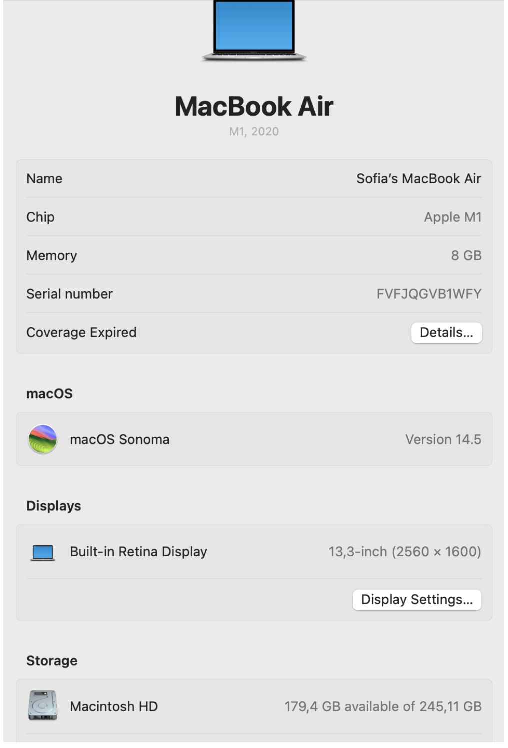
2. System Configuration

To proceed with the development of a thorough configuration manual, initially it is needed to understand the system configuration requirements taken into account before and during the implementation of the project.
The system requirements are as follows:

2.1 Hardware Requirement

Hardware	**Configuration**
System	Macbook Air (M1,2020)
Operating System	macOS Sonoma 14.5 (23F79)
RAM	8GB
Hard Disk	245.11GB
Graphics Card	8-core GPU
Processor	Apple M1 chip

***TABLE** 1- Hardware Requirements*


MacBook Air

M1, 2020

Name	Sofia's MacBook Air
Chip	Apple M1
Memory	8 GB
Serial number	FVFJQGVB1WFY
Coverage Expired	Details...

macOS

macOS Sonoma	Version 14.5

Displays

Built-in Retina Display	13,3-inch (2560 × 1600)
	Display Settings...

Storage

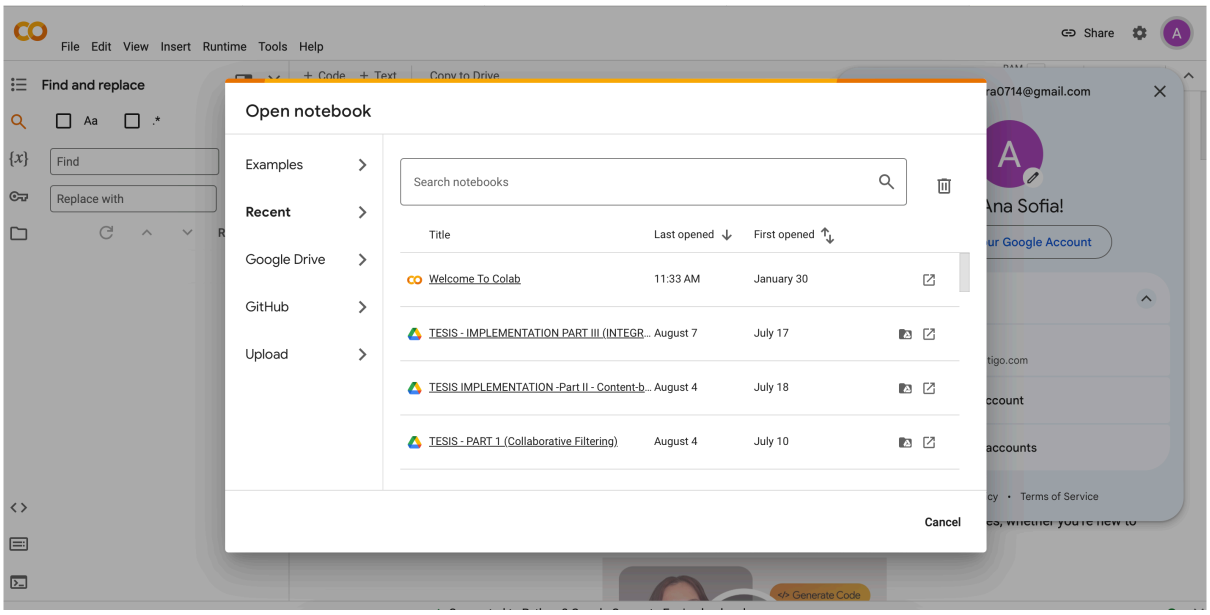
Macintosh HD	179,4 GB available of 245,11 GB

FIGURE 1 - Operating System Configuration

2.2 Software Requirement

On the other hand, the software configurations required are as follows:

Software	Version
Google Colab Community (Python)	2024 (Python 3.10)



Open notebook

Title	Last opened	First opened
Welcome To Colab	11:33 AM	January 30
TESIS - IMPLEMENTATION PART III (INTEGR...	August 7	July 17
TESIS IMPLEMENTATION -Part II - Content-b...	August 4	July 18
TESIS - PART 1 (Collaborative Filtering)	August 4	July 10

FIGURE 2- GOOGLE COLAB COMMUNITY INTEGRATION (PYTHON LANGUAGE)

3. Project Implementation

3.1 Data Collection

For the development of this project, the model and all the technical processes were based in the collection of a specific dataset named originally "Booking.com reviews dataset", gathered from Kaggle who collected it from the original website of booking.com through Crawl Feeds. The dataset in this project was renamed as "Hotel.reviews.csv". It is of public domain, available for everyone to download since 2021 when it was imported to Kaggle by @opensnippets. The dataset has a size of 47.02 MB.

3.2 Data Summary

Regarding the content of the dataset, it counts with 15 columns and 26,386 rows as can be seen in TABLE 2. Each row represents a review of an hotel made by a particular user. The data itself, as mentioned displays the reviews made by users on the different hotels experienced, both leaving a text review in the majority of the cases and a rating. The Hotels encountered in the dataset are located in Belgium or Belarus as those are the only two countries involved in the data; some of the hotels are replicated as diverse reviews were made on them, meaning that the number of unique hotels in general actually is 819.

review_title	Title of the review
reviewed_at	Date of the review
reviewed_by	User who wrote the review
Images	Images attached to the review
crawled_at	The date the review was tracked
URL	URL of the review
hotel_name	The name of the hotel
hotel_url	URL of the hotel in booking.com
avg_rating	Average rating
Nationality	Nationality of the user
# rating	Rating made by the user
review_text	Text of the review
raw_review_text	Raw text of the review
Tags	Tags of the review
Meta	Raw data of the review

***TABLE** 2 - "Hotel_Reviews.csv" dataset columns*

During the understanding of the data, an initial Exploratory Data Analysis was done to gather insights about the data and its content. It was discovered that the data counted with only two countries as mentioned, Belgium and Belarus, being Brussels the city with the most hotels (2160 Hotels). The hotel with the most reviews is the Marivaux Hotel in Brussels with 449 reviews. Moreover, only 209 reviews had images attached to the review submitted and the dataset contains reviews from July 2018 to July 2021.
For further understanding, the hotels are rated from 0 to 10, the reviewers come from 123 different nationalities, in its majority from the UK.

3.3 Data Preparation

The next step in the methodology carried out to develop the project is the preparation of the data. The original dataset as can be seen in the TABLE 2, counted with one column called hotel_url, indicating the location of the hotel. To be able to fully comprehend the results of the EDA, first it was needed to complete the dataset with the additional information underlying in some of the data already collected as for example the specific City and Country the Hotel is located in, allowing to gather more details about the dataset itself. To do this, filtering the hotels in excel, allowed to encountered the unique hotels and insert two new columns to the initial version of the dataset "City" and "Country" now counting with 17 columns.

3.4 Data Preprocessing

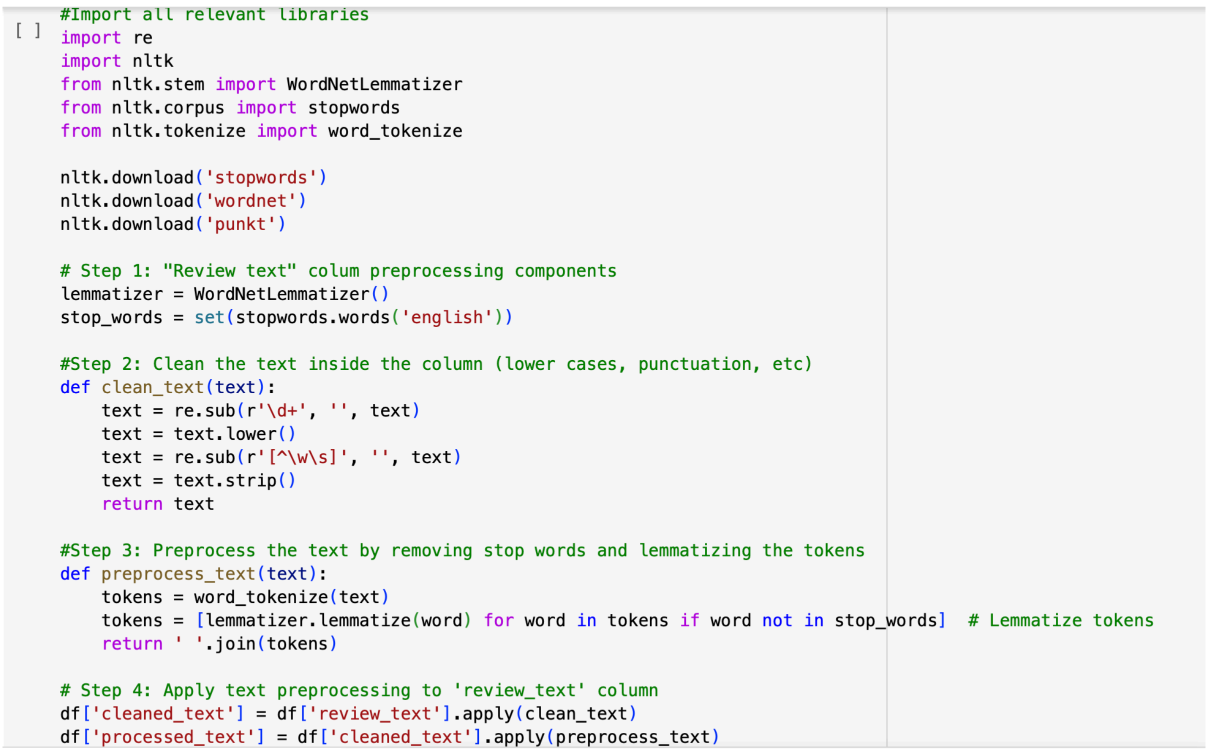
Later on, the dataset was completed and the EDA offered a more detailed understanding of the patterns within the data as mentioned. The next step was the data preprocessing, for this, the data needed first to be cleaned.

3.4.1 Data Cleaning

During the data cleaning stage, the libraries "Re" and "NLTK" were imported to proceed with the cleaning efforts. The following directories were downloaded to support said operations:

- Wordnet: Lemmatization of words. Converting words into their roots.
- Stopwords: Eliminating common words such as "is" "at", etc to reduce code disruption.
- Punkt: tokenization of words, transforming words into tokens to create lists for easier processing.

In addition, data cleaning consists of removing digits or special characters, convert all into lower cases and remove white spaces within text as shown in IMAGE 1.

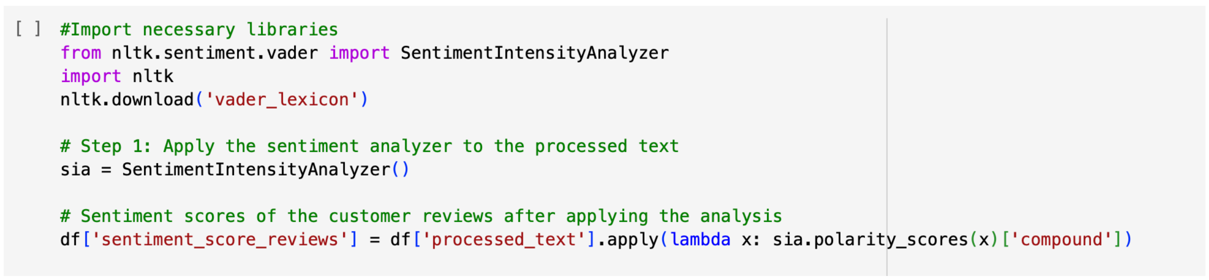


```
[ ]  #Import all relevant libraries
     import re
     import nltk
     from nltk.stem import WordNetLemmatizer
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize

     nltk.download('stopwords')
     nltk.download('wordnet')
     nltk.download('punkt')

     # Step 1: "Review text" colum preprocessing components
     lemmatizer = WordNetLemmatizer()
     stop_words = set(stopwords.words('english'))

     #Step 2: Clean the text inside the column (lower cases, punctuation, etc)
     def clean_text(text):
         text = re.sub(r'\d+', '', text)
         text = text.lower()
         text = re.sub(r'[^\w\s]', '', text)
         text = text.strip()
         return text

     #Step 3: Preprocess the text by removing stop words and lemmatizing the tokens
     def preprocess_text(text):
         tokens = word_tokenize(text)
         tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]  # Lemmatize tokens
         return ' '.join(tokens)

     # Step 4: Apply text preprocessing to 'review_text' column
     df['cleaned_text'] = df['review_text'].apply(clean_text)
     df['processed_text'] = df['cleaned_text'].apply(preprocess_text)
```

IMAGE 1 - Data Preprocessing stage code in Colab Notebook Part I.

3.4.2 Sentiment Analysis

In terms of the next step of the data preprocessing, the text values of the dataset needed to be converted or understood by the machine learning algorithm for further processing which is why is needed to perform sentiment analysis. The dataset contains a large amount of text due to the reviews included, the sentiment analysis was carried out as shown in IMAGE 2. From the library "NLTK", the Sentiment Intensity Analyzer was imported along the vader_lexicon which is a directory for sentiment scores. The analyzer was applied to the text which resulted in the compound sentiment scores of the reviews.



```
[ ] #Import necessary libraries
    from nltk.sentiment.vader import SentimentIntensityAnalyzer
    import nltk
    nltk.download('vader_lexicon')

    # Step 1: Apply the sentiment analyzer to the processed text
    sia = SentimentIntensityAnalyzer()

    # Sentiment scores of the customer reviews after applying the analysis
    df['sentiment_score_reviews'] = df['processed_text'].apply(lambda x: sia.polarity_scores(x)['compound'])
```

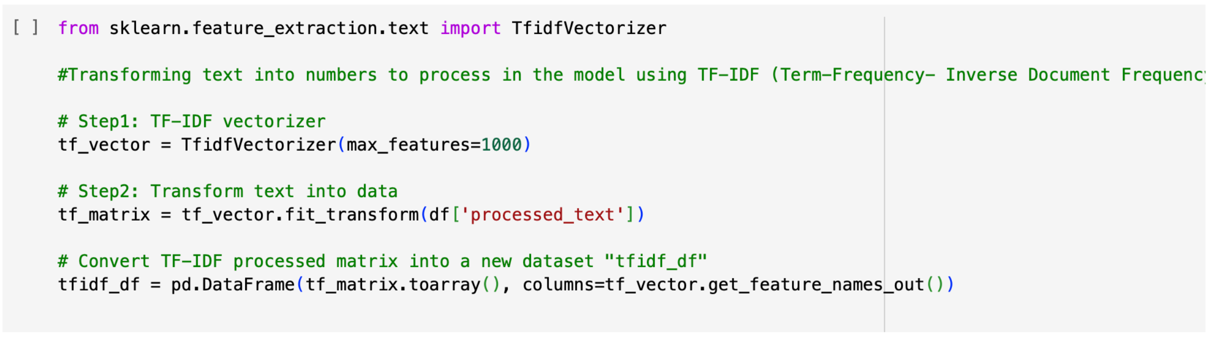
***IMAGE 2**- Sentiment Analysis code*

3.5 Data Transformation

As the data specially the text in it is cleaned and process, transformation is required to configure the final dataset to embedded into the model.

3.5.1 Feature Engineering

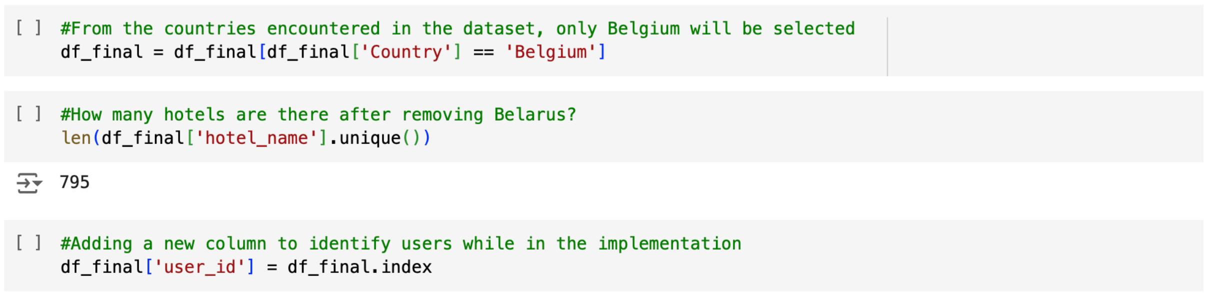
The transformation of the data starts with engineering its features, for this, primarily, it was required to transform the categorical "text" values into numerical ones, so they could actually be processed for the models further along. To do this, TF-IDF vectorizer was implemented, meaning that by converting words into vectors, the model can actually identify their numerical values after being introduced into the TF-IDF matrix as seen in IMAGE 3.



```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

    #Transforming text into numbers to process in the model using TF-IDF (Term-Frequency- Inverse Document Frequenc

    # Step1: TF-IDF vectorizer
    tf_vector = TfidfVectorizer(max_features=1000)

    # Step2: Transform text into data
    tf_matrix = tf_vector.fit_transform(df['processed_text'])

    # Convert TF-IDF processed matrix into a new dataset "tfidf_df"
    tfidf_df = pd.DataFrame(tf_matrix.toarray(), columns=tf_vector.get_feature_names_out())
```

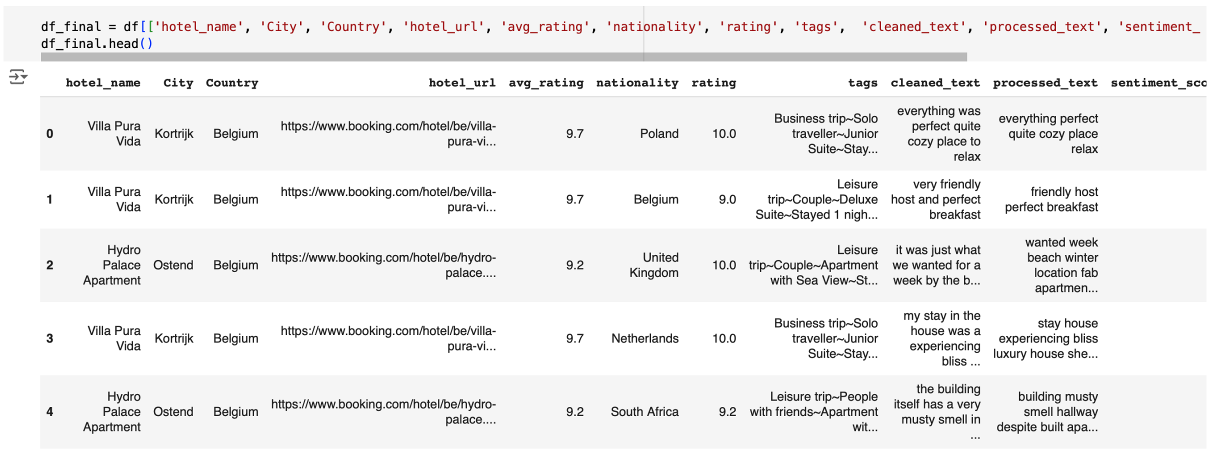
***IMAGE 3** - TD-IDF vectorizer - Text into numbers*

Next, it was decided that the model would only take into account the Hotels in Belgium, removing Belarus from its data, the reason behind said decision was to further increase the model accuracy by offering recommendation targeted to only specific country and its particular Hotel industry. As IMAGE 4 shows, the final dataset only shows one country, only 795 unique hotels and the addition of one extra column "user_id", to simplify the deployment and evaluation of the model when later on its tested. By only having to select a number, its easier to check the performance as to defer to the original dataset and check an specific user name. IMAGE 5 then, presents the final data frame “df_final” when the data selection stage was encountered and specific columns targeted to be part of the embedding into the models.



```
[ ] #From the countries encountered in the dataset, only Belgium will be selected
    df_final = df_final[df_final['Country'] == 'Belgium']

[ ] #How many hotels are there after removing Belarus?
    len(df_final['hotel_name'].unique())

    795

[ ] #Adding a new column to identify users while in the implementation
    df_final['user_id'] = df_final.index
```

IMAGE 4 - Data Selection - Country Selection and Column Addition



```
df_final = df[['hotel_name', 'City', 'Country', 'hotel_url', 'avg_rating', 'nationality', 'rating', 'tags',  'cleaned_text', 'processed_text', 'sentiment_
df_final.head()
```

	hotel_name	City	Country	hotel_url	avg_rating	nationality	rating	tags	cleaned_text	processed_text	sentiment_sco
0	Villa Pura Vida	Kortrijk	Belgium	https://www.booking.com/hotel/be/villa-pura-vi...	9.7	Poland	10.0	Business trip~Solo traveller~Junior Suite~Stay...	everything was perfect quite cozy place to relax	everything perfect quite cozy place relax	
1	Villa Pura Vida	Kortrijk	Belgium	https://www.booking.com/hotel/be/villa-pura-vi...	9.7	Belgium	9.0	Leisure trip~Couple~Deluxe Suite~Stayed 1 nigh...	very friendly host and perfect breakfast	friendly host perfect breakfast	
2	Hydro Palace Apartment	Ostend	Belgium	https://www.booking.com/hotel/be/hydro-palace....	9.2	United Kingdom	10.0	Leisure trip~Couple~Apartment with Sea View~St...	it was just what we wanted for a week by the b...	wanted week beach winter location fab apartmen...	
3	Villa Pura Vida	Kortrijk	Belgium	https://www.booking.com/hotel/be/villa-pura-vi...	9.7	Netherlands	10.0	Business trip~Solo traveller~Junior Suite~Stay...	my stay in the house was a experiencing bliss ...	stay house experiencing bliss luxury house she...	
4	Hydro Palace Apartment	Ostend	Belgium	https://www.booking.com/hotel/be/hydro-palace....	9.2	South Africa	9.2	Leisure trip~People with friends~Apartment wit...	the building itself has a very musty smell in ...	building musty smell hallway despite built apa...	

IMAGE 5 - Data Selection - Final data frame

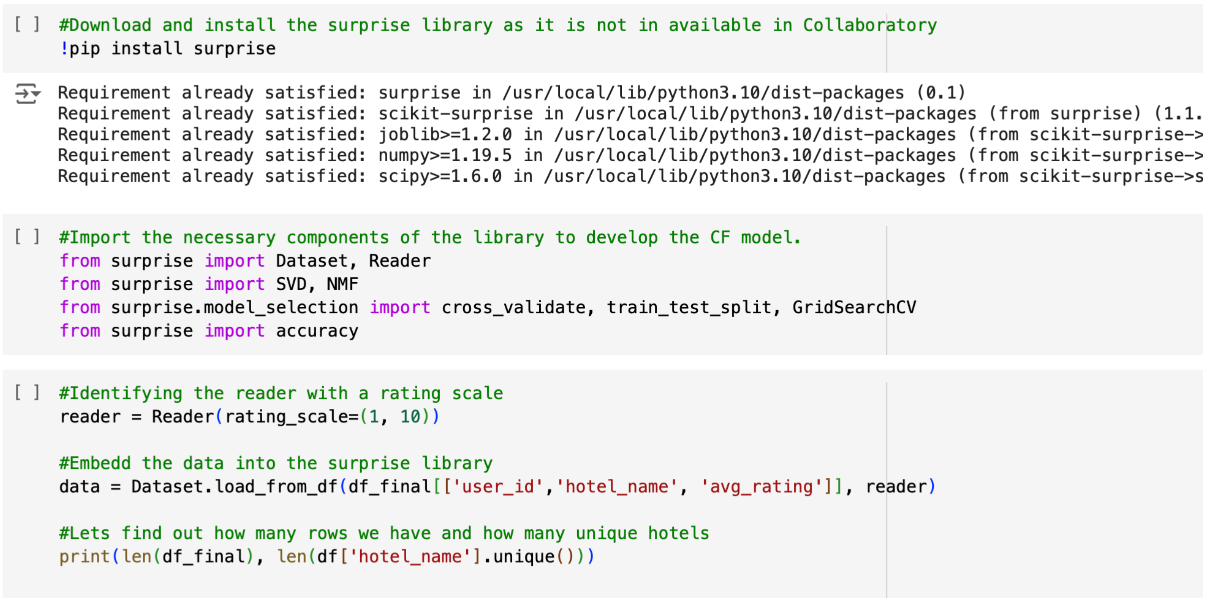
4. Model Building

4.1 COLLABORATIVE FILTERING

As mentioned, this project was focused on the development of a hybrid recommendation system by integrating two different models: Collaborative Filtering and Content-based filtering.

a) Libraries Importation

As per the first model, collaborative filtering is implemented by installing the surprise library which was not initially in the repertory of Colab Community. Then, some important functions need to be imported such as Reader, SVD, NMF and some other functions to later on test the model such as cross validate, train_test split, GridSearch and Accuracy as represented by IMAGE 6. This first model was developed by using Singular Value Decomposition in which the Reader will interpret the ratings made by the reviewers by the scale used from 0 to 10, needing three main attributes to produce the recommendations “user_id”, “hotel_name” “avg_rating”.



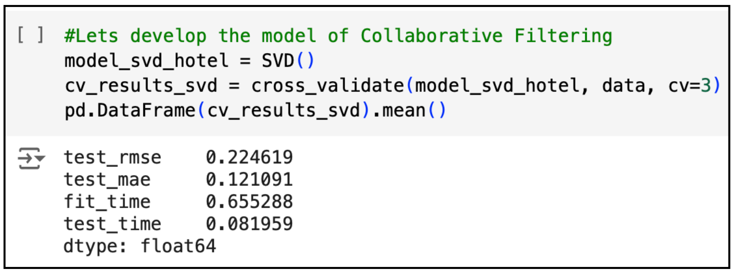
```
[ ] #Download and install the surprise library as it is not in available in Collaboratory
    !pip install surprise

    Requirement already satisfied: surprise in /usr/local/lib/python3.10/dist-packages (0.1)
    Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.10/dist-packages (from surprise) (1.1.
    Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->
    Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->
    Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->s
```

```
[ ] #Import the necessary components of the library to develop the CF model.
    from surprise import Dataset, Reader
    from surprise import SVD, NMF
    from surprise.model_selection import cross_validate, train_test_split, GridSearchCV
    from surprise import accuracy
```

```
[ ] #Identifying the reader with a rating scale
    reader = Reader(rating_scale=(1, 10))

    #Embedd the data into the surprise library
    data = Dataset.load_from_df(df_final[['user_id','hotel_name', 'avg_rating']], reader)

    #Lets find out how many rows we have and how many unique hotels
    print(len(df_final), len(df['hotel_name'].unique()))
```

***IMAGE** 6 - Collaborative Filtering initial setup*

b) Singular Value Decomposition



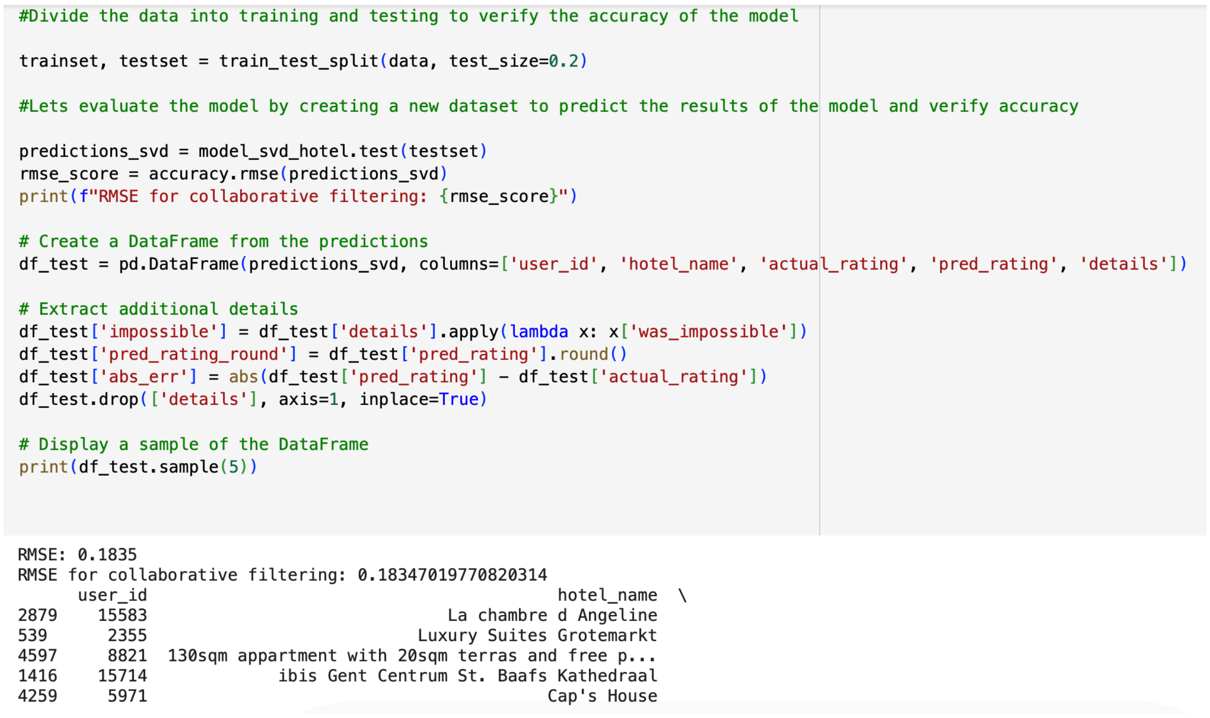
```
[ ] #Lets develop the model of Collaborative Filtering
    model_svd_hotel = SVD()
    cv_results_svd = cross_validate(model_svd_hotel, data, cv=3)
    pd.DataFrame(cv_results_svd).mean()

    test_rmse     0.224619
    test_mae      0.121091
    fit_time      0.655288
    test_time     0.081959
    dtype: float64
```

***IMAGE** 7 - SVD Model Training*

Regarding the model itself, SVD allows to create vectors based on the user preferences on ratings, identifying patterns of similarities to cluster similar users together and create recommendations based on those who are similar. IMAGE 7 demonstrates the implementation of SVD during training and the initial results of its evaluation which contains RMSE, MAE and the time spent on fitting the data into the model and testing said data to conjure results, which at first hand are positive and the prediction are not so far off from the actual ratings.

c) Testing the Model - Root Mean Square Error

To evaluate the model, further testing was done by splitting the data into training (already done) and testing sets as shown by IMAGE 8. The predictions were calculated based on the SVD model and later on tested by the RMSE metric. The testing involved



```
#Divide the data into training and testing to verify the accuracy of the model

trainset, testset = train_test_split(data, test_size=0.2)

#Lets evaluate the model by creating a new dataset to predict the results of the model and verify accuracy

predictions_svd = model_svd_hotel.test(testset)
rmse_score = accuracy.rmse(predictions_svd)
print(f"RMSE for collaborative filtering: {rmse_score}")

# Create a DataFrame from the predictions
df_test = pd.DataFrame(predictions_svd, columns=['user_id', 'hotel_name', 'actual_rating', 'pred_rating', 'details'])

# Extract additional details
df_test['impossible'] = df_test['details'].apply(lambda x: x['was_impossible'])
df_test['pred_rating_round'] = df_test['pred_rating'].round()
df_test['abs_err'] = abs(df_test['pred_rating'] - df_test['actual_rating'])
df_test.drop(['details'], axis=1, inplace=True)

# Display a sample of the DataFrame
print(df_test.sample(5))
```

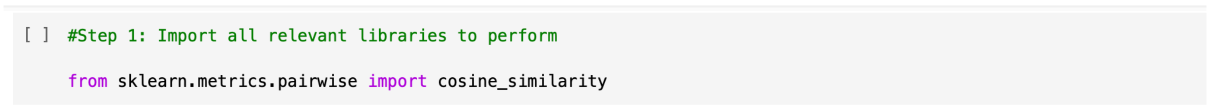
```
RMSE: 0.1835
RMSE for collaborative filtering: 0.18347019770820314
      user_id                                         hotel_name  \
2879    15583                                La chambre d Angeline
539      2355                              Luxury Suites Grotemarkt
4597     8821  130sqm appartment with 20sqm terras and free p...
1416    15714                 ibis Gent Centrum St. Baafs Kathedraal
4259     5971                                          Cap's House
```

IMAGE 8 - Testing the Collaborative Filtering Model

4.2 CONTENT-BASED FILTERING

A) Libraries Importation

For the development of the second model, content-based filtering, the library SKlearn will be imported, mostly its function of cosine_similarity.

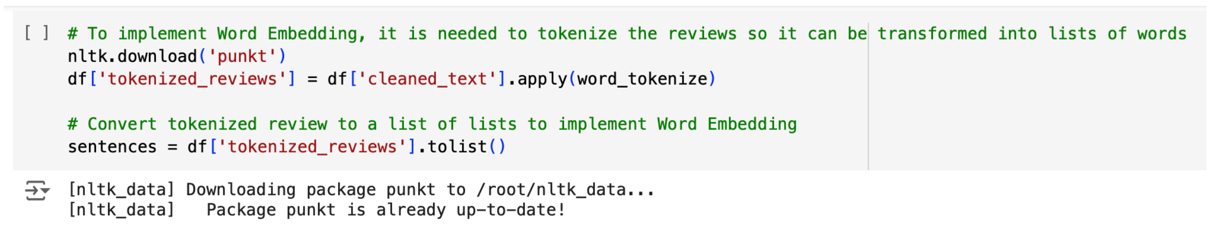


```
[ ] #Step 1: Import all relevant libraries to perform

    from sklearn.metrics.pairwise import cosine_similarity
```

IMAGE 9 - Library importation

b) Review Tokenization of words

To accommodate the data for the model Word2Vec, it was needed to verify that the data was properly tokenized since to fully vectorize it, it is needed to identify words as tokens. This was done by downloading the Punkt directory and apply it to the cleaned text gathered in the processing stage.



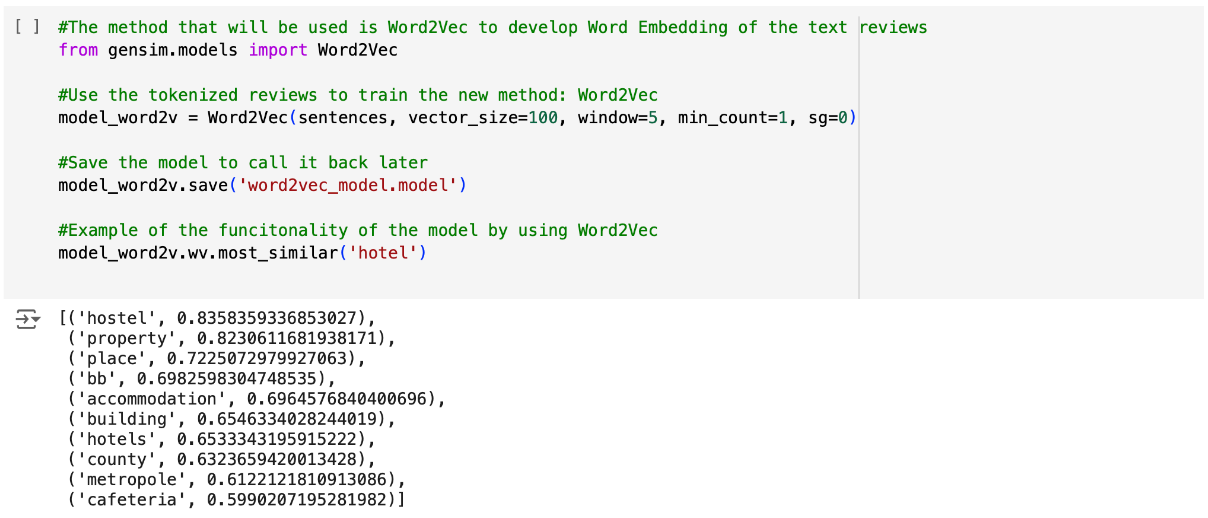
```
[ ] # To implement Word Embedding, it is needed to tokenize the reviews so it can be transformed into lists of words
    nltk.download('punkt')
    df['tokenized_reviews'] = df['cleaned_text'].apply(word_tokenize)

    # Convert tokenized review to a list of lists to implement Word Embedding
    sentences = df['tokenized_reviews'].tolist()

    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Package punkt is already up-to-date!
```

IMAGE *10- Word Tokenization for Word2Vec preparation*

c) Word2Vec

Word2Vec is the model selected to carry on the content-based filtering. Its nature of developing vectors support the need to find item-based similarities between the hotels by taking into account its reviews. Word2Vec gives support to Word Embedding techniques mostly implemented by importing the gensim library and the Word2Vec directory.

The vectors generated were going to have a size of 100 due to some limitations of the RAM storage available from Colab Notebooks with a minimum count of 1 and skip-gram 0, since the target word is defined and the context words predicted.
As an example, the word hotel was introduced in the model, and the similarities encountered as shown in IMAGE 11 are the most similar to the word Hotel.



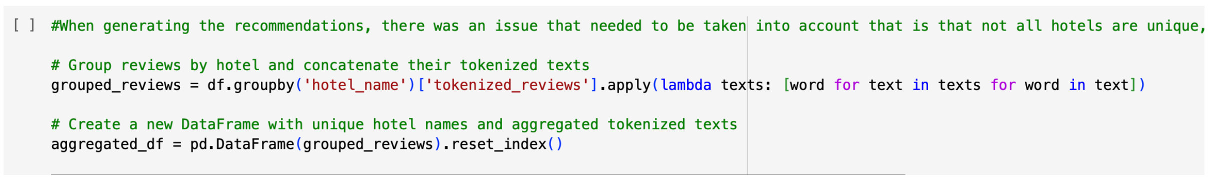
```
[ ] #The method that will be used is Word2Vec to develop Word Embedding of the text reviews
    from gensim.models import Word2Vec

    #Use the tokenized reviews to train the new method: Word2Vec
    model_word2v = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)

    #Save the model to call it back later
    model_word2v.save('word2vec_model.model')

    #Example of the funcitonality of the model by using Word2Vec
    model_word2v.wv.most_similar('hotel')
```

```
[('hostel', 0.8358359336853027),
 ('property', 0.8230611681938171),
 ('place', 0.7225072979927063),
 ('bb', 0.6982598304748535),
 ('accommodation', 0.6964576840400696),
 ('building', 0.6546334028244019),
 ('hotels', 0.6533343195915222),
 ('county', 0.6323659420013428),
 ('metropole', 0.6122121810913086),
 ('cafeteria', 0.5990207195281982)]
```

IMAGE 11 - Word2Vec Model

d) Aggregated Group of Reviews - Solving an issue

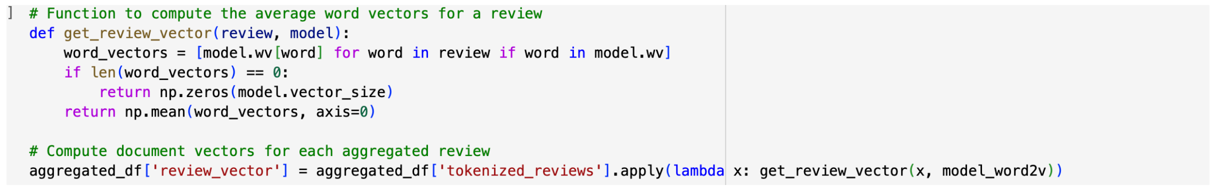
As the vectors were taking each Hotel as independent and unique hotels, it was needed to create group of reviews for each hotel that could have a list of lists of tokens to identify which reviews were from the same hotel to proceed with the development.



```
[ ] #When generating the recommendations, there was an issue that needed to be taken into account that is that not all hotels are unique,

    # Group reviews by hotel and concatenate their tokenized texts
    grouped_reviews = df.groupby('hotel_name')['tokenized_reviews'].apply(lambda texts: [word for text in texts for word in text])

    # Create a new DataFrame with unique hotel names and aggregated tokenized texts
    aggregated_df = pd.DataFrame(grouped_reviews).reset_index()
```

IMAGE 12 - GROUP REVIEWS

E) Vectors for the Reviews - Average Vectors

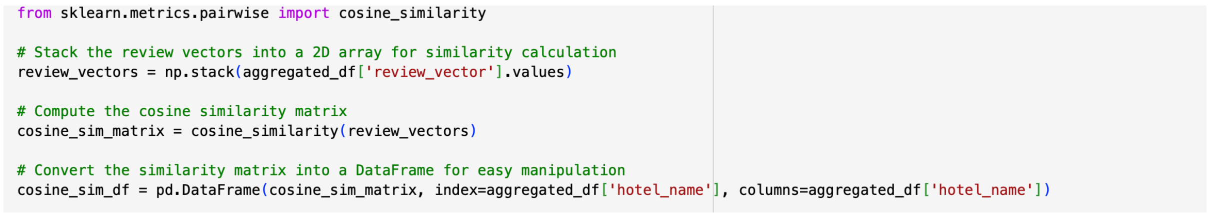


```python
# Function to compute the average word vectors for a review
def get_review_vector(review, model):
    word_vectors = [model.wv[word] for word in review if word in model.wv]
    if len(word_vectors) == 0:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

# Compute document vectors for each aggregated review
aggregated_df['review_vector'] = aggregated_df['tokenized_reviews'].apply(lambda x: get_review_vector(x, model_word2v))
```

***IMAGE** 13 - AVERAGE REVIEW VECTOR*

As now there were group of reviews with list of list of tokens and vectors that could identify them, it was needed to create average vectors per group of reviews for each unique hotel to have a main value of representation as shown in image 13.

F) Cosine Similarity



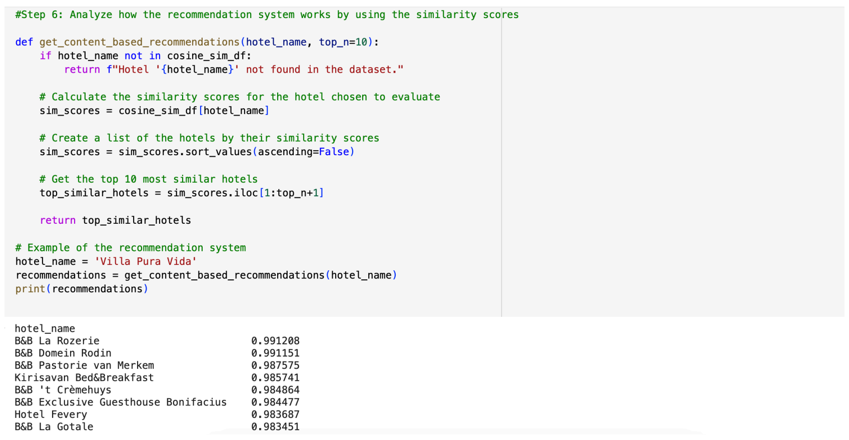
```python
from sklearn.metrics.pairwise import cosine_similarity

# Stack the review vectors into a 2D array for similarity calculation
review_vectors = np.stack(aggregated_df['review_vector'].values)

# Compute the cosine similarity matrix
cosine_sim_matrix = cosine_similarity(review_vectors)

# Convert the similarity matrix into a DataFrame for easy manipulation
cosine_sim_df = pd.DataFrame(cosine_sim_matrix, index=aggregated_df['hotel_name'], columns=aggregated_df['hotel_name'])
```

***IMAGE** 14 - COSINE SIMILARITY CALCULATION*

The next step in the development is cosine similarity. As can be seen in image 14, the process was done by stacking the reviews into a different infrastructure of 2 arrays to calculate the similarities between them by throwing the vectors into the matrix.

G) Testing of the model

the last part of the implementation of content-based filtering was to test the model. IMAGE 15 shows a clear example of the functionality and the recommendations generated. For this example, the hotel Villa Pura Vida was selected to generate recommendations which according to the model, the hotels in the image are the top similar to this particular hotel.



```
#Step 6: Analyze how the recommendation system works by using the similarity scores

def get_content_based_recommendations(hotel_name, top_n=10):
    if hotel_name not in cosine_sim_df:
        return f"Hotel '{hotel_name}' not found in the dataset."

    # Calculate the similarity scores for the hotel chosen to evaluate
    sim_scores = cosine_sim_df[hotel_name]

    # Create a list of the hotels by their similarity scores
    sim_scores = sim_scores.sort_values(ascending=False)

    # Get the top 10 most similar hotels
    top_similar_hotels = sim_scores.iloc[1:top_n+1]

    return top_similar_hotels

# Example of the recommendation system
hotel_name = 'Villa Pura Vida'
recommendations = get_content_based_recommendations(hotel_name)
print(recommendations)
```

```
hotel_name
B&B La Rozerie                        0.991208
B&B Domein Rodin                      0.991151
B&B Pastorie van Merkem               0.987575
Kirisavan Bed&Breakfast               0.985741
B&B 't Crèmehuys                      0.984864
B&B Exclusive Guesthouse Bonifacius   0.984477
Hotel Fevery                          0.983687
B&B La Gotale                         0.983451
```

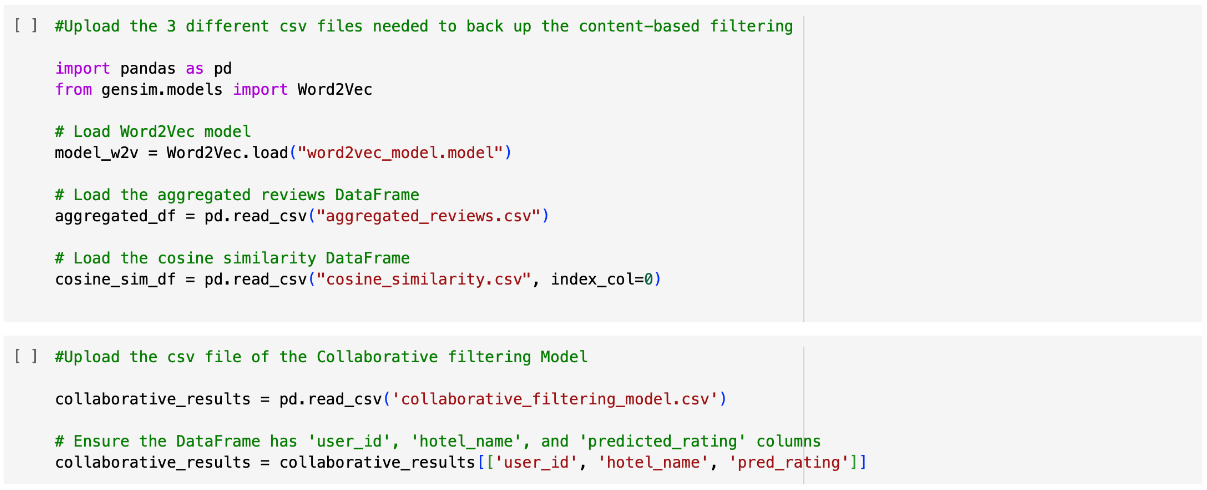
***IMAGE** 15 - TESTING OF THE SECOND MODEL*

4.3 HYBRID RECOMMENDATIONS

As per the final model, hybrid recommendation integrates both collaborative filtering and content based filtering.

A) File uploading

As the models were developed in different notebooks, it was needed to upload the required files from the past two notebooks to develop the final integration. For collaborative filtering, it was needed the collaborative filtering.csv and for the content based filtering more files where needed such as aggregated reviews, cosine similarity and model_w2v as shown in image 16. For the content-based files, gensim library is needed for decoding.



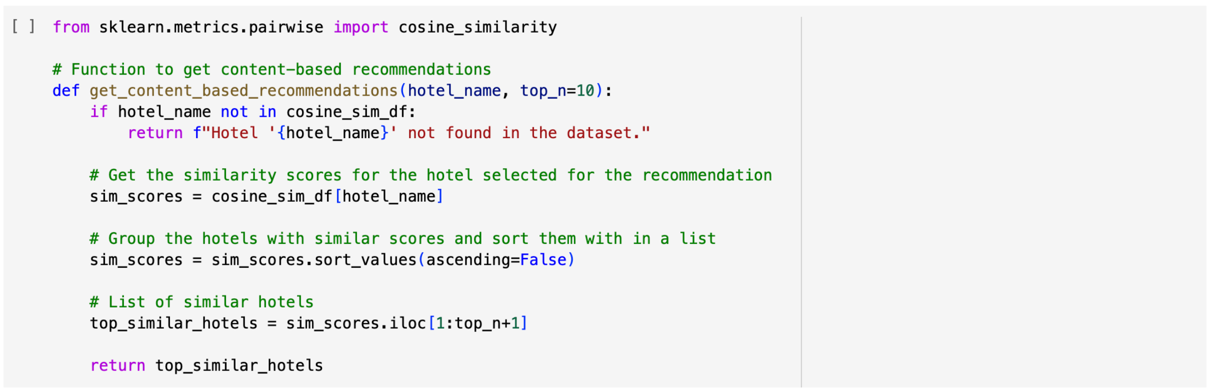
```
[ ]  #Upload the 3 different csv files needed to back up the content-based filtering

     import pandas as pd
     from gensim.models import Word2Vec

     # Load Word2Vec model
     model_w2v = Word2Vec.load("word2vec_model.model")

     # Load the aggregated reviews DataFrame
     aggregated_df = pd.read_csv("aggregated_reviews.csv")

     # Load the cosine similarity DataFrame
     cosine_sim_df = pd.read_csv("cosine_similarity.csv", index_col=0)
```

```
[ ]  #Upload the csv file of the Collaborative filtering Model

     collaborative_results = pd.read_csv('collaborative_filtering_model.csv')

     # Ensure the DataFrame has 'user_id', 'hotel_name', and 'predicted_rating' columns
     collaborative_results = collaborative_results[['user_id', 'hotel_name', 'pred_rating']]
```

***IMAGE** 16 - library and files uploading*

B) CONTENT-BASED RECOMMENDATIONS

now that the files are uploaded and the system recognized the past history of the algorithm, it was suggested by third parties (video tutorials such as) to corraborate that the content based recommendations were in place. As image 17 shows, the content based recommendations were initiated by importing cosine similarity functions.

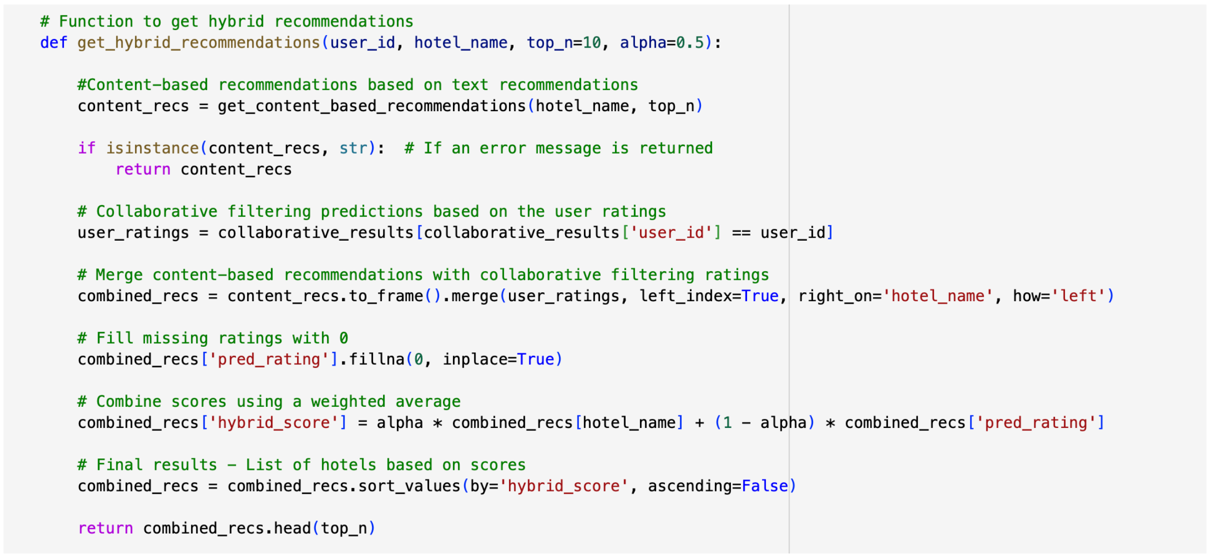


```
[ ] from sklearn.metrics.pairwise import cosine_similarity

    # Function to get content-based recommendations
    def get_content_based_recommendations(hotel_name, top_n=10):
        if hotel_name not in cosine_sim_df:
            return f"Hotel '{hotel_name}' not found in the dataset."

        # Get the similarity scores for the hotel selected for the recommendation
        sim_scores = cosine_sim_df[hotel_name]

        # Group the hotels with similar scores and sort them with in a list
        sim_scores = sim_scores.sort_values(ascending=False)

        # List of similar hotels
        top_similar_hotels = sim_scores.iloc[1:top_n+1]

        return top_similar_hotels
```

IMAGE 17 - CONTENT-BASED RECOMMENDATIONS

C) HYBRID RECOMMENDATIONS

now that the content-based recommendations are in place, since there were so many files needed in its conversion, the hybrid recommendations are generated.



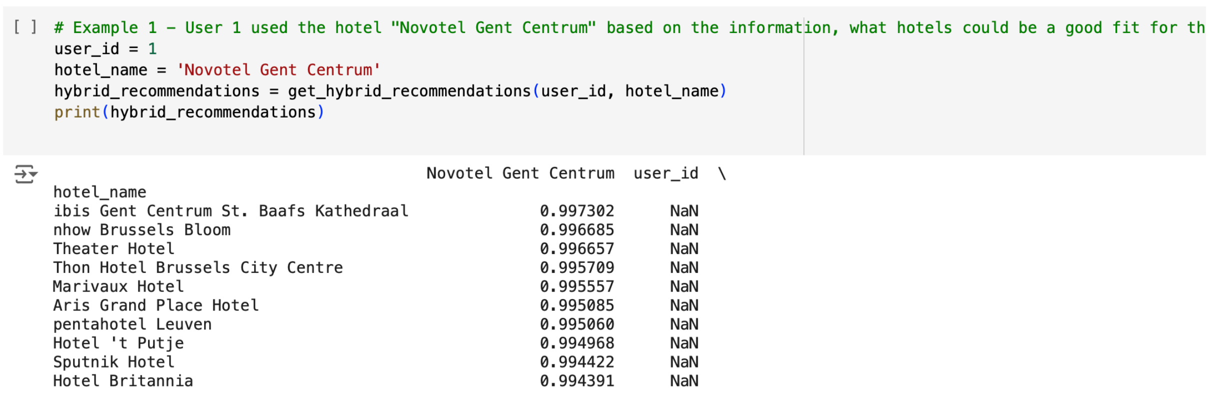
```
# Function to get hybrid recommendations
def get_hybrid_recommendations(user_id, hotel_name, top_n=10, alpha=0.5):

    #Content-based recommendations based on text recommendations
    content_recs = get_content_based_recommendations(hotel_name, top_n)

    if isinstance(content_recs, str):  # If an error message is returned
        return content_recs

    # Collaborative filtering predictions based on the user ratings
    user_ratings = collaborative_results[collaborative_results['user_id'] == user_id]

    # Merge content-based recommendations with collaborative filtering ratings
    combined_recs = content_recs.to_frame().merge(user_ratings, left_index=True, right_on='hotel_name', how='left')

    # Fill missing ratings with 0
    combined_recs['pred_rating'].fillna(0, inplace=True)

    # Combine scores using a weighted average
    combined_recs['hybrid_score'] = alpha * combined_recs[hotel_name] + (1 - alpha) * combined_recs['pred_rating']

    # Final results - List of hotels based on scores
    combined_recs = combined_recs.sort_values(by='hybrid_score', ascending=False)

    return combined_recs.head(top_n)
```

IMAGE 18 - HYBRID RECOMMENDATION CALCULATION

D) EXAMPLE OF HYBRID RECOMMENDATION (USER_ID=1, HOTEL-NOVOTEL GENT CENTRUM

The following experiment is to corroborate the right functionality of the model and evaluate initially its accuracy and the recommendations generated. This example takes the user_id=1 who liked or reviewed the hotel " Novotel Gent Centrum". The top similar hotels are being recommended to this particular user based on similar hotels from the one liked and hotels liked by similar users.

EXAMPLES OF THE HYBRID **RECOMMENDATIONS**

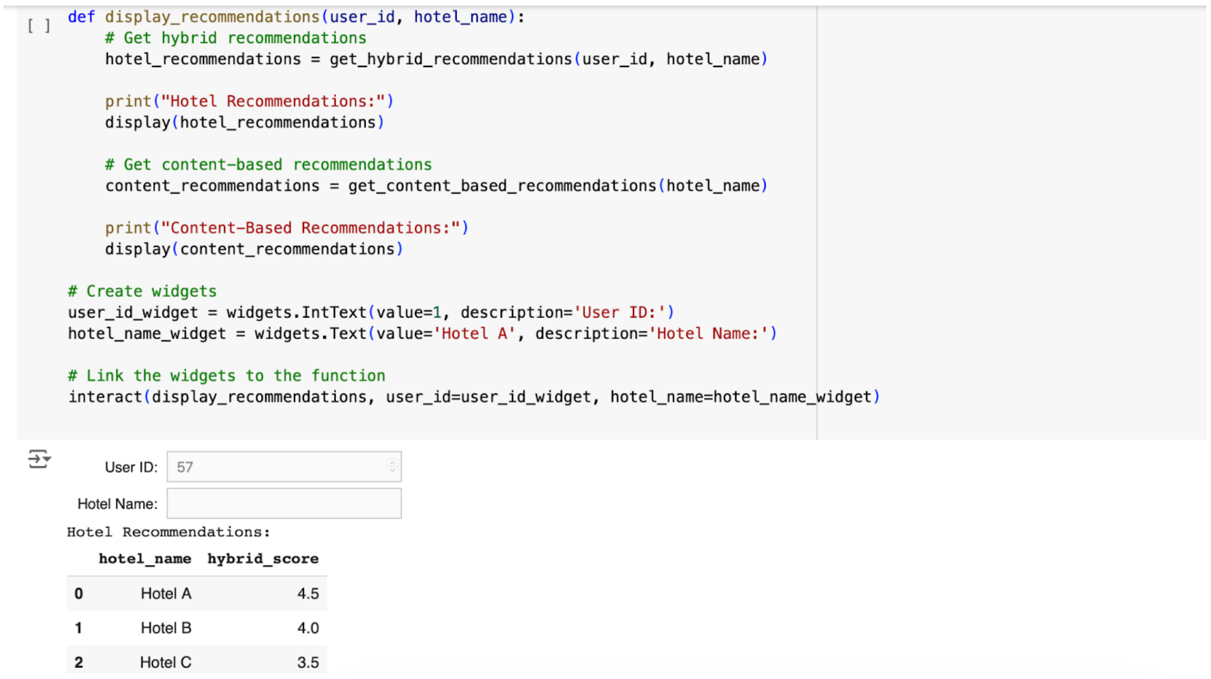


```
[ ] # Example 1 - User 1 used the hotel "Novotel Gent Centrum" based on the information, what hotels could be a good fit for th
    user_id = 1
    hotel_name = 'Novotel Gent Centrum'
    hybrid_recommendations = get_hybrid_recommendations(user_id, hotel_name)
    print(hybrid_recommendations)
```

```
                                        Novotel Gent Centrum  user_id  \
hotel_name
ibis Gent Centrum St. Baafs Kathedraal              0.997302      NaN
nhow Brussels Bloom                                 0.996685      NaN
Theater Hotel                                       0.996657      NaN
Thon Hotel Brussels City Centre                     0.995709      NaN
Marivaux Hotel                                      0.995557      NaN
Aris Grand Place Hotel                              0.995085      NaN
pentahotel Leuven                                   0.995060      NaN
Hotel 't Putje                                      0.994968      NaN
Sputnik Hotel                                       0.994422      NaN
Hotel Britannia                                     0.994391      NaN
```

***IMAGE** 19 - HYBRID RECOMMENDATION - EXPERIMENT 1*

E) Visual and interactive interface
Project wise, it was decided to include an interactive component to develop the recommendations, with the support of video tutorials it was possible to use colab widgets to create an interactive interface to test the hybrid model as shown in image 20.



```
[ ] def display_recommendations(user_id, hotel_name):
        # Get hybrid recommendations
        hotel_recommendations = get_hybrid_recommendations(user_id, hotel_name)

        print("Hotel Recommendations:")
        display(hotel_recommendations)

        # Get content-based recommendations
        content_recommendations = get_content_based_recommendations(hotel_name)

        print("Content-Based Recommendations:")
        display(content_recommendations)

    # Create widgets
    user_id_widget = widgets.IntText(value=1, description='User ID:')
    hotel_name_widget = widgets.Text(value='Hotel A', description='Hotel Name:')

    # Link the widgets to the function
    interact(display_recommendations, user_id=user_id_widget, hotel_name=hotel_name_widget)
```

User ID: 57

Hotel Name:

Hotel Recommendations:

	hotel_name	hybrid_score
0	Hotel A	4.5
1	Hotel B	4.0
2	Hotel C	3.5

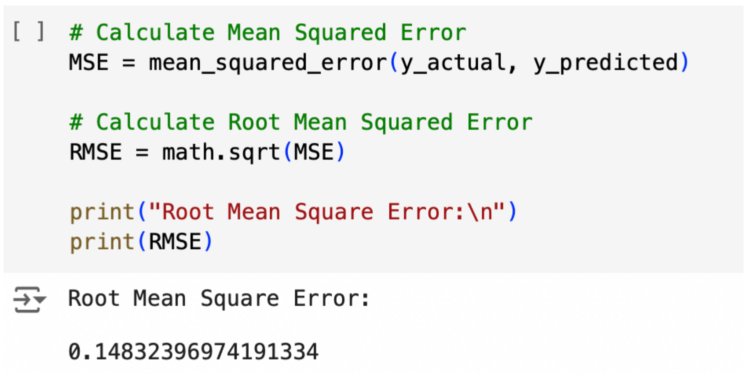
***IMAGE** 20 - INTERACTIVE INTERFACE - HYBRID RECOMMENDATIONS*

4.4 EVALUATION

As per the testing of the hybrid recommendation, three different metrics were used: Root Mean Square error, precision and F1-Score.

RMSE was selected due to its valuable insights in rating performance, supporting the collaborative filtering suggestions while precision was selected to evaluate the content based filtering, due to the fact that it evaluates the false positives, signaling if the recommendations are in fact correct. Lastly, F1 score was decided to be included since it is an integration between precision and recall giving a more complete insight on how accurate is the hybrid model.

A) RMSE

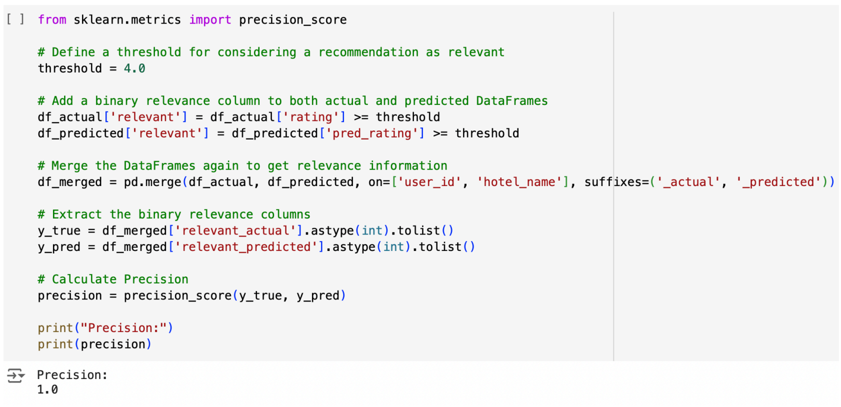


```
[ ] # Calculate Mean Squared Error
    MSE = mean_squared_error(y_actual, y_predicted)

    # Calculate Root Mean Squared Error
    RMSE = math.sqrt(MSE)

    print("Root Mean Square Error:\n")
    print(RMSE)
```

```
Root Mean Square Error:

0.14832396974191334
```

***IMAGE** 21 - RMSE Results*

B) Precision

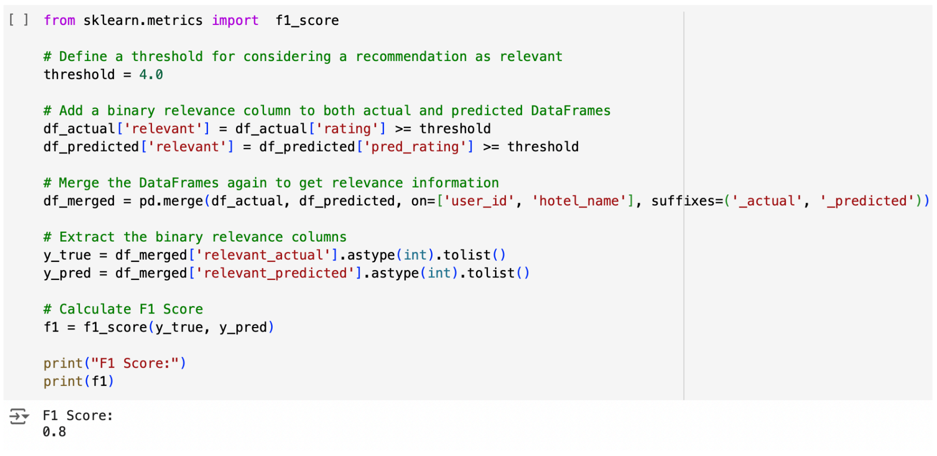


```
[ ] from sklearn.metrics import precision_score

    # Define a threshold for considering a recommendation as relevant
    threshold = 4.0

    # Add a binary relevance column to both actual and predicted DataFrames
    df_actual['relevant'] = df_actual['rating'] >= threshold
    df_predicted['relevant'] = df_predicted['pred_rating'] >= threshold

    # Merge the DataFrames again to get relevance information
    df_merged = pd.merge(df_actual, df_predicted, on=['user_id', 'hotel_name'], suffixes=('_actual', '_predicted'))

    # Extract the binary relevance columns
    y_true = df_merged['relevant_actual'].astype(int).tolist()
    y_pred = df_merged['relevant_predicted'].astype(int).tolist()

    # Calculate Precision
    precision = precision_score(y_true, y_pred)

    print("Precision:")
    print(precision)
```

```
Precision:
1.0
```

***IMAGE** 22 - PRECISION METRIC RESULTS*

C) F1 Score



```
[ ] from sklearn.metrics import  f1_score

    # Define a threshold for considering a recommendation as relevant
    threshold = 4.0

    # Add a binary relevance column to both actual and predicted DataFrames
    df_actual['relevant'] = df_actual['rating'] >= threshold
    df_predicted['relevant'] = df_predicted['pred_rating'] >= threshold

    # Merge the DataFrames again to get relevance information
    df_merged = pd.merge(df_actual, df_predicted, on=['user_id', 'hotel_name'], suffixes=('_actual', '_predicted'))

    # Extract the binary relevance columns
    y_true = df_merged['relevant_actual'].astype(int).tolist()
    y_pred = df_merged['relevant_predicted'].astype(int).tolist()

    # Calculate F1 Score
    f1 = f1_score(y_true, y_pred)

    print("F1 Score:")
    print(f1)
```

```
F1 Score:
0.8
```

IMAGE 23 - F1 SCORE RESULTS

References

- GitHub. (n.d.). Data-Cleaning-With-Python/README.md at master · HarunMbaabu/Data-Cleaning-With-Python. [online] Available at: https://github.com/HarunMbaabu/Data-Cleaning-With-Python/blob/master/README.md [Accessed 1 Aug. 2024].
- Programming, P. (2023). Data Cleaning Techniques with Python. [online] Medium. Available at: https://medium.com/@LearnPythonProgramming/data-cleaning-techniques-with-python-2debc7c0eb10.
- Zhao, A. (2022). adashofdata/nlp-in-python-tutorial. [online] GitHub. Available at:https://github.com/adashofdata/nlp-in-python-tutorial/blob/master/1-Data-Cleaning.ipynb.
- GitHub. (n.d.). TextSentimentAnalysis/Model_Code.md at main · khanfarhan10/TextSentimentAnalysis. [online] Available at: https://github.com/khanfarhan10/TextSentimentAnalysis/blob/main/Model_Code.md [Accessed 2 Aug. 2024].
- Arora, S. (2022). Sentiment Analysis Using Python. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/.
- Cheng, R. (2023). Understanding TF-IDF: A Traditional Approach to Feature Extraction in NLP. [online] Medium. Available at: https://towardsdatascience.com/understanding-tf-idf-a-traditional-approach-to-feature-extraction-in-nlp-a5bfbe04723f.
- ncsaayali (2021). Feature Engineering-Count & Tfidf vectorizer. [online] Kaggle.com. Available at: https://www.kaggle.com/code/ncsaayali/feature-engineering-count-tfidf-vectorizer [Accessed 2 Aug. 2024].
- GitHub. (n.d.). collaborative-filtering-python/collaborative-filtering-model-based.ipynb at master · klaudia-nazarko/collaborative-filtering-python. [online] Available at: https://github.com/klaudia-nazarko/collaborative-filtering-python/blob/master/collaborative-filtering-model-based.ipynb [Accessed 2 Aug. 2024].

- Amy @GrabNGoInfo (2022). Recommendation System: User-Based Collaborative Filtering. [online] GrabNGoInfo. Available at: https://medium.com/grabngoinfo/recommendation-system-user-based-collaborative-filtering-a2e76e3e15c4.
- devalindey (2024). Recommender-Systems-using-Word-Embeddings/Recommender.ipynb at master · devalindey/Recommender-Systems-using-Word-Embeddings. [online] GitHub. Available at: https://github.com/devalindey/Recommender-Systems-using-Word-Embeddings/blob/master/Recommender.ipynb [Accessed 4 Aug. 2024].
- Haidar Joumaa (2024). Building Recommendation Systems: Content-Based, Collaborative Filtering, Integration of LLM. [online] Medium. Available at: https://medium.com/@haidar.joumaa/building-recommendation-systems-content-based-collaborative-filtering-integration-of-llm-9e889edaf006
- Anon, (2020). RMSE - Root Mean Square Error in Python - AskPython. [online] Available at: https://www.askpython.com/python/examples/rmse-root-mean-square-error.
- scikit-learn. (2024). precision_score. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html# [Accessed 6 Aug. 2024].
- Grossman, M. (2023). Accuracy, Recall, Precision, & F1-Score with Python. [online] Medium. Available at: https://medium.com/@maxgrossman10/accuracy-recall-precision-f1-score-with-python-4f2ee97e0d6.