Team 6 Final Capstone Project:

European Vacation Hotel Recommender

Presented by: Addis Ashenafi, Carlos Capado, Neil Ralston

Introduction:

After 26 weeks of Bootcamp, we are ready to travel abroad for some rest and relaxation! A weekend trip to Europe sounds refreshing – but selecting a hotel seems daunting as there are so many to choose from!

Never fear, your cohorts on Team 6 have developed an application to help simplify the hotel selection process. Using a Kaggle dataset containing over 500,000 customer reviews of nearly 1,500 luxury hotels across Europe, our application processes customer review comments to recommend other hotels with similar reviews.

Our inspiration for this project came from two primary sources: the Goodreads book recommender application developed by a predecessor bootcamp team and an example hotel recommendation system article provided on the Clever Programmer website.

The expected behavior from our application is that with a user-defined input – a target hotel name – similar hotels based on customer review comments will be returned.

Methodology:

Our application uses a Natural Language Processing (NLP) machine learning model to process customer review comments and determine similarity patterns within a content-based model.

According to our Bootcamp course module, NLP is defined as:

> **Natural language processing (NLP)** is a growing field of study that combines linguistics and computer science for computers to understand written, spoken, and typed natural language. NLP is the process of converting normal language to a machine-readable format, which allows a computer to analyze text as if it were numerical data.

The process breaking NLP down into a series of smaller, less-complex tasks is referred to as the "NLP Pipeline". The NLP Pipeline from the course module is shown below:

Raw Text:

Due to the large file size of the raw dataset (over 230mb), it was uploaded to an Amazon S3 bucket.

The raw dataset included the following columns:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 515738 entries, 0 to 515737
Data columns (total 17 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   Hotel_Address                               515738 non-null  object
 1   Additional_Number_of_Scoring                515738 non-null  int64
 2   Review_Date                                 515738 non-null  object
 3   Average_Score                               515738 non-null  float64
 4   Hotel_Name                                  515738 non-null  object
 5   Reviewer_Nationality                        515738 non-null  object
 6   Negative_Review                             515738 non-null  object
 7   Review_Total_Negative_Word_Counts           515738 non-null  int64
 8   Total_Number_of_Reviews                     515738 non-null  int64
 9   Positive_Review                             515738 non-null  object
 10  Review_Total_Positive_Word_Counts           515738 non-null  int64
 11  Total_Number_of_Reviews_Reviewer_Has_Given  515738 non-null  int64
 12  Reviewer_Score                              515738 non-null  float64
 13  Tags                                        515738 non-null  object
 14  days_since_review                           515738 non-null  object
 15  lat                                         512470 non-null  float64
 16  lng                                         512470 non-null  float64
dtypes: float64(4), int64(5), object(8)
memory usage: 66.9+ MB
```

At the outset of the project, the following adjustments were made to clean the raw dataset:

- Converted Review_Date to a date/time format
- Dropped rows missing latitude (lat) and longitude (lng) coordinate data
    - Fewer than one percent of rows were dropped
- Replaced the country name “United Kingdom” with “UK”
- Added a new column containing the hotel country only (as extracted from the hotel address)
- Added a new column with overall rating sentiment based on binned review scores
- Replaced “No Negative” and “No Positive” review comments with “None”

The cleaned dataset advanced into the next phase of the NLP Pipeline included the following columns:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 512470 entries, 0 to 515737
Data columns (total 19 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   Hotel_Address                               512470 non-null  object
 1   Additional_Number_of_Scoring                512470 non-null  int64
 2   Review_Date                                 512470 non-null  datetime64[ns]
 3   Average_Score                               512470 non-null  float64
 4   Hotel_Name                                  512470 non-null  object
 5   Reviewer_Nationality                        512470 non-null  object
 6   Negative_Review                             512470 non-null  object
 7   Review_Total_Negative_Word_Counts           512470 non-null  int64
 8   Total_Number_of_Reviews                     512470 non-null  int64
 9   Positive_Review                             512470 non-null  object
 10  Review_Total_Positive_Word_Counts           512470 non-null  int64
 11  Total_Number_of_Reviews_Reviewer_Has_Given  512470 non-null  int64
 12  Reviewer_Score                              512470 non-null  float64
 13  Tags                                        512470 non-null  object
 14  days_since_review                           512470 non-null  object
 15  lat                                         512470 non-null  float64
 16  lng                                         512470 non-null  float64
 17  Hotel_Country                               512470 non-null  object
 18  Sentiment                                   512470 non-null  category
dtypes: category(1), datetime64[ns](1), float64(4), int64(5), object(8)
memory usage: 74.8+ MB
```

Tokenization and Stop Word Filtering:

The following adjustments were made to the cleaned dataset before proceeding with tokenizing and filtering:

- Positive and negative review comments were combined into one column named “All_Reviews”.
- Only the first review comment for each hotel was kept; all duplicate instances of a hotel name were dropped. This greatly simplified the dataset for use in downstream NLP analysis tasks.

The next step was to extract keywords from the combined “All_Reviews” column using the Rapid Automatic Keyword Extraction (RAKE) from the NLTK library. This process removed stopwords (words that have little or no linguistic value), white space, and punctuation from the text in the “All_Reviews” column – leaving a clean set of keywords.

To prepare for final processing, the keywords were further refined as follows:

- Tokenizing: separating the words from paragraphs or sentences, into individual words.
- Stemming: a technique used to extract the base form of the words by removing affixes from them.
    - Example: the stem of the words eating, eats, and eaten is eat.
- Lemmatizing: the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.$
    - Example: runs, running, ran are all forms of the word run, therefore run is the lemma of all these words

A sample of the key word development process is shown below:

All_Reviews	key_words	name_key_words	clean_keywords
Only the park outside of the hotel was beauti...	[park, outside, hotel, beautiful, angry, made,...	[hotel, arena]	park outside hotel beautiful angry made post a...
Very comfortable beds smart bathroom good sho...	[comfortable, beds, smart, bathroom, good, sho...	[k, hotel, george]	comfortable beds smart bathroom good shower lo...
They have followed through special requests C...	[followed, special, requests, comfortable, bed...	[apex, temple, court, hotel]	followed special requests comfortable bed wide...
The size of the room- The first night we were...	[size, room, first, night, given, extremely, s...	[park, grand, london, paddington]	size room first night given extremely small ba...
Nice hotel Room was beautiful and bed very co...	[nice, hotel, room, beautiful, bed, comfortabl...	[monhotel, lounge, spa]	nice hotel room beautiful bed comfortable expe...

Term Frequency/Inverse Document Frequency (TF-IDF) Processing:

TF-IDF is a statistical weight showing the importance of a word in a document. The TF-IDF algorithm uses a relatively simple but intuitive approach to weighting words, making it a great starting point for our application.

Term frequency (TF) measures the frequency of a word occurring in a document, while **inverse document frequency (IDF)** measures the significance of a word across a set of documents.

Multiplying these two numbers determines the TF-IDF score. The following explanation is provided in the module:

> Consider the previous example of the technology article. If "Python" appears 20 times in a 1,000-word article, the TF weight would be 20 divided by 1,000 which equals 0.02. Assume that this article resides in a database with other articles, totaling 1,000,000 articles.
>
> IDF takes the number of documents that contain the word Python and compares to the total number of documents, then takes the logarithm of the results. For example, if Python is mentioned in 1000 articles and the total amount of articles in the database is 1,000,000, the IDF would be the log of the total number of articles divided by the number of articles that contain the word Python.
>
> IDF = log(total articles / articles that contain the word Python)
>
> IDF = log(1,000,000 / 1000) = 3
>
> Remember that the log (in base 10) of a number is the power of 10 that raises 10 to that number. So, since 1,000,000 / 1,000 = 1000, and 10 raised to the 3rd power is 1000, the log of 1000 is therefore 3. Now that we have both numbers, we can determine the TF-IDF which is the multiple of the two.
>
> TF-IDF = TF * IDF = 0.2 * 3 = .6

Next, the data needs to be converted into a numerical vector. TF-IDF vectorization is accomplished by determining the TF-IDF score for each word in a dataset, and then transforming the data into a vector. With these vectors, similarity between datasets can be determined by comparing TF-IDF vectors using cosine similarity.

In our application, cosine similarity is used to measure the similarity of the TF-IDF scores of keywords derived from the hotel review comments.

The resulting cosine similarity matrix is shown below:

```
#calculate cosine matrix
cosine_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)
print(cosine_similarities)
```

```
C:\Users\neil.ralston.MAC\Anaconda3\envs\PythonData\lib\site-packages\sklearn\utils\validation.py:598: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
  FutureWarning,
```

```
[[1.         0.02775123 0.         ... 0.04576608 0.01198679 0.01373626]
 [0.02775123 1.         0.01933517 ... 0.00948075 0.         0.0231052 ]
 [0.         0.01933517 1.         ... 0.         0.         0.        ]
 ...
 [0.04576608 0.00948075 0.         ... 1.         0.         0.00624459]
 [0.01198679 0.         0.         ... 0.         1.         0.00428577]
 [0.01373626 0.0231052  0.         ... 0.00624459 0.00428577 1.        ]]
```

The following link provides access to an excellent article that explains further details about TF-IDF:

https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

Machine Learning:

Using the following code, our content filtering recommender model uses the TF-IDF scores and resulting cosine similarity calculations to determine which hotels have comparable review comments to a target selection.

```
#get recommendations based on tf-idf and cosine_similarities (hotel name)

def recommend_name(Hotel_Name, cosine_similarities = cosine_similarities):

    recommended_hotel = []
    recommended_hotel_address = []
    recommended_hotel_rating = []
    recommended_hotel_score = []

    idx = indices[indices == Hotel_Name].index[0]

    score_series = pd.Series(cosine_similarities[idx]).sort_values(ascending = False)

    top_10_indices = list(score_series.iloc[1:11].index)

    for i in top_10_indices:
        recommended_hotel.append(list(hotel_sub_df['Hotel_Name'])[i])
        recommended_hotel_address.append(list(hotel_sub_df['Hotel_Address'])[i])
        recommended_hotel_rating.append(list(hotel_sub_df['Average_Score'])[i])
        recommended_hotel_score.append(score_series[i])

    data = {'Hotel': recommended_hotel,
           'Address': recommended_hotel_address,
            'Rating': recommended_hotel_rating,
            'Score': recommended_hotel_score
                      }

    rec_df = pd.DataFrame(data)

    return rec_df
```

The cosine similarity matrix was saved as a NumPy array (.npy) in an AWS S3 bucket and accessed via s3fs, which provides a direct mapping of S3 to a local filesystem.

As an example of the output provided by our model, if the Senator Hotel Vienna is the selected target, the hotel with the highest review similarity score (with a minimum user rating of 8.0) is the Guitart Grand Passage.

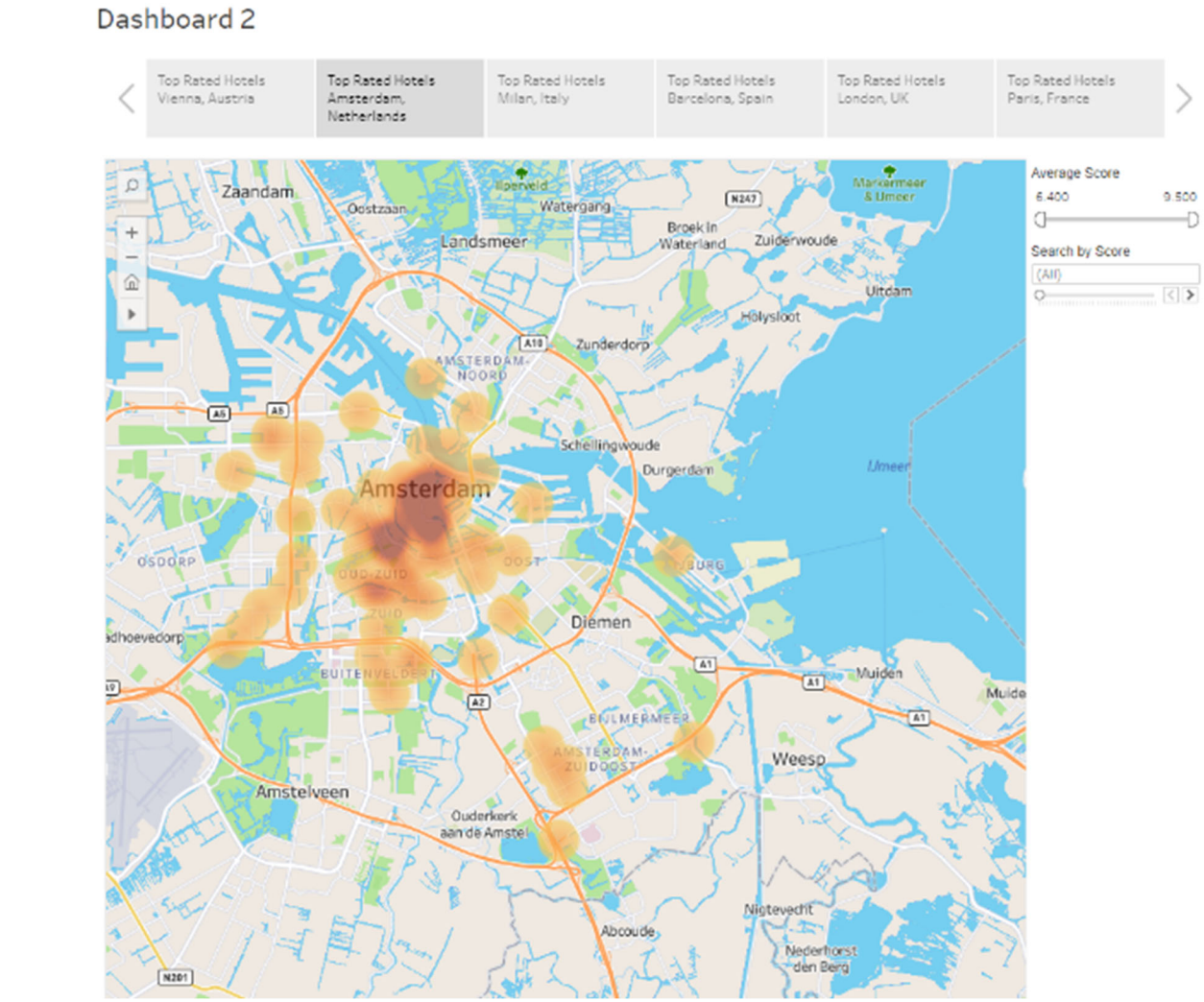
```
rec_filter_df = recommend_name('Senator Hotel Vienna')
rec_filter_df = rec_filter_df[(rec_filter_df['Rating'] > 8)]
rec_filter_df.head()
```

	Hotel	Address	Rating	Score
0	Guitart Grand Passage	Muntaner 212 Eixample 08036 Barcelona Spain	8.1	0.203923
1	Best Western Premier Faubourg 88	88 rue du Faubourg Poissoni re 10th arr 75010 ...	8.4	0.160422
4	Hotel Xenia Autograph Collection	160 Cromwell Road Kensington Kensington and Ch...	9.0	0.128471
5	H tel Pont Royal	5 7 Rue De Montalembert 7th arr 75007 Paris Fr...	8.6	0.128003
6	The Grosvenor	101 Buckingham Palace Road Westminster Borough...	8.4	0.121670

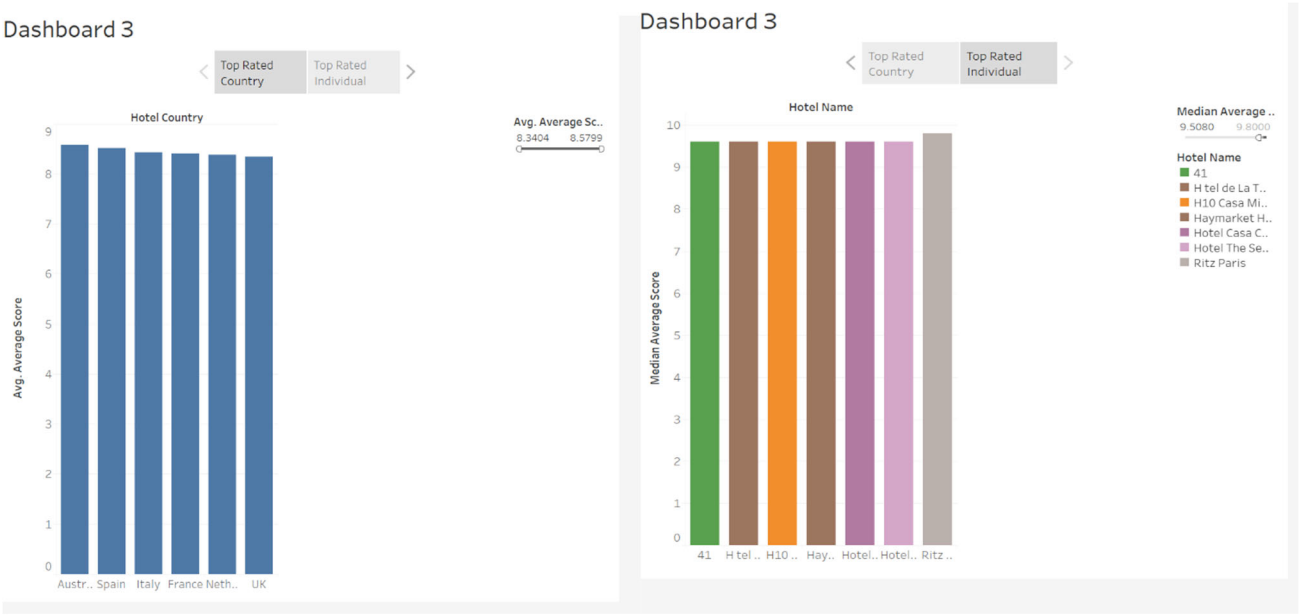
Visualization Development:

One of the most important aspects of data analysis is being able to tell a story through data visualizations. As such, we created several dashboards which assist us in telling a story for users trying to find the perfect hotel to stay at in Europe. In the first illustration, we can see all the hotels in Europe which have been rated. Through the filters, it is easy to parse out hotels by rating by average score.

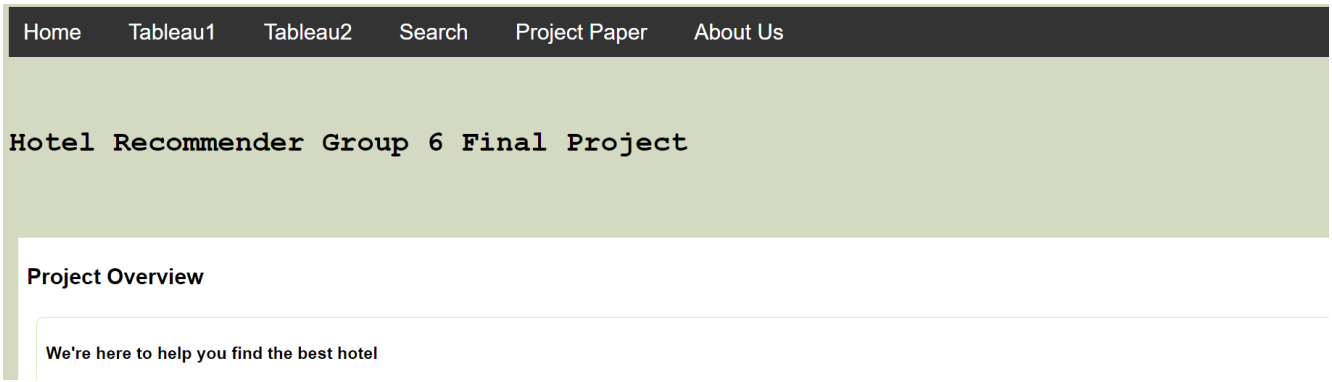
In the following dashboards we visualized top rated hotels in individual Countries throughout Europe which greatly help narrow down the search. Additionally, the heat density maps are equipped with filters to be able to search by average score as well as a more detailed individual score. There were of 6 individual heat density maps created for this project. An example heat density map is shown below.



In the last dashboard, we visualized the data via bar graph to display the top-rated country for hotel stays based on the average score; additionally, we also visualized the top hotel overall in Europe based on average reviews.



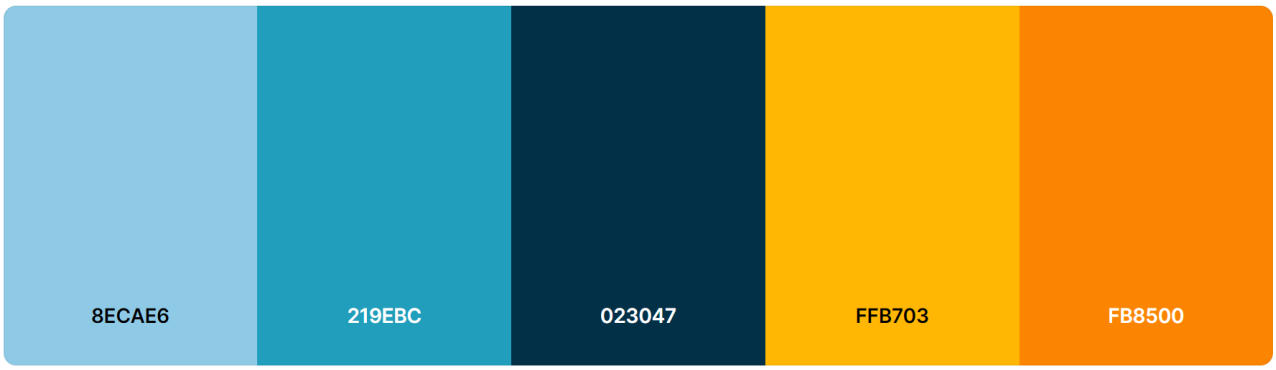
Website Development:



Our website consists of six pages, as follows:

- Home page with introductory material and an interactive matrix to view hotels based on selected country, rating, and number of review filters.
- Hotel recommender search page where the user can input a target hotel name and see other hotels with similar review comments.
- Two pages of Tableau visualizations to provide context to the dataset. The visualizations are interactive for the user to interrogate the data.
- An embedded copy of our final Project Report.
- Brief introductions about the project team.

We selected the following color palette that uses vibrant colors to represent the equally vibrant travel destinations in Europe represented by the hotels in our dataset.



Limitations and Future Work:

Our original intention was to create a hotel recommender that would allow the user to input a review keyword, such as “pool” or “airport shuttle”, that would return top-ranked hotels with similar words in the review comments. However, developing a machine learning model to accomplish this task was beyond our current capabilities. Instead, our model is based on a target hotel name, from which hotels with similar review comments are identified. Adding the keyword similarity search functionality would be a logical next step for this application.

Although relatively straightforward and simple to use, our TF-IDF model has some inherent drawbacks – an inability to consider the semantic context of words, for example. Ideally, the application would add an additional

recommendation algorithm like the countvectorizer model to provide validation of the results. A sentiment analysis feature could be added to the model as well, perhaps coupled with a K-Nearest Neighbors (KNN) algorithm.

Finally, integrating the model into the Flask application proved to be more difficult than originally anticipated. As it turned out, a simple indention error in our model helper file was the culprit! We learned the hard way not only to pay attention to code content but formatting as well.

Conclusion:

Our application successfully delivers the intended result – a hotel recommender that returns results of hotels with similar review comments. Our interactive visualizations provide additional context around the data, and our website pulls together the machine learning model and visuals into a stand-alone package.

Our small-but-mighty team addressed technical challenges by working together and seeking help from the instructional team. While there is certainly room for enhancements and improvements, we are proud of what we have been able to accomplish to close out our Bootcamp experience.

If only the Griswold family had used our app during their European Vacation…