Safari Travel Advisor

Introduction, Business Statement, and Business Understanding

Introduction and Problem Statement

Traveling is one of the most cherished experiences globally, but finding the perfect destination that aligns with individual preferences, interests, or vacation goals remains a significant challenge for many. Travelers often spend hours researching potential destinations, sifting through reviews, or consulting friends and family to decide where to go. However, this process can be overwhelming due to the abundance of information available online, coupled with the difficulty of aligning their unique interests with the offerings of different destinations.

This project aims to solve this problem by leveraging machine learning to suggest and predict personalized travel destinations based on users' interests or the activities they wish to engage in during their vacations. By analyzing destination characteristics, the model can provide tailored suggestions, saving users time and effort while increasing their satisfaction with travel planning.

Stakeholders:

- Travel Enthusiasts: Individuals seeking new destinations that align with their personal interests (e.g., art lovers wanting to visit galleries, nature enthusiasts looking for scenic hikes).
- Travel Agencies and Platforms: Businesses like Expedia, Booking.com, or TripAdvisor, which can integrate this system to enhance their customer experience and increase user engagement.
- Destination Marketers: Local tourism boards or global travel organizations that can use the model to promote destinations based on specific target audience preferences.

These stakeholders would use the model to simplify decision-making, enhance customer experiences, and drive engagement or revenue growth by promoting destinations aligned with user interests.

Business Understanding

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Travel planning involves a complex interplay of preferences, budgets, and activities, often leaving individuals overwhelmed by choices or dissatisfied with their final decisions. For example, someone interested in art galleries might unknowingly miss an underrated artistic hub. Similarly, adventure seekers might struggle to identify destinations with off-the-beaten-path hiking opportunities due to limited information.

Real-World Problem: The real-world problem is the gap between the vast number of global travel destinations and the ability of travelers to identify those that best align with their personal interests and activities. This misalignment leads to dissatisfaction, wasted time, and potentially missed opportunities for both travelers and businesses.

Value Proposition: This project addresses these challenges by providing a system that:

- For Travelers: Reduces decision fatigue by offering personalized suggestions tailored to their unique interests.
- For Travel Businesses: Increases user engagement, loyalty, and potential upselling opportunities by curating destinations that resonate with users.
- For Destination Marketers: Enables targeted marketing campaigns, focusing on promoting destinations to the most relevant audiences.

By solving this problem, the project creates a win-win scenario for travelers seeking memorable experiences and businesses aiming to enhance their service offerings and revenue streams.

Objectives

The primary objective of this project is:

- To create a machine learning model that can interpret user preferences and predict suitable country destinations using text classification techniques.

The secondary objectives are:

- To analyze the common descriptors used for top destinations on travel websites, using Lonely Planet's sample data as a benchmark.
- To compare attraction distribution across countries to identify imbalances, using Lonely Planet's sample data as a benchmark.
- To determine which countries are overrepresented on travel websites.
- To analyze international travel websites' marketing of Kenyan destinations and identify popular attractions and descriptive language used.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Data Understanding

The success of this machine learning model hinges on the quality and relevance of the dataset, as it directly impacts the ability to provide accurate and meaningful suggestions. For this project, data was scraped from Lonely Planet's website, focusing on their curated list of must-see attractions across 25 countries. For example, U.S. top attractions. The dataset is well-suited to addressing the business problem because it encapsulates rich descriptive information about attractions, which is directly aligned with the model's goal of predicting the most relevant destination based on user interests. Here is the Python File showing the scraping process.

1. Dataset Size

The scraped dataset contains:

18,040 rows, representing 18,040 unique text descriptions of must-see attractions across 25 countries. This dataset size is sufficient for training a machine learning model to generalize well while covering a diverse range of attractions. Each row corresponds to a single attraction, and the dataset offers both breadth and depth, with numerous attractions for each country. This enables the model to learn the nuanced differences in attraction types and their associations with specific destinations.

2. Data Sources and Suitability

The dataset includes information about the must-see attractions in each of the 25 countries, which was scraped from a reputable travel platform, Lonely Planet. Lonely Planet is a trusted resource in the travel industry, known for its in-depth and authentic coverage of global destinations. This ensures that the dataset is both reliable and relevant for a model designed to suggest/recommend travel destinations.

Key features of the data include:

- Description (Feature): The primary input for the model, offering detailed linguistic cues about each attraction.
- Country (Target): The output of the model, representing the predicted destination for a user’s input.
- Attraction Name: Contextual information included but not used directly in the model. The description feature allows the model to capture user preferences and connect them to relevant destinations, while the country serves as the

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

interpretable classification target. The attraction descriptions serve as the core feature for the model, as they encapsulate the essence of what travelers may be seeking (e.g., cultural landmarks, artistic experiences, natural beauty). This aligns with the business problem of connecting user inputs (e.g., "art galleries" or "hiking trails") to potential destinations.

3. Utility for the Real-World Problem

The dataset is diverse and granular, with 18,040 unique attraction descriptions across 25 countries. Its richness and alignment with user interests make it suitable for creating a system that predicts destinations based on minimal user input. The data enables the model to generalize across a wide range of preferences, effectively addressing the challenge of personalized travel suggestions/recommendations.

Data Limitations

While the dataset provides a solid foundation for a destination prediction system, several limitations could impact the model's performance and generalizability:

- Imbalanced Dataset: Some countries have significantly more attractions than others, potentially biasing the model toward over-represented countries. To address this, techniques like oversampling (e.g., SMOTE) or undersampling will be applied, and evaluation metrics like F1-score will ensure fair assessment across classes.
- Non-English Text: Some descriptions contain non-English words, which may introduce noise as the primary target language is English. This will be handled by translating non-English text where feasible or filtering it out during preprocessing.
- Text Cleaning: Raw text often includes irrelevant characters, stopwords, or inconsistencies. Cleaning will involve removing punctuation, stopwords, and applying lemmatization to standardize and refine the input data.
- Limited Geographic Scope: The dataset covers only 25 countries, limiting global applicability. Future iterations can incorporate additional data from other platforms or regions to expand coverage, with potential use of transfer learning to adapt the model to new data.
- By addressing these challenges through targeted preprocessing and robust modeling strategies, the project aims to ensure accurate and scalable predictions while laying the groundwork for future enhancements.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [231…

```
# Import Statements
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px

import string
import regex as re

from nltk.corpus import stopwords
# nltk.download('stopwords')
# nltk.download('punkt')
from nltk import word_tokenize
from nltk import FreqDist

import warnings
warnings.filterwarnings('ignore')

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score, accuracy_score, f1_score, confusic
from sklearn.utils import class_weight
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin
from sklearn import set_config

from PIL import Image
from wordcloud import WordCloud
from textwrap import wrap
import joblib
```

Data Loading

We will load the dataset as obtained through the scraping process which can be accessed in this python file.

In [232…

```
df = pd.read_csv('/Users/rosew/Desktop/Moringa/phase_5/Travel-WordFinder/Dat

df.head()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[232…

	Attraction	Description	Country	Continent
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa

Explore the Data

In [233…

```
df. info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18040 entries, 0 to 18039
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Attraction   18040 non-null  object
 1   Description  18040 non-null  object
 2   Country      18040 non-null  object
 3   Continent    18040 non-null  object
dtypes: object(4)
memory usage: 563.9+ KB
```

The dataset has 18,040 columns and 4 columns (Attraction, Description, Country, and Continent)

In [234…

```
df.describe()
```

Out[234…

	Attraction	Description	Country	Continent
count	18040	18040	18040	18040
unique	17185	18024	25	7
top	Cathedral	Buddhist ruins in Si Satchanalai-Chaliang Hist...	Canada	Asia
freq	19	4	1200	4480

There are four unique columns (Attraction, Description, country, and continent) which are all non-numeric. 18,040 entries in total, with 25 unique countries and 7 unique continents.

In [235

```
df. shape
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[235…

```
(18040, 4)
```

18040 rows and 4 columns

In [236…

```
#No null values because we scraped everything ourselves. Just to double-chec
df.isna().sum()
```

Out[236…

```
Attraction     0
Description    0
Country        0
Continent      0
dtype: int64
```

There are no null values as we scaped all the data ourselves

Duplicates

In [237…

```
df.duplicated().sum()
```

Out[237…

```
9
```

There are 9 duplicates. Let's identify them below

In [238…

```
all_duplicates = df[df.duplicated(keep=False)]
print(all_duplicates)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
                                         Attraction  \
3439                                    Yuexiu Park   
3440                                    Yuexiu Park   
3479   Palace of Moon & Water Kwun Yum Temple   
3480   Palace of Moon & Water Kwun Yum Temple   
3639                             Rakadrak Hermitage   
3640                             Rakadrak Hermitage   
3679                                Huilan Pavilion   
3680                                Huilan Pavilion   
4999                               Pithoragarh Fort   
5000                               Pithoragarh Fort   
5157                          Himadri Hans Handloom   
5160                          Himadri Hans Handloom   
11559                                   Kids Market   
11560                                   Kids Market   
11637         Africville Heritage Trust Museum   
11641         Africville Heritage Trust Museum   
14359                           Cementerios 1 & 2   
14360                           Cementerios 1 & 2   

                                             Description Country  \
3439   A crenellated roadway between attractions in t...   China   
3440   A crenellated roadway between attractions in t...   China   
3479   Not to be confused with Kwun Yum Temple nearby...   China   
3480   Not to be confused with Kwun Yum Temple nearby...   China   
3639   This hermitage high above Lhasa has three simp...   China   
3640   This hermitage high above Lhasa has three simp...   China   
3679   Lit up at night, this graceful pavilion decora...   China   
3680   Lit up at night, this graceful pavilion decora...   China   
4999   This renovated historic fort was built by Gurk...   India   
5000   This renovated historic fort was built by Gurk...   India   
5157   Just north of town on the road to Binsar is th...   India   
5160   Just north of town on the road to Binsar is th...   India   
11559  A kaleidoscopic mini shopping mall for under-1...  Canada   
11560  A kaleidoscopic mini shopping mall for under-1...  Canada   
11637  Learn the story of Africville, Halifax's predo...  Canada   
11641  Learn the story of Africville, Halifax's predo...  Canada   
14359  The city's most illustrious, influential and i...   Chile   
14360  The city's most illustrious, influential and i...   Chile   

           Continent  
3439            Asia  
3440            Asia  
3479            Asia  
3480            Asia  
3639            Asia  
3640            Asia  
3679            Asia  
3680            Asia  
4999            Asia  
5000            Asia  
5157            Asia  
5160            Asia  
11559  North America  
11560  North America  
11637  North America  
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
11641  North America
14359  South America
14360  South America
```

The duplicated attractions contains the same exact information, so we can drop them from the dataframe

In [239…
```
df = df.drop_duplicates()
```

In [240…
```
all_duplicates = df[df.duplicated(keep=False)]
print(all_duplicates)
```

```
Empty DataFrame
Columns: [Attraction, Description, Country, Continent]
Index: []
```

There are now no duplicates

Class Imbalance

In [ ]:
```
# Displaying the total unique countries
display(df.Country.unique())
print('Total Unique Countries:', len(df.Country.unique()))
```

```
array(['Kenya', 'South Africa', 'Egypt', 'Morocco', 'Japan', 'China',
       'India', 'Thailand', 'France', 'Italy', 'Germany', 'United States',
       'Canada', 'Mexico', 'Brazil', 'Argentina', 'Chile', 'Peru',
       'Australia', 'New Zealand', 'Fiji', 'United Arab Emirates',
       'Turkey', 'Israel', 'Jordan'], dtype=object)
Total Unique Countries: 25
```

In [ ]:
```
# Each of their value counts
df.Country.value_counts(normalize=True)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Out[ ]: Country
Germany                   0.066552
France                    0.066552
Australia                 0.066552
United States             0.066552
Japan                     0.066552
Italy                     0.066552
Canada                    0.066441
India                     0.066441
China                     0.066330
Mexico                    0.059897
Turkey                    0.057678
Thailand                  0.048805
South Africa              0.035494
Brazil                    0.033276
Egypt                     0.028839
New Zealand               0.022184
Morocco                   0.019966
Argentina                 0.019966
Peru                      0.019966
Chile                     0.017692
Israel                    0.008874
Kenya                     0.008874
Jordan                    0.008874
United Arab Emirates      0.006655
Fiji                      0.004437
Name: proportion, dtype: float64
```

```
In [ ]: # Grouping the df by country
countries = df.groupby('Country').count()
```

```
In [ ]: countries.reset_index(inplace=True)
```

```
In [ ]: # Sorting the countries by number of attractions (highest first)
sorted_countries = countries.sort_values(by='Attraction', ascending=False)
sorted_countries
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

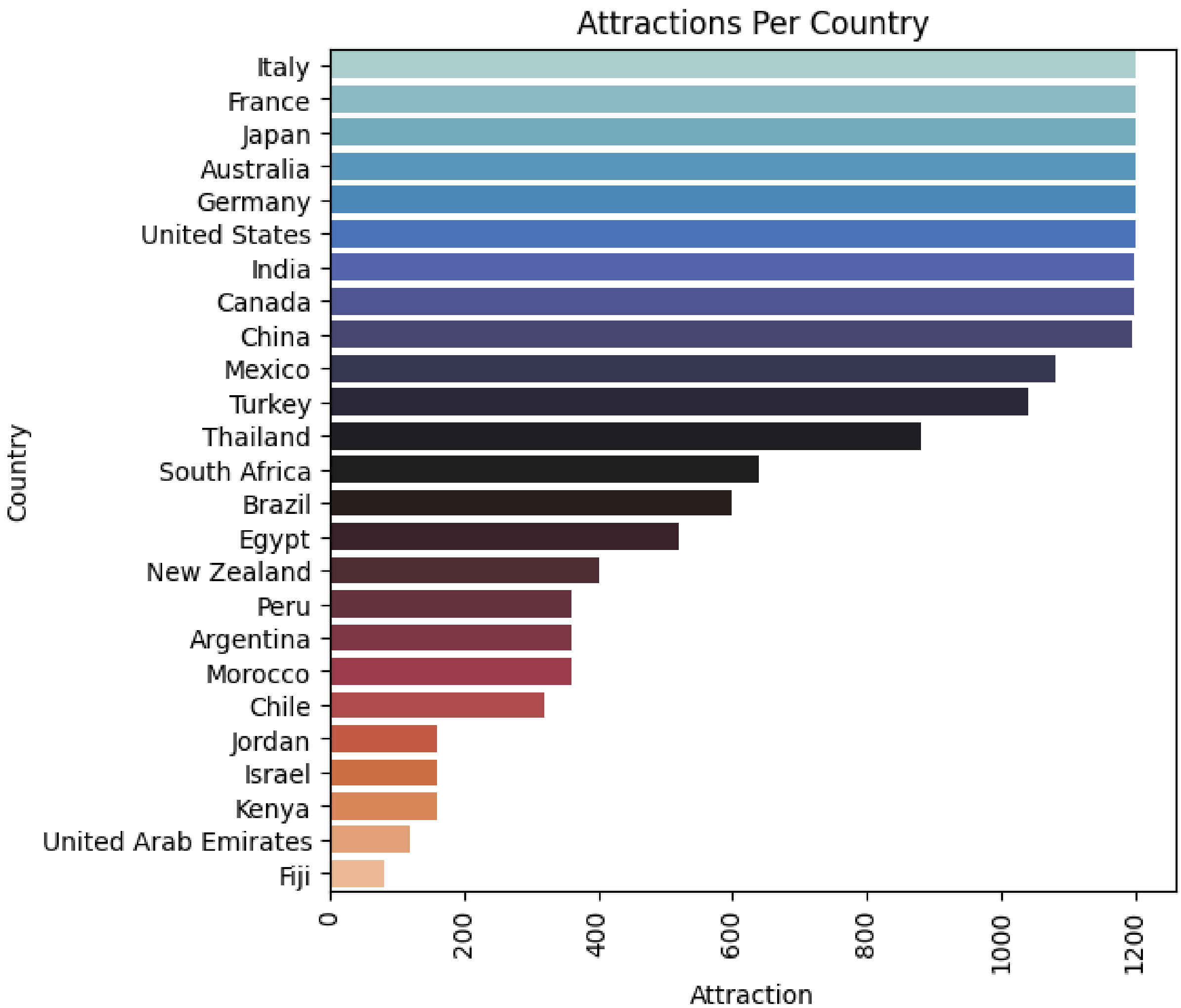
Out[ ]:

	Country	Attraction	Description	Continent
12	Italy	1200	1200	1200
8	France	1200	1200	1200
13	Japan	1200	1200	1200
1	Australia	1200	1200	1200
9	Germany	1200	1200	1200
24	United States	1200	1200	1200
10	India	1198	1198	1198
3	Canada	1198	1198	1198
5	China	1196	1196	1196
16	Mexico	1080	1080	1080
22	Turkey	1040	1040	1040
21	Thailand	880	880	880
20	South Africa	640	640	640
2	Brazil	600	600	600
6	Egypt	520	520	520
18	New Zealand	400	400	400
19	Peru	360	360	360
0	Argentina	360	360	360
17	Morocco	360	360	360
4	Chile	319	319	319
14	Jordan	160	160	160
11	Israel	160	160	160
15	Kenya	160	160	160
23	United Arab Emirates	120	120	120
7	Fiji	80	80	80

In [ ]:

```
# Plot the class imbalance for the countries
plt.figure(figsize=(6,6))
sns.barplot(x='Attraction', y='Country', data=sorted_countries, palette='ice
plt.title('Attractions Per Country')
plt.xticks(rotation=90)
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



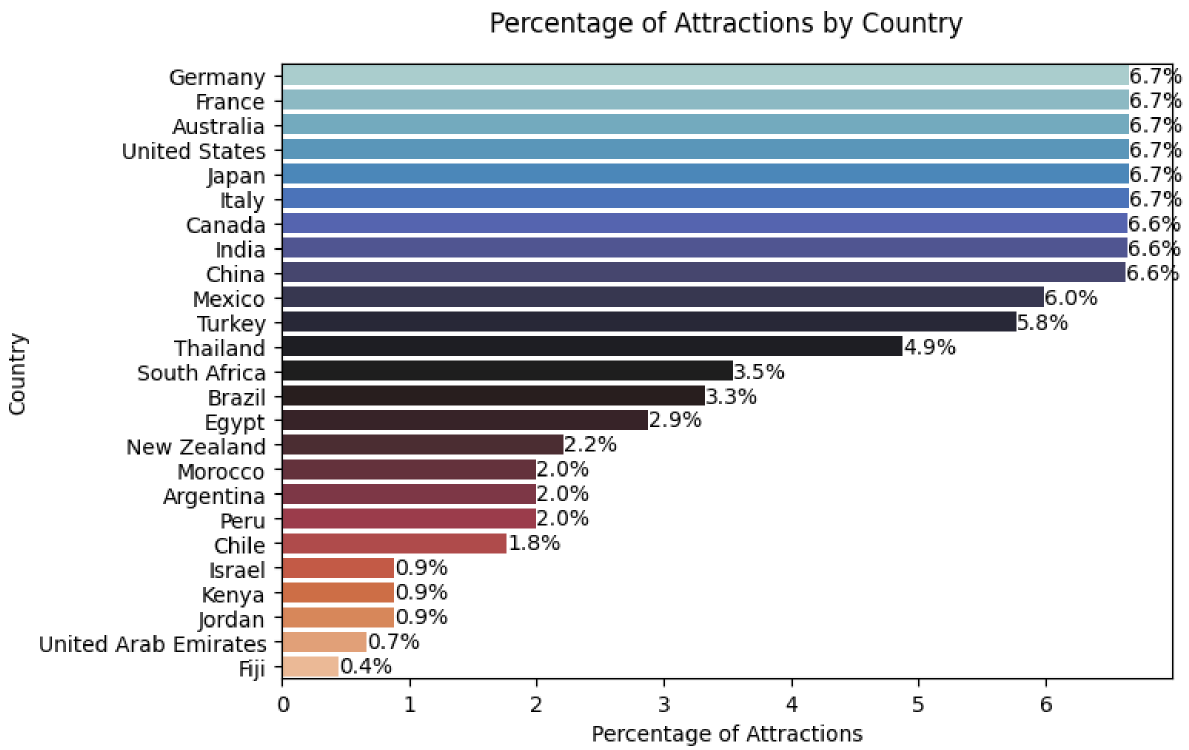
In [ ]:

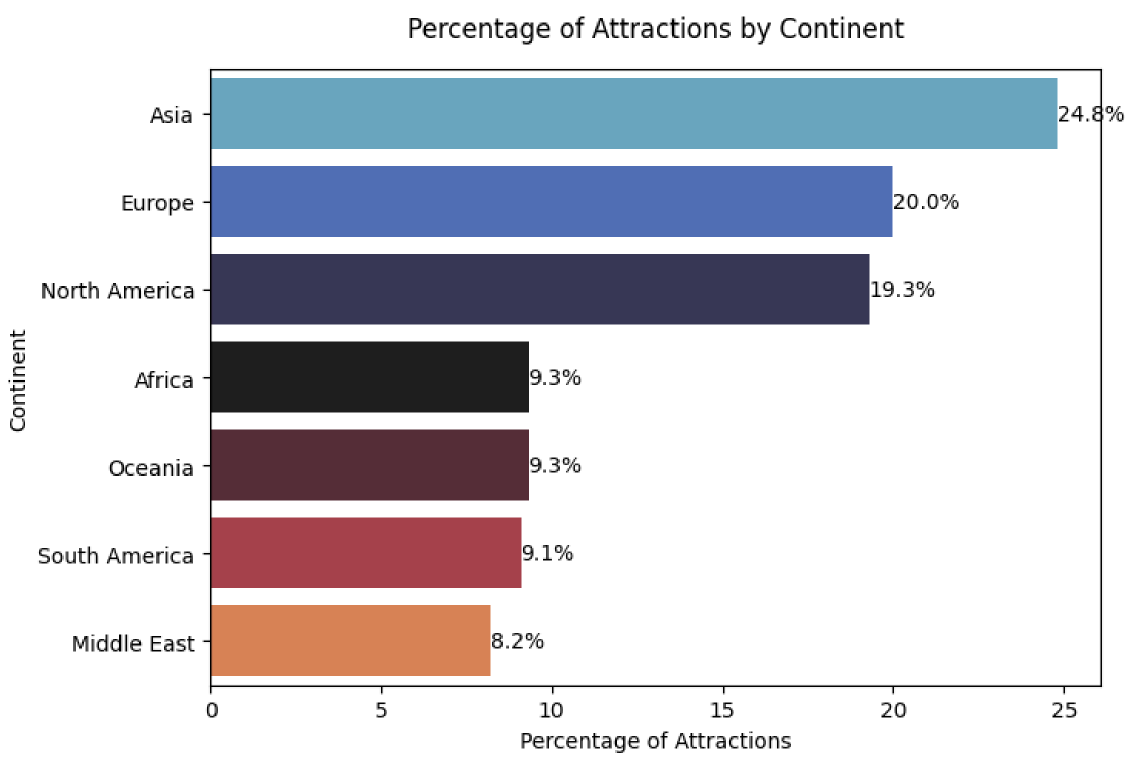
```
# Plotting class imbalance for continents
# Count attractions per country and continent
country_distribution = df['Country'].value_counts()
continent_distribution = df['Continent'].value_counts()
import matplotlib.pyplot as plt
import seaborn as sns


# Calculate percentages
country_pct = (country_distribution / country_distribution.sum() * 100)
continent_pct = (continent_distribution / continent_distribution.sum() * 100

# Create figure and subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 10))

# Plot country distribution
sns.barplot(
    x=country_pct.values,
    y=country_pct.index,
    ax=ax1,
    color='#3b82f6', palette = 'icefire'
)
ax1.set_title('Percentage of Attractions by Country', pad=15)
ax1.set_xlabel('Percentage of Attractions')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
ax1.set_ylabel('Country')
# Add percentage labels on country bars
for i in ax1.patches:
    width = i.get_width()
    ax1.text(width, i.get_y() + i.get_height()/2,
            f'{width:.1f}%',
            ha='left', va='center')

# Plot continent distribution
sns.barplot(
    x=continent_pct.values,
    y=continent_pct.index,
    ax=ax2,
    color='#8b5cf6', palette= 'icefire'
)
ax2.set_title('Percentage of Attractions by Continent', pad=15)
ax2.set_xlabel('Percentage of Attractions')
ax2.set_ylabel('Continent')

# Add percentage labels on continent bars
for i in ax2.patches:
    width = i.get_width()
    ax2.text(width, i.get_y() + i.get_height()/2,
            f'{width:.1f}%',
            ha='left', va='center')

# Adjust layout
plt.tight_layout()
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js





This will likely be an issue when modeling, so we will try to use class weights to fix this problem

Text Cleaning, Preprocessing , And Further Exploration

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js nd numbers

- Lowercasing everything
- Removing stopwords
- Creating a document term matrix grouped by Country
  - Count Vectorization
  - TF-IDF Vectorization
  - Bi-grams
- Creating a document term matrix grouped by Continent
  - Count Vectorization
  - TF-IDF Vectorization
  - Bi-grams
- Visualize most frequent words
  - Word clouds
  - Bar plot or histogram

In [248…
```
# Create a list of stop words
stopwords_list = stopwords.words('english')
stopwords_list+= list(string.punctuation)
```

In [249…
```
# Preview the list
stopwords_list[:10]
```

Out[249…
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

In [251…
```
# Lowercase all words in each corpus
df_to_clean = df.copy()
df_to_clean['Cleaned'] = df_to_clean['Description'].apply(lambda x: x.lower(
df_to_clean
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[251…

	Attraction	Description	Country	Continent	Cleaned
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa	amboseli belongs in the elite of kenya’s natio...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	this 16th-century fort and unesco world herita...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupying a plot within nairobi national park,...
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa	welcome to kenya’s most accessible yet incongr...
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa	kenya’s wonderful national museum, housed in a...
...	...	...	...	...	...
18035	Byzantine Basilica	Near the Citadel's archaeological museum is th...	Jordan	Middle East	near the citadel's archaeological museum is th...
18036	Sharif Al Hussein Bin Ali Mosque	This grand and beautiful gleaming white mosque...	Jordan	Middle East	this grand and beautiful gleaming white mosque...
18037	North Theatre	The North Theatre is overgrown and missing muc...	Jordan	Middle East	the north theatre is overgrown and missing muc...
18038	Shops	The shells of a row of shops remain in the wes...	Jordan	Middle East	the shells of a row of shops remain in the wes...
18039	Rakhabat Canyon	Close to Rum village, the labyrinthine siqs of...	Jordan	Middle East	close to rum village, the labyrinthine siqs of...

18031 rows × 5 columns

In [252…

```
# Remove commas, hyphens, colons, and other punctuation
df_to_clean['Cleaned'] = df_to_clean['Cleaned'].apply(lambda x: re.sub('[%s]
df_to_clean.head()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[252…

	Attraction	Description	Country	Continent	Cleaned
0	Amboseli National Park	Amboseli belongs in the elite of Kenya's natio...	Kenya	Africa	amboseli belongs in the elite of kenya's natio...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	this 16thcentury fort and unesco world heritag...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupying a plot within nairobi national park ...
3	Nairobi National Park	Welcome to Kenya's most accessible yet incongr...	Kenya	Africa	welcome to kenya's most accessible yet incongr...
4	National Museum	Kenya's wonderful National Museum, housed in a...	Kenya	Africa	kenya's wonderful national museum housed in an...

In [253…

```
# Use regex to get rid of numbers
df_to_clean['Cleaned'] = df_to_clean['Cleaned'].apply(lambda x: re.sub('\w*\
df_to_clean.head(10)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[253…

	Attraction	Description	Country	Continent	Cleaned
0	Amboseli National Park	Amboseli belongs in the elite of Kenya's natio...	Kenya	Africa	amboseli belongs in the elite of kenya's natio...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	this fort and unesco world heritage treasure ...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupying a plot within nairobi national park ...
3	Nairobi National Park	Welcome to Kenya's most accessible yet incongr...	Kenya	Africa	welcome to kenya's most accessible yet incongr...
4	National Museum	Kenya's wonderful National Museum, housed in a...	Kenya	Africa	kenya's wonderful national museum housed in an...
5	Giraffe Centre	This centre, which protects the highly endange...	Kenya	Africa	this centre which protects the highly endanger...
6	Lamu Museum	The best museum in town (and the second best i...	Kenya	Africa	the best museum in town and the second best in...
7	Galana River	Running through the heart of the park and mark...	Kenya	Africa	running through the heart of the park and mark...
8	Mzima Springs	Mzima Springs is an oasis of green in the west...	Kenya	Africa	mzima springs is an oasis of green in the west...
9	Ngulia Rhino Sanctuary	At the base of Ngulia Hills, this 90-sq-km are...	Kenya	Africa	at the base of ngulia hills this area is surr...

In [254…

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [273…

```
# Lemmatize the text using spacy
lemmatized = spacy.load('en_core_web_sm')

df_to_clean['Lemmatized'] = df_to_clean['Cleaned'].apply(lambda x: ' '.join(
                                    [token.lemma_ for token in list(lemmatiz
df_to_clean.head(10)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[273…

	Attraction	Description	Country	Continent	Cleaned	Lemmatized
0	Amboseli National Park	Amboseli belongs in the elite of Kenya's natio...	Kenya	Africa	amboseli belongs in the elite of kenya's natio...	amboseli belong elite kenya national park easy...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	this fort and unesco world heritage treasure ...	fort unesco world heritage treasure mombasa ...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupying a plot within nairobi national park ...	occupy plot nairobi national park nonprofit tr...
3	Nairobi National Park	Welcome to Kenya's most accessible yet incongr...	Kenya	Africa	welcome to kenya's most accessible yet incongr...	welcome kenya accessible incongruous safari ex...
4	National Museum	Kenya's wonderful National Museum, housed in a...	Kenya	Africa	kenya's wonderful national museum housed in an...	kenya wonderful national museum house impose b...
5	Giraffe Centre	This centre, which protects the highly endange...	Kenya	Africa	this centre which protects the highly endanger...	centre protect highly endanger rothschild gira...
6	Lamu Museum	The best museum in town (and the second best i...	Kenya	Africa	the best museum in town and the second best in...	good museum town second good kenya house grand...
7	Galana River	Running through the heart of the park and mark...	Kenya	Africa	running through the heart of the park and mark...	run heart park mark northernmost point park vi...
8	Mzima Springs	Mzima Springs is an oasis of green in the west...	Kenya	Africa	mzima springs is an oasis of green in the west...	mzima spring oasis green west park produce inc...
9	Ngulia Rhino Sanctuary	At the base of Ngulia Hills, this 90-sq-km are...	Kenya	Africa	at the base of ngulia hills this area is surr...	base ngulia hill area surround electric fe...

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# Group the corpora by Country and join them
df_to_group = df_to_clean[['Country', 'Lemmatized']]
df_grouped = df_to_group.groupby(by='Country').agg(lambda x:' '.join(x))
df_grouped
```

Out[ ]:

Country	Lemmatized
Argentina	earth dynamic accessible ice field glaciar per...
Australia	definitively sydney bondi world great beach cl...
Brazil	tijuca s leave atlantic rainforest surround ri...
Canada	canada sight banff national park justifiably r...
Chile	dub serengeti southern cone parque nacional ...
China	cablehaule funicular railway scale ascent hi...
Egypt	amunra local god karnak luxor new kingdom prin...
Fiji	coloisuva pronounce tholoeesoova oasis lush ...
France	fantastic space museum citys eastern outskirt ...
Germany	east gallery embodiment berlin grit gut cologn...
India	rise perpendicular impregnable rocky hill stan...
Israel	formal garden flow steep terrace resplendent...
Italy	found pope julius ii early century enlarge s...
Japan	fujisan japan revered timeless attraction insp...
Jordan	spectacular sandstone city petra build centu...
Kenya	amboseli belong elite kenya national park easy...
Mexico	tulum visit archaeological zone mexico good re...
Morocco	french fashion designer yve saint laurent part...
New Zealand	maungakiekie large spiritually significant māo...
Peru	large lake cordillera blanca — snowcappe range...
South Africa	location unique flora combine botanical gard...
Thailand	wat pho absolute favorite bangkok big sight fa...
Turkey	right heart istanbul historic center sacred b...
United Arab Emirates	burj al arabs graceful silhouette – mean evoke...
United States	story smoky mountain begin primordial time cla...

Look at different vectorization strategies

- Try different vectorization strategies and visualize them with word clouds

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- Count Vectorization
- TF-IDF Vectorization
- Bi-grams

In [257…

```
# Create a document term matrix using count vectorization
# Using count vectorization (most simple way to vectorize)
cv = CountVectorizer(analyzer='word', stop_words=stopwords_list)
data = cv.fit_transform(df_grouped['Lemmatized'])
df_dtm = pd.DataFrame(data.toarray(), columns=cv.get_feature_names_out())
df_dtm.index = df_grouped.index
df_dtm.head()
```

Out[257…

	aachen	aah	aalara	aalto	aaron	aaronsohn	aarti	aath	ab	aba
Country										
Argentina	0	0	0	0	0	0	0	0	0	
Australia	0	0	0	0	0	0	0	0	0	
Brazil	0	0	0	0	0	0	0	0	0	
Canada	0	0	0	0	0	0	0	0	0	
Chile	0	0	0	0	0	0	0	0	0	

5 rows × 32172 columns

In [ ]:

```
# Create a document term matrix using TF-IDF vectorization
# Might be good for classifying countries
tfidf = TfidfVectorizer(analyzer='word', stop_words=stopwords_list)
data2 = tfidf.fit_transform(df_grouped['Lemmatized'])
df_dtm2 = pd.DataFrame(data2.toarray(), columns=tfidf.get_feature_names_out(
df_dtm2.index = df_grouped.index
df_dtm2.head()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]:

	aachen	aah	aalara	aalto	aaron	aaronsohn	aarti	aath	ab	ab
Country										
Argentina	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Australia	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Brazil	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Canada	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Chile	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 32172 columns

Top Words with Count Vectorization

In [276…

```
# A function to Generate Word Clouds
def generate_wordcloud(data, title):
    cloud = WordCloud(width=400, height=330, max_words=150, colormap='tab20c
    plt.figure(figsize=(10,8))
    plt.imshow(cloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('\n'.join(wrap(title,60)), fontsize=13)
    plt.show()
```

In [ ]:

```
# Look at top words with count vectorizer (in total, not per country)
sum_words = data.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv.vocabulary_.item
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

Out[ ]:

```
[('museum', 2567),
 ('build', 1694),
 ('park', 1381),
 ('house', 1213),
 ('art', 1043),
 ('old', 1040),
 ('building', 992),
 ('temple', 968),
 ('city', 965),
 ('town', 962),
 ('small', 920),
 ('de', 881),
 ('large', 874),
 ('beach', 850),
 ('church', 844)]
```

One of the top words is 'km', short for kilometer which does not point to anything

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js re small, large, de(Frech for of), Include, Know, like, sq, la, di, and ad. We could consider adding these to the stop words list

Top Words with TF-IDF Vectorization

In [259…

```
# Look at top words with tf-idf vectorization (for total words, not per cour
sum_words = data2.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in tfidf.vocabulary_.i
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

Out[259…

```
[('museum', 4.338283338806132),
 ('build', 2.847216592794639),
 ('park', 2.5753723067208036),
 ('house', 2.102425356015084),
 ('de', 2.059704840493331),
 ('art', 1.7656707773228493),
 ('building', 1.7376203154296406),
 ('small', 1.6824509665578642),
 ('city', 1.6807341150324604),
 ('old', 1.6741619719375596),
 ('temple', 1.629421021562768),
 ('town', 1.6075814968969786),
 ('beach', 1.573155156763898),
 ('church', 1.5325291672646273),
 ('large', 1.4327499375576518)]
```

This is very similar to the top words to count vectorication, with words like km, de,include, being repeated. However, there is no much overlap since TF-IFD finds more words thata re unique to the countries, telling is that this is probably a better technique.

Top Bi-Grams

In [ ]:

```
cv2 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram_rang
data3 = cv2.fit_transform(df_grouped['Lemmatized'])
df_dtm3 = pd.DataFrame(data3.toarray(), columns=cv2.get_feature_names_out())
df_dtm3.index = df_grouped.index
df_dtm3
# Transposing document term matrix
df_dtm3 = df_dtm3.transpose()
# Look at top bi-grams (in total, not per country)
sum_words = data3.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv2.vocabulary_.ite
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Out[ ]: [('national park', 303),
 ('de la', 131),
 ('small museum', 129),
 ('museum house', 119),
 ('date century', 101),
 ('old town', 99),
 ('art museum', 95),
 ('sq km', 92),
 ('contemporary art', 86),
 ('build century', 86),
 ('museum display', 84),
 ('art gallery', 82),
 ('world heritage', 79),
 ('look like', 72),
 ('worth visit', 68)]
```

This gives us a better indication of the words that we should remove since they are creating noise in the data but are commonly featured in the countries. These are:

- sq, km, south, north, west, east, de, la, southeast, northeast, northwest, look, like, southwest, de, san, and northern. Now that we have confirmation, we will add them to our stop words lists to make our data cleaner for visualizations and analysis.

Removing Noise from the Data

We need to remove these words that are not unique to countries.

```
In [ ]: # let's add these words to the stopwords list
stopwords_list += ['sq', 'km', 'south', 'west', 'north', 'east', 'de', 'la',
```

```
In [ ]: # Check whether this has worked.
cv2 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram_rang
data3 = cv2.fit_transform(df_grouped['Lemmatized'])
df_dtm3 = pd.DataFrame(data3.toarray(), columns=cv2.get_feature_names_out())
df_dtm3.index = df_grouped.index
df_dtm3
# Transposing document term matrix
df_dtm3 = df_dtm3.transpose()
# Look at top bi-grams (in total, not per country)
sum_words = data3.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv2.vocabulary_.ite
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)

words_freq[:15]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Out[ ]: [('national park', 303),
 ('small museum', 129),
 ('museum house', 119),
 ('date century', 101),
 ('old town', 99),
 ('art museum', 95),
 ('contemporary art', 86),
 ('build century', 86),
 ('museum display', 84),
 ('art gallery', 82),
 ('world heritage', 79),
 ('worth visit', 68),
 ('early century', 67),
 ('buddhist temple', 66),
 ('world large', 65)]
```

Functions To Make Preprocessing Easier

```
In [ ]: def preprocess_df(df, column, preview=True, lemmatize=True):
    """
    Input df with raw text descriptions.
    Return df with preprocessed text.
    If preview=True, returns a preview of the new df.
    """


    df[column] = df['Description'].apply(lambda x: x.lower())
    df[column] = df[column].apply(lambda x: re.sub('[%s]' % re.escape(string
    df[column] = df[column].apply(lambda x: re.sub('\w*\d\w*','', x))


    if lemmatize:
        df[column]= df[column].apply(lambda x: ' '.join(
                                    [token.lemma_ for token in list(lemmatiz


    if preview:
        display(df.head(10))

    return df
```

```
In [ ]: def group_text_per_country(df, column):
    """
    Groups the preprocessed text per country.
    """
    df_to_group = df[['Country', column]]
    df_grouped = df_to_group.groupby(by='Country').agg(lambda x:' '.join(x))
    return df_grouped
```

```
In [ ]: def create_doc_term_matrix(df, column, count_vec=True, ngram_range=(1,1)):
    """
    Creates a document term matrix.
    Defaults to count vectorizer with optional n-gram param.
    If count_vec==False, uses a TF-IDF vectorizer.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    """
    df_grouped = group_text_per_country(df, column)

    if count_vec:
        vec = CountVectorizer(analyzer='word', stop_words=stopwords_list, ng
    else:
        vec = TfidfVectorizer(analyzer='word', stop_words=stopwords_list)

    data = vec.fit_transform(df_grouped[column])
    df_dtm = pd.DataFrame(data.toarray(), columns=vec.get_feature_names_out(
    df_dtm.index = df_grouped.index
    return df_dtm.transpose()
```

In [ ]:
```
preprocessed_df = preprocess_df(df, 'Lemmatized')
preprocessed_df
```

	Attraction	Description	Country	Continent	Lemmatized
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa	amboseli belong elite kenya national park easy...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	fort unesco world heritage treasure mombasa ...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupy plot nairobi national park nonprofit tr...
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa	welcome kenya accessible incongruous safari ex...
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa	kenya wonderful national museum house impose b...
5	Giraffe Centre	This centre, which protects the highly endange...	Kenya	Africa	centre protect highly endanger rothschild gira...
6	Lamu Museum	The best museum in town (and the second best i...	Kenya	Africa	good museum town second good kenya house grand...
7	Galana River	Running through the heart of the park and mark...	Kenya	Africa	run heart park mark northernmost point park vi...
8	Mzima Springs	Mzima Springs is an oasis of green in the west...	Kenya	Africa	mzima spring oasis green west park produce inc...
9	Ngulia Rhino Sanctuary	At the base of Ngulia Hills, this 90-sq-km are...	Kenya	Africa	base ngulia hill area surround electric fe...

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]:

	Attraction	Description	Country	Continent	Lemmatized
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa	amboseli belong elite kenya national park easy...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	fort unesco world heritage treasure mombasa ...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupy plot nairobi national park nonprofit tr...
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa	welcome kenya accessible incongruous safari ex...
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa	kenya wonderful national museum house impose b...
...	...	...	...	...	...
18035	Byzantine Basilica	Near the Citadel's archaeological museum is th...	Jordan	Middle East	near citadels archaeological museum small byza...
18036	Sharif Al Hussein Bin Ali Mosque	This grand and beautiful gleaming white mosque...	Jordan	Middle East	grand beautiful gleam white mosque – icon aqab...
18037	North Theatre	The North Theatre is overgrown and missing muc...	Jordan	Middle East	north theatre overgrown miss original blackbas...
18038	Shops	The shells of a row of shops remain in the wes...	Jordan	Middle East	shell row shop remain western section colonnad...
18039	Rakhabat Canyon	Close to Rum village, the labyrinthine siqs of...	Jordan	Middle East	close rum village labyrinthine siqs rakhabat c...

18031 rows × 5 columns

Visualizations for All Countries

In [ ]:

```
# Top Words After All the Preprocessing Steps (In total for all the countrie
sum_words = data.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv.vocabulary_.item
```

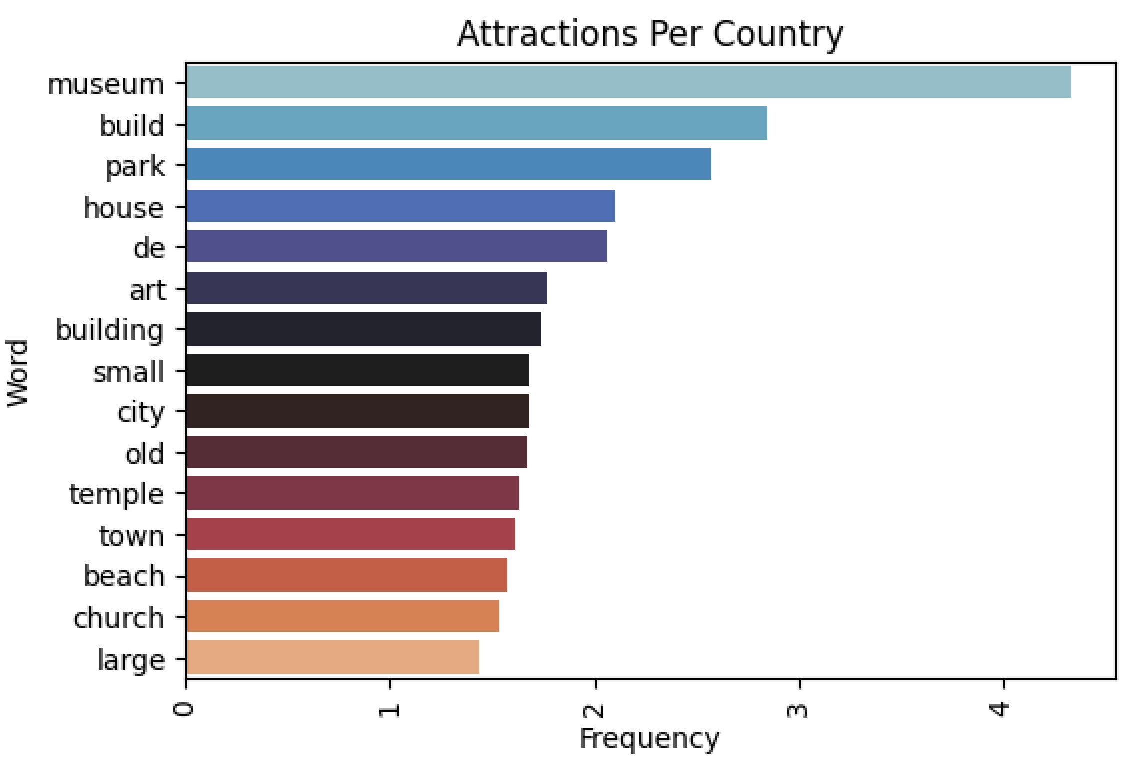
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

Out[ ]:
```
[('museum', 2567),
 ('build', 1694),
 ('park', 1381),
 ('house', 1213),
 ('art', 1043),
 ('old', 1040),
 ('building', 992),
 ('temple', 968),
 ('city', 965),
 ('town', 962),
 ('small', 920),
 ('de', 881),
 ('large', 874),
 ('beach', 850),
 ('church', 844)]
```

Top Words for All The Countries

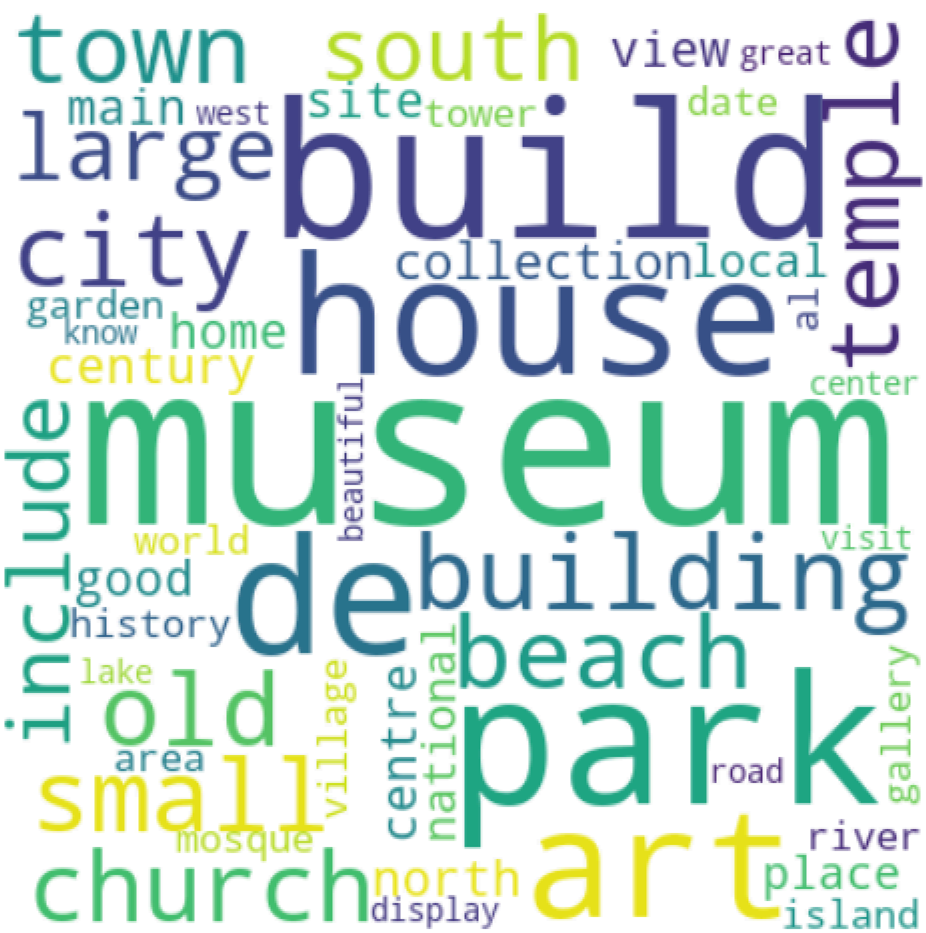
In [260…
```
# Plot the 15 top words in total
words_freq_df = pd.DataFrame(words_freq[:15], columns=['Word', 'Frequency'])
words_freq_df

plt.figure(figsize=(6,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_df, palette='icefire')
plt.title('Attractions Per Country')
plt.xticks(rotation=90)
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [261…

```python
# Word Cloud for Top 15 words in Total
%matplotlib inline
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq)

# Generate the word cloud
wordcloud = WordCloud(width=400, height=400, max_words=50, colormap='viridis

# Display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Top Bi-Grams For All the Countries

In [ ]:

```python
# Top Bi-Grams for All the Countries
cv3 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram_rang
df_grouped = group_text_per_country(preprocessed_df, 'Lemmatized')
data4 = cv3.fit_transform(df_grouped['Lemmatized'])
df_dtm4 = pd.DataFrame(data4.toarray(), columns=cv3.get_feature_names_out())
.index
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
df_dtm4
# Transposing document term matrix
df_dtm4 = df_dtm4.transpose()
# Look at top bi-grams (in total, not per country)
sum_words = data4.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv3.vocabulary_.ite
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

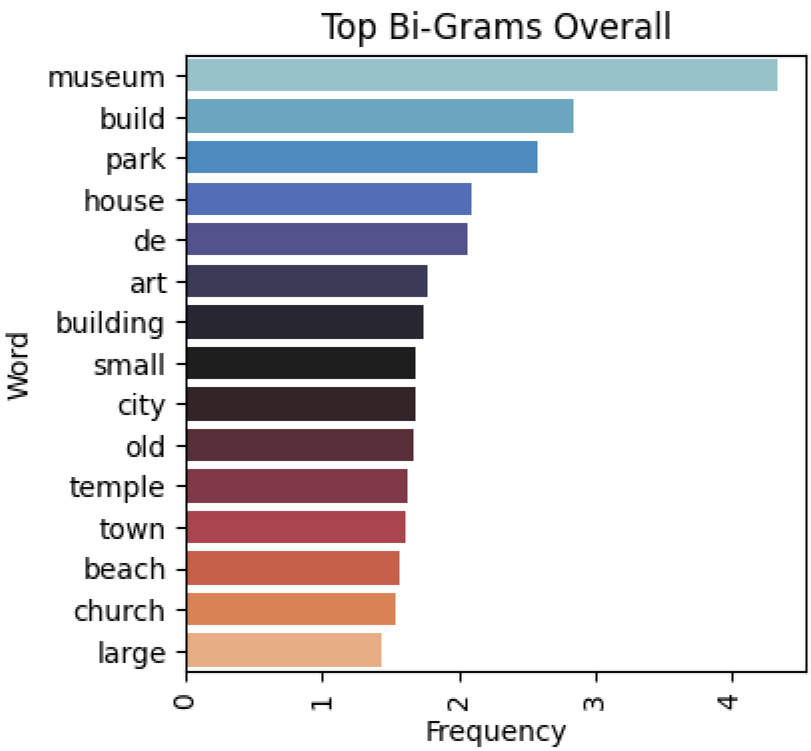
Out[ ]:

```
[('national park', 303),
 ('small museum', 129),
 ('museum house', 119),
 ('date century', 101),
 ('old town', 99),
 ('art museum', 95),
 ('contemporary art', 86),
 ('build century', 86),
 ('museum display', 84),
 ('art gallery', 82),
 ('world heritage', 79),
 ('worth visit', 68),
 ('early century', 67),
 ('buddhist temple', 66),
 ('world large', 65)]
```

In [262…

```
# Plot the 15 top Bi-Grams in total
words_freq_bi_df = pd.DataFrame(words_freq[:15], columns=['Word', 'Frequency
words_freq_bi_df

plt.figure(figsize=(4,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_bi_df, palette='icefire
plt.title('Top Bi-Grams Overall')
plt.xticks(rotation=90)
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [263…

```
# Word Cloud for Top 15 BI-Grams in Total
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq)

# Generate the word cloud
wordcloud = WordCloud(width=400, max_words=50, height=400, colormap='viridis

# Display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

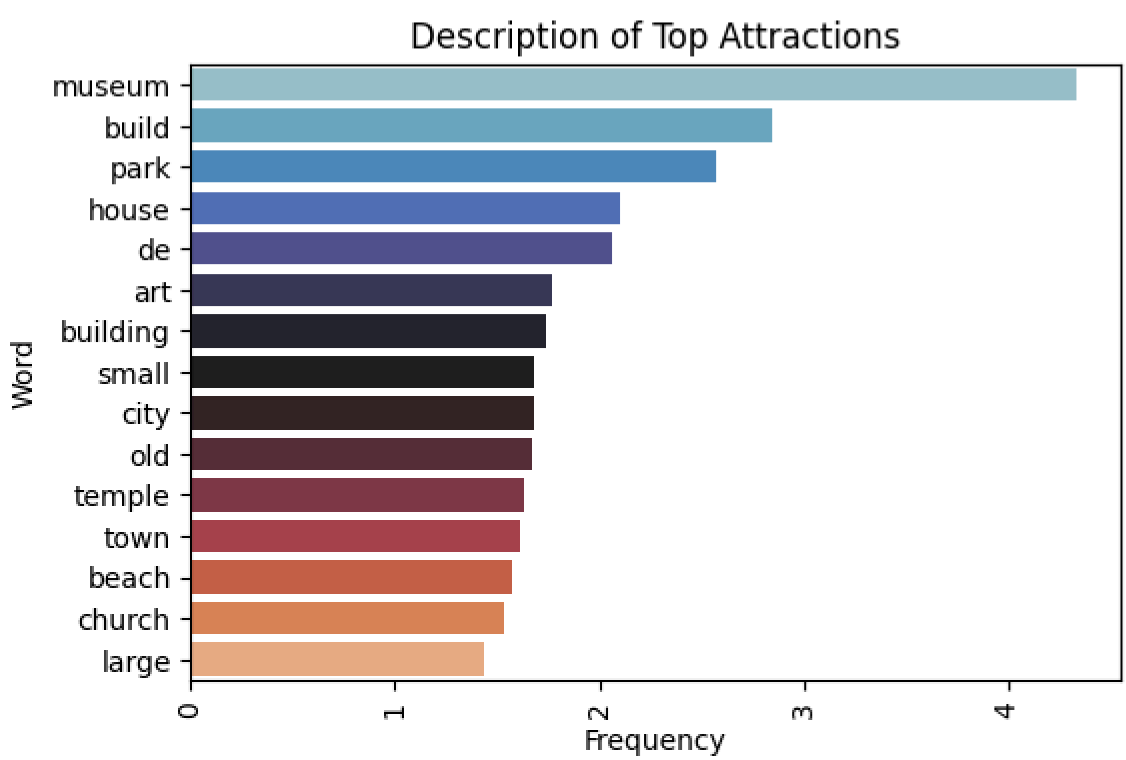
Top Tri-Grams for All Countries

In [ ]:
```
# Top Tri-Grams for All the Countries
cv4 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram_rang
df_grouped = group_text_per_country(preprocessed_df, 'Lemmatized')
data5 = cv4.fit_transform(df_grouped['Lemmatized'])
df_dtm5 = pd.DataFrame(data5.toarray(), columns=cv4.get_feature_names_out())
df_dtm5.index = df_grouped.index
df_dtm5
# Transposing document term matrix
df_dtm5 = df_dtm5.transpose()
# Look at top tri-grams (in total, not per country)
sum_words = data5.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in cv4.vocabulary_.ite
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq[:15]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Out[ ]: [('world heritage site', 47),
 ('unesco world heritage', 46),
 ('national historic site', 21),
 ('museum tell story', 20),
 ('world heritage list', 18),
 ('final resting place', 16),
 ('national park cover', 15),
 ('date early century', 14),
 ('traditional ornately decorate', 12),
 ('ornately decorate residence', 12),
 ('world large collection', 11),
 ('museum worth visit', 11),
 ('large national park', 10),
 ('art gallery house', 10),
 ('haveli traditional ornately', 10)]
```

```
In [264…
# Plot the 15 top Tri-Grams in total
words_freq_tri_df = pd.DataFrame(words_freq[:15], columns=['Word', 'Frequenc
words_freq_tri_df

plt.figure(figsize=(6,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_tri_df, palette='icefir
plt.title('Description of Top Attractions')
plt.xticks(rotation=90)
plt.show()
```




```
In [265…
# Word Cloud for Top 15 Tri-Grams in Total
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# Generate the word cloud
wordcloud = WordCloud(width=600, max_words=50, height=400, colormap='viridis

# Display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Visualizations for Kenya

In [ ]:
```
# Creating a DF for Kenya only
kenya_df = preprocessed_df[preprocessed_df['Country'] == 'Kenya']
kenya_df
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]:

	Attraction	Description	Country	Continent	Lemmatized
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa	amboseli belong elite kenya national park easy...
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa	fort unesco world heritage treasure mombasa ...
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa	occupy plot nairobi national park nonprofit tr...
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa	welcome kenya accessible incongruous safari ex...
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa	kenya wonderful national museum house impose b...
...	...	...	...	...	...
155	Malindi Museum	Part of the Malindi Historic Circuit, this mod...	Kenya	Africa	malindi historic circuit moderately interestin...
156	Lake Oloiden	Lake Naivasha may be a freshwater lake, but it...	Kenya	Africa	lake naivasha freshwater lake alkaline water n...
157	Portuguese Church	This thatched-roofed church gets its name beca...	Kenya	Africa	thatchedroofe church get portuguese explorer v...
158	Lamu Market	Atmospheric and somewhat chaotic, this quintes...	Kenya	Africa	atmospheric somewhat chaotic quintessential la...
159	Buffalo Springs National Reserve	The twin sister of Samburu National Reserve, w...	Kenya	Africa	twin sister samburu national reserve sit oppos...

160 rows × 5 columns

In [ ]:
```
data_kenya = cv.fit_transform(kenya_df['Lemmatized'])
```

In [ ]:
```
print("Shape of data_kenya:", data_kenya.shape)
print("Size of cv.vocabulary_:", len(cv.vocabulary_))
```

```
Shape of data_kenya: (160, 1134)
Size of cv.vocabulary_: 1134
```

In [ ]:
```
# Sum word occurrences across all rows
sum_words_kenya = data_kenya.sum(axis=0)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

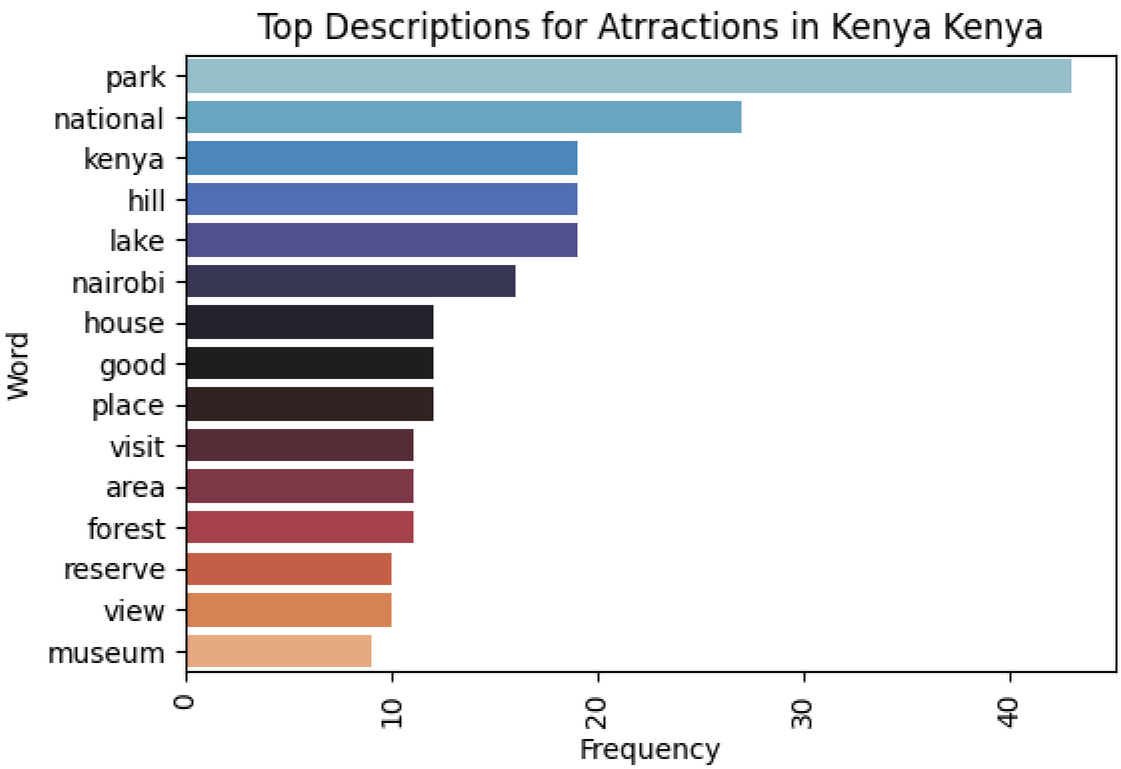
```python
# Extract word frequencies
words_freq_kenya = [(word, sum_words_kenya[0, idx]) for word, idx in cv.voca

# Sort by frequency in descending order
words_freq_kenya = sorted(words_freq_kenya, key=lambda x: x[1], reverse=True

# Display the result
words_freq_kenya[:15]
```

Out[ ]:
```
[('park', 43),
 ('national', 27),
 ('kenya', 19),
 ('hill', 19),
 ('lake', 19),
 ('nairobi', 16),
 ('house', 12),
 ('good', 12),
 ('place', 12),
 ('visit', 11),
 ('area', 11),
 ('forest', 11),
 ('reserve', 10),
 ('view', 10),
 ('museum', 9)]
```

Top Words for Kenya

In [267…
```python
# Plot the 15 top words in total for Kenya
words_freq_df_kenya = pd.DataFrame(words_freq_kenya[:15], columns=['Word', '

plt.figure(figsize=(6,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_df_kenya, palette='icef
plt.title('Top Descriptions for Atrractions in Kenya Kenya')
plt.xticks(rotation=90)
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [266…

```
# Word Cloud for Top 15 Words in Kenya
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq_kenya)

# Generate the word cloud
wordcloud = WordCloud(width=600, max_words=50, height=400, colormap='viridis

# Display the word cloud
plt.figure(figsize=(6, 4))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Top Bi-Grams for Kenya

In [ ]:

```
# Top Bi-Grams for Kenya

cv_Kenya_bi = CountVectorizer(analyzer='word', stop_words=stopwords_list, ng
kenya_grouped = group_text_per_country(kenya_df, 'Lemmatized')
data_kenya_bi = cv_Kenya_bi.fit_transform(kenya_df['Lemmatized'])
df_dtm_kenya = pd.DataFrame(data_kenya_bi.toarray(), columns=cv_Kenya_bi.get


# # Transposing document term matrix
df_dtm_kenya = df_dtm_kenya.transpose()
# # Look at top bi-grams
sum_words_bi = data_kenya_bi.sum(axis=0)
words_freq_bi = [(word, sum_words_bi[0, idx]) for word, idx in cv_Kenya_bi.v
words_freq_bi = sorted(words_freq_bi, key=lambda x: x[1], reverse=True)
words_freq_bi[:15]
```

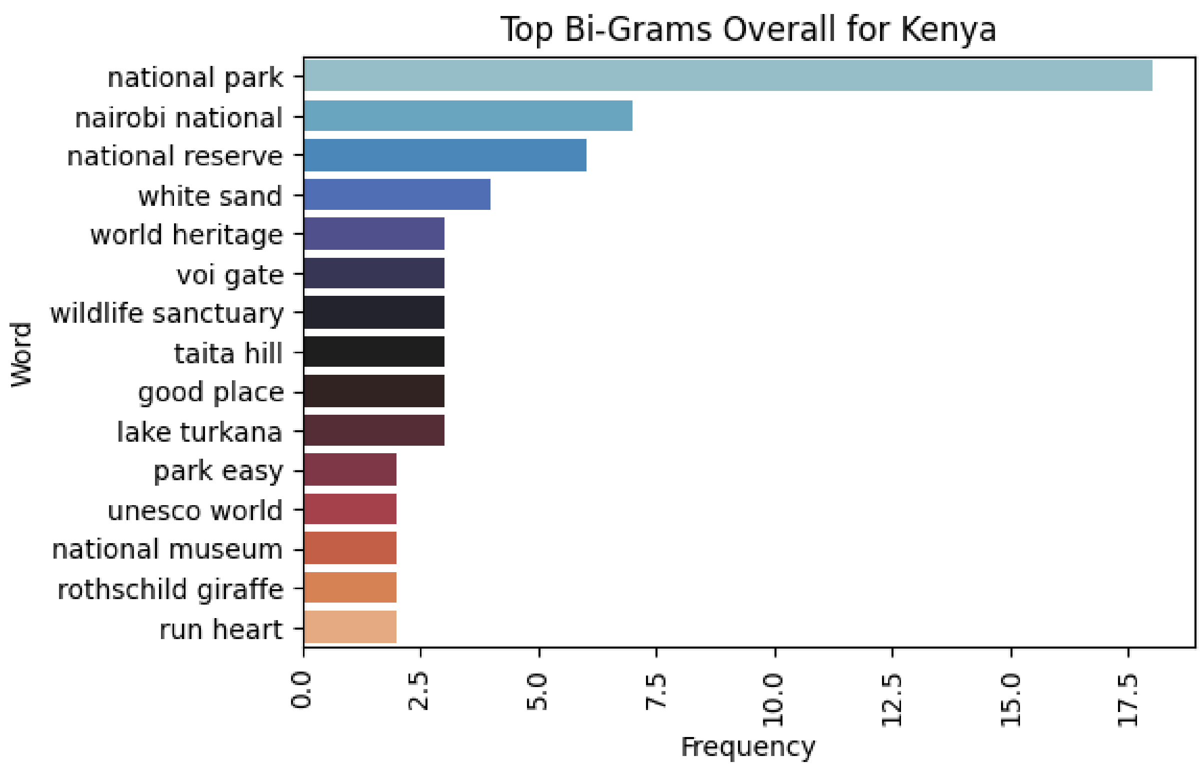
Out[ ]:

```
[('national park', 18),
 ('nairobi national', 7),
 ('national reserve', 6),
 ('white sand', 4),
 ('world heritage', 3),
 ('voi gate', 3),
 ('wildlife sanctuary', 3),
 ('taita hill', 3),
 ('good place', 3),
 ('lake turkana', 3),
 ('park easy', 2),
 ('unesco world', 2),
 ('national museum', 2),
 ('rothschild giraffe', 2),
 ('run heart', 2)]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [268…

```
# Plot the 15 top Bi-Grams in total for Kenya
words_freq_bi_df_kenya = pd.DataFrame(words_freq_bi[:15], columns=['Word', '
words_freq_bi_df_kenya

plt.figure(figsize=(6,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_bi_df_kenya, palette='i
plt.title('Top Bi-Grams Overall for Kenya')
plt.xticks(rotation=90)
plt.show()
```



In [269…

```
# Word Cloud for Top 15 Bi-grams in Kenya
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq_bi)

# Generate the word cloud
wordcloud = WordCloud(width=600, max_words=50, height=400, colormap='viridis

# Display the word cloud
plt.figure(figsize=(6, 4))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Top Tri-Grams for Kenya

In [ ]:

```
# Top Tri-Grams for Kenya


cv_Kenya_tri = CountVectorizer(analyzer='word', stop_words=stopwords_list, r
kenya_grouped = group_text_per_country(kenya_df, 'Lemmatized')
data_kenya_tri = cv_Kenya_tri.fit_transform(kenya_df['Lemmatized'])
df_dtm_kenya_tri = pd.DataFrame(data_kenya_tri.toarray(), columns=cv_Kenya_t


# # Transposing document term matrix
df_dtm_kenya_tri = df_dtm_kenya_tri.transpose()
# # Look at top bi-grams
sum_words_tri = data_kenya_tri.sum(axis=0)
words_freq_tri = [(word, sum_words_tri[0, idx]) for word, idx in cv_Kenya_tr
words_freq_tri = sorted(words_freq_tri, key=lambda x: x[1], reverse=True)
words_freq_tri[:15]
```

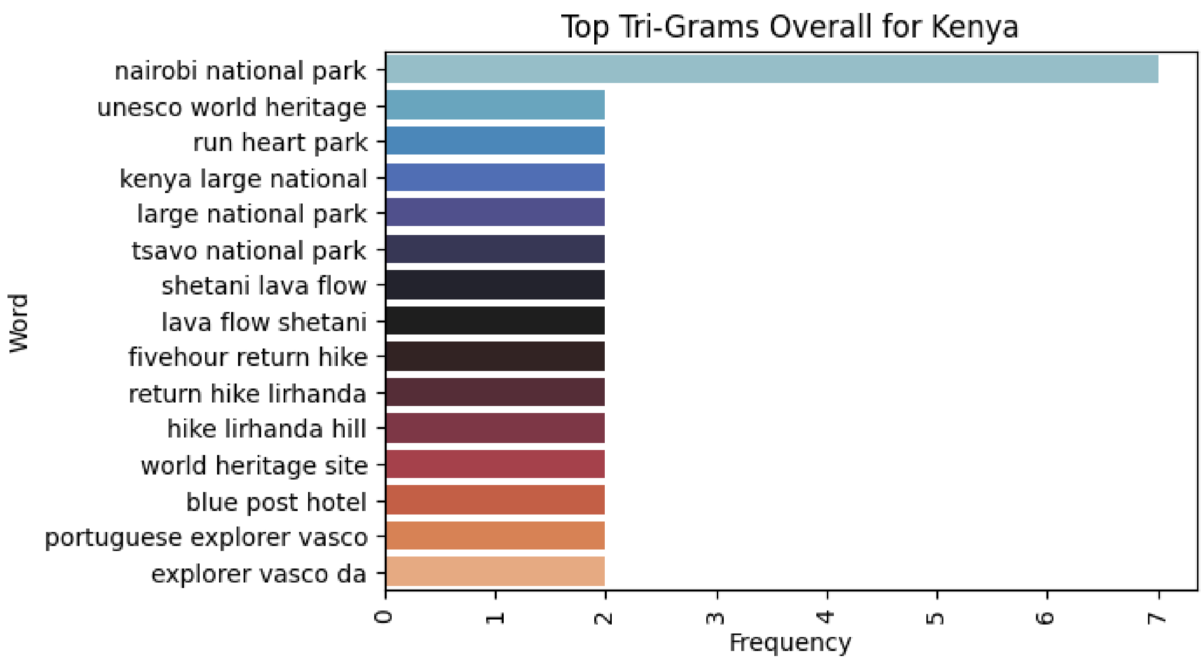
Out[ ]:

```
[('nairobi national park', 7),
 ('unesco world heritage', 2),
 ('run heart park', 2),
 ('kenya large national', 2),
 ('large national park', 2),
 ('tsavo national park', 2),
 ('shetani lava flow', 2),
 ('lava flow shetani', 2),
 ('fivehour return hike', 2),
 ('return hike lirhanda', 2),
 ('hike lirhanda hill', 2),
 ('world heritage site', 2),
 ('blue post hotel', 2),
 ('portuguese explorer vasco', 2),
 ('explorer vasco da', 2)]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [270…

```
# Plot the 15 top Tri-Grams in total for Kenya
words_freq_tri_df_kenya = pd.DataFrame(words_freq_tri[:15], columns=['Word',
words_freq_tri_df_kenya

plt.figure(figsize=(6,4))
sns.barplot(x='Frequency', y='Word', data=words_freq_tri_df_kenya, palette='
plt.title('Top Tri-Grams Overall for Kenya')
plt.xticks(rotation=90)
plt.show()
```



In [271…

```
# Word Cloud for Top 15 Tri-grams in Kenya
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Convert word frequencies into a dictionary
word_freq_dict = dict(words_freq_tri)

# Generate the word cloud
wordcloud = WordCloud(width=600, max_words=50, height=400, colormap='viridis

# Display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

EDA Conclusions and Recommendations

- Countries like Canada, India, Australia, United States, Italy, and France have the highest number of attractions, each showing around 1,000+ attractions in the dataset. These high counts likely indicate these countries have diverse or popular tourist destinations, making them significant for the tourism industry.
- Asia leads in the number of attractions, followed by Europe and North America. This could imply that Asia has a vast diversity of attractions or that it has been highly represented in this dataset.
- The higher number of attractions in continents like Asia and Europe could reflect established tourism infrastructure and popular cultural or historical sites.
- The presence of multiple attractions in diverse countries highlights a broad global interest in travel, with each continent offering unique experiences.
- Regions with fewer attractions listed, such as Africa and the Middle East, could represent untapped tourism potential. They may benefit from increased marketing efforts or infrastructure development to attract more tourists.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- Kenya is not well represented, with Nairobi National Park standing out as a phrase

Modeling

In [ ]:
```
# Re-import the data to get a fresh start
data = pd.read_csv('/Users//rosew/Downloads/best_travel_destinations_for_202
data.head()
```

Out[ ]:

	Attraction	Description	Country	Continent
0	Amboseli National Park	Amboseli belongs in the elite of Kenya’s natio...	Kenya	Africa
1	Fort Jesus	This 16th-century fort and Unesco World Herita...	Kenya	Africa
2	David Sheldrick Wildlife Trust	Occupying a plot within Nairobi National Park,...	Kenya	Africa
3	Nairobi National Park	Welcome to Kenya’s most accessible yet incongr...	Kenya	Africa
4	National Museum	Kenya’s wonderful National Museum, housed in a...	Kenya	Africa

Preprocessing and Train Test Split

In [ ]:
```
# Perform train-test split before cleaning.preprocessing
X = data['Description']
y= data['Country']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
X_train.shape, X_test.shape
```

Out[ ]:
```
((14424,), (3607,))
```

In [ ]:
```
# Since this is a series, it will need to be changed to a DF for preprocessi
X_train.head()
```

Out[ ]:
```
119      A watering hole that attracts animals, includi...
9709     This museum offers a good overview of the natu...
11414    Formed as early as 1977 from a desire to prese...
4584     One of the few nature sanctuaries within day-t...
12856    This small plantation, which produces almost-o...
Name: Description, dtype: object
```

In [ ]:
```
# Preprocessing
X_train_preprocessed = preprocess_df(pd.DataFrame(X_train, columns = ['Descr
X_test_preprocessed = preprocess_df(pd.DataFrame(X_test,  columns =['Descrip
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	Description	Lemmatized
119	A watering hole that attracts animals, includi...	watering hole attract animal include elephant ...
9709	This museum offers a good overview of the natu...	museum offer good overview natural cultural hi...
11414	Formed as early as 1977 from a desire to prese...	form early desire preserve memory story poss...
4584	One of the few nature sanctuaries within day-t...	nature sanctuary daytrip reach mumbais city li...
12856	This small plantation, which produces almost-o...	small plantation produce almostorganic shadegr...
4622	A few partially rebuilt wall stubs are all tha...	partially rebuild wall stub remain palace comp...
16990	The Roman harbour at the base of Kaleiçi's slo...	roman harbour base kaleiçis slope antalyas lif...
9767	Was it the fall of 1966 or the winter of ’67? ...	fall winter ' haight saying go remember summ...
12534	Staff at the visitors center of the stunning C...	staff visitor center stunning chipinque park o...
4282	The International Society for Krishna Consciou...	international society krishna consciousness ww...

	Description	Lemmatized
15937	About 5km south of Cooktown, this 47-hectare w...	south cooktown wetland favourite birdwatch...
7759	Nero had his Domus Aurea constructed after the...	nero domus aurea construct fire ad rumour st...
7950	A popular diving destination, these protected ...	popular diving destination protect water res...
1020	At the perennially popular Gardens there are a...	perennially popular garden actually site near ...
2864	Exhibits in this museum include the crown and ...	exhibit museum include crown personal item dai...
4463	Isa Khan was a noble of the Sher Shah era, and...	isa khan noble sher shah era grandiose afghans...
17248	Old cars and horse-drawn carts are housed in t...	old car horsedrawn cart house silk factory gar...
1870	The Five Mountains of Aso are the smaller moun...	mountain aso small mountain asosan caldera out...
5365	Standing 14m high and weighing in at 30 tonnes...	stand high weigh tonne beautiful bronze st...
15321	Family-owned winery producing award-winning ri...	familyowne winery produce awardwinne riesle sh...

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# Redefining stop words list
stopwords_list = stopwords.words('english')
stopwords_list += list(string.punctuation)
stopwords_list += ['sq', 'km', 'one','two', 'south', 'west', 'north', 'east'
```

In [ ]:
```
# Vectorize the text data to be suitable for modeling
vectorizer = TfidfVectorizer(analyzer='word', stop_words=stopwords_list, dec
# vectorizer = TfidfVectorizer(analyzer='word')
X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_tfidf = vectorizer.transform(X_test_preprocessed['Lemmatized'])
```

In [ ]:
```
def evaluate_model(model, X_train, X_test):
    y_preds_train = model.predict(X_train)
    y_preds_test = model.predict(X_test)

    print('Training Accuracy:', accuracy_score(y_train, y_preds_train))
    print('Testing Accuracy:', accuracy_score(y_test, y_preds_test))
    print("Train and Test Accuracy Difference:", accuracy_score(y_train, y_p
    print('\n---------------\n')
    print('Training F1:', f1_score(y_train, y_preds_train, average='weightec
    print('Testing F1:', f1_score(y_test, y_preds_test, average='weighted'))
    print('\n---------------\n')
    print(classification_report(y_test, y_preds_test))
```

1. Multinomial Naive Bayes(MNB)

MNB Iteration One

In [ ]:
```
nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)
```

Out[ ]:
```
▾ MultinomialNB
MultinomialNB()
```

In [ ]:
```
# View the classes
nb.classes_
```

Out[ ]:
```
array(['Argentina', 'Australia', 'Brazil', 'Canada', 'Chile', 'China',
       'Egypt', 'Fiji', 'France', 'Germany', 'India', 'Israel', 'Italy',
       'Japan', 'Jordan', 'Kenya', 'Mexico', 'Morocco', 'New Zealand',
       'Peru', 'South Africa', 'Thailand', 'Turkey',
       'United Arab Emirates', 'United States'], dtype='<U20')
```

In [ ]:
```
# Evaluate the model
evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.7425124792013311
Testing Accuracy: 0.5234266703631827
Train and Test Accuracy Difference: 0.21908580883814843

---------------

Training F1: 0.6812673313389652
Testing F1: 0.4754250157939932

---------------

                      precision    recall  f1-score   support

           Argentina       0.00      0.00      0.00        73
           Australia       0.38      0.71      0.49       252
              Brazil       1.00      0.09      0.17       127
              Canada       0.37      0.63      0.47       241
               Chile       0.00      0.00      0.00        60
               China       0.52      0.67      0.59       249
               Egypt       0.94      0.16      0.28       104
                Fiji       0.00      0.00      0.00        15
              France       0.62      0.60      0.61       245
             Germany       0.66      0.68      0.67       252
               India       0.65      0.64      0.64       255
              Israel       0.00      0.00      0.00        29
               Italy       0.64      0.72      0.68       246
               Japan       0.48      0.71      0.57       238
              Jordan       0.00      0.00      0.00        29
               Kenya       0.00      0.00      0.00        35
              Mexico       0.51      0.68      0.58       203
             Morocco       0.00      0.00      0.00        74
         New Zealand       0.00      0.00      0.00        74
                Peru       1.00      0.01      0.03        70
        South Africa       1.00      0.06      0.11       115
            Thailand       0.96      0.62      0.75       169
              Turkey       0.59      0.76      0.66       188
United Arab Emirates       0.00      0.00      0.00        24
       United States       0.39      0.58      0.47       240

            accuracy                           0.52      3607
           macro avg       0.43      0.33      0.31      3607
        weighted avg       0.54      0.52      0.48      3607
```

MNB Iteration Two- Using Count Vectorizer

In [ ]:

```
# Trying Count Vectorizer to see the difference
# Vectorize the text data to be suitable for modeling
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=stopwords_list,
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])

nb = MultinomialNB()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
nb.fit(X_train_cv, y_train)
evaluate_model(nb, X_train_cv, X_test_cv)
```

```
Training Accuracy: 0.8264004437049363
Testing Accuracy: 0.5389520377044635
Train and Test Accuracy Difference: 0.2874484060004727

---------------

Training F1: 0.8097368278257866
Testing F1: 0.5072964114748401

---------------

                      precision    recall  f1-score   support

           Argentina       1.00      0.03      0.05        73
           Australia       0.40      0.71      0.51       252
              Brazil       0.93      0.31      0.46       127
              Canada       0.39      0.62      0.48       241
               Chile       1.00      0.05      0.10        60
               China       0.53      0.65      0.59       249
               Egypt       0.89      0.46      0.61       104
                Fiji       0.00      0.00      0.00        15
              France       0.62      0.58      0.60       245
             Germany       0.63      0.65      0.64       252
               India       0.64      0.60      0.62       255
              Israel       0.00      0.00      0.00        29
               Italy       0.65      0.69      0.67       246
               Japan       0.52      0.67      0.58       238
              Jordan       0.00      0.00      0.00        29
               Kenya       0.00      0.00      0.00        35
              Mexico       0.49      0.67      0.57       203
             Morocco       1.00      0.08      0.15        74
         New Zealand       1.00      0.03      0.05        74
                Peru       1.00      0.06      0.11        70
        South Africa       0.81      0.23      0.35       115
            Thailand       0.81      0.69      0.75       169
              Turkey       0.55      0.76      0.64       188
United Arab Emirates       0.00      0.00      0.00        24
       United States       0.39      0.57      0.46       240

            accuracy                           0.54      3607
           macro avg       0.57      0.36      0.36      3607
        weighted avg       0.60      0.54      0.51      3607
```

This is much more overfit, so we can keep working with TF-IDF Vectorization

MNB Iteration Three- Using Class Weights to Improve Class Imbalance

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Using Class Weights to improve class imbalance

In [ ]:

```
# Compute class weights
from sklearn.utils import class_weight
import numpy as np

# Compute class weights
class_weights = class_weight.compute_class_weight(class_weight='balanced',
                                                  classes=np.unique(y_train)
                                                  y=y_train)
weights_dict = dict(zip(np.unique(y_train), class_weights))
weights_dict

# Use class weights dictionary to calculate sample weight (needed for Multir
sample_weights = y_train.map(weights_dict)
sample_weights

nb = MultinomialNB()
nb.fit(X_train_tfidf,
       y_train,
       sample_weight=sample_weights)

evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.8640460343871326
Testing Accuracy: 0.5450512891599667
Train and Test Accuracy Difference: 0.31899474522716587

---------------

Training F1: 0.8722675838024745
Testing F1: 0.5710646547235012

---------------

                      precision    recall  f1-score   support

           Argentina       0.29      0.45      0.35        73
           Australia       0.62      0.49      0.55       252
              Brazil       0.60      0.63      0.62       127
              Canada       0.57      0.45      0.50       241
               Chile       0.42      0.37      0.39        60
               China       0.69      0.54      0.61       249
               Egypt       0.69      0.76      0.72       104
                Fiji       0.17      0.60      0.27        15
              France       0.75      0.51      0.60       245
             Germany       0.75      0.51      0.61       252
               India       0.81      0.50      0.62       255
              Israel       0.20      0.48      0.28        29
               Italy       0.78      0.58      0.66       246
               Japan       0.74      0.57      0.64       238
              Jordan       0.18      0.72      0.29        29
               Kenya       0.13      0.57      0.21        35
              Mexico       0.73      0.52      0.61       203
             Morocco       0.38      0.53      0.44        74
         New Zealand       0.22      0.45      0.30        74
                Peru       0.39      0.53      0.45        70
        South Africa       0.40      0.64      0.49       115
            Thailand       0.74      0.76      0.75       169
              Turkey       0.66      0.71      0.68       188
United Arab Emirates       0.11      0.62      0.18        24
       United States       0.63      0.40      0.49       240

            accuracy                           0.55      3607
           macro avg       0.51      0.56      0.49      3607
        weighted avg       0.64      0.55      0.57      3607
```

The test accuracy increases a bit but the model is more overfit than the previous one

Oversampling

MNB Iteration Four- Random Oversampling

In [ ]:
```
# pip install imblearn
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
In [ ]: # Using Random Oversampling
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy='not majority', random_stat


processed = pd.DataFrame(X_train_preprocessed['Lemmatized'])
X_train_res, y_train_res = oversample.fit_resample(processed, y_train)

X_train_res = X_train_res.squeeze()

ros_tfidf = TfidfVectorizer(analyzer='word', stop_words=stopwords_list, deco
X_train_ros = ros_tfidf.fit_transform(X_train_res)
X_test_ros = ros_tfidf.transform(X_test_preprocessed['Lemmatized'])


model = MultinomialNB()

model.fit(X_train_ros, y_train_res)

resampled = model.predict(X_test_ros)
train_pred= model.predict(X_train_ros)
accuracy_score_train = accuracy_score(y_train_res, train_pred)
accuracy_score_test = accuracy_score(y_test, resampled)


print("Random Over Sampling Training Accuracy score:", accuracy_score_train)
print("Random Over Sampling Testing Accuracy score:", accuracy_score_test)
print("Difference between Train and Test Accuracy:", accuracy_score_train-ac
```

```
Random Over Sampling Training Accuracy score: 0.9340956340956341
Random Over Sampling Testing Accuracy score: 0.5572497920709731
Difference between Train and Test Accuracy: 0.376845842024661
```

MNB Iteration Five SMOTE

```
In [ ]: # Using SMOTE on class imbalance
from collections import Counter
class_counts = Counter(y_train)
print("Class distribution:", class_counts)
```

```
Class distribution: Counter({'Japan': 962, 'United States': 960, 'Canada': 9
57, 'France': 955, 'Italy': 954, 'Germany': 948, 'Australia': 948, 'China':
947, 'India': 943, 'Mexico': 877, 'Turkey': 852, 'Thailand': 711, 'South Afr
ica': 525, 'Brazil': 473, 'Egypt': 416, 'New Zealand': 326, 'Peru': 290, 'Ar
gentina': 287, 'Morocco': 286, 'Chile': 259, 'Jordan': 131, 'Israel': 131,
'Kenya': 125, 'United Arab Emirates': 96, 'Fiji': 65})
```

```
In [ ]: # finding the majority class size
majority_class_size = max(class_counts.values())
threshold = 0.1 * majority_class_size  # Classes with <10% of the majority a
minority_classes = [cls for cls, count in class_counts.items() if count < th
print("Minority classes:", minority_classes)
```

```
Minority classes: ['Fiji', 'United Arab Emirates']
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
In [ ]: # Implementing SMOTE
from imblearn.over_sampling import SMOTE
vectorizer_smote = TfidfVectorizer()
X_train_numeric = vectorizer_smote.fit_transform(X_train_preprocessed['Lemma
X_test_numeric = vectorizer_smote.transform(X_test_preprocessed['Lemmatized'
# Target only minority classes for balancing
smote = SMOTE(sampling_strategy={cls: majority_class_size for cls in minorit
# processed = pd.DataFrame(X_train_preprocessed['Lemmatized'])
X_train_res, y_train_res = smote.fit_resample(X_train_numeric, y_train)

# X_train_res = X_train_res.squeeze()

model22 = MultinomialNB()
# Build a pipeline using the TF-IDF Vectorizer and Logistic Regression
model22.fit(X_train_res, y_train_res)

resampled22 = model22.predict(X_test_numeric)
train_pred22= model22.predict(X_train_res)
accuracy_score_train = accuracy_score(y_train_res, train_pred22)
accuracy_score_test = accuracy_score(y_test, resampled22)
# Verify new class distribution
# from collections import Counter
# print("New class distribution:", Counter(y_train_res))
print("SMOTE Testing Accuracy score:", accuracy_score_train)
print("SMOTE Training Accuracy score:", accuracy_score_test)
print("Difference between Test and Train Accuracy:", accuracy_score_train- a
```

```
SMOTE Testing Accuracy score: 0.7654290480014827
SMOTE Training Accuracy score: 0.514000554477405
Difference between Test and Train Accuracy: 0.2514284935240777
```

The random oversampled model is the most overfit of all the iterations, while SMOTE is less overfit, but still doesn't perform as well as the first iteration

MNB Iteration Six- Try Using Bi-Grams

```
In [ ]: # Using Bi-Grams
bigram = TfidfVectorizer(analyzer='word',
                         stop_words=stopwords_list,
                         decode_error='ignore',
                         ngram_range=(2,2))
X_train_bg = bigram.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_bg = bigram.transform(X_test_preprocessed['Lemmatized'])
nb_bg = MultinomialNB()
nb_bg.fit(X_train_bg,
          y_train)
evaluate_model(nb_bg, X_train_bg, X_test_bg)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.8517748197448697
Testing Accuracy: 0.32242861103410037
Train and Test Accuracy Difference: 0.5293462087107693

--------------

Training F1: 0.7943808600689665
Testing F1: 0.2932248657223727

--------------
```

	precision	recall	f1-score	support
Argentina	0.00	0.00	0.00	73
Australia	0.30	0.44	0.35	252
Brazil	0.83	0.04	0.08	127
Canada	0.32	0.41	0.36	241
Chile	0.00	0.00	0.00	60
China	0.34	0.38	0.36	249
Egypt	0.67	0.02	0.04	104
Fiji	0.00	0.00	0.00	15
France	0.27	0.39	0.32	245
Germany	0.37	0.37	0.37	252
India	0.41	0.40	0.41	255
Israel	0.00	0.00	0.00	29
Italy	0.35	0.43	0.39	246
Japan	0.18	0.57	0.28	238
Jordan	0.00	0.00	0.00	29
Kenya	0.00	0.00	0.00	35
Mexico	0.38	0.39	0.38	203
Morocco	0.00	0.00	0.00	74
New Zealand	0.00	0.00	0.00	74
Peru	0.00	0.00	0.00	70
South Africa	1.00	0.05	0.10	115
Thailand	0.87	0.31	0.46	169
Turkey	0.51	0.48	0.49	188
United Arab Emirates	0.00	0.00	0.00	24
United States	0.32	0.38	0.35	240
accuracy			0.32	3607
macro avg	0.28	0.20	0.19	3607
weighted avg	0.36	0.32	0.29	3607

Bigrams improve the train accuracy but the testing accuracy is highly lowered, making the model very overift.

At this point, the best model is still iteration one

2. Random Forest

- The benefit of this is the ability to see feature importances and get more

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

el is working with the text data

In [ ]:

```
# Vectorizing
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words=stopwords_list,
                             decode_error='ignore')
X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_tfidf = vectorizer.transform(X_test_preprocessed['Lemmatized'])
```

Putting class weight as balanced deals with class imbalance

In [ ]:

```
# Fitting Random Forest
rf = RandomForestClassifier(class_weight='balanced')
rf.fit(X_train_tfidf, y_train)
```

Out[ ]:

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced')
```

In [ ]:

```
# Evaluating the model
evaluate_model(rf, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 1.0
Testing Accuracy: 0.5012475741613529
Train and Test Accuracy Difference: 0.4987524258386471

---------------

Training F1: 1.0
Testing F1: 0.5027904614118391

---------------

                      precision    recall  f1-score   support

           Argentina       0.56      0.19      0.29        73
           Australia       0.39      0.60      0.47       252
              Brazil       0.82      0.54      0.65       127
              Canada       0.36      0.45      0.40       241
               Chile       0.68      0.22      0.33        60
               China       0.49      0.46      0.47       249
               Egypt       0.60      0.68      0.64       104
                Fiji       0.70      0.47      0.56        15
              France       0.48      0.48      0.48       245
             Germany       0.51      0.53      0.52       252
               India       0.58      0.53      0.55       255
              Israel       0.69      0.31      0.43        29
               Italy       0.63      0.49      0.55       246
               Japan       0.44      0.56      0.49       238
              Jordan       0.52      0.38      0.44        29
               Kenya       0.93      0.40      0.56        35
              Mexico       0.47      0.59      0.53       203
             Morocco       0.97      0.38      0.54        74
         New Zealand       0.70      0.28      0.40        74
                Peru       0.49      0.27      0.35        70
        South Africa       0.36      0.43      0.39       115
            Thailand       0.83      0.63      0.72       169
              Turkey       0.59      0.63      0.61       188
United Arab Emirates       0.87      0.54      0.67        24
       United States       0.36      0.47      0.41       240

            accuracy                           0.50      3607
           macro avg       0.60      0.46      0.50      3607
        weighted avg       0.54      0.50      0.50      3607
```

This is the worst performing model, with the training accuracy being one.

In [ ]:
```
#Get feature importances
feat_imps = pd.Series(rf.feature_importances_,
                      index=vectorizer.get_feature_names_out())
feat_imps[:11]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

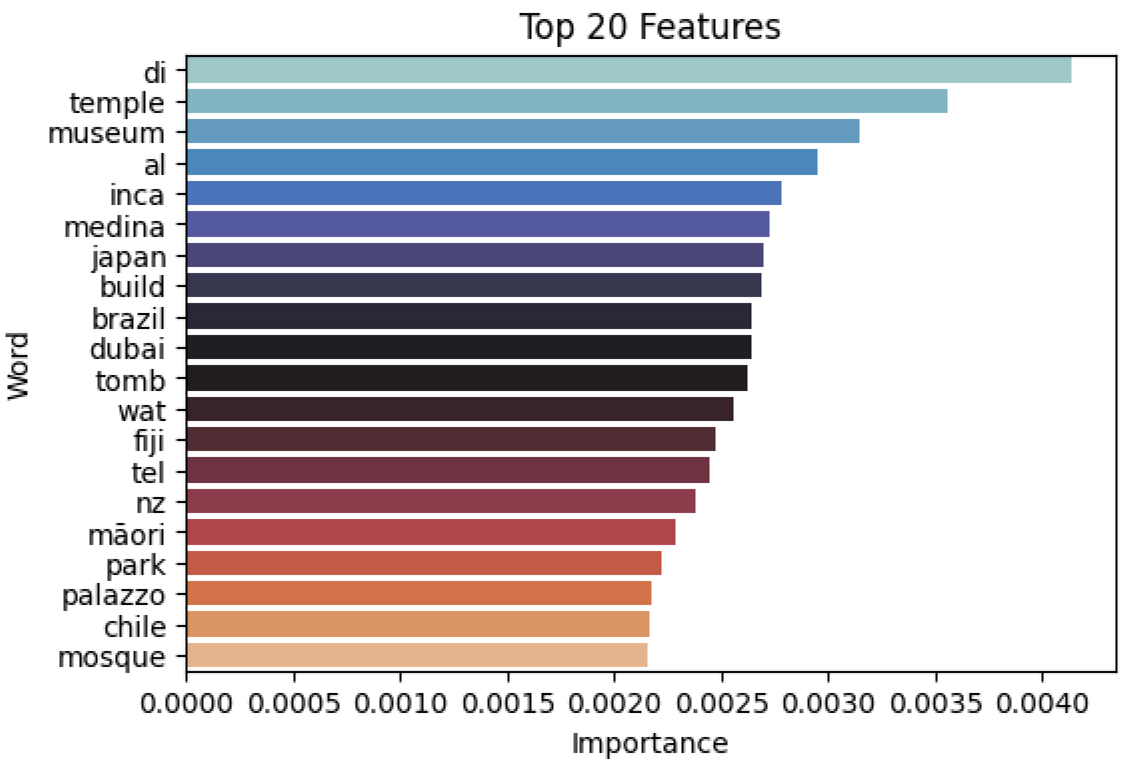
Out[ ]:
```
aachen       9.905318e-06
aah          1.342975e-06
aalara       2.150395e-08
aalto        1.828490e-06
aaron        7.676560e-05
aaronsohn    5.639301e-05
aath         2.334608e-06
ab           3.703046e-06
abaca        8.201820e-05
abad         3.222686e-05
abancay      1.133380e-06
dtype: float64
```

In [ ]:
```
# The top 20 features
top_20_feats = feat_imps.sort_values(ascending=False).head(20)
top_20_feats
```

Out[ ]:
```
di          0.004140
temple      0.003562
museum      0.003150
al          0.002947
inca        0.002785
medina      0.002723
japan       0.002697
build       0.002686
brazil      0.002646
dubai       0.002637
tomb        0.002622
wat         0.002556
fiji        0.002469
tel         0.002441
nz          0.002377
māori       0.002289
park        0.002220
palazzo     0.002170
chile       0.002168
mosque      0.002160
dtype: float64
```

In [ ]:
```
# Visualizing the top 20 features according to Random Forest
plt.figure(figsize=(6,4))
sns.barplot(x=top_20_feats, y=top_20_feats.index, palette='icefire')
plt.title('Top 20 Features')
plt.ylabel('Word')
plt.xlabel('Importance')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



This model is also overfit. Interestingly, the feature importances show a lot of country-specific words, such as Japan, Brazil, Dubai, nz, and fiji. In the future, it might be a good idea to take these kinds of words out, but for the model's use-case we can leave them in for now.

Iteration 1 is the best model so far.

3. GradientBoost

Gradient Boost Iteration One

In [ ]:
```
# Default metrics
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, G
gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train_tfidf, y_train)
```

Out[ ]:
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [ ]:
```
# Evaluate the model
evaluate_model(gb, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.7866749861342207
Testing Accuracy: 0.5167729415026338
Train and Test Accuracy Difference: 0.26990204463158696

---------------

Training F1: 0.804464004794689
Testing F1: 0.5371919203376678

---------------

                      precision    recall  f1-score   support

           Argentina       0.33      0.25      0.28        73
           Australia       0.58      0.47      0.52       252
              Brazil       0.80      0.54      0.64       127
              Canada       0.51      0.45      0.48       241
               Chile       0.31      0.25      0.28        60
               China       0.53      0.48      0.50       249
               Egypt       0.59      0.62      0.60       104
                Fiji       0.43      0.60      0.50        15
              France       0.68      0.49      0.57       245
             Germany       0.73      0.50      0.60       252
               India       0.70      0.56      0.62       255
              Israel       0.35      0.31      0.33        29
               Italy       0.66      0.54      0.59       246
               Japan       0.58      0.55      0.56       238
              Jordan       0.36      0.28      0.31        29
               Kenya       0.67      0.57      0.62        35
              Mexico       0.63      0.57      0.60       203
             Morocco       0.65      0.45      0.53        74
         New Zealand       0.50      0.42      0.46        74
                Peru       0.43      0.33      0.37        70
        South Africa       0.46      0.48      0.47       115
            Thailand       0.83      0.63      0.71       169
              Turkey       0.72      0.62      0.67       188
United Arab Emirates       0.61      0.58      0.60        24
       United States       0.19      0.65      0.30       240

            accuracy                           0.52      3607
           macro avg       0.55      0.49      0.51      3607
        weighted avg       0.59      0.52      0.54      3607
```

While this model is also overfit, it performs much better than the Random Forest, with less variation between the train and test accuracy. However, the first iteration of MNB is the best yet.

We will try making some changes to the Gradient Boost model to see if it improves (using Count Vectorization, Oversampling, using bi-grams, and class weights)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Gradient Boost Iteration Two- Count Vectorization

In [ ]:
```
# Trying Count Vectorizer to see the difference
# Vectorize the text data to be suitable for modeling
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=stopwords_list,
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])

gb.fit(X_train_cv, y_train)
evaluate_model(gb, X_train_cv, X_test_cv)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.7608153078202995
Testing Accuracy: 0.5411699473246465
Train and Test Accuracy Difference: 0.21964536049565297

---------------

Training F1: 0.7806536674890038
Testing F1: 0.5665453201599471

---------------

                      precision    recall  f1-score   support

           Argentina       0.70      0.29      0.41        73
           Australia       0.53      0.57      0.55       252
              Brazil       0.74      0.52      0.61       127
              Canada       0.52      0.46      0.49       241
               Chile       0.37      0.18      0.24        60
               China       0.58      0.48      0.53       249
               Egypt       0.66      0.67      0.67       104
                Fiji       0.75      0.60      0.67        15
              France       0.67      0.54      0.60       245
             Germany       0.82      0.54      0.65       252
               India       0.71      0.56      0.63       255
              Israel       0.56      0.31      0.40        29
               Italy       0.66      0.56      0.60       246
               Japan       0.66      0.57      0.61       238
              Jordan       0.56      0.31      0.40        29
               Kenya       0.71      0.57      0.63        35
              Mexico       0.74      0.60      0.66       203
             Morocco       0.76      0.43      0.55        74
         New Zealand       0.65      0.42      0.51        74
                Peru       0.64      0.40      0.49        70
        South Africa       0.52      0.48      0.50       115
            Thailand       0.84      0.62      0.71       169
              Turkey       0.75      0.65      0.70       188
United Arab Emirates       0.79      0.62      0.70        24
       United States       0.19      0.70      0.29       240

            accuracy                           0.54      3607
           macro avg       0.64      0.51      0.55      3607
        weighted avg       0.63      0.54      0.57      3607
```

The test accuracy has been improved, with the difference between train and test accuracy also reducing, making this the best model so far. It also has the better f1 score(weighted to accomodate for class imbalance), which makes it have the best balance of precision and accuracy.

Gradient Boost Iteration Three - Using Class Weights to fix Class Imbalance

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:

```
# Compute class weights
from sklearn.utils import class_weight
import numpy as np

# Compute class weights
class_weights = class_weight.compute_class_weight(class_weight='balanced',
                                                  classes=np.unique(y_train),
                                                  y=y_train)
weights_dict = dict(zip(np.unique(y_train), class_weights))
weights_dict

# Use class weights dictionary to calculate sample weight (needed for Multin
sample_weights = y_train.map(weights_dict)
sample_weights

gb.fit(X_train_tfidf,
       y_train,
       sample_weight=sample_weights)

evaluate_model(gb, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.7844564614531336
Testing Accuracy: 0.5206542833379539
Train and Test Accuracy Difference: 0.2638021781151797

---------------

Training F1: 0.80425582848918
Testing F1: 0.5491481743066695

---------------

                      precision    recall  f1-score   support

           Argentina       0.31      0.33      0.32        73
           Australia       0.66      0.46      0.54       252
              Brazil       0.73      0.60      0.66       127
              Canada       0.60      0.44      0.51       241
               Chile       0.27      0.27      0.27        60
               China       0.68      0.45      0.54       249
               Egypt       0.61      0.64      0.63       104
                Fiji       0.40      0.53      0.46        15
              France       0.67      0.49      0.57       245
             Germany       0.77      0.53      0.63       252
               India       0.73      0.51      0.60       255
              Israel       0.29      0.28      0.28        29
               Italy       0.73      0.49      0.59       246
               Japan       0.64      0.55      0.59       238
              Jordan       0.32      0.59      0.41        29
               Kenya       0.39      0.54      0.45        35
              Mexico       0.70      0.60      0.65       203
             Morocco       0.58      0.45      0.50        74
         New Zealand       0.38      0.45      0.41        74
                Peru       0.39      0.46      0.42        70
        South Africa       0.43      0.53      0.47       115
            Thailand       0.82      0.64      0.72       169
              Turkey       0.82      0.66      0.73       188
United Arab Emirates       0.50      0.67      0.57        24
       United States       0.18      0.61      0.27       240

            accuracy                           0.52      3607
           macro avg       0.54      0.51      0.51      3607
        weighted avg       0.62      0.52      0.55      3607
```

Oversampling

Gradient Boost Iteration Four- Random Oversampling

In [ ]:
```
# Using Random Oversampling
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy='not majority', random_stat
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
processed = pd.DataFrame(X_train_preprocessed['Lemmatized'])
X_train_res, y_train_res = oversample.fit_resample(processed, y_train)

X_train_res = X_train_res.squeeze()

ros_tfidf = TfidfVectorizer(analyzer='word', stop_words=stopwords_list, deco
X_train_ros = ros_tfidf.fit_transform(X_train_res)
X_test_ros = ros_tfidf.transform(X_test_preprocessed['Lemmatized'])


# model = GradientBoostingClassifier(random_state=42)
# Build a pipeline using the TF-IDF Vectorizer and Logistic Regression
gb.fit(X_train_ros, y_train_res)

resampled = gb.predict(X_test_ros)
train_pred= gb.predict(X_train_ros)
accuracy_score_train = accuracy_score(y_train_res, train_pred)
accuracy_score_test = accuracy_score(y_test, resampled)


print("Random Over Sampling Training Accuracy score:", accuracy_score_train)
print("Random Over Sampling Testing Accuracy score:", accuracy_score_test)
print("Train and Test Accuracy Difference:", accuracy_score_train - accuracy
```

```
Random Over Sampling Training Accuracy score: 0.851060291060291
Random Over Sampling Testing Accuracy score: 0.5261990573884114
Train and Test Accuracy Difference: 0.3248612336718796
```

This is much more overfit.

Gradient Boost Iteration Five- SMOTE

In [ ]:

```
# Use SMOTE
from imblearn.over_sampling import SMOTE
vectorizer_smote = TfidfVectorizer()
X_train_numeric = vectorizer_smote.fit_transform(X_train_preprocessed['Lemma
X_test_numeric = vectorizer_smote.transform(X_test_preprocessed['Lemmatized'
# Target only minority classes for balancing
smote = SMOTE(sampling_strategy={cls: majority_class_size for cls in minorit

X_train_res, y_train_res = smote.fit_resample(X_train_numeric, y_train)

gb.fit(X_train_res, y_train_res)

resampled = gb.predict(X_test_numeric)
train_pred= gb.predict(X_train_res)
accuracy_score_train = accuracy_score(y_train_res, train_pred)
accuracy_score_test = accuracy_score(y_test, resampled)

print("SMOTE Training Accuracy score:", accuracy_score_train)
print("SMOTE Testing Accuracy score:", accuracy_score_test)
print("Difference between Test and Train Accuracy:", accuracy_score_train- a
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
SMOTE Training Accuracy score: 0.8169518749613888
SMOTE Testing Accuracy score: 0.5084557804269476
Difference between Test and Train Accuracy: 0.30849609453444116
```

This is not the best model. The difference is higher than the second GB iteration

4. Vector Class (SVC)

SVC Iteration One

In [ ]:
```
# Using default metrics
from sklearn.svm import SVC
svc = SVC(random_state=42, probability=True)
svc.fit(X_train_tfidf, y_train)
evaluate_model(svc, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.9865501941209096
Testing Accuracy: 0.5492098696978098
Train and Test Accuracy Difference: 0.4373403244230998

---------------

Training F1: 0.9856247825588891
Testing F1: 0.5301118056758048

---------------

                      precision    recall  f1-score   support

           Argentina       1.00      0.08      0.15        73
           Australia       0.39      0.69      0.50       252
              Brazil       0.92      0.38      0.54       127
              Canada       0.42      0.64      0.50       241
               Chile       1.00      0.05      0.10        60
               China       0.58      0.62      0.60       249
               Egypt       0.82      0.53      0.64       104
                Fiji       0.00      0.00      0.00        15
              France       0.62      0.61      0.61       245
             Germany       0.61      0.70      0.65       252
               India       0.62      0.63      0.62       255
              Israel       1.00      0.03      0.07        29
               Italy       0.70      0.63      0.66       246
               Japan       0.53      0.66      0.59       238
              Jordan       0.00      0.00      0.00        29
               Kenya       1.00      0.06      0.11        35
              Mexico       0.57      0.66      0.61       203
             Morocco       1.00      0.19      0.32        74
         New Zealand       1.00      0.09      0.17        74
                Peru       0.92      0.17      0.29        70
        South Africa       0.64      0.31      0.42       115
            Thailand       0.93      0.59      0.72       169
              Turkey       0.61      0.71      0.66       188
United Arab Emirates       1.00      0.04      0.08        24
       United States       0.34      0.61      0.44       240

            accuracy                           0.55      3607
           macro avg       0.69      0.39      0.40      3607
        weighted avg       0.63      0.55      0.53      3607
```

This model is also very overfit and performs almost the same as the Random Forest one.

SVC Iteration Two- Count Vectorization

In [ ]:

```
# Trying Count Vectorizer to see the difference
# Vectorize the text data to be suitable for modeling
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=stopwords_list,
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
svc.fit(X_train_cv, y_train)
evaluate_model(svc, X_train_cv, X_test_cv)
```

```
Training Accuracy: 0.956946755407654
Testing Accuracy: 0.4665927363459939
Train and Test Accuracy Difference: 0.49035401906166004

---------------

Training F1: 0.9546058921480152
Testing F1: 0.4448988373995546

---------------

                      precision    recall  f1-score   support

           Argentina       1.00      0.07      0.13        73
           Australia       0.36      0.62      0.45       252
              Brazil       0.89      0.31      0.47       127
              Canada       0.35      0.53      0.42       241
               Chile       1.00      0.02      0.03        60
               China       0.48      0.48      0.48       249
               Egypt       0.79      0.47      0.59       104
                Fiji       0.00      0.00      0.00        15
              France       0.48      0.52      0.50       245
             Germany       0.44      0.65      0.52       252
               India       0.51      0.56      0.53       255
              Israel       0.00      0.00      0.00        29
               Italy       0.65      0.48      0.55       246
               Japan       0.46      0.58      0.51       238
              Jordan       0.00      0.00      0.00        29
               Kenya       1.00      0.06      0.11        35
              Mexico       0.40      0.62      0.48       203
             Morocco       1.00      0.12      0.22        74
         New Zealand       1.00      0.03      0.05        74
                Peru       0.86      0.09      0.16        70
        South Africa       0.45      0.32      0.38       115
            Thailand       0.91      0.52      0.66       169
              Turkey       0.52      0.65      0.58       188
United Arab Emirates       0.00      0.00      0.00        24
       United States       0.35      0.44      0.39       240

            accuracy                           0.47      3607
           macro avg       0.56      0.32      0.33      3607
        weighted avg       0.54      0.47      0.44      3607
```

It is also ver overfit and judging from the results, we will only tune the hyperparameters of MNB and GradientBoost

5. Logistic Regression

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```python
# Iteration One Using Default metrics
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train_tfidf, y_train)
evaluate_model(lr, X_train_tfidf, X_test_tfidf)
```

```
Training Accuracy: 0.824667221297837
Testing Accuracy: 0.5652897144441364
Train and Test Accuracy Difference: 0.25937750685370053

---------------

Training F1: 0.8072042031104955
Testing F1: 0.5454027840479614

---------------

                      precision    recall  f1-score   support

           Argentina       0.88      0.10      0.17        73
           Australia       0.44      0.67      0.53       252
              Brazil       0.90      0.45      0.60       127
              Canada       0.45      0.61      0.52       241
               Chile       1.00      0.08      0.15        60
               China       0.58      0.61      0.60       249
               Egypt       0.78      0.58      0.66       104
                Fiji       0.00      0.00      0.00        15
              France       0.59      0.62      0.60       245
             Germany       0.62      0.69      0.66       252
               India       0.61      0.66      0.64       255
              Israel       1.00      0.07      0.13        29
               Italy       0.67      0.64      0.66       246
               Japan       0.52      0.68      0.59       238
              Jordan       0.00      0.00      0.00        29
               Kenya       1.00      0.09      0.16        35
              Mexico       0.52      0.68      0.59       203
             Morocco       1.00      0.19      0.32        74
         New Zealand       1.00      0.14      0.24        74
                Peru       0.87      0.19      0.31        70
        South Africa       0.60      0.43      0.50       115
            Thailand       0.84      0.69      0.76       169
              Turkey       0.58      0.72      0.64       188
United Arab Emirates       1.00      0.04      0.08        24
       United States       0.40      0.59      0.48       240

            accuracy                           0.57      3607
           macro avg       0.67      0.41      0.42      3607
        weighted avg       0.62      0.57      0.55      3607
```

This is not as bad as Random Forest and SVC. Let's see if count vectorization, which improved on the GradientBoost model, improves this one too

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Logistic Regression Iteration Two- Count Vectorization

In [ ]:

```
# Trying Count Vectorizer to see the difference
# Vectorize the text data to be suitable for modeling
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=stopwords_list,
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])

lr.fit(X_train_cv, y_train)
evaluate_model(lr, X_train_cv, X_test_cv)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.9979894620077648
Testing Accuracy: 0.572775159412254
Train and Test Accuracy Difference: 0.42521430259551085

---------------

Training F1: 0.9979881489190341
Testing F1: 0.5654514593642909

---------------

                      precision    recall  f1-score   support

           Argentina       0.53      0.25      0.34        73
           Australia       0.49      0.58      0.53       252
              Brazil       0.74      0.53      0.62       127
              Canada       0.47      0.56      0.51       241
               Chile       0.58      0.18      0.28        60
               China       0.58      0.60      0.59       249
               Egypt       0.78      0.67      0.72       104
                Fiji       1.00      0.20      0.33        15
              France       0.59      0.58      0.59       245
             Germany       0.59      0.65      0.62       252
               India       0.64      0.64      0.64       255
              Israel       0.86      0.21      0.33        29
               Italy       0.66      0.63      0.65       246
               Japan       0.54      0.67      0.59       238
              Jordan       0.46      0.21      0.29        29
               Kenya       0.64      0.26      0.37        35
              Mexico       0.54      0.63      0.58       203
             Morocco       0.66      0.28      0.40        74
         New Zealand       0.55      0.31      0.40        74
                Peru       0.62      0.34      0.44        70
        South Africa       0.46      0.56      0.50       115
            Thailand       0.79      0.73      0.76       169
              Turkey       0.60      0.72      0.66       188
United Arab Emirates       0.67      0.08      0.15        24
       United States       0.46      0.57      0.51       240

            accuracy                           0.57      3607
           macro avg       0.62      0.47      0.50      3607
        weighted avg       0.59      0.57      0.57      3607
```

While the test accuracy improves a bit, the model is very overfit

6. Decision Tree

In [ ]:
```
# Iteration One using Default Metrics
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state= 42)
dt.fit(X_train_tfidf, y_train)
evaluate_model(dt, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 1.0
Testing Accuracy: 0.3834211255891322
Train and Test Accuracy Difference: 0.6165788744108678

---------------

Training F1: 1.0
Testing F1: 0.38239108000085115

---------------

                      precision    recall  f1-score   support

           Argentina       0.23      0.18      0.20        73
           Australia       0.36      0.40      0.38       252
              Brazil       0.62      0.44      0.52       127
              Canada       0.37      0.36      0.37       241
               Chile       0.33      0.15      0.21        60
               China       0.35      0.36      0.36       249
               Egypt       0.51      0.53      0.52       104
                Fiji       0.50      0.13      0.21        15
              France       0.41      0.40      0.41       245
             Germany       0.35      0.35      0.35       252
               India       0.45      0.41      0.43       255
              Israel       0.22      0.14      0.17        29
               Italy       0.43      0.40      0.42       246
               Japan       0.39      0.46      0.42       238
              Jordan       0.39      0.24      0.30        29
               Kenya       0.12      0.09      0.10        35
              Mexico       0.36      0.45      0.40       203
             Morocco       0.33      0.26      0.29        74
         New Zealand       0.26      0.23      0.24        74
                Peru       0.20      0.20      0.20        70
        South Africa       0.25      0.39      0.30       115
            Thailand       0.65      0.56      0.60       169
              Turkey       0.41      0.48      0.44       188
United Arab Emirates       0.33      0.25      0.29        24
       United States       0.32      0.33      0.32       240

            accuracy                           0.38      3607
           macro avg       0.37      0.33      0.34      3607
        weighted avg       0.39      0.38      0.38      3607
```

The Decision Tree Model is very overfit, with a training accuracy of 1.0. Clearly, the Random Forest and Decision Tree which are tree-based models are overfitting a lot.

7. KNeighbors Classifier

In [ ]:
```
# Using default metrics
from sklearn.neighbors import KNeighborsClassifier
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
knn= KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto'
knn.fit(X_train_tfidf, y_train)
evaluate_model(knn, X_train_tfidf, X_test_tfidf)
```

```
Training Accuracy: 0.6956461453133667
Testing Accuracy: 0.4835042971998891
Train and Test Accuracy Difference: 0.21214184811347758

---------------

Training F1: 0.6975934842058451
Testing F1: 0.4858197764896338

---------------

                      precision    recall  f1-score   support

           Argentina       0.19      0.33      0.24        73
           Australia       0.31      0.58      0.40       252
              Brazil       0.36      0.54      0.43       127
              Canada       0.34      0.52      0.41       241
               Chile       0.25      0.20      0.22        60
               China       0.44      0.53      0.48       249
               Egypt       0.58      0.66      0.62       104
                Fiji       0.31      0.27      0.29        15
              France       0.52      0.51      0.51       245
             Germany       0.56      0.47      0.51       252
               India       0.62      0.51      0.56       255
              Israel       0.78      0.24      0.37        29
               Italy       0.70      0.44      0.54       246
               Japan       0.66      0.56      0.61       238
              Jordan       0.79      0.38      0.51        29
               Kenya       0.71      0.29      0.41        35
              Mexico       0.50      0.56      0.53       203
             Morocco       0.84      0.28      0.42        74
         New Zealand       0.60      0.20      0.30        74
                Peru       0.48      0.33      0.39        70
        South Africa       0.49      0.37      0.42       115
            Thailand       0.67      0.65      0.66       169
              Turkey       0.65      0.66      0.65       188
United Arab Emirates       0.62      0.21      0.31        24
       United States       0.47      0.28      0.35       240

            accuracy                           0.48      3607
           macro avg       0.54      0.42      0.45      3607
        weighted avg       0.53      0.48      0.49      3607
```

While this model is also less overfit (0.22 difference between train and test accuracies), its accuracy is lower than the second iteration of GradientBoost, making GradientBoost still the best option. Its F1 score is also much lower.

In [ ]:

```
# Trying Count Vectorizer to see the difference
# Vectorize the text data to be suitable for modeling
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=stopwords_list,
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])

knn.fit(X_train_cv, y_train)
evaluate_model(knn, X_train_cv, X_test_cv)
```

```
Training Accuracy: 0.3103854686633389
Testing Accuracy: 0.09453839756029941
Train and Test Accuracy Difference: 0.21584707110303947

---------------

Training F1: 0.3038122065452747
Testing F1: 0.06070338989647092

---------------

                      precision    recall  f1-score   support

           Argentina       0.15      0.03      0.05        73
           Australia       0.08      0.56      0.14       252
              Brazil       0.08      0.02      0.04       127
              Canada       0.16      0.02      0.04       241
               Chile       0.33      0.02      0.03        60
               China       0.13      0.03      0.05       249
               Egypt       1.00      0.04      0.07       104
                Fiji       0.00      0.00      0.00        15
              France       0.25      0.01      0.02       245
             Germany       0.25      0.01      0.02       252
               India       0.47      0.03      0.06       255
              Israel       0.00      0.00      0.00        29
               Italy       0.17      0.00      0.01       246
               Japan       0.33      0.01      0.02       238
              Jordan       1.00      0.03      0.07        29
               Kenya       0.00      0.00      0.00        35
              Mexico       0.08      0.40      0.13       203
             Morocco       0.00      0.00      0.00        74
         New Zealand       0.24      0.05      0.09        74
                Peru       0.03      0.11      0.05        70
        South Africa       0.15      0.08      0.10       115
            Thailand       0.16      0.23      0.19       169
              Turkey       0.47      0.09      0.14       188
United Arab Emirates       0.00      0.00      0.00        24
       United States       0.44      0.02      0.03       240

            accuracy                           0.09      3607
           macro avg       0.24      0.07      0.05      3607
        weighted avg       0.25      0.09      0.06      3607
```

Both accuracies are very low and the model performs very poorly. This is the worst performance

Based on all these results, we will only try to tune the MNB and GradientBoost Models. Scoring will be **weighted**- Calculate metrics for each label, and find

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

their average weighted by support (the number of true instances for each label). This alters ‘macro’ to account for label imbalance, which we have in the dataset.

Hyper Parameter Tuning- Tuning MNB and GradientBoost Models

Tuning MNB

In [ ]:
```
# Commented out because it takes too long to run. Obtained parameters have b
# from sklearn.model_selection import GridSearchCV

# params_mn = {
#     'alpha':[.001, .01, .05, .1, .2, .4, .6, .8, 1],
#     'fit_prior': [True, False]
# }
# nb_gridsearch = GridSearchCV(estimator = nb, param_grid=params_mn, cv = 5,
# nb_gridsearch.fit(X_train_tfidf, y_train)
```

In [ ]:
```
# # Report best score and parameters
# print(f"Best score: {nb_gridsearch.best_score_:.3f}")
# print(f"Best parameters: {nb_gridsearch.best_params_}")
```

Let's refit it again with the new parameters and get the new test and train accuracies- Best parameters: {'alpha': 0.05, 'fit_prior': False}

In [ ]:
```
nb = MultinomialNB(alpha=0.05, fit_prior=False)
nb.fit(X_train_tfidf, y_train)
evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.9884914032168608
Testing Accuracy: 0.6165788744108678
Train and Test Accuracy Difference: 0.371912528805993

---------------

Training F1: 0.9884992190487341
Testing F1: 0.6128367682930148

---------------

                      precision    recall  f1-score   support

           Argentina       0.47      0.33      0.39        73
           Australia       0.51      0.63      0.56       252
              Brazil       0.73      0.63      0.68       127
              Canada       0.47      0.59      0.52       241
               Chile       0.67      0.27      0.38        60
               China       0.65      0.66      0.66       249
               Egypt       0.78      0.73      0.75       104
                Fiji       1.00      0.27      0.42        15
              France       0.63      0.61      0.62       245
             Germany       0.69      0.71      0.70       252
               India       0.73      0.64      0.68       255
              Israel       0.58      0.24      0.34        29
               Italy       0.76      0.70      0.73       246
               Japan       0.67      0.70      0.69       238
              Jordan       0.69      0.31      0.43        29
               Kenya       0.61      0.31      0.42        35
              Mexico       0.62      0.72      0.67       203
             Morocco       0.66      0.45      0.53        74
         New Zealand       0.42      0.34      0.37        74
                Peru       0.63      0.39      0.48        70
        South Africa       0.47      0.53      0.50       115
            Thailand       0.72      0.80      0.76       169
              Turkey       0.65      0.76      0.70       188
United Arab Emirates       0.56      0.21      0.30        24
       United States       0.46      0.54      0.50       240

            accuracy                           0.62      3607
           macro avg       0.63      0.52      0.55      3607
        weighted avg       0.63      0.62      0.61      3607
```

While the test accuracy increases, the model is now very overfit compared to the first iteration

Tuning GradientBoost Iteration Two

In [ ]:
```
# Commented out because it was taking too long to run, but the best paramete
# Since we were only trying to tune the number of estimators, we will try wi
# param_grid_gb = {
#     'n_estimators': [50, 100]
# }
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# grid_search_gb =GridSearchCV(estimator = gb,param_grid=param_grid_gb, scor
# grid_search_gb.fit(X_train_cv, y_train)
```

In [ ]:

```
gb_50 = GradientBoostingClassifier(n_estimators=50, random_state=42)
gb_50.fit(X_train_cv, y_train)
evaluate_model(gb_50, X_train_cv, X_test_cv)
```

```
Training Accuracy: 0.6460759844703272
Testing Accuracy: 0.4976434710285556
Train and Test Accuracy Difference: 0.1484325134417716

---------------

Training F1: 0.6856463180001228
Testing F1: 0.5371876347599285

---------------

                      precision    recall  f1-score   support

           Argentina       0.74      0.27      0.40        73
           Australia       0.60      0.48      0.53       252
              Brazil       0.88      0.48      0.62       127
              Canada       0.50      0.45      0.47       241
               Chile       0.42      0.18      0.26        60
               China       0.56      0.44      0.49       249
               Egypt       0.66      0.62      0.64       104
                Fiji       0.60      0.60      0.60        15
              France       0.65      0.47      0.54       245
             Germany       0.82      0.48      0.60       252
               India       0.68      0.55      0.61       255
              Israel       0.53      0.31      0.39        29
               Italy       0.64      0.52      0.57       246
               Japan       0.71      0.45      0.55       238
              Jordan       0.39      0.31      0.35        29
               Kenya       0.71      0.57      0.63        35
              Mexico       0.71      0.51      0.60       203
             Morocco       0.79      0.42      0.55        74
         New Zealand       0.81      0.41      0.54        74
                Peru       0.62      0.37      0.46        70
        South Africa       0.50      0.44      0.47       115
            Thailand       0.86      0.60      0.70       169
              Turkey       0.86      0.57      0.68       188
United Arab Emirates       0.75      0.62      0.68        24
       United States       0.15      0.74      0.25       240

            accuracy                           0.50      3607
           macro avg       0.65      0.47      0.53      3607
        weighted avg       0.64      0.50      0.54      3607
```

While the model is less overfit, the accuracies have reduced by a lot

Trying with 200 n_estimators below

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
gb_200 = GradientBoostingClassifier(n_estimators=200, random_state=42)
gb_200.fit(X_train_cv, y_train)
evaluate_model(gb_200, X_train_cv, X_test_cv)
```

```
Training Accuracy: 0.8598863006100943
Testing Accuracy: 0.5719434433046854
Train and Test Accuracy Difference: 0.2879428573054089

---------------

Training F1: 0.8680622642837107
Testing F1: 0.5876688345753117

---------------

                      precision    recall  f1-score   support

           Argentina       0.70      0.29      0.41        73
           Australia       0.52      0.60      0.55       252
              Brazil       0.76      0.53      0.62       127
              Canada       0.53      0.54      0.54       241
               Chile       0.34      0.20      0.25        60
               China       0.61      0.56      0.58       249
               Egypt       0.71      0.68      0.70       104
                Fiji       0.75      0.60      0.67        15
              France       0.66      0.57      0.61       245
             Germany       0.80      0.58      0.67       252
               India       0.73      0.60      0.66       255
              Israel       0.56      0.31      0.40        29
               Italy       0.69      0.58      0.63       246
               Japan       0.63      0.62      0.63       238
              Jordan       0.57      0.28      0.37        29
               Kenya       0.71      0.57      0.63        35
              Mexico       0.71      0.64      0.67       203
             Morocco       0.76      0.42      0.54        74
         New Zealand       0.60      0.42      0.49        74
                Peru       0.61      0.39      0.47        70
        South Africa       0.54      0.50      0.52       115
            Thailand       0.82      0.68      0.74       169
              Turkey       0.73      0.69      0.71       188
United Arab Emirates       0.79      0.62      0.70        24
       United States       0.22      0.66      0.34       240

            accuracy                           0.57      3607
           macro avg       0.64      0.53      0.56      3607
        weighted avg       0.63      0.57      0.59      3607
```

The accuracies have increased but the model is more overfit. Ultimately, the best performing model is the second iteration of the GradientBoost model- With Count Vectorization.

Reasoning:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- The model achieves the best balance between test and train accuracy, without compromising on the values themselves.
- Compared to the other contender(the first iteration of MNB), it has the best F1 score, which means that it has the best balance of precision and accuracy, which is important for the destination suggestions and predictions.

As a final check, let's remove the country names by adding them to the stopwords list to see how this impacts the model. The precense of these words in the top feature names means that they could be making the models biased.

In [ ]:
```
new_stopwords = stopwords_list + ['Argentina', 'Australia', 'Brazil', 'Canac
       'Egypt', 'Fiji', 'France', 'Germany', 'India', 'Israel', 'Italy',
       'Japan', 'Jordan', 'Kenya', 'Mexico', 'Morocco', 'New Zealand',
       'Peru', 'South Africa', 'Thailand', 'Turkey',
       'United Arab Emirates', 'United States']
```

Final Model

In [ ]:
```
# This is the final model (Iteration two of GradientBoost)
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, G
vectorizer_cv = CountVectorizer(analyzer='word', stop_words=new_stopwords, c
X_train_cv = vectorizer_cv.fit_transform(X_train_preprocessed['Lemmatized'])
X_test_cv = vectorizer_cv.transform(X_test_preprocessed['Lemmatized'])

gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train_cv, y_train)
evaluate_model(gb, X_train_cv, X_test_cv)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Training Accuracy: 0.7608153078202995
Testing Accuracy: 0.5411699473246465
Train and Test Accuracy Difference: 0.21964536049565297

---------------

Training F1: 0.7806536674890038
Testing F1: 0.5665453201599471

---------------

                      precision    recall  f1-score   support

           Argentina       0.70      0.29      0.41        73
           Australia       0.53      0.57      0.55       252
              Brazil       0.74      0.52      0.61       127
              Canada       0.52      0.46      0.49       241
               Chile       0.37      0.18      0.24        60
               China       0.58      0.48      0.53       249
               Egypt       0.66      0.67      0.67       104
                Fiji       0.75      0.60      0.67        15
              France       0.67      0.54      0.60       245
             Germany       0.82      0.54      0.65       252
               India       0.71      0.56      0.63       255
              Israel       0.56      0.31      0.40        29
               Italy       0.66      0.56      0.60       246
               Japan       0.66      0.57      0.61       238
              Jordan       0.56      0.31      0.40        29
               Kenya       0.71      0.57      0.63        35
              Mexico       0.74      0.60      0.66       203
             Morocco       0.76      0.43      0.55        74
         New Zealand       0.65      0.42      0.51        74
                Peru       0.64      0.40      0.49        70
        South Africa       0.52      0.48      0.50       115
            Thailand       0.84      0.62      0.71       169
              Turkey       0.75      0.65      0.70       188
United Arab Emirates       0.79      0.62      0.70        24
       United States       0.19      0.70      0.29       240

            accuracy                           0.54      3607
           macro avg       0.64      0.51      0.55      3607
        weighted avg       0.63      0.54      0.57      3607
```

This does not make a difference

Final Model

Ultimately, this model should tell people where they should travel based on what they want to do when on vacation. Let's take a look at some of the sample predictions this model would give them

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# Function to preprocess text
def preprocess_text(text):
    """
    Input raw text.
    Return preprocessed text.
    """
    nlp = spacy.load('en_core_web_sm')
    preprocessed = nlp(text)

    preprocessed = text.lower()
    preprocessed = re.sub('[%s]' % re.escape(string.punctuation), '', prepro
    preprocessed = re.sub('\w*\d\w*','', preprocessed)


    return [preprocessed]
```

In [ ]:
```
# The vectorizer
vectorizer_final = CountVectorizer(analyzer='word', stop_words=new_stopwords
```

In [ ]:
```
# Fitting the vectorizer
X_train_final = vectorizer_final.fit_transform(X_train_preprocessed['Lemmati
X_test_final = vectorizer_final.transform(X_test_preprocessed['Lemmatized'])
```

In [ ]:
```
# Fitting the final model once again
final_model = GradientBoostingClassifier(random_state=42)
final_model.fit(X_train_final, y_train)
```

Out[ ]:
```
GradientBoostingClassifier

GradientBoostingClassifier(random_state=42)
```

In [ ]:
```
# Obtaining the predictions
y_preds_test = final_model.predict(X_test_final)
```

In [ ]:
```
# Confirming that accuracy score is still the same
accuracy_score(y_test, y_preds_test)
```

Out[ ]:
```
0.5411699473246465
```

Testing out several texts that could be input into the model

Test Out Model

In [ ]:
```
raw_text = 'Best place for hiking and snorkeling'
preprocessed_text = preprocess_text(raw_text)
preprocessed_text
```

Out[ ]:
```
['best place for hiking and snorkeling']
```

In [ ]:
```
final_model.predict(vectorizer_final.transform(preprocessed_text))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]: array(['Mexico'], dtype=object)

In [ ]:
```
preprocessed2 = preprocess_text('Where can I go hiking and swimming in the c
print(preprocessed2)
final_model.predict(vectorizer_final.transform(preprocessed2))
```

```
['where can i go hiking and swimming in the ocean']
```

Out[ ]: array(['Australia'], dtype=object)

In [ ]:
```
preprocessed3 = preprocess_text('Which is the best place to do Wine tastings
print(preprocessed3)
final_model.predict(vectorizer_final.transform(preprocessed3))
```

```
['which is the best place to do wine tastings long walks on the beach and di
nners on the beach']
```

Out[ ]: array(['South Africa'], dtype=object)

In [ ]:
```
preprocessed4 = preprocess_text('Where can I do yoga on the beach?')
print(preprocessed4)
final_model.predict(vectorizer_final.transform(preprocessed4))
```

```
['where can i do yoga on the beach']
```

Out[ ]: array(['United States'], dtype=object)

In [ ]:
```
preprocessed5 = preprocess_text('Where can I visit historical museums?')
print(preprocessed5)
final_model.predict(vectorizer_final.transform(preprocessed5))
```

```
['where can i visit historical museums']
```

Out[ ]: array(['United States'], dtype=object)

In [285…
```
preprocessed6 = preprocess_text('Where can I see alpine meadows and glaciers
print(preprocessed6)
final_model.predict(vectorizer_final.transform(preprocessed6))
```

```
['where can i see alpine meadows and glaciers']
```

Out[285… array(['Chile'], dtype=object)

In [ ]:
```
preprocessed7 = preprocess_text('Where can I see alpine meadows, rivers, lak
print(preprocessed7)
final_model.predict(vectorizer_final.transform(preprocessed7))
```

```
['where can i see alpine meadows rivers lakes and glaciers']
```

Out[ ]: array(['Chile'], dtype=object)

Visualization Comparison of The Tested Models

We will visualize the Train Accuracy, Test Accuracy, Difference between accuracies, and F1 Scores of the best performing iterations of all the models.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# Create a table with these variables
models = ['MNB', 'RF', 'GB', 'SVC', 'LR', 'DT', 'KNN']
train_accuracies = [0.74, 1.00, 0.76, 0.99, 0.82, 1.00, 0.70]
test_accuracies= [0.52, 0.51, 0.54, 0.55, 0.57, 0.38, 0.48]

f1_scores= [0.48, 0.51, 0.57, 0.44, 0.54, 0.38, 0.49 ]
models_comparison =  pd.DataFrame({
    'Model': models,
    'Train Accuracy': train_accuracies,
    'Test Accuracy': test_accuracies,
    'F1 Score': f1_scores
})
models_comparison['Accuracies Difference'] = models_comparison['Train Accura
models_comparison
```

Out[ ]:

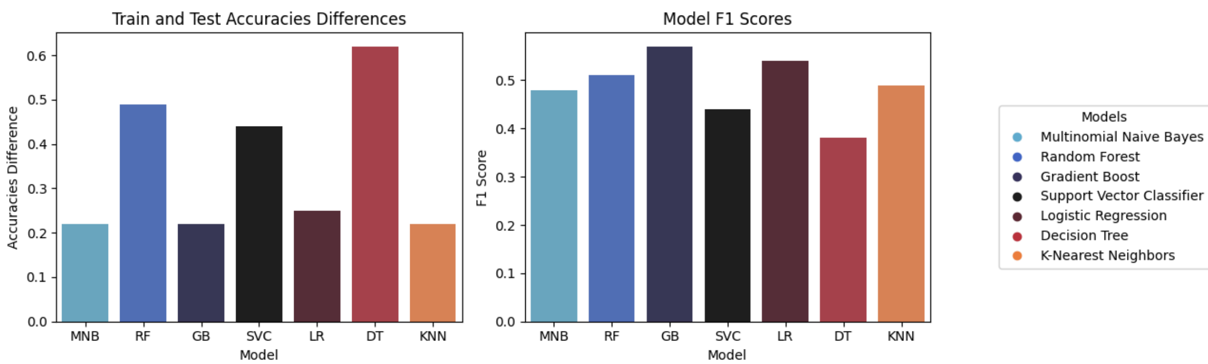
	Model	Train Accuracy	Test Accuracy	F1 Score	Accuracies Difference
0	MNB	0.74	0.52	0.48	0.22
1	RF	1.00	0.51	0.51	0.49
2	GB	0.76	0.54	0.57	0.22
3	SVC	0.99	0.55	0.44	0.44
4	LR	0.82	0.57	0.54	0.25
5	DT	1.00	0.38	0.38	0.62
6	KNN	0.70	0.48	0.49	0.22

Accuracies and F1 Score

In [ ]:
```
# Visualize the accuracies and F1 scores
import matplotlib.pyplot as plt
import seaborn as sns

# Create a mapping between abbreviations and full names
model_full_names = {
    'MNB': 'Multinomial Naive Bayes',
    'RF': 'Random Forest',
    'GB': 'Gradient Boost',
    'SVC': 'Support Vector Classifier',
    'LR': 'Logistic Regression',
    'DT': 'Decision Tree',
    'KNN': 'K-Nearest Neighbors'
}

fig, ax = plt.subplots(ncols=2, figsize=(10, 4))

# Plot 1: Train and Test Accuracies Differences
sns.barplot(x='Model', y='Accuracies Difference', ax=ax[0], data=models_comp
ax[0].set_title('Train and Test Accuracies Differences')

# Plot 2: F1 Scores
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
sns.barplot(x='Model', y='F1 Score', ax=ax[1], data=models_comparison, palet
ax[1].set_title('Model F1 Scores')

# Add a legend with full model names
handles = [
    plt.Line2D([0], [0], color=sns.color_palette('icefire', len(models_compa
    for _, abbr in enumerate(models_comparison['Model'].unique())
]

fig.legend(handles=handles, title="Models", bbox_to_anchor=(1.05, 0.5), loc=

# Adjust layout
plt.tight_layout()
plt.show()
```



Conclusions

The final model is the GradientBoost Classifier, which can predict a destination with 54% accuracy and a 57% F1 score (Iteration two of the GB Classifier in this notebook with Count Vectorization). The higher the F1 score, the better is the performance of our model, and this model has the best F1 score, and the least variation between the test and train accuracies, making it the least overfit. It will generalize best to unseen data.

The data put into this model is lowercased, punctuations removed, lemmatized, and with stop words removed.

Model Fit, Evaluation, and Selection

Accuracy and F1 score were used to evaluate model performance. With 25 target classes, accuracy is critical to gauge overall correctness. However, due to class imbalance, the weighted F1 score was prioritized to account for false positives and false negatives, offering a balanced perspective between precision and recall.

The selected model demonstrated one of the highest accuracies while avoiding
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js g candidate compared to other models. Although it

performs well in predicting across 25 classes, further improvement is needed through additional data and fine-tuning.

Final Model Performance:

- Training Accuracy: 0.76, F1 Score: 0.78
- Testing Accuracy: 0.54, F1 Score: 0.57

Multiple iterations were conducted with various models, including Multinomial Naive Bayes (MNB), Random Forest, Gradient Boosting, Decision Trees, Logistic Regression, Support Vector Classifier (SVC), and K-Nearest Neighbors (KNN). Key efforts included:

- Addressing class imbalance using oversampling techniques (Random Oversampling, SMOTE) and class weights.
- Exploring TF-IDF vectorization versus CountVectorization.
- Including bi-grams for feature engineering.
- Adding country names to the stop word list.
- Hyperparameter tuning to optimize model performance.

These iterative approaches highlight the model's potential while identifying areas for further development.

Recommendations

- **Travel Enthusiasts/Travelers**: The interactive dashboard created offers an opportunity for travel enthusiasts to shorten the time involved in decision making based on their likes and interests. Through this, they could get an opportunity to explore their best interests despite having limited time. The product simplifies their search for an appropriate travel destination.

- **Travel platforms and websites**: :Travel platforms should broaden their content to include a wider range of countries, particularly those currently underrepresented. This approach would offer more balanced visibility to diverse regions with unique attractions.

- **Destination Marketers:**

  > - The project highlights the limitations in vocabulary used to descibe top attractions in countries such as museum, art gallery, unesco world heritage, which could point to a bias in the marketing of top attractions, focusing on specific types of attractions only. Destination marketers can apply this knowledge and integrate a broader marketing approach that could highlight

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

> the rare but unique destinations to present a more balanced image.

> - Enhance Kenyan Destination Marketing:Promote Kenya's coastal beaches, urban culture, and adventure sports alongside its wildlife offerings. Use comprehensive language in promotional materials to portray Kenya as a multi-faceted destination, attracting a broader range of tourists.

Future Implementation

1. Refine Machine Learning Model:Improve the text classification model's accuracy, especially for underrepresented countries. Steps include:

    - *Balance the dataset*: Ensure even distribution of countries in the training data.
    - *Expand feature set*: Incorporate advanced text processing techniques to capture nuanced descriptors.
    - *Tune the model*: Experiment with various machine learning models and hyperparameters.
    - *Implement user feedback*: Incorporate a mechanism for users to rate and refine suggestions, enabling continuous improvement.
2. Integrate with Travel Platforms:Implement the machine learning model as a personalized recommendation tool on travel websites and apps. This AI-driven feature could help users discover new destinations based on their preferences.

3. Data Expansion and Enrichment:

    - *Incorporate Additional Data*: Include diverse travel websites, lesser-known attractions, and user-generated content.
    - *Geospatial Data*: Integrate location data to enhance recommendation accuracy based on destination types.
4. Advanced NLP Techniques:

    - *Deep Learning Models*: Utilize transformers (e.g., BERT, GPT) for improved text classification accuracy.
    - *Topic Modeling*: Apply techniques like Latent Dirichlet Allocation to uncover hidden topics in travel descriptions.
5. User Profiling and Personalization:

    - *User Profiles*: Create profiles based on travel history and preferences for personalized recommendations.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- *Adaptive Recommendations*: Refine suggestions based on user interactions and feedback.

6. Mobile App Development:

- Create an app offering real-time recommendations based on user preferences, travel deals, and seasonal factors.
- Incorporate user feedback for continuous model improvement.

Deployment

Code can be viewed in the Deployment folder Here is the final product

In [ ]:
```
# Import streamlit and pickle for deployment
import streamlit as st
import pickle
```

We will use cosine similarity to match actual attractions to the predicted country. Steps are below

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Example user query
user_query = "Where can I see alpine meadows and glaciers?"

# Step 1: Preprocess the input query
processed_input = preprocess_text(user_query)

# Step 2: Predict the country
predicted_country = final_model.predict(vectorizer_final.transform(processed
print("Predicted Country:", predicted_country)

# Step 3: Filter attractions by the predicted country
filtered_data = preprocessed_df[preprocessed_df['Country'] == predicted_cour

if filtered_data.empty:
    print("No attractions found for the predicted country.")
else:
    # Step 4: Rank Attractions by Similarity
    vectorizer_attractions = TfidfVectorizer(stop_words='english')
    filtered_tfidf = vectorizer_attractions.fit_transform(filtered_data['Des
    query_tfidf = vectorizer_attractions.transform([user_query])

    # Calculate similarity
    similarity_scores = cosine_similarity(query_tfidf, filtered_tfidf).flatt
    filtered_data['Similarity'] = similarity_scores

    # Step 5: Output Top Attractions
top_attractions = filtered_data.sort_values(by='Similarity', ascending=False

                              ctions:")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
for idx, row in top_attractions.iterrows():
    print(f"Attraction: {row['Attraction']}")
    print(f"Country: {row['Country']}")
    print(f"Description: {row['Description']}")
    print("-" * 50)
```

```
Predicted Country: Chile

Recommended Attractions:
Attraction: Parque Nacional Bernardo O'Higgins
Country: Chile
Description: Virtually inaccessible, Parque Nacional Bernardo O'Higgins rema
ins an elusive cache of glaciers. It can be entered only by boat. From Puert
o Natales, full…
--------------------------------------------------
Attraction: Parque Nacional Nevado Tres Cruces
Country: Chile
Description: It requires effort to get here, but the rewards are tremendous.
This remote national park is home to soaring peaks, glittering alpine lakes
and abundant…
--------------------------------------------------
Attraction: Laguna Miñiques
Country: Chile
Description: The smaller of two dramatic alpine lakes, the shimmering blue s
urface of Miñiques looks all the more stunning against a backdrop of chisele
d snow-covered…
--------------------------------------------------
Attraction: Parque Nacional Hornopirén
Country: Chile
Description: Relatively unknown and not often accessed, Parque Nacional Horn
opirén protects a lush wilderness of alpine terrain. It remains obscure main
ly because…
--------------------------------------------------
Attraction: Parque Nacional Huerquehue
Country: Chile
Description: The 125-sq-km preserve, founded in 1912, is a little wonderland
of waterfalls, alpine lakes and araucaria forests and the creatures that tra
verse them,…
--------------------------------------------------
```

In [ ]:

```python
# Save the final model and vectorizer for use in the deployment folder
import joblib
joblib.dump(final_model, 'deployment/final_model.pkl')
joblib.dump(vectorizer_final, 'deployment/vectorizer_final.pkl')
```

Out[ ]: ['deployment/vectorizer_final.pkl']

In [ ]:

```python
# Save preprocessed_df as a CSV file for use in the deployment folder
preprocessed_df.to_csv('deployment/preprocessed_df.csv', index=False)
```

Here is a link to the deployed web app. Travel Word Finder

In [296…

```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# Load and display the image
img = mpimg.imread('/Users/rosew/Desktop/Moringa/phase_5/Travel-WordFinder/I
plt.imshow(img)
plt.axis('off')  # Turn off axis labels
plt.show()
```

This notebook was converted with convert.ploomber.io

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js